Run-time management of multi-core architectures using the BBQ framework

Targeting Applications and Platform “Variability” Challenges

Speaker 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!
Platforms Evolution
Power Trend: The 2004 Inflection Point

From single-core to multi-core processors

Platforms Evolution
Which New Architectures We Have to Target?

- From ad-hoc HW... to generic HW

- Specialized Accelerator
  - Multi-Core Processor
  - Memory
  - Accelerator
  - GPU

- Programmable Accelerator
  - Multi-Core Processor
  - Computation Fabric
  - Memory
  - GPU

- Computing Fabric

- Multi-Core Computing Cluster

- HW Synchronizer, CDMA, D&TU, T&MU, E&VU
  - Core
  - L1
  - L2
  - Core
  - L1
  - L1
  - Core
  - L1
  - L1
  - Core
  - L1
  - L1

April 30, 2014
Platforms Evolution
How They can Be Exploited?

Example: Software Defined Radios (SDR)
Platforms Evolution
What Are the Software Challenges?

- 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

http://fractal.ow2.org
http://threadingbuildingblocks.org
Platforms Evolution
Which Standards?

- OpenCL: “the” industrial standard

- OpenVX: the upcoming standard which introduces the concept of “task manager”
Platforms Evolution
How To Exploit Many-Cores?

- Same principle used when playing with LEGO$	ext{®}$s

  - “collect, put together”
  - from Danish “leg godt” = “play well”
Platforms Evolution
We Have a Data-Center in the Pocket

- Embedded is moving towards many-core architectures
  Many computing units rather similar
  Complex applications decomposed in parallelizable modules

- Functionality of the device is polymorphic
  Depends on the way of programming
  Can change at run-time to adapt to the usage scenarios

Very similar to what happen in HPC

![Barcelona Supercomputing Centre Image]

Same benefits but “programmable”

![Tile-Gx100 Image]

It's just a change of “scale factor”

10.240 processors

Tilera Tile-Gx100
100 independent cores

April 30, 2014
Introduction to RTRM
overall view on goals, requirements and design
Introduction to RTRM
Why Run-Time Resources Management?

- 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
Introduction to RTRM
What is Run-Time Resources Management About?

- Run-Time Resources Management (RTRM) is about finding the optimal trade-off 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
different criticality
time-varying requirements
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**
Introduction to RTRM

How we compare?

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}

\textit{Both commercial and open source solutions}

\begin{quote}
\textit{Commercial: e.g. OpenVZ, VServer, Montavista Linux; Open: e.g. KVM, Linux Containers}
\end{quote}

\begin{quote}
\textit{require HW support on the target system}
\end{quote}

User-space approaches

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

\begin{quote}
\textit{mostly limited to CPU assignment}
\end{quote}

\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}
Introduction to RTRM

How we compare?

Different approaches targeting resources allocation

Linux scheduler extensions

mostly based on adding new scheduler classes [2,4,7]

More dynamic usage of **Linux Control Groups** to manage *multiple resources* with a *portable and modular* RTRM running in user-space

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.harpa-project.eu
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

- Overheads contingency
  design-time profiling
  run-time optimization

- Support different granularities
  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

Legend

The BarbequeRTRM
Overall View on Run-Time Resource Management

Coarse grained control on platform available resources:
- resource accounting, partitioning and abstraction
- high level HW events handling
- manage applications priorities
- power/thermal "coarse tuning"

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

Dynamic Code Generation
Task Mapping
DDM
Critical Apps
Best Effort Apps
RTLib
Res Accounting
Res Partitioning
Res Abstraction
MRAPI
Platform DRV
Platform Proxy

supported platforms
kernel
user space

Business Intelligence
Monitoring and Security

Guide Assistance
Access Control
Notify
Configure
Optimization Policy
Requirements

SThorm Run-Time

The BarbequeRTRM
Example: Multi-Core NUMA Platforms

Congested workloads

Critical Apps
Best-Effort Apps

RTLib
RTLib

Res Accounting
Res Partitioning

Res Abstraction

CGroups

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

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

ChipEx 2014
April 30, 2014
The BarbequeRTRM
Example: Many-Core STHorm Platform

Critical Apps

Best-Effort Apps

RTLib

RTLib

Res Accounting

Res Partitioning

Res Abstraction

P2012 PIL

Congested workloads

Regular Workload

Memory mapped resources abstraction layer

Application-Specific RTM
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"

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

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

Managed by a user-space daemon

April 30, 2014
The Proposed Control Solution
Distributed Hierarchical Control

- 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
Scheduling Policy
YaMS - A modular multi-objective scheduler

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

Resources Availability and Status

Resources Requirements

BBQ Resource Scheduler

Optimization Goals

Evaluation Metrics

BBQ Validation Policy

Resource Partitioning

Proposed Schedule

BBQ Resource Synchronizer

Platform Integration Layer

Applications

Resources

ChipEx 2014

April 30, 2014
Scheduling Policy
System-Wide Controller – Scalable and “Fast Response”

Resources Availability and Status

Resources Requirements

BBQ Resource Scheduler

Proposed Schedule

Evaluation Metrics

Optimization Goals

BBQ Validation Policy

Resource Partitioning

Scheduling Time vs EXCs (BbqRTLlibTestApp)

Time [ms]

EXCs Count

Average scheduling time [ 5 AWM ]

34 ms
25 ms
22 ms

10 → 7 ms
3.5 → 3 ms

0 10 20 30 40

Time [ms]

4 cluster 16 PE

Scheduling rate [n. scheduling / s ]

16 applications → 300
32 applications → 100 .. 130
64 applications → 30 .. 45

Speedup

+36%

+54%

April 30, 2014
Scheduling Policy
System-Wide Controller – Scalable and “Fast Response”

Linux kernel 3.2
Creation overheads: ~500ms
Update overheads: ~100ms
(1/3 on quadcore 7)

Evaluation Metrics
BBQ Validation
Policy
Resource
Partitioning

SyncP vs EXCs (BbqRTLlibTestApp)

EXCs Count

Time [m/s]

1.0 \times 10^4
1.0 \times 10^3
5.0 \times 10^2
2.5 \times 10^2

40
30
20

Completion time

Time [s]

150
100
50
0

2
4
6
8
10
12

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)
Scheduling Policy
System-Wide Controller – Grant Stability and Robustness
Application Integration Layer
System-Wide RTRM: RTLib details

Dynamic Code Generation

Critical Apps
Best-Effort Apps

Res Accounting
Res Partitioning

RTLib
RTLib

Res Abstraction

MRAPI
Platform Proxy

Platform Driver

Platform Firmware

Application-Specific RTM
- coarse 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

ChipEx 2014
April 30, 2014
Run-time **reconfigurable** workloads
e.g. Scalable Video Coding (SVC)

*single input stream, different decoding configurations*

Different **decoding profiles** which corresponds to different **quality-vs-performances** requirements

2PARMA Project Demo - BarbequeRTRM v0.6 (Angus)
Stream processing applications
which means not only multimedia processing
\textit{e.g. packet sniffing and analysis, pattern matching, ...}

Well defined \textbf{Abstract Execution Model (AEM)}
loop of actions, until no more workload to process
\textit{Setup, Configure, Running, Monitor}
Application Integration Layer
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
AEM Abstract API

- **callbacks** based with default implementations
- hide all the RTM boilerplate code
Applications Integration

MOST DSE Tool integration
Conclusions
The Barbeque OpenSource 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
Thanks for your attention!

For more information and contacts for porting BBQ on your architecture

Prof. William Fornaciari
Politecnico di Milano – DEIB
Via Ponzio 34/5 – 20133 Milano, Italy
william.fornaciari@polimi.it
home.deib.polimi.it/fornacia

If you are interested, please check the project website for further information and to keep update with the developments

http://bosp.dei.polimi.it

Carousel demo of BBQ features when running on x86 architectures
https://www.youtube.com/watch?v=276Yo_K1Wag

ChipEx 2014
April 30, 2014