2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

ReLU-QP: A GPU-Accelerated Quadratic Programming Solver for Model-Predictive Control

Arun L. Bishop[1*], John Z. Zhang[1*], Swaminathan Gurumurthy[1], Kevin Tracy[1], Zachary Manchester[1]

***Abstract*— We present ReLU-QP, a GPU-accelerated solver for quadratic programs (QPs) that is capable of solving high-dimensional control problems at real-time rates. ReLU-QP is derived by exactly reformulating the Alternating Direction Method of Multipliers (ADMM) algorithm for solving QPs as a deep, weight-tied neural network with rectified linear unit (ReLU) activations. This reformulation enables the deployment of ReLU-QP on GPUs using standard machine-learning toolboxes. We evaluate the performance of ReLU-QP across three model-predictive control (MPC) benchmarks: stabilizing random linear dynamical systems with control limits, balancing an Atlas humanoid robot on a single foot, and performing a whole-body pick-up motion on a quadruped equipped with a six-degree-of-freedom arm. These benchmarks indicate that ReLU-QP is competitive with state-of-the-art CPU-based solvers for small-to-medium-scale problems and offers order-of-magnitude speed improvements for larger-scale problems.**

I. Introduction

Convex quadratic programming is a core technology in many areas of robotics, including inverse kinematics [1], contact dynamics [2], actuator design [3], and model-predictive control (MPC) [4]. In many cases, problem instances with thousands or tens of thousands of variables need to be solved in milliseconds to achieve real-time performance.

In this paper, we focus on MPC, which places particularly challenging demands on solvers by requiring large problems to be solved robustly and quickly. Model-predictive control (MPC) is a widely used control strategy that solves a receding-horizon optimization problem while reasoning about linear (or linearized) system dynamics and additional linear equality and inequality constraints. Successful applications of MPC include autonomous driving [5], rocket landing [6], and legged locomotion [7]. MPC problems frequently include hundreds to tens of thousands of variables, with solver time complexity scaling cubically with state and control dimensions and linearly with the time horizon.

To reduce solve times, it is common practice to use simplified point-mass [6], [8] or centroidal [9], [10] models that limit the controller's ability to reason about system dynamics and state and control constraints, such as joint and torque limits. When whole-body models are used in MPC controllers, the increased computational burden results in longer solve times and lower sample rates, requiring a faster low-level PD controller to compensate [11], and often producing more conservative and less robust performance.

[1] Authors are with the Robotics Institute, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA {arunleob, johnzhang, swamig, ktracy, zacm}@cmu.edu

*These authors contributed equally to this paper

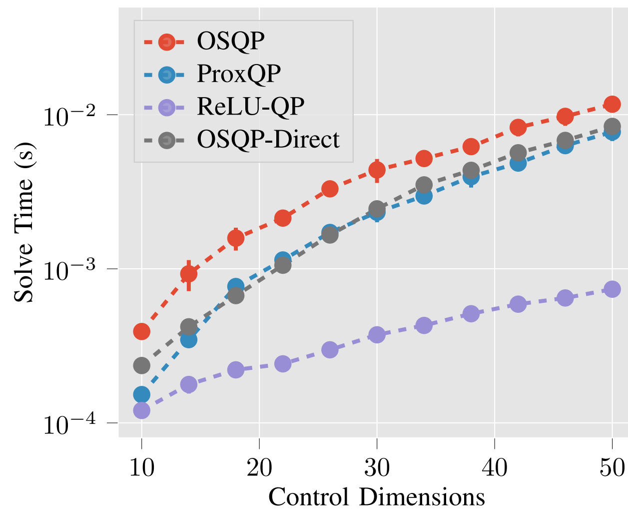

Fig. 1: MPC solve times for random linear systems with control limits (horizon = 40, 3:1 state-to-control dimension ratio). ReLU-QP is an order of magnitude faster on high-dimensional control problems. The preconditioned condensed formulation is used except for *OSQP-Direct*, which solves the sparse direct formulation.

Current methods for solving quadratic programs generally fall into three categories: interior-point methods, active-set methods, and the closely related augmented-Lagrangian method and alternating-direction method of multipliers (ADMM). Interior-point solvers like GUROBI [12], MOSEK [13], and IPOPT [14], offer robust convergence with a few iterations of Newton's method, but each iteration requires the expensive solution of a linear system. Interior-point methods are also difficult to warm start, making them less well suited to MPC applications where similar problem instances are encountered at each time step.

Active-set methods like QuadProg [15] and qpOASES [16] treat active inequality constraints as equalities and remove inactive ones from the problem. They are easy to warm start and can be very fast if the active set is guessed correctly, but their worst-case time complexity is combinatorial in the number of constraints [17].

Augmented-Lagrangian and ADMM methods, including ProxQP [18], OSQP [19], and ALTRO [20], have recently gained popularity in the robotics community for MPC. These solvers are easy to warm start and converge quickly to coarse tolerances, but generally have slow "tail convergence" when trying to achieve tight constraint tolerances.

979-8-3503-8457-4/24/$31.00 ©2024 IEEE

While most existing MPC algorithms run on CPUs, there has been significant recent interest in exploiting the parallel computing capabilities of GPUs to speed up MPC. One such approach is model predictive path integral control (MPPI) [21], [22], which randomly samples control sequences and performs thousands of simulation rollouts in parallel to determine the next control action. While fast, MPPI can struggle to stabilize open-loop unstable systems and suffers from poor sample efficiency, making it difficult to scale to high-dimensional systems.

There have also been recent efforts to parallelize ADMM algorithms. A GPU version of OSQP uses an indirect conjugate-gradient method in place of a direct matrix factorization to exploit fast parallel matrix-vector products [23]. Another parallelized ADMM implementation pre-computes matrix inverses and relies on custom CUDA kernels for computing matrix-vector products on the GPU, but otherwise maintains the conventional sequential structure of ADMM [24].

In this work, we introduce ReLU-QP, a GPU-accelerated quadratic programming solver that achieves speed-ups of an order of magnitude over existing solvers on large-scale problems. ReLU-QP achieves this by analytically mapping the sequential updates of the ADMM algorithm into the structure of a deep, weight-tied neural network with rectified linear unit (ReLU) activations. With this mapping, each iteration of the ADMM algorithm corresponds to a single layer of the network, comprising a dense matrix-vector product followed by a projection onto the positive orthant (clamping negative values to zero).

ReLU-QP can handle large MPC problem instances, enabling the use of detailed whole-body models subject to joint and torque limits, friction cones, and other linear constraints. Despite a large number of constraints and high-dimensional state and input spaces in these problems, ReLU-QP can achieve kilohertz solution rates, allowing us to replace the standard hierarchical control stack — in which an MPC controller is cascaded with low-level PD controllers — with a single, unified, MPC controller that can more fully reason about a robot's dynamics for better performance, safety, and robustness. The simplicity of ReLU-QP also makes it easy to implement and deploy with standard machine-learning toolboxes such as PyTorch, TensorFlow, and JAX.

Our contributions are:

- An analytically exact transformation of the ADMM algorithm into a deep, weight-tied, neural network
- Open-source implementations of the ReLU-QP solver in PyTorch and Julia
- Experimental validation of ReLU-QP on several representative model-predictive control benchmarks with comparisons to state-of-the-art CPU-based solvers

Our paper is organized as follows: In Section II, we review quadratic programs, the ADMM algorithm, and MPC. In Section III, we derive the mapping between ADMM and a deep, weight-tied neural network and present the ReLU-QP solver. In Section IV, we discuss the details of our MPC implementation for use with ReLU-QP. In Section V we benchmark ReLU-QP against OSQP and ProxQP on dense QPs and three MPC applications: random linear problems, the Atlas humanoid balancing on one foot, and a quadruped equipped with a 6-DoF arm performing a whole-body pick-up motion. Finally, we summarize our results, discuss the limitations of our work, and outline future research directions in Section VI.

II. Background

A. Convex Quadratic Programs

Convex quadratic programs are defined as follows [25]:

$$\begin{aligned} \underset{y}{\text{minimize}} \quad & \frac{1}{2} y^T H y + g^T y \\ \text{subject to} \quad & c \leq G y \leq d, \end{aligned} \tag{1}$$

where $g \in \mathbb{R}^n$ and the symmetric positive-definite matrix $H \in \mathbb{R}^{n \times n}$ definite the cost function, $G \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^m$ and $d \in \mathbb{R}^m$ define the constraints, and n and m are the number of decision variables and constraints, respectively.

B. Alternating Direction Method of Multipliers

The Alternating Direction Method of Multipliers (ADMM) is an efficient approach for solving (1) that has been widely deployed in robotics and machine learning in recent years [19]. The ADMM algorithm introduces "splitting variables" $\bar{y}$ and z to transform (1) into

$$\begin{aligned} \underset{y, \bar{y}, z}{\text{minimize}} \quad & \frac{1}{2} \bar{y}^T H \bar{y} + g^T \bar{y} \\ \text{subject to} \quad & G \bar{y} = z, \\ & y = \bar{y}, \\ & c \leq z \leq d. \end{aligned} \tag{2}$$

The augmented Lagrangian for (2) is,

$$\begin{aligned} L_\rho(y, z, \lambda) = & \frac{1}{2} y^T H y + g^T y + \frac{\sigma}{2} ||\bar{y} - y||^2 \\ & + \mu^T (\bar{y} - y) + \frac{1}{2} (G\bar{y} - z)^T \rho (G\bar{y} - z) \\ & + \lambda^T (G\bar{y} - z) + \mathbb{I}_{(c,d)}(z), \end{aligned} \tag{3}$$

where $\sigma \in \mathbb{R}$ is a penalty weight, ρ is a diagonal penalty matrix, μ and λ are Lagrange multipliers, and $\mathbb{I}$ is the indicator function:

$$\mathbb{I}_{(c,d)}(z) = \begin{cases} 0 & c \leq z \leq d \\ \infty & otherwise \end{cases} \tag{4}$$

The diagonal elements in ρ are used to weight progress towards satisfying each constraint.

ADMM alternates minimizing over y and z by performing the following steps [19]:

$$\bar{y}^+ = (H + \sigma I + G^T \rho G)^{-1}(-g + \sigma y - \mu + G^T(\rho z - \lambda)), \tag{5}$$

$$y^+ = \bar{y}^+ + \sigma^{-1}\mu, \tag{6}$$

$$z^+ = \Pi_{(c,d)}(G\bar{y}^+ + \rho^{-1}\lambda), \tag{7}$$

$$\mu^+ = \mu + \sigma(\bar{y}^+ - y^+), \tag{8}$$

$$\lambda^+ = \lambda + \rho(Gy^+ - z^+), \tag{9}$$

where $\Pi_{(c,d)}$ is a projection onto the box constraints $c \leq z \leq d$, (5) is usually computed through a direct matrix factorization or using an indirect method like conjugate gradient, and steps (6)-(9) are composed of simple addition, multiplication, and clamping operations. We note that, from the second iteration onwards, both equations (8) and (6) ensure that $\mu = 0$. Consequently, $\bar{y}$ and y are equivalent, making it possible to eliminate $\bar{y}$ and μ and simplify the overall algorithm, leaving us with y, z, and λ as the essential variables.

1) Convergence Criteria: A natural choice of stopping criterion of the ADMM algorithm is the ∞-norm of the primal and dual residuals of (2) [26]:

$$||Gy - z||_\infty \leq \epsilon_{prim}, \tag{10}$$

$$||Hy + g + G^T\lambda||_\infty \leq \epsilon_{dual}, \tag{11}$$

where $\epsilon_{prim} \geq 0$ and $\epsilon_{dual} \geq 0$ are user-specified accuracy tolerances.

2) Preconditioning: Numerical conditioning plays a key role in the convergence behavior of ADMM [19]. Heuristic preconditioners [27], [28] are often used to reduce condition numbers and increase convergence rate. Variants of Ruiz equilibration [29] are common in modern solvers [18], [19].

3) Adaptive Penalty: The most important parameter in the ADMM algorithm is the parameter ρ [19]. Empirically, convergence rates can be slow when a fixed penalty parameter is used [26]. A common heuristic that works well in practice is to adapt the penalty based on the ratio between the primal and dual residuals [30]. OSQP [19] implements a similar strategy that adjusts ρ based on the relative magnitudes of the residuals:

$$\rho^+ = \rho \sqrt{\frac{||r_p||_\infty \max(||Hy||_\infty, ||G^T\lambda||_\infty, ||g||_\infty, 10^{-4})}{||r_d||_\infty \max(||Gy||_\infty, ||z||_\infty, 10^{-4})}}, \tag{12}$$

where r_p and r_d are the primal and dual residuals from (10) and (11). While adjusting the penalty term online is critical for fast convergence, each update requires a new matrix factorization, which is computationally expensive. Therefore, minimizing overall solution time requires a carefully balanced strategy.

4) Warm Starting: When solving many similar problem instances sequentially, a standard warm-starting strategy for ADMM is to initialize the solver with y, λ, and ρ from the previous solution and set $z = Gy$.

C. Model-Predictive Control

Linear MPC solves the receding-horizon optimal control problem:

$$\begin{aligned} \underset{x_{0:N},\, u_{0:N-1}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \frac{1}{2}(x_k^T Q x_k + u_k^T R u_k) + \frac{1}{2} x_N^T Q_N x_N \\ \text{subject to} \quad & x_{k+1} = A x_k + B u_k \quad \forall k \in 0, \ldots, N-1 \end{aligned} \tag{13}$$

where x_0 is the current state of the system, Q and R are stage cost weights on the states and controls, respectively, Q_N is the terminal state cost weight, A is the discrete state transition matrix, B is the discrete control Jacobian, and N is the MPC horizon length. This optimization problem can be put into the standard form of (1) with the following cost and constraint matrices:

$$y = \begin{bmatrix} u_0 \\ x_1 \\ u_1 \\ \vdots \\ x_N \end{bmatrix}, \quad H = \begin{bmatrix} R & 0 & 0 & \ldots & 0 \\ 0 & Q & 0 & \ldots & 0 \\ 0 & 0 & R & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & Q_N \end{bmatrix},$$

$$G = \begin{bmatrix} B & -I & 0 & 0 & \ldots & 0 \\ 0 & A & B & -I & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \ldots & -I \end{bmatrix}, \quad c = d = \begin{bmatrix} -Ax_0 \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

where G, c, and d encode the dynamics constraints, current state, and potentially additional state and control constraints. In this form, it is also straightforward to add additional constraints on the state and control variables. In the absence of additional constraints, (13) is a Linear-Quadratic Regulator (LQR) problem and can be efficiently solved with a Riccati recursion [31].

We refer to (13) as the "direct" formulation of the MPC problem, as the decision variables include both states and controls. In practice, an MPC controller repeatedly re-solves this problem using the latest state information to produce a feedback policy.

III. The ReLU-QP Algorithm

In this section, we derive a reformulation of the ADMM algorithm as a deep, weight-tied neural network with ReLU activations. This reformulation involves only GPU-friendly operations such as matrix-vector products and projection onto the positive orthant (clamping negative values to zero).

A. Modified ADMM Iterations

We make two key modifications to the ADMM iteration in Section II-B. First, we move the λ update before the y and z updates.

$$\lambda^+ = \lambda + \rho(Gy - z), \tag{14}$$

$$y^+ = (H + \sigma I + G^T\rho G)^{-1}(-g + \sigma y + G^T(\rho z - \lambda^+)), \tag{15}$$

$$z^+ = \Pi_{(c,d)}(Gy^+ + \rho^{-1}\lambda^+). \tag{16}$$

We then combine (14) and (15) into a single matrix equation, followed by the projection (16), where $\tilde{c}$ and $\tilde{d}$ are $(-\infty, +\infty)$ for y and λ:

$$\begin{bmatrix} y^+ \\ z^+ \\ \lambda^+ \end{bmatrix} = \Pi_{(\tilde{c},\tilde{d})}\left(W \begin{bmatrix} y \\ z \\ \lambda \end{bmatrix} + b\right), \tag{17}$$

where

$$W = \begin{bmatrix} D(\sigma I - G^T\rho G) & 2DG^T\rho & -DG^T \\ GD(\sigma I - G^T\rho G) + G & 2GDG^T\rho - I & -GDG^T + \rho^{-1} \\ \rho G & -\rho & I \end{bmatrix},$$

$$D = (H + \sigma I + G^T\rho G)^{-1}, \qquad b = \begin{bmatrix} -D \\ -GD \\ 0 \end{bmatrix} g.$$

Note that, despite some additional computation introduced by reordering the ADMM update steps, the matrix-vector product and projection operations in (17) can be easily parallelized row-wise, and can therefore be efficiently computed on a GPU. In addition, the reordering does not affect the convergence properties of the algorithm. The only effect is on the residuals since λ is now lagged by one iteration, which has negligible practical performance impact.

B. Algorithm Summary

The ReLU-QP algorithm includes both offline and online stages: The offline stage involves pre-computing a set of matrices W in (17) for a predefined set of penalty parameters. In the online stage, we initialize the primal and dual variables and then perform the modified ADMM iterations (17) until convergence criteria (10) and (11) are satisfied, as summarized in Algorithm 1.

Algorithm 1 ReLU-QP

```
Given: ε, ρ_i, check_interval, max_iters
Offline:
Scale problem
Compute KKT inverse, W and b for each ρ ∈ ρ_i
Online:
Update b, c̃, and d̃ with initial conditions
v = [y; z; λ]
for i = 1:max_iters do
    v ← clamp(Wv + b, c̃, d̃)
    if mod(i, check_interval) == 0 then
        W, b, ρ ← rho_update(v)
        if residuals(v) < ε then break
        end if
    end if
end for
return y
```

C. Implementation details

1) Penalty Scaling Heuristics: By default, we define a list of penalty weights ρ_i sampled logarithmically from 10^{-3} to 10^3 and set the initial value to $\rho = 0.1$.

During the solve, we compute a nominal penalty parameter from (12) and use the closest value in the predefined list by choosing the corresponding weight W and bias b.

2) Preconditioning: We use a modified Ruiz equilibration heuristic [29] for ill-conditioned problems. Similar to OSQP, [19], we scale both the constraints and cost to improve the convergence rate.

3) Convergence Checking: The conventional wisdom that ADMM iterations are computationally inexpensive relative to computing convergence criteria (10) and (11) still holds for our solver. We check the convergence criteria every 25 iterations by default.

4) Software Implementation: We implemented the ReLU-QP solver in two popular machine-learning libraries: Flux.jl [32] and PyTorch [33]. Both implementations demonstrate similar performance, especially on large problems where the solve time is dominated by low-level GPU operations on large matrices. Open-source implementations of ReLU-QP can be found at: `https://github.com/RoboticExplorationLab/ReLUQP.jl`.

IV. Dense Model-Predictive Control

We solve a condensed form of the MPC problem (13) to take advantage of existing GPU-accelerated dense matrix operations from standard machine-learning libraries. A common technique for condensing MPC problems is to eliminate the states as variables and only optimize over the controls given an initial condition x_0. This idea is similar to single shooting methods where the solver only optimizes over dynamically feasible trajectories, resulting in fewer decision variables. However, for open-loop unstable systems, condensing the MPC problem in this way also introduces severe numerical ill-conditioning [34].

We find that the modified Ruiz equilibration is often insufficient for addressing ill-conditioning in these problems. Instead, we use an infinite-horizon LQR feedback controller as an effective preconditioner, expressing the controls as $u_k = -Kx_k + \Delta u_k$, where K is the LQR gain matrix and Δu_k are the new decision variables in the MPC problem. This modification has been shown to eliminate the ill-conditioning associated with condensed MPC problems and improve solver convergence [35], [36].

In summary, we define the following transformations: :

$$y = S\tilde{y} + Mx_0, \qquad \bar{A} = A - BK,$$

$$\tilde{y} = \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ \vdots \\ \Delta u_N \end{bmatrix}, \qquad M = \begin{bmatrix} -K \\ \bar{A} \\ \vdots \\ -K\bar{A}^{N-1} \\ \bar{A}^N \end{bmatrix},$$

$$S = \begin{bmatrix} I & 0 & \dots & 0 \\ B & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \\ -K\bar{A}^{N-2}B & -K\bar{A}^{N-3}B & \dots & I \\ \bar{A}^{N-1}B & \bar{A}^{N-2}B & \dots & B \end{bmatrix}.$$

The Hessian, gradient, and constraint matrices of the "direct" form are then modified as follows:

$$\bar{H} = S^T H S, \qquad \bar{g} = S^T H M x_0, \qquad \bar{G} = GS,$$
$$\bar{c} = c - GMx_0, \qquad \bar{d} = d - GMx_0.$$

The rows of G corresponding to the dynamics constraints are eliminated in this formulation as they are included in the new cost Hessian.

V. Simulation Experiments

In this section, we present the results of benchmarking experiments demonstrating ReLU-QP on both random, dense QPs and simulated closed-loop MPC problems, including balancing the Atlas humanoid on one foot and performing a whole-body pick-up motion on a quadruped robot equipped with a six-degree-of-freedom arm.

All experiments were performed on a desktop equipped with an Intel i7-12700K CPU with 64 GB of memory and an NVIDIA GeForce RTX 3080 GPU with 10 GB of VRAM. For OSQP [19] and (dense) ProxQP [18], we report internal solver timings and use the same convergence tolerances across all solvers to ensure fair comparisons. Unless otherwise specified, we solve each problem three times and average the solution times. All benchmarks are run in Julia [37] with the Flux.jl [32] implementation of ReLU-QP.

A. Random Dense QPs

We first benchmark on random dense QPs with both equality and inequality constraints. We generated problems for ten different decision-variable dimensions (n) logarithmically spaced from 10 to 2000. The number of equality and inequality constraints in each problem are both $n/4$. For each problem size, we randomly generate ten dense QPs and solve each problem until the residuals are below 10^{-6}. Note that, except for tolerance settings, default parameters are used for all solvers. Timing results are shown in Fig. 2. Compared to OSQP [19] and ProxQP [18], our method provides competitive solve times on medium-sized problems ($n = 100$) and is up to 30 times faster on large ($n = 2000$) problems.

B. Random Linear MPC Problems

We randomly generate controllable linear systems with control dimensions ranging from 10 to 50, where the state dimensions are always chosen to be triple the control dimensions, by sampling the A and B matrices in (13). We then use MPC to drive each system to the origin under control limits from a random initial state, making sure the initial states are large enough to activate the constraints for a significant portion of the trajectory. For OSQP and ReLU-QP, we perform one solver iteration per MPC step. For ProxQP, we solve to 10^{-2} tolerance, which normally corresponds to 1 outer iteration. All solvers are warm started with the primal and dual solutions from the previous MPC step. Solve times are shown in Fig. 1, where ReLU-QP is an order of magnitude faster on high-dimensional problems.

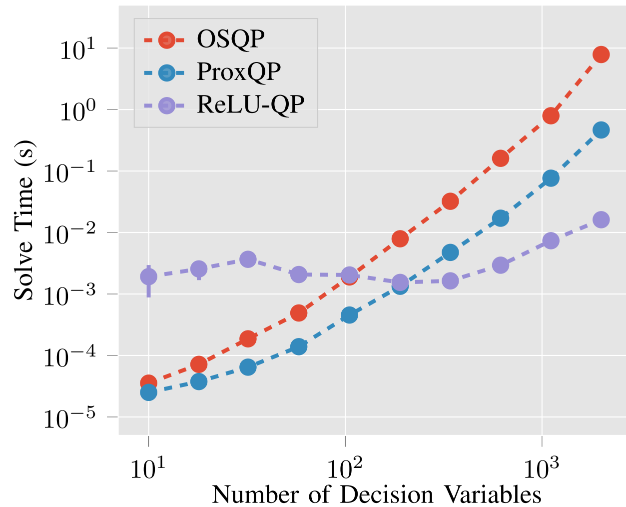

Fig. 2: Solve times for random dense QPs with both equality and inequality constraints demonstrating that ReLU-QP is up to 30 times faster than OSQP and ProxQP on large problems.

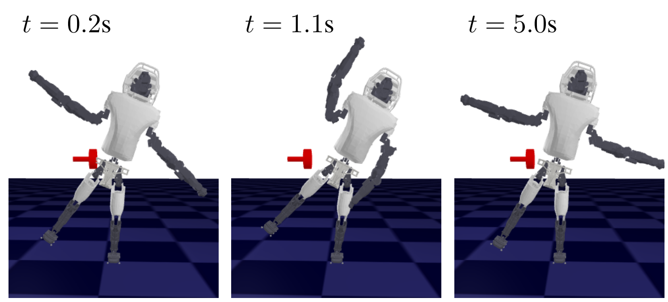

Fig. 3: An Atlas humanoid robot balancing on one foot subject to control limits and disturbances. ReLU-QP successfully solves the task at up to 2600 Hz.

Note that all experiments in Fig. 1 solve the LQR-preconditioned dense MPC formulation, except *OSQP-Direct*, which solves the "direct" MPC formulation (13) using a sparse linear system solver (QDLDL) [19] that efficiently exploits matrix sparsity.

C. Atlas Balance

We use MPC to stabilize a full-body model of the Atlas humanoid robot balancing on one foot subject to initial velocity disturbances and control limits. The model has 58 states and 29 control inputs, and we linearize about a nominal reference pose. We benchmark our solver against OSQP and ProxQP using the preconditioned condensed formulation, with MPC horizons of 0.3s, 0.4s, and 0.5s. We warm start each solver by solving an initial problem instance to 10^{-6} tolerances. After the initial solve, we run OSQP and ReLU-QP for 2 iterations per MPC step and ProxQP to 10^{-4} tolerances – both of which were empirically found to be the coarsest and fastest settings that could successfully stabilize the robot.

Timing results for each horizon length are shown in Table I. ReLU-QP demonstrates speed improvements of up to 20

TABLE I: Average MPC solve frequencies for a whole-body Atlas stabilizing task. ReLU-QP demonstrates 10-20 times speed ups.

MPC Horizon	OSQP	ProxQP	ReLU-QP	speed up
0.3 s	206 Hz	192 Hz	**2599** Hz	**10x**
0.4 s	104 Hz	153 Hz	**1925** Hz	**13x**
0.5 s	65 Hz	66 Hz	**1348** Hz	**20x**

TABLE II: Average MPC solve frequencies for a whole-body quadruped model with an arm picking up an object from the ground. ReLU-QP demonstrates a 2× speed up over OSQP.

MPC Horizon	OSQP	ProxQP	ReLU-QP	speed up
0.3 s	325 Hz	– Hz	**834** Hz	**2.6x**

times and is able to successfully stabilize the non-linear Atlas model at up to 2600 Hz, introducing the possibility of using whole-body MPC on high-dimensional robots without lower-level joint space or impedance controllers.

D. Whole-Body Pick-up Motion on a Quadruped with an Arm

We used MPC to control a quadruped pick-up motion with a 6 DoF arm. The model has 52 states, 20 control inputs, and 12 foot-contact force variables, and was linearized around a standing pose. The problem is shown below, where u contains the controls and foot forces. C is the linearized pinned-foot constraint and $\mathcal{U}$ represents the feasible set for u which includes the pyramidal friction cone, normal force, and torque limit constraints, totaling 52 constraints per knot point in the dense formulation.

$$\begin{aligned} \underset{x_{0:N},\, u_{0:N-1}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \frac{1}{2}(x_k^T Q x_k + u_k^T R u_k) + \frac{1}{2} x_N^T Q_N x_N \\ \text{subject to} \quad & x_{k+1} = A x_k + B u_k \quad \forall k \in 0, \dots, N-1, \\ & C x_k = 0 \qquad \forall k \in 1, \dots, N, \\ & u_k \in \mathcal{U} \qquad \forall k \in 1, \dots, N-1 \end{aligned} \tag{18}$$

We benchmarked ReLU-QP against ProxQP and OSQP, with ReLU-QP and ProxQP solving the preconditioned condensed formulation and OSQP solving the sparse direct formulation. Each solver was warm-started with ReLU-QP and OSQP running for 15 iterations per MPC step, which were empirically found to be the coarsest and fastest settings that could achieve the task. However, even when solving to a tolerance of 10^{-6} ProxQP was unable to stabilize the system. The results of the benchmarks are shown in Table II. ReLU-QP was able to run over two times faster than OSQP with a frequency of 834 Hz, compared to 325 Hz for OSQP.

VI. Discussion and Conclusions

This paper presents ReLU-QP, a GPU-accelerated quadratic programming solver capable of solving medium-to-large problem instances at real-time rates. Our solver excels

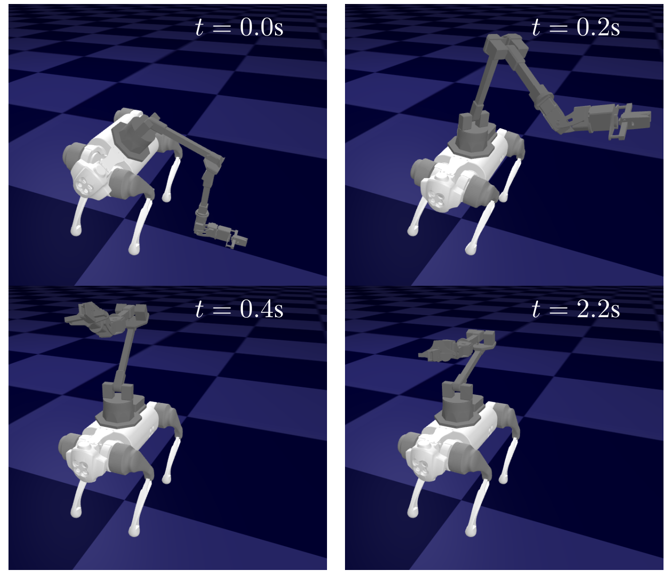

Fig. 4: A Go1 robot equipped with a 6 DoF arm performing a pick-up motion using linear MPC. ReLU-QP solves the problem at 834 Hz.

at quickly solving large, dense QPs to coarse tolerances, and the LQR-preconditioned condensed MPC formulation enables high-performance model-predictive control on complex, high-dimensional systems.

A. Limitations

The primary limitation of our method is the need to pre-compute weight matrices offline, which prevents online updating of dynamics matrices or solving nonlinear problems with sequential quadratic programming. We also note that the speed-ups achieved on the quadruped example were less than on Atlas, which we attribute to the large number of constraints in the problem and ReLU-QP's current inability to leverage matrix sparsity, unlike OSQP.

B. Future Work

Several directions remain for future work: First, we plan to extend ReLU-QP to better exploit problem sparsity. Second, we will investigate indirect linear-system solvers like conjugate gradient to avoid pre-computing inverses offline and enable online updates and nonlinear SQP methods. Finally, we plan to investigate the possibility of initializing neural network control policies using ReLU-QP before performing additional policy tuning using reinforcement-learning techniques.

References

[1] K. Shankar, J. W. Burdick, and N. H. Hudson, “A Quadratic Programming Approach to Quasi-Static Whole-Body Manipulation,” en, in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, ser. Springer Tracts in Advanced Robotics, H. L. Akin, N. M. Amato, V. Isler,

and A. F. van der Stappen, Eds., Cham: Springer International Publishing, 2015, pp. 553–570.

[2] S. Redon, A. Kheddar, and S. Coquillart, "Gauss' least constraints principle and rigid body simulations," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1, Washington, DC, United States, 2002, pp. 517–522.

[3] A. Spielberg, B. Araki, C. Sung, R. Tedrake, and D. Rus, "Functional co-optimization of articulated robots," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 5035–5042.

[4] B. Kouvaritakis and M. Cannon, *Model Predictive Control* (Advanced Textbooks in Control and Signal Processing). Cham: Springer International Publishing, 2016.

[5] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive Active Steering Control for Autonomous Vehicle Systems," *IEEE Transactions on Control Systems Technology*, vol. 15, no. 3, pp. 566–580, May 2007, Conference Name: IEEE Transactions on Control Systems Technology.

[6] B. Açıkmeşe, J. M. Carson, and L. Blackmore, "Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2104–2113, Nov. 2013, Conference Name: IEEE Transactions on Control Systems Technology.

[7] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ISSN: 2153-0866, Oct. 2018, pp. 1–9.

[8] B. Acikmese and S. R. Ploen, "Convex programming approach to powered descent guidance for mars landing," *Journal of Guidance, Control, and Dynamics*, vol. 30, no. 5, pp. 1353–1366, 2007, Publisher: American Institute of Aeronautics and Astronautics _eprint: https://doi.org/10.2514/1.27553.

[9] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 2245–2252.

[10] S. Kuindersma, R. Deits, M. Fallon, *et al.*, "Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot," *Autonomous Robots*, vol. 40, no. 3, pp. 429–455, Mar. 1, 2016.

[11] M. Neunert, M. Stäuble, M. Giftthaler, *et al.*, "Whole-Body Nonlinear Model Predictive Control Through Contacts for Quadrupeds," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1458–1465, Jul. 2018, Conference Name: IEEE Robotics and Automation Letters.

[12] Gurobi Optimization, LLC, *Gurobi Optimizer Reference Manual*, 2023.

[13] M. ApS, "MOSEK Optimization Toolbox for MATLAB," en,

[14] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, Mar. 2006, Publisher: Springer Nature.

[15] D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," en, *Mathematical Programming*, vol. 27, no. 1, pp. 1–33, Sep. 1983.

[16] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," en, *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, Dec. 2014.

[17] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2e. New York, NY, USA: Springer, 2006.

[18] A. Bambade, S. El-Kazdadi, A. Taylor, and J. Carpentier, "PROX-QP: Yet another Quadratic Programming Solver for Robotics and beyond," en, Jun. 2022.

[19] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," en, *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, Dec. 2020.

[20] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A Fast Solver for Constrained Trajectory Optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ISSN: 2153-0866, Nov. 2019, pp. 7674–7679.

[21] G. Williams, N. Wagener, B. Goldfain, *et al.*, "Information theoretic MPC for model-based reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 1714–1721.

[22] J. Pravitra, K. A. Ackerman, C. Cao, N. Hovakimyan, and E. A. Theodorou, "L1-Adaptive MPPI Architecture for Robust and Agile Control of Multirotors," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, ISSN: 2153-0866, Oct. 2020, pp. 7661–7666.

[23] M. Schubiger, G. Banjac, and J. Lygeros, "GPU acceleration of ADMM for large-scale quadratic programming," en, *Journal of Parallel and Distributed Computing*, 2020.

[24] L. Yu, A. Goldsmith, and S. Di Cairano, "Efficient Convex Optimization on GPUs for Embedded Model Predictive Control," en, in *Proceedings of the General Purpose GPUs*, Austin TX USA: ACM, Feb. 2017, pp. 12–21.

[25] J. Nocedal and S. J. Wright, "Quadratic Programming," en, in *Numerical Optimization*, ser. Springer

Series in Operations Research and Financial Engineering, New York, NY: Springer, 2006, pp. 448–492.
[26] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," English, *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, Jul. 2011, Publisher: Now Publishers, Inc.
[27] P. Giselsson and S. Boyd, "Metric selection in fast dual forward–backward splitting," *Automatica*, vol. 62, pp. 1–10, Dec. 2015.
[28] P. Giselsson and S. Boyd, "Linear Convergence and Metric Selection for Douglas-Rachford Splitting and ADMM," *IEEE Transactions on Automatic Control*, vol. 62, no. 2, pp. 532–544, Feb. 2017, Conference Name: IEEE Transactions on Automatic Control.
[29] D. Ruiz, "A scaling algorithm to equilibrate both rows and columns norms in matrices," Jan. 2001.
[30] B. S. He, H. Yang, and S. L. Wang, "Alternating Direction Method with Self-Adaptive Penalty Parameters for Monotone Variational Inequalities," en, *Journal of Optimization Theory and Applications*, vol. 106, no. 2, pp. 337–356, Aug. 2000.
[31] P. Scokaert and J. Rawlings, "Constrained linear quadratic regulation," *IEEE Transactions on Automatic Control*, vol. 43, no. 8, pp. 1163–1169, Aug. 1998, Conference Name: IEEE Transactions on Automatic Control.
[32] M. Innes, "Flux: Elegant machine learning with Julia," en, *Journal of Open Source Software*, vol. 3, no. 25, p. 602, May 2018.
[33] A. Paszke, S. Gross, F. Massa, *et al.*, *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, Pages: 8024–8035 Publication Title: Advances in Neural Information Processing Systems 32 original-date: 2016-08-13T05:26:41Z, 2019.
[34] R. Tedrake, *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832)*, 2019.
[35] J. A. Rossiter, B. Kouvaritakis, and M. J. Rice, "A numerically robust state-space approach to stable-predictive control strategies," *Automatica*, vol. 34, no. 1, pp. 65–73, Jan. 1998.
[36] I. McInerney, E. C. Kerrigan, and G. A. Constantinides, "Horizon-Independent Preconditioner Design for Linear Predictive Control," *IEEE Transactions on Automatic Control*, vol. 68, no. 1, pp. 580–587, Jan. 2023, Conference Name: IEEE Transactions on Automatic Control.
[37] T. Koolen and R. Deits, "Julia for robotics: Simulation and real-time control in a high-level programming language," May 2019.