Hardware-Aware Co-Design for High-Frequency Visual MPC on Edge SoCs

Prithvi Raj Singh[1]

***Abstract*— Numerical MPC solvers can operate at kilohertz frequencies on modern edge hardware, yet integrating high-latency visual perception collapses end-to-end throughput to sub-10 Hz – a disparity we term the *Sensing-to-Actuation Gap*. We characterize this gap on two representative heterogeneous SoCs: the Raspberry Pi 4B (quad-core ARM Cortex-A72) and the NVIDIA Jetson TX2 (asymmetric Denver 2 + ARM A57 + Pascal GPU). Profiling reveals a *Data-Age Ratio* of up to 115:1 on the Pi and 86:1 on the Jetson, meaning the controller repeatedly acts on stale visual state estimates. We propose a hardware-aware asynchronous multirate architecture that exploits each platform's heterogeneous core clusters: the timing-critical MPC solver is pinned to high-performance cores while perception runs independently on auxiliary compute. This decoupling recovers solver frequencies of $>$1 kHz while the vision pipeline operates at its own natural cadence. Scaling experiments show the Jetson maintains the 100 Hz real-time threshold up to state-space dimension n=300, compared to n<200 on the Pi. Our results highlight that perception throughput-not solver arithmetic-is now the primary bottleneck in visual MPC, and that heterogeneous core allocation is a critical design lever.**

I. Introduction

High-speed autonomous robotics requires tight coupling between perception and control. Advances in embedded convex optimization–illustrated by TinyMPC [1], which exploits ADMM sparsity to run MPC at KHz rates on microcontrollers–have largely solved the *solver* bottleneck. However, integrating visual state estimation with these high-speed solvers on resource-constrained edge SoCs introduces a new system-level bottleneck: the camera and its processing pipeline operate orders of magnitude slower than the control loop.

On a baseline serial stack running on the Raspberry Pi 4B, we measure an end-to-end visual MPC frequency of only 9.6 Hz, despite the MPC solver achieving 1,073 Hz in isolation. This $\approx$100$\times$ disparity-the Sensing-to-Actuation Gap-means the control law is repeatedly computed from a state estimate that is many control cycles stale. For agile platforms, such as quadrotors or ground vehicles, this staleness can cause divergence.

Prior work on hardware-software co-design for robotics (e.g., Robomorphic Computing [2]) has focused on accelerating the solver kernel itself, often assuming zero-latency state feedback. While advances like EMPC [3] exploits edge network heterogeneity and MPCGPU [4] leverages parallelism on GPU to achieve KHz-rate nonlinear trajectory optimization, these frameworks do not address the within-SoC mismatch between perception and control timescales. We extend the co-design principle to the *system integration* layer, asking: how should a heterogeneous SoC's asymmetric core clusters be allocated to minimize the perception-control gap?

Contributions. (1) A quantitative characterization of the *Sensing-to-Actuation* Gap on two representative edge SoCs, introducing the *Data-Age Ratio* (DAR) as a performance metric. (2) A hardware-aware asynchronous multirate architecture that decouples the perception and control threads via core-affinity pinning. (3) Scaling analysis identifying the "computational ceiling" for each platform as a function of state-space dimension.

II. Problem Formulation

We consider a discrete-time linear system $\mathbf{x}_{k+1} = A\mathbf{x}_k + B\mathbf{u}_k$, where $x_k \in \mathbb{R}^n$ is the state and $u_k \in \mathbb{R}^m$ is the control input, controlled by solving the finite-horizon quadratic program characterized by equation 1

$$\min_{\mathbf{u}_{0:N-1}} \sum_{i=0}^{N-1} \left(\mathbf{x}_i^\top Q\mathbf{x}_i + \mathbf{u}_i^\top R\mathbf{u}_i\right) + \mathbf{x}_N^\top P\mathbf{x}_N, \qquad (1)$$

subject to $\mathbf{x} \in \mathcal{X}$, $\mathbf{u} \in \mathcal{U}$. Total system latency decomposes as $\tau_{\text{total}} = \tau_p + \tau_c + \tau_s$, where τ_p, τ_c, and τ_s denote perception, communication, and solver latency, respectively. The control input u_k at time k is computed using a state estimate $\hat{x}$ that corresponds to the system state at time $t - \tau_p$. In our async architecture, the control law is given by $u_k = MPC(\hat{x}_k - R)$, where R is the DAR.

Data-Age Ratio (DAR). Since $\tau_p \gg \tau_s$ on all tested platforms, the controller at time k uses a state estimate $\hat{\mathbf{x}}$ that is $R = \lfloor \tau_p/\tau_s \rfloor$ control cycles old. We call R the *Data-Age Ratio*; high R implies the robot is acting "blind" for many cycles between visual updates, which degrades closed-loop stability for dynamic tasks.

III. Hardware-Aware Asynchronous Architecture

Asynchronous multirate decoupling. In a regular serial implementation, the MPC thread blocks on each new vision frame, limiting throughput to the camera pipeline rate. We replace this with a *shared-memory double-buffer*: the vision thread writes the latest state estimate asynchronously, while the MPC thread polls the buffer at its own maximum rate without blocking. This single change transforms the 9.6 Hz serial stack into a 1,043 Hz control loop on the Pi 4B and 999 Hz on the Jetson TX2.

Core-affinity pinning on the Jetson TX2. Naive multithreading on the Jetson causes resource contention: the perception and solver threads compete for shared L2 cache and memory-bus bandwidth, degrading both. Following the

[1]Author is with the Department of Engineering & Computer Science Email: psingh8@mcneese.edu

TABLE I: Preliminary Experimental Results

Table I: Platform Comparison (n=100)

Metric	RPi 4B	Jetson TX2
MPC Freq. (Hz)	1,073.4	998.9
MPC Latency (ms)	0.93	1.00
Vision Throughput (FPS)	9.30	11.60
Vision Latency (ms)	106.2	86.2
Data-Age Ratio	115:1	86:1

Table II: Jetson TX2 Scaling

n	Latency (ms)	Freq. (Hz)
100	1.59	627.4
200	3.18	314.3
300	8.87	**112.8**
400	12.20	82.0
600	24.07	41.6

Table III: RPi 4B Scaling

n	Latency (ms)	Freq. (Hz)
100	2.94	340.2
200	9.44	**106.0**
300	20.07	49.8
400	46.00	21.7
600	144.00	6.9

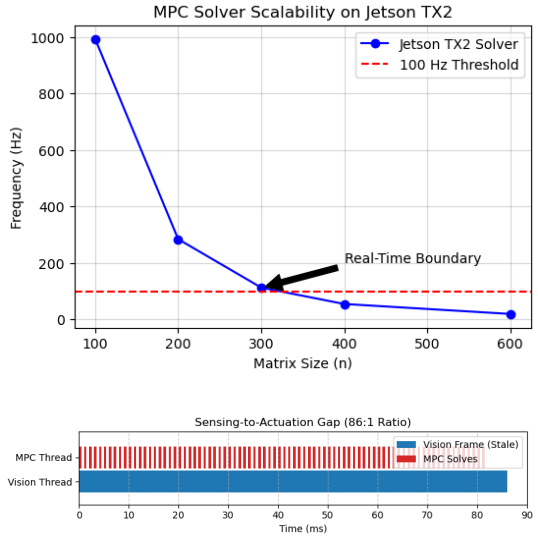

Fig. 1: **(Top)** MPC solver frequency vs. state dimension n on the Jetson TX2, showing the 100 Hz real-time boundary. **(Bottom)** Thread timeline (86:1 DAR on Jetson TX2): MPC completes $\sim$86 solves per stale vision frame. Note: figures from current profiling; TensorRT acceleration is future work.

paradigm of co-design solvers for target hardware architectures [5], we mitigate the resource contention problem by explicitly partitioning the TX2's asymmetric clusters pinning the time-critical MPC solver to the high-performance Denver 2 cores while offloading the vision pipeline to the ARM A57 cluster. This task-isolation strategy ensures that vision-pipeline jitter does not perturb the deterministic 999 Hz control loop. On the Pi 4B (homogeneous quad-core Cortex-A72), we instead apply thread-priority scheduling (SCHED_FIFO) to minimize preemption of the solver by the vision thread.

Scope and limitations. The current implementation uses a baseline Python/OpenCV vision pipeline (no TensorRT or NEON-vectorized inference) to isolate and characterize the architectural effects. Full GPU-accelerated perception via TensorRT is planned as future work (Section V).

IV. Experimental Evaluation

Platforms. Raspberry Pi 4B: quad-core ARM Cortex-A72 @ 1.8 GHz, 8 GB LPDDR4. NVIDIA Jetson TX2: Denver 2 dual-core + ARM A57 quad-core, 256-core Pascal GPU, 8 GB LPDDR4. MPC implemented using ADMM-based linear QP (following TinyMPC [1]); vision pipeline uses a standard OpenCV detection backbone.

Platform comparison (Table I). With the asynchronous architecture enabled, both platforms achieve >1 kHz MPC throughput. The Jetson TX2 reduces vision latency by 18 ms (18%) and nearly 25% frame throughput (9.3 to 11.60 FPS), narrowing the DAR from 115:1 to 86:1. Crucially, neither CPU cluster achieves real-time vision (>30 FPS); this motivates future GPU-accelerated perception (Section V).

Complexity scaling (Tables II and III). We sweep state dimension n from 100 to 600 to identify each platform's "computational ceiling." The Jetson TX2 maintains the 100 Hz real-time threshold through n=300 (8.87 ms, 112.75 Hz); the Pi 4B fails at n=200 (9.44 ms, 105.96 Hz) and reaches 6.94 Hz at n=600. The Jetson's Denver 2 advantage in out-of-order execution yields a 3$\times$ latency improvement at n=300 (8.87 vs. 20.07 ms) and a widening gap at higher dimensions. This shows that asymmetric cores handle complex math significantly better.

V. Discussion and Future Work

Our results seeks to establish a key insight for the robotics systems community: on heterogeneous edge SoCs today, *numerical optimization is no longer the bottleneck for visual MPC–perception is*. Even on the Jetson TX2 with its Pascal GPU idle for perception, the best achievable DAR is 86:1, meaning the robot acts on millisecond-stale information for the vast majority of control cycles.

This motivates two complementary directions. First, **GPU-accelerated perception**: offloading the vision backbone to the TX2's Pascal GPU via TensorRT is expected to reduce vision latency by 5–10$\times$, pushing FPS toward 60–100. This direction builds upon the benefits of high-throughput GPU optimization established in [4], but pivots the accelerator's role towards perception to reduce the data-age ratio (DAR) below 10:1. Second, **predictive state estimation**: using the spare Denver 2 capacity to run a lightweight Kalman or IMU-propagation filter between visual updates can partially mitigate staleness without faster cameras. Together, these form a full hardware-software co-design loop: solver $\rightarrow$ core-affinity $\rightarrow$ GPU perception $\rightarrow$ predictive filter.

This work-in-progress is a systems-level characterization study that exposes a *new bottleneck profile* for visual MPC on commodity SoCs – one that the RoboARCH community is well-positioned to address through accelerator design, compiler pipelines (e.g., Apache TVM [6]), and co-designed scheduling.

REFERENCES

[1] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester, "Tinympc: Model-predictive control on resource-constrained microcontrollers," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024.

[2] S. M. Neuman, B. Plancher, T. Bourgeat, T. Tambe, S. Devadas, and V. J. Reddi, "Robomorphic computing: A design methodology for domain-specific accelerators parameterized by robot morphology," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2021, pp. 674–686.

[3] Y.-Y. Lou, J. Spencer, K. T. Kim, and M. Chiang, "E-mpc: Edge-assisted model predictive control," *arXiv preprint arXiv:2410.00695*, 2024.

[4] E. Adabag, X. Bu, K. Nguyen, S. Schoedel, and B. Plancher, "Mpcgpu: Real-time nonlinear model predictive control through parallelism on gpus," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2024. [Online]. Available: https://a2r-lab.org/publication/mpcgpu/

[5] B. Plancher, E. Adabag, X. Bu, K. Nguyen, S. Schoedel, A. Alavilli, M. D. Atal, W. Gerard, E. Nedumaran, and Z. Manchester, "Optimizing at all scales: Edge (non)linear model predictive control from mcus to gpus," in *Frontiers of Optimization for Robotics Workshop at Robotics: Science and Systems (RSS)*, 2024. [Online]. Available: https://brianplancher.com/publication/optimizingatallscales/

[6] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: An automated end-to-end optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.