ARTICLE TEMPLATE

ParNMPC – A parallel optimization toolkit for real-time nonlinear model predictive control

Haoyang Deng[a] and Toshiyuki Ohtsuka[a]

[a]Department of Systems Science, Graduate School of Informatics, Kyoto University, Kyoto, Japan

ARTICLE HISTORY
Compiled June 17, 2020

ABSTRACT
Real-time optimization for nonlinear model predictive control (NMPC) has always been challenging, especially for fast-sampling and large-scale applications. This paper presents an efficient implementation of a highly parallelizable method for NMPC, called ParNMPC. The implementation details of ParNMPC are introduced, including a dedicated discretization method suitable for parallelization, a framework that unifies search direction calculation done using Newton's method and the parallel method, line search methods for guaranteeing convergence, and a warm start strategy for the interior-point method. To assess the performance of ParNMPC under different configurations, three experiments including a closed-loop simulation of a quadrotor, a real-world control example of a laboratory helicopter, and a closed-loop simulation of a robot manipulator are shown. These experiments show the effectiveness and efficiency of ParNMPC both in serial and parallel.

KEYWORDS
Nonlinear model predictive control; parallel computing; real-time optimization

1. Introduction

Nonlinear model predictive control (NMPC), which is formulated as a finite-horizon optimal control problem (FHOCP), is a powerful and flexible control method due to its ability to handle multi-input multi-output systems, deal explicitly with nonlinearities and constraints, and directly minimize economic costs (Rawlings, Angeli, & Bates, 2012). Compared with its linear counterpart, where the underlying FHOCP is exactly a quadratic program (QP), NMPC requires heavier computation in general. This heavy computation comes from the predictive feature of NMPC, in which a future trajectory is maintained and the number of optimized input and state variables grows with the discrete prediction horizon. Along the prediction horizon, the state trajectory has to be simulated as an initial value problem considering both numerical stability and accuracy. Moreover, first- and second-order derivatives of the cost function, dynamics, and constraints have to be evaluated or estimated online. It is generally needed to solve equations or to perform matrix inversion to make steps toward the optimal trajectory. Furthermore, extra care has to be taken to guarantee convergence of the nonlinear

CONTACT Haoyang Deng. Email: deng.haoyang.23r@st.kyoto-u.ac.jp
CONTACT Toshiyuki Ohtsuka. Email: ohtsuka@i.kyoto-u.ac.jp

program (NLP).

Considerable effort has been devoted in recent years to real-time optimization methods and implementations for NMPC. The so-called continuation/generalized minimal residual (C/GMRES) method was proposed by Ohtsuka (2004), where the solution to the Karush-Kuhn-Tucker (KKT) conditions is traced without any line search or Newton iteration. The linear system is solved by the GMRES method (Saad & Schultz, 1986), enabling Hessian-free operations and fast computation if a limited number of GMRES iterations are performed. The C/GMRES method has been implemented in the AutoGenU toolkit (Ohtsuka, 2004). Another approximate method is the real-time iteration (RTI) scheme (Diehl, Bock, & Schlöder, 2005), which can be seen as the sequential QP (SQP) method with only one QP iteration performed each sampling time. The RTI scheme has been implemented in many toolkits, such as ACADO (Houska, Ferreau, & Diehl, 2011), acados (Verschueren et al., 2018), and MATMPC (Chen, Bruschetta, Picotti, & Beghi, 2019), and efficient QP solvers optimized for MPC, such as qpOASES (Ferreau, Kirches, Potschka, Bock, & Diehl, 2014), HPIPM (Frison, Sørensen, Dammann, & Jørgensen, 2014) and qpDUNES (Frasch, Sager, & Diehl, 2015), can be integrated. An efficient implementation of the interior-point method for NMPC is reported in the FORCES NLP (Zanelli, Domahidi, Jerez, & Morari, 2017) software, which benefits from a structure-exploiting linear system solver. Efficient implementations of first-order methods tailored to NMPC have been reported in, such as, GRAMPC (Englert, Völz, Mesmer, Rhein, & Graichen, 2019), VIATOC (Kalmari, Backman, & Visala, 2015), and FalcOpt (Torrisi, Grammatico, Smith, & Morari, 2018). Generally, the iteration of first-order methods is cheap to perform. For example, GRAMPC conducts a forward state simulation and a backward costate (Lagrange multiplier corresponding to the state equation) simulation. The search direction is calculated by using simple derivatives, and a tailored line search is performed to minimize the cost function. These methods are very efficient, and various applications have been reported. However, applying NMPC in real time for emerging complicated systems, such as robotic systems and power networks, is still challenging. Considering the stagnating single-core performance of the processors, it is no surprise that the computational demand is being shifted to fast-growing parallel computing platforms. However, current state-of-the-art methods and implementations are not tailored to parallel computing, that is, they can only be parallelized to a certain degree. Therefore, according to Amdahl's law (Amdahl, 1967), only a limited performance improvement can be achieved on parallel platforms and there is an increasing demand for high degree-of-parallelism (DOP) methods and efficient implementations thereof.

Parallel computing can be performed on various platforms, such as field programmable gate arrays (FPGAs), graphics processing units (GPUs), and multi-core processors. Efficient FPGA implementations for linear MPC exploiting parallelizable matrix-vector operations have been reported for the interior-point method (Jerez, Constantinides, & Kerrigan, 2011) with a minimum residual linear system solver and first-order methods (Jerez et al., 2014), e.g., Nesterov's fast gradient method (Nesterov, 1983). Due to the inherent complexity of NMPC, processors are advantageous in performing complicated operations, such as nonlinear function evaluations and matrix inversions. Instruction-level parallelism depends on, e.g., single-instruction multiple-data (SIMD) instructions, which are common in many modern multi-core processors and capable of executing vector operations simultaneously. Frison, Kufoalor, Imsland, and Jørgensen (2014) used SIMD instructions to speed up matrix-matrix operations in the interior-point method. Task-level parallelism involves decomposing the optimization problem into pieces and distributing them into different cores of a processor. Methods

with task-level parallelism typically exploit the particular structure of NMPC. Multiple shooting (Bock & Plitt, 1984) based NMPC can have functions, e.g., dynamics and its derivatives, evaluated in parallel intrinsically. Moreover, the banded structure of the KKT matrix can be exploited to reduce the computational complexity from $\mathcal{O}(N)$ to $\mathcal{O}(\log(N))$ with the so-called parallel cyclic reduction method (Soudbakhsh & Annaswamy, 2013), where N is the number of the discretization grid points in the prediction horizon. Another $\mathcal{O}(\log(N))$ method by Nielsen and Axehill (2018) splits the original FHOCP into independent subproblems and constructs a master problem with a smaller size. A similar idea can be found in the work by Kouzoupis, Quirynen, Houska, and Diehl (2016), where the original FHOCP is decoupled into subproblems and a consensus QP is solved to update the coupling dual variables. Instead of decomposition, the so-called advanced-multi-step NMPC proposed by Yang and Biegler (2013) and the parallel precomputation method (Kawakami, Ono, Ohtsuka, & Inoue, 2018) solve in advance future FHOCPs in parallel on the basis of predicted states, which can be seen as a higher level of parallelism. It should be noted that these different levels of parallelism can be combined easily to achieve an additive speed-up.

Unfortunately, there is still a lack of efficient implementations for the parallel optimization of NMPC to the authors' knowledge. Current existing parallel algorithms either require a large problem size, e.g., long prediction horizon, to surpass serial algorithms or depend on careful designs for parallelization, which are not user-friendly for implementation and have difficulty incorporating existing nonlinear optimization techniques to achieve robust numerical performance. In this paper, ParNMPC, which is an effective and efficient parallel implementation for the optimization of NMPC, is presented. ParNMPC is a MATLAB open-source (https://github.com/deng-haoyang/ParNMPC) software, and the parallelization part is implemented by using OpenMP (Dagum & Menon, 1998), which is designed for shared-memory multi-core processors and supported by most of the operating systems. ParNMPC comes with the following features.

- The NMPC problem can be formulated symbolically as easily as other toolkits.
- The DOP is made configurable so that it can be adapted to the rate of convergence and the number of cores, enabling deployment to both single- and multi-core processors.
- Parallel code can be automatically generated.

The parallel search direction calculation is based on our previous work (Deng & Ohtsuka, 2018, 2019), in which a highly parallelizable algorithm with a fast rate of convergence was proposed. The main contributions of this paper are:

- The parallel code generation toolkit ParNMPC is introduced. The primal-dual interior-point method is inherently integrated in ParNMPC to deal with the inequality constraints, and its warm start strategy in the context of NMPC is introduced.
- The search direction calculation procedure is generalized from algebraic operations (Deng & Ohtsuka, 2019) to NLPs so that the existing nonlinear optimization techniques, such as Hessian approximation, regularization, and condensing, can be applied. As a consequence, the method can be applied to a wider class of problems, and a speed up of more than $2\times$ was observed in Section 6.

The performance of ParNMPC is assessed with several challenging applications.

This paper is organized as follows. The NMPC problem is formulated, and a dedicated discretization method suitable for parallelization is introduced in Section 2.

Search direction calculation done using Newton's method and the parallel method are given in a unified framework in Section 3. Section 4 introduces two commonly used line search methods for guaranteeing convergence. Warm start and the barrier strategy are discussed in Section 5. The performance evaluation of ParNMPC is introduced in Section 6. Finally, this paper is summarized in Section 7.

1.1. Notations

Let $v_{(i)}$ be the i-th component of a vector $v \in \mathbb{R}^n$. For a matrix $P \in \mathbb{R}^{n\times n}$, we denote $P > 0$ and $P \geq 0$ as P being positive-definite and positive-semidefinite, respectively. The weighted norm is defined as $\|v\|_P := \sqrt{v^T P v}$. The ℓ_1 norm is defined as $\|v\|_1 := \sum_{i=1}^{n} |v_{(i)}|$. For an optimization variable v, we denote v^k as the value of v at the k-th iteration and v^* as the optimal solution. Let $\odot$ and $\oslash$ be the element-wise product and division, respectively. We denote $|v|$ as the element-wise absolute value of v. For a differentiable function $f(v) : \mathbb{R}^n \to \mathbb{R}^m$, we denote $\nabla_v f \in \mathbb{R}^{m\times n}$ as the Jacobian matrix of f.

2. Problem formulation

Consider a continuous-time nonlinear system governed by the following differential equation:

$$\dot{x}(t) = f\left(u(t), x(t), p(t)\right), \tag{1}$$

where $x \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{n_u}$, and $p \in \mathbb{R}^{n_p}$ are the system state, control input, and given time-dependent parameter, respectively. In this section, a discretized NMPC problem for system (1) is formulated, and its relaxed problem under the framework of the interior-point method is given.

2.1. Discretization

NMPC based on the direct approach (Stryk & Bulirsch, 1992) requires continuous-time system (1) to be discretized. Generally, different discretization methods lead to different discretization accuracies, computational costs, and problem structures. It was shown by Deng and Ohtsuka (2019) that the so-called reverse-time discretization method can yield an NMPC problem with a particular structure, which can be exploited to design a highly parallelizable algorithm. The reverse-time discretization of (1) is given by

$$x_- + \mathscr{F}(u, x, p) = 0, \tag{2}$$

where x_- is the predecessor state. The reverse-time discretization method comes from the fact that the predecessor state x_- can be directly obtained from x. The simplest reverse-time discretization method is the backward Euler method with $\mathscr{F}(u, x, p) = hf(u, x, p) - x$ (h is the step size).

For a given explicit discretization method, its reverse-time variation involves simply applying the discretization backward in time, i.e., with a negative step size. Therefore, the evaluation of the left-hand side of (2) has exactly the same computational cost as

its explicit version. It should be noted that in solving the NMPC problem, the state integration is still propagated forward in time; thus, stable dynamics stay stable when the discretization accuracy is high. However, extra computation is needed to solve (2) for a given value of x_-.

Regarding the discretization accuracy, we show in Proposition A.1 in Appendix A that the discretization accuracy of an explicit method is preserved in its reverse-time variation. Several commonly used Runge-Kutta methods with different orders are available in ParNMPC.

2.2. NMPC

For a prediction horizon $T > 0$, we consider the following N-stage NMPC problem based on discretized system (2) with a discretization step size of $\Delta\tau := T/N$:

$$\begin{aligned} \min_{X,U} \ & \sum_{i=1}^{N} L(u_i, x_i, p_i) \\ \text{s.t.} \quad & x_0 = \bar{x}_0, \\ & x_{i-1} + \mathscr{F}(u_i, x_i, p_i) = 0, \quad i \in \{1, \cdots, N\}, \\ & C(u_i, x_i, p_i) = 0, \quad i \in \{1, \cdots, N\}, \\ & G(u_i, x_i, p_i) \geq 0, \quad i \in \{1, \cdots, N\}, \end{aligned} \tag{3}$$

where $\bar{x}_0$ is the initial state, $X := (x_0, x_1, x_2, \cdots, x_N)$ and $U := (u_1, u_2, \cdots, u_N)$ are the state and input sequences along the horizon, respectively, $L(u, x, p) : \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}$ is the stage cost function, $C(u, x, p) : \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_\mu}$ is the stage equality constraint function, and $G(u, x, p) : \mathbb{R}^{n_u} \times \mathbb{R}^{n_x} \times \mathbb{R}^{n_p} \to \mathbb{R}^{n_z}$ is the stage polytopic constraint function on u and x, that is, G can be expressed as $G(u, x, p) = A(p)u + B(p)x + c(p)$, where A and B are parameter-depend matrices and c is a parameter-depend vector. For the following reasons, we formulate the NMPC problem with a polytopic inequality constraint. First, the polytopic constraint can be easily satisfied with a backtrack line search. Second, problems with polytopic constraints can benefit from the primal-dual interior-point method (e.g., Nocedal & Wright, 2006). Third, polytopic constraints, such as box or softened box constraints, are common in the context of NMPC. NMPC problems with nonlinear inequality constraints can be reformulated into the above form by introducing slack input variables. Since x_N is the terminal state, the commonly used terminal cost and terminal constraint for guaranteeing stability are encoded in NMPC formulation (3).

We adopt the interior-point method to relax NMPC problem (3) by transferring the inequality constraint $G(u, x, p) \geq 0$ into a logarithmic barrier function added to the cost. Notice that the inequality constraint $G(u, x, p) \geq 0$ is a single-side constraint, which may lead to an unbounded solution since the corresponding barrier function tends to be $-\infty$ as its argument goes to ∞. To prevent this, a linear damping term (Wächter & Biegler, 2006b) is added to the barrier function, and we obtain the fol-

lowing relaxed NMPC problem:

$$\begin{aligned} \min_{X,U} & \sum_{i=1}^{N} L(u_i, x_i, p_i) + \rho\Phi(u_i, x_i, p_i) \\ \text{s.t.} \quad & x_0 = \bar{x}_0, \\ & x_{i-1} + \mathscr{F}(u_i, x_i, p_i) = 0, \quad i \in \{1, \cdots, N\}, \\ & C(u_i, x_i, p_i) = 0, \quad i \in \{1, \cdots, N\}, \end{aligned} \tag{4}$$

where $\rho > 0$ is the barrier parameter and

$$\Phi(u, x, p) := \sum_{j=1}^{n_z} \big(-\ln(G_{(j)}(u, x, p)) + \sigma G_{(j)}(u, x, p) \big)$$

with a fixed small damping constant $\sigma > 0$ (e.g., 10^{-4}).

2.3. KKT conditions

For the sake of brevity, we define

$$s := (\lambda, \mu, u, x).$$

Let $\mathcal{H}(s, p)$ be the Hamiltonian defined by

$$\mathcal{H}(s, p) := L(u, x, p) + \lambda^T \mathscr{F}(u, x, p) + \mu^T C(u, x, p),$$

where λ (costate) $\in \mathbb{R}^{n_x}$ and $\mu \in \mathbb{R}^{n_\mu}$ are the Lagrange multipliers corresponding to the state equation and the equality constraint $C(u, x, p) = 0$, respectively. Let $\mathscr{K}(x_{i-1}, s_i, \lambda_{i+1}, p_i)$ be defined by

$$\begin{aligned} & \mathscr{K}(x_{i-1}, s_i, \lambda_{i+1}, p_i) \\ & := \begin{bmatrix} x_{i-1} + \mathscr{F}(u_i, x_i, p_i) \\ C(u_i, x_i, p_i) \\ \nabla_u \mathcal{H}(s_i, p_i)^T + \rho \nabla_u \Phi(u_i, x_i, p_i)^T \\ \lambda_{i+1} + \nabla_x \mathcal{H}(s_i, p_i)^T + \rho \nabla_x \Phi(u_i, x_i, p_i)^T \end{bmatrix}. \end{aligned} \tag{5}$$

The KKT conditions for relaxed NMPC problem (4) are

$$\mathscr{K}(x^*_{i-1}, s^*_i, \lambda^*_{i+1}, p_i) = 0, \ \forall i \in \{1, \cdots, N\},$$

with $x^*_0 = \bar{x}_0$ and $\lambda^*_{N+1} = 0$.

3. Search direction calculation

Search direction calculation is the most computationally expensive part of real-time optimization. In some particular algorithms dedicated for NMPC, such as the C/GMRES method and the RTI scheme, a full-step iteration is performed each sampling

time, requiring only the computation of the search direction. In this section, search direction calculation performed using Newton's method and the parallel method is given in a unified framework.

We introduce the following shorthand at the k-th iteration:

$$\begin{aligned}\mathscr{K}_i^k &:= \mathscr{K}(x_{i-1}^k, s_i^k, \lambda_{i+1}^k, p_i),\\ \nabla_u \mathscr{F}_i^k &:= \nabla_u \mathscr{F}(u_i^k, x_i^k, p_i), \quad \text{etc.}\end{aligned}$$

3.1. Newton's method

For Newton's method, the search direction $(\Delta s_1, \cdots, \Delta s_N)$ is obtained by solving the following regularized primal-dual (Nocedal & Wright, 2006) system:

$$\left[\begin{array}{c|ccc|c} \ddots & I & & & \\ \hline I & 0 & 0 & & \\ & 0 & -\delta_C I & J_i^k & \\ & & (J_i^k)^T & M_i^k & I \\ \hline & & & I & \ddots \end{array}\right] \left[\begin{array}{c} \vdots \\ \Delta s_i \\ \vdots \end{array}\right] = \left[\begin{array}{c} \vdots \\ \mathscr{K}_i^k \\ \vdots \end{array}\right], \tag{6}$$

where

$$J_i^k := \left[\begin{array}{cc} \nabla_u \mathscr{F}_i^k & \nabla_x \mathscr{F}_i^k \\ \nabla_u C_i^k & \nabla_x C_i^k \end{array}\right]$$

and

$$\begin{aligned} M_i^k := & \underbrace{\left[\begin{array}{cc} \nabla_{uu}^2 \mathcal{H}_i^k & \nabla_{ux}^2 \mathcal{H}_i^k \\ \nabla_{xu}^2 \mathcal{H}_i^k & \nabla_{xx}^2 \mathcal{H}_i^k \end{array}\right]}_{①} + \delta_H I \\ & + \underbrace{\left[\begin{array}{cc} \nabla_u G_i^k & \nabla_x G_i^k \end{array}\right]^T \operatorname{diag}(z_i^k \oslash G_i^k) \left[\begin{array}{cc} \nabla_u G_i^k & \nabla_x G_i^k \end{array}\right]}_{②}. \end{aligned} \tag{7}$$

Here, $z_i \in \mathbb{R}^{n_z}$ denotes the Lagrange multiplier corresponding to the inequality constraint $G(u_i, x_i, p_i) \geq 0$ and $\delta_C, \delta_H \geq 0$. Apart from the regularization terms $-\delta_C I$ and $\delta_H I$, solving system (6) is equivalent to applying Newton's method to KKT conditions (5) for relaxed problem (4), however, with the Hessian of the barrier function $\rho\Phi$ estimated by ② in (7), which is in contrast with the primal system where the exact Hessian is used. The primal-dual system is preferred against the primal system since the primal system tends to perform poorly when a small barrier parameter ρ is chosen (Nocedal & Wright, 2006). After obtaining the search direction $(\Delta s_1, \cdots, \Delta s_N)$, the search direction of z is obtained from

$$\Delta z_i = (z_i^k \odot G_i^{k+1} - \rho e) \oslash G_i^k, \tag{8}$$

where $G_i^{k+1} := G(u_i^k - \Delta u_i, x_i^k - \Delta x_i, p_i)$ and $e := [1, \cdots, 1]^T$.

It is generally easy to guarantee the full rank property of the Jacobian of $\mathscr{F}$, e.g., when the discretization step size $\Delta\tau$ is small. We add a regularization term $\delta_C I \geq 0$ to avoid singularity (see Nocedal & Wright, 2006) of the primal-dual matrix in (6) caused by the rank deficiency of $[\nabla_u C_i^k \ \nabla_x C_i^k]$. In ParNMPC, δ_C is chosen to be sufficiently small (10^{-9}).

To guarantee that the obtained search direction is a descent direction for certain merit functions, a regularization term $\delta_H I \geq 0$ is added to the Hessian so that the primal-dual matrix has a desired inertia. The descent property is required to guarantee convergence when the current iterate is distant from the optimal solution. However, choosing a proper δ_H on-the-fly requires successive factorizations of the primal-dual matrix, which is computationally heavy for real-time optimization. In the context of NMPC, δ_H can be decided offline with prior knowledge of the NMPC problem. We show in Section 3.3 that the descent property is ensured inherently if the Hessian matrix is approximated properly, thus requiring no regularization procedure, i.e., $\delta_H = 0$.

We show next that solving (6) can be interpreted as solving a series of subproblems. We first define the following single-stage subproblem:

$$\begin{aligned} &\min_{x_i,u_i} L(u_i,x_i,p_i) + \rho\Phi(u_i,x_i,p_i) \\ &\qquad + \underbrace{(x_i - x_i^k)^T\lambda_{i+1} + \frac{1}{2}\|x_i - x_i^k\|^2_{W_{i+1}}}_{①} \\ &\text{s.t.} \quad x_{i-1} + \mathscr{F}(u_i,x_i,p_i) = 0, \\ &\qquad C(u_i,x_i,p_i) = 0, \end{aligned} \tag{9}$$

where its regularized primal-dual matrix at the k-th iteration is given by

$$\left[\begin{array}{cc|c} 0 & 0 & \\ 0 & -\delta_C I & J_i^k \\ \hline & (J_i^k)^T & M_i^k + \left[\begin{array}{cc} 0 & 0 \\ 0 & W_{i+1} \end{array}\right] \end{array}\right]. \tag{10}$$

Given a barrier parameter ρ, we denote

$$(s_i, W_i) \leftarrow \mathcal{P}_i^k(x_{i-1}, \lambda_{i+1}, W_{i+1}) \tag{11}$$

as the operation of performing a full-step primal-dual iteration of (9) at the k-th iteration for given parameters of x_{i-1}, λ_{i+1}, and W_{i+1}. Here, $W_i \in \mathbb{R}^{n_x \times n_x}$ denotes the sensitivity matrix of λ_i with respect to the initial state x_{i-1} of subproblem (9). Specifically, W_i is the negation of the n_x-th leading principal submatrix of the inversion of (10), which is a by-product of iteration (11).

For the sake of brevity, we introduce notations as follows.

$$\begin{aligned} S &:= (s_1, \cdots, s_N) \\ Z &:= (z_1, \cdots, z_N) \\ P &:= (p_1, \cdots, p_N) \\ \Omega &:= (W_1, \cdots, W_N) \end{aligned}$$

Notice that the primal-dual matrix in (6) is a block-tridiagonal matrix. It has been

shown by Deng and Ohtsuka (2018) that the so-called backward correction method (Deng & Ohtsuka, 2018) is equivalent to solving (6) by using the block Gaussian elimination method, which is also equivalent to performing the Riccati recursion for the linear quadratic problem having regularized KKT conditions (6). After the backward recursion of the block Gaussian elimination method, we can obtain a linear system with a block lower-triangular coefficient matrix, where its block diagonal entries are given by (10). The forward recursion of the block Gaussian elimination method solves systems with coefficient matrices (10). Together with the fact that (10) is the regularized primal-dual matrix of (9), it can be interpreted that the block Gaussian elimination method, or equivalently, the backward correction method, is equivalent to solving single-stage subproblems (9) recursively as shown in Algorithm 1.

Algorithm 1 Search direction calculation using Newton's method

Input: $\bar{x}_0$, P, S^k, Z^k, ρ
Output: ΔS, ΔZ
1: *Initialization*: $x_0^k = x_0^{k+1} = \bar{x}_0$, $\lambda_{N+1}^{k+1} = 0$, $W_{N+1}^k = 0$
2: **for** $i = N$ to 1 **do**
3: $\quad (s_i^{k+1}, W_i^k) \leftarrow \mathcal{P}_i^k(x_{i-1}^k, \lambda_{i+1}^{k+1}, W_{i+1}^k)$
4: **end for**
5: **for** $i = 1$ to N **do**
6: $\quad (s_i^{k+1}, -) \leftarrow \mathcal{P}_i^k(x_{i-1}^{k+1}, \lambda_{i+1}^{k+1}, W_{i+1}^k)$
7: $\quad \Delta s_i = s_i^k - s_i^{k+1}$
8: $\quad \Delta z_i \leftarrow (8)$
9: **end for**

3.2. Parallel method

The search direction calculation done using Newton's method in Algorithm 1 involves a recursion of computing the sensitivity matrix W_i from $i = N$ to 1. The parallel method proposed by Deng and Ohtsuka (2018) iterates on the basis of the information stored at the previous iteration so that the recursion is broken, and the sensitivity matrices are then updated, which can be referred to as "first iterate, then update," while in Newton's method, it is "first update, then iterate." In the parallel method, the regularized primal-dual matrices are independent of each other and can be factorized in parallel. The parallel method is summarized in Algorithm 2, in which the parallelizable part is explicitly given. As shown in Algorithm 2, not only are the function evaluations performed in parallel, matrix factorizations, which are the most computationally expensive part of (11), are also done in parallel. The parallelization is not fully shown in this paper, and details can be found in the work by Deng and Ohtsuka (2018, 2019), in which the non-parallelizable part consists only of $2N$ matrix-vector multiplications with a complexity of $\mathcal{O}(Nn_x^2)$.

Remark 1. Newton's method and the parallel method have DOPs of one and N, respectively. It should be noted that, for each iteration, the parallel method has exactly the same computational cost as Newton's method, and parallelization is achieved with no extra computational cost. Compared with the quadratic rate of convergence for Newton's method, a superlinear rate of convergence is shown by Deng and Ohtsuka (2019) for the parallel method. It is therefore expected to have a faster rate of convergence when fewer sensitivity matrices W_i are approximated by shifting between or

Algorithm 2 Search direction calculation using parallel method

Input: $\bar{x}_0$, P, S^k, Z^k, Ω^{k-1}, ρ
Output: ΔS, ΔZ, Ω^k

Initialization: $x_0^k = x_0^{k+1} = \bar{x}_0$, $\lambda_{N+1}^{k+1} = 0$, $W_{N+1}^{k-1} = 0$
for $i = 1$ to N **do in parallel**
    Evaluate $\mathcal{K}_i^k$ (KKT function)
    Evaluate J_i^k (Jacobian)
    Evaluate M_i^k (Hessian)
    Factorize (10) (KKT matrix)
end for
for $i = N$ to 1 **do**
    $(s_i^{k+1}, W_i^k) \leftarrow \mathcal{P}_i^k(x_{i-1}^k, \lambda_{i+1}^{k+1}, W_{i+1}^{k-1})$
end for
for $i = 1$ to N **do**
    $(s_i^{k+1}, -) \leftarrow \mathcal{P}_i^k(x_{i-1}^{k+1}, \lambda_{i+1}^{k+1}, W_{i+1}^{k-1})$
    $\Delta s_i = s_i^k - s_i^{k+1}$
    $\Delta z_i \leftarrow (8)$
end for

combining these two methods. In practice, depending on the number of cores, the DOP is made configurable to speed convergence up without sacrificing the computational performance. When the DOP is set to one, ParNMPC decays to Newton's method in Algorithm 1 and behaves exactly the same as the structure-exploiting primal-dual interior-point method. When the DOP is set to N, a fully parallelized method that can use N cores is obtained as shown in Algorithm 2.

The convergence of the parallel method is expected to behave like Newton's method when the system is fast-sampled, less parallelized (i.e., DOP is small), and the dynamics and parameters are slowly varying. That is, the sensitivity matrices W_i can be updated in real time. In practice, the parallel method was observed to also converge fast even when it was fully parallelized (DOP = N) and for highly nonlinear systems, such as the robot manipulator in Section 6 and the double inverted pendulum on a cart, which can be found on ParNMPC's homepage.

Remark 2. Compared with the parallelization given by algebraic operations (Deng & Ohtsuka, 2018, 2019), the parallel method is generalized in this paper from algebraic operations to NLPs (9) so that existing nonlinear optimization techniques, such as regularization, condensing, Hessian approximation, and line search, can be applied. These techniques broaden the range of application of ParNMPC and make the algorithm numerically more efficient and robust. For example, the Hessian matrices in robot applications have to be approximated to avoid time-consuming evaluations of their exact values, and a large speed up was observed in Section 6 due to condensing.

In Newton's method, the nonsingular and descent regularization procedures are performed on the overall primal-dual matrix in (6), while, in the parallel method, it is generally difficult to have an explicit form of the overall primal-dual matrix. An interpretation of ① in (9) is that ① approximates the optimal cost-to-go in a quadratic form (the constant term is ignored). Consequently, each subproblem (9) approximates an NMPC problem consisting of stages i to N with an initial state of x_{i-1}. When

$i = 1$, (9) approximates the original relaxed problem (4) locally. We require that, after regularizing each subproblem, its primal-dual matrix (10) is nonsingular and has a desired inertia, that is, the lower-right submatrix is positive-definite on the null space of J_i^k.

Instead of inverting (10) directly (Deng & Ohtsuka, 2018, 2019) by using LU factorization, a more efficient way of solving each subproblem is to condense the primal-dual system by eliminating Δx_i and $\Delta \lambda_i$. Compared with $\mathcal{O}((2n_x + n_\mu + n_u)^3)$ for inverting (10) directly, condensing can lead to significant computational improvement and is used by default in ParNMPC. It should be noted that the inherent implicit property of the state and costate equations due to the use of the reverse-time discretization method incurs an extra cost on equation solving, i.e., requiring $\nabla_x \mathscr{F}_i^k$ to be inverted, which is usually not needed for the sequential methods (Diehl, Ferreau, & Haverbeke, 2009) where a forward and backward simulation is performed so that the state and costate equations are eliminated. We show that this extra cost for inverting $\nabla_x \mathscr{F}_i^k$ can be avoided by using approximation. Consider, for example, the reverse-time Euler method with $\mathscr{F}(u, x, p) = f(u, x, p)\Delta\tau - x$. We can have the following approximation based on truncated Neumann series when the condition $\|\nabla_x f_i^k \Delta\tau\| \leq 1$ is satisfied:

$$(\nabla_x \mathscr{F}_i^k)^{-1} \approx -I - \nabla_x f_i^k \Delta\tau,$$

where the truncation error is $\mathcal{O}(\Delta\tau^2)$. It is not difficult to show that, for the family of reverse-time Runge-Kutta methods, the approximation is obtained by

$$(\nabla_x \mathscr{F}_i^k)^{-1} \approx -2I - \nabla_x \mathscr{F}_i^k. \tag{12}$$

A good approximation can be obtained if (1) is not stiff and $\Delta\tau$ is small. Approximation (12) stays an option in ParNMPC.

3.3. Hessian approximation

Although ① in (7) is evaluated in parallel as shown in Algorithm 2, it sometimes still dominates the entire computation, e.g., when complicated dynamics and high-order discretization schemes are involved. Moreover, even when the exact Hessian is obtained, the descent property of the search direction is not guaranteed; thus, a regularization procedure is potentially required. Hessian approximation plays an important role in both numerical efficiency and robustness. For example, the Hessian of a high-order discretized problem can be approximated by that of a low-order discretized problem with a cheaper cost. We discuss in this subsection a commonly used Hessian approximation method in NMPC.

When the cost function is in the least-squares form:

$$L(u, x, p) := \frac{1}{2}\|l(u, x, p)\|_2^2,$$

where l is a vector-valued function, the generalized Gauss-Newton method (Bock, 1983) approximates the Hessian matrix ① in (7) by

$$\begin{bmatrix} \nabla_u l_i^k & \nabla_x l_i^k \end{bmatrix}^T \begin{bmatrix} \nabla_u l_i^k & \nabla_x l_i^k \end{bmatrix}. \tag{13}$$

The Gauss-Newton method performs well if the least-square residual is small and func-

tions $\mathscr{F}(u,x,p)$ and $C(u,x,p)$ are less nonlinear such that their second-order derivatives are negligible. The Gauss-Newton Hessian approximation method is favoured in the context of NMPC, where a quadratic cost function is commonly chosen, e.g., in regulation and tracking control problems. Meanwhile, the positive definiteness of the lower-right block of (10) can be guaranteed for any DOP settings, which is shown in the following proposition.

Proposition 3.1. *Assume $\delta_H = 0$, $\delta_C > 0$ is chosen such that* (10) *is nonsingular, and the Jacobian of l is of full column rank. If W_i is initialized to satisfy $W_i \geq 0$ for $i \in \{1, \cdots, N\}$, then the lower-right block of* (10) *generated by the Gauss-Newton method is always positive-definite.*

Proof. See Appendix B. □

Another type of Hessian approximation method is the quasi-Newton method, e.g., the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method (e.g. Nocedal & Wright, 2006), which requires only the first-order derivatives of the Hamiltonian to approximate the Hessian matrix and can estimate the curvature in the Hamiltonian. However, it is not easy to guarantee the positive definiteness of the Hessian update, especially for constrained problems. Although there are methods, such as the damped BFGS method (Nocedal & Wright, 2006), for guaranteeing the positive definiteness, quasi-Newton Hessian approximation methods are less favoured in the context of real-time optimization due to their fluctuating performance as reported by Quirynen (2017).

4. Line search

After obtaining the search directions ΔS and ΔZ, we first determine the maximum step size of an iteration so that the primal and dual feasibility conditions $G(u,x,p) \geq 0$ and $z > 0$ are satisfied. It is recommended by Nocedal and Wright (2006) to have different step sizes for s and z (say, $\alpha_s^{\max}$ and $\alpha_z^{\max}$) to improve performance. $\alpha_s^{\max}$ and $\alpha_z^{\max}$ are obtained by using the fraction-to-the-boundary rule (Nocedal & Wright, 2006):

$$\alpha_z^{\max} = \max\{\alpha \in (0,1] : z_i^k - \alpha \Delta z_i \geq (1-\tau) z_i^k,\ \forall i \in \{1, \cdots, N\}\}$$

$$\alpha_s^{\max} = \max\{\alpha \in (0,1] : G_i^k - \alpha \Delta G_i \geq (1-\tau) G_i^k,\ \forall i \in \{1, \cdots, N\}\}$$

where

$$\Delta G_i = G_i^k - G(u_i^k - \Delta u_i, x_i^k - \Delta x_i, p_i)$$

and the fraction-to-the-boundary parameter τ is chosen to be $\tau = \min\{\tau^{\min}, \rho\}$ with $\tau^{\min} = 0.005$ (typical value). The update of z is performed by

$$Z^{k+1} = Z^k - \alpha_z^{\max} \Delta Z.$$

Although ΔS is a descent direction, the convergence of a full feasible step iteration cannot be guaranteed, especially when the current iterate is far from the optimal

solution, e.g., a far reference. To ensure convergence, we perform the iterate

$$S^{k+1} = S^k - \alpha_s \Delta S,\ \alpha_s \in (0, \alpha_s^{\max}],$$

only when a specified merit function $Q(U, X, P)$ has been decreased by trying a sequence of values of α_s with a constant decay rate or α_s has reached its specified minimum value, e.g., 10^{-3}. The step size α_s is accepted if

$$\begin{aligned} &Q(U^k - \alpha_s \Delta U, X^k - \alpha_s \Delta X, P) \\ &\leq Q(U^k, X^k, P) + \nu \alpha_s D_Q(U^k, X^k, P; \Delta U, \Delta X), \end{aligned}$$

where $\nu \in (0, 1)$ and $D_Q(U^k, X^k, P; \Delta U, \Delta X)$ denotes the directional derivative of Q in the direction $(\Delta U, \Delta X)$. At the k-th iteration, we choose the merit function to be

$$\begin{aligned} Q(U, X, P) = \sum_{i=1}^{N} \{ & L(u_i, x_i, p_i) + \rho \Phi(u_i, x_i, p_i) \\ & + (\lambda^{\max})^T |x_{i-1} + \mathscr{F}(u_i, x_i, p_i)| \\ & + (\mu^{\max})^T |C(u_i, x_i, p_i)| \}, \end{aligned} \tag{14}$$

where $x_0 = \bar{x}_0$, $\lambda^{\max} \in \mathbb{R}^{n_x}$, and $\mu^{\max} \in \mathbb{R}^{n_z}$. We here choose the following heuristic penalty coefficients for better practical performance:

$$\lambda_{(j)}^{\max} = \max_{i \in \{1, \cdots, N\}} |\lambda_{i(j)}^k - \alpha_s^{\max} \Delta \lambda_{i(j)}|$$

and

$$\mu_{(j)}^{\max} = \max_{i \in \{1, \cdots, N\}} |\mu_{i(j)}^k - \alpha_s^{\max} \Delta \mu_{i(j)}|,$$

where $\lambda_{(j)}^{\max}$ and $\mu_{(j)}^{\max}$ are the j-th component of $\lambda^{\max}$ and $\mu^{\max}$, respectively.

An alternative to ensure convergence (Wächter & Biegler, 2006a) is the filter line search method (Fletcher & Leyffer, 2002), in which a trial step is accepted if it decreases the cost function or improves the constraint violation instead of a combination of those two in (14). This method is favoured against the merit-function-based line search method because it has a smaller computational cost per step and does not need to maintain a merit function, which depends on the choices of the penalty terms $\lambda^{\max}$ and $\mu^{\max}$. Both methods can be easily parallelized and are available in ParNMPC.

It should be noted that line search introduces an extra computational cost, which is not favoured in the context of real-time optimization. Moreover, special care should be taken with expensive computational costs, such as a feasibility restoration phase (Nocedal & Wright, 2006) and second-order corrections (Nocedal & Wright, 2006), when the step size becomes too small or the trial step has been rejected. Under what circumstances should we enable line search? Of course, line search is recommended offline in the case where the very first FHOCP is deterministic and can be solved offline to provide an accurate initial guess. We discuss the online case in Remark 3.

Remark 3. For NMPC, where successive optimization problems are solved, line search can be avoided, i.e., by setting $\alpha_s = \alpha_s^{\max}$, if the initial state $\bar{x}_0$ and given

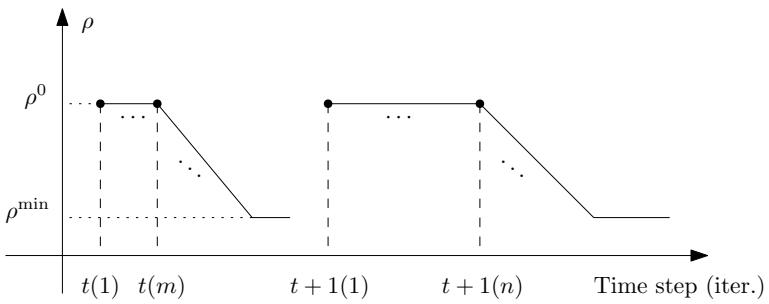

Figure 1. Barrier parameter ρ under two-phase strategy

parameter p, e.g., reference signal, vary smoothly with respect to time. This strategy has been applied in many of the real-time optimization methods for NMPC. For example, under certain conditions, the C/GMRES method can trace a KKT solution without line search. For the case of a distant solution caused by, e.g., far reference, line search should be enabled so that the convergence to a local optimum can be guaranteed (Yamashita, 1998) under certain assumptions.

5. Warm start and barrier strategy

In many optimization methods for MPC, such as the active-set method, warm start can significantly reduce the number of iterations (NOI) since the optimal solution stays close to that of the last sampling time. However, the warm start of the interior-point method is less straightforward and remains challenging. In the interior-point method, relaxed problem (4) is solved successively with a decreasing barrier parameter $\rho > 0$, and an accurate solution is obtained when the barrier parameter ρ is decreased to be sufficiently small. The difficulty of warm start for the interior-point method is that the barrier parameter has to return back to its initial value at the next sampling time, which makes the optimization problem no longer close to its previous one.

One solution is to choose a fixed value (Wang & Boyd, 2010) of ρ, i.e., to obtain an approximate solution. This method is pretty acceptable in the context of MPC, where we care more about the closed-loop performance or cost. However, when a small value of ρ is chosen, it always leads to poor performance since the optimization problem is highly nonlinear at the boundary of the constraints. The iterates can hardly escape from the boundary due to the large entries of ② in (7).

To resolve the problem of warm start when pursuing a highly accurate solution, we adopt the two-phase strategy proposed by Zanelli, Quirynen, Jerez, and Diehl (2017). In the first phase, the relaxed problem is solved with a fixed barrier parameter ρ^0. Then, the barrier parameter is decreased with a constant rate at every iteration in the second phase. The strategy is illustrated in Fig. 1 showing the variation of ρ. At sampling time t, the relaxed problem with an initial barrier parameter ρ^0 is solved and the solution is stored to warm start the first iteration of the next sampling time $t+1$, that is, the solution at $t(m)$ is used to warm start $t+1(1)$.

The overall parallel optimization algorithm, together with the two-phase strategy, is summarized in Algorithm 3. The termination criterion is met when the optimality error for the relaxed problem is smaller than a predefined tolerance or the maximum

NOI has been reached. We denote e_i^x, e_i^μ, e_i^u, e_i^λ as the ℓ_∞ norms of the four expressions in (5), respectively. The optimality error e_o is defined as

$$e_o := \max_{i\in\{1,\cdots,N\}} \left\{ e_i^x,\ e_i^\mu,\ \frac{e_i^u}{\sigma_u},\ \frac{e_i^\lambda}{\sigma_\lambda} \right\},$$

where $\sigma_u, \sigma_\lambda \geq 1$ are the scaling parameters. We adopt Wächter and Biegler's scaling strategy (Wächter & Biegler, 2006b) to choose σ_u and σ_λ as

$$\sigma_u = \max \left\{ \frac{\sum_{i=1}^N (\|\lambda_i\|_1 + \|\mu_i\|_1)}{N(n_x + n_\mu)},\ \sigma^{\max} \right\} / \sigma^{\max}$$

and

$$\sigma_\lambda = \max \left\{ \frac{\sum_{i=1}^N \|\lambda_i\|_1}{N n_x},\ \sigma^{\max} \right\} / \sigma^{\max}.$$

Here, $\sigma^{\max} \geq 1$ is a fixed number, e.g., $\sigma^{\max} = 100$.

Algorithm 3 Parallel optimization with two-phase strategy

Input: $\bar{x}_0$, P, S^{init}, Z^{init}, Ω^{init}
Output: S^{init}, Z^{init}, Ω^{init}, S^*, Z^*

1: *Initialization:* $k = 0$; initial guesses: $S^0 = S^{\text{init}}$, $Z^0 = Z^{\text{init}}$, $\Omega^{-1} = \Omega^{\text{init}}$; barrier parameters: $\rho^0 > 0$, $\eta \in (0, 1]$, $\rho^{\min} \in (0, \rho^0]$, $\rho = \rho^0$
2: **repeat**
3: $(S^{k+1}, Z^{k+1}, \Omega^k) \leftarrow \mathit{Iter}(\bar{x}_0, P, S^k, Z^k, \Omega^{k-1}, \rho)$
4: $k \leftarrow k + 1$
5: **until** termination criterion is met
6: $(S^{\text{init}}, Z^{\text{init}}, \Omega^{\text{init}}) \leftarrow (S^k, Z^k, \Omega^{k-1})$
7: **repeat**
8: $\rho = \max\{\rho^{\min}, \eta\rho\}$
9: $(S^{k+1}, Z^{k+1}, \Omega^k) \leftarrow \mathit{Iter}(\bar{x}_0, P, S^k, Z^k, \Omega^{k-1}, \rho)$
10: $k \leftarrow k + 1$
11: **until** termination criterion is met
12: $(S^*, Z^*) \leftarrow (S^k, Z^k)$

Procedure *Iter*
Input: $\bar{x}_0$, P, S^k, Z^k, Ω^{k-1}, ρ
Output: S^{k+1}, Z^{k+1}, Ω^k
Calculate search directions (Algorithm 2)
Perform line search (Section 4)
End

Regarding the initialization of the very first optimization problem, for those which can be solved offline, an accurate initial guess can be provided with the offline solution. For those which cannot be solved offline, e.g., with an unknown initial state beforehand, the basic requirements for S^{init}, Z^{init}, and Ω^{init} are that the inequality $G(u, x, p) > \epsilon$ ($\epsilon > 0$ is a small number) should be satisfied, its corresponding Lagrange multiplier z should be positive, and the sensitivity matrices should be positive-semidefinite. Conventional initialization techniques used in other methods can be applied to S^{init} and

Z^{init}. For example, a state trajectory can be obtained by interpolating from the initial state to the set-point. The initialization of the sensitivity matrices Ω^{init} is less straightforward. We show a possible initialization for the input-constrained problem with a quadratic cost function, that is, the stage cost function of relaxed problem (4) is in the form of

$$L(u_i, x_i, p_i) + \rho\Phi(u_i, p_i) = \frac{1}{2}\|x_i - x_{ref}\|^2_{Q_i} + \phi(u_i, p_i, \rho),$$

where x_{ref} is the given state reference, $Q_i > 0$ is the weighting matrix, and $\phi(u_i, p_i, \rho)$ is a function encoding the input-related cost. The following equality can be shown:

$$\lim_{\Delta\tau \to 0} W_i = \sum_{j=i}^{N} Q_i,\ i \in \{1, \cdots, N\},$$

and the right-hand side can be used as W_i^{init}.

6. Performance evaluation

In this section, the performance of ParNMPC is evaluated by using three examples. In the first, an example of quadrotor control, we demonstrate the problem formulation and parallel code generation procedures of ParNMPC. The computation time (CT) of ParNMPC is compared in detail against several state-of-the-art NMPC toolkits, and the performance of the two-phase and inversion approximation strategies is assessed as well. We show in the second example, in which a real-world laboratory helicopter is controlled, that ParNMPC converges fast when tracking a dramatically changed reference signal. The third experiment is a closed-loop simulation of a robot manipulator that demonstrates the Gauss-Newton Hessian approximation method and the parallel performance for systems with complicated dynamics.

All experiments were performed on a hexa-core 2.9-GHz (Turbo Boost and Hyper-Threading were disabled) Intel Core i9-8950HK laptop. The helicopter experiment was run in Simulink on Windows 10, and the others were run on Ubuntu 18.04. All code was automatically generated by using ParNMPC. To reduce the effect of the computing environment, the CT at each time step was measured by taking the minimum one of ten runs of the closed-loop simulation.

It should be noted that all three experiments were started from deterministic states, which made it possible to solve the very first optimization problem offline to provide an accurate initial guess, and line search in ParNMPC was only enabled then.

6.1. Quadrotor

6.1.1. Problem formulation

Consider a quadrotor with four inputs and nine states. The state vector of the quadrotor is $x = [X, \dot{X}, Y, \dot{Y}, Z, \dot{Z}, \gamma, \beta, \alpha]^T \in \mathbb{R}^9$, where (X, Y, Z) and (γ, β, α) are the position and angles of the quadrotor, respectively. The input vector is $u = [a, \omega_X, \omega_Y, \omega_Z]^T$, where a represents the thrust and $(\omega_X, \omega_Y, \omega_Z)$ the rotational rates. The dynamics of

the quadrotor are given by the following equations (Hehn & D'Andrea, 2011):

$$\begin{aligned}
\ddot{X} &= a(\cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha)\\
\ddot{Y} &= a(\cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha)\\
\ddot{Z} &= a\cos\gamma\cos\beta - g\\
\dot{\gamma} &= (\omega_X\cos\gamma + \omega_Y\sin\gamma)/\cos\beta\\
\dot{\beta} &= -\omega_X\sin\gamma + \omega_Y\cos\gamma\\
\dot{\alpha} &= \omega_X\cos\gamma\tan\beta + \omega_Y\sin\gamma\tan\beta + \omega_Z,
\end{aligned}$$

where $g = 9.81$ m/s^2. The goal was to control the quadrotor to position $(1, 1, 1)$ under the constraints of the inputs: $[0, -1, -1, -1]^T \le u \le [11, 1, 1, 1]^T$. We chose the cost function to be quadratic as

$$L(u, x, p) = \frac{1}{2}(\|x - x_{ref}\|_Q^2 + \|u - u_{ref}\|_R^2),$$

where x_{ref} encodes the position reference, $u_{ref} = [g, 0, 0, 0]^T$, and the weighting matrices were $Q = \mathrm{diag}(10, 1, 10, 1, 10, 1, 1, 1, 1)$ and $R = 0.01 \times I$. The prediction horizon was $T = 0.5$ s, which was discretized into $N = 24$ grids by using the reverse-time Heun's method.

The above NMPC problem is formulated in ParNMPC as shown in Listing 1.

```
% Create an OCP(n_u,n_x,n_p,N)
OCP = OptimalControlProblem(4,9,0,24);
X = OCP.x(1); dX = OCP.x(2); ...
a = OCP.u(1); omegaX = OCP.u(2); ...
OCP.setT(0.5);
OCP.setDiscretizationMethod('RK2');
f = [dX;a*(cos(gamma)*sin(beta)*cos(alpha)+sin(gamma)*sin(alpha));...];
Q = diag([10, 1, 10, 1, 10, 1, 1, 1, 1]); R = 0.01*eye(4);
xRef = [1;0;1;0;1;0;0;0;0]; uRef = [g;0;0;0];
L = 0.5*(OCP.x-xRef).'*Q*(OCP.x-xRef)+0.5*(OCP.u-uRef).'*R*(OCP.u-uRef);
G =[[11;1;1;1]-OCP.u;[0;1;1;1]+OCP.u];
OCP.setf(f); OCP.setL(L); OCP.setG(G);
OCP.codeGen();
```

Listing 1 Problem formulation in ParNMPC (some repeated code was omitted)

After formulating the NMPC problem, the NMPC solver is configured as shown in Listing 2. The exact Hessian is calculated, and the regularization parameters are made default in this example, i.e., $\delta_C = 10^{-9}$ and $\delta_H = 0$. The exact value of $(\nabla_x \mathscr{F})^{-1}$ is used by default.

```
% Configurate the NMPC solver
sol = NMPCSolver(OCP);
sol.setHessianApproximation('Newton');
% sol.setNonsingularRegularization(1e-9); % \delta_C
% sol.setDescentRegularization(0); % \delta_H
% sol.setInvFxMethod('exact');
sol.codeGen();
```

Listing 2 NMPC solver configuration in ParNMPC

6.1.2. Code generation

The parallel code for DOP = 6 of the NMPC controller can be automatically generated as shown in Listing 3. We fix the barrier parameter to be $\rho^0 = \rho^{\min} = 10^{-3}$. The two-phase strategy in Section 5 can be adopted by adjusting parameters ρ^0, $\rho^{\min}$, and η. Regarding the termination criterion, the optimality tolerance is 10^{-3}, and the maximum NOI is 10. The generated code is self-contained, easy-to-use, and can be easily deployed, e.g., using Visual Studio with *OpenMP Support* enabled or GCC with an extra *-fopenmp* flag. Given an initial state, the corresponding optimal control input can be obtained by simply calling function *NMPC_Solve* from the generated code.

```
options = createOptions();
options.DoP = 6;
options.isLineSearch = false;
options.rhoInit = 1e-3; % \rho^0
options.rhoEnd = 1e-3; % \rho^min
% options.rhoDecayRate = 0.1; % \eta
options.tolEnd = options.rhoEnd; % tolerance
options.maxIterTotal = 10;
% Generate parallel C code
NMPC_Solve_CodeGen('lib','C',options);
```

Listing 3 Parallel code generation in ParNMPC

6.1.3. Evaluation of computation time

The CT of ParNMPC was evaluated by performing a three-second closed-loop simulation starting from $\bar{x}_0 = 0$ with a sampling period of 10 ms. We compared the following toolkits (the last three toolkits are based on the first-order methods):

- ParNMPC (version 1903-1): the toolkit introduced in this paper.
- ParNMPC-primitive: the primitive version of ParNMPC, of which the algorithm is introduced in (Deng & Ohtsuka, 2019). The mechanism for handling inequalities was modified in this paper to be the primal interior-point method.
- ACADO Code Generation Tool with qpOASES and qpDUNES: the SQP method based on a dense QP solver, qpOASES, and a sparse QP solver, qpDUNES.
- GRAMPC: a gradient-based augmented Lagrangian method.
- VIATOC: a gradient projection method for NMPC problems with input and state box constraints.
- FalcOpt: a projected gradient descent method.

Apart from the default parameters, some key tuning parameters for the different toolkits are shown in Table 1.

Table 1. Tuning parameters for different toolkits in quadrotor example (e.g., RT-Heun stands for reverse-time Heun's method)

Toolkit	Discretization	Tuning parameters
ParNMPC	RT-Heun	As shown in Listings 1, 2, and 3
ParNMPC-primitive	RT-Euler	Barrier parameter: 0.001
ACADO (qpOASES)	Heun	GAUSS_NEWTON, MULTIPLE_SHOOTING, FULL_CONDENSING_N2, HOTSTART_QP, tolerence: 0.001
ACADO (qpDUNES)	Heun	GAUSS_NEWTON, MULTIPLE_SHOOTING, tolerence: 0.001
ACADO-RTI (qpDUNES)	Heun	Same as above but performing RTI scheme
GRAMPC	Heun	MaxMultIter: 1, MaxGradIter: 10
VIATOC	Heun	Number of iterations: 10
FalcOpt	Euler	Tolerance: 0.01

The closed-loop trajectories of the inputs and position when using ParNMPC are shown in Figs. 2 and 3, respectively. The CTs during the closed-loop simulation for the different toolkits are shown in Fig. 4. In addition, the optimality for each toolkit

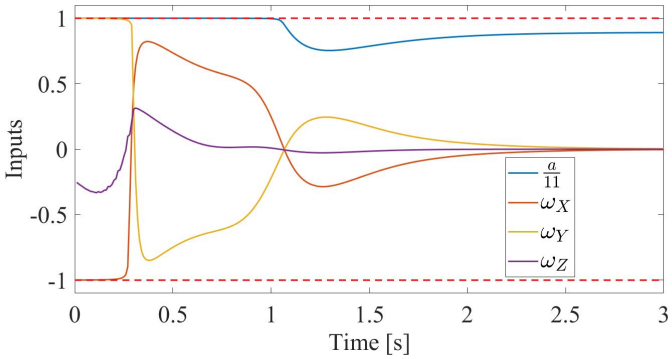

Figure 2. Time histories of inputs for quadrotor example

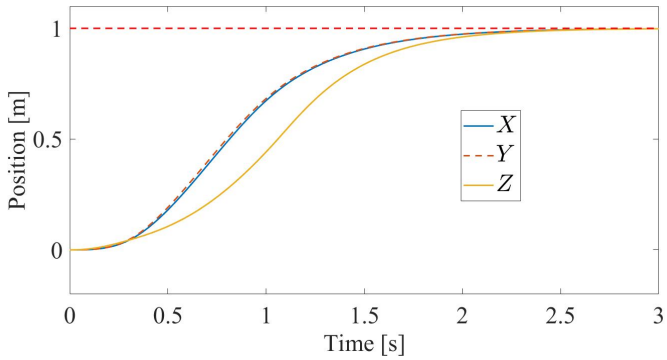

Figure 3. Time histories of position for quadrotor example

is defined as the normalized distance to its corresponding optimal trajectory, i.e.,

$$\frac{\sum_{t/0.01=0}^{300}\{L(\tilde{u}(t),\tilde{x}(t),p(t))-L(u^*(t),x^*(t),p(t))\}}{\sum_{t/0.01=0}^{300}L(u^*(t),x^*(t),p(t))}\times 100\%, \tag{15}$$

where $\tilde{u}(t)$ is the control input obtained by each toolkit, $u^*(t)$ is the optimal control input obtained by solving the corresponding OCP exactly, and $\tilde{x}(t)$ and $x^*(t)$ are the corresponding closed-loop responses, respectively. The closed-loop optimalities of the different toolkits are shown in Table 2.

Table 2. Closed-loop optimalities [%] of different toolkits for quadrotor example

ParNMPC	0.1177	GRAMPC	1.5233
ParNMPC-primitive	0.0819	VIATOC	1.2743
ACADO (qpDUNES)	0.0002	FalcOpt	0.9719
ACADO-RTI (qpDUNES)	0.0108	-	-

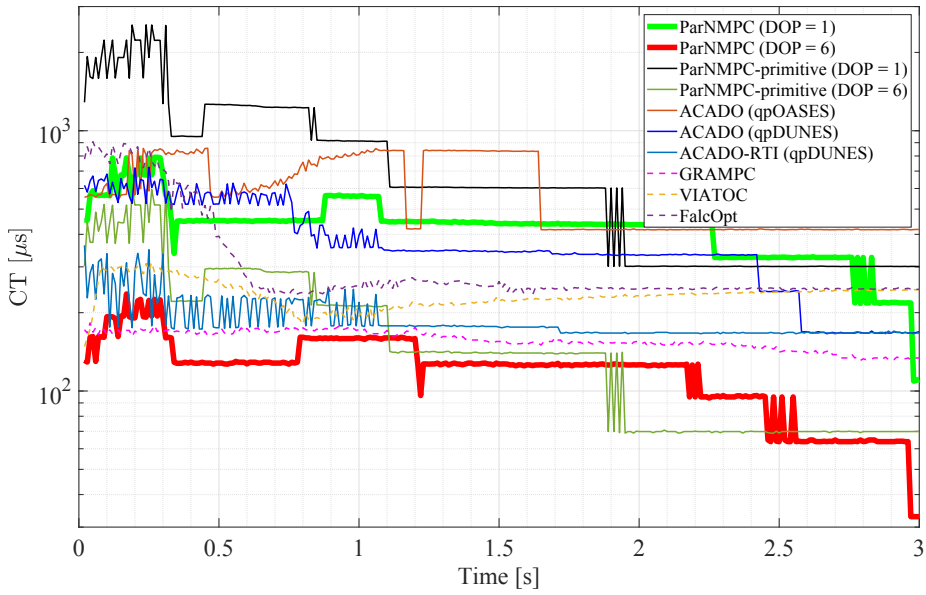

Figure 4. CTs per time step of closed-loop simulation

Note that even though the first-order methods (GRAMPC, VIATOC, and FalcOpt) have relatively low optimalities, they can still drive the quadrotor to the reference position. In some circumstances, a sub-optimal solution obtained by performing only several iterations is acceptable. That is, we are also interested in the CT per iteration, which is compared for different N for these toolkits in Table 3.

We conclude from the comparison results as shown in Fig. 4 and Tables 2 and 3 that

- For ParNMPC, after parallelization, a speed-up of more than $4\times$ was achieved for $N = 96$ and even $2.5\times$ for a small number of discretion grid points of $N = 6$.
- Compared with ParNMPC-primitive, ParNMPC was a factor of 2-3 faster in terms of CT per iteration as shown in Table 3 both in serial and parallel, which was caused by condensing.
- Compared with ACADO, where a QP has to be solved at each iteration, the CT per iteration of ParNMPC in Table 3 was shorter than that of ACADO (qpDUNES) even in serial and did not vary too much during the closed-loop simulation due to the usage of the interior-point method. That is, the CT per time step can be roughly estimated from the NOI. However, ACADO was more effective than ParNMPC in terms of the closed-loop optimality as shown in Table 2.
- The first-order methods had short CTs per iteration, and ParNMPC with parallelization was only several times slower than first-order methods as shown in Table 3. However, the first-order methods converged slowly as indicated by their closed-loop optimalities in Table 2.

In summary, ParNMPC has a good trade-off between optimality and CT, resulting in an overall advantageous CT for the closed-loop control in Fig. 4, while maintaining a good closed-loop optimality as shown in Table 2.

Table 3. CTs [μs] per iteration for different N for quadrotor example

N		6	12	24	48	96
ParNMPC (DOP = 1)	min	27	54	108	218	437
	median	28	55	110	222	444
	max	29	57	114	241	456
ParNMPC (DOP = 6)	min	11	18	31	55	101
	median	11	19	32	56	103
	max	12	20	33	63	111
ParNMPC-primitive (DOP = 1)	min	78	152	301	627	1251
	median	79	154	302	631	1261
	max	82	166	322	640	1287
ParNMPC-primitive (DOP = 6)	min	20	37	69	135	263
	median	21	38	70	137	267
	max	24	42	76	141	284
ACADO (qpOASES)	min	30	81	269	1015	4571
	median	32	99	417	2463	20377
	max	34	102	427	2496	20573
ACADO (qpDUNES)	min	41	83	120	248	520
	median	43	86	171	355	751
	max	60	112	245	492	1226
ACADO-RTI (qpDUNES)	min	42	83	166	345	719
	median	44	87	176	368	774
	max	79	161	363	819	1855
GRAMPC	min	3	6	13	26	53
	median	4	8	16	31	62
	max	5	9	18	35	71
VIATOC	min	6	8	15	29	57
	median	7	12	23	46	94
	max	8	13	31	100	469
FalcOpt	min	3	6	12	24	49
	median	3	6	12	25	51
	max	4	8	17	35	73

6.1.4. Evaluation of the two-phase and inversion approximation strategies

In the above quadrotor example, the performance of the two-phase and inversion approximation (12) strategies is not evaluated. In this part, we extended the simulation time to six seconds and set the position reference x_{ref} to zero for the last three seconds. We ran the experiment with different DOPs, barrier strategies (with different ρ^0 and $\rho^{\min}$), and $(\nabla_x \mathscr{F})^{-1}$ calculation methods as shown in Table 4. It should be noted that all of these patterns converged to the same optimality tolerance with the same terminal barrier parameter, and therefore, the same level of optimality.

Due to the choice of the small terminal barrier parameter $\rho^{\min}$, input signals approached the constraints very closely, and the closed-loop optimality (15) was decreased to 0.002. The NOIs and CTs for different patterns are shown in Table 5. By comparing patterns (b)(d)(e)(f) with (a)(c), it can be seen that the two-phase strategy illustrated in Fig. 1 significantly reduced the maximum NOIs for both DOPs. The approximation of $(\nabla_x \mathscr{F})^{-1}$ using (12) for patterns (e)(f) reduced the CT by 14% and 6% per iteration compared with patterns (b)(d), respectively, however, with a slight increase in the NOIs.

Table 4. Configurations

Pattern	DOP	Two-phase strategy ($\eta = 0.1$) ρ^0	Two-phase strategy ($\eta = 0.1$) $\rho^{\min}$	$(\nabla_x \mathscr{F})^{-1}$
(a)	1	10^{-5}	10^{-5}	Exact
(b)	1	1	10^{-5}	Exact
(c)	6	10^{-5}	10^{-5}	Exact
(d)	6	1	10^{-5}	Exact
(e)	1	1	10^{-5}	Approx.
(f)	6	1	10^{-5}	Approx.

Table 5. NOIs and CTs under different configurations

Pattern	Max. NOI	Avg. NOI	Median CT/Iter. [μs]
(a)	37	6.0	111
(b)	19	13.0	111
(c)	38	6.5	32
(d)	21	13.4	32
(e)	20	13.2	95
(f)	21	13.4	30

6.2. Helicopter

In this experiment, we controlled Quanser's 3 degree-of-freedom (DOF) helicopter as shown in Fig. 5. The data acquisition and communication were done by using Quanser Q8-USB, which is able to provide a maximum closed-loop control rate of 2 kHz. The hardware blocks in Simulink were provided by Quanser's real-time control software, QUARC.

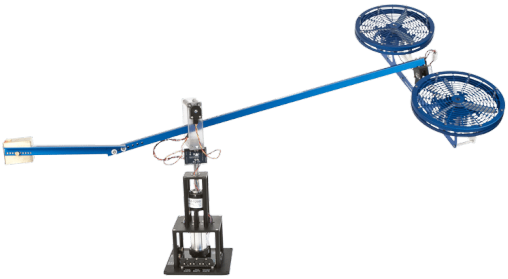

Figure 5. Quanser's 3 DOF helicopter (https://www.quanser.com/products/3-dof-helicopter/)

The helicopter has two inputs $u = [V_f, V_b]^T$: the voltage on the front motor V_f and the voltage on the back motor V_b. It has six states including three angles $q = [q_\epsilon, q_\rho, q_\lambda]^T$ (the elevation angle q_ϵ, pitch angle q_ρ, and yaw angle q_λ) and their time derivatives. We use the benchmark model by Brentari, Bosetti, Queinnec, and Zaccarian (2018) in the following form:

$$\ddot{q} = -\begin{bmatrix} \sin q_\epsilon(a_{\epsilon_1} + a_{\epsilon_2}\cos q_\rho) + C_\epsilon \dot{q}_\epsilon \\ -a_\rho \cos q_\epsilon \sin q_\rho + C_\rho \dot{q}_\epsilon \\ C_\lambda \dot{q}_\epsilon \end{bmatrix} + K_f \begin{bmatrix} b_\epsilon \cos q_\rho & 0 \\ 0 & b_\rho \\ b_\lambda \cos q_\epsilon \sin q_\rho & 0 \end{bmatrix} \begin{bmatrix} V_f + V_b \\ V_f - V_b \end{bmatrix},$$

where the parameters are given as: $a_{\epsilon_1} = 2.356$, $a_{\epsilon_2} = 0.799$, $a_\rho = 0.858$, $C_\epsilon = 0.053$, $C_\rho = 0.048$, $C_\lambda = 0.274$, $b_\epsilon = 0.719$, $b_\rho = 9.336$, $b_\lambda = 0.327$, and $K_f = 0.1188$. The goal was to control the helicopter to track a given reference q_{ref} under the constraints of the input voltages: $V_f, V_b \in [5, 10]$. We choose the cost function to be quadratic as follows.

$$L(u, x, p) = \frac{1}{2}(\|q - q_{ref}\|^2_{Q_q} + \|\dot{q}\|^2_{Q_{\dot{q}}} + \|u - \bar{u}\|^2_R)$$

Here, $\bar{u} = [7.5, 7.5]^T$ denotes the approximate voltage needed to eliminate the effect

of gravity, $p = q_{ref}$, and the weighting matrices are $Q_q = 10 \times I$, $Q_{\dot{q}} = 0.1 \times I$, and $R = 0.1 \times I$. The prediction horizon was $T = 4$ s, which was discretized into $N = 24$ grids by using the reverse-time Runge-Kutta method method. The barrier parameter and the optimality tolerance were fixed to 0.001, and DOP = 1. The exact Hessian and $(\nabla_x \mathscr{F})^{-1}$ were used.

The experiment was performed for 90 s with a sampling period of 5 ms. We first controlled the helicopter to track several step yaw angle references and then a sine signal with a skew rate of $\pi/2$ rad/s. The closed-loop responses of the angles are shown in Figs. 6 and 7, and the corresponding input signals are shown in Fig. 8. Despite the offset when tracking the piecewise constant reference, the NMPC controller could track the given reference well while satisfying the input constraints. As shown in Fig. 9, the proposed method converged to the specified tolerance with only five iterations at most, even though the reference signal was changed dramatically, resulting in sudden changes in the input signals flipping from one side to another. The time histories of the CT are shown in Fig. 10.

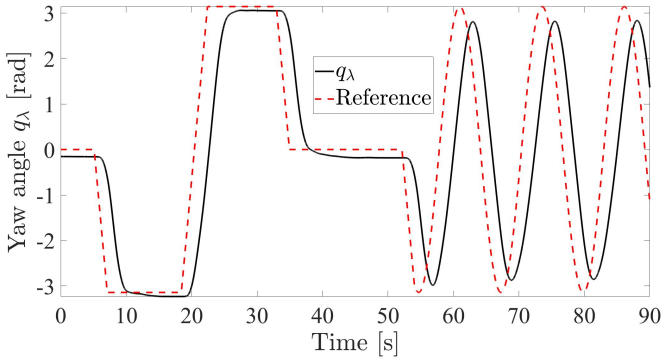

Figure 6. Time histories of yaw angle

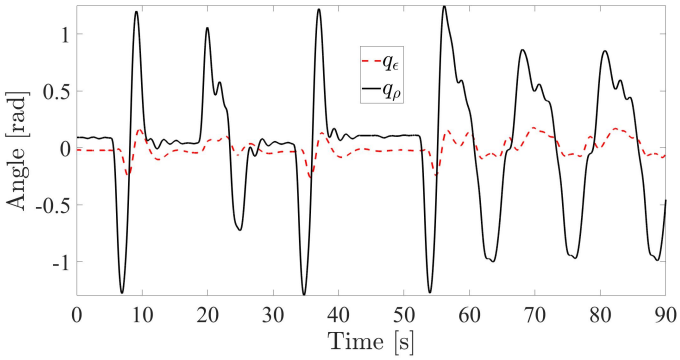

Figure 7. Time histories of elevation and pitch angles

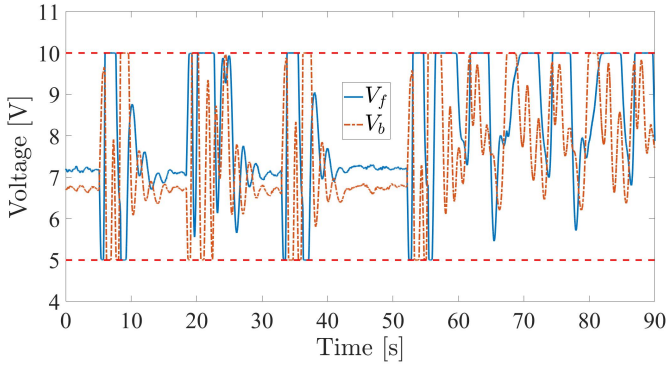

Figure 8. Time histories of input signals

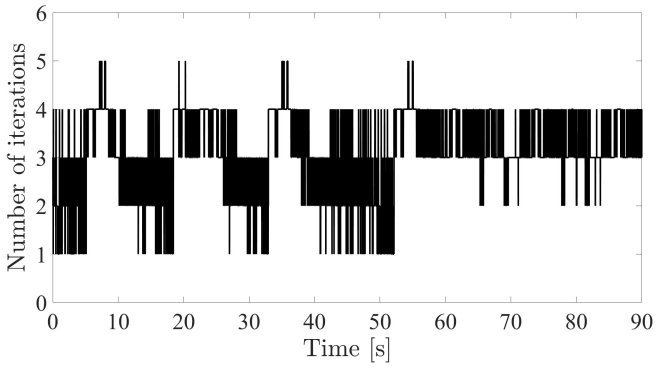

Figure 9. Time histories of NOIs

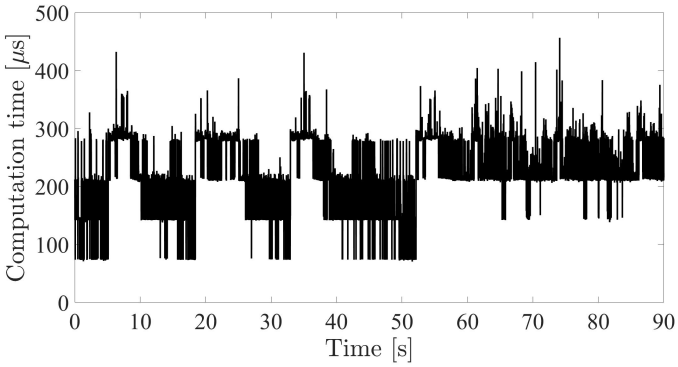

Figure 10. Time histories of CT

6.3. Robot manipulator

The manipulator is a 7-DOF lightweight robot manipulator KUKA LBR iiwa 14, which has seven joint torque inputs $u = \tau \in \mathbb{R}^7$ and 14 states including seven joint angles $q \in \mathbb{R}^7$ and their corresponding angular velocities $\dot{q} \in \mathbb{R}^7$. The goal was to control the manipulator to track given joint angles' references q_{ref} under the constraints of the

torques and angular velocities; each joint had a maximum torque output of 10 Nm, and the angular velocity for each joint was limited to have a maximum value of $\pi/2$ rad/s in order to achieve smooth movement. For the NMPC problem formulation, the angular velocity constraints were softened by introducing a slack variable $v \geq 0$, which denotes the violation of the constraints. The inequality constraint is

$$G(u,x,p) = \begin{bmatrix} \tau + 10e \\ -\tau + 10e \\ v \\ \dot{q} + \frac{\pi}{2}e + ve \\ -\dot{q} + \frac{\pi}{2}e + ve \end{bmatrix} \geq 0,$$

where $e = [1, \cdots, 1]^T$. We choose the cost function to be quadratic as follows.

$$L(u,x,p) = \frac{1}{2}(\|q - q_{ref}\|^2_{Q_q} + \|\dot{q}\|^2_{Q_{\dot{q}}} + \|\tau\|^2_R + 1000v^2)$$

Here, $p = q_{ref}$ and the weighting matrices were $Q_q = I$, $Q_{\dot{q}} = 0.1 \times I$, and $R = 0.001 \times I$. The weighting imposed on v was 1000, which was at least three orders of magnitude larger than the other weightings. The prediction horizon was $T = 1$ s, which was discretized into $N = 18$ grids by using the reverse-time Euler's method method. The Gauss-Newton Hessian approximation method was chosen. The barrier parameter and the optimality tolerance were fixed to 5×10^{-4}, and the exact $(\nabla_x \mathscr{F})^{-1}$ was used. The gravity was set to zero. The system function $f(u,x,p)$ and its derivatives were implemented using Pinocchio (Carpentier et al., 2019), which is a C++ library for efficient rigid multi-body dynamics computations.

For the closed-loop simulation, the system was started from an initial state of $\bar{x}_0 = 0$ and $q_{ref} = 0$. The simulation was performed for 8 s with a sampling period of 1 ms. The joints' reference was set to $q_{ref} = [0, \pi/2, 0, \pi/2, 0, \pi/2, 0]^T$ for the first four seconds and $q_{ref} = [\pi/2, 0, \pi/2, 0, \pi/2, 0, \pi/2]^T$ for the last four seconds. The reference was fed to the controller with a rate limitation of 4π rad/s.

In this experiment, we compared the CTs under different DOP settings (in serial and parallel). We note again that when DOP is set to one, ParNMPC behaves exactly the same as the structure-exploiting primal-dual interior-point method. For the measurement of the CT, we ran the simulation 10 times and chose the minimal one to eliminate the effect of performance fluctuation. Five sampled plots for the simulated manipulator are shown in Fig. 11, and the time histories of the joint torque inputs, joint angles, and the angular velocities are shown in Figs. 12, 13, and 14, respectively. Here, the last three torques are omitted in Fig. 12 because their magnitudes are in the range of $[-1, 1]$. We can see from the simulation results that the NMPC controller could control the robot manipulator to its desired reference position smoothly while satisfying the specified constraints. The NOIs for DOP = 1 and 6 are compared in Fig. 15, which shows that both converged to the specified tolerance with nearly the same rates, even though approximate information was used in the parallel mode. However, the parallel one was much faster than the serial one in terms of the CT shown in Fig. 16. A speed-up of more than $4\times$ could be achieved on the hexa-core processor. We noticed that, for both DOPs, the NMPC controller could still drive the manipulator to its reference position by using even only one iteration each sampling time. Considering that the CT per iteration for the parallel method was less than 170 μs, a higher sampling rate can be achieved with either more cores or fewer iterations.

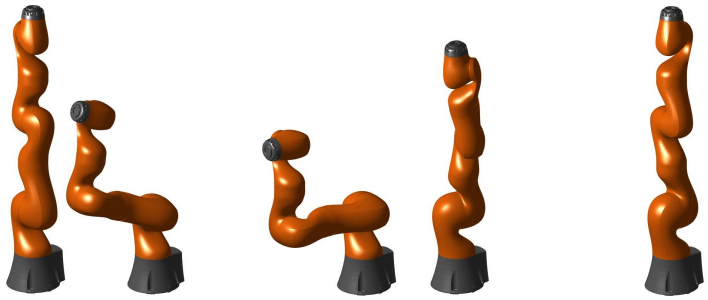

Figure 11. Plots of manipulator at 0, 1, 4, 5, and 8 s

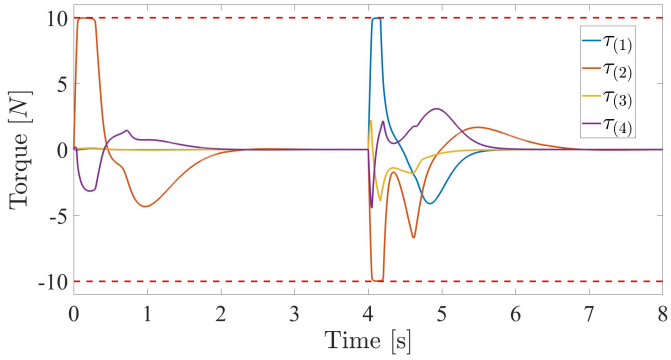

Figure 12. Time histories of first four joint torque inputs

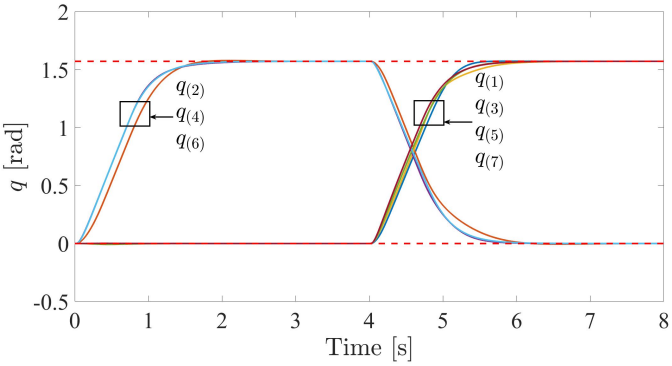

Figure 13. Time histories of joint angles q

6.4. Discussion

Since ParNMPC is exactly the primal-dual interior-point method applied to NMPC when DOP = 1, it inherits the good convergence and robustness of the interior-point method. Moreover, the increase in speed for ParNMPC with parallelization (DOP > 1) can be large on a hexa-core processor, so ParNMPC with parallelization is expected to behave better with more cores. Parallel performance can be achieved even for highly

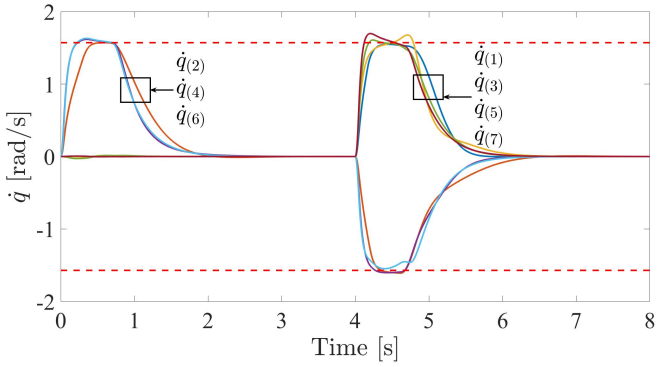

Figure 14. Time histories of joint angular velocities $\dot{q}$

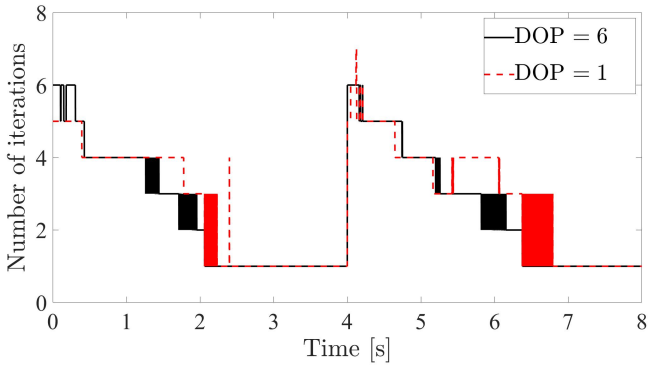

Figure 15. Time histories of NOIs for parallel method (DOP = 6) and Newton's method (DOP = 1)

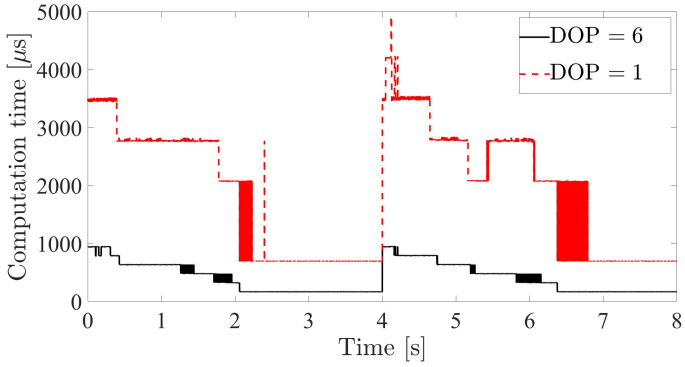

Figure 16. Time histories of CTs for parallel method (DOP = 6) and Newton's method (DOP = 1).

nonlinear systems (e.g., a robot manipulator) and a small N, with almost the same number of iterations. However, parallelization with OpenMP introduces an overhead time, which is usually proportional to DOP and is not negligible when either the overall CT is small or the DOP is large.

7. Conclusion

This paper presented an efficient implementation for the parallel optimization of NMPC. The reverse-time discretization method dedicated for parallelization is introduced and its accuracy was shown. Since the parallel method is in the same framework as the conventional primal-dual interior-point method, nonlinear optimization techniques, such as regularization, Hessian approximation, and line search, are applied and optimized for the parallel optimization of NMPC, making the implementation both fast and numerically robust. Three experiments showed that ParNMPC was effective and efficient both in serial and parallel.

Future directions include integrating algorithmic differentiation to calculate the required derivatives for large-scale systems and replacing OpenMP with lower level parallel computing interfaces to reduce the overhead time.

Disclosure statement

No potential conflict of interest was reported by the authors.

Appendix A. Proposition A.1

Proposition A.1. *Let $y(t) \in \mathbb{R}^m$ and consider the ordinary differential equation*

$$\dot{y}(t) = f(y(t), t).$$

Let h be the discretization step size and the differential equation be discretized by using an explicit discretization method with a local truncation error of $\mathcal{O}(|h|^n)$:

$$Y(t+h) = F(y(t), h, t),$$

that is, $Y(t+h)$ is the approximation to $y(t+h)$ satisfying

$$\|Y(t+h) - y(t+h)\| = \mathcal{O}(|h|^n).$$

Let $\tilde{Y}(t)$ be obtained by using the corresponding reverse-time discretization method, i.e.,

$$y(t-h) = F(\tilde{Y}(t), -h, t).$$

Then, the local truncation error of the reverse-time discretization method is also $\mathcal{O}(|h|^n)$, that is,

$$\|\tilde{Y}(t) - y(t)\| = \mathcal{O}(|h|^n). \tag{A1}$$

Proof. Without loss of generality, we assume $h > 0$. It can also be obtained that

$$\|y(t-h) - Y(t-h)\| = \mathcal{O}(h^n), \tag{A2}$$

where

$$Y(t-h) = F(y(t), -h, t).$$

According to the mean value theorem for vector-valued functions (McLeod, 1965), the following equation holds:

$$\begin{aligned} &F(\tilde{Y}(t), -h, t) - F(y(t), -h, t) \\ &\qquad = \sum_{i=1}^{n} \lambda_i \nabla_y F(\xi_i, -h, t) \left(\tilde{Y}(t) - y(t) \right), \end{aligned} \tag{A3}$$

where $\xi_i \in [y(t), Y(t)]$, $\lambda_i \geq 0$, and $\sum_{i=1}^{n} \lambda_i = 1$. Since $y(t-h) = F(\tilde{Y}(t), -h, t)$ and $Y(t-h) = F(y(t), -h, t)$, by taking norms of both sides, we obtain

$$\begin{aligned} &\|\tilde{Y}(t) - y(t)\| \\ &\leq \left\| \left(\sum_{i=1}^{n} \lambda_i \nabla_y F(\xi_i, -h, t) \right)^{-1} \right\| \|(y(t-h) - Y(t-h))\|. \end{aligned}$$

The result (A1) then follows from (A2) and the fact that $\nabla_y F(\xi_i, -h, t) \to I$ when $h \to 0$. □

Appendix B. Proof of Proposition 3.1

Proof. From the full rank property of the Jacobian of l, we know that (13), i.e., ① in (7), is positive-definite. Since $z_i^k > 0$ and $G_i^k > 0$, ② in (7) always satisfies ② ≥ 0. Consequently, we know that $M_i^k > 0$ always holds. In addition, due to the fact that the initial value of W_{i+1} satisfies $W_{i+1} \geq 0$, the lower-right block of (10) is positive-definite for the initial value of W_{i+1}. We show next the definiteness of the updated W_i for $i \in \{1, \cdots, N\}$.

For each $i \in \{1, \cdots, N\}$, the Schur complement of the lower-right block of matrix (10) is nonsingular since we have shown that the lower-right block is positive-definite and (10) is nonsingular from the choice of δ_C. Specifically, the upper-left block of the inversion of (10) can be expressed as

$$\left(\begin{bmatrix} 0 & 0 \\ 0 & -\delta_C I \end{bmatrix} - J_i^k \left(M_i^k + \begin{bmatrix} 0 & 0 \\ 0 & W_{i+1} \end{bmatrix} \right)^{-1} (J_i^k)^T \right)^{-1}. \tag{B1}$$

For an arbitrary vector $v \in \mathbb{R}^{n_x + n_\mu}$, the inequality

$$v^T J_i^k \left(M_i^k + \begin{bmatrix} 0 & 0 \\ 0 & W_{i+1} \end{bmatrix} \right)^{-1} (J_i^k)^T v \geq 0 \tag{B2}$$

holds. Considering that $\delta_C > 0$, (B2) holds, and (B1) is nonsingular, we know that (B1) < 0. Since the updated W_i is the negation of the n_x-th leading principal submatrix of (B1), $W_i > 0$ always holds for $i \in \{1, \cdots, N\}$. In turn, the result then follows. □

References

Amdahl, G. M. (1967). Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the Sprint Joint Computer Conference* (pp. 483–485). Atlantic City, USA.

Bock, H. G. (1983). Recent advances in parameter identification techniques for ODE. In P. Deuflhard & E. Hairer (Eds.), *Numerical Treatment of Inverse Problems in Differential and Integral Equations* (Vol. 2). Birkhäuser Boston.

Bock, H. G., & Plitt, K. J. (1984). A multiple shooting algorithm for direct solution of optimal control problems. In *Proceedings of the 9th IFAC World Congress* (pp. 1603–1608). Budapest, Hungary.

Brentari, M., Bosetti, P., Queinnec, I., & Zaccarian, L. (2018). Benchmark model of Quanser's 3 DOF helicopter. *Rapport LAAS 18040*, hal-01711135.

Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., & Mansard, N. (2019). The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *Proceedings of the 11th IEEE/SICE International Symposium on System Integration* (pp. 614–619). Paris, France.

Chen, Y., Bruschetta, M., Picotti, E., & Beghi, A. (2019). MATMPC - A MATLAB based toolbox for real-time nonlinear model predictive control. In *Proceedings of the 18th European Control Conference* (pp. 3365–3370). Naples, Italy.

Dagum, L., & Menon, R. (1998). OpenMP: an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, *5*(1), 46–55.

Deng, H., & Ohtsuka, T. (2018). A highly parallelizable Newton-type method for nonlinear model predictive control. In *Proceedings of the 6th IFAC Conference on Nonlinear Model Predictive Control* (pp. 426–432). Madison, USA.

Deng, H., & Ohtsuka, T. (2019). A parallel Newton-type method for nonlinear model predictive control. *Automatica*, *109*, 108560.

Diehl, M., Bock, H. G., & Schlöder, J. P. (2005). A real-time iteration scheme for nonlinear optimization in optimal feedback control. *SIAM Journal on Control and Optimization*, *43*(5), 1714–1736.

Diehl, M., Ferreau, H. J., & Haverbeke, N. (2009). Efficient numerical methods for nonlinear MPC and moving horizon estimation. In L. Magni, D. M. Raimondo, & F. Allgöwer (Eds.), *Nonlinear Model Predictive Control* (Vol. 384). Springer, Berlin, Heidelberg.

Englert, T., Völz, A., Mesmer, F., Rhein, S., & Graichen, K. (2019). A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC). *Optimization and Engineering*, *20*(3), 769–809.

Ferreau, H. J., Kirches, C., Potschka, A., Bock, H. G., & Diehl, M. (2014). qpOASES: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, *6*(4), 327–363.

Fletcher, R., & Leyffer, S. (2002). Nonlinear programming without a penalty function. *Mathematical Programming*, *91*(2), 239–269.

Frasch, J. V., Sager, S., & Diehl, M. (2015). A parallel quadratic programming method for dynamic optimization problems. *Mathematical Programming Computation*, *7*(3), 289–329.

Frison, G., Kufoalor, D. K. M., Imsland, L., & Jørgensen, J. B. (2014). Efficient implementation of solvers for linear model predictive control on embedded devices. In *Proceedings of the 2014 IEEE Conference on Control Applications* (pp. 1954–1959). Antibes, France.

Frison, G., Sørensen, H. H. B., Dammann, B., & Jørgensen, J. B. (2014). High-performance small-scale solvers for linear model predictive control. In *Proceedings of the 2014 European Control Conference* (pp. 128–133). Strasbourg, France.

Hehn, M., & D'Andrea, R. (2011). A flying inverted pendulum. In *Proceedings of the IEEE International Conference on Robotics and Automation* (pp. 763–770). Shanghai, China.

Houska, B., Ferreau, H., & Diehl, M. (2011). An auto-generated real-time iteration algorithm for nonlinear MPC in the microsecond range. *Automatica*, *47*(10), 2279–2285.

Jerez, J. L., Constantinides, G. A., & Kerrigan, E. C. (2011). An FPGA implementation of a

sparse quadratic programming solver for constrained predictive control. In *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (pp. 209–218). Monterey, USA.

Jerez, J. L., Goulart, P. J., Richter, S., Constantinides, G. A., Kerrigan, E. C., & Morari, M. (2014). Embedded online optimization for model predictive control at megahertz rates. *IEEE Transactions on Automatic Control*, *59*(12), 3238–3251.

Kalmari, J., Backman, J., & Visala, A. (2015). A toolkit for nonlinear model predictive control using gradient projection and code generation. *Control Engineering Practice*, *39*, 56 - 66.

Kawakami, S., Ono, T., Ohtsuka, T., & Inoue, K. (2018). Parallel precomputation with input value prediction for model predictive control systems. *IEICE Transactions on Information and Systems*, *101*(12), 2864–2877.

Kouzoupis, D., Quirynen, R., Houska, B., & Diehl, M. (2016). A block based ALADIN scheme for highly parallelizable direct optimal control. In *Proceedings of the 2016 American Control Conference* (pp. 1124–1129). Boston, USA.

McLeod, R. M. (1965). Mean value theorems for vector valued functions. *Proceedings of the Edinburgh Mathematical Society*, *14*(3), 197–209.

Nesterov, Y. E. (1983). A method for solving the convex programming problem with convergence rate $O(1/k^2)$. In *Soviet Mathematics Doklady* (Vol. 269, pp. 543–547).

Nielsen, I., & Axehill, D. (2018). Direct parallel computations of second-order search directions for model predictive control. *IEEE Transactions on Automatic Control*, *64*(7), 2845–2860.

Nocedal, J., & Wright, S. J. (2006). *Numerical optimization, second edition.* New York: Springer Science and Business Media.

Ohtsuka, T. (2004). A continuation/GMRES method for fast computation of nonlinear receding horizon control. *Automatica*, *40*(4), 563–574.

Quirynen, R. (2017). *Numerical simulation methods for embedded optimization* (Unpublished doctoral dissertation). Albert-Ludwigs-Universität Freiburg.

Rawlings, J. B., Angeli, D., & Bates, C. N. (2012). Fundamentals of economic model predictive control. In *Proceedings of the 51st IEEE Conference on Decision and Control* (pp. 3851–3861). Maui, Hawaii.

Saad, Y., & Schultz, M. H. (1986). GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, *7*(3), 856–869.

Soudbakhsh, D., & Annaswamy, A. M. (2013). Parallelized model predictive control. In *Proceedings of the 2013 American Control Conference* (pp. 1715–1720). Washington, DC, USA.

Stryk, O. V., & Bulirsch, R. (1992). Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, *37*(1), 357–373.

Torrisi, G., Grammatico, S., Smith, R. S., & Morari, M. (2018). A projected gradient and constraint linearization method for nonlinear model predictive control. *SIAM Journal on Control and Optimization*, *56*(3), 1968–1999.

Verschueren, R., Frison, G., Kouzoupis, D., van Duijkeren, N., Zanelli, A., Quirynen, R., & Diehl, M. (2018). Towards a modular software package for embedded optimization. In *Proceedings of the 6th IFAC Conference on Nonlinear Model Predictive Control* (pp. 426–432). Madison, USA.

Wächter, A., & Biegler, L. (2006a). Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, *16*(1), 1–31.

Wächter, A., & Biegler, L. T. (2006b). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, *106*(1), 25–57.

Wang, Y., & Boyd, S. (2010). Fast model predictive control using online optimization. *IEEE Transactions on Control Systems Technology*, *18*(2), 267–278.

Yamashita, H. (1998). A globally convergent primal-dual interior point method for constrained optimization. *Optimization Methods and Software*, *10*(2), 443–469.

Yang, X., & Biegler, L. T. (2013). Advanced-multi-step nonlinear model predictive control.

Journal of Process Control, *23*(8), 1116–1128.

Zanelli, A., Domahidi, A., Jerez, J., & Morari, M. (2017). FORCES NLP: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, 1–17.

Zanelli, A., Quirynen, R., Jerez, J., & Diehl, M. (2017). A homotopy-based nonlinear interior-point method for NMPC. In *Proceedings of the 20th IFAC World Congress* (pp. 13188–13193). Toulouse, France.