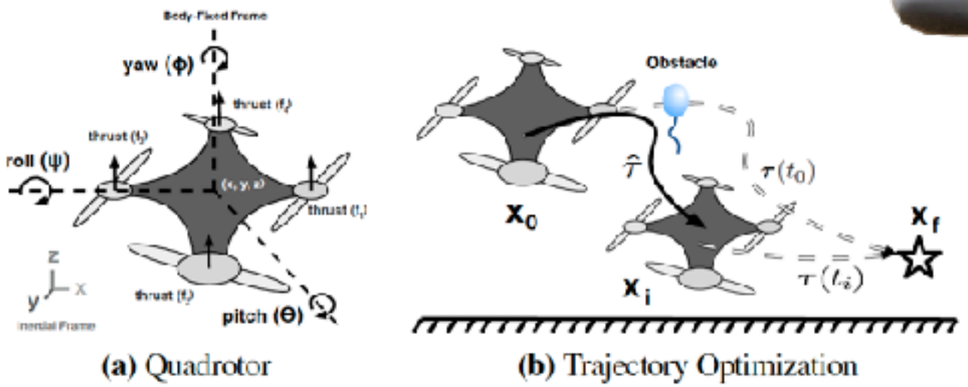
RoboX: An End-to-End Solution to Accelerate Autonomous Control in Robotics

Jacob Sacks, Divya Mahajan Richard, C. Lawson
and Hadi Esmaeilzadeh
Georgia Institute of Technology and University of
California, San Diego

Why?

Unmanned Aerial Vehicles

- 131.4W for the motor
- Total power budget of all other things — 5.6W



(a) Quadrotor

(b) Trajectory Optimization

Figure 1: (a) States and control inputs of a quadrotor. (b) Quadrotor planning trajectory $\tau(t_0)$ towards target (star) and adjusting course to $\hat{\tau}$ due to obstacle (balloon), while staying above altitude $\underline{z}$ due to a constraint.

Why RoboX

- General-purpose processor is not very energy-efficient
- Compute power can be more significant as actuators keep evolving

Recap: What can we do?

- Holistic/cross-layered system design instead of single-point/ local optimizations.
- Distributed computing instead of centralized computing.
- Massive, wimpy processing units instead of single, powerful processing unit.
- Domain-specific design instead of general-purpose architectures.

The opportunity for an accelerator

- The common part in their algorithms — using Cholesky decomposition and forward/backward substitution to solve the following linear system — This process is repeated until convergence, after which the control decisions are supplied to the quadrotor.

$$\begin{bmatrix} H & g_{\Delta z}^T & h_{\Delta z}^T & \\ g_{\Delta z} & & & \\ h_{\Delta z} & & I & \\ & & S & \Lambda \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta \mu \\ \Delta \lambda \\ \Delta s \end{bmatrix} = - \begin{bmatrix} J_{\Delta z} \\ g \\ h \\ S\Lambda \end{bmatrix} \qquad (6)$$

What?

What is "RoboX"

- An end-to-end solution includes
  - Hardware **accelerator** for the core model predicative control (MPC) algorithms
  - Domain specific language (DSL) for RoboX

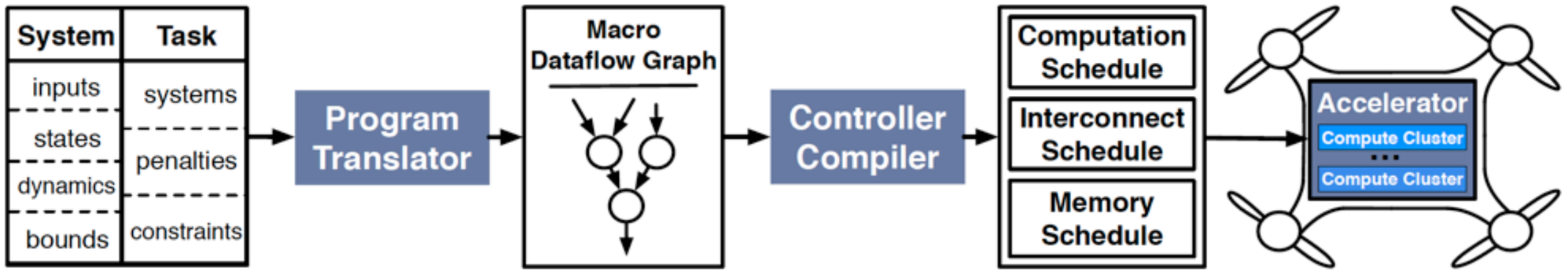


Figure 2: *RoboX* constitutes a domain-specific language, a unique hardware accelerator, and an automated compilation workflow.

The accelerator

- Compute Units
- Compute Clusters
- Compute-enabled interconnect
- Programmable Memory Access Engine

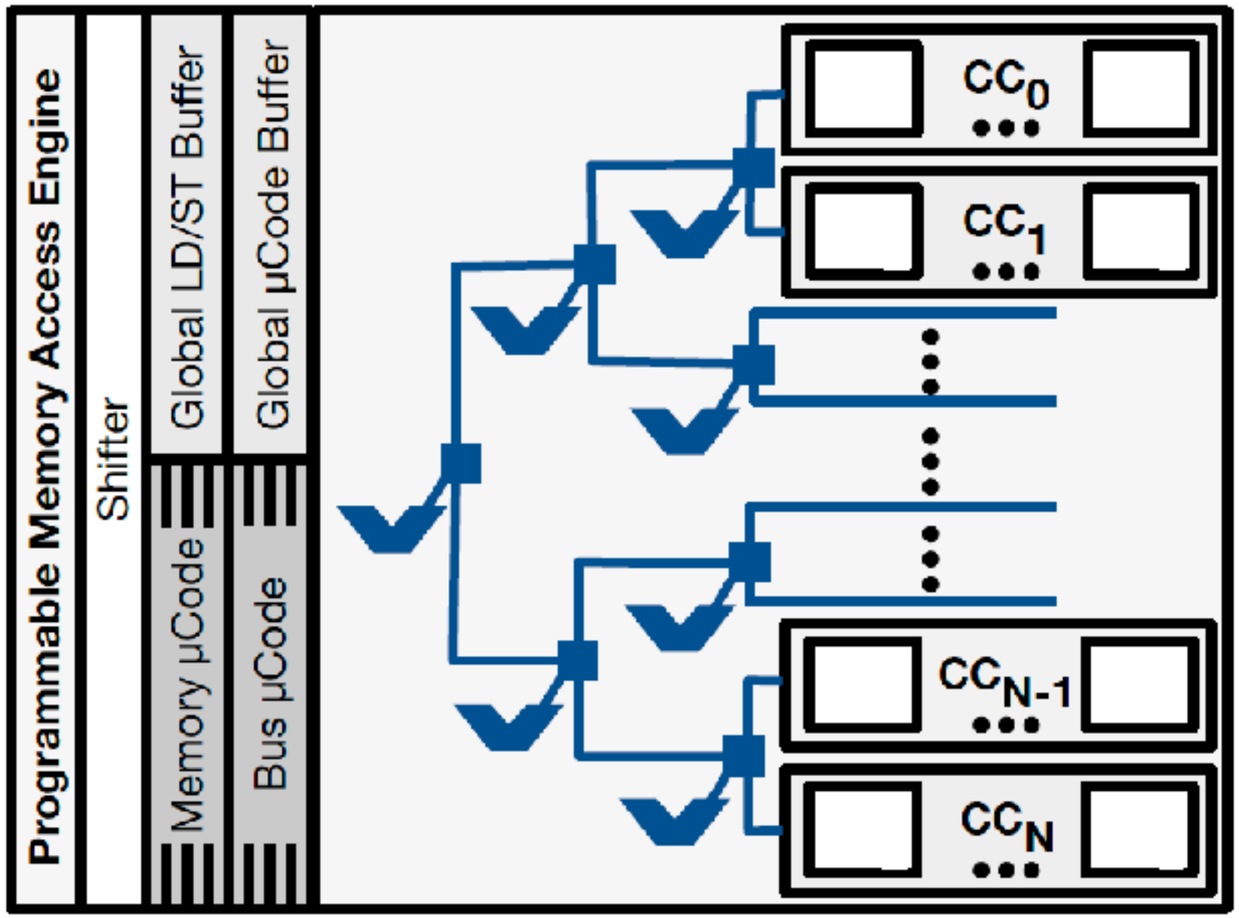
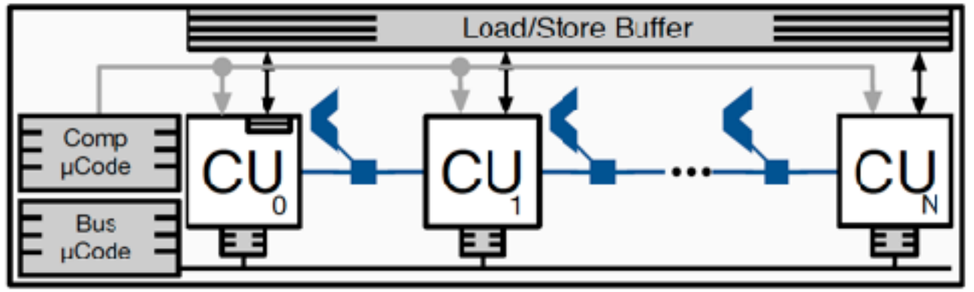


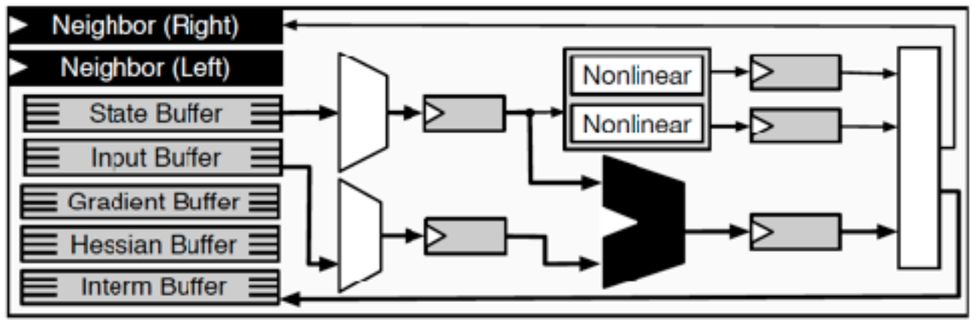
Figure 3: *RoboX* architecture which comprises: a hierarchical composition of Compute Units (CUs) into Compute Clusters (CCs); a compute-enabled interconnect; and a memory access engine.

Compute Units and Compute Clusters

- Why compute-clusters?
  - Data-level parallelism as many inputs are vectors/matrices



(a) Compute Clusters (CC)



(b) Compute Units (CU)

Compute-enabled interconnect

- Multiple-add
- Why compute-enabled interconnect? — Reduce the bandwidth for propagating data
- How to decide the usage of them? — DSL + Compiler

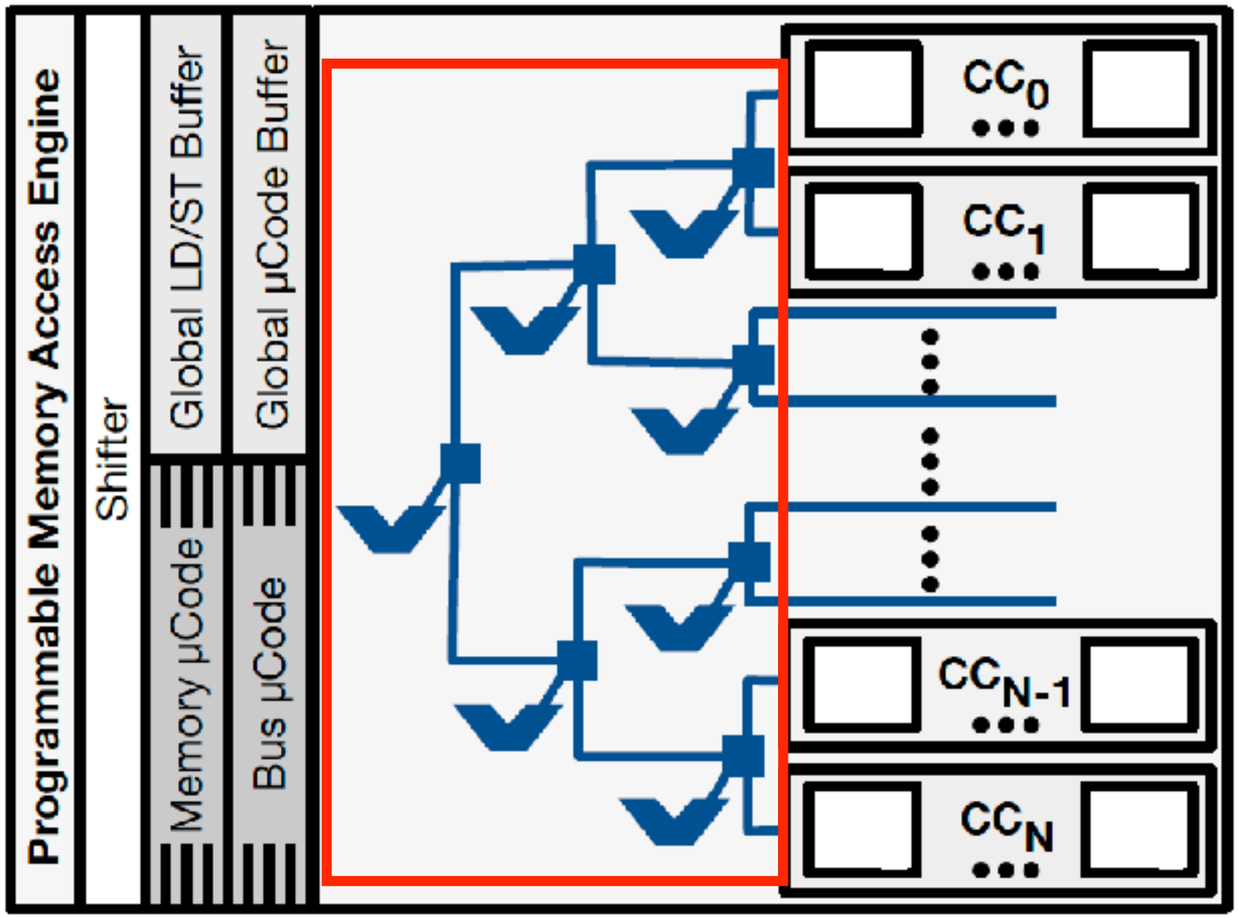


Figure 3: *RoboX* architecture which comprises: a hierarchical composition of Compute Units (CUs) into Compute Clusters (CCs); a compute-enabled interconnect; and a memory access engine.

The language

Table I: Language constructs of *RoboX*.

Type	Keyword	Description
Component	System	Definition of robot type comprising all necessary attributes
	Task	Task definition comprising penalties and constraints
Data Types	input	Robot control input
	state	Robot system state
	param	Constant parameter
	penalty	Term minimized during Task execution
	constraint	Constraint on terms in Task
	reference	Potentially time-varying term used to define penalties
	range	Defines an range for array access and group operations
Fields	lower_bound, upper_bound	Inequality constraints on a variable
	equals	Equality constraints on a variable
	running	Indicates enforced everywhere except last step of horizon
	terminal	Indicates enforced only at last step of horizon
	dt	Time derivative of state variable
	weight	Relative weight of penalty term
Mathematical Operations	+,-,*, /	Elementary operators
	sin, cos, ..., sqrt	Nonlinear operations
	sum, norm, min, max	Group operations

The ISA

Table II: The *RoboX* ISA, which is divided into separate compute, communication, and memory instructions.

	Bits	31 - 29	28 - 27	26 - 25	24 - 22	21 - 19	18 - 17	16 - 14	13 - 11	10 - 8	7 - 5	4 - 3	2 - 0
Compute	Scalar Queue Op	Opcode = 000	Function		Destination Namespace	Source 1 Namespace	Source 1 Pop	Source 1 Index	0		Source 2 Namespace	Source 2 Pop	Source 2 Index
	Vector Queue Op	Opcode = 001							Vector Length				
	Scalar Imm Op	Opcode = 010							0		Immediate		
	Vector Imm Op	Opcode = 011							Vector Length				
Communication	Unicast	Opcode = 000	Destination PU Quarter	Destination PE Quarter	Destination Namespace	Source Namespace	Source Pop	Source Index	Destination PE	Destination PU	0	Source PE Quarter	Source PE ID
	PE Multicast	Opcode = 010	Destination PE Quarter Mask						PE Mask				
	PU Multicast	Opcode = 011	Destination PU Quarter Mask						PU Mask				
	Broadcast	Opcode = 111	0						0				
	PE Aggregation	Opcode = 100	Destination PE Quarter Mask						PE Mask			Function	0
	PU Aggregation	Opcode = 101	Destination PU Quarter Mask						PU Mask				
Memory	Load	Opcode = 000	Offset					Shift Amt	PE Mask			PE Quarter	Namespace
	Store	Opcode = 001											
	Set Block	Opcode = 010	Block Number									0	Namespace
	End of Code	Opcode = 011	Not Used										

Example code

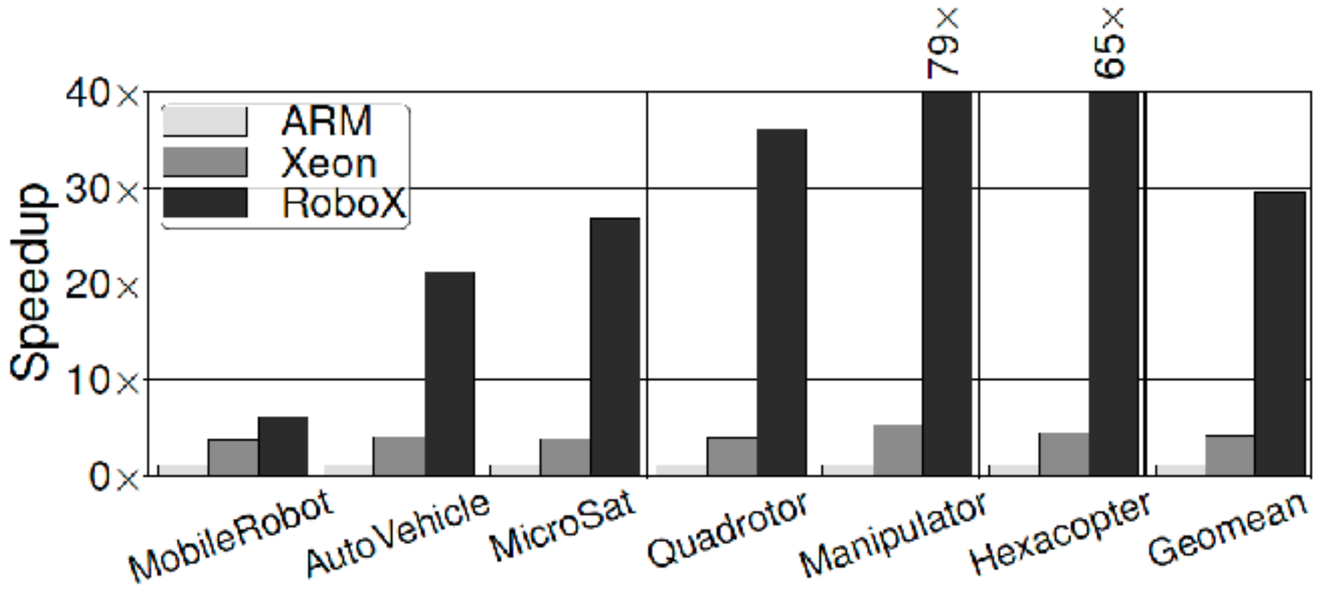
```
System MobileRobot(...) {
  Task moveTo(
    reference desired_x,
    reference desired_y,
    param weight,
    param radius) {
    // penalize distance from target
    penalty target_x, target_y;
    target_x.terminal = pos[0] - desired_x;
    target_y.terminal = pos[1] - desired_y;
    target_x.weight <= weight;
    target_y.weight <= weight;
    // constraints on position
    constraint pos_bound;
    pos_bound.running = pos[0]^2 + pos[1]^2;
    pos_bound.upper_bound <= radius;} }
```

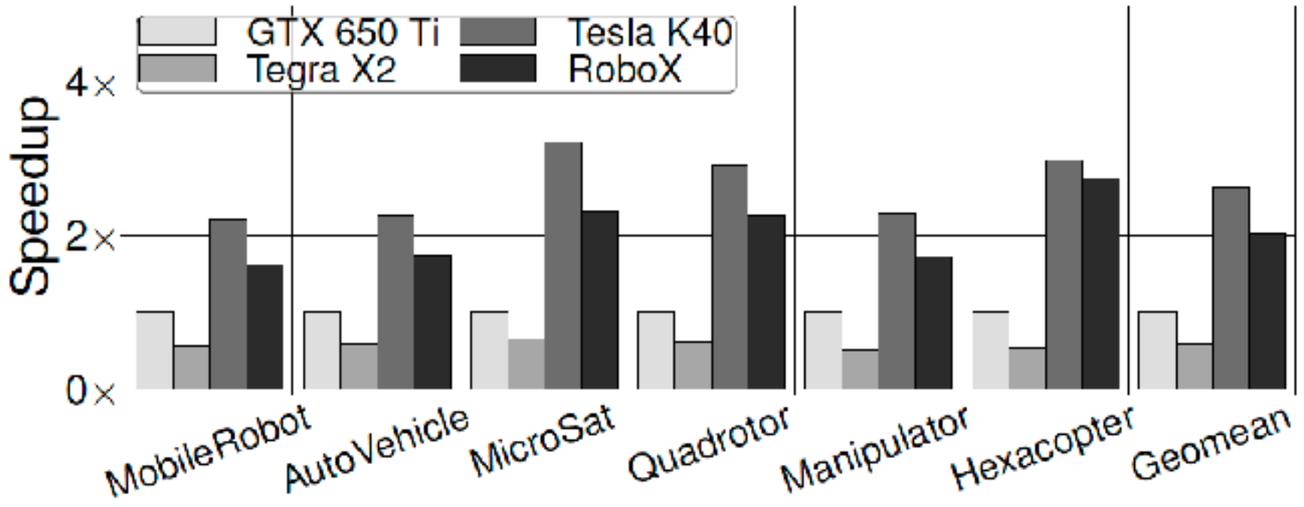
How?

Methodology

Using cycle-accurate simulations, we evaluate *RoboX* over six different robots: mobile robot, autonomous vehicle, manipulator, micro-satellite, quadrotor, and hexacopter. We compare *RoboX* to an optimized CPU implementation on a Intel Xeon E3 and ARM Cortex A57 and a custom GPU implementation on a Tegra X2, GTX 650 Ti, and Tesla K40. On average, *RoboX*, under a power budget of 3.4 Watts at 45 nm, provides 29.4× (7.3×) speedup over ARM A57 (Xeon E3) and a 7.8×, 65.5×, and 71.8× average performance-per-watt improvement compared to the Tegra X2, GTX 650 Ti, and Tesla K40, respectively. Results suggest that *RoboX* marks a first step towards enabling comprehensive solutions for robotic acceleration from high-level mathematical specifications.

How're they doing? — Speedup





Performance-per-watt

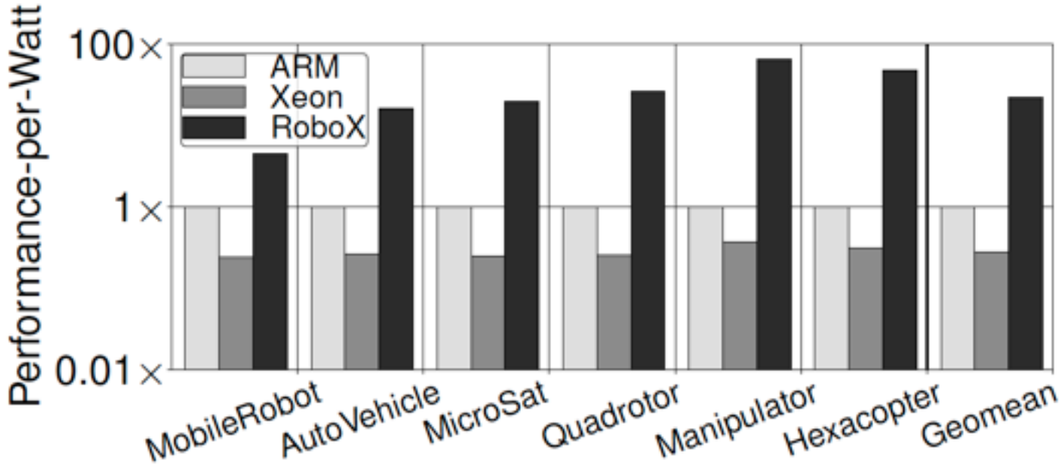


Figure 7: Performance-per-Watt improvement of Xeon E3 and *RoboX* over ARM A57 baseline.

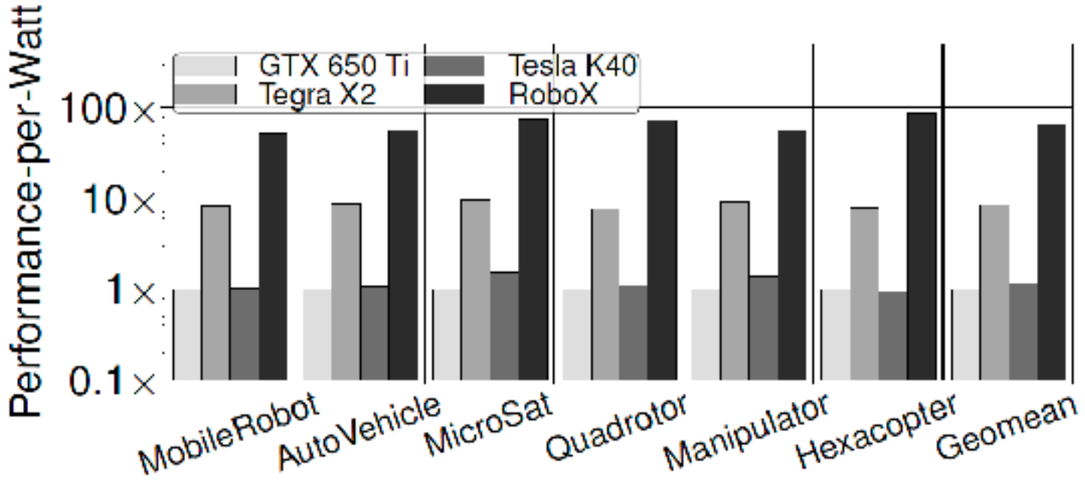


Figure 8: Performance-per-Watt improvement of GPUs and *RoboX* over GTX 650 Ti baseline.

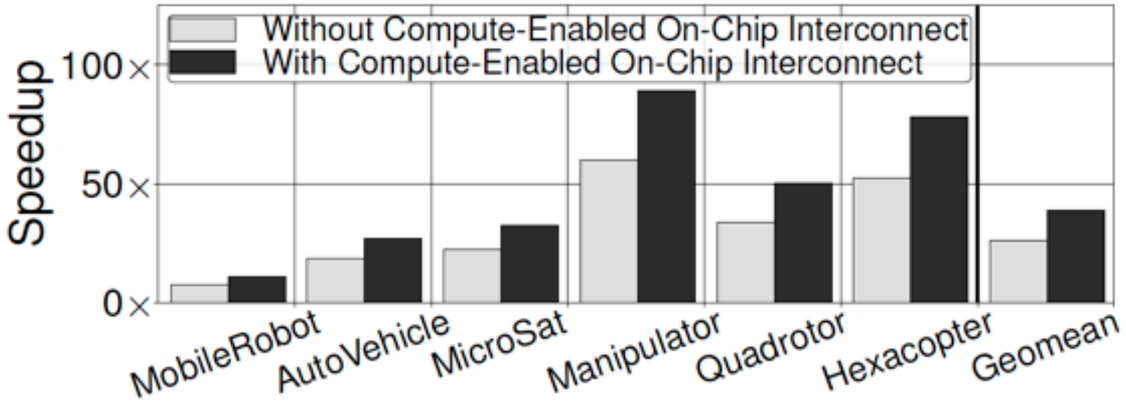


Figure 10: Average speedup of *RoboX* with and without the compute-enabled on-chip interconnect over ARM A57.

Contributions of the paper

- To move beyond conventional approaches of profiling and partially mapping code regions to specialized hardware modules, we build our acceleration solution atop an algorithmic understanding of the application domain. We observe that many diverse robotic applications are formulated as constrained optimization problems which can be solved online usingModel Predictive Control. Using this insight, we develop RoboX, an end-to-end acceleration solution for robotic motion planning and control.
- RoboX encapsulates a domain-specific language which enables programmers to express robotic applications close to their concise mathematical description.We provide a compiler which transforms this high-level specification into a concrete MPC formulation and solver, which is mapped to the accelerator.
- The RoboX architecture introduces compute-enabled on-chip interconnections, where hops integrate simple functional units. This functional unit, if required, can perform operations on in-transit data to reduce the burden on compute units. To support this architecture, we devise an ISA that offers three categories of instructions, where each program the interconnect, compute units, and memory interface, respectively. This flexibility allows efficiently executing different phases of robotic applications that alternate between dynamics and solver computation.

Discussions

Discussion Questions

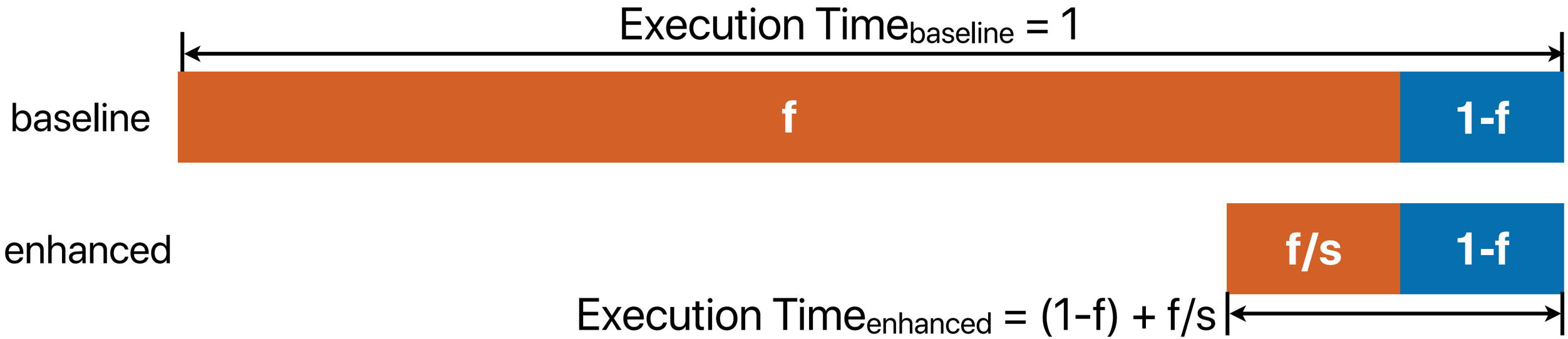
- Is “offloading to cloud” really an alternative?

Discussion Questions

- Do you agree with “To move beyond conventional approaches of profiling and partially mapping code regions to specialized hardware modules, we build our acceleration solution atop an algorithmic understanding of the application domain.”?

Amdahl's Law

$$Speedup_{enhanced}(f, s) = \frac{1}{(1 - f) + \frac{f}{s}}$$



$$Speedup_{enhanced} = \frac{Execution\ Time_{baseline}}{Execution\ Time_{enhanced}} = \frac{1}{(1 - f) + \frac{f}{s}}$$

Discussion Questions

- Comments on the description language?

Tensorflow

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

Discussion Questions

- Comments on evaluation methodology?

Announcement

- Sign up — you need to present to pass
  - We still have two slots for 1/21
  - One for 1/28
  - One for 2/4
  - Two for 2/27