MARS Framework
A Framework for resource management

Reflective System Model
- Kernel policy Model 1
- Kernel policy Model n
- Perf/Power Model

Resource Management Policies
- Policy model 0
- Policy 1
- Policy n

Actuators
- AA
- KA
- HA

System stack
- Applications
- Linux kernel
  - Scheduling, resource allocation, DVFS, etc
- HMP HW platform

Sensors
- AS
- KS
- HS

Sensed data

Pred. Sen. data

What if?

Predicted decisions
A Framework for resource management
Example: reflective task mapping

Reflective System Model
- What if task A migrated to core 0?
- Linux DVFS Gov. Model
- Core 0 freq. set to max.
- Perf/Power Model

Resource Management Policies
- Task Mapping policy
- Task A perf. increases

Actuators
- AA
- KA
- New Task->Core assignment

System stack
- Applications
- Linux kernel
  - Scheduling, resource allocation, DVFS, etc
- HMP HW platform

Sensors
- AS
- KS

Sensed data
MARS usage

• Crating a resource manager with two policies:

```java
class Simple_Map_Policy : public Policy_Manager {
    void setup() {
        registerPolicy (new Simple_Map_Policy(), 150);
        registerPolicy (new Simple_DVFS_Policy(), 50);
    }
};
```
MARS interface to user policies

• **sense** - returns sensed data for the latest execution period.
  
  E.g.: for a policy that executes every 50ms
  
  ```
  sense<SEN_INSTR_TOTAL>(task)
  ```

  returns the total number of instructions in the last 50ms

• **actuate** - sets an actuation knob
  
  E.g.: changing a task mapping:
  
  ```
  actuate<ACT_TASK_MAP>(task,core.id);
  ```

• **senself** - returns predicted sensed data for the next execution period.

• **tryActuate** - “What if I perform an actuation ?”
  
  • tryActuate affects senself results but it does not actually set physical actuation values
Simple_Map_Policy implementation

- If a task performance drops below established goal, searches for a new task mapping

```java
class Simple_Map_Policy : public Policy {
    void execute()
    {
        auto task = sys_info().tasks[0];
        double ips = sense<SEN_INSTR_TOTAL>(task) / sense<SEN_TIME_TOTAL_S>(task);
        if (ips < GOAL)
        {
            for (auto core : sys_info().cores)
            {
                tryActuate<ACT_TASK_MAP>(task, core.id);
                ips = sensel_self<SEN_INSTR_TOTAL>(task) / sensel_self<SEN_TIME_TOTAL_S>(task);
                if (ips >= GOAL)
                {
                    actuate<ACT_TASK_MAP>(task, core.id);
                    break;
                }
            }
        }
    }
};
```
Reflection process
MARS deployment
Implementation on Linux
Offline implementation
Offline simulation flow

PoliCym framework

Current actuations
Task-core-mapping, core freq, etc

App traces
One trace per app and per plat. config

Plat specs
#cores, core types, freqs, etc

Trace playback

Tasks' busy/idle periods

Scheduled trace playback

Hotspot
Reestimate temp. since plat. topology and task placement changes

Aging models

Performance, power

Temp, aging

Scheduling + tasks' core utilization

Linsched
Case studies

- Energy efficient task mapping for HMPs
- Managing chip wear-out on mobile platforms
- Design space exploration of HMPs
Case study: energy efficient task mapping

- **SPARTA**: Runtime Task Allocation for Energy Efficient Heterogeneous Many-cores
  - Light-weight QoS-aware task mapping
    - maximize the energy efficiency given performance constraints
  - Orthogonal to other policies
    - Performance/Power predictive models take in to account the behavior of the Linux’s **DVFS governor** and **CFS policy**.


SPARTA Overview

Sensing

cores: Huge Big Medium Little

t_2 t_1 t_3 idle

tasks: t_1 t_2 t_3

Prediction

IPS saturated
Target throughput met

5/16/2018
SPARTA Overview

- Effective throughput eventually saturated unless the task is 100% CPU bound
- Opportunity to move task to a slower core and save energy

![Graph showing Effective throughput vs Execution throughput](image)

- IPS saturated Target throughput met
- Meets target throughput and improves IPS/Watt
SPARTA Overview

- Due to workloads not being 100% cpu bound (i.e. interactive apps)
- Memory latency or throughput bound workloads
- Opportunity to move task to a slower core and save energy

- IPS saturated
- Target throughput met
- Meets target throughput and improves IPS/Watt
SPARTA Overview

cores: Huge ➔ Big ➔ Medium ➔ Little

tasks: \( t_1 \) ➔ \( t_2 \) ➔ idle ➔ \( t_3 \)

Select new mapping

IPS saturated Target throughput met

Meets target throughput and improves IPS/Watt
When `tryActuate/senseIf` is used, a model of Linux’s `ondemand` DVFS governor predicts the next core frequencies for the new mapping.

**Performance/Power model** predicts IPS and power given new mapping and frequencies:
- Offline trained model predicts per-task IPS/Power
- Scheduler model predicts resource utilization

List scheduling heuristic to explore new mappings:
- `tryActuate/senseIf` to find which cores meet QoS with highest energy efficiency
- Map tasks in order of energy efficiency