Compiling Image Processing Applications for Many-Core Accelerators
Pierre Guillou

To cite this version:

Pierre Guillou. Compiling Image Processing Applications for Many-Core Accelerators. Journées de seconde année de l’Ecole Doctorale, Jun 2015, Paris, France. hal-01178938

HAL Id: hal-01178938
https://minesparis-psl.hal.science/hal-01178938
Submitted on 21 Jul 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
**Image Processing**

- Image analysis: detect geometrical structures in an image
- Mathematical morphology: image analysis theory and technique based on lattice theory

---

**Mathematical Morphology Base Operators**

- Arithmetic operators
  - Addition, subtraction, multiplication, division
  - Bitwise operations
- Morphological operators
  - Opening, closing, erosion, dilation
- Reduction operators
  - Minimum, maximum
- Other operators
  - Thresholding, mask, log2, ...

---

**Sigma-C, a Dataflow Programming Language**

Example: Licence Plate Extraction

![Example: Licence Plate Extraction](image)

```c
agent foo() {
  // describe agent interface
  interface {
    input inp0 [2], inp1, output outp [1], output outp [2];
    // describe subgraph interface
    interface { /* ... */ }
    // connect agents
    agent a3 = new Subgraph3(); // ...
    agent a1 = new Agent1();
    connect(a3.output, a5.input1);
    connect(a1.output0, a2.input);
    connect(a5.output, output1);
    connect(input0, a1.input0);
  }
  // loop over the state
  void start() {
    exchange(input0, inp0[2], input1, inp1, output outp[1], output outp[2]);
    // describe subgraph interface
    spec { input0[2], input1, output[3];
      // loop over the state
      void start() {
        exchange(input0, inp0[2], input1, inp1, output outp[1], output outp[2]);
        // describe subgraph interface
        spec { input0[2], input1, output[3];
          // loop over the state
          return outp[1];
        }
      }
    }
  }
}
```

---

**The MPPA-256 Chip**

![The MPPA-256 Chip](image)

**Compilation Chain**

- Source-to-source compiler
- Call graph optimisations
- Target-specific compiler
- Compute libraries

**Runtime Environment**

- Control code
- Host runtime
- Accelerator runtime
- Compute clusters

**Optimisations**

- Unrolling of converging loops
- Arithmetic operators aggregation
- Generation of kernel-specific convolutions
- Data parallelization for compute-intensive operators

---

**Results: Execution Times and Energy Consumption**

(MPPA-256 = 1, lower is better)

![Results: Execution Times and Energy Consumption](image)

---

**Future Work**

- Other programming models:
  - C, Fortran, OpenMP on compute clusters, communication library between clusters
  - OpenCL via local memory pagination
- Improve data-parallelism to better advantage of the current architecture
- Implement more complex algorithms: watershed, arrow, labelling, minima, ...

---

**References**

Pierre Guillou, Fabien Coelho, and François Irigoin.
Automatic Streamlining of Image Processing Applications.
The 27th International Workshop on Languages and Compilers for Parallel Computing (LCPC), 2014.