# *Parallelized Model Predictive Control*

# Parallelized Model Predictive Control

Damoon Soudbakhsh and Anuradha M. Annaswamy

*Abstract*— **Model predictive control (MPC) has been used in many industrial applications because of its ability to produce optimal performance while accommodating constraints. However, its application on plants with fast time constants is difficult because of its computationally expensive algorithm. In this research, we propose a parallelized MPC that makes use of the structure of the computations and the matrices in the MPC. We show that the computational time of MPC with prediction horizon $N$ can be reduced to $O(\log(N))$ using parallel computing, which is significantly less than that with other available algorithms.**

## I. INTRODUCTION

Model Predictive Control (MPC) has been used in many industrial applications successfully [1], since it can compute optimal control inputs while accommodating constraints. However, its implementation for plants with high bandwidths is challenging because of its computationally expensive algorithm. The problem of controlling multiple applications using shared resources, which is typical in a distributed embedded system, DES, can introduce large delays due to arbitration in the network where other applications with higher priority may need to be serviced. A part of these delays may be known, as significant information is available regarding the structure of the DES [2]. An MPC that incorporates this known information about the delay provides an opportunity to realize improved control performance. This however exacerbates the computational complexity further, as the dimension of the underlying control problem increases further. Therefore, development of a fast MPC algorithm that can significantly reduce the overall computational lag is highly attractive.

In recent years, significant amount of work has been done on efficient implementations of MPC. One way to implement MPC is to compute a piecewise affine control law offline to reduce the online control computation to a stored look up table [3]. A drawback of this method is the exponential relation of the number of regions with the size of the control problem [4], [5]. Another approach is to obtain a compact form of the optimization problem by substituting the predicted states by the planned inputs [6], [3]. A common procedure that is adopted in the literature to reduce computational complexity of this problem is the use of any underlying sparsity in the matrices [7], [8] and using primal-dual methods to solve MPC problem [7], [6], [4]. The computational time of this type of formulation can be reduced further by limiting the number of iterations and using suboptimal solutions [4]. These approaches can reduce the computational time of MPC to be linear with prediction horizon at best. Although this is a huge improvement over using the original form of MPC, it is an obstacle in extending MPC applications.

In this paper, we propose a parallelized implementation of the MPC in order to arrive at an efficient implementation with reduced computational time. This is accomplished by making use of the underlying sparsity in the MPC parameters and suitable pipelining of the computations into more than one processor. Similar concepts have been explored in [6], [9]. Unlike [6], where a compact form of MPC with no delay was implemented, in this paper we include a delay, and use a sparse formulation of MPC to take advantage of its structure and reduce the computational time. The MPC design of [9] is based on a sparse formulation; however, the overall computational time of MPC with prediction horizon $N$ is in order of $O(N^2)$. Here, we developed a parallelized MPC algorithm whose computation is in the order of $log(N)$. The implications of these results on hardware implementations on platforms such as Field Programmable Gate Arrays (FPGA) are also briefly explored.

The paper is organized as follows. First the dynamics of the plant is presented in §II. Then the design of MPC is presented in §III. In this section, the stage cost, dynamics of the system as well as input and state constraints are written in sparse form to reveal some of structures within them. The overall procedure to find the optimal input is given in §III-D. In §IV, we exploit the structures within the MPC matrices further and possibilities for parallelization to reduce the computational time to $O(logN_d)$ are presented. Some concluding remarks are given in §V.

## II. NCS MODEL

The problem is to control a plant using a distributed embedded architecture that is required to service multiple applications. Due to such servicing, during the control of a plant, the sensor signal, the computations of the controller block, and actuator signal experience delays $\tau_s$, $\tau_c$, and $\tau_a$ respectively (see Fig. 1). The problem is to co-design the controller and the DES architecture so that the plant can be controlled accurately with minimal computational time.

### A. Problem Statement

The specific plant to be controlled is assumed to be linear time-invariant and has the following state-space form:

$$\dot{x}(t) = Ax(t) + Bu(t-\tau) \tag{1}$$

where, $x(t) \in \Re^p$ and $u(t) \in \Re^q$ are states and inputs, respectively. We assume that the plant is periodically sampled with a fixed period $T$, and define $\tau = \tau_a + \tau_s + \tau_c$ (Fig.

This work was supported in part by the NSF Grant No. ECCS-1135815 via the CPS initiative.

D. Soudbakhsh and A. M. Annaswamy are with the Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge Ma 02139, email: {damoon, aanna} @mit.edu

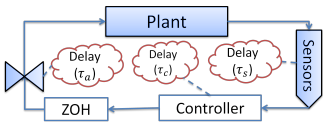


Fig. 1. Schematic overview of the control of a plant using a DES.

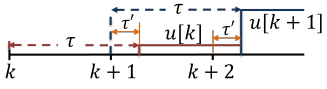


Fig. 2. Delay dependence of the effective inputs.

1). We accommodate the possibility of $\tau > T$ and define $\tau' = \tau - \lfloor \frac{\tau}{T} \rfloor T$, where $\lfloor y \rfloor$ denotes the floor($y$), i.e., the largest integer not greater than $y$ and is illustrated in Fig. 2. We now discretize (1) to obtain

$$x[k+1] = A_d x[k] + B_{d1} u[k-d_1] + B_{d2} u[k-d_2] \quad (2)$$

where $A_d \stackrel{\text{def}}{=} e^{AT}$, $B_{d1} \stackrel{\text{def}}{=} (\int_0^{T-\tau'} e^{Av} dv) B$ , and $B_{d2} \stackrel{\text{def}}{=} (\int_{T-\tau'}^{T} e^{Av} dv) B$. We note that $\tau' < T$. In (2), $d_1 \stackrel{\text{def}}{=} \lfloor \frac{\tau}{T} \rfloor$, $d_2 \stackrel{\text{def}}{=} \lceil \frac{\tau}{T} \rceil$, where $\lceil y \rceil$ denotes the ceiling($y$), i.e. the smallest integer not less than $y$. It is easy to see that $d_2 = d_1 + 1$. We also note that $B_{d2} = 0$ if $\tau = mT$, where $m$ is an integer. For ease of exposition, we denote the time instant $t_k$ as time $k$.

## III. Design of Model Predictive Control

The MPC is designed to control plant (1) with delay $\tau$ seconds to track a desired path while minimizing the control input and its rate. The states and inputs are limited by linear inequalities. Toward finding the optimal input, the controller estimates the future predicted outputs and applies the first input. This procedure is repeated at every step and a new input is applied at each step. In what follows, $y[k]$ denotes the actual value of $y$ at time $k$, and $y[k+i|k]$ is the predicted/planned value of $y$ at time $k+i$ at the current time $k$. For ease of exposition, we denote $y[k+1|k]$ as $y_{k+1}$.

We first define an extended state $X[k] \stackrel{\text{def}}{=} [u[k-d_2]^T, x_k^T]^T$ using which we rewrite (2):

$$X_{k+i+1} \stackrel{\text{def}}{=} \begin{bmatrix} u_{k+i-d_1} \\ x_{k+i+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ B_{d2} & A_d \end{bmatrix} \begin{bmatrix} u_{k+i-d_1-1} \\ x_{k+i} \end{bmatrix} + \begin{bmatrix} I \\ B_{d1} \end{bmatrix} u_{k+i-d_1} = F X_{k+i} + G u_{k+i-d_1} \quad (3)$$

At $i = d_1$, it follows that the control input $u_k$ will become available and affect $X_{k+d_1+1}$. Therefore at this instant, (3) can be simplified as:

$$X_{k+d_1+1} = \begin{bmatrix} u_k \\ x_{k+d_1+1} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ A_d B_{d1} + B_{d2} & A_d^{d_1} \end{bmatrix} \begin{bmatrix} u[k-1] \\ x_k \end{bmatrix} + \begin{bmatrix} 0 \\ \sum_{j=1}^{d_1-1} A_d^j (A_d B_{d1} + B_{d2}) u[k-j-1] \end{bmatrix} + \begin{bmatrix} 0 \\ A_d^{d_1-1} B_{d2} u[k-d_1-1] \end{bmatrix} + \begin{bmatrix} I \\ B_{d1} \end{bmatrix} u_k \quad (4)$$

### A. Stage cost

The objective of the controller is to minimize an objective function while satisfying a set of constraints. We define the cost function as:

$$J_k \stackrel{\text{def}}{=} \sum_{i=0}^{\infty} \frac{1}{2} \Big\{ e[k+i+1|k]^T Q e[k+i+1|k] + u_r[k+i|k]^T R_1 u_r[k+i|k] + \Delta u[k+i|k]^T R_2 \Delta u[k+i|k] \Big\} \quad (5)$$

where $e_{k+i} \stackrel{\text{def}}{=} x_{k+i} - x_{ref}[k+i]$ is the tracking error, $u_{r_{k+i}} \stackrel{\text{def}}{=} u_{k+i} - u_{ref}[k+i]$ is the input error, and $\Delta u_{k+i} \stackrel{\text{def}}{=} u_{k+i} - u_{k+i-1}$. The optimization problem in (5), which includes an infinite cost can be recast as a finite horizon problem leading to:

$$\min_{u,x} \; J_k = \sum_{i=1}^{N-1} \frac{1}{2} e_{k+i}^T Q e_{k+i} + \sum_{i=0}^{N_d-1} \frac{1}{2} \{ {u_{r_{k+i}}}^T R_1 u_{r_{k+i}} + \Delta u_{k+i}^T R_2 \Delta u_{k+i} \} + \frac{1}{2} e_{k+N}^T Q_{1t} e_{k+N} \quad (6)$$

where $N$ is finite, $N_d \stackrel{\text{def}}{=} N - d_1$, and $Q_t$ is the terminal cost that can be estimated by the solution of a Lyapunov discrete equation. For regulator case, the terminal cost $Q_t$ is the solution of the following Lyapunov equation:

$$Q_t = (F+GK)^T Q (F+GK) - \left( \begin{bmatrix} R_2 & 0 \\ 0 & Q \end{bmatrix} + K^T (R_1 + R_2) K + 2 K^T \begin{bmatrix} R_2 & 0 \end{bmatrix} \right) \quad (7)$$

where $K$ is the unconstrained optimal control law.

To develop the parallelized MPC, we start with the cost function over the finite horizon (6). First we note that the underlying plant is a delayed system, hence the input has no effect from $i = 0$ to $i = d_1$. Also, the quantities $x_{ref}$ and $u_{ref}$ have no effect on the optimization. Using both these facts, we can rewrite stage cost (6) as:

$$\begin{aligned} J_k = & \sum_{i=1}^{N_d-1} \Bigg\{ - \begin{bmatrix} 0 \\ x_{ref}[k+i+d_1] \end{bmatrix}^T \overbrace{\begin{bmatrix} 0 & 0 \\ 0 & Q \end{bmatrix}}^{M_2} X_{k+i+d_1} - \\ & u_{ref}[k+i]^T R_1 u_{k+i} \Bigg\} - \begin{bmatrix} 0 \\ x_{ref}[k+N] \end{bmatrix}^T \begin{bmatrix} 0 & 0 \\ 0 & Q_t \end{bmatrix} X_{k+N} + \\ & \frac{1}{2} \sum_{i=1}^{N_d-1} \left\{ \begin{bmatrix} X_{k+i+d_1} \\ u_{k+i} \end{bmatrix}^T \overbrace{\begin{bmatrix} \begin{bmatrix} R_1+R_2 & 0 \\ 0 & Q \end{bmatrix} & \begin{bmatrix} -R_2 \\ 0 \end{bmatrix} \\ \begin{bmatrix} -R_2 & 0 \end{bmatrix} & R_2 \end{bmatrix}}^{M_1} \begin{bmatrix} X_{k+i+d_1} \\ u_{k+i} \end{bmatrix} \right\} \\ & + \frac{1}{2} X_{k+N}^T \overbrace{\begin{bmatrix} R_1 & 0 \\ 0 & Q_t \end{bmatrix}}^{M_{1t}} X_{k+N} + u_k^T \overbrace{(R_1+R_2)}^{M_0} u_k \\ & - u_{ref}[k]^T R_1 u_k - u_k^T R_2 u[k-1]. \end{aligned} \quad (8)$$

Using the new variables $M_1, M_2$, and $M_{1t}$, we rewrite (8) as:

$$\begin{aligned} J_k = & \begin{bmatrix} u_k \\ X_{k+d_1+1} \\ \vdots \\ u_{k+N_d-1} \\ X_{k+N} \end{bmatrix}^T \overbrace{\begin{bmatrix} M_0 & 0 & \cdots & 0 & 0 \\ 0 & M_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & M_1 & 0 \\ 0 & 0 & \cdots & 0 & M_{1t} \end{bmatrix}}^{\mathscr{M}_1} \overbrace{\begin{bmatrix} u_k \\ X_{k+d_1+1} \\ \vdots \\ u_{k+N_d-1} \\ X_{k+N} \end{bmatrix}}^{\Phi} \\ & - \begin{bmatrix} X_{ref}[d_1+1] \\ \vdots \\ X_{ref}[N] \end{bmatrix}^T \overbrace{\begin{bmatrix} M_2 & \cdots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & M_2 & 0 \\ 0 & \cdots & 0 & M_{1t} \end{bmatrix}}^{\mathscr{M}_2} \begin{bmatrix} X_{k+d_1+1} \\ \vdots \\ X_{k+N} \end{bmatrix} \\ & - \begin{bmatrix} u_{ref}[k] \\ \vdots \\ u_{ref}[k+N_d-1] \end{bmatrix}^T \begin{bmatrix} R_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & R_1 \end{bmatrix} \begin{bmatrix} u_k \\ \vdots \\ u_{k+N_d-1} \end{bmatrix} \end{aligned} \quad (9)$$

Since $x_{ref}$ and $u_{ref}$ are independent variables, the last two terms are essentially linear combinations of $\Phi$. Hence (9) can be expressed as

$$J_k = \Phi^T \mathscr{M}_1 \Phi + c^T \Phi \quad (10)$$

In a tracking problem, $c$ should be suitably updated at each $k$. It should be noted that both $\mathscr{M}_1$ and $\mathscr{M}_2$ are block-diagonal.

### B. Inequality Constraints

We assume that the constraints on the state and control input are present in the form $A_x x \le a_x$, and $A_u u \le a_u$, where $A_x \in \Re^{m_1 \times p}$ and $A_u \in \Re^{m_2 \times q}$. We rewrite the state constraints in the forms of the extended states as:

$$A_X \stackrel{\text{def}}{=} \begin{bmatrix} A_u & 0 \\ 0 & A_x \end{bmatrix}$$

Using the above definition for $A_X$, the constraints can be written as:

$$\begin{bmatrix} A_u & 0 & 0 & \cdots & 0 \\ 0 & A_X & 0 & \cdots & 0 \\ 0 & 0 & A_u & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & A_X \end{bmatrix} \begin{bmatrix} u_k \\ X_{k+d_1+1} \\ \vdots \\ u_{k+N_d} \\ X_{k+N} \end{bmatrix} \preccurlyeq \begin{bmatrix} a_u \\ a_X \\ \vdots \\ a_X \end{bmatrix} \tag{11}$$

Denoting $s \stackrel{\text{def}}{=} [{s_1}^T \ \cdots \ {s_{N_d}}^T]^T \ge 0$ as the slack variable, and defining matrices $A_c$ and $a$, we rewrite (11) as

$$A_c \Phi + s - a = 0 \tag{12}$$

The equality constraints are given by the plant dynamics in (3) and (4). Rewriting (4) as

$$X_{k+d_1+1} = F_0 X_k + G_0 + G u_k, \tag{13}$$

with known $X_k$, all equality constraints can be written as

$$A_e \Phi - g = 0 \tag{14}$$

where $g$, and $A_e$ are defined as below:

$$g = \begin{bmatrix} -F_0 X - G_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ A_e = \begin{bmatrix} G & -I & 0 & 0 & 0 & \cdots & 0 \\ 0 & F & G & -I & 0 & \cdots & 0 \\ 0 & 0 & 0 & F & G & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & & \cdots & -I \end{bmatrix}.$$

### C. Primal-Dual Method

Combining the objective function (9), the inequality constraints (12), and the equality constraints (14), we obtain the Lagrangian:

$$\begin{aligned} \mathscr{L}(x, \lambda, \nu, s) = \ & \Phi^T \mathscr{M}_1 \Phi + c^T \Phi + \nu^T (A_e \Phi - g) \\ & + \lambda^T (A_c \Phi + s - a) \end{aligned} \tag{15}$$

where $\lambda \stackrel{\text{def}}{=} [{\lambda_1}^T, \ \cdots, \ {\lambda_{N_d}}^T]^T$ and $\nu \stackrel{\text{def}}{=} [{\nu_1}^T, \ \cdots, \ {\nu_{N_d}}^T]^T$ are the Lagrange multipliers. The KKT (Karush-Kuhn-Tucker) conditions [10] of this quadratic problem are:

$$\begin{cases} \mathscr{M}_l \Phi + c + A_c^T \lambda + A_e^T \nu = 0, \\ A_e \Phi - g = 0 \\ A_c \Phi + s - a = 0 \\ \Lambda S \mathbf{1} = 0, \lambda \ge 0, \ s \ge 0 \end{cases} \tag{16}$$

where $\Lambda \stackrel{\text{def}}{=} \text{diag}\left(\lambda_1, \lambda_2, \cdots, \lambda_{N_d}\right), S \stackrel{\text{def}}{=} \text{diag}(s_1, ..., s_{N_d})$, and $\mathbf{1}$ is a vector of ones. The solution of (16) provides an optimal $\Phi$, and in turn an optimal $U(k)$. The actual control input $u(k)$ is then determined using this optimal $U(k)$.

The solution of (16) is typically arrived at by minimizing the vector of residuals $r = (r_X, r_\nu, r_\lambda . r_s)$ over $y = (\Phi, \nu, \lambda, s)$, with the following components:

$$\begin{cases} r_X &= -\mathscr{M}_1 \Phi - c - A_c^T \lambda - A_e^T \nu, \\ r_\nu &= -A_e \Phi + g \\ r_\lambda &= -A_c \Phi - s + a \\ r_s &= -\Lambda S \mathbf{1} + \sigma \mu \mathbf{1} \end{cases} \tag{17}$$

where $\mu = \frac{\lambda^T s}{N_d m_1 + N_d m_2}$ is a measure of duality; and $m_1$ and $m_2$ are number of inequality constraints on states and inputs, respectively. $\sigma \in [0, 1]$ is the centering parameter [11]. Residual minimization of $r$ in (17) over $y$ is typically carried out by finding the optimal changes to $y = y_0$. This is done by first solving the following linear equations for the primal and dual step search directions

$$\begin{bmatrix} \mathscr{M}_1 & A_e^T & A_c^T & 0 \\ A_e & 0 & 0 & 0 \\ A_c & 0 & 0 & I \\ 0 & 0 & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta\Phi \\ \Delta\nu \\ \Delta\lambda \\ \Delta s \end{bmatrix} = \begin{bmatrix} r_X \\ r_\nu \\ r_\lambda \\ r_s \end{bmatrix} \tag{18}$$

and then iterating $y^{k+1} = y^k + \eta \Delta y$, using a line-search method to find the optimal step size $\eta$ [10]. The iteration is completed when $\|r\|_2 < \varepsilon$, an arbitrarily small constant. To find the primal and dual step search directions in (18), we simplify the problem by block elimination and get:

$$\Delta s = -s + \Lambda^{-1}(\sigma \mu \mathbf{1} - S \Delta \lambda) \tag{19}$$

$$\Delta \lambda = -S^{-1} \Lambda (A_c \Delta \Phi + s + r_\lambda) + S^{-1} \sigma \mu \mathbf{1} \tag{20}$$

$$\Delta \nu = \Gamma^{-1} \left( -r_\nu + A_e Z^{-1} R_{\nu 1} \right) \stackrel{\text{def}}{=} \Gamma^{-1} R_{\nu 2} \tag{21}$$

$$\Delta \Phi = Z^{-1}(R_{\nu 1} - A_e^T \Delta \nu) \tag{22}$$

where

$$R_{\nu 1} \stackrel{\text{def}}{=} r_X + A_c^T S^{-1} \Lambda (s + r_\lambda) - A_c^T S^{-1} \sigma \mu \mathbf{1} \tag{23}$$

we also note that matrices $Z$ and $\Gamma$ defined as

$$Z \stackrel{\text{def}}{=} L_z L_z^T \stackrel{\text{def}}{=} \mathscr{M}_1 + A_c^T S^{-1} \Lambda A_c \tag{24}$$

$$\Gamma \stackrel{\text{def}}{=} \left( A_e Z^{-1} A_e^T \right) \tag{25}$$

where, $L_z$ is the Cholesky decomposition matrices of $Z$. It can be observed that $Z$ is block diagonal and $\Gamma$ is tridiagonal. Each block $Z_{i,i} = L_{zi} L_{zi}^T$ of $Z$ can be derived to be

$$Z_{i,i} \stackrel{\text{def}}{=} \begin{cases} M_0 + A_u^T E_{u1} A_u & i = 1 \\ M_1 + \begin{bmatrix} A_u^T E_{ui} A_u & 0 & 0 \\ 0 & A_x^T E_{xi} A_x & 0 \\ 0 & 0 & A_u^T E_{uxi} A_u \end{bmatrix}, & i = 2 : N_d - 1 \\ M_{1t} + \begin{bmatrix} A_u^T E_{uxN} A_u & 0 \\ 0 & A_x^T E_{xN} A_x \end{bmatrix} & i = N_d \end{cases} \tag{26}$$

where $E_{ui}$, $E_{uxi}$, and $E_{xi}$ are the associated blocks of the matrix $E \stackrel{\text{def}}{=} S^{-1}\Lambda$, and $i = 1 : N_d$. From (26), one can compute $Z^{-1}$ in a straight forward manner using Cholesky factorization. We then repartition $Z^{-1}$ using matrices $Z_{Qi} \in \Re^{(p+q)\times(p+q)}$, $Z_{Ri} \in \Re^{q \times q}$, and $Z_{Si} \in \Re^{q \times (p+q)}$, we get the following form for $Z^{-1}$:

$$Z^{-1} = \begin{bmatrix} Z_{R1} & 0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & Z_{Q2} & Z_{s2} & 0 & \cdots & 0 & 0 \\ 0 & Z_{s2}^T & Z_{R2} & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & Z_{Q3} & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ & & & & Z_{sN_d} & Z_{RN_d} & 0 \\ 0 & 0 & 0 & \cdots & & 0 & Z_{Q_t} \end{bmatrix} \tag{27}$$

We also note that the blocks in the tridiagonal matrix $\Gamma$ can be derived to be:

$$\begin{cases} \Gamma_{1,1} = G Z_{R_1} G^T + Z_{Q_2} \\ \Gamma_{i,i} = G Z_{R_i} G^T + F Z_{Q_i} F^T + Z_{Q_{i+1}} + (G Z_{S_i}^T F^T + F Z_{S_i} G^T), \\ \qquad\qquad i = 2 : N_d \\ \Gamma_{i,i+1} = \Gamma_{i+1,i}^T = -Z_{Q_{i+1}} F^T - Z_{S_{i+1}} G^T, \quad i = 1 : N_d - 1 \end{cases} \tag{28}$$

The solution of (21) can be found efficiently using a recursive algorithm by computing the first block of $\Delta\nu$ and use it to

compute the second block and so forth. However, the sequential nature of such algorithm results in linear dependency on prediction horizon. In §IV we will show an alternative method to reduce the computational time of solving (21).

### D. Numerical procedure to find the Newton step

In §II, a design procedure to compute optimal input using MPC approach was presented. In this design an extended state was defined and structures of MPC matrices were exploited to reduce asymptotic complexity of computations. The procedure is summarized below:

The MPC control input $u[k]$ is determined by solving the minimization problem in (8) which in turn is solved through a primal-dual method. The latter is solved by finding a solution $\Delta y$ in (18), using a numerically efficient procedure that takes advantage of the sparsity of the underlying matrices. Key steps of this numerical procedure are:

1) Set $\Phi = \Phi_0, s = s_0$, and $\lambda = \lambda_0$, with all elements of the vectors $s_0$ and $\lambda_0$ being positive. Set $E = E_0$ where $E_0 = S_0^{-1}\Lambda_0$.
2) Compute residuals $r = (r_X, r_\nu, r_\lambda, r_s)$ from (17)
3) Solve for $\Delta\nu$ in (21) . The determination of $\Delta\nu$ can be further subdivided into the following tasks:
   a) Find inverse of $Z$ (by decomposition)
   b) Compute $R_{\nu 1}$ and $R_{\nu 2}$ using (23) and (21)
   c) Compute $\Gamma = A_e Z^{-1} A_e^T$
   d) Solve $\Gamma\Delta\nu = R_{\nu 2}$ for $\Delta\nu$
4) Compute $\Delta\Phi, \Delta\lambda$, and $\Delta s$ using (22), (20), and (19). We note that the inverse of $Z$ for computation of $\Delta\Phi$ is readily available.
5) Carry out line search to determine the optimal step size $\eta_{opt}$ using the following two steps:

Step 5-1 Choose $\eta' = \sup\{\eta \in [0,1] | \lambda + \eta\Delta\lambda > 0, s + \eta\Delta s > 0\}$ , Set $\eta'' = (1-\varepsilon_1)\eta'$, where $\varepsilon_1$ is a small parameter $\varepsilon_1 \in (0,1)$ chosen to avoid one-dimensional line search. set $\eta_0 = \eta''$.

Step 5-2 Iterate $\eta^j = \beta\eta^{j-1}$ until $\|r(y^k)\|_2 \leq (1 - \alpha\eta^i)\|r(y^{j-1})\|_2$ where $\alpha = \min(1-\varepsilon, 1/\eta'')$, and $\varepsilon$ is arbitrarily small.

## IV. Design of Parallel MPC

We now introduce a parallelization for the computation of $\Delta y$ in (18). The parallelization is introduced in each of the above steps discussed in §III-D. Of that, the most important component is step 3, which involves the computation of $\Gamma^{-1}$ (decomposition of $\Gamma$) in (21). In contrast to the $N_d$ steps that were shown to be needed in step 3 in §III-D, we will show that the proposed parallelization requires only $\lceil \log_2 N_d \rceil$ steps.

### A. Parallel computations

We examine each of the five steps of §III-D and check parallelizability of them.

*Step 1:* The underlying computations pertains to those of performing the Cholesky decomposition of matrix $Z(= L_z L_z^T)$ and computing vector $R_{\nu 1}$. Due to block diagonal structure of $Z$, the computation of $L_Z$ can be fully parallelized. Using $N_d$ processors, each $L_{zi}$ is computed in the $i^{\text{th}}$ processor. This is indicated using green cells in step 1 in Fig. 3. For computations of the blocks in $R_{\nu 1}$ defined in (23), denoting the $i^{\text{th}}$ block of $R_{\nu 1}$ as $R_{\nu 1}(i)$, we note that every block $i$ can be computed independently, and is represented by the purple cells in step 1.

*Step 2:* The focus of this step is the computation of $\Gamma$ and $R_{\nu 2}$. Noting that $\Gamma$ has a tridiagonal structure, the same procedure as in Step 1 can be used to perform parallel computations of the $(i,j)^{\text{th}}$ blocks of $\Gamma$, denoted by the green cells in step 2 in Fig. 3, and computations of block $i$ of $R_{\nu 2}$, denoted by the purple cells. The number of green cells in this case is $2N_d$ and the number of purple cells is $N_d$.

*Step 3:* This step involves the computation of $\Gamma^{-1}$. As $\Gamma$ is tridiagonal, computation of its inverse, in general, is not parallelizable. For ease of exposition we consider a case in which $N_d = 2^m$, with $m$ being a positive integer. In what follows we propose a specific procedure that overcomes this hurdle, and is based on an extended version of Parallel Cyclic Reduction (PCR). The main idea behind PCR is to take a tridiagonal matrix, select groups of the elements according to their even or odd matrices and reduce the problem to two tridiagonal matrices with each of the sizes being roughly half the size of the original matrix[12]. $\Gamma$ as defined in (28) is a

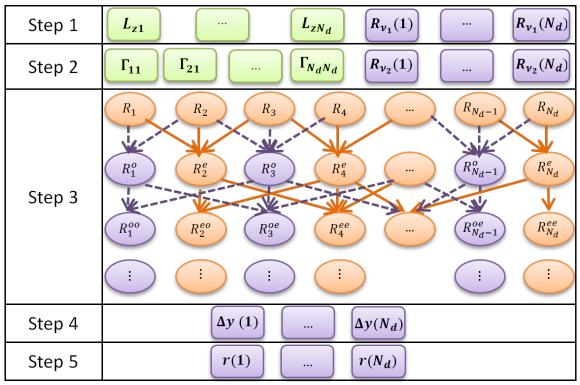


Fig. 3. Parallelizations within Steps for computing MPC input

symmetric tridiagonal matrix, and has the following form:

$$\Gamma = \begin{bmatrix} \Gamma_{11} & \Gamma_{12} & & & & \\ \Gamma_{12}^T & \Gamma_{22} & \Gamma_{23} & & & \\ 0 & \Gamma_{23}^T & \Gamma_{33} & \Gamma_{34} & & \\ & & \ddots & \ddots & \ddots & \\ & & & & \Gamma_{N_d-1,N_d}^T & \Gamma_{N_d,N_d} \end{bmatrix} \tag{29}$$

The problem at hand is the solution of

$$\Gamma\Delta\nu = R_{\nu 2} \tag{30}$$

where $\Delta\nu$ is computed of $N_d$ blocks, each of dimension $(p+q)$. We define $\Delta\nu(i)$ as the $i^{\text{th}}$ block. Using (29), (30) can be written as:

$$\begin{aligned} &(-\Gamma_{i-1,i}^T\Gamma_{i-1,i-1}^{-1}\Gamma_{i-2,i-1}^T)\Delta\nu(i-2) + \\ &\Big(\Gamma_{i,i} - (\Gamma_{i-1,i}^T\Gamma_{i-1,i-1}^{-1}\Gamma_{i-1,i}) - \Gamma_{i,i+1}\Gamma_{i+1,i+1}^{-1}\Gamma_{i,i+1}^T\Big)\Delta\nu(i) \\ &+ (-\Gamma_{i,i+1}\Gamma_{i+1,i+1}^{-1}\Gamma_{i+1,i+2})\Delta\nu(i+2) \\ &\quad = R_{\nu 2}(i) - (\Gamma_{i-1,i}^T\Gamma_{i-1,i-1}^{-1})R_{\nu 2}(i-1) \\ &\qquad - (\Gamma_{i,i+1}\Gamma_{i+1,i+1}^{-1})R_{\nu 2}(i+1) \qquad i = 1 : N_d \end{aligned} \tag{31}$$

where $\Gamma_{i,j} = 0$ for $i,j < 1$ and $i,j > N_d$. In (31), each even (odd) block of $\Delta\nu(i)$ depends only on the even (odd) blocks

$\Delta\nu(i-2)$, and $\Delta\nu(i+2)$. We define new matrices $\Gamma^o$ and $\Gamma^e$ using the coefficients of the odd and even blocks of $\Delta\nu$, respectively. Expressions for $\Gamma^o$ and $\Gamma^e$ are given below:

$$\begin{cases} \Gamma^o_{j,j+1} = {\Gamma^o_{j+1,j}}^T \stackrel{\text{def}}{=} -\Gamma_{2j-1,2j}\Gamma^{-1}_{2j,2j}\Gamma_{2j,2j+1}, \quad j = 1 : \frac{N_d}{2}-1 \\ \Gamma^o_{j,j} \stackrel{\text{def}}{=} \Gamma_{2j-1,2j-1} - \Gamma^T_{2j-2,2j-1}\Gamma^{-1}_{2j-2,2j-2}\Gamma_{2j-2,2j-1} \\ \qquad -\Gamma_{2j-1,2j}\Gamma^{-1}_{2j,2j}\Gamma^T_{2j-1,2j} \qquad j = 1 : \frac{N_d}{2} \end{cases}$$

$$\begin{cases} \Gamma^e_{j,j+1} = {\Gamma^o_{j+1,j}}^T \stackrel{\text{def}}{=} -\Gamma_{2j,2j+1}\Gamma^{-1}_{2j+1,2j+1}\Gamma_{2j+1,2j+2}, \\ \qquad\qquad\qquad\qquad\qquad\qquad j = 1 : \frac{N_d}{2}-1 \\ \Gamma^e_{j,j} \stackrel{\text{def}}{=} \Gamma_{2j,2j} - \Gamma^T_{2j-1,2j}\Gamma^{-1}_{2j-1,2j-1}\Gamma_{2j-1,2j} \\ \qquad -\Gamma_{2j,2j-1}\Gamma^{-1}_{2j+1,2j+1}\Gamma^T_{2j,2j+1}, \qquad j = 1 : \frac{N_d}{2}-1 \end{cases}$$

Using the above for $\Gamma^o$ and $\Gamma^e$ we can write the following new sets of equations for odd and even blocks of $\Delta\nu$:

$$\begin{aligned} &\Gamma^o_{j,j-1}\Delta\nu(2j-3) + \Gamma^o_{j,j}\Delta\nu(2j-1) + {\Gamma^o_{j+1,j}}^T\Delta\nu(2j+1) \\ &\quad = R^o_{\nu 2}(j), \qquad\qquad j = 1,2,...,\lceil\frac{N_d}{2}\rceil \end{aligned} \tag{32}$$

$$\begin{aligned} &\Gamma^e_{j,j-1}\Delta\nu(2j-2) + \Gamma^e_{j,j}\Delta\nu(2j) + {\Gamma^e_{j+1,j}}^T\Delta\nu(2j+2) \\ &\quad = R^e_{\nu 2}(j), \qquad\qquad j = 1,2,...,\lfloor\frac{N_d}{2}\rfloor \end{aligned} \tag{33}$$

In (32) and (33), $R^o_{\nu 2}(j)$ and $R^e_{\nu 2}(j)$ are defined as:

$$\begin{aligned} R^o_{\nu 2}(j) &= R_{\nu 2}(2j-1) - (\Gamma^T_{2j-2,2j-1}\Gamma^{-1}_{2j-2,2j-2})R_{\nu 2}(2j-2) \\ &\quad - (\Gamma_{2j-1,2j}\Gamma^{-1}_{2j,2j})R_{\nu 2}(2j), \qquad j = 1 : \frac{N_d}{2} \end{aligned} \tag{34}$$

$$\begin{aligned} R^e_{\nu 2}(j) &= R_{\nu 2}(2j) - (\Gamma^T_{2j-1,2j}\Gamma^{-1}_{2j-1,2j-1})R_{\nu 2}(2j-1) \\ &\quad - (\Gamma_{2j,2j+1}\Gamma^{-1}_{2j+1,2j+1})R_{\nu 2}(2j+1), \quad j = 1 : \frac{N_d}{2} \end{aligned} \tag{35}$$

where $R_{\nu 2}(i) = 0$ for $i < 1$ and $i > N_d$. The above simplifications allowed us to reduce the $N_d \times N_d$ block tridiagonal matrix $\Gamma$ to two block tridiagonal matrices $\Gamma^o$ and $\Gamma^e$, each with $\frac{N_d}{2} \times \frac{N_d}{2}$ blocks. The second line in step 3 in Fig 3 illustrates this transformation, with the purple cells $R^o_1, R^o_3, \cdots, R^o_{N_d}$ indicating (32), and orange cells $R^e_2, R^e_4, \cdots, R^e_{N_d-1}$ indicating (33). Using the same procedure that we used to convert the solution of $\Gamma$ to the solution of $\Gamma^e$ and $\Gamma^o$, one can convert the solution of $\Gamma^e$ and $\Gamma^o$ to that of $\Gamma^{oo}, \Gamma^{oe}, \Gamma^{eo}$, and $\Gamma^{ee}$, each of which has a block size $\frac{N_d}{4} \times \frac{N_d}{4}$. Repeating this procedure, one can arrive at $\frac{N_d}{2}$ number of equations each of which has coefficients with a block size $2 \times 2$, with two blocks of unknown parameters $\Delta\nu$ in each equation. Each of these equations can therefore be solved simultaneously to yield the entire vector $\Delta\nu$. Noting that at each step $k$, the problem reduces a block size of $\frac{N_d}{2^k} \times \frac{N_d}{2^k}$ to $\frac{N_d}{2^{k+1}} \times \frac{N_d}{2^{k+1}}$, a total number of $m-1$ steps is sufficient to reduce the block size from $N_d$ to two with the $m^{\text{th}}$ step providing the complete solution to (30). The total number of steps is therefore $m$, which equals $\log_2 N_d$. For any general $N_d$, it can be shown that $\Delta\nu$ can be determined in $\lceil\log_2 N_d\rceil$ steps with $N_d$ processors.

*Step 4:* Once $\Delta\nu$ is computed in step 3, other elements of $\Delta y$ can be computed with a series of matrix vector multiplications. This step can be divided to three sub-steps: step 4-1 to compute $\Delta\Phi$ using (22), step 4-2 to compute $\Delta\lambda$ using (20), and step 4-3 to compute $\Delta s$ using (19). We note that each of these sub-steps are parallelizable, and can be done independent of prediction horizon steps.

*Step 5:* This step is devoted to the line search algorithm to find the amount of changes in $y$ in each step that eventually results in finding the optimal input. Most of the computations of step 5 are the matrix-vector multiplications to compute residuals. It should be noted that the residuals $r = (r_X, r_\nu, r_\lambda, r_s)$ are independent of each other, and each of them has $N_d$ blocks that can be computed independently as well. However, because of iterative nature of this step, the computation of the residuals has to be repeated until satisfactory results are achieved or the maximum number of iterations is reached.

### *B. Overall Computational Time*

Up to now, we discussed the parallel structure of the MPC computations without considering the computational time of each step. In this section, we investigate the computational cost of each step assuming parallel structure of §IV is implemented. In what follows we discuss the computational time of each step of §III-D based on the parallel structure shown in Fig. 3.

Step 1 mainly focuses on computation of $L_z$ and $R_{\nu 1}$. Matrix $Z$ is block diagonal with each block defined in (26), and using $N_d$ processors, $Z$ can be decomposed in one step with cost of $O(8q^3 + p^3)$. Computations of each block of $R_{\nu 1}$ can be done independently from the prediction horizon steps as well, provided there are additional $N_d$ processors. The computational time of $R_{\nu 1}$ is in the order of $O(5m_1p^2 + 14m_2q^2)$.

In Step 2, components of block tridiagonal matrix $\Gamma$ are calculated using $N_d$ processors. The computational complexity of this task is $O(2(p+q)^3)$, while computational complexity of $R_{\nu_2}$ using $N_d$ processors is $O((p+2q)^3)$. However, the overall time taken for the first two steps is relatively small compared to step 3 since they are independent of the prediction horizon $N_d$.

Step 3 is for the computation of $\Gamma$ matrices. To estimate the total cost of step 3, we divide the computation of these new $\Gamma$s, which is associated with each row of step 3 in Fig. 3, to three sub-steps. Step 3-1 is for the factorization of diagonal blocks at each step and has a computational complexity of $O((p+q)^3)$ if $N_d$ processors are used. Step 3-2 computes the expressions $\Gamma^{-1}_{i,i}\Gamma_{i+1,i+2}$ and $\Gamma^{-1}_{i,i}\Gamma_{i,i-1}$ and has the computational time of $O((p+q)^3)$ with $2N_d$ processors. Step 3-3 pertains to the computations of the elements of the new tridiagonal matrices and using $N_d$ processors, it corresponds to a computational time of $O((p+q)^3)$. Fig 4 illustrates these three sub-steps for the computation of $\Gamma^o$ and $\Gamma^e$ from $\Gamma$, which corresponds to the second row of step 3 in Fig. 3. Since there are $\lceil\log_2 N_d\rceil$ rows in step 3 and the overall computational time is the number of steps multiplied by computations within each step, the computational time of step 3 is in the order of $O(3\lceil\log_2 N_d\rceil(p+q)^3)$.

Step 4 is for the matrix-vector multiplications to compute elements of $\Delta y$ and using $N_d$ processors, its computational complexity is $O((p+q)^3)$.

The line search algorithm of Step 5 using $N_d$ processors has the computational complexity of $O(2p^2 + 5q^2)$ for each iteration. We can estimate the order of computational time

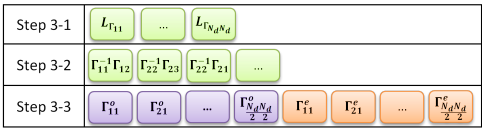


Fig. 4. Computations associated with the second row of step 3

for computing the control input by considering the computational time in each step.

As shown in this paper, by taking advantage of the structure of MPC and parallelizing the control algorithm, we can significantly reduce the computational time. This in turn allows the algorithm to be efficiently implemented using different hardware structures such as multi-core processors and FPGA. An efficient hardware implementation can in turn facilitate the application of MPC to many portable, low-power, embedded, and high bandwidth applications [13]. The emergence of FPGA has generated even more attention to the topic of hardware implementation of MPC [6], [9]. This may be due to the fact that building FPGA prototypes costs much less than building an ASIC (Application-Specific Integrated Circuit) prototype [14]. Also, compared to General Purpose Processor (GPU), implementing MPC algorithm on FPGA is more energy efficient [9]. Given that the algorithm proposed in this paper results in a much shorter computation time compared to those in [6], [9], [13], we expect its implementation in FPGA to be equally advantageous as well. In particular assuming that the system is implemented on an FPGA with $N_d$ available processors and the computation of $\Gamma$s in step 3 of the algorithm takes $\delta$ seconds, our results above imply that the time to compute $\Delta v$ would be in the order of $O(3\lceil \log_2 N_d \rceil (p+q)^3 \delta)$.

*Remark 1:* It should be noted that the parallelism and reduction of number of time unit steps was achieved by introducing more computations in each step of solving (21). Also, there are more communications among the processors that can add to the computational time as a result of the communication delays. The communication delay increases linearly with length and is in the order of nano seconds [15]. The overall computational time is therefore expected to be much less than other methods and justifies using the proposed algorithm at the expense of adding communication delays.

*Remark 2:* Many commercial software packages use Mehrotra's predictor-corrector algorithm [16] in solving interior point problems, which typically results in fewer number of outer iterations. This is achieved by solving the step search problem (18) twice in each step with different right hand sides that makes it most effective with factorization methods. However, in a problem with large prediction horizon, the computational time of solving the problem using the proposed method of this paper is much shorter than using factorization methods.

## V. Conclusions

In this research a parallel algorithm to take full advantage of the structures of MPC with quadratic cost function is proposed. First, MPC was reformulated to get a form with sparse matrices. The numerical algorithm to solve the interior point problem associated with finding the optimal input was divided to 5 steps. The computational time of different steps were analyzed to detect computational bottlenecks. We showed that the involved matrices in MPC problem have block diagonal and block tridiagonal structures. Using sparse format of the derived matrices, computations within each steps were parallelized to minimize the time to finish the task. Step 3, which is solving a system of equations involving block tridiagonal matrix $\Gamma$, is the most burdensome part of computations. We proposed to compute the step search direction $\Delta y$ by making use of tridiagonal symmetric structure of $\Gamma$ to reduce the computational time relative to the prediction horizon to $O(\lceil \log_2 N_d \rceil)$. This reduction of computational time paves the way for more high bandwidth and real-time application of MPC.

## Acknowledgment

The authors would like to thank Prof. Samarjit Chakraborty and Dr. Dip Goswami of TU Munich for their helpful discussions and suggestions.

## References

[1] S. Qin and T. Badgwell, "A survey of industrial model predictive control technology," *Control engineering practice*, vol. 11, no. 7, pp. 733–764, 2003.

[2] A. Annaswamy, D. Soudbakhsh, R. Schneider, D. Goswami, and S. Chakraborty, "Arbitrated network control systems: A co-design of control and platform for cyber-physical systems," in *Workshop on Control of Cyber Physical Systems*, 2013.

[3] A. Bemporad, M. Morari, V. Dua, and E. Pistikopoulos, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[4] 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.

[5] J. L. Jerez, E. C. Kerrigan, and G. A. Constantinides, "A condensed and sparse QP formulation for predictive control," in *CDC-ECC'11*, 2011, pp. 5217 –5222.

[6] K. Ling, S. Yue, and J. Maciejowski, "A FPGA implementation of model predictive control," in *ACC'06*, 2006, pp. 1930–1935.

[7] S. Wright, "Applying new optimization algorithms to model predictive control," in *Fifth International Conference on Chemical Process Control CPC V*, 1997, pp. 147–155.

[8] S. Boyd and B. Wegbreit, "Fast computation of optimal contact forces," *IEEE Transactions on Robotics*, vol. 23, no. 6, pp. 1117 – 1132, 2007.

[9] J. Jerez, G. Constantinides, E. Kerrigan, and K. Ling, "Parallel mpc for real-time FPGA-based implementation," in *IFAC World Congress*, 2011.

[10] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge Univ Pr, 2004.

[11] S. Wright, *Primal-dual interior-point methods*. Society for Industrial Mathematics, 1997, vol. 54.

[12] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[13] A. Wills, G. Knagge, and B. Ninness, "Fast linear model predictive control via custom integrated circuit architecture," *IEEE Transactions on Control Systems Technology*, vol. 20, no. 1, pp. 59 –71, 2012.

[14] P. Fasang, "Prototyping for industrial applications [industry forum]," *Industrial Electronics Magazine, IEEE*, vol. 3, no. 1, pp. 4 –7, 2009.

[15] T. Mak, P. Sedcole, P. Cheung, and W. Luk, "Average interconnection delay estimation for on-FPGA communication links," *Electronics Letters*, vol. 43, no. 17, pp. 918 –920, 2007.

[16] S. Mehrotra, "On the implementation of a primal-dual interior point method," *SIAM Journal on optimization*, vol. 2, no. 4, pp. 575–601, 1992.