A RTRM proposal for multi/many-core platforms and reconfigurable applications

Patrick Bellasi
DEI - Politecnico di Milano
Via Ponzio, 34/5
Milano, Italy
Email: bellasi@elet.polimi.it

Giuseppe Massari
DEI - Politecnico di Milano
Via Ponzio, 34/5
Milano, Italy
Email: massari@elet.polimi.it

William Fornaciari
DEI - Politecnico di Milano
Via Ponzio, 34/5
Milano, Italy
Email: fornacia@elet.polimi.it

***Abstract*—Emerging multi/many-core architectures, targeting both High Performance Computing (HPC) and mobile devices, increase the interest for self-adaptive systems, where both applications and computational resources could smoothly adapt to the changing of the working conditions. In these scenarios, an efficient Run-Time Resource Manager (RTRM) framework can provide a valuable support to identify the optimal trade-off between the Quality-of-Service (QoS) requirements of the applications and the time varying resources availability.**

This paper introduces a new approach to the development of a system-wide RTRM featuring: a) a hierarchical and distributed control, b) the exploitation of design-time information, c) a rich multi-objective optimization strategy and d) a portable and modular design based on a set of tunable policies. The framework is already available as an Open Source project, targeting a NUMA architecture and a new generation multi/many-core research platform. First tests show benefits for the execution of parallel applications, the scalability of the proposed multi-objective resources partitioning strategy, and the sustainability of the overheads introduced by the framework.

I. Introduction

Modern parallel computing architectures have broadened the boundaries of the High Performance Computing (HPC) world, emerging in the Embedded Systems context too. The computational power provided can be translated into higher Quality-of-Experience (QoE) for the users. Nowadays, mobile systems (e.g., smartphones and tablets) are benefiting from this platforms too, becoming powerful enough to run applications ranging from advanced multimedia to augmented reality, and from web-browsing to 3D gaming. Considering network computing contexts instead, the availability of a large amount of resources allows the deployment of multi-user systems (e.g., cloud computing), where the computational power can be properly partitioned among users, according to the purchased service level.

Furthermore, multi/many-core based System-on-Chip, provide to embedded systems designers more possibilities to implement a system feature as a software task instead of a hardware module, compared to the past. As a result, a clear gain can be obtained in terms of flexibility, costs, and time-to-market. In addition, the intrinsic "redundancy" of these platforms is the base to improve system reliability and fault tolerance. Indeed, there is more space to detect and a handle at run-time process corners of the chip manufacture, by properly redefine the resource assignments. However, exploiting these opportunities is far from being trivial. To get the maximum from highly parallelized architectures requires effort at different levels. Since this is no more the only objective to pursue, the whole picture becomes really complex. For instance, mobile systems usually need of a smarter usage of the computational resources, in order to maximize the battery lifetime. On their side, mission-critical systems can be very sensitive to thermal status variations of the die, requiring dynamic re-allocations of tasks.

Moreover, platforms are growing in complexity, with processing elements grouped into clusters, memories structured into several layers, and communication channels arranged into Network-on-Chip (NoC) and buses. A resource assignment decision cannot disregard these issues, since mapping the same request on a different set of resources can show very different behaviors in terms of performance and non-functional behaviors. All that considered, it is easy to understand how a Run-time Resource Manager (RTRM) plays a key role in modern systems. It should operate as a system-wide arbiter of the resource contention, being aware of the run-time dynamism of the involved actors, namely resources and applications, and taking into account multiple objectives. Indeed, resources can change their status in terms of usage levels and non-functional properties. In turn, the applications may vary in terms of numbers, priorities and Quality-of-Service (QoS) requirements (mixed workload), with poor degrees of predictability. By supporting these dynamic behaviors, the RTRM brings into the system enhanced capabilities of adaptivity and reconfigurability.

In this paper we propose the BarbequeRTRM[1], as a portable and extensible framework for run-time resource management, supporting both homogeneous and heterogeneous platforms. The paper is organized as follows. Section I-A provides a comprehensive review of the current approaches to the run-time management of resources. Section II explains the support to the applications reconfigurability, provided by the proposed framework. Section III describes how the resource partitioning is performed and controlled at run-time. Experimental results are reported in Section IV, where stress tests have been

[1]BOSP: The Barbeque Open Source Project, http://bosp.dei.polimi.it

TABLE I: RTRM properties

Property	Details
HETEROGENEOUS RESOURCES	Target systems with resources of the same type having different capabilities (e.g., CPU+GPU, multiple memory modules,...)
HOMOGENEOUS RESOURCES	Target systems with resources of the same type having the same capabilities (e.g., Multi-Core CPUs)
MULTI-OBJECTIVE SCHEDULING/ ALLOCATION	Scheduling/Resource Allocation based on multi-objective policies
RECONFIGURABILITY/ ADAPTIVITY	Applications can be reconfigured to run according to different resource assignment configurations
MULTIPLE RESOURCES	More than one resource can be required and assigned
HIERARCHICAL RESOURCE MAPPING	It is possible to map the resource requirements among multiple sets, hierarchically grouped (e.g., tiles, clusters,...)
FORMAL CONTROL THEORY MODEL	Scheduling/Resource Allocation schema formalized by a C-T model
DESIGN-TIME RESULTS EXPLOITATION	Run-time exploitation of Design Space Exploration or off-line profiling results
PORTABILITY	The manager does not target specific systems or application domains.

TABLE II: RTRM comparisons

Resource Managers	Heter-Platforms	Homog-Platforms	Multi-Objective	Reconfig./Adapt.	Multi-Resource	Hierarchical Mappings	Control Theory Model	Design-time exploitation	Portability
STARPU [1]	✓	-	-	✓	✓	-	-	-	-
Binotto et al. [4]	✓	-	-	✓	✓	-	-	✓	-
Fu et al. [7]	-	✓	-	✓	✓	-	✓	✓	-
ACTORS [3]	-	✓	-	✓	✓	-	✓	-	-
SEEC [8]	-	✓	✓	✓	✓	-	✓	-	✓
DISTRM [10]	-	✓	✓	✓	-	-	-	✓	✓
OUR PROPOSAL	✓	✓	✓	✓	✓	✓	-	✓	✓

performed to assess the framework's performance, and first benefits for the applications are reported. Conclusions and the outline of our current research effort are drawn in Section V.

A. Related Works

To date, the Run-Time Resource Management is a quite new challenge for the embedded systems world, since the appearance of Multi/Many-Core architectures is a recent occurrence. Reasonably, a first form of resource management has been introduced by *virtualization* tools. Born in the scope of server-oriented architectures, in the last few years they found a market segment among the MPSoC-based embedded systems too. The isolation put in place by these tools determines a practical and effective resource partitioning among several software execution environments. However, the price to pay is expressed in terms of overheads introduced by the "hypervisor" module and the platform abstraction layer. More OS-integrated solutions ("lighweight virtualization"), like Linux Containers (LXC) [6], offer similar isolation capabilities, with considerably lower overhead. However, reconfigurability and adaptivity of the partitions are poorly supported.

Regarding the hardware resources instead, a point to consider is the difference between *homogeneous* and *heterogeneous* platforms. In the former case, resources of the same type provide the same capabilities. For instance, this means that the system provides a symmetric set of processing elements, hence featuring the same architecture and performances. Conversely, in the latter case, the platforms can be equipped with different and specialized computing architectures (often optimized for specific execution paradigms). This last case includes a very common architectural design, where the computational power is distributed between a host CPU (possibly multi-core) and a hardware accelerator. The accelerator provides support for the execution of highly parallelized code. Latest System-on-Chips, featuring multi/many-core Graphical Processing Units (GPUs) are noticeable examples of such platforms.
StarPU [1] and [4] propose run-time frameworks specifically focused on performance optimization of heterogeneous platforms. In both cases, task scheduling strategies do not target on other objectives, but performance maximization and load balancing.

The most part of proposals target resource management on homogeneous platforms. ACTORS [3], along with [7] and [15], are examples based on Control Theory Models. Basically they all provide scheduling strategies focusing power consumption optimization for real-time systems, taking into account the current performance requirements of the task. Nevertheless, the applicability of these very interesting approaches is constrained by the specific application domain. Hoffman et al. [8], propose an extremely flexible control system framework for adaptive computing. Moreover, the simplicity of its control mechanism does not allow a straightforward implementation of a resource mapping policy, that could take into account multiple options and a hierarchical layouts.

Unlike the *centralized* solutions seen so far, DistRM [10] suggests a *distributed* approach, based on a multi-agent system. The idea is to deploy an intelligent agent per application, acting as a local resource manager. Our approach is a somehow an "hybrid" solution, as we are going to see, with the resource allocation mainly defined by a centralized policy and then refined in a distributed hierarchical manner. We addressed this potential issue by considering the minimization of reconfigurations and migrations as one of the scheduling policy objectives. In the end, Tab. I summarizes the properties we considered in the evaluation of a run-time resource management system, while Tab. II provides a taxonomy of the most evaluable approaches, according to such properties.

II. APPLICATIONS RECONFIGURABILITY

An effective usage of the computational resources starts from the capability of the applications to run according to different configurations, that means different resource usage

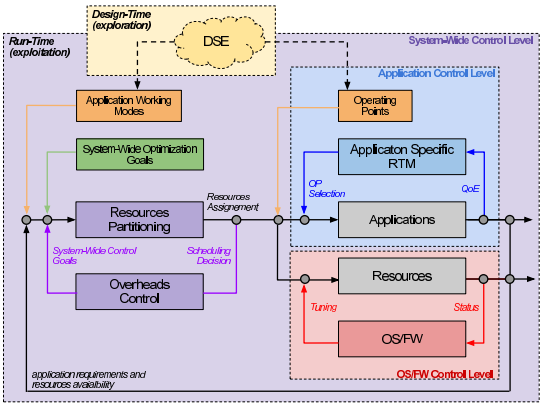

Fig. 1: The proposed distributed control.

levels. We think that design-time activities can provide a good contribution in profiling applications, and thus identify their possible configurations. To this purpose, we currently exploit a *Design-Time Exploration* (DSE) tool[2], which performs an optimal quantization of the configuration space of run-time tunable applications, identifying a finite set of configurations. The effectiveness of such an approach to efficiently support run-time resources management has been already demonstrated on many prior works [11], [12], [14].

The configuration of a run-time tunable application is defined by a set of parameters; some of them could impact just on the applications behaviors while others have a direct impact on the class or amount of the required computational resources. For example, if we consider a video decoding application, a tunable 'framerate' parameter impacts just on the application behaviors, while the 'CPU processing time' is a parameter directly impacting the amount of a required computational resource. This distinction allows to classify application tunable parameters into two different granularity levels:

- *Operating Point (OP)*: a collection of application specific parameters, corresponding to an expected QoS for the end user;
- *Application Working Mode (AWM)*: a collection of resources requirements, corresponding to an expected QoS for the application.

Indeed, a single AWM could support multiple OPs, i.e. the same kind and amount of computational resources could accommodate multiple values of application specific parameters. Therefore, OPs are bound to application-specific properties, i.e. impacting just the specific application, while AWMs describes system-wide properties, i.e. affecting system resources and thus every other system entity competing on their usage.

A. Distributed Hierarchical Control

The set of AWMs and OPs identified at design-time become valuable feed-forward signals, for the proposed run-time control solution. They lead to an efficient and low-overhead identification of control actions, as well as to a reduction of the convergence time required to reach a new stable system configuration.

We propose a distributed control scheme, where controllers are distributed throughout the system, acting on a specific subsystem. This design allows: a) to spread the control complexity, b) to design subsystem specific optimal controllers, and c) to scale better with the overall system complexity and number of subsystems.

The hierarchical approach allows the control at different granularities and abstraction levels. This simplifies the design of the overall control solutions for a complex system. Indeed, higher abstraction levels target system-wide optimizations, which are usually associated to non-negligible overheads and reduced optimization opportunities. Conversely, lower abstraction levels are targeted to fine tunings and optimizations, which usually relate to almost negligible overheads and frequent adjustment opportunities.

We propose three main classes of subsystems, each one corresponding to a different control level, and driven by a specific controller. For each run-time tunable application an "application control level" subsystem is defined. Based on the specific characteristics of the target application an *Application Specific Run-Time Manager (AS-RTM)*, is in charge to define a suitable policy to select an OP based on the amount of resources being assigned to the application (AWM), and the actual quality of experience obtained at run-time. Such an approach allows to exploit the detailed knowledge on: applications internals, performances evaluation and user perceived quality. Hence, this control level is in charge to: a) evaluate actual run-time application behaviors and b) tune its specific parameters or eventually to request more resources to the higher control level.

At the same hierarchical level of the previous one, we relay an "OS/FW control level", which is platform-specific. Here are located mechanisms like DVFS and thermal control capabilities, that can be exploited to react to risky conditions, e.g. hot spots and thermal alarms. Finally, the *System-Wide Run-Time Resource Manager (SW-RTRM)* targets a set of optimization goals which are achieved by a proper assignment of the available resources to the demanding applications. The rest of this paper will be mainly devoted to this level.

B. Application Program Interface

In this distributed and hierarchical view, the applications play an active role on the self-adaptiveness of the system. All these activities are supported by a *Run-Time Library (RTLib)*, which provides a rich set of features related not only to properly manage the interaction with the framework, but also to support the application specific run-time management activities.

AEM API. To simplify the coding (and the integration of existing) stream processing application, the RTlib provides the *Abstract Execution Model (AEM)*. Basically, it is defined via a callback based API, meaning that the developer just needs to implement the application specific logic into the body of few

[2]Actually this step is optional, but a complete description of the scenarios where this could be avoided is outside the scope of this paper.

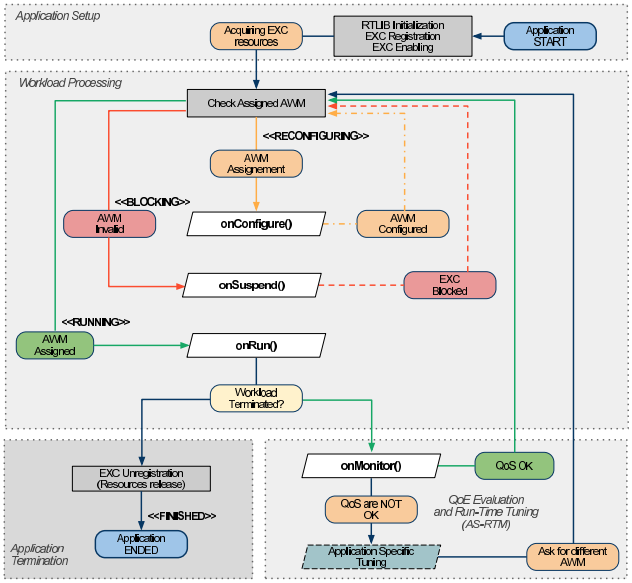

Fig. 2: Abstact Execution Model (AEM).

methods[3], represented by the parallelograms in Fig. 2. The implementation must define proper actions related to[4] a) the processing of a bunch of data (onRun), e.g. decoding a frame, b) reconfiguring into a different working mode (onConfigure) or c) suspending the workload processing (onSuspend), e.g. when the required computational resources are currently not available.

AS-RTM API. As previously discussed in Sec. II-A, the AS-RTM is basically in charge to monitor the application specific QoS in order to either fine-tune its parameters or to switch to a different AWM, once an agreement has been met with the SW-RTRM. The RTLib provides a collection of utilities to monitor metrics of interest (e.g. timings and usages on different resources, such as memory or bandwidth). Given such metrics, an Operating Point Manager (OP-Manager) can easily order and filter the set of operating points, on the basis of an optimization goal. By selecting an OP, the AS-RTM tunes some application specific parameters, without affecting the resource requirements. However, whenever the target QoS is not satisfactory, the application can ask for a change of AWM to the RTRM.

With reference to Fig. 2, this API is related to the collection of methods which could be exploited during the so called "QoS Evaluation and Run-Time Tuning" phase of the AEM. This step is always entered after the completion of a workload processing run, thus it gives an application the chance to evaluate and eventually tune its behavior based on the performance obtained during the previous computations.

[3]A default implementation is provided for methods which are not of interest for a specific application.

[4]Some more callbacks are defined to support more advanced usages.

TABLE III: Control properties

Property	Details
HIERARCHICAL	System-wide resource partitioning Application specific parameters tuning OS/FW resources monitoring and control
FEED-FORWARD	OP and AWM to increase convergence speed
DISTRIBUTED	Local control loop assisted subsystems
OPTIMAL	User defined goal functions, overheads control
ROBUST	Adapt to the small system variations
ADAPTIVE	The control algorithm is run-time computed
OBSERVABLE	Monitored applications and resources
CONTROLLABLE	Bounded and measurable response time

III. THE SYSTEM-WIDE RTRM

The BarbequeRTRM is implemented as an open source modular framework. Its layered design, with a main software stack, spans from target platform up to managed applications, besides a set of optional components which provide advanced features. However, due to lacking of space we will not provide a detailed review of the architecture of the framework in this paper.

A. Scheduling Control

An overall view of the proposed system-wide resource partitioning strategy is depicted in Fig. 3. The main goal of this control level is to identify an optimal partitioning of the available computational resources among the demanding applications. It is worth to notice that what we propose is an *event based scheduler*, where a new scheduling decision is based on events. Events are related either to changes on resources availability (or their status) or a change on applications requirements in terms of computational resources. Each time such an event happens, a new run of a "resource scheduler" is triggered to identify a new scheduling. This is the first step of our partitioning strategy.

The new proposed resources assignment is based on the scheduler optimization function, which exploits the design-time identified AWMs, according to a set of system-wide optimization goals. These goals encompass different system optimization aspects, ranging from applications performance to power and thermal profile as well as some overheads and control robustness aspects, thus actually supporting a *run-time tunable* and *multi-objective* optimization strategy.

However, the scheduling identified is not immediately enforced on the system. Indeed, the second step of the proposed strategy is related to "overheads control" and targets a validation of the scheduling decision based on a set of system-wide metrics and a scheduler validation policy. The goal of this policy is two-fold: a) to inhibit the actuation of scheduling decisions that are not compliant with constraints on system-wide optimization goals, and b) to tune some of the scheduler parameters, in order to improve the quality of the proposed solution on its following runs.

Since a new run of the resource scheduler could be eventually

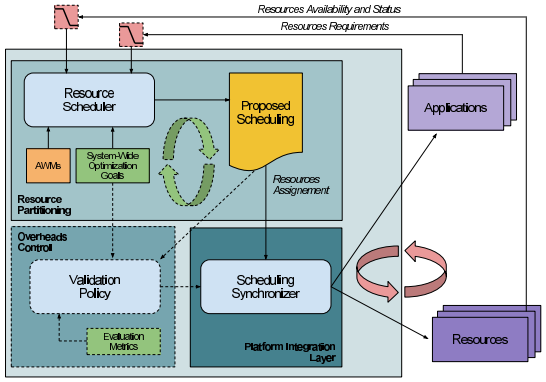

Fig. 3: System-Wide Controller.

started by the validation policy, the first two steps of the proposed strategy actually represent an *inner feedback control loop*, for the system-wide control level. Finally, the *outer feedback control loop* is closed by a *scheduler synchronizer* which is in charge to actuate a proposed scheduling decision by properly configuring both applications and resources. Therefore, reconfigured applications are properly notified about the new assigned AWM, i.e. the set of computational resources reserved to the application, and computational resources are set according to the scheduling decision. The last point is performed through a platform-specific module called *Platform Integration Layer (PIL)*.

It is worth to notice that the main goal of the inner loop is to reduce the chances to produce instabilities, or higher overheads on the outer loop. To the same purpose respond the two low-pass filters at the input of the resource scheduler. These are targeted to increase *control robustness*, by avoiding (or eventually delaying) the triggering of new scheduling events following small variations on either application requirements or resources availability. A resume of the main properties of the proposed control solution is reported in Tab. III

B. Resources Partitioning Policy

The resource partitioning, among the demanding applications, must consider 1) the requirements of each application 2) the status of the available resources and 3) a set of optimization goals. It is worth to notice that a) resources are *clustered*, thus we do not admit the possibility to schedule a single application across multiple clusters[5], and b) *every application has a predefined set of possible configurations* (the AWMs identified at design-time), each one corresponding to a certain amount of required resources. The partitioning policy is in charge to select (at most) one AWM for each active application, where it is assumed that there are:

- R platform resources, each one with availability $R_{r,c}$, where r is the resource and c one of the C clusters;
- A active applications, each one with priority P_i and a set of W_i possible configurations, i.e. AWMs;

[5]This is a restriction which is going to be removed in a next version of the partitioning policy.

TABLE IV: The set of considered optimization goals

Goal	Description
PERFORMANCE	the expected QoE an application could get, i.e. the sustainable framerate for a video decoding application
OVERHEAD	the overheads involved to switch from a previous mapping to a different one, e.g. time and energy
CONGESTION	how much resources are used compared to a predefined threshold level, e.g. a certain bandwidth utilization on a network channel
FAIRNESS	Bounded unfair resources allocation to same priority applications, e.g. almost same CPU bandwidth
STABILITY	Allows reconfigurations as long as overheads are "reasonable" and applications QoE are not noticeably affected
ROBUSTNESS	Absorb small modifications regarding both applications requirements and resource availability
ENERGY	Enforce bounded per-application energy consumptions
THERMAL	Hot spots avoidance and thermal leveling

This configures as a well studied problem in combinational optimization which is known as *multi-choice multi-dimension multiple knapsacks* problem (MMMKP) [9], indeed we have respectively:

- *items* (i.e. AWMs) subdivided into multiple *classes* (i.e. applications), with exactly one AWM to be selected for each application;
- multiple *constraints*, represented by the availability $R_{r,c}$ of each resource r within each cluster c;
- multiple *containers*, represented by the clusters, each one with its own resources availability.

Moreover, the formulation of the MMMKP problem is completed by the introduction of a *profit* P to be associated to each possible choice. In our problem a choice is defined by a *mapping* $M_{i,j,k} :=< A_i, W_j, C_k >$, which represents the scheduling of the j-th AWM (W_j) of the i-th application (A_i) into the k-th cluster (C_k). The profit assigned to such a choice is defined by a multi-objective optimization policy π, described thereafter, and thus $P_{i,j,k} = \pi(M_{i,j,k}, G)$, where $M_{i,j,k}$ is a mapping and G is the set of the optimization goals. Such a formulated MMMKP problem is known to be NP-hard [9], and algorithms for exact solutions are too slow and thus not suitable for an efficient run-time management exploitation. Fortunately, state-of-the-art heuristics have been developed, which allows to find near-optimal solutions and are fast enough for the considered environment.

The optimization policy we propose belongs to this class of solutions and it has been inspired by a *greedy heuristic* (henceforth referenced as "original heuristics"), proposed by Ykman-Couvreur et al. [13], which has been considered since it has been demonstrated to exhibit reasonable overhead, even in the specific context of resource constrained and real-time embedded system. The original heuristic was targeted to a problem similar to our with respect of which we consider a) *multiple clusters*, instead of just one, and b) *multiple optimization goals*, instead of just the energy minimization. These two main differences extend the original problem to be a multi-knapsack with a multi-objective optimization goal (MMMKP),

instead of just a MMKP, and for these reasons the original heuristic has been updated to consider the aforementioned differences.
The main steps of the original heuristics are still valid considering these four main modifications: 1) the initial "Pareto filtering preprocessing" step is completely delegated to the Design Space Exploration, which is performed at design time; 2) the "Multi-dimensional resource reduction" step has been replaced by a *"Mappings Computation (MC)"* which targets also the multi-cluster extension (a); 3) the "Pareto point sorting" is implemented by a *"Mappings Ordering (MO)"* step designed to consider the multi-objective optimization extension (b); and finally 4) the "Greedy algorithm" has been replaced by a similar near-optimal solution finder, which targets the *"Mappings Selection (MS)"* considering the system-wide optimization goals. While a detailed description of the first step is outside the scope of this paper, last three steps deserve more interests and will be detailed.

Mappings Computation (MC). This step reduces the multi-dimensional solution space into a single-dimension space which is the input for the mappings selection step. As previously described, in the formulation of the MMMKP problem, each mapping (i.e. each possible scheduling choice) could be associated to a value of profit. This step is in charge to compute such profit P for each possible mapping and contextually order all the mappings by decreasing profit value. The profit function π we propose has been defined to consider all the optimization goals we are targeting, which are defined in Tab. IV. Each one of these goals represents an *optimization metrics* and it is associated to a function $\pi_g(M_i, g)$ which evaluates a mapping M_i with respect to the specific optimization goal g and returns a normalized index m_i, i.e. $m_i \in [0..1]$.
The set of optimization metrics could be geometrically interpreted as vectors defined in a M-dimensional space, where M is the number of optimization goals we consider. For example, in Fig. 4 it is represented (just for readability) a bi-dimensional optimization space corresponding to the optimization metrics g_1 and g_2; each mapping is represented by a point which identifies a vector on this space, e.g. $\overrightarrow{P_{3,j,k}}$ represents the profit of application A3 for the mapping of its j-th AWM into the k-th cluster.

Mappings Ordering (MO). The optimization goals of the System-Wide RTRM belong also the set of metrics defined in Tab. IV. Indeed, our solution provides support for the definition of a system-wide optimization strategy by giving a relative importance to each one of these goals. Thus, the system-wide optimization goal could also be represented as a vector $\overrightarrow{G}$, within this M-dimensional space, which identifies an *optimization direction*. The ordering of mappings, by decreasing profit value, is thus a straightforward operation which requires just to compare the component of metric vectors $\overrightarrow{P_{i,j,k}}$ on the optimization direction:

$$Comp_{\overrightarrow{G}}\overrightarrow{P_{i,j,k}} = \frac{\overrightarrow{G} \cdot \overrightarrow{P_{i,j,k}}}{\| \overrightarrow{G} \|} = \frac{\overrightarrow{O_{i,j,k}}}{\| \overrightarrow{G} \|} \quad (1)$$

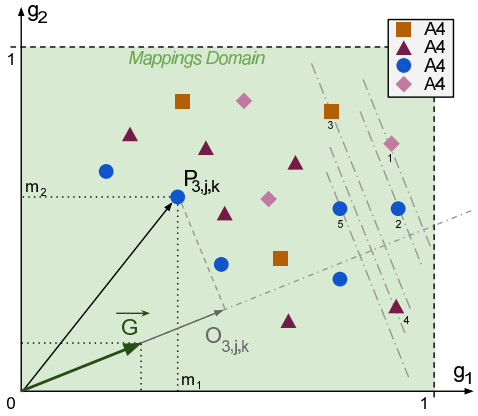

Fig. 4: Example of 2D optimization space

This implies that an higher chance to be scheduled is given to the mappings with better indexes for each optimization goal. Furthermore, since at each optimization run the system-wide optimization goal vector $\overrightarrow{G}$ is supposed not to change, for the purposes of the ordering we can just compute the numerator of Eq. 1, which is as simple as the vector scalar product identified by $\overrightarrow{O_{i,j,k}}$.
The asymptotic complexity of this step is essentially defined by that of the ordering algorithm: $O(A \cdot W \cdot C \cdot log(A \cdot W \cdot C))$ [5].

Mappings Selection (MS). For each application to be scheduled, the best candidate mapping is selected, starting from the highest profit ones. The selection is performed until the required amount of resources is available, otherwise the next candidate is picked. The step is completed whenever an AWM has been assigned to each application, or the resources have been saturated. The worst case complexity of this last step is $O(A \cdot W \cdot C)$.

IV. Experimental Results

An experimental evaluation of the proposed framework has been done in order to assess its overheads and scalability properties, as well as the framework behaviors and benefits on the management of highly congestioned workload scenarios. The experiments have been carried out on a four nodes NUMA machine, each one consisting of a Quad-Core AMD Opteron Processor 8378 running at 2.8GHz[6], for a total count of 16 Processing Elements (PE). These PEs have been organized into a *host partition* with 4 CPUs, used to run the framework and all the other generic system services and applications. The remaining 12 CPUs have been placed into a *device partition*, used to run only the BarbequeRTRM managed applications, and further divided into 3 clusters with 4 CPUs each one.

A. Performance

A first experiment evaluates the scalability of the resource partitioning algorithms described in Sec. III-B. The results are reported in Fig. 5a, where the average scheduling time

[6]Experiments have been conducted by locking the CPU to that frequency

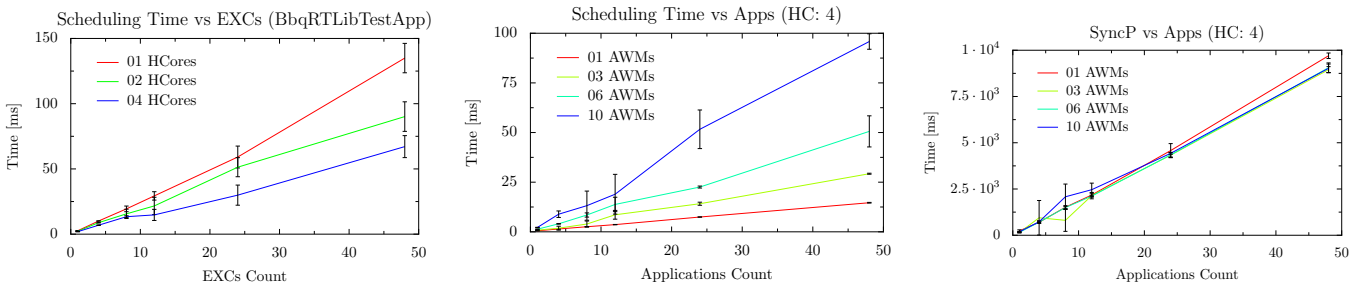

Fig. 5: Burst start performance: a) Scheduling scalability, b) Scheduling overheads and c) Synchronization overheads

TABLE V: Control Loops Timings [ms]

Overheads	Min	Max	Avg	Var	Rate
OUTER LOOP	181.050	192.215	187.735	4.817	*5*
INNER LOOP	1.930	2.111	1.997	0.081	*474*

has been measured for an increasing number of *concurrently starting applications* (EXCs[7]) and different number of CPUs used to run the framework (HCores). This is a sort of worst case analysis, since we consider bursts of applications starting at the same time. Each application has 10 different AWMs, thus requiring the computation of up to 1440^8 mappings in the case of 48 applications starting concurrently.
Two are the main results: 1) the required computation time is in the order to few hundreds of $milliseconds$ even for large bursts; 2) the computation time scaled well with the parallelism offered by the HOST device (around 50% speedup moving from a single- to a quad-core).

The second experiment evaluated the scheduler overheads for a different number of AWMs per managed application and still considering a worst case analysis, i.e. burst of starting applications. The results are reported in Fig. 5b where, as expected, the required scheduling time scales linearly with respect to the number of mappings to be computed. These timings impact directly on the behaviors of the inner control loop discussed in Sec. III-A. Based on these experiments we can infer the expected *maximum control rate*, which is reported in Tab. V. For the inner loop, a worst case of 2[ms] for a single application with 10 AWMs corresponds to a scheduling rate near to 500 scheduled applications per second. In Fig. 5c are reported the results of a third experiment targeting the characterization of the outer control loop, which refers to the system synchronization mentioned in Sec. III-A. It is worth to notice that, in this experimental setup, the platform integration layer exploits the CGroups framework provided by the most recent available Linux kernel (v3.3.1). Almost all the synchronization overhead is related to the creation of cgroups, which requires up to 10[s] to schedule 48 applications, and scale linearly with the number of applications. As summarized in Tab. V, the time required to *create* a new cgroup is in the order of 200[ms], but this latency is required just at application start time, while its reconfiguration has been measured to be one order of magnitude lower.
Two are the main observation here: 1) the synchronization overhead on a NUMA machine using CGroups is not negligible and 2) these times are expected to be negligible on different platforms where the accelerator is not CGroups controlled. In this case, the platform integration layer should reasonably rely on more effective low level interfaces (e.g., a device driver).

[7]Multiple instances of a custom synthetic workload generator.
[8]10 AWMs, mapped in 3 possible clusters, for each of the 48 application.

B. Use Case

In order to better evaluate the benefits from the application side, workloads from the PARSEC v2.1 benchmarks suite [2] are going to be integrated with the RTLib provided by the BarbequeRTRM. So far, just the *bodytrack* workload has been integrated within our API and some tests have been conducted, targeting the same NUMA machine configuration but with CPUs frequency locked at 800 Mhz. In this test scenario an increasing number of instances of bodytrack has been launched, sampling the completion time and the power consumption, this last by collecting information exported by the available *Intelligent Platform Management Interface (IPMI)*.

A comparison between the original version of bodytrack and the RTRM-managed one is reported in Fig. 6a. In the first case, the resources assignment is in charge of the Linux scheduler, while in the second, the scheduling policy of the BarbequeRTRM defines resource partitioning by properly configuring CGroups at run-time. The first plot reports a considerable improvement in the completion time which is explained by the better capability of the RTRM to manage concurrency by enforcing resources partitioning. Indeed, in highly congested scenarios where multiple applications compete on reduced resources, our framework enforces an optimal partition and an eventual sequential scheduling, thus reducing access conflicts and corresponding effects (e.g. caches trashing, time delays). As expected, these improved computation behaviors translate to a lowered power consumption, as reported in 6b. This figure shows that the best-effort Linux scheduler, which tries its best to co-schedule all the applications concurrently, produces an average power consumption which is always greater than the one we get using the BarbequeRTRM, even for reduce number of workloads. Energy benefits are even greater since our scheduler reduces also the completion time.

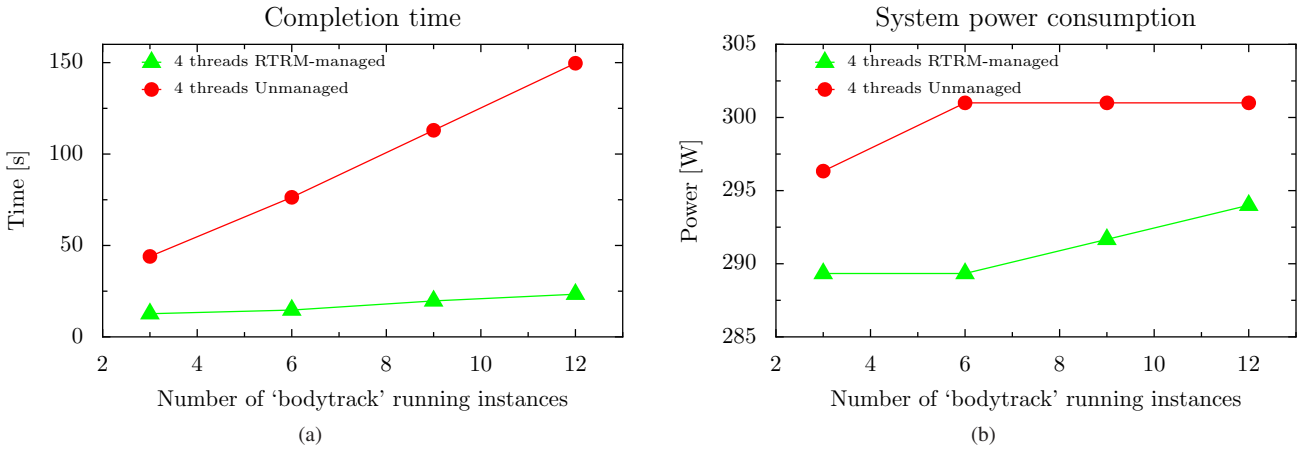

Fig. 6: PARSEC 'bodytrack' benchmark comparison between RTRM-managed and unmanaged execution.

Overall, these initial results demonstrate how reconfigurability and platform-aware resource partitioning are two key points for a more efficient and effective resource usage.

V. Concluding Remarks

The paper describes a framework providing system-wide run time management of resources, tailored to multi/many core architectures. It has been shown the benefits of the underlying multi-level design methodology, basically in terms of:

- effective exploitation of system resources while fulfilling predefined application QoS
- flexibility and scalability of the RTRM strategy, thanks to its hierarchical and distributed control structure
- acceptable overheads allowing usage in real cases, including those with variable workload
- availability of a simple API interface making straightforward for the programmers to take full advantages from framework services
- 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

Work is in progress within the 2PARMA FP7-ICT-248716 FP7 EU-sponsored project[9] and some demo videos showing the BarbequeRTRM framework running on real architectures are available on the web. Current research effort is the direction of improving the control policies to ensure better stability and robustness and a dynamic tuning of the policy itself, according to possible changing of the usage scenarios or in the presence of faults. Porting on Android is also in progress, as well as the support for the P2012 many-core platform by STMicroelectronics.

References

[1] C. Augonnet, S. Thibault, R. Namyst, and P.-a. Wacrenier. StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. *Writing*, 23(2):187–198, 2009.

[2] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 72–81, New York, NY, USA, 2008. ACM.

[3] E. Bini, G. Buttazzo, J. Eker, S. Schorr, R. Guerra, G. Fohler, K.-E. Arzen, V. Romero, and C. Scordino. Resource Management on Multicore Systems: The ACTORS Approach. *IEEE Micro*, 31(3):72–81, May 2011.

[4] A. P. D. Binotto, B. M. V. Pedras, M. Goetz, A. Kuijper, C. E. Pereira, A. Stork, and D. W. Fellner. Effective Dynamic Scheduling on Heterogeneous Multi/Manycore Desktop Platforms. *2010 22nd International Symposium on Computer Architecture and High Performance Computing Workshops*, pages 37–42, 2010.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2nd edition, 2001.

[6] V. Danen. Introducing Linux virtual containers with LXC, 2010.

[7] X. Fu and X. Wang. Utilization-Controlled Task Consolidation for Power Optimization in Multi-core Real-Time Systems. In *2011 IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 73–82. IEEE, Aug. 2011.

[8] L. A. Hoffmann H., Maggio M., Santambrogio M. D. SEEC: A Framework for Self-aware Computing. *Artificial Intelligence*, 2010.

[9] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[10] S. Kobbe, L. Bauer, D. Lohmann, W. Schroder-Preikschat, and J. Henkel. DistRM: Distributed resource management for on-chip many-core systems. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2011 Proceedings of the 9th International Conference on*, pages 119–128, 2011.

[11] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*, pages 196–201.

[12] P. Yang and F. Catthoor. Pareto-optimization-based run-time task scheduling for embedded systems. In *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign & system synthesis - CODES+ISSS '03*, page 120, New York, New York, USA, Oct. 2003. ACM Press.

[13] C. Ykman-Couvreur, V. Nollet, F. Catthoor, and H. Corporaal. *Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management*. IEEE, 2006.

[14] C. Ykman-Couvreur, V. Nollet, T. Marescaux, E. Brockmeyer, F. Catthoor, and H. Corporaal. Pareto-Based Application Specification for MP-SoC Customized Run-Time Management. In *Embedded Computer Systems: Architectures, Modeling and Simulation, 2006. IC-SAMOS 2006. International Conference on*, pages 78–84, 2006.

[15] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. *ACM SIGOPS Operating Systems Review*, 37(5):149, 2003.

[9] 2PARMA Project Website, http://www.2parma.eu