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

Jacob Sacks
Divya Mahajan
Richard C. Lawson
Hadi Esmaeilzadeh†

Alternative Computing Technologies (ACT) Lab

Georgia Institute of Technology
†University of California, San Diego

ISCA ’18
Los Angeles, California
Challenges in Autonomous Robotics

- Many diverse applications
- Battery constraints
- Compute-intensive
- Limited power budget
Challenges in Autonomous Robotics

Flight Time

CPU

Mobile Processor
Challenges in Autonomous Robotics

Flight Time

Power
Accelerating Planning and Control

Model Predictive Control
RoboX Workflow

Domain-Specific Language

System Quadrotor() {
    state position[3], angle[3];
    input torque[4];
    ...
    Task takeOff() {
        penalty target_height;
        constraint max_height;
        ...
    }
}

Macro Dataflow Graph

Controller Compiler

Statically-Scheduled Instructions

- Computation Schedule
- Communication Schedule
- Memory Schedule

Concise mathematical description

Automatically synthesize DFG

Statically schedule on accelerator
Background: System Models

- yaw (\( \phi \))
- roll (\( \psi \))
- thrust (\( f_1 \))
- thrust (\( f_2 \))
- thrust (\( f_3 \))
- thrust (\( f_4 \))
- pitch (\( \theta \))
Background: Dynamics and Constraints

General nonlinear dynamics

\[ \dot{x} = f(x, u) \]

time derivative states inputs

State and input constraints

\[ f \leq f_i \leq \bar{f} \quad z \leq \bar{z} \]
Background: Objective Function
Background: Objective Function
Background: Objective Function

\[ J = P_{\text{term}}(x(t_f)) + \int_{t_0}^{t_f} P_{\text{run}}(x(t)) \, dt \]

- terminal cost
- running cost
Components of MPC

- Objective Function
- Dynamics
- Input Constraints
- State Constraints
Domain-Specific Language

Aims of RoboX DSL

- Distill MPC into modular components
- Remain close to mathematical expressions
- Independent of implementation

System
Task
Symbolic expressions
Group operations
DSL: System Component

```c
System MobileRobot() {
    state pos[2];
    state angle;
    input vel;
    input ang_vel;
    ...
}
```
DSL: System Component

```cpp
System MobileRobot() {
    state pos[2];
    state angle;
    input vel;
    input ang_vel;

    pos[0].dt = vel * cos(angle);
    pos[1].dt = vel * sin(angle);
    angle.dt = ang_vel;
    ...
}
```
System MobileRobot(...) {
    Task moveTo(...) {
        penalty target_x, target_y;
        target_x.running = pos[0] - desired_x;
        target_y.running = pos[1] - desired_y;
        ...
    }
}
System MobileRobot(...) {
    Task moveTo(...) {

        penalty target_x, target_y;
        target_x.running = pos[0] - desired_x;
        target_y.running = pos[1] - desired_y;

        constraint pos_bound;
        pos_bound.running = sqrt(pos[0]^2 + pos[1]^2);
        pos_bound.upper_bound <= radius;
    }
}
Flexible dataflow architecture organized as a two-level hierarchy to handle large amount of data dependencies.
RoboX Accelerator Architecture

Compute-enabled interconnect to perform simple operations on in-transit data
Each computer cluster executes separate compute and communication microprograms and can operate in a SIMD mode.
Compute units do not initiate communication requests but consume data from single-hop connections and a shared bus.
The compute unit is a three-stage pipeline and divides its memory into separate buffers to simplify communication scheduling.
Programmable memory access engine *prefetches* instructions and data according to its own statically-scheduled microprogram.
Instruction Set Architecture

Compute Instructions

- Scalar
- SIMD

Communication Instructions

- Data Transfer
- In-Network

Memory Instructions

- Load
- Store
Program Translator

Domain-Specific Language

- States and inputs
- Dynamics function
- Objective function
- Automatic differentiation for necessary gradients

Parameterized Solver Template
Controller Compiler

Mapping and Scheduling

- Computation Instruction Schedule
- Communication Instruction Schedule
- Memory Instruction Schedule

Decode

Programmable Memory Access Engine

Global LD/ST Buffer

Global Code Buffer

Shifter

Bus µCode

Compute Cluster 1
Compute Cluster 2
Compute Cluster N-1
Compute Cluster N
## Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>System</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileRobot</td>
<td>Two-Wheel Mobile Robot</td>
<td>Trajectory Tracking</td>
</tr>
<tr>
<td>Manipulator</td>
<td>Two-Link Manipulator</td>
<td>Reaching</td>
</tr>
<tr>
<td>AutoVehicle</td>
<td>Four-Wheel Vehicle</td>
<td>High-Speed Racing</td>
</tr>
<tr>
<td>MicroSat</td>
<td>Miniature Satellite</td>
<td>Orbit Control</td>
</tr>
<tr>
<td>Quadrotor</td>
<td>Four-Rotor Micro UAV</td>
<td>Motion Planning</td>
</tr>
<tr>
<td>Hexacopter</td>
<td>Six-Rotor Micro UAV</td>
<td>Attitude Control</td>
</tr>
</tbody>
</table>
Platforms

- **Low Power**
- **High Performance**

### CPU
- ARM Cortex A57
- Intel Xeon E3

### GPU
- Tegra X2
- GTX 650 Ti
- Tesla K40
On average, RoboX achieves a **29.4X** and **7.3X** speedup over the ARM A57 and Xeon E3, respectively.
On average, RoboX achieves a **2.0X** and **3.5X** speedup over the GTX and Tegra, respectively, and is **1.3X** slower than the Tesla.
On average, RoboX achieves a **65.5X, 7.9X, and 71.8X** performance-per-watt improvement over the GTX, Tegra, and Tesla, respectively.
Conclusion

Domain-general acceleration solution by leveraging algorithmic understanding of robotics

Deliver significant performance and energy gains while abstracting away details of controls, optimization, and hardware

First step towards enabling full-stack solutions for robotics from high-level mathematical specifications