Managing Adaptability in Heterogeneous Architectures through Performance Monitoring and Prediction

Cristina Silvano
cristina.silvano@polimi.it
Politecnico di Milano
Motivations

- **System adaptivity** is a key issue for enabling applications running on many-core heterogeneous architectures to operate close to optimal efficiency in the face of:

  - Changing conditions by adjusting their behavior to their operating environments, usage contexts and resource availability
  
  - Meeting the requirements on performance, power budget and Quality-of-Service
This talk introduces an application auto-tuning framework for heterogeneous platforms in an adaptive multi-application environment.

The auto-tuning framework should manage at runtime the assignment of system resources to the active concurrent applications (request-level parallelism).
Application auto-tuning

- The approach exploits the concepts of:
  - Orthogonality between application auto-tuning and runtime management of system resources to support multiple adaptive applications.
  - Combination of design-time and run-time techniques to create an effective way of “self-aware” computing with limited runtime overhead.
Application & Platform Domains

Stream processing applications

Best-Effort Applications
Gaming
Engineering
Critical Applications

OpenCL application
Host program
Kernels

System-wide Run-time Resource Manager

Multi-core platforms, HPC, Mobile devices...

Heterogeneous Many-Core Platform
Heterogeneous parallel platforms

- Examples:
  - STMicroelectronics P2012 (STHORM)
  - Adapteva Parallela board
  - Nvidia Tegra K1

- Highly-parallel software-programmable accelerators
  - Locally homogeneous, globally heterogeneous
  - Computation offloading: separate host and device memory address spaces

- Different programming abstractions: native vs. OpenCL to support application code portability
Target problem (1)

- How to **efficiently program** (resource-aware programming) applications on heterogeneous many-core architectures to sustain performance?
Target problem (2)

- How to adapt at runtime the application behavior to sustain the performance?
Target problem (3)

- How to efficiently **allocate and manage** the system resources at runtime?
Most of the applications are **configurable** in terms of a set of parameters

- Color
- Shape
- Size
Key idea is that most of the applications are configurable in terms of a set of parameters by using run-time knobs to trade-off Quality of Results and Throughput.
Key idea is that most of the applications are configurable in terms of a set of parameters by using run-time knobs to trade-off Quality of Results and Throughput.
Why Run-Time Knobs & Auto-Tuning?

- In some applications internal knobs can be used at run-time to trade-off quality of results and performance.
Single application – Multiple instances

Autonomous Video-surveillance System

Video Frame Rate

Video Resolution
1. **Design-time exploration phase based on code profiling** to define the relations between application parameters and performance metrics (such as energy, throughput and accuracy).
2. Profiling information (OPs) are used to build a prediction model of application behavior, characterizing the effect of tunable parameters on performance metrics.
3. Prediction model supports run-time decisions based on monitoring to tune the application behavior according to available system resources.
Application Auto-tuning Framework (3)
Application Auto-tuning Framework (3)

Monitoring Application + AS-RTM

External events

User goal

Application-Specific Run-Time Manager (AS-RTM)

OP List

Monitors

Goal +/-
Application Auto-tuning Framework (20)
Orthogonality Concept

- Target HW Platform
- Platform OS
- OpenCL

Requests → Resources
Orthogonality Concept: App Auto-tuning

Target HW Platform

Orthogonality Concept: App Auto-tuning

Platform OS

Target HW Platform

Resource Availability

Requests

Resources

User goal

External events

Application

Application + AS-RTM

OpenCL APIs

OpenCL Kernels

Application-Specific Run-Time Manager (AS-RTM)

OP List

monitors

goal

Ext

Ext

User

User

Performance

Accuracy

Power consumption

Requests

Resources
Orthogonality Concept: App Auto-tuning + RTRM

- **Run-Time Resource Manager**
  - Application + AS-RTM
  - Monitors
  - User goal
  - OP List
  - Requests
  - Resources
  - Resource Availability
  - Platform OS
  - Target HW Platform

- **Working Modes**

- **External Events**
  - User
  - External events

- **Goal**

- **Application-Specific Run-Time Manager (AS-RTM)**
  - OpenCL APIs
  - OpenCL Kernels

- **Performance**
  - Accuracy
  - Power consumption

- **http://bosp.dei.polimi.it**
The Multi-View Case Study

2 eyes → third dimension
Pixel disparity

Left camera

Right camera

reference disparity

5 Application Knobs

QoR Disparity Error
Scenario: Multiple Multi-view applications

AMD NUMA 4 clusters quad-core Opteron @ 2.4 GHz

Linux CGroups

CPU-quota
Experimental Setup

- **Target Platform**
  - AMD NUMA Architecture: 4 nodes - 4 cores
    - OpenCL 1.2 run-time provided by AMD

- **Workload Definition:**
  - Single application – multiple instances
  - Dynamic workload in terms of:
    - Start time upon user request;
    - Amount of frame data to process;
    - Frame-rate goal demanded by user
Application Auto-Tuning Effects

![Graph showing FPS and ERR for different core counts (1-core, 2-cores, 3-cores, 4-cores) as a function of frame-rate goal. The graph depicts smooth and uneven performance metrics with smiley and frowning faces indicating satisfaction and dissatisfaction, respectively.]
Comparative Analysis

Application Auto-tuning

<table>
<thead>
<tr>
<th>Run-Time Resource Management</th>
<th>OFF</th>
<th>ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>PLAIN-LINUX</td>
<td>ADAPTIVE-LINUX</td>
</tr>
<tr>
<td>ON</td>
<td>PLAIN-RTRM</td>
<td>ADAPTIVE-RTRM</td>
</tr>
</tbody>
</table>
Multiple-Application Run-Time Results

**PLAIN-LINUX**

**ADAPTIVE-RTRM**

Throughput [frames/sec]

Error [%]

Time [s]
Multiple-Application Run-Time Results

ADAPTIVE-LINUX

ADAPTIVE-RTRM
Dynamic workload analysis by varying the number of MV instances

Evaluation metrics:

- **Average Normalized Actual Penalty (NAP):** User satisfaction to measure the distance of the run-time performance from the required frame-rate goal (Quality metric)

- **Runtime throughput degradation w.r.t. the expected throughput based on offline profiling** (Predictability metric)

- **Normalized output quality loss:** User satisfaction in terms of quality of the resulting image (Error metric)
Contestion on shared resources

SW-RTRM

APP1

APP2

LLC

Memory

LLC misses

Num. instances

Plain Linux
Proposal

0 1 2 3

1e9
Conclusions

- We proposed an approach to support application adaptivity in many-core architectures exploiting:
  - A runtime resource manager operating at system-level combined with an application auto-tuning framework
  - Performance/quality trade-offs and low overhead due to the combination of design-time and run-time techniques based on prediction models
- Most recent projects: **FP7-2PARMA** and **FP7-HARPA** (on-going)