Runtime Resource Management

Politecnico di Milano
Seminar Room A. Alario
27 November, 2015

Antonio R. Miele
Marco D. Santambrogio
Politecnico di Milano
Modern architectures are heterogeneous

- A modern computer architecture includes
  - One or more CPUs
  - One or more GPUs
  - Optionally other HW accelerators
Modern architectures are heterogeneous

- Each unit presents a different profile in terms of
  - Computational efficiency
  - Power consumption
- Example: ARM big.LITTLE
Modern architectures are heterogeneous

• Each unit presents a different profile in terms of
  – Computational efficiency
  – Power consumption
• Example: AMD Accelerated Processing Unit

Computational efficiency depends also on the “internal structure” of the application
Workloads are dynamic

- System’s workload are highly evolving, variable and heterogeneous in many application scenarios (from mobile phones to HPC servers)
  - Different types of applications
  - Different amount of data and input parameters per each run of the application
    - Applications with different use-mode
  - Different execution times
    - Depending on the application and the amount of processed data
  - Different possible Quality-of-Service (QoS) requirements
    - In terms of throughput, turnaround time, deadlines, ...
  - Unknown arrival times
    - Depending on the user requests
Workloads are dynamic

- On a mobile phone:
  - Phone calls
  - Short message service
  - Web browsing
  - Audio/video playing
  - Gaming

Generally short execution times, low amount of processed data, actually no QoS requirements or not-challenging ones

Generally considerable amount of data to be processed with specific throughputs to be fulfilled, high demanding elaborations
Workloads are dynamic

• On a HPC server:
  – Financial modeling and analysis
  – Fluid dynamic simulations
  – Weather and climatic modeling
  – ...
Energy/power consumption issues

- Energy and power consumption - *Are they a constraint for...*
Energy/power consumption issues

- Energy and power consumption - Are they a constraint for...
  - Embedded & IoT
  - Mobile devices
Energy/power consumption issues

- Energy and power consumption - Are they a constraint for...
  - Embedded & IoT
  - Mobile devices

Batteries capacities
Energy/power consumption issues

- Energy and power consumption - *Are they a constraint for...*
  - Embedded & IoT
  - Mobile devices
  - Desktops
  - Servers
  - HPC Clusters
Energy/power consumption issues

- Energy and power consumption - Are they a constraint for...

<table>
<thead>
<tr>
<th>- Embedded &amp; IoT</th>
<th>Batteries capacities</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Mobile devices</td>
<td></td>
</tr>
<tr>
<td>- Desktops</td>
<td>Energy costs</td>
</tr>
<tr>
<td>- Servers</td>
<td></td>
</tr>
<tr>
<td>- HPC Clusters</td>
<td></td>
</tr>
</tbody>
</table>
Energy/power consumption issues

• Energy and power consumption - Are they a constraint for…
  – Embedded & IoT
  – Mobile devices
  – Desktops
  – Servers
  – HPC Clusters

+ Thermal issues!

AMD Athlon II x2 240 – Dual Core running SPEC CPU 2006
Energy/power consumption issues

- Energy and power consumption - Are they a constraint for...
  - Embedded & IoT
  - Mobile devices
  - Desktops
  - Servers
  - HPC Clusters

+ Thermal issues!
- Main cause is the power consumption
- Modern chips are characterized by a Thermal Design Power (TDP)
Energy/power consumption issues

- Energy and power consumption - Are they a constraint for...
  - Embedded & IoT
  - Mobile devices
  - Desktops
  - Servers
  - HPC Clusters

+ Thermal issues!

- Consequences of over-heating
  - Higher cooling costs
  - Accelerated aging
  - Chip burn
Energy/power consumption issues

• Energy and power consumption - **Are they a constraint for**…
  – Embedded & IoT
  – Mobile devices
  – Desktops
  – Servers
  – HPC Clusters

• Any optimization needs a deep understanding of the phenomenon
Overall picture

Various types of applications
Different arrival times
Application-specific QoS requirements
Power consumption requirements
Energy consumption requirements
Thermal constraints
...

• How can we accurately handle all these requirements?
A problem as a new opportunity

- Programming has become very difficult
  - Impossible to balance all constraints manually
A problem as a new opportunity

- Programming has become very difficult
  - Impossible to balance all constraints manually
- More computational horse-power than ever before
  - Cores are free
A problem as a new opportunity

- Programming has become very difficult
  - Impossible to balance all constraints manually
- More computational horse-power than ever before
  - Cores are free
- Energy is new constraint
  - Software must become energy and space aware
A problem as a new opportunity

• Programming has become very difficult
  – Impossible to balance all constraints manually
• More computational horse-power than ever before
  – Cores are free
• Energy is new constraint
  – Software must become energy and space aware

We cannot handle all these aspects manually and at design time due to the unknown and unpredictable runtime working scenario
A problem as a new opportunity

• Programming has become very difficult
  – Impossible to balance all constraints manually
• More computational horse-power than ever before
  – Cores are free
• Energy is new constraint
  – Software must become energy and space aware

• Modern computing systems need context-awareness
  – be aware of the surrounding environment conditions
  – know internal state
• To optimize and meet their requirements taking advantage as much as possible of the context to pursue concurrent goals
Need for runtime resource management

- In order to deal with a highly dynamic and evolving working scenario we need a runtime strategy

New software component on top of (or within) the operating system observing the system behavior and acting on it.
Self-adaptive resource management

• Self-adaptive systems: systems that can observe
  their runtime behavior, learn, and take actions to
  meet desired goals

• Characteristics:
  – Goal-oriented
    • Tell the system what you want
    • System’s job to figure out how to get there
  – Approximate
    • Does not expend any more effort than necessary to meet
      goals
Observe-decide-act loop

- Self-adaptive systems generally implements the so-called Observe-Decide-Act loop.
Observe-decide-act loop

Self-adaptive systems generally implement the Observe-Decide-Act loop.

The controller collects raw observations from the available sensors/monitors and computes high level metrics.
Observe-decide-act loop

Self-adaptive systems generally implement the Observe-Decide-Act loop. Examples of raw observations are the heartbeat, the current power consumption, the current temperature.
Observe-decide-act loop

Systems generally implement the Observe-Decide-Act loop.

Examples of metrics:
- Throughput
- Average power consumption
- Average temperature

Diagram:
- Observe
- Proposed Metrics
- Decisions
- Control Knobs
- Act
- Goals / Constraints
Observe-decide-act loop

• Self-adaptive systems generally implement the so-called Observe-Decide-Act loop.

The controller takes decisions on the resource management on the basis of an integrated policy.
Observe-decide-act loop

- Self-adaptive systems generally implement the so-called Observe- Decide-Act loop.

Inputs of the decision policy are the computed metrics and the specified goals/constraints.
Observe-decide-act loop

- Self-adaptive systems generally implement the so-called Observe-Decide-Act loop.

Examples of goals/constraints are an application QoS or a power/energy budget.
Observe-decide-act loop

- Self-adaptive systems generally implement the so-called Observe-Decide-Act loop.

Many strategies can be used to implement the policy:
- Heuristics
- Control theory
- Machine learning
- ...
Observe-decide-act loop

- Self-adaptive systems generally implement the so-called Observe-Decide-Act loop.

Finally decisions are actuated by controlling the available architecture/application knobs.
Observe-decide-act loop

- Self-adaptive systems generally implement the so-called Observe-Decide-Act loop.

Examples of architecture knobs are the DVFS or the core power gating.
Observe-decide-act loop

- Self-adaptive systems generally implements the so-called Observe-Decide-Act loop.

Examples of application knobs are the selection of the number of threads or the application mapping.
The runtime resource management layer is placed on top of (or within) the operating system.
Resource management policies in modern OS

• The definition of policies for runtime resource management is still a research topic

• Modern OS employs simple policies not-aware about the application requirements

• E.g.:
  – Completely Fair Scheduler
  – DVFS governor
  – Heterogeneous MultiProcessing (HMP) scheduler
Resource management policies in modern OS

• Completely Fair Scheduler is the Linux scheduler
  – Dispatches running processes on available processors
  – Targeted for symmetric multiprocessor systems
  – Aims at maximizing overall CPU utilization
  – Aims at maximizing interactive performance
Resource management policies in modern OS

• DVFS governor
  – Controls the voltage/frequency of the cores
  – Decides the voltage/frequency according to the observed CPU utilization level
  – Aimed at
    • Providing computational power when applications require
    • Saving power when the system is unloaded
Resource management policies in modern OS

- HMP scheduler is the Linux scheduler for big.LITTLE architectures
  - Dispatches running processes on processors
  - Move computational intensive processes on the big cores
  - Move non-computational intensive on the LITTLE cores
  - Aimed at guaranteeing high performance only to processors that requ
Research on runtime resource management

• Academic/industrial research is intensively investigating adaptive policies for runtime resource management

• Studies on
  – Mechanisms to monitor end-to-end QoS and define related requirements
  – Mechanisms to actuate control decisions
  – Policies for the optimization of the trade-off between application requirements and system ones
    • Performance vs. power vs. energy vs ...
Application monitoring and actuation

- Definition of mechanisms for monitoring and actuating on the application
The Heart Rate Monitor

- Set performance goal
- Run the app and update progress

Statistics automatically updated

e.g.:
min: 25hb/sec
max: 35hb/sec
The Heart Rate Monitor

• Heartbeats signal either progresses or availability
  – video encoder: 1 heartbeat = 1 frame
  – web server: 1 heartbeat = 1 request
  – database server: 1 heartbeat = transaction

• Heart rate as a performance measure and goal
  – High-level, application-specific performance measurements and goals (e.g., video encoder: 30 heartbeats/s = 30 frames/s)

• This performance monitor is implemented with a library used to instrument the application
Actuation on the application

• Setting of the execution specific parameters
  – Select the implementation to be used (CPU, GPU,...)
  – Set the number of threads

• Setting of algorithmic-specific parameters
  – Some applications have parameters to trade off result quality vs. execution latency
Actuation on the application

- Setting of the execution specific parameters
  - Select the implementation to be used (CPU, GPU, ...)
  - Set the number of threads
- Setting of algorithmic-specific parameters
  - Some applications have parameters to trade off result quality vs. execution latency
- All these parameters can be controlled only directly from the application source code
  - Also in this case the runtime resource manager have to be connected with the application to actuate
Adaptive application template

• Template of adaptive application:

```c
int main() {
    ...
    // initialization block
    register_monitor();
    ...
    for(i=0; i<NUM_OF_CHUNKS; i++) {
        mapping = get_mapping();
        if(mapping = CPU_MAPPING) {
            ...
        } else if(mapping = GPU_MAPPING) {
            ...
        }
    ...
    heartbeat();
}
// final block
...
deregister_monitor();
}
```

- Communicate required performance
- Get current mapping
- Update performance measurements
- Communicate termination
Self-adaptive policies

- Some examples of policies...
Run without any control

- Architecture: Intel Core i7 quad-core CPU
- Workload: 1 x264
Core allocations

• **QoS requirement:**
  – fulfill a given throughput level

• **Strategy:**
  – allocate a set of cores and
  – set the proper number of threads
Core allocations
Performance-aware fair scheduler

• In a scenario where applications
  – are competing for the same set of resources
  – require predictable performance, expressed through high-level, application-specific metrics

• The scheduler has to become Performance-Aware to automatically allocate resources to match performance goals
The controlled run

- Architecture: Intel Core i7 quad-core CPU
- Workload: 1 x264 with throughput requirements
Two controlled runs, different goals

• Architecture: Intel Core i7 quad-core CPU
• Workload: 2 x264 with different throughput requirements
Two controlled runs, different goals

- Architecture: Intel Core i5 dual core CPU
- Workload: 2 Black&Scholes with different minimum throughput requirements

- We are able to adapt to the working scenario
Three controlled runs, different goals

- Architecture: Intel Core i5 CPU + 8 Maxeler DFEs
- Workload: 3 Black&Scholes with different deadlines
Five controlled runs, different goals

- Architecture: Intel Core i5 dual core CPU + Nvidia GPU
- Workload: complex workload
Four controlled runs, different goals

- Architecture: ARM big.LITTLE + ARM MALI GPU
- Workload: complex workload

The side-effect is that we may save power since we are slowing-down application execution when required
DVFS tuning

• **QoS requirement:**
  – fulfill a given throughput level and
  – minimize power and/or energy consumption

• **Strategy:**
  – allocate a set of cores
  – set the proper number of threads and
  – tune voltage and frequency levels
Scheduling and DVFS-control

- Resource allocation can be used in conjunction with DVFS control to
  - Meet application requirements
  - Save power consumption

- Different possible strategies: race-to-idle, never-idle, ...
Scheduling and DVFS-control

- Architecture: ARM big.LITTLE
- Workload: 1 blackscholes

![Graphs showing throughput, power, temperature, and board energy over time for different policies and energy management policies.](image)
DVFS tuning

- **QoS requirement:**
  - Control temperature
  - Maximize performance

- **Strategy:**
  - Tune voltage and frequency levels
  - Inject idle cycles
Temperature Control/Management

set a temperature cap

Temperature Control/Management

DVFS is dangerous

Temperature Control/Management

The AcOS refreshment: ThermOS

SAVE orchestrator

• The framework for runtime resource management in the SAVE project...
Orchestrator framework

- The targeted HSA is composed of a set of heterogeneous processing resources
- Resources are organized in homogeneous clusters
Orchestrator framework

- The workload is composed of loop-based and computational-intensive applications
- Arrival times and amount of data to be processed are unknown
- Applications are slightly instrumented to enable adaptiveness
The orchestrator is the RTRM middleware

- It controls a set of processing element’s managers (PE managers)
- It implements an Observe-Decide-Act loop
The orchestrator is the RTRM middleware.
- It controls a set of processing element’s managers (PE managers).
- It implements an Observe-Decide-Act loop.
Orchestrator framework

- The orchestrator monitors the applications’ performance by means of specific API
- Metrics: throughput, latency, performance/Watt
Orchestrator framework

- The orchestrator monitors the resource status by means of the manager interface.
- Metrics:
  - Power consumption
  - Energy consumption
  - Utilization
  - Temperature
  - Voltage/frequency
Orchestrator framework

- The orchestrator collects:
  - System-level requirements (e.g. energy/power budget)
  - Applications requirements (e.g. minimum throughput)
Orchestrator framework

- The decision module consists of two activities
  - Dispatch applications to clusters
  - Set resource constraints to clusters
Orchestrator framework

- The orchestrator actuates decisions by commanding
  - The PE managers
  - The applications through the APIs
Adaptive applications

- Applications are slightly instrumented with a specific API to interact with the orchestrator
  - Communicate performance goal
  - Monitor current performance (Heartbeat library)
  - Actuate on the application dispatching and other parameters (e.g. #threads)
PE managers

- PE managers are the **actuators** for the orchestrator.

- A **PE manager** controls a cluster of homogeneous resources (CPUs, GPUs, DFEs).

- This **hierarchical organization** offers scalability and extendibility of the middleware.
PE managers

• The orchestrator assigns to the PE managers
  – The applications to be executed
  – The “resources’ budget”

• Each manager implements an ODA loop
  – It tries to fulfill applications’ requirements within the given resources’ budget
  – The manager communicates possible goal failures to the orchestrator
  – If necessary, the orchestrator explores other mapping solutions
PE managers

• Available actuation knobs:
  – CPU manager:
    • Task mapping
    • Dynamic voltage and frequency scaling
    • Idle cycle injection
  – GPU manager:
    • Task mapping
    • Dynamic voltage and frequency scaling
  – DFE manager:
    • Group setting
The orchestrator is implemented in C++ on Linux OS:

- It consists of a single process running in userspace.
- CPU/GPU managers are implemented in the same process.
- DFE manager is an external process.
A custom application monitoring/control library has been implemented

- Heartbeat mechanism to enable performance measurement
- Communications by means of shared memory
- Configuration descriptors in JSON
- Applications should implement a specific template to enable monitoring and control
Orchestrator implementation

- PE Managers have been implemented relying on OS facilities
  - Application mapping on CPUs performed by `set_affinity`
  - HW monitoring and control by means of Linux virtual file systems
  - DFE controller adapted and extended
Case studies

• Considered architectures:
  – Intel Core i5 dual core CPU + Nvidia GeForce GPU
  – ARM big.LITTLE + ARM MALI GPU
  – Intel Core i5 dual core CPU + 8 Maxeler DFEs

• Live demo...