The Parallelization of Riccati Recursion

Forrest Laine and Claire Tomlin

Abstract—A method is presented for parallelizing the computation of solutions to discrete-time, linear-quadratic, finite-horizon optimal control problems, which we will refer to as LQR problems. This class of problem arises frequently in robotic trajectory optimization. For very complicated robots, the size of these resulting problems can be large enough that computing the solution is prohibitively slow when using a single processor. Fortunately, approaches to solving these type of problems based on numerical solutions to the KKT conditions of optimality offer a parallel solution method and can leverage multiple processors to compute solutions faster. However, these methods do not produce the useful feedback control policies that are generated as a by-product of the dynamic-programming solution method known as Riccati recursion. In this paper we derive a method which is able to parallelize the computation of Riccati recursion, allowing for super-fast solutions to the LQR problem while still generating feedback control policies. We demonstrate empirically that our method is faster than existing parallel methods.

I. INTRODUCTION

When solving a continuous-state, continuous-action trajectory optimization problem for robotic path planning, many of the well known methods for solving these problems result in subproblems taking the form of a discrete-time, time-varying, finite-horizon optimal control problem with linear dynamics and a quadratic objective [1], [2], [3], [4], [5]. The class of such problems has a long history of research, and is commonly referred to as the Linear Quadratic Regulator (LQR) [6]. In applications in which the trajectory optimization must be computed online, such as in Model Predictive Control frameworks [5], we require methods for solving this subproblem very efficiently.

Fortunately, LQR problems offer efficient methods for computing solutions based on dynamic programming, sometimes referred to as the discrete-time Riccati recursion. Alternatively, one could solve these types of problem by direct methods based on the KKT system of equations given for the optimization problem. Both of these methods require computation that scales linearly with respect to the number of discrete time-points in the trajectory [7], [3]. The direct methods based on solving the KKT system of equations rely on the banded nature of the coefficient matrix of the equations. For this reason, the solution to these problems can also utilize parallelized methods for solving banded systems of equations [8], [9]. These parallel methods allow applications to utilize all of the available cores available on a computer to compute solutions as fast as possible.

Though there exist parallelized methods for computing optimal trajectories in the LQR problem, because they are based on direct methods, they only compute numerical values of the trajectories and do not include the feedback control policies that are generated through Riccati recursion. In online Model Predictive Control applications, these policies can be used to stabilize the system to the planned trajectory until the next trajectory is generated. These policies can also be used to stabilize single-shooting approaches used in nonlinear optimization methods [10].

It would therefore be desirable to have a method which can parallelize the solution to Riccati recursion, so that feedback control policies can be generated in a way that also leverages multiple processors to divide up computation. There are two main ingredients that are needed to derive such a method. First, a means of breaking up the trajectory optimization problem, which we will call the global problem, into subproblems so that they can be computed in parallel. Second, a method is needed which can solve those subproblems such that their solutions can be reconstructed into the solution of the global problem. The contributions of this paper are derivations of both ingredients, and as such, a resulting method for parallelizing Riccati recursion.

In section II we present our method. In that presentation, we will first introduce the global problem, and propose a method for how it can be parallelized. We then will present a means for computing the solution to the resulting subproblems, and show how the solutions can be reconstructed. In section III we discuss properties of our method, and demonstrate empirically that our method in fact computes the solutions to the global problem. We also show that our method is faster than the fastest-known method for solving the KKT system of equations in parallel.

II. PARALLELIZINGRICCATI RECURSION

The global LQR problem which we wish to parallelize will be defined as the following:

\[
\min_{x_0, u_0, ..., u_{T-1}, x_T} \sum_{t=0}^{T-1} cost_t(x_t, u_t) + cost_T(x_T) \tag{1a}
\]

s.t. \( \text{dynamics}_t(x_{t+1}, x_t, u_t) = 0 \ \forall t \in \{0, ..., T - 1\} \tag{1b} \)

\( x_0 = x_{\text{init}} \tag{1c} \)

Where \( x_t \in \mathbb{R}^n \), \( u_t \in \mathbb{R}^m \), the functions \( cost_t : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R} \), \( cost_T : \mathbb{R}^n \rightarrow \mathbb{R} \), \( \text{dynamics}_t : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n \), are
functions defined as:

\[
\text{cost}_t(x, u) = \frac{1}{2} x^\top \begin{pmatrix} 1 \\ \mu_1 \end{pmatrix} \begin{pmatrix} 0 & q_{11t}^\top \\ q_{11t} & Q_{xx_t} Q_{uu_t} Q_{ux_t} \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix} \quad \text{(2a)}
\]

\[
\text{cost}_t(x) = \frac{1}{2} x^\top \begin{pmatrix} 1 \\ \mu_1 \end{pmatrix} \begin{pmatrix} 0 & q_{11t}^\top \\ q_{11t} & Q_{xx_t} Q_{uu_t} Q_{ux_t} \end{pmatrix} \begin{pmatrix} 1 \\ x \end{pmatrix} \quad \text{(2b)}
\]

dynamics_t(x_{t+1}, x_t, u_t) = x_{t+1} - (F_{x_t} x_t + F_{u_t} u_t + f_t) \quad \text{(2c)}

We assume that the block coefficients $Q_{uu_t}$ of the quadratic functions $\text{cost}_t$ are positive-definite, and the terms $Q_{xx_t} - Q_{uu_t} Q_{uu_t}^{-1} Q_{xx_t}$ is positive semi-definite, so that problem \[ \text{I} \] is strictly convex.

\section*{A. Dividing into Sub-trajectories}

The way problem \[ \text{I} \] will be broken into independent sub-problems will come from the KKT conditions of optimality. Together, along with the dynamic constraints \[ \text{II} \], the KKT conditions for problem \[ \text{I} \] can be stated as

\[
\begin{pmatrix} Q_{xx_t} & Q_{uu_t} \\ Q_{uu_t} & Q_{uu_t} \end{pmatrix} \begin{pmatrix} x_t \\ u_t \end{pmatrix} + \begin{pmatrix} q_{11t} \\ q_{11t} \end{pmatrix} + \begin{pmatrix} I & -F_{x_t}^\top \\ -F_{u_t}^\top & -F_{u_t} \end{pmatrix} \begin{pmatrix} \lambda_t \\ \lambda_{t+1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \forall t \in \{0, ..., \tau - 1\} \quad \text{(3a)}
\]

\[
Q_{xx_t} x_t^2 + q_{11t} + \lambda_T = 0 \quad \text{(3b)}
\]

Here $\lambda_{t+1}$ is the Lagrange multiplier associated with constraints \[ \text{II} \], and $\lambda_t$ is the multiplier associated with constraint \[ \text{Ic} \]. Finding a set of $x_t, u_t, \lambda_t$ that satisfy these equations will result in the optimal solution to the global problem \[ \text{I} \].

Notice that we can modify the equations \[ \text{I} \] around some time $\tau$ to obtain another set of equivalent equations:

\[
\begin{pmatrix} Q_{xx_t} & Q_{uu_t} \\ Q_{uu_t} & Q_{uu_t} \end{pmatrix} \begin{pmatrix} x_t \\ u_t \end{pmatrix} + \begin{pmatrix} q_{11t} \\ q_{11t} \end{pmatrix} + \begin{pmatrix} I & -F_{x_t}^\top \\ -F_{u_t}^\top & -F_{u_t} \end{pmatrix} \begin{pmatrix} \lambda_t \\ \lambda_{t+1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad \forall t \in \{0, ..., \tau - 1\} \quad \text{(4a)}
\]

\[
\lambda_T = \mu_T = 0 \quad \text{(4b)}
\]

\[
x_T = x_{\text{link}} \quad \text{(4c)}
\]

\[
Q_{xx_T} x_T^2 + q_{11T} + \lambda_T = 0 \quad \forall t \in \{\tau, ..., T - 1\} \quad \text{(4d)}
\]

\[
Q_{xx_T} x_T^2 + q_{11T} + \lambda_T = 0 \quad \text{(4e)}
\]

Here, $x_{\text{link}} = x_T$, is the optimal point from the global problem. This has the effect of constraining $x_T$ to be its optimal value. The term $\mu_T$ is an auxiliary variable whose purpose will be come clear shortly. Notice that these set of equations \[ \text{II} \] correspond to the KKT conditions for two independent optimization problems:

\[
\min_{x_0, x_0, ..., x_{T-1}, u_1} \sum_{t=0}^{T-1} \text{cost}_t(x_t, u_t) \quad \text{(5a)}
\]

\[
\text{s.t. } \text{dynamics}_t(x_{t+1}, x_t, u_t) = 0 \quad \forall t \in \{0, ..., \tau - 1\} \quad \text{(5b)}
\]

\[
x_0 = x_{\text{init}} \quad \text{(5c)}
\]

\[
x_T = x_{\text{link}} \quad \text{(5d)}
\]

\[
\min_{x_0, x_0, ..., x_{T-1}, u_1} \sum_{t=0}^{T-1} \text{cost}_t(x_T, u_T) + \sum_{t=\tau}^{T-1} \text{cost}_t(x_t, u_t) \quad \text{(5e)}
\]

\[
\text{s.t. } \text{dynamics}_t(x_{t+1}, x_t, u_t) = 0 \quad \forall t \in \{\tau, ..., T - 1\} \quad \text{(5f)}
\]

\[
x_T^2 = x_{\text{link}} \quad \text{(5g)}
\]

$x_T$ and $x_T^2$ are used to distinguish the variables in the two optimization problems, since $x_T$ appears in both problems. From this, it can be seen that the term $\mu_T$ appearing in equation \[ \text{II} \] is the Lagrange multiplier corresponding to constraint \[ \text{II} \].

Therefore, if the value of $x_{\text{link}}$ could be determined somehow, solving the problems \[ \text{III} \] independently would provide a means for solving the global problem in parallel. It turns out that by computing the optimal solutions and the Lagrange multipliers corresponding to the endpoint constraints in each of the sub-problems \[ \text{III} \] as functions of the unknown point $x_{\text{link}}$, the resulting solutions can be used to solve for the point $x_{\text{link}}$. Since the solutions of the sub-problems are functions of $x_{\text{link}}$, by plugging it back into those solutions, the global solution can be recovered.

The method for computing solutions as a function of the endpoint will presented in the following section. For now assume that such a method exists, giving solutions to each
of the subproblems (5) as:

\[ x^1_t = x^2_t = x_{\text{link}} \]
\[ \lambda^1_t = L^1_{xt}x_{\text{link}} + l^1_t \]
\[ \lambda^2_t = L^2_{xt}x_{\text{link}} + l^2_t \]
\[ x^i_t = R_i x_{\text{link}} + r_i, \quad t \in \{0, \ldots, \tau - 1, \tau + 1, \ldots, T\} \]
\[ u^i_t = S_i x_{\text{link}} + s_i, \quad t \in \{0, \ldots, T - 1\} \]
\[ \lambda_x = J_{xt}x_{\text{link}} + l_x, \quad t \in \{0, \ldots, \tau - 1, \tau + 1, \ldots, T\} \]
\[ \mu_t = E_t x_{\text{link}} + e_t. \]

By noticing that the condition \( \mu_t = -\lambda^2_x \) is required if the KKT conditions are to be equivalent to the equations in (4), we obtain:

\[ (E_t + L^2_{xt})x_{\text{link}} + (e_t + l^2_t) = 0 \]

This is a system of \( n \) equations in \( n \) unknowns, which can be solved for \( x_{\text{link}} \). This assumes \( E_t + L^2_{xt} \) is full rank, which it will be when the global problem is strictly convex. With the solution of \( x_{\text{link}} \), the solutions of the subproblems (and by virtue of the equivalence, the global problem) given in (6) can be evaluated.

This argument made relating the Lagrange multipliers of neighboring trajectories holds for an arbitrary number of sub-trajectories, given that a solution to each sub-optimization problem will result in the overall method for computing solutions in the manner outlined will be very efficient. We present such a method now.

### B. Constraint-explicit LQR

We present a method for computing control policies of the following end-point constrained LQR problem, explicitly as a function of the values \( x_{\text{init}} \) and \( x_{\text{term}} \):

\[
\min_{x_0, u_0, \ldots, u_{T-1}, x_T} \quad cost_T(x_T) + \sum_{t=0}^{T-1} cost_t(x_t, u_t) \\
\text{s.t.} \quad \text{dynamics}(x_{t+1}, x_t, u_t) = 0 \quad \forall t \in \{0, \ldots, T - 1\} \\
x_0 = x_{\text{init}} \\
x_T = x_{\text{term}}
\]

Here the functions appearing in (8a, 8b) take the same form as in (1). It is assumed that the constraints (8b, 8d) are linearly independent. If there are linearly dependent constraints, they can simply be removed without changing the problem.

The method we will use to solve this problem is based on the method presented in [12], except we will maintain all control policies, value-functions and constraints as explicit functions of the terminal state \( x_{\text{term}} \).

The derivation of the control policies for the constrained follows a dynamic programming approach. Starting from the end of the trajectory and working towards the beginning, a given control input \( u_t \) will be chosen such that for any value of the corresponding state \( x_t \) and terminal constraint value \( x_{\text{term}} \), the remaining trajectory will be cost-optimal and the residual parts of the terminal constraint will be satisfied if possible.

To formalize and calculate these controls, a time-varying quadratic value function, \( \text{cost}_J(x, x_{\text{term}}) : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R} \) will be maintained, representing the minimum possible cost remaining in the trajectory from stage \( t \) onward as a function of state at time \( t \) and the terminal constraint state \( x_{\text{term}} \). Additionally, a linear function \( \text{constraint}_J(x, x_{\text{term}}) : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n \) will be maintained which encompasses all future constraints. Let the following terms represent these functions:

\[
\text{cost}_J(x, x_{\text{term}}) = \frac{1}{2} \begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} x & x_{\text{term}} \end{pmatrix}^\top \begin{pmatrix} V_{xx} & V_{xl} \end{pmatrix} \begin{pmatrix} x \end{pmatrix} + H_{x} x + H_{z} x_{\text{term}} + h_1.
\]

We initialize these terms at time \( T \) as follows:

\[
H_{x} = I \quad H_{z} = -I \\
h_{x} = 0 \quad V_{xx} = Q_{xx} \\
v_{xl} = 0 \quad V_{z} = 0
\]

Here we abuse notation slightly, letting \( 0 \) represent both the zero vector and matrix, with corresponding size which should be clear from context. \( I \) is the \( n \times n \) identity matrix.

According to the procedure outlined in [12], the following optimization problem is solved starting at time \( t = T - 1 \) and
continuing back to $t = 0$:

$$u_t^* = \arg \min_u cost_t(x_t, u_t) + cost_{t_0 \to t_1}(x_{t_1}, x_{term}) \quad (12a)$$

s.t.  

$$0 = dynamics_{t}(x_{t_1}, x_t, u_t) \quad (12b)$$

$$u_t \in \arg \min_u \| constraint_{t_0 \to t_1}(x_{t_1}, x_{term}) \|_2 \quad (12c)$$

Again, as shown in [12], problem (12) is equivalent to the following problem:

$$y_t^*, w_t^* = \arg \min \frac{1}{2} \| N_u x_t + N_u P_{y_t} y_t + N_{x_{term}} + n_t \|_2 +$$

$$\frac{1}{2} \begin{pmatrix} x_t \cr m_{x_1} \cr M_{u\dot{x}} \end{pmatrix}^T \begin{pmatrix} 1 \\ m_{x_1}^T \cr M_{u\dot{x}}^T \cr M_{x_{term}}^T \cr z_{x_{term}} \cr x_{term} \end{pmatrix} \begin{pmatrix} 1 \\ x_t \cr m_{x_1} \cr m_{x_1} \cr M_{u\dot{x}} \cr M_{x_{term}} \cr M_{x_{term}} \cr x_{term} \end{pmatrix}$$

$$u_t^* = P_{y_t} y_t^* + Z_{x_{term}} w_t^* \quad (13a)$$

Where the above terms are defined as:

$$m_{x_1} = q_{x_1} + F_{x_1} x_{t_1}, \quad m_{x_1} = q_{x_1} + F_{x_1} x_{t_1}, \quad M_{x_{term}} = 0$$

$$M_{x_1} = q_x + F_{x} x_{t_1}, \quad M_{u\dot{x}} = M_{x_{term}}$$

Here, $z_{x_{term}}$ is chosen such that the columns form an orthonormal basis for the null-space of $N_u$, and $P_{y_t}$ is chosen such that its columns form an orthonormal basis for the range-space of $N_u$. Hence $N_u$ and $P_{y_t}$ are orthogonal and their columns together span $\mathbb{R}^m$.

Problem (13), which is an unconstrained problem, has a solution given by the following:

$$y_t^* = -(N_u P_{y_t})^T (N_u x_t + N_{x_{term}} + n_t) \quad (14)$$

$$w_t^* = -(Z_{x_{term}} M_{u\dot{x}} Z_{x_{term}})^{-1} Z_{x_{term}}^T (M_{x_{term}} x_t + M_{x_{term}}^T n_{x_{term}}) \quad (15)$$

Here $\dagger$ represents the pseudo-inverse. In the case that $P_{y_t}$ is a size-zero matrix (i.e. $\dim(null(N_u)) = m$), $z_{x_{term}}$ is $I_m$ (Identity matrix $\in \mathbb{R}^{m \times m}$), and $y_t$ has dimension 0. Correspondingly, when $z_{x_{term}}$ is a size-zero matrix, $P_{y_t}$ is $I_m$ and $y_t$ has dimension 0. Therefore we ignore one of the correspondingly empty update equations above (14 or 15).

The control $u_t$ can now be expressed in closed-form as an affine function of the state $x_t$:

$$u_t^* = K_{x_t} x_t + K_{x_{term}} + k_t \quad (16)$$

$$K_{x_t} = -(P_{y_t}^T (N_u P_{y_t})^T N_u + Z_{x_{term}}^T M_{x_{term}} Z_{x_{term}})^{-1} Z_{x_{term}}^T M_{x_{term}} \quad (17)$$

$$K_{x_{term}} = -(P_{y_t}^T (N_u P_{y_t})^T N_{x_{term}} + Z_{x_{term}}^T M_{x_{term}} Z_{x_{term}})^{-1} Z_{x_{term}}^T M_{x_{term}} \quad (18)$$

$$k_t = -(P_{y_t}^T (N_u P_{y_t})^T n_t + Z_{x_{term}}^T M_{x_{term}} Z_{x_{term}})^{-1} Z_{x_{term}}^T m_{x_1} \quad (19)$$

Since the control is a function of the state, we can also express the constraint residual, i.e. the value of the term $\text{cost}_{t_0 \to t_1}(x_{t_1}, x_{term})$, as a function of the state. We can update the constraint-to-go to be this constraint residual, requiring preceding controls to enforce the residual is zero. We substitute into the right-hand side of (12c) to obtain:

$$\text{constraint}_{t_0 \to t_1}(x_t) = N_u x_t + N_{x_{term}} + n_t - N_u P_{y_t} (N_u P_{y_t})^T (N_u x_t + N_{x_{term}} + n_t) \quad (20)$$

This results in the update for the terms $H_{x_t}, H_{x_t}$ and $h_{x_t}$:

$$H_{x_t} = (I - N_u P_{y_t} (N_u P_{y_t})^T) N_u$$

$$H_{x_t} = (I - N_u P_{y_t} (N_u P_{y_t})^T) N_{x_{term}}$$

$$h_{x_t} = (I - N_u P_{y_t} (N_u P_{y_t})^T) n_t \quad (21)$$

The expression for the control is plugged in to the objective function of our optimization problem (13) to obtain an update on the cost-to-go as a function of the state and $x_{init}$:

$$V_{x_{t_1}} = M_{x_{t_1}} + 2M_{x_{t_1}}^T K_{x_t} + M_{x_{t_1}}^T K_{x_{term}} \quad (22)$$

$$V_{z_{x_{term}}} = M_{z_{x_{term}}} + 2M_{z_{x_{term}}}^T K_{x_{term}} + M_{z_{x_{term}}}^T K_{x_{term}} \quad (23)$$

We have now presented forms for all terms in the cost $\text{cost}_{t_0 \to t_1}$ and constraint $\text{constraint}_{t_0 \to t_1}$, functions, and computed linear feedback terms $K_{x_t}, K_{x_{term}}$, and $k_t$ in the process. This procedure can be repeated for time all time back to $t = 0$. If $H_{x_{t_1}} + H_{x_{term}} + h_{x_{t_1}} = 0$, then the sequence of control policies $\{K_{x_t}, K_{x_{term}}\}_{t\in[0,...,T-1]}$ will produce a sequence of states and controls that are optimal for our original problem (11). Otherwise, no feasible solution exists.

Trajectory: Once we have computed all of the control policies for the entirety of the trajectory, the system can be forward simulated from the $x_{init}$ to obtain a form for all states and controls of the trajectory as functions of the variables $x_{init}$ and $x_{term}$. We have:

$$x_t^* = R_{x_{init}} x_{init} + R_{x_{term}} x_{term} + r_t \quad (29)$$

$$u_t^* = S_{x_{init}} x_{init} + S_{x_{term}} x_{term} + s_t \quad (30)$$

Here the terms are initialized as $R_{x_{init}} = I, R_{x_{term}} = 0$, and $r_t = 0$. Otherwise, terms are recursively defined as:

$$S_{x_{init}} = K_{x_t} R_{x_{init}} \quad S_{x_{term}} = K_{x_{term}} K_{x_{term}}$$

$$s_{x_{term}} = K_{x_{term}} s_{x_{term}} + k_{x_{term}}$$

$$R_{x_{term}} = F_{x_{term}} R_{x_{term}} + F_{x_{term}} S_{x_{term}}$$

$$R_{x_{term}} = F_{x_{term}} R_{x_{term}} + F_{x_{term}} S_{x_{term}}$$

$$r_{x_{term}} = F_{x_{term}} r_{x_{term}} + F_{x_{term}} s_{x_{term}} + f_t$$

Lagrange Multipliers: Similar to the trajectory, the Lagrange multipliers associated with constraints (10) are functions of $x_{init}$ and $x_{term}$. Writing the KKT conditions of the problem (8) that involve the multipliers, we have:

$$\begin{bmatrix} Q_{x_{init}} \quad Q_{x_{term}}^T \quad Q_{x_{term}} \end{bmatrix} \begin{bmatrix} x_t^* \quad u_t^* \quad q_{x_{init}} \end{bmatrix} + \begin{bmatrix} I \quad -F_{x_{term}}^T \quad -F_{x_{term}} \end{bmatrix} \begin{bmatrix} \lambda_{x_{term}} \quad \lambda_{x_{term}} \quad \lambda_{x_{term}} \end{bmatrix} = \begin{bmatrix} 0 \quad 0 \quad 0 \end{bmatrix}$$

\[ \forall t \in \{0,...,T-1\} \quad (31) \]

$$Q_{x_{term}} x_t^* + q_{x_{term}} + \lambda x_t^* + \mu x_t^* = 0 \quad (32)$$
As before, $\lambda_0 \in \mathbb{R}^n$ is the Lagrange multiplier associated with the constraint (8c), $\lambda_{t+1} \in \mathbb{R}^n$, $0 \leq t < T$ are the Lagrange multipliers associated with constraints (8b), and $\mu_T \in \mathbb{R}^n$ is the Lagrange multiplier associated with constraint (8d).

Since the values of all optimal state and control variables, are known from (29-30), the resulting system has $(T-1)(n+m) + n$ equations with $(T+1)n$ unknowns, and is an overdetermined system. Stacking these equations into a large matrix equation gives:

$$A^T\lambda = b$$

Here we have made use of the terms:

$$\Sigma = I + F_x F_x^T + F_u F_u^T$$

Looking at terms given in (34), we have:

$$AA^T = \begin{bmatrix}
I & -F_x^T \\
-F_x & \Sigma_0 & -F_x^T \\
& \ddots & \ddots \\
& & -F_{T-2} & \Sigma_{T-2} & -F_{T-1}^T \\
& & & -F_{T-1} & \Sigma_{T-1} & I \\
& & & & I & I
\end{bmatrix}$$

$$Ab = \begin{bmatrix}
D_{a0} \\
\vdots \\
D_{aT}
\end{bmatrix} \begin{bmatrix}
x_{\text{init}} \\
\vdots \\
x_{\text{term}}
\end{bmatrix} + \begin{bmatrix}
d_{10} \\
\vdots \\
d_{1T}
\end{bmatrix}$$

Notice that because (35) is a block-tridiagonal matrix, and the form of $Ab$ is explicit with respect to the variables $x_{\text{init}}$ and $x_{\text{term}}$, we can use a block substitution or sparse gaussian elimination method on the system (34) to determine the Lagrange multipliers as explicit functions of $x_{\text{init}}$ and $x_{\text{term}}$, in a computational complexity that scales linearly with respect to the length of the control horizon.

Performing such a calculation will result in a simple expression for each of the multipliers of the form

$$\lambda_t = L_{a0} x_{\text{init}} + L_{a1} x_{\text{term}} + l_{1t}, \quad t \in \{0, \ldots, T\}$$

$$\mu_T = E_{aT} x_{\text{init}} + E_{cT} x_{\text{term}} + e_{1T}.$$
the property that even though the resulting optimal trajectories are equivalent, the feedback policies generated by our method are slightly different. We illustrate this relationship with a simple example.

A. Parallel vs. Serial Riccati Recursion

Consider the following LQR problem:

\[
\begin{align*}
\min_{u_0, \ldots, u_{T-1}} & \quad \alpha_T \|x_T\|^2 + \sum_{t=0}^{T-1} \alpha_t \|x_t\|^2 + \beta_t \|u_t\|^2 \\
\text{s.t.} & \quad x_{t+1} = \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} dt \\ 0 \end{bmatrix} u_t \\
& \quad x_0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} 
\end{align*}
\]  

(49a) (49b) (49c)

Here \( \alpha_t = 10 \), \( \alpha_T = 10^3 \), and \( \beta_t = 10^{-2} \). We compute the LQR feedback policies using standard Riccati recursion, as well as using the parallel method outlined above with \( J = 3 \) sub-trajectories. Figure 2 demonstrates that, as we expect, the resulting optimal trajectories for both methods are identical (labels “Parallel Undisturbed” and “Serial Undisturbed”). We also apply the feedback policies generated in both methods to a disturbed system given by the following dynamics:

\[
\begin{align*}
x_{t+1} &= \begin{bmatrix} 1 & dt \\ 0 & 1 \end{bmatrix} x_t + \begin{bmatrix} dt \\ 0 \end{bmatrix} u_t + \begin{bmatrix} 0 \\ x_t(1)/(x_t(1)^2 + 10^{-4}) \end{bmatrix} 
\end{align*}
\]

(50)

We see in Figure 2 that when applied to the disturbed systems, the feedback policies now generate different solutions. Because each sub-trajectory of the parallel version are constrained to end at the would-be-optimal point, we see strong corrections to these points. When the cost of these two trajectories are computed, we have \( \text{cost}_s = 2.896e3 \), and \( \text{cost}_p = 2.203e3 \) for the serial and parallel policies, respectively. This behavior is interesting. On one hand, such correction can induce added stability to the policy, and result in improved performance as seen in this toy example. On the other hand, this may not always be beneficial. An optimal point for the assumed system dynamics may not be feasible for the true system, and as such a constraint forcing to one of these points might force the controller to apply unnecessarily strong actions in attempting to achieve the infeasible constraint.

A middle-ground between these two methods can be obtained by re-computing the feedback policies for the first \( J-1 \) subtrajectories using the value function \( \text{cost}_j \| \dot{x}_{j+1} \| \) (value function at beginning of subtrajectory \( j \)) as an initialization for the \( \text{cost}_j \| \dot{x}_{j+1} \| \) function at the end of subtrajectory \( j \) in the second pass, the sub-trajectories no longer need to be constrained to terminate at the link-points. This has the effect of smoothing the control policies, and the result can also be seen in Figure 2. Note that this smoothing still results in the same optimal trajectory as the serial and parallel case when executed on the undisturbed dynamics. The cost of this smoothing procedure is roughly twice that of the non-smoothed parallel method, but might still be much faster than the serial method in some cases.

B. Computation Time vs. ScaLAPACK

We compare the computation time for our parallel method against a parallel solver for banded matrices. The KKT system of equations for an LQR problem form a banded coefficient matrix [3] which can be solved using specialized solvers for such systems. ScaLAPACK [13] is a package which provides fortran routines for solving linear algebra problems on distributed processors. It provides a method ‘PDGBSV’ which solves general banded systems using distributed processors and memory. Taking an approach based on the parallelization of banded systems was proposed in [14].

Table I compares the computation time of our method against the ScaLAPACK method. Here the dimensions of the state and control match that of the Cassie Robot, built by Agility Robotics. (Recall \( n \) is the dimension of the state, and \( m \) is the dimension of the control). All times are taken as the minimum over 10 runs on a Xeon Phi 1.3GHz 64-core processor. We use very long trajectory lengths to demonstrate the true effectiveness of parallelized methods. Our method outperforms the routine provided by ScaLAPACK in every case for this problem.

| \( n \) | \( m \) | \( T \) | Cores | ScaLAPACK (s) | Parallel Riccati (s) |
|---|---|---|---|---|---|
| 40 | 10 | 1024 | 32 | 0.074 | 0.043 |
| 40 | 10 | 1024 | 64 | 0.057 | 0.037 |
| 40 | 10 | 2048 | 1 | 0.797 | 0.727 |
| 40 | 10 | 2048 | 32 | 0.116 | 0.074 |
| 40 | 10 | 2048 | 64 | 0.085 | 0.057 |

C. Other Methods

The authors in [16] present a method which also parallelizes the computation of solutions to LQR problems. Their method is based on using approximations of the value function, grabbed from previous solutions of the control problem when used in a Model Predictive Control framework, to avoid performing the dynamic programming on the entire length of the trajectory. One issue with such a method is that the resulting solutions are only approximate, and information will take at least \( J \) cycles of planning to propagate to the start of the trajectory, where \( J \) is the number of sub-trajectories.
There exist other methods as well for parallelizing computation of controllers for LQR problems, such as in [17]. However, the approach in that work focuses on time-invariant, infinite-horizon problems and is therefore not applicable for use in trajectory optimization for nonlinear systems with time-varying objectives.

IV. CONCLUSION

We have developed a method for parallelizing Riccati recursion, allowing for very efficient computation of feedback control policies for LQR problems. An interesting relationship between Lagrange multipliers of neighboring trajectories was developed, allowing for the distribution of optimization to multiple processors. We used a simple example to illustrate the differences between the serial and parallel versions of Riccati recursion, and presented some examples showing computational speedup using our method compared to existing methods.

REFERENCES

[1] D. P. Bertsekas, Nonlinear programming. Athena scientific Belmont, 1999.
[2] S. J. Wright, “Interior point methods for optimal control of discrete time systems,” Journal of Optimization Theory and Applications, vol. 77, no. 1, pp. 161–187, 1993.
[3] ——, “Applying new optimization algorithms to more predictive control,” Argonne National Lab., IL (United States), Tech. Rep., 1996.
[4] S. Wright and J. Nocedal, “Numerical optimization,” Springer Science, vol. 35, no. 67-68, p. 7, 1999.
[5] F. Allgöwer and A. Zheng, Nonlinear model predictive control. Birkhäuser, 2012, vol. 26.
[6] H. Kwakernaak and R. Sivan, Linear optimal control systems. Wiley-Interscience New York, 1972, vol. 1.
[7] F. Borrelli, A. Bemporad, and M. Morari, Predictive control for linear and hybrid systems. Cambridge University Press, 2017.
[8] J. J. Dongarra and L. Johnsson, “Solving banded systems on a parallel processor,” Parallel Computing, vol. 5, no. 1-2, pp. 219–246, 1987.
[9] P. Arbenz, A. Cleary, J. Dongarra, and M. Hegland, “A comparison of parallel solvers for diagonally dominant and general narrow-banded linear systems,” in Parallel and Distributed Computing Practices (PCDP) 2000. Citeseer, 1999.
[10] M. Gifithaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, “A family of iterative gaus-newton shooting methods for nonlinear optimal control,” arXiv preprint arXiv:1711.11006, 2017.
[11] S. Boyd and L. Vandenberghe, Convex optimization. Cambridge university press, 2004.
[12] F. Laine and C. Tomlin, “Efficient computation of feedback control for constrained systems,” arXiv preprint arXiv:1807.00794, 2018.
[13] L. S. Blackford, J. Choi, A. Cleary, E. D’Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, et al., ScalAPACK users’ guide. SIAM, 1997.
[14] S. J. Wright, “Solution of discrete-time optimal control problems on parallel computers,” Parallel Computing, vol. 16, no. 2-3, pp. 221–237, 1990.
[15] Y. Tassa, T. Erez, and E. Todorov, “Synthesis and stabilization of complex behaviors through online trajectory optimization,” in Intelligent Robots and Systems (IROS). 2012 IEEE/RSJ International Conference on. IEEE, 2012, pp. 4906–4913.
[16] F. Farshidian, E. Jelavić, A. Satapathy, M. Gifithaler, and J. Buchli, “Real-time motion planning of legged robots: A model predictive control approach,” arXiv preprint arXiv:1710.04029, 2017.
[17] P. Benner, E. S. Quintana-Orti, and G. Quintana-Orti, “Solving linear-quadratic optimal control problems on parallel computers,” Optimization Methods & Software, vol. 23, no. 6, pp. 879–909, 2008.