La loi de convexité énergie-fréquence de la consommation des programmes : modélisation, thermosensibilité et applications

Karel de Vogeleer

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2015-ENST-0046

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « Informatique et Réseaux »

présentée et soutenue publiquement par

Karel De Vogeleer

4 Septembre 2015

La loi de convexité énergie-fréquence de la consommation des programmes : modélisation, thermosensibilité et applications

Co-Directeur de la thèse	:	**Gérard MEMMI**
Co-Directeur de la thèse	:	**Pierre JOUVELOT**
Encadrant de la thèse	:	**Fabien COELHO**

Jury

M. Maurice GAGNAIRE, Prof., TELECOM ParisTech	Examinateur
M. William JALBY, Prof., Université de Versailles Saint-Quentin-en-Yvelines	Rapporteur
M. Georg HAGER, Dr. rer. nat. habil., Friedrich-Alexander Uni Erlangen-Nürnberg	Rapporteur
M. Christophe GUETTIER, Dr., SAGEM	Examinateur

T H È S E

TELECOM ParisTech
école de l'Institut Mines-Télécom - membre de ParisTech
46 rue Barrault 75013 Paris - (+33) 1 45 81 77 77 - www.telecom-paristech.fr

The Energy/Frequency Convexity Rule of Energy Consumption for Programs: Modeling, Thermosensitivity and Applications

Karel De Vogeleer

Abstract

Both theoretical and experimental evidence is presented in this work for the existence of an Energy/Frequency Convexity Rule, which relates energy consumption and microprocessor frequency for nanometer-scale microprocessors. Typical nanometer-scale application processors were monitored running specific compute-intensive kernels using high-resolution power gauges. Data gathered during several week-long acquisition campaigns suggest that energy consumed is strongly correlated with the microprocessor's frequency, and, more interestingly, the curve exhibits a clear minimum over the processor's frequency range. An analytical model for this behavior is provide and motivated, which fits well with the data. The circumstances are discussed under which this convexity rule can be exploited, and when other methods are more effective, with the aim of improving the microprocessor's energy efficiency. The Energy/Frequency Convexity Rule is potentially more exploitable by low-power systems, such as battery-powered and embedded systems, and less likely by high-performance computer systems. The Energy/Frequency Convexity Rule is also applied to multi-buddy systems, Amdahl's law and heterogeneous computing.

Given that the microprocessor's energy consumption is temperature-dependent, a macro-level temperature/power relationship for application processors is introduced and experimentally validated. By adopting a holistic view, this model is able to take into account many of the physical effects that occur within such systems. Via measurements on two pertinent platforms sporting nanometer-scale application processors, it is shown that the power/temperature relationship is indeed very likely exponential over a 20°C to 85°C temperature range. The data suggest that, for a temperature range between 20°C and 55°C, a quadratic model is still accurate and a linear approximation is acceptable. Power transformation models are also presented that aim at canceling the temperature biases in power traces. These transformation models are developed to increase the accuracy and meaningfulness of power measurement traces.

Besides static power measurements, the transient power and thermal behavior are also analyzed by means of the cooling laws and the temperature/power relationship models. Exponential cooling models are justified for actively-cooled microprocessors. For passively cooled processors however, as frequently found in embedded systems, an exponential law may not be theoretically justified. Here, the tractability of the exact cooling law for a passively-cooled body is analyzed, subject to radiative cooling and a modest level of heat loss via convection. Focusing then on embedded microprocessors, the performance difference between the new passive cooling law and the conventionally-used exponential one is compared. It is shown that, for large surface sizes, the radiative cooling component can be comparable to the convective cooling one. However, for large cooling surface areas of the order of $10\,\mathrm{cm}^2$ or more, it is shown that the differences between the passive cooling law and the exponential cooling law are significant. The results thus suggest that, in the absence of accurate temperature measurements, an exponential cooling law is only accurate enough for small-sized SoC systems that require low processing overhead.

Résumé

Cette these s'intéresse à la consommation énergétique d'un système embarqué durant l'execution d'un programme. Une preuve théorique est présentée dans cette thèse et expérimentale de l'existence d'une loi de convexité énergie-fréquence de la consommation des programmes, qui concerne la consommation d'énergie et la fréquence des microprocesseurs à l'échelle nanométrique. Des noyaux de calcul intensif spécifiques ont été exécutés sur des processeurs d'applications typiques, à l'échelle nanométrique, et leurs caractéristiques mesurées en utilisant des capteurs de puissance à haute résolution. Les données recueillies lors de nombreuses campagnes d'acquisition de données longues de plusieurs semaines chacune suggèrent que la consommation est fortement corrélée avec la fréquence du microprocesseur et, ce qui est extrêmement intéressant, que la courbe présente un minimum clair sur la gamme de fréquences utilisables sur les processeurs. Un modèle analytique de ce comportement est fourni et motivé; il cadre particulièrement bien avec les données. Les circonstances dans lesquelles cette règle de convexité peut être exploitée sont discutées, en particulier dans le but d'améliorer l'efficacité énergétique du microprocesseur. La loi de convexité énergie-fréquence de la consommation des programmes est potentiellement plus exploitable par les systèmes de faible puissance, tels que les systèmes embarqués et alimentés par piles ou batteries, et moins susceptible de l'être par les systèmes informatiques de haute performance. La loi de convexité énergie-fréquence de la consommation des programmes est également appliquée aux systèmes multi-coeurs, à la loi d'Amdahl et aux systèmes informatiques hétérogènes.

Étant donné que la consommation d'énergie du microprocesseur dépend de sa température, une relation température/puissance au niveau macro pour les processeurs d'application est également introduite et validée expérimentalement dans cette thèse. En adoptant une vision holistique, ce modèle est capable de prendre en compte de nombreux effets physiques qui se produisent dans de tels systèmes. Via des mesures sur deux plates-formes pertinentes comportant des processeurs d'applications à l'échelle nanométrique, il est montré que la relation puissance/température se comporte de manière exponentielle entre 20°C et 85°C. Les données suggèrent de plus que, pour une plage de températures comprise entre 20°C et 55°C, un modèle quadratique est toujours suffisamment précis et qu'une approximation linéaire est même acceptable. Des modèles de transformation d'énergie visant à annuler les biais liés à la température dans les mesures de puissance sont également présentés. Ces modèles de transformation ont été mis au point afin d'augmenter la précision et la pertinence des traces de mesure de puissance.

Outre les mesures statiques de puissance, les comportements transitoires en puissance et température sont également analysés à l'aide des lois de refroidissement et des modèles température/puissance. Il s'avère que des modèles de refroidissement exponentiels sont justifiés pour des microprocesseurs refroidis de manière active. Cependant, pour les processeurs refroidis passivement que l'on trouve fréquemment dans les systèmes embarqués, une loi exponentielle ne peut pas être justifiée théoriquement. En conséquence, la loi ex-

acte de refroidissement pour un corps à refroidissement passif est analysée, sous condition de refroidissement radiatif et d'un niveau modeste de perte de chaleur par convection. Si l'on se concentre sur les microprocesseurs embarqués, il y a une différence de performance entre la nouvelle loi de refroidissement passif et celle, exponentielle, classiquement utilisée. On montre que, pour les grandes surfaces, le refroidissement par rayonnement peut être comparable à celui lié à la convection. Toutefois, pour les grandes surfaces de refroidissement de l'ordre de $1\,\mathrm{dm}^2$ ou plus, les différences entre la loi de refroidissement passif et la loi exponentielle de refroidissement sont importantes. Ces résultats suggèrent donc que, en l'absence de mesures précises de la température, une loi exponentielle de refroidissement n'est suffisamment précise que pour les petits systèmes SoC ne nécessitant qu'une faible charge de traitement.

Acknowledgements

I would like to show much appreciation and gratitude to my doctoral advisors and supervisors. Gerard Memmi's wisdom extends far beyond the particular research field handled in this work. He always had interesting comments and insight to share that greatly improved this work. I also thank him for the administrative support during my years at TELECOM ParisTech. Pierre Jouvelot's experience and scientific expertise made this work more valuable. We also had many non-work related conversations in the park, or at the school's premises, that were inspiring and amusing. Gerard and Pierre devoted significant amounts of time in shaping me as a researcher, which I appreciate vastly. Fabien Coelho I also like to thank for his comments and contributions to this work. I also like to thank the reviewers of my thesis, William Jalby and Georg Hager, for their comments and expertise.

Even tough I resided most of the time at TELECOM ParisTech's premises, I also visited CRI at MINES ParisTech in Fontaintebleau on a regular basis, where Pierre and Fabien are based. I thank the whole staff there, notably Francois Irigoin, for being open, welcoming and supportive to collaboration between institutes.

I also thank the people at the department of Biosystems Engineering of Ghent University who provided support for a part of this work. I had a great pleasure the time I stayed there, and the faculty was always helpful.

In fact, I started my doctoral studies in Sweden at Blekinge Institute of Technology, before I came to Paris. I made good friends there throughout the years. Thanks goes out also to my former supervisors there.

Last, but not least, I appreciate the support of my family too, during my Ph.D studies. The distance created by studying/working abroad is not always easy for the family. In particular, I want to wish the best of luck to my sister, who is also perusing a doctoral degree. My greatest fear was that she would graduate before me. I curbed this threat nicely.

Karel De Vogeleer,
Paris, France
December, 2015

Contents

List of Figures

List of Tables

List of Procedures

Table of Symbols

Symbol	Description	Units
cc_{b}	program clock cycles	-
f_{k}	clock frequency thieves	1/s
cc_{m}	memory clock cycles	-
f	clock frequency	1/s
f_{cpu}	microprocessor clock frequency	1/s
f_{mem}	memory clock frequency	1/s
σ_{x}	out-of-order execution coverage	-
c	number of active cores	-
Δt	execution time / timespan	s
M_a	number of external memory accesses	-
β	average waiting time per clock cycle	s
Q_{sys}	heat flux of the system	$\mathrm{W/m^2}$
Q_{in}	heat flux into the system	$\mathrm{W/m^2}$
Q_{ihg}	heat flux of the internal heat conversion	$\mathrm{W/m^2}$
Q_{out}	heat flux to the environment	$\mathrm{W/m^2}$
Q_{d}	heat flux resulting from conduction	$\mathrm{W/m^2}$
Q_{v}	heat flux resulting from convection	$\mathrm{W/m^2}$
Q_{r}	heat flux resulting from radiation	$\mathrm{W/m^2}$
m	body mass	kg
c_p	specific heat capacity at constant pressure	$\mathrm{J/kg{\cdot}K}$
ρ	density	$\mathrm{kg/m^3}$
α	thermal diffusivity	$\mathrm{m^2/s}$
k	thermal conductivity	$\mathrm{W/m{\cdot}K}$
A	surface area	$\mathrm{m^2}$
n	normal vector	-
h	convective heat transfer coefficient	$\mathrm{W/m^2{\cdot}K}$
h_{ac}	active cooling heat transfer coefficient	$\mathrm{W/m^2{\cdot}K}$
h_{pc}	passive cooling heat transfer coefficient	$\mathrm{W/m^2{\cdot}K}$
T_s	surface temperature	°C or K
T_a	ambient temperature	°C or K
T_0	temperature at $t = 0$	°C or K
σ	Boltzmann constant	$\mathrm{W/m^2{\cdot}K^4}$
ϵ	emissivity	-
σ	electrical conductivity	Ω/m
r	radius	m
z	height	m
φ	angle	c
θ_{JB}	junction-to-board (thermal) resistance	$\mathrm{m^2{\cdot}K/W}$

Symbol	Description	Units
ω	temperature scaling factor	-
ω_*	polynomial roots	-
P_r	reference power	W
P_m	measured power	W
ξ	microprocessor's power profile	W/Hz·V^2
γ	microprocessor's leakage current profile	V^{-1}
T_r	reference temperature	°C or K
T_m	measured temperature	°C or K
η_*	fitting parameters	W/°C*
ΔT	temperature difference	°C or K
m_*	voltage/frequency regression coefficients	V/Hz*
s_*	power profile regression coefficients	W/Hz*
s_x	power profile scaling factor	-
E_{sys}	energy consumption of the system	J
E_{n}	normalized energy consumption of the system	J
η_*	internal heat generation regression coefficients	W/K*
S	surface area	m^2
C	heat capacity of the body	J/K
T_{tr}	point of thermal runaway	°C or K
$*^\circ$	energy or power in idle-state	J or W
$*^+$	energy or power in active-state	J or W
$\bar{*}$	static energy or power component	J or W
$\tilde{*}$	dynamic energy or power component	J or W
δ	number of energy state transactions	-
ϱ	proportion of code that is parallelizable	-
n	number of cores	cores
k	number of active cores	cores
p	dimensionless time penalty due to frequency scaling	-
q	dimensionless power penalty due to frequency scaling	-
Λ	dimensionless performance scaling factor	-
κ_*	polynomial coefficients	-
c_0	integration constant	-

Résumé en Français

Introduction

Même une petite quantité d'énergie économisée grâce à un meilleure contrôle sur la planification de la fréquence et des effets thermiques sur le comportement énergétique des microprocesseurs peut avoir des impacts sociaux, monétaires et écologiques importants. Comprendre et modéliser ces relations avec précision peut avoir un impact au-delà de la seule gestion optimisée de fonctionnement des systèmes informatiques. Cela souligne l'aspect crucial de l'optimisation de l'énergie dans les microprocesseurs, en particulier pour les systèmes embarqués et de calcul haute performance (HPC). Ce point est particulièrement important pour les systèmes utilisant une batterie électrique, tels que les smartphones, capteurs, etc. Quelle que soit leur origine, tous les appareils portatifs partagent en effet le même talon d'Achille: la *batterie* [123]. A l'autre bout du spectre, les systèmes HPC sont eux, entre autres, contraints par l'alimentation électrique disponible.

La disponibilité du service offert par des dispositifs alimentés par batteries est une question critique pour quasiment tout utilisateur ou exploitant. L'autonomie de la batterie est devenue un facteur important pour l'expérience utilisateur, comme cela a été démontré par des études académiques [37] ainsi que par des enquêtes menées par les canaux d'information de détail [1] depuis la création des appareils portables. Même si la capacité des batteries et leurs performances augmentent au fil du temps, l'amélioration de l'efficacité énergétique des systèmes à base de batterie est et reste essentielle, parce que entre autres, la demande de puissance devance les développements dans les capacités des batteries [94, 117]. Les super-ordinateurs, d'autre part, sont eux limités par la puissance disponible fournie par le réseau alimentant le centre de données. Ces ordinateurs doivent donc fonctionner avec une puissance et un budget contraints. En outre, l'expérience utilisateur, les produits et la conception de circuits sont limités par la dissipation en puissance de crête des composants électriques. Comprendre les différents aspects de la consommation de l'énergie des systèmes (embarqués) est donc une question clé. La fréquence du processeur, ainsi que la durée et la température de l'exécution, sont, entre autres, des facteurs importants qui influencent la consommation d'énergie et le profil de puissance de l'exécution des programmes lié à l'architecture hardware sur laquelle il s'exécute. Fournir des modèles de consommation d'énergie peut ainsi ouvrir la voie à l'optimisation de l'énergie à l'interface entre logiciel et matériel.

Historiquement, les performances en termes de temps d'exécution ou de consomation mémoire ont été la principale mesure de la performance au niveau des couches logicielles et matérielles. Aujourd'hui, l'efficacité énergétique et la dissipation d'énergie deviennent des indicateurs de performance tout autant importants, surtout pour les appareils alimentés

[1] Autonomie : préoccupations des utilisateurs mobiles: http://www.cnn.com/2005/TECH/ptech/09/22/telephone

par des batteries. Le temps d'exécution demeure bien sûr une mesure de performance pour les systèmes qui nécessitent une performance soutenue, comme les super-ordinateurs. Mais pour les systèmes qui sont conçus pour offrir une interface homme-machine riche, la puissance de calcul soutenue n'est pas nécessairement un objectif primordial ; en revanche, les temps de réponse de l'application doivent suffire à satisfaire l'expérience de l'utilisateur. Ces approches différenciées de l'informatique peuvent nécessiter des méthodes d'optimisation de l'énergie différentes. Par exemple, des bouffées sporadiques de dissipation de puissance dépassant l'enveloppe thermique (TDP) (Thermal Design Power) peuvent être inoffensives pour le matériel et offrir un soutien temporaire pour un système devant être prêt â répondre rapidement [90].

Optimiser de manière efficace le profil de puissance du microprocesseur et le flux de chaleur qu'il génère est une question cruciale pour les technologies futures. Les technologies actuelles et futures ne peuvent permettre, dans les cas les plus extrêmes, qu'à un sous-ensemble de la logique du microprocesseur d'être actif à un moment donné pour rester compatible avec les niveaux acceptables de dissipation de puissance maximale du microprocesseur. La fraction de logique ainsi désactivée est appelée *silicium noir*. En outre, les limites de puissance et de temps de transmission physique ont conduit à maintenir constante la fréquence d'horloge des technologies actuelles et futures [33]. En conséquence, pour améliorer la performance, le parallélisme est aujourd'hui de plus en plus employé dans les microprocesseurs. Le multi-threading simultané (SMT) sur multi-coeurs a fourni des économies d'énergie substantielles avec le matériel récent [34].

Outre l'optimisation de l'efficacité énergétique au niveau matériel, le logiciel peut également être conçu de façon à minimiser les exigences de consommation d'énergie. Pour les systèmes embarqués, six aspects d'optimisation de l'énergie peuvent se trouver au niveau logiciel: les systèmes d'exploitation orientés-énergie, la gestion efficace des ressources, l'impact de la configuration de l'interaction des utilisateurs avec les appareils mobiles et les applications, les interfaces sans fil, la gestion des capteurs et la prise en compte des services informatiques de Cloud [117]. Toutes ces facettes pourraient être optimisées individuellement, mais elles doivent plutôt être optimisées de concert pour atteindre le plus grand gain d'énergie. Il s'agit là d'une tâche ambitieuse et complexe. Au niveau logiciel, les techniques d'optimisation d'énergie et de puissance peuvent être grossièrement divisées en méthodes *statique* et *dynamique*. Les méthodes statiques comprennent, entre autres, la conception efficace de logiciels et l'optimisation du compilateur. Pour que ces approches permettent d'optimiser énergie et puissance, un profil énergétique du matériel sur lequel l'exécution du logiciel est prévue est nécessaire. Cela comporte l'inconvénient que le logiciel ne sera optimal, pour une valeur d'énergie ou de puissance donnée, que pour un type de configuration matérielle spécifique. Avec ce type d'optimisation statistique, cependant, la spécification du matériel peut ne pas être connue complètement et des informations de contexte peuvent ne pas toujours être disponibles, par exemple, lors de l'étape de développement. En outre, la conception de l'architecture économe en énergie est très sensible à la charge, nécessairement variable, de travail [34] du processeur. Cela impose une incertitude sur l'efficacité finale des techniques d'optimisation statiques.

L'optimisation dynamique, d'autre part, présente l'avantage que des informations de contexte sont disponibles et les spécifications du matériel sont connues. Avec ces informations supplémentaires, si elle sont accessibles, le système et le logiciel peuvent dynamiquement s'adapter pour améliorer l'efficacité énergétique et de puissance. Parmi les méthodes en ligne, on trouve : l'optimisation dynamique de compilation, l'optimisation du bytecode, et les techniques d'optimisation au niveau du système. Comme pour l'optimisation

statique, la connaissance des profils énergétiques du matériel et des logiciels contribue à améliorer les décisions prises par les techniques dynamiques d'optimisation d'énergie et de puissance.

L'objectif de cette thèse est de se concentrer sur les profils énergétiques et, plus précisément, sur la façon dont deux paramètres fondamantaux : la température et la fréquence d'horloge du microprocesseur affectent la consommation d'énergie et les conditions de fonctionnement optimales d'un système informatique.

La loi de convexité énergie – fréquence

Les courants circulant dans un microprocesseur, ou circuit intégré (IC), respectent les lois fondamentales d'électricité (Ohm, Kirchhoff, ...). Ainsi ils produisent une dissipation de chaleur proportionnelle à I^2R, où R est la résistance et I le courant électrique global du système étudié. La première loi de la thermodynamique indique que, en fonctionnement régulier, l'apport d'énergie d'un système est égal à la consommation d'énergie du système. Ainsi, en l'absence d'autres interactions de l'énergie, et en négligeant l'information liée à l'énergie elle-même, la seule forme d'énergie émanant d'un microprocesseur est la chaleur engendrée par les courants circulant à travers des éléments résistifs [17]. Par conséquent, la dissipation de chaleur du microprocesseur est très proche de sa consommation d'énergie. Le processus de *génération de chaleur interne* du microprocesseur est parfois aussi appelé *conversion de chaleur interne*. Un microprocesseur présente un comportement thermique transitoire où les délais de refroidissement et de chauffage dépendent de sa capacité calorique. Le comportement thermique transitoire dure plus longtemps pour les systèmes ayant des capacités caloriques plus grandes que pour ceux ayant des capacités plus petites. Ces systèmes sont traditionnellement modélisés par des circuits RC, en se fondant sur l'équivalence courant/chaleur (voir la section 2.3.3), à savoir un filtre passe-bas, où la température du système est proportionnelle à la tension aux bornes des condensateurs [27].

En réalité, le comportement (transitoire) thermique est plus compliqué, parce que la résistance R et les courants I dépendent de la température du microprocesseur. D'une manière générale, il est démontré expérimentalement que le besoin de puissance croît super-linéairement avec la température du système. Dans le chapitre 3, cette relation entre température et puissance est analysée en détail. En particulier, les comportements thermiques de trois processeurs d'applications pour systèmes embarqués sont discutés. Il sera montré que la relation température / puissance est correctement modelisée par une loi exponentielle. Pour de petites variations de température, que l'on rencontre fréquemment dans les systèmes embarqués, des approximations linéaires et quadratiques sont adéquates. En outre, des modèles de transformation de puissance sont conçus afin de servir à éliminer les biais de température à partir de traces de mesure, et de réduire ainsi les effets de distance et de dispersion d'un capteur de température lointain. De tels modèles d'alimentation et de transformation sont intéressants pour les raisons suivantes.

- Les mesures sont essentielles pour la compréhension puis l'optimisation des systèmes énergie critique [34, 117]. Un manque de mesures de puissance détaillées portera atteinte aux efforts visant à réduire la consommation d'énergie sur un logiciel moderne [34].

- Des travaux de recherche ont essayé de décrire la relation température / puissance en se concentrant sur un sous-ensemble des courants de fuite décrits par BSIM [36, 68, 69, 108, 114]. De tels modèles font l'hypothèse que les courants de fuite sont

uniquement dépendants de la température. Les courants, cependant, sont également fonction du temps, car les tensions appliquées aux bornes des transistors changent avec le temps, ce qui introduit une complexité supplémentaire de modélisation. Les courants de fuite dépendent d'une multitude de facteurs spécifiques à chaque microprocesseur, qui ne sont pas tous encore connus précisément. Par conséquent, il peut être intéressant de proposer une modélisation agrégée de la relation température / puissance, comme approche alternative aux modèles dérivés de BSIM.

- En outre, il est essentiel de bien comprendre l'importance des erreurs dues aux mesures de puissance en fonction des différents régimes de température du système à l'étude. Par exemple, certaines de nos mesures de puissance exhibent jusqu'à 10% d'écart en raison d'effets transitoires de température [28]. Ainsi, si les mesures de puissance doivent être comparées, il est interessant que les effets de la température puissent être controlés de manière à obtenir une base équitable d'analyse.

- Du point de vue de la mesure, il est également important de comprendre l'impact de la génération de chaleur interne sur la température et de l'effet de capteurs distants. Des protocoles de mesures de puissance précis et reproductibles sont difficiles à établir, en raison du comportement thermique transitoire des composants électriques. Plus précisément, pour un programme de test donné, la mesure de l'alimentation du microprocesseur peut conduire à des valeurs différentes pour différentes températures du microprocesseur. Par souci de précision et de comparaison équitable entre différentes mesures de puissance, il est donc d'une importance vitale de pouvoir contrôler ou annuler les effets du comportement thermique transitoire.

La dépendance en température fait partie des spécifications relatives à un microprocesseur; des températures élevées ont un effet négatif sur sa durée de vie. La fiabilité des produits électroniques est influencée par les gradients spatiaux ou temporels de température, ainsi que par leurs valeurs, dans l'absolu [63]. Les gradients thermiques, qui se produisent à la fois dans l'espace et dans le temps, induits par la variabilité de la charge de microprocesseur et des opérations, engendrant des cycles thermiques qui ont un effet négatif sur le taux de panne des systèmes [60]. L'International Technology Roadmap for Semiconductors (ITRS) affirme même que les coûts de traitement et les spécifications de performance peuvent être limités par la durée de vie, la fiabilité étant la principale préoccupation dans la phase de conception d'un microprocesseur [51]. Le temps moyen avant fin de bon fonctionnement (MTTF) des équipements électroniques diminue de façon exponentielle avec la température ; les causes possibles de panne sont l'électromigration, les réactions chimiques, la rupture diélectrique, ou le fluage dans les matériaux de collage [17]. Une augmentation de la température de 10° C à 15°C peut réduire de moitié la vie d'un microprocesseur [125]. Par conséquent, la température d'un système est souvent limitée pour contrôler la dissipation de la chaleur au maximum de puissance, augmenter le MTTF, minimiser la consommation d'énergie, éviter l'autodestruction, ainsi que pour des raisons de sécurité. Les smartphones sont souvent plafonnés thermiquement autour de 50°C, de sorte que les utilisateurs ne se brûlent pas, mais aussi afin d'utiliser efficacement la capacité électrique de la batterie. Soulignons que la température de la surface au contact des systèmes électroniques devrait être limitée à 41°C ou 45°C, en fonction du matériau, afin d'assurer le confort tactile de l'utilisateur [9].

Les équipements électroniques avancés emploient des techniques de gestion de la température (Thermal Management Units (TMUs) ou Dynamic Thermal Management

(DTM)) qui sont capables de ralentir ou adapter les systèmes de façon à ce que leurs contraintes thermiques, parfois strictes, soient remplies. Ces techniques de gestion thermique peuvent être mises en place dès la phase de conception du système ou être dynamiquement déployées au moment de l'exécution. Une pléthore de méthodes de contrôle thermique pour microprocesseurs et systèmes sur une puce (SoC) [8, 60] existent ; elles agissent sur les compromis entre profil de température, paramètres de fréquence, consommation d'énergie et complexité de mise en œuvre [138]. Certains microprocesseurs complexes peuvent employer des TMUs implantées sur le matériel, comme, par exemple, certains microprocesseurs d'Intel [31, 77], tandis que les TMUs logicielles sont fréquemment mises en œuvre dans les systèmes embarqués. Pour utiliser efficacement les TMUs, il est important de comprendre le comportement thermique transitoire du système. Le comportement thermique transitoire d'un système refroidi de manière active peut être fidèlement décrit par une relation exponentielle. Mais, pour les systèmes refroidis passivement, tels que les smartphones, les capteurs sans fil ou encore de nombreux objets participant à l'Internet des objets (IoT), l'hypothèse exponentielle ne tient pas. Les systèmes à refroidissement passif comptent sur le refroidissement par conduction, convection naturelle, et rayonnement et sur la gestion thermique de la TMU ou DTM, qui présentent des propriétés non-linéaires. Dans le chapitre 5, le comportement thermique d'un microprocesseur refroidi passivement est étudié par l'intermédiaire du développement d'une loi de refroidissement passif utilisant la conversion de chaleur interne et les mécanismes de refroidissement convectif et radiatif. Des solutions approchées sont également présentées, car la solution exacte est assez complexe et donne le temps t en fonction de la température T alors qu'il serait naturel d'avoir l'inverse, c'est à dire la temperature en fonction du temps. La performance des solutions approchées est aussi évaluée. L'*approximation des coefficients* se révèle une bonne solution heuristique dans la majorité des cas, avec un minimum d'erreurs en présence de variations de température d'applications embarquées typiques. Pour un objet comme un microprocesseur, il est montré que, pour les grandes surfaces de refroidissement, la loi de refroidissement passive diffère sensiblement d'une loi exponentielle de refroidissement. Pour les surfaces de refroidissement plus petites qu'environ 1 dm^2, la loi de refroidissement passive reste proche d'une loi exponentielle. Pour de tels systèmes, une loi exponentielle de refroidissement, même moins précise, reste avantageuse en termes de mise en œuvre et de complexité de calcul.

En plus de la température, d'autres paramètres influencent le profil de la consommation d'énergie d'un microprocesseur. Les caractéristiques de temps d'exécution et les exigences de puissance du logiciel et du système sont les principaux facteurs qui définissent la consommation finale d'énergie. Ceci est une conséquence directe de la définition de la consommation d'énergie électrique : l'intégrale de la puissance électrique au fil du temps. Le temps d'exécution est influencé par le type et le coût des opérations effectuées par le logiciel en question, y compris les instructions accédant à la mémoire externe et aux registres. Chaque unité fonctionnelle au sein d'un microprocesseur et composant du système a son propre profil de puissance et de temps d'exécution. En conséquence, chaque logiciel a des exigences différentes de puissance. Par exemple, Carroll et Heiser [16] ont montré que, pour leur système embarqué, en exécutant `eQuake`, `vpr`, et `gzip` de SPEC CPU2000, la consommation d'énergie du microprocesseur dépasse la consommation due à la seule mémoire RAM, tandis que c'est l'inverse pour `crafty` et `mcf`, issus de la même suite logicielle. Minimiser le nombre d'opérations d'accès à la mémoire est une technique d'optimisation énergétique courante. Par exemple, Intel a introduit, avec la puce E5 Xeon Data Direct I/O (DDIO), le carte réseau (NIC) Ethernet peut charger des données

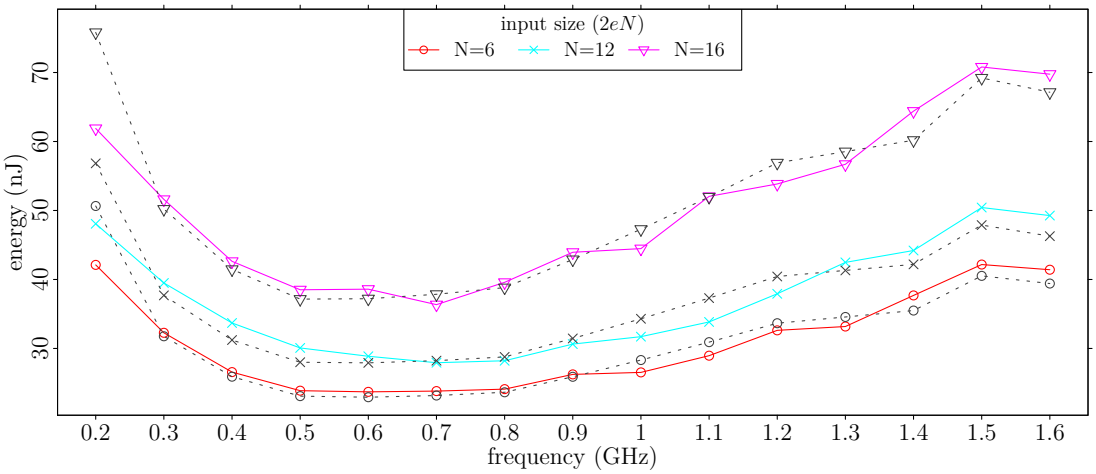


Figure 1: Données expérimentales et théoriques de consommation d'énergie d'un programme s'executant sur les microprocesseurs ARM Cortex A9. La consommation d'énergie pour divers points caractéristiques avec différentes tailles des entrées est représentée pour l'algorithme Rader, ainsi que les repères BEEBS. Les traits pleins représentent les données mesurées, tandis que les lignes en pointillé représentent la courbe d'énergie et de temps d'exécution calculée.

directement dans le cache du microprocesseur [57], minimisant ainsi l'accès a la mémoire mémoire vive (RAM). En évitant les opérations d'entrées-sorties (I/O) , la performance, mais aussi la consommation d'énergie du système, est améliorée. Une caractéristique intéressante de la consommation d'énergie d'une séquence de code, c'est que le produit de son temps d'exécution et de la consommation d'énergie du microprocesseur possède, sous certaines hypothèses, des propriétés convexes, ce qui est discuté dans le chapitre 4; cette loi est appelée loi de convexité énergie-fréquence de la consommation des programmes: elle est illustrée dans la figure 1. Cette règle stipule qu'il existe une fréquence d'horloge pour l'exécution de chaque séquence de code qui minimise la consommation d'énergie de ladite séquence de code. Dans certaines conditions, cette fréquence d'horloge optimale, qui réduit la consommation d'énergie, se trouve entre les fréquences d'horloge minimale et maximale de fonctionne du microprocesseur. Comme on le verra, le choix de la fréquence d'horloge optimale conduit à un compromis entre performance en termes de temps d'exécution et économies d'énergie. Pour les applications nécessitant de nombreuses interactions humaines, il a été montré que la fréquence d'horloge peut être réduite sans affecter considérablement l'expérience de l'utilisateur [101]. Une preuve expérimentale de l'existence de la loi de convexité énergie-fréquence de la consommation des programmes, qui concerne la consommation d'énergie et la fréquence d'horloge des microprocesseurs sur les appareils mobiles, est présentée. La propriété de convexité semble garantir l'existence d'une fréquence optimale où la consommation d'énergie est minimale. Cette affirmation de l'existence d'une telle propriété est fondée sur des preuves à la fois théoriques et pratiques. Les données recueillies par l'intermédiaire de campagnes d'acquisition de donnée sur plusieurs plates-formes suggèrent que l'énergie consommée par élément d'entrée est fortement corrélée avec la fréquence d'horloge du microprocesseur et, ce qui est encore plus intéressant, que la courbe correspondante présente un minimum clair sur une fenêtre de fréquences spécifique au système informatique et au programme. Un modèle analytique de ce comportement est aussi motivé, qui cadre bien avec les données présentées. Une analyse de sensibilité des paramètres est réalisée afin d'évaluer l'influence des paramètres sur la

fréquence optimale. Il est également montré que la fréquence optimale augmente lorsque les besoins en puissance du système, à l'exclusion des microprocesseurs, augmentent. La présence de cycles d'horloge perdus pour la gestion propre du système accroît également la fréquence optimale. La fréquence optimale telle que dérivée du cadre théorique présenté ici est cependant indépendante du nombre d'instructions à exécuter.

Dans le chapitre 6, la loi de convexité énergie-fréquence de la consommation des programmes, comme le montre bien la Figure 1, est utilisée dans des applications pratiques. On montre comment on peut calculer les fréquences optimales minimisant la consommation d'énergie du système pour des systèmes intégrant plusieurs entités ayant chacune leur propre fréquence d'horloge, ce qui correspond, en substance, à une superposition de plusieurs systèmes unitaires. La loi de convexité énergie-fréquence de la consommation des programmes est également appliquée à un système informatique soumis à des contraintes temporelles répétitives et à l'adaptation de la fréquence d'un système multithread. Il est ainsi montré que des économies d'énergie de l'ordre de 30 % sont réalisables dans ces cas. Dans le prolongement de la loi d'Amdahl, on analyse le rendement d'énergie d'un système informatique hétérogène. Cette mesure de performance met en lumière les avantages des systèmes à haute performance ne nécessitant qu'une faible puissance.

Contributions

Ce travail fournit une meilleure compréhension à la fois théorique et pratique de la consommation d'énergie d'un microprocesseur. L'analyse du comportement thermique transitoire et les gains d'énergie via l'adaptation de la fréquence d'horloge sont d'un intérêt tout particulier. Les principales contributions sont :

- un cadre théorique pour la loi de convexité énergie-fréquence de la consommation des programmes et des mesures expérimentales – une analyse de sensibilité de la loi de convexité énergie-fréquence de la consommation des programmes est effectuée pour estimer l'impact des multiples paramètres;
- une loi de refroidissement exacte pour un objet isotherme soumis au rayonnement, à la convection, et à la génération de chaleur interne – la loi exacte est appliquée à un objet de type microprocesseur en vue d'évaluer son comportement transitoire;
- des modèles température / puissance appliqués à l'Exynos 5410 – ces modèles sont utiles pour annuler le biais de température dans les mesures de puissance et d'augmenter la précision de celles-ci;
- des modèles de puissance explicites pour l'Exynos 5410 système sur une puce (SoC) intégrant la fréquence d'horloge, la température, et le nombre de cœurs actifs – ces modèles peuvent être directement réutilisés dans toute simulation pour d'autres processeurs d'architecture similaire;
- plusieurs approximations à la loi de refroidissement exacte – ces approximations sont plus simples et plus convenablement formulées pour une utilisation pratique;
- des méthodes et lignes directrices concrètes pour améliorer la précision et la pertinence des mesures d'énergie et de puissance;
- des règles générales pour évaluer quand le refroidissement passif devient non négligeable par rapport à un refroidissement actif dans les systèmes embarqués;

- et la mise en œuvre sous forme de prototypes de tous ces modèles de refroidissement – les lois exactes et approximatives pour les objets refroidis passivement sont fournies avec une évaluation de la precision perdues pour les approximations qui d'autres part, permetent un calcul rapide (utilisable dans des applications temps réel).

Résumé des chapitres

Généralités sur la consommation d'énergie électrique et le transfert de chaleur

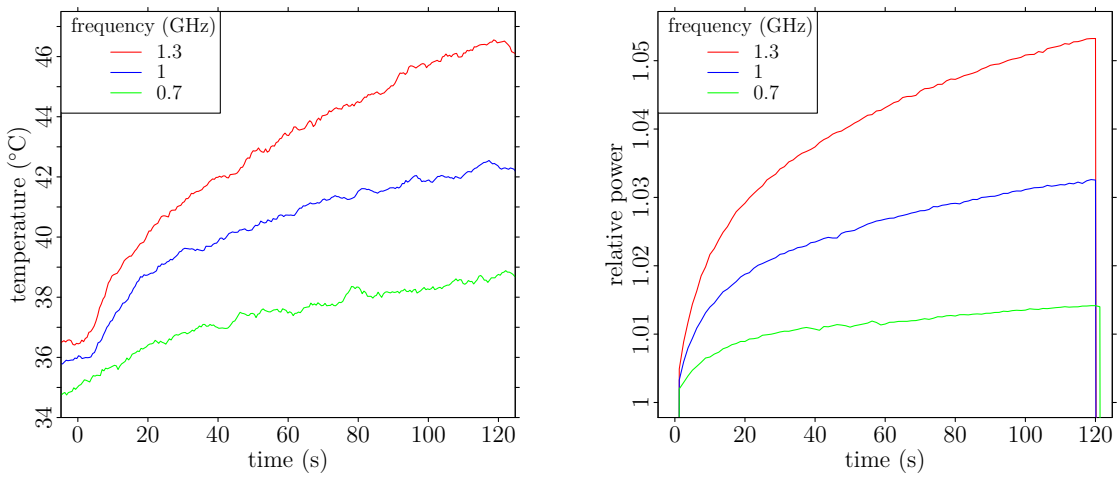
Les concepts fondamentaux sur la *consommation d'énergie électrique* et le *transfert de chaleur* sont rappelés dans ce chapitre. Pour comprendre et modeliser la consommation d'énergie électrique, les différentes sources de consommation d'énergie lors de l'exécution d'un programme sont prises en compte. Les modèles de temps d'exécution sont également nécessaires. Au total, ces modèles permettront d'étudier la consommation d'énergie des différents types d'applications dans les chapitres suivants. La température joue un rôle majeur dans la consommation d'énergie lors de l'exécution d'un programme. De ce fait, des modèles de transfert de chaleur sont étudiés ; ils seront utilisés pour comprendre le comportement thermique transitoire d'un microprocesseur.

La section 2.1 présente les notions de consommation d'énergie électrique et de profil de puissance dans le cadre des systèmes informatiques. Les différentes sources de consommation d'énergie dans un microprocesseur sont mises en évidence. Dans la section 2.2, des modèles de temps d'exécution d'un programme sont utilisés conjointent au modèles énergétiques . Des modèles thermiques seront utilisés pour comprendre l'interaction entre l'énergie et la température ; donc la section 2.3 présente un résumé des principes thermiques, y compris les modes de transfert de chaleur et les propriétés thermiques des matériaux.

La relation de température / puissance dans les microprocesseurs

Lorsqu'on étudie la consommation d'énergie sur la base de mesures de puissance, il est impératif d'obtenir des échantillons de puissance précis et des traces de mesures reproductibles. En effet, la température a un impact clair sur les mesures de puissance des microprocesseurs. L'exemple de la figure 2 montre une augmentation de la consommation d'énergie de 5% pour une augmentation de température de 10°C. Auparavant, il avait été également souligné que les courants de fuite sont dépendants de la température, et une équation avait été introduite pour en fournir une approximation. Par conséquent, la puissance d'un microprocesseur est également dépendante de la température et la consommation d'énergie du microprocesseur en est affectée.

La relation entre la consommation d'énergie et la température est ici démystifiée. Les modèles décrits dans la littérature sont remis en question et des traces de mesures expérimentales sont utilisées pour valider certains de ces modèles. L'utilisation pratique des modèles de température / puissance est illustrée avec le but d'améliorer la précision des mesures. L'importance de l'emplacement de la sonde de température par rapport aux sources de chaleur est mise en évidence quantitativement. Un autre modèle de transformation est avancé, qui prend en compte l'emplacement et le comportement des capteurs de température. Quand un capteur de température est placé à distance à partir d'une source de chaleur, le capteur mesure la température avec un retard et avec une amplitude



(a) température (b) puissance

Figure 2: Température (a) et puissance (b) d'un Exynos 4210 en vertue de la charge de travail constante, opérationnel à différentes fréquences d'horloge microprocesseur. On observe que la consommation d'énergie augmente rapidement due à l'augmentation de la température croissante du microprocesseur, étant donné que tous les autres paramètres ont été maintenus constants [28].

diminuée. Ce chapitre traite ainsi de la transformation des traces de puissance, en mettant l'accent sur la température du système, pour arriver à des échantillons de puissance significatifs et reproductibles.

La loi de convexité énergie-fréquence de la consommation des programmes

Dans le chapitre 2, des modèles de puissance et de temps d'exécution ont été présentés, lesquels, quand ils sont combinés, produisent un modèle de consommation d'énergie du microprocesseur, selon équation 2.2. Dans ce chapitre, le comportement de ce modèle de consommation d'énergie est étudié via une analyse de sensibilité en fonction d'une selection de parametres et sa validité est montrée en pratique grâce à des mesures expérimentales de temps d'exécution et de puissance.

Le chapitre commence par un large état de l'art présentant des courbes d'énergie / fréquence trouvées dans la littérature. La loi de convexité énergie-fréquence de la consommation des programmes pour un microprocesseur à cœur unique est développée dans la section 4.2. Ensuite, dans la section 4,3, des mesures expérimentales de la puissance et du temps d'exécution de deux processeurs sont présentées ; elles montrent l'existence de la convexité énergie / fréquence. Ces modèles fondés sur des campagnes de mesures seront également utilisés dans les chapitres suivants. La section 4.4 analyse, quant à elle, la sensibilité des paramètres de la loi de convexité énergie-fréquence de la consommation des programmes.

Le refroidissement passif des microprocesseurs

Une solution analytique exacte du transfert de chaleur au sein d'un microprocesseur refroidi passivement est développée dans ce chapitre, sous les hypothèses de présence de

convection, rayonnement et génération de chaleur interne. Ces modèles seront utiles aux Thermal Management Units (TMUs) et Dynamic Thermal Management (DTM) pour évaluer et prédire le comportement thermique d'un microprocesseur. Alors qu'on a auparavant étudié la dépendance de la température sur la puissance du microprocesseur, on se penche ici sur le comportement thermique transitoire. Analyser le comportement thermique exact d'un microprocesseur est intractable en raison de la nature complexe de la physique impliquée dans un circuit intégré y compris la dynamique des fluides, tous les modes de transfert de chaleur, les configurations des composants électroniques complexes, la turbulence, etc. Pour maitriser cette complexité, des hypothèses de simplification sont formulées, par example, ou représente un microprocesseur, par une pièce isotherme d'oxyde de silice qui est placée dans un espace ouvert infini soumis à la convection (naturelle) et au rayonnement. Ces hypothèses sont formulées supposées lorsqu'il s'agit de prendre en compte la loi de refroidissement de Newton.

Tout d'abord, dans la section 5.2, le comportement temporel de la température d'un microprocesseur est considéré, en partant de la loi générale de refroidissement de Newton et la génération de chaleur interne. Ensuite, dans la section 5.3, l'impact de la radiation enrichit notre modèle, pour développer une équation de chaleur modélisant le refroidissement passif d'un microprocesseur avec une génération de chaleur interne. La différence de performance entre les lois de refroidissement passif et actif est également évaluée dans la section 5.4. Par ailleurs, des approximations de la solution exacte de la loi de refroidissement passif sont proposées dans la section 5.3.6, cette dernière se révélant être une équation assez complexe. L'emballement thermique de l'équation de la chaleur passive appliquée à un modèle de microprocesseur réaliste est également étudié.

Applications aux domaines multi-horloge de la consommation optimale d'énergie

La loi de convexité énergie-fréquence de la consommation des programmes du chapitre 4 est appliquée à des cas d'usage spécifiques dans ce chapitre. Les systèmes d'agents coopérants sont étudiés dans la section 6.1, et une solution analytique est développée pour trouver la fréquence d'horloge optimale pour tous les participants, dans le but de minimiser la consommation énergétique globale de l'ensemble du système. Il est démontré que, pour trouver la fréquence optimale pour chaque participant, la règle de convexité doit être appliquée de nouveau chaque fois que le système change son état d'énergie. Dans la section 6.2, la règle de convexité est appliquée à un système soumis à des contraintes temporelles. Les résultats montrent que l'on peut économiser jusqu'à 55% de l'énergie, dans le meilleur des cas. Mais ces économies peuvent devenir marginales lorsque beaucoup de travail doit être effectué avant la date limite. La section 6.3 examine si l'application de la règle de convexité peut conduire à économiser de l'énergie par ajustement de la fréquence d'horloge de threads parallèles individuels. Il sera montré que, pour une consommation d'énergie de fond comparable à la consommation d'énergie du microprocesseur, des économies d'énergie entre 3% et 10% par rapport à la consommation énergétique induite par le gestionnaire de fréquence d'horloge typiquement mis en œuvre par Linux sont possibles. La section 6.4 intègre la notion d'énergie dans la loi d'Amdahl pour analyser la *performance par Joule* d'un microprocesseur. On montre comment la fréquence d'horloge qui minimise la consommation d'énergie se comporte sous la loi d'Amdahl. Dans la section 6.5, on s'attache à démystifier les relations qu'on suppose traditionnellement entre calcul hétérogène et énergie. Le contraste entre le comportement d'un microprocesseur

basse-puissance et un autre à haute performance est mis en évidence en utilisant la loi de convexité énergie-fréquence de la consommation des programmes et la loi de Amdahl. Une taxonomie spécifique est introduite pour distinguer les différentes facettes de la notion de puissance.

Conclusions

Résumé des résultats

Dans ce travail, la loi de convexité énergie-fréquence de la consommation des programmes est établie analytiquement et validée par une vast bibliographie ainsi que des mesures expérimentales. La thermosensibilité de la loi de convexité énergie-fréquence de la consommation des programmes et le profil de puissance des microprocesseurs ont également été discutés. En outre, des procédés pour améliorer la précision des mesures de l'énergie et de la puissance ont été présentés.

La loi de convexité énergie-fréquence de la consommation des programmes s'applique à un système informatique dont la consommation d'énergie E_{sys} peut être décrite par:

$$E_{\text{sys}} = \Big((1+\gamma V)\,\xi f V^2 + P_{\text{back}}\Big)\cdot cc_{\text{b}}\left(\frac{1}{f-f_{\text{k}}}+\beta\right), \tag{4.2}$$

où : γ est un paramètre associé aux courants de fuite ; V est la tension du processeur ; ξ est la demande de puissance du processeur ; f est la fréquence d'horloge du processeur ; P_{back} est la demande de puissance du système, excepté celle du processeur ; f_{k} sont les *voleurs de temps* (temps d'attente typiquement sur des entrées/sorties); et β est le *slack time* du processeur (du au fonctionnement du système).

$$E_{\text{n}} = E_{\text{n}}^{\text{A}} + E_{\text{n}}^{\text{B}}$$

$$\frac{\partial E_{\text{n}}^{\text{A}}}{\partial f} = (4af^3+3bf^2+2cf+d)\cdot\beta \tag{11.a}$$

$$\frac{\partial E_{\text{n}}^{\text{B}}}{\partial f} = \frac{3af^4+(2b-4a\,f_{\text{k}})f^3+(c-3b\,f_{\text{k}})f^2+2c\,f_{\text{k}}f+(P_{\text{back}}-d\,f_{\text{k}})}{(f-f_{\text{k}})^2} \tag{11.b}$$

$$\frac{\partial^2 E_{\text{n}}^{\text{A}}}{\partial f^2} = (12af^2+6bf+2c)\cdot\beta \tag{11.c}$$

$$\frac{\partial^2 E_{\text{n}}^{\text{B}}}{\partial f^2} = \frac{-6af^4+(16a\,f_{\text{k}}-2bc\,f_{\text{k}}^2)f^3+(6b\,f_{\text{k}}-12a\,f_{\text{k}}^2)f^2}{(f-f_{\text{k}})^3} - \frac{6b\,f_{\text{k}}^2 f+2(d\,f_{\text{k}}+P_{\text{back}})}{(f-f_{\text{k}})^3} \tag{11.d}$$

Si la dérivée première de l'énergie E_{n} a une seule racine sur $\mathbb{R}^+$ et la dérivée seconde est une fonction croissante monotone, alors la consommation d'énergie du système présente un minimum global avec à une valeur optimale de la fréquence d'horloge f_{opt}. Par définition, cette fréquence d'horloge optimale donne la plus petite consommation d'énergie pour l'exécution d'une séquence d'instructions. Si la fréquence d'horloge optimale se situe dans la plage des fréquences d'horloge admissibles du microprocesseur, elle peut être exploitée. Dans le cas contraire, le microprocesseur est le plus économe en énergie pour une fréquence d'horloge se trouvant en limite de plage : la fréquence maximale ou minimale de l'horloge. La consommation d'énergie est une fonction de plusieurs paramètres: les demandes en

puissance du microprocesseur (ξ), les capacités d'adaptation tension / fréquence (f/V), la consommation de base (P_{back}), le slack time (β), les voleurs de cycles d'horloge (f_{k}), la température T, et quelques autres.

Dans le chapitre 4, une analyse de sensibilité des paramètres a été effectuée. La consommation d'arrière-plan P_{back} s'avère être le paramètre ayant la plus grande influence sur la fréquence optimale d'horloge, celle qui réduit au maximum la consommation d'énergie du système. En règle générale, f_{opt} est exploitable, c'est-à-dire $f_{\min} < f_{\text{opt}} < f_{\max}$, si les besoins en puissance du microprocesseur sont plus petits que P_{back}. Pour certains types d'applications, cependant, f_{opt} est indépendante de P_{back}, par exemple pour le traitement de tâches répétitives au sein des systèmes informatiques, comme expliqué dans la section 6.2. Le nombre de voleurs de cycle d'horloge affecte également f_{opt} de manière significative, dans le sens que f_{opt} augmente quand moins de cycles d'horloge sont disponibles pour le calcul proprement dit. On a également montré que le profil de puissance du microprocesseur ou encore le nombre d'instructions exécutées n'ont pas d'influence significative sur la fréquence d'horloge optimale, ce qui est une conséquence directe de la formulation de la consommation d'énergie donnée par l'équation 4.2, formule qui a également expliqué que la technique d'optimisation de l'énergie *race-to-halt* est seulement efficace quand $f_{\text{opt}} > f_{\max}$.

Le chapitre 6 applique la loi de convexité énergie-fréquence de la consommation des programmes à des applications pratiques. Il y est montré comment la loi de convexité énergie-fréquence de la consommation des programmes pouvait être appliquée à un système informatique composé d'entités multiples, qu'on appelle parfois "amis", disposant de fréquences d'horloge indépendantes. L'énergie peut être optimisée à un niveau global en divisant le traitement effectué par le système en étapes temporelles dans lesquelles les états d'énergie des amis ne changent pas dans une étape donnée. Ensuite, la fréquence optimale pour chaque ami dans chaque étape peut être calculée. Il a été montré que, dans le cas d'exécution dans le désordre (OOE), la détermination de f_{opt} peut devenir compliquée, parce que les fréquences d'horloge des amis ne peuvent pas être modifiées indépendamment les unes des autres pendant l'exécution. Une exemple d'un système à quatre amis a été présentée dans la section 6.3, où les fréquences d'horloge des quatre threads ont été modifiées de façon indépendante et en collaboration pour atteindre un système caractérisé par une consommation d'énergie minimale. Par rapport au controleur par défaut de la fréquence d'horloge implementé par Linux, un algorithme coopératif des fréquences d'horloge entre amis a montré qu'un gain en énergie allant jusqu'à 40 % pouvait être obtenu dans les conditions les plus favorables. Cependant, le gain d'énergie tombe à 10 % lorsque les autres composants du système ont besoin approximativement de la même puissance que le microprocesseur. En fait, ce gain d'énergie représente cependant un compromis avec le temps d'exécution, qui est environ deux fois plus important quand on gagne 40 % en énergie. La loi d'Amdahl a également été étendue pour intégrer le réglage de la fréquence d'horloge, loi à partir de laquelle la *métrique de performance par Joule* a été déduite. Cette métrique a été appliquée à l'Exynos 5410 SoC, qui abrite un microprocesseur *à faible consommation* (A7) et un micro processeur à haute performance (A15). La performance par Joule la plus convenable a été montrée être dix fois plus élevée pour l'A7 que pour l'A15, alors que le temps d'exécution n'augmente seulement qu'environ 5 fois pour l'A7 par rapport à l'A15.

Ces observations de la loi de convexité énergie-fréquence de la consommation des programmes et ses applications ont à la fois été fondées sur un cadre théorique novateur et étayées par des données expérimentales. Pour obtenir ces données expérimentales,

des profils de temps d'exécution ont été tirés de Bristol Energy Efficiency Benchmark Suite (BEEBS) et de l'algorithme *bit-reverse* de Golden-Rader, ces programmes ayant été exécutés sur deux plates-formes constituées de SoCs multimédia. Les profils de puissance utilisés ont été enregistrés sur les mêmes SoCs, dont un est composé d'un Cortex A9 et l'autre d'un double microprocesseur Cortex A7 et A15. Des modèles de puissance pour processeurs A7 et A15 ont été fournis dans la section 3.6 ; ces modèles logiciels peuvent être utilisés directement dans des simulations. Ce qui se révèle comme particulièrement intéressant à propos de ces modèles de puissance P, c'est que la température T du microprocesseur est un paramètre d'entrée. Dans le chapitre 3, la dépendance de la température sur les caractéristiques de puissance d'un microprocesseur a été discutée. Outre d'autres propriétés physiques, les courants de fuite sont les plus grands contributeurs à cette dépendance de la puissance du microprocesseur de la température, ce qui est largement discuté dans la section 3.2.1. Il est montré par des données empiriques tirées du processeur A15 que, dans le cas le plus extrême, les besoins en énergie à 85 °C sont 20 % plus élevés qu'à 25°C. Un modèle fondé sur une loi exponentielle

$$P = a_1 e^{T/a_2} + a_0,$$

s'avère modéliser fidèlement la relation température / puissance dans l'intervalle allant de 20°C à 85°C. Pour une plage de variations de température plus pertinente, entre 20 °C et 50°C, des approximations linéaire et du second degré sont acceptables. Cette relation température / puissance est utilisée pour montrer comment supprimer le biais de la température à une trace de mesure de puissance. Il est également indiqué et illustré que la distance d'un capteur de température à partir d'une source de chaleur peut introduire des erreurs sur les mesures de puissance, comme un décalage dans le temps et une amplitude réduite. Une fonction de transformation, pratique mais un peu ad-hoc, a été proposée pour annuler un tel comportement.

Le comportement thermique transitoire d'un microprocesseur est influencé par la relation température / puissance, en ce que la génération de chaleur interne dépend de la température. Le comportement thermique transitoire est également dicté par la façon dont le microprocesseur dissipe sa chaleur dans l'environnement. Les systèmes refroidis activement libèrent leur chaleur dans l'environnement par convection forcée, par exemple de l'air ou d'autres types de fluide, ce qui domine les autres modes de transfert de chaleur. Le comportement transitoire de ces systèmes est bien décrit par une loi de refroidissement exponentielle. Les dispositifs refroidis passivement, cependant, comptent sur la dissipation naturelle de la chaleur, y compris par rayonnement. Le chapitre 5 explore les effets de la présence de refroidissement radiatif sur le comportement thermique transitoire par l'intermédiaire d'un cadre analytique exact. La loi de refroidissement en présence de radiation s'exprime de la façon suivante :

$$t = -\frac{1}{\kappa_4}\Bigg(A\ln|T-\omega_1| + B\ln|T-\omega_2| + \frac{C}{2}\ln|(T-\alpha)^2+\beta^2| + \frac{\alpha C + D}{\beta}\arctan\left(\frac{T-\alpha}{\beta}\right) + c_o\Bigg). \quad (5.17)$$

Un cas d'utilisation appliqué à un objet ressemblant un microprocesseur montre que la composante liée au rayonnement ne peut être négligée pour les objets offrant une surface de refroidissement plus grande que $1\,\mathrm{dm}^2$, ce qui pourrait être une surface de refroidissement typique d'un smartphone. Pour les surfaces de refroidissement plus petites, la loi de

refroidissement passive exacte s'approche d'une loi de refroidissement exponentielle. Dans ces conditions, une loi exponentielle de refroidissement doit être favorisée, car elle est beaucoup moins complexe que la loi de refroidissement passif exacte. Des approximations à la loi de refroidissement exacte ont également été fournies dans la section 5.3.6 ; elles sont destinées à être utilisées dans des applications pratiques. On a montré que la loi dit de *rapprochement des coefficients*

$$T = \frac{\sqrt{\kappa_1^2 - 4\kappa_2\kappa_0}(1 - c_o e^{-\frac{\kappa_2}{A}t})}{2\kappa_2(1 + c_o e^{-\frac{\kappa_2}{A}t})} - \frac{\kappa_1}{2\kappa_2}, \tag{5.28}$$

fonctionne le mieux dans la plupart des circonstances. Pour les petits écarts à la température ambiante, la méthode dite de *rapprochement de second ordre d'O'Sullivan*

$$T = \frac{w}{2m}\tanh\left(\frac{w}{2C}t + c_o\right) - \frac{n}{2m} + T_a, \tag{5.36}$$

peut aussi être utilisée de manière satisfaisante. Les taux d'erreur sont à chaque fois calculés.

Travaux Futurs

Même si aucune question ouverte n'est formulée explicitement au long la recherche présentée, il semble très intéressant de poursuivre plusieurs pistes de réflexion possibles découlant de ce travail.

La précision du protocole de mesure de la relation température / puissance peut être améliorée, principalement en utilisant plusieurs capteurs de température plus précis. C'est une tâche difficile, car les capteurs de température présents sur les circuits ont une résolution faible et peuvent conduire à des taux d'erreur importans. Les dispositifs extérieurs de mesure de la température, tels que les capteurs à base d'infrarouge, peuvent fournir une précision améliorée, mais ils doivent être utilisés dans un environnement contrôlé et de préférence avec le circuit exposé. La température ambiante et la température du circuit doivent également être contrôlées de façon à éviter les effets d'*hystérésis* de température pouvant interférer avec la corrélation température / puissance. En particulier, l'hypothèse d'isothermie doit être valider.

D'un point de vue théorique, il pourrait être intéressant d'évaluer le facteur d'impact exact des processus qui dépendent de la température, en supplément du courant de fuite, et qui affectent le profil de puissance des microprocesseurs. En utilisant ces connaissances, un modèle plus complet de la relation température / puissance pour microprocesseur pourrait être construit, fondé sur des principes physiques, en complément du modèle température / puissance heuristique présenté auparavant. En outre, le modèle de transformation permettant de prendre en compte le *syndrome de capteur distant* pourrait être déduit d'une analyse théorique plus solide, en remplacement d'un polynôme approximatif, bien que, comme on l'a montré, un tel effort conduirait probablement à une formulation mathématique très complexe qui pourrait être insurmontable pour des applications pratiques.

Du point de vue de la mesure expérimentale de la puissance, des données plus précises seraient également avantageuses pour affiner le modèle de température / puissance. Un taux d'échantillonnage plus élevé permettrait d'observer plus en détail des parties spécifiques des séquences d'instructions. Cela mènerait à une estimation de ξ avec un grain plus fin,

ce qui pourrait être bénéfique pour obtenir des informations ξ pour les unités fonctionnelles du microprocesseur, meilleures que l'estimation du niveau d'application qui a été supposé dans cette thèse. Cependant, des taux d'échantillonnage plus élevés induisent des exigences de traitement post-mesure de données plus grandes et plus coûteuses. Dans cette thèse, il a déjà été parfois fastidieux de traiter des traces de mesures de puissance à 4 kHz. Des outils d'acquisition de données haut de gamme peuvent avoir des taux d'échantillonnage de 100 kHz à 1 GHz; plus d'espace disque et de patience seront donc nécessaires pour analyser des traces de puissance plus riches.

Des profils de ξ plus précis peuvent alors également conduire à une optimisation de la gestion dynamique de la tension et de la fréquence (DVFS) plus agressive, ce qui donnerait des gains énergétiques bien supérieurs à ceux fournis par les gouverneurs interactifs de fréquence présents dans les distributions Linux. Les profils de ξ et de temps d'exécution pourraient également être intégrés dans le code comme directives pour une optimisation dynamique de l'énergie. Il pourrait être difficile, pour les architectures matérielles actuelles, d'étendre leur traduction de code binaire afin de prendre en charge les profils d'énergie et de temps d'exécution. Toutefois, si ce type d'acquisition de données était incorporé dans le code binaire, un processus de machines virtuelles pourrait décoder ces informations et rediriger les données vers une unité de gestion de l'énergie. Même les informations de profil énergétique issues du code binaire, récoltées via une interface native de Machine virtuelle (VM), pourraient être transmises, de manière utile, aux unités de gestion de l'énergie. Comme indiqué dans cette thèse, les économies d'énergie maximales sont atteintes par la coopération de toutes les applications, pas nécessairement via l'optimisation de chaque application individuelle. Une unité de gestion de l'énergie possédant tous les paramètres ξ des VMs actives pourrait ainsi optimiser le système de manière coopérative, par exemple grâce à la méthode *amis* et le DVFS présentés dans ces travaux. Une coopération basique au niveau application existe déjà dans Android via ce qui est appelé *wakelocks*, une technique qui gère les dépenses d'énergie des ressources du système, comme par exemple l'écran et les capteurs.

La loi de convexité énergie-fréquence de la consommation des programmes a été étudiée en adoptant un point de vue théorique, sans prendre en compte la notion de ressenti humain. Pour des systèmes HPC qui doivent produire des résultats aussi rapidement que possible, les ressentis humains ne sont d'aucune valeur. Par contre, les dispositifs conçus avec une interactions homme-machine (HCI) riche, tels que les smartphones, ou tablettes se doivent absolument de prendre en compte les aspects émotionnels humains dans l'utilisation de la loi de convexité énergie-fréquence de la consommation des programmes. Comme il a été présenté dans ce travail, la loi de convexité énergie-fréquence de la consommation des programmes est un compromis entre la consommation d'énergie, d'un côté, et le temps d'exécution, de l'autre. Si une HCI est impliquée, l'expérience utilisateur sera potentiellement affectée par un changement trop grand de temps d'exécution. Lorsque le temps d'exécution augmente, le système informatique devient moins réactif et l'expérience de l'utilisateur va se détériorer, de façon non-linéaire. Comprendre l'impact émotionnel de la loi de convexité énergie-fréquence de la consommation des programmes devrait donc être un facteur clé lors de la conception d'un système optimisé pour en fonction de la qualité de l'expérience humaine. L'expérience de l'utilisateur peut alors être utilisée comme une contrainte supplémentaire pour définir la fréquence d'horloge optimale. Cette contrainte induirait une limite inférieure de la fréquence d'horloge qui indiquerait le moment à partir duquel l'utilisateur ne tolérerait plus aucun ralentissement du système.

Chapter 1

Introduction

Even a tiny amount of energy gained via a better understanding of frequency scheduling and thermal effects on energy behavior of microprocessors may have significant monetary and ecological impacts. Understanding and accurately modeling these relationships may bear impact beyond optimized system operation management. This stresses the crucial aspect of energy optimization in microprocessors, especially for embedded and High Performance Computing (HPC) systems. This point is particularly acute for system running on an electrical battery, such as smartphones, sensors etc. Whatever their origin, all hand-held devices share the same Achilles' heel: the *battery* [123]. HPC systems on the other hand are, amongst others, constrained by the available power supply.

The service uptime of battery-powered devices is a sensitive issue for nearly any end-user or operator. Battery life became an important user experience factor, as has been shown by academic studies [37] as well as by surveys conducted by retail information channels[1] since the dawn of portable devices. Even though battery capacity and performance are hoped to increase steadily over time, improving the energy efficiency of current battery-powered systems is essential because power demands are outpacing the developments in battery capacities [94, 117]. Supercomputers, on the other hand, are limited by the available power supplied from the grid feeding the data center. Hence their computers must operate within an available power and financial budget. Moreover, user experience, product and circuit design are constrained by the peak-power heat dissipation of electrical components. Understanding the energy consumption of the different aspects of (battery-powered) computer systems is thus a key issue. The processor's frequency, execution time and temperature are, amongst others, important factors influencing energy and power consumption of microprocessors while executing programs. Providing models for energy consumption can pave the way for energy optimization at the hardware and software layer.

Historically, performance in terms of execution time has been the main performance metric at the software and hardware layers. Nowadays, energy efficiency and power dissipation become important performance metrics as well, especially for battery-powered devices. Execution time remains a performance metric for systems that require sustained performance, such as super computers. But for systems that are build for human-computer interaction (HCI), sustained power is not necessarily an objective; instead, responsiveness is of key importance. Such different approaches to computing may require different methodologies in energy optimization. For example, sporadic bursts of power dissipation

[1] Battery life concerns mobile users: http://www.cnn.com/2005/TECH/ptech/09/22/phone

exceeding the thermal design power (TDP) can be harmless for hardware and furnish support for a responsive system [90].

Improved Integrated Circuit (IC) technologies have been expected to provide a doubling of the number of transistors on chip about every 18 months, presumably following *Moore's Law*. Dennard's law, stating that the power density of scaled technology roughly stays the same, however, stalled a decade ago. The confluence of Moore's law and the stall of Dennard scaling implies that improved technology comes along with increased power densities. Hence, bounding the microprocessor's power dissipation and draining its heat flow efficiently are critical issues for future technologies. Contemporary and future technologies, in extreme cases, can only enable a subset of the microprocessor's logic at a given moment to meet the maximum power dissipation levels and the power budget of the microprocessor. This fraction of disabled logic is referred to as *dark silicon*. Moreover, physical power and wire-delays limits have let to a stagnating clock frequency for current and future technologies [33]. As a result, to boost performance, parallelism is nowadays encouraged in microprocessors. Simultaneous multithreading (SMT) on multicores has delivered substantial energy savings for recent hardware and for in-order processors [34].

Besides optimizing hardware for energy efficiency, the software layer can also be enhanced to minimize the energy consumption and power requirements of the hardware. For embedded systems six different facets of energy optimization can be identified at the software layer: energy-aware operating systems, efficient resource management, impact of the users' interaction pattern with mobile devices and applications, wireless interfaces and sensor management, and benefiting from cloud computing services [117]. All facets could be optimized individually, but rather should be optimized cooperatively to attain the largest energy gain system-wide. This is a grand and not straightforward task. At the software level, energy and power optimization techniques can grossly be divided in *off-line* and *on-line* methods. Off-line methods include, amongst others, efficient software design and coding, and compiler optimization. For these practices to optimize for energy and power, an energy profile of the hardware, on which the software is expected to run, is needed. This comes with the drawback that the software will be energy or power optimal for a specific hardware configuration. Off-line, however, the hardware specification may not be known completely and context information may not always be available, e.g., at the development stage. Moreover, energy-efficient architecture design is very sensitive to the workload [34]. This imposes some uncertainty on the final effectiveness of off-line optimization techniques.

On-line methods, on the other hand, have the advantage that context information is available and hardware specifications are well defined. With this additional information, if accessible, the system and software can dynamically adapt to its context to improve its energy and power efficiency dynamically. On-line methods include, but are not limited to: dynamic compiler optimization, byte-code optimization, and system-level optimization techniques. As for off-line optimization, knowledge of energy profiles of hardware and software helps to improve the decision making of dynamic energy and power optimization techniques. The focus of this thesis is on energy profiles, more specifically, on how the microprocessor's clock frequency and temperature affects the energy consumption and optimal operation conditions of a computer system.

1.1 Summary

Since the dawn of microprocessors it is known that the currents flowing through a microprocessor, or IC, produce a heat dissipation proportional to I^2R, where R is the resistance and I the electric current. The first law of thermodynamics states that in steady operation the energy input of a system is equal to the energy output of the system. Thus in the absence of other energy interactions, and neglecting the energy related information itself, the only form of energy leaving a microprocessor is heat generated by currents flowing through resistive elements [17]. Therefore the microprocessor's heat dissipation is very close to its power consumption. The microprocessor's *internal heat generation* process is sometimes also referred to as *internal heat conversion*. A microprocessor exhibits transient thermal behavior, where the time frame of cooling and heating depends on its heat capacity. The transient thermal behavior is more elongated for systems with larger heat capacities than systems with smaller heat capacities. Such systems can be thought of as RC-circuits based on the current/thermal equivalence (see Section 2.3.3), i.e., a low-pass filter, where the temperature of the system is proportional to the voltage over the capacitors [27].

In reality the (transient) thermal behavior is more complicated, because the resistance R and the current I are known to depend on the microprocessor's temperature. Generally speaking, it is shown experimentally that the power dissipation grows super-linearly with the temperature of the system. In Chapter 3 this temperature/power relationship is discussed in detail. In particular, the thermal behavior of three application processors for embedded systems are discussed. It will be shown that the temperature/power relationship is well described by an exponential law. Within small temperature ranges, as frequently occurring in embedded systems, linear and quadratic approximations are adequate, as shown in Section 3.6. Also, power transformation models are presented that can remove temperature biases from measurement traces, and reduce the effects of time-lag and magnification of a distant temperature sensor. Such power and transformation models are interesting for the following reasons:

- Measurements are essential for the understanding and optimization of energy-aware systems [34, 117]. A lack of detailed power measurements is impairing efforts to reduce energy consumption on modern software [34].

- Previous research tried to describe the temperature/power relationship by focusing on a subset of leakage currents described by BSIM[2] [36, 68, 69, 108, 114]. Such models assumed the leakage current models to be only temperature-dependent. The currents, however, are also time-dependent as the terminal voltages applied to the transistors change over time, which introduces another complexity to the calculations. The leakage currents depend on a plethora of factors specific to every microprocessor, which may not be known exactly. Therefore it may be interesting to understand the aggregated behavior of the temperature/power relationship as an alternative approach to the BSIM derived models.

- Moreover, it is vital to understand the impact of the error in power measurements due to different temperature regimes of the system under study. For instance, exemplary power traces show up to 10% deviation because of transient temperature

[2] BSIM contains MOSFET transistor models for integrated circuit design [70].

effects [28]. Thus, if power measurements are to be compared, temperature effects need to be canceled out to obtain a fair basis for analysis.

- From a measurement perspective it is also important to understand the implications of the internal heat generation's temperature dependency, and the effect of remote sensors. The reproducibility of accurate power measurements are challenging by virtue of the electrical components' transient thermal behavior. More specifically, for a fixed benchmark a microprocessor power measurement yields different values at various microprocessor temperatures. For the sake of accuracy and fair comparison between different power measurements it is thus of vital importance to control or cancel the effects of transient thermal behavior.

The temperature dependency of the microprocessor's power requirements inducing elevated temperatures has an adverse affect on its lifespan. The reliability of electronic products is influenced in terms of spatial or temporal gradients, and absolute temperatures [63]. Thermal gradients that occur both in space and time, induced by the variability in microprocessor load and operations, generate thermal cycles that have a negative affect on the failure rate of systems [60]. The International Technology Roadmap for Semiconductors (ITRS) even states that processor costs and performance specifications may be limited by lifetime reliability and is of primary concern in the microprocessor's design phase [51]. The failure rate increases exponentially for electronic equipment with increasing temperatures: possible causes of failure are electromigration, chemical reactions, dielectric breakdown, or creep in the bonding materials [17]. A 10°C to 15°C temperature increase may halve a microprocessor's lifetime [125]. Therefore the system's temperature is often limited to control the peak-power heat dissipation, increase the Mean Time To Failure (MTTF), minimize power consumption, avoid self-destruction, as well as for safety reasons. Smartphones are often thermally capped around 50°C such that its users don't burn any body parts, but also to efficiently use the battery's electrical capacity. Moreover, the maximum skin temperature of electronic systems should be limited to 41°C to 45°C, depending on the material, to assure the user's touch comfort [9]. Advanced electronic equipments employ Thermal Management Units (TMUs) or Dynamic Thermal Managements (DTMs) techniques, which are able to throttle or scale the system such that stringent thermal constraints are met. Such thermal management techniques may be used at the system design phase or can be deployed dynamically at run time. A plethora of thermal control methods for microprocessors, and Systems-on-Chips (SoCs) [8, 60], exist that show trade-offs between temperature profile, frequency settings, power consumption and implementation complexity [138]. Complex microprocessors may employ hard-wired TMUs, e.g, some Intel chip sets [31, 77], whereas software implementations of TMUs are frequently seen in embedded devices. For TMUs it is important to understand the transient thermal behavior of the system. The transient thermal behavior of actively cooled system can be well described via an exponential relationship. But for passively cooled systems, such as smartphones, wireless sensors or possibly many objects participating in the Internet of Things (IoT), the exponential assumption does not hold. Passively-cooled systems rely on cooling via conduction, natural convection, radiation and on the thermal management of the TMU or DTM, which exhibit non-linear properties. In Chapter 5 the thermal behavior of a passively cooled microprocessor is studied by the development of a passive cooling law incorporating internal heat conversion, and convective and radiative coolings. Approximate solutions are also presented, as the exact solution is fairly complex and of the form $f(T) = t$, whereas for convenience a $f(t) = T$ expression is desired. The

performance of the approximate solutions are assessed. The *coefficient approximation* is put forward as a good all-round approximation with minimal error within the temperature range of typical embedded applications. For a microprocessor-like object it is shown that for large cooling surface areas the passive cooling law differs substantially from an exponential cooling law. For cooling surface areas smaller than about $1\,\mathrm{dm}^2$, the passive cooling law approximates an exponential law. For such system's the trade-off between a less accurate exponential cooling law is advantageous in terms of implementation and computational complexity.

Besides the temperature, other parameters affect the energy consumption profile of a microprocessor. The execution time characteristics and power requirements of the software and the system are the main drivers that define the final energy consumption. This is a direct result of the definition of electrical energy consumption: the integral of electrical power over time. The execution time is influenced by the type and the amount of operations contained by the software of concern, including register-based and external memory-based instructions. Each functional unit within a microprocessor and each component of the computer system have their own respective power and execution time profiles. As a result, every software has different power and execution time demands. For example, Carroll and Heiser [16] showed that, for an embedded system running `equake`, `vpr`, and `gzip` from the SPEC CPU2000 benchmark suite, the microprocessor energy consumption exceeds the RAM memory consumption, whereas `crafty` and `mcf` from the same suite showed to be straining more energy from the device RAM memory. Minimizing memory operations is a common energy optimization technique. For example, Intel's Data Direct I/O (DDIO) feature, introduced with the E5 Xeon chip, allows the Ethernet Network Interface Cards (NICs) to load data directly into the microprocessor's cache [57], minimizing Random Access Memory (RAM) memory accesses. By avoiding input/output (I/O) operations, the performance is improved but also the energy consumption of the system. An interesting feature of a code sequence's energy consumption, i.e., the product of its execution time and the microprocessor's power consumption, under certain assumptions, is that it has convex properties, which is elaborated in Chapter 4 and referred to as the *Energy/Frequency Convexity Rule*. The rule states that there exists a clock frequency for the execution of each sequence of code that minimizes the energy consumption of that code sequence. Under certain conditions this optimal clock frequency, minimizing energy consumption, lies between the minimum and maximum clock frequency. As will be shown, running at the optimal clock frequency is a trade-off between performance, in terms of execution time, and energy savings. For applications requiring human interaction, it has been shown that the clock frequency can be scaled down considerably without affecting the user's experience [101]. Experimental evidence for the existence of an Energy/Frequency Convexity Rule that relates energy consumption and microprocessor clock frequency on mobile devices is presented. The convexity property seems to ensure the existence of an optimal frequency where energy consumption is minimal. This existence claim is based on both theoretical and practical evidence. Data gathered via acquisition campaigns on multiple platforms suggest that the energy consumed per input element is strongly correlated with microprocessor clock frequency and, more interestingly, that the corresponding curve exhibits a clear minimum over a frequency window specific to the computer system. An analytical model of this behavior is also motivated, which fits well with the presented data. A parameter sensitivity analysis is carried out to assess the influence of the parameters on the optimal frequency. The optimal frequency is shown to increase when the power requirements of the computer system excluding the micropro-

cessor increases. Clock cycles lost for routine maintenance of the system also force the optimal frequency up. The optimal frequency as derived from the theoretical framework is, however, independent of the number of instructions to be executed.

In the final Chapter 6 the Energy/Frequency Convexity Rule is applied to practical applications. It is shown how to compute optimal frequencies minimizing system-wise energy consumption for systems incorporating multiple entities with independent clock frequencies, which is in essence a superposition of single entity systems. The Energy/Frequency Convexity Rule is also applied to a computer system subject to repetitive deadlines and the frequency scaling of a multithreaded system. It is shown that energy savings up to 30 % are achievable. The performance per Joule, as an extension of Amdahl's law, is analyzed of a heterogeneous computer system. This performance metric highlights the advantages of low-power and high-performance computing.

1.2 Contributions

This thesis provides a deeper theoretical and practical understanding of a microprocessor's energy consumption. Transient thermal behavior and energy gains via clock frequency scaling are of particular interest. The main contributions of this thesis are:

- A theoretical framework for the Energy/Frequency Convexity Rule and supportive experimental data: a sensitivity analysis of the Energy/Frequency Convexity Rule is carried out to estimate the impact of the multiple input parameters;

- An exact cooling law for an isothermal body subject to radiation, convection, and internal heat generation: the exact law is applied to a microprocessor-like object to assess its transient and steady-state behavior;

- Temperature/power relationship models applied to the Exynos 5410; these models are useful to cancel temperature biases in power measurement traces and to increase the accuracy thereof;

- Explicit power models for the Exynos 5410 SoC incorporating the microprocessor's clock frequency, temperature, and number of active cores; these models can be copy-pasted directly into any simulation;

- Multiple approximations to the exact cooling law are developed: these approximations are simpler and formulated more suitably for practical use;

- Methods and practical guidelines to increase the accuracy and meaningfulness of energy and power measurements;

- Actionable rules-of-thumb to assess when passive cooling becomes non-negligible compared to active cooling in embedded systems;

- Prototype implementations of all models; the exact and approximate cooling laws for passively cooled objects are provided for the convenience of the reader;

1.3 Outline

The remainder of this thesis is organized as follows. Chapter 2 introduces prerequisite knowledge that may help to understand the content of this thesis. Fundamental concepts of electrical energy consumption, execution time modeling, and heat transfer are presented. Then, Chapter 3 develops on the notion that electrical power is temperature dependent. Based on experimental evidence, a temperature-dependent power macro-level model is developed and analyzed for three different processors on two testbeds. Also, power/temperature transformation functions are presented to improve the accuracy and meaningfulness of power/temperature measurements. In Chapter 4 the Energy/Frequency Convexity Rule is introduced. Its theoretical foundation is laid out and experimental data is presented to support the analytical framework. A parameter sensitivity analysis explores the behavior, including the thermosensitivity, of the Energy/Frequency Convexity Rule. Following, the transient thermal behavior of a passively cooled microprocessor-like object is studied in Chapter 5. An exact cooling law is derived and is compared with the active cooling law. Approximations to the exact cooling law are also presented. Chapter 6 applies the Energy/Frequency Rule to practical applications, including deadline-based code execution, parallelism and Amdahl's law. The thesis is concluded in Chapter 7 with final remarks and possible directions of future research.

1.4 List of Publications

- K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "The Energy/Frequency Convexity Rule: modeling and experimental validation on mobile devices," in *Proceedings of the 10th Conference on Parallel Processing and Applied Mathematics*. Springer Verlag, Sep. 2013.

- K. De Vogeleer, G. Memmi, P. Jouvelot, and F. Coelho, "Modeling the temperature bias of power consumption for nanometer-scale CPUs in application processors," in *14th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, Jul. 2014, pp. 172-180.

- K. De Vogeleer, P. Jouvelot, and G. Memmi, "The impact of surface size on the radiative thermal behavior of embedded systems," CoRR, vol. abs/1410.0628, 2014, (submitted to *IEEE TMC* in 2014).

- K. De Vogeleer, G. Memmi, and P. Jouvelot, "Parameter Sensitivity Analysis of the Energy/Frequency Convexity Rule for Nanometer-scale Application Processors," CoRR, vol. abs/1508.07740, 2015, (in submission to *The Elsevier Journal of Parallel and Distributed Computing*, 2015).

Chapter 2

Background on Electrical Energy Consumption and Heat Transfer

Fundamental concepts on *electrical energy consumption* and *heat transfer* are recalled in this chapter. These fundamentals are prerequisites for understanding the sequel of this work. To understand electrical energy consumption, the different sources of power consumption in a microprocessor are highlighted and appropriate models are developed. Execution time models are also necessary to model the energy consumption of computer systems. Altogether, these models will be used to study the energy consumption of different kinds of applications in the next chapters. It will also be shown that temperature plays a major role in the magnitude of energy consumption during the execution of a program. Additionally, heat transfer models are presented, which will be used to understand the transient thermal behavior of a microprocessor.

Section 2.1 presents the notion of electrical energy consumption and power profiles in the context of computer systems. The different sources of energy consumption in a microprocessor are highlighted. In Section 2.2 models for the execution time of a program are presented. Thermal models will be used to understand the interaction between energy and temperature; therefore Section 2.3 presents a summary of thermal principles, including heat transfer modes and thermal properties of materials.

2.1 Power Consumption of Computer Systems

Computer systems, including a microprocessor, consume a specific amount of *energy* E, in Joules (J), to perform a given task. During this task the system uses *power* P (W=J/s) at a specific rate. The relationship between the power and the energy consumed by a computer system over a time period Δt follows the classic formula

$$E = \int_0^{\Delta t} P(t)\ \mathrm{d}t = \int_0^{\Delta t} I(t) \cdot V(t)\ \mathrm{d}t, \tag{2.1}$$

where $I(t)$ is the *current* supplied to the system, and $V(t)$ the *voltage* drop over the system. In a microprocessor $V(t)$ is quasi constant over time; hence $\int P(t)\ \mathrm{d}t$ only depends on $I(t)$. If both current and voltage are constant over time, the energy integral becomes the product of voltage, current and time, or alternatively power and time:

$$E = P \cdot \Delta t. \tag{2.2}$$

In practical applications the power consumption $P(t)$ of a microprocessor is anything but constant. Even more, the power fluctuations, while executing a piece of software, can be significant. Variations of 25 % are not uncommon in experimental traces [41]. This highly depends, however, on the type of hardware and the software being executed. The software dictates the microprocessor what parts of the microprocessor itself and what other hardware components should be enabled. All the sources of power demands in the computer system, besides the microprocessor, are referred to as the *background power*. For example, system power may include a Global Positioning System (GPS) sensor, needing power when it is sensing for signals, or an AMOLED LCD display requiring power not only proportional to the brightness level but also relative to the kind of color that is displayed [23], and other sources of power usages, e.g., memory sub-system maintenance of I/O devices (including memory), power supplies etc. Further on, however, the background power usage is deemed constant over time.

Now, the total power P_{sys} used by a computer system, including a microprocessor and background power, is then the sum of the said aspects:

$$P_{\text{sys}} = P_{\text{cpu}} + P_{\text{back}}, \tag{2.3}$$

where P_{cpu} is the power for the microprocessor and P_{back} is the power of the rest of the system. P_{back} can further be divided in components whose energy expenditure can be controlled by the microprocessor (P_{ctrl}), and the components that consume energy completely independent of the microprocessors (P_{fix}). For example, a hard disk goes into a low power mode when there haven't been data request from the microprocessor for a certain amount of time. In some sense the microprocessor thus controls the energy consumption of the hard disk. On the other hand, the inefficiencies of the power supply can drain some energy. This is an example of a fixed background power.

2.1.1 Microprocessors

A microprocessor is build up from logic gates, which are constructed from a network of transistors and capacitors. As the microprocessor is toggling gates at each clock cycle, a certain amount of energy is needed to drive the logic gates state changes, and some energy is leaking through the microprocessor by default. The energy required to drive a microprocessor's clock cycle can thus be attributed to three different physical processes, i.e., the electrical charge required to maintain the new logic gate state, the energy flushed during the logic gate state change, and the energy leaking through the microprocessor. A specific nomenclature is associated with these physical process: the *switching power* P_{switch}, the *short-circuit power* P_{short}, and the *leakage power* P_{leak}, respectively. The sum of the charge and short-circuit power is also referred to as the *dynamic power* P_{dynamic}, whereas the leakage power is sometimes also referred to as a *static power* component P_{static}. The switching power is the power usefully spend to drive the microprocessor's logic whereas the short-circuit and leakage powers can be considered as parasitic effects which are in some sense a *waste* of power. The microprocessor's power consumption may thus be represented as the sum of the referenced sources of power:

$$\begin{aligned} P_{\text{cpu}} &= P_{\text{dynamic}} + P_{\text{static}} \\ &= (P_{\text{switch}} + P_{\text{short}}) + P_{\text{leak}} \\ &= (I_{\text{switch}} + I_{\text{short}} + I_{\text{leak}}) \cdot V. \end{aligned} \tag{2.4}$$

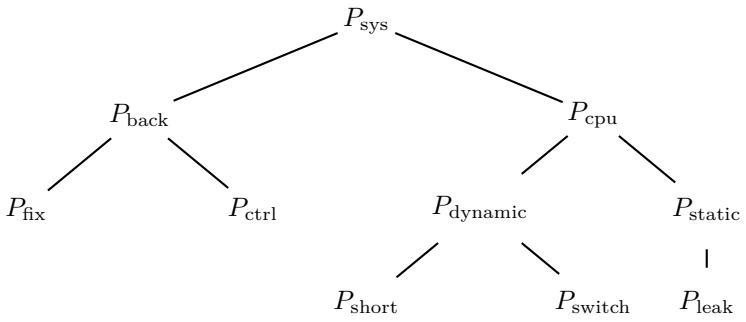


Figure 2.1: Power breakdown of a computer system as per Equation 2.4. In the literature P_{leak} is sometimes referred to as P_{static}, and P_{back} is sometimes used to reference P_{sys}.

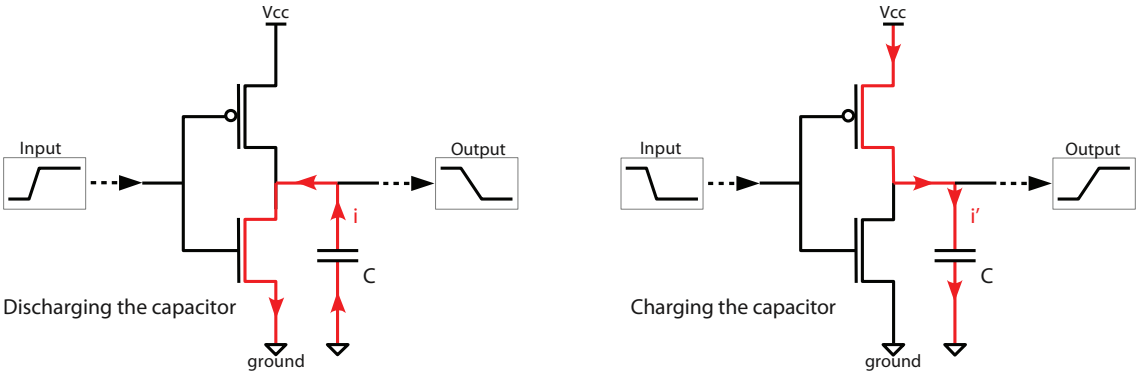


Figure 2.2: Inverter exemplifying the switching power. Electrical charge is stored in the capacitor `C` by the current i' via V_{cc}, or discharged into the ground via the current i, to represent the logical state of the inverter. Figure source: Bonamy [12].

Figure 2.1 provides a graphical representation of the power breakdown considered in the this work. The following sections go deeper into each of the microprocessor's power components.

2.1.2 Switching Power

Logic gates, which constitute the microprocessor, are built up from transistors that are configured such as to form logic gates, e.g., XOR, AND, flip-flops etc. Contemporary transistors are based on complementary metal-oxide-semiconductor (CMOS) technology, which have a typical temperature operation range for commercial electronics between 0°C and 85°C, and for military applications between -55°C and 125°C. The digital states of the logic gates, being `1` or `0`, are physically represented as a charge in a capacitor. When gates are toggled, these capacitors are charged or discharged. The energy required to toggle all the logic gates in the microprocessor contributes to the *switching power* P_{switch}. Figure 2.2 shows the internals of an exemplary inverter, or NOT-logic gate. For the inverter to invert its output, an electrical charge is placed in the capacitor to represent the logical state `1` or `0`. The charge is placed there by controlling the transistors that drive the logic gate.

In the literature P_{switch} is usually [128] modeled as

$$P_{\text{switch}} = \alpha C f V_{\text{cc}} V_{\text{swing}} \approx \alpha C f V^2, \tag{2.5}$$

where α is the *activity factor*, C the capacitance of the whole clock frequency domain,

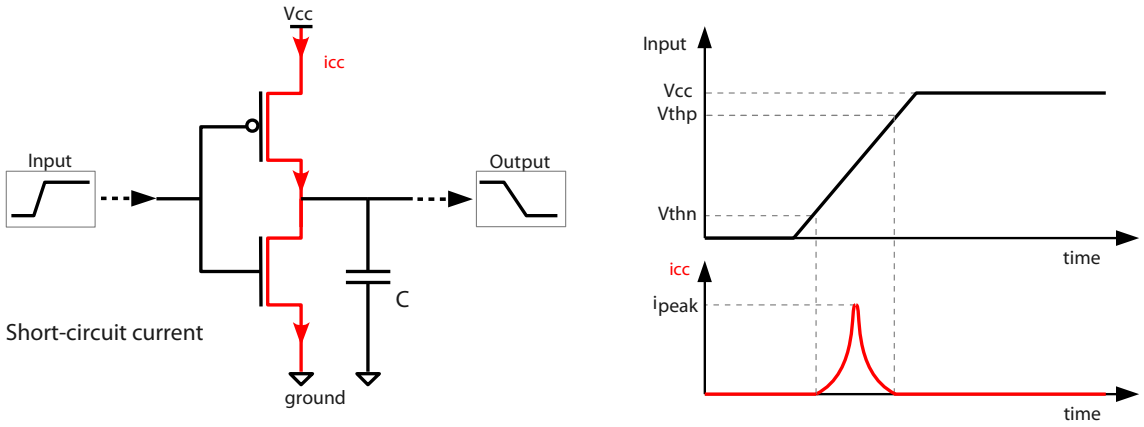


Figure 2.3: Inverter exemplifying the short-circuit power. While the gate is transitioning between states, the transistors in the gate are conducting simultaneously for a very brief moment and a short-circuit current (i_{cc}) is able to flow from the supply into the ground. Figure source: Bonamy [12].

f the frequency at which the domain is switching, V_{cc} or V the microprocessor's supply voltage, and V_{swing} the voltage swing across C. Usually $V_{swing} \approx V_{cc}$, and so the commonly known quadratic dependency of the switching power on the microprocessor supply voltage is obtained.

During each clock cycle, all the logic gates in the microprocessor are not toggling exactly at the same time. Thus, not the whole capacitance C of the microprocessor is charged or flushed each time. Therefore, the capacitance of the microprocessor is scaled with the activity factor $\alpha \in [0, 1]$ to represent only that part of the microprocessors' capacitance that is actually active. The activity factor is different for each clock cycle. But given the large number of clock cycles per time unit, e.g., 1 GHz, the activity factor may be approximated with an average value over time for a specific application, or a subpart thereof. For simple logic it can be assumed that the activity factor is lower than 0.25 [128]. For more elaborate circuits of logic gates, for example a complete microprocessor, the switching activity may even be lower, as not all the functional units are used at once. For example, a shift operation doesn't require the Floating Point Unit (FPU).

If the voltage is linearly dependent on the frequency, then Equation 2.5 becomes a third-order polynomial. In the literature [112] this is also referred to as the *cube root rule*: $P_{switch} \approx K_v V^3 \approx K_f f^3$, where K_v and K_f are constants.

2.1.3 Short-circuit Power

The short-circuit power consumption P_{short} originates during the transition of logic gates between two states. During this transition, or switching, the transistors inside the logic gate may conduct simultaneously for a very brief moment in time, creating a direct path between V_{cc} and the ground. Even though this peak current happens over a very small time interval, given the high microprocessor clock frequencies and large amount of logic gates, the short-circuit current may be non-negligible. Figure 2.3 shows an example of the short-circuit current flow with the help of an inverter. When the inverter is inverting, the upper transistor will close and the bottom transistor will open, or vice versa. The time for these transistors to open and close is, however, finite. Therefore, the two transistors will be conducting simultaneously for a very brief moment and drawing some current directly

from the source into the ground. This current is not contributing to the representation or maintenance of digital data and can therefore be considered undesirable. Yet, by virtue of the non-ideal behavior of transistors short-circuit currents are not avoidable.

The short-circuit currents are technology- and logical gate-dependent. For example, for a CMOS inverter [119] without load P_{short} is shown to adhere to the equation

$$P_{\text{short}} = \frac{H}{12}(V_{\text{cc}} - 2V_{\text{thn}})^3 \tau f, \tag{2.6}$$

where H is a technology-dependent scaling factor, τ is the input signal rise time, V_{cc} is the supply voltage, V_{thn} is the threshold voltage, and f is the switching frequency. For other types of gates with different transistor configurations P_{short} will differ. Accurate formulations for P_{short} for all types of gates are not always available. Moreover, a detailed value of P_{short} is tedious to compute for fast clock frequencies and complex microprocessors. This would require the knowledge of the physical properties and dimensions of transistors, logic gate configurations and detailed gate activity. This information is, indeed, not publicly available for commercial microprocessors. Efforts have been devoted to establish a P_{short} model applicable to arbitrary logic gates. For example Vemuru and Scheinberg [120] estimated the average short-circuit power dissipation in arbitrary CMOS logic gates based on an α-power law model that includes the velocity saturation effects of short channel metal-oxide semiconductor field-effect transistors (MOSFETs). Though, their resulting model is complex. Sylvester and Keutzer [115] approximated the short-circuit power with a heuristic model based on the timespan t_{base} and magnitude of the short-circuit current I_{peak}:

$$P_{\text{short}} = N \frac{\alpha_{\text{sc}}}{2} t_{\text{base}} I_{\text{peak}} V_{\text{cc}} f, \tag{2.7}$$

where N is the number of logic gates, and α_{sc} is a model parameter describing the saturation current of the transistors. A typical value for α_{sc} is 1.3.

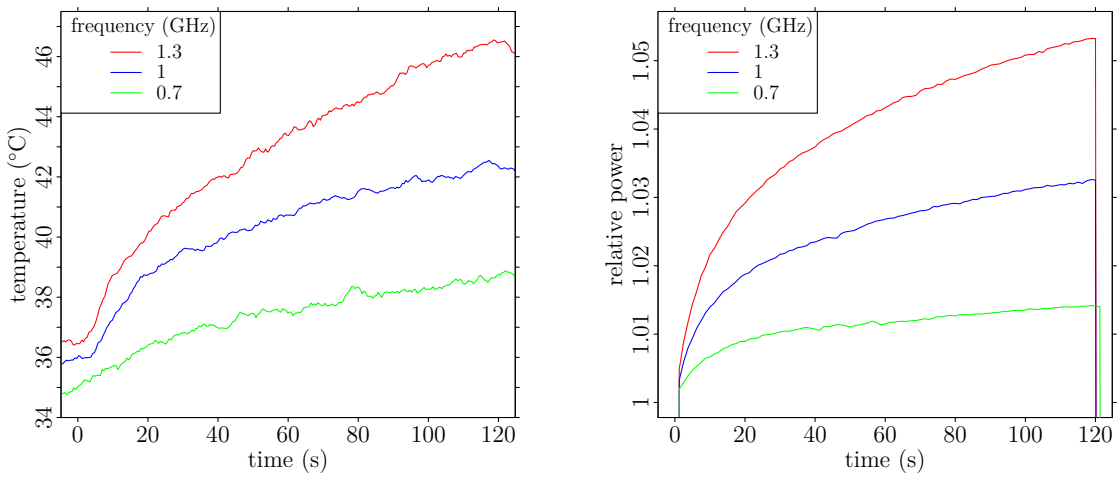
In Equations 2.6 and 2.7 it is seen that the short-circuit power is proportional to the switching rate of the gate. Thus it can be stated that $P_{\text{short}} \propto \alpha f$, where α is the switching activity and f is the frequency in Equation 2.5. As a result, the proportionality constant υ can be considered as the ratio of the short-circuit power P_{short} over the switching power P_{switch}. Sylvester and Keutzer [115], who developed the Berkeley Advanced Chip Performance Calculator (BACPAC) for sub-micron technology, estimated that the short-circuit power is approximately 15 % of the dynamic power. The dynamic power of a microprocessor, the sum of the switching and short-circuit power, can then be developed:

$$\begin{aligned} P_{\text{dynamic}} &= P_{\text{switch}} + P_{\text{short}} \\ &= P_{\text{switch}} + \upsilon P_{\text{switch}} \\ &= (1+\upsilon) P_{\text{switch}} = \xi f V^2, \end{aligned} \tag{2.8}$$

where $\xi = (1+\upsilon)\alpha C$. P_{dynamic} represents the microprocessor's average power while the microprocessor is switching. Next, power dissipation is expounded, which is independent of the switching activities of the microprocessor.

2.1.4 Leakage Currents

The power P_{leak} emanates from leakage currents which flow between the differently doped parts of MOSFETs, the basic building block of microprocessors. The energy carried by these currents are lost and don't contribute to the information that is processed or held



(a) time trace (b) power trace

Figure 2.4: Temperature (a) and power (b) traces of an Exynos 4210 application microprocessor under constant workload, run at different microprocessor clock frequencies. It is observed that the power consumption is inflated due to the microprocessor's increasing temperature, given that all other parameters were kept constant [28].

by the logic gates. On a micro-level they also directly affect the transistor's amplification characteristics, the main task of the transistor. Six distinct sources of leakage are identified [70]: *reverse-bias pn-junction leakage*, *sub-threshold leakage*, *oxide tunneling current*, *gate current due to hot-carrier injection*, *gate-induced drain leakage* (or *band-to-band tunneling*), and *channel punch-through current*.

Some types of leakage currents are induced during the *on* or *off*-state of the transistor, or both. Despite the presence of multiple sources of leakage in MOSFET transistors, the sub-threshold leakage current, gate leakage, and band-to-band tunneling (BTBT) dominate the others for sub-100 nm technologies [3]. Leakage current models, e.g., as incorporated in the Berkeley Short-channel IGFET Model (BSIM) [70], are accurate, nevertheless complex since they depend on multiple variables. The BSIM leakage currents contain variables that, inter alia, are technology-specific, e.g., transistor dimensions, gate oxide capacitance. Moreover, the leakage power fluctuates as it also depends on the temperature T of the microprocessor. The temperature itself appears several times in BSIM for the sub-threshold and BTBT leakage models; the gate leakage however is not temperature-dependent. Mukhopadhyay *et al.* [80] showed via simulation that for 25 nm technology the sub-threshold leakage current is dominant over the BTBT leakage current, but the latter cannot be neglected. You *et al.* [134] shows, for a 0.1 μm microprocessor, that a temperature increase from 30°C to 40°C leads to a 3 % leakage power increase. Liao *et al.* [69] noted that due to temperature-dependent leakage current the power increases by 38 % between 65°C and 110°C. Figure 2.4 shows excerpts of temperature/power traces of a microprocessor under constant workload. It is observable that the temperature and power profile are not constant over time. Consequently the leakage power should not be considered a static part of the microprocessor's power time-wise. This is the reason why in this work the name P_{leak} is considered instead of P_{static}, as often used contradictorily in other literature, as P_{leak} is not constant. Now, the leakage power P_{leak} is conveniently represented as

$$P_{\text{leak}}(T) = I_{\text{leak}}(T) \cdot V_{\text{cc}}, \tag{2.9}$$

where $I_{\text{leak}}(T)$ is the temperature-dependent leakage current flowing through the entire microprocessor. Theoretical research based on simulation provides a plethora of models to address the temperature dependency of a microprocessor power profile [36, 69, 108, 114]. Skadron *et al.* [109] in particular deducted a relationship between the leakage power P_{leak} and dynamic power P_{dynamic} based on International Technology Roadmap for Semiconductors (ITRS) measurement traces (variables indexed with 0 are reference values):

$$R_T = \frac{P_{\text{leak}}}{P_{\text{dynamic}}} = \frac{R_0}{V_0 T_0^2} e^{\frac{B}{T_0}} V T^2 e^{\frac{-B}{T}} = \gamma V. \tag{2.10}$$

If the temperature T is stable across different operating voltages, then the value of R_T is a function of V multiplied by a parameter, lets say γ, that includes the temperature-dependent variables and other constants. Using Equation 2.8, the system power P_{sys} in Equation 2.4 then becomes:

$$\begin{aligned} P_{\text{cpu}} &= P_{\text{dynamic}} + P_{\text{leak}} \\ &= P_{\text{dynamic}} + \gamma V P_{\text{dynamic}} \\ &= (1 + \gamma V) \cdot \xi f V^2. \end{aligned} \tag{2.11}$$

Or alternatively the leakage current can be kept separated from the dynamic power component by maintaining a proper leakage current component I_{leak}:

$$P_{\text{cpu}} = P_{\text{dynamic}} + P_{\text{leak}} = \xi f V^2 + I_{\text{leak}} V. \tag{2.12}$$

Equations 2.11 are 2.12 are equivalent and represent the microprocessor's power. They can, and will, be used interchangeably.

Hardware designers strive to minimize leakage and short-circuit power by, e.g., advanced computer-aided design (CAD) tools, by superior Very-Large-Scale Integration (VLSI) technology, or by the voltage gating of functional units, memory, and cache [98, 112]. For example, You *et al.* [134] extended an Instruction Set Architecture (ISA) with instructions to support the control of power-gating at the component level. Via simulations the authors claim to reduce the power consumption of their benchmarks on the average by 23.05 %. Besides putting a system to sleep via power-gating, waking up the system at the right moment is necessary for minimizing the possible time lost while waiting for the system to wake up. Abramovici *et al.* [2] invented an autonomous power management system that predicts the optimal sleeping and wake up time with the aim of saving energy and reducing waiting times due to the power-gating process.

A more detailed discussion on leakage currents will be presented in Chapter 3.

2.1.5 Voltage and Frequency Scaling

The power models in the previous section depend on the clock frequency of the microprocessor. Nowadays microprocessors are able to scale their clock frequency dynamically. The act of dynamically scaling the clock frequency is referred to as *Dynamic Frequency Scaling (DFS)*, *nonsupply-voltage-scaling (NSVS)* or *CPU throttling*. When the supply voltage of the microprocessor is scaled accordingly, one talks about *Dynamic Voltage and Frequency Scaling (DVFS)*. The motivation behind scaling the supply voltage along with the clock frequency is that the electronic circuits need to react faster when the clock frequency is scaled up. This is achieved by applying a higher supply voltage. Accordingly, if the frequency is scaled down, to save energy, the supply voltage may be scaled down

as well. There exists a minimum voltage level for each clock frequency setting for the microprocessor to operate without errors. If logic gates state changes cannot keep up with the pace of the clock frequency, errors will be introduced in the digital circuits. In *Dynamic Voltage Scaling (DVS)* systems the voltage is controlled to scale the energy consumption of the system irrespective of the clock frequency. For systems that can tolerate computational errors [18, 61], the supply voltage may be dropped below the minimum supply voltage level, i.e., *undervolting*.

Due to physical constraints of electronic regulators, the clock frequency and supply voltage cannot be scaled continuously. The scaling of the frequency and supply voltage happens in fixed discrete steps dictated by the analogue electronic drivers. Dedicated power supply and clock frequency regulators exist for the modern microprocessors and SoCs. Moreover, the voltage/frequency ratio should not be assumed constant. Minor variations occur for all clock frequency settings. Exemplary voltage/frequency graphs are shown in Section 4.2.

2.1.6 Power Measurement Instruments

The power of a computer system can be measured following several different protocols. Each of them has its advantages and disadvantages. First, the current mirror-based measurement approach is addressed and then the shunt resistors. Software measurement approaches and multimeters are also discussed.

The more accurate power measurement devices are based on *current mirrors*-like features. A current mirror duplicates an arbitrary current. This current copy can then be stored in a capacitor and its magnitude can be read out over fixed time intervals. The energy stored is then proportional to the power drawn by the original current. Also, such power measurement method is able to handle all kinds of irregular wave-forms, even at arbitrary high frequencies. However the current mirror has the disadvantage that, given the presence of a current copy, twice the amount of energy is consumed. For battery powered devices this is clearly not desirable. A simpler, smaller, yet less accurate method is often used for embedded applications and microprocessors. On the power supply rail, to the system under study, a small shunt resistor, in the order of milliohms, is installed in series. The voltage drop over the resistor is then proportional to the power supplied to the system. The disadvantage of such system is that a little power is lost over the resistor and the system under study is perturbed. Moreover, the sampled voltage over the resistor is a snapshot of reality. Events occurring in between two samples will not be seen by such power measurements. The current mirror, on the other hand, has no problem with such blind spots as it integrates the current in between two samples.

Shunt resistor-based systems can be implemented on-chip together with other logic, but dedicated chips also exist. For example, the INA231 chip implements the power measurement logic for the ODROID testbed used hereafter. An Inter-Integrated Circuit (I^2C) interface provides a standardized communication channel between the chip and any other system for power measurement read-out retrieval. External power measurement tools, e.g, the *Monsoon Power Monitor*[1] can also be used. This power monitor based on the current mirror principle is popular with large research institutions and organizations. The Monsoon Power Monitor is a device, the size of a 500 pages book, which is intended to be used to replace the power source, e.g., for the battery of a smartphone or wireless sensor. This implies that the total power of the system is measured. It is thus difficult,

[1]Monsoon power monitor: http://www.msoon.com/LabEquipment/PowerMonitor/

but not impossible, to measure selectively the power consumption of components on a circuit-board. A shunt resistor is therefore a better solution if a more targeted power usage is to be measured.

When the SoC's temperature is measured alongside the power it is useful that both aspects are recorded by the same processor. Synchronizing both traces is then more straightforward as a common timestamp clock exists. In the case of the Monsoon power monitor temperature and power traces are to be synchronized manually as the power monitor's graphical user interface (GUI) doesn't support temperature measurements, nor some sort of time stamp synchronization facilities. Manual synchronization may introduce errors in the measurements and have to be minimized.

A third category of power measurement devices are based on "multimeter" measurement techniques. A multimeter can be placed in series and/or in parallel with a power supply. Often the voltage supply is assumed constant and then only the current is measured. The *wattsup*[2] power meter is a commonly used multimeter-oriented measurement device in the energy consumption devices research domain. The wattsup meter is placed between an electricity socket and an electronic device. The meter is thus only usable with the larger electronic appliances requiring electricity from the grid. The sampling rate of cheap multimeters are often too low and automated read-outs are not always possible. Cheap multimeter setups should be avoided by all means if one aims for accuracy in power measurements. Professional accurate multimeter systems, on the other hand, such as provided by LabVIEW, also exist, which provide more elaborate features and are priced accordingly. It is noted that the alternate current (AC) mode of multimeters are designed to measure sine-waves and report Root Mean Square (RMS) values. When talking about power one should thus refer there to volt-ampere (VA) and not Watt.

A multitude of software applications for all kinds of platforms exist that "measure" the power of the underlying hardware [81]. These applications can be divided in two groups. The first group presents power estimates from hardware power sensors. Modern microprocessors and SoC may have on-chip or off-chip power sensors, such as the Intel's Sandy Bridge micro-architecture [97] and the ODROID XU+E platform. These hardware-backed applications provide a power estimate with descent accuracy. The second group of power measurement applications consist of power models that are fed by some kind of software and hardware statistics to produce a power estimate. These models are derived from real power measurements and tuned for specific platforms. Linear multidimensional regression models, or, fancy put *perceptrons*, are commonly used. The input of these power models can be, e.g., historical context information or hardware performance counters (HPCs). The accuracy of these application power models are often limited around 5 %.

2.2 Execution Time Modeling

Estimating the energy consumption of software, or a sequence of instructions, executing on a microprocessor via Equation 2.1 requires the knowledge of the power $P(t)$ and the *execution time* Δt. The previous section elaborated on the power. This section focuses on modeling the execution time of an instruction sequence, sometimes also referred to as the *makespan* or *timespan*.

A piece of binary software consists of a sequence of instructions which are executed one-by-one by a scalar microprocessor. Each instruction requires a certain amount of

[2]Watts up? power meters: https://www.wattsupmeters.com

clock cycles to complete. The number of clock cycles required depends on the microprocessor's architecture, the type and number of executed instructions, and the context. For example, ARMv7, a low-power 32-bit Reduced Instruction Set Computing (RISC) microprocessor architecture, requires one clock cycle for a data operation, three clock cycles to load a single register, and two additional cycles if the *program counter* is the destination.[3] Microprocessors where instructions act on a single datum at a time are called *scalar microprocessors*, or also referred to as Single Instruction Single Data (SISD) microprocessors. *Vector microprocessors*, on the other hand, have instructions that can operate on a data vector with a single instruction, which is also generally known as Single Instruction Multiple Data (SIMD). Instructions requiring memory access need additional clock cycles for the data to be fetched or stored. This time penalty may vary whether data is available in a cache, or the main memory needs to be accessed. While instructions are waiting for memory accesses to complete, out-of-order execution (OOE) microprocessors are able to execute data-independent instructions. Processors may also employ pipelines which allow the overlap of instruction executions. A pipelined microprocessor will have a clock-cycles per instruction (CPI) value close to 1. Features such as advanced pipelines, SIMD, and OOE reduce the execution time effectively, but increase the complexity of microprocessor design and may not necessarily be in favor of energy consumption. More information about microprocessor architecture can be found in the work of Stallings [113].

2.2.1 Sequential Execution Model

Let's assume that a code sequence requires cc_b clock cycles to complete execution on a basic scalar microprocessor. Then the total *execution time* Δt of the code sequence is proportional to the inverse of the number of microprocessor clock cycles f_cpu available to the code sequence:

$$\Delta t = \frac{cc_\mathrm{b}}{f_\mathrm{cpu}}. \tag{2.13}$$

It can be observed that the execution time Δt approaches asymptotically infinity when the frequency f_cpu approaches zero: $\lim_{f_\mathrm{cpu}\to 0} \Delta t = \infty$.

On advanced microprocessors, e.g., with pipelines and OOE, or in multi-task environments, such as frequently found in multi-purpose/advanced embedded systems, a code sequence may be stalled for the microprocessor to perform other tasks. Indeed, an Operating System (OS) needs periodically some clock cycles to perform recurrent or occasional tasks, e.g., interrupt handling, I/O device requests, process scheduling, system calls, or processing kernel events. Also, if other programs are scheduled on the same microprocessor then the microprocessor clock cycles are likely divided fairly among the contestants. Moreover, the hardware may halt the execution of software sometimes momentarily to handle exceptions, e.g., during pipeline stalls due to branch miss-prediction, misaligned memory access, page faults, operation intervention etc. From a heuristic point of view, it can be assumed that the OS kernel and microprocessor need on the average a fixed number of clock cycles f_k per time unit to complete their tasks. Thus, the execution time of a code sequence becomes

$$\Delta t = \frac{cc_\mathrm{b}}{f_\mathrm{cpu} - f_\mathrm{k}}, \quad \text{where } f_\mathrm{k} \leq f_\mathrm{cpu}, \tag{2.14}$$

and where f_cpu is the microprocessor's clock frequency, and f_k are the number of microprocessor clock cycles not available to the code sequence cc_b. Δt values for $f_\mathrm{cpu} < f_\mathrm{k}$ are

[3] ARM Architecture Reference Manual: http://infocenter.arm.com

not defined. Note that now Δt tends to infinity at the vertical asymptote defined by f_{k}.

Such a formulation for the execution time as Equation 2.14 is also found in the work of Snowdon *et al.* [110] (Equation 1), though presented differently. Snowdon presents the ratio between the lengthening of the execution time by OS interrupts:

$$\frac{T_{\mathrm{tot}}}{T_{\mathrm{work}}} = \frac{1}{1 - C_{\mathrm{tick}} f_{\mathrm{tick}} / f_{\mathrm{cpu}}}.$$

Equation 2.14 refers to f_{k} and $\frac{cc_{\mathrm{b}}}{f}$ whereas Snowdon refers to $C_{\mathrm{tick}} f_{\mathrm{tick}}$ and T_{work}, respectively, which eventually leads to the same result. Snowdon indicates that for their tested platforms, laptop computers from mid 2000, $f_{\mathrm{cpu}} \gg f_{\mathrm{k}}$. From experimental experience with application microprocessors, it may be observed that f_{k} can not always be neglected, especially for the low microprocessor frequency settings. Also, f_{k} seems to be varying for different benchmarks and architectures.

2.2.2 Execution Model with Slack Time

The sequential execution time model of a code sequence, as given by Equation 2.14, is based on a simple scalar microprocessor. Let's extend the model by the stalling time caused by I/O access, e.g., memory accesses, and add OOE support.

The execution of a code sequence can be slowed down if data needs to be fetched outside the microprocessor registers. The required data may be present in a cache or must be fetched in the main memory of the computer system. Either way, accessing the cache and the main memory incurs a time penalty, compared to accessing the local registers in the microprocessor. Access times for caches are much shorter than the main memory; yet, caches are much smaller and monetary more expensive. The external memory may have its own clock frequency, and can be working synchronously with the bus frequency, or independently, i.e., asynchronously. Asynchronous memory is generally faster and more energy-efficient than synchronous memory, but is more complex to implement [25]. Let us focus on synchronous memories as they are more frequently found in the embedded applications of our interest.

Basic microprocessors with synchronous memory will stall upon external memory accesses and resume the code execution when the requested data is available. Accessing the external memory will only happen when the requested data is not available in the cache. The number of *external memory accesses* M_a is proportional to the *cache miss-rate* ε of the highest cache level. The number of *external memory accesses* is proportional to

$$M_a = \varepsilon n_{\$} c c_{\mathrm{b}}, \tag{2.15}$$

where $n_{\$}$ is the proportion of code sequence that requires *cache transactions*. When the external memory is accessed, usually data is retrieved to replace a whole cache line at once, in so-called burst-mode access. The total execution time of a code sequence then includes the time while waiting for data being retrieved from the external memory:

$$\Delta t = \frac{cc_{\mathrm{b}}}{f_{\mathrm{cpu}} - f_{\mathrm{k}}} + M_a \frac{cc_{\mathrm{m}}}{f_{\mathrm{mem}}} = cc_{\mathrm{b}} \left(\frac{1}{f_{\mathrm{cpu}} - f_{\mathrm{k}}} + \beta \right), \tag{2.16}$$

where cc_{m} is the number of memory clock cycles required for a burst-mode transaction, f_{mem} is the memory clock frequency, and $\beta = \frac{M_a}{cc_{\mathrm{b}}} \frac{cc_{\mathrm{m}}}{f_{\mathrm{mem}}}$. The β parameter represents the average amount of time per clock cycle the microprocessor is waiting for the external

memory. β is similar to Hsu and Kremer's [54] β_{cpu} parameter, which measures, what they refer to as, the CPU-boundedness. More general, the β parameter may also be representative for all events that forces the microprocessor to wait for external inputs, e.g., sensor or radio interface data.

Assuming that the microprocessor stalls during external memory accesses is realistic for in-order execution microprocessors. OOE-enabled microprocessors, on the other hand, can continue to execute data-independent code during external memory accesses, while the data-dependent code is waiting for the memory access to return. The number of instructions being able to execute in an out-of-order fashion is code specific, i.e., data-independent code must be available for execution. However, the average time gained via OOE can be defined by introducing a variable σ_{x} in Equation 2.16; σ_{x} is a ratio representing the portion of time spend on external memory accesses that is not able to be covered by OOE. Then the OOE-enabled execution time of Equation 2.16 becomes

$$\Delta t = \frac{cc_{\mathrm{b}}}{f_{\mathrm{cpu}} - f_{\mathrm{k}}} + \sigma_{\mathrm{x}} M_a \frac{cc_{\mathrm{m}}}{f_{\mathrm{mem}}} = cc_{\mathrm{b}} \left(\frac{1}{f_{\mathrm{cpu}} - f_{\mathrm{k}}} + \sigma_{\mathrm{x}} \beta \right). \tag{2.17}$$

The presence of the external memory accesses implies that the execution time is dependent on the highest cache-level miss rate. In particular the cache miss rate may be less predictable when multiple processes are communicating with a shared cache, e.g., the kernel, parallel threads or applications. As a result the execution time estimates of repeated measurements may fluctuate considerably when the microprocessor is required to communicate frequently with the external memory in a multi-task environment.

2.2.3 Multi-Core Execution Model

The execution model with slack time of Equation 2.17, applies to the execution of a code sequence on a single core microprocessor with separate clock frequencies for the memory and the microprocessor. To support single-frequency domain multi-core microprocessors, f_{cpu} needs to be multiplied with the *number of available cores* c, assuming that the code sequence is parallelizable. In this context f_{k} will also incorporate the clock cycles not utilized during the serial portion of parallel applications[4]. The execution time model for a single-frequency domain multi-core microprocessors then becomes:

$$\Delta t = \frac{cc_{\mathrm{b}}}{c f_{\mathrm{cpu}} - f_{\mathrm{k}}} + \sigma_{\mathrm{x}} \frac{M_a cc_{\mathrm{m}}}{f_{\mathrm{mem}}}. \tag{2.18}$$

Specific parts of a code sequence can be executed, deterministically or non-deterministically, with multiple microprocessor and memory clock frequencies. In such cases the execution time must be considered during the different clock frequency regimes separately. As such, a generalized execution time model is obtained for a multi-frequency domain, multi-core microprocessor:

$$\Delta t = \sum_{f_x \in f_{\mathrm{cpu}}} \frac{cc_{\mathrm{b},f_x}}{c f_x - f_{\mathrm{k},f_x}} + \sum_{f_y \in f_{\mathrm{mem}}} \sigma_{\mathrm{x},f_y} \frac{M_{a,f_y} cc_{m,f_y}}{f_y}, \tag{2.19}$$

where f_{cpu} and f_{mem} are the set of frequencies the microprocessor cores and memory run at, respectively. Dimensioning the values for cc_{b}, f_{k} and M_a, on-line or off-line, for the

[4]Code that is written for parallel processing often contains a portion that must be executed serially nonetheless. This notion is captured by *Amdahl's law*, which Section 6.4 dwells upon.

multiple frequencies may pose great challenges. Detailed knowledge is required of the microprocessor operations, memory accesses and context to know the exact execution of a code sequence.

2.3 Principles of Heat Transfer

Before, it was indicated that the electrical energy consumption of a microprocessor is a contribution of multiple physical processes, amongst others, the leakage currents. In Section 2.1.4 it was mentioned that these currents are temperature-dependent. Chapters 3 and 5 talk in detail about leakage currents in microprocessors and the thermal behavior of embedded systems. To comprehend this discussion, fundamentals of heat transfer and thermal behavior are presented here. To know more about the subject, the reader is referred to the work of Cengel and Ghajar [17].

In the following analyses the microprocessor is assumed to be a *lumped system*. A lumped system is a system in which the temperature can be described as a function of time only: $T(t)$, i.e., whose interior temperature is assumed to be quasi uniform [17]. The *heat transfer rate* Q [J/s], or *heat flux*, is the amount of heat transferred per unit of time. The heat flux of the system Q_{sys} is defined as the sum of the heat flux Q_{out} released to the environment, the heat absorbed from the environment Q_{in}, and the internal heat generation of the system Q_{ihg}:

$$Q_{\text{sys}} = Q_{\text{in}} + Q_{\text{ihg}} - Q_{\text{out}}. \tag{2.20}$$

Here Q_{out} is a negative flux as heat is moving from the system to the environment. For a system in equilibrium, or steady state, i.e., at a constant system temperature T_e, the outgoing heat flux Q_{out} is equal to the sum of the heat flux into the system Q_{in} and the internal heat generation Q_{ihg}; hence $Q_{\text{sys}} = 0$. If the temperature of the system is in a transient state the sum of Q_{in}, Q_{out} and Q_{ihg} will differ from zero: $Q_{\text{sys}} \neq 0$; if $Q_{\text{sys}} > 0$ the system is heating up; for $Q_{\text{sys}} < 0$ the system is cooling down.

Steady state is attained when the temperature within the body and its surfaces is independent of time. When the temperature is in a transient state, either energy is being stored or removed from the body. The rate at which energy is stored is denoted by:

$$Q_{\text{sys}} = mc_p \frac{\partial T}{\partial t}, \tag{2.21}$$

where m [kg] is the body's mass, and c_p is the body's *specific heat capacity* [J/(kg·K)] at constant pressure. Heat can be added to, or extracted from, the system via the three fundamental modes of heat transfer, which are *conduction*, *convection* and *radiation*, which will be introduced in Section 2.3.2. But first, let us recall some thermal properties of materials that are required to develop the modes of heat transfer.

2.3.1 Thermal Properties of Materials

Thermal properties of materials are quantified by several parameters addressing different physical aspects. The *thermal conductivity* k [W/(m·K)] quantifies the ability of a material to conduct heat. The larger k, the better the material is conducting heat. The thermal conductivity varies for each material and phase. For solids the thermal conductivity is primarily only temperature-dependent. For gases, k is also temperature-dependent

Table 2.1: Typical values for the *thermal conductivity* (k) of applicable materials for computer systems at room temperature. Data compiled from multiple sources [1, 66, 79, 87, 124].

Material	k	Additional Information
air	0.025	at atmospheric pressure
Al	200	for heat sink purposes
Au	318	electronic interconnects
Cu	386	for heat spreader purposes
Ge	0.6	pure crystal structure
IC package	0.40	HL832 integrated-circuit plastic package
PCB	0.35	FR4 printed circuits
Si	1.5	pure crystal structure
SiO_2	1.4	pure crystal structure
thermal grease	0.79	Si-Free 1020 metal-oxide synthetic grease

Table 2.2: Typical values for the *thermal diffusivity* (α) of applicable materials for computer systems at room temperature. Data compiled from multiple sources [66, 79, 86, 87, 124].

Material	α (10^{-6})	Additional Information
air	25	at atmospheric pressure
Al	84.18	for example as heat sink
Cu	112.34	for example as heat spreader
Ge	36	pure crystal structure
IC package	0.019	HL832 integrated-circuit plastic package
PCB	0.032	FR4 printed circuits
Si	92	pure crystal structure
SiO_2	0.087	pure crystal structure
thermal grease	1.3	T670 high bulk thermal conductivity

and quasi independent to pressure close to atmospheric pressure. Table 2.1 shows some examples of k-data for applicable materials in the context of computer systems.

The *thermal diffusivity* α [m^2/s] is the thermal conductivity divided by density and specific heat capacity at constant pressure:

$$\alpha \equiv \frac{k}{\rho c_p}, \tag{2.22}$$

where ρ is the *density* of the material and c_p the *specific heat capacity* at constant pressure. The larger the thermal diffusivity, the faster heat is diffused in a material. For gases the thermal diffusivity is highly dependent on temperature and pressure. Typical values for α for applicable materials in the context of computer systems are reported in Table 2.2.

2.3.2 Heat Transfer Modes

Heat transfer happens via the three fundamental modes, namely *conduction*, *convection* and *radiation*. Each heat transfer mode is elaborated in this section. These modes may be part of Q_{out} of Equation 2.20.

All modes of heat transfer are present in thermal processes, though some are more prominent, and others may be insignificant. For example, for actively-cooled systems the convective component may dominate the radiation component by several orders of magnitude. Then the radiative component may be ignored without loss of accuracy. For passively-cooled systems on the other hand, the convective component may be comparable in magnitude and hence should be considered.

Conduction

A temperature gradient in a homogeneous body induces a heat flux Q_d through a surface A described by Fourier's law:

$$Q_d = -kA\frac{\partial T}{\partial n}, \tag{2.23}$$

where n is the *normal* in the direction of the area A, and k is the *thermal conductivity* as seen in Table 2.1. When a homogeneous body is in *steady state* and k is fixed then the solution to Fourier's law reduces to a linear system.

The *general conduction equation* is a result of the first law of thermodynamics applied to a three-dimensional volume:

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} + q = \frac{1}{\alpha}\frac{\partial T}{\partial t}, \tag{2.24}$$

where q is the rate of *internal heat generation* per unit volume, and x, y, z are the three dimensions of the volume. Internal heat generation is the process that generates heats inside the volume. Equation 2.24 is generally know, as *Fourier's law of heat conduction*.

Convection

A solid body exposed to a moving fluid is subject to energy exchange if their temperatures differ. Energy is *convected* from the body if the moving fluid has a lower temperature than the body. The energy transfer rate Q_v between the moving fluid and the surface of the body is also formally known as *Newton's law of cooling*:

$$Q_v = h_v A(T_s - T_a), \tag{2.25}$$

where T_a is the temperature of the moving fluid, T_s the temperature of the surface of the body, and h_v the *convective heat transfer coefficient*. It must be noted, however, that the actual energy exchange at a solid-fluid boundary is by conduction. It is the fluid that convects the energy away from the body. The formulation of Equation 2.25 can be attained by applying Fourier's law in Equation 2.23.

The well-known solution to Newton's law of cooling is easily obtained:

$$T(t) = T_a + (T_0 - T_a)e^{-dt}, \tag{2.26}$$

where T_0 is the temperature at $t = 0$, and $d = h_v A$.

Radiation

Radiative heat transfer happens trough exchange of electromagnetic wave propagation, possibly through both vacuum or a medium. Stefan-Boltzmann's law states that the power radiated from a *black body* is proportional to its temperature. A *black body* is a body that absorbs all incident radiation and, at thermal equilibrium, emits its energy over a well defined frequency spectrum. In particular, Stefan-Boltzmann's law states that the radiant heat transfer Q_r is proportional to the black body's temperature to the fourth power:

$$Q_r = \epsilon\sigma A T^4, \tag{2.27}$$

where σ is the Boltzmann constant $5.6697 \cdot 10^{-8}$ $[\mathrm{W}/(\mathrm{m}^2 \cdot \mathrm{K}^4)]$, A is the surface area subject to radiation, and $\epsilon \in [0, 1]$ is the *emissivity* of a gray body's surface (dimensionless). A

Table 2.3: Analogies between heat flow and electrical current transport [6].

Parameter	Electrical Current		Heat	
current	I	[A]	P	[W]
current density	j	[A/m^2]	q	[W/m^2]
potential or temperature	V	[V]	T	[K or °C]
conductance or conductivity	σ	[℧]	k	[W/(m· K)]
resistance	R	[Ω]	R	[K/W]
capacitance	C	[F]	C	[J/K]

gray body is a body that doesn't absorbs all incident radiation; it partly reflects incident radiation.

The total heat flux of a body via radiation is equal to the incident energy minus the radiated energy:

$$Q_r = \epsilon_1 \sigma A T_a^4 - \epsilon_2 \sigma A T^4 = \epsilon \sigma A (T_a^4 - T^4), \tag{2.28}$$

when assuming that $\epsilon = \epsilon_1 = \epsilon_2$; ϵ_1 is the emissivity of the background deemed a black body at temperature T_a, also referred to as the ambient radiation, and ϵ_2 is the emissivity of the body itself. Equation 2.28 can be rewritten as:

$$Q_r = h_r A (T_a - T), \qquad \text{where} \qquad h_r = \epsilon \sigma \frac{T_a^4 - T^4}{T_a - T}. \tag{2.29}$$

The radiative heat transfer coefficient h_r is clearly a non-linear function of T, which will be challenging to deal with in Chapter 5.

2.3.3 Current-Thermal Equivalence

The thermal analysis of complex systems is frequently modeled via equivalent electrical circuits laws [6, 55, 131]. Well-established tools are available to solve electrical circuits, e.g, Kirchhoff's current and voltage laws, which can then be applied to transient and steady-state heat transfer problems. For example Equation 2.25 can be thought of as the current flowing through a resistor, where $T_s - T_a$ is equivalent to the voltage drop over a resistor and $\frac{1}{h_v A}$ is equivalent to the electrical resistance. Then the heat transfer rate Q follows the same law as a current flowing through a resistor. Indeed, the definition of the heat transfer rate, Fourier's law (Equation 2.23), is equivalent to the electrical current I, Ohm's Law:

$$I = A\sigma \frac{\partial V}{\partial n},$$

where σ [Ω/m] is the electrical conductivity. Table 2.3 shows the complete list of analogies between heat flow and electrical current transport.

Chapter 3

Temperature/Power Relationship in Microprocessors

When studying the energy consumption based on power measurements it is necessary to obtain accurate power samples and reproducible traces. Indeed, the temperature has a measurable impact on power measurements of microprocessors. In the example of Figure 2.4 a 5% power consumption increase was demonstrated for a 10°C temperature increase. Previously, it was also pointed out that the leakage currents are temperature-dependent, Equation 2.10 was put forward as an adequate approximation. Consequently, the power consumption of a microprocessor is also temperature-dependent and hence, the energy consumption of the microprocessor is affected accordingly.

The relationship between the temperature and power consumption is demystified here. Literature models are surveyed and experimental measurement traces are used to validate some of the models. Practical usage of the temperature/power models are demonstrated with the aim of improving accuracy. Also, the importance of the temperature sensor's location relative to heat sources is highlighted. Another transformation model is advanced that captures, and cancels, the behavior of remote temperature sensors. When a temperature sensor is placed at a distance from a heat source, the sensor will measure the temperature with a time delay and diminished magnitude. The presented temperature transformation model attempts at canceling such effects. This chapter deals with transforming power traces, with a focus on the system's temperature, to arrive at meaningful and reproducible power samples.

3.1 Introduction

Under normal operation conditions, the temperature of the microprocessor's internals varies continuously depending on the present and past load of the microprocessor, and the ambient temperature. When the temperature increases, the failure rate of the system increases accordingly due to several side-effects [60], e.g., electromigration, chemical reactions, dielectric breakdown, and creep in the bonding materials. The transient temperature will also induce variable power characteristics, and hence also affect the energy consumption. As a consequence, because of the temperature-dependent energy consumption, to have a fair comparison of energy consumption between the execution of different code pieces, one should compare the measurements at a reference tempera-

ture. For example, basic experimental smartphone power measurement traces may show that a temperature difference of only 10°C can increase the power consumption by about 7% in the worst-case scenario [30]. Also, TMUs and DTMs may benefit from accurate temperature/power models.

Ideally, to model the temperature dependency of the power, a formulation for the current I_{cpu} is needed where the temperature T can be isolated from possible other parameters $g(\cdot)$: $I_{\text{cpu}}(T, \cdot) = f(T)g(\cdot)$. In such a formulation $f(T)$ is a scaling factor of $g(\cdot)$, and hence of I_{cpu}, that is independent of the system's technicalities, e.g., logic gate states or physical dimensions. Finding a temperature scaling factor for the power and energy consumptions is however not a straightforward task. Nevertheless, approximative scaling factors for the leakage currents, which contribute to the power consumption, have been analytically obtained or experimentally defined via simulations (mainly SPICE[1]) [36, 68, 69, 108, 114]. Most of the theoretical approximations are derived from the leakage current definitions defined by the BSIM [70]. From our practical experience with experimental temperature and power measurement traces, none of the models cited above were able to satisfactorily describe collected power data from our testbeds. This is because the rationale on which these approximations are based, assume conditions that are not entirely realistic, to simplify the leakage current micro models. Most previous research works focus solely on the sub-threshold leakage effect, neglecting other leakage effects. This may not be appropriate for deep sub-micron technologies [3, 132]. Moreover, existing models require the knowledge of multiple (transistor) parameters and the leakage current at a reference temperature, which is not always available at hand or at run time.

3.2 Contributors to Temperature Fluctuations

Let us recall that the microprocessor's power consumption P_{cpu} is the result of all currents that flow within the microprocessor accounted for by different physical processes [29]. Firstly, the *dynamic power* P_{dynamic} is the power used to charge and discharge the capacities that drive the logic gates within the microprocessor. Also, leakage currents flow through the transistors as a result of their non-ideal behavior, resulting in a leakage power P_{leak}. The total microprocessor power consumption can then be thought of as the sum of the mentioned processes, whence

$$P_{\text{cpu}} = P_{\text{dynamic}} + P_{\text{leak}}. \tag{2.11}$$

P_{leak} is known to be temperature-dependent and well defined for an isolated transistor. P_{dynamic} is prone to temperature fluctuations resulting from changing physical properties. The voltage supply level may be temperature-dependent as well. P_{cpu} is dissipated as heat and contributes to the *internal heat generation*. The total internal heat generation is a result of the combined effect of the mentioned processes whose relations are not totally clear, and moreover could lag in time. Therefore, further on, the modeling of the temperature/power relationship is approached from a macro-level perspective such that the aggregated behavior of the mentioned, and possibly other, physical processes are captured. Some sources of temperature dependency are highlighted in the following sections.

[1] SPICE (Simulation Program with Integrated Circuit Emphasis) is an open-source widely used analog electronic circuit simulator for general purposes.

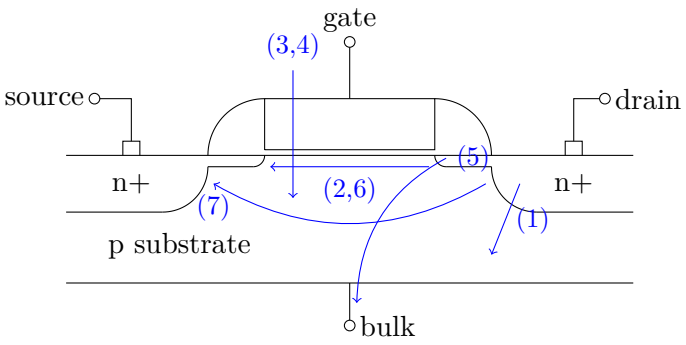


Figure 3.1: Schematic overview of the NMOS, or N-channel, transistor's internals and the different sources of leakage current are shown: (1) reverse-bias pn-junction, (2) sub-threshold, (3) gate-oxide tunneling, (4) hot-carrier injection, (5) GIDL, (6) DIBL, and (7) channel punch-through. Gate-oxide tunneling occurs during the *on*-state of the transistor; the other currents manifest during the *off*-state; hot-carrier injection may also occur during transistor bias state in transition.

3.2.1 Leakage Currents

Leakage current effects are inherent to the silicon-based metal-oxide semiconductor field-effect transistors (MOSFETs) that build up the microprocessor nowadays. These leakage effects originate from currents that flow between the differently doped areas, terminals and insulators, or through its junctions. The leakage currents are dependent on the state of the transistor, i.e., the terminal voltage levels. Figure 3.1 shows a schematic overview of a NMOS, or N-channel, transistor's internals and seven distinct sources of leakage are identified [22, 59, 70, 102, 129]:[2]

1. *reverse-bias* pn-*junction leakage*: this is a small leakage current flowing in the reverse voltage bias direction of the *pn*-junction, due to electron-hole pair generation in the depletion layer and minority carriers, which diffuse to the junction. The reverse-bias *pn*-junction leakage is a function of doping concentration and junction area;

2. *sub-threshold leakage* or *weak inversion leakage*: it occurs when an N-channel transistor is in *off*-mode, i.e., the voltage bias between the ground and the source is smaller than the threshold level. Then a leakage current flows between the drain and the source due to a minimal electrical field in the channel, which allows for carriers to diffuse across. The sub-threshold leakage current is claimed to be the largest leakage component and increases considerably when the technology is scaled down;

3. *gate-oxide tunneling current*: nano-scale oxide layers, or insulating layers, reduce the width of the energy barrier that separates the transistor's gate from the channel. Via quantum mechanical effects (QME) electron-hole pairs are therefore able to tunnel easier through the insulator layer, e.g., via Fowler-Nordheim tunneling. Gate oxide tunneling leakage current becomes important if the gate-oxide thickness is scaled down below 3 nm. High-k dielectrics are developed to replace gate-oxide in order to reduce the tunnel current;

[2] Analytical formulations for the leakage currents can be found in the BSIM manual [70].

4. *hot-carrier injection*: electrons-hole pairs, with larger than average energies, are attracted to the gate node due to very high electric fields in the drain region. The hot-carriers may be injected into the gate-oxide, degrading the oxide layer and the Si-SiO_2 interface, while producing gate and substrate leakage currents and compromising operation. Hot-carrier injection can be managed by larger channel lengths, which is contradictory to technology scaling;

5. *gate-induced drain leakage (GIDL)*: due to thin oxides, band-to-band tunneling (BTBT) currents arise between the drain and the substrate in the high electric-field deep-depleted drain region of the gate-drain overlap. GIDL is a complex mechanism that may cause other defects such as Fowler-Nordheim tunneling of electrons from the drain to gate;

6. *drain-induced barrier lowering (DIBL)*: when the drain voltage is enhanced, e.g., for faster inversion, the potential barrier near the channel surface decreases when the depletion region of the drain interacts with the source. Electrons are able to flow between source and drain because of the reduced potential barrier, even when the the gate-to-source voltage is lower than the threshold voltage. DIBL leakage can be reduced by shallower source/drain junction depths and higher surface and channel doping;

7. *channel punch-through current*: if the depletion regions surrounding the source and drain merge in the channel, punch-through occurs and a current is able to flow between the source and drain. At this point the gate loses control of the sub-gate channel region. Punch-through can be minimized with larger substrate doping, thinner oxides, shallower junctions, and longer channels. Channel punch-through is also regarded as a subsurface version of DIBL.

Gate-oxide tunneling occurs during the *on*-state of the transistor; the other currents manifest during the *off*-state; hot-carrier injection may also occur during transistor bias state in transition. Although there are multiple sources of leakage in MOSFET transistors, the sub-threshold leakage current, gate leakage, and GIDL dominate the others for sub-100 nm technologies [3, 132]. Leakage current models, e.g., as incorporated in the BSIM [70], are accurate, nevertheless complex since they depend on multiple parameters. Detailed knowledge of the transistors and states are necessary to assess the precise magnitude of the leakage currents, e.g., dimensions, materials, and terminal voltages. Even more, when transistors are stacked, e.g., in logic gates, leakage currents may be amplified throughout the stack [44, 132]. Moreover, as a microprocessor changes state each clock cycle, keeping track of the exact leakage current may practically be a daunting task.

The physical dimensions of transistors are scaled with each new technology and fabrication materials are improved to enhance their performance and power efficiency. One should therefore be careful to assess the magnitude of leakage currents based on the research of a decade ago. It has been shown that for example gate-oxide leakage currents become more prominent for shrinking transistor dimensions [72]. Mostly the sub-threshold leakage currents are accounted for in previous research, which is not necessarily a wrong assumption for the larger transistor sizes studied in the past. However, composed leakage current effects impose us to develop a model that captures aggregated behavior over time and temperature dependency, on a macro-level, for all transistors in the microprocessor.

Often coarse-grained models are inspired by the intrinsic behavior of a single transistor's leakage current. Table 3.1 provides an overview of models found in the literature.

Table 3.1: Literature models capturing the coarse-grained behavior of leakage currents. a_* are parameters; T is the microprocessor's temperature. The last two equations are popular choices as they are closely related to the BSIM formulation of the sub-threshold leakage current.

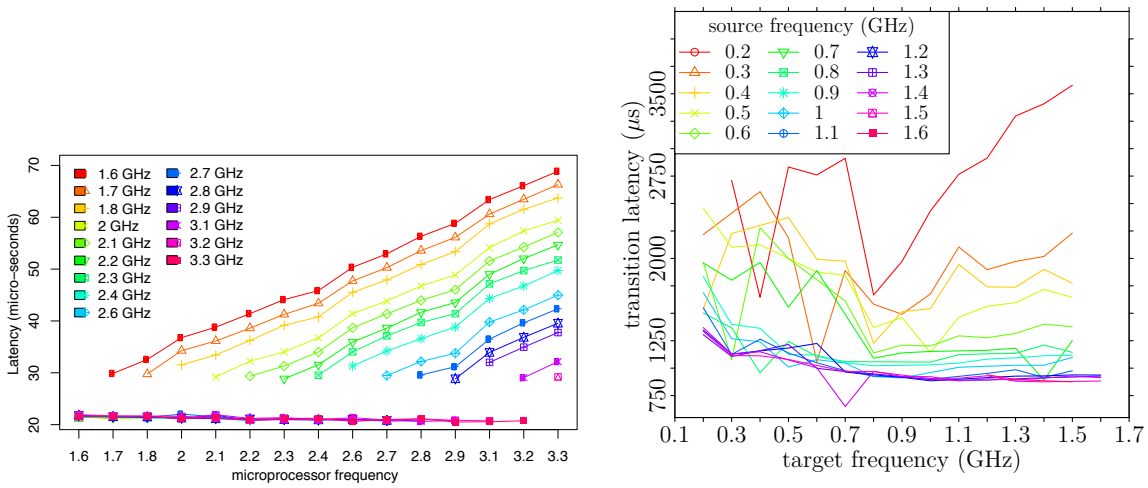
Authors	Model
Su *et al.* [114]	$P_{\text{leak}} \propto a_2 T^2 + a_1 T + a_0$
Liao *et al.* [69]	$P_{\text{leak}} \propto a_2 + a_1 T^2 e^{a_0/T}$
Liu *et al.* [71]	$P_{\text{leak}} \propto a_0 T + a_1$
Ferré and Figueras [36]	$P_{\text{leak}} \propto a_1 e^{a_0/T}$
Sinha and Chandrakasan [108]	
Liao *et al.* [68]	
Skadron *et al.* [109]	$P_{\text{leak}} \propto a_1 T^2 e^{a_0/T}$
Chandrakasan *et al.* [19]	$P_{\text{leak}} \propto a_2 (1 - e^{a_1/T}) e^{a_0/T}$
Butts and Sohi [13]	
Skadron *et al.* [141]	$P_{\text{leak}} \propto a_2 T^2 (1 - e^{a_1/T}) e^{a_0/T}$

Liao *et al.* [69] stated that for their 65 nm benchmark the sub-threshold and gate leakages dominate the leakage process. Only the former is temperature dependent, the authors claim. In another paper by Liao *et al.* [68] a power consumption model for adders was presented. The temperature-dependent part of the power model looks slightly different from their previous work. Skadron *et al.* [109] deducted a relationship between the leakage power P_{leak} and dynamic power P_{dynamic} based on International Technology Roadmap for Semiconductors (ITRS) power traces. It can be observed that their equation is inspired by the sub-threshold leakage current. In another publication, Skadron *et al.* [141] adopted the exact formulation of the sub-threshold leakage current for the transistor in *off*-state. Liu *et al.* [71] put forward a linearized leakage current equation based again on the gate and threshold leakage currents, which was studied via SPICE simulations. Yet, in their humble attempt to model the thermal behavior of leakage current with finite elements, they forgot to account for the inflating power consumption due to leakage currents. Su *et al.* [114] modeled the leakage currents of so-called *standard cells* based on SPICE and custom thermal simulations. The authors identified a satisfactory quadratic correlation between the temperature and leakage currents. Chandrakasan *et al.* [19], and Butts and Sohi [13], adopted the exact formulation of the sub-threshold leakage current from the BSIM manual. Ferré and Figueras [36], and also Sinha and Chandrakasan [108], based on an approximation of the sub-threshold leakage current, assumed a pure exponential relationship between temperature and leakage. This is a justified simplification of the sub-threshold leakage current expression as the $(1 - e^{-V_{\text{dd}}/V_{\text{T}}})$ component, where V_{T} is the thermal voltage, turns out to be close to one for all V_{dd}.

Most authors assert that they were able to model the leakage current adequately for their dedicated testbed. This implies that leakage currents may very well be application-specific, i.e., for given transistor dimensions and materials etc. A simulation-based approach is not surprising as physically measuring the leakage current on a real testbed is not a straightforward task. Though, a glimpse of its behavior can be observed by controlling the system's temperature levels.

3.2.2 Voltage Regulators

Each microprocessor has a voltage regulator that supplies the microprocessor with a constant voltage supply. The voltage regulator of a DVFS-enabled microprocessor can alter the magnitude of the supplied voltage and clock-frequency on demand, though with a



(a) SandyBrigde (b) Exynos 4210

Figure 3.2: Microprocessor clock frequency transition latency. The target frequency (GHz) is on the abscissa, the initial frequency is color coded. On the left, for the SandyBrigde (4 cores) as published by Mazouz *et al.* [75]. On the right, transition latencies are measured as seen from the application layer on the Exynos 4210 testbed. For the SandyBridge, at the instruction level, the transition latency is below 80 μs, whereas from the application layer, for the Exynos 4210, the transition latency is below 3.5 ms.

small transition delay. This transition delay is frequency-dependent. For high-end microprocessors the transition delay remains below 100 μs at the instruction-level [75], as shown in Figure 3.2. Also, frequency transition latencies are shown as measured on the Exynos 4210 testbed (defined in the sequel) at the application layer. Both graphs show the transition latency's frequency-dependency. The transition latency for the Exynos is considerably larger, with a maximum around 3.5 ms, as the system call has to propagate through the operating system to arrive at the actual instructions that trigger the frequency transition. Moreover, such system calls may be interrupted and deferred by other processes inside the kernel, which introduces some noise as can be observed in Figure 3.2(b).

A voltage regulator is build up from resistors, capacitors, inductors, and transistors. Therefore the voltage regulator, affected by temperature fluctuations, may also supply a voltage which is temperature-dependent. The resulting macro-level effect is that, for a fixed resistance, the current supplied to a resistor will increase and augment the internal heat generation. For example, if a 1.5 voltage drop over a 1 kΩ resistor increases by 1%, the resistor's power dissipation increases approximately 2%.

The temperature sensitivity of the voltage regulator is usually listed in its datasheet and probably minimized by design. The specifications for the S2MPS11 voltage regulator of the ODROID testbed (defined in Section 3.5) are unfortunately not available. But its temperature drift can be estimated via the onboard INA231 voltage sensor. In the most extreme case[3] a voltage rise from 1.25 V to 1.265 V was observed between 30°C and 90°C for the A15 microprocessor, which is a 1.2% voltage increase. The maximum gain error for the INA231 is listed to be 0.5%. This leaves us with an estimated voltage regulator temperature drift of around 0.8%. For the more energy-efficient A7 microprocessor in idle mode a 0.25% rise was noted in the most extreme case. The quantization noise and gain

[3] All microprocessor's cores busy at the maximum frequency.

error, however, render the latter observation unreliable.

In general, the voltage regulator may not always be exposed to the full temperature swings stemming from switching logic inside the microprocessor. This depends on the distance of the voltage regulator to the microprocessor. As a result, during peak microprocessor activity, escalating internal heat generation, resulting from leakage current swells, will kick-in faster than the increased dissipation from the inflating voltage supply. This is a result of the finite propagation time of heat between the microprocessor's logic and the voltage regulator. As an illustration, in both our testbeds the voltage regulator (S2MPS11 and MAX8997) is located about 1 cm away from the actual application microprocessor. Practically this implies that the transient thermal behavior of a microprocessor will look different when the whole system is heated homogeneously, e.g., while being exposed to the sun, than when a temperature rise in the microprocessor emanates from the execution of a job.

3.2.3 Physical Properties of the Processor

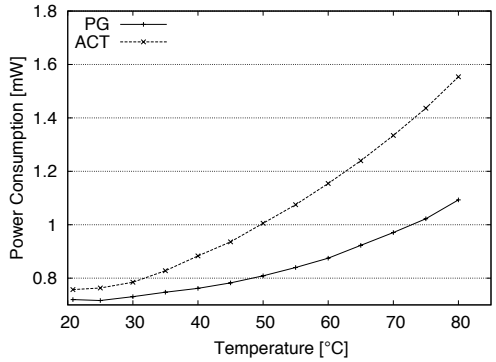
Physical properties of the materials that constitute the microprocessor, and the computer system, e.g., packaging and printed circuit board (PCB), are temperature-dependent, including but not limited to, the electrical resistance and thermal diffusivity. For example the resistivity of copper and aluminum increases about 12% between 0°C and 50°C. A practical illustration: a thick film resistor's, e.g., surface mount device (SMD), electrical resistivity decreases 1% over 50°C. The electrical resistivity of semiconductors typically decreases with rising temperatures. Similarly, the thermal conductivities of both copper and aluminum change about 0.9% between 0°C and 50°C. The thermal characteristics of silicon depend also on the type of dopant, and most likely the concentration [76]. Ramalingam *et al.* [91] claims that omitting the influences of the thermal conductivity nonlinearities of silicon could lead to a microprocessor temperature profile that is off by 10°C,[4] and could cause inaccurate results in reliability analysis. However, given the complex mix of chemical elements in microprocessors and computer systems the aggregated thermal effect of their physical properties is in practice unclear. This question is, however, outside the scope of this work.

3.3 Temperature/Power Models in the Literature

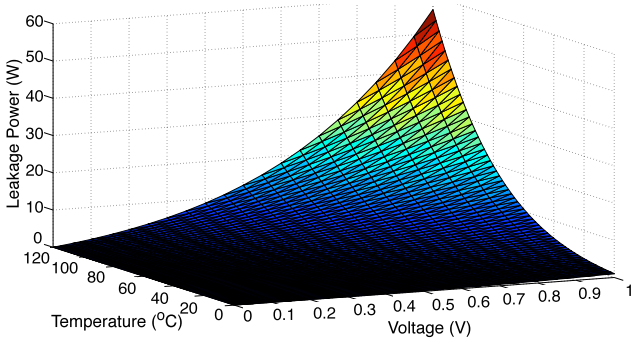
With the objective of adequate performance within certain temperature constraints, DVFS controllers may employ temperature/power models. Also for TMUs or DTMs it may be useful to understand the thermal behavior of the temperature/power relationship. Computer energy consumption decompositions account often for the leakage currents where the temperature/power dependency is referenced. A summary of temperature/power data and models found in the literature is listed below. Figure 3.3 shows excerpts of the data portrayed in the papers.

Weissel and Bellosa [127] developed a TMU for data center computers. Based on a handful of temperature/power measurements in a limited temperature range (35°C to 60°C) they assumed the temperature/power relationship to be quadratic, quasi linear. The accuracy of the fitting is however questionable, as the linear fit doesn't seem to be the least square, and the quadratic fit is dominated by an outlier. Hanumaiah and

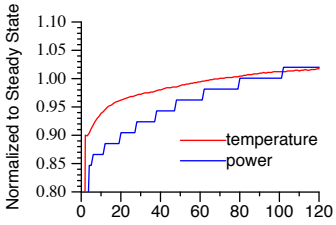
[4] Even though the 10% temperature bias is apparently well motivated by the authors, it appears large.



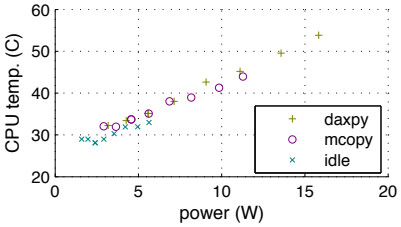
(a) Ikebuchi *et al.* [56]



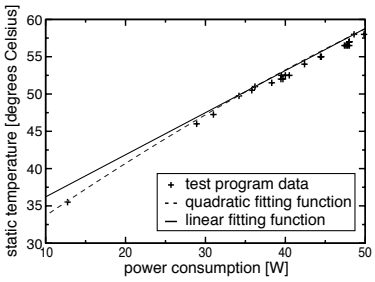
(b) Hanumaiah and Vrudhula [52]



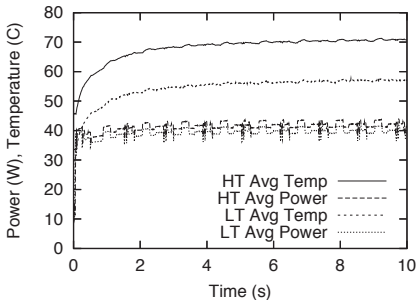
(c) Singh *et al.* [107]



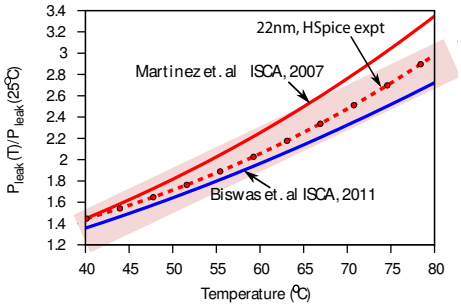
(d) Hansom *et al.* [50]



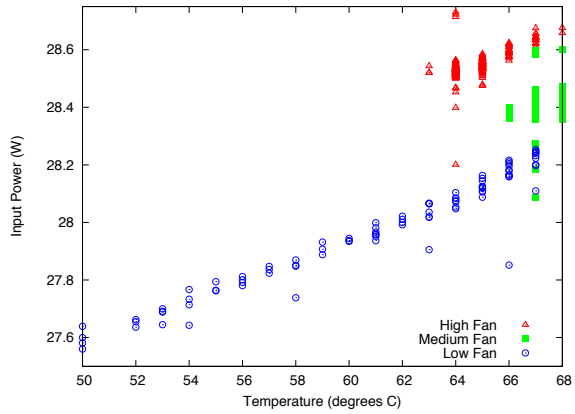
(e) Weissel and Bellosa [127]



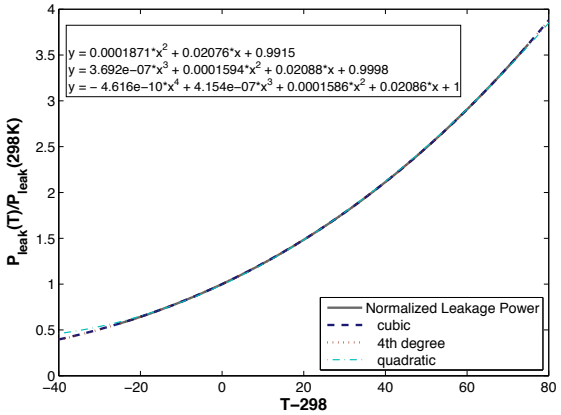
(f) Mesa-Martinez *et al.* [78]



(g) Sarangi *et al.* [100]



(h) Snowdon [110]



(i) Biswas *et al.* [11]

Figure 3.3: Excerpts of temperature/power plots as found in the literature. All figures were originally published in the papers referenced in the respective captions.

Vrudhula [52] developed a DVFS controller for systems with hard real-time and temperature constraints. They employ a linearized version of the exponential temperature/power assumption, which was based on the BSIM leakage current models. The authors also assume that the power increases linearly with the supply voltage. The temperature in their experiments ranged between 35°C and 110°C. While studying the thermal response to DVFS of an Intel Pentium M microprocessor, Hansom *et al.* [50] assumed a linear relationship between power and temperature. The temperature ranged between 20°C and 55°C in their experiments. Singh *et al.* [107] attempted to model an AMD microprocessor's power consumption based on a subset of performance counters. Temperature/power traces are shown but only with relative figures. Their traces show that during the execution of some benchmarks the microprocessor's temperature inflates about 10% (of °C), resulting in a 20% power increase. With a bit of good will a super-linear relationship can be identified. Ikebuchi *et al.* [56] shows temperature/power traces for their Geyser-1 MIPS microprocessor. The temperature/power relationship, measured between 20°C and 80°C, shows a clear exponential relationship. The authors also show that with the help of power-gating the effects of leakage currents on power consumption can be diminished. Mesa-Martinez *et al.* [78] shows empirical results for what they refer to as *an unnamed modern high-performance microprocessor* with high-resolution infrared cameras to obtain accurate die temperatures. Their traces show temperatures between 20°C and 70°C. Snowdon [110] analyzed the power consumption and microprocessor temperature of his laptop for the development of an energy-aware operating system. He also assessed the impact of the fan on the temperature of the system and its power consumption.

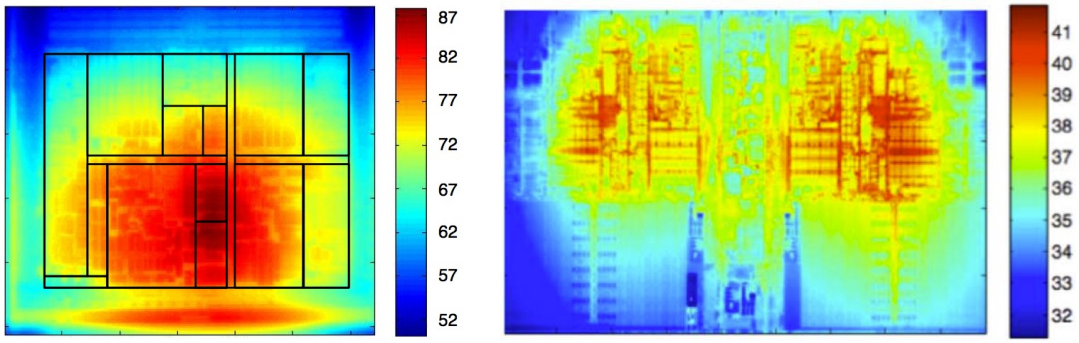
In the context of the design of an ultra-fast temperature simulator Sarangi *et al.* [100] quantified the temperature dependency of the leakage power empirically based on HSPICE simulations between 45°C and 70°C. The authors deem a linear approximation appropriate. Similarly, using HSPICE, Biswas *et al.* [11] evaluated the temperature dependency of the leakage power between -10°C and 100°C, for a power management technique aiming to avoid thermal hot-spots. The authors adopted a third-order polynomial to represent the temperature dependency. Quang and Zhang [89] studied the feasibility of hard real-time periodic task set scheduling under the peak temperature constraint. For this purpose the authors approximated the temperature/power analysis linearly between 40°C and 110°C based on the analytical formulations of a subset of leakage currents.

All the work listed above shows temperature/power traces for at most three benchmarks and for specific microprocessor settings, or small dedicated circuits, often for illustrative purposes. Usually high-performance MIPS microprocessors are targeted as the objects of study, as part of large server farms. The authors claim that their data is representative for their specific platforms.

Based on elaborate measurements further on, an experimental temperature/power relationship is identified for different microprocessor configurations and loads for our application microprocessors. Such application microprocessors are expected to function in embedded systems, e.g., smartphones or appliances.

3.4 Temperature Transformation Model

To obtain a satisfactory accurate microprocessor's temperature/power model, one must be sure that the temperature is measured at the source of the power dissipation. If the power dissipation is not uniform in space, i.e., thermal hot-spots or thermal gradients are present,



(a) Mesa-Martinez *et al.* [78] (b) Reda *et al.* [93]

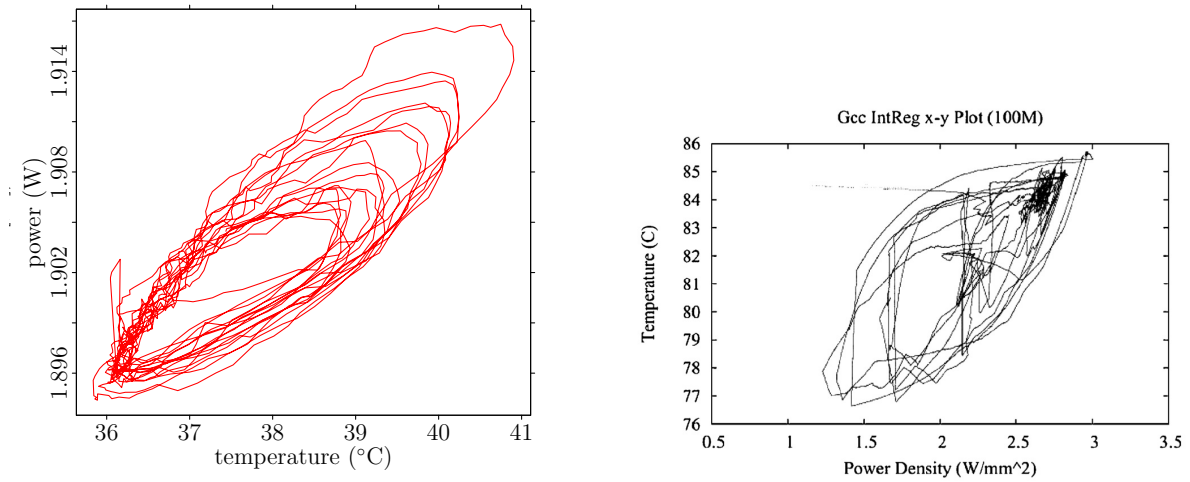
Figure 3.4: Examples of thermal images for two high-performance microprocessors. The temperature varies between 87°C to 60°C on the left, respectively 42°C and 34°C on the right. It is observed that the temperature is not uniform.

getting an accurate temperature representation becomes more challenging. Temperature sensors don't always yield an accurate representation of the actual temperature where the majority of the power is dissipated. In these cases, a temperature transformation model could be used to transform the temperature measurements as they would have been measured at a particular location. This requires knowledge of the system's thermal properties and the events that take place at the source of the heat generation. Such a transformation model is discussed in the sequel.

3.4.1 Optimal Temperature Sensor Placement

The location of on-die temperature sensors is critical to obtain a realistic thermal view of what is happening in the microprocessor with the aim of controlling heat dissipation and preventing thermal runaway situations. Such thermal runaway situations, and mechanical stress due to thermal gradients, lower the MTTF and may lead to irreversible destruction in the microprocessor, and eventually causing complete hardware failure [17]. In realistic environments the microprocessor's die temperature is not thermally uniform, and commonly referred to as non-isothermal. Thermal hot-spots exist, which are located in the parts of the microprocessor which are most active. For example, the FPU will be a large source of heat generation when many floating points operations are executed whereas the address decoder is a possible source of heat when many memory operations are performed. For a microprocessor that is cooled excessively, e.g., with forced air cooling, hot-spots will be more prevalent than for systems where the heat has to dissipate passively. Heat prefers the path of least resistance, which is usually through the heat spreader instead of through the microprocessor's silicon itself. A heat spreader usually connects the microprocessor's silicon with some sort of heat sink.

Figure 3.4 shows practical examples of thermal hot-spots in microprocessors. Mesa-Martinez *et al.* [78] measured the temperature of an unnamed modern high-performance microprocessor with a $10\times10\,\mu$m spatial resolution infrared (IR) camera running `crafty` from the SPECINT2000 benchmark. The thermal hot-spot's center is around 87°C whereas the borders of the chip are around 60°C. Reda *et al.* [93] shows thermal images of a dual-core AMD Athlon II 240 microprocessor while running various CPU SPEC06 workloads at 2.1 GHz. As can be observed, the hottest areas are around 42°C and the cooler parts are around 34°C. The figures exemplify well how the placement of the temperature sensors



(a) Samsung Galaxy S2

(b) Skadron *et al.* [109]

Figure 3.5: Temperature/power hysteresis loops induced by the *distant-sensor-syndrome*. Heat needs a finite time to propagate from a heat source to a temperature sensor. If this propagation time-lag is sufficiently long, temperature/power loops may be observed as shown for the Samsung Galaxy S2 testbed (a). Temperature/power hysteresis loops may also emanate from data smoothing and large heat capacities such as visible in Skadron *et al.*'s measurement traces (b).

is essential to grasp a realistic view of the microprocessor's static and transient thermal behavior. Also the effectiveness of TMUs and DTMs will depend on the availability of accurate and representative thermal data. Work has been devoted to optimal temperature sensor placement. The aim is to minimize the number of thermal sensors, while guaranteeing full detection of all hot-spots and worst-case temperature gradient [92]. An overview of optimal temperature sensor placement strategies can be found in the work of Li *et al.* [67] and Kong *et al.* [60].

What happens exactly when a sensor is placed at a finite distance from a thermal hot-spot? Heat needs a finite time to propagate through a solid or fluid from the hot-spot to the thermal sensor. Consequently, a time delay will occur between the actual heat generation and the measurement at the heat sensor. Moreover, the magnitude of the temperature at the temperature sensor will also be smaller, as the heat density decreases when it spreads out over a space. This hysteresis effect is addressed as the *distant-sensor-syndrome*. The temperature time-lag of the distant-sensor-syndrome can be observed when loops appear in power/temperature graphs, which always rotate clock-wise when the temperature is on the abscissa. This is because it is the temperature that lags behind on the power dissipation. Figure 3.5 shows temperature/power loops as measured on the Galaxy S2 platform (defined in Section 3.5), and measured by Skadron *et al.* [109]. Skadron *et al.* pointed out that said behavior emanates from thermal capacities acting as low-pass filters. Temperature/power loops may also be observed when the data is averaged, or smoothed, over a large time window. The exponential moving average (EMA) is more affected by such side-effects than a symmetric simple moving average (SMA) as EMA is in essence a low-pass filter. Temperature/power loops are more pronounced for data recorded over a non-fixed load of the microprocessor. In Skadron *et al.*'s work, a mixture of a non-fixed microprocessor load and thermal capacities is likely the cause of their temperature/power loop. Temperature/power loops are frequently present in measurement data, even for on-die temperature sensors, however, more subtle than shown

in Figure 3.5.

To prevent the consequences of the distant-sensor-syndrome, a transformation model, as a function of space and time, is necessary to translate the sensor's temperature data as if it were measured at the heart of the power dissipation. In the next section a heat diffusion model is developed to arrive at a temperature transformation model. As the temperature sensor is located in the vicinity of the hot-spot, in the order of a few centimeters, only the conductive heat transfer mode is considered. Also, one should distinguish between *off-chip* and *on-chip* temperature sensors. For off-chip temperature sensors the transformation model will probably be more complex as the heat traverses a more elaborate path from the heat source to the temperature sensor.

3.4.2 Heat Diffusion Models

The heat diffusion in solids is a well-studied subject based on Fourier's equation of heat diffusion (Equation 2.23). A microprocessor is mostly a slice of heterogeneous material based on Silicon (Si) but doped or combined with other chemical elements such as oxygen, boron, arsenic, or phosphorus. For simplicity the microprocessor is deemed henceforth an isotropic material.

The simplest application of Fourier's law is a semi-infinite one-dimensional slab of homogeneous material; that is, an infinite slab of material with a heat source applied at its origin. Heat then propagates from the origin into infinity. When a heat source with constant temperature is applied to the origin, a time-dependent heat flux will flow into the semi-infinite material. The solution of Fourier's law $T(x,t)$, with the initial and boundary conditions: $T(x,0) = T_i$ and $T(0,t) = T_s$, is then given by

$$T(x,t) = (T_i - T_s)\,\mathrm{erfc}\left(\frac{x}{\sqrt{4\alpha t}}\right) + T_i, \tag{3.1}$$

where α is the heat diffusivity of the material, and `erfc` is the complementary error function [17]. When a constant heat flux q is applied at the origin, $x = 0$, of the semi-infinite slab of homogeneous material the solution becomes much more complex

$$T(x,t) = 2\frac{q_0''}{k}\sqrt{\frac{\alpha t}{\pi}}e^{\frac{-x^2}{4\alpha t}} - \frac{q}{k}x\left[\mathrm{erfc}\left(\frac{x}{2\sqrt{\alpha t}}\right)\right]. \tag{3.2}$$

For an annulus extruded in the third dimension, with outer radius b, inner radius a and height z, cooled on the bottom with a convective heat transfer coefficient h, and a constant heat flux applied to its inner surface at $r = a$, the steady-state solution to Fourier's law is given by:

$$T(r) = \lambda K_0\left(\frac{r}{L}\right), \quad \text{and} \quad -k\frac{\mathrm{d}T(r,\lambda)}{\mathrm{d}r}\bigg|_{r=a} = \frac{P}{2\pi bz}, \tag{3.3}$$

where $K_0(\cdot)$ is the modified Bessel function of the second kind of order zero, k is the heat conductivity of the material, P is the dissipated power, L is the characteristic length $\sqrt{kz/l}$, and z is the height of the cylinder [121]. The annulus model could serve as a first approximation for an on-chip temperature sensor transfer function. However, if the temperature sensor is mounted on a plate, e.g., PCB, next to the actual heat generator then a more elaborated model must be deployed. Let us think about the PCB as a finite disk, in cylindrical coordinates, with radius b, height z, and a heat source is located at

the origin with radius a injecting a constant heat flux Q into the disk. The analytical solution to such problems becomes increasingly difficult to derive. In the literature the steady-state solution of said problem is given if the bottom surface of the disk is cooled with a convective heat transfer coefficient h and the other surfaces are adiabatic [65]:

$$T(\gamma,\zeta) = \frac{qa}{k}\left[\epsilon\left(\frac{1}{\mathrm{Bi}}+\zeta\right) + 2\sum_{n=1}^{\infty}\frac{J_1(\lambda_n\epsilon)J_0(\lambda_n\gamma)}{\lambda_n^2 J_0^2(\lambda_n)}\frac{\cosh(\lambda_n\zeta)}{\cosh(\lambda_n\tau)}\frac{\tanh(\lambda_n\zeta)+\frac{\lambda_n}{\mathrm{Bi}}}{1+\frac{\lambda_n}{\mathrm{Bi}}\tanh(\lambda_n\tau)}\right], \tag{3.4}$$

where the disk's axisymmetric coordinates for height j and radius r are normalized as $\zeta = j/z$ and $\gamma = r/b$, respectively, $\epsilon = a/b$, $\tau = t/b$, $\mathrm{Bi} = hb/k$, and $J_0(\cdot)$ and $J_1(\cdot)$ are Bessel functions of the first kind of the order zero and one, respectively. The eigenvalue λ_n is nth root that satisfies $J_1(\lambda_n) = 0$.

Additionally, Yovanovich [135] provides a similar solution when the sides of the cylinder/disk are also cooled, with a convective heat transfer coefficient h_s, and the cooled surfaces are not necessarily isothermal. Then the temperature difference θ of the cylinder with the ambient temperature is given by

$$\theta(r,z) = \sum_{n=1}^{\infty}[E_n\cosh(\lambda_n z) + F_n\sinh(\lambda_n z)]J_0(\lambda_n r), \tag{3.5}$$

$$\text{where } F_n = -E_n\phi_n, \quad E_n = \frac{Q}{\pi bk}\frac{2J_1(\delta_n\epsilon)}{\phi_n\delta_n^2[J_0^2(\delta_n)+J_1^2(\delta_n)]}, \quad \phi_n = \frac{\mathrm{Bi}_e+\delta_n\tanh(\delta_n\tau)}{\delta_n+\mathrm{Bi}_e\tanh(\delta_n\tau)},$$

and $\delta_n = \lambda_n b$, $\mathrm{Bi} = h_s b/k$, $\mathrm{Bi}_e = hb/k$. The eigenvalues λ_n satisfy $\delta_n J_1(\delta_n) = \mathrm{Bi}J_0(\delta_n)$. Equivalent equations are given by Ellison [32] for Cartesian coordinates and non-unity rectangular aspect ratios. Lee *et al.* [65] showed that the heat diffusion of a circular heat source applied to the surface of a cylinder is representative for the heat diffusion process of other geometries as well. Transformation functions are given to swap between geometries. Equations 3.1 and 3.2 are transient solutions whereas Equations 3.3, 3.4 and 3.5 are steady-state solutions. In fact, the last three equations were initially developed to assess the spreading resistance of cylinders. It is clear that for the steady-state solution the analytical definition is already rather complex. Adding transient behavior, and also non-linear components such as a temperature-dependent heat source (and later on radiation), will complicate the analytical modeling. Therefore numerical simulations, and a fair deal of fitting, will be used to arrive at the envisioned temperature transformation function.

3.4.3 Temperature Transformation Function

The goal is to seek a temperature transformation $H(\cdot)$ function that translates a temperature trace T_{sensor}, measured at a finite distance from a heat source, as if it were measured at the heat source T_{source}:

$$T_{\text{source}} = H(r,t,\ldots)\cdot T_{\text{sensor}}. \tag{3.6}$$

The transformation function $H(\cdot)$ may be a function of time t, distance to sensor r, temperature, T and others. To arrive at a first approximation for an off-chip temperature sensor let's assume some simplifications. As the sensor is assumed to be in the proximity of the heat source, it is presumed that heat conduction is the primary heat transfer mode. Also, the system is assumed to be a three-dimensional axisymmetric (r,z,φ) disk with radius b and height u. As such it is sufficient to model the system only in two-dimensions, i.e., r and z. A circular heat source with radius a is centered at ($r = 0, z = 0$). The heat

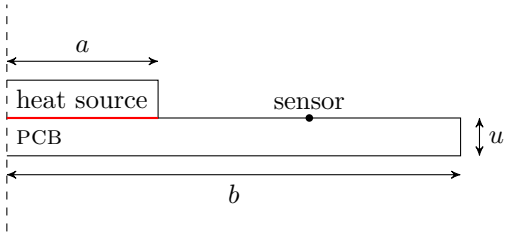
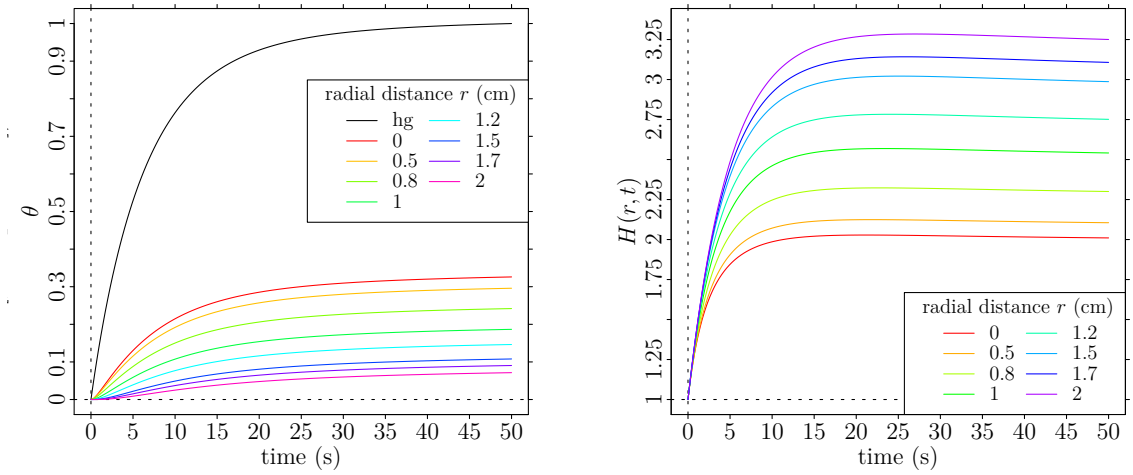


Figure 3.6: Axisymmetric model of a temperature sensor on a PCB at a finite distance from a heat source, representing a microprocessor. The PCB, essentially a thin disk, has a height z and radius b; the sensor is placed at a distance; the heat source has a radius a. The thermal resistance θ_{JB} between the heat source and the PCB is indicated in red.



(a) normalized temperature

(b) transfer function

Figure 3.7: Normalized transient thermal behavior of a heat generator (hg) injecting a temperature-dependent heat flux into a thin disk. The normalized temperature is shown for the heat generator and for different points on the thin disk at $z = 0$. A propagation delay is visible that becomes larger the further the measurement point from the origin.

source shares the thermal properties of a heat generator injecting a constant heat flux into the disk. The resistance θ_{JB} between the disk and the heat generator is also referred to as *junction-to-board* resistance. Figure 3.6 shows the axisymmetric model as described.

A numerical simulation was set up in COMSOL to mimic the thermal behavior of a microprocessor and a distant temperature sensor mounted on a PCB. The properties of the materials used are estimated for an isothermal microprocessor, centered at $r = 0$, with internal heat generation resembling the Exynos 4210 SoC. The microprocessor is mounted on a glass-reinforced polymer FP4 PCB with a temperature sensor at r =16.5 mm and $z = 0$.

Figure 3.7 shows the step response of a constant microprocessor load, and the generated temperature-dependent heat flux. The normalized transient temperature is shown at specific points on the PCB for $z = 0$. Because of the thermal resistance θ_{JB} between the heat source and the PCB a significant temperature drop exists between the two. Furthermore, it can be observed indeed that heat needs time to propagate a given distance from the heat source to the temperature sensor. This propagation delay is larger the further away from the heat source, most visible at $0 < t < 5$. If a sensor is placed 1 cm away from

the center of the heat source it takes about 9 s to reach 10% of the normalized equilibrium temperature. The farther away from the heat source the slower the temperature rises at that point.

The temperature transfer function $H(r,t)$ for each measurement point from Figure 3.7(a) is shown in Figure 3.7(b). The temperature transfer function was computed as the ratio of the temperature at the heat source over the temperature at the sensor:

$$H(r,t) = \frac{T_{\text{source}}(t)}{T_{\text{sensor}}(r,t)}, \tag{3.7}$$

where the temperature is expressed in degrees Celsius. The transfer functions shown in Figure 3.7(b) starting from $t = 0$ increase, have a maximum around 10 to 15 seconds and slightly decreases afterwards. Given that the analytical expression for the transfer function is complex, as shown in the previous section, the transfer function can be approximated via polynomial fitting. To get a fitting with acceptable error, it was observed that a polynomial approximation of an order higher than seven is required. For example, for a sensor at $r = 16.5\,\text{mm}$ from the origin, the temperature transfer function becomes

$$\begin{aligned} H(0.00165,t) = {} & 1.003 + 0.464t - 4.743 \times 10^{-2}t^2 + 2.827 \times 10^{-3}t^3 \\ & + 1.024 \times 10^{-4}t^4 + 2.195 \times 10^{-6}t^5 - 2.553 \times 10^{-8}t^6 + 1.235 \times 10^{-10}t^7. \end{aligned} \tag{3.8}$$

The transfer function $H(0.00165,t)$ has a maximum and average relative error of 0.362% and 0.092%, respectively. This transformation function only makes sense if the total heat flux generated by the internal heat source, without its temperature dependency, is indeed a step-function, i.e., if the temperature-independent heat flux Q_{ihg} equals

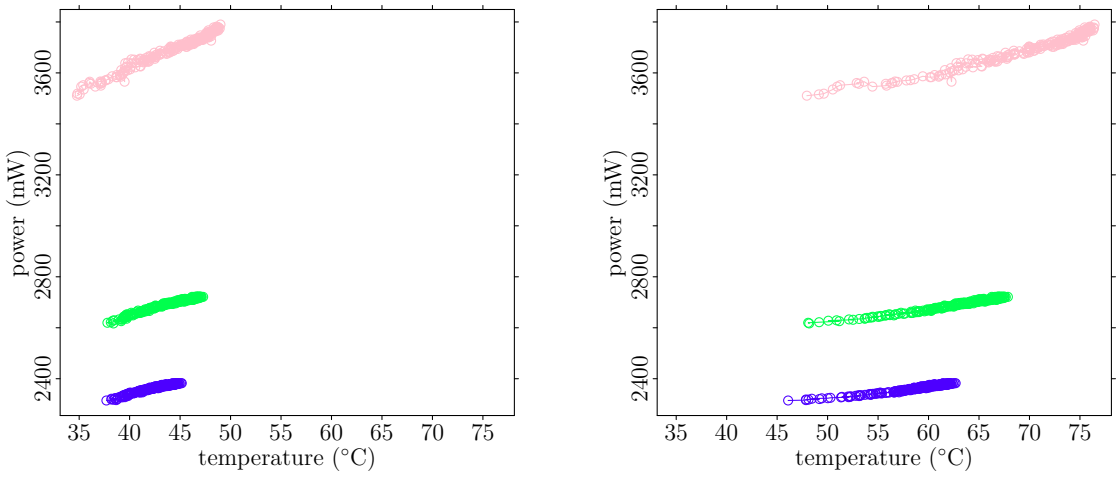
$$Q_{\text{ihg}} = \begin{cases} Q_1 & \text{if } t < 0 \\ Q_2 & \text{if } t \geq 0 \end{cases}, \quad \text{where } Q_1 \neq Q_2. \tag{3.9}$$

This corresponds to the power dissipation of a microprocessor whose load and type of computations change from one fixed state to another at $t = 0$.

As a practical example, Figure 3.8 shows temperature traces from an off-chip sensor at 16.5 mm from the center of a heat source. The traces were collected on a Samsung Galaxy S2, defined in the next section. Figure 3.8(a) shows the originally recorded traces whereas, on the right, in Figure 3.8(b), the transformed power traces are shown using Equation 3.8. The original traces show a more or less linear relationship, even slightly bending downwards, and reside around 35°C to 50°C. The transformed traces have an upward bending shape. This is what would be expected if the temperature sensor were close to a microprocessor's hot-spot. The temperature is also about 25°C larger for the transformed temperature cases. This is evident as the temperature at the heat source is always greater than at a finite distance of the heat source. In Section 3.6 the temperature transformation function $H(0.00165,t)$ of Equation 3.8 will be used to transform more extensive measurement traces.

3.5 Testbed Description

Whereas before the subject was tackled from an experimental point of view. Now, the temperature/power relationship is approached from a practical stand point. First, the hardware used is described. After, after the measurements are presented and discussed.



(a) original (b) transformed

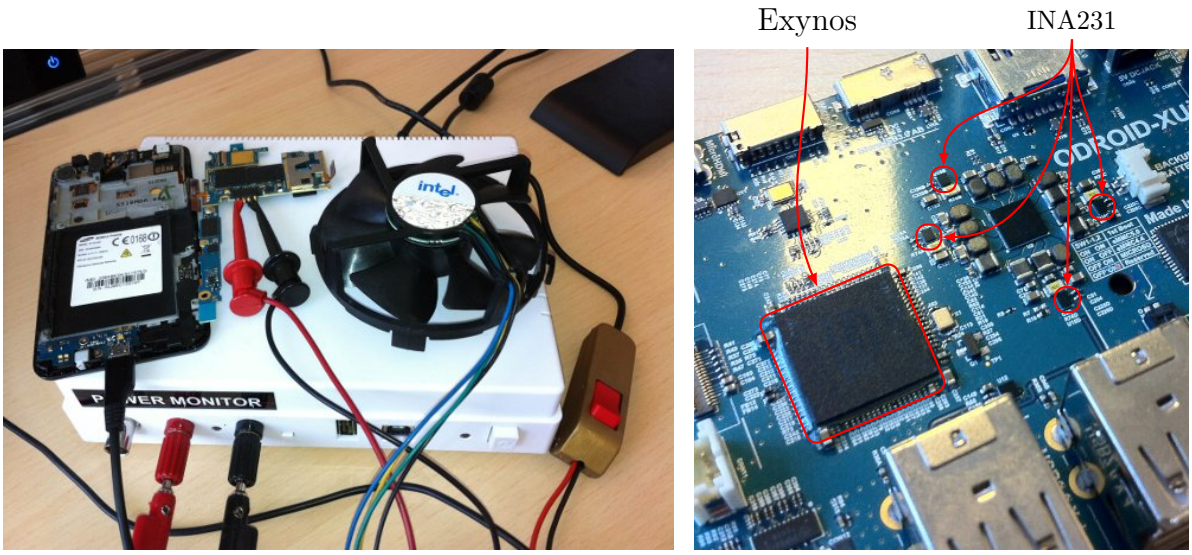
Figure 3.8: Original power/temperature traces (a) and their transformed counterparts (b) using Equation 3.8, a seventh-order polynomial. The original traces, measured at a 16.5 mm distance from the microprocessor, were transformed as they would appear at the center of the microprocessor's heat generation process.

3.5.1 Hardware

The following two platforms were used to collect temperature/power traces with the aim of assessing the temperature/power relationship; they will also be used in the next chapters to study the energy consumption of the platforms. Part of these platforms were chosen as they provide an adequate accuracy at a descent cost. Besides, they operate on open-source software, which is very useful for experimentation, and a descent amount of information, e.g., data sheets and PCB designs, is also available about the hardware.

A Samsung Galaxy S2 was used, sporting a Samsung Exynos 4210 SoC 45 nm dual-core, and a Hardkernel ODROID XU+E, featuring the Samsung Exynos 5410 SoC 28 nm quad-core. The Galaxy implements an A9 Cortex ARM microprocessor, whereas the ODROID has both an A7 and an A15 Cortex ARM microprocessor. The two platforms were running a custom compiled Linux kernel, version 3.0.31 and 3.4.67, respectively. Some kernel functions and modules were amended and added to the kernel, to enable swift temperature and power sample recording and retrieval, and minimize testbed perturbation during experimentation. The kernel frequency scaling governor was set to operate in *userspace* mode to prevent automated frequency and voltage scaling on-the-fly.

Figure 3.9 shows the Monsoon power monitor and the ODROID INA231s. The Monsoon power monitor replaced the battery of the smartphone which can be seen decomposed on top of the monitor in Figure 3.9(a). Also a fan, originally from a Pentium 4 processor's heat sink, is shown, which was used to provide a rudimentary temperature control of the smartphone. The temperature on both platforms were measured via on-board temperature sensors with a resolution of 1°C. Power and temperature samples were collected at a rate of 5 Hz. Forced cooling and forced heating were applied to the SoCs packaging (including the microprocessor) to force its temperature up and down. The left-most black chip on the PCB above the black and red clip probes is the Exynos 4210 SoC of the smartphone. The Monsoon power monitor has a Microsoft Windows-based GUI and produces power samples at 4 kHz with 1 mW accuracy. Unfortunately, the GUI made it impossible



(a) Monsoon Power Monitor and equipment (b) ODROID PCB

Figure 3.9: Power measurement devices as used hereafter. The Monsoon power monitor operates based on the current mirror principle, whereas the INA231 is a shunt resistor-based power consumption recorder: (a) the Monsoon power monitor replaces the battery, in this case, of a smartphone. (b) four INA231s are mounted on a PCB next to the power supply of a SoC.

to fully automatize the data acquisition. The Exynos 5410 SoC is the largest black chip in figure 3.9(b) and on its right, surrounded by other components, is the SoC's voltage regulator. Four INA231s with shunt resistors are located around the voltage regulator, measuring each a different aspect of the SoC. The INA231s are about 4 mm^2 large, are interconnected with a I^{2}C bus and can be read out via the Linux kernel at a maximum rate of 4 Hz and 1.25 mW accuracy. The read-out speed is limited by the I^{2}C protocol. Full automation with low overhead of power and temperature sampling is possible as the read-out of the temperature and power samples happen on a low level in the Linux OS. As the Monsoon power monitor replaces the battery in the Samsung Galaxy S2, and the ODROID is powered by an adapter, long-term nor short-term battery aging was addressed in this study.

3.5.2 Software

During the recording of the temperature and power traces a contentious quasi-constant load was applied to one or more cores of the microprocessors.

The Galaxy was loaded with 4096 kB *bit-reverse* calculations, part of the ubiquitous Cooley-Tukey Fast Fourier Transformation (FFT) algorithm, which rearranges deterministically elements in an array. The Gold-Rader implementation of the bit-reverse algorithm was used [42], as shown in Figure 3.10. The ODROID spun over the native square root function from the `math.h` library invoked by the `stress` program.

The root calculations were forked up to four times to assess the temperature/power impact of the four cores in the A7 and A15 microprocessor; inactive cores were not hot-plugged. On the A9 platform only one core was enabled; the other core was unplugged. It must be noted that the benchmarks ran on top of an OS, so there must be some power accounted for the system's overhead.

```
void bitreverse_gold_rader (int N, complex *data) {
    int n = N;
    int nm1 = n-1;
    int i = 0, j = 0;

    for (; i < nm1; i++) {
        int k = n >> 1;
        if (i < j) {
            complex temp = data[i];
            data[i] = data[j];
            data[j] = temp;
        }
        while (k <= j) {
            j -= k;
            k >>= 1;
        }
        j += k;
    }
}
```

Figure 3.10: The bit-reverse algorithm as per the Gold-Rader implementation [42]. The algorithm rearranges deterministically an array of elements of size N, in this case complex numbers implemented as a pair of integers. The bit-reverse algorithm is part of the Cooley-Tukey Fast Fourier Transformation (FFT) algorithm.

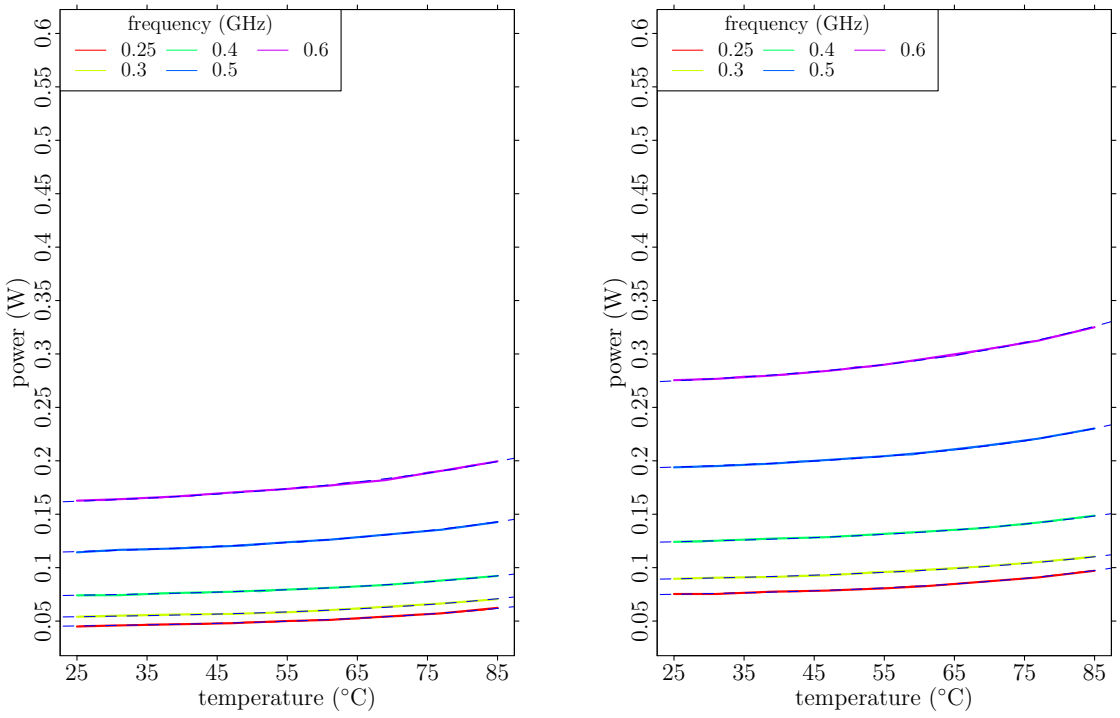
3.6 Temperature/Power Modeling

Now this chapter proceeds to model the power consumption and temperature/power relationship, using the temperature transformation models elaborated before to enhance the accuracy of the power measurements traces.

3.6.1 Measurements & Fitting Discussion

Because of the temperature dependency of some currents flowing through the microprocessor, the microprocessor's power consumption inflates for an increasing temperature. Figure 2.4 on page 14 showed a basic practical example of the temperature/power relationship. A measurement campaign was set up to assess the magnitude of this inflation on the A7, A9 and A15 Cortex microprocessors. The temperature of the A7 and A15 microprocessors was swept between 25°C and 85°C; for the A9 the temperature was swept between 25°C and 55°C. Unfortunately the A9 broke down, most likely due to excessive thermal stress; as a result the temperature ranges and benchmarks for the different microprocessors may differ. The power consumption and temperature were measured for the A7 between 250 MHz and 600 MHz, the A9, between 200 MHz and 1.6 GHz, and the A15, between 0.8 GHz and 1.6 GHz. The traces for the A7 and A15 are shown in Figure 3.11 and Figure 3.12, respectively.

The temperature/power traces were fitted with four models to assess their suitability. Preferably, the aggregated model's complexity should remain tractable for system-level design analysis and mathematical derivations. In such applications accuracy is often a trade-off with complexity. The curves considered are listed in Table 3.2, which were derived from the models of Table 3.1. The models $P = a_1 e^{a_2/T} + a_0$ and $P = a_1 T^2 e^{a_2/T} + a_0$ were also considered, but eventually left out, because they were unable to model the shape



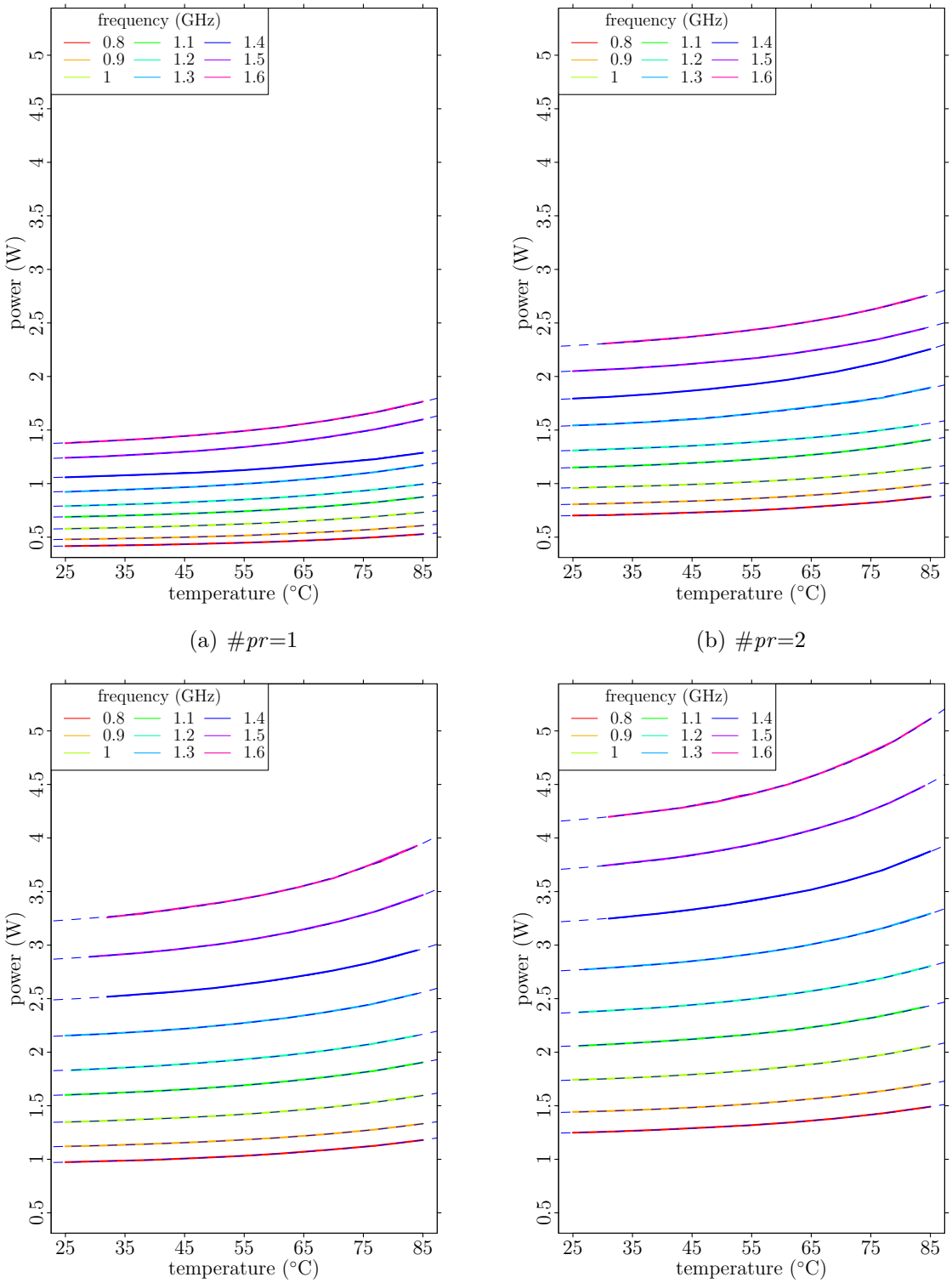
(a) $\#pr$=1

(b) $\#pr$=2

(c) $\#pr$=3

(d) $\#pr$=4

Figure 3.11: Temperature/power trace for the Cortex A7 microprocessor with $\#pr$ cores active between 250 MHz and 600 MHz. A super-linear temperature dependency on the power is clearly visible. The power increases when more cores are active.



(a) $\#pr$=1

(b) $\#pr$=2

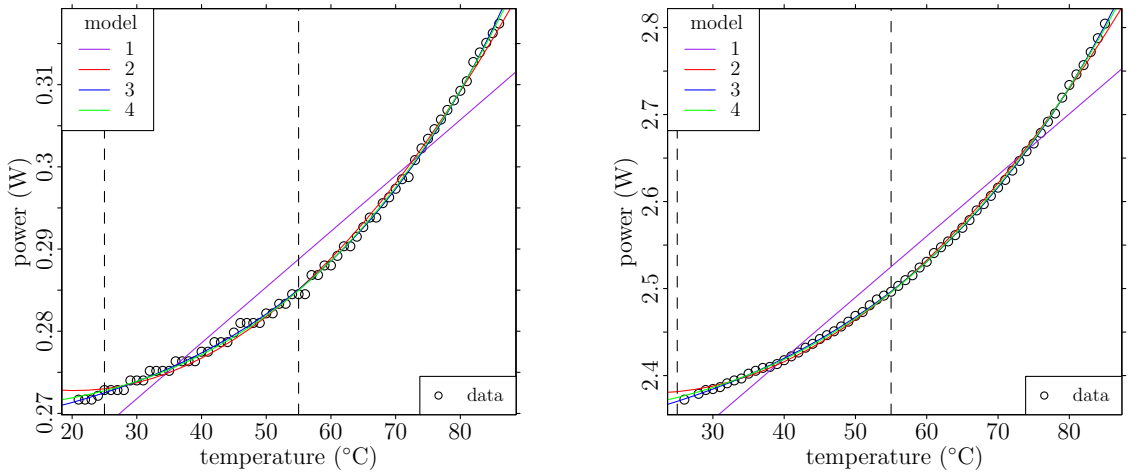
(c) $\#pr$=3

(d) $\#pr$=4

Figure 3.12: Temperature/power traces of the Cortex A15 microprocessor with $\#pr$ cores active between 800 MHz and 1.6 GHz. A super-linear temperature dependency on the power is clearly visible. The power increases when more cores are active.

Table 3.2: Power models $P(T)$ considered as candidates for an adequate temperature/power relationship model. a_* are parameters to be defined via fitting. Three basic curves are considered, which frequently appear in theoretical research on leakage currents: linear, quadratic and exponential. Figure 3.13 shows practical examples of these curves.

Model	Expression	Base
1	$P = a_1 T + a_0$	linear
2	$P = a_2 T^2 + a_1 T + a_0$	quadratic
3	$P = a_1 e^{T/a_2} + a_0$	exponential
4	$P = a_1 T^2 e^{T/a_2} + a_0$	quadratic-exponential



(a) A7 microprocessor - 500 MHz - $\#pr$=3

(b) A15 microprocessor - 1.2 GHz - $\#pr$=4

Figure 3.13: Temperature/Power relationship as measured on the A7 and A15 microprocessors for three and four active cores, respectively. The traces for the other microprocessors and configurations show similar behavior. Note the quantization noise in the case of the A7 microprocessor, most prevalent at low temperatures. Models are listed in Tale 3.2

of the data adequately.

Figure 3.13(a) and 3.13(b) shows single temperature/power traces for the A7 and A15 microprocessors, respectively, with the models from Table 3.2 fitted on the data. All traces look similar to these two examples; therefore not all curves are shown and only aggregated fitting errors are presented. The aggregated fitting errors are given in Table 3.3 for fitting over the 25°C to 85°C temperature range. The mean squared error (MSE)[5] is aggregated over all the traces measured with the same active core count.

The sum of squared errors for the quadratic case is observed to be 1.25 to 2.5 times larger than the exponential fit errors, and the linear fit errors are about 3 to 6 times larger than the quadratic. The two exponential-based models perform nearly equally good, but model 3 seems to have the upper hand. The p-values of the *sign test* between the four models on the A7 and A15 stay well below the 0.01 significance level, confirming that the exponential-based models are a significantly better fit than the quadratic model, while the latter is significantly better than the linear fit. The sign test also indicates that the exponential model is better than the quadratic-exponential model 95 % of the time. It may thus be assumed that the exponential-based models are the most representative of

[5]The *mean squared error* is defined in Appendix A.3.

Table 3.3: Aggregated temperature/power fitting errors (MSE) for the models: (1) linear, (2) quadratic, (3) exponential, and (4) quadratic-exponential over the temperature range 25°C to 85°C for the A7 and A15 microprocessors (PR) and a given active core count (#CO).

		Model			
PR	#CO	1	2	3	4
A7	1	0.016175	0.004070	0.003274	0.003316
A7	2	0.019185	0.004308	0.003333	0.003364
A7	3	0.022967	0.005080	0.003313	0.003604
A7	4	0.026978	0.005851	0.003444	0.003906
A15	1	0.112919	0.020049	0.007726	0.010625
A15	2	0.146620	0.026332	0.011888	0.015327
A15	3	0.180399	0.031024	0.012865	0.016726
A15	4	0.229522	0.040283	0.016049	0.020562

Table 3.4: Aggregated temperature/power fitting errors (MSE) for the models: (1) linear, (2) quadratic, (3) exponential, and (4) quadratic-exponential over the temperature range 25°C to 55°C for the A9 and A15 microprocessors (PR) and a given active core count (#CO).

		Model			
PR	#CO	1	2	3	4
A9	1	0.000491	0.000277	0.001460	0.000285
A15	1	0.009520	0.004649	0.004590	0.003331
A15	2	0.008536	0.004559	0.004596	0.005003
A15	3	0.006327	0.003244	0.003207	0.003342
A15	4	0.006318	0.003000	0.002966	0.003874

the proposals in Table 3.2. Figures 3.13(a) and 3.13(b) back up this observation. Indeed, the exponential-based models seem to follow the measurements very well. The quadratic curve overestimates the power for the lower temperatures but performs very well for larger ones. The linear curve does not adequately represent the temperature/power relationship in this temperature range compared to the other proposed models.

Let's see how the curves behave in a more pertinent temperature range; between 25°C–55°C. Aggregated errors are given in Table 3.4. Due to the quantization noise within this temperature span the fitting for the A7 traces don't always converge properly. This hampers the fitting process and renders the results unreliable; therefore the A7 analysis is omitted here. For the A15 microprocessor, it is observed that the competitive advantage of the exponential-based curves has shrunk. The sign test significantly favors the quadratic curve over the linear curve. The quadratic curve in the 25°C–55°C temperature range is, however, as good as the exponential curve following the same sign test. Also, the quadratic-exponential model performs significantly less than the exponential model. In the case of the A9 microprocessor, based merely on the aggregated errors, the quadratic curve presents itself as the best fit, but its performance is not significantly different from the exponential-quadratic model according a sign test with 0.01 significance level. Nevertheless, the linear curve is also a good match compared to the quadratic and the exponentials. The exponential model does perform worst in this case, which is contradictory for the observation made for the A7 and A15 microprocessors.

Based on the A7 and A15 traces, the two exponential-based model's performance best,

Table 3.5: Practical guidelines for choosing a temperature/power model with the aim of minimizing model complexity and fitting error. The models are chosen depending on the temperature range ΔT of the system.

TEMPERATURE RANGE					MODEL
		ΔT	$<$	20°C	linear
20°C	$<$	ΔT	$<$	40°C	quadratic
40°C	$<$	ΔT			exponential

where the exponential model has the upper hand. Following Ockham's razor, model 3 is preferred as it is less complex, which is in line with the sign tests.

Even though the curves are most likely exponential-based, in this limited temperature range the quadratic curve performs as well as the exponential curve. The performance of the linear curve is also acceptable nonetheless. Table 3.5 shows guidelines for choosing a proper model depending on the temperature range as discussed. This is a positive conclusion for TMUs and previous research that assumed a linear or quadratic relationship between temperature and power. Analytical derivations can be notably simplified by virtue of said assumptions. It is noted, however, that the A9 traces suffer from the so called *distant-sensor-syndrome*, explained in Section 3.4.1, and were hence transformed as per Equation 3.8 before being processed here.

3.6.2 Parametric Temperature/Power Model

For diverse applications, amongst others simulations, it is useful to construct a power model, for arbitrary temperature, clock frequency and number of active cores. The variables a_* in Table 3.2 need to be defined in function of the temperature, clock frequency, and active cores, in order to arrive at a parametric power model. Two different modeling approaches are taken; first the parameters a_* themselves are modeled, as they are actually functions of f, and, secondly, the power's temperature shift is modeled.

Let us look at the modeling of the a_* parameters first. Here, the expression of the exponential is assumed in a slightly different form than was shown in Table 3.2:

$$a_1^\star e^{T/a_2} + a_0 = e^{(T-a_1)/a_2} + a_0, \quad \text{where} \quad a_1^\star = e^{a_1/a_2}.$$

From analyzing the A7 and A15 traces it appears that a_2 is seemingly constant for all measurement data. Observing the fitted values for a_1 reveal that these are linearly correlated with frequency and active core count. For the a_0 case, the values are quadratically correlated with the frequency and linearly with the active core count. Moreover, the lines pertaining a_2 values for a fixed frequency, when extrapolated, seem to converge to a single point on the abscissa. Following these observations the following expressions for a_0 and a_1 are obtained:

$$\begin{aligned} g_s &= m_1 + m_2 f + m_3 f^2 \\ g_o &= g_s / m_4 \\ a_0 &= g_s c + g_o \qquad (3.10a) \\ a_1 &= m_5 f + m_6 + (5 - c) m_7, \qquad (3.10b) \end{aligned}$$

where f is the microprocessor's clock frequency, c the active core count, and m_* case specific parameters, to be defined via fitting of the power traces. For both microprocessors it is observed that m_4 and m_7 are almost equal. Whereas this model is based on correlation

Table 3.6: Frequency (GHz) and voltage (mV) settings for the A7, A9, and A15 microprocessors. The data for the A7 and A15 is assessed via measurements. The A9 voltage levels are retrieved from the Linux kernel.

	FREQUENCY (GHz)															
	0.2	0.25	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6
A7	–	950	950	987	1100	1225	–	–	–	–	–	–	–	–	–	–
A9	925	–	950	950	950	975	1000	1025	1075	1125	1175	1225	1250	1275	1325	1350
A15	–	–	–	–	–	–	–	900	925	962	1000	1025	1062	1100	1137	1162

procedure POWER-MODEL-1.A7(T, f, c)
$g_s \leftarrow 0.028 - 0.093f + 0.371f^2$
$g_o \leftarrow g_s/2.202$
$a_0 \leftarrow g_s c + g_o$
$a_1 \leftarrow -38.242f + 187.668 + (5 - c) \cdot 8.430$
$a_2 \leftarrow 33.105$
return $\exp((T - a_1)/a_2) + a_0$
end procedure

Procedure 3.1: Cortex A7 power consumption model based on the modeling of the a_* parameters of exponential model number 3 in Table 3.2, parameters: temperature T (°C), CPU frequency f (GHz), and active core count c [1:4].

with the microprocessor's clock frequency, the correlation of the microprocessor's supply voltage and the a_* parameters was also assessed. However, no satisfactory correlation was identified. For the sake of completeness the frequency/voltage relationship is shown for the A7, A9 and A15 microprocessors in Table 3.6.

A prototype implementation of the power models for the A7 and A15 is given in Procedure 3.1 and Procedure 3.2, respectively, which can be copy-pasted directly into any simulation or script. Based on the collected traces the A15 power model shows a median error of 1.19% and a maximum error of 7.11%; for the A7 microprocessor the median error is 2.89% and the maximum is 8.31%. In absolute terms both models deviate about equally from the measurements, but as the A7 consumes less power its relative error is larger. The error of the models is not negligible. After analyzing our data three sources were identified that introduce errors/noise: the initial temperature conditions that vary for each trace, temperature and power sensor noise (the former being larger than the latter).

As an alternative to estimating the power consumption via Equations 3.10, a model

procedure POWER-MODEL-1.A15(T, f, c)
$g_s \leftarrow 0.220 - 0.315f + 0.467f^2$
$g_o \leftarrow g.s/2.202$
$a_0 \leftarrow g_s c + g_o$
$a_1 \leftarrow -56.652f + 165.896 + (5 - c) \cdot 8.430$
$a_2 \leftarrow 33.105$
return $\exp((T - a_1)/a_2) + a_0$
end procedure

Procedure 3.2: Cortex A15 power consumption model based on the modeling of the a_* parameters of exponential model number 3 in Table 3.2, parameters: temperature T (°C), CPU frequency f (GHz), and active core count c [1:4].

```
procedure POWER-MODEL-2.A7(T, f, c)
    fV ← cbind(c(0.6,1.225),c(0.5,1.100),c(0.4,0.987),c(0.3,0.950),c(0.25,0.950))
    params ← cbind(c(0.001548354,-0.2271537,0.2527928),
              c(0.001076922,-0.1989523,0.4096225), c(-0.001276455,-0.2114213,0.5880604),
              c(-0.004916091,-0.2525054,0.8008379))
    omega ← cbind(c(0.9165315,36.64675,128.5582),c(0.9446261,33.04807,133.2030),
              c(0.9550139,31.69125,135.3539),c(0.9624107,30.69523,136.8430))
    mV ← fV[(fV[,1] == f),2]
    Pref ← params[c,1] + (1+mV*params[c,2])*params[c,3]*f*mV^2
    scale ← omega[c,1] + exp((T-omega[c,3])/omega[c,2])
    return scale * Pref
end procedure
```

Procedure 3.3: Alternative Cortex A7 power consumption model scaling a reference power consumption to an arbitrary temperature, parameters: temperature T (°C), CPU frequency f (GHz), and active core count c [1:4]. The pseudo code is R-based.

```
procedure POWER-MODEL-2.A15(T, f, c)
    fV ← cbind(c(0.8,0.900),c(0.9,0.925),c(1.0,0.962),c(1.1,1.000),c(1.2,1.025),
              c(1.3,1.062),c(1.4,1.100),c(1.5,1.137),c(1.6,1.162))
    params ← {{0.06304262,0.8507527,0.3143365),c(0.09711657,0.6573983,0.5889670),
              c(0.15714696,0.9571463,0.6840582),c(0.20643708,1.1880969,0.7855338))
    omega ← cbind(c(0.9275326,33.37058,124.3985),c(0.9368108,33.98474,130.4629),
              c(0.9468184,33.19642,134.5030),c(0.9489654,32.72711,134.2357))
    mV ← fV[(fV[,1] == f),2]
    Pref ← params[c,1] + (1+mV*params[c,2])*params[c,3]*f*mV^2
    scale ← omega[c,1] + exp((T-omega[c,3])/omega[c,2])
    return scale * Pref
end procedure
```

Procedure 3.4: Alternative A15 power model scaling a reference power consumption to an arbitrary temperature, parameters: temperature T (°C), CPU frequency f (GHz), and active core count c [1:4]. The pseudo code is R-based.

can be set up that is based on the exponential temperature/power model. The idea is that the power consumption at a reference temperature P_{ref} is computed and then scaled by ω to any arbitrary temperature:

$$P_{\text{cpu}}(T) = \omega(T) \cdot P_{\text{ref}} = \left(\omega_2 + e^{\frac{T-\omega_0}{\omega_1}}\right) \cdot P_{\text{ref}}. \tag{3.11}$$

Procedures 3.3 and 3.4 show a prototype implementation. This power model takes the same parameters as arguments as the model of Procedures 3.1 and 3.2, and comes along with more fixed values compared to those models. Mainly because, besides the modeling of the exponential relationship, a power model is also present. Moreover, the clock frequency parameter for these procedures is supposed to be chosen within the discrete set of microprocessor clock frequencies. As a contrast, procedures 3.1 and 3.2 allow for continues values for the clock frequency. The median and maximum relative errors of Procedures 3.3 and 3.4 are, respectively, 0.55% and 4.46% for the A7, 0.56% and 4.88% for the A15. The model errors of these procedures are about half as small as the model errors of Procedures 3.1 and 3.2. One could thus favor the power models of Procedures 3.3 and 3.4 when it comes to accuracy. On the other hand, the advantage of the power models

```
procedure POWER-TEMPERATURE-TRANSFORMATION.A7(Tinit,Tdest,c)
    temp ← cbind(c(0.9165315,36.64675,128.5582),c(0.9446261,33.04807,133.2030),
               c(0.9550139,31.69125,135.3539),c(0.9624107,30.69523,136.8430))
    init ← temp[c,1] + exp((Tinit-temp[c,3])/temp[c,2])
    dest ← temp[c,1] + exp((Tdest-temp[c,3])/temp[c,2])
    return (dest/init)
end procedure
```

Procedure 3.5: Cortex A7 power transformation model based on the exponential temperature/power model, parameters: initial temperature `Tinit` (°C), destination temperature `Tdest` (°C), and active core count `c` [1:4].

```
procedure POWER-TEMPERATURE-TRANSFORMATION.A15(Tinit,Tdest,c)
    temp ← cbind(c(0.9275326,33.37058,124.3985),c(0.9368108,33.98474,130.4629),
               c(0.9468184,33.19642,134.5030),c(0.9489654,32.72711,134.2357))
    init ← temp[c,1] + exp((Tinit-temp[c,3])/temp[c,2])
    dest ← temp[c,1] + exp((Tdest-temp[c,3])/temp[c,2])
    return (dest/init)
end procedure
```

Procedure 3.6: Cortex A15 power transformation model based on the exponential temperature/power model, parameters: initial temperature `Tinit` (°C), destination temperature `Tdest` (°C), and active core count `c` [1:4].

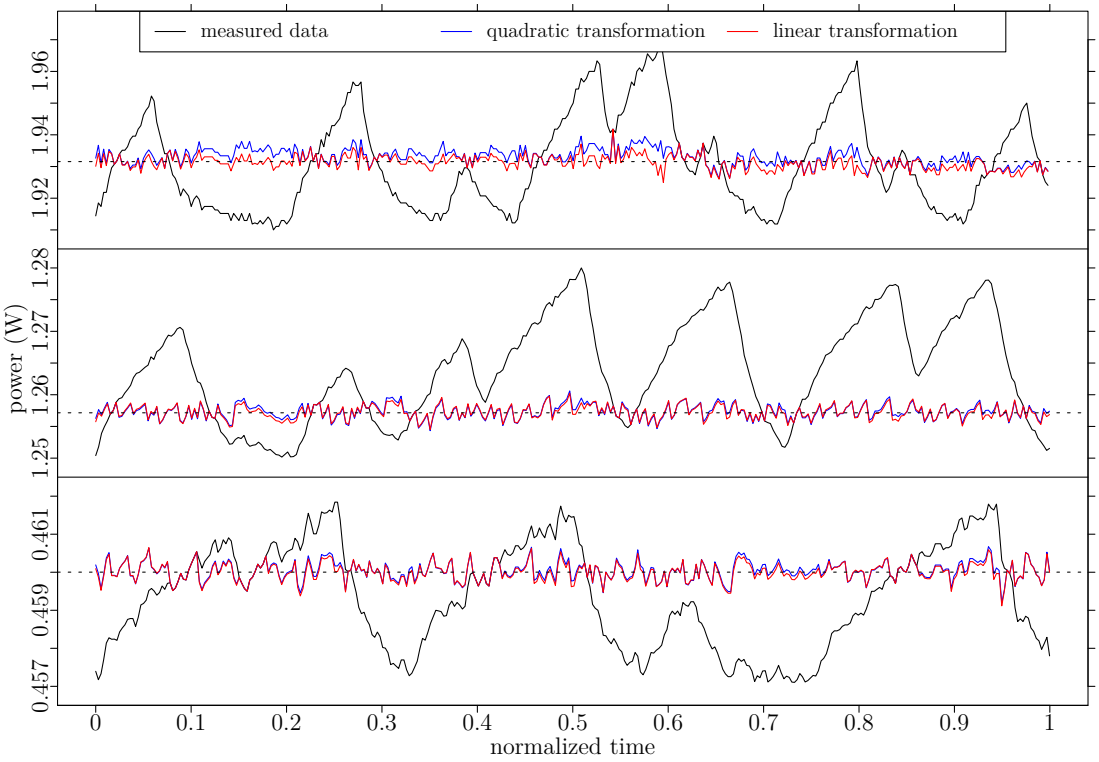
of Procedures 3.1 and 3.2 is the continuous clock frequency parameters. Depending on the context in the sequel one or the other procedure will be favored.

The temperature scaling factor ω from Equation 3.11 can also be used to transform a power sample, measured at one temperature, to another arbitrary temperature. This is basically achieved by normalizing the temperature and then scaling it to an arbitrary temperature. The procedures to convert the power consumption for the A7 and the A15 are shown in Procedure 3.5 and 3.6 respectively. Knowledge of the exponential temperature/power curve is necessary to achieve this transformation, and therefore incorporated into the procedures. This temperature transformation of the power is also the subject of the next section.

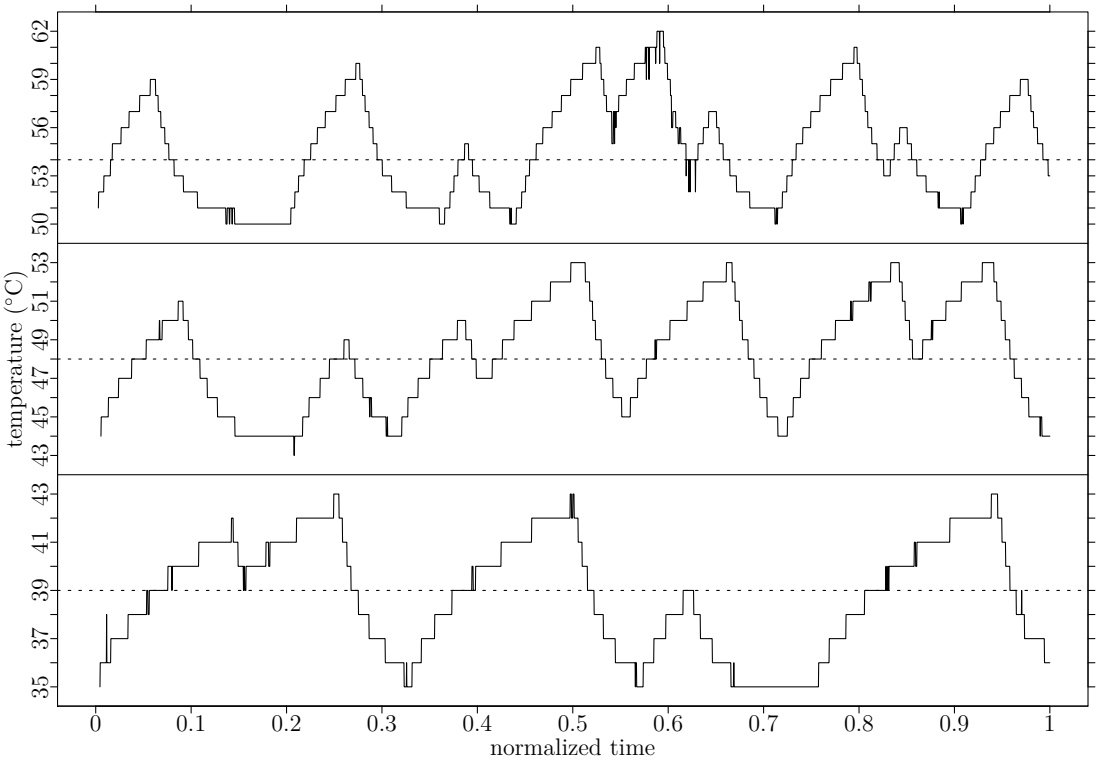
3.7 Temperature/Power Relationship Application

In the previous section it was shown that the temperature/power relationship follows well an exponential-based model. For a limited temperature range a linear or quadratic relationship is adequate as well. Using the temperature/power relationship, it is demonstrated how to cancel power inflations stemming from temperature fluctuations. This improves the power measurement's accuracy and provides a more valuable indication of power demands, and hence also energy consumption. Figure 3.14(a) shows three random power measurement traces. The temperature traces, recorded at the same time as the power traces, are shown in Figure 3.14(b).

The goal here is to convert all the measured power samples as if they were measured at a fixed arbitrary *reference temperature*. Based on the temperature/power relationship models in the previous section it may be assumed that there must exist a transformation



(a) power trace



(b) temperature trace

Figure 3.14: Figure (a) power/time traces (black line): (top) an A15 running at 1.3 GHz with 3 active cores, (middle) an A15 running at 0.9 GHz and (bottom) an A7 running at 0.6 GHz with 4 active cores. The blue and red lines are the transformed power traces in the case of linear and quadratic temperature/power approximations, respectively. An arbitrary reference temperature was chosen at, from top till bottom, 40°C, 46°C, and 55°C. The accompanying temperature/time traces are shown in figure (b).

function (linear/quadratic/exponential) such that the transformed power is constant[6]. As a result of the transformation, the power traces, as shown in Figure 3.14, should appear flat after the transformation. A linear relationship between temperature and power ($P = \eta_1 T + \eta_0$) yields the following transformation function:

$$\begin{aligned} P_r - \eta_1 T_r &= P_m - \eta_1 T_m \\ P_r &= P_m + \eta_1 \Delta T, \end{aligned} \tag{3.12}$$

where T_r is the arbitrary reference temperature; P_r is the power consumption at the reference temperature, P_m and T_m are the measured power and temperature, and $\Delta T = T_r - T_m$. Similarly, a quadratic temperature/relationship ($P = \eta_2 T^2 + \eta_1 T + \eta_0$) yields the following power transformation function:

$$P_r = P_m + \eta_2 (T_r^2 - T_m^2) + \eta_1 \Delta T. \tag{3.13}$$

To find the optimal transformation function, it isn't required to know the microprocessor's precise thermal behavior. A linear or quadratic regression between temperature and power of the collected trace suffices to obtain the η_* values. As stated before, the linear and quadratic approach is appropriate when the testbed temperature variations are no more than what is advised in Table 3.5; otherwise one needs to resort to exponential fits to maintain acceptable accuracy.

Figure 3.14 shows power traces of a microprocessor under constant load with variable convective cooling applied. Their resulting linear (blue) and quadratic (red) power transformations are also shown. As can be observed, the jerky power traces are converted into somewhat stable traces, besides the presence of the measurement instrument's noise. Temperature noise and sensor inaccuracy are a known problem for TMUs [60]. The most important feature of the power transformation is that the arbitrary distribution is transformed into a symmetric distribution. Statistical measures, such as the mean, variance and median, have only valuable meaning when dealing with symmetric distributions. Figure 3.15 shows the distributions of the measured and the transformed power traces. The measured power trace distributions in Figure 3.15(a) have a shape dictated by the temperature variation of the system. The transformed power trace distributions in Figure 3.15(b) have smaller variances, and are more symmetric. This facilitates the reproducibility of measurement results, and ameliorates the meaningfulness and stability of statistical estimators.

Table 3.7 shows an overview of the transformation's performance at different reference temperatures. The measured power fluctuation (MPF):

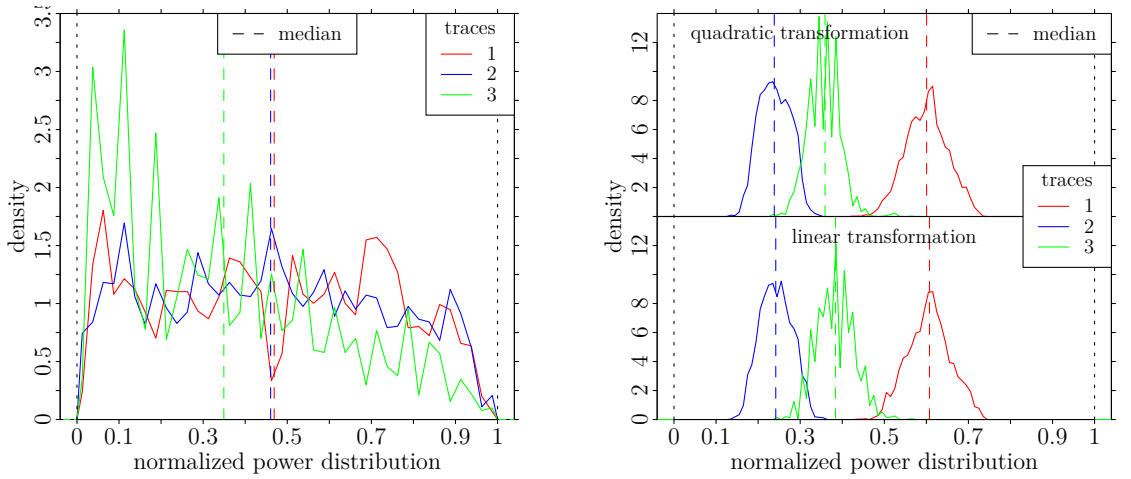
$$\text{MPF} = \frac{\max(P_m) - \min(P_m)}{\text{median}(P_m)} \tag{3.14}$$

due to varying temperature is shown to be between 1.21% and 3%. The relative power fluctuation (RPF) is computed as the transformed power (P_r) fluctuation over the measured power (P_m) fluctuations:

$$\text{RPF} = \frac{\max(P_r) - \min(P_r)}{\text{median}(P_r)} \Bigg/ \frac{\max(P_m) - \min(P_m)}{\text{median}(P_m)}. \tag{3.15}$$

It can be seen that the power fluctuations, due to the power transformation, are diminished by a factor of about three to four in all cases. This can also be visually verified

[6] In practice, due to noisy measurements, the variance of the power is to be minimized.



(a) original normalized power density

(b) transformed normalized power density

Figure 3.15: Power distribution of the original and transformed power traces from Figure 3.14. The power traces were normalized before being plotted. Densities are plotted for a bin-size of 0.025 on the left, and 0.01 on the right. It can be observed that the original distributions are transformed into distributions that are more symmetric and smaller in variance.

in Figure 3.14. Moreover, the resulting transformation's distribution is quasi symmetric. This is shown with the RAT metric in Table 3.7, which represents the departure of the median from the mean. If the median differs significantly from the mean then the distribution is not symmetric. It can be seen, however, that in all cases the median and mean are very close too each other, indicating that the transformation produces a quasi symmetric distribution. Now that the power is converted as if it were measured at a reference temperature, statistical methods have more practical value. If non-robust statistical methods would be applied directly to the measured power, the estimators would produce estimates that are biased based on the arbitrary distribution of the measured power samples. This non-robust estimator mismatch should be avoided by all means[7].

It is noted that it is advisable to choose a reference temperature within the measured temperature range, preferably not too close to the extremities, to minimize transformation errors.

3.8 Conclusion

The power consumption of a microprocessor is temperature-dependent and was shown via experimental measurements on two SoCs. Also, an overview of temperature/power models was presented from the literature. Yet, these models focus usually on the temperature dependency of the leakage currents, which contribute to the power consumption of the microprocessor. Other sources of temperature dependency were highlighted; they can also contribute to fluctuations in power consumption.

The temperature/power analysis on experimental traces showed that a simple exponential model can describe the relationship very well for a temperature range between 25°C and 85°C. For a more limited temperature range, e.g., 25°C $< T <$ 55°C, a linear or quadratic relationship is acceptable as well. Moreover, a practical example was given

[7]More information on robust statistics can be found in the work of Maronna *et al.* [74].

Table 3.7: Performance metrics of the power transformation, for different reference temperatures T (°C), and microprocessor configurations, for the traces in Figure 3.14. The measured power fluctuation is provided (MPF) as a measure of variance; the relative fluctuation (RPL) for the linear (L) and quadratic case (Q) are shown. The relative increase of the median over the mean is also stated (RAT) to quantize the distribution's symmetry.

A15 @ 1.3 GHz					
T	MPFL	RPL-L	RPL-Q	RAT-L	RAT-Q
35	1.21	0.297	0.311	$0.753{\cdot}10^{-3}$	$0.403{\cdot}10^{-3}$
38	1.21	0.295	0.310	$0.750{\cdot}10^{-3}$	$0.401{\cdot}10^{-3}$
41	1.21	0.294	0.309	$0.747{\cdot}10^{-3}$	$0.400{\cdot}10^{-3}$
A15 @ 0.9 GHz					
T	MPF	RPL-L	RPL-Q	RAT-L	RAT-Q
43	1.80	0.231	0.212	$-1.706{\cdot}10^{-3}$	$-2.591{\cdot}10^{-3}$
46	1.80	0.229	0.211	$-1.697{\cdot}10^{-3}$	$-2.579{\cdot}10^{-3}$
49	1.80	0.228	0.210	$-1.688{\cdot}10^{-3}$	$-2.565{\cdot}10^{-3}$
52	1.80	0.227	0.208	$-1.679{\cdot}10^{-3}$	$-2.551{\cdot}10^{-3}$
A7 @ 0.6 GHz					
T	MPF	RPL-L	RPL-Q	RAT-L	RAT-Q
50	2.99	0.359	0.327	$-0.249{\cdot}10^{-3}$	$8.677{\cdot}10^{-3}$
54	2.99	0.355	0.324	$-0.247{\cdot}10^{-3}$	$8.615{\cdot}10^{-3}$
58	2.99	0.352	0.321	$-0.245{\cdot}10^{-3}$	$8.532{\cdot}10^{-3}$
62	2.99	0.349	0.317	$-0.243{\cdot}10^{-3}$	$8.430{\cdot}10^{-3}$

showing how to use the temperature/power relationship to cancel temperature bias in power measurements to improve their accuracy. Another model is advanced that allows for backtracking temperature measurements that were recorded at a finite distance of a heat source. Such models allow for a better understanding of the microprocessor's heat dynamics from an observer's point of view, including the distant-sensor-syndrome. After fine tuning, all these models are also applicable to other microprocessors. Temperature traces should be phrased adequately before they are correlated with power traces in order to avoid side-effects, including the temperature/power hysteresis loops.

The models developed in this chapter and the concepts broached, especially w.r.t. the microprocessor's thermal behavior, help in producing power measurements that are meaningful and reproducible. This is an important aspect of energy estimation and modeling. The latter is the subject of the following chapters.

Chapter 4

The Energy/Frequency Convexity Rule

In chapter 2 a power and an execution time model, was presented which, when multiplied, produces the microprocessor's energy consumption model following Equation 2.2. In this chapter, the behavior of this energy consumption model is analyzed via sensitivity analysis and demonstrated practically with experimental execution time and power measurement traces.

The chapter starts with a state of the art presenting energy/frequency curves found in the literature. The Energy/Frequency Convexity Rule for a single-core microprocessor is developed theoretically in Section 4.2. Then, in Section 4.3, experimental power and execution time measurements of two application processors are presented that demonstrate the existence of the energy/frequency convexity. These measurement-based models will also be used in the following chapters. Section 4.4 analyses the sensitivity of the Energy/Frequency Convexity Rule's parameters. The chapter is concluded in Section 4.5.

4.1 State of the Art

To preempt the reader's intrigue by virtue of the catchy chapter title, it is noted that the energy consumption of a microprocessor shows convex properties with regards to its clock frequency. The convex property of the energy consumption curve has been hinted at before in the literature. A series of papers, approaching the problem from an architectural point of view, have shown a convex energy consumption curve with respect to Dynamic Voltage and Frequency Scaling (DVFS) [35, 64, 105, 111]. The literature puts forward some motivation for the energy consumption's convexity, but rarely provides analytical frameworks based on physical explanations. For example, Senn *et al.* [103] and Austin and Wright [5] provide a heuristic model. Other studies, e.g., Hager *et al.* [46] and Freeh *et al.* [40], discuss what the consequences are of said behavior and how to exploit them, from a high-level point of view. Other researchers have also shown energy measurements under DVFS processes but no convexity is shown by the measurements, e.g., Sinha and Chandrakasan [108], and Šimunić *et al.* [106], who are not running their benchmarks on top of an OS. Authors, such as Austin and Wright [5] and Snowdon [35, 111], have shown more specifically that for applications with certain behavioral patterns no energy convexity is observed. However, the energy consumption model about to be scrutinized can explain such behavior.

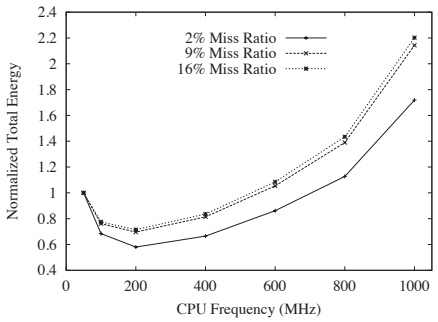
In the VLSI design domain, voltage scaling has also been discussed but usually for a fixed frequency [14, 62, 139]. The aim of the voltage scaling is to find a minimum energy operation point where the digital circuit yields the correct output. The major trade-off is between increased circuit latency and leakage power, and decreasing dynamic power. This trade-off also yields a convex energy consumption curve, but for a fixed frequency. In this chapter, however, the combined effect of voltage/frequency scaling is of interest. Clock frequency modulation is another approach to energy optimization, similar to DVFS. Clock frequency modulation is, however, not discussed in this work. Experimental work, e.g., by Cicotti *et al.* [26] and Wang *et al.* [126], has pointed out similar behavior between clock frequency modulation and DVFS in terms of energy consumption with potential convex properties.

There is some work that covers the energy/frequency convexity properties in a limited analytical framework. Figure 4.1, continued in Figure 4.2, show excerpts of convex energy graphs provided by the cited work. Yuki and Rajopadhye [137] explored the energy consumption of high performance computers in the context of compiler optimization and optimal frequency conditions of the microprocessor. One of their conclusions is that for power-hungry systems the *race-to-halt*[1] energy optimization technique is more effective than DVFS. Hager *et al.* [46], on the other hand, showed that race-to-halt is not always the most effective strategy in a multi-core context with bandwidth-bound codes. The authors studied the energy consumption of modern multi-core chips via simple machine models and showed how to minimize the energy consumption with respect to number of cores, serial code performance, and clock frequency. Austin and Wright [5] examined the energy consumption of micro-benchmarks and applications on a Cray CX30 super computer system. The authors developed a simple linear heuristic energy model. They also stressed that the frequency/energy minimum is applications-specific. Cho and Chang [25] assessed the optimal frequency conditions for a microprocessor in conjunction with a memory. Their resulting model is fairly complex; yet the authors show the feasibility of a microprocessor's optimal frequency conditions in conjecture with a memory system. Cho and Melhelm [24] produced a convex model derived from Amdahl's law and extended with the notion of energy. The authors use a simplifying assumption for the representation of power and execution time. They show via their model that there is a certain clock frequency range that yields both energy and speed improvements. Similarly, Rizvandi *et al.* [96] devised a convex model but, just as Cho and Melhelm, simplified representations of power and execution time were assumed. Tudor and Teo [116] developed an analytical energy consumption model for multi-core ARM-based servers. Their time model is elaborate, based on M/G/1 queues and Pareto distributions, but their power model is basic. Despite their extensive time modeling efforts, the analytical model, however, doesn't fit perfectly on the authors' experimental measurements.

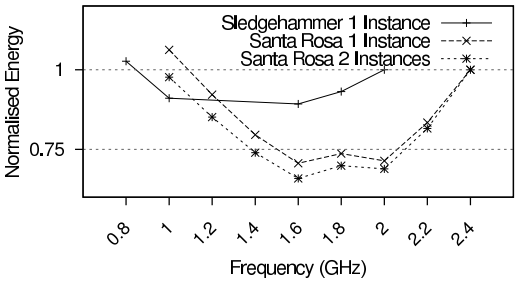
From an experimental perspective, Halimi *et al.* [48] claim to save up to 39 % of energy, and Qiu *et al.* [88] advertise an energy gain of 25 %, by adjusting the microprocessor's clock frequency via an experimental algorithm with predefined user or application constraints. Although no theoretical framework was provided by the authors about the energy/frequency convexity, their algorithm is essentially chasing the energy convex minimum. Senn *et al.* [103] showed also convex energy/frequency curves, based on a simplified system model, for their TI C55 C62 C64 C67 platforms.

The work presented here focuses on embedded systems, in contrast with Yuki and

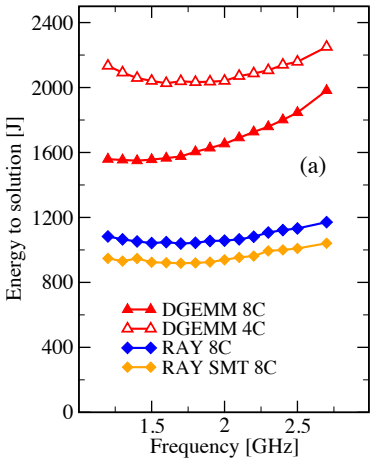
[1]The energy optimization technique *race-to-halt* runs the microprocessor at full speed until all tasks are completed, then the microprocessor is put in a low-power mode.



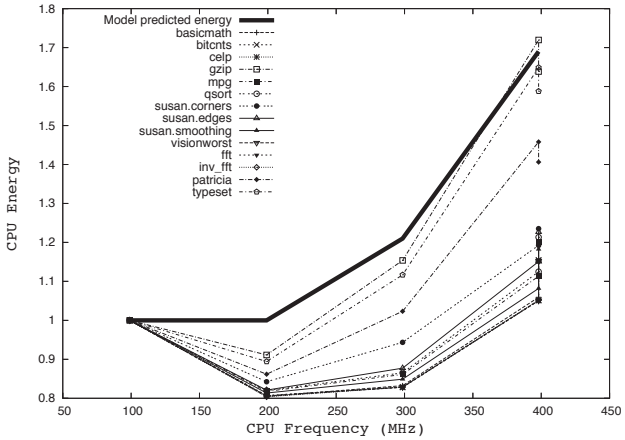
(a) Fan *et al.* [35]



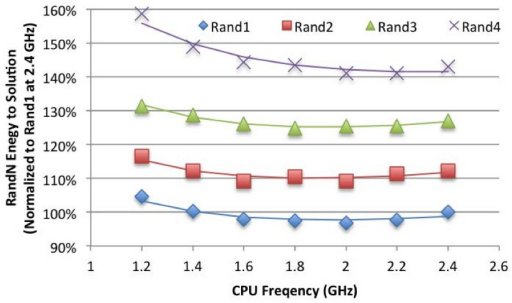
(b) Le Sueur and Heiser [64]



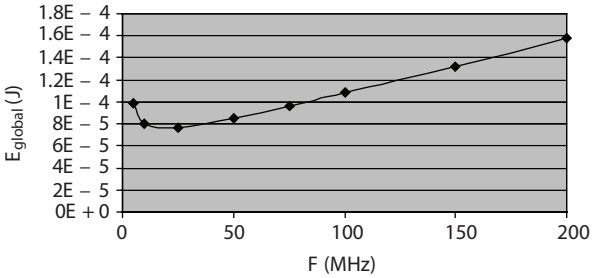
(c) Hager *et al.* [46]



(d) Snowdon *et al.* [111]

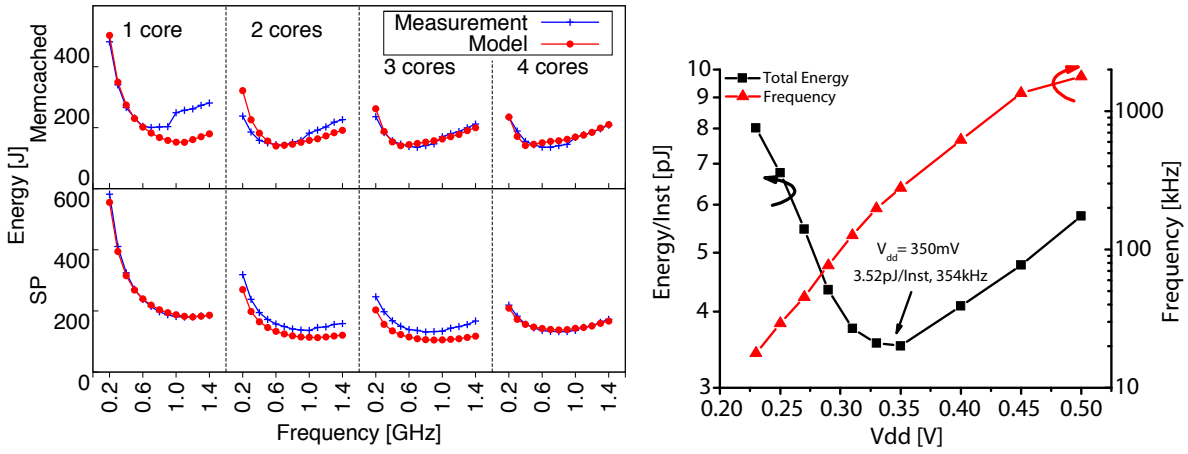


(e) Austin and Wright [5]



(f) Senn *et al.* [103]

Figure 4.1: Excerpts of energy/frequency measurements as found in the literature. Convex minimums are observable for the energy at a certain microprocessor clock frequency, depending on the microprocessor and architecture. In the sequel the behavior of this convex minimum is analyzed. All figures were originally published in the papers referenced in their respective captions.



(a) Tudor and Teo [116]

(b) Seok *et al.* [105]

Figure 4.2: Excerpts of energy/frequency measurements as found in the literature. All figures were originally published in the papers referenced in their respective captions.

Rajopadhye's, Hager *et al.* and Austin and Wright's work, which is dedicated to more powerful computer architectures. In the sequel, the sensitivity of the parameters that constitute the energy consumption equation are also analyzed via both an analytical approach and via experimental data, the former fitted with data from the latter. The convex energy model presented is, in contrast with the mentioned works, more extensive, which allows for a more realistic modeling of reality. For example, temperature has not been a subject of interest and a sensitivity analysis of parameters has also not been carried out in any of the referenced works.

4.2 Single-Core Convexity Model

The energy consumption of a computer system comprising a microprocessor and possibly other components is equal to the integral of the system's power consumption over time:

$$E = \int_0^{\Delta t} P(t)\ \mathrm{d}t = \int_0^{\Delta t} I(t) \cdot V(t)\ \mathrm{d}t. \tag{2.2}$$

If the power is considered constant the integral is equivalent to the product of the power consumption and the timespan of interest. V can often be considered constant by design; for example, portable devices such as smartphones are supplied by 3.7 V lithium-ion batteries, and microprocessors operate at very specific voltage levels. I, on the other hand, may show a certain variance, even over small time frames. The current's variance depends on the context, its history and the state of the microprocessor. However, at the time frame of an instruction execution, henceforth referred to as a *time quanta*, the energy consumption can be deemed quasi constant. Following this definition, the parameters that define the energy consumption during a time quanta are also constant. As such, similar to the rationale behind the *Riemann sum*, the total system energy consumption E_{sys} of a code sequence can be thought of as the sum of the energy consumption $E_{\text{sys},i}$ during each time quanta:

$$E_{\text{sys}} = \sum_{i=1}^{n} E_{\text{sys},i} = \sum_{i=1}^{n} P_{\text{sys},i} \cdot \Delta t_i, \tag{4.1}$$

where n is the number of time quanta, and $P_{\text{sys},i}$ the power during time quanta Δt_i.

Let us consider the execution of a code sequence executed by a microprocessor during a time quanta with a length Δt (Equation 2.16) requiring a constant power P_{sys} (Equation 2.11). The energy consumption of said computer system is then equal to:

$$\begin{aligned} E_{\text{sys}} &= P_{\text{sys}} \cdot \Delta t \\ &= (P_{\text{dynamic}} + P_{\text{leak}} + P_{\text{back}}) \cdot \Delta t \\ &= \left((1+\gamma V)\,\xi f V^2 + P_{\text{back}}\right) \cdot c c_{\text{b}} \left(\frac{1}{f - f_{\text{k}}} + \beta\right). \end{aligned} \tag{4.2}$$

Here, P_{sys} is a monotonic increasing function of f, whereas Δt is a monotonic decreasing function of f, given that all its parameters are elements of $\mathbb{R}^+$. When P_{sys} and Δt are multiplied the resulting function may very well have convex properties. Such convexity would be of interest as there would exist a microprocessor configuration that minimizes the energy consumption for that particular context. Let us explore the constraints under which such convexity is exploitable by the microprocessor. The following derivation is similar to Yuki and Rajopadhye [137]; however, different frequency and voltage relationships are used, mainly more contemporary, and the leakage current is scaled more realistically. Note that P_{back} can be arbitrarily large; its value is inherent to the computer system and independent of the microprocessor. In the remainder of this work it is also assumed that the temperature of the microprocessor remains constant unless otherwise noted. Section 4.4.4 analyzes the influence of the temperature on the energy/frequency convexity in particular.

4.2.1 Voltage/Frequency Relationship

For modern microprocessors, the frequency and supply voltage in the DVFS process are approximately linearly related as shown in Figure 4.3. The exact relationship is dependent on the physical abilities of the microprocessor, but also on the capability of the microprocessor's voltage and frequency regulator to scale the voltage and frequency on-demand. When the frequency of a microprocessor is ramped up, the transistors inside need to switch faster to meet timing and delay constraints. As subparts of transistors are essentially very small capacitors as well, a finite time is required to switch the transistor from one state to another. Thus if stringent timing delays need to be met, the microprocessor voltage needs be increased accordingly. The higher voltage supply will decrease the transistors' transition time and capacitors' charging time.

A linear relationship between the voltage and frequency is expressed as follows:

$$V = m_1 f + m_2, \tag{4.3}$$

where m_1 and m_2 are regression coefficients. Figure 4.3 shows the voltage and frequency relationship for several microprocessors. It is noted that the S3C6410 and the PXA320 are fairly dated microprocessors with low performance, whereas the Exynos series and the Intel M are more recent microprocessors designed for embedded multimedia applications, e.g., smartphones and tablets. The values $m_1 = \frac{2}{3}$ and $m_2 = \frac{1}{3}$, the dashed blue line in Figure 4.3, are motivated to be adequate for high-performance microprocessors based on theoretical values [137]. Here, the values $m_1 = \frac{1}{3}$ and $m_2 = \frac{4}{5}$ are shown to better represent the voltage/frequency relationship for microprocessors for embedded applications. These values are approximates of a linear fit on the combined data of the Exynos and the Intel M microprocessor.

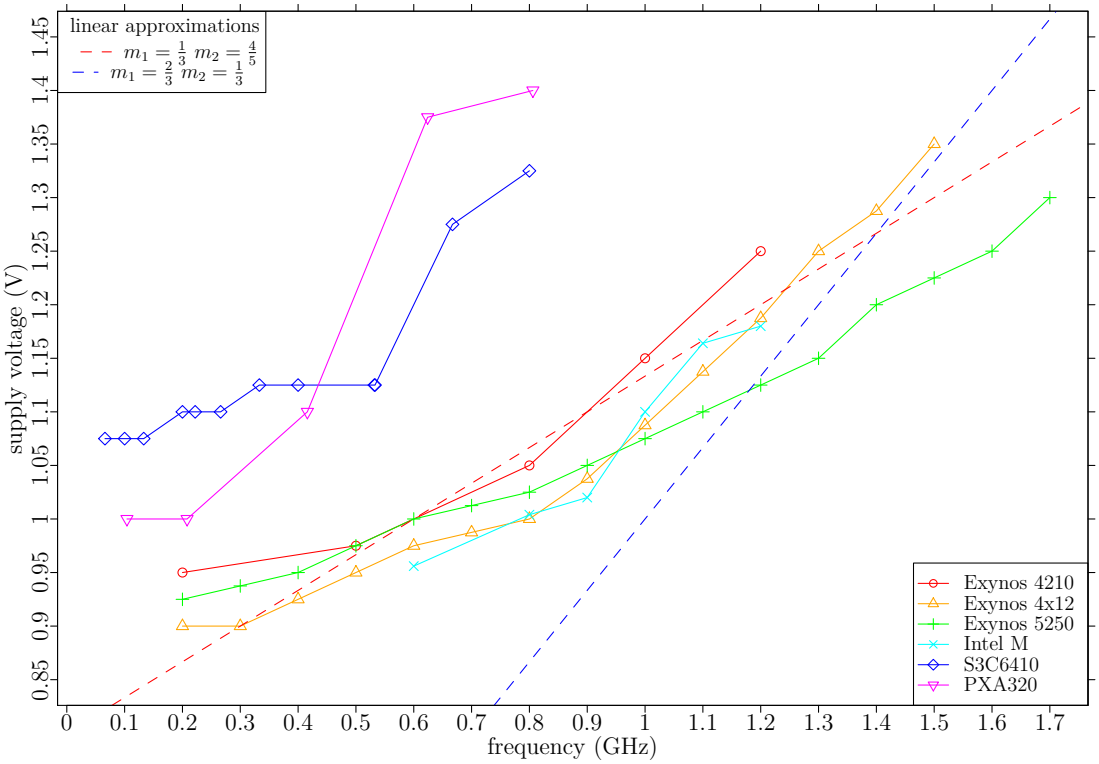


Figure 4.3: Frequency/voltage relationship of multiple microprocessors as found in the Linux kernel. Two dashed linear curves are draw: $V = m_1 f + m_2$; the red is fitted on the depicted data of the first three microprocessors, the blue is borrowed from Yuki and Rajopadhye's [137].

Henceforth, the microprocessor's *default clock frequency window* ($\mathcal{F}_{\text{cpu}}$) over f is defined as the clock frequency range bounded by the minimum and maximum clock frequency of the microprocessor:

$$\mathcal{F}_{\text{cpu}} = f_{\min} \leq f \leq f_{\max}. \tag{4.4}$$

If $f_{\text{k}} > f_{\min}$, then the execution time for $f_{\min} \leq f \leq f_{\text{k}}$ has no physical meaning as $\lim_{f \to^{+} f_{\text{k}}} \Delta t = +\infty$. Hence, the *exploitable clock frequency window* ($\mathcal{F}_{\text{epx}}$) is defined as the frequency range with an upper bound characterized by the microprocessor's maximum frequency $f_{\max}$, and the lower bound defined by the largest of the microprocessor's minimum frequency $f_{\min}$ and f_{k}:

$$\mathcal{F}_{\text{epx}} = \max(f_{\min}, f_{\text{k}}) \leq f \leq f_{\max}. \tag{4.5}$$

It is the exploitable clock frequency window that is open for optimization.

4.2.2 Energy Consumption Model

The energy consumption model independent of V is obtained by inserting the V/f dependency (Equation 4.3) in the definition of E_{sys}:

$$\begin{aligned} E_{\text{sys}} &= \left((1+\gamma V)\xi f V^2 + P_{\text{back}}\right) \cdot \Delta t \\ &= \left((1+\gamma(m_1 f + m_2))\xi f (m_1 f + m_2)^2 + P_{\text{back}}\right) \cdot \Delta t \\ &= (af^4 + bf^3 + cf^2 + df + P_{\text{back}}) \cdot cc_{\text{b}} \left(\frac{1}{f - f_{\text{k}}} + \beta\right), \end{aligned} \tag{4.6}$$

where $a = \gamma\xi m_1^3$, $b = m_1^2\xi(1 + 3\gamma m_2)$, $c = m_1 m_2 \xi(3\gamma m_2 + 2)$, and $d = m_2^2\xi(\gamma m_2 + 1)$. Note that following Equation 4.6, cc_b is a scaling factor of E_sys and implies that the energy consumption of a piece of code is linearly dependent on its code size. Moreover, this also implies that compiler optimization techniques that target code size reduction will indirectly also lead to an improved energy profile of the code. On the other hand, a microprocessor can also reduce energy consumption by overlapping code execution, increasing power demands but reducing execution time. Similar observations between the interaction of energy and power consumption were made by Valluri and John [118].

For further analysis the normalized energy consumption E_n for code size-independent analysis is introduced. The normalized energy consumption is defined as

$$E_\mathrm{n} = \frac{E_\mathrm{sys}}{cc_\mathrm{b}}. \tag{4.7}$$

Normalizing the energy consumption E_sys has no effect whatsoever on its tentative convex properties as it's merely a scaling factor.

The energy function in Equation 4.7 is called strictly convex over the exploitable clock frequency window, if and only if (iff),

$$\forall f_1 \neq f_2 \in f_\mathrm{cpu}, \forall t \in (0,1): \quad E_\mathrm{n}(tf_1 + (1-t)f_2) < tE_\mathrm{n}(f_1) + (1-t)E_\mathrm{n}(f_2). \tag{4.8}$$

In other words, if E_sys is strictly convex, then E_sys possesses no more than one minimum over the exploitable frequency window. If the minimum of E_sys is not one of the microprocessor's boundaries: $f_\mathrm{min} < f_\mathrm{opt} < f_\mathrm{max}$, then the minimum can be found via the first and second derivative of E_sys:

$$\left(\frac{\partial E_\mathrm{n}}{\partial f}\right)_{f=f_\mathrm{opt}} = 0 \quad \text{and} \quad \frac{\partial^2 E_\mathrm{n}}{\partial f^2} > 0. \tag{4.9}$$

To simplify the derivative calculation of Equation 4.6, E_n is split into a polynomial and non-polynomial part, namely E_n^A and E_n^B:

$$E_\mathrm{n}^\mathrm{A} = (af^4 + bf^3 + cf^2 + df + P_\mathrm{back}) \cdot \beta \tag{4.10a}$$

$$E_\mathrm{n}^\mathrm{B} = (af^4 + bf^3 + cf^2 + df + P_\mathrm{back}) \cdot \frac{1}{f - f_\mathrm{k}} \tag{4.10b}$$

$$E_\mathrm{n} = E_\mathrm{n}^\mathrm{A} + E_\mathrm{n}^\mathrm{B},$$

The respective derivatives are then as follows:

$$\frac{\partial E_\mathrm{n}^\mathrm{A}}{\partial f} = (4af^3 + 3bf^2 + 2cf + d) \cdot \beta \tag{4.11a}$$

$$\frac{\partial E_\mathrm{n}^\mathrm{B}}{\partial f} = \frac{3af^4 + (2b - 4af_\mathrm{k})\, f^3 + (c - 3bf_\mathrm{k})\, f^2 - (2cf_\mathrm{k}f + P_\mathrm{back} + df_\mathrm{k})}{(f - f_\mathrm{k})^2} \tag{4.11b}$$

$$\frac{\partial^2 E_\mathrm{n}^\mathrm{A}}{\partial f^2} = (12af^2 + 6bf + 2c) \cdot \beta \tag{4.11c}$$

$$\frac{\partial^2 E_\mathrm{n}^\mathrm{B}}{\partial f^2} = \frac{6af^4 + (2b - 16af_\mathrm{k})\, f^3 + (12af_\mathrm{k}^2 - 6bf_\mathrm{k})\, f^2}{(f - f_\mathrm{k})^3} + \frac{6bf_\mathrm{k}^2 f + 2(P_\mathrm{back} + cf_\mathrm{k}^2 + df_\mathrm{k})}{(f - f_\mathrm{k})^3}. \tag{4.11d}$$

Intuitively, convex properties can already be observed for E_n. For $f \xrightarrow{+} f_\mathrm{k}$: E_n^A will approach βP_back, whereas E_n^B is amplified, and tends to positive infinity because of the presence of f in the denominator. When $\frac{f}{2} < f_\mathrm{k}$, the system is spending more energy in overhead than in the actual program, as the overhead has priority over the program. In the limit, E_n goes to infinity at f_k. At this point the system is overloaded and is not reactive anymore from the point of view of cc_b. For $f \to \infty$, it is E_n^A that inflates whereas E_n^B approaches zero. In other words, for the smaller frequencies, by virtue of the increased execution time more energy due to leakage currents needs to be accounted for. The execution time for large frequencies are dramatically lower, but the dynamic power consumption of the microprocessor increases cubically and the leakage currents increase quartically with frequency. As a result, the global minimum of the energy function, at the optimal frequency f_opt, is the point where a balance is found between the consequences of the inflated execution time and the quartic power consumption of the microprocessor.

4.2.3 Approximate Energy Consumption Model

The power model in the energy consumption formulation of Equation 4.6 is of the fourth order. In Section 4.3.2, it will be seen that the fourth-order power equation can be adequately approximated with a quadratic polynomial, therefore simplifying the derivations somewhat. The power consumption of the system can then be represented as:

$$P_\mathrm{sys} = af^4 + bf^3 + cf^2 + df + P_\mathrm{back} \approx kf^2 + lf + m + P_\mathrm{back}, \tag{4.12}$$

and accordingly the energy consumption of the system becomes

$$E_\mathrm{sys} = (kf^2 + lf + m + P_\mathrm{back}) \cdot cc_\mathrm{b} \left(\frac{1}{f - f_\mathrm{k}} + \beta \right). \tag{4.13}$$

$k \in \mathbb{R}_0^+$, though $\{l, m\} \in \mathbb{R}$. The first and second derivative of the normalized energy consumption are then as follows

$$\frac{\partial E_\mathrm{n}}{\partial f} = \beta(2kf + l) - \frac{f_\mathrm{k}(2kf + l) - kf^2 + m + P_\mathrm{back}}{(f - f_\mathrm{k})^2} \tag{4.14a}$$

$$\frac{\partial^2 E_\mathrm{n}}{\partial f^2} = 2k\beta + \frac{2(f_\mathrm{k}^2 k + f_\mathrm{k} l + m + P_\mathrm{back})}{(f - f_\mathrm{k})^3}. \tag{4.14b}$$

There exists a convex minimum if $\frac{\partial E_\mathrm{n}}{\partial f}$ has a root and $\frac{\partial^2 E_\mathrm{n}}{\partial f^2}$ is a monotonous increasing function. In other words:

$$\begin{aligned} 0 &= \beta(2kf + l)(f_\mathrm{k} - f)^2 - (f_\mathrm{k}(2kf + l) - kf^2 + m + P_\mathrm{back}) \\ &= 2k\beta f^3 + (k + \beta(l - 4f_\mathrm{k}k))f^2 + 2f_\mathrm{k}(\beta(f_\mathrm{k}k - l) - k)f \\ &\qquad -(m + P_\mathrm{back} + f_\mathrm{k} l(1 - \beta f_\mathrm{k})) \qquad (4.15) \\ 2k\beta &\geq -2\frac{f_\mathrm{k}^2 k + f_\mathrm{k} l + m + P_\mathrm{back}}{(f - f_\mathrm{k})^3}. \end{aligned}$$

The solution to Equation 4.15 is the frequency that minimizes energy consumption. Via the well-known Ferarri[2] solution for the calculation of the roots of a third order polynomial,

[2]Ferarri developed an analytical method to solve the roots of both a third and forth order polynomial. Appendix A.2 shows Ferarri's derivation for a fourth order polynomial is shown, which includes the derivation of a third order.

the optimal frequency can be determined analytically. Yet, the analytical formulation to calculate the roots of a cubic polynomial is still elaborate. Lets assume some further simplifications. For $\beta = 0$ one gets that

$$\begin{aligned} f_{\text{opt}} &= f_{\text{k}} + \frac{\sqrt{4k^2 f_{\text{k}}^2 + 4k(m + P_{\text{back}} + f_{\text{k}} l)}}{2k} \\ 0 &\leq 2\frac{f_{\text{k}}^2 k + f_{\text{k}} l + m + P_{\text{back}}}{(f - f_{\text{k}})^3}. \end{aligned} \tag{4.16}$$

If all parameters are elements of $\mathbb{R}^+$, the latter inequality holds whenever $f_{\text{k}} < f$. Additionally for $f_{\text{k}} = 0$ one obtains

$$\begin{aligned} f_{\text{opt}} &= \sqrt{\frac{m + P_{\text{back}}}{k}} \\ 0 &\leq \frac{2(m + P_{\text{back}})}{f^3}, \end{aligned} \tag{4.17}$$

but only valid for $-P_{\text{back}} < m$. These simplified models for $\beta = f_{\text{k}} = 0$ may be used when the context allows for, i.e., when cc_{b} is executed without any interruption. For example, from practical experience and in the literature, f_{k} is often observed to be close to zero in a multi-core context. β may vary considerably for different applications and should be assessed per case before deeming insignificant.

4.3 Experimental Results

In the sequel experimentally-obtained power and execution time measurement traces are presented. The measurement traces were obtained while running benchmarks on specific testbeds. The next subsections describe the testbed and the used benchmarks. The actual collected measurements are presented and discussed after.

4.3.1 Platform and Benchmark Description

The exact same platforms were used as previously expounded in Section 3.5. Benchmarks were run on an ODROID XU+E and a Samsung Galaxy S2. The ODROID features a Cortex A7 and Cortex A15 quad-core microprocessor, while the Samsung Galaxy S2 sports a Cortex A9 dual-core microprocessor. The A7 runs at {0.25,0.3,0.4,0.5,0.6} GHz, the A15 from 0.8 GHz to 1.6 GHz in steps of 100 MHz, and the Galaxy S2 from 0.2 GHz to 1.6 GHz in steps of 100 MHz.

Two benchmarks were used during the experimentation: the Bristol Energy Efficiency Benchmark Suite (BEEBS) and the Gold-Rader bit-reverse algorithm. BEEBS [83] is a select set of benchmarks representative for typical embedded systems applications. The control code of the benchmarks however were amended such that the input size of the benchmarks could be scaled. This was successfully accomplished for all benchmarks except for `fdct` and `cubic`. Table 4.1 shows an overview and description of the benchmarks. The benchmarks are rated *high* (H), *medium* (M) and *low* (L), regarding the following properties: *branching frequency* (B), *memory access intensity* (M), *integer operations* (I), *floating-point operations* (FP). BEEBS was run on the ODROID testbed.

The Gold-Rader bit-reverse algorithm was run on the Galaxy S2 testbed. The Gold-Rader implementation of the bit-reverse algorithm was used, part of the ubiquitous FFT

Table 4.1: Overview of the Bristol Energy Efficiency Benchmark Suite (BEEBS). The benchmarks are labeled high (H), medium (M) and low (L), according to their properties: branching frequency (B), memory access intensity (M), integer operations (I), floating-point operations (FP). More details about the benchmark can be found in the work of Pallister *et al.* [83].

NAME	B	M	I	FP	CATEGORY	DESCRIPTION
`blowfish`	L	M	H	L	networking	symmetric block cipher encryption algorithm
`crc32`	M	L	H	L	automotive	cyclic redundancy check
`cubic`	L	M	H	L	networking	root calculation of cubic equations
`dijkstra`	M	L	H	L	consumer	graph shortest-path algorithm
`fdct`	H	H	L	H	automotive	image compression algorithm
`float_matmult`	M	H	M	M	automotive	floating-point matrix multiplication
`int_matmult`	M	M	H	L	security	integer matrix multiplication
`rijndael`	H	L	M	L	networking	symmetric block cipher encryption algorithm
`sha`	H	M	M	L	automotive	cryptographic hash function
`2dfir`	H	M	L	H	security	image transformation algorithm

algorithm, which rearranges deterministically elements in an array. The reason that not all benchmarks were run on all the testbeds is that a hardware failure rendered the Exynos 4210 SoC of the Galaxy S2 unusable, very likely due to excessive thermal stress.

4.3.2 Execution Time and Power Measurements

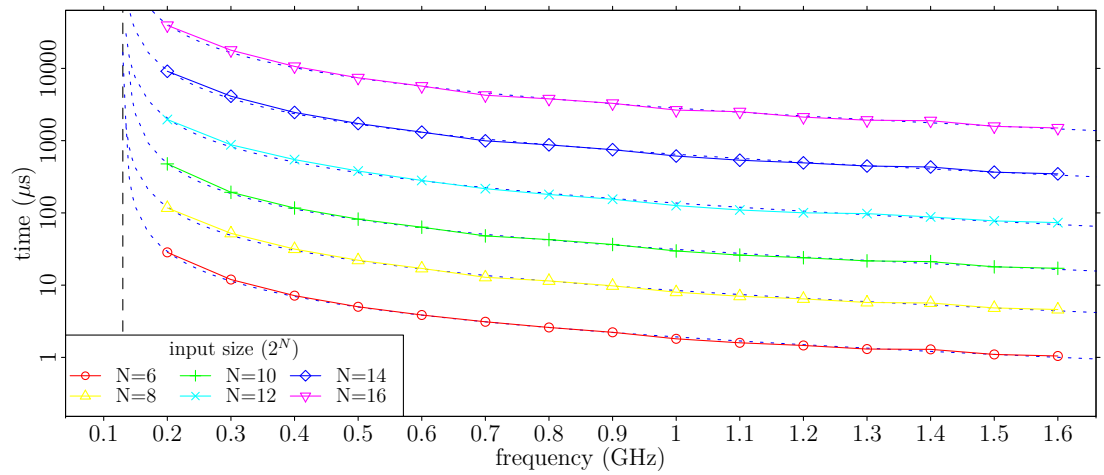
The power consumption and the execution time of the benchmarks were measured separately to obtain an estimate of the energy consumption using Equation 4.2.

Execution Time

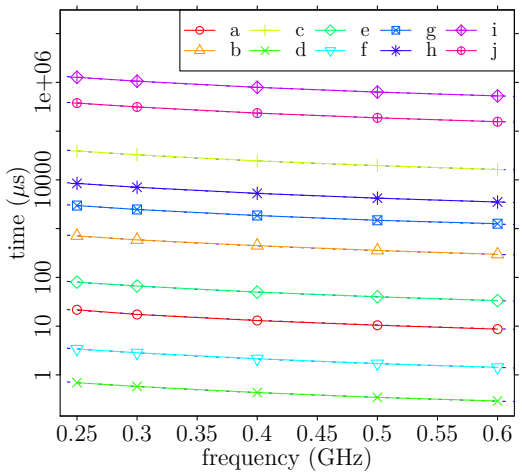
Figure 4.4 shows the execution time of the Gold-Rader algorithm on the A9 and an excerpt of the BEEBS benchmarks on the A7 and A15. The traces for different input sizes look similar and therefore not depicted.

Table 4.2 shows a subset of the fitted execution time parameters as per Equation 2.16. For the A7 and A15 traces the parameters f_k and β were initially measured to be negligibly small or zero and were therefore left out of the final fitting process to prevent overfitting practices. This was not the case for the A9 traces, however. The reason for this is that the second core of the A9 was disabled whereas all the cores were available on the A7 and the A15. Thus on the A9 the benchmark had to share the microprocessor with other processes. On the A7 and A15 the benchmark was scheduled on a separate core where the benchmark had all the clock cycles available for itself, thus $f_k \approx 0$. Memory accesses were limited or well hidden by the cache for the A7 and A15 as β was measured to be close to zero.

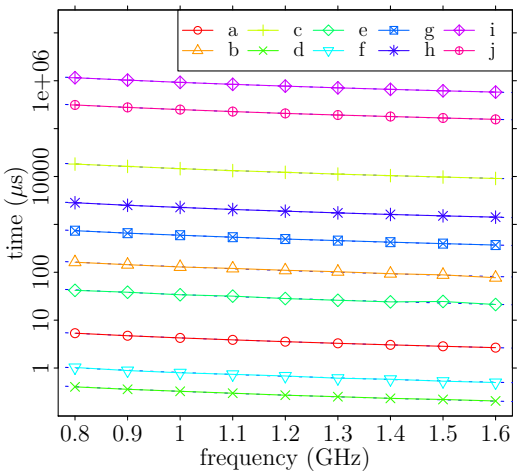
The fitted execution time model has a relative absolute error with 90 % of the errors between 0.18 % and 7.36 % and a median of 3.12 % for the A9 execution time traces. The A15 execution time model fit has 90 % of its values between 0.02 % and 3.57 % and a median of 0.31 %. The A7 execution time model fit has 90 % of its values between 0.01 % 2.32 % and a median of 0.15 %. In general, the errors are an order of magnitude larger for the A9 traces compared to the A7 and A15 traces. This may be because the benchmark in the case of the A9 was running alongside other code on the same microprocessor core, e.g., the OS, which was not the case for the A7 and A15 cases. This resource sharing may introduce some variation in the observed execution times.



(a) Gold-Rader on A9



(b) BEEBS on A7



(c) BEEBS on A15

Figure 4.4: Execution time experimental measurement traces on the Cortex A7, A9 and A15 microprocessors. The execution time of different input sizes is shown for the Gold-Rader algorithm and the BEEBS benchmarks. The solid lines represent the measured data whereas the dotted lines is the data fitted on the execution time model (Equation 2.16). The BEEBS benchmarks are labeled as follows: a) `2dfir`, b) `blowfish`, c) `crc32`, d) `cubic`, e) `dijkstra`, f) `fdct`, g) `float_matmult`, h) `int_matmult`, i) `rijndael`, and j) `sha`. The execution time for the BEEBS benchmarks were measured for multiple input sizes, only a random input size is selected to be plotted however, as they all look alike.

Table 4.2: Benchmark execution time model parameters: cc_b, f_k and β as per Equation 2.16 for running the Gold-Rader algorithm on the A9 microprocessor and the BEEBS benchmark on the A7 and A15 microprocessors. These values were used for the fitted models in Figure 4.4. f_k and β for the A7 and A15 execution time models were found to be negligibly small or zero. Only a subset of cc_b is shown for the A7 and A15.

	GOLD-RADER – INPUT SIZE 2^N – A9					
N	6	8	10	12	14	16
cc_b	1.943	8.596	31.1	144.359	670.8	2918.837
f_k	0.134	0.129	0.137	0.13	0.13	0.129
β	-0.166	-0.167	-0.152	-0.202	-0.183	-0.182

	BEEBS – A7 & A15									
	$cc_b \times 10^{-3}$									
	2dfir	blowfish	crc32	cubic	dijkstra	fdct	float_matmult	int_matmult	rijndael	sha
A7	0.005	0.178	9.816	0.173×10^{-3}	0.02	0.001	0.744	2.113	317.435	93.995
A15	0.004	0.13	14.604	0.326×10^{-3}	0.034	0.001	0.593	2.26	926.348	249.964

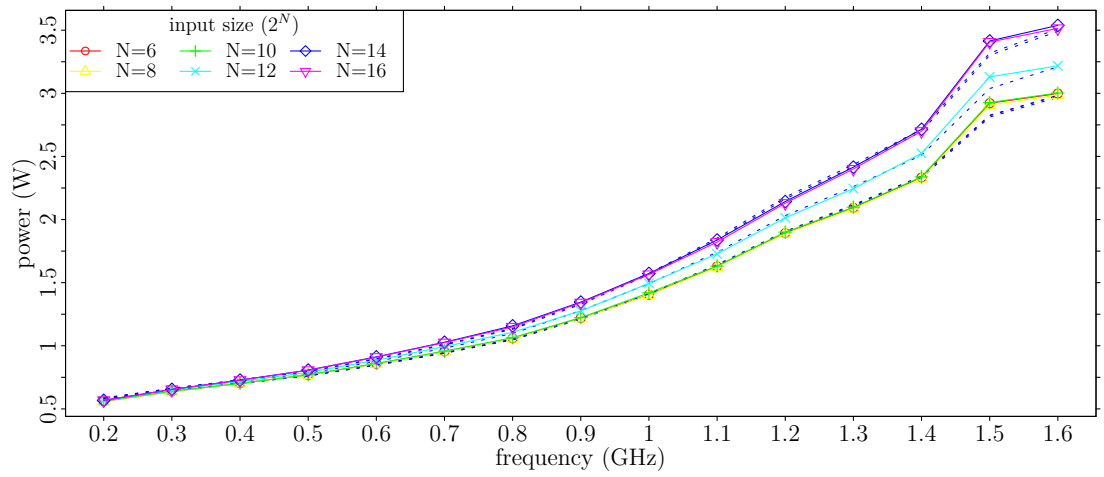
Power Consumption

Figure 4.5 shows the power consumption of the Gold-Rader algorithm on the A9 microprocessor and excerpts of the BEEBS benchmarks on the A7 and A15 microprocessors. The traces for each benchmark with different input sizes of BEEBS look similar and therefore not all are depicted. All traces were recorded while the temperature of the hardware fluctuated. During the recording of the A9 traces the temperature of the testbed was artificially oscillated around 37°C and then the power samples with a temperature of 37°C were selected. The A7 and A15 power traces were converted to 37°C based on the transformation formula given by Procedure 3.12.

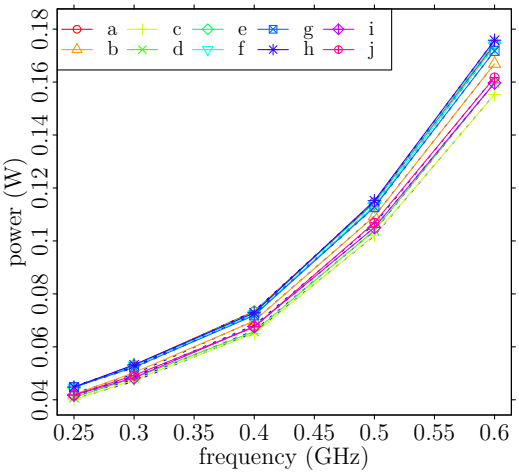
Table 4.3 shows the fitted values for ξ, γ and P_{back} as per Equation 2.11 for the A9 microprocessor, and a subset of the parameters for the A7 and A15 microprocessors. Discrete voltage/ frequency pairs were used to fit the measured data as reported in Figure 4.3 for the Exynos 4210 and Exynos 5410 microprocessors.

As observed from Figure 4.5 the power model fits well on the experimental data. The fitting errors for the A7 are between 0.01 % and 1.42 % with a median of 0.31 %. For the A15 the fitting errors are between 0.06 % and 2.79 % with a median of 0.67 %. And for the A9 the fitting errors are between 0.07 % and 3.18 % with a median of 0.86 %. The fitted model for the A9 in Figure 4.5(a) for $f = 1.5\,\text{GHz}$ seems to deviate persistently from the measured data. This could be due to a slightly higher supply voltage at 1.5 GHz than reported in Figure 4.3 for the Exynos 4210 microprocessor.

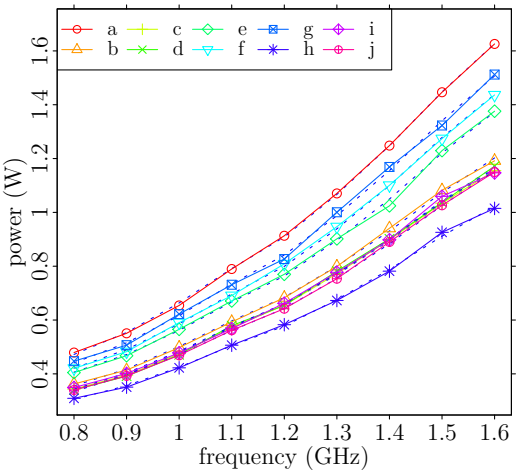
The fitted model parameters in Table 4.3 for the A9 seem to be consistent for an input size up to 2^{12}. The fitted model parameters for larger input sizes seem to be much different. Note that array sizes up to 2^9 fit in the L1 cache, while sizes over 2^{18} are too big to fit in the L2 cache. Therefore external memory accesses and microprocessor slack time may influence the power expenditure of the microprocessor. This is also reflected in the power model parameters of the BEEBS benchmarks; there seem to be no consistency



(a) Gold-Rader on A9



(b) BEEBS on A7



(c) BEEBS on A15

Figure 4.5: Power consumption experimental measurement traces on the Cortex A7, A9 and A15 microprocessors. The power consumption of the benchmarks with different input sizes is shown for the Gold-Rader algorithm and a subset of the traces for the BEEBS benchmarks. The solid lines represent the measured data whereas the dotted lines is the data fitted on the power model (Equation 2.11). The BEEBS benchmarks are labeled as follows: a) `2dfir`, b) `blowfish`, c) `crc32`, d) `cubic`, e) `dijkstra`, f) `fdct`, g) `float_matmult`, h) `int_matmult`, i) `rijndael`, and j) `sha`. The power consumptions for the BEEBS benchmarks were measured for multiple input sizes; only a subset is selected to be plotted however.

Table 4.3: Benchmark power model parameters: ξ, γ and P_{back} as per Equation 2.11 for running the Gold-Rader algorithm on the A9 microprocessor and the BEEBS benchmark on the A7 and A15 microprocessors. These values were used for the fitted models in Figure 4.5. Only a subset of ξ, γ and P_{back} is shown for the A7 and A15.

	GOLD-RADER – INPUT SIZE 2^N – A9					
N	6	8	10	12	14	16
ξ	0.101	0.108	0.134	0.137	0.44	0.011
γ	5.578	5.127	4.030	4.36	1.035	65.985
P_{back}	0.480	0.480	0.477	0.469	0.394	0.407

		BEEBS – A7 & A15									
		$cc_{\text{b}} \times 10^{-3}$									
		2dfir	blowfish	crc32	cubic	dijkstra	fdct	float_matmult	int_matmult	rijndael	sha
A7	ξ	0.565	0.595	0.73	1.254	0.748	1.058	0.886	0.836	0.634	0.639
	$\gamma \times 10^{-3}$	6.959	5.906	7.095	8.409	7.551	9.019	8.767	7.778	7.178	6.818
	P_{back}	0.11	0.103	0.087	0.066	0.097	0.08	0.087	0.092	0.095	0.097
A15	ξ	2.192	0.618	0.624	0.599	20.155	6.492	2.874	1.675	0.656	4.14
	$\gamma \times 10^{-3}$	84.526	39.107	32.402	21.595	109.447	99.307	89.409	59.195	37.847	74.986
	P_{back}	0.201	0.313	0.302	0.313	0.024	0.073	0.151	0.151	0.296	0.086

in the values for the different benchmarks. After all, consistency is not expected as each benchmark performs different types of computations and hence stresses different parts of the microprocessor. This results in different power consumption profiles for each distinct benchmark and implies that power consumption is application-specific. Overall, the power consumption variation for the different benchmarks and different input sizes are not as large as what was observed for the case of the execution time. The power magnitude of all traces are all of the same order, whereas for the execution time the order of magnitude may differ by multiple orders.

In the sequel, it will be useful to have an expression for the upper and lower bounds on the power consumption of the Exynos 5410. The goal is not to be as accurate as possible. A simple expression is desired to describe the power consumption space with as few parameters as possible. Figure 4.6 shows the upper bounds and lower bounds of the measured power consumption of the A7 and the A15 microprocessors taken over all BEEBS benchmarks. A quadratic polynomial was used to fit the upper-bounds and lower bounds. A single scaling factor is introduced to scale the polynomial's coefficients such that a representative power consumption curve in function of the microprocessor's frequency is generated. Table 4.4 shows the polynomial's coefficients and their respective scaling factor for the A7 and A15 microprocessor. The power consumption, in function of the microprocessor's frequency f, is calculated as follows: $P(f) = s_x(s_2 f^2 + s_1 f + s_0)$. As can be seen from Figure 4.6 the quadratic approximation is reasonably well able to describe the upper bounds and lower bounds of the power consumption.

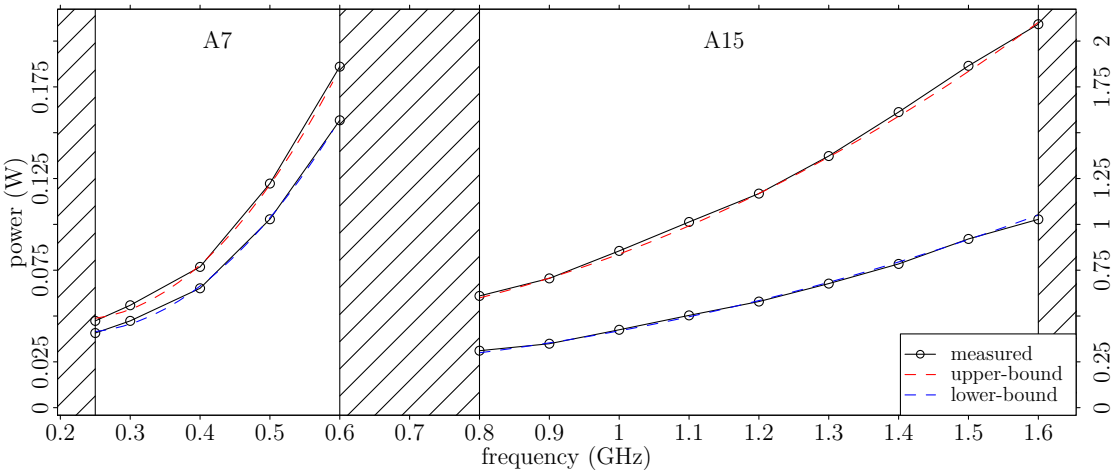


Figure 4.6: Power consumption upper bound and lower bound for the A7 and A15 microprocessors. The boundaries are fit with quadric polynomials. A single scaling factor, further on referred to as s_x, is able to scale between the upper and lower bounds for each microprocessor. The coefficients and scaling factors are shown in Table 4.4. The left vertical axis indicating power is dedicated to the A7 microprocessor whereas the right axis is representative for the A15 microprocessor.

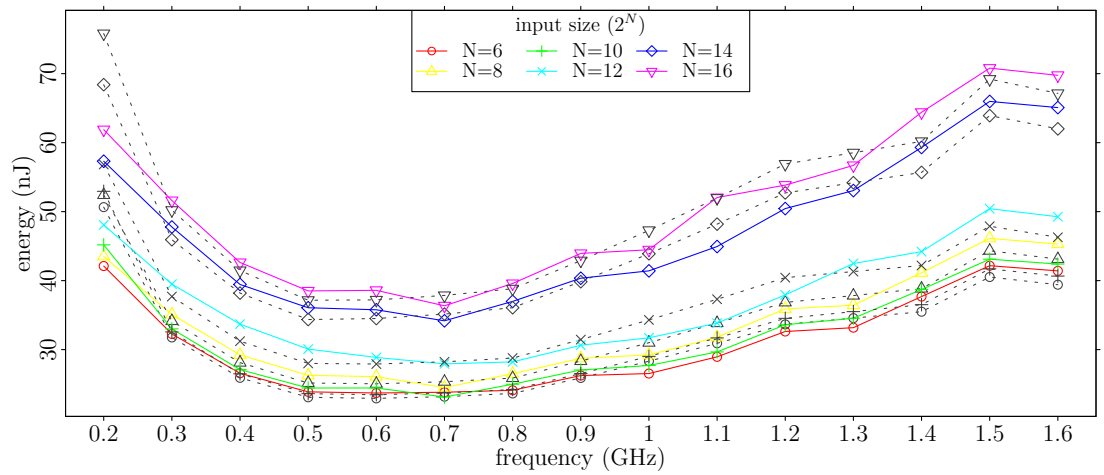
Table 4.4: Coefficients for the upper and lower bounds on the power $P(f)$ of the Cortex A7 and Cortex A15 microprocessor. The power consumption, in function of the microprocessor's frequency f, is computed as $P(f) = s_x(s_2 f^2 + s_1 f + s_0)$. The scaling factor is used to scale the polynomial coefficients to generate the upper and lower bounds, but also to generate intermediate power consumption levels.

	Scaling Factor s_x		Polynomial Coefficients		
	$\min(s_x)$	$\max(s_x)$	s_0	s_1	s_2
A7	1	1.175	0.083	-0.373	0.828
A15	1	2	0.276	-0.426	0.569

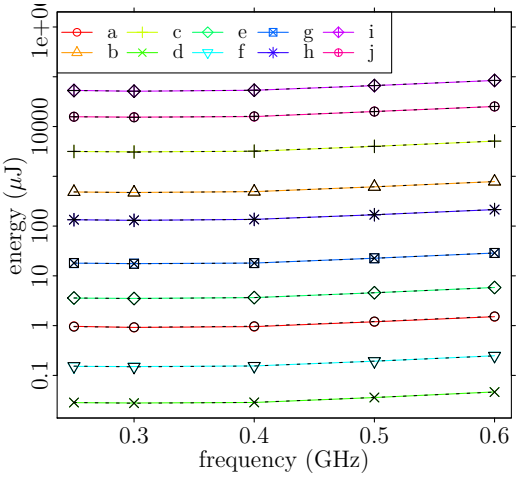
4.3.3 Energy Consumption

The estimated experimental energy consumption are obtained by multiplying the power traces with the execution time traces for each frequency. This was done for both the experimental traces and the fitted power and execution time models. Figure 4.7 shows the energy consumption of the Gold-Rader algorithm on the A9 microprocessor and an excerpt of the BEEBS benchmarks on the A7 and A15 microprocessors. The traces for different input sizes look similar and therefore not depicted. The fitted errors are the sum of the errors of the power and execution time traces separately. It is observed that the fitting errors are larger for the A9 case than for the A7 and A15.

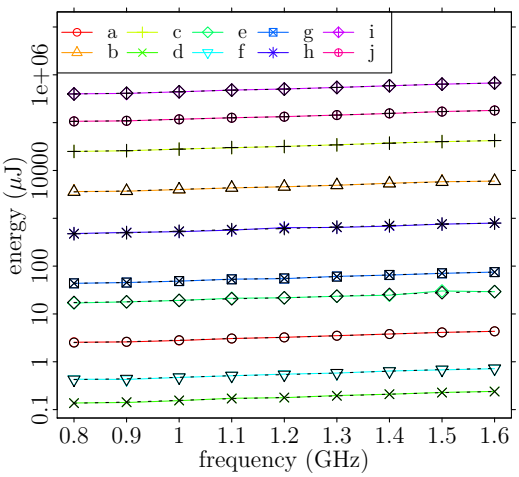
For the A9 traces a clear energy convex minimum is seen between 500 MHz and 800 MHz. The A7 traces show a convex minimum at 300 MHz to 200 MHz, however, the minimum is not as clear as in the A9 case. For the A15 traces a convex minimum is not present within this frequency range, the minimum energy consumption is at the microprocessor's minimum frequency: 0.8 GHz.



(a) Gold-Rader on A9



(b) BEEBS on A7



(c) BEEBS on A15

Figure 4.7: Experimental energy consumption data for the Cortex A7, A9 and A15 microprocessors. The energy consumption of the benchmarks with different input sizes is shown for the Gold-Rader algorithm and the BEEBS benchmarks. The solid lines represent the measured data whereas the dotted lines is the product of the fitted power and execution time models from Figure 4.5 and Figure 4.4, respectively. The BEEBS benchmarks are labeled as follows: a) `2dfir`, b) `blowfish`, c) `crc32`, d) `cubic`, e) `dijkstra`, f) `fdct`, g) `float_matmult`, h) `int_matmult`, i) `rijndael`, and j) `sha`. The execution time for the BEEBS benchmarks were measured for multiple input sizes; only a subset of the input sizes are selected to be plotted.

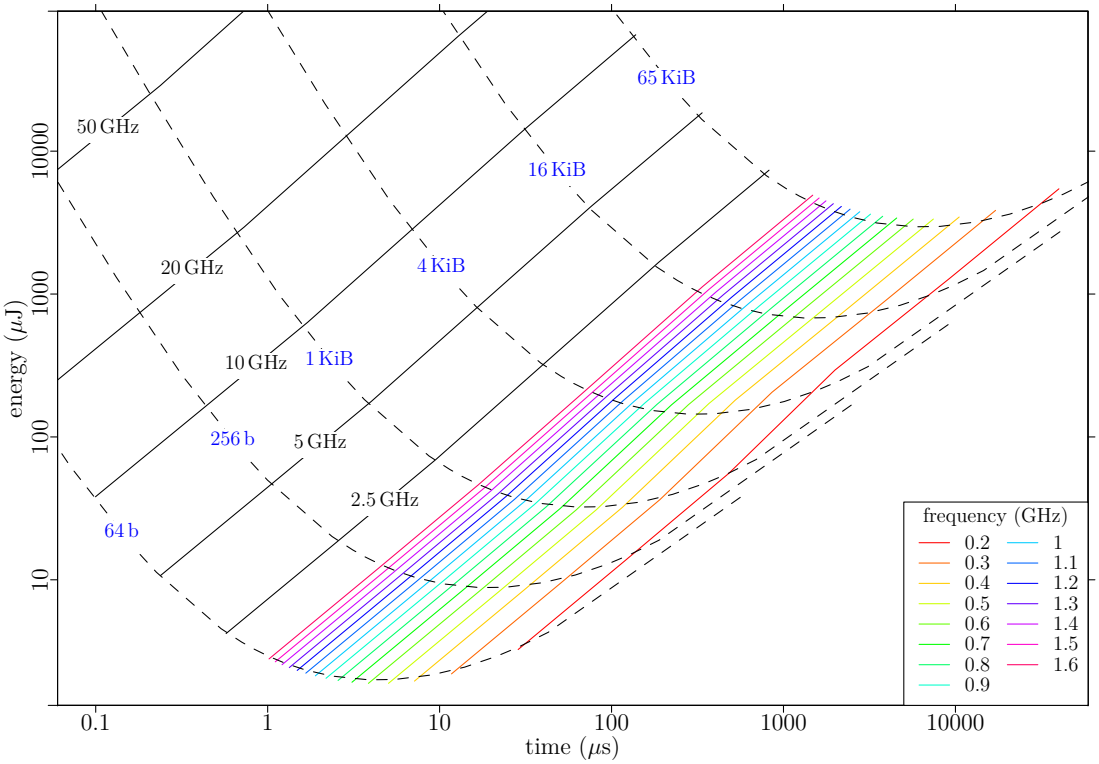


Figure 4.8: Energy consumption and execution time relationship at different frequencies for the bit-reverse algorithm benchmark on the A9. The dashed lines represent the benchmark's performance with a given input size (data size). The solid lines is the benchmark's performance at different microprocessor clock frequencies. The colored lines are within the microprocessor's operating range, whereas the solid black lines are extrapolated theoretical values.

4.3.4 Energy Consumption and Execution Time Relationship

Figure 4.8 shows the relationship between energy consumption and the execution time of the benchmark, based on the A9 measurements and Equation 4.2. It can be observed that for this benchmark the energy consumption and the execution time exhibit a linear relationship. This is also apparent from Equation 4.2 where cc_{b} is a scaling factor of E_{sys}. This implies that performance optimization is the same as optimizing for energy in this case. Such findings have been pointed out before in the literature. Pallister *et al.* [84] showed via different compiler settings that the energy needed to execute a sequence of code is approximately linearly proportional to the time needed to execute that sequence, i.e., $\propto cc_{\mathrm{b}}$. Bellosa [7] showed similar measurements for integer and floating point operations. This is in line with the observation that the energy needed to execute a single instructions doesn't differ largely [20, 108]. The total microprocessor's energy consumption however also depends on the length and the state of the pipeline, inter-instruction Hamming distance, number of operands, or the inter-instruction effects [20]. A random sequence of instructions will nonetheless yield a total energy consumption that is approximately proportional to the number of instructions in the sequence.

We point out that increasing the microprocessor frequency from 0.2 GHz to 0.4 GHz speeds up the program by a factor of 4, whereas a frequency increase from 0.8 GHz to 1.6 GHz improves execution time by a factor of 2.5. Decreasing the execution time requires a super-linear clock frequency need. Moreover, the energy required to sustain this frequency increase becomes relatively increasingly larger than the time gain. For example,

for an input size of 4 KiB, to speed-up by three-fold between 0.7 GHz and 1.6 GHz, 1.46 times more energy is required. Similarly, to speed up the benchmark three-fold from 1.6 GHz the processor needs to be theoretically clocked at 3.6 GHz, that would require 1.9 times more energy. These conclusion can also be deducted from Equation 4.2.

4.4 Sensitivity of the Convexity Model

To analyze the behavior and sensitivity of the convexi model of Equation 4.2, the Cortex A9 processor of the Exynos 4210 is used as reference use case, representative for embedded multimedia applications, e.g., smartphones [29]: $m_1 = 0.330$ [V/f], $m_2 = 0.808$ [V], $\beta = 0$ [s], $\gamma = 3.137$ [V^{-1}], $f_k = 0.130$ [GHz], $\xi_{max} = 0.181$ [W/(GHz·V^2)], ξ_{min}= 0.155 [W/(GHz·V^2)], and the microprocessor's clock frequency starts at 200 MHz and goes to 1.6 GHz. ξ is a parameter that describes the power profile of an application. The values for β, f_k, γ and ξ were defined via fitting as presented in the previous sections. The microprocessor's clock frequency is also considered a continuous variable in this section. In reality the clock frequency is limited to a discrete set. But for analytical purposes, not to mention the aesthetics of the graphs, the clock frequency is deemed continuous.

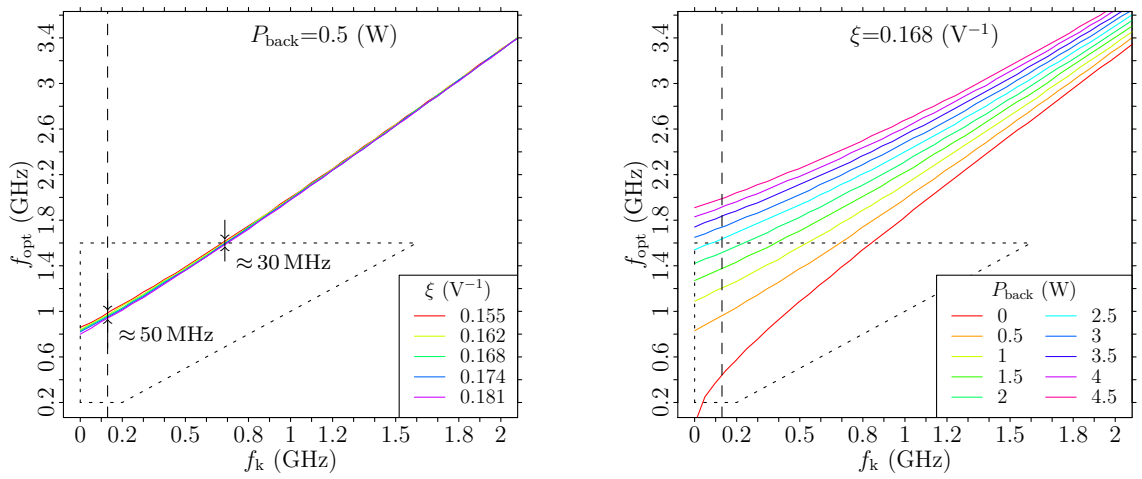
4.4.1 What About Those frequency thieves?

When considering the execution time of a code sequence, f_k was previously defined as the number of clock cycles per time unit not available to the execution of the code. These clock cycles are spent, for example, to handle microprocessor exceptions, or to execute operating system routine tasks. f_k can therefore be regarded as little frequency thieves. From a mathematical point of view, the presence of f_k in Equation 4.2 also introduces some complexity to derivations such as Equation 4.11. Bear in mind that the microprocessor's clock frequency f must always be larger than f_k; otherwise the execution time is not defined. Consequently, $f_k < f_{max}$ must be satisfied as the microprocessor becomes overloaded at $f_k \geq f$, and hence unusable.

Figure 4.9 shows the sensitivity of f_k with regards to the optimal frequency f_{opt}, the microprocessor power ($P_{cpu} \propto \xi$), and the background power P_{back}. In the left plot it is seen that $f_{opt}(f_k = 0, P_{back} = 0.5) \approx 0.8$ GHz. The optimal frequency increases for increasing values of f_k and hits the microprocessor's maximum frequency $f_{max} = 1.6$ GHz around $f_k = 0.7$ GHz. At this point, about 45 % ($\approx 0.7/1.6$) of the clock cycles would not be available to the code sequence. Furthermore, it is observed that $f_{opt} > f_k$ always holds. The effect of the microprocessor's power demands on f_{opt} is fairly small, expressed by the ξ parameter. A 30 MHz to 50 MHz difference in f_{opt} is observed between the minimum and maximum microprocessor's power usage as ξ varies between 0.155 V^{-1} and 0.181 V^{-1}.

The background power usage P_{back} has a bigger impact on f_{opt} than ξ. For $P_{back} = 0$, f_{opt} even drops below the minimum operation frequency of the microprocessor. Increasing P_{back} inflates f_{opt}. For $f_k = 0$ and $P_{back} \approx 2.5$ W the optimal frequency already surpasses f_{max}. For a typical value of f_k (130 MHz) an increase in f_{opt} is observed for increasing values of P_{back}; yet, the increase becomes smaller for larger values of P_{back}. The average difference between $f_{opt}(f_k = 0)$ and $f_{opt}(f_k = 0.13)$, within the microprocessor's clock frequency range, is approximately 100 MHz.

In the rest of this section it will be assumed for simplicity that $f_k \ll f$ unless otherwise stated. For a more realistic estimate of f_{opt}, in case f_k is not negligible, it was observed from the graphs that adding 100 MHz to f_{opt} is a reasonable assumption.



(a) $f_{\text{opt}}(f_{\text{k}},\xi)$ (b) $f_{\text{opt}}(f_{\text{k}},P_{\text{back}})$

Figure 4.9: Optimal microprocessor frequency f_{opt} for variable levels of f_{k} in function of ξ, on the left, and P_{back} on the right. A typical value for f_{k} is drawn at 0.13 GHz (dashed vertical line). The area encapsulated by the dotted line signals the microprocessor's default clock frequency window: $\max(0.2\,\text{GHz}, f_{\text{k}}) \leq f \leq 1.6\,\text{GHz}$.

4.4.2 Absence of frequency thieves

It is not unthinkable that, in particular contexts, f_{k} is negligibly small compared to f: $f_{\text{k}} \ll f$. For example, such occasions may occur when the clock frequency microprocessor is reasonably fast, or the code sequence of concern is running only on one of the available cores of a multi-core microprocessor without interruption. Assuming f_{k} negligible considerably simplifies Equation 4.11. The system of equations for $f_{\text{k}} = 0$ becomes:

$$\frac{\partial E_{\text{n}}^{\text{A}}}{\partial f} = (4af^3 + 3bf^2 + 2cf + d) \cdot \beta \tag{4.18a}$$

$$\frac{\partial E_{\text{n}}^{\text{B}}}{\partial f} = 3af^2 + 2bf + c - \frac{P_{\text{back}}}{f^2} \tag{4.18b}$$

$$\frac{\partial^2 E_{\text{n}}^{\text{A}}}{\partial f^2} = (12af^2 + 6bf + 2c) \cdot \beta \tag{4.18c}$$

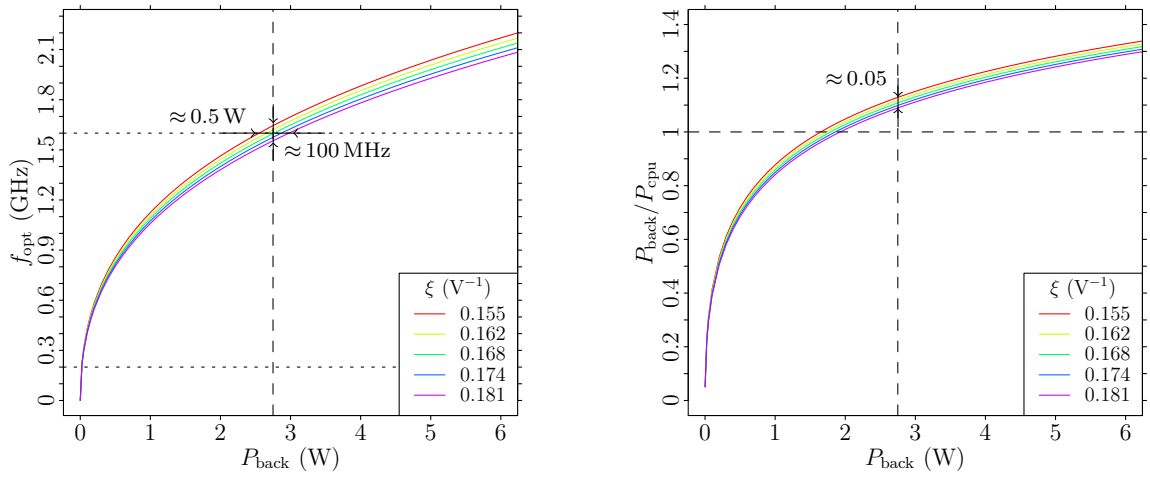
$$\frac{\partial^2 E_{\text{n}}^{\text{B}}}{\partial f^2} = 6af + 2b + 2\frac{P_{\text{back}}}{f^3}. \tag{4.18d}$$

For $\max(f_{\text{min}}, f_{\text{k}}) < f_{\text{opt}} < f_{\text{max}}$, E_{n} was said to be strictly convex iff there exist only one point in the exploitable clock frequency window for which $\frac{\partial E_{\text{n}}}{\partial f} = 0$ and $\frac{\partial^2 E_{\text{n}}}{\partial f^2} > 0$. Given the system of Equations 4.18, these two requirements translate, respectively, into:

$$4a\beta f_{\text{opt}}^3 + 3(a + b\beta)f_{\text{opt}}^2 + 2(b + c\beta)f_{\text{opt}} + (d\beta + c) = \frac{P_{\text{back}}}{f_{\text{opt}}^2} \tag{4.19a}$$

$$12a\beta f_{\text{opt}}^2 + 6(a + b\beta)f_{\text{opt}} + 2(b + c\beta) + 2\frac{P_{\text{back}}}{f_{\text{opt}}^3} > 0. \tag{4.19b}$$

Recall that for all constants in this system of equations: $\{a, b, c, d, \beta\} \in \mathbb{R}^+$. Thus the requirement in Equation 4.19b is satisfied by default as the left-hand side will never be negative. Accordingly, the root requirement of Equation 4.19a is also satisfiable. It is

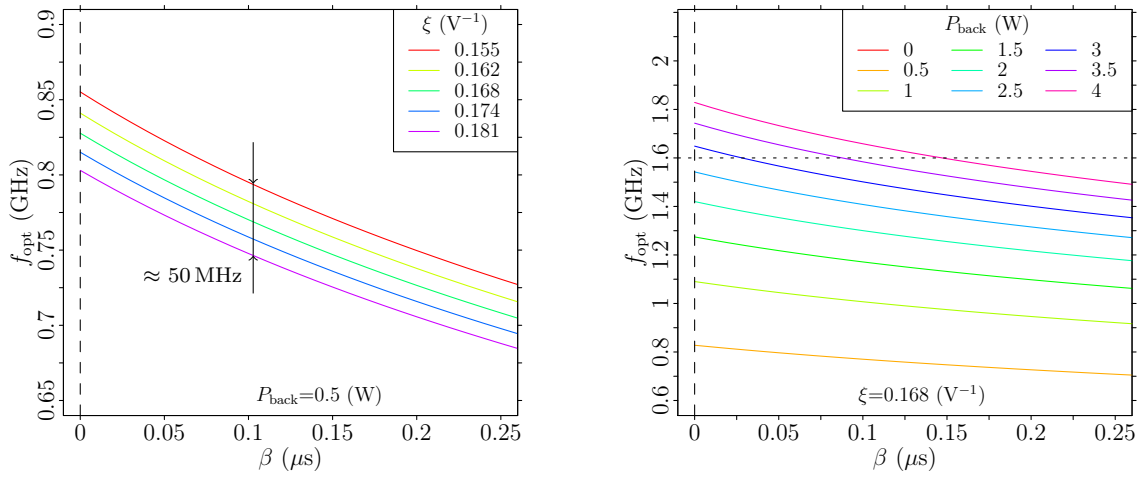


(a) $f_{opt}(P_{back}, \xi)$ (b) P_{back}/P_{cpu} ratio at f_{opt}

Figure 4.10: Optimal microprocessor frequency f_{opt} for variable background power consumption P_{back}. On the left f_{opt} is shown for various microprocessor loads ξ. On the right the ratio between the background power and the microprocessor power P_{cpu} at f_{opt} is shown. The area between the dotted lines signals the default clock frequency window: $0.2\,\text{GHz} \leq f \leq 1.6\,\text{GHz}$.

immediately clear that the background power demands P_{back} directly controls the optimal frequency f_{opt}. The constants $\{a, b, c, d\}$ describe the microprocessor's power usage whereas P_{back} describes the power demands of everything in the computer system besides the microprocessor. For systems with a large P_{back}, e.g., servers or desktop computers, f_{opt} will therefore be higher than for systems with a low P_{back}, e.g., wireless sensors. Moreover, f_{opt} may be so high that it is larger than the maximum microprocessor's clock frequency.

Figure 4.10 shows the optimal frequency for a variable background power consumption P_{back} and microprocessor loads ξ. Also, the ratio between the microprocessor P_{cpu} and the background P_{back} power consumption is given. The area encapsulated by the dotted line signals the operating range of the microprocessor. For the microprocessor to be able to exploit the minimum-energy operation frequency, the background power consumption needs to be between 0.02 W and about 2.75 W, depending on the exact microprocessor load. The influence of the different microprocessor loads on P_{back} is not significant; at 1.6 GHz there is a 0.5 W difference between P_{back} for ξ_{min} and ξ_{max}. If P_{back} is larger than 2.75 W, it is advised to run the microprocessor at the maximum clock frequency to minimize energy consumption of the system. Under such conditions, the energy optimization technique known as *race-to-halt* is a good strategy. This was also Yuki and Rajopadhye's [137] main conclusion while studying high-performance computers. The optimal frequency f_{opt} surpasses the microprocessor's maximum frequency roughly around the point where the background power demands become larger than the microprocessor's power usage. Battery-powered electronic systems such as embedded systems, wireless sensors or smartphones aim at minimizing their background power demands, which thus increases the feasibility of f_{opt} exploitation. For more powerful computers, however, such as servers, the optimal frequency will be very likely out of reach of the microprocessor's capabilities: $f_{opt} > f_{max}$. For example, Seo *et al.* [104] claim that DVFS in general hardly improves the energy efficiency of mobile multimedia electronics. The testbed power measurements of their embedded system show, however, that their P_{cpu} to P_{back} ratio is



(a) $f_{\text{opt}}(\beta,\xi)$ (b) $f_{\text{opt}}(\beta,P_{\text{back}})$

Figure 4.11: Optimal microprocessor frequency f_{opt} for variable levels of β in function of ξ, on the left, and P_{back}, on the right. The area between the horizontal dotted lines signals the microprocessor's default clock frequency window (0.2 GHz $\leq f \leq$ 1.6 GHz).

smaller than 1 to 18, and their m_1 is very small. For their specific testbed, f_{opt} is very likely larger than f_{max}, and race-to-halt should indeed be most beneficial when aiming for energy savings.

4.4.3 Out-of-Order Execution

Out-of-order execution (OOE) is parametrized via $\sigma_{\text{x}} \in [0,1]$, which is a subpart of β in Equation 4.6: $\sigma_{\text{x}} = 1$ for a system without OOE, $\sigma_{\text{x}} = 0$ when OOE is perfectly able to cover the time during external memory accesses with data-independent code execution. The system's normalized energy consumption, assuming $f_{\text{k}} \approx 0$, is given by:

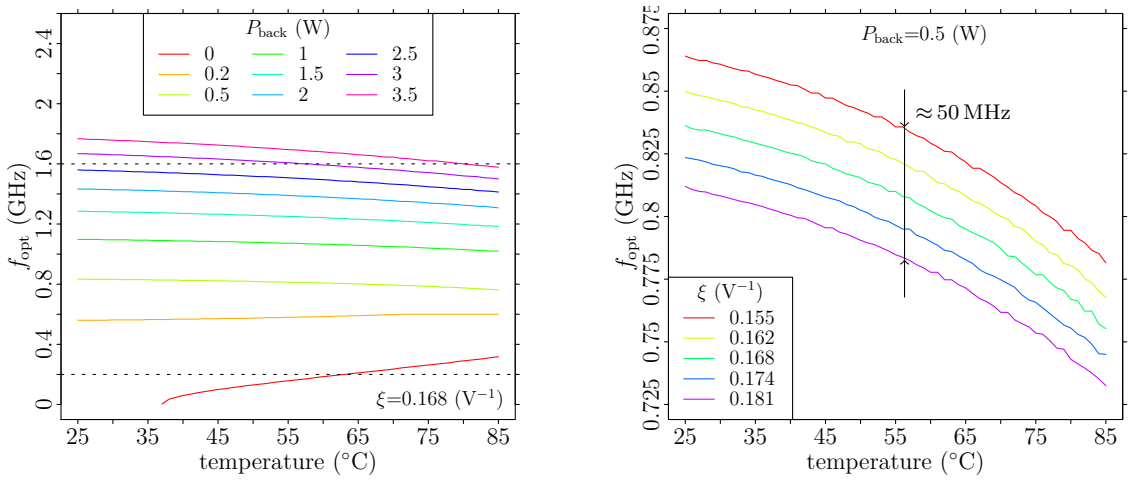
$$E_{\text{n}} = (af^4 + bf^3 + cf^2 + df + P_{\text{back}}) \cdot \left(\frac{1}{f} + \sigma_{\text{x}}\beta\right).$$

Its requirements for convexity are then defined as:

$$(4af_{\text{opt}}^3 + 3bf_{\text{opt}}^2 + 2cf_{\text{opt}} + d) \cdot \sigma_{\text{x}}\beta + \frac{3af_{\text{opt}}^4 + 2bf_{\text{opt}}^3 + cf_{\text{opt}}^2}{f_{\text{opt}}^2} = \frac{P_{\text{back}}}{f_{\text{opt}}^2} \tag{4.20a}$$

$$(12af_{\text{opt}}^2 + 6bf_{\text{opt}} + 2c) \cdot \sigma_{\text{x}}\beta + \frac{6af_{\text{opt}}^4 + 2bf_{\text{opt}}^3 + 2P_{\text{back}}}{f_{\text{opt}}^3} > 0. \tag{4.20b}$$

It can be observed that for $\sigma_{\text{x}} = 0$ the most left-hand term in Equation 4.20a becomes zero, resulting in an increased f_{opt} for the equality to be satisfied. Similarly, the larger σ_{x} or β, the more f_{opt} needs to decrease for the inequality of Equation 4.20b to hold. Figure 4.11 shows the sensitivity of the β parameter on the optimal frequency f_{opt}. The parameter β is scaled here because the exact value of M_m is not known. Scaling β has the same effect as changing σ_{x}. Indeed, from the figure, it is observed that f_{opt} decreases for increasing β. Moreover, f_{opt} changes about 100 MHz over a 0 to 0.25 μs β range for medium levels of P_{back}. The larger P_{back}, the larger the spread in f_{opt} for variable β. For P_{back} over 4 W, the f_{opt} spread between $\beta = 0$ and $\beta = 0.25$ increases to more than 200 MHz.



(a) $f_{opt}(T,P_{back})$ (b) $f_{opt}(T,\xi)$

Figure 4.12: Optimal microprocessor clock frequency f_{opt} for variable temperature levels T in function of ξ, on the left, and P_{back}, on the right. The temperature is varied between 25°C and 85°C. The area between the horizontal dotted lines signals the operating range of the microprocessor (0.2 GHz $\leq f \leq$ 1.6 GHz). The power model of the A9 is used, mixed with the temperature/power model of the A15 because of the lack of a decent A9 temperature/power model.

In theory, σ_x can be frequency-dependent as well. That is, the memory clock frequency can be scaled along with the microprocessor's frequency, this to ensure the timely delivery of data in the microprocessor registries and caches. σ_x in such case would not be constant over f. In Section 6.1 a model will be developed for the optimal clock frequency in n-buddy systems, where multiple entities collaborate while having their respective energy profile and clock frequencies.

Here, it was assumed that the microprocessor's clock frequency, once set at f_{opt}, doesn't change over time. Another common approach to save energy is to have a variable clock frequency to minimize OOE slack-time and also energy consumption. This is discussed in Chapter 6.

4.4.4 Temperature Dependency

The temperature of the microprocessor has influence on its power and energy consumption. This primarily originates because of the leakage currents that are shown to be temperature-dependent as discussed in Chapter 2. The parameter γ controls the leakage current of the power equation, and is a function of temperature and others, as defined in Equation 2.10. In Section 2.1.4, it was shown, however, that such formulation to describe the leakage current is probably not the best possible. The A9 traces are unfortunately too noisy to be useful to derive an accurate temperature/power relationship. As an alternative, the temperature/power model of the A15 is borrowed and plugged into Equation 2.12 to get an estimate of f_{opt}'s temperature dependency.

Figure 4.12 shows f_{opt} for various temperature levels T in function of the microprocessor's power, represented by ξ, and the background power demands P_{back}. Over the temperature range 25°C $< T <$ 85°C, it can be seen in Figure 4.12(a), that the optimal frequency f_{opt} may variate about 200 MHz. The optimal frequency is decreasing for in-

creasing temperatures above $f_{opt} = 600\,\text{MHz}$. That is because the larger the temperature, the larger the leakage currents and power drawn. To offset this leakage power the processor runs slower to consume less dynamic power. Below 600 MHz, on the other hand, the optimal frequency increases for increasing temperatures. Now, the increased leakage power is offset by a faster execution. The execution time decrease per Hertz is larger closer to 0 Hz than at higher frequencies, because of its asymptotic properties. Therefore, below 600 MHz, running faster decreases the execution time more than the total power consumption increases, proportionally. Hence energy consumption is minimized.

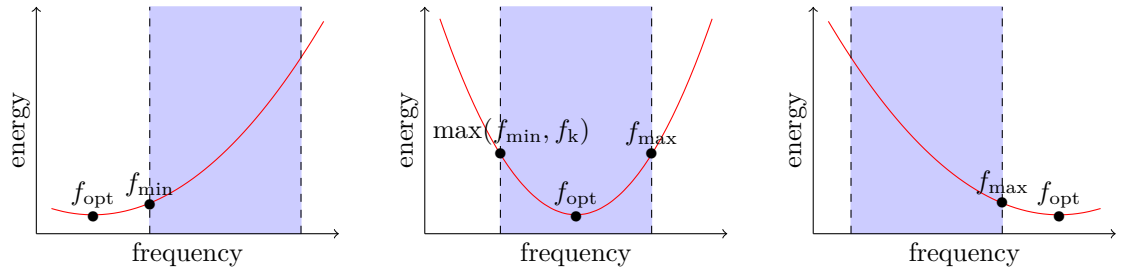
Figure 4.12(b) shows the optimal frequency for various levels of microprocessor power levels. As seen in the previous graph, the optimal frequency decreases for increasing temperatures. The optimal frequency f_{opt} changes about 50 MHz over the whole temperature range and maximum and minimum power levels of the microprocessor. Given that a contemporary microprocessor often operates with 100 MHz clock frequency steps, the temperature thus has likely minimal effect on the optimal clock frequency for various application power profiles. However, as observed in Figure 4.12(a) the difference in frequency in the temperature extremities becomes larger for larger levels of background power demands.

4.5 Conclusion

In this chapter the energy consumption equation of a microprocessor operating in a computer system with other components is developed and analyzed. An analytical analysis, along with numerical simulation and measurement data, is used to study the behavior and sensitivity of its parameters. It was shown via an analytical framework, measurements, and literature review that the energy consumption curve shows convex properties with regard to the clock frequency of the microprocessor. The convex minimum is the point with a given clock frequency f_{opt} where the computer system consumes the minimum amount of energy while executing a code sequence. The energy saving gained by running at the optimal clock frequency is a trade-off with the performance of the system, in terms of execution time. For applications requiring human interaction, it has been shown by Seeker *et al.* [101], however, that the clock frequency can be scaled down considerably without affecting the user's experience.

Given the energy/frequency convex behavior, three classes of code execution can be distinguished as shown in Figure 4.13. When the optimal clock frequency f_{opt} is left of the default clock frequency window: $f_{opt} < f_{min}$, executing as slow as possible yields the best energy gains; if $\max(f_{min}, f_k) < f_{opt} < f_{max}$ then chasing f_{opt} will earn the best energy efficiency; and when $f_{opt} > f_{max}$, then the race-to-halt energy optimization technique was shown to be most effective. It was noted by Rizvandi [95] that under certain circumstances it can be more efficient, in terms of energy consumption, to have a binary frequency scheme with the maximum and minimum clock frequency rather than scaling the clock frequency through the whole frequency space. The presented performance-oriented work, and also the user-oriented work of Seeker *et al.* [101], suggest that this is, in fact, not the case. f_{opt} may assume any frequency within the default clock frequency window, and may fluctuate throughout the code execution depending on the kind of operations scheduled.

The existence of the energy/frequency convexity property was further confirmed via experimental measurement traces on two multimedia microprocessors commonly used for embedded system applications. The main conclusions of the analysis are:



(a) $f_{opt} < f_{min}$ (b) f_{opt} exploitable (c) $f_{opt} > f_{max}$

Figure 4.13: The location of the optimal frequency f_{opt} w.r.t. default clock frequency window (blue) is an indication for which energy optimization technique is most effective: (a) when f_{opt} is left of the exploitable clock frequency window: $f_{opt} < f_{min}$, one should set the clock frequency as low as possible; (b) if $\max(f_{min}, f_k) < f_{opt} < f_{max}$ then chasing f_{opt} will yield the best energy efficiency; (c) when $f_{opt} > f_{max}$ then the race-to-halt energy optimization technique is most effective.

- Energy/frequency convexity occurs always, but, to be exploitable, f_{opt} should be within the exploitable clock frequency window: $\max(f_{min}, f_k) < f_{opt} < f_{max}$;
- The background power requirement (P_{back}) is the parameter that influences the optimal frequency the most; the larger the background power demands, the larger the optimal clock frequency: when P_{back} equals P_{cpu}, f_{opt} will be close to the maximum microprocessor clock frequency;
- An application's power profile (ξ) has a minimal effect on the optimal frequency, mostly because the variations in power profiles are fairly small, an average of 50 MHz in f_{opt} between the power profile's extremities was oserved;
- The number of instructions of a code sequence (cc_b) has no influence on the optimal clock frequency, following the energy consumption model, but does scale the energy consumption linearly on the premise that ξ has minimal effect;
- Application concurrency and clock cycle thieves (f_k) significantly affect the optimal frequency; the less clock cycles available to the applications, the larger the optimal clock frequency: on average for a 1 GHz increase in f_k, f_{opt} increases by 2 GHz;
- Microprocessor slack time (β), during off-chip operations, forces the optimal clock frequency down: 300 MHz for $0 < \beta < 0.25$ in the extreme case;
- The temperature, influencing mostly the leakage currents (γ), has a notable effect on the optimal frequency, in the extreme case 200 MHz between $25°C < T < 85°C$;
- The *race-to-halt* strategy is justified only when the optimal clock frequency is larger than the microprocessor's maximum frequency.

Given that P_{back} has a large effect on the optimal frequency f_{opt} it was shown that a system with a P_{back} of the order of P_{cpu} and larger will have a f_{opt} likely outside the reach of the microprocessor's clock frequency range. Thus chasing the optimal clock frequency f_{opt} is especially beneficial for low-power systems, such as for embedded applications, as their P_{back} is much smaller than what would be expected for high performance computer systems.

Chapter 5

Passive Cooling of Microprocessors

An exact analytical solution to the heat transfer equation of a passively cooled microprocessor is developed in this chapter, subject to convection, radiation, and internal heat generation. Such models are useful for Thermal Management Units (TMUs) and Dynamic Thermal Management (DTM) to evaluate and predict the thermal behavior of a microprocessor. Whereas before the power-dependent thermal behavior of microprocessor was studied, here the transient thermal behavior is developed. Analyzing the exact thermal behavior of a microprocessor is, however, intractable due to the complex nature of the physics involved including fluid dynamics, all heat transfer modes, complex electronic component configurations, turbulence, reattachement, conjugate problems etc. [99] To curb the complexity, a simplified reality will be assumed where an isothermal piece of silica oxide, representing a microprocessor, is placed in an infinite open space subject to (natural) convection and radiation. Such conditions are also assumed when dealing with Newton's law of cooling.

First, the temperature's temporal behavior of a microprocessor is considered, subject to Newton's law of cooling and internal heat generation, in Section 5.2. Then, in Section 5.3, the impact of radiation and internal heat generation is added to the model, to develop the passively cooled heat equation for a microprocessor with internal heat generation. The performance difference between the passive and active coolings law is also assessed in Section 5.4. Besides, some approximations to the exact solution of the passive cooling law are proposed in Section 5.3.6, as it will turn out to be a fairly elaborate equation. The thermal runaway of the passive heat equation applied to a realistic microprocessor model is also studied.

5.1 Cooling in Thermal Management Techniques

Thermal management techniques for microprocessors have been devised to control their heat dissipation. Excessive heat dissipation may have adverse effects on performance, energy consumption, and the short term and long term failure rates of the microprocessor. Basic run-time thermal management techniques can be rudimentary, such as clock gating. Yet, if service continuation is needed, more advanced thermal techniques are required. Thermal-aware design of microprocessors can also be effective to minimize peak and average heat and power dissipation during run time. The challenge here, however, lies in decision making based on incomplete design and run-time details.

To get a current perspective on how such issues are addressed in the literature, top computer architecture and VLSI conferences are surveyed for papers devoted to micro-

processor thermal models for TMUs and DTMs and temperature-aware design methods based on heat transfer theory. The conferences surveyed are ISCA, MICRO, ASPLOS, HPCA, PACT, ISLPED, ICCAD, DAC, DATE and ASP-DAC from 2010 to 2014. 35 papers were identified focusing on the thermal optimization of microprocessors using heat transfer models. 90 % of these papers base their results solely upon simulation or numerical analysis; the remaining ones use either actual measurements or a combination of simulation and measurements to make their point. Beside custom thermal simulators and models, non-commercial and open-source thermal simulators are mostly used: these are based on finite-element methodologies. Commercial applications such as COMSOL Multiphysics®, Autodesk Simulation CFD or FLoTHERM®, which support the radiative heat transfer mode, are not used in the selected papers. About 40 % of the selected papers deploy Hotspot for their thermal simulations. Hotspot [55] is a self-proclaimed accurate and fast thermal model designed for microprocessor architectural analysis, e.g., floor planning. The basic setup of Hotspot includes active cooling via a heat sink. No passive cooling capabilities are available in Hotspot. Other experimental simulators, such as LightSim [100], CONTILTS [49], ISAC [133] and PowerBlurr [85], also allow for thermal analysis of microprocessors, but are less popular and none support radiative cooling. In most of the simulations, both the temperature at steady state and transient temperatures are available, where the steady-state case is much faster to compute than the transient behavior.

It is worthwhile to ponder upon why no non-commercial simulators support radiative cooling. One reason could be that the non-linear behavior of radiation is not easy to handle in mathematical formulations and advanced finite element techniques need to be employed in numerical simulations. Also it is not always clear to what extent radiation actually affects the thermal behavior of semiconductors. As a result, given the lack of passive cooling support in most simulators, it is not surprising that passive cooling has not gotten much attention in the thermal management research community. In fact, only one paper [122] was found, about 3D integrated circuits, which mentions that radiation may influence the thermal behavior of microprocessors; yet in this work no further reference to radiation is found. Nevertheless, 30 % of the papers surveyed claim that their research is applicable to mobile embedded systems, a situation in which passive cooling is usually of the essence.

Beside generic thermal microprocessor simulators, dedicated embedded system thermal simulators were also developed. Therminator [131], for example, is a thermal simulator designed to simulate heat dissipation in smartphones. Finite element methodologies are used to compute the heat propagation through an arbitrary smartphone configuration, which may include a PCB, battery, case, display etc. The authors show that their dedicated thermal simulator produces results that are close to what commercial computational fluid dynamics (CFD) software would calculate. Therminator takes the convective and conductive heat transfer modes into account. Heat loss via radiation, however, is not implemented in their thermal simulator. Luo et al. [73] analyzed the issue of thermal management on mobile phones based on numerical simulation and basic thermal models. The authors came up with design proposals on how to improve the thermal management of mobile phones by studying the steady-state behavior of the system. Even though radiation is mentioned in the introduction, including formulations, radiation is not present in their steady-state analysis. Gurrum et al. [45] decomposed, just as Luo et al. [73], a hand-held device in multiple subparts with different physical properties and analyzed its thermal behavior. Radiation, however, did not come to their attention.

From our literature survey it is concluded that the numerical tools used for thermal behavior of embedded systems can be classified into three categories. First, the general-purpose CFD software, which is able to simulate arbitrary systems including all modes of heat transfer. These systems require the most efforts to produce interesting results. The second class corresponds to dedicated embedded system simulators. It was observed that the designers of the simulators are aware of surface radiation but they do not provide support in their simulators. And last, which are the most popular, are the generic microprocessor thermal simulators. None of these microprocessor simulators were noticed to support the radiative heat transfer mode. This provides a strong motivation for this work, which strives to understand the possible impact of radiation on the transient and steady-state thermal behaviors of micro-processors in the context of embedded systems.

5.2 Active cooling: the Newtonian Approach

Actively cooled computer systems spend energy to forcibly cool down the system. The most basic active cooling technique is an air fan mounted directly on the microprocessor or on a heat sink. Water and cooling fluids-based cooling devices are more effective but also more expensive, more complex to maintain and more hazardous for the hardware. Examples of technologies under development for active thermal management of portable electronic devices are phase change materials, micro heat pipes, conductivity materials such as carbon [43], thermoelectric cooling, and two-phase refrigerant cooling.

Newton's law of cooling states that the temperature rate of change of an object is proportional to the difference between the ambient temperature and the object's temperature. TMUs and DTMs often assume the system to cool down following the resulting exponential Newton's law of cooling. Accordingly, it is implicitly assumed that the system's power consumption, as a result, also exhibits exponential behavior over time, comparable to an RC network. Such assumptions are frequently found in experimental thermal management systems [21, 27, 39, 53, 58, 60, 140]. For actively cooled systems, i.e., cooled using forced convection (such as computer fans or water cooling), an exponential assumption is a good approximation when radiative and conductive cooling may be neglected.

Assume that for such systems the stored energy is approximated by the sum of the heat transfer induced by *convectional cooling*: $h_{\text{ac}}S(T_a - T)$, and an *internal heat generation* (ihg), which is deemed linear as a first-order approximation: $\eta_1 T + \eta_0$, then

$$\begin{aligned} C\frac{dT}{dt} &= \text{convection} + \text{internal heat generation} \\ &= h_{\text{ac}}S(T_a - T) + (\eta_1 T + \eta_0)\,, \end{aligned} \tag{5.1}$$

where C is the body's heat capacity and T_a the ambient temperature. Note that, if the active cooling system consists of a fan and heat sink, then h_{ac} depends upon the dimensions of the heat sink or body surface area, and the revolutions per minute (rpm) of the fan. Moreover, η_1 and η_0 are also dependent on the microprocessor's clock frequency, type of computations, and the load on the system. Similar to Weissel and Bellosa's [127] work, resulting from Equation 5.1:

$$\begin{aligned} C\frac{dT}{dt} &= h_{\text{ac}}S(T_a - T) + (\eta_1 T + \eta_0) \\ C\frac{dT}{dt} &= -(h_{\text{ac}}S - \eta_1)T + (\eta_0 + h_{\text{ac}}ST_a) \end{aligned}$$

$$\begin{aligned}
\int \frac{1}{T - \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1}} dT &= -\int \frac{(h_{\text{ac}} S - \eta_1)}{C} dt \\
\ln\left(T - \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1}\right) &= -\frac{(h_{\text{ac}} S - \eta_1)}{C} t + c_o \\
T - \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1} &= c_o e^{-\frac{(h_{\text{ac}} S - \eta_1)}{C} t},
\end{aligned} \tag{5.2}$$

while imposing the initial condition at $t = 0$: $T(0) = T_0$. Therefore $c_0 = T_0 - \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1}$, and thence

$$T_{\text{ac}}(t) = \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1} + \left(T_0 - \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1}\right) e^{-\frac{(h_{\text{ac}} S - \eta_1)}{C} t}. \tag{5.3}$$

It is clear that such a system is only stable if the cooling process with constant h_{ac} convects heat away from the system faster than the system is generating heat internally. The system is stable if there exists an equilibrium temperature T_e for the system, which is equivalent to stating that

$$0 = h_{\text{ac}} S(T_a - T_e) + (\eta_1 T_e + \eta_0) \quad \Rightarrow \quad h_{\text{ac}} = \frac{\eta_1 T_e + \eta_0}{S(T_e - T_a)}, \tag{5.4}$$

where all constants $\{T_e, T_a, \eta_1, \eta_0\} \in \mathbb{R}^+$. It can be stated, given that h_{ac} must be positive: $\exists h \rightarrow T_e > T_a$. From Equation 5.4 it can also be concluded that h_{ac} is always larger than η_1/S. If $h_{\text{ac}} < \eta_1/S$, the exponent in Equation 5.3 would go to infinity over time. In practical applications the value of h_{ac} must be dimensioned properly such that the system's T_e stays below the maximum operation temperature.

Not surprisingly, Newtonian cooling with linear internal heat generation yields again an exponential relationship between temperature and time. Consequently, the power P consumed by the system, which is an affine transformation of temperature ($P = \eta_1 T + \eta_0$), will also exhibit exponential behavior. An exponential model for actively cooled systems with linear (or constant, $\eta_1 = 0$) internal heat generation is therefore a valid approximation. The exponential assumption is however not quite the same as assuming Newtonian cooling, as the coefficients in both models are different, mainly due to the presence of the internal heat generation. In the case of the presence of internal heat generation, the equilibrium temperature T_e of the system will be larger than the ambient temperature T_a:

$$T_e = T(\infty) = \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1} + \left(T_0 - \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1}\right) e^{-\infty} = \frac{\eta_0 + h_{\text{ac}} S T_a}{h_{\text{ac}} S - \eta_1}.$$

This is the same result as obtained in Equation 5.4.

The resulting cooling law for a system with an internal heat generation following a second-order polynomial can be derived similarly as before. Its solution is then given by:

$$\begin{aligned}
C\frac{dT}{dt} &= h_{\text{ac}} S(T_a - T) + (\eta_2 T^2 + \eta_1 T + \eta_0) \\
T_{\text{ac}}(t) &= \frac{\omega_1 + \omega_2 e^{-\frac{\kappa_2}{A} t}}{1 + e^{-\frac{\kappa_2}{A} t}} + c_o,
\end{aligned}$$

where the constants κ_2, c_o and ω_* are as follows:

$$\begin{aligned}\kappa_2 &= \eta_2 \\ \omega_* &= \text{RootsOf}[\eta_2 T^2 + (\eta_1 - h_{\text{ac}} S)T + (\eta_0 + h_{\text{ac}} S T_a)] \\ A &= \frac{1}{\omega_2 - \omega_1} \\ c_o &= T_e - \omega_1.\end{aligned}$$

Such system is, however, a bit more complex than Newton's exponential law of cooling but will be similar in behavior.

5.3 Passive Cooling with Internal Heat Generation

Microprocessors that are not actively cooled must rely on passive cooling for the system to attain a temperature equilibrium state. Passive cooling implies that no energy is spend to enforce the system to cool down, e.g, via a spinning fan. Passive cooling, just as active cooling, happens through the three fundamental modes of heat transfer. Though, the convection in this case may be considerably smaller than when the system is actively cooled. The convection arising here may be originating from buoyancy forces, or natural movement of air, e.g., wind. For this reason, sometimes the convection is referred to as *natural convection* as the movement of air is not forced onto the system.

Assume an isothermal body subject to radiative cooling and convection with internal heat generation. The temperature change of such an object at any given point in time is equal to the heat absorbed from the environment, plus the internal heat generation, minus the heat released to the environment. Absorption of heat, and the release of heat to the environment can happen via radiation and convection. The temperature change of such a system, with internal heat generation (ihg), can then be represented by the following equation:

$$\begin{aligned}C\frac{dT}{dt} &= \text{radiation} + \text{convection} + \text{internal heat generation} \\ &= \epsilon\sigma S(T_a^4 - T^4) + h_{\text{pc}} S(T_a - T) + (\eta_1 T + \eta_0),\end{aligned} \tag{5.5}$$

where C is the heat capacity of the body, S is the radiation surface of the object, ϵ is the emissivity of the body, and σ is the Boltzmann constant. Here it is assumed that the internal heat generation is linearly dependent on the temperature of the body: $H(T) = \eta_1 T + \eta_0$. Yet, higher order polynomials (up to the 4th order) can be used as well for the following derivation to hold (see Section 5.3.3).

By rearranging Equation 5.1 the following is obtained:

$$\frac{dT}{dt} = \frac{1}{C}[-\epsilon\sigma S T^4 + (\eta_1 - h_{\text{pc}} S)T + (\eta_0 + S(h_{\text{pc}} T_a + \epsilon\sigma T_a^4))]. \tag{5.6}$$

Here, the right-hand side is a 4th-order polynomial. This becomes clear when the equation is rearranged and the following variables are introduced $\kappa_4 = \frac{\epsilon\sigma S}{C}$, $\kappa_1 = \frac{\eta_1 - h_{\text{pc}} S}{C}$, and $\kappa_0 = \frac{\eta_0 + (h_{\text{pc}} T_a + \epsilon\sigma T_a^4) S}{C}$:

$$\frac{dT}{dt} = -\kappa_4 T^4 + \kappa_1 T + \kappa_0. \tag{5.7}$$

In the next section the differential equation, given by Equation 5.7, is solved. Checking the units of this equation yields a unit equality (keep in mind that $W = J/s$):

$$\begin{aligned}
\kappa_4 &= \frac{e\sigma S}{C} = \frac{W}{m^2K^4}m^2\frac{K}{J} = \frac{1}{sK^3} \\
\kappa_1 &= \frac{(\eta_1 - h)S}{C} = \left(\frac{W}{m^2K} - \frac{W}{m^2K}\right)m^2\frac{K}{J} = \frac{1}{s} \\
\kappa_0 &= \frac{\eta_0 + (hT_a + e\sigma T_a^4)S}{C} = \left[W + \left(\frac{W}{m^2K}K + \frac{W}{m^2K^4}K^4\right)m^2\right]\frac{K}{J} = \frac{K}{s}.
\end{aligned}$$

And so the units of Equation 5.7 equate:

$$\frac{K}{s} = \frac{1}{sK^3}K^4 + \frac{1}{s}K + \frac{K}{s}.$$

5.3.1 Exact Solution of $f(T) = t$

The exact solution is now developed for a differential equation of the form

$$\frac{dT}{dt} = -\kappa_4 T^4 + \kappa_3 T^3 + \kappa_2 T^2 + \kappa_1 T + \kappa_0, \tag{5.8}$$

where the constants $\kappa_4 \in \mathbb{R}_0^+$ and $\kappa_{\{0,1,2,3\}} \in \mathbb{R}^+$. The solution to this equation is found by separating first the temperature and time variables on the opposite sides:

$$\begin{aligned}
\frac{dT}{-\kappa_4 T^4 + \kappa_3 T^3 + \kappa_2 T^2 + \kappa_1 T + \kappa_0} &= dt \\
\int \frac{1}{T^4 - \frac{\kappa_3}{\kappa_4}T^3 - \frac{\kappa_2}{\kappa_4}T^2 - \frac{\kappa_1}{\kappa_4}T - \frac{\kappa_0}{\kappa_4}}dT &= -\kappa_4 \int dt.
\end{aligned} \tag{5.9}$$

The integration of the fraction on the left-hand side can be achieved via partial fraction decomposition:

$$\int \frac{1}{T^4 - \frac{\kappa_3}{\kappa_4}T^3 - \frac{\kappa_2}{\kappa_4}T^2 - \frac{\kappa_1}{\kappa_4}T - \frac{\kappa_0}{\kappa_4}}dT = \int \frac{1}{(T-\omega_1)(T-\omega_2)(T-\omega_3)(T-\omega_4)}dT \tag{5.10}$$

The roots of the 4th-order polynomial in the denominator are obtained via Ferrari's theorem [15]. Given that there exist a maximum at one or two real values for T that satisfy

$$\kappa_4 T^4 = \sum_{i=0}^{3} \kappa_i T^i,$$

it can be stated that two roots are real, let's say $\omega_{\{1,2\}}$; the other two roots are complex conjugates (see Section 5.3.3)[1]. This means that $\Re(\omega_3) = \Re(\omega_4)$ and $\Im(\omega_3) = -\Im(\omega_4)$, and simplifies a few things. As the initial differential equation is real, a real solution is looked for as well; thus the imaginary part must equate to zero. This is, however, automatically taken care of as the product of the two complex roots yield a real sum:

$$\begin{aligned}
\frac{1}{(T-\omega_3)(T-\omega_4)} &= \frac{1}{(T - [\Re(\omega_3) + i\Im(\omega_3)])(T - [\Re(\omega_4) - i\Im(\omega_4)])} \\
&= \frac{1}{(T - \Re(\omega_3))^2 + \Im(\omega_3)^2}.
\end{aligned}$$

[1] Appendix A.1.1 explains the case where there are four real, and four complex roots. Such configurations have no physical meaning in the context of our application, however.

Whence, Equation 5.10 becomes

$$\int \frac{1}{T^4 - \frac{\kappa_3}{\kappa_4}T^3 - \frac{\kappa_2}{\kappa_4}T^2 - \frac{\kappa_1}{\kappa_4}T - \frac{\kappa_0}{\kappa_4}} dT = \int \left[\frac{A}{(T-\omega_1)} + \frac{B}{(T-\omega_2)} + \frac{CT+D}{(T-\Re(\omega_3))^2 + \Im(\omega_3)^2} \right] dT. \quad (5.11)$$

Henceforth $\alpha = \Re(\omega_3)$ and $\beta = |\Im(\omega_3)|$ and $\omega_1 < \omega_2$ are defined. Then the following equality must be solved to obtain the values of A, B, C and D:

$$\frac{1}{(T-\omega_1)(T-\omega_2)((T-\alpha)^2+\beta^2)} = \frac{A}{(T-\omega_1)} + \frac{B}{(T-\omega_2)} + \frac{CT+D}{(T-\alpha)^2+\beta^2}, \quad (5.12)$$

which can be expressed as a system of equations:

$$\begin{cases} 0 &= A + B + C \\ 0 &= D - \omega_1(B+C) - \omega_2(A+C) - 2\alpha(A+B) \\ 0 &= \alpha^2(A+B) + \beta^2(A+B) + 2\alpha(\omega_2 A + \omega_1 B) - (\omega_1+\omega_2)D + \omega_1\omega_2 C \\ 1 &= -\alpha^2(\omega_2 A + \omega_1 B) - \beta^2(\omega_2 A + \omega_1 B) + \omega_1\omega_2 D \end{cases}$$

and can be solved via Gaussian elimination:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ -\omega_2 - 2\alpha & -\omega_1 - 2\alpha & -\omega_1 - \omega_2 & 1 \\ \alpha^2 + \beta^2 + 2\alpha\omega_2 & \beta^2 + 2\alpha\omega_1 + \alpha^2 & \omega_1\omega_2 & -\omega_1 - \omega_2 \\ -\alpha^2\omega_2 - \beta^2\omega_2 & -\alpha^2\omega_1 - \beta^2\omega_1 & 0 & \omega_1\omega_2 \end{bmatrix} \times \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}. \quad (5.13)$$

So the expressions for A, B, C and D are obtained:

$$A = \frac{1}{(\omega_1-\omega_2)((\alpha^2+\beta^2) - \omega_1(2\alpha-\omega_1))} \quad (5.14a)$$

$$B = -A\frac{\alpha^2+\beta^2-\omega_1(2\alpha-\omega_1)}{\alpha^2+\beta^2-\omega_2(2\alpha-\omega_2)} \quad (5.14b)$$

$$C = -(A+B) \quad (5.14c)$$

$$D = A(2\alpha-\omega_1) + B(2\alpha-\omega_2). \quad (5.14d)$$

Continuing with Equation 5.11 yields:

$$\begin{aligned} \int \frac{A}{(T-\omega_1)} + \frac{B}{(T-\omega_2)} + \frac{CT+D}{(T-\alpha)^2+\beta^2} dT &= \int \frac{A}{(T-\omega_1)} dT + \int \frac{B}{(T-\omega_2)} dT + \int \frac{CT+D}{(T-\alpha)^2+\beta^2} dT \\ &= A\ln|T-\omega_1| + B\ln|T-\omega_2| + c_o + \int \frac{CT+D}{(T-\alpha)^2+\beta^2} dT, \end{aligned}$$

where c_o is an integration constant. The last term of the right-hand side may be integrated via substitution, where $u = (T-\alpha)^2$, then $du = 2(T-\alpha)dT$, and also $v = \frac{T-\alpha}{\beta}$, then

$dv = \frac{1}{\beta} dT$:

$$\begin{aligned}
\int \frac{CT + D}{(T-\alpha)^2 + \beta^2} dT &= \int \frac{C(T-\alpha)}{(T-\alpha)^2 + \beta^2} dT + \int \frac{\alpha C + D}{(T-\alpha)^2 + \beta^2} dT \\
&= \frac{C}{2} \int \frac{2(T-\alpha)}{(T-\alpha)^2 + \beta^2} dT + \frac{\alpha C + D}{\beta^2} \int \frac{1}{\left(\frac{T-\alpha}{\beta}\right)^2 + 1} dT \\
&= \frac{C}{2} \int \frac{1}{u + \beta^2} du + \frac{\alpha C + D}{\beta} \int \frac{1}{(v^2 + 1)} dv \\
&= \frac{C}{2} \ln |u + \beta^2| + \frac{\alpha C + D}{\beta} \arctan(v) + c_o \\
&= \frac{C}{2} \ln |(T-\alpha)^2 + \beta^2| + \frac{\alpha C + D}{\beta} \arctan\left(\frac{T-\alpha}{\beta}\right) + c_o.
\end{aligned}$$

Then the solution to Equation 5.8 is as follows

$$\begin{aligned}
\int \frac{1}{T^4 - \frac{\kappa_3}{\kappa_4} T^3 - \frac{\kappa_2}{\kappa_4} T^2 - \frac{\kappa_1}{\kappa_4} T - \frac{\kappa_0}{\kappa_4}} dT &= A \ln |T - \omega_1| + B \ln |T - \omega_2| \\
&\quad + \frac{C}{2} \ln |(T-\alpha)^2 + \beta^2| + \frac{\alpha C + D}{\beta} \arctan\left(\frac{T-\alpha}{\beta}\right) + c_o, \quad (5.15)
\end{aligned}$$

where A, B, C and D are given by Equations 5.14, and ω_* are the real roots of the polynomial in the denominator on the left-hand side, $\alpha = \Re(\omega_3)$, $\beta = \Im(\omega_3)$, and c_o is an integration constant. Now Equation 5.9 can be completed:

$$-\kappa_4 \int dt = \int \frac{1}{T^4 - \frac{\kappa_3}{\kappa_4} T^3 - \frac{\kappa_2}{\kappa_4} T^2 - \frac{\kappa_1}{\kappa_4} T - \frac{\kappa_0}{\kappa_4}} dT \quad (5.16)$$

$$\begin{aligned}
t &= -\frac{1}{\kappa_4} \Bigg(A \ln |T - \omega_1| + B \ln |T - \omega_2| + \frac{C}{2} \ln |(T-\alpha)^2 + \beta^2| \\
&\quad + \frac{\alpha C + D}{\beta} \arctan\left(\frac{T-\alpha}{\beta}\right) + c_o \Bigg), \quad (5.17)
\end{aligned}$$

where the kappa variables for a linear internal heat generation $H(T)$ are:

$$\kappa_4 = -\frac{e\sigma S}{C}, \quad \kappa_3 = \kappa_2 = 0, \quad \kappa_1 = \frac{\eta_1 - h_{\text{pc}} S}{C}, \quad \text{and} \quad \kappa_0 = \frac{(h_{\text{pc}} T_a + e\sigma T_b^4) S + \eta_0}{C}.$$

Similarly, for a second-order $H(T)$ the kappa variables are as follows:

$$\kappa_4 = -\frac{e\sigma S}{C}, \quad \kappa_3 = 0, \quad \kappa_2 = \frac{\eta_2}{C}, \quad \kappa_1 = \frac{\eta_1 - h_{\text{pc}} S}{C}, \quad \text{and} \quad \kappa_0 = \frac{(\eta_1 T_a + e\sigma T_b^4) S + \eta_0}{C}.$$

c_o must satisfy the initial conditions $f(T_0) = 0$:

$$\begin{aligned}
c_o = &-A \ln |T_0 - \omega_1| - B \ln |T_0 - \omega_2| \\
&- \frac{C}{2} \ln |(T_0 - \alpha)^2 + \beta^2| - \frac{\alpha C + D}{\beta} \arctan\left(\frac{T_0 - \alpha}{\beta}\right). \quad (5.18)
\end{aligned}$$

A prototype implementation in R of Equation 5.17 is given in Appendix B.1.1.

A system, whose transient behavior is described by Equation 5.17, is in an equilibrium state if its derivative (Equation 5.8) equates to zero. This happens when the temperature T equals one of the roots of Equation 5.8. Before $\omega_1 < \omega_2$ was defined, and the two other roots are complex conjugates. As ω_1 is almost always negative in the context of this work, see Section 5.3.3, the equilibrium temperature T_e of the system is defined to be ω_2.

5.3.2 Dimensionless Solution $\chi(\theta) = \tau$

Let's normalize the temperature, of the exact solution to the passive heat equation, by introducing the dimensionless quantity

$$\theta = \frac{T}{T_e} \tag{5.19}$$

The equilibrium temperature T_e was previously defined to be the positive real root ω_2 of the 4th-order equation. Given θ and $T_e d\theta = dT$, then:

$$-\kappa_4 T_e^4 \int dt = \int \frac{T_e}{\theta^4 - \frac{\kappa_3}{\kappa_4 T_e}\theta^3 - \frac{\kappa_2}{\kappa_4 T_e^2}\theta^2 - \frac{\kappa_1}{\kappa_4 T_e^3}\theta - \frac{\kappa_0}{\kappa_4 T_e^4}} d\theta$$

The solution to this equation is similar to the previously defined solution in Equation 5.17:

$$t = -\frac{1}{\kappa_4 T_e^3}\Bigg(A \ln|\theta - \omega_1| + B \ln|\theta - \omega_2| + \frac{C}{2}\ln|(\theta-\alpha)^2 + \beta^2| + \frac{\alpha C + D}{\beta}\arctan\left(\frac{\theta-\alpha}{\beta}\right) + c_o\Bigg).$$

When A is brought to the front then

$$t = -\frac{A}{\kappa_4 T_e^3}\Bigg(\ln|\theta - \omega_1| + \frac{B}{A}\ln|\theta - \omega_2| + \frac{C}{2A}\ln|(\theta-\alpha)^2 + \beta^2| + \frac{\alpha C + D}{\beta A}\arctan\left(\frac{\theta-\alpha}{\beta}\right) + c_o\Bigg).$$

Here the fraction A/κ_4, via Equation 5.14a, has the dimension s as

$$\kappa_4 = \frac{1}{K^3 s} \quad \text{and} \quad A = \frac{1}{(K)((K^2) - K(K))} = \frac{1}{K^3}, \quad \text{thus} \quad \frac{A}{\kappa_4} = \frac{sK^3}{K^3} = s.$$

Now, γ can be used to introduce the dimensionless time quantity

$$\tau = \frac{t}{\gamma}. \tag{5.20}$$

Thus the dimensionless solution $\chi(\theta) = \tau$ to the exact heat equation is given by

$$\tau = -\frac{1}{T_e^3}\Bigg(\ln|\theta - \omega_1| + \frac{B}{A}\ln|\theta - \omega_2| + \frac{C}{2A}\ln|(\theta-\alpha)^2 + \beta^2| + \frac{\alpha C - D}{\beta A}\arctan\left(\frac{\theta-\alpha}{\beta}\right) + c_o\Bigg). \tag{5.21}$$

Surprisingly this solution is similar to the solution presented by Besson [10], even though he modeled a different physical problem. In Section 4 of Besson's work on analytical solutions to cooling laws, the author focuses on the combined effect of radiative and convectional cooling, but not internal heat generation. The differential equation formulating his cooling problem shows similarities to our passively cooled microprocessor thermal model. Besson however assumed some simplifications and solved the differential equation

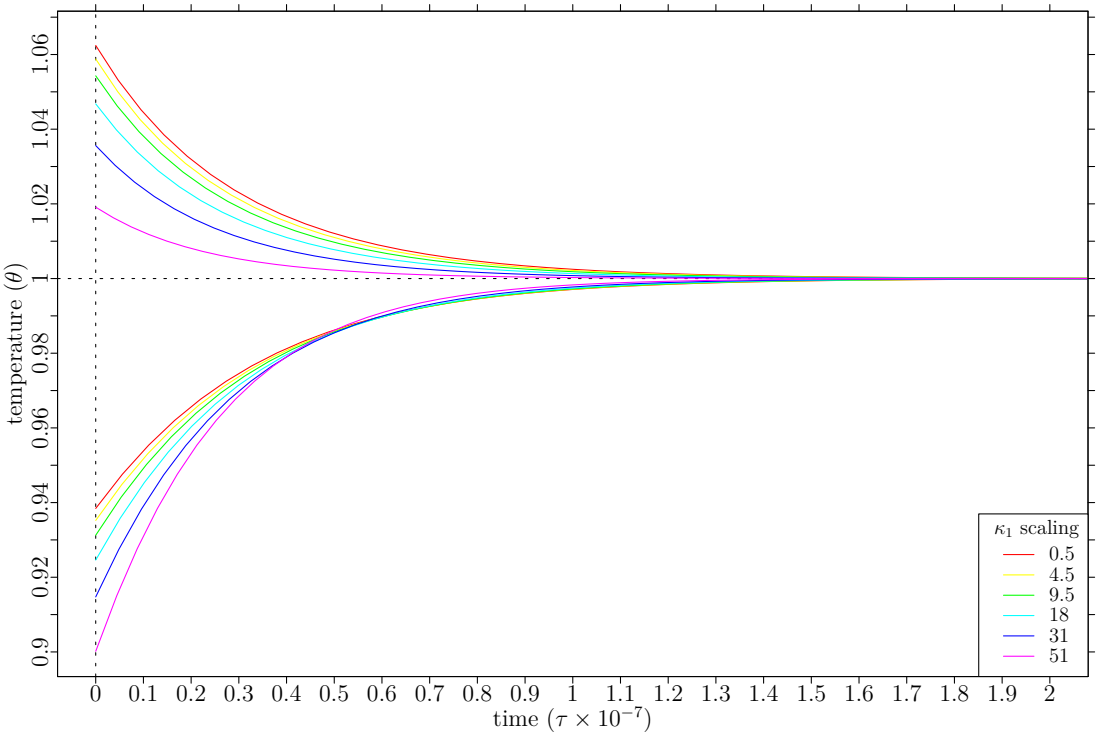


Figure 5.1: Typical graph of the dimensionless cooling law $\chi(\theta)$ as per Equation 5.21 showing the cooling and heating of a body. Constants used are $\kappa_4 = -2.24 \cdot 10^{-10}$, $\kappa_3 = \kappa_2 = 0$, $\kappa_1 = -2.39 \cdot 10^{-13}$, and $\kappa_0 = 0.1$, while κ_1 is scaled according to the legend. The scaling of κ_1 can be understood as a non-linear scaling of the internal heat generation's slope.

via other methods, not via Ferrari's theorem as done here. Nonetheless, his solution also contains three *logarithms*, one of which contains a second-order polynomial, and an *arctan*. Because of Besson's simplifying assumptions, however, his equation is limited to the case where $T - T_a = T + \frac{\eta_1}{\eta_0}$.

Figure 5.1 shows an example of the dimensionless cooling and heating processes as per Equation 5.21. If one looks carefully, especially where the temperature is approaching its asymptote, one can observe that the cooling process happens faster than the heating process. Given the dependence of the radiative cooling on the fourth-power of the temperature, cooling happens faster at larger temperatures than smaller temperatures. If it was not for the presence of the *internal heat generation*, the equilibrium temperature would be equal to the ambient temperature T_a.

5.3.3 Applicability of the Exact Heat Equation

Previously it was assumed that the internal heat generation $H(T)$ was a linear function, i.e., polynomial of the first-order with regression constants $\eta_{\{0,1\}} \in \mathbb{R}^+$. Given that the radiation absorbed or emitted by a body follows a fourth-order polynomial, the implications of an arbitrary internal heat generation $H(T)$ up to the third order is discussed. A heuristic reasoning will show that the analytic solution in the sequel holds for $H(T)$ up to the third order under certain conditions.

Let's define a body that is radiating energy at a rate δ and is subject to other heat transfer mechanisms described by a polynomial $K(T)$, e.g., internal heat generation. Let

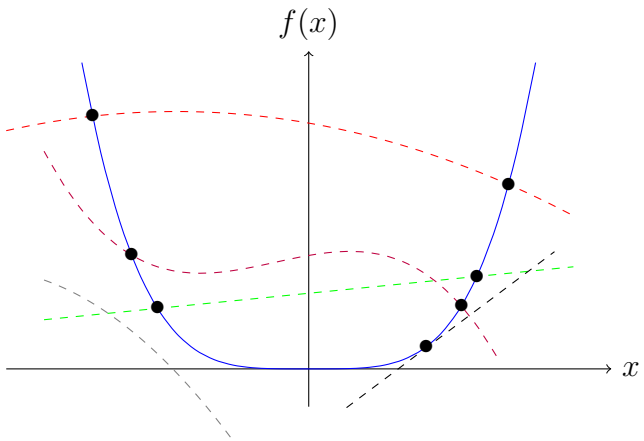


Figure 5.2: Visualization of Equation 5.23 for several variations of the right-hand side polynomial ($K(T)$). Polynomials: 1st order (green,black), 2nd order (red, gray), 3rd order (purple), and $\kappa_4 T^4$ (blue). The black bullets represent the intersects of each polynomial with $\kappa_4 T^4$.

$K(T)$ be a polynomial of an order not higher than three. Then the energy stored (Equation 2.21) into the body at any given time is equal to:

$$\begin{aligned} C\frac{\mathrm{d}T}{\mathrm{d}t} &= -\delta T^4 + K(T) \\ &= -\delta T^4 + (\kappa_3 T^3 + \kappa_2 T^2 + \kappa_1 T + \kappa_0), \end{aligned} \tag{5.22}$$

where $\delta \in \mathbb{R}_0^+$, $\kappa_{0,1,2,3} \in \mathbb{R}$. δ_4 must be positive as $-\delta_4 T^4$ represents the heat emitted by the body via radiation. $\kappa_{0,1,2,3}$ are the constants of a polynomial describing the function $K(T)$. To solve the differential in Equation 5.22, the roots need to be found. This is easily done analytically via Ferrari's theorem [15]. In particular, the differential equation for a fourth order polynomial was solved assuming two real and two complex conjugate roots. Equation 5.22 is evaluated at T_e where $C\mathrm{d}T/\mathrm{d}t = 0$ to find its roots:

$$\delta T_e^4 = \kappa_3 T_e^3 + \kappa_2 T_e^2 + \kappa_1 T_e + \kappa_0. \tag{5.23}$$

This equality is visualized in Figure 5.2, where the solid blue curve represents the contribution on the left-hand side, and the other dashed lines are possible examples of the polynomial on the right-hand side. It can be seen that it is easy to construct polynomials that have two intersections with δT^4. Also, curves can be constructed that intersect δT^4 only in one point (for example the dashed black line in Figure 5.23); such points are counted as double roots. The dashed gray line is an example of a polynomial without any intersection with δT^4.

The *fundamental theorem of algebra* states that an nth-order polynomial has exactly n roots. Thus Equation 5.22 must have exactly four roots. If $K(T)$ has one or two intersections with δT^4, it can be concluded that there exists two real roots, and two complex roots, which are conjugates following the *complex conjugate root* theorem. The complex conjugate root property can be understood as follows. If z is a complex root of a polynomial f of the nth order: $0 = f(z) = \sum_{i=0}^{n} \kappa_i z^i$, then

$$\overline{0} = \overline{\sum_{i=0}^{n} \kappa_i (z)^i} = \sum_{i=0}^{n} \kappa_i \left(\overline{z}\right)^i = 0.$$

Table 5.1: Parameters assumed for the COMSOL validation simulation. Specific values were used for the convective heat transfer coefficient (h) and internal heat generation (ihg) such that the proper equilibrium temperature is attained.

CONSTANTS			VARIABLES			
symbol	value	dim.	symbol	VALUE		dim.
				ihg min	ihg max	
σ	5.670×10^{-8}	W/(m^{2}K^4)	h_{heat}	-4.359	11.144	W/(m^{2}K)
ϵ	0.94	-	h_{cool}	2.764	76.939	W/(m^{2}K)
T_a	20	°C	η_1	1.053	9.407	mW/K
D	2	mm	η_0	0.098	1.318	W
S	0.01	m^2				
C	$S \times D \times 1548709$	J/K				

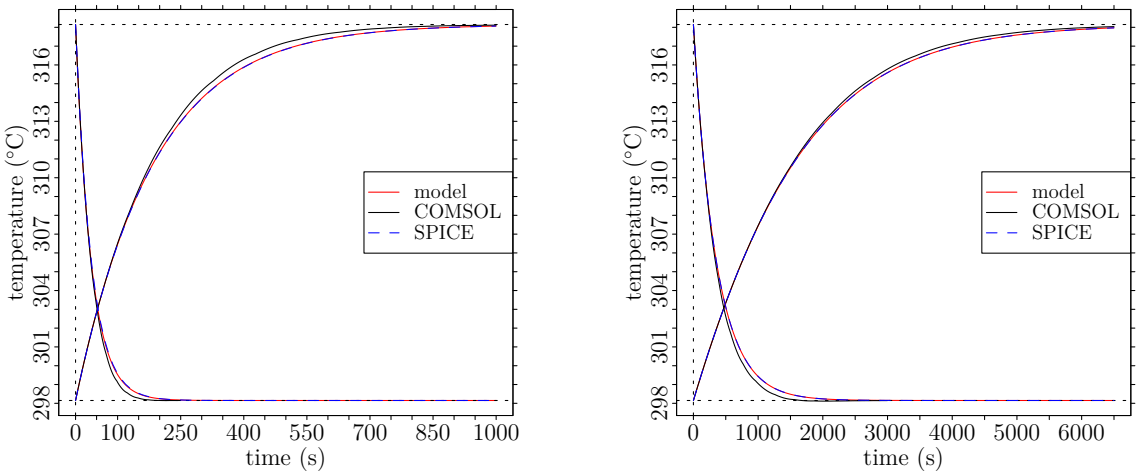
This shows that, if z is a complex root of a polynomial, then its complex conjugate $\overline{z}$ is a root as well.

In Section 5.3.1 an analytical solution was calculated for the differential equation as defined in Equation 5.22. The solution presented is applicable for an arbitrary polynomial $K(T)$ up to the third order with two real and two complex conjugate roots. In Appendix A.1.1 the solutions are given for a polynomial $K(T)$ with four imaginary roots and four real roots.

5.3.4 Validation with Finite Element Analysis

To validate, in the absence of accurate temperature/time experimental data, the passive cooling solution defined in Equation 5.17, a set of CFD simulations is set up in COMSOL, where the transient thermal behavior of a slice of silica glass (SiO_2) is analyzed, as glass has the thermal properties close to the ones of a microprocessor. A 3D conjugate heat transfer scenario was created, with simulation settings as shown in Table 5.1. The exact same values, as listed in this table, were also used for the theoretical model. As can be seen in the settings table, h_{heat} igh min is negative. This implies that heat is added to the system to attain the desired equilibrium temperature. To simulate an isothermal object in COMSOL the thermal conductivity of the silica glass is multiplied by 10^3. The silica glass has a surface area of 0.01 m^2. For the heating process T_0 is set to 25°C and T_e is scaled between T_0 and 45°C. Similarly, for the cooling process T_0=45°C and T_e is scaled between T_0 and 25°C. The temperature values chosen correspond to what is typically encountered when using a mobile device. Linear internal heat generation is used with the parameters as shown in Table 5.1. The convective heat transfer coefficient was set such that, with the given internal heat generation, the proper equilibrium temperature is attained. The two different levels of internal heat generation were derived from ARM Cortex A15 quad-core processor power measurements on the Exynos 5410 SoC introduced in Section 3.5. The minimum internal heat generation represents the A15 processor running a single applications at minimum processor frequency, whereas the maximum internal heat generation represents the A15 processor running at maximum frequency while executing four applications in parallel.

Figure 5.3 shows the transient thermal behavior of the silica glass as described above. On the left, the case for maximum internal heat generation is shown and, on the right, for minimum internal heat generation. Both the cooling and heating processes are shown in the same graph. Data for various surface areas and equilibrium temperatures was generated but all graphs look similar; hence not all are shown. Our model curves follow the COMSOL curves well. The maximum temperature difference between our model and



(a) maximum internal heat generation

(b) minimum internal heat generation

Figure 5.3: Transient thermal behavior of an isothermal slice of silica glass as per COMSOL, the analytical model and SPICE simulations given the parameters in Table 5.1. The SPICE simulations follow the analytical model well. The discrepancy between the CFD results and the analytical model is no larger than 0.5°C.

the COMSOL results is less than 0.5°C. Though, the COMSOL transient data seems to have a slightly steeper slope than our theoretical model. This could be originating from the fact that the COMSOL object is not 100 % isothermal. Besides the small temperature discrepancy between our analytical model and the COMSOL data, the model is deemed an appropriate solution for passive cooling with internal heat generation.

Figure 5.3 also shows the results of a simulated electrical circuit in SPICE, modeling the same cooling problem, based on the current/thermal equivalence[2]. The SPICE simulations follow the analytical results systematically well. The maximum difference is around 25 mK, which is negligibly small.

5.3.5 Thermal Runaway

Thermal runaway of a system occurs when the system's internal heat generation becomes larger than its total heat dissipation capabilities. At this point the system enters an irreversible process where the heat generation engages in a feedback loop that leads to a self heating process. This temperature feedback loop continues until the system destroys itself, something that should, for obvious reasons, be avoided. Systems with temperature-dependent internal heat generation, such as microprocessors, are most prone to thermal runaway. For microprocessors, the temperature-dependent leakage currents contribute most to the effect of thermal runaway. During a thermal runaway the power consumption, and hence also the energy consumption, increases as the temperature inflates. Liao *et al.* [69] formulated mathematically the conditions for a thermal runaway:

1. $\mathrm{d}T/\mathrm{d}t > 0$: the temperature must be increasing;

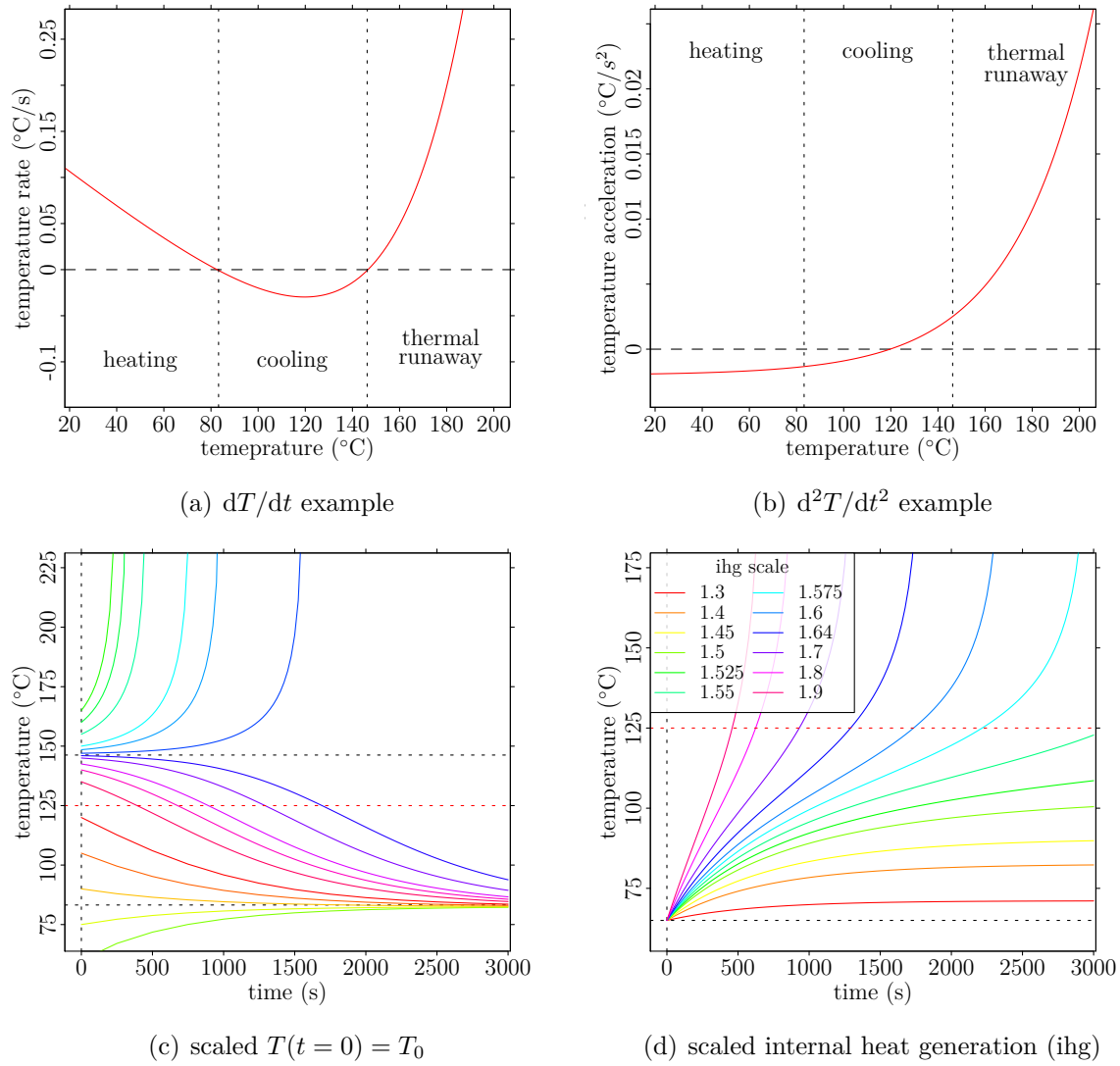
2. $\mathrm{d}^2T/\mathrm{d}t^2 > 0$: the increment of internal heat generation is larger than the increment in system's power heat removal capabilities.

[2]The source code for the SPICE simulation is given in Appendix B.1.3.

When both criteria are met, the system falls back to an infinite heating loop. Thus under no circumstances should the system surpass this critical temperature point. IC designers usually state a maximum operation temperature for ICs that should not be surpassed. For example, the OMAP 4470 dual-core applications processor's datasheet states that the absolute maximum junction temperature has to remain below 125°C and the maximum average junction temperature be below 110°C. Such temperature limits lay below the point of thermal runaway by a safe margin. This safety temperature limit does not only prevent thermal runaways, but also prevents possible irreversible hardware damage to the IC due to excessive temperature, e.g., dielectric breakdown, or creep in the bonding materials [17].

Figure 5.4 provides visual aids for understanding thermal runaway. In Figure 5.4(a) the first derivative of temperature is shown, which is essentially the graphical representation of Equation 5.5. Three distinct parts are identified in the graph: $\mathrm{d}T/\mathrm{d}t > 0$ on the left for heating, $\mathrm{d}T/\mathrm{d}t < 0$ in the middle for cooling, and on the right again $\mathrm{d}T/\mathrm{d}t > 0$ for thermal runaway. The system is in a state of thermal runaway only in the right most part as there the second derivative is positive, see Figure 5.4(b). Figure 5.4(c) shows the effect of thermal runaway for various initial temperatures $T(t = 0) = T_0$. The equilibrium temperature of the system here is 83.21°C; all curves for T_0 below the point of thermal runaway T_{tr} asymptotically approach this temperature. All curves above the point of thermal runaway, i.e., T_{tr}=146.25°C, accelerate into infinity. Note that the system is very sensitive around the point of thermal runaway. This is better visible in Figure 5.4(d); here the internal heat generation is scaled for a fixed temperature T_0 at $t = 0$. In this case, when the temperature doesn't approach a horizontal asymptote it is deemed potentially thermally unstable, and prone to thermal runaway. The point where the system becomes potentially thermally unstable in this example is around a *ihg scale* of 1.51. Here, the ihg scaling factor could reflect the microprocessor's clock frequency of Procedure 3.2. The internal heat generation increases for higher clock frequencies, see Chapter 3. The maximum internal heat generation of this power model is at 1.6 GHz. This means that this system has the potential to become unstable when residing too long in maximum power mode. In Figure 5.4(d), under those conditions, the temperature starts to increase exponentially around 1500 s or 25 min. Very small steps in frequency around 1.51 GHz yield a large difference in thermal runaway behavior. In all cases the temperature tends to infinity, but $\mathrm{d}T/\mathrm{d}t$ is very sensitive for small frequency steps. The power model used to generate this example, and Figure 5.5, was shown to be representative between 25°C and 85°C in Section 3.6. Here the model is extrapolated up to 300°C; the resulting graphs should thus be taken with a grain of salt.

Let us analyze the point of thermal runaway for a system as was described, for the COMSOL simulations, in Table 5.1, for passively cooled and actively cooled systems, respectively. Figure 5.5 provides visual aids to understand thermal runaway. Figure 5.5(a) shows the point of thermal runaway T_{tr} for a passively cooled system for different levels of internal heat generation. T_{tr} drops for smaller surface areas. The smaller the surface area, the less effectively a system can dissipate heat to its environment; therefore it is understandable that T_{tr} decreases for smaller surface areas. The temperature difference between the smallest and largest surface areas is between 150°C and 175°C, respectively, depending on the magnitude of the internal heat generation. For each increased clock frequency step, or about 15 % more internal heat generation, T_{tr} drops 8°C. The ratio of T_{tr} for an actively cooled system over a passively cooled system with equal equilibrium temperature is depicted in Figure 5.5(b). The average discrepancy between the T_{tr} for



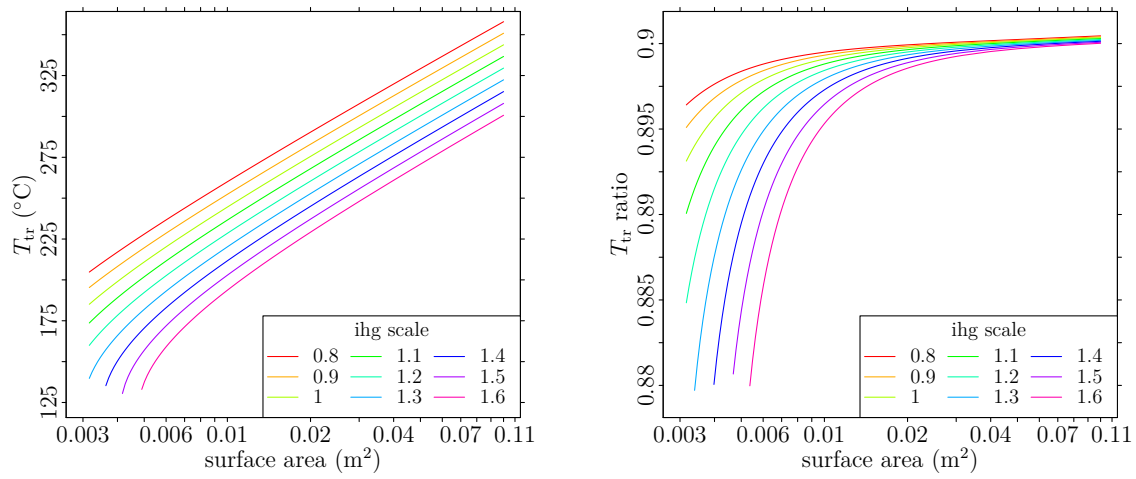
(a) $\mathrm{d}T/\mathrm{d}t$ example

(b) $\mathrm{d}^2T/\mathrm{d}t^2$ example

(c) scaled $T(t=0)=T_0$

(d) scaled internal heat generation (ihg)

Figure 5.4: Thermal runaway exemplified for a system subject to radiative, convective cooling and internal heat generation: h=6 W/(m^2·K), T_a=20°C, S=0.01 m^2, C=30 J/K, and internal heat generation (ihg) as per Procedure 3.2 for f=1.4, and c=4. For a fixed ihg in Figure (c), the equilibrium temperature is T_e=83.21°C, and the point of thermal runaway is at T_{tr}=146.25°C. Figures (a) and (b) show the conditions for a thermal runaway: $\mathrm{d}T/\mathrm{d}t > 0$ and $\mathrm{d}^2T/\mathrm{d}t^2 > 0$. Figure (c) shows the transient temperature for various values of $T(0)$=T_0. Figure (d) scales the internal heat generation for a fixed T_0=65°C. A red horizontal dotted line is drawn at T=125°C indicating a typical maximum junction temperature for applications microprocessors.



(a) T_{tr} for passive cooling

(b) ratio T_{tr} of passive and active cooling

Figure 5.5: Point of thermal runaway T_{tr} for a passively cooled system (a) with properties as listed in Table 5.1, with different levels of internal heat generation (ihg) and surface areas. (b) shows the ratio of T_{tr} for an actively cooled system over a passively cooled system with the same equilibrium temperature is shown on the left.

active and passively cooled systems is around 10 %. For smaller surface areas, the ratio seems to drop fast, the latter for the same reasons as will be explained in Section 5.4.

5.3.6 Approximate Solutions to $f(t) = T$

The exact solution for the passive heat Equation 5.17 is of the form $f(T) = t$. Ideally, for practical motivations, the inverse $f(t) = T$ is likely to be known. For example, this may be convenient for the equation to be used by DTMs, TMUs, and in proportional-integral-derivative (PID) controller systems. Calculating the inverse of Equation 5.17 is, however, a challenging endeavor. Therefore, some effective approximations are assumed to obtain an invertible heat equation.

Finding a useful expression $f(t) = T$ requires isolating T in Equation 5.17. Mainly the presence of the `arctan` throws a monkey wrench in the mathematical derivation. Linearization or differential approximation will not provide any help as the derivative within the pertinent temperature range, i.e., between 20°C and 55°C, is far from being constant. Converting the arctan into a logarithm introduces imaginary numbers, yet, applying complex exponentiation rules will not get rid of the `arctan` either. The `arctan` keeps recurring further on in the derivation. So a different approach is to be taken to come to a solution for $f(t) = T$, mainly by assuming some simplifications. In the sequel the following chosen approximations are analyzed: quadratic approximation of the radiation, and binomial expansion of the first and second-order, also referred to as the O'Sullivan approximations. Figure 5.6 shows an illustrative example of the exact solution and the approximations. The approximations were computed with the exact same values as the exact solution. It can already be observed that the approximations are not perfectly accurate. Section 5.3.7 is devoted to the performance comparison of the approximations and the exact solution.

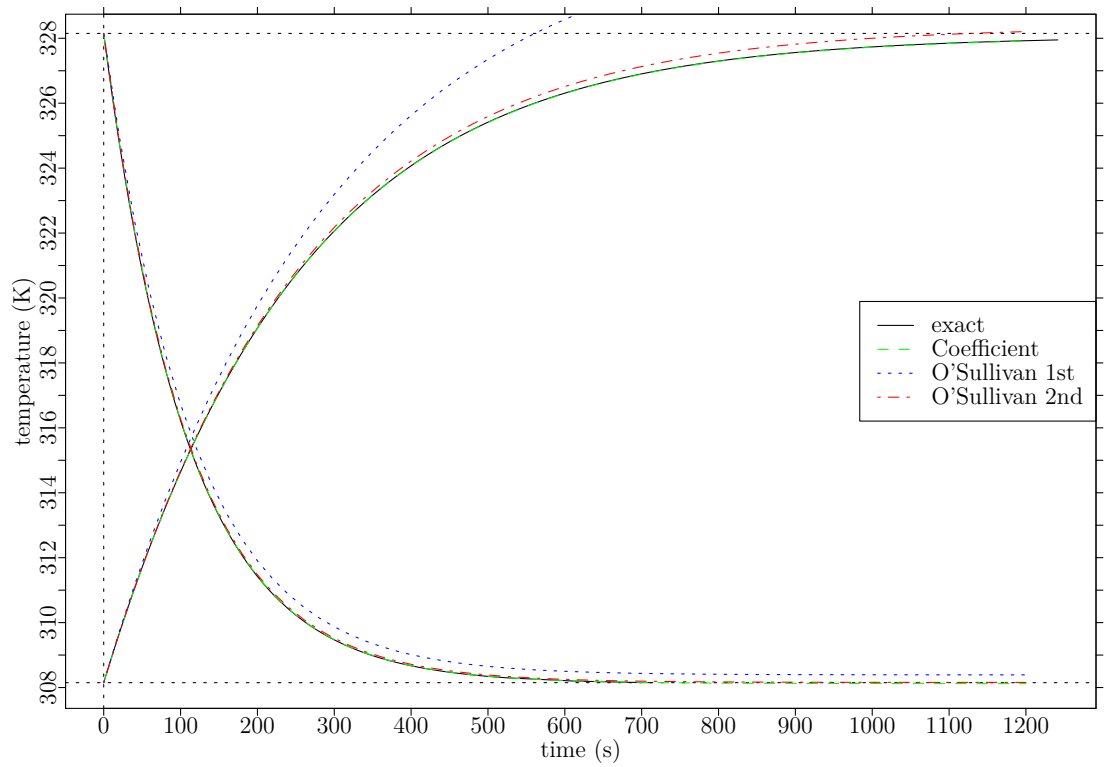


Figure 5.6: Illustration of the discussed approximate solutions to $f(t) = T$. Both the cooling and heating processes of a slice of silica glass are depicted: S=0.1m^2, $T_{\max}$=55°C, $T_{\min}$=35°C. The same values were used to generate the exact solution and the approximations. Note that the approximations are not perfectly following the exact solution.

The Coefficient Approximation

Stefan-Boltzmann's law of radiation states that the energy emitted by radiation is proportional to T^4 (Stefan-Boltzmann's law: Equation 2.27). Because of this term the polynomial of Equation 5.7 is of the fourth-order. More specifically, it are the two imaginary roots of the fourth-order polynomial that introduce the `arctan` in Equation 5.17. If T^4 were to be approximated with a second-order polynomial then the heat equation is asserted to have two real roots, and the `arctan` would disappear. Hence, isolating T will be less difficult. The quadratic approximation

$$\begin{aligned} T^4 &= q_0 + q_1 T + q_2 T^2 \\ &= 29700057265 - 251483462\,T + 598262\,T^2, \end{aligned} \tag{5.24}$$

where T is expressed in *Kelvin*, introduces an error for 20°C $< T <$ 65°C up to 0.072 %, which is very acceptable. Then the quadratic approximation to Equation 5.8 would be equal to solving

$$\frac{dT}{dt} = \kappa_2 T^2 + \kappa_1 T + \kappa_0. \tag{5.25}$$

The solution to this equation, assuming two real roots $\omega_* = (-\kappa_1 \pm \sqrt{\kappa_1^2 - 4\kappa_2\kappa_0})/(2\kappa_2)$ and that $\kappa_2 < 0$:

$$t = -\frac{1}{\kappa_2}\left(A \ln|T - \omega_1| + B \ln|T - \omega_2| + c_o\right), \tag{5.26}$$

where $A = 1/(\omega_2 - \omega_1)$ and $B = -A$. Now T can be isolated as follows:

$$\begin{aligned} t + \frac{c_o}{\kappa_2} &= -\frac{A}{\kappa_2}\left(\ln|T-\omega_1| - \ln|T-\omega_2|\right) \\ -\frac{\kappa_2 t + c_o}{A} &= \ln\left(\frac{|T-\omega_1|}{|T-\omega_2|}\right) \\ c_o e^{-\frac{\kappa_2}{A}t} &= \frac{|T-\omega_1|}{|T-\omega_2|}. \end{aligned} \tag{5.27}$$

Let's define ω_1 and ω_2 such that $\omega_1 < \omega_2$. In this case, T is always larger than ω_1, which will be approximately around 100 K for problems around room temperature. Thus, as the temperature range $0°\text{C} < T < 100°\text{C}$ is of primal concern, T will always be larger than ω_1. Hence it can be assumed that $T - \omega_1 > 0$. The presence of $|T - \omega_2|$ in Equation 5.27 forces us to distinguish two cases, i.e., where $T > \omega_2$ and $T < \omega_2$. This corresponds either to the heating or the cooling process, respectively. Bear in mind that ω_2 is also the equilibrium temperature T_e of the system. Thus if $T = \omega_2 = T_e$, then the temperature of the system is stable over time. Let's look at the cooling process first. If $T > \omega_2$, then

$$\begin{aligned} T - \omega_1 &= (T-\omega_2)c_o e^{-\frac{\kappa_2}{A}t} \\ T - c_o e^{-\frac{\kappa_2}{A}t}T &= \omega_1 - \omega_2 c_o e^{-\frac{\kappa_2}{A}t} \\ T &= \frac{\omega_1 - \omega_2 c_o e^{-\frac{\kappa_2}{A}t}}{1 - c_o e^{-\frac{\kappa_2}{A}t}} \\ T &= \frac{-\kappa_1 + \sqrt{\kappa_1^2 - 4\kappa_2\kappa_0} + (\kappa_1 + \sqrt{\kappa_1^2 - 4\kappa_2\kappa_0})c_o e^{-\frac{\kappa_2}{A}t}}{2\kappa_2(1 - c_o e^{-\frac{\kappa_2}{A}t})} \\ T &= \frac{\sqrt{\kappa_1^2 - 4\kappa_2\kappa_0}(1 + c_o e^{-\frac{\kappa_2}{A}t})}{2\kappa_2(1 - c_o e^{-\frac{\kappa_2}{A}t})} - \frac{\kappa_1}{2\kappa_2} \end{aligned} \tag{5.28}$$

and accordingly, for $T < \omega_2$, or the heating process, one gets:

$$T = \frac{\sqrt{\kappa_1^2 - 4\kappa_2\kappa_0}(1 - c_o e^{-\frac{\kappa_2}{A}t})}{2\kappa_2(1 + c_o e^{-\frac{\kappa_2}{A}t})} - \frac{\kappa_1}{2\kappa_2}, \tag{5.29}$$

where c_o is an integration constant used to meet the initial condition $f(0) = T_0$:

$$c_o = \frac{|T_0 - \omega_1|}{|T_0 - \omega_2|} = \frac{|2\kappa_2 T_0 + \kappa_1 - \sqrt{\kappa_1^2 - 4\kappa_2\kappa_0}|}{|2\kappa_2 T_0 + \kappa_1 + \sqrt{\kappa_1^2 - 4\kappa_2\kappa_0}|}.$$

The equilibrium temperature T_e is defined by the positive real root ω_2. A prototype implementation of Equation 5.29 is provided in Appendix B.1.2.

In the above expound, the coefficients q_* in Equation 5.24 are fixed. These values were chosen to fit best in a certain temperature range. To be more universally applicable, however, the coefficients could be generated dynamically such that they are optimally tailored to the temperature range of concern.

The First Order O'Sullivan Approximation

O'Sullivan [82] presented an approximation for a cooling law including convection and radiation, but without the presence of internal heat generation. Though, his approximation

can be extended with internal heat generation. An alternative formulation of the internal heat generation is used such that a variable substitution can be applied more easily later on: $H(T) = \eta_1 T + \eta_0 = \eta_1(T - T_a) + \eta_1 T_a + \eta_0$. The initial definition of the passive heat Equation 5.5 then becomes

$$\begin{aligned} -C\frac{\mathrm{d}T}{\mathrm{d}t} &= \epsilon\sigma S(T^4 - T_a^4) + hS(T - T_a) - (\eta_1(T - T_a) + \eta_1 T_a + \eta_0) \\ &= \epsilon\sigma S(T^4 - T_a^4) + (hS - \eta_1)(T - T_a) - (\eta_1 T_a + \eta_0). \end{aligned}$$

Let's introduce the variable $\theta = T - T_a$:

$$-C\frac{\mathrm{d}\theta}{\mathrm{d}t} = \epsilon\sigma S((\theta + T_a)^4 - T_a^4) + (hS - \eta_1)\theta - (\eta_1 T_a + \eta_0).$$

Now, binomial expansion[3] to $(\theta - T_a)^4$ can be applied, whence:

$$\begin{aligned} -C\frac{\mathrm{d}\theta}{\mathrm{d}t} &= \epsilon\sigma S[(\theta^4 + 4T_a\theta^3 + 6T_a^2\theta^2 + 4T_a^3\theta + T_a^4) - T_a^4] + (hS - \eta_1)\theta - (\eta_1 T_a + \eta_0) \\ &= \epsilon\sigma S(\theta^4 + 4T_a\theta^3 + 6T_a^2\theta^2) + (hS - \eta_1 + 4\epsilon\sigma S T_a^3)\theta - (\eta_1 T_a + \eta_0) \\ &= k\theta^4 + l\theta^3 + m\theta^2 + n\theta + p, \end{aligned} \tag{5.30}$$

where the coefficients are as follows, with magnitude estimates for surface areas $\approx$1 dm^2:

$$\begin{aligned} k &= \epsilon\sigma S \quad (\sim 10^{-10}) \\ l &= 4\epsilon\sigma S T_a \quad (\sim 10^{-7}) \\ m &= 6\epsilon\sigma S T_a^2 \quad (\sim 10^{-5}) \\ n &= (hS - \eta_1 + 4\epsilon\sigma S T_a^3) \quad (\sim 0.01) \\ p &= -(\eta_1 T_a + \eta_0) \quad (\sim 1). \end{aligned}$$

Now, if $(T - T_a)$ is not too large the series on the right-hand of Equation 5.30 side converges reasonably fast [82]. Depending on the accuracy desired, the higher-orders may be dropped. Let's see how well the first-order and second-order approximations fit. As expected, the first-order approximation yields also an exponential:

$$\begin{aligned} -C\frac{\mathrm{d}\theta}{\mathrm{d}t} &= n\theta + p \\ \int \frac{1}{\theta + p/n}\mathrm{d}\theta &= -\frac{n}{C}\int \mathrm{d}t \\ \ln(\theta + p/n) &= -\frac{n}{C}t + c_o \\ \theta &= c_o e^{-\frac{n}{C}t} - \frac{p}{n}, \end{aligned}$$

where c_o is an integration constant such that $\theta(t = 0) = T_0$:

$$c_o = \theta_0 + \frac{p}{n} = (T_0 - T_a) + \frac{p}{n}.$$

And so the first-order O'Sullivan approximate solution is:

$$T = \left(T_0 - T_a + \frac{p}{n}\right) e^{-\frac{n}{C}t} - \frac{p}{n} + T_a. \tag{5.31}$$

[3] Binomial expansion of the 4th order: $(x + y)^4 = x^4 + 4x^3y + 6x^2y^2 + 4xy^3 + y^4$.

Surprisingly, this approximation is a pure exponential function. The equilibrium temperature T_e and the convection heat transfer coefficient h at T_e are given by:

$$\begin{aligned} T_e &= \frac{\eta_0 + \eta_1 T_a}{hS - \eta_1 + 4\epsilon\sigma S T_a^3} + T_a \\ h &= \frac{\eta_0 + \eta_1 (T_e - T_a)}{S(T_e - T_a)} - 4\epsilon\sigma T_a^3 \end{aligned}$$

A prototype implementation of Equation 5.31 is provided in Appendix B.1.2.

The Second Order O'Sullivan Approximation

The second-order O'Sullivan approximation is a bit more complex compared to the first-order O'Sullivan approximation. Moreover, the derivation looks also significantly different to the original derivation of O'Sullivan [82], given the presence of the constant term p in Equation 5.30. The second-order O'Sullivan approximation is similar to the quadratic approximation in the sense that solving

$$-C\frac{\mathrm{d}\theta}{\mathrm{d}t} = m\theta^2 + n\theta + p \tag{5.32}$$

is similar to solving Equation 5.25. Thus the solution for the second-order O'Sullivan approximation will be the same as for the quadratic approximation, besides the constants. It can thus be stated that the second-order O'Sullivan approximation is given by:

$$T = \frac{\omega_1 \pm \omega_2 c_o e^{\frac{m}{AC}t}}{1 \pm c_o e^{\frac{m}{AC}t}} + T_a, \tag{5.33}$$

where "$\pm$" becomes "$+$" for $T > T_e$, and "$-$" for $T < Te$. ω_* is given by:

$$\omega_1 = \frac{-\sqrt{n^2 - 4pm} - n}{2m} \quad \text{and} \quad \omega_2 = \frac{\sqrt{n^2 - 4pm} - n}{2m}.$$

The constant A and integration constant c_o such that $\theta(0) = \theta_0$ are:

$$A = -\frac{1}{\omega_2 - \omega_1} \quad \text{and} \quad c_o = \frac{|\theta_0 - \omega_1|}{|\theta_0 - \omega_2|}, \tag{5.34}$$

where $\theta_0 = T_0 - T_a$. The equilibrium temperature T_e is defined by $\omega_2 + T_a$.

Alternative Second Order O'Sullivan Approximation

Alternatively, the second-order O'Sullivan approximation can also be computed as follows, yielding a solution with a hyperbolic tangent. Rearranging Equation 5.32 reveals an alternative formulation:

$$\begin{aligned} m\theta^2 + n\theta + p &= m\left(\theta^2 + 2\frac{n}{2m}\theta + \frac{n^2}{4m^2} - \frac{n^2}{4m^2} + \frac{p}{m}\right) \\ &= m\left(\left(\theta + \frac{n}{2m}\right)^2 - \left(\frac{n^2 - 4mp}{4m^2}\right)\right), \end{aligned} \tag{5.35}$$

which when integrating its inverse $(1/f(\theta))$, yields an inverse hyperbolic tangent, knowing that strictly $\{m, n\} \in \mathbb{R}_0^+$ and $p \in \mathbb{R}_0^-$:

$$\int \frac{1}{(\theta+a)^2 - b} \mathrm{d}\theta = -\frac{1}{\sqrt{b}} \operatorname{atanh}\left(\frac{\theta+a}{\sqrt{b}}\right).$$

Though, the hyperbolic arctangent comes with the constraint that the function is only defined within $\left|\frac{(\theta+a)}{\sqrt{b}}\right| < 1$. Thus translating this constraint to Equation 5.35 yields:

$$\begin{aligned}\left(\theta + \frac{n}{2m}\right)^2 - \frac{n^2-4mp}{4m^2} &< 0 \\ T &< \sqrt{\frac{n^2-4mp}{4m^2}} - \frac{n}{2m} + T_a = T_e,\end{aligned}$$

which means concretely that, for the hyperbolic tangent to be satisfied, $T < T_e$. This corresponds to the heating process. For the cooling process, the hyperbolic tangent is to be replaced with the hyperbolic cotangent `coth` in the sequel. The hyperbolic cotangent is defined for $\left|\frac{(\theta+a)}{\sqrt{b}}\right| > 1$. Let's express the alternative solution to the second-order O'Sullivan approximation for the heating process:

$$\begin{aligned}\int \frac{1}{m\theta^2 + n\theta + p} \mathrm{d}\theta &= -\frac{1}{C}\int \mathrm{d}t \\ \frac{1}{m}\int \frac{1}{\left(\theta + \frac{n}{2m}\right)^2 - \left(\frac{n^2-4mp}{4m^2}\right)} \mathrm{d}\theta &= -\frac{1}{C}\int \mathrm{d}t \\ -\frac{1}{m\sqrt{\frac{n^2-4mp}{4m^2}}} \operatorname{atanh}\left(\frac{\theta + \frac{n}{2m}}{\sqrt{\frac{n^2-4mp}{4m^2}}}\right) &= -\frac{1}{C}t + c_o,\end{aligned}$$

where `atanh` is the *inverse hyperbolic tangent*. Let's define the variable $w = \sqrt{n^2 - 4mp}$:

$$\begin{aligned}\operatorname{atanh}\left(\frac{2m\theta + n}{w}\right) &= \frac{w}{2C}t + c_o \\ \theta &= \frac{w}{2m}\tanh\left(\frac{w}{2C}t + c_o\right) - \frac{n}{2m} \\ T &= \frac{w}{2m}\tanh\left(\frac{w}{2C}t + c_o\right) - \frac{n}{2m} + T_a, \end{aligned} \tag{5.36}$$

where c_o an integration constant that must satisfy the initial condition $\theta(t=0) = T_0 - T_a$:

$$c_o = \operatorname{atanh}\left(\frac{2m(T_0 - T_a) + n}{\sqrt{n^2 - 4mp}}\right).$$

Similarly, for the cooling process:

$$T = \frac{w}{2m}\coth\left(\frac{w}{2C}t + c_o\right) - \frac{n}{2m} + T_a, \quad \text{and} \quad c_o = \operatorname{acoth}\left(\frac{2m(T_0 - T_a) + n}{\sqrt{n^2 - 4mp}}\right), \tag{5.37}$$

where `coth` is the *hyperbolic cotangent*, and `acoth` is the *inverse hyperbolic cotangent*. The equilibrium temperature T_e of the system is defined for $t \to \infty$:

$$T_e = \frac{\sqrt{n^2 - 4mp} - n}{2m} + T_a. \tag{5.38}$$

Table 5.2: Recapitulation of approximations to the exact inverse cooling law $f(t) = T$. The quadratic coefficient and the first and second-order O'Sullivan approximations are shown together with their equilibrium temperature.

APPROXIMATION	$T(t)$	$T(\infty) = T_e$
Coefficient	$T = \frac{\omega_1 \pm \omega_2 c_o e^{-\frac{\kappa_2}{A}t}}{1 \pm c_o e^{-\frac{\kappa_2}{A}t}}$	$T_e = \frac{\sqrt{\kappa_1^2 - 4\kappa_2\kappa_0} - \kappa_1}{2\kappa_2}$
O'Sullivan 1st	$T = \left(T_0 - T_a + \frac{p}{n}\right) e^{-\frac{n}{C}t} - \frac{p}{n} + T_a$	$T_e = -\frac{p}{n} + T_a$
O'Sullivan 2nd	$T = \frac{w}{2m} \tanh\left(\frac{w}{2m}t + c_0\right) - \frac{n}{2m} + T_a$	$T_e = \frac{\sqrt{n^2 - 4mp} - n}{2m} + T_a$

A prototype implementation of Equations 5.36 and 5.37 is provided in Appendix B.1.2

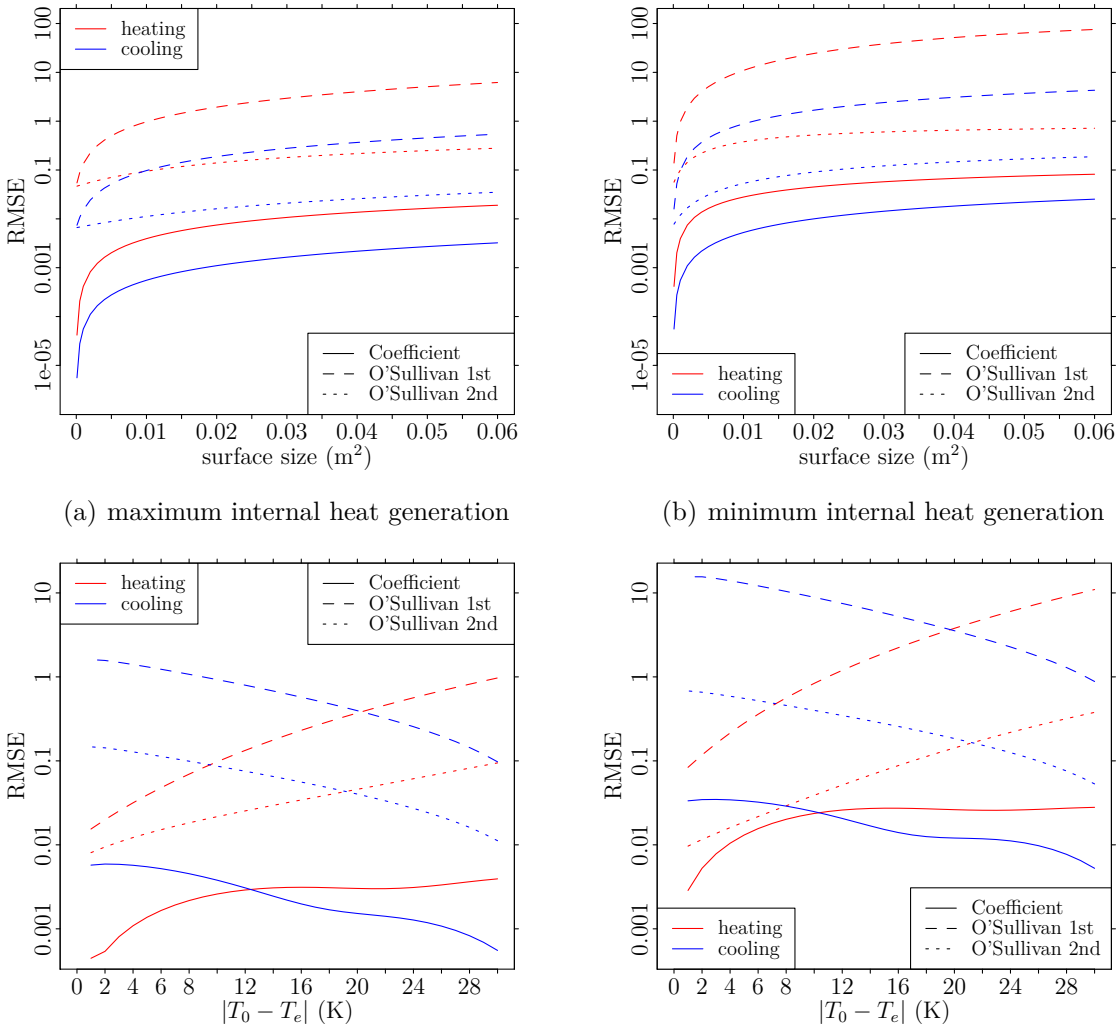
Surely, the solution of the alternative second-order O'Sullivan approximation in Equation 5.36 must be equal to the exponential-based solution of Equation 5.33. This can be shown by replacing the `tanh` by its exponential representation and expressing the `atanh` by its logarithmic representation. The resulting proof is given in Appendix A.1.2.

5.3.7 Battle of the Approximate Solutions

Let's analyze the accuracy of the approximations of Section 5.3.6. The measure of accuracy is defined as the root-mean-square error (RMSE)[4] between the exact cooling solution and an approximate solution. The number of samples over which the RMSE is computed was set to 500 and equally spaced between $0 < t < f(0.99 \cdot T_e)$. The exact cooling law and its approximations are generated with the same constants as the COMSOL simulation in the previous section; these are listed in Table 5.1. The accuracy for changing surface area S is investigated, together with internal heat generation (ihg), equilibrium temperature T_e, and for the cooling and warming process separately. T_0=25°C is set for the heating process and T_0=55°C for the cooling process. The equilibrium temperature T_e is scaled between 25°C and 55°C. The convective heat transfer coefficient is computed accordingly to attain the respective equilibrium temperatures based on Equations 5.4 and 5.38. The variables generated for the exact cooling law are then used to compute the approximations.

Table 5.2 shows an overview of the three different approximations considered. Figure 5.7 shows the RMSE of the approximations for different surface areas, internal heat generation and equilibrium temperature settings. From all graphs the quadratic approximation is clearly performing best. Also, the second-order O'Sullivan approximation is considerably better than the first-order O'Sullivan approximation. However, for very small surface areas the errors in all approximations are acceptable. Interestingly, the first-order O'Sullivan approximation does well for small surface areas, because the radiative part in the heat equation becomes negligible for smaller surface areas, and so the passive heat equation tends towards an exponential cooling law. Consequently the first-order O'Sullivan approximation, being an exponential function, is able to approximate the accurate cooling law well for very small surface areas: $S < 0.005\,\mathrm{m}^2$.

[4]The *root-mean-square error* is defined in Appendix A.3.



(a) maximum internal heat generation

(b) minimum internal heat generation

(c) maximum internal heat generation

(d) minimum internal heat generation

Figure 5.7: Root mean-square error (RMSE) between the exact cooling law and the approximations. On top (a,b) the surface area of the object under study is a variable (T_e=55°C), whereas the equilibrium temperature T_e is variable in the bottom graphs (S=0.01m^2) (c,d). The *Quadratic Approximation* seems to outperform the other approximations. The *Second-order O'Sullivan approximation* is performing acceptably well for small values of $|T - T_a|$.

The errors for small internal heat generation seem to be systematically larger than the errors for the maximum internal heat generation case. The same observation can be made for the heating and cooling processes. The heating approximation seems to be more erroneous than the cooling process.

For variable equilibrium temperatures it is seen that, the higher the temperature, the error increases for the heating process and descends for the cooling process. In the derivation of the O'Sullivan approximations it was assumed that $T - T_a$ remains relatively small. This implies that the larger T departs from T_a the more imprecise the approximation becomes. For the cooling process T_0=55°C and the equilibrium temperature T_e was scaled between 25°C and 55°C. Similarly, for the heating process T_0 was set to 55°C and T_e was scaled between 25°C and 55°C. In both cases T_a was fixed to 20°C. Thus as the cooling process approaches T_a for increasing $|T_0 - T_e|$, $T - T_a$ becomes smaller, and hence also the error between the O'Sullivan approximations and the exact cooling law. The reverse observation is also valid for the heating process; RMSE becomes larger for larger values of $T - T_a$. The error properties in the case of the quadratic approximation depends on the fit of the second-order polynomial on the (quadratic) radiation function.

Let's define an additional measure of accuracy. Whereas the RMSE is perhaps hard to mentally represent, the temperature error Θ is introduced to be more tangible. The temperature error Θ is defined as the distance between the exact solution ψ and its approximations ϕ evaluated at their respective equilibrium temperature:

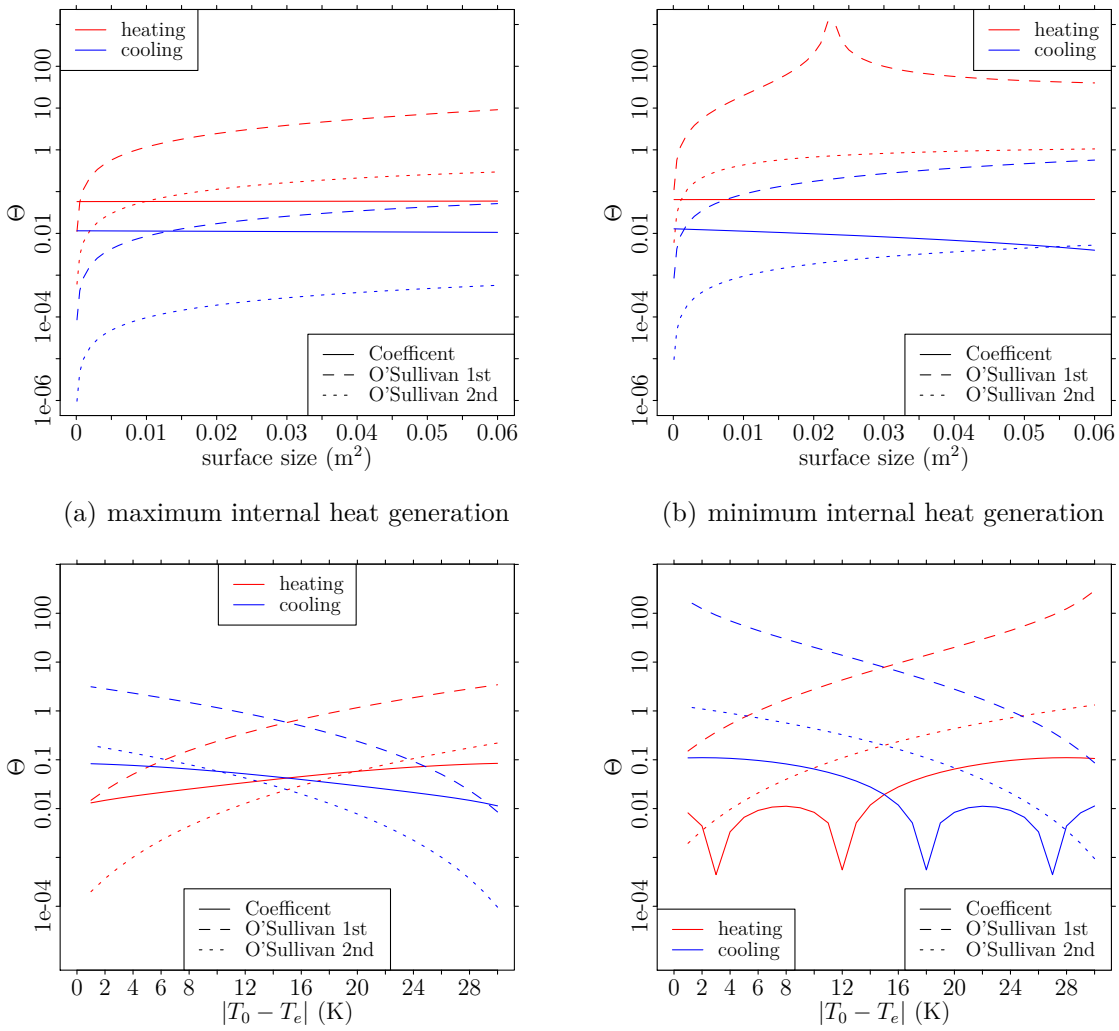
$$\Theta = |\psi(t = \infty) - \phi(t = \infty)|. \tag{5.39}$$

Figure 5.8 shows Θ for variable surface area S and equilibrium temperature T_e, with the same conditions as before. The quadratic approximation is no longer the overall winner based on the temperature difference at equilibrium temperature. In Figure 5.8(a) and 5.8(b), especially for a small surface area, the cooling of the O'Sullivan approximations perform well. For the heating process, the errors become quite large, even to such an extent that the first O'Sullivan approximation should really be avoided. In Figure 5.8(c) and 5.8(d), as $|T - T_a|$ becomes smaller, the O'Sullivan approximations perform increasingly better, as expected. Though, as before, the first-order O'Sullivan approximation should be used with caution, notably when the internal heat generation is small. The second-order O'Sullivan approximation and the quadratic approximation stay almost always below 1°C temperature error.

Overall, it is not advised to use the first-order O'Sullivan approximation, unless the surface area is really small. The second-order O'Sullivan approximation can be used but with caution. The equilibrium temperature should not depart too much from the ambient temperature T_a; $T - T_a < 15$°C seems acceptable. The use of the quadratic approximation is, however, recommended, even though the solution isn't much elegant when the large polynomial coefficients are introduced.

5.3.8 Impulse Responses

The solutions $f(t) = T$ from the previous sections represent the temperature response of the system to a (Heaviside) step function , i.e., the *step response*. The Heaviside function can practically be thought of as a microprocessor's transition from one regime to another, having different ξ values. To obtain the temperature response for an arbitrary power input, the convolution of the impulse response of $f(t)$ with the arbitrary power trace can be computed. The impulse response $\delta(t)$ of $f(t)$ is obtained by the first derivative of the



(a) maximum internal heat generation

(b) minimum internal heat generation

(c) maximum internal heat generation

(d) minimum internal heat generation

Figure 5.8: Temperature difference at equilibrium temperature between the exact heat equation and its approximations. On top (a,b) the surface area of the object under study is a variable (T_e=55°C), whereas the equilibrium temperature T_e is variable in the bottom graphs (S=0.01m^2) (c,d). The *quadratic approximation* seems to the best choice on the average. The *second-order O'Sullivan approximation* is performing acceptably as well.

step response. Then, the convolution of the impulse response with an arbitrary power trace is given by

$$(P * \delta)(t) = \int_{-\infty}^{\infty} P(\tau)\, \delta(t-\tau)\; d\tau. \tag{5.40}$$

As the exact inverse of $f(T) = t$ is not know, for reasons explained in the previous sections, the impulse response is obtained from its approximations. Let us state the impulse response $\delta(t)$ of the quadratic approximation and the tangent expression of the second-order O'Sullivan approximation:

- Quadratic Approximation:

$$\delta(t) = \frac{-\omega_2 c_0 \frac{\kappa_2}{A} e^{-\frac{\kappa_2}{A}t}}{1 - c_0 e^{-\frac{\kappa_2}{A}t}} - \frac{c\frac{\kappa_2}{A} e^{-\frac{\kappa_2}{A}t}(\omega_2 c_0 e^{-\frac{\kappa_2}{A}t} + \omega_1)}{(1 - c_0 e^{-\frac{\kappa_2}{A}t})^2};$$

- Second-order O'Sullivan Approximation:

$$\delta(t) = \frac{w}{2m}\left(1 - \coth^2\left(\frac{w}{2C}t + c_o\right)\right).$$

Computing the convolution of the power with any of these impulse responses may incur considerable overhead. For TMUs or DTMs in low-power systems this may not be a tractable approach to energy optimization.

5.4 Passive and Active Cooling Law Comparison

Given the intrinsic complexity of the (inverse) function describing passive cooling compared to the rather straightforward exponential specification of other cooling modes, it is worth investigating in which cases dealing with it in practice is necessary. A series of simulations are run to understand under what circumstances the passive and active cooling law differ from each other. The main difference between the active and the passive cooling law is the presence of the radiative heat transfer mode. Thus, if the radiative heat transfer is negligible compared to the convective heat transfer, the passive cooling law will approach an exponential cooling law. Such situations are explored based on concrete microprocessor use cases. Let's recall the equations governing the cooling process of a passively and actively cooled isothermal body with internal heat generation:

$$\text{active cooling:} \quad C\frac{dT}{dt} = h_{\text{ac}} S(T_a - T) + H(T) \tag{5.1}$$

$$\text{passive cooling:} \quad C\frac{dT}{dt} = h_{\text{pc}} S(T_a - T) + H(T) + \epsilon\sigma S(T_a^4 - T^4), \tag{5.5}$$

and their respective convective heat transfer coefficients h, at equilibrium temperature:

$$\text{active cooling:} \quad h_{\text{ac}} = \frac{H(T_e)}{S(T_e - T_a)} \tag{5.42a}$$

$$\text{passive cooling:} \quad h_{\text{pc}} = \frac{\epsilon\sigma S(T_a^4 - T_e^4) + H(T_e)}{S(T_e - T_a)}, \tag{5.42b}$$

where $H(T)$ is a function defining the temperature-dependent internal heat generation of the body. In Section 3.6, it was shown that $H(T)$ is well described by an exponential

Table 5.3: Parameters assumed for the comparison of the active and passive cooling laws. Two internal heat generation settings are used defined by the first and second values of α, β and γ.

Constants			Variables		
symbol	value	dim.	symbol	value	dim.
σ	5.670×10^{-8}	$W/(m^2{\cdot}K^4)$	S	$[0,6]\times 10^{-3}$	m^2
ϵ	0.94	-	C	$S \times D \times c_v$	J/K
T_a	20	°C	T	$[25,85]$	°C
D	2	mm	h	(see Equation 5.42)	$W/(m^2{\cdot}K)$
c_v	1708800	$J/(m^3{\cdot}\ K)$	α	$\{0.396, 4.030\}$	W
			β	$\{29.015, 32.010\}$	-
			γ	$\{82.738, 149.797\}$	-

equation. Even more, within the temperature range $25°C < T < 55°C$, the exponential can be approximated adequately with a linear or quadratic polynomial. Though, for a more extended temperature ranges, $25°C < T < 85°C$, an exponential function is advised.

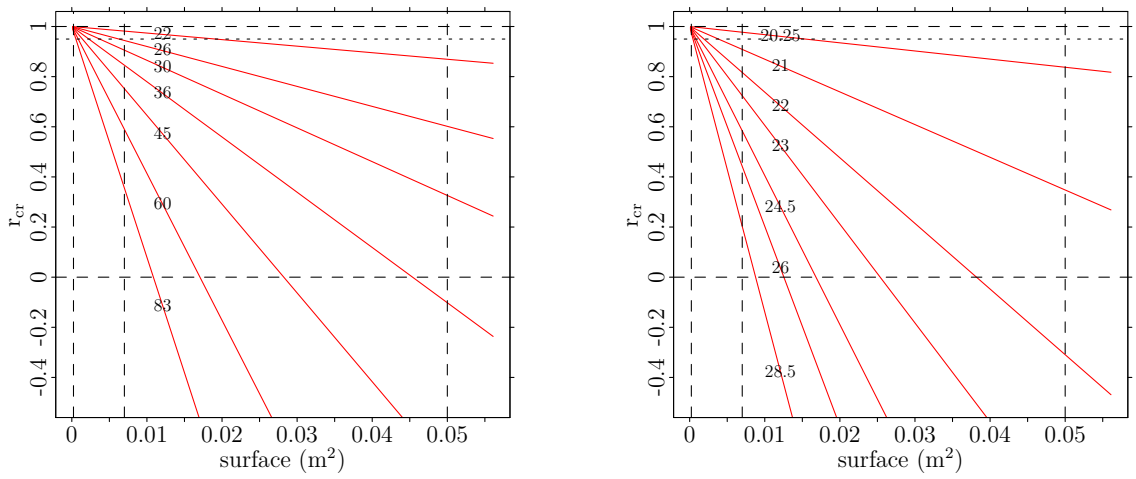
The active and passive cooling of a microprocessor are compared in the context of an embedded system, i.e., a low-power microprocessor subject to internal heating generation and cooling. In order to do so, a simplified microprocessor model was assumed: an isothermal volume, with the physical properties of silicon-dioxide (SiO_2), and cooled via convection and radiation. Table 5.3 shows the values used in the simulations. The table lists the fixed variables: σ, ϵ, T_a and D. The emissivity of PVC for ϵ was chosen and was fixed T_a to be representative for room temperature. The height of our microprocessor D is characteristic for a modern SoC. The variables that may vary during the analysis are also listed. The impact of the surface area S over which the device cools via convection and radiation was studied. The minimum size was set to a square with a side of 1 cm. This is representative for a small SoC; for example, the Samsung Exynos 5410 SoC has a side length of 1.6 cm. The maximum surface area was set to 0.1 m^2, which is a representative area for a large tablet. The behavior of the system was analyzed within the temperature range $T \in [25, 85]$°C. Throughout the analysis the internal heat generation $H(T)$ was defined to be an exponential function $(\alpha + e^{(T-\gamma)/\beta})$; the coefficients are shown in Table 5.3 as pairs. The left values are for minimal internal heat generation, the right values for maximum internal heat generation. The values for α, β and γ were derived from the A15 power and temperature measurements of the Samsung Exynos 5 SoC as reported in Chapter 3. The system's behavior is studied when the A15 is running at full capacity, i.e., at 1.6 GHz, and when the A15 is running in lowest-power mode; at 800 MHz. The heat capacity of the body C is the product of its volume $S \times D$ and its specific heat capacity and density ($\approx 0.712 \cdot 2.4 \times 10^6$).

5.4.1 Convective Heat Transfer Coefficient Ratio

First, the ratio of the convective heat transfer coefficients of the passive and the active cooling cases is looked at. The temperature T_0 at t=0 is set to 25°C. Then the respective convective heat transfer coefficients are computed as per Equations 5.42 based on a series of equilibrium temperatures T_e. The ratio of the convective heat transfer coefficients r_{cr} is given by

$$r_{cr} = \frac{h_{pc}}{h_{ac}} = \frac{\epsilon\sigma S(T_a^4 - T_e^4) + H(T_e)}{H(T_e)}.$$

r_{cr} describes how much the active and passive cooling law will resemble. If $r_{cr} = 1$ there is no difference between the two cooling cases. The more r_{cr} tends to zero, the more the



(a) maximum internal heat generation

(b) minimum internal heat generation

Figure 5.9: Ratio between the convective heat transfer coefficients of active and passive cooling at a given equilibrium temperatures T_e (see curve labels in °C). In Figure (a) the internal heat generation is set to a maximum, while in (b) it is set to a minimum. The vertical dashed lines represent typical surfaces of a SoC ($\approx 2.5\,\mathrm{cm}^2$), a smartphone ($\approx 70\,\mathrm{cm}^2$) and a tablet ($\approx 5\,\mathrm{dm}^2$). A horizontal dotted reference line is drawn at $r_{\mathrm{cr}} = 0.95$.

two cooling laws will deviate in behavior.

Figure 5.9 shows the ratio of the convective heat transfer coefficient of the passive and the active cooling cases. Given that r_{cr} stays well above 0.95, it is observed that, for a small object, similar to SoCs (left most vertical dashed line), the difference between active and passive cooling will be very small for all equilibrium temperatures ranging between 20°C to 85°C. For a moderate surface size, e.g., the size of an average smartphone (middle vertical dashed line), the radiative cooling starts to become more prominent already for temperatures close to the ambient temperature T_a. For equilibrium temperatures more than about 5°C above T_a, signs of deviating behavior will become clearly visible. Large surface sizes and equilibrium temperatures close to T_a will yield a r_{cr} which is smaller than 0.95. This implies that the radiative cooling for large surfaces should be taken into account. As a general rule of thumb, the larger the equilibrium temperature and the cooling surface, the more behavioral differences between passive and active coolings will occur. So how large are the differences temperature-wise in particular?

When looking at the temperature differences between the passive and active cooling laws at specific points in time, we must differentiate between the cooling and heating processes. Convective heat transfer is proportional to the difference of the body temperature and the ambient temperature, and is therefore *independent* of the body's absolute temperature and environment. This results in a symmetry between the heating and the cooling process for the convective heat transfer. The radiative heat transfer, on the other hand, is *dependent* on the absolute value of the body and the environment. This is illustrated as follows for the convective and radiative heat transfer respectively:

$$\begin{aligned} |hS(T-(T-\Delta T))| &= |hS(T-(T+\Delta T))| \\ |\epsilon\sigma S(T^4-(T-\Delta T)^4)| &\neq |\epsilon\sigma S(T^4-(T+\Delta T)^4)|. \end{aligned} \tag{5.43}$$

As a consequence, due to the last inequality, the radiative heat transfer process will not be symmetric for the cooling and the heating processes. Moreover, when radiative heat

transfer is combined with convective heat transfer, the symmetry property of the heating and cooling processes will not hold either.

5.4.2 Temperature Lag

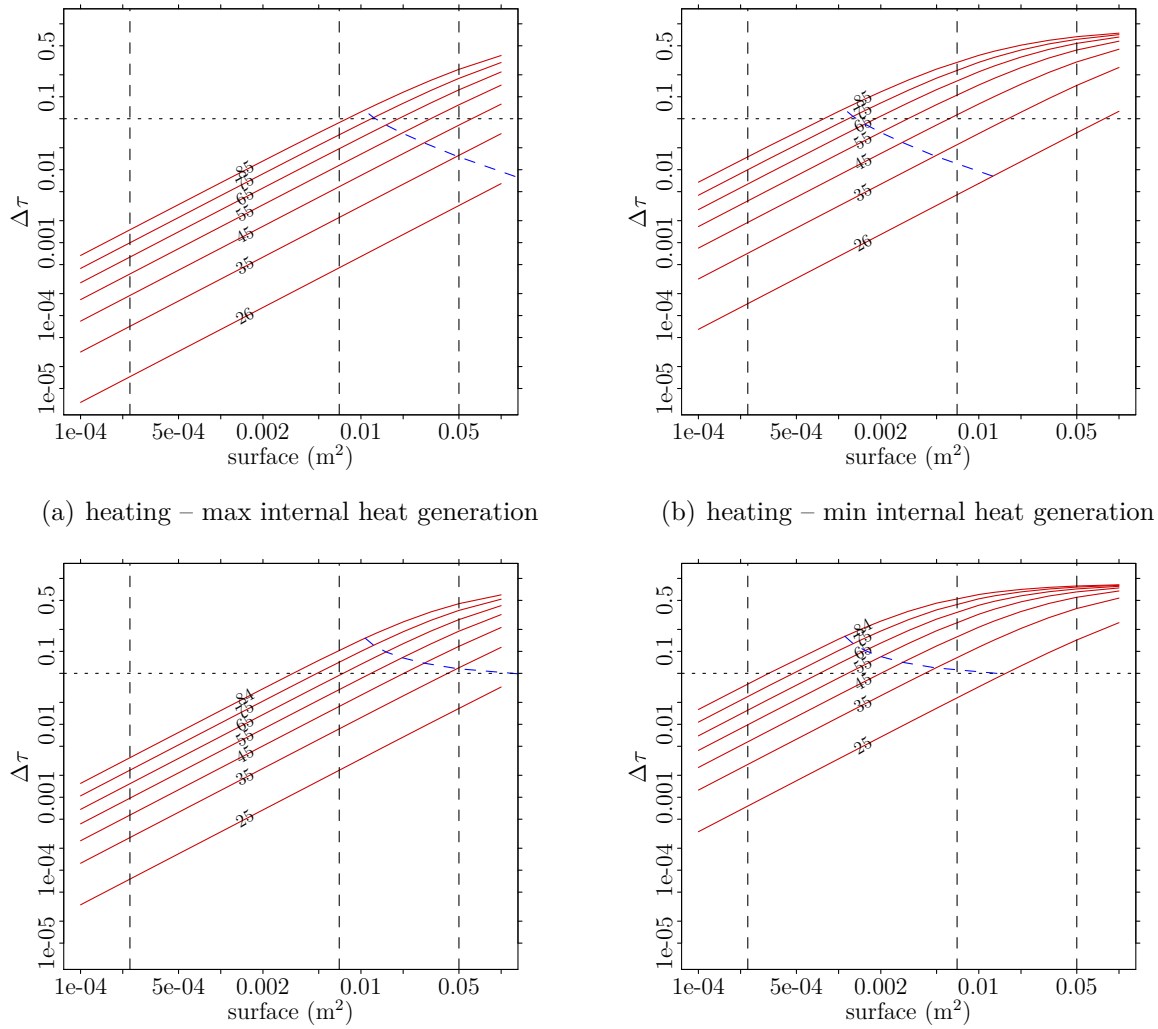
Let's define the temperature lag ΔT between an actively and passively cooled identical body, measured at the moment when the passively cooled body reaches a reference temperature T_{pc}. The reference temperature T_{pc} is henceforth defined as $T_{\text{pc}} = 0.85(T_e - T_0) + T_0$, i.e., when the body temperature has reached 85 % of its equilibrium temperature, starting from T_0. It is also assumed that both the passively and actively cooled bodies have the same internal heat generation process and initial condition T_0 at $t = 0$. Figure 5.10 shows the relative temperature lag $\Delta\tau$, which is defined as the absolute temperature lag ΔT divided by the temperature difference at $t = 0$ and at equilibrium, i.e. , $|T_e - T_0|$:

$$\Delta\tau = \frac{\Delta T}{|T_e - T_0|} = \frac{T_{\text{pc}} - T_{\text{ac}}}{|T_e - T_0|}. \tag{5.44}$$

The relative temperature lag $\Delta\tau$ is depicted in Figure 5.10 for both a large and small internal heat generation, as defined before, and for the heating and cooling processes separately. A dotted reference line is drawn at $\Delta\tau = 5\,\%$. Data points on the right of the dashed blue line show configurations with one or more negative convective heat transfer coefficients. This implies that in these cases additional heat needs to added to the system to attain the target equilibrium temperature.

For the case of large internal heat generation, the relative temperature lag $\Delta\tau$ for small surfaces stays below 0.5 % meaning that the presence of radiative heating will be quasi unnoticeable here. $\Delta\tau$ stays around 5 %, in the case of small internal heat generation, which may be difficult to spot. Contemporary on-die processor temperature sensors report frequently temperatures in steps of 1°C. Given this quantization noise, a relative temperature lag of 5 % could be hard to identify when $|T_e - T_0| > 20$°C. So for small microprocessor temperature variations, it is again unlikely that a contemporary microprocessor temperature sensor is able to distinguish between active and passive cooling. For a smartphone-size cooling surface, the relative temperature lag varies significantly depending on the situation. For a large internal heat generation and heating, there is less than 5 % difference between passive and active cooling. For the other cases, however, the discrepancy between the passive and active cooling can run up from nil to as high as 10 %, depending on the equilibrium temperature. $\Delta\tau = 10\,\%$ is already noticeable at $|T_e - T_0| > 10$°C in the presence of 1°C quantization noise. The data for the tablet-sized cooling surfaces is mostly not of our concern as the convective heat transfer coefficients for passive and active cooling are negative. This means that heat needs to be added to the system to attain the given equilibrium temperature.

Generally speaking, it was noticed that the relative temperature lag $\Delta\tau$ for the heating case is smaller than for the cooling case. This can be explained via the inequality of Equation 5.43. The radiative heat transfer coefficient will have greater weight when the body's temperature is larger than the equilibrium temperature than when the temperature is below the equilibrium, hence inflating the discrepancy between active and passive cooling. Also, the amount of internal heat generation affects the relative temperature lag. It appears that, the larger the internal heat generation, the smaller $\Delta\tau$ becomes. Indeed, given the differential representation of the cooling law in Equation 5.5, for a fixed equilibrium temperature, it can be seen that the convective cooling part can outweigh



(a) heating – max internal heat generation

(b) heating – min internal heat generation

(c) cooling – max internal heat generation

(d) cooling – min internal heat generation

Figure 5.10: Relative time lag $\Delta\tau$ (Equation 5.44) for the internal heat generation set to the maximum (a,c), and set to the minimum (b,d). The different curves are generated for different equilibrium temperatures (see curve labels in °C). On the top row the heating process is depicted (a,b), and the cooling process on the bottom row (c,d). The three vertical dashed lines represent typical surfaces for a SoC ($\approx 2.5\,\mathrm{cm}^2$), a smartphone ($\approx 70\,\mathrm{cm}^2$) and a tablet ($\approx 5\,\mathrm{dm}^2$). Data points on the right of the blue dashed lines have negative convective heat transfer coefficients.

the radiative cooling the larger the internal heat generation becomes. Thus the larger the internal heat generation, the less sensitive the body becomes to changes in the radiative or convective cooling, and the more active and passive cooling will resemble.

5.5 Conclusion

The exact cooling law for passively cooled objects subject to radiation, (natural) convection, and internal heat generation was developed. The passive cooling law is analytically more complex than the commonly accepted exponential cooling law, which is technically sound for forcibly cooled objects. Unfortunately, the exact solution for the passively cooled object is a function of temperature: $t(T)$. Numerical approaches can be used for the computation of the exact inverse: $T(t)$. The validation of the passive cooling law's accurate solution via CFD simulations demonstrated the cooling law's adequacy. Approximations for the inverse solution $T(t)$ to the passive heat equation were proposed and their performance was assessed. The coefficient approximation came out as the best candidate. The second-order O'Sullivan approximation performs reasonably but only for small temperature departures from the ambient temperature.

Via analytical simulations, it was shown that the difference between active and passive cooling depends on three factors: 1) the surface size of the object, 2) the internal heat generation, and 3) the equilibrium temperature. For large surfaces areas, it was shown that the difference between active and passive cooling can be significant. For medium-sized surface areas, depending on the magnitudes of the internal heat generation and equilibrium temperature, the discrepancy between active and passive cooling could probably go unnoticed. For small surfaces, e.g., SoCs, an exponential cooling law is shown to be an appropriate approximation. It was also highlighted that the quantization noise of temperature sensors may conceal temporal information between active and passive cooling. As the cooling law for passively cooled devices is quite elaborate to work with and the benefit of a scientifically sound cooling law is limited by the current lack of accurate temperature sensors, it can be stated that, for systems minimizing overhead, assuming an exponential cooling law will likely not induce large perceptual deviations from reality.

Chapter 6

Optimal Energy/Frequency Applications to Multi-Clock Domains

The Energy/Frequency Convexity Rule from Chapter 4 is applied to specific applications in this chapter. Multi-buddy systems are studied, in Section 6.1, and their analytical formulations are developed to find the optimal clock frequency for all clock frequency-enabled buddies, with the aim of global system-wide energy consumption. It is shown that, to find the optimal clock-frequencies for each buddy, the convexity rule should be reapplied each time the system changes energy states. In Section 6.2 the convexity rule is applied to a system subject to deadlines. The findings show that up to 55 % of energy can be save in the best case, but savings can become marginally small when a lot of work has to be done before the deadline. Section 6.3 investigates whether the application of the convexity rule can save energy by clock frequency scaling individual parallel threads. It will be shown that, for a background power consumption comparable to the energy consumption of the microprocessor, energy savings are possible between 10 % and 3 % compared to a default Linux-like frequency governor. Section 6.4 incorporates the notion of energy in Amdahl's law to analyze the performance per Joule of a microprocessor. It is shown how the clock frequency that minimizes energy consumption behaves under Amdahl's law. In Section 6.5 the heterogeneous computing w.r.t. energy folklore is demystified. The contrast between a low-power and a high-performance microprocessor is highlighted using the Energy/Frequency convexity Rule and the extended Amdahl's law.

Henceforth, a specific taxonomy is introduced for referring to specific power components. A dynamic energy or power component is denoted by a tilde over the symbol: $\tilde{}$, whereas a static component is denoted by a bar: $\bar{}$. Similarly, a $^+$ refers to the component in *active* state, and $^\circ$ refers to the component in idle state. For example $\tilde{E}^+$ is the dynamic energy consumption in active state, and $\bar{P}^\circ$ is the static power in idle state.

6.1 Two-Buddy Convexity Model

In the Section 4.2 a single microprocessor was considered in a system with constant background power demands. In various applications, however, the microprocessor coexists with other clock-driven components that have their own power profiles. The micropro-

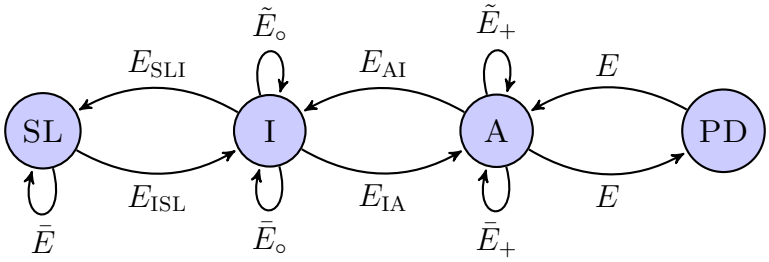


Figure 6.1: Energy state machine representation of a computer component. Three distinct states are depicted: *power-down mode* (PD), *idle mode* (I), *sleep mode* (SL), and *active mode* (A), and their transition energy requirements E_*. There exist *dynamic* $\tilde{E}$ and *static* $\bar{E}$ energy costs to maintain the states, and an energy cost to change between states.

cessor may interact with these external components by exchanging some form of data. Here, such systems are referred to as n-buddy systems, where n is the number of components with independent clock-frequency domains. Some examples of n-buddy system are: the coexistence of a microprocessor and an external memory, or the collaboration of multiple cores inside a single microprocessor that are separately clock frequency scalable, a microprocessor next to a radio driver or multimedia decoder. While the microprocessor is polling external components it may be stalling until a response is received, or the microprocessor may continue other data-independent work in the meantime. Similarly, the external components may be waiting for the microprocessor for a request before initiating any work. The microprocessor and other external clock-driven components can be represented with an energy *state machine*. Three distinct states can be distinguished for the energy state machine:

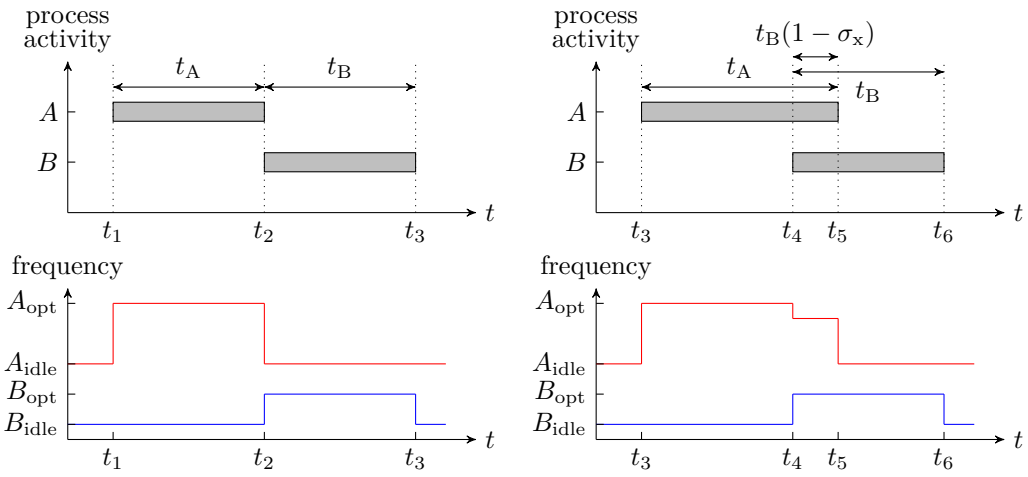
1. *active mode*: performing computations;
2. *idle mode*: waiting or standing-by, or performing no useful operations;
3. *sleeping mode*: sleeping in a low-power state;
4. *power-down mode*: no energy supplied to the device.

A computer component is in one state at each time. A computer component's power dissipation in the active and idle mode have both a dynamic and a static power consumption component. Moreover, transitioning between each state can induce an energy *transition cost*. Figure 6.1 shows a graphical representation of the energy state machine.

Let us develop the optimal frequency model for a system with two buddies, A and B with their own respective clock frequency and energy profile. In such a system, each of the two buddies will have their own proper optimal frequency: $f^{\mathrm{A}}_{\mathrm{opt}}$ and $f^{\mathrm{B}}_{\mathrm{opt}}$ at any time.

Let us assume for starters a simple synchronous system where A is the master and B is the slave, as shown in Figure 6.2(a). Over a timespan Δt, when A is polling B, A is idle while B is active, and B is idle whenever A is active. The energy consumption of such a system can be expressed as the sum of the energy consumption of each buddy separately: E_{A} and E_{B}, and their energy state transition cost E_{tr}:

$$\begin{aligned} E_{\mathrm{sys}} &= E_{\mathrm{A}} + E_{\mathrm{B}} + \delta E_{\mathrm{tr}} \\ &= (t_{\mathrm{A}} P^{+}_{\mathrm{A}} + t_{\mathrm{B}} P^{\circ}_{\mathrm{A}}) + (t_{\mathrm{B}} P^{+}_{\mathrm{B}} + t_{\mathrm{A}} P^{\circ}_{\mathrm{B}}) + \delta E_{\mathrm{tr}} \\ &= t_{\mathrm{A}}(P^{+}_{\mathrm{A}} + P^{\circ}_{\mathrm{B}}) + t_{\mathrm{B}}(P^{\circ}_{\mathrm{A}} + P^{+}_{\mathrm{B}}) + \delta E_{\mathrm{tr}}, \end{aligned} \tag{6.1}$$



(a) master/slave (b) out-of-order execution

Figure 6.2: Frequency time line in (a) master/slave and (b) out-of-order two-buddy system. When process A is active process B is idle and vice versa in case of the master/slave configuration. Between t_4 and t_5 process A is able to cover idle time with data-independent code execution or out-of-order execution. During this timespan, each buddy may have a unique frequency assigned to minimize energy consumption.

where δ are the number of transitions between states, t_A is the time during which buddy A is active, and t_B the time during which buddy B is active. t_A and t_B are related by the equation $\Delta t - t_A = t_B$. Here it is also assumed implicitly that whenever buddy A or B is active they operate at a single frequency, and in idle state they operate at a lower clock frequency. If it is assumed that buddy A starts in active state, and also ends in active state over the timespan Δt then δ is assumed to be even and E_{tr} can be defined as the sum of E_{AI} and E_{IA}: $E_{tr} = E_{AI} + E_{IA}$.

The execution time t_A and t_B are proportional to the amount of clock cycles executed cc by the buddies and their respective clock frequency f: $t = cc/f$, given by Equation 2.14, f_k and β are assumed to be zero. Additionally, let us also assume that buddies A and B can perform out-of-order execution (OOE). Then the idle time of buddy A and B are reduced by the parameter σ_x, proportional to the extent to which OOE is able to cover the tasks in A or B with the execution of data-independent code. $\sigma_x = 0$ means perfect OOE coverage, whereas $\sigma_x = 1$ indicates that no OOE is performed. Figure 6.2(b) shows an explanatory time line of the *two*-buddy system with OOE and proper clock frequencies at active state X_{opt} and idle state X_{idle}. From t_4 to t_5 process A is able to cover process' B with data-independent code execution.

The total energy consumption of Equation 6.1 now becomes the sum of the energy spent during the initial timespan t_A and t_B, minus the idle energy during the OOE overlap, over the timespan $t_B(1-\sigma_x)$, plus the energy spent between state transition:

$$\begin{aligned} E_{sys} &= t_A(P_A^+ + P_B^\circ) + t_B(P_B^+ + P_A^\circ) - t_B(1-\sigma_x)(P_A^\circ + P_B^\circ) + \delta E_{tr} \\ &= \frac{cc_A}{f_A}(P_A^+ + P_B^\circ) + \frac{cc_B}{f_B}[(P_B^+ + P_A^\circ) - (1-\sigma_x)(P_A^\circ + P_B^\circ)] + \delta E_{tr}. \end{aligned} \tag{6.2}$$

The dynamic power components $\tilde{P}$, and also the static power $\bar{P}$ components, are frequency dependent via Equation 2.8 and Equation 2.10:

$$P = \tilde{P} + \bar{P} = (1 + \gamma V)\, \xi f V^2.$$

Let us also assume for now that the power consumption in idle mode $P_\circ = \tilde{P}_\circ + \bar{P}_\circ$ is a fixed constant, independent of a clock frequency. Substituting these equations in the last definition of E_{sys} creates an overwhelmingly large and complex equation, which is refrained from citing. It is noted, however, that the resulting equation is, again, of the fourth order because of the assumption of linear dependency of the voltage on clock frequency.

For $\max(f_{\min}, f_{\text{k}}) \leq f_* \leq f_{\max}$ the energy consumption of the two-buddy system E_{sys} may have the convex property, and thus a global minimum, iff its first derivative with respect to f has a root and its second derivative is monotonically increasing. This is expressed in Equation 4.9. Taking the first and second derivative of f_{A} and f_{B} yields:

$$\frac{\partial E_{\text{sys}}}{\partial f_{\text{A}}} = cc_{\text{A}}(3a_A f_{\text{A}}^2 + 2b_A f_{\text{A}} + c_A) - \frac{cc_{\text{A}}}{f_{\text{A}}^2} P_{\text{B}}^\circ \tag{6.3a}$$

$$\frac{\partial E_{\text{sys}}}{\partial f_{\text{B}}} = cc_{\text{B}}(3a_B f_{\text{B}}^2 + 2b_B f_{\text{B}} + c_B) - \frac{cc_{\text{B}}}{f_{\text{B}}^2}[P_{\text{A}}^\circ - (1-\sigma_{\text{x}})(P_{\text{A}}^\circ + P_{\text{B}}^\circ)] \tag{6.3b}$$

$$\frac{\partial^2 E_{\text{sys}}}{\partial f_{\text{A}}^2} = cc_{\text{A}}(6a_A f_{\text{A}} + 2b_A) + 2\frac{cc_{\text{A}}}{f_{\text{A}}^3} P_{\text{B}}^\circ \tag{6.3c}$$

$$\frac{\partial^2 E_{\text{sys}}}{\partial f_{\text{B}}^2} = cc_{\text{B}}(6a_B f_{\text{B}} + 2b_B) + 2\frac{cc_{\text{B}}}{f_{\text{B}}^3}[P_{\text{A}}^\circ - (1-\sigma_{\text{x}})(P_{\text{A}}^\circ + P_{\text{B}}^\circ)], \tag{6.3d}$$

where $a_* = \gamma\xi s_1^3$, $b_* = s_1^2\xi(1+3\gamma s_2)$, $c_* = s_1 s_2 \xi(3\gamma s_2 + 2)$ for each microprocessor separately. All parameters in Equation 6.3 are positive: $\{\cdot\} \in \mathbb{R}_0^+$. Hence, convexity for $\frac{\partial^2 E_{\text{sys}}}{\partial f_{\text{A}}^2}$ is achievable. However, for $\frac{\partial^2 E_{\text{sys}}}{\partial f_{\text{B}}^2}$ the constraint

$$3a_B f_{\text{B}} + b_B > \frac{1}{f_{\text{B}}^3}[P_{\text{B}}^\circ - \sigma_{\text{x}}(P_{\text{A}}^\circ + P_{\text{B}}^\circ)]$$

must hold. The optimal frequency $f_{\text{opt}}^{\text{A}}$ and $f_{\text{opt}}^{\text{B}}$ are found by equating Equation 6.3a and 6.3b to zero:

$$0 = 3a_A (f_{\text{opt}}^{\text{A}})^2 + 2b_A f_{\text{opt}}^{\text{A}} + c_A - \frac{1}{(f_{\text{opt}}^{\text{A}})^2} P_{\text{B}}^\circ \tag{6.4a}$$

$$0 = 3a_B (f_{\text{opt}}^{\text{B}})^2 + 2b_B f_{\text{opt}}^{\text{B}} + c_B - \frac{1}{(f_{\text{opt}}^{\text{B}})^2}[P_{\text{A}}^\circ + (\sigma_{\text{x}} - 1)(P_{\text{A}}^\circ + P_{\text{B}}^\circ)]. \tag{6.4b}$$

One can observe that Equation 6.4a is independent of f_{B}, and Equation 6.4b is independent of f_{A}. Thus computing the proper frequencies of the buddies is independent of the active power consumption of the other buddy.

Now, the interesting part here is that Equation 6.3 assumed a single optimal frequency for the active states of each buddy. Alternatively, it is also possible to define an independent optimal frequency for the timespan during which both buddies are active; for $t_4 < t < t_5$ in Figure 6.2, and for the other time periods where one buddy is active and the other idle. The length of the timespan where both buddies are active is dependent of the amount of code that buddy A can execute out-of-order while buddy B is also active, lets say for cc_{C} clock cycles:

$$E_{\text{C}} = \frac{cc_{\text{C}}}{f_{\text{A}}}(P_{\text{A}}^+ + P_{\text{B}}^+) \tag{6.5a}$$

$$\frac{\partial E_{\text{C}}}{\partial f_{\text{A}}} = cc_{\text{C}}\left(3a_A f_{\text{A}}^2 + 2b_A f_{\text{A}} + c_A - \frac{1}{f_{\text{A}}^2} P_{\text{B}}^+\right) \tag{6.5b}$$

$$\frac{\partial^2 E_{\text{C}}}{\partial f_{\text{A}}^2} = cc_{\text{C}}\left(6a_A f_{\text{A}} + 2b_A + \frac{2}{f_{\text{A}}^3} P_{\text{B}}^+\right) \tag{6.5c}$$

To find the optimal operating frequencies $f^{\mathrm{A}}_{\mathrm{opt}}$ and $f^{\mathrm{B}}_{\mathrm{opt}}$, Equation 6.5b needs to equal zero, as well as the $\frac{\partial E_{\mathrm{C}}}{\partial f_{\mathrm{B}}}$ counterpart:

$$3a_A(f^{\mathrm{A}}_{\mathrm{opt}})^2 + 2b_A f^{\mathrm{A}}_{\mathrm{opt}} + c_A - \frac{1}{(f^{\mathrm{A}}_{\mathrm{opt}})^2} P^{+}_{\mathrm{B}} = 0 \tag{6.6a}$$

$$3a_A(f^{\mathrm{A}}_{\mathrm{opt}})^4 + 2b_A(f^{\mathrm{A}}_{\mathrm{opt}})^3 + c_A(f^{\mathrm{A}}_{\mathrm{opt}})^2 = a_B(f^{\mathrm{B}}_{\mathrm{opt}})^4 + b_B(f^{\mathrm{B}}_{\mathrm{opt}})^3 + c_B(f^{\mathrm{B}}_{\mathrm{opt}})^2 + d_B f^{\mathrm{B}}_{\mathrm{opt}} \tag{6.6b}$$

and similarly for f_{B} the following must hold

$$3a_A(f^{\mathrm{B}}_{\mathrm{opt}})^4 + 2b_A(f^{\mathrm{B}}_{\mathrm{opt}})^3 + c_A(f^{\mathrm{B}}_{\mathrm{opt}})^2 = a_B(f^{\mathrm{A}}_{\mathrm{opt}})^4 + b_B(f^{\mathrm{A}}_{\mathrm{opt}})^3 + c_B(f^{\mathrm{A}}_{\mathrm{opt}})^2 + d_B f^{\mathrm{A}}_{\mathrm{opt}}. \tag{6.6c}$$

An analytically solution for the latter equation is fairly complex. Numerical methods will come at hand here. Following this rationale, each buddy in an n-buddy system has its respective optimal clock frequency for each combination of buddy energy states. One can argue that such a system is more optimized energy-wise than a system that has a distinct clock frequency for each buddy throughout the whole execution. Therefore the former will be superior, in terms of energy consumption, compared to the latter approach.

6.2 Single Core with Deadlines

When the theoretical foundations of the energy/frequency rule were laid out in Section 4.2, only the energy during the execution of the code sequence was accounted for. In practical situations, e.g., when dealing with deadlines, however, a microprocessor may be in an idle-state during certain periods of time. Then the energy consumed during idle-state should be accounted for when assessing the optimal frequency at which energy consumption is minimal.

6.2.1 Modeling Including P_{idle}

The microprocessor of concern is governed by the energy consumption law of Equation 4.2. The microprocessor in idle-state requires P_{idle} Watt and it is also assumed that microprocessor clock frequency transitions don't incur a significant time or energy penalty.

Let us assume a repetitive task k, which needs cc_{b} clock cycles to complete, is scheduled such that within a fixed duration $t_{\max}$ the task is to be completed. The execution time Δt of the task k is bounded by $t_{\max}$:

$$t_{\max} \geq cc_{\mathrm{b}} \left(\frac{1}{f - f_{\mathrm{k}}} + \beta \right). \tag{6.7}$$

The upper-bound on the execution time implies that the task has a minimum frequency $f_{\min}$ for which $t_{\max} = \Delta t$ is satisfied. As repetitive tasks are dealt with, the background power requirements are not going to affect the optimal frequency given that it is there during the repetition of each task. The background power consumption should thus not be considered when finding the optimal processor clock frequency in this case. The energy consumption of the microprocessor is the sum of the energy spent executing the task and the energy consumed during idle-state:

$$\begin{aligned} E_{\mathrm{pr}} &= E_{\mathrm{act}} + E_{\mathrm{idle}} \\ &= P_{\mathrm{act}}\ \Delta t + P_{\mathrm{idle}}\ (t_{\max} - \Delta t) \\ &= [(1+\gamma V)\ \xi f V^2 - P_{\mathrm{idle}}]\Delta t + P_{\mathrm{idle}}\ t_{\max}. \end{aligned}$$

Senn *et al.* [103] used a similar formulation for E_{pr}, though, their power and time models are more rudimentary. For analytical simplicity, let us assume that the time penalty for external memory accesses and clock cycle thieves is negligibly small ($\beta = 0$ and $f_{\text{k}} = 0$), and Δt is a function of f:

$$\begin{aligned} E_{\text{pr}} &= \left[(1+\gamma V)\,\xi f V^2 - P_{\text{idle}}\right]\frac{cc_{\text{b}}}{f} + P_{\text{idle}}\,t_{\max} \\ &= cc_{\text{b}}\left(af^3 + bf^2 + cf + d - \frac{P_{\text{idle}}}{f}\right) + P_{\text{idle}}\,t_{\max}, \end{aligned}$$

where $a = \gamma\xi s_1^3$, $b = s_1^2\xi(1+\gamma s_2+2\gamma s_2)$, $c = s_1 s_2 \xi(\gamma s_2 + 2\gamma s_2 + 2)$, and $d = s_2^2\xi(\gamma s_2 + 1)$, as described in Section 4.2. $t_{\max}$, cc_{b}, f_{k}, and P_{idle} are fixed. All parameters are elements of $\mathbb{R}_0^+$. As before, E_{pr} may have a convex minimum, within the exploitable clock frequency window, if its first derivative has a root and the second derivative is a monotonous increasing function. For a fixed P_{idle} one gets:

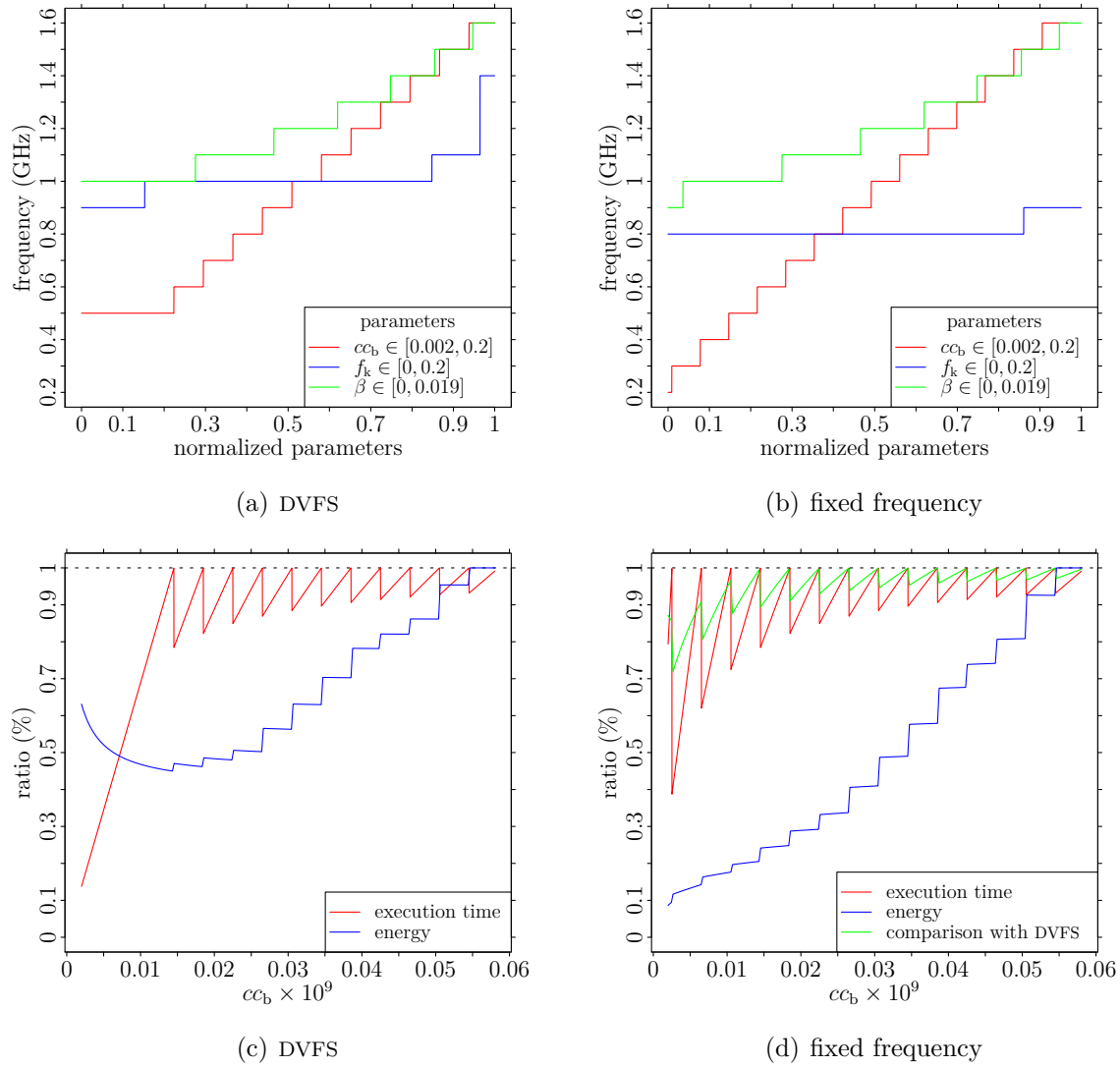
$$\begin{aligned} \frac{\mathrm{d}E_{\text{pr}}}{\mathrm{d}f} &= cc_{\text{b}}\left(3af^2 + 2bf + c + \frac{P_{\text{idle}}}{f^2}\right) \\ \frac{\mathrm{d}^2E_{\text{pr}}}{\mathrm{d}f^2} &= cc_{\text{b}}\left(6af + 2b - 2\frac{P_{\text{idle}}}{f^3}\right). \end{aligned}$$

Keep in mind that the constraint of Equation 6.7 must hold as well. Here $\frac{\mathrm{d}E_{\text{pr}}}{\mathrm{d}f}$ is a monotonic increasing function of f as all parameters are elements of $\mathbb{R}_0^+$. The second derivative $\frac{\mathrm{d}^2E_{\text{pr}}}{\mathrm{d}f^2}$ is in fact only a monotonous increasing function iff $6af + 2b > 2\frac{P_{\text{idle}}}{f^3}$ for all $f \in f_{\text{cpu}}$. For the system to show convex energy properties P_{idle} should be as small as possible. Then, the lower P_{idle}, the lower f_{opt}, being the root of the first derivative, without violating the $t_{\max}$ time constraint.

6.2.2 Experimental Results

Let us insert realistic values from the previously defined Galaxy S2 testbed where the video decoding by the microprocessor, with a frame-rate of 25 Hz, is simulated. A frame-rate of 25 Hz translates into a decoding deadline of $t_{\max} = 40$ ms. The power requirements in idle-state P_{idle} can be chosen either as the power demands of the microprocessor's lowest frequency, or the idle power demands at the optimal frequency. The first would apply frequency scaling on-the-fly, whereas in the second case a static frequency throughout the decoding is assumed. The power parameters γ, ξ, and time parameters f_{k} and $\beta = 0$ for the Cortex A9 microprocessor are assumed as reported in Table 4.2 and 4.3 for $N = 10$ unless stated otherwise. Accordingly, the discrete voltage/frequency pairs of the A9 from Table 3.6 are also used. The influence of cc_{b}, f_{k} and β are assessed during the simulations. Figure 6.3 shows the results. In Figure 6.3(a) the microprocessor's frequency is scaled to the minimum frequency in idle mode (DVFS), whereas in Figure 6.3(b) the microprocessor's frequency remains unaltered throughout the simulation.

For the case where the microprocessor's frequency in idle mode is scaled, Figure 6.3(a), it can be seen that while cc_{b} increases the system minimizes energy by increasing the microprocessor's frequency. The reasons are two-fold for this frequency increase; the larger cc_{b} the faster the microprocessor needs to run to meet its deadline but also the energy/frequency convex minimum shifts to higher frequencies as explained in Chapter 4. The red line in Figure 6.3(c) shows the ratio of the decoding time over the remaining time



(a) DVFS (b) fixed frequency

(c) DVFS (d) fixed frequency

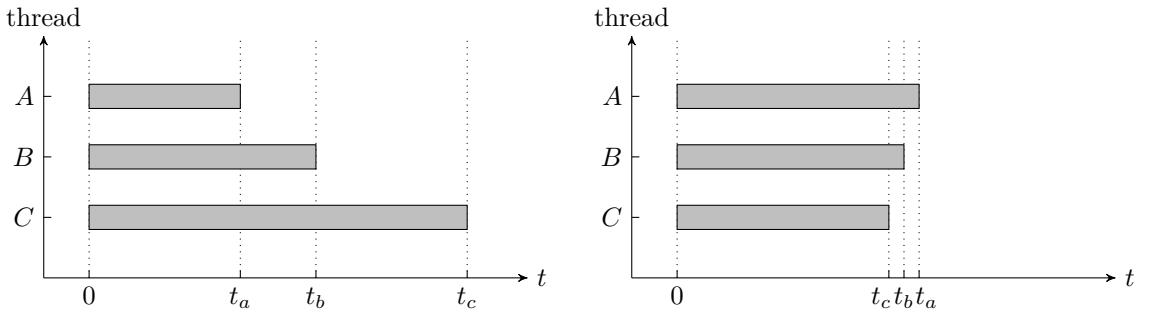
Figure 6.3: Optimal microprocessor clock frequency minimizing energy consumption of a code piece executed repetitively before a deadline. In Figure (a) and (c) the microprocessor frequency is scaled to the minimum frequency in idle mode via DVFS. In Figure (b) and (d) the microprocessor's frequency remains unaltered throughout the simulation. Maximum energy savings can be seen in (c) for $cc_b \approx 15 \times 10^{-9}$ where smart frequency scaling can save about 55 % of energy. The parameters in the legend of Figure (a) and (b) are normalized over their range where max(parameter) = 1, and min(parameter) = 0.

before the dead-line. The smaller cc_b, the less work to be done, so the smaller the execution time ratio. But also, the more work to be done, the higher the microprocessor frequency to finish the decoding before the deadline. This is visible in Figure 6.3(c) and 6.3(d); when the execution time ratio is about to hit 1, the execution time is about to miss its deadline, and therefore the microprocessor needs to gear up the frequency to meets its deadline. This also explains the stair-case-like behavior of the curves in Figure 6.3(a) and 6.3(b). For small values of cc_b the optimal frequency does not go below 0.5 Ghz. This is so because the slower microprocessor frequencies inflate the execution time asymptotically. The smaller active power consumption is not able to offset proportionally the increased execution time to gain an advantage energy-wise. Therefore the optimal frequency doesn't drop below a certain frequency, in this particular case 0.5 GHz. The energy savings of the optimal frequency for a variable cc_b are shown with the blue line in Figure 6.3(c) as the ratio of the energy consumption while operating at the optimal frequency to the energy consumption while running at the maximum frequency in active mode. The latter corresponds to the *on-demand* and *interactive* frequency governor of mainstream Linux distributions. A maximum energy can be saved of around 55 % around $cc_b = 0.015 \times 10^9$ after which the energy savings gradually drop. At the maximum cc_b there is no energy saving as the optimal operating frequency is equal to the maximum microprocessor frequency. Running at an optimal frequency can thus save considerable amounts of energy if favorable conditions are encountered. In this application it is also shown that the *race-to-halt* technique is not always the optimal strategy to save energy.

The f_k parameter influences the execution time; the larger f_k the longer the execution time as less clock cycles are available. From both Figure 6.3(a) and 6.3(b) it can be observed that f_k increases the optimal frequency. An increased f_k will thus also result in a decreased possible energy gain. Similar conclusions can be drawn for the β parameter. β incurs a time penalty for the code execution which leads to increased execution time. β also increases the optimal execution time and hence is also detrimental for energy gains.

The stair-case-like behavior is also exhibited when the microprocessor is running at a constant frequency, i.e., no DVFS, as shown in Figure 6.3(c). The optimal frequency, however, drops all the way to the minimum frequency for small values of cc_b in contrast with the DVFS scenario. Here, the extra power consumption in idle-mode weighs through more compared to the DVFS case. In the DVFS case, in idle-mode, the microprocessor frequency was geared to the minimum frequency whereas in the constant frequency case the frequency is not altered. This has some influence on the energy gains as well. The blue line in Figure 6.3(d) shows a monotonic increasing energy gain curve which is between 0 and 90 % for the extreme values of cc_b. The reader should however not be fooled; the green line in the same figure shows the ratio of the energy gains in the DVFS case over the energy gains in the constant frequency case. One can thus observe that the energy gains in the DVFS case are up to 20 % better for small values of cc_b but the competitive advantage shrinks fast, to about 1 %, for larger values of cc_b. Note that in this application with deadlines the parameter cc_b has influence on the optimal clock frequency whereas cc_b doesn't have any effect on optimal frequency when a single code sequence is regarded. The same conclusions for the parameters cc_b and β can be drawn for the constant clock frequency case as in the case where the microprocessor runs at a lower frequency in idle-mode.

In practice, a performance profile of the application is required, especially with regards to cc_b, so that the microprocessor's clock frequency can be efficiently scaled. The profile can be defined offline or via online profiling techniques. Online parameter profiling and



(a) global clock frequency scaling (b) individual clock frequency scaling

Figure 6.4: Energy optimization via individual thread clock frequency scaling. Leftmost, all cores run at the same frequency incurring excessive idle time for some threads. On the right, the threads are throttled individually to achieve global or local minimum energy consumption.

estimation techniques have been shown to be viable based on application cycle probability distributions [136].

It is also noted that in this example the microprocessor was assumed to decode a video stream. In fact, SoCs often provide dedicated decoding and encoding circuits to process popular multimedia formats such as MPEG-1 or MPEG-2 Audio Layer III (MP3) and DivX. Though, the "media" in this example can be replaced with any other computation that must meet deadlines.

6.3 Multi-core Code Execution

A program may have multiple data-independent threads running in parallel on multiple cores. At a given point in time these threads need to synchronize such that the serial portion of the program can continue. If these threads run on separate cores, where clock frequencies can be scaled independently, then there must exist a frequency scheme to minimize the energy consumption of the microprocessor and the whole computer system. Figure 6.4 illustrates an example of the individual frequency scaling of three concurrent threads. All threads must cooperate with each other, in terms of clock frequency scaling, to attain the minimum energy consumption globally, a process henceforth referred to as *thread-cooperation*. It may not be presupposed that the execution time of the shortest threads must be matched with the longest thread to minimize energy consumption. This is experimentally shown with the following optimization definition of thread-cooperation, where $f = \{f_i\}$ is the set of clock frequencies of n cores:

$$\begin{aligned} \underset{f}{\text{minimize}} \quad & E_{\text{tot}}(f) \\ \text{subject to} \quad & \forall i \;\; \frac{cc_{\text{b},i}}{f_i} \leq t_{\max} \\ & f_{\min} \leq f_i \leq f_{\max}, \end{aligned}$$

where $t_{\max}$ is a predefined maximum execution time of all threads. $E_{\text{tot}}(f) : \mathbb{R}^m \to \mathbb{R}$ is the objective function to be minimized over f:

$$E_{\text{tot}}(f) = P_{\text{back}} t_{\max} + \sum_{i=0}^{n} \left[\frac{cc_{\text{b},i}}{f_i} P^{+} + \left(t_{\max} - \frac{cc_{\text{b},i}}{f_i} \right) P^{\circ} \right], \tag{6.8}$$

where P^+ is the power consumption when the core is performing work and P° is the power consumption of the core while being idle. A core is in idle mode whenever the thread finished execution and is waiting for another parallel thread to finish.

The optimization problem in Equation 6.8 is fairly complex as the power terms are non-linear. Instead of formulating a discrete optimization problem, in the case of discrete frequency/voltage pairs, an exhaustive search simulator is developed in `C++` to find the minimum energy of the objective function. This is feasible as the search space is limited. The performance of three clock frequency assignment schemes are analyzed in the simulation:

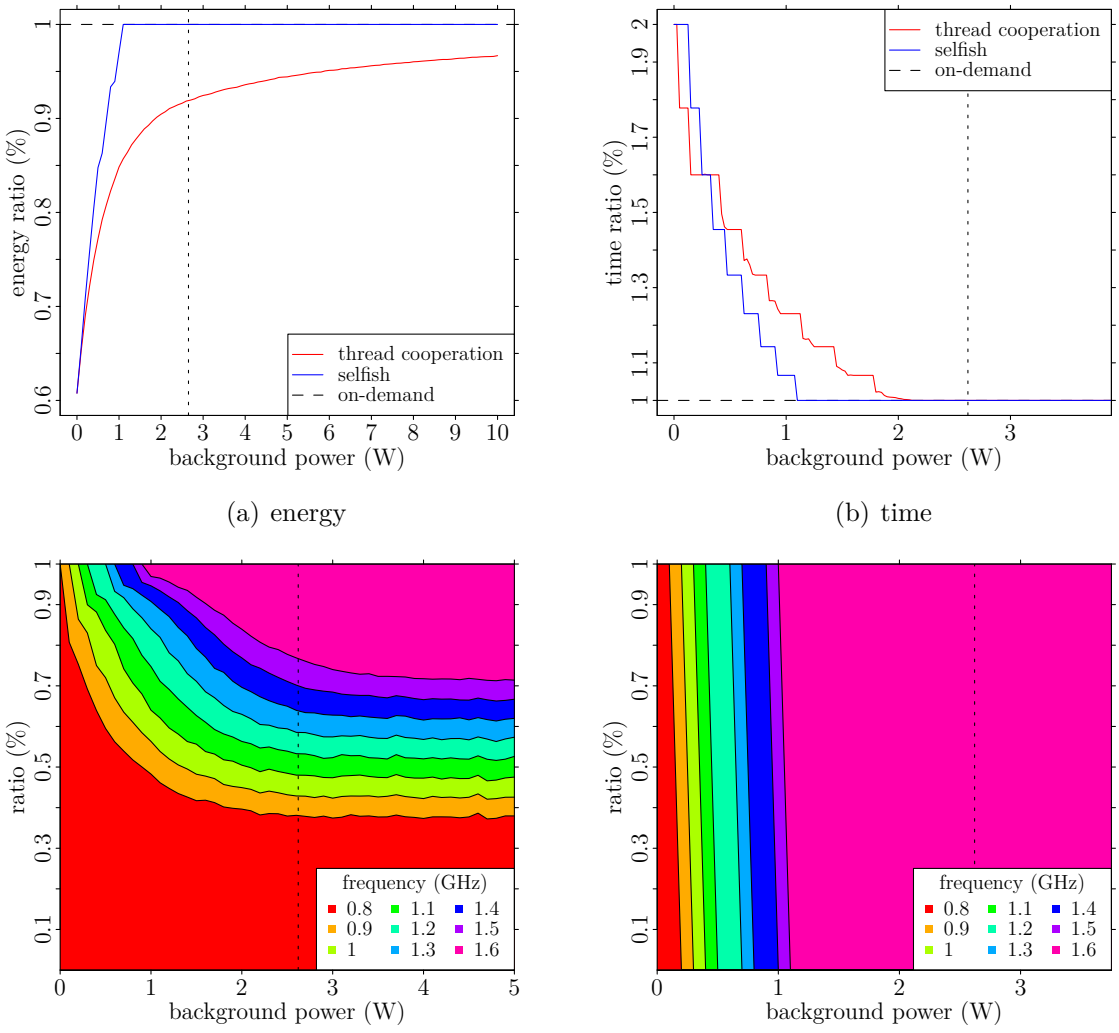
1. *On-demand*: when a core has work to do, its frequency is set to the maximum. If no work is to be done, the core is set to an idle frequency, the lowest frequency. This frequency scheme corresponds to the *on-demand* and *interactive* frequency governor in the Linux kernel, and is used as a reference.

2. *Selfish*: each core is frequency scaled to be energy optimal by its own, disregarding the energy consumption of the other cores.

3. *Thread-cooperation*: the frequencies of the cores are scaled such that all the cores combined consume the least amount of energy globally.

Four threads are supposed to run in parallel, and need a random number of clock cycles cc_b to complete. The number of clock cycles was drawn from a uniform distribution where $0 \leq cc_\mathrm{b} \leq 4.9 \times 10^9$. The Cortex A15 power model from Procedure 3.2 and the frequency/voltage pairs are used to model the power consumption of the cores. Though in the original A15 the cores are not individually scalable, there is only one global frequency governing all cores, for the simulator individual core frequency scaling was introduced for the A15. The power was estimated at a reference temperature of 37°C. The average energy consumption, averaged over 4096 runs, of the three frequency governors is measured for background power requirements varying between 0 and 10 W. For the thread execution time it is also assumed that the time penalty for external data accesses and clock cycle thieves is negligibly small ($\beta = 0$ and $f_\mathrm{k} = 0$). The power consumption in idle mode is set to the power consumption while executing no-operations at the lowest microprocessor clock frequency.

Figure 6.5 shows the performance of the on-demand, selfish, and thread-cooperation clock frequency assignment governors. Figure 6.5(a) shows the energy consumption of the different frequency schemes relative to the on-demand governor. A vertical dotted line is drawn around 2.8 W where the power consumption of the microprocessor running on-demand equals the background power consumption. The thread-cooperation governor performs the best, energy-wise. The selfish governor performs better than the on-demand frequency governor for a background power consumption smaller than about 1 W. After 1 W of background power consumption, the selfish governor and the on-demand governor consume the same amount of energy. For zero background power consumption the energy gains of the thread-cooperation and selfish frequency governor are about 40 % compared to the on-demand scheme. The energy savings for the thread-cooperation frequency governor decrease in an apparent logarithmic fashion. At the 2.8 W landmark the energy savings are 10 %, whereas the energy gains of the selfish frequency governor dropped to 0 %. For background power consumption larger than 6 W the energy savings drop below 5 %. Overall, the thread-cooperation frequency governor outperforms the selfish and on-demand frequency governors energy-wise.



(a) energy

(b) time

(c) frequency distribution: *thread-cooperation*

(d) frequency distribution: *selfish*

Figure 6.5: Energy consumption of multi-thread code executing on multiple cores. The performance of three frequency assignment governors is shown: *on-demand*, *selfish*, and *thread-cooperation*, with variable background power consumption. The vertical dotted line at 2.8 W indicates where the background power consumption P_{back} equals the power consumption of the microprocessor. The *on-demand* frequency governor is used as a reference in figure (a) and (b). The *selfish* governor outperforms w.r.t to energy, but *on-demand* is best time-wise.

Figure 6.5(b) shows the impact of the frequency governors on the execution time relative to the on-demand frequency governor. As the on-demand frequency is always running at the highest frequency possible, the on-demand governor is the lowest-bound of execution time. Both the thread-cooperation and selfish frequency governor are equal or induce an increased execution time compared to the lower-bound. The selfish frequency governor performs slightly better than the thread-cooperation and selfish frequency governor timewise. For a background power consumption of zero Watt, the execution time is twice as large compared to the on-demand frequency governor performance. The execution time discrepancy drops fairly fast to equal the on-demand frequency governor around 1 W to 2 W. For P_{back} levels below 2 W the thread-cooperation frequency governor is a trade-off between energy savings and prolonged execution time. Energy savings vary between 40 % and 10 % whereas the execution time is extended up to 100 %. For $P_{\text{back}} > 2\,\text{W}$ there is no execution time penalty, though energy gains are possible between 10 % and 3 %. Similar conclusions can be made for the selfish frequency governor.

Figure 6.5(c) and Figure 6.5(d) show the frequency assignment distribution for the thread-cooperation and selfish frequency governors at different levels of background power consumption levels. The on-demand governor always runs at the highest frequency and therefore its probability plot would be a pink plane, which is not very interesting to look at, unless you're a fan of contemporary art. The selfish frequency governors gradually ramps up the frequencies seemingly uniformly for all cores. All cores are assigned the maximum frequency for $P_{\text{back}} > 1$, which explains why the performance of the selfish and on-demand frequency governors become the same at that point. For the thread-cooperation frequency governor, the frequency assignments are more diverse. A great amount of threads are running at the slowest and greatest frequencies. The mid-frequencies are seemingly equally distributed. For $P_{\text{back}} > 2.8\,\text{W}$ the frequency distribution is more or less constant whereas for $P_{\text{back}} < 2.8\,\text{W}$ the smaller frequencies dominate. The ratio between the background and microprocessors' power consumption seems to be an indicator of the stability of the frequency distribution.

Halimi *et al.* [47] has studied a similar clock frequency assignment scheme that aims at minimizing energy consumption of parallel programs. The authors claim to be able to save up to 34 % of energy by accelerating and slowing down threads to minimize slack time among the threads. For the computation of the clock frequencies Halimi *et al.* assumed the energy/frequency convex model presented in this work.

6.4 Amdahl's Law Extension for Energy Efficiency

6.4.1 Extension of Amdahl's Law

In 1967 Gene Amdahl [4] provided an intuitive argument about performance gains, which was latter on casted into a mathematical formulation. Concretely, Amdahl's law predicts a theoretical upper-bound on the speedup of algorithmic or architectural enhancements, e.g., parallel computations for which it is most known to date. In a parallel context, Amdahl's law states that the non-parallel execution rapidly diminishes the performance scalability for parallel applications, irrespective of the number of parallel computations [130].

According to Amdahl's law the execution time $t(n)$ of a code running on n cores, with a proportion ϱ which can be parallelized at will follows:

$$t(n) = t(1)\, p \left((1 - \varrho) + \frac{\varrho}{n} \right), \tag{6.9}$$

where $n \in \mathbb{N}_0^+$, $0 \leq \varrho \leq 1$. $0 < p < \infty$ is added to Amdahl's law to addresses the time penalty incurred by clock frequency scaling. The *performance scaling factor*

$$\Lambda = \frac{1}{p\left((1-\varrho) + \frac{\varrho}{n}\right)} \tag{6.10}$$

is deemed the speedup of the parallelized code: $t(n) = t(1)/\Lambda$. Woo and Lee [130] have extended Amdahl's law to assess the *performance per Joule* given a certain percentage of parallelization. Performance per Joule is a metric for evaluating the performance achievable in a reference battery life cycle. In essence the performance per Joule is the reciprocal of the energy/delay product.

To model the power consumption of a multi-core microprocessor let us introduce the parameter k that represents the fraction of power consumed in idle mode. An active core consumes a power of P^+. When a microprocessor is executing the serial part of the code on one core, the other $(n-1)$ cores are in idle-mode and consume $kP^+(n-1)$. In total, the active core and the remaining cores consume $P^+ + kP^+(n-1) = P^+[1 + k(n-1)]$. During the parallel execution phase all cores are assumed active and consume nP^+ totally. The average power demand W of the microprocessor is then:

$$W = \frac{q\left(p(1-\varrho)P^+[1+(n-1)k] + p\frac{\varrho}{n}nP^+\right)}{t(1)\ p\left((1-\varrho) + \frac{\varrho}{n}\right)} = \frac{q\left(1+(n-1)k(1-\varrho)\right)}{(1-\varrho) + \frac{\varrho}{n}} \frac{P^+}{t(1)}, \tag{6.11}$$

where $0 < q < \infty$ is a scaling factor added to Woo and Lee's model to address the power consumption change as a result of dynamic frequency scaling. The performance per joule $\frac{\Lambda}{J}$ is then defined as:

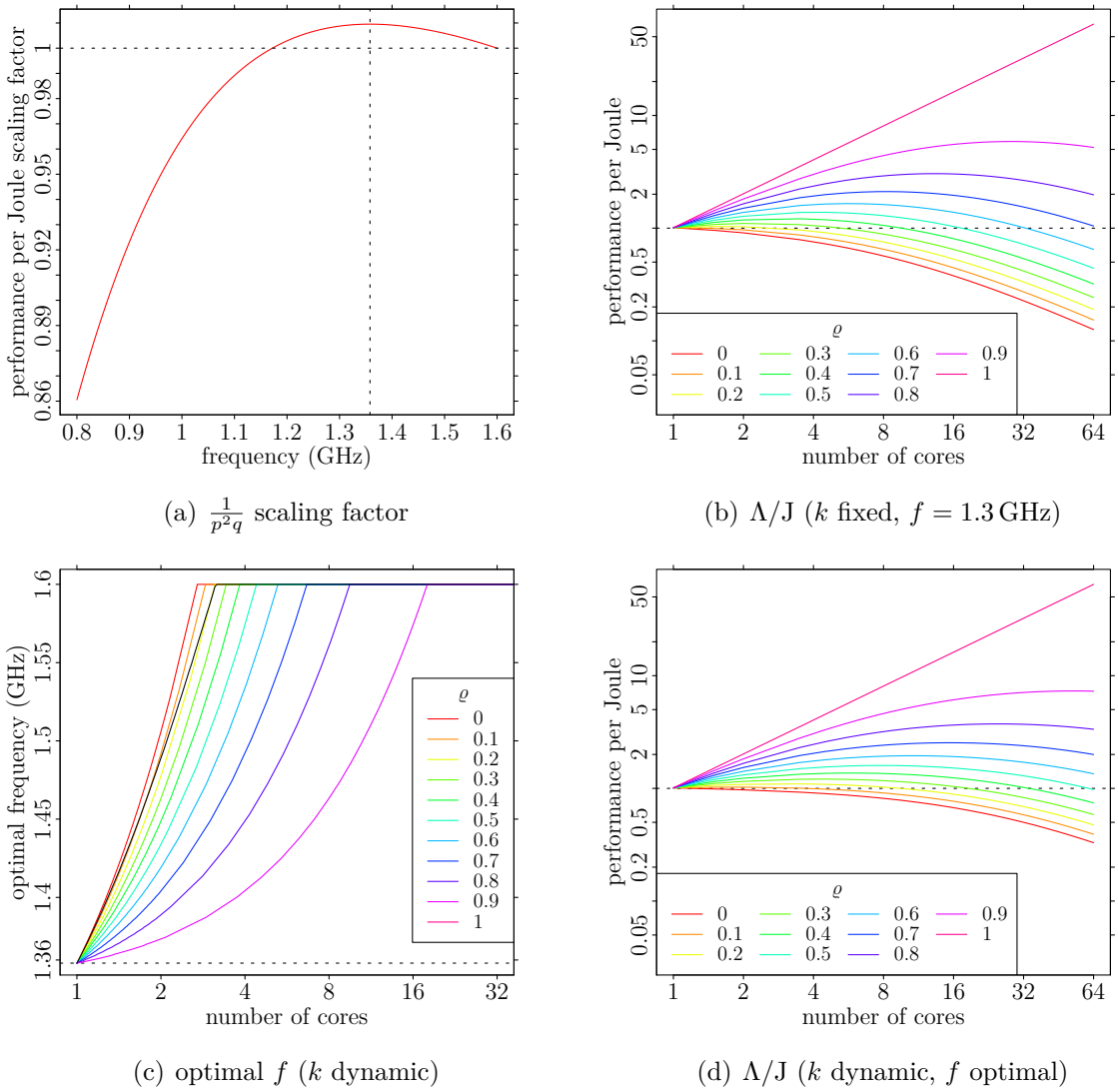
$$\frac{\Lambda}{J} = \frac{\Lambda}{W \times t} = \frac{1}{p^2 q} \cdot \frac{1}{P^+} \cdot \frac{1}{(1-\varrho) + \frac{\varrho}{n}} \cdot \frac{1}{1+(n-1)k(1-\varrho)}. \tag{6.12}$$

Now, k can be chosen to be the relative power consumption in idle-mode at an arbitrary frequency. If the microprocessor's clock frequency is dynamically scaled for idle-mode then k is a function of the microprocessor's frequency, just as p and q. If the clock frequency is static k is a constant. Then for a fixed ϱ and n the two last terms in Equation 6.12 are constant, and the fraction $\frac{1}{p^2 q}$ becomes a scaling factor. The power factor P^+ is itself a reciprocal scaling factor of the performance per joule.

The differences between the model of Equation 6.12 and that of Woe and Lee are the presence of the variables p and q, and here the system power is not normalized.

6.4.2 Experimentation

Let us plug in some realistic values for p, q, and k derived from the A15 model, sampled at 37°C, as resulting from Procedure 3.2. $k = 0.111$ and the values for p and q, relative to the microprocessor's performance at maximum frequency, are listed in Table 6.1. The Cortex A15 has only 4 cores; in this simulation however, up to 64 cores are added to observe how the performance is affected. Figure 6.6 shows the performance per Joule for varying number of cores n, degree of parallelization ϱ, and this for a fixed microprocessor frequency and Dynamic Voltage and Frequency Scaling (DVFS). P^+ is set to 1 such that the variables k, p and q can be at the center of attention. The effect of P^+ would be a vertical scaling of the performance per joule graphs.



(a) $\frac{1}{p^2 q}$ scaling factor

(b) Λ/J (k fixed, $f = 1.3\,\mathrm{GHz}$)

(c) optimal f (k dynamic)

(d) Λ/J (k dynamic, f optimal)

Figure 6.6: Performance per Joule (Λ/J), as an extension of Amdahl's law for energy efficiency, for a microprocessor which runs a portion of a code in parallel. The cases for fixed microprocessor frequency (k fixed) and different microprocessor frequency in active-mode and idle-mode (k dynamic) are addressed. For fixed k, there exists a global optimal frequency for all values of ϱ and n at 1.358 GHz, see Figure (a). For a dynamic k the optimal frequency is shown in Figure (c), where the idle-mode frequency is set to be the microprocessor's minimum frequency. The performance per Joule at the optimal frequency in both cases is shown in Figure (b) and (c). The black line in Figure (c) is the optimal frequency/number-of-cores pair. $P^+ = 1$, and values for p, q and k are derived from the Cortex A15 power model at 37°C in Procedure 3.2.

Table 6.1: Amdahl's law scaling factors: q is the execution time scaling factor ($f_{\mathrm{k}} = 0$ and $\beta = 0$); p is the frequency scaling factor; and k is the active power to idle power ratio. k is computed against the idle-mode at minimum frequency. For a static frequency $k = 0.111$. The scaling factors are relative to the microprocessor's maximum performance, i.e., at maximum clock frequency.

	Frequency								
	0.8	0.9	1	1.1	1.2	1.3	1.4	1.5	1.6
p	0.291	0.343	0.405	0.478	0.561	0.654	0.759	0.874	1.000
q	2.000	1.778	1.600	1.455	1.333	1.231	1.143	1.067	1.000
k	0.111	0.094	0.080	0.068	0.058	0.049	0.043	0.037	0.032

Figure 6.6(a) shows the performance per joule scaling factor. For a fixed degree of parallelization (ϱ) and number of parallel cores the scaling factor simulates the effect of running at a fixed frequency relative to the maximum microprocessor frequency. When a core is idle, its frequency is not altered. From the figure one can observe that the microprocessor frequency that yields the highest *performance per joule* in this case is 1.358 GHz. As in practice the frequency settings of the microprocessor are discrete, thus the optimal frequency will either be 1.3 GHz or 1.4 GHz. At the optimal frequency, the performance per Joule is 0.95 % larger than the reference performance, which is not that overwhelmingly impressive. The performance per Joule at 1.172 GHz is equal to the performance per Joule at 1.6 GHz. For slower frequencies the performance per Joule is below the unit. At 0.8 GHz the performance per Joule is 13.94 % less than at 1.6 GHz.

When the microprocessor gears to another frequency in idle-mode then k is not constant, and hence, the optimal frequency that maximizes the performance per Joule will not be constant. In the most extreme case where the executed code knows no serial part ($\varrho = 1$), the core of the microprocessor doesn't reside in idle state; thus the optimal frequency equals to the optimal frequency for the constant k case, i.e., 1.358 GHz. On the contrary, when $\varrho = 0$, i.e., no parallel code execution is performed, the optimal performance per Joule is evidentially at 1.6 GHz. The optimal frequency that maximizes the performance per energy varies between 1.358 GHz and 1.6 GHz for all values of ϱ and n. The optimal frequency increases for increasing number of cores and degree of parallelization. At around 16 cores the performance per Joule is maximized for $\varrho > 0.9$ when $f = 1.6$ GHz.

The performance per Joule for the constant k and dynamic k cases are shown in Figure 6.6(b) and Figure 6.6(d), respectively. Both graphs look similar: the lower the degree of parallelization, the more the curves deviate between both cases. In the most extreme case, for $\varrho = 1$, the curves are identical, as the system doesn't know any idle time. The performance per Joule decreases systematically for smaller degrees of parallelism. The performance per Joule drops faster for the case of fixed k; that is because more energy is wasted in idle-mode compared to the dynamic k case. The performance per Joule at $n = 8$ and $n = 64$ is 14 times and 515 times larger, respectively between $\varrho = 1$ and $\varrho = 0$ for k fixed. For the dynamic k case the performance per Joule at $n = 8$ and $n = 64$ is 10 times and 196 times larger, respectively, between $\varrho = 1$ and $\varrho = 0$. The performance per Joule for a dynamic k is 1.44 times larger at $n = 8$, and 2.63 times larger at $n = 64$, compared to the fixed k case. It is thus clear that a dynamic k has the upper hand over a fixed k in this particular scenario. Increasing the level of parallelism is also favorable for the performance per Joule. From these figures it is also understood that the performance

per Joule is maximized at a certain number of cores, which is different for each value of ϱ. The link between the optimal frequency and the optimal number of cores is depicted in Figure 6.6(c) with the solid black line. It can be seen that the optimal frequency increases fast from 1.358 GHz to 1.6 GHz. In fact, the optimal frequency/number-of-cores pair that is not at 1.6 GHz is for one and two number of cores. This implies that the maximum performance per Joule is nearly almost attained at maximum frequency.

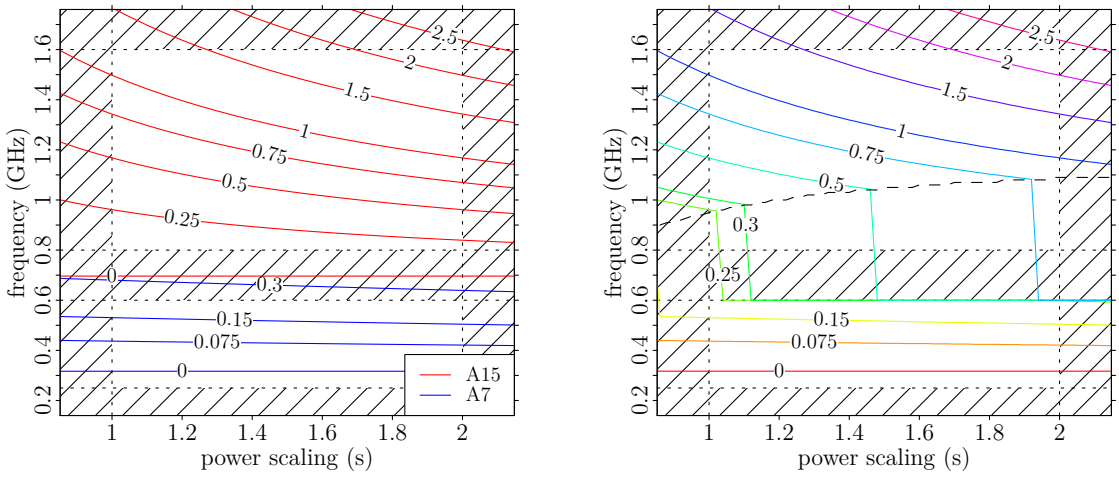
In both Figure 6.6(b) and Figure 6.6(d) the higher the degree of parallelization (ϱ), the better the performance per Joule. Parallelization can thus be seen as method to optimize energy efficiency. In fact, the microprocessor developed in the last decades were able to improve on performance and energy efficiency, not necessarily by scaling up the microprocessor's clock frequency, but rather by providing multi-core support. This is advantageous for multi-threaded applications, but not for single-threaded applications. In a practical situation, however, the parallelizability of an applications is inherent to its properties and abilities, and can only be exploited by an architecture to a limited extent. Therefore, the maximum performance per Joule is not always practically feasible.

6.5 big-LITTLE Heterogeneous Computing

The big-LITTLE computer architecture, as marketed by Samsung, is a heterogeneous computing concept that couples a low-power microprocessor (LITTLE) with a high-performance microprocessor (big). At the same time the LITTLE microprocessor has low performance while the big core is power-hungry. The OS can switch seamlessly between the big and LITTLE cores as seen fit for its context. The registers' states are automatically copied between microprocessors while switching microprocessors. SoCs implementing the big-LITTLE architecture normally activate the LITTLE core for low loads or when fast execution time is not of the essence. The big core is employed for performance-demanding tasks. The first commercially available big-LITTLE implementation, i.e., the default Exynos 5410, only allows for one of the two microprocessors to be active at the same time. Future implementation of the big-LITTLE architecture may allow for the two microprocessors to be active at the same time; this is both a hardware and a software challenge. Now the question arises; how does the energy/frequency convex minimum behave on such a big-LITTLE architecture?

The ODROID XU+E testbed sports an Exynos 5410 which implements the big-LITTLE architecture: an ARM Cortex A7 for the LITTLE microprocessor and an ARM Cortex A15 for the big microprocessor. The A7 runs at 0.25 GHz to 0.6 GHz in steps of 100 MHz. The A15 runs at 0.8 GHz to 1.6 GHz also in steps of 100 MHz. The power model from Table 4.4 is used to generate representative microprocessor power consumption curves. Figure 6.7 shows the optimal frequency for the two microprocessors running independently and coupled. Figure 6.7(a), for the two microprocessors running independently, is essentially the same as Figure 4.9 but differently represented. Figure 6.7(b) shows the optimal frequency if the microprocessors are coupled and enable seamless migration from one to the other.

It can be observed from Figure 6.7(a) that the optimal frequency for the low-power core is mostly exploitable for low levels of background power consumption below 0.3 Watt. The high-performance microprocessor is then more suitable for higher levels of background power consumption, up to 2 Watt. When the two microprocessors allow for seamless migration, then the microprocessor with the best energy characteristics may be chosen at any time. In Figure 6.7(b) it can be seen that the different curves jump between 0.9 GHz and 1.1 GHz on the A15 core to 0.6 GHz on the A7. There exists thus an operation point



(a) independent microprocessors

(b) coupled microprocessors

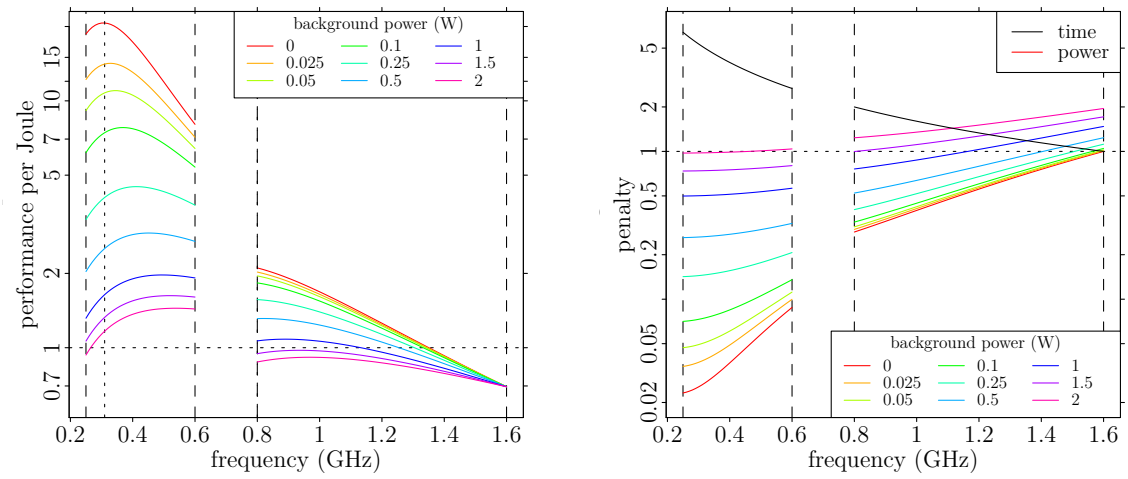
Figure 6.7: Optimal frequency for the Exynos 5410, a heterogeneous platform, featuring two microprocessors with different performance and power profiles: a low-power Cortex A7 and an high-performance Cortex A15. The A7 operates between 0.25 GHz and 0.6 GHz; the A15 is active between 0.8 GHz and 1.6 GHz. The optimal frequency for different levels of background power (Watt) is displayed as a label on the curves. Figure (a) shows the optimal frequency for the different cores, whereas Figure (b) shows the optimal frequency when seamless switching between microprocessors is enabled. The black dashed line in Figure (b) shows the border where the optimal frequency switches between microprocessors. For the execution time model $\beta = 0$ and $f_k = 0$ were assumed.

for the A15 where its energy consumption is equal to the A7, which is shown by the curved dashed line in Figure 6.7(b). From this point of view the performance dimensioning of the SoC was done reasonably well, as the point where the frequency that minimizes the energy consumption switches between cores is close to the minimum frequency of the A15 and maximum frequency of the A7.

Figure 6.7 assumed that the energy consumption is the only metric by which the performance of the microprocessor is measured. Let us introduce the concept of *performance per Joule* from Section 6.4, which also incorporates the time-wise performance, but then only for a single active application (ϱ=0). The time-wise and power-wise performances at the maximum frequency of the A15 microprocessor are taken as a reference. Figure 6.8(a) shows the performance per Joule at various levels of background power consumption. The figure implies that, at $s = 1.5$[1] and $P_{\text{back}} = 0$, the performance per Joule is about ten times higher for the A7 microprocessor (20.67 at 0.31 GHz) than for the A15 microprocessor (2.10 at 0.8 GHz) at the optimal frequency for each core individually. It is noted that the performance per Joule for $P_{\text{back}} = 0$ corresponds to the performance of the microprocessor itself, whereas if $P_{\text{back}} > 0$ the performance per Joule describes a computer system including the microprocessor and arbitrary other components. The ratio between the peaks of the A7 and A15 increases with s. But the ratio decreases close to 1.6 for increasing background power consumption levels, and doesn't drop below the unit.

Figure 6.8(b) shows the time and power penalty in function of the frequency for different levels of background power consumption relative to the A15 microprocessor's

[1]The parameter s was defined in Section 4.3.2 on page 69 as a single parameter that can scale the microprocessor's power consumption for different application energy profiles.



(a) performance per Joule

(b) time & power penalty

Figure 6.8: Performance per Joule of the Exynos 5410, a typical embedded system, for a single application running on a single core with various levels of background power consumption (Figure (a)). Figure (b) shows the scaling factors, or penalty, for power (color) and time (black) while running at different frequency and background power levels. The penalty is relative to the system running at the maximum frequency. The plots were generated with the following parameters: $s = 1.5$, $\beta = 0$ and $f_k = 0$, where s is a power scaling factor as shown in Figure 4.6.

best performance (highest frequency). As the frequency drops, the execution time becomes longer; therefore the time penalty is descending in function of the frequency and larger than the unit. The power penalty is an increasing function of frequency and smaller than the unit as the microprocessors consumes less power for lower frequencies. The maximum time penalty is 6.4 at 0.25 GHz. Execution time is not affected by the background power consumption; therefore, only one curve is depicted. The power is penalized more for larger levels of background power consumption. This is logical as the system consumes more power while performing the same computation, the execution time is unaffected, and thus deteriorating the performance per Joule. The maximum power penalty is 1.95 at 1.6 GHz for $P_{\text{back}} = 2$ Watt. For this level of P_{back} the power penalty drops just below the unit at the lowest frequency. Yet, the power penalty may go as low as 0.023 at 0.25 GHz when there is no background power consumption. It is the combined effect of the increasing power penalty and decreasing time penalty that creates the proper environment to conceive a convex energy/frequency behavior.

Overall, it can be stated that the designers of the Exynos 5410 did a great job dimensioning the SoC, as the performance, with regards to energy and execution time, is well balanced between the two microprocessors. The low-power core was shown to have indeed a superior performance per Joule; yet the high-performance core outperforms time-wise the low-power core with ease.

6.6 Conclusion

In this chapter the Energy/Frequency Convexity Rule which was previously expounded in Chapter 4 was applied. There, the convexity model was developed for a single core. In the first section it was shown that a system comprising of multiple cores or entities

with a proper clock frequency domain shows similar convexity properties. The different clock frequency domains should cooperate to minimize energy consumption. In fact, it was shown that the system's clock frequency assignment can be divided in discrete time blocks corresponding to the different combinations in energy states of the participating entities. The clock frequency of each of the entities should be optimized for each discrete time block to attain minimum energy consumption of the global system. This approach works for a system with arbitrary entities; yet the complexity of finding the optimal clock frequency increases with the number of participating entities as the convexity equation system grows accordingly.

Furthermore, the Energy/Frequency Convexity Rule was applied to a single core system subject to recurrent tasks with deadlines, which is common in real-time systems. It was indicated that in such a context the background consumption doesn't affect the optimal frequency, but does affect the energy consumption of the system. It was shown in Figure 6.3(c) with a practical application that, in the best case, the microprocessor can save up to 55 % of energy. This case corresponded to a task with a fairly small clock cycle demand. For tasks that require a lot of computation before the deadline this energy gain may shrink to 0 %. For large tasks the race-to-halt technique can be most effective, but for smaller tasks a more refined clock scheduling, following the Energy/Frequency Convexity Rule, may yield better energy consumption efficiency. These findings may imply that it could be beneficial to over-dimension a system and carefully set its configuration to minimize energy consumption.

A clock frequency assignment based on the Energy/Frequency Convexity Rule, stimulating *thread-cooperation*, for a multi-threaded code execution was also examined. The performance of three clock frequency assignment schemes was assessed, with the Linux-like *on-demand* frequency governor as a reference. Figure 6.5(a) showed that the frequency governor based on the Energy/Frequency Convexity Rule outperformed the other frequency governors energy-wise; the microprocessor could save up to 40 % of energy in the best case, and around 10 % when the background power consumption equals the microprocessor's power consumption. However, the time-wise performance of the Energy/Frequency Convexity Rule-based frequency governor lagged behind on the reference for a maximum of 20 %.

Woo and Lee's Amdahl law was also extended for energy efficiency and frequency scaling. The *performance per Joule* metric was defined as a function of multiple parameters, including the system's power consumption, voltage and frequency scaling constants and a parameter defining the degree of parallelization. It was shown in Figure 6.6(a) that for the Exynos 5410 SoC testbed the voltage and frequency scaling factor $1/(p^2q)$ had an optimal performance around 1.3 GHz. As expected, it was shown that the performance per Joule increases for higher degrees of parallelization. Moreover, there are an optimal level number of threads and optimal frequency which varies for the degree of parallelization.

Finally, a heterogeneous architecture of a coupled low-power and high-performance microprocessor was studied. It was shown that the optimal frequency for the coupled microprocessors was well balanced with the aim of minimizing energy consumption for different loads of the microprocessor. Via the performance per Joule metric it was also demonstrated that for the Exynos 5410 indeed the low-power core provides a superior performance per Joule, while the high-performance cores is more power hungry but offers better performance time-wise. The performance per Joule of the low-power core can be up to twenty times better then for the high performance core, while the low-power core is maximum six times slower than the high performance core.

Chapter 7

Conclusion

7.1 Final Remarks and Results

In this work the Energy/Frequency Convexity Rule was developed and supported with experimental data. The thermosensitivity of the Energy/Frequency Convexity Rule and microprocessors' power profile was also discussed. Furthermore, methods to improve the accuracy energy and power measurements were presented.

The Energy/Frequency Convexity Rule applies to a computer system whose energy consumption can be described by Equation 4.2. If its first derivative has a single root and its second derivative is a monotonous increasing function (see Equation 4.11), then the system's energy consumption has a global minimum at a given (optimal) clock frequency f_{opt}. This optimal clock frequency yields the smallest energy consumption for the execution of an instruction sequence. If the optimal clock frequency is within the microprocessor's clock frequency range, it can be exploited. If not, the microprocessor is most energy-efficient at a boundary clock frequency, either at the maximum or minimum clock frequency. The energy consumption is a function of multiple parameters: the microprocessor's power profile (ξ), voltage/frequency scaling capabilities (f/V), background consumption (P_{back}), slack time (β), clock cycle thieves (f_{k}), temperature T, and some others.

In Chapter 4 a parameter sensitivity analysis of the Energy/Frequency Convexity Rule was carried out. The background consumption turned out to be the parameter with the largest influence on the optimal clock frequency, minimizing the system-wide energy consumption. As a rule of thumb, f_{opt} is exploitable, i.e., $f_{\text{opt}} < f_{\max}$, if the microprocessor's power requirements are smaller than P_{back}. For some types of applications, however, f_{opt} is independent of P_{back}, for example for computer systems processing repetitive jobs as discussed in Section 6.2. The number of clock cycle thieves also affects f_{opt} significantly in the sense that f_{opt} increases the fewer clock cycles are available for computation. The microprocessor's power profile was shown to have a minimal effect on f_{opt}. The number of executed instructions has no influence on the optimal clock frequency, which is a direct result of the energy consumption formulation of Equation 4.2. It was also indicated that the *race-to-halt* energy optimization technique is only effective when $f_{\text{opt}} > f_{\max}$.

Chapter 6 applied the Energy/Frequency Convexity Rule to practical applications. It was shown how the Energy/Frequency Convexity Rule could be applied to a computer system that includes multiple entities, or buddies, with independent clock frequencies. Energy can be minimized system-wide by dividing the system into time frames such that the energy states of the buddies do not change within a single time frame. Then the

optimal frequency for each buddy within each time frame is calculated. It was shown that in the case of out-of-order execution (OOE), f_{opt} can become complicated as the buddies' clock frequencies cannot be scaled independently during OOE. An application of a four-buddy system was presented in Section 6.3 where the clock frequency of four threads were scaled independently and cooperatively to attain a system-wide energy consumption minimum. Compared to the default Linux-like clock frequency governor in the use case, a cooperative clock frequency scaling algorithm between the multiple threads showed that a maximum energy of 40 % could be saved under the most favorable conditions. The energy gain shrinks to 10 % when the background components of the system require about the same power as the microprocessor. This energy gain is however a trade-off with execution time, which is about two times larger at 40 % energy gain. Amdahl's law was also extended to incorporate clock frequency scaling, from which the *performance-per-Joule* metric was derived based on previous research. This metric was applied to the Exynos 5410 SoC, which houses a low-power (A7) and a high performance (A15) microprocessors. The performance-per-Joule metric at the most optimal configuration was shown to be ten times higher for the A7 than for the A15, whereas the execution time, i.e., the trade-off with performance-per-joule, increases only about 5 times.

These observations on the Energy/Frequency Convexity Rule and their applications were based on a theoretical framework and backed up by experimental data. To obtain the experimental data, execution time profiles were derived from the Bristol Energy Efficiency Benchmark Suite (BEEBS) and the Gold-Rader bit-reverse algorithm, which were ran on two platforms consisting of flag ship[1] multimedia SoCs testbeds. The power profiles used were recorded on the same SoC testbed of which one implements a Cortex A9, and the other a dual microprocessor including a Cortex A7 and A15. Power models for the A7 and A15 processors were provided in Section 3.6.2 which can be directly used in simulations. The neat thing about these power models is that the microprocessor's temperature is an input parameter. In Chapter 3 the temperature dependency of the microprocessor's power requirements was discussed. Amongst other physical properties, the leakage currents are most likely the greatest contributors to the temperature dependency of the microprocessor's power profile, which was extensively discussed in Section 3.2.1. It was shown via empirical data of the A15 processor that, in the most extreme case, the power requirements at 85°C were 20 % larger than at 25°C. An exponential-based model was shown to best fit the temperature/power relationship over the 20°C to 85°C temperature range. In a more pertinent temperature range, between 20°C and 50°C, linear and quadratic approximations are acceptable. This temperature/power relationship was used to demonstrate how to remove the temperature bias from a power measurement trace. It was also stressed and illustrated that the distance of a temperature sensor from a heat source may introduce power measurement errors, including time lag and diminished magnitude. A workable but rather not-so-elegant transformation function was constructed to cancel such behavior.

The temperature/power relationship influences the transient thermal behavior of a microprocessor, as its internal heat generation is dependent on the temperature. The transient thermal behavior is also dictated by how the microprocessor dissipates its heat to the environment. Actively cooled systems release their heat to the environment via forced convection, of e.g., air or other types of fluid, which dominates the heat transfer modes. The transient behavior of such systems is well described by an exponential cooling law. Passively cooled devices, however, rely on natural dissipation of heat, in-

[1] Flag ship SoCs at the time of their commercial release.

cluding radiation. Chapter 5 explored the effects of the presence of radiative cooling on transient thermal behavior via an exact analytical framework. A use case applied to a microprocessor-like object showed that the radiation component cannot be neglected for objects with a cooling surface area larger than $1\,\mathrm{dm}^2$, which is a representative cooling surface area for a smartphone. For smaller cooling surface areas the exact passive cooling law approximates an exponential cooling law. Under such conditions an exponential cooling law is favored as it is much less complex than the exact passive cooling law. For when the exact passive cooling is desired, approximations to the exact cooling law were also provided in Section 5.3.6, designed to be used in practical applications. The *coefficient approximation* was shown to perform best under various circumstances. For small departures from the ambient temperature the *second-order O'Sullivan approximation* performed satisfyingly as well.

7.2 Future Work

Although no explicit open questions were stated in the presented research, there are several lines of possible inquiry arising from this work which could be interesting to pursue.

The accuracy of the protocol to measure the temperature/power relationship can be improved, mainly by employing more accurate temperature sensors. This is, however, a challenging task as on-die temperature sensors have poor resolution and may have a temperature-dependent error. External temperature measurement devices, such as IR-based sensors, can provide a descent accuracy but they are to be used in a controlled environment and preferably with the die exposed. The ambient temperature and the die temperature should also be controlled properly to avoid the effects of temperature hysteresis loops interfering with the temperature/power correlation.

From a theoretical point of view, it is interesting to assess the exact impact factor of temperature-dependent processes, besides the leakage current, that affect the power profile of microprocessors. With such knowledge a microprocessor temperature/power model could be constructed, based on physical principles. This is in contrast with the heuristic temperature/power model that was presented before. Also, the transformation model to combat the *distant-sensor-syndrome* could be derived from a more sound theoretical foundation, instead of an approximate polynomial. Though, as shown, such endeavor will likely yield a very complex mathematical formulation which could be intractable for practical applications.

From the power measurement point of view, more accurate power measurements will have also benefits for the temperature/power model. Higher sampling rates could allow to look in more detail to specific parts of instruction sequences. This leads to a finer grained estimation of ξ. It could be beneficial to obtain the ξ information of the microprocessor's functional units, compared to the application-level estimation of ξ that was assumed in this thesis. Yet, higher sampling rates come along with a larger price tag and larger post-measurement data processing requirements. Here it was already sometimes tedious to process 4 kHz power measurement traces. High-end data acquisition tools may have sampling rates from 100 kHz to 1 GHz; enough disk space and patience are hence advised to analyze extensive power traces.

More accurate ξ profiles can then also lead to more aggressive DVFS schemes, which would yield better energy gains than the current interactive on/off frequency governors as seen in prominent Linux distributions, ideally dynamic DVFS schemes with energy savings comparable to what hard-coded DVFS can achieve. ξ and execution time profiles could

also be embedded into the code as directives for dynamic energy optimization. It could be difficult for current hardware architectures to extend their binary code translation to support energy and time profiles. However, when embedded in byte code, (process) Virtual Machines (VMs) can decode such information and redirect the data to an energy management unit. Even energy profile information of binary code, which is called via a native interface from a VM, can be passed to, and be useful for, energy management units. As shown in this thesis, maximum energy savings are achieved by application cooperation, not necessarily by individual application optimization. An energy management unit receiving ξ parameters from all active VMs could thus optimize the system in a cooperative manner, via the presented *buddy* rationale and DVFS for example. Basic application cooperation already exists in Android via the so called *wakelocks*, which manage system resources energy expenditure, e.g., display and sensors.

The Energy/Frequency Convexity Rule was investigated from a theoretical point of view without the notion of human sentiment. For HPC systems that are expected to produce results as swiftly as possible, human sentiments are of no value. Devices designed for human-computer interaction (HCI), such as smartphones, tablets, or phablets, assert the need for a human emotional aspect into the trade-off of the Energy/Frequency Convexity Rule. As presented in this work, the Energy/Frequency Convexity Rule is a trade-off between energy consumption on one side, and execution time on the other side. If HCI is involved, the user experience will be affected by a change in execution time. When execution time increases, the computer system becomes less reactive and the user experience will deteriorate, in a non-linear fashion. Understanding the emotional impact of the Energy/Frequency Convexity Rule should thus be a key factor when designing a system optimized for human experience. The user's experience can then be used as an additional constraint when defining the optimal clock frequency. This constraint would put a lower bound on the clock frequency that indicates the point at which the user doesn't tolerate any slow down of the system.

Appendix A

Mathematical Fundamentals and Derivations

A.1 Derivations

A.1.1 Deviating Root Configurations

In Section 5.3.1 the following differential equation was solved

$$\frac{dT}{dt} = -\kappa_4 T^4 + \kappa_3 T^3 + \kappa_2 T^2 + \kappa_1 T + \kappa_0, \tag{A.1}$$

where $\kappa_{0,1,2,3} \in \mathbb{R}^+$ and $\kappa_4 \in \mathbb{R}_0^+$. The polynomial on the right-hand side was assumed to have two real roots and two complex conjugate roots. But what happens if there are four imaginary roots, i.e., two times two complex conjugate roots? And what about four real roots? The following two sections provide a guide to these solutions.

The Real Deal

Let us assume that the polynomial on the right hand side of Equation A.1 has exactly four real roots. Then as before, the polynomial can be split via partial fraction decomposition:

$$\frac{1}{(T-\omega_1)(T-\omega_2)(T-\omega_3)(T-\omega_4)} = \frac{A}{(T-\omega_1)} + \frac{B}{(T-\omega_2)} + \frac{C}{(T-\omega_3)} + \frac{D}{(T-\omega_4)},$$

where A, B, C and D are constants and ω_* are the real roots of the polynomial. The values of ω_* can be obtained as before by Ferarri's theorem (see Section A.2), and the constants A, B, C and D can be found by solving the following system of equations:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -(\omega_2+\omega_3+\omega_4) & -(\omega_1+\omega_3+\omega_4) & -(\omega_1+\omega_2+\omega_4) & -(\omega_1+\omega_2+\omega_3) \\ \omega_2\omega_3+\omega_2\omega_4 & \omega_1\omega_3+\omega_1\omega_4 & \omega_1\omega_2+\omega_1\omega_4 & \omega_1\omega_2+\omega_1\omega_3 \\ +\omega_3\omega_4 & +\omega_3\omega_4 & +\omega_2\omega_4 & +\omega_2\omega_3 \\ -\omega_2\omega_3\omega_4 & -\omega_1\omega_3\omega_4 & -\omega_1\omega_2\omega_4 & -\omega_1\omega_2\omega_3 \end{bmatrix} \times \begin{bmatrix} A \\ B \\ C \\ \\ D \end{bmatrix}.$$

Then the solution to Equation A.1 with four real roots takes a simpler form as before:

$$t = -\frac{1}{\kappa_4}(A \ln|T-\omega_1| + B \ln|T-\omega_2| + C \ln|T-\omega_3| + D \ln|T-\omega_4| + c_o), \tag{A.2}$$

where c_0 is a constant such that $T(0) = T_0$.

The Case for Four Imaginary Roots

Let us assume that the polynomial on the right hand side of Equation A.1 has exactly four imaginary roots. Then as before, the polynomial can be split via partial fractions decomposition:

$$\frac{1}{(T-\omega_1)(T-\omega_2)(T-\omega_3)(T-\omega_4)} = \frac{AT+B}{((T-\alpha_1)^2+\beta_1)} + \frac{CT+D}{((T-\alpha_2)^2+\beta_2)},$$

where A, B, C and D are constants and ω_* are the complex roots of the polynomial, and $\alpha_1 = \Re(\omega_1) = \Re(\omega_2)$, $\alpha_2 = \Re(\omega_3) = \Re(\omega_4)$, $\beta_1 = \Im(\omega_1) = -\Im(\omega_2)$ and $\beta_2 = \Im(\omega_3) = -\Im(\omega_4)$. The values of ω_* can be obtained as before by Ferarri's theorem (see Section A.2), and the constants A, B, C and D can be found by solving the following system of equations:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ \alpha_1^2+\beta_2 & 2\alpha_2 & \alpha_1^2+\beta_1 & -2\alpha_1 \\ -2\alpha_2 & 1 & -2\alpha_1 & 1 \\ 0 & \alpha_2^2+\beta_2 & 0 & \alpha_1^2+\beta_1 \end{bmatrix} \times \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix}.$$

Then the solution to Equation A.1 with four imaginary roots is given by:

$$t = -\frac{1}{\kappa_4}\left(\frac{A}{2}\ln|(T-\alpha_1)^2+\beta_1^2| + \frac{\alpha_1 A - B}{\beta_1}\arctan\left(\frac{T-\alpha_1}{\beta_2}\right) + \frac{C}{2}\ln|(T-\alpha_2)^2+\beta_2^2| + \frac{\alpha_2 C - D}{\beta_2}\arctan\left(\frac{T-\alpha_2}{\beta_2}\right) + c_o\right), \quad \text{(A.3)}$$

where c_0 is a constant such that $T(0) = T_0$.

A.1.2 Equality of the Second-order O'Sullivan Approximations

Via different derivation methodologies, what appears to be, two different solutions were obtained for the second-order O'Sullivan approximation in Section 5.3.6, namely Equation 5.33 and Equation 5.36. In fact, these two solutions are equal:

$$\frac{\omega_1 - \omega_2 c_o e^{-\frac{m}{AC}t}}{1 + c_o e^{-\frac{m}{AC}t}} + T_a = \frac{w}{2m}\tanh\left(\frac{w}{2C}t + c_o\right) - \frac{n}{2m} + T_a.$$

Let us focus on the heating process to prove the equality of these two formulations, the derivation for the cooling process is similar.

Let's begin with substituting the `tanh` with its exponential representation:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{1 - e^{-2x}}{1 + e^{-2x}},$$

then Equation 5.36 becomes:

$$T = \frac{w}{2m}\frac{1 - e^{-2\left(\frac{w}{2C}t + c_o\right)}}{1 + e^{-2\left(\frac{w}{2C}t + c_o\right)}} - \frac{n}{2m} + T_a.$$

Isolating and replacing $c_o = \text{atanh}\frac{2m\theta_0+n}{w}$ yields:

$$\begin{aligned} T &= \frac{w}{2m}\frac{1-e^{-\frac{w}{C}t}e^{-2c_o}}{1+e^{-\frac{w}{C}t}e^{-2c_o}} - \frac{n}{2m} + T_a \\ T &= \frac{w}{2m}\frac{1-e^{-\frac{w}{C}t}e^{-2\text{atanh}\left(\frac{2m\theta_0+n}{w}\right)}}{1+e^{-\frac{w}{C}t}e^{-2\text{atanh}\left(\frac{2m\theta_0+n}{w}\right)}} - \frac{n}{2m} + T_a. \end{aligned}$$

The `arctanh` can be replaced with its logarithmic formulation:

$$\text{atanh}(x) = \frac{1}{2}\ln\left(\frac{1+x}{1-x}\right) \quad \text{for} \quad |x| < 1,$$

and continuing with the derivation:

$$\begin{aligned} T &= \frac{w}{2m}\frac{1-e^{-\frac{w}{C}t}e^{-2\frac{1}{2}\ln\left(\frac{1+\frac{2m\theta_0+n}{\sqrt{n^2-4mp}}}{1-\frac{2m\theta_0+n}{\sqrt{n^2-4mp}}}\right)}}{1+e^{-\frac{w}{C}t}e^{-2\frac{1}{2}\ln\left(\frac{1+\frac{2m\theta_0+n}{\sqrt{n^2-4mp}}}{1-\frac{2m\theta_0+n}{\sqrt{n^2-4mp}}}\right)}} - \frac{n}{2m} + T_a \\ &= \frac{w}{2m}\frac{1-e^{-\frac{w}{C}t}\left(\frac{1-\frac{2m\theta_0+n}{\sqrt{n^2-4mp}}}{1+\frac{2m\theta_0+n}{\sqrt{n^2-4mp}}}\right)}{1+e^{-\frac{w}{C}t}\left(\frac{1-\frac{2m\theta_0+n}{\sqrt{n^2-4mp}}}{1+\frac{2m\theta_0+n}{\sqrt{n^2-4mp}}}\right)} - \frac{n}{2m} + T_a \\ &= \frac{w}{2m}\frac{1-e^{-\frac{w}{C}t}\left(\frac{w-2m\theta_0-n}{w+2m\theta_0+n}\right)}{1+e^{-\frac{w}{C}t}\left(\frac{w-2m\theta_0-n}{w+2m\theta_0+n}\right)} - \frac{n}{2m} + T_a. \end{aligned} \tag{A.4}$$

Now, the term between the large brackets is the inverted integration constant c_o (Equation 5.34) from Equation 5.33:

$$\frac{w-2m\theta_0-n}{w+2m\theta_0+n} = \frac{\frac{w-n}{2m}-\theta_0}{\frac{w+n}{2m}+\theta_0} = \frac{\omega_2-\theta_0}{\omega_1+\theta_0} = \frac{1}{c_0}, \tag{A.5}$$

and $w = -\frac{m}{A}$ as can be shown:

$$\frac{m}{A} = m(\omega_2-\omega_1) = \left(\frac{-n+w}{2} + \frac{n-w}{2}\right) = -w. \tag{A.6}$$

Substituting Equation A.6 and A.5 in Equation A.4 gives

$$\begin{aligned} T &= \frac{w}{2m}\frac{1+\frac{1}{c_o}e^{\frac{m}{AC}t}}{1-\frac{1}{c_o}e^{\frac{m}{AC}t}} - \frac{n}{2m} + T_a \\ &= \frac{w}{2m}\frac{c_o e^{-\frac{m}{AC}t}-1}{c_o e^{-\frac{m}{AC}t}+1} - \frac{n}{2m} + T_a \\ &= \frac{w\frac{c_o e^{-\frac{m}{AC}t}-1}{c_o e^{-\frac{m}{AC}t}+1} - n\frac{c_o e^{-\frac{m}{AC}t}+1}{c_o e^{-\frac{m}{AC}t}+1}}{2m} + T_a \\ &= \frac{w(c_o e^{-\frac{m}{AC}t}-1) - n(c_o e^{-\frac{m}{AC}t}+1)}{2m(c_o e^{-\frac{m}{AC}t}+1)} + T_a \\ &= \frac{-w-n}{2m}\frac{1}{c_o e^{-\frac{m}{AC}t}+1} + \frac{w-n}{2m}\frac{c_o e^{-\frac{m}{AC}t}}{c_o e^{-\frac{m}{AC}t}+1} + T_a \\ &= \frac{\omega_1-\omega_2 c_o e^{-\frac{m}{AC}t}}{1+c_o e^{-\frac{m}{AC}t}} + T_a. \end{aligned} \tag{A.7}$$

The later equation is equal to Equation 5.33, and so it is proven that Equation 5.33 and 5.36 are, in fact, the same!

A.2 Ferarri's Theorem

In the 15th century Lodovico Ferrari developed an algebraic algorithm to find the roots of a *quatric equation*, or simply a fourth order polynomial. Given the fourth order polynomial of the form

$$\kappa_4 x^4 + \kappa_3 x^3 + \kappa_2 x^2 + \kappa_1 x + \kappa_0 = 0, \tag{A.8}$$

where $\kappa_4 \in \mathbb{R}_0^+$ and $\kappa_{0,1,2,3} \in \mathbb{R}^+$, the four roots $\omega_{1,2,3,4}$ can be obtained as follows.

Let's define first

$$y = x + (b/4), \tag{A.9}$$

then for aesthetical motivations $\kappa_4, \kappa_3, \kappa_2, \kappa_1, \kappa_0$ are defined as a, b, c, d, e, respectively, now the variables α, β, and γ are introduced

$$\begin{aligned}
\alpha &= -\frac{3b^2}{8a^2} + \frac{c}{a} \\
\beta &= \frac{b^3}{8a^3} - \frac{bc}{2a^2} + \frac{d}{a} \\
\gamma &= -\frac{3b^4}{256a^4} + \frac{cb^2}{16a^3} - \frac{bd}{4a^2} + \frac{e}{a}.
\end{aligned}$$

If $\beta = 0$ then the roots can immediately be computed as follows

$$\omega = -\frac{b}{4a} \pm \sqrt{\frac{-\alpha \pm \sqrt{\alpha^2 - 4\gamma}}{2}}, \tag{A.10}$$

otherwise one continues:

$$\begin{aligned}
P &= -\frac{\alpha^2}{12} - \gamma \\
Q &= -\frac{\alpha^3}{108} + \frac{\alpha\gamma}{3} - \frac{\beta^2}{8}.
\end{aligned}$$

$$\begin{aligned}
R_p &= \frac{Q}{2} + \sqrt{Q^{1/2} + P^{1/9}} \\
R_m &= \frac{Q}{2} - \sqrt{Q^{1/2} + P^{1/9}} \\
U &= R_m^{1/3}
\end{aligned}$$

$$\begin{aligned}
\text{for } U = 0: \quad y &= -\frac{5}{6}\alpha - U \\
\text{for } U \neq 0: \quad y &= -\frac{5}{6}\alpha - U + \frac{P}{3U}
\end{aligned}$$

Then the four roots of Equation A.8 can be calculated as follows

$$\omega = -\frac{b}{4a} \pm \frac{1}{2}\left(\sqrt{\alpha + 2y} \pm \sqrt{-\left(3\alpha + 2y + \frac{2\beta}{\sqrt{\alpha + 2y}}\right)}\right)$$

Given the units for the κ values presented in Equation 5.8, the unit of ω can be verified. Let us start with α, β, and γ:

$$\begin{aligned}\alpha &= \frac{K^6 s}{K^4 s} + \frac{K^3 s}{K s} = K^2 \\ \beta &= \frac{K^9 s}{K^6 s} + \frac{K^6 s^2}{K^2 s \cdot K s} + \frac{K^3 s}{s} = K^3 \\ \gamma &= \frac{K^{12} s^4}{K^8 s^4} + \frac{K^9 s^3}{K s \cdot K^4 s^2} + \frac{K^6 s^2}{K^2 s \cdot s} + \frac{K \cdot K^3 s}{s} = K^4,\end{aligned}$$

then the unit of ω is

$$\omega = \frac{K^3 s}{K^2 s} + \sqrt{K^2 + \sqrt{K^4 + K^4}} = K. \tag{A.11}$$

A.3 Statistical Estimator Error

In statistics the divergence of an estimator $\hat{\theta}$ from a true value θ, also referred to as the *error*, can be expressed in several ways.

The *absolute error* is given by

$$|\hat{\theta} - \theta|. \tag{A.12}$$

The *relative error* is given by

$$\frac{|\hat{\theta} - \theta|}{\theta}. \tag{A.13}$$

The *mean squared error (MSE)* over a set of n samples is given by

$$\frac{1}{n}\sum_{i=0}^{n}(\hat{\theta}_i - \theta_i)^2. \tag{A.14}$$

The *root-mean-square error (RMSE)* over a set of n samples given by

$$\sqrt{\frac{1}{n}\sum_{i=0}^{n}(\hat{\theta}_i - \theta_i)^2}. \tag{A.15}$$

A.4 Functions

A.4.1 Heaviside function

The *heaviside function* $\theta(x)$, also known as a *step function*, in one variable x is defined as

$$\theta(x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x > 0. \end{cases} \tag{A.16}$$

A.4.2 Sign Function

The *sign function* $\Phi(x)$ has a tertiary output depending on its input:

$$\Phi(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{if } x = 0, \\ -1 & \text{if } x < 0. \end{cases} \tag{A.17}$$

A.4.3 Dirac's Delta function

Dirac's delta function δ is defined by the following property

$$\delta(x) = \begin{cases} 0 & \text{if } x \neq 0, \\ \infty & \text{if } x = 0. \end{cases} \tag{A.18}$$

additionally

$$\int_{-\infty}^{\infty} \delta(x)\ dx = 1. \tag{A.19}$$

In fact Dirac's delta function is not a function by definition.

Appendix B

Source Code Excerpts

B.1 Implementation of the Passive Cooling Law

B.1.1 Exact Solution

A prototype implementation of the passive cooling law, as per Equation 5.17, is presented for R. The function `passive_cooling_coefs` computes the coefficients of the passive cooling law given a temperature T_0 at t=0. a, b, c, d and e are the coefficients of the polynomial on the right hand side of Equation 5.8, i.e., $\kappa_{4,3,2,1,0}$ respectively. The function `passive_cooling` computes the time at which the system attains the temperature T, with the coefficients computed by `passive_cooling_coefs` as a parameter.

```
passive_cooling_coefs <- function(a,b,c,d,e,T0) {
  my_roots <- polyroot(c(e/a,d/a,c/a,b/a,1))
  my_complex <- apply(as.matrix(my_roots), 2, function(x) {return(abs(Im(x)) > 1e-3)})
  my_real <- which(my_complex == FALSE)
  my_imag <- which(my_complex == TRUE)

  omega <- Re(my_roots[my_real])[order(Re(my_roots[my_real]))]
  alpha <- Re(my_roots[my_imag][1])
  beta <- Im(my_roots[my_imag][1])

  A <-  1/((omega[1]-omega[2])*((alpha^2+beta^2)-omega[1]*(alpha*2-omega[1])))
  B <- -1/((omega[1]-omega[2])*((alpha^2+beta^2)-omega[2]*(alpha*2-omega[2])))
  C <- -(A+B)
  D <- A*(alpha*2-omega[1])+B*(alpha*2-omega[2])

  co <- -(A*log(abs(T0-omega[1]))
          +B*log(abs(T0-omega[2]))
          +C/2*log(abs((T0-alpha)^2+beta^2))
          +(alpha*C+D)/beta*atan((T0-alpha)/beta))

  output <- rbind(c(a,b,c,d,e,A,B,C,D,omega[1],omega[2],alpha,beta,co,abs(omega[2])))
  colnames(output) <- c("a","b","c","d","e","A","B","C","D",
                        "w1","w2","alpha","beta","co","Te")
  return(output)
}

passive_cooling <- function(coefs, T) {
  a <- coefs[,"a"]
  A <- coefs[,"A"]
  B <- coefs[,"B"]
```

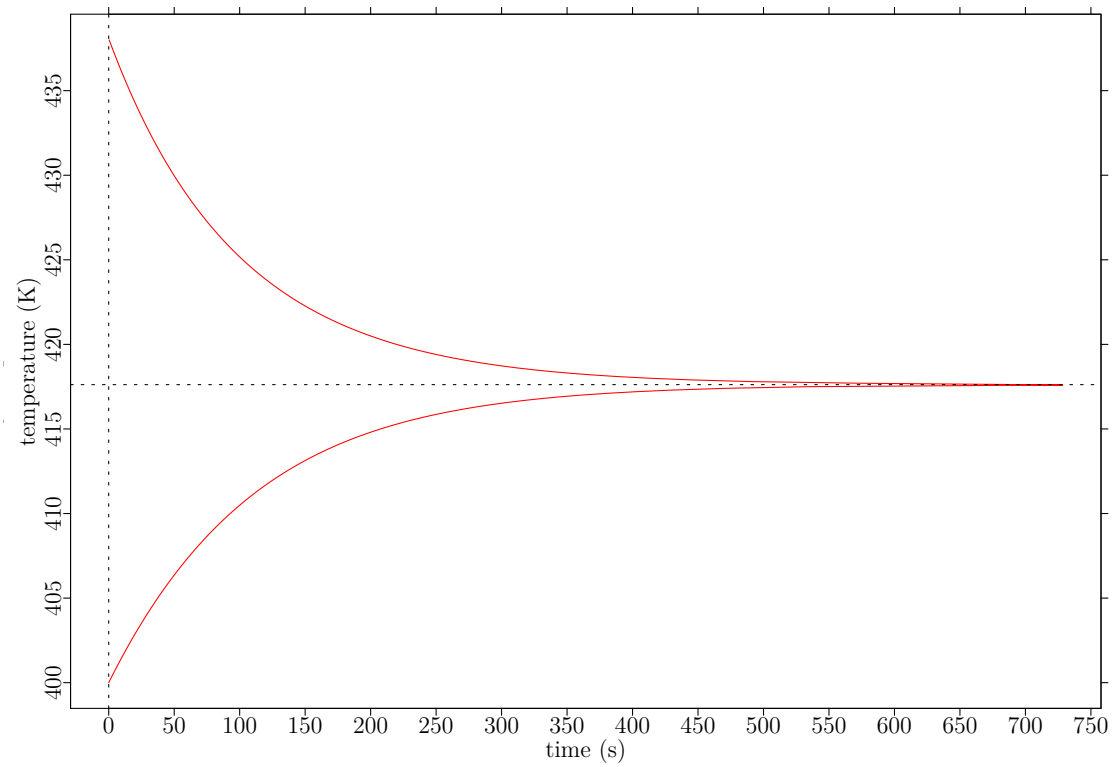


Figure B.1: Illustration of the prototype implementation of $f(t) = T$ in `R` for $a = -3.56 \times 10^{-11}$, $b = c = 0$, $d = 9 \times 10^{-4}$, $e = 7.07 \times 10^{-1}$, $T_0 = 400$.

```
  C <- coefs[,"C"]
  D <- coefs[,"D"]
  omega <- c(coefs[,"w1"],coefs[,"w2"])
  alpha <- coefs[,"alpha"]
  beta <- coefs[,"beta"]
  co <- coefs[,"co"]

  return(1/a*(A*log(abs(T-omega[1]))
              +B*log(abs(T-omega[2]))
              +C/2*log(abs((T-alpha)^2+beta^2))
              +(alpha*C+D)/beta*atan((T-alpha)/beta)
              +co))
}
```

A realistic example of the passive cooling goes as follows:

```
my_coefs       <- passive_cooling_coefs(-3.56e-11,0,0,9e-4,7.07e-1,400)
my_temperature <- seq(400,438,0.1)
my_time        <- passive_cooling(my_coefs, my_temperature)
plot(x=my_time, y=my_temperature, type="l", col="red")
abline(h=my_coefs["Te"], v=0, col="black", lty="dashed")
```

which should have the output as shown in Figure B.1.

B.1.2 Approximations

The `R` functions in the sequel approximate the exact cooling law, and are formulated as $f(t) = T$. The following arguments are used: `e` is the emissivity of the object, `h` is the convective heat transfer coefficient, `S` the surface area, `Ta` the ambient temperature, `C` the

thermal capacity of the object, `eta1` and `eta2` coefficients describing the internal heat conversion, assuming linear internal heat generation, and `T0` is the temperature of the object at t=0.

Coefficient Approximation

```
paristech_stefan_analytical_approx_radiation
      <- function(t, e, h, S, Ta, C, eta1, eta0, T0) {

  k4 <- -emission*sigma*S/C
  k1 <- (eta1 - h*S)/C
  k0 <- ((h*S*Ta + e*sigma*S*Ta^4 - eta1) + eta0)/C

  x <-  k4*598262
  y <- -k4*251483462+k1
  z <-  k4*29700057265+k0

  omega1 <- (-y+sqrt(y^2-4*x*z))/(2*x)
  omega2 <- (-y-sqrt(y^2-4*x*z))/(2*x)

  A  <- 1/(omega2-omega1)
  B  <- -A
  c0 <- abs(T0-omega1)/abs(T0-omega2)

  if(omega2 > T0) {
    return( (omega1+omega2*c0*exp(-(t*x)/A))/((1+c0*exp(-(t*x)/A))) )
  } else {
    return( (omega1-omega2*c0*exp(-(t*x)/A))/((1-c0*exp(-(t*x)/A))) )
  }
}
```

First order O'Sullivan Approximation

```
approximation_osullivan_first_order
      <- function(t,e,h,S,Ta,C,eta0,eta1,T0) {

  sigma <- 5.6704e-8
  kelvin <- 273.15

  k <- e*sigma*S
  l <- 4*e*sigma*S*Ta
  m <- 6*e*sigma*S*Ta^2
  n <- h*S - eta1 + 4*e*sigma*S*Ta^3
  p <- -(eta1*Ta + eta0)

  return((T0+(-Ta+p/n))*exp(-n/C*t)+(-p/n+Ta))
}
```

Second order O'Sullivan Approximation

```
approximation_osullivan_second_order
      <- function(t, e, h, S, Ta, C, eta1, eta0, T0) {
```

```
  sigma <- 5.6704e-8
  kelvin <- 273.15

  k <- e*sigma*S
  l <- 4*e*sigma*S*Ta
  m <- 6*e*sigma*S*Ta^2
  n <- h*S - eta1 + 4*e*sigma*S*Ta^3
  p <- -(eta1*Ta + eta0)

  omega1 <- (-n-sqrt(n^2-4*m*p))/(2*m)
  omega2 <- (-n+sqrt(n^2-4*m*p))/(2*m)

  theta0 <- T0-Ta
  A      <- -1/(omega2-omega1)
  B      <- -A
  co     <- abs(theta0-omega1)/abs(theta0-omega2)

  if((omega2+Ta) > T0) {
    return( (omega1+omega2*co*exp(-m/(C*A)*t))/((1+co*exp(-m/(C*A)*t)))+Ta )
  } else {
    return( (omega1-omega2*co*exp(-m/(C*A)*t))/((1-co*exp(-m/(C*A)*t)))+Ta )
  }
}
```

Alternative Second order O'Sullivan Approximation

```
approximation_osullivan_second_order_alternative
      <- function(t,e,h,S,Ta,C,eta0,eta1,T0) {

  library(pracma)

  sigma <- 5.6704e-8
  kelvin <- 273.15

  k <- e*sigma*S
  l <- 4*e*sigma*S*Ta
  m <- 6*e*sigma*S*Ta^2
  n <- h*S - eta1 + 4*e*sigma*S*Ta^3
  p <- -(eta1*Ta + eta0)

  w   <- sqrt(n^2-4*m*p)

  if((omega2+Ta) > T0) {
    co  <- atanh((2*m*(T0-Ta)+n)/w)
    return( w/(2*m)*tanh(w/(2*C)*t+co) - n/(2*m) + Ta )
  } else {
    co  <- acoth((2*m*(T0-Ta)+n)/w)
    return( w/(2*m)*coth(w/(2*C)*t+co) - n/(2*m) + Ta )
}
```

B.1.3 SPICE simulation

The following piece of code simulates the thermal behavior of a slice of silica glass subject to radiative and convective cooling. The resistor representing the radiative cooling is replaced by an equivalent *voltage controlled current source*. This is possible via the *source absorption theorem* which establishes, quoting Ferreira *et al.* [38]:

> The voltage source absorption theorem establishes that if, in one branch of a circuit with current I, there is a voltage source controlled by I, the source can be replaced by a simple impedance with value equal to the source controlling factor.

In this case the reverse theorem is of use. Then the following electrical circuit may be constructed:

```
*Current/thermal equivalent network

.PARAM kelvin  = 273.15
.PARAM sigma   = 5.670373e-8

.PARAM hac     = 2.764
.PARAM T0      = 'kelvin+25'
.PARAM Ta      = 'kelvin+20'
.PARAM S       = 0.01
.PARAM epsilon = 0.94
.PARAM eta0    = 0.099733
.PARAM eta1    = 0.001053
.PARAM D       = 0.002
.PARAM Hcap    = 1548709
.PARAM Cth     = 'S*D*Hcap'

B1 0  n1 I=(eta1*(v(n1)+Ta)+eta0)
B2 n1 0  I='V(n1)/(1/(hac*S))'
B3 n1 0  I=V(n1)/((Ta-(Ta+V(n1)))/(sigma*epsilon*S*(Ta^4-(Ta+V(n1))^4)))
C1 n1 0  Cth IC=T0

.control
tran 0.2 3500 uic
*.print TRAN v(n1)
plot v(n1)
.endc
.end
```

Bibliography

[1] Aavid Thermalloy. Thermal Greases, Dec. 2013.

[2] Abramovici, M., Bradley, P. A., Levin, P. L., and Memmi, G. Integrated circuit with autonomous power management. US Patent Applicaton #20060218424, 2006.

[3] Agarwal, A., Mukhopadhyay, S., Kim, C., Raychowdhury, A., and Roy, K. Leakage power analysis and reduction: models, estimation and tools. *Computers and Digital Techniques, IEEE Proceedings - 152*, 3 (May 2005), 353–368.

[4] Amdahl, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference* (New York, NY, USA, 1967), AFIPS '67 (Spring), ACM, pp. 483–485.

[5] Austin, B., and Wright, N. J. Measurement and interpretation of microbenchmark and application energy use on the Cray XC30. In *Proceedings of the 2Nd International Workshop on Energy Efficient Supercomputing* (Piscataway, NJ, USA, 2014), E2SC '14, IEEE Press, pp. 51–59.

[6] Azar, K. *Thermal Measurements in Electronics Cooling*. Taylor & Francis, 1997.

[7] Bellosa, F. The benefits of event: Driven energy accounting in power-sensitive systems. In *Proceedings of the 9th Workshop on ACM SIGOPS European Workshop: Beyond the PC: New Challenges for the Operating System* (New York, NY, USA, 2000), EW 9, ACM, pp. 37–42.

[8] Benini, L., Bogliolo, A., and De Micheli, G. A survey of design techniques for system-level dynamic power management. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 8*, 3 (2000), 299–316.

[9] Berhe, M. K. Ergonomic temperature limits for handheld electronic devices. In *ASME InterPACK Conference* (July 2007), vol. 2, pp. 1041 – 1047.

[10] Besson, U. Cooling and warming laws: an exact analytical solution. *European Journal of Physics 31*, 5 (2010), 1107–1121.

[11] Biswas, S., Tiwari, M., Sherwood, T., Theogarajan, L., and Chong, F. Fighting fire with fire: Modeling the datacenter-scale effects of targeted superlattice thermal management. In *Computer Architecture (ISCA), 2011 38th Annual International Symposium on* (June 2011), pp. 331–340.

[12] Bonamy, R. *Modélisation, Exploration et Estimation de la Consommation pour les Architectures Hétérogènes Reconfigurables Dynamiquement*. PhD thesis, IRISA / INRIA équipe CAIRN, Université de Rennes, July 2013.

[13] Butts, J., and Sohi, G. A static power model for architects. In *Microarchitecture, 2000. MICRO-33. Proceedings. 33rd Annual IEEE/ACM International Symposium on* (2000), pp. 191–201.

[14] Calhoun, B., Wang, A., Verma, N., and Chandrakasan, A. Sub-threshold design: The challenges of minimizing circuit energy. In *Low Power Electronics and Design, 2006. Proceedings of the International Symposium on* (2006), pp. 366–368.

[15] CANDIDO, G. Le risoluzioni della equazione di quarto grado (Ferrari-Eulero-Lagrange). *Period. Mat. 21*, 4 (1941), 88–106.

[16] CARROLL, A., AND HEISER, G. An analysis of power consumption in a smartphone. In *Proceedings of the USENIX conference on USENIX* (Berkeley, CA, USA, 2010).

[17] CENGEL, Y., AND GHAJAR, A. *Heat and Mass Transfer: Fundamentals and Applications*. McGraw-Hill Education, 2010.

[18] CHANDRA, V., AND AITKEN, R. Impact of technology and voltage scaling on the soft error susceptibility in nanoscale cmos. In *Defect and Fault Tolerance of VLSI Systems, 2008. DFTVS '08. IEEE International Symposium on* (Oct 2008), pp. 114–122.

[19] CHANDRAKASAN, A. P., BOWHILL, W. J., AND FOX, F. *Design of High-Performance Microprocessor Circuits*, 1st ed. Wiley-IEEE Press, 2000.

[20] CHANG, N., KIM, K., AND LEE, H. G. Cycle-accurate energy consumption measurement and analysis: case study of ARM7TDMI. In *Proceedings of the 2000 international symposium on Low power electronics and design* (New York, NY, USA, 2000), ISLPED '00, ACM, pp. 185–190.

[21] CHAPARRO, P., GONZALEZ, J., MAGKLIS, G., CAI, Q., AND GONZALEZ, A. Understanding the thermal implications of multi-core architectures. *IEEE Transactions on Parallel and Distributed Systems 18*, 8 (2007), 1055–1065.

[22] CHAUDHRY, A. *Fundamentals of Nanoscaled Field Effect Transistors*. Springer New York, 2013.

[23] CHEN, X., CHEN, Y., MA, Z., AND FERNANDES, F. C. A. How is energy consumed in smartphone display applications? In *Proceedings of the 14th Workshop on Mobile Computing Systems and Applications* (New York, NY, USA, 2013), HotMobile '13, ACM, pp. 3:1–3:6.

[24] CHO, S., AND MELHEM, R. G. On the interplay of parallelization, program performance, and energy consumption. *IEEE Transactions on Parallel and Distributed Systems 21*, 3 (2010), 342–353.

[25] CHO, Y., AND CHANG, N. Energy-aware clock-frequency assignment in microprocessors and memory devices for dynamic voltage scaling. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 26*, 6 (June 2006), 1030–1040.

[26] CICOTTI, P., TIWARI, A., AND CARRINGTON, L. Efficient speed (ES): adaptive DVFS and clock modulation for energy efficiency. In *2014 IEEE International Conference on Cluster Computing, CLUSTER 2014, Madrid, Spain, September 22-26, 2014* (2014).

[27] COHEN, A., FINKELSTEIN, F., MENDELSON, A., RONEN, R., AND RUDOY, D. On estimating optimal performance of CPU dynamic thermal management. *IEEE Computer Architecture Letters 2*, 1 (2003), 6.

[28] DE VOGELEER, K., MEMMI, G., JOUVELOT, P., AND COELHO, F. Energy consumption modeling and experimental validation on mobile devices. Tech. Rep. 2013D008, TELECOM ParisTech, Dec. 2013.

[29] DE VOGELEER, K., MEMMI, G., JOUVELOT, P., AND COELHO, F. The Energy/Frequency Convexity Rule: modeling and experimental validation on mobile devices. In *Proceedings of the 10th Conference on Parallel Processing and Applied Mathematics* (Sept. 2013), Springer Verlag.

[30] DE VOGELEER, K., MEMMI, G., JOUVELOT, P., AND COELHO, F. Modeling the temperature bias of power consumption for nanometer-scale cpus in application processors. In *XIVth International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation* (July 2014), pp. 172–180.

[31] DUARTE, D., GEANNOPOULOS, G., MUGHAL, U., WONG, K., AND TAYLOR, G. Temperature sensor design in a high volume manufacturing 65nm CMOS digital process. In *Custom Integrated Circuits Conference, 2007. CICC '07. IEEE* (2007), pp. 221–224.

[32] Ellison, G. Maximum thermal spreading resistance for rectangular sources and plates with nonunity aspect ratios. *Components and Packaging Technologies, IEEE Transactions on 26*, 2 (June 2003), 439–454.

[33] Esmaeilzadeh, H., Cao, T., Yang, X., Blackburn, S. M., and McKinley, K. S. Looking back and looking forward: Power, performance, and upheaval. *Commun. ACM 55*, 7 (July 2012), 105–114.

[34] Esmaeilzadeh, H., Cao, T., Yang, X., Blackburn, S. M., and McKinley, K. S. What is happening to power, performance, and software? *IEEE MICRO 32*, 3 (March 2012).

[35] Fan, X., Ellis, C. S., and Lebeck, A. R. The synergy between power-aware memory systems and processor voltage scaling. In *Proceedings of the Third international conference on Power - Aware Computer Systems* (Berlin, Heidelberg, 2004), Springer-Verlag, pp. 164–179.

[36] Ferre, A., and Figueras, J. Characterization of leakage power in CMOS technologies. In *Electronics, Circuits and Systems, 1998 IEEE International Conference on* (1998), vol. 2, pp. 185 –188 vol.2.

[37] Ferreira, D., Dey, A., and Kostakos, V. Understanding human-smartphone concerns: A study of battery life. In *Pervasive Computing*, K. Lyons, J. Hightower, and E. Huang, Eds., vol. 6696 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2011, pp. 19–33.

[38] Ferreira, F. F., de Oliveira, P. G., and Tavares, V. G. *Guide to the study of Multistage Differential Amplifiers*. Universidade do Porto, 2004.

[39] Forte, D., and Srivastava, A. Energy- and thermal-aware video coding via encoder/decoder workload balancing. *ACM Trans. Embed. Comput. Syst. 12*, 2s (May 2013), 96:1–96:26.

[40] Freeh, V. W., Lowenthal, D. K., Pan, F., Kappiah, N., Springer, R., Rountree, B. L., and Femal, M. E. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Trans. Parallel Distrib. Syst. 18*, 6 (June 2007), 835–848.

[41] Ge, R., Feng, X., Song, S., Chang, H.-C., Li, D., and Cameron, K. Powerpack: Energy profiling and analysis of high-performance systems and applications. *Parallel and Distributed Systems, IEEE Transactions on 21*, 5 (May 2010), 658–671.

[42] Gold, B., and Rader, C. M. *Digital processing of signals*. McGraw-Hill NY, 1969.

[43] Grimes, R., Walsh, E., and Walsh, P. Active cooling of a mobile phone handset. *Applied Thermal Engineering 30*, 16 (2010), 2363 – 2369. Selected Papers from the 12th Conference on Process Integration, Modelling and Optimisation for Energy Saving and Pollution Reduction.

[44] Gu, R., and Elmasry, M. Power dissipation analysis and optimization of deep submicron CMOS digital circuits. *Solid-State Circuits, IEEE Journal of 31*, 5 (May 1996), 707–713.

[45] Gurrum, S., Edwards, D., Marchand-Golder, T., Akiyama, J., Yokoya, S., Drouard, J., and Dahan, F. Generic thermal analysis for phone and tablet systems. In *Electronic Components and Technology Conference (ECTC), 2012 IEEE 62nd* (May 2012), pp. 1488–1492.

[46] Hager, G., Treibig, J., Habich, J., and Wellein, G. Exploring performance and power properties of modern multi-core chips via simple machine models. *Concurrency and Computation: Practice and Experience* (2013), n/a–n/a.

[47] Halimi, J., Pradelle, B., Guermouche, A., and Jalby, W. Forest-mn: Runtime DVFS beyond communication slack. In *International Green Computing Conference, IGCC 2014, Dallas, TX, USA, November 3-5, 2014* (2014), IEEE, pp. 1–6.

[48] Halimi, J.-P., Pradelle, B., Guermouche, A., Triquenaux, N., Laurent, A., Beyler, J., and Jalby, W. Reactive dvfs control for multicore processors. In *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing* (Aug 2013), pp. 102–109.

[49] HAN, Y., KOREN, I., AND KRISHNA, C. M. TILTS: A fast architectural-level transient thermal simulation method. *J. Low Power Electronics 3*, 1 (2007), 13–21.

[50] HANSON, H., KECKLER, S., GHIASI, S., RAJAMANI, K., RAWSON, F., AND RUBIO, J. Thermal response to DVFS: analysis with an intel pentium m. In *Low Power Electronics and Design, ACM/IEEE Symposium on* (2007), pp. 219–224.

[51] HANUMAIAH, V., AND VRUDHULA, S. Reliability-aware thermal management for hard real-time applications on multi-core processors. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011* (Mar. 2011), pp. 1–6.

[52] HANUMAIAH, V., AND VRUDHULA, S. Temperature-aware DVFS for hard real-time applications on multicore processors. *IEEE Transactions on Computers 61*, 10 (2012), 1484–1494.

[53] HEATH, T., CENTENO, A. P., GEORGE, P., RAMOS, L. E., JALURIA, Y., AND BIANCHINI, R. Mercury and freon: temperature emulation and management for server systems. In *ASPLOS* (2006), ACM, pp. 106–116.

[54] HSU, C.-H., AND KREMER, U. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. *SIGPLAN Not. 38*, 5 (May 2003), 38–48.

[55] HUANG, W., GHOSH, S., VELUSAMY, S., SANKARANARAYANAN, K., SKADRON, K., AND STAN, M. Hotspot: a compact thermal modeling methodology for early-stage VLSI design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 14*, 5 (May 2006), 501–513.

[56] IKEBUCHI, D., SEKI, N., KOJIMA, Y., KAMATA, M., ZHAO, L., AMANO, H., SHIRAI, T., KOYAMA, S., HASHIDA, T., UMAHASHI, Y., MASUDA, H., USAMI, K., TAKEDA, S., NAKAMURA, H., NAMIKI, M., AND KONDO, M. Geyser-1: a MIPS R3000 CPU core with fine-grained run-time power gating. In *ASP-DAC* (2010), IEEE, pp. 369–370.

[57] INTEL CORP. Intel Data Direct I/O Technology Overview, 2012.

[58] JAYASEELAN, R., AND MITRA, T. Temperature aware task sequencing and voltage scaling. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design* (Piscataway, NJ, USA, 2008), ICCAD '08, IEEE Press, pp. 618–623.

[59] KESHAVARZI, A., ROY, K., AND HAWKINS, C. Intrinsic leakage in low power deep submicron CMOS ICs. In *Test Conference, 1997. Proceedings., International* (Nov 1997), pp. 146–155.

[60] KONG, J., CHUNG, S. W., AND SKADRON, K. Recent thermal management techniques for microprocessors. *ACM Comput. Surv. 44*, 3 (June 2012), 13:1–13:42.

[61] KOREN, I., AND KRISHNA, C. *Fault-Tolerant Systems*. Elsevier Science, 2010.

[62] KURODA, T. Optimization and control of vdd and vth for low-power, high-speed CMOS design. In *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on* (Nov 2002), pp. 28–34.

[63] LALL, P., PECHT, M., AND HAKIM, E. *Influence of Temperature on Microelectronics and System Reliability: A Physics of Failure Approach*. The electronic packaging series. CRC Press, 1997.

[64] LE SUEUR, E., AND HEISER, G. Dynamic voltage and frequency scaling: the laws of diminishing returns. In *Proceedings of the 2010 international conference on Power aware computing and systems* (Berkeley, CA, USA, 2010), HotPower'10, pp. 1–8.

[65] LEE, S., SEAHO SONG, V. A., AND MORAN, K. P. Constriction/spreading resistance model for electronics packaging. *ASME/JSME Thermal Engineering Conference 4* (1995).

[66] LEITON GMBH. Datasheet Rev. 2.2 - FR4 Printed Circuits, Dec. 2013.

[67] LI, X., RONG, M., LIU, T., AND ZHOU, L. Research of thermal sensor allocation and placement based on dual clustering for microprocessors. *SpringerPlus 2*, 1 (2013), 253.

[68] Liao, W., Basile, J. M., and He, L. Leakage power modeling and reduction with data retention. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design* (New York, NY, USA, 2002), ACM, pp. 714–719.

[69] Liao, W., He, L., and Lepak, K. M. Temperature and supply voltage aware performance and power modeling at microarchitecture level. *Trans. Comp.-Aided Des. Integ. Cir. Sys. 24*, 7 (Nov. 2006), 1042–1053.

[70] Liu, W., Jin, X., Kao, K., and Hu, C. BSIM4 v4.8.0 MOSFET model-user's manual. Tech. rep., EECS Dept., Univ. of California, Berkeley, Nov. 2013.

[71] Liu, Y., Dick, R. P., Shang, L., and Yang, H. Accurate temperature-dependent integrated circuit leakage power estimation is easy. In *Proceedings of the Conference on Design, Automation and Test in Europe* (San Jose, CA, USA, 2007), DATE '07, EDA Consortium, pp. 1526–1531.

[72] Liu, Z., and Kursun, V. Leakage power characteristics of dynamic circuits in nanometer CMOS technologies. *Circuits and Systems II: Express Briefs, IEEE Transactions on 53*, 8 (2006), 692–696.

[73] Luo, Z., Cho, H., Luo, X., and il Cho, K. System thermal analysis for mobile phone. *Applied Thermal Engineering 28*, 14-15 (2008), 1889 – 1895.

[74] Maronna, R., Martin, D., and Yohai, V., Eds. *Robust Statistics: Theory and Methods*. John Wiley & Sons, New York City, NY, USA, 2006.

[75] Mazouz, A., Laurent, A., Pradelle, B., and Jalby, W. Evaluation of CPU frequency transition latency. *Computer Science - R&D 29*, 3-4 (2014), 187–195.

[76] McConnell, A., Uma, S., and Goodson, K. E. Thermal conductivity of doped polysilicon layers. *Microelectromechanical Systems, Journal of 10*, 3 (Sep 2001), 360–369.

[77] McGowen, R., Poirier, C., Bostak, C., Ignowski, J., Millican, M., Parks, W., and Naffziger, S. Power and temperature control on a 90-nm itanium family processor. *Solid-State Circuits, IEEE Journal of 41*, 1 (2006), 229–237.

[78] Mesa-Martinez, F., Brown, M., Nayfach-Battilana, J., and Renau, J. Measuring power and temperature from real processors. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on* (April 2008), pp. 1–5.

[79] MGC inc. BT Materials for IC Plastic Package, Dec. 2013.

[80] Mukhopadhyay, S., Raychowdhury, A., and Roy, K. Accurate estimation of total leakage current in scaled CMOS logic circuits based on compact current modeling. In *Design Automation Conference, 2003. Proceedings* (June 2003), pp. 169 – 174.

[81] Noureddine, A., Rouvoy, R., and Seinturier, L. A review of energy measurement approaches. *Operating Systems Review 47*, 3 (2013), 42–49.

[82] O'Sullivan, C. T. Newton's law of cooling – a critical assessment. *American Journal of Physics 58*, 10 (Oct. 1990), 956–960.

[83] Pallister, J., Hollis, S., and Bennett, J. BEEBS: Open benchmarks for energy measurements on embedded platforms. *CoRR abs/1308.5174* (2013).

[84] Pallister, J., Hollis, S., and Bennett, J. The impact of different compiler options on energy consumption. In *First LPGPU Workshop on Power-Efficient GPU and Many-core Computing* (New York, NY, USA, 2013), PEGPUM '13, ACM.

[85] Park, J.-H., Shin, S., Christofferson, J., Shakouri, A., and Kang, S.-M. Experimental validation of the power blurring method. In *Semiconductor Thermal Measurement and Management Symposium, 2010. SEMI-THERM 2010. 26th Annual IEEE* (Feb. 2010), pp. 240–244.

[86] Parker Chomerics. Thermal Grease, Dec. 2013.

[87] PITTS, D. R., AND SISSOM, L. E. *Heat Transfer*. Schaum's Outlines. McGraw-Hill, 2012.

[88] QIU, M., MING, Z., LI, J., LIU, S., WANG, B., AND LU, Z. Three-phase time-aware energy minimization with {DVFS} and unrolling for chip multiprocessors. *Journal of Systems Architecture 58*, 10 (2012), 439 – 445.

[89] QUAN, G., AND ZHANG, Y. Leakage aware feasibility analysis for temperature-constrained hard real-time periodic tasks. In *Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems* (Washington, DC, USA, 2009), ECRTS '09, IEEE Computer Society, pp. 207–216.

[90] RAGHAVAN, A., LUO, Y., CHANDAWALLA, A., PAPAEFTHYMIOU, M., PIPE, K. P., WENISCH, T. F., AND MARTIN, M. M. K. Computational sprinting. In *Proceedings of the 2012 IEEE 18th International Symposium on High-Performance Computer Architecture* (Washington, DC, USA, 2012), HPCA '12, IEEE Computer Society, pp. 1–12.

[91] RAMALINGAM, A., LIU, F., NASSIF, S., AND PAN, D. Accurate thermal analysis considering nonlinear thermal conductivity. In *Quality Electronic Design, 2006. ISQED '06. 7th International Symposium on* (March 2006), pp. 6 pp.–649.

[92] RANIERI, J., VINCENZI, A., CHEBIRA, A., ATIENZA, D., AND VETTERLI, M. Eigenmaps: Algorithms for optimal thermal maps extraction and sensor placement on multicore processors. In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE* (June 2012), pp. 636–641.

[93] REDA, S., COCHRAN, R., AND NOWROZ, A. Improved thermal tracking for processors using hard and soft sensor allocation techniques. *Computers, IEEE Transactions on 60*, 6 (June 2011), 841–851.

[94] REZZI, F., COLLAMATI, L., COSTAGLIOLA, M., AND CUTRUPI, M. Battery management in mobile devices. In *Frequency References, Power Management for SoC, and Smart Wireless Interfaces*, A. Baschirotto, K. A. Makinwa, and P. Harpe, Eds. Springer International Publishing, 2014, pp. 147–168.

[95] RIZVANDI, N., TAHERI, J., ZOMAYA, A., AND LEE, Y. C. Linear combinations of dvfs-enabled processor frequencies to modify the energy-aware scheduling algorithms. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on* (May 2010), pp. 388–397.

[96] RIZVANDI, N. B., TAHERI, J., AND ZOMAYA, A. Y. Some observations on optimal frequency selection in dvfs-based energy consumption minimization. *Journal of Parallel and Distributed Computing 71*, 8 (2011), 1154 – 1164.

[97] ROTEM, E., NAVEH, A., RAJWAN, D., ANANTHAKRISHNAN, A., AND WEISSMANN, E. Power-management architecture of the intel microarchitecture code-named sandy bridge. *IEEE Micro 32*, 2 (2012), 0020–27.

[98] ROY, K., MUKHOPADHYAY, S., AND MAHMOODI-MEIMAND, H. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits. *Proceedings of the IEEE 91*, 2 (Feb. 2003), 305 – 327.

[99] SABRY, M.-N. Compact thermal models for electronic systems. *Components and Packaging Technologies, IEEE Transactions on 26*, 1 (March 2003), 179–185.

[100] SARANGI, S., ANANTHANARAYANAN, G., AND BALAKRISHNAN, M. Lightsim: A leakage aware ultrafast temperature simulator. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific* (Jan. 2014), pp. 855–860.

[101] SEEKER, V., PETOUMENOS, P., LEATHER, H., AND FRANKE, B. Measuring qoe of interactive workloads and characterising frequency governors on mobile devices. In *2014 IEEE International Symposium on Workload Characterization, IISWC 2014, Raleigh, NC, USA, October 26-28, 2014* (2014), pp. 61–70.

[102] Semenov, O., Pradzynski, A., and Sachdev, M. Impact of gate induced drain leakage on overall leakage of submicrometer CMOS vlsi circuits. *Semiconductor Manufacturing, IEEE Transactions on 15*, 1 (Feb 2002), 9–18.

[103] Senn, E., Laurent, J., Julien, N., and Martin, E. Softexplorer: Estimating and optimizing the power and energy consumption of a C program for DSP applications. *EURASIP J. Adv. Sig. Proc. 2005*, 16 (2005), 2641–2654.

[104] Seo, Y., Kim, J., and Seo, E. Effectiveness analysis of dvfs and dpm in mobile devices. *Journal of Computer Science and Technology 27*, 4 (2012), 781–790.

[105] Seok, M., Jeon, D., Chakrabarti, C., Blaauw, D., and Sylvester, D. Pipeline strategy for improving optimal energy efficiency in ultra-low voltage design. In *Proceedings of the 48th Design Automation Conference* (New York, NY, USA, 2011), DAC '11, ACM, pp. 990–995.

[106] Simunic, T., Benini, L., and De Micheli, G. Cycle-accurate simulation of energy consumption in embedded systems. In *Design Automation Conference, 1999. Proceedings. 36th* (1999), pp. 867 –872.

[107] Singh, K., Bhadauria, M., and McKee, S. A. Real time power estimation and thread scheduling via performance counters. *SIGARCH Comput. Archit. News 37*, 2 (July 2009), 46–55.

[108] Sinha, A., and Chandrakasan, A. P. Jouletrack: a web based tool for software energy profiling. In *Proceedings of the 38th annual Design Automation Conference* (New York, NY, USA, 2001), DAC '01, ACM, pp. 220–225.

[109] Skadron, K., Stan, M. R., Sankaranarayanan, K., Huang, W., Velusamy, S., and Tarjan, D. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Archit. Code Optim. 1*, 1 (Mar. 2004), 94–125.

[110] Snowdon, D. C. *OS-Level Power Management*. PhD thesis, School of Computer Science and Engineering, University of NSW, Sydney 2052, Australia, Mar. 2010.

[111] Snowdon, D. C., Ruocco, S., and Heiser, G. Power management and dynamic voltage scaling: Myths and facts. In *2005 WS Power Aware Real-time Comput.* (New Jersey, USA, Sept. 2005).

[112] Srikant, Y. N., and Shankar, P., Eds. *The Compiler Design Handbook: Optimizations and Machine Code Generation*. CRC Press, 2008.

[113] Stallings, W. *Computer Organization and Architecture*. Pearson Education, 2015.

[114] Su, H., Liu, F., Devgan, A., Acar, E., and Nassif, S. Full chip leakage estimation considering power supply and temperature variations. In *Proceedings of the 2003 international symposium on Low power electronics and design* (New York, NY, USA, 2003), ISLPED '03, ACM, pp. 78–83.

[115] Sylvester, D., and Keutzer, K. System-level performance modeling with BACPAC - berkeley advanced chip performance calculator. In *1st International Workshop on System-Level Interconnect Prediction* (1999), pp. 109–114.

[116] Tudor, B. M., and Teo, Y. M. On understanding the energy consumption of arm-based multicore servers. *SIGMETRICS Perform. Eval. Rev. 41*, 1 (June 2013), 267–278.

[117] Vallina-Rodriguez, N., and Crowcroft, J. Energy management techniques in modern mobile handsets. *Communications Surveys Tutorials, IEEE 15*, 1 (First 2013), 179–198.

[118] Valluri, M., and John, L. Is compiling for performance == compiling for power? In *Interaction between Compilers and Computer Architectures*, G. Lee and P.-C. Yew, Eds., vol. 613 of *The Springer International Series in Engineering and Computer Science*. Springer US, 2001, pp. 101–115.

[119] Veendrick, H. Short-circuit dissipation of static CMOS circuitry and its impact on the design of buffer circuits. *Solid-State Circuits, IEEE Journal of 19*, 4 (Aug. 1984), 468–473.

[120] VEMURU, S. R., AND SCHEINBERG, N. Short-circuit power dissipation estimation for cmos logic gates. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on 41*, 11 (Nov 1994), 762–765.

[121] VERMEERSCH, B., AND DE MEY, G. Dependency of thermal spreading resistance on convective heat transfer coefficient. *MICROELECTRONICS RELIABILITY 48*, 5 (2008), 734–738.

[122] VINCENZI, A., SRIDHAR, A., RUGGIERO, M., AND ATIENZA, D. Fast thermal simulation of 2D/3D integrated circuits exploiting neural networks and GPUfs. In *Proceedings of the 17th IEEE/ACM International Symposium on Low-power Electronics and Design* (Piscataway, NJ, USA, 2011), ISLPED '11, IEEE Press, pp. 151–156.

[123] VIREDAZ, M. A., BRAKMO, L. S., AND HAMBURGEN, W. R. Energy management on handheld devices. *Queue 1*, 7 (Oct. 2003), 44–52.

[124] VIRGINIA SEMICONDUCTOR. The General Properties of Si, Ge, SiGe, SiO_2 and Si_3N_4, June 2012.

[125] VISWANATH, R., WAKHARKAR, V., WATWE, A., AND LEBONHEUR, V. Thermal performance challenges from silicon to systems. *Intel Technology Journal*, Q3 (Aug. 2000), 1–16.

[126] WANG, W., PORTERFIELD, A., CAVAZOS, J., AND BHALACHANDRA, S. Using per-loop CPU clock modulation for energy efficiency in OpenMP applications. In *International Conference on Parallel Processing* (2015), ICPP.

[127] WEISSEL, A., AND BELLOSA, F. Dynamic thermal management for distributed systems. In *Proceedings of the First Workshop on Temperatur-Aware Computer Systems (TACS'04)* (Munich, Germany, June 2004).

[128] WESTE, N. H. E., AND ESHRAGHIAN, K. *Principles of CMOS VLSI design: a systems perspective.* Addison-Wesley Longman Publishing Co., Inc., Boston, USA, 1985.

[129] WILLIAMS, B. W. *Power electronics : devices, drivers, applications and passive components*, 2nd ed. ed. Macmillan Basingstoke ; London, 1992.

[130] WOO, D. H., AND LEE, H.-H. S. Extending amdahl's law for energy-efficient computing in the many-core era. *Computer 41*, 12 (Dec. 2008), 24–31.

[131] XIE, Q., DOUSTI, M. J., AND PEDRAM, M. Therminator: A thermal simulator for smartphones producing accurate chip and skin temperature maps. In *Proceedings of the 2014 International Symposium on Low Power Electronics and Design* (New York, NY, USA, 2014), ISLPED '14, ACM, pp. 117–122.

[132] XU, Y.-J., LUO, Z.-Y., LI, X.-W., LI, L.-J., AND HONG, X.-L. Leakage current estimation of CMOS circuit with stack effect. *J. Comput. Sci. Technol. 19*, 5 (Sept. 2004), 708–717.

[133] YANG, Y., GU, Z., ZHU, C., DICK, R., AND SHANG, L. ISAC: Integrated space-and-time-adaptive chip-package thermal analysis. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 26*, 1 (Jan. 2007), 86–99.

[134] YOU, Y.-P., LEE, C., AND LEE, J. K. Compiler analysis and supports for leakage power reduction on microprocessors. In *Proceedings of the 15th international conference on Languages and Compilers for Parallel Computing* (Berlin, Heidelberg, 2005), LCPC'02, Springer-Verlag, pp. 45–60.

[135] YOVANOVICH, M. M. Thermal resistances of circular source on finite circular cylinder with side and end cooling. *J. Electron. Packag. 125*, 2 (2003), 169–177.

[136] YUAN, W., AND NAHRSTEDT, K. Energy-efficient CPU scheduling for multimedia applications. *ACM Trans. Comput. Syst. 24*, 3 (Aug. 2006), 292–331.

[137] YUKI, T., AND RAJOPADHYE, S. Folklore confirmed: Compiling for speed = compiling for energy. In *Proceedings of the 26th International Workshop on Languages and Compilers for Parallel Computing* (2013).

[138] Zanini, F., Atienza, D., Jones, C. N., Benini, L., and De Micheli, G. Online thermal control methods for multiprocessor systems. *ACM Trans. Des. Autom. Electron. Syst. 18*, 1 (Jan. 2013), 6:1–6:26.

[139] Zhai, B., Blaauw, D., Sylvester, D., and Flautner, K. Theoretical and practical limits of dynamic voltage scaling. In *Proceedings of the 41st annual Design Automation Conference* (New York, NY, USA, 2004), DAC '04, ACM, pp. 868–873.

[140] Zhang, S., and Chatha, K. S. Approximation algorithm for the temperature-aware scheduling problem. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design* (Piscataway, NJ, USA, 2007), ICCAD '07, IEEE Press, pp. 281–288.

[141] Zhang, Y., Parikh, D., Sankaranarayanan, K., Skadron, K., and Stan, M. Hotleakage: A temperature-aware model of subthreshold and gate leakage for architects. Tech. Rep. CS-2003-05, Univ. of V., Mar. 2003.