GPU-accelerated dynamic nonlinear optimization with ExaModels and MadNLP Franc ¸ois Pacaud and Sungho Shin

François Pacaud, Sungho Shin

GPU-accelerated dynamic nonlinear optimization with ExaModels and MadNLP

François Pacaud and Sungho Shin

***Abstract*— We investigate the potential of Graphics Processing Units (GPUs) to solve large-scale nonlinear programs with a dynamic structure. Using ExaModels, a GPU-accelerated automatic differentiation tool, and the interior-point solver MadNLP, we significantly reduce the time to solve dynamic nonlinear optimization problems. The sparse linear systems formulated in the interior-point method is solved on the GPU using a hybrid solver combining an iterative method with a sparse Cholesky factorization, which harness the newly released NVIDIA cuDSS solver. Our results on the classical distillation column instance show that despite a significant pre-processing time, the hybrid solver allows to reduce the time per iteration by a factor of 25 for the largest instance.**

I. INTRODUCTION

There is a strong interest in using high-performance computing for accelerating the solution of dynamic nonlinear programs, as this is critical for Nonlinear Model Predictive Control (NMPC) and optimal control applications [1], [2]. It is well known that for problems with a dynamic structure, the Newton step is equivalent to a linear-quadratic problem, solvable using dynamic programming or Riccati recursions [3]. By doing so, the method exploits the dynamic structure *explicitly*. Alternatively, one can leverage the structure *implicitly* inside a sparse direct solver, in charge of finding an appropriate ordering to reduce the fill-in in the sparse factorization [4]. That kind of many-degrees-of-freedom approaches are known to scale better with the problem's size.

A. Related works

When solving generic large-scale nonlinear programs, the two bottlenecks are the computation of the Newton step and the evaluation of the derivatives. On the one hand, the solution of the Newton step can be accelerated either by exploiting the structure explicitly or by using efficient sparse linear algebra routines [5], [6], [7]. On the other hand, efficient automatic differentiation routines have been introduced, which now evaluate the first and second-order derivatives in a vectorized fashion for performance [8]. As a result, the dynamic optimization problem can be solved with near real-time performance [9] when coupled with an optimization solver [10], [11], [12].

With their focus on embedded applications, the solvers listed in the previous paragraph have been heavily optimized on CPU architectures, going as far as using dedicated linear algebra routines [13]. Aside, NVIDIA has recently released the NVIDIA Jetson GPU, developed primarily for embedded applications. Hence, solving MPC problems on GPU/SIMD architectures is gaining more traction [14], with new applications in robotics and autonomous vehicles [15], [16], [17].

B. Contributions

In this article, we investigate the capability of the modeler ExaModels and the interior-point solver MadNLP [18] — both leveraging GPU acceleration — to solve nonlinear programs with dynamic structure. MadNLP implements a filter line-search interior-point method [19], which results in solving a sequence of sparse indefinite linear systems with a saddle-point structure [20]. The linear systems are increasingly ill-conditioned as we are approaching the solution, preventing a solution with Krylov-based solvers. The alternative is to use an inertia-revealing sparse direct solver, generally implementing the Duff-Reid factorization [21]. Unfortunately, it is well known that such factorization is not practical on the GPU, as they rely on expensive numerical pivoting operations for stability [22], [23]. The usual workaround is to densify the solution of the linear systems using a null-space method, as was investigated in our previous work [24], [25]. Instead, we propose to solve the linear systems with a hybrid sparse linear solver mixing a sparse Cholesky routine with an iterative method. We present two alternative methods for the hybrid solver. On the one hand, Lifted-KKT [18] uses an equality relaxation strategy to reduce the indefinite linear system down to a sparse positive definite matrix, factorizable using Cholesky factorization. On the other hand, HyKKT [26] uses the Golub and Greif method [27] (itself akin to an Augmented Lagrangian method) to solve the linear system with an inner direct solve used in conjunction with a conjugate gradient. Both methods are fully implementable on the GPU and rely only on basic linear algebra routines. We show on the classical distillation column instance [7] that despite a significant pre-processing time, both Lifted-KKT and HyKKT reduce the time per IPM iteration by a factor of 25 and 18 respectively, compared to the HSL solvers.

II. PROBLEM FORMULATION

We formulate the dynamic nonlinear optimization problem as a generic nonlinear program. Let n the number of variables. The objective is encoded by a function $f : \mathbb{R}^n \to \mathbb{R}$, the dynamic and algebraic constraints by a function $g : \mathbb{R}^n \to \mathbb{R}^{m_e}$ and the remaining inequality constraints by a function $h : \mathbb{R}^n \to \mathbb{R}^{m_i}$. By introducing slack variables $s \geq 0$, the problem writes:

$$\begin{aligned} \min_{x \in \mathbb{R}^n, s \in \mathbb{R}^{m_i}} \; & f(x) \\ \text{s.t. } & g(x) = 0 \;,\; h(x) + s = 0 \;,\; s \geq 0. \end{aligned} \tag{1}$$

We note $y \in \mathbb{R}^{m_e}$ (resp. $z \in \mathbb{R}^{m_i}$) the multiplier attached to the equality constraints (resp. the inequality constraints). The Lagrangian of (1) is defined as

$$L(x, s, y, z, \nu) = f(x) + y^\top g(x) + z^\top (h(x) + s) - \nu^\top s \ . \quad (2)$$

A primal-dual variable $w := (x, s, y, z, \nu)$ is solution of Problem (1) if it satisfies the Karush-Kuhn-Tucker (KKT) equations

$$\begin{cases} \nabla_x L(x, s, y, z, \nu) = 0 \ , \\ z - \nu = 0 \\ g(x) = 0 \ , \\ h(x) + s = 0 \ , \\ 0 \leq s \perp \nu \geq 0 \ , \end{cases} \quad (3)$$

where we use the symbol $\perp$ to denote the complementarity constraints $s_i \times \nu_i = 0$ for $i = 1, \cdots, m_i$.

We note the active set $\mathcal{B}(x) = \{i = 1, \cdots, m_i \mid h_i(x) = 0\}$, and denote the active Jacobian as $A(x) = \left[\nabla_x g(x)^\top \ \nabla_x h_{\mathcal{B}}(x)^\top\right]^\top$ with $h_{\mathcal{B}}(x) := \{h_i(x)\}_{i \in \mathcal{B}(x)}$. We suppose the following assumption holds at a primal-dual solution $w^\star := (x^\star, s^\star, y^\star, z^\star, \nu^\star)$ of (1).

- Linear Independance Constraint Qualification (LICQ): the active Jacobian $A(x^\star)$ is full row-rank.
- Strict complementarity (SCS): for every $i \in \mathcal{B}(x^\star)$, $z_i^\star > 0$.
- Second-order sufficiency (SOSC): for every $v \in \text{null}(A(x^\star))$, $v^\top (\nabla^2_{xx} L(w^\star)) v > 0$.

III. INTERIOR-POINT METHOD

The primal-dual interior-point method (IPM) reformulates the non-smooth KKT conditions (3) using an homotopy method [28, Chapter 19]. For a barrier parameter $\mu > 0$, IPM solves the smooth system of nonlinear equations $F_\mu(w) = 0$ for $(s, \nu) > 0$ and $F_\mu(\cdot)$ defined as

$$F_\mu(w) = \begin{bmatrix} \nabla_x L(x, s, y, z, \nu) \\ g(x) \\ h(x) + s \\ S\nu - \mu e_{m_i} \end{bmatrix} . \quad (4)$$

We set $S = \text{diag}(s)$, $V = \text{diag}(\nu)$, and $e_{m_i} \in \mathbb{R}^{m_i}$ a vector filled with 1. As we drive $\mu \to 0$, we recover the original KKT conditions (3).

A. Newton method

The system of nonlinear equations (4) is solved using a Newton method. The primal-dual variable is updated as $w_{k+1} = w_k + \alpha d_k$, where d_k is a descent direction solution of the linear system

$$\nabla_w F_\mu(w_k) d_k = -F_\mu(w_k) \ . \quad (5)$$

The step α is computed using a fraction-to-boundary rule, guaranteeing that $(s, \nu) > 0$ throughout the iterations [28].

B. Augmented KKT system

We note the local sensitivities $W_k = \nabla^2_{xx} L(w_k) \in \mathbb{R}^{n \times n}$, $H_k = \nabla_x h(x_k) \in \mathbb{R}^{m_i \times n}$ and $G_k = \nabla_x g(x_k) \in \mathbb{R}^{m_e \times n}$. The solution of the linear system (5) translates to the *augmented KKT system*:

$$\overbrace{\begin{bmatrix} W_k & 0 & G_k^\top & H_k^\top \\ 0 & D_s & 0 & I \\ G_k & 0 & 0 & 0 \\ H_k & I & 0 & 0 \end{bmatrix}}^{K_{aug}} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \end{bmatrix} = - \begin{bmatrix} r_1 \\ r_2 \\ r_3 \\ r_4 \end{bmatrix} , \quad (6)$$

with the diagonal matrix $D_s = S_k^{-1} V_k$. The right-hand-sides are given respectively by $r_1 = \nabla f(x_k) + \nabla g(x_k)^\top y_k + \nabla h(x_k)^\top z_k + \mu X^{-1} e$, $r_2 = z_k + \mu S^{-1} e$, $r_3 = g(x_k)$, $r_4 = h(x_k) + s_k$.

The system (6) is sparse, symmetric and exhibits a saddle-point structure. Most nonlinear optimization solvers solve the system (6) using a sparse LBL factorization [21]. Using [20, Theorem 3.4], the system (6) is invertible if the Jacobian $J_k = \begin{bmatrix} G_k & 0 \\ H_k & I \end{bmatrix}$ is full row-rank and

$$\text{null}\left(\begin{bmatrix} W_k & 0 \\ 0 & D_s \end{bmatrix}\right) \cap \text{null}\left(\begin{bmatrix} G_k & 0 \\ H_k & I \end{bmatrix}\right) = \{0\} \ . \quad (7)$$

To ensure (7) holds, the solver checks the inertia $\text{In}(K_{aug})$ (the tuple (n_+, n_0, n_-) encoding respectively the number of positive, null and negative eigenvalues in K_{aug}). If

$$\text{In}(K_{aug}) = (n + m_i, 0, m_i + m_e) \ , \quad (8)$$

then the system K_{aug} is invertible and the solution of the system (6) is a descent direction. Otherwise, the solver regularizes (6) using two parameters $(\delta_x, \delta_c) > 0$ and solves

$$\begin{bmatrix} W_k + \delta_x I & 0 & G_k^\top & H_k^\top \\ 0 & D_s + \delta_x I & 0 & I \\ G_k & 0 & -\delta_c I & 0 \\ H_k & I & 0 & -\delta_c I \end{bmatrix} . \quad (9)$$

The parameter (δ_x, δ_c) are computed so as the regularized system satisfies (8). There exists inertia-free variants for IPM [29], but experimentally, inertia-based method are known to converge in fewer iterations.

C. IPM and optimal control

IPM is a standard method to solve MPC and optimal control problems [6]. In particular, if (1) encodes a problem with a dynamic structure, the Hessian W_k and the Jacobian H_k are block diagonal, the Jacobian G_k playing the role of the coupling matrix.

There exist interesting refinements of the IPM method for problems with a dynamic structure [12]. Notably:

- The primal regularization δ_x can be computed recursively using dynamic programming, in a way that keeps the primal solution of the system (6) intact [30].
- Similarly, the dual regularization δ_c can be refined to take into account the problem's structure, implicitly (inside the linear solver [31]) or explicitly (using exact penalty [32]).

IV. CONDENSED KKT SYSTEM

Solving the augmented KKT system (6) is numerically demanding, and is often the computational bottleneck in IPM. Furthermore, the sparse LBL factorization is known to be non trivial to parallelize, as it relies on extensive numerical pivoting operations [23]. Fortunately, the KKT system (6) can be reduced down to a positive definite matrix, whose factorization can be computed efficiently using a Cholesky factorization.

First, we exploit the structure of the system (6) using a condensation step[1]. The system (6) is reduced by removing the blocks associated to the slack d_s and to the inequality multiplier d_z. We obtain the equivalent *condensed KKT system*,

$$\overbrace{\begin{bmatrix} K_k & G_k^\top \\ G_k & 0 \end{bmatrix}}^{K_{cond}} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} r_1 + H_k^\top (D_s r_4 - r_2) \\ r_3 \end{bmatrix} , \quad (10)$$

with the *condensed matrix* $K_k := W_k + \delta_x + H_k^\top D_s H_k$. Using the solution of the system (10), we recover the updates on the slacks and inequality multipliers with $d_s = -r_4 - H_k d_x$ and $d_z = -r_2 - D_s d_s$. We note that the condensed matrix K_k retains the block structure of the Hessian W_k and Jacobian H_k.

Using Haynsworth's inertia additivity formula, we have the equivalence

$$\begin{aligned} \text{In}(K_{aug}) = (n + m_i, 0, m_i + m_e) \iff \\ \text{In}(K_{cond}) = (n, 0, m_e) . \end{aligned} \quad (11)$$

Usually, the solver uses a sparse LBL factorization to solve the KKT system K_{aug} or K_{cond}. Here, we move one step further and reduce the condensed KKT system (10) down to a (sparse) positive definite matrix, using either Lifted-KKT [18] or HyKKT [26].

A. Solution 1: Lifted-KKT

We observe in (10) that without equality constraints, we obtain a $n \times n$ system which is guaranteed to be positive definite if the primal regularization parameter δ_x is chosen appropriately. Hence, we relax the equality constraints in (1) using a small relaxation parameter $\tau > 0$, and solve the relaxed problem

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{s.t.} \quad -\tau \leq g(x) \leq \tau \; , \; h(x) \leq 0 \; . \quad (12)$$

The problem (12) has only inequality constraints. After introducing slack variables, the condensed KKT system (10) reduces to

$$K_k d_x = -r_1 - H_k^\top (D_s r_4 - r_2) \; . \quad (13)$$

Using inertia correction method, the parameter δ_x is set to a value high enough to render the matrix K_k positive definite. As a result, it can be factorized efficiently using a sparse Cholesky method.

[1]Here, the condensation removes the blocks associated to the slacks and the inequalities in the KKT system. As such, it has a different meaning than the condensing procedure used in model predictive control, which eliminates the state variables at all time except at 0 to obtain a dense KKT system [12].

B. Solution 2: HyKKT

A substitute method is to exploit directly the structure of the condensed KKT system (10), without reformulating the initial problem (1). To do so, we observe that if K_k were positive definite, the solution of the system (10) can be evaluated using the Schur complement $G_k K_k^{-1} G_k^\top$. Unfortunately the original problem (1) is nonconvex: we have to convexify it using an Augmented Lagrangian technique. For $\gamma > 0$, we note the KKT system (10) is equivalent to

$$\begin{bmatrix} K_k + \gamma G_k^\top G_k & G_k^\top \\ G_k & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_y \end{bmatrix} = - \begin{bmatrix} r_\gamma \\ r_3 \end{bmatrix} . \quad (14)$$

with $r_\gamma := r_1 + H_k^\top (D_s r_4 - r_2) + \gamma G_k^\top r_3$.

We note Z a basis of the null-space of the Jacobian G_k. We know that if $Z^\top K_k Z$ is positive definite and G_k is full row-rank, then there exists a threshold value $\underline{\gamma}$ such that for all $\gamma \geq \underline{\gamma}$, $K_\gamma := K_k + \gamma G_k^\top G_k$ is positive definite [33]. This fact is exploited in the Golub and Greif method [27], which has been recently revisited in [26]. If K_γ is positive definite, we can solve the system (14) using a Schur-complement method, by computing the dual descent direction d_y as solution of

$$(G_k K_\gamma^{-1} G_k^\top) d_y = r_3 - G_k K_\gamma^{-1} r_\gamma \; . \quad (15)$$

Then, we recover the primal descent direction as $K_\gamma d_x = r_\gamma - G_k^\top d_y$. The method is tractable for two main reasons. First, the matrix K_γ is positive definite, meaning it can be factorized efficiently without numerical pivoting. Second, the Schur complement system (15) can be solved using a conjugate gradient algorithm converging in only a few iterations. In fact, the eigenvalues of $S_\gamma := G_k K_\gamma^{-1} G_k^\top$ converge to $\frac{1}{\gamma}$ as $\gamma \to +\infty$ [26], implying that the conditioning of S_γ converges to 1 (a setting particularly favorable for iterative methods).

However, the conditioning of K_γ increases with the parameter γ and the values in the diagonal matrix D_s. Fortunately, the impact of the ill-conditioning remains limited in IPM, and we can recover accurate solution when solving the system (15) (see e.g. [34]).

V. IMPLEMENTATION

We have implemented the algorithm in the Julia language, using the modeler ExaModels and the interior-point solver MadNLP [18]. Except for a few exceptions, all the array data is exclusively resident on the device memory, and the algorithm has been designed to run fully on the GPU to avoid expensive data transfers between the host and the device.

A. Evaluation of the model with ExaModels

Despite being nonlinear, the problem (1) usually has a highly repetitive structure that eases its evaluation. CasADi allows for fast evaluation of problems with dynamic structures [8] but is not compatible with GPU. Instead, we use the modeler ExaModels [18], which detects the repeated patterns inside a nonlinear program to evaluate them in a vectorized fashion using SIMD parallelism. Using the multiple dispatch feature of Julia, ExaModels generates

highly efficient derivative computation code, compiled for each computational pattern found in the model. Derivative evaluation is implemented via array and kernel programming in the Julia Language, using the wrapper CUDA.jl to dispatch the evaluation on the GPU.

B. Hybrid linear solver

The two KKT solvers introduced in §IV, Lifted-KKT and HyKKT, both require a sparse Cholesky factorization and an iterative routine. We use the sparse Cholesky solver cuDSS [35], recently released by NVIDIA. The most expensive operation in cuDSS is the computation of the symbolic factorization. However, it is important to note that the symbolic factorization can be only computed once and refactorized efficiently if the matrix's sparsity pattern remains the same. This setting is particularly favorable for IPM, as the sparsity pattern of the condensed matrix K_k is fixed throughout the iterations. Furthermore, within MPC framework, the symbolic factorization can be performed offline, and thus, does not affect the online computation time.

Lifted-KKT factorizes the matrix K_k using cuDSS, and uses the resulting factor to solve the linear system (13) using a backsolve. The matrix K_k becomes increasingly ill-conditioned as we approach the solution. Hence, it has to be refined afterwards using an iterative refinement algorithm, to increase the accuracy of the descent direction. We use Richardson iterations in the iterative refinement, as in [19]. Lifted-KKT sets the relaxation parameter to $\tau = 10^{-6}$.

HyKKT also factorizes the matrix K_γ with cuDSS. The resulting factor is used afterwards to evaluate the residual r_γ and solve the Schur complement system (15) using a conjugate gradient algorithm (we only evaluate matrix-vector products $S_\gamma x$ to avoid computing the full matrix S_γ). HyKKT leverages the conjugate gradient algorithm implemented in the GPU-accelerated library Krylov.jl [36] to solve the system (15) entirely on the GPU. As the conditioning of the Schur complement S_γ improves with the parameter γ, the CG method converges in less than 10 iterations on average, and does not require a preconditioner. In practice, we set $\gamma = 10^7$.

C. GPU-accelerated interior-point solver

The two KKT solvers Lifted-KKT and HyKKT have been implemented inside MadNLP [18], a nonlinear solver implementing the filter line-search interior-point method [19] in pure Julia. MadNLP builds upon the library CUDA.jl to dispatch the operations seamlessly on the GPU, and leverages the libraries cuSPARSE and cuBLAS for basic linear algebra operations. The assembling of the two matrices K_k and K_γ occurs entirely on the GPU, using a custom GPU kernel.

VI. NUMERICAL RESULTS

We assess the performance of Lifted-KKT and HyKKT on the GPU, by comparing them with the performance we obtain with the HSL solvers ma27 and ma57 on the CPU.

A. Optimization of a distillation column

As a test case, we use the classical distillation column instance from [7], here implemented with ExaModels. The size of the discretization grid is parameterized by N. The distillation column has 32 trays. The vapor-liquid equilibrium equations are encoded with algebraic equations, the evolution of the material balances on each tray with differential equations. We note $\alpha = 1.6$ the constant relative volatility, L_t the liquid flow rate in the rectification section, S_t the liquid flow rate in the stripping section, $x_{n,t}$ the liquid-phase mole fraction at tray n and time t, $y_{n,t}$ the vapor-phase mole fraction, u_t the reflux ratio, $D = 0.2$ the constant distillate flow rate, $F = 0.4$ the constant feed flow rate. The objective weights are $\gamma = 1000$ and $\rho = 1$. The problem minimizes the quadratic deviation from the setpoints $(\bar{x}_1, \bar{u})$:

$$\min \sum_{t=1}^{N} \left(\gamma(x_{1,t} - \bar{x}_1)^2 + \rho(u_t - \bar{u})^2\right)$$

subject to, for all $t = 1, \cdots, N$ and for a fixed time-step $\Delta t := 10/N$,

$$
\begin{aligned}
& L_t = u_t D\,,\ V_t = L_t + D\,,\ S_t = F + L_t\,, \\
& y_{n,t} = \frac{\alpha x_{n,t}}{1 + (\alpha - 1)x_{n,t}}\,, \quad \forall n \in \{1, .., 32\}\,, \\
& \dot{x}_{1,t} = \frac{1}{M_1} V_t (y_{2,t} - x_{1,t}) \\
& \dot{x}_{n,t} = \frac{1}{M_n}\big(L_t(x_{n-1,t} - x_{n,t}) - V_t(y_{n,t} - y_{n+1,t})\big) \\
& \quad \forall n \in \{2, .., 16\}\,, \\
& \dot{x}_{17,t} = \frac{1}{M_{17}}\big(F x_f + L_t x_{16,t} - S_t x_{17,t} - V_t(y_{17,t} - y_{18,t})\big) \\
& \dot{x}_{n,t} = \frac{1}{M_n}\big(S_t(x_{n-1,t} - x_{n,t}) - V_t(y_{n,t} - y_{n+1,t})\big) \\
& \quad \forall n \in \{18, .., 31\}\,, \\
& \dot{x}_{32,t} = \frac{1}{M_{32}}\big(S_t(x_{31,t} - (F - D_t)x_{32,t} - V_t y_{32,t}\big) \\
& \dot{x}_{n,t} = \frac{1}{\Delta t}\big(x_{n,t} - x_{n,t-1}\big)\,, \quad \forall n \in \{1, .., 32\}\,, \\
& x_{n,0} = \bar{x}_{n,0}\,,\ 1 \le u_t \le 5\,,
\end{aligned}
$$

B. Results

We use our local workstation, equipped with an AMD Epyc 7443 (24-core, 3.1GHz) and a NVIDIA A30 (24GB of local memory). [2]

1) Performance of the cuDSS solver: We start by assessing the performance of the sparse Cholesky solver cuDSS [35]. We use MadNLP to generate the condensed matrix K_k for the distillation column instance, for different discretization sizes N. We benchmark individually (i) the time to perform the symbolic analysis, (ii) the time to refactorize the matrix, (iii) the time to compute the backsolve. The results are displayed in Table I. We note that for the largest instance ($N = 50,000$) we have more than 3.3M

[2] A script to reproduce the results is available at `https://github.com/exanauts/nlp-on-gpu-paper/tree/main/HybridKKT.jl/benchmarks/cdc`.

variables n in the optimization problem, but only 0.0002% of nonzeros in the sparse matrix K_k. We observe that cuDSS spends most of its time in the symbolic analysis: once the symbolic factorization is computed, the refactorization and the backsolve are surprisingly fast (less than 0.5 seconds to recompute the factorization of the largest instance). In other words, the costs of the symbolic factorization is amortized as soon as we use many refactorizations afterward. Furthermore, during the online computation within control systems, the symbolic factorization can simply be reused and does not affect the online computation time.

TABLE I

PERFORMANCE OF THE LINEAR SOLVER CUDSS

N	n	nnz	SYM (s)	FAC (s)	SOLVE (s)
100	6,767	83,835	0.058	0.003	0.001
500	33,567	418,635	0.209	0.011	0.004
1,000	67,067	837,135	0.381	0.019	0.006
5,000	335,067	4,185,135	2.185	0.072	0.022
10,000	670,067	8,370,135	4.396	0.115	0.037
20,000	1,340,067	16,740,135	9.057	0.220	0.072
50,000	3,350,067	33,450,135	20.187	0.432	0.165

*SYM, FAC, and SOLVE are the time spent in the symbolic analysis, refactorization, and backsolve, respectively (all displayed in seconds).

2) Performance of the MadNLP solver: We analyze now the performance in the MadNLP solver, using both the Lifted-KKT and HyKKT hybrid solvers together with cuDSS. As a baseline, we give the time spent in the HSL solvers ma27 and ma57. We set MadNLP tolerance to `tol=1e-6`, and look at the time spent to achieve convergence in IPM. The results are displayed in Table II. We observe that MadNLP converges in the same number of iterations with HSL ma27, HSL ma57 and HyKKT, as these KKT solvers all solve the same KKT system (6). On its end, Lifted-KKT solves the relaxed problem (12) and requires twice as much iterations to achieve convergence. In concordance with Table I, the pre-processing times (column `init`) are significant for Lifted-KKT and HyKKT. Computing the symbolic factorization with cuDSS is the bottleneck: in comparison, ma27 is approximately twice as fast during the pre-processing. However, the time spent in the symbolic factorization is amortized throughout the IPM iterations: Lifted-KKT and HyKKT solve the problem respectively 26x and 18x faster than HSL ma27 on the largest instance ($N = 50,000$).

We display the time per IPM iteration in Figure 1. We observe that Lifted-KKT is slightly faster than HyKKT, as the performance of the later method depends on the total number of CG iterations required to solve the system (15) at each IPM iteration. The solver HSL ma27 is consistently better than ma57 on that particular instance, as the problem is highly sparse.

VII. CONCLUSIONS AND FUTURE WORKS

A. Conclusions

In this paper, we have presented two hybrid solvers to solve the sparse KKT systems arising in IPM on the GPU:

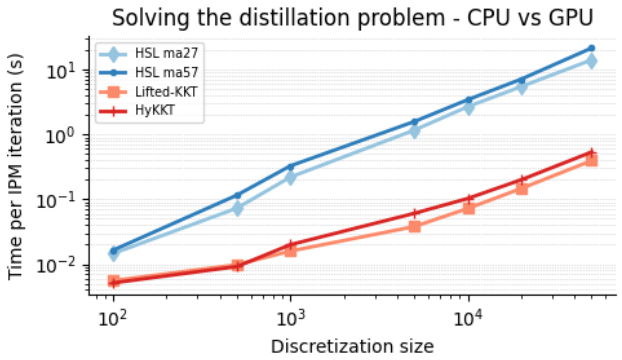

Fig. 1. Time per IPM iteration (s), CPU versus GPU.

Lifted-KKT and HyKKT. We have implemented the two hybrid solvers within MadNLP, using the newly released cuDSS linear solver to compute the sparse Cholesky factorization on the GPU. Our results on the distillation column instance show that once the symbolic factorization is computed, both Lifted-KKT and HyKKT are significantly faster than HSL running on the CPU, with a time per iteration reduced by a factor of 25 on the largest instance. This setting is relevant for NMPC, as the symbolic factorization can be computed only once and reused when we solve the problem a second time with updated data.

B. Future Works

In this article, the dynamic structure has not been exploited explicitly. However, it is known that for dynamic nonlinear programs, the condensed matrix K_k is block-banded and can be factorized efficiently using a block-structured linear solver. Indeed, we can interpret K_k as a matrix encoding a linear-quadratic program and solve the resulting problem in parallel using partitioned dynamic programming [5]. We are planning to investigate this promising outlook in future research.

REFERENCES

[1] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear MPC and moving horizon estimation," *Nonlinear model predictive control: towards new challenging applications*, pp. 391–417, 2009.

[2] C. Kirches, L. Wirsching, S. Sager, and H. G. Bock, "Efficient numerics for nonlinear model predictive control," in *Recent Advances in Optimization and its Applications in Engineering: The 14th Belgian-French-German Conference on Optimization*, pp. 339–357, Springer, 2010.

[3] J. C. Dunn and D. P. Bertsekas, "Efficient dynamic programming implementations of Newton's method for unconstrained optimal control problems," *Journal of Optimization Theory and Applications*, vol. 63, no. 1, pp. 23–38, 1989.

[4] V. M. Zavala, C. D. Laird, and L. T. Biegler, "Interior-point decomposition approaches for parallel solution of large-scale nonlinear parameter estimation problems," *Chemical Engineering Science*, vol. 63, no. 19, pp. 4834–4845, 2008.

[5] S. J. Wright, "Partitioned dynamic programming for optimal control," *SIAM Journal on Optimization*, vol. 1, no. 4, pp. 620–642, 1991.

[6] C. V. Rao, S. J. Wright, and J. B. Rawlings, "Application of interior-point methods to model predictive control," *Journal of Optimization Theory and Applications*, vol. 99, no. 3, pp. 723–757, 1998.

TABLE II

PERFORMANCE COMPARISON OF MADNLP ON CPU AND GPU.

	HSL ma27				HSL ma57				Lifted-KKT				HyKKT			
N	init	AD	linsolve	**total**	init	AD	linsolve	**total**	init	AD	linsolve	**total**	init	AD	linsolve	**total**
100	0.0	0.0	0.1	**0.1**	0.0	0.0	0.1	**0.1**	0.1	0.0	0.2	**0.3**	0.1	0.0	0.1	**0.2**
500	0.1	0.0	0.6	**0.7**	0.0	0.0	1.0	**1.0**	0.2	0.0	0.3	**0.6**	0.2	0.0	0.1	**0.3**
1,000	0.1	0.0	1.7	**1.8**	0.6	0.0	2.3	**3.0**	0.5	0.0	0.4	**0.9**	0.4	0.0	0.2	**0.6**
5,000	0.6	0.1	8.8	**9.5**	3.6	0.1	13.2	**16.9**	2.3	0.0	0.5	**2.8**	2.3	0.0	0.4	**2.7**
10,000	1.6	0.2	20.6	**22.4**	7.7	0.2	26.6	**34.4**	4.8	0.0	0.9	**5.8**	4.9	0.0	0.8	**5.7**
20,000	4.6	0.4	40.3	**45.3**	17.4	0.4	59.0	**76.8**	10.2	0.1	2.0	**12.3**	10.6	0.1	2.0	**12.6**
50,000	15.4	0.9	109.1	**125.5**	49.5	0.9	146.3	**196.8**	27.9	0.5	5.4	**33.8**	29.7	0.1	4.6	**34.5**

[7] A. Cervantes and L. T. Biegler, "Large-scale DAE optimization using a simultaneous NLP formulation," *AIChE Journal*, vol. 44, no. 5, pp. 1038–1050, 1998.

[8] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: a software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, pp. 1–36, 2019.

[9] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. v. Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados—a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, vol. 14, no. 1, pp. 147–183, 2022.

[10] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, pp. 327–363, 2014.

[11] J. V. Frasch, S. Sager, and M. Diehl, "A parallel quadratic programming method for dynamic optimization problems," *Mathematical programming computation*, vol. 7, pp. 289–329, 2015.

[12] G. Frison and M. Diehl, "HPIPM: a high-performance quadratic programming framework for model predictive control," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020.

[13] G. Frison, D. Kouzoupis, T. Sartor, A. Zanelli, and M. Diehl, "BLASFEO: Basic linear algebra subroutines for embedded optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 44, no. 4, pp. 1–30, 2018.

[14] E. C. Kerrigan, G. A. Constantinides, A. Suardi, A. Picciau, and B. Khusainov, "Computer architectures to close the loop in real-time optimization," in *2015 54th IEEE conference on decision and control (CDC)*, pp. 4597–4611, IEEE, 2015.

[15] D.-K. Phung, B. Hérissé, J. Marzat, and S. Bertrand, "Model predictive control for autonomous navigation using embedded graphics processing unit," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 11883–11888, 2017.

[16] L. Yu, A. Goldsmith, and S. Di Cairano, "Efficient convex optimization on GPUs for embedded model predictive control," in *Proceedings of the General Purpose GPUs*, (New York, NY, USA), pp. 12–21, Association for Computing Machinery, 2017.

[17] K. M. M. Rathai, M. Alamir, and O. Sename, "GPU based stochastic parameterized NMPC scheme for control of semi-active suspension system for half car vehicle," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 14369–14374, 2020.

[18] S. Shin, M. Anitescu, and F. Pacaud, "Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods," *Electric Power Systems Research*, vol. 236, p. 110651, 2024.

[19] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[20] M. Benzi, G. H. Golub, and J. Liesen, "Numerical solution of saddle point problems," *Acta numerica*, vol. 14, pp. 1–137, 2005.

[21] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear," *ACM Transactions on Mathematical Software (TOMS)*, vol. 9, no. 3, pp. 302–325, 1983.

[22] B. Tasseff, C. Coffrin, A. Wächter, and C. Laird, "Exploring benefits of linear solver parallelism on modern nonlinear optimization applications," *arXiv preprint arXiv:1909.08104*, 2019.

[23] K. Świrydowicz, E. Darve, W. Jones, J. Maack, S. Regev, M. A. Saunders, S. J. Thomas, and S. Peleš, "Linear solvers for power grid optimization problems: a review of GPU-accelerated linear solvers," *Parallel Computing*, p. 102870, 2021.

[24] D. Cole, S. Shin, F. Pacaud, V. M. Zavala, and M. Anitescu, "Exploiting GPU/SIMD architectures for solving linear-quadratic MPC problems," in *2023 American Control Conference (ACC)*, pp. 3995–4000, IEEE, 2023.

[25] F. Pacaud, S. Shin, M. Schanen, D. A. Maldonado, and M. Anitescu, "Accelerating condensed interior-point methods on SIMD/GPU architectures," *Journal of Optimization Theory and Applications*, pp. 1–20, 2023.

[26] S. Regev, N.-Y. Chiang, E. Darve, C. G. Petra, M. A. Saunders, K. Świrydowicz, and S. Peleš, "HyKKT: a hybrid direct-iterative method for solving KKT linear systems," *Optimization Methods and Software*, vol. 38, no. 2, pp. 332–355, 2023.

[27] G. H. Golub and C. Greif, "On solving block-structured indefinite linear systems," *SIAM Journal on Scientific Computing*, vol. 24, no. 6, pp. 2076–2092, 2003.

[28] J. Nocedal and S. J. Wright, *Numerical optimization*. Springer series in operations research, New York: Springer, 2nd ed., 2006.

[29] N.-Y. Chiang and V. M. Zavala, "An inertia-free filter line-search algorithm for large-scale nonlinear programming," *Computational Optimization and Applications*, vol. 64, pp. 327–354, 2016.

[30] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl, "A sparsity preserving convexification procedure for indefinite quadratic programs arising in direct optimal control," *SIAM Journal on Optimization*, vol. 27, no. 3, pp. 2085–2109, 2017.

[31] W. Wan and L. T. Biegler, "Structured regularization for barrier NLP solvers," *Computational Optimization and Applications*, vol. 66, pp. 401–424, 2017.

[32] D. Thierry and L. Biegler, "The $\ell 1$—exact penalty-barrier phase for degenerate nonlinear programming problems in Ipopt," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6496–6501, 2020.

[33] G. Debreu, "Definite and semidefinite quadratic forms," *Econometrica: Journal of the Econometric Society*, pp. 295–300, 1952.

[34] M. H. Wright, "Ill-conditioning and computational error in interior methods for nonlinear programming," *SIAM Journal on Optimization*, vol. 9, no. 1, pp. 84–111, 1998.

[35] NVIDIA cuDSS documentation, "NVIDIA cuDSS (Preview): A high-performance CUDA Library for Direct Sparse Solvers." `https://docs.nvidia.com/cuda/cudss/index.html`.

[36] A. Montoison and D. Orban, "Krylov. jl: A julia basket of hand-picked krylov methods," *Journal of Open Source Software*, vol. 8, no. 89, p. 5187, 2023.