Optimizing at All Scales: Edge (Non)linear Model Predictive Control from MCUs to GPUs

Carnegie Mellon University Robotics Institute

REXLAB

Emre Adabag[1*], Xuyei Bu[1*], Khai Nguyen[2*], Sam Schoedel[3*], Anoushka Alavilli[4], Miloni Atal[1], William Gerard[1], Elakhya Nedurmaran[4], Zac Manchester[3], and Brian Plancher[5]

1: Fu Foundaiton School of Engineering and Applied Science, 5: Barnard College, Columbia University

2: Mechanical Engineering, 3: Robotics Institute, 4: Electrical & Computer Engineering, Carnegie Mellon University

A²R The Barnard Accessible and Accelerated Robotics Lab

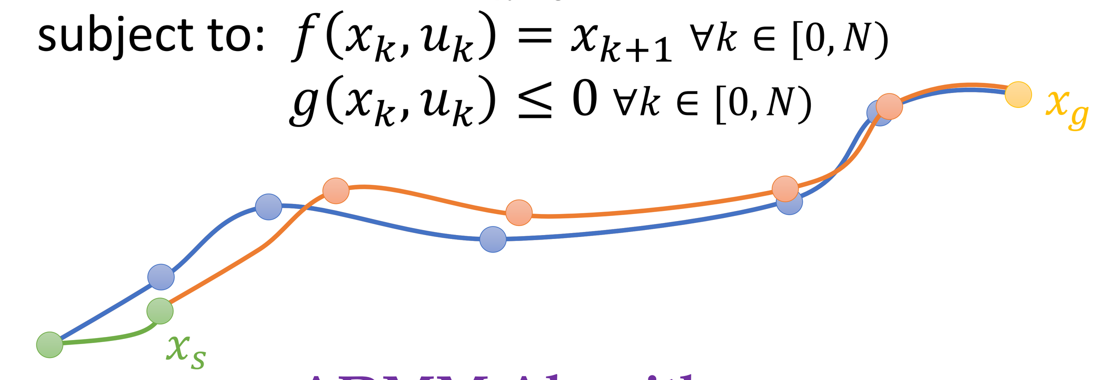
The Big Picture:

In our recent works, by leveraging a combination of parallelism, approximation, and structure exploitation, we have **enabled and accelerated (nonlinear) trajectory optimization solvers for real-time performance on non-standard computational hardware**, ranging from microcontrollers (MCUs) to graphical processing units (GPUs). This has led to real-time **MPC onboard an MCU powered 27g quadrotor** for dynamic obstacle avoidance, as well as simulated **whole-body nonlinear MPC at kHz rates for a GPU powered manipulator** for high speed trajectory tracking.

$$\min_{X,U} \; l_f(x_N) + \sum_{k=0}^{N-1} l(x_k, u_k)$$

subject to: $f(x_k, u_k) = x_{k+1} \; \forall k \in [0, N)$

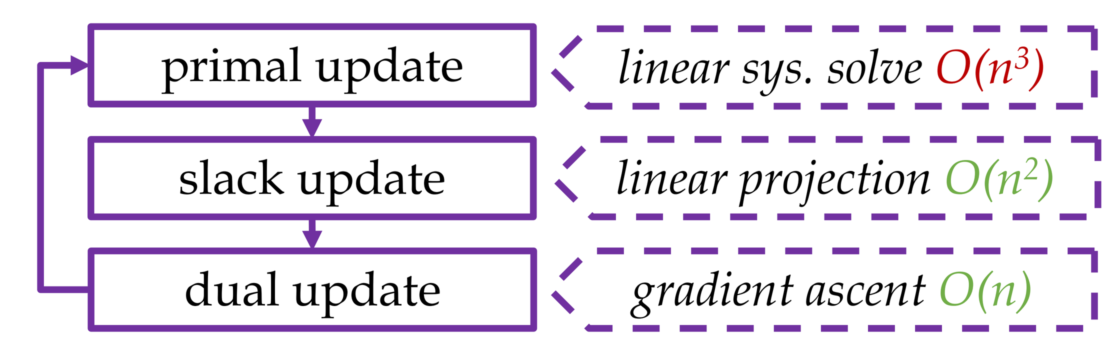
$g(x_k, u_k) \leq 0 \; \forall k \in [0, N)$



Model Predictive Control Background:

In most Model Predictive Control (MPC) formulations, a **trajectory optimization problem is used to re-optimize the robot's trajectory at each control step**. These problems computes a robot's optimal path through an environment as a series of states, x, and controls, u, by minimizing a cost function, $l(\cdot)$, subject to discrete time dynamics, $f(\cdot)$, and additional constraints, $g(\cdot)$, (e.g., obstacle avoidance, torque limits). The **alternating direction method of multipliers (ADMM)** algorithm can be used to solve such problems by breaking the problem into a **three step process** where slack variables are added to decouple the solution of additional constraints $g(\cdot)$, from the base dynamics constrained problem over $l(\cdot)$ and $f(\cdot)$. In this work **we explore two separate approaches to accelerate and compress the solution of the rate-limiting primal update.**

ADMM Algorithm



MPCGPU: Real-Time Nonlinear Model Predictive Control through Symmetric Stair Preconditioned Conjugate Gradient on the GPU:

One key computation for direct trajectory optimization problems is the repeated solving of the resulting Karush-Kuhn-Tucker linear system, which can be solved using the **symmetric positive definite and block tridiagonal *Schur Complement*, S**. Iterative linear system solves can accelerate such problems on parallel processors. However, these methods **require a preconditioner, $\Phi^{-1} \approx S^{-1}$**, as their convergence properties are related to the clustering and magnitude of the eigenvalues of $\Phi^{-1}S$. MPCGPU solves the NMPC problem through a three-step process, leveraging the symmetric-stair preconditioner for improved performance:

1) On the GPU it **computes S, γ, and Φ^{-1} in parallel**,
2) Uses the **GPU-Accelerated Block-Diagonal PCG algorithm (GBD-PCG) to compute λ^*** and reconstruct δX*, δU* efficiently by re-factoring the PCG algorithm,
3) Uses a **parallel line search** to form the final trajectory.

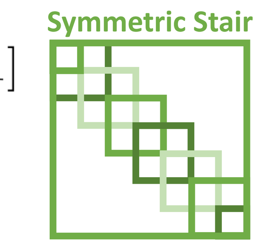
This trajectory is passed to the (simulated) robot and the current state of the (simulated) robot is measured and fed back into our solver which is run again, **warm-started with our last solution**.

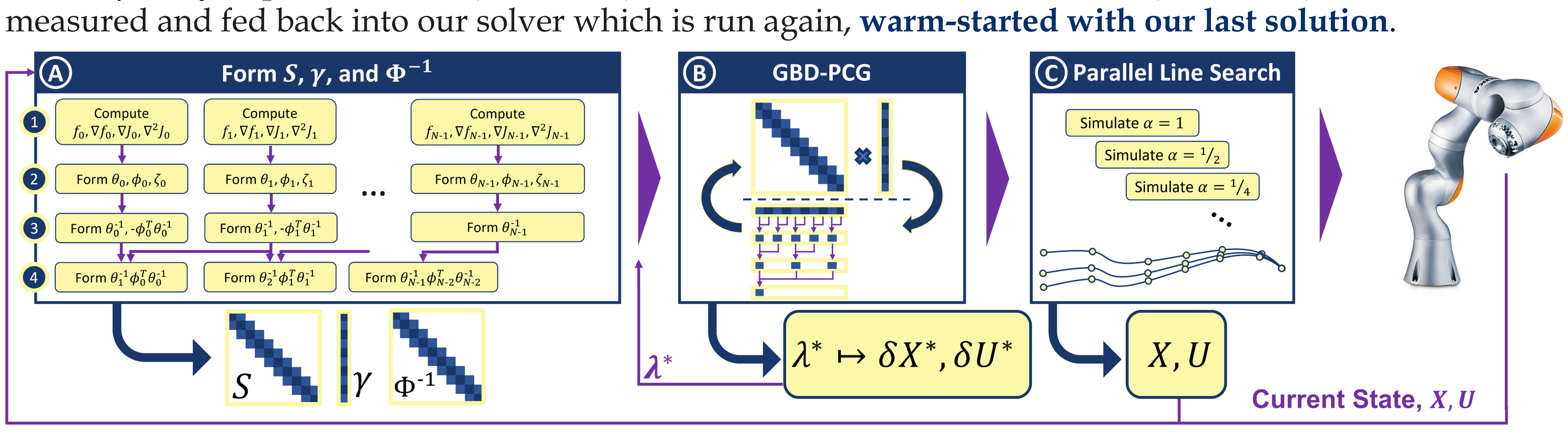
$$S = \begin{bmatrix} D_1 & O_1 & 0 \\ O_1^T & D_2 & O_2 \\ 0 & O_2^T & D_3 \end{bmatrix} \Rightarrow \Psi_l = \begin{bmatrix} D_1 & 0 & 0 \\ O_1^T & D_2 & O_2 \\ 0 & 0 & D_3 \end{bmatrix}$$

$$\Psi_l^{-1} = \begin{bmatrix} D_1^{-1} & 0 & 0 \\ -D_2^{-1}O_1^T D_1^{-1} & D_2^{-1} & -D_2^{-1}O_2 D_3^{-1} \\ 0 & 0 & D_3^{-1} \end{bmatrix}$$

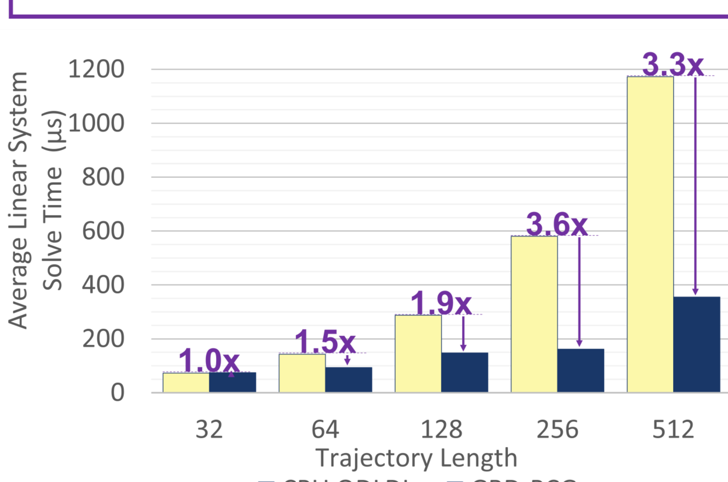
$$\Phi_{sym}^{-1} = \Psi_l^{-1} + \Psi_r^{-1} - D^{-1}$$

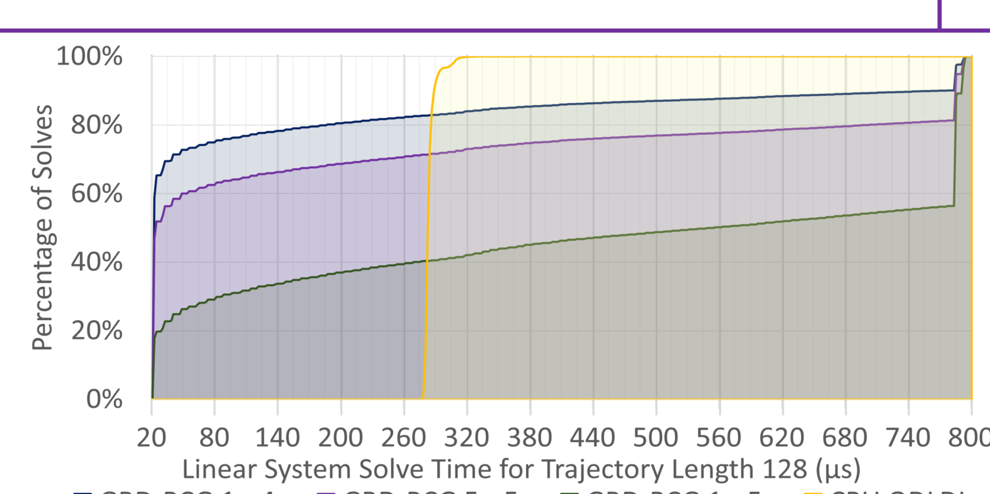
$$\lambda(\Phi_{sym}^{-1} S) \in (0, 1]$$





- GBD-PCG's advantage **scales with problem size, with up to a 3.6x average speedup** over QDLDL on the CPU.
- GBD-PCG, under multiple different exit tolerances, ϵ, exhibits a **bi-modal solve time distribution** which is usually much faster than the uni-modal distribution for QDLDL on the CPU. E.g., for ϵ = 1e^{-4}:
  - **>65% of GBD-PCG solves are ≥10x faster** than the fastest QDLDL solve,
  - **<10% of GBD-PCG solves are ≥2x slower**, and the slowest is 2.5x slower, than the slowest QDLDL solve.





MPCGPU with GBD-PCG		Knot Points				
		32	64	128	256	512
Control Rate	250Hz	22.2	19.7	15.4	5.2	4.4
	500Hz	10.3	10.6	8.0	4.6	3.0
	1kHz	4.9	5.2	3.7	2.4	1.7

- Our **GPU-first approach** enables MPCGPU to scale **to 512 knot points at 1kHz** and execute 8 iterations for **128 knot points** at 500Hz, for a per-iteration rate of **4kHz**.

TinyMPC: Conic Model-Predictive Control on Resource-Constrained Microcontrollers through Code Generation

TinyMPC **trades generality for speed and low-memory utilization** to enable real-time use on MCUs by exploiting the structure of the MPC problem. Specifically, we leverage the **closed-form Riccati solution to the LQR problem to compute the primal update** by leveraging a single linearization of the system dynamics, $f(\cdot)$, and solving the infinite horizon LQR problem offline. This enables us to cache bottleneck computations and **avoid any matrix inversions or divisions online**. The ADMM framework also enables us to support **both linear and conic inequality constraints for $g(\cdot)$ through a simple projection.**

LQR

$$K_k = (R + B^\intercal P_{k+1} B)^{-1}(B^\intercal P_{k+1} A) \longrightarrow K_\infty$$

$$d_k = (R + B^\intercal P_{k+1} B)^{-1}(B^\intercal p_{k+1} + r_k)$$

$$P_k = Q + K_k^\intercal R K_k + (A - BK_k)^\intercal P_{k+1}(A - BK_k) \longrightarrow P_\infty$$

$$p_k = q_k + (A - BK_k)^\intercal(p_{k+1} - P_{k+1}Bd_k) + K_k^\intercal(Rd_k - r_k)$$

Offline vs. Online

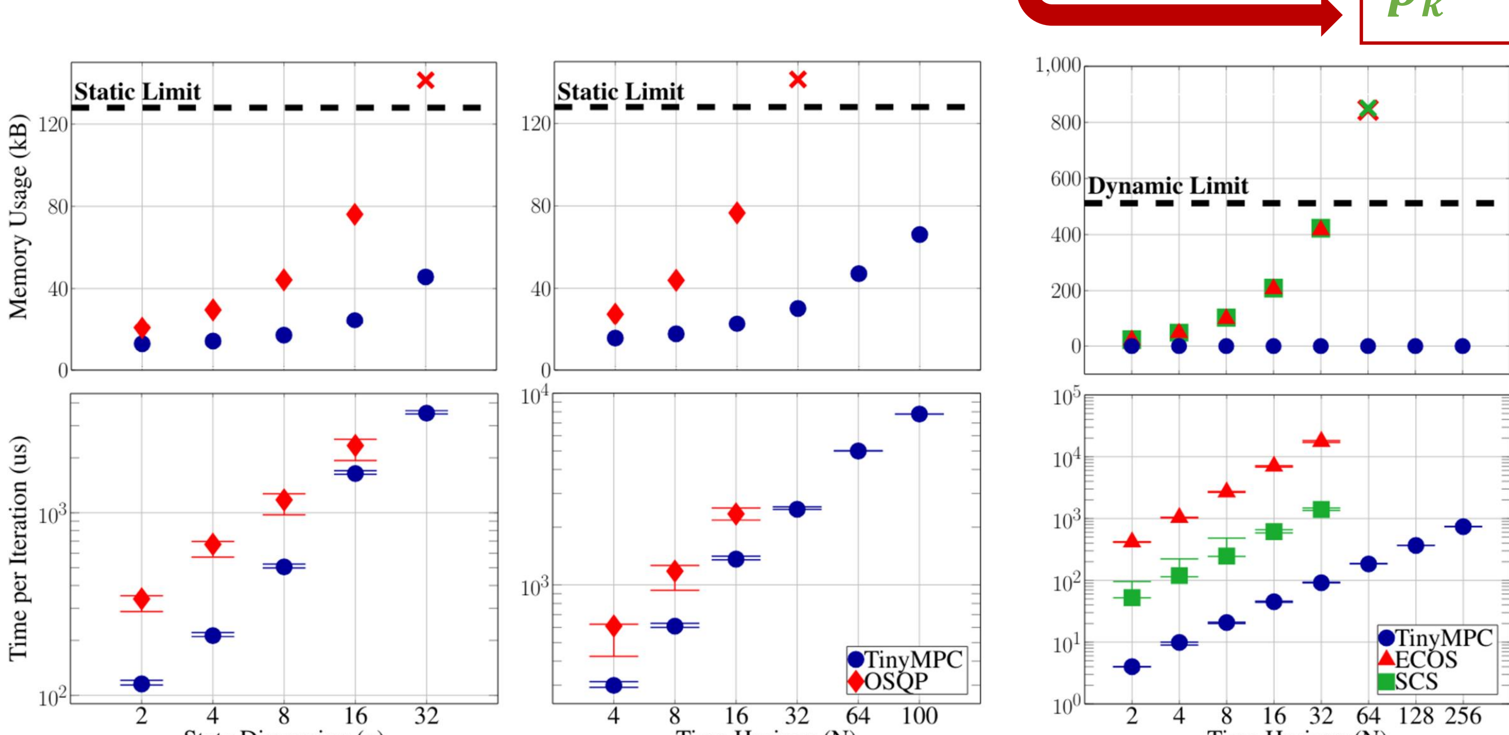
$$C_1 = \left(R + B^T P_\infty B\right)^{-1}$$

$$C_2 = (A - BK_\infty)^T$$

$$d_k = C_1(B^T p_{k+1} + r_k)$$

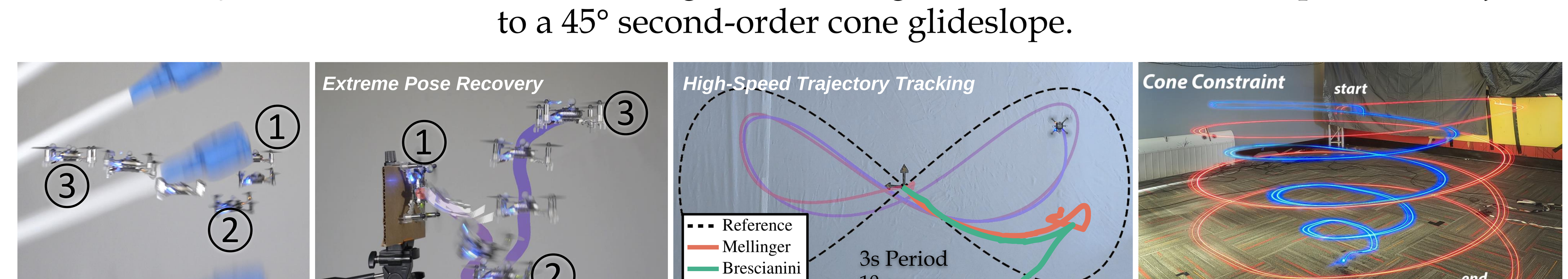
$$p_k = q_k + C_2 p_{k+1} - K_\infty^T r_k$$

- On a 168 MHz STM32F405 with 1 MB of Flash and 128 kB of RAM we find that **TinyMPC scales better than OSQP** across both state dimension and time horizon, **using far less memory for faster iterations**.
- On a 600 MHz Teensy 4.1 with 7.75 MB of flash and 512 kB of tightly coupled RAM, **TinyMPC again scales better than ECOS and SCS across both memory usage and iteration speed**.
- In both settings, **unlike the other solvers, TinyMPC fits inside the resource limits of embedded hardware**. These computational results **enable real-time optimal control onboard tiny robots** like the Crazyflie 2.1, a 27 gram nano-quadrotor. Examples include:
  - Real-time dynamic obstacle avoidance,
  - Recovery from a 90° attitude error,
  - High-speed figure-8 trajectory tracking,
  - And tracking a descending helical reference with its position subject to a 45° second-order cone glideslope.



Predictive safety filtering

Rocket soft-landing



This material is based upon work supported by the National Science Foundation (under Award 2246022). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding organizations.

*These authors contributed equally to this work.