The BarbequeRTRM Framework
Targeting Applications and Platform “Variability” Challenges

Prof. William Fornaciari
william.fornaciari@polimi.it

Politecnico di Milano - DEIB
Dipartimento di Elettronica, Informazione e Bioingegneria
Milano, Italy
Platforms Evolution

Some big: good... many small: better!
Outline of the keynote

- Introduction: towards *multi-problem* and multi-core Challenges for new generation of applications
- Effective and flexible exploitation of new platform capabilities
  - Adaptability
- The BBQ RunTime Resource Management approach
  - Tradeoff, achievements, the BOSP open source project
- Screen-cast of use cases
  - BBQ in action
- Work in progress
  - Roadmap and new FP7 projects
From single-core to multi-core processors

- Consider frequency control at SoC and cluster granularity
- Introduce PVT (Process, Voltage, Temperature) sensors
- Joint design of firmware and OS layers
Platforms Evolution
Which New Architectures We Have to Target?

- From ad-hoc HW... to generic HW
Example: Software Defined Radios (SDR)

Re-Programmable approach

Classical approach
- Support for **parallel code development**
- Foster **reusable software components**
  
  independent and parallelized SW modules (filters)
  
  well defined interfaces to support composition (pipelines)

- **New programming paradigms**
  
  to better support parallelized modules development
  
  not binded to a specific target
  
  “write one run anywhere”

- **Usable development environments**
  
  high level of abstraction design of applications
  
  target specific simulation and optimization support
  
  support for multiple programming models
Platforms Evolution
Which Programming Paradigms?

- Proprietary and/or platform specific
  Fractal
    defined by OW2 Consortium
    modular and extensible middleware
    language agnostic (e.g. C, Java, .NET)

Native Programming Model
  defined by STMicroelectronics
  collection of primitives
  to support decomposition

Thread Building Blocks (TBB)
  defined by Intel
  mostly targeting HPC
  supporting just x86
Platforms Evolution
Which Standards?

- OpenCL: “the” industrial standard

- OpenVX: the upcoming standard which introduces the concept of “task manager”
- Same principle used when playing with LEGOss

“collect, put together”

from Danish “leg godt” = “play well”
- Embedded is moving towards many-core architectures
  - Many similar computing elements
  - Complex applications are decomposed in parallel modules
- Device functionality is polymorphic
  - Depends on the programming
  - Can change at run-time adapting to the new scenario
- Resemble the HPC style
  - See last FP7 calls...

Barcelona Supercomputing Centre

10,240 processors

It's just a change of “scale factor”

Tilera Tile-Gx100
100 independent cores

Same benefits
but “programmable”
Introduction to RTRM
overall view on goals, requirements and design

BBQ
The BarbequeRTRM Framework
Computing platforms convergence

*targeting both HPC and high-end embedded and mobile systems*

parallelism level ranging from few to hundreds of PEs

thanks to silicon technology progresses

Emerging new set of non-functional constraints

*thermal* management, system *reliability* and fault-*tolerance*

area and power are typical design issues

*embedded systems are loosing exclusiveness*

*effective resource management policies required to properly exploit modern computing platforms*
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
Many-core platforms enable a new set of applications. Computer vision is just one of the main interesting.

Multi-functional embedded devices are widespread. Concurrently running applications differ in criticality and time-varying requirements.
- 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
Different approaches targeting resources allocation

- Linux scheduler extensions
  mostly based on adding new scheduler classes\cite{2,4,7}
  \textit{force the adoption of a customized kernel}

Virtualization

\textit{Hypervisor acting as a global system manager}

Both commercial and open source solutions

- Commercial: e.g. OpenVZ, VServer, Montavista Linux;
- Open: e.g. KVM, Linux Containers

\textit{require HW support on the target system}

User-space approaches

\textit{more portable solutions}\cite{3,6,11}

\textit{mostly limited to CPU assignment}

\begin{itemize}
  \item \cite{2} Bini et. al., \textit{Resource management on multicore systems: The actors approach"}, Micro 2011.
  \item \cite{3} Blagodurov and Fedorova, \textit{“User-level scheduling on numa multicore systems under linux”}, Linux Symposium 2011.
  \item \cite{4} Fu and Wang, \textit{“Utilization-controlled task consolidation for power optimization in multi-core real-time systems”}. RTCSA 2011.
  \item \cite{6} Hofmeyr et. al., \textit{“Load balancing on speed”}. Ppopp 2010.
  \item \cite{7} Li et. al., \textit{“Efficient operating system scheduling for performance-asymmetric multi-core architectures”}. SC 2007.
  \item \cite{11} Sondag and Rajan, \textit{“Phase-based tuning for better utilization of performance-asymmetric multicore processors”}. CGO 2011.
\end{itemize}
Different approaches targeting resources allocation

- Linux scheduler extensions
- More dynamic usage of Linux Control Groups to manage multiple resources with a portable and modular RTRM running in user-space
- Hypervisor acting as a global system manager
- Both commercial and open source solutions
  Commercial: e.g. OpenVZ, VServer, Montavista Linux; Open: e.g. KVM, Linux Containers
  require HW support on the target system

User-space approaches
  more portable solutions \[3,6,11\]

mostly limited to CPU assignment

The Barbeque Approach to RTRM
an overall view on proposed tool architecture
The BarbequeRTRM

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

http://www.2parma.eu

http://bosp.dei.polimi.it
The BarbequeRTRM
A Bird Eye View on the Proposed Approach

- Track run-time variabilities
  application requirements
  resources availabilities

- Overhead contingency
  design-time profiling
  run-time optimization

- Support different granularity
  system-wide optimization
  application-specific tuning

- Integrated work-flow
  single framework to support
  both design-time and run-time
The BarbequeRTRM
Overall View on Run-Time Resource Management

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"

BarbequeRTRM

**The BarbequeRTRM**  
Overall View on Run-Time Resource Management

- **Applications**
- **Requirements**
- **Optimization Policy**
- **Configure**
- **Notify**

**Monitor and Security**  
**Business Intelligence**

---

The BarbequeRTRM
Example: Multi-Core NUMA Platforms

Congested workloads

Critical Apps
Best-Effort Apps

RTLib
RTLib

Res Accounting
Res Partitioning

Res Abstraction

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

CGroups

Extend advanced and efficient resources control capability offered by modern Linux Kernels
with suitable resources partitioning policies
running in user-space

CGroups based resources abstraction layer
The BarbequeRTRM
Example: Many-Core STHorm Platform

Congested workloads

Critical Apps

Best-Effort Apps

RTLib

RTLib

Res Accounting

Res Partitioning

Res Abstraction

P2012 PIL

Extend SThorm resident run-time scheduler capability offered by current p12runtime with suitable resources partitioning policies

Managed by a user-space daemon
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**
- Introduction of a new modular policy (YaMS)
  partition available resources (R) on applications (A)
  considering A priorities and R “residual” availabilities
  multi-objective optimization
    support 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
System-Wide Controller – Overall View

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

Resources Availability and Status
Resources Requirements

BBQ Resource Scheduler

Optimization Goals
Evaluation Metrics

BBQ Validation Policy

Resource Partitioning

Scheduling Time vs EXCs (BbqRTrLib/TestApp)

Average scheduling time [ 5 AWM ]
34 ms
25 ms
22 ms
10 → 7 ms
3.5 → 3 ms

Scheduling rate [ n. scheduling / s ]
16 applications → 300
32 applications → 100..130
64 applications → 30..45

Speedup
+36%
+54%
Scheduling Policy
System-Wide Controller – Scalable and “Fast Response”

Linux kernel 3.2
Creation overheads: ~500ms
Update overheads: ~100ms
(1/3 cn quad core i7)

SyncP vs EXCs (BbqRTLlibTestApp)

Time [ms]
1.0 × 10^3
7.5 × 10^2
5.0 × 10^2
2.5 × 10^2
0.5 × 10^2
0

EXCs Count

System power consumption

Power [W]
305
300
295
290
285

Number of ‘bodytrack’ running instances

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

The BarbequeRTRM Framework
Scheduling Policy
System-Wide Controller – Grant Stability and Robustness
Run-time **reconfigurable** workloads
e.g. Scalable Video Coding (SVC)

*single input stream, different decoding configurations*

**Temporal Scalability**
- 30 fps
- 15 fps
- 7.5 fps

**Spatial Scalability**
- QCIF
- CIF
- TV

**Quality Scalability**

*Different decoding profiles which correspond to different quality-vs-performances requirements*
Stream processing applications which means not only multimedia processing e.g. packet sniffing and analysis, pattern matching, ...

Well defined **Abstract Execution Model (AEM)** loop of actions, until no more workload to process Setup, Configure, Running, Monitor
- 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
AEM Abstract API

- **callbacks** based with default implementations

- hide all the RTM boilerplate code
The Barbeque Open Source Project (BOSP)

- 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
We cannot cover internal details
please check project website and past presentations


Complete Framework Review + Hands On Sessions

Results on Multi-Core NUMA machine


Official Project Website

http://bosp.dei.polimi.it
The BarbequeRTRM v0.8 (Betty Bacon)
Outline of the Demo Video

MP Tasks

HP Tasks

BBQ Manged Resources

2PARMA Projects
Second Review

BarbequeRTRM

Demo Test Cases

Live demo...
(or visit http://youtu.be/Hcz1ob23WWA)
The BarbequeRTRM Framework
Conclusions and Future Works
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
Wide spectrum of activities
covering different abstraction level
Under negotiation in FP7

*Strep – run time for reliability and QoS guaranteed. HPC and ES synergy*

*IP – mixed criticalities, WSN+cloud*

Prof. William FORNACIARI

*home.deib.polimi.it/fornacia*

*william.fornaciari@polimi.it*

The POLIMI team

*BBQ Fmk: P.Bellasi, G.Massari*

*Metrics for thermal: D.Zoni, S.Corbetta, F.Terraneo*

*New runtime directions: L.Rucco, C.Caffarri, C.Brandolese*

*DSE: C.Silvano, G.Palermo, V.Zaccaria, E.Paone*

*Progr. Paradigms: G.Agosta, Scandale*
Backup Slides
Experimental Setup
Hardware Platform and Workloads

- Workloads: increasing number of concurrently running applications
  
  Bodytrack (BT) (PARSEC v2.1)
  modified to be run-time tunable and integrated with the BarbequeRTRM

- Platform: Quad-Core AMD Opteron 8378
  
  4 core host partition, 3x4 CPUs accelerator partition
  running up to 2.8GHz, 16 Processing Elements (PE)
  CPUFreq and its on-demand policy

Goal: assess framework capability to efficiently manage resources on increasingly congested workload scenarios
• Compare Bodytrack original vs integrated version using same maximum amount of thread. The BBQ Managed version could reduce this number at Run-Time.

• Original version controlled by Linux scheduler, integrated version managed by BarbequeRTRM.

• Performances profiling using standard frameworks.

IPMI Interface for system-wide power consumption [W]

Using Linux *perf* framework to collect HW/SW performance counter.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTIME</td>
<td>Time [s] - Workload completion time [s]</td>
</tr>
<tr>
<td>POWER</td>
<td>Power [W] - System power consumption [W]</td>
</tr>
<tr>
<td>TASK-CLOCK</td>
<td>Ticks - Task clock ticks</td>
</tr>
<tr>
<td>CTX</td>
<td>Context-Switches - Total number of context switches</td>
</tr>
<tr>
<td>MIG</td>
<td>Migrations - Total number of CPU migrations</td>
</tr>
<tr>
<td>PF</td>
<td>Page-Faults - Total number of page faults</td>
</tr>
<tr>
<td>CYCLES</td>
<td>Cycles - Total number of CPU cycles</td>
</tr>
<tr>
<td>FES</td>
<td>Front-End Stalls - Total number of front-end stalled-cycles</td>
</tr>
<tr>
<td>FEI</td>
<td>Front-End Idles - Total number of front-end idle-cycles</td>
</tr>
<tr>
<td>BES</td>
<td>Back-End Stalls - Total number of back-end stalled-cycles</td>
</tr>
<tr>
<td>BEI</td>
<td>Back-End Idles - Total number of back-end idle-cycles</td>
</tr>
<tr>
<td>INS</td>
<td>Instructions - Total number of executed instructions</td>
</tr>
<tr>
<td>SPC</td>
<td>SPC - Effective Stalled-Cycles-per-Instruction</td>
</tr>
<tr>
<td>B</td>
<td>Branches - Total number of branches</td>
</tr>
<tr>
<td>B-RATE</td>
<td>Branches-Rate - Effective rate of branch instructions</td>
</tr>
<tr>
<td>B-MISS</td>
<td>Branch-miss - Total number of missed branches</td>
</tr>
<tr>
<td>B-MISS-RATE</td>
<td>Branch-miss Quota - Effective percentage of missed branches</td>
</tr>
<tr>
<td>GHZ</td>
<td>GHz - Effective processor speed</td>
</tr>
<tr>
<td>CPU-USED</td>
<td>CPUs utilized - CPUs utilization</td>
</tr>
<tr>
<td>IPC</td>
<td>IPC - Effective Instructions per Cycles</td>
</tr>
</tbody>
</table>

(*) The lower the better, for all metrics but the IPC.
Results
Workload Burst Performance Comparison

Completion Time
- BBQ managed apps pinned to assigned CPUs

CPU Migrations
- improved code execution efficiency

CTX Switches

Power [W]
- IPC: 1.080 => 1.235

1 Thread

High System congestion

8 Threads

BBQ partially serialize the execution of concurrent workloads

Reduced OS overhead
- Improved code efficiency
  > x1.3 faster

Up to x6 more energy efficient

Improvements [%] - BBQ Manged vs Unmanaged

<table>
<thead>
<tr>
<th>Scenario</th>
<th>ctime [%Δ]</th>
<th>power [%Δ]</th>
<th>energy [%Δ]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A 1 Thread - 1 Instance</td>
<td>80</td>
<td>0.2</td>
<td>16</td>
</tr>
<tr>
<td>B 1 Thread - 12 Instances</td>
<td>84</td>
<td>7.8</td>
<td>665</td>
</tr>
<tr>
<td>C 8 Thread - 1 Instance</td>
<td>35</td>
<td>9.7</td>
<td>339</td>
</tr>
<tr>
<td>D 8 Thread - 12 Instances</td>
<td>84</td>
<td>7.2</td>
<td>604</td>
</tr>
</tbody>
</table>

Statistics based on: 30 runs, 99% confidence interval

BBQ
The BarbequeRTRM Framework
Results
Benefits and Loss Comparison

Normalized speedups for all collected performance counters

Same order of magnitude for “migrations” on lower congestions

“page faults” and “branch rate” always degraded because of code organization for BBQ integration

loop-unrolling could not be applied, but... an improved integration has already been identified

Instruction stream optimization could be achieved by trying compile time optimization with effective resources assignment

positive bar corresponds to an improvement while a negative bar represents a deficiency of the managed application with respect of the original one

BBQ
The BarbequeRTRM Framework
Support monitoring, management and control at different granularity levels to reduce overheads

Different granularity
- accelerated 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
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
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
- **Environment**: the chef’s secret
- **Reliability Issues**: dropping the flesh
- **Policy**: the cooking recipe
- **Priority**: how thick is the meat or how much you are hungry
- **Mixed Workload**: sausages, steaks, chops and vegetables
- **Thermal Issues**: burning the flesh
- **Resources**: coals and grill

**The BarbequeRTRM Framework**

- Why such a name?!?