Parallel Structure-Exploiting Multistage Optimization in PIQP

Fenglong Song, Roland Schwan, Yuwen Chen, and Colin N. Jones
Automatic Control Laboratory, EPFL, Lausanne, Switzerland

***Abstract*—This paper presents a parallel Cholesky factorization and triangular solve algorithm for the Karush–Kuhn–Tucker (KKT) systems arising in multistage optimization problems, with a focus on model predictive control and trajectory optimization. The proposed approach exposes parallelism directly at the linear-algebra level when solving block-tridiagonal-arrow KKT systems arising in interior-point methods. The algorithm is implemented as a new backend of the PIQP solver and released as open source. Numerical experiments on the chain-of-masses benchmarks and a minimum-curvature race line optimization problem demonstrate substantial performance gains compared to other state-of-the-art solvers.**

I. Introduction

Optimal control problems (OCPs), which arise in applications such as robotic trajectory optimization, must often be solved in real time, as the sampling periods are on the order of milliseconds. This stringent requirement makes computational efficiency a critical challenge in solver design. Fortunately, OCPs often exhibit sparse and structured formulations, particularly when discretized from continuous-time ordinary differential equations (ODEs) using multiple shooting or direct collocation that yield block tridiagonal Karush-Kuhn-Tucker (KKT) systems. A rich class of solvers that exploits such temporal sparsity has been developed in [2]–[5], with a computational complexity proportional to the horizon length N, i.e. $\mathcal{O}(N)$, under mild assumptions. These methods typically involve solving block-sparse KKT systems via sparse Cholesky or condensed Riccati-based factorizations, which are inherently done sequentially.

Recently, several studies have explored parallelism for the temporal sparsity pattern within OCP solvers. From a dynamic programming perspective, [6] proposes to solve the unconstrained linear quadratic OCP with the parallel scan algorithm [7], making $\mathcal{O}(\log N)$ complexity achievable when a sufficient number of parallel computing units are available. In addition, other works perform parallelization directly at the linear algebra level when solving the KKT system, such as parallel cyclic reduction [8]. A parallel Riccati-recursion is proposed in [9], combined with a dedicated data structure that facilitates vectorized batched dense linear algebra operations, further improving throughput.

The work most closely related to ours is [10]. In contrast to their focus on block-tridiagonal matrices, we consider the more general block-tridiagonal-arrow structure and use a different parallelization strategy. Furthermore, while [10] targets GPU implementations, we focus on multi-core CPU architectures. A corresponding GPU-based extension of our approach is presented in [11].

This paper is a concise version of the authors' previous work [1].

In this work, we introduce parallelization *directly at the linear-algebra level* for solving the KKT systems without altering the outer optimization algorithm. This design accelerates the computation while preserving the numerical robustness and theoretical properties of the underlying optimization method. Specifically, we extend the parallel Cholesky factorization scheme of [12] to efficiently handle block-tridiagonal–arrow KKT matrices that commonly arise in multistage optimization problems. The proposed parallelization strategy is general and can be integrated into a wide range of quadratic programming solvers that exploit the intrinsic OCP structure.

Our main contributions are summarized as follows:

- We extend the parallel Cholesky factorization for block-tridiagonal matrices in [12] to support block-tridiagonal-arrow matrices, which allows dealing with more general problem types. We also extend the parallelism for the forward and backward substitutions, enabling thread-safe parallelism without race conditions.
- We analyze the computational complexity of the proposed parallel method, derive an optimal strategy for distributing workload across multiple threads to maximize parallel efficiency, and quantify the theoretically achievable speedups relative to the sequential factorization in [13].
- We provide an open-source implementation of the proposed method and integrate it as a new backend in the PIQP solver from [14]. We show that PIQP with our multi-threaded KKT system solver outperforms state-of-the-art solvers PIQP, HPIPM [3], and Clarabel [15] through numerical experiments.

Notation: We denote the set of real numbers by $\mathbb{R}$, the set of positive integers by $\mathbb{N}_+$, the set of n-dimensional real-valued vectors by $\mathbb{R}^n$, and the set of $n \times m$-dimensional real-valued matrices by $\mathbb{R}^{n\times m}$. The floor and ceiling operators are denoted by $\lfloor\cdot\rfloor$ and $\lceil\cdot\rceil$, respectively. For symmetric matrices, we use $\star$ to represent the entries (or blocks) in the upper triangular part for brevity. Finally, D_{*k} denotes the collection of all D_{ik} with a fixed index k.

II. Problem Formulation

We consider the following QP formulation following [13]:

$$\begin{aligned}
\min_{x,g} \quad & \sum_{i=0}^{N-1} \ell_i(x_i, x_{i+1}, g) + \ell_N(x_N, g) \\
\text{s.t.} \quad & \bar{A}_i x_i + \bar{B}_i x_{i+1} + \bar{E}_i g = \bar{b}_i, \;\; i = 0, \dots, N-1, \\
& \bar{C}_i x_i + \bar{D}_i x_{i+1} + \bar{F}_i g \leq \bar{h}_i, \;\; i = 0, \dots, N-1, \\
& \bar{A}_N x_N + \bar{E}_N g = \bar{b}_N, \\
& \bar{D}_N x_N + \bar{F}_N g \leq \bar{h}_N,
\end{aligned} \tag{1}$$

with coupled stage cost from stage 0 to $N-1$

$$\ell_i(x_i, x_{i+1}, g) \coloneqq \frac{1}{2}\begin{bmatrix} x_i \\ x_{i+1} \\ g \end{bmatrix}^\top \begin{bmatrix} \bar{Q}_i & \bar{S}_i^\top & \bar{T}_i^\top \\ \bar{S}_i & 0 & 0 \\ \bar{T}_i & 0 & 0 \end{bmatrix} \begin{bmatrix} x_i \\ x_{i+1} \\ g \end{bmatrix} + \bar{c}_i^\top x_i,$$

and terminal cost at stage N

$$\ell_N(x_N, g) \coloneqq \frac{1}{2}\begin{bmatrix} x_N \\ g \end{bmatrix}^\top \begin{bmatrix} \bar{Q}_N & \bar{T}_N^\top \\ \bar{T}_N & \bar{Q}_g \end{bmatrix} \begin{bmatrix} x_N \\ g \end{bmatrix} + \bar{c}_N^\top x_N + \bar{c}_g^\top g,$$

where $x_i \in \mathbb{R}^{n_i}$ are the stage-wise decision variables, $g \in \mathbb{R}^{n_g}$ is a global decision variable, and $N \in \mathbb{N}_+$ is the horizon.

The formulation (1) is more general than the classical OCP formulation since it can describe the inter-stage coupling not only through the system dynamics but also other general inter-stage costs and constraints. Moreover, it can easily capture the structure where global variables are present, e.g., time-optimal MPC and scenario-based robust/stochastic MPC.

Solving the multistage optimization problem (1) via PIQP [14] includes solving the following Karush-Kuhn-Tucker (KKT) system in each iteration:

$$\Psi \Delta x = r, \tag{2}$$

where

$$\Psi \coloneqq \begin{bmatrix} \Psi_{0,0} & \Psi_{1,0}^\top & 0 & \cdots & \Psi_{g,0}^\top \\ \Psi_{1,0} & \Psi_{1,1} & \Psi_{1,2}^\top & \ddots & \Psi_{g,1}^\top \\ 0 & \Psi_{1,2} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \Psi_{N,N} & \Psi_{g,N}^\top \\ \Psi_{g,0} & \Psi_{g,1} & \cdots & \Psi_{g,N} & \Psi_{g,g} \end{bmatrix}, \tag{3}$$

with $\Psi_{i,i} \in \mathbb{R}^{n_i \times n_i}$, $\Psi_{g,g} \in \mathbb{R}^{n_g \times n_g}$, $\Psi_{i+1,i} \in \mathbb{R}^{n_{i+1} \times n_i}$ and $\Psi_{g,i} \in \mathbb{R}^{n_g \times n_i}$. The KKT matrix Ψ is symmetric positive definite (SPD) and has a structured block-tridiagonal-arrow form in (3). While the detailed derivation is omitted here for brevity, interested readers are referred to [13] for the complete algorithmic framework.

Computing the solution of the KKT system (2) typically constitutes the computational bottleneck in quadratic programming (QP) solvers. In the next section, we present a novel parallel algorithm leveraging parallel computing platforms to accelerate computation related to solving (2).

III. Methodology

A. Parallel Cholesky Factorization

The core idea for exposing parallelism in the factorization of the block-tridiagonal-arrow KKT matrix is to break the inter-block dependencies along the time or stage dimension. We partition the original KKT matrix Ψ into groups of consecutive stages, as illustrated in Fig. 1a. The matrix Ψ exhibits strong coupling between neighboring segments of $\{D_{*k}, E_{*k}, G_{*k}\}$ through the coupling blocks F_k, A_{k+1}, B_{k+1}, and Q_{k+1}.

To break such coupling, we use a proper permutation matrix P to obtain $\hat{\Psi} \coloneqq P\Psi P^\top$ as shown in Figure 1b. With the permutation, we reorder the coupling blocks $\{F_k, A_{k+1}, B_{k+1}\}$, $1 \le k \le p-1$, to appear in the last rows and columns but before the blocks associated with the global variables. Hence, the processing of the coupling blocks is postponed, allowing the independent portions of the matrix, i.e., the operations related to $D_{*k}, E_{*k}, G_{*k}, B_k, F_k$ for each k on an individual thread, to be factorized in parallel. The remaining coupled part (the bottom-right block) is processed serially last. This leads to a two-phase routine, including:

- a *parallel* phase, in which operations are distributed among p threads for parallel computation, and
- a *sequential* phase, in which the remaining coupled part (bottom-right block in Figure 1b) is processed serially.

The detailed derivations are omitted here for brevity. We refer interested readers to [1] for more details.

B. Parallel Triangular Solve

Once the Cholesky factor $\hat{L}$ is computed such that $\hat{\Psi} = \hat{L}\hat{L}^\top$, we solve the linear system (2) via the permuted system:

$$\underbrace{P\Psi P^\top}_{\hat{\Psi}} \underbrace{P\Delta x}_{\Delta\hat{x}} = \underbrace{Pr}_{\hat{r}},$$

where $\Delta\hat{x}$ can be computed via a standard forward-backward substitution:

$$\hat{L}\Delta\hat{y} = \hat{r}, \quad \hat{L}^\top \Delta\hat{x} = \Delta\hat{y}.$$

Thanks to this dedicated permutation, parallelism is inherently exposed during both substitution phases. Because the factorized matrix $\hat{L}$ is lower triangular and retains the decoupled block structure in its upper-left region, the forward substitution begins with a *parallel phase*. The independent upper segments of $\Delta\hat{y}$ are computed concurrently across the p threads, after which a *sequential phase* is executed to solve the bottom-right coupled portion of $\Delta\hat{y}$.

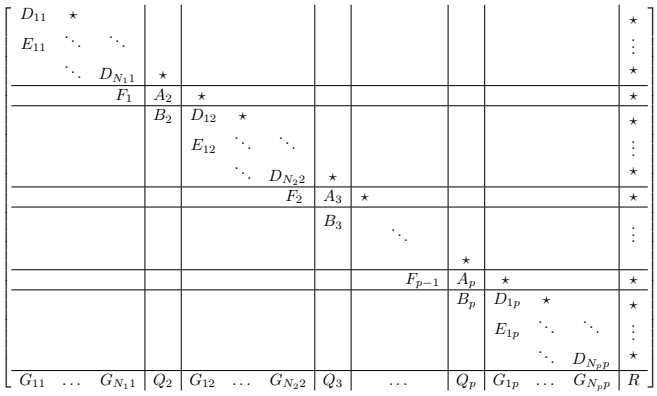
Conversely, the backward substitution reverses this order. Because $\hat{L}^\top$ is upper triangular, the bottom-right coupled variables of $\Delta\hat{x}$ must be solved first sequentially. Once computed, these shared variables are substituted back into the upper equations, completely decoupling the remaining system and allowing the upper segments of $\Delta\hat{x}$ to be solved in parallel.

C. Flop Analysis and Optimal Partitioning

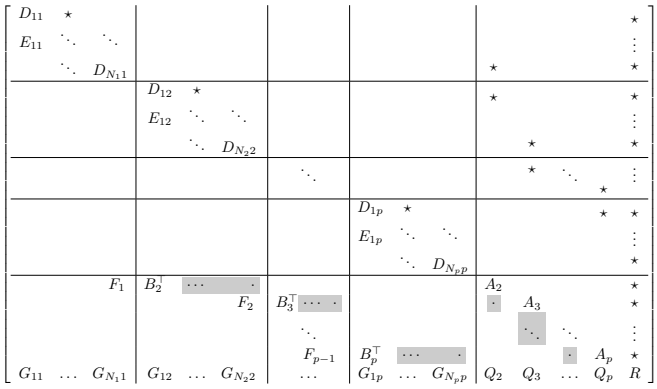
In the parallel phase, assuming a uniform block size b and no global variables, the first segment does not cause fill-in blocks, requiring $(\frac{7}{3}-1)N_1 b^3$ flops. In contrast, any k-th segment ($1 < k < p$) must also compute and update dense fill-in blocks, increasing its cost to $(\frac{19}{3}N_k - 1)b^3$ flops. Because Cholesky factorization heavily dominates the total computational effort, we balance the workload across all p threads by equating these costs:

$$\left(\frac{7}{3}N_1 - 1\right)b^3 = \left(\frac{19}{3}N_k - 1\right)b^3 \implies \sigma \coloneqq \frac{N_1}{N_k} = \frac{19}{7}.$$

Thus, the ideal segment length is $\bar{N}_k^* \coloneqq (N-p+1)/(p+\sigma)$, which we round up or down to minimize the maximum thread complexity.

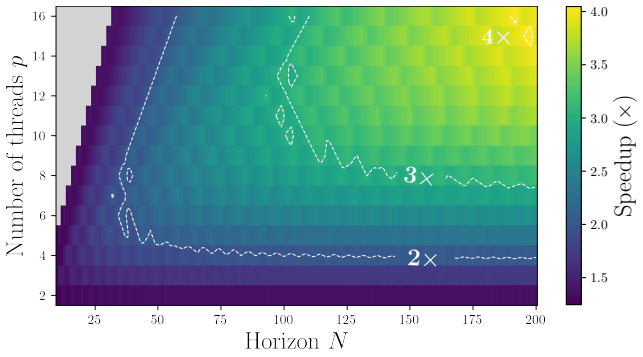


(a) Original KKT matrix Ψ.

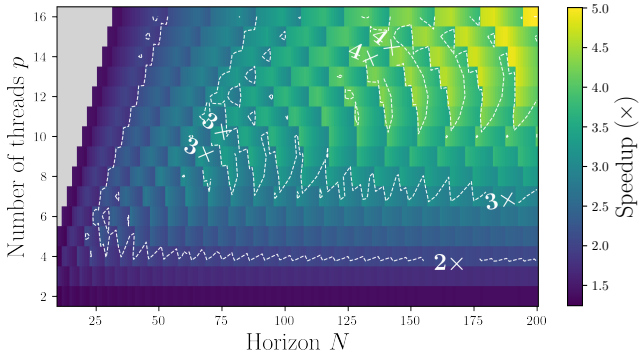


(b) Permuted KKT matrix $\hat{\Psi} := P\Psi P^\top$.

Fig. 1: The partition of original KKT matrix Ψ and the permuted KKT matrix $\hat{\Psi}$. The colored entries in Figure 1b indicate fill-in blocks after performing Cholesky factorization on $\hat{\Psi}$.



(a) Theoretical Speedup for Cholesky factorization



(b) Theoretical Speedup for triangular solve

Fig. 2: Theoretical speedups using our parallel method compared to the sequential method. The gray region indicates the condition $N \geq 2p$ is not satisfied, a constraint required for the parallel method across p threads to be meaningful.

The overall time complexity of our parallel method is:

$$\mathcal{O}_{par}(N;p) = \left(\max\left\{\frac{7}{3}N_1^*, \frac{19}{3}N_k^*\right\} + \frac{10}{3}p - \frac{19}{3}\right) b^3,$$

where N_1^* and N_k^* are the optimally rounded segment lengths. In comparison, the sequential factorization in [13] requires

$$\mathcal{O}_{seq}(N) = \left(\frac{7}{3}N - 2\right) b^3.$$

We illustrate the theoretical speedup of factorization $\gamma := \mathcal{O}_{\text{seq}}(N)/\mathcal{O}_{\text{par}}(N;p)$ in Figure 2 for various numbers of threads $p \in [2, 16]$ and prediction horizons $N \in [10, 200]$. For a fixed number of threads, the speedup exhibits an overall increasing trend with minor fluctuations as the horizon length N grows. These fluctuations arise from changes in the rounding strategy for $\bar{N}_k^*$, i.e., when the rounding switches between floor and ceiling at certain values of N. In addition, the parallel Cholesky factorization and triangular solve must perform additional operations for the fill-in blocks, which offset the gains from parallelization. As a result, the speedup tends to saturate once the horizon N becomes sufficiently long.

IV. Numerical Results

We have implemented the proposed factorization and triangular solve routines as a new backend within the solver PIQP [14]. The implementation utilizes C/C++ with the Eigen3 library while leveraging BLASFEO [16] for optimized linear algebra operations and OpenMP for multi-thread parallelization in the new backend. Additionally, we parallelize other naturally parallel operations, including data copies as well as block-wise matrix–matrix and matrix–vector multiplications in data preparation.

We benchmark our algorithm with the sequential multistage solver from PIQP [13] and HPIPM [3]. To assess the impact of hardware-level optimizations, all solvers are compiled with the `AVX2` instruction set enabled on x86 architecture for fair comparison. All experiments are conducted on an AMD Ryzen 9 3900X processor with 12 cores in a single socket. The turbo boost is disabled to ensure consistent thermal conditions. In addition, we bind OpenMP threads to physical cores to improve performance.

In the remainder of this section, we refer to the PIQP implementation with the general sparse KKT solver in [14]

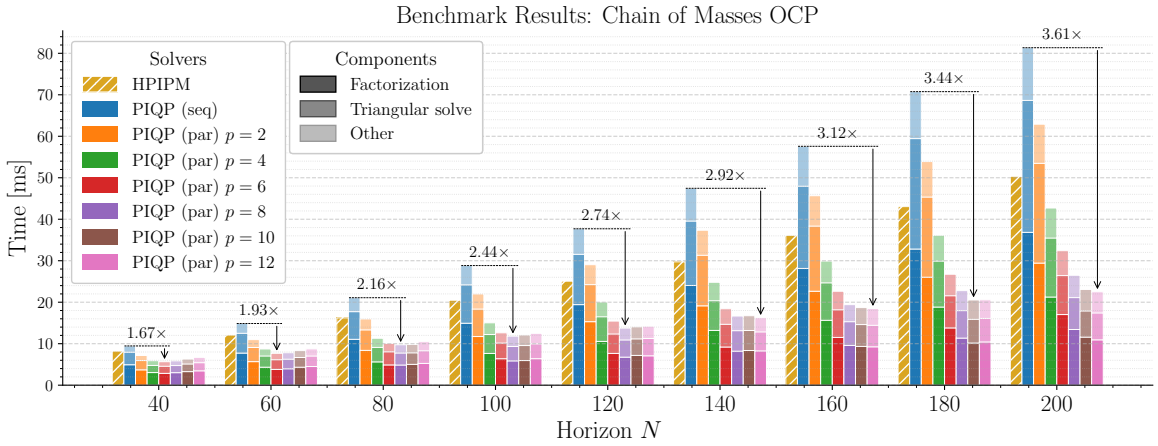


Fig. 3: Computation time of PIQP variants and HPIPM for the chain of masses OCP benchmark. The stacked bars show the solver time decomposition into factorization, triangular solve, and other components, distinguished by transparency levels. The arrows indicate the maximum speedup achieved by the fastest PIQP (par) configuration over the PIQP (seq) baseline.

TABLE I: Runtimes (mean ± std, in ms) and speedups relative to PIQP (seq) for total solver run, factorization, triangular solve, and other components in the race line optimization problem. Bold values indicate the fastest result within each category.

Solver	Total		Factorization		Triangular solve		Other	
	Time [ms]	Speedup	Time [ms]	Speedup	Time [ms]	Speedup	Time [ms]	Speedup
PIQP (seq)	137.92 ± 1.43	1.0×	51.81 ± 0.31	1.0×	63.59 ± 1.07	1.0×	22.52 ± 0.24	1.0×
PIQP (par) $p = 2$	98.99 ± 0.54	1.39×	38.80 ± 0.17	1.34×	43.77 ± 0.31	1.45×	16.43 ± 0.14	1.37×
PIQP (par) $p = 4$	68.62 ± 0.86	2.01×	26.29 ± 0.25	1.97×	28.72 ± 0.50	2.22×	13.62 ± 0.21	1.65×
PIQP (par) $p = 6$	56.48 ± 0.48	2.44×	**20.40 ± 0.30**	**2.54×**	23.00 ± 0.26	2.77×	13.07 ± 0.13	1.72×
PIQP (par) $p = 8$	53.00 ± 0.57	2.60×	20.43 ± 0.34	2.54×	20.30 ± 0.29	3.13×	12.27 ± 0.13	1.84×
PIQP (par) $p = 10$	50.92 ± 0.47	2.71×	20.79 ± 0.48	2.49×	18.26 ± 0.04	3.48×	11.87 ± 0.05	1.90×
PIQP (par) $p = 12$	**50.30 ± 0.53**	**2.74×**	21.10 ± 0.37	2.46×	**17.43 ± 0.17**	**3.65×**	11.77 ± 0.23	1.91×
PIQP (sparse)	111.40 ± 1.32	1.24×	59.32 ± 0.32	0.87×	44.14 ± 0.76	1.44×	**7.94 ± 0.22**	**2.84×**
Clarabel	252.77 ± 1.58	0.55×	—	—	—	—	—	—

as *PIQP (sparse)*, the one with the sequential multistage KKT solver in [13] as *PIQP (seq)*, and to our implementation of the proposed parallel multistage KKT solver as *PIQP (par)*.

A. Chain of Masses Problems

We evaluate the performance of solvers on the chain-of-masses system from [17], following the same setup as in [13]. A system with M masses includes $2M$ states and $M - 1$ inputs. We set $M = 20$ and vary the prediction horizon $N \in \{40, 60 \ldots, 200\}$ and the number of threads $p \in \{2, 4, \ldots, 12\}$.

Computation times for the chain-of-masses OCP with varying horizons N. The stacked bars show the solver time decomposition into factorization, triangular solve, and other components. The arrows indicate the maximum speedup achieved by the fastest PIQP (par) configuration over the PIQP (seq) baseline. For HPIPM, only its total solver runtimes are reported because its internal timings are not exposed.

Figure 3 reports the computation times of multiple variants of PIQP and the state-of-the-art solver HPIPM averaged over 30 runs, as well as the decomposed timings. The results show that the proposed parallel solver achieves substantial acceleration. For a horizon of $N = 200$ and $p = 12$ threads, PIQP (par) achieves a $3.61\times$ overall speedup over PIQP (seq). Compared to state-of-the-art solver HPIPM [3], PIQP (par) still achieves up to a $2.24\times$ speedup, demonstrating the effectiveness of the proposed method.

B. Minimum-curvature Race Line Optimization

We consider the minimum-curvature race line optimization problem similar to [18], formulated as a multistage QP. The trajectory is parameterized by cubic splines, where we directly optimize the spline coefficients to minimize the sum of squared curvatures along the track. Crucially, enforcing C^2 continuity for a closed circuit introduces global variables to the system. This coupling yields an arrow-shaped KKT matrix, as opposed to the standard block-tridiagonal structure, which cannot be efficiently handled by solvers such as HPIPM.

We apply our method to compute the minimum curvature race line for the Silverstone Formula One race track that is approximately 5.89km long and divided into 2356 segments, leading to $N = 2356$. Table I summarizes the computation times of PIQP variants and Clarabel [15]. As p increases, both the factorization and triangular-solve stages exhibit substantial speedups, leading to an overall improvement of up to $2.74\times$ at $p = 12$. The factorization and triangular solve achieve $2.46\times$ and $3.65\times$ speedup, respectively, compared to the sequential baseline.

REFERENCES

[1] F. Song, R. Schwan, Y. Chen, and C. N. Jones, "Parallel KKT solver in PIQP for multistage optimization," 2025.

[2] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen, "High-performance small-scale solvers for linear model predictive control," in *IEEE European Control Conference (ECC)*, 2014, pp. 128–133.

[3] 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.

[4] K. F. Løwenstein, D. Bernardini, and P. Patrinos, "QPALM-OCP: A newton-type proximal augmented lagrangian solver tailored for quadratic programs arising in model predictive control," *IEEE Control Systems Letters*, vol. 8, pp. 1349–1354, 2024.

[5] L. Vanroye, A. Sathya, J. De Schutter, and W. Decré, "Fatrop: A fast constrained optimal control problem solver for robot trajectory optimization and control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023, pp. 10 036–10 043.

[6] S. Sarkka and A. F. Garcia-Fernandez, "Temporal parallelization of dynamic programming and linear quadratic control," *IEEE Transactions on Automatic Control*, vol. 68, no. 2, pp. 851–866, Feb. 2023.

[7] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with cuda," in *GPU Gems 3*. Addison-Wesley, 2007, ch. 39.

[8] E. Bertolazzi and D. Stocco, "Parallel cyclic reduction of padded bordered almost block diagonal matrices," *Journal of Computational and Applied Mathematics*, vol. 458, p. 116331, 2025.

[9] P. Pas and P. Patrinos, "Cyqlone: A parallel, high-performance linear solver for optimal control," 2026.

[10] D. Jin, A. Montoison, and S. Shin, "Harnessing batched BLAS/LAPACK kernels on GPUs for parallel solutions of block tridiagonal systems," 2025.

[11] R. Schwan, D. Kuhn, and C. N. Jones, "GPU-accelerated Cholesky factorization of block tridiagonal matrices," 2026.

[12] T. D. Cao, J. F. Hall, and R. A. van de Geijn, "Parallel cholesky factorization of a block tridiagonal matrix," in *International Conference on Parallel Processing Workshop*, 2002, pp. 327–335.

[13] R. Schwan, D. Kuhn, and C. N. Jones, "Exploiting multistage optimization structure in proximal solvers," in *2025 IEEE 64th Conference on Decision and Control (CDC)*, 2025, pp. 4677–4683.

[14] R. Schwan, Y. Jiang, D. Kuhn, and C. N. Jones, "PIQP: A proximal interior-point quadratic programming solver," in *IEEE Conference on Decision and Control (CDC)*, 2023, pp. 1088–1093.

[15] P. J. Goulart and Y. Chen, "Clarabel: An interior-point solver for conic programs with quadratic objectives," 2024.

[16] G. Frison, D. Kouzoupis, T. Sartor, A. Zanelli, and M. Diehl, "BLASFEO: Basic linear algebra subroutines for embedded optimization," *ACM Trans. Math. Softw.*, vol. 44, no. 4, Jul. 2018.

[17] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Transactions on Control Systems Technology*, vol. 18, no. 2, pp. 267–278, 2010.

[18] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, "Minimum curvature trajectory planning and control for an autonomous race car," *Vehicle System Dynamics*, vol. 58, no. 10, pp. 1497–1527, 2020.