GPU-Accelerated Dual Gradient-Projection Algorithm for Embedded Linear Model Predictive Control

Luke Nuculaj, Joseph Volcic, Shreyas Renukuntla, and Matthew Redoute
ECE 5770: GPU Accelerated Computing

November 2, 2024

1 Project Proposal

Our group is interested in developing a parallel implementation of the dual gradient-projection algorithm (GPAD) for solving the quadratic programming (QP) problem associated with linear model predictive control (MPC). This approach was proposed in [1] and its embedded performance explored in [2], finding an approximate primal solution to the MPC control problem, requiring a computational effort similar to existing QP solvers, and is easily programmable with basic and parallelizable operations. The problem centers around the finite-time optimal control problem formulation for MPC

$$
\begin{aligned}
V^*(p) = \min_{z} \quad & \ell_N(x_N) + \sum_{k=0}^{N-1} \ell_N(x_k, u_k) && \text{(1a)}\\
\text{s.t.} \quad & x_0 = p && \text{(1b)}\\
& x_{k+1} = A_k x_k + B_k u_k + f_k && \text{(1c)}\\
& F_k x_k + G_k u_k \leq c_k && \text{(1d)}\\
& \qquad k = 0, \dots, N-1 && \text{(1e)}\\
& F_N x_N \leq c_N && \text{(1f)}
\end{aligned}
$$

where N is the prediction horizon, $p \in \mathbb{R}^{n_x}$ is the current state vector, $u_k \in \mathbb{R}^{n_u}$ is the vector of control variables at prediction time k for n_x. Using quadratic stage and terminal costs, (1) can be expressed more succinctly

$$
\begin{aligned}
V^*(p) \triangleq \min_{z} \quad & \frac{1}{2} z^\top M z + (Cp + g)^\top z + \frac{1}{2} p^\top Y p && \text{(2a)}\\
\text{s.t.} \quad & Gz \leq Ep + b && \text{(2b)}
\end{aligned}
$$

where $z \triangleq \begin{bmatrix} u_0^\top & u_1^\top & \dots & u_{N-1}^\top \end{bmatrix}^\top$, $M \in \mathbb{S}^n_{++}$ is the Hessian matrix, $C \in \mathbb{R}^{n_x \times n}$ and $g \in \mathbb{R}^m$ are associated with the linear term of the cost function. The GPAD algorithm will be utilized to solve (2) for the optimal control sequence z^*.

The GPAD algorithm is expressed by the following iterations:

$$w_\nu = y_\nu + \beta_\nu(y_\nu - y_{\nu-1}) \tag{3a}$$
$$\hat{z}_\nu = -M_G w_\nu - g_P \tag{3b}$$
$$z_\nu = (1-\theta_\nu)z_{\nu-1} + \theta_\nu \hat{z}_\nu \tag{3c}$$
$$y_{\nu+1} = [w_\nu + G_L \hat{z}_\nu + p_D]_+ \tag{3d}$$
$$y_0 = y_{-1} = 0, \;\; z_{-1} = 0 \tag{3e}$$

where $y \in \mathbb{R}^m$ is the dual vector, $M_G \triangleq M^{-1}G^\top$, $g_P \triangleq M^{-1}(Cp+g)$, $G_L \triangleq \frac{1}{L}G$, $p_D \triangleq -\frac{1}{L}(Ep+b)$, $\nu \in \mathbb{N}$ is the iteration number, and β_ν is a scalar sequence developed by the following recursions

$$\theta_{\nu+1} = \frac{\sqrt{\theta_\nu^4 + 4\theta_\nu^2} - \theta_\nu^2}{2} \tag{4a}$$
$$\beta_\nu = \theta_\nu(\theta_{\nu-1}^{-1} - 1) \tag{4b}$$
$$\theta_0 = \theta_{-1} = 1. \tag{4c}$$

A flops analysis reveals that $(3a)$ requires $3m$ flops, $(3b)$ requires $\approx 2n_uNm$ flops, $(3c)$ requires $3n_uN$ flops, and $(3d)$ requires $\approx 2n_uNm$ flops, where m is the number of constraints, n_u is the size of the control vector u, and N is the length of the prediction horizon. As an applied example case, we are considering the series-connected battery balancing problem, which has a number of constraints $m = 4n_uN + 2N$.

The averaged execution times are visualized in Figure 1, which are averaged over various sizes of the optimization variable z and executed in MATLAB R2023b on an Intel i7 CPU running at 2.9 GHz with 32 GB of RAM.

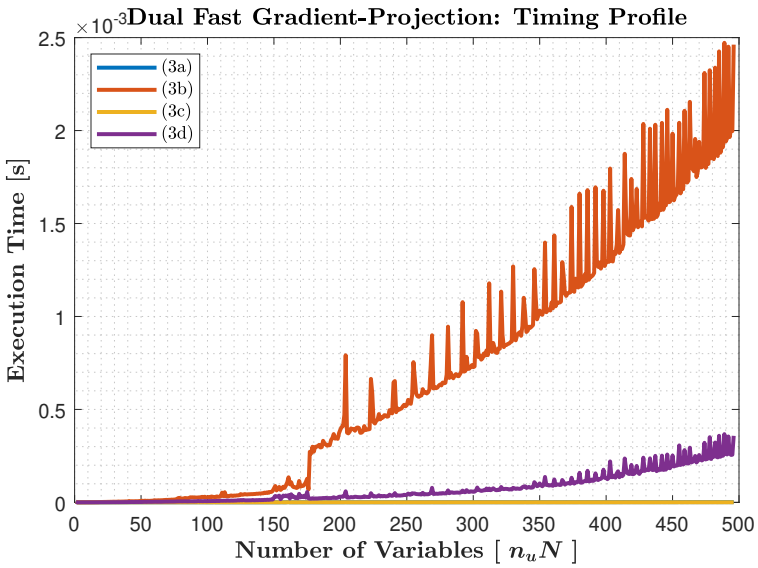

Figure 1: Timing profile for the GPAD algorithm averaged over 100 trails for each number of variables.

It is clear to see that (3b) and (3d) are the bottlenecks in the GPAD algorithm – this is reinforced by the fact that their computation grows quadratically with the number of

variables in the MPC problem while (3a) and (3c) are only linear. Upon inspection, the bottlenecks are matrix multiplications, additions, and element-wise comparisons, which are easily parallelizable ripe for a GPU implementation. To complicate things further, the larger matrices involved in these computations are sparse (almost all of their elements are zero), so special considerations will have to be made to matrix multiplication kernel functions so as to cut back on unnecessary computation and memory overhead.

References

[1] A. Bemporad and P. Patrinos, "Simple and certifiable quadratic programming algorithms for embedded linear model predictive control," *IFAC Proceedings Volumes*, vol. 45, no. 17, pp. 14–20, 2012.

[2] P. Patrinos, A. Guiggiani, and A. Bemporad, "A dual gradient-projection algorithm for model predictive control in fixed-point arithmetic," *Automatica*, vol. 55, pp. 226–235, 2015.