BBQ - The Barbeque RTRM
System-Wide run-time resource management for multi-many cores in the 2Parma Projects

Speaker and Project Technical Manager: Prof. William Fornaciari
fornacia@elet.polimi.it

Chief Developer: PhD Patrick Bellasi
bellasi@elet.polimi.it

Developer and Integrator: G. Massari

Politecnico di Milano
Dipartimento di Elettronica e Informazione

Last review: Apr, 19 2012
Introduction to RTRM

*Overall View on Goals, Requirements and Design*
Find the optimal trade-off between QoS requirements and resources availability

- Shared HW resources, mixed SW workloads
  upcoming many-core devices are complex systems
  process variations and run-time issues
  resource sharing and competition
- Simple SW solutions are required
  support frequently changing use-case
  suitable for both critical and best-effort applications
The RTRM Tool
Objectives of RunTime 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
Support monitoring, management and control at different granularity levels to reduce overheads

Different granularity

- accelerared application
- operating system
- computation fabric
- computation clusters

How to reduce the control complexity?

Each granularity level collects requirements from higher level and it provides constraints to lower levels
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
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
### RTRM Imposed Requirements

**Resources Related Requirements**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RI.1</td>
<td><strong>definition of resources working-modes</strong></td>
</tr>
<tr>
<td>RI.2</td>
<td><strong>definition of resources control points</strong></td>
</tr>
<tr>
<td>RI.3</td>
<td><strong>resources observability</strong></td>
</tr>
</tbody>
</table>

1. **RI.1 definition of resources working-modes**
   - Any component in a use-case which represent a resource must completely define its *working modes* and notify them to this tool.

2. **RI.2 definition of resources control points**
   - Subsystems must expose their *control points* and define how modification impact on their behaviors (e.g. power and performances)

3. **RI.3 resources observability**
   - Subsystem are expected to expose some *observability points*, represented by metrics that can be used to identity their behaviors.

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

#### Name

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RI.4</td>
<td>priority based access to resources</td>
</tr>
<tr>
<td>RI.5</td>
<td>user-space representing application</td>
</tr>
<tr>
<td>RI.6</td>
<td>query resource availability</td>
</tr>
<tr>
<td>RI.7</td>
<td>get and release resources</td>
</tr>
<tr>
<td>RI.8</td>
<td>handle RTPM notifications</td>
</tr>
</tbody>
</table>

#### Description

- **RI.4** priority based access to resources: Critical applications could **preempt** resources used by best-effort applications.
- **RI.5** user-space representing application: Every RTRM client tool has a corresponding **controlling user-space application** which defines its resource access rights and priority.
- **RI.6** query resource availability: Applications (best-effort) are required to **query** the RTPM tool to know **about resources availability**.
- **RI.7** get and release resources: Each tool user is required to **notify** when a resource is used and released.
- **RI.8** handle RTPM notifications: Applications (best-effort) could be requested to **adapt to resource availability changing conditions**.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC.1 monitor resource</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>performance</td>
<td></td>
</tr>
<tr>
<td>RC.2 dynamic resource</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>partitioning</td>
<td></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</td>
<td>System-wide and multiple metrics optimization policy.</td>
</tr>
<tr>
<td>optimization policy</td>
<td></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</td>
<td>Reduced impact on the performances of the controlled system.</td>
</tr>
<tr>
<td>overheads</td>
<td></td>
</tr>
</tbody>
</table>
RTRM Covered Requirements
The Proposed Run-Time Control Solution

Run-Time (exploitation)
- Application Working Modes
- System-Wide Optimization Goals
- Resources Partitioning
- Overheads Control

Design-Time (exploration)
- DSE

System-Wide Control Level
- Operating Points
- Applicaton Specific RTM
- Applications
- Resources
- OS/FW

Application Control Level
- OP Selection
- QoE
- Tuning
- Status

OS/FW Control Level

application requirements and resources availability
Hierarchical control
- System-wide level (e.g., resource partitioning)
- Application specific (tuning of application parameters, DMM)
- Firmware/OS level (frequency selection, thermal alarm, resource availability, adaptive voltage scaling, ...)

Closed control loop enhanced with feed forward
- OP and AWM increase convergence speed

Enforce observability and controllability
- The control algorithm is run-time computed
- Rich set of sensing and control capabilities, both for applications and platforms
- Bounded and measurable response time

Distributed control
- Several subsystems have their own control loop

Optimal
- User defined goal functions (including overheads)

Robust
- Can adapt to the model characteristics

Adaptive
- The control algorithm is run-time computed
The Barbeque Approach to RTRM

*an overall view on proposed tool architecture*
- Complete support for application development
  - RTLlib
    - *Plain and AEM-based API*
  - Applications mgmt
  - Resource accounting
  - Scheduling policies
    - *randon (stress test)*
    - *YaMCA*
  - Synchronization Policy
    - *SASB*
- Platform integration
  - CGroup based (x86)
  - Memory Mapped (P2012)
System-Wide RTRM
Main Achievements Since Last Review

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 H/W events handling e.g., critical conditions, faults...
- manage applications priorities
- power/threrral "coarse tuning"

Barbeque provided code/modules

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

POLITECNICO DI MILANO
- Defines the (expected) application behavior
  Loop of actions, until no more workload to process
  *GetWorkingMode, [Reconfigure|Suspend], Running, Monitor*

- Abstract the communication channel
  *Right now using “threaded FIFOs”, planned Binder/DBus support*

- Hide Synchronization-Protocol details

- Provide also an AEB-based abstract API
### Barbeque RTRM

**RTLib – Linux Performance Counters Support (1/3)**

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

#### HW
- cpu-cycles OR cycles
- stalled cycles-frontend OR idle-cycles-frontend
- stalled cycles-backend OR idle-cycles-backend
- instructions
- cache-references
- cache-misses
- branch-instructions OR branches
- branch-misses
- bus-cycles
- cpu-clock
- task-clock
- page-faults OR faults
- minor-faults
- major-faults
- context-switches OR cs
- cpu-migrations OR migrations
- alignment-faults
- emulation-faults

#### SW
- sched: sched_kthread_stop
- sched: sched_kthread_stop_ret
- sched: sched_wakeup
- sched: sched_wakeup_new
- sched: sched_switch
- sched: sched_migrate_task
- sched: sched_process_free
- sched: sched_process_exit
- sched: sched_wait_task
- sched: sched_process_wait
- sched: sched_process_fork
- sched: sched_stat_wait
- sched: sched_stat_sleep
- sched: sched_stat_ioawait
- sched: sched_stat_runtime
- sched: sched_pid_setpriority

#### OS
- L1-dcache-loads
- L1-dcache-load-misses
- L1-dcache-stores
- L1-dcache-store-misses
- L1-dcache-prefetches
- L1-dcache-prefetch-misses
- L1-icache-loads
- L1-icache-load-misses
- L1-icache-stores
- L1-icache-store-misses
- L1-icache-prefetches
- L1-icache-prefetch-misses
- LLC-loads
- LLC-load-misses
- LLC-stores
- LLC-store-misses
- LLC-prefetches
- LLC-prefetch-misses
- dTLB-loads
- dTLB-load-misses
- dTLB-stores
- dTLB-store-misses
- dTLB-prefetches
- dTLB-prefetch-misses
- sTLB-loads
- sTLB-load-misses
- branch-loads
- branch-stores
- branch-load-misses
- node-loads

#### Caches
- [Hardware cache event]
- [Software cache event]

#### Power
- [Tracepoint event]
- [Tracepoint event]
Example: Bodytrack with 8 or 4 threads

Running with 4 thread is "more efficient"

Performance counters

Derived metrics

Statistics (stddev/avg)
- 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
  `BBQUE_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)]
- Started the 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)
BBQ Validation Policy
- enforce certain control properties (energy budget, stability and robustness)
- authorize resources synchronization
Introduction of a new modular policy (YaMS)

multi-objective optimization

supports a set of tunable goals

DONE: performances, overheads, congestion, fairness

WIP: stability, robustness, thermal and power

increases overall system value

considering discrete and tunable improvements

partition of available resources (R) on applications (A)

considering A priorities
considering R “residual” availabilities

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
Reconfigurations are allowed as long as:
- reconfiguration overheads are “reasonable”
  - don't spent more energy on scheduling than on processing
- applications QoS are not noticeably affected
  - maintain applications re-configurations 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 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
Average scheduling time [5 AWM]

- 34 ms
- 25 ms
- 22 ms

10 → 7 ms
3.5 → 3 ms

**Speedup**
+36%
+54%

**Scheduling rate [n. scheduling / s]**

- 16 applications → ~300
- 32 applications → 100..130
- 64 applications → 30..45
Apps with 3 AWM, 3 Clusters => 9 configurations per application
BBQ running on NSJ, 4 CPUs @ 2.5GHz (max)
- Initial 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
  run-time generation of new CGroup to control applications
  *requires RTLib integration (of course)*

- Started the P2012 integration
  initial agreement on control and monitoring interface
  advanced run-time control features for OpenCL apps
BBQ Validation Policy
- enforce certain control properties
  - energy budget, stability and robustness
- authorize resources synchronization
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**

**CGroups PIL**
Barbeque RTRM
Conclusions and Future Plans

- **Current status**
  
  Initial working implementation of x86-NUMA PIL
  Improved multi-objective resource partitioning policy
  *overheads and scalability measurements in backup slides*

- **Future works**
  
  P2012 Platform Integration Layer
  YaMS extension
  *to enforce Robustness and Stability*
  *to cover Power/Energy Optimization*
  *to target thermal management*
  2PARMA reference workload integration
Dynamic CGroups management is possible using per-application CGroups created on-the-fly at application start tuned at run-time based on resources assignment.

CGroups could also be used to define managed resources.

CGroup creation and management has a non-negligible overhead measured available on backup slides this could impact on burst application starts requires proper implementations to preserve critical code paths.

Run-Time CGroups tuning requires synchronization not synced apps could crash on CPU count changes e.g. TBB optimized loops.
Real stream processing workloads could be integrated into the “Abstract Execution Model” (AEM), exposed by the RTLib, is suitable to map real applications examples of pthread and Intel TBB based workloads 

_i.e. Bodytrack and Ferret (PARSEC v2.1), CANNY, FAST, SURF (OpenCV)_

Most of the required effort spent to make existing applications run-time tunable 

integrated applications could be used as templates 

*based on thread-pools with tunable ticketing mechanism*

RTLib integration is almost only “copy&past” or “ifdef” existing code should be placed into required AEM callbacks

Application-Specific Run-Time Management (AS-RTM) policies should be developed to **monitor** application run-time behaviors and **issue requirements** to SW-RTRM
Run-time application behaviors could exhibit non linear performance/efficiency profiles
   performance counters could be on hand to support DSE
   performance counters could support AS-RTM policies

Application-Specific RTM policies are required
   but just for optimal application tuning
   best-effort integration achieved by just linking the RTLib

The RTLib could provide a valuable support
   for-free performance counters support
      AWM-specific profiling
   for-free DSE integration (WIP)
Barbeque RTRM
v0.8 (Betty Bacon)
Grant resources to **critical workloads** while optimize resource usage by **best-effort workloads**

- Considering mixed-workload scenarios

  critical workload (High Priority)

  **Application**: SVCDemo

  app using SVC Video Decoding library by HHI (x86_32 version)

  other workloads runs concurrently (Lower Priority)

  **Applications**: 1 SVCDemo + 3 BbqRTLibTestApp

  BbqRTLibTestApp as background “disturbing” app

- **Goals**: **application integration** and **dynamic resource partitioning**

  according to apps priority and resources demand

  synthetic recipes (no DSE based), mechanisms evaluation (behaviors, overheads, stability,...)

  **YaMS policies and CGroup based platform integration**
Demo of the Barbeque RTRM
Demo 1 – Applications Priority Management

Resources Scheduling

Live demo available...
http://youtu.be/gUXY98wFkFM
... while other apps adapted by scaling down their resolution in order to meet the FPS goal with the reduced resources (12%) assigned ;-) 

More resources (50%) have been assigned to the demanding application... 

Live Demo Available... 
http://youtu.be/4DIPqY8F6SY
Based on (a customization of) Android building system freely available for download and (automatized) building

https://bitbucket.org/bosp/barbeque/wiki/Getting_the_Barbeque_Sources

Framework dependencies
External libs, tools, ...

Framework Sources
BarbequeRTRM, RTLib

Framework Tools
PyGrill (loggrapher), ...

Contributions
Tutorials, demo
Because of its “sweet analogy” with something everyone know...

- **QoS**: how good is the grill
- **Applications**: the stuff to cook
- **Overheads**: Cook fast and light
- **Task mapping**: the chef’s secret
- **Priority**: how thick is the meat or how much you are hungry
- **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

---

**BBQ - The Barbeque RTRM**

41