The BarbequeRTRM Framework
2PARMA Framework for Run Time Resource Management of Multi-Core Computing Platforms

Speaker: Patrick Bellasi
bellasi@elet.polimi.it

Demo: Giuseppe Massari
massari@elet.polimi.it

Politecnico di Milano
Dipartimento di Elettronica e Informazione

Presentation Outline

- Introduction to RTRM
- The Barbeque approach to RTRM
- BarbequeRTRM internals
- BOSP: The BarbequeRTRM Open Source Project
- Demo video and practical overview
Why Run-Time Resources Management?

- Run-Time Resources Management (RTRM) is about finding the optimal tradeoff between QoS requirements and resources availability.

  - **Target scenario**
    - Shared HW resources
      - *upcoming many-core devices are complex systems*
      - *process variations and run-time issues*
    - Mixed SW workloads
      - *resources sharing and competition*
        - among applications with different and time-varying requirements
  
  - **Simple solutions are required**
    - support for frequently changing use-cases
    - suitable for both critical and best-effort applications
Introduction to RTRM

Goals of Run-Time Resources Management

- Multiple devices, subsystems
  Heterogeneous -> Homogeneous (Many-Cores)

  Scalability and Retargetability

- Shared resources among different devices and applications
  Computation, memory, energy, bandwidth...

  System-wide resources management

- Multiple applications and usage scenarios
  Run-time changing requirements

  Time adaptability

Main Goals: 1/3 - Hierarchical Control

Support monitoring, management and control at different granularity levels to reduce overheads

Different granularity
- accellerated application
- operating system
- computation fabric
- computation clusters

How to reduce control complexity?
Each granularity level collects requirements from lower levels and it provides constraints to lower levels
Introduction to RTRM
Main Goals: 2/3 - Resources Virtualization

*Map “virtual resources” on “physical resources” at run-time to achieve optimal platform usage*

Considering run-time phenomena
- process variation
- hot-spot and failures
- workload variation

*How to support optimal system resource exploitation?*

Virtual resources representation to support accounting; map on physical ones at run-time to handle variations

---

Introduction to RTRM
Main Goals: 3/3 - Dynamic Resources Partitioning

*Grant resources to critical workloads while optimize resource usage by best-effort workloads*

Considering a *mixed-workload* scenario
- critical workloads could be off-line optimized (e.g., using DSE)
- other workloads runs concurrently

*How to handle resources granted to critical applications?*

Dynamically grant these resources to best-effort workloads while not required by critical ones
### Introduction to RTRM

#### Resources Related Requirements

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RI.1</td>
<td>definition of resources working-modes&lt;br&gt;Any component in a use-case which represent a resource must completely define its <strong>working modes</strong> and notify them to this tool.</td>
</tr>
<tr>
<td>RI.2</td>
<td>definition of resources control points&lt;br&gt;Subsystems must expose their <strong>control points</strong> and define how modification impact on their behaviors (e.g. power and performances)</td>
</tr>
<tr>
<td>RI.3</td>
<td>resources observability&lt;br&gt;Subsystem are expected to expose some <strong>observability points</strong>, represented by metrics that can be used to identity their behaviors.</td>
</tr>
</tbody>
</table>

Most of these requirements impact on the lower abstraction levels of the RTRM, i.e. the platform specific code.

---

### Applications Related Requirements

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RI.4</td>
<td>priority based access to resources&lt;br&gt;<strong>Critical applications could preempt</strong> resources used by best-effort applications.</td>
</tr>
<tr>
<td>RI.5</td>
<td>user-space representing application&lt;br&gt;<strong>Every RTRM client tool has a corresponding controlling user-space application</strong> which defines its resource access rights and priority.</td>
</tr>
<tr>
<td>RI.6</td>
<td>query resource availability&lt;br&gt;Applications (best-effort) are required to <strong>query</strong> the RTPM tool to know <strong>about resources availability</strong>.</td>
</tr>
<tr>
<td>RI.7</td>
<td>get and release resources&lt;br&gt;<strong>Each tool user is required to notify</strong> when a resource is used and released.</td>
</tr>
<tr>
<td>RI.8</td>
<td>handle RTRM notifications&lt;br&gt;Applications (best-effort) could be requested to <strong>adapt to resource availability changing conditions</strong>.</td>
</tr>
</tbody>
</table>
## The Barbeque RTRM Framework

### Introduction to RTRM

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC.1 monitor resource performance</td>
<td>Export an updated view of the usage and behaviors of each subsystem, and their resources, with different details levels.</td>
</tr>
<tr>
<td>RC.2 dynamic resource partitioning</td>
<td>Grant resources to critical workloads while dynamically yield these resources to best-effort workloads when (and only while) they are not required by critical ones, thus optimize resource usage and fairness.</td>
</tr>
<tr>
<td>RC.3 resource abstraction</td>
<td>Handle a decoupled resource view between the users and the underlying hardware.</td>
</tr>
<tr>
<td>RC.4 multi-objective optimization policy</td>
<td>System-wide and multiple metrics optimization policy.</td>
</tr>
<tr>
<td>RC.5 dynamic optimization policy</td>
<td>Support multiple and tuneble policies to fit well specific optimization scenarios.</td>
</tr>
<tr>
<td>RC.6 low run-time overheads</td>
<td>Reduced impact on the performances of the controlled system.</td>
</tr>
</tbody>
</table>

---

### The Barbeque Approach to RTRM

*an overall view on proposed tool architecture*
The BarbequeRTRM Framework

Overall Contributions

- **Methodology** to support system-wide run-time resource management exploiting design-time information hierarchical and distributed control

- **BarbequeRTRM Framework**
  - multi-objective optimization strategy
  - easily portable and modular design
  - run-time tunable and scalable policies
  - open source project

### System-Wide RTRM: Overall Framework

**Application-Specific RTM**
- Fine grained control on application allocated resources:
  - task ordering
  - virtual processor assignment
  - DVFS
  - application parameters monitoring

**System-Wide RTRM**
- Coarse grained control on platform available resources:
  - resource accounting, partitioning and abstraction
  - high-level HW events handling e.g., critical conditions, faults...
  - manage applications priorities
  - power/thermal "coarse tuning"

---

Legend:
- SW Interface (API)
- SW/HW Meta-data
Each application uses a certain quantity of resources. Working modes define the amount of required resources for each kind of "platform" provided resources. Metadata (a) and (b) defines these quantities.

The RTRM keeps track of used/free resources, thus each application must "ask" the RTRM for resources...

Applications see only virtual resources:
- To improve portability
- Mapping on physical ones is performed by lower abstraction layers

Applications view virtual resources described by working modes, assigned by the RTRM. I.e., they know how much PEs can be used, but not what physical PEs will be actually assigned.

Platforms define physical resources described by meta-data (d), both kind and hierarchy.

Mapping from virtual to physical resources supported by Barbeque lower abstraction layer using generic (D) or platform specific (E) interfaces.

The platform driver should actively support both abstraction and mapping.
The BarbequeRTRM Framework

Main Components: Resources Partitioning

- Application may have different “priority levels”
  critical vs best-effort, different usage scenarios
  => different user QoE requirements
- Applications may have different “working modes”
  working modes maps on different resource requirements
  e.g. different amount, different quality, …
  only some applications could be off-line tuned
  e.g. critical apps could be off-line profiled using DSE techniques

- Meta-data a and b
  application working modes and priorities
- Meta-data c
  re-compilation hooks
- Meta-data e
  should be embedded with a and b

Host-Side Integration: The RTLib

- Runs as a (real-time) user-space process (deamon)
- Applications access BBQ features using the RTLib
  shared library masking the RTRM as much as possible
  used to collect application requirements
  e.g. working modes, run-time constraints, …
  notifies applications about changes on resources availability
  i.e., it could trigger a (forced) working mode change
- Hides the actual RTM communication channel (C)
  e.g., IPCs, dbus, binder, sockets, …
- Exploit platform specific services
  meta-data specification
  e.g. XML, recipes, dconf
  communication channels
  e.g. binder, dbus
- Either standard (D) or platform specific interfaces (E) implemented as (compile time selectable) modules
- Multicore Association standard support
  i.e. MRAPI and MCAPI
- Platform description (d) required
  resources and hierarchical layout
  possibly using open standards
  e.g. using a Flattened Device Tree
- Monitoring interface (F)
  with very low overhead and jitters
- Possibly support customization interfaces (H) and (I)
  to support enforced task mapping and resource tuning
  e.g. shrink OpenCL run-time resources availability, tune memory allocation parameters
### The BarbequeRTRM

**System-Wide RTRM: Presentation Outline**

**System-Wide RTRM**
- Coarse grained control on platform available resources:
  - resource accounting, partitioning and abstraction
  - high-level HW events handling e.g., critical conditions, faults...
  - manage applications priorities
  - power/thermal “coarse tuning”

**Application-Specific RTM**
- Fine grained control on application allocated resources:
  - task ordering
  - virtual processor assignment
  - DVFS
  - application parameters monitoring

**Legend**
- SW Interface (API)
- SW/HW Meta-data

---

**The BarbequeRTRM**

**System-Wide RTRM: Distributed Hierarchical Control**

**Legend**
- SW Interface (API)
- SW/HW Meta-data
Different subsystems have their own control loop (CL)
- System-wide level (resources partitioning, system-wide optimization, ...)
- Application specific (application tuning, dynamic memory management, ...)
- Firmware/OS level (F/V control, thermal alarms, resource availability, ...)

- FF closed CL using OP and AWM
- Optimal user defined goal functions including overheads
- Robust
- Adaptive
BBQ Validation Policy
- enforce certain control properties
  - energy budget, stability and robustness
- authorize resources synchronization

The BarbequeRTRM Framework
System-Wide RTRM: Resource Partitioning Strategy

Application-Specific RTRM
- fine grained control on application allocated resources:
  - task ordering
  - virtual processor assignment
  - DVFS
  - application parameters monitoring

System-Wide RTRM
- coarse grained control on platform available resources:
  - resource accounting, partitioning and abstraction
  - high-level HW events handling e.g., critical conditions, faults...
  - manage applications priorities
  - power/thermal "coarse tuning"

Legend
- SW Interface (API)
- SW/HW Meta-data
Introduction of a new modular policy (YaMS) partition available resources (R) on applications (A) considering A priorities and R “residual” availabilities.

- multi-objective optimization supports a set of tunable goals:
  - DONE: performances, overheads, congestion, fairness
  - WIP: stability, robustness, thermal and power

increase overall system value considering discrete and tunable improvements.

- LP theory, MMKP heuristic promote scheduling of some AWMs which improve optimization goals
demote scheduling of others AWMs which degrade solution metrics e.g. stability and robustness.

Scheduling Policy YaMS - Scalability

Average scheduling time [ 5 AWM ]

<table>
<thead>
<tr>
<th>Time [ms]</th>
<th>01 HCores</th>
<th>02 HCores</th>
<th>04 HCores</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Speedup:

- +36%
- +54%

Scheduling rate [n. scheduling / s ]

- 16 applications → ~ 300
- 32 applications → 100 .. 130
- 64 applications → 30 .. 45
Reconfigurations are allowed as long as:
- reconfiguration overheads are “reasonable"
  - don’t spend more energy on scheduling than on processing
- applications QoS are not noticeably affected
  - maintain applications re-configuration within reasonable timings

Enforce some control properties

stability
- considering a thread-off between reconfigurations and benefits
- constraint reconfigurations within a tunable power budget
- delay or inhibit anomalous application requirements

robustness
- absorb small modification on resources requirements and availabilities
- resource partitioning and AWMs scheduling based on (tunable) minimal gains
- enforce minimal ratio between reconfiguration overhead vs gain

Scheduling Policy
System-Wide Controller – Overall View

- BBQ Resource Scheduler
- Optimization Goals
- Evaluation Metrics
- BBQ Validation Policy
- BBQ Resource Synchronizer
- Platform Integration Layer
- Propose Schedule
- Resources Availability and Status
- Resources Requirements
- Applications
- Resources

BBQ Validation Policy
- enforce certain control properties
  - energy budget, stability and robustness
- authorize resources synchronization
Scheduling Policy
System-Wide Controller – Inner-Loop “Scheduling”

- Scheduling is done per (application) priority level considering status of available resources
  - i.e. those remaining from the scheduling of previous priority levels
  - considering application resources requirements
    - i.e. defined by AWMs (identified at DSE time) and run-time tunings

- MMKP heuristic, three main steps:
  - Computation of AWMs “mapping value”
    - evaluate the value of an AWM being mapped on certain resources
      - \( \text{value} = f(\text{optimization goals, mapped resources}) \)
  - Order AWMs according their “mapping value”
    - from higher to lower
  - Selection of most promising AWMs
    - select the most promising (i.e. higher value) AWM
      - which fits on available resources

BBQ Validation Policy
- enforce certain control properties
  - energy budget, stability and robustness
- authorize resources synchronization
The BarbequeRTRM Framework

**System-Wide RTRM: Platform Integration**

- **Application-Specific RTRM**
  - Fine grained control on application allocated resources:
    - task ordering
    - virtual processor assignment
    - DVFS
    - application parameters monitoring

- **System-Wide RTRM**
  - Coarse grained control on platform available resources:
    - resource accounting, partitioning and abstraction
    - high-level HW events handling (e.g., critical conditions, faults...)
    - manage applications priorities
    - power/thermal "coarse tuning"

---

Apps with 3 AWM, 3 Clusters => 9 configuration per application
BBQ running on NSJ, 4 CPUs @ 2.5GHz (max)
- Support for generic Linux SMP/NUMA machines
  portable solution, based on Linux Control Group
  
  *CPUs and Memory nodes assignments*
  *Memory amount assignment*
  *CPU bandwidth quota assignment (requires kernel 3.2)*

- Support both resources *monitoring and control*
  pre-configured CGroup to define BBQ controlled resources
  *at Barbeque start, by parsing a pre-configured cgroups*
  … *than Barbeque takes control over these resources*
  
  *tun-time generation of new CGroup to control applications*
  *requires RTLib integration (of course)*

- Working on P2012 integration
  new genration many-core platform from STMicroelectronics
  *first version: 4 cluster with 16 Processing Element each one*

**Synchronization Policy**

**System-Wide Controller – Outer-Loop “Synchronization”**

- *BBQ Resource Scheduler*
  - Optimization Goals
  - Evaluation Metrics
- *BBQ Validation Policy*
  - Resource Partitioning
- *BBQ Resource Synchronizer*
  - Proposed Schedule
- *Platform Integration Layer*
  - Resources
  - Resources Requirements
  - Applications

**BBQ Validation Policy**
- *enforce certain control properties*
  - energy budget, stability and robustness
- *authorize resources synchronization*
Synchronization Policy
System-Wide Controller – Outer-Loop Overheads

min AWM 25% CPU Time, 3 Clusters x 4CPUs => max 48 syncs
BBQ running on NSJ, 4 CPUs @ 2.5GHz (max)

Linux kernel 3.2
Creation overheads: ~500ms
Update overheads: ~100ms
(1/3 on quadcore i7)
Application dependent

The BarbequeRTRM Framework
Power Optimizations

Initial experiments on congested workloads
increasing running instances of Bodytrack (PARSEC)
3AWM: [1,2,4] Threads
system-wide power measurements
via the standard IPMI interface

X86_64 NUMA machine: 3 Clusters x 4CPUs
BBQ running on NSJ, 4 CPUs @ 800MHz
BarbequeRTRM Internals
v0.8 (Betty Bacon) – Last Stable Release (02/2012)

The BarbequeRTRM Internals
System-Wide RTRM: Overall Framework

Application-Specific RTM
Fine grained control on application allocated resources:
- task ordering
- virtual processor assignment
- DVFS
- application parameters monitoring

System-Wide RTRM
Coarse grained control on platform available resources:
- resource accounting, partitioning and abstraction
- high-level HW events handling e.g., critical conditions, faults...
- manage applications priorities
- power/thermal "coarse tuning"

Legend
- SW Interface (API)
- SW/HW Meta-data

BarbequeRTRM
The BarbequeRTRM Internals
Framework Architecture

- Complete support for application development
  RTLib
  *Plain and AEM-based API*
  Applications mgmt
  Resources accounting
  Scheduling policies
  *random (stress test)*
  YaMCA and YaMS
  Synchronization Policy
  SASB
- x86 Platform Integration
  c-group based PIL
- P2012 Platform Integration
  work in progress, first support for OpenCL apps

The BarbequeRTRM Internals
System-Wide RTRM: RTLib details

Application-Specific RTM
Fine grained control on application allocated resources:
- task ordering
- virtual processor assignment
- DVFS
- application parameters monitoring

System-Wide RTRM
Coarse grained control on platform available resources:
- resource accounting, partitioning and abstraction
- high-level HW events handling e.g., critical conditions, faults...
- manage applications priorities
- power/thermal “coarse tuning”
The BarbequeRTRM Internals
Run-Time Library (RTLib)

- Defines the (expected) application behavior loop of actions, until no more workload to process
- Abstract the communication channel using "threaded FIFOs", (WIP) Binder support on Android
- Provides APIs at **three different abstraction levels** Plain API, AEM API and AS-RTM API
- Hides the **Synchronization-Protocol** details

The "Abstract Execution Model" (AEM)

- Targets integration of **stream-processing** applications defines the (required) application behavior
  
  *Setup, GetAWM, [Reconfigure|Suspend], Run, Monitor*
  
- Support enforced reconfiguration (by System-Wide RTRM)
- Configuration assisted by a Synchronization Protocol ensure proper reconfiguration, collect suitable statistics, completely hided by the RTLib
The BarbequeRTRM Framework
Run-Time Library (RTLib)
Overall Work-flow and Abstract API

- Internal defined state-machine
  - granted state management
  - complexity hiding
- Call-back based user API
  - simple apps integration
- Monitoring support
  - transparent data collection
  - targeting Design/Run-Time

Run-Time Library (RTLib)
Linux Performance Counters Support (1/3)

- RTLib instrumentation for “perf” support
  - standard Linux interface to access performance counters

- HW
- SW
- OS
- Caches
- Power
Run-Time Library (RTLib)
Linux Performance Counters Support (2/3)

- Allows AWM-specific monitoring
  “for-free”, for each application integrated with the RTLib
  could support integration with DSE techniques
  negligible run-time overheads
  could support BBQ run-time monitoring and AWM tuning
- Simple usage by exec environment configuration
  BBQ_RTLIB_OPTS="p" <your_application>
- Many flags to customize the reporting
  detail level (simple, +D$, +I$, +Bus cycles)
  kernel overheads
  RTLib overheads
  [output format (text vs CSV)]

Run-Time Library (RTLib)
Linux Performance Counters Support (3/3)

- Example: Bodytrack (PARSECv2.1), 8 vs 4 threads

Running with 4 thread is “more efficient”
The BarbequeRTRM Internals

Platform Integration

- Complete support for platform integration
generic Linux SMP machine
portable solution, based on Linux Control Group

  **CPU and Memory nodes assignments**
  **CPU quota** (work-in-progress, requires kernel 3.2)

- Support both resources monitoring and control
  pre-configured CGroup to define BBQ controlled resources
  *at BarbequeRTRM start, by parsing a pre-defined cgroup*
  *... than Barbeque takes control over these resources*
tun-time generation of new CGroup to control applications
  requires RTLlib integration (of course)

- Initial support for STHORM platform integration
  based on shared-memory and firmware hosted controller
  low-overheads for both configuration and monitoring
The BarbequeRTRM Framework

System-Wide RTRM: Applications Integration

- Resource accounting, partitioning, and abstraction
- High-level HW events handling, e.g., critical conditions, faults...
- Manage applications priorities
- Power/thermal "coarse tuning"

Application-Specific RTRM

- Fine-grained control on application allocated resources:
  - Task ordering
  - Virtual processor assignment
  - DVFS
  - Application parameters monitoring

Legend
- SW Interface (API)
- SW/HW Meta-data

Applications Integration
MOST DSE Tool integration

Metrics API
Applications Integration
The Recipe file

- Description of application requirements
  - System priority,
  - Application Working Modes (AWMs),
  - Resources requirements (platform specific)

- Deployed at application installation time into global system folder

- XML Schema possible merge with Android manifest
Applications Integration
Integration Workflow (S-W vs A-S RTM)

Applications Integration
Application-Specific API – Overall View

Application Working Modes vs Operating Points

System Wide resources assignment

AWM1

AWM2

AWMn

Application Specific tuning parameters

AS-RTM

OP-Manager

Goals

OPFilter

Monitors
Applications Integration

Integration of “interesting/real” Applications (1/2)

- Partial integration of PARSEC v2.1 benchmarks
  Bodytrack and Ferret (pthreads version) already done
  two different examples: data-parallelism vs pipelining

- Example of the required effort:
  code re-arrangement to support run-time re-configurations
  
  Because this application is not designed for Run-Time management

RTLib integration (mostly copy-and-paste code)
Applications Integration
Integration of “interesting/real” Applications (2/2)

- 2PARMA reference workloads
  Multiview, OpenCL on x86, OpenCL-P2012 WIP
  Scalable Video Coding (SVC), both x86 and NPM-P2012

- Internal developed applications
  OCVDemo, OpenCV+TBB on x86
  http://youtu.be/4DlpqY8F6SY

- Overall considerations
  most of the effort spent on developing run-time tunable applications
  or adapting applications not designed to be run-time tunable
  RTLib integration is straight-forward
  almost only copy-and-paste code

The BarbequeRTRM Open Source Project
The on-line BarbequeRTRM community
Based on (a customization of) Android building system freely available for download and (automatized) building

Framework dependencies
- External libs, tools, ...

Framework Sources
- BarbequeRTRM, RTLib

Framework Tools
- PyGrill (loggrapher), ...

Contributions
- Tutorials, demo

Public GIT repository
https://bitbucket.org/bosp

Integration with Kconfig
- simpler configuration of BBQ compilation/installation options
**Demo video and Practical Overview**

*BarbequeRTRM v0.8 (Betty Bacon)*

---

**Outline of the Demo Video**

**MP Tasks**

**HP Tasks**

**BarbequeRTRM**

**Demo Test Cases**

---

*Live demo... (or visit http://youtu.be/Hcz1ob23WWA)*
The BarbequeRTRM Framework

Outline of the Practical Session

- BOSP source tree
  File-system structure
- Building the framework
- Configuring the framework
  RTRM options and (x86) platform description
- Using the framework
  Exploiting the BOSP shell
  Running a simple example

- Case study: Bodytrack integration (PARSEC v2.1)
  A simple recipe
  Adding RTRM support
  RTLib integration

The BarbequeRTRM Framework
Conclusions and Future Works
**The BarbequeRTRM Framework**

### Conclusions

- **Framework for System-Wide RTRM**
  - **flexibility and scalability** of the RTRM strategy
    - thanks to its hierarchical and distributed control structure
  - **acceptable overheads** for real usage scenarios
    - including those with variable workload
  - tunable **multi-objective optimization** policies
    - to cope with several design constraints and goals
      - e.g., performance, power, thermal and reliability, ...
  - promising results in terms of **performance improving**
  - and **power consumption reduction**
    - for a highly parallel workload, on a NUMA multi-core architecture

- availability of a simple API interface
  - making straightforward for the programmers to take full advantages from framework services

### Future Works

- **Optimization policies**
  - robustness and stability assessment
  - improving power/energy optimizations
  - thermal and reliability management

- **Platform integration**
  - extended support for Android targets
  - complete the integration with the P2012 many-core platform

- **2PARMA reference workloads integration**
  - two workloads: MultiView and SVC Decoder
  - two use-cases: x86-NUMA and STHORM Platform
  - additional R&S assessment
Because of its “sweet analogy” with something everyone knows...

**QoS**
how good is the grill

**Applications**
the stuff to cook

**Overheads**
Cook fast and light

**Priority**
how thick is the meat or how much you are hungry

**Task mapping**
the chef’s secret

**Mixed Workload**
sausages, steaks, chops and vegetables

**Reliability Issues**
dropping the flesh

**Thermal Issues**
burning the flesh

**Policy**
the cooking recipe

**Resources**
coals and grill

*Thanks for your attention!*