A Low-Overhead Heuristic for Mixed Workload Resource Partitioning in Cluster-Based Architectures

Davide Zoni, Patrick Bellasi, and William Fornaciari

Politecnico di Milano – Dipartimento di Elettronica e Informazione
Via G. Ponzio 34/5, 20133 Milano, Italy
{zoni,bellasi,fornacia}@elet.polimi.it

Abstract. The execution of multiple multimedia applications on a modern Multi-Processor System-on-Chip (MPSoC) rises up the need of a Run-Time Management (RTM) layer to match hardware and application needs. This paper proposes a novel model for the run-time resource allocation problem taking into account both architectural and application standpoints. Our model considers clustered and non-clustered resources, migration and reconfiguration overheads, quality of service (QoS) and application priorities. A near optimal solution is computed focusing on spatial and computational constraints. Experiments reveal that our first implementation is able to manage tens of applications with an overhead of only fews milliseconds and a memory footprint of less than one hundred KB, thus suitable for usage on real systems.

1 Introduction

Modern multimedia applications for new generation multi-core mobile embedded systems require specific resources and exhibit dynamic QoS requirements. The required support to parallel application execution with both high performance and low energy costs, raised up new classes of problems. *Quality of service (QoS) management*, which concerns to the definition of mechanisms and policies to enhance user experience in a cost effective manner [8], represents a first problem. The system design should consider the dynamic nature of user requirements and the support of different optimization strategies, e.g. multimedia boosting and resource saving. This can be done with a run-time fine tuning on applications' behavior, for example by dynamically changing the level of parallelism of the applications. A suitable run-time support cannot disregard *mixed workload* usage scenarios, where both device critical services (e.g., a radio stack) and user-installed applications run aside competing on the usage of the same resources. This requires the ability to specify a *priority level* for each application and to properly handle it at run-time. The presence of *clustered resources* is another main issue to focus on. Especially on many-core systems, while some resources are globally accessible (e.g., an external shared memory) others are organized into groups that can only be assigned as a whole, e.g., clusters of processing elements. Finally, run-time *migration and reconfiguration overheads* should be considered to take optimal run-time resource scheduling decision. Indeed, the opportunity to reconfigure an active application should be evaluated considering that migration and reconfiguration actions introduce both performance and energy additional costs [1]. An effective resource assignment task cannot disregard these aspects.

Considering all of these new classes of problems, the design of modern and efficient embedded systems targeting mobile multimedia applications could benefit from the introduction of a System-Wide Run-Time Management (SW-RTM) layer to match dynamically changing applications requirements with platform resources availability [7]. This paper presents a novel formal model and a heuristic providing an adaptive system optimization level that captures all emerging design aspects and related challenges. The main contributions are four. First, we propose a new formal model which allows to describe a run-time manager able to consider application priorities. Second, the proposed model can manage both clustered and non clustered resources. Third, migration and reconfiguration aspects are considered both in the problem formulation and its solution, thereby paying attention on run-time overheads. Finally, the proposed heuristic represents another contribution which provides a fast near optimal solution to the new formulated problem, perfectly suitable to be used in realistic implementation.

The rest of the paper is organized in four distinct parts. Section 2 provides an overview on previous works in this area. Section 3 details the proposed formal model and heuristic. Section 4 discusses the experimental results, while conclusions are drawn in Section 5.

2 Background

Among the approaches to run-time resource management, the "Pareto optimality" is a widely used concept [6,11,10]. Kahn et al. [6,5] describe a "utility model" which allows to obtain a resources allocation solution aiming to approach the Pareto optimality. This model constructs a multi-dimensional multi-choice knapsack problem (MMKP) formalizing the concept of profiled applications with QoS constraints and Pareto optimal resource allocation. Many approaches have been proposed that are based on the Kahn's utility model. In [7], Nollet et al. describe a generic run-time resource manager and provides useful guidelines to design its main functional blocks. Based on this work, Ykman-Couvereur et al. [12] present a runtime manager for multi-processors, which is capable to manage applications with different operating configurations and QoS constraints. The runtime manager selects a system configuration for all the active applications as a MMKP solution. Other works define different models of the solution. Shojaei et al. [10] proposes a solution for chip-multiprocessors (CMP), exploiting the Pareto Algebra [2]. Worth to notice are also the heuristic defined by Akbar et al. [9], which aims to boost the computation of the optimal resource allocation by exploiting multi-core architectures, and the approximated solution to the MMKP described by Hifi [3], which is based on a local search paradigm.

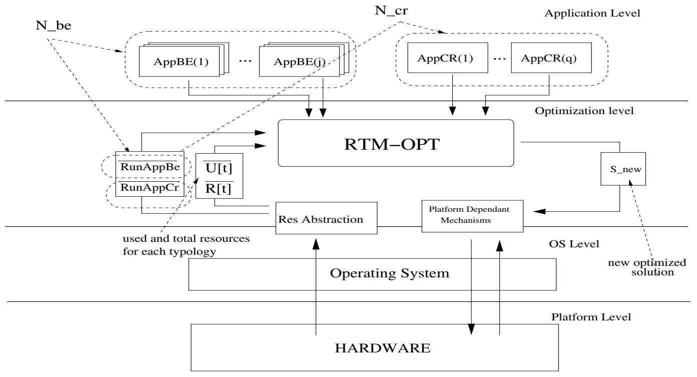


Fig. 1. Logical view for the proposed run-time optimization manager

Finally, many other different approaches have been investigated, especially on application scenarios different from that of embedded systems. Each proposals to find an efficient heuristic solution to the Khan's utility model described so far, implicitly suffers of the same lacks of the original model itself. Apart from multiple AWMs, these proposals do not consider mixed workload, clustered resources and migration/reconfiguration costs.

3 Proposal

The proposed run-time manager can conceptually be seen as a module at the Operating System (OS) level, devoted to optimize both the resource assignment to running applications and the platform objectives, e.g. power saving. The underlying system is not explicitly defined in the model depicted in Fig. 1, since the flexibility of the proposed approach allows to optimize a set of different multi-core real platforms. However, to better contextualize the proposal, a brief description of two possible real architectures could be provided. A first example is represented by a non-uniform memory access multi-core architecture. Resources classes can be represented by processing elements, local and global memory subsystems and interconnection subsystems. The RTM layer represents an optimization layer on top of the OS, and it uses the OS interfaces to directly control performance and resource assignment (see Fig. 1 blocks *Res Abstraction* and *Platform Dependant Mechanism*). Another multi-core architecture is represented by a host processor (where the OS runs) joined to a four cluster computation fabric, each cluster containing 16 processing elements and a shared memory. The proposed RTM framework is in charge to allocate applications on the fabric resources on behalf of the operating system, using the interfaces provided by the fabric device driver.

The proposed work does not substitute operating system or the low level drivers, but rather provides a framework to manage multiple optimization goals in modern multi- and many-core architectures, using specific platform-dependent information provided by the operating system or by a dedicated driver in charge of controlling a specific peripheral or resource. Moreover, our work focuses on the methodological aspects decoupling the optimization phase from the platform-dependent phase, providing better results in terms of portability, optimization and flexibility.

To complete the description of the proposed RTM, sketched in Fig. 1, a set of definitions should be provided. We define a solution $\mathcal{S}$ as an allocation of platform resources to the active applications, and we assume that the RTM (i.e., the RTM-OPT block in Fig. 1) constructs a new solution $\mathcal{S}$ when an event E occurs in the system. An *event* can be either application or platform dependent. An application event occurs when applications enter/exit the system or assert a new resource requirement, while a platform event identifies a change in the abstract vision of the resources (e.g., some PEs becoming unavailable due to a thermal alarm). Hence, the proposed RTM can be seen as a state machine $RTM_{fsm} := S_{old} \xrightarrow{E} S_{new}$, where S_{old} represents the current allocation of the resources to the active applications, S_{new} defines the new allocation of the resources to the applications and E identifies the application or hardware event which triggered the system reconfiguration. Moreover, every time the RTM_{fsm} constructs a new solution, it is assumed that N_{be} and N_{cr} applications are active, where *be* and *cr* identify respectively best-effort and critical applications. Indeed, to assess the validity of this work we discuss both model and heuristic considering a framework with two different priority levels: critical and best-effort. Moreover, from the architectural point of view, we consider both clustered and non-clustered resources. Specifically we consider a single clustered resource and a variable set of clustered resources.
It is worth to notice that these assumptions have been made to provide a manageable way to present model, heuristic and results, however they do not constraint the expressiveness of the proposed solution and could be easily removed as discussed later in the paper.

3.1 Formal ILP Based Model

Considering the above hypothesis, we defined the resource allocation problem in terms of a *formal model* based on Integer Linear Programming (ILP). The same model has been used as a reference ("golden model") for the evaluation of the proposed *heuristic* described in Section 3.3. This evaluation strategy has been motivated by the lack in literature, from the best of our knowledge, of formal proposals and quantitative models which are able to capture all the terms we are considering: migration and reconfiguration costs, application priorities and clustered resources.

Our formal model, for the identification of "the best" scheduling choice, is defined by a goal function (Fig. 2) which represents a maximization problem for the utility ($\nu_{a,m}$) of each application considering a system with N_{be} best-effort applications, each one with M working modes, running on a system with K clusters. It is worth noting that, since critical applications have only one working mode which is granted to be always scheduled by the run-time manager, the

$$MAX \sum_{a=1}^{N_{be}} \sum_{m=1}^{M} \sum_{k=1}^{K} \nu_{a,m} * x^{be}_{a,m,k} - \sum_{k=1}^{K} y_k * \Theta_k - \sum_{a=1}^{N_{be}} \sum_{m_1=1}^{M} \sum_{m_2=1}^{M} xRec^{be}_{a,m_1,m_2} * \Delta^{be}_{m_1,m_2}$$
$$- \sum_{a=1}^{N_{cr}} \sum_{k_1=1}^{K} \sum_{k_2=1}^{K} xMig^{cr}_{a,k_1,k_2} * \Psi^{cr}_{k1,k_2} - \sum_{a=1}^{N_{be}} \sum_{k_1=1}^{K} \sum_{k_2=1}^{K} xMig^{be}_{a,k_1,k_2} * \Psi^{be}_{k1,k_2}$$

Fig. 2. The goal function of the proposed ILP based golden model

objective function of the proposed model considers only best-effort applications, which do not have a null value. Critical applications are accounted for by a set of constraints explained in the following.

Negative terms are used to introduce penalties: the cost Θ_k to use "new" clusters, the overhead Ψ^{cr} to migrate critical-applications and the penalties to re-configure (Δ^{be}) or to migrate (Ψ^{be}) best-effort applications. Reconfiguration and migration costs, represented by Ψ^* and Δ^* coefficients, are defined independently for both critical and best-effort applications. These coefficients allow to fine tune the allocation algorithm considering relative overheads for each of these operations. However, the definition of a proper methodology to obtain a quantitative evaluation of these coefficients is out of the scope of this paper. Indeed, in this work the focus is just on the definition of a suitable model to properly capture these aspects, thus preparing the ground to exploit them for the solution of the scheduling problem on real system where of course such coefficients must be properly identified.
Binary variables are used in the model to join the state of the system ($\mathcal{S}$) with the state of the running applications. In particular, each application has a state variable defined as:

$$x^{cr}_{a,k} \in \{0,1\}, \qquad \forall a \in [1..N_{cr}], \forall k \in [1..K]$$
$$x^{be}_{a,m,k} \in \{0,1\}, \qquad \forall a \in [1..N_{be}], \forall m \in [1..M], \forall k \in [1..K]$$

where cr and be identify the critical (cr) and best-effort (be) class of the application, respectively. This variable ($x^*_{...}, * \in \{cr, be\}$) describes on which cluster k is allocated the application a and the running working mode m if the application is a best-effort one.
Three other sets of variables capture, for each application, migrations and reconfigurations:

$$xMig^{cr}_{a,k1,k2} \in \{0,1\}, \qquad \forall a \in [1..N_{cr}], \forall k1, k2 \in [1..K]$$
$$xMig^{be}_{a,k1,k2} \in \{0,1\}, \qquad \forall a \in [1..N_{be}], \forall k1, k2 \in [1..K]$$
$$xRec^{be}_{a,m1,m2} \in \{0,1\}, \qquad \forall a \in [1..N_{be}], \forall m1, m2 \in [1..M]$$

while best-effort applications have monitored both reconfiguration and migration aspects, only the migration factor is evaluated for critical applications, since only a single working mode is available.

Moreover, a set of parameters, one for each active application, summarizes their previous configuration:

$$xOld^{cr}_{a,k} \in \{0,1\}, \qquad \forall a \in [1..N_{cr}], \forall k \in [1..K]$$
$$xOld^{be}_{a,m,k} \in \{0,1\}, \qquad \forall a \in [1..N_{be}], \forall m \in [1..M], \forall k \in [1..K]$$

Finally, each cluster has a binary variable, indicating whether or not it is active:

$$y_k \in \{0,1\}, \qquad \forall k \in [1..K]$$

At any point in time the system state $\mathcal{S}$ has to deal with some constraints, that guarantee the feasibility of the resource assignment. First of all the proprieties of the different best-effort and critical applications are introduced:

$$\sum_{m=1}^{M}\sum_{k=1}^{K} x^{be}_{a,m,k} \leq 1, \qquad \forall a \in [1..N_{be}]$$
$$\sum_{k=1}^{K} x^{cr}_{a,k} = 1, \qquad \forall a \in [1..N_{cr}]$$

They ensure the execution of critical applications, while offering a best effort service with respect to best-effort applications. Furthermore, only a single AWM must be active at any point in time for every application.
Another set of constraints limits the solution feasibility to the assignments that do not exceed the available both clustered and non-clustered resources, $\forall t \in [1..T_{NCL}]$ and $\forall k \in [1..K]$:

$$\sum_{a=1}^{N_{be}}\sum_{m=1}^{M}\sum_{k=1}^{K} r^{NCL}_{a,m,t} x^{be}_{a,m,k} + \sum_{a=1}^{N_{cr}}\sum_{k=1}^{K} r^{NCL}_{n,t}\, x^{cr}_{a,k} \leq R^{NCL}_{tot}[t]$$
$$\sum_{a=1}^{N_{be}}\sum_{m=1}^{M} r^{CL}_{a,m,t} x^{be}_{a,m,k} + \sum_{a=1}^{N_{cr}} r^{CL}_{n}\, x^{cr}_{a,k} \leq R^{CL}_{tot}[k]$$

where $R^{NCL}_{tot}[t]$ represents, for each non-clustered resource of type t, the available resources, while $R^{CL}_{tot}[k]$ represents the total amount of available resources on each cluster k. Note that a single index to identify R^{CL}_{tot} symbol is needed since, for the sake of simplicity, a single clustered resource class has been considered. The $r^{NCL}_{a,m,t}$ details the resource of type t required by the application a to run in working mode m, while $r^{NCL}_{n,t}$ represents the resources of type t required from critical application n for its single configuration. Finally $r^{CL}_{a,m}$ and r^{CL}_{n} represent the required clustered resources for a best-effort application a, running in working mode m, and a critical application n, respectively.
Moreover, a set of constraints are added to control the state of every clustered resources, that must be active or switched off based on the scheduled applications, $\forall k \in [1..K]$:

$$\sum_{a=1}^{N_{be}}\sum_{m=1}^{M} x^{be}_{a,m,k} + \sum_{a=1}^{N_{cr}} x^{cr}_{a,k} \leq (N_{cr} + N_{be}) * y_k$$

As last part of the model, a set of constraints are provided to exploit the migration and reconfiguration variables defined above:

$$x^{be}_{a,m1,k1} + xOld^{be}_{a,m2,k2} \leq 1 + xRec^{be}_{a,m1,m2} \ , \quad \forall a \in [1..N_{be}] \wedge m1, m2 \in [1..M] \wedge k1, k2 \in [1..K] \wedge m1 \neq m2$$

$$x^{be}_{a,m1,k1} + xOld^{be}_{a,m2,k2} \leq 1 + xMig^{be}_{a,k1,k2} \ , \quad \forall a \in [1..N_{be}] \wedge m1, m2 \in [1..M] \wedge k1, k2 \in [1..K] \wedge k1 \neq k2$$

$$x^{cr}_{a,k1} + xOld^{cr}_{a,k2} \leq 1 + xMig^{cr}_{a,k1,k2} \ , \quad \forall a \in [1..N_{cr}] \wedge k1, k2 \in [1..K] \wedge k1 \neq k2$$

3.2 Heuristic Approach

Since a RTM framework has to operate "on-line" with a negligible overhead, this Section proposes a heuristic to allocate both critical and best-effort applications producing a near-optimal scheduling with respect to the optimal ILP solution described so far. Thus, the performance of the heuristic is evaluated by comparing its solution with that computed using the ILP based model. Since *critical applications* could be exposed to time constraints, they are scheduled before the best-effort ones. Moreover, the scheduling algorithm is designed to avoid migrations of already running critical applications. However, when such an application must be migrated, for instance because of an hardware failure, this application is treated as new and scheduled before any other ones. Our heuristic schedules critical applications on a (possibly) already active cluster with the lower usage of local resources, thereby reducing clusters switch-on costs and performance degradations due to high contention on clustered resources. The goal of *best-effort applications* scheduling, accordingly to the ILP model, is to maximize the overall system utility. This requires to reduce reconfigurations and migration overheads as well as to avoid resource wasting, i.e. to reduce the number of used clusters. The proposed approach exploits a per-cluster support data structure, which contains all the working modes, of both new and running best-effort applications, ordered according to the descending value of a metric τ, whose computation is described later in Section 3.3. The scheduling heuristic is based on few simple steps. Starting with the first cluster, working modes are ordered according to the τ metric to produce the ordered list. Then, the actual scheduling is produced by visiting the list and scheduling each AWM such that the corresponding application is not already scheduled and the resources demand is compliant with the corresponding resource availability. When cluster resources have been filled-up or no more working modes are available in the list, the heuristic moves to the next cluster where the same steps are applied but considering just applications which have not yet been scheduled. This process stops when all applications have been scheduled or no more resources are available.

3.3 Working Mode Ordering Metric

The ordering of the applications working modes according to a certain metric τ is pivotal since it allows to simplify application scheduling by executing the

$$\tau^{heu}_{(a,m_1,m_2,k_1,k_2)} = \Big(\nu[a,m_2] - \varPsi^{be}[k_1,k_2] - \varDelta^{be}[m_1,m_2]\Big) * \left(\sum_{\rho=1}^{T} \frac{\delta[a,m_2,\rho] * \mu(\rho)}{\phi[\rho]/\kappa[\rho]}\right)^{-1}$$

Fig. 3. The working mode ordering metrics used by the proposed heuristic

straightforward “selection” algorithm described so far, and can capture all the key concepts of our formal model. The τ metric, formally defined by the equation in Fig. 3, is a function of the considered application a, its current and future working mode (m_1,m_2), and the current and future cluster (k_1,k_2) on which the application runs. Starting from the utility ν_{a,m_2} of a candidate m_2 target AWM for the application a, the numerator takes care of migrations and reconfigurations costs while the denominator accounts for resource contention. Precisely, the utility of each working mode is decreased by the *migration costs* $\varPsi^{be}(k_1, k_2)$ for moving the application from cluster k_1 to cluster k_2 and, dually, the value is also decreased by the *reconfiguration overhead* $\varDelta^{be}(m_1, m_2)$ to switch it from the current working mode m_1 to the candidate working mode m_2. This simple equation allows to penalize the value of working modes which corresponds to either a migration or reconfiguration of an already running application. The penalty is defined by a pair of cost matrices, respectively $\varPsi^{be}$ for migrations and $\varDelta^{be}$ for reconfigurations, which allow to define a different value for each pair of current and candidate configuration. The second factor of the metric τ, the denominator, considers resource contention using a sum for each resource type ρ. The idea of these terms is to penalize more working modes which request scarce resources by adopting a “demand-offer approach”: the more a resource is contended the more the working mode metrics will be penalized with respect to others. Its numerator computes the product between the *resource demand* $\delta[a, m_2, \rho]$, that is the amount of resource the application a requires in working mode m_2, and the factor $\mu(\rho)$ which is computed considering all the applications to be scheduled and represents, for each resource type ρ, the total amount of resource ρ requested by the *minimum value* working mode. The denominator of the second factor of τ computes, for each resource type, the ratio between *resource offer* $\phi[\rho]$, that corresponds to the amount of free resources, divided by a “clustering factor” $\kappa[\rho]$, that represents the number of clusters for the specific resource type ρ and for non clustered resources is equal to 1.

4 Experimental Results

A number of experiments on an implementation of the proposed heuristic has been carried out to assess both memory and computational overheads. We considered a “system load phase”, where an increasing number of applications enter in the system, one at the time, to be scheduled for execution. We collected both memory consumption and computational time to find a solution; these values

Table 1. Description of the scenarios

DS	N_{be}	AWM	$R_1..R_{10}$
3	15	10	[75, 75, 75, 75, 75, 75, 75, 75, 75, 75]
4	20	10	[100, 100, 100, 100, 100, 100, 100, 100, 100, 100]
5	25	10	[125, 125, 125, 125, 125, 125, 125, 125, 125, 125]
6	30	10	[150, 150, 150, 150, 150, 150, 150, 150, 150, 150]
51	50	10	[207, 205, 203, 236, 187, 200, 232, 222, 235, 220]
52	50	10	[293, 295, 297, 264, 313, 300, 268, 278, 265, 280]

have been compared considering a C-coded heuristic and the optimal one obtained by solving the ILP model. The optimal model has been coded in MathProg language, using the open source GLPSOL linear programming solver. It is worth to notice that this formulation does not consider at all the previous state, thus disregarding migration and reconfiguration costs. In fact, it just schedules all the applications in one single step thereby finding the best allocation considering all the other parameters, e.g., applications priorities and resource clustering. This approach is more similar to the one used by most of the prior art contributions, which actually performs just a single-shot scheduling of all the applications.

For the experiments we used an updated version of the dataset [4] adopted also by Ykman-Couvreur et al. [12], where we added the information supported by our model, namely application priority, resource clustering and migration/reconfiguration overheads. We report the experimental results for six different datasets which are representative of very different load conditions. For each scenario, the number of applications and their working mode are summarized in Table 1, along with the number of resources available on the platform. The first resource (R_1 in Table 1) is considered to be clustered, thus the value reported by the table has been equally distributed among the available clusters. These scenarios have been analyzed considering two different clustering size. Moreover, three different metrics have been used to assess the quality of the proposed heuristic. Since we are dealing with a utility maximization problem, the first metric, called *ν degradation*, measures the gap, expressed in percentage, between the utility obtained by the heuristic solution with respect to the ILP one. The lower this value is the better the heuristic solution will be. The second metric (*memory*) considers the memory used to compute the solution and it is expressed in KB. Finally, *time* is a metric that measures the time used to find the solution and it is expressed in milliseconds.

Results are reported in Table 2 for 4 and 8 clusters[1]. The first column reports the scenery identifier, while the second one collects the evaluation metrics. The third column reports the results obtained by the heuristic with respect to the evaluation metrics. For every metric the mean and the standard deviation are given. The rightmost column reports the same results obtained by the ILP model. The experiments show that the proposed heuristic scales well under different clusterization factors; the profit degradation seems to be fairly independent on the number of clusters. Moreover, the heuristic allows to find a solution with reasonable processing time at very different load conditions, on a system with 4

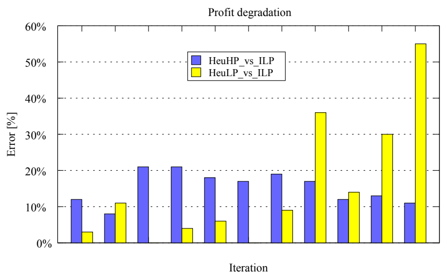
[1] Experiments with other clustering factors have been omitted due to lack of space.

Table 2. Synthetic results using 4 and 8 clusters

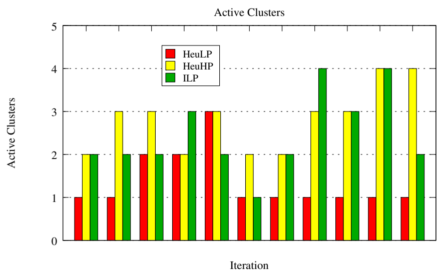
DS	METRICS	4 Clusters				8 Clusters			
		HEU		ILP		HEU		ILP	
		Avg	StdDev	Avg	StdDev	Avg	StdDev	Avg	StdDev
	ν degrad.	25%	5%	-	-	26%	11%	-	-
3	memory [KB]	$1.28 * 10$	6.58	$2.32 * 10^4$	$1.48 * 10^4$	$1.57 * 10$	7.93	$8.29 * 10^4$	$4.46 * 10^4$
	time [ms]	0.34	0.23	$2.49 * 10^4$	$4.77 * 10^4$	0.66	0.43	$5.57 * 10^4$	$5.65 * 10^4$
	ν degrad.	9%	5%	-	-	13%	5%	-	-
4	memory [KB]	$1.63 * 10$	8.35	$2.83 * 10^4$	$1.64 * 10^4$	$2.03 * 10$	$1.06 * 10$	$1.09 * 10^5$	$5.96 * 10^4$
	time [ms]	0.51	0.34	$2.45 * 10^4$	$4.78 * 10^4$	0.98	0.68	$4.40 * 10^4$	$5.64 * 10^4$
	ν degrad.	8%	10%	-	-	20%	9%	-	-
5	memory [KB]	$2.04 * 10$	$1.10 * 10$	$3.36 * 10^4$	$1.85 * 10^4$	$2.49 * 10$	$1.32 * 10$	$1.34 * 10^5$	$7.46 * 10^4$
	time [ms]	0.74	0.55	$6.31 * 10^3$	$2.36 * 10^4$	1.35	0.99	$5.33 * 10^4$	$5.71 * 10^4$
	ν degrad.	9%	10%	-	-	17%	9%	-	-
6	memory [kb]	$2.42 * 10$	$1.32 * 10$	$4.00 * 10^4$	$2.22 * 10^4$	$2.95 * 10$	$1.59 * 10$	$1.60 * 10^5$	$8.93 * 10^4$
	time [ms]	1.02	0.80	$1.00 * 10^4$	$2.87 * 10^4$	1.89	1.50	$7.14 * 10^4$	$5.71 * 10^4$
	ν degrad.	26%	6%	-	-	26%	5%	-	-
51	memory [KB]	$3.71 * 10$	$1.89 * 10$	$6.61 * 10^4$	$3.74 * 10^4$	$4.74 * 10$	$2.58 * 10$	$2.64 * 10^5$	$1.49 * 10^5$
	time [ms]	2.22	1.69	$5.05 * 10^4$	$5.68 * 10^4$	4.71	3.86	$9.29 * 10^4$	$4.91 * 10^4$
	ν degrad.	27%	2%	-	-	28%	2%	-	-
52	memory [KB]	$3.94 * 10$	$2.19 * 10$	$6.57 * 10^4$	$3.71 * 10^4$	$4.78 * 10$	$2.65 * 10$	$2.64 * 10^5$	$1.49 * 10^5$
	time [ms]	2.52	2.15	$4.35 * 10^4$	$5.75 * 10^4$	4.76	4.03	$8.01 * 10^4$	$5.52 * 10^4$

and 8 clusters. The results show that our approach is able to find a solution in the order of few tens of milliseconds, even for relatively high load scenarios with 50 applications. The memory required is also always less then 100KB and such value is almost uncorrelated with the number of clusters. These numbers make our solution perfectly suitable for the usage on a real system, whereas the optimal model could not be profitably applied due to the huge memory requirements, which easily goes over the tens of MB, and the prohibitive processing time, which is in the order of minutes. Moreover, the *profit degradation*, i.e. how far the value of the approximate solution is from the value of the solution computed by ILP, is always lower that 30% which makes it perfectly acceptable if we also consider the negligible overheads. It is worth to notice that the ILP problem disregard migration and reconfiguration costs, thus the profit degradation we measured is a conservative estimation of the real one.

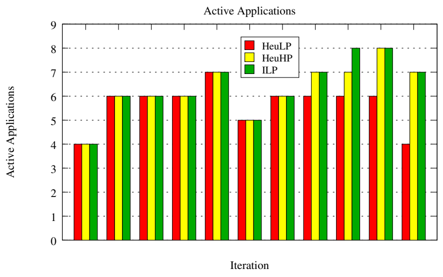
A second set of measurements evaluates a more realistic scenario where applications arrive and leave the system at different times and the RTM generates a new optimal allocation for every event, namely the arrival or the exit of one or more applications from the system. This scenario considers a system with 10 PEs for each of the 4 clusters and a total of 20 applications, five of which are considered critical, where details for each application are not reported for sake of brevity. A tuning parameter, which allows to trade application performance against resource usage, allows us to evaluate two different scheduling policies for our heuristic: HeuLP and HeuHP. The former limits resource usage, eventually reducing application performance by scheduling low-value working modes, while the latter focuses on application performance with respect to resource preservation. Both HeuLP and HeuHP implement the heuristic described in Section 3.3, however HeuLP has a control at the end of each cluster (k) allocation, that prevent the scheduled applications (A) on the cluster k, to be allocated to cluster (k) itself according to the equation $\sum_{i \in A} \nu_i < \Theta_k$. This control prevents allocation if the cost to switch on the cluster (Θ_k) is greater than the total utility gained by scheduling the set of application A ($\sum_{i \in A} \nu_i$). The evaluation of the



(a) profit degradation



(b) active clusters



(c) scheduled applications

Fig. 4. Realistic scenario results showing profit degradation (a), number of active clusters (b) and number of scheduled applications (c) using two different scheduling policies, HeuLP and HeuHP, and comparing their results against the optimal ILP solution

solution generated by both HeuLP and HeuHP has been assessed by three different metrics. As in the previous experiments, we use ν *degradation* to evaluate the gap between the utility reached by the heuristic solution with respect to the ILP one (see Fig. 4-a). The second metric considers the number of active clusters (see Fig. 4-b). The last metric counts the number of active applications for the solution (see Fig 4-c). Since we assume that only critical applications have guaranteed execution, it is useful to monitor the trade off between resource saving and the rejected best-effort applications.
Of course, the two heuristic flavors exhibit very different behaviors in terms of scheduled applications, resources usage and profit degradation. HeuLP is more prone to reject applications thus paying an higher profit degradation (Fig.4-a), especially towards the last iterations. Indeed, we can see that (Fig. 4-b and Fig. 4-c) in the really last iteration this scheduling policy uses less clusters with respect to HeuHP (1 instead of 4) which in turns could accommodate for only a reduced number of applications (4 instead of 7). This allows to keep powered on a reduced number of clusters, a condition which could be eventually exploited by a energy saving policy. On the other hand, the HeuHP policy shows a more regular profit degradation (Fig.4-a) which is also quite small and assesses around 15% in average.

5 Conclusions

Both a novel model and a corresponding heuristic for run-time resource allocation has been presented. The formal model captures some new issues emerging from both new complex architectures and applications adaptivity needs, i.e., clustered resources, application priorities, migration and reconfiguration costs. The proposed heuristic builds a near optimal solution to the allocation problem and experiments reveal some valuable proprieties. First of all the proposed heuristic approach can manage up to fifty applications with a negligible computational time limited to tens of milliseconds while also memory usage is limited to 100KB. Moreover, results show a good scalability using a different clusterization factors.

References

1. Acquaviva, A., Alimonda, A., Carta, S., Pittau, M.: Assessing Task Migration Impact on Embedded Soft Real-Time Streaming Multimedia Applications. EURASIP Journal on Embedded Systems, 1–16 (2008)
2. Geilen, M., Basten, T., Theelen, B., Otten, R.: An Algebra of Pareto Points. In: IEEE ACSD 2005, pp. 88–97 (2005)
3. Hifi, M., Michrafy, M., Sbihi, A.: A Reactive Local Search-Based Algorithm for the Multiple-Choice Multi-Dimensional Knapsack Problem. Computational Optimization and Applications 33(2-3), 271–285 (2005)
4. Hifi, M.: Benchmark Data, `http://www.laria.u-picardie.fr/hifi/OR-Benchmark/MMKP/`
5. Khan, S., Li, K.F., Manning, E.G., Akbar, M.M.: Solving the Knapsack Problem for Adaptive Multimedia Systems. Studia Informatica Universalis 2(1), 157–178 (2002)
6. Khan, S.: Quality Adaptation in a Multisession Multimedia System: Model, Algorithms and Architecture. Ph.D. thesis, University of Victoria (1998)
7. Nollet, V., Verkest, D., Corporaal, H.: A Safari Through the MPSoC Run-Time Management Jungle. Journal of Signal Processing Systems (2008)
8. Raina, S.: Introduction to Quality of Service (QoS). In: Telecommunications Quality of Service: The Business of Success (QoS 2004), pp. 48–51 (2004)
9. Shahriar, A.Z.M., Akbar, M.M., Rahman, M.S., Newton, M.A.H.: A multiprocessor based heuristic for multi-dimensional multiple-choice knapsack problem. The Journal of Supercomputing 43(3), 257–280 (2007)
10. Shojaei, H., Ghamarian, A.H., Basten, T., Geilen, M., Stuijk, S., Hoes, R.: A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for CMP run-time management. In: 46th ACM/IEEE DAC 2009, pp. 917–922 (2009)
11. Xu, P., Michailidis, G., Devetsikiotis, M.: Profit-Oriented Resource Allocation Using Online Scheduling in Flexible Heterogeneous Networks. Accepted by Telecommunication Systems (2005)
12. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. In: International Symposium on System-on-Chip, pp. 1–4. IEEE (November 2006)