FP7 – 216693 - MULTICUBE Project

MULTI-OBJECTIVE DESIGN SPACE EXPLORATION OF MULTI-PROCESSOR SOC ARCHITECTURES FOR EMBEDDED MULTIMEDIA APPLICATIONS

Deliverable D3.3.1: Initial report on run-time component for task allocation and scheduling based on Pareto information

[Revision 7]

Delivery due date: M24 (December 2009)
Actual submission date: February 1st, 2010
Lead beneficiary: IMEC

Dissemination Level of Deliverable		
PU	Public	X
PP	Restricted to other program participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	
Nature of Deliverable		
R	Report	X
P	Prototype	
D	Demonstrator	
O	Other	

Author(s):	Chantal Ykman-Couvreur, Giovanni Mariani		
Reviewer(s):	Prabhat Avasare, Gianluca Palermo		
WP/Task No:	3.3	**Number of pages:**	22
Identifier:	D3.3.1_IMEC_12.2009	**Dissemination level:**	Public
Issue Date:	Febraury 1st, 2010		

Keywords: Prototype, Open-Source, Exploration

Abstract: In the context of MULTICUBE, we describe in this deliverable D3.3.1 the exploration framework we designed for a run-time resource management. This deliverable presents the results of the activities carried out from M18 to M24 in Task 3.3 under the leadership of IMEC.

The main goal of this task is to prove the feasibility of the exploration framework, which uses the Pareto information from the design-time exploration to optimize the run-time task allocation and scheduling.

This deliverable presents the initial analysis performed for the usage of design-time exploration information by the run-time resource management.

In particular, the designed framework consists of two phases:

- First, a design-time exploration per application to derive a multi-dimensional Pareto set of optimal configurations. This exploration makes use of the design space explorers and of the MP-SoC platform simulators developed in MULTICUBE.
- Second, a low-complexity run-time manager addressing the challenges imposed by future multi-core embedded platforms. This run-time manager is based on a very fast optimization strategy to globally select optimal configurations for the active applications, according to the available platform resources, in order to minimize the platform power consumption, while satisfying the constraints imposed by the external word or the user.

In the next deliverable D3.3.2 related to the same task 3.3, we intend to describe the experiments and results obtained by the proposed exploration framework (considering both design-time analysis and run-time management) applied to a real-life embedded multi-media application.

Approved by the Project Coordinator: 	**Date:** Febraury 1st, 2010

Table of contents

I. Executive summary

In the context of MULTICUBE, we describe in this deliverable D3.3.1 our exploration framework for a Run-time Resource Management (RRM). This framework consists of two phases:

- First, a design-time exploration per application to derive a multi-dimensional Pareto set of optimal configurations. This exploration makes use of the design space explorers and of the MP-SoC platform simulators developed in MULTICUBE.
- Second, a low-complexity run-time manager addressing the challenges imposed by future multi-core embedded platforms. This run-time manager is based on a very fast optimization strategy to globally select optimal configurations for the active applications, according to the available platform resources, in order to minimize the platform power consumption, while satisfying the constraints imposed by the external word or the user.

The contents of the deliverable D3.3.1 is organized as follows. First, an introduction is given to motivate the need of an alleviated RRM using the Pareto information derived by a design-time exploration. Then, the state of the art in this context is overviewed to understand the added value of the proposed framework. For the sake of clarity, all assumptions and terminologies used in the deliverable are listed, and the RRM problem is defined, before presenting the exploration framework.

II. Introduction

The future of embedded computing is shifting to multi-core designs to boost performance due to the unacceptable power consumption and operating temperature increase of fast single-core CPU's. This introduces new big challenges.

The first challenge is the support for a variety of applications: mobile communications, networking, automotive and avionic applications, multimedia in the automobile, Internet interfaced to many embedded control systems. These applications may run concurrently, start and stop at any time. Each application may have multiple configurations, with different constraints imposed by the external word or the user (deadlines and quality requirements, such as audio and video quality, output accuracy), usages of various types of platform resources (processing elements, memory and communication), and costs (performance, power and energy consumption).

The second challenge is the platform heterogeneity, happening between platforms and within a platform. Even for similar platforms, process variation endows them with different performance characteristics. Furthermore the resources required for each application may vary over time, as applications are launched or complete, and due to hardware adaptation to physical constraints (power, temperature, battery life, and aging). Hence, it is untenable to ask software vendors to adapt or optimize their applications for each platform.

Finally, the challenge of time to market, which makes software development productivity of paramount importance.

The natural way to address these challenges is to follow a platform-based design methodology [19] and to develop and automate a Run-time Resource Manager (RRM) fulfilling the following features:

- A **holistic view of platform resources and management of diverse quality aspects** is needed:
  - For global resource allocation decisions, arbitrating between all applications, and optimizing a utility function (also called Quality of user Experience (QoE)), given the available resources. This QoE models the user benefit for all applications. It allows trade-off, negotiated with the user, between diverse quality aspects and costs.
  - For concurrently dealing load balancing and communication [17].
  - For careful management of the energy stored in the battery since power consumption, processor execution speed and timeliness of applications are interdependent.
  - For failure and anomaly detection and recovery.
- **Transparent** optimization of resource usage and application mapping on the platform allows to facilitate the application development and mapping from diverse application domains. It also allows managing the quality requirements without rewriting the applications.
- **Adaptivity** enables the best usage of resources and achieves a high efficiency under changing environment and run-time requirements. To that end:

  - Dynamic resource allocation and dynamic reconfiguration of applications must be supported.
  - Quality requirements and resources must be scaled dynamically (e.g., by adjusting the processor clock frequency, or by switching off some functions) in order to control the power consumption and the heat dissipation of the platform.
- **System-level power management** must be included:
  - As platform components get smaller, high die temperature can be reached, and the negative impact on the hardware reliability should be kept under control.
  - In battery-powered devices, functions with high power consumption should be turned off when the battery status gets low, in order to save energy for functions with low power consumption and though high user value.
  - To minimize the power consumption of the platform, while still meeting the quality requirements.
- Since such an RRM is intended for embedded platforms, a **lightweight implementation** only is acceptable. To alleviate the run-time decision making and to avoid conservative worst-case assumptions, this RRM should consist of two phases [40]:
  - First, a **design-time exploration** per application to derive a multi-dimensional Pareto set of optimal configurations. Each configuration is characterized by a code version together with an optimal combination of required constraints, used platform resources, and costs. The different code versions refer to different parallelizations of the application into parallel tasks and data transfers to shared and local memories. These optimal configurations are stored once within the OS before running the application.
  - Second, a **low-complexity run-time manager**, incorporated on top of the basic services of the platform OS, and acting as an exception handler. Whenever the environment is changing (e.g., when a new application/use case starts, or when the user requirements change), for each active application, the run-time manager reacts as follows:
    - It selects a configuration from its Pareto set, according to the available platform resources, in order to minimize the costs, while satisfying the constraints. This selection is a combinatorial problem, called Multi-dimension Multiple-choice Knapsack Problem (MMKP), and belonging to the NP-hard class [13].
    - It reconfigures and maps the application software on the platform. I.e., it assigns the platform resources, it adapts the platform parameters, it loads the application tasks, and it issues the application execution according the newly selected configuration.

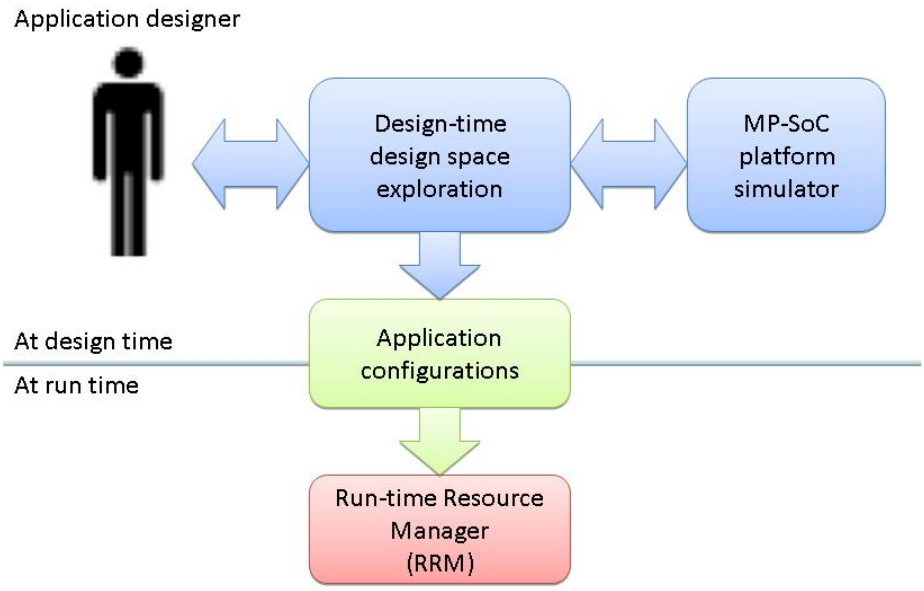

Figure 1: Exploration framework for the RRM in MULTICUBE

In the context of MULTICUBE, an exploration framework, depicted in Figure 1, and fulfilling these two phases of the RRM, is developed:

- For each application, a set of optimal configurations is identified at design time, by analyzing and exploring parallelization and quality impacts on the constraints, the platform resource usage, and the costs. This can be done through one of the design space explorers coupled with platform simulators at multiple abstraction levels:
  - The available design space explorers are:
    - *modeFRONTIER* [2], commercially available and extended in the project.
    - *Multicube Explorer* [3], developed in the project.

    As emphasized in [14] and also previously mentioned in this section, these design-space exploration tools are needed within the embedded system design cycle. They allow to alleviate the run-time decision making and to avoid conservative worst-case assumptions.
  - The available MP-SoC platform simulators are:
    - *HLSim* [4], a fast high-level timed simulator. This simulator can be used for extensive explorations. This simulator reports execution time and power consumption estimations of the platform running the application with a given configuration. For these estimations, a TSMC 90nm process technology is assumed.
    - *Scope* [9], a SystemC platform model for behavioral simulation and performance estimation.
    - *TLMsim*, a SystemC-based cycle-accurate transaction-level model, built using the CoWare virtual platform prototyping tools [1]. This simulator

consumes more time, but it is more accurate. It can be used for selective explorations on relevant regions of the design space.

As pointed in [8], platform simulations can be done at many abstraction levels: e.g., functional-level simulation, timed simulation, cycle-accurate simulation. Recently, simulations based on a cycle-accurate Transaction-Level Model (TLM) have gained importance due to standardization efforts. But the key problem of all these simulation levels is the following one: the more accurate the simulator, the more time it takes to perform a simulation. Hence from the design-space exploration point of view, which needs a large number of simulations, there is an important trade-off between the result accuracy and the simulation time.

- The run-time manager runs on one available processor of the platform. As optimization strategy to globally select optimal configurations for the active applications, it extends a fast MMKP heuristic for MP-SoC run-time management, presented in [39]. As new contribution to this heuristic, in case of soft real-time applications, and when the optimization strategy can find no solution, the run-time manager also takes the application priorities into account in order to relax the constraints and to reach a solution after all. Another new contribution is the integration at design time of further filtering and sorting of available configurations. This allows alleviating the run-time selection heuristic.

The remainder of the deliverable is organized as follows. Section III overviews the state-of-the-art on run-time resource management for embedded multi-core platforms. Section IV introduces all assumptions and notations used in this deliverable. Section V formulates the RRM problem, whereas Section VI presents our exploration framework to solve this RRM problem.

III. Related work on RRM

In the context of RRM, traditional approaches can be roughly classified into either pure design-time approaches or pure run-time approaches. Nevertheless, they suffer from the following drawbacks:

- Some of them are applicable only for single-processor platforms [35], or for homogeneous multi-processor platforms [42], but not for heterogeneous multi-processor platforms.

- None of the existing approaches proposes a complete framework:

    - The ones are based only on task scheduling, i.e. on task ordering and assignment. A good overview of available design-time algorithms can be found in [31].

    - Some others are based only on slowing or shutting down the platform resources [6] and on Dynamic Voltage and Frequency Scaling (DVFS) [11, 18, 24, 30].

- The objective of the majority of these approaches is performance optimization [5, 7, 10, 20, 23], and not power consumption minimization.

- Design-time approaches involve slow heuristics [30, 32, 33] using ILP and cannot be used at run time.

- To reach a lightweight implementation, run-time approaches hide the specification of the internal application tasks, and they don't fully exploit the task mapping choices of the target platform. Hence these approaches are sub-optimal.

Hence neither the existing pure design-time approaches nor the existing pure run-time approaches are efficient to solve this complex RRM problem. To alleviate the run-time decision making and to avoid worst-case assumptions, new research directions are ongoing and propose a mixed design-time and run-time approach:

- The Task Concurrency Management (TCM) methodology, proposed in [36, 37, 38], explores the energy-performance trade-off at the system level. To reach an efficient usage of the platform resources, this methodology models the application at a finer granularity than traditional task graphs. It identifies the sub-tasks of the application that can run in parallel on a heterogeneous multi-processor platform. It also includes data access and memory management at the task level [25,40].

- Scenario-based approaches, proposed in [16, 27], is based on the concept of application scenarios identified at design time, as follows. First, a profiling-based analysis of various run-time situations of the application is performed. Then, these run-time situations are clustered into a few dominant application scenarios and a backup

scenario. At run time, the actual scenario is detected with a simple detector, and the application is executed with the configuration decided for that scenario.

- The task scheduling techniques proposed in [36, 37] schedule one task at a time, making use of the entire platform for each task. This leads to inefficient usage of the platform. The techniques proposed in [39, 34] allow parallel execution of the tasks, while sharing the platform resources. But they assume that all tasks start at the same time. This assumption can lead to idle platform processors until the next RRM call. These techniques are extended in [28], which allows overlapped sharing of the platform resources. This latter technique considers the start time and the periodic information present in applications such as frame processing in video decoding, or packet processing in wireless applications.

- RRM for multi-core embedded platforms seeks to extend the previous approaches by exploiting the number of available cores, in addition to voltage and frequency. In particular, different parallelized versions of a single application can be used to trade the available platform resources with the performance and the power consumption [21, 22].

- The addition of the parallelism to the set of controllable platform parameters significantly increases the design space of operating modes. Innovative and efficient techniques for RRM are needed to extend the traditional approaches for power consumption optimization. Recent studies in this field address the problem by modeling it as an MMKP and solving it through dedicated heuristics [39, 34]. Another approach [26] proposes a run-time management technique for task-level parallelism in order to optimize the performance under a power consumption budget.

- Advanced technologies such as sub-45nm CMOS and 3D integration are known for the increased number of reliability failure mechanisms. Nevertheless, classical reliability-aware approaches are no longer viable, since they propose ad-hoc failure or worst-case solutions, which incur a significant cost penalty. In [29], the state of the art in reliability management techniques is summarized, and a new proactive energy management approach is proposed, which handles both temperature and lifetime at run time.

To allow integration and collaboration of all these complementary techniques, a global run-time resource management framework for cross-domain embedded multi-core platforms has been developed [41], with the most relevant generic services.

The goal of the MULTICUBE task 3.3 is to propose a feasible exploration framework for a run-time resource management, combining the design-time exploration tools developed in MULTICUBE with a low-complexity run-time manager.

IV. Assumptions and terminologies

In this section the assumption and terminologies used in the next sections are listed, in order to formally define the run-time management problem and describe the exploration framework.

Platform assumptions:

- The target platform is a homogeneous MP-SoC platform.
- The platform resources being currently considered are: the processors and their possible clock frequencies that can be dynamically scaled independently from each other.
- The set of possible clock frequencies per processor is Φ.
- The platform consists of ρ^{max} processors.

RRM assumptions:

- The constraints to be satisfied are the application deadlines.
- The cost to be minimized is the total power consumption of the platform.

Application assumptions:

- p applications are simultaneously active on the platform. Both p and active applications can change at run time.
- Each active application receives an identifier $\alpha \in A = \{\alpha_1, \ldots, \alpha_p\}$.
- The deadline of an application α is τ_α^{max}. This deadline can change at run time by the external world or the user.
- The priority of an application α is ω_α.

Application configuration characterization:

- For each application α, any configuration is characterized by the tuple

  $$c_\alpha = <\rho, \underline{\phi}, \pi, \tau>$$

  where:
  - $\rho \leq \rho^{max}$, or $c_\alpha[\rho]$, is the number of needed processors.
  - An application binary, being parallelized over ρ processors, is available.
  - $\underline{\phi}$, or $c_\alpha[\underline{\phi}]$, $= <\phi_1, \ldots, \phi_\rho>$ is the combination of clock frequencies for each of the ρ processors.
  - π, or $c_\alpha[\pi]$, is the average platform power consumption associated with c_α.
  - τ, or $c_\alpha[\tau]$, is the average execution time to run the application binary corresponding with c_α.
- For each application α, the set of available configurations c_α is C_α, and its size is $N_\alpha \leq N^{max}$, where N^{max} is the maximum number of configurations in any C_α.

V. Run-time management problem definition

The run-time manager has to select exactly one configuration from each active set C_α, according to the available platform resources, in order to minimize the total power consumption of the platform, while satisfying the application deadlines.

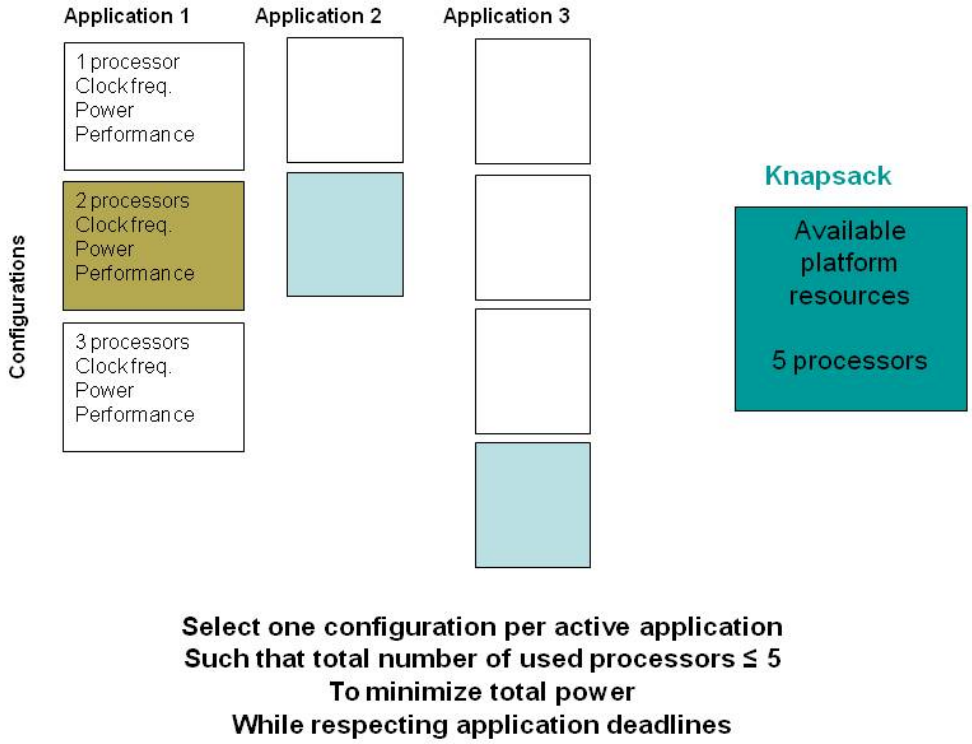

Figure 2: Run-time management problem

This problem, depicted in Figure 2**,** can be mathematically formulated as follows:

- Given p active applications $\alpha_1, \ldots, \alpha_p$ with deadlines $\tau_{\alpha 1}^{max}, \ldots, \tau_{\alpha p}^{max}$, and set of available configurations $C_{\alpha 1, .., } C_{\alpha p}$
- Identify a set of configurations

  $$S = \{c_{\alpha 1} \in C_{\alpha 1}, \ldots, c_{\alpha p} \in C_{\alpha 1}\}$$

  in order to minimize the total power consumption:

  $$\Sigma_{\alpha \in A}\, c_\alpha[\pi]$$

  according to the resource constraint:

  $$\Sigma_{\alpha \in A}\, c_\alpha[\rho] \leq \rho^{max},$$

and to the application deadlines:

$$c_{\alpha 1}[\tau] \leq \tau_{\alpha 1}{}^{max},$$
$$\ldots,$$
$$c_{\alpha p}[\tau] \leq \tau_{\alpha p}{}^{max}.$$

According to [13], this problem is an MMKP belonging to the NP-hard class with respect to ρ^{max}, p, and $N_{\alpha 1}, \ldots, N_{\alpha p}$.

Dependent on the application deadlines, this problem can have no solution. In case of soft real-time applications, for which the deadline can be missed with the lowest penalty possible, their priority is taken into account in order to relax the deadlines and to reach a solution after all (see Algorithm 2).

VI. Exploration framework for RRM

Our exploration framework for RRM, depicted in Figure 1, combines a design-time exploration of the design space of any application with a lightweight RRM. This run-time manager is incorporated on top of the basic services of the platform OS. It runs on one available processor of the platform.

In the remainder of this section, first, we describe our design-time and run-time heuristics, and we analyze their complexities. Then, we describe the interface which allows us to combine the design-time exploration with the RRM.

VI.1. Design-time generation of C_α

VI.1.1. Heuristic description

Algorithm 1 performs a design-time exploration of configurations for any application α, and it derives a Pareto set of optimal ones, according to the available platform resources, the corresponding power consumption of the platform, and the execution time of the application.

1. **for** each $\alpha \in A$ **do**
2. C_α = empty set
3. **for** each $\rho \leq \rho^{max}$ **do**
4. C_α^{new} = multi-objective exploration (α, ρ, Φ) with optimization objectives == minimize both π and τ
5. Append C_α^{new} to C_α
6. **end for**
7. Pareto filtering of C_α with optimization objectives == minimize ρ, π, and τ
8. **end for**

Algorithm 1: Design-time generation of C_α

This algorithm works as follows:

- **Lines 1 and 2.** The algorithm iterates over the applications and the number of used processors per application.
- **Line 3.** For any application α using ρ processors, the algorithm explores the clock frequency combinations $\underline{\phi}$ (one clock frequency per used processor) and derives the set of Pareto configurations C_α^{new} according to the power consumption π and the execution time τ of α.
    - The size of the design space to be explored rapidly increase with ρ and with the number of possible clock frequencies. Therefore, an exhaustive exploration for each ρ is not possible, and the exploration is performed by one of the multi-

objective optimization strategies developed either in modeFRONTIER or in Multicube Explorer. Currently:

- For $\rho \leq 3$, an exhaustive exploration is performed.
- For $\rho \geq 4$, the exploration is performed by the NSGA-II multi-objective genetic algorithm [15]. For each ρ, the genetic algorithm runs with a population size $\lfloor |\Phi| \times \rho \rfloor$ for *ngen* generations, resulting for the worst case in $ngen \times \lfloor |\Phi| \times \rho \rfloor$ simulations.

o π and τ are estimated through one platform simulator, either HLSim, or Scope, or TLMsim.

o Any Pareto clock frequency combination derived from the exploration determines a new configuration $c_\alpha = <\rho, \phi, \pi, \tau>$. This configuration is added to the set C_α^{new}.

- **Line 5**. This new set of configurations C_α^{new} is appended to the current C_α.
- **Line 7**. At the end of the exploration, the set of configurations C_α is further filtered to only keep the Pareto configurations according to the number of used processors too.

VI.1.2. Complexity analysis

It is worth mentioning that the processing for one iteration over a given application parallelization (line 3 in Algorithm 1) is completely independent of the processing for the other parallelizations. Thus, to save exploration time, all iterations can be performed in parallel. Since every iteration mainly consists of a multi-objective optimization (with the required platform simulations), the worst-case execution time for processing one application can then be approximated as the worst-case execution time of the multi-objective optimization for one parallelization, which is
$O(ngen \lfloor |\Phi| \rho)$.

VI.2. Design-time pre-processing

VI.2.1. Description

To improve the performance of the run-time decision making described in Section VI.3, and to reduce the effort needed to indentify the optimal application configurations, the following steps are performed at design time.

Step 1 looks in each set C_α for the configuration with the highest power consumption π. Let c^0_α denotes this configuration.

Step 2 derives the value $c_\alpha[\nu] = c^0_\alpha[\pi] - c_\alpha[\pi]$ for each configuration c_α in C_α. Indeed, the run-time heuristic will maximize the total value $\Sigma_{\alpha \in A} c_\alpha[\nu]$ (see Eq. 1 page 16), instead of minimizing the total power consumption $\Sigma_{\alpha \in A} c_\alpha[\pi]$.

Step 3 sorts the configurations of each set C_α in ascending order according first to their resource usage ρ and then to their value ν. First, this step allows reducing Step2 of the run-time heuristic to a linear scan of each active set C_α. Second, this step makes the selection of the initial solution easier in Step3 of the run-time heuristic.

VI.2.2. Complexity analysis

The complexity analysis of each step and per application is as follows:

- The complexity of both **Steps 1 and 2** is $O(N^{max})$, where N^{max} is the maximum number of configurations in any C_α.
- **Step** 3 is a sorting. From [12], its worst-case complexity is $O(N^{max} \log(N^{max}))$.

VI.3. Run-time selection of c_α

VI.3.1. Heuristic description

The optimization strategy used in the run-time manager is a fast MMKP heuristic for MP-SoC run-time management, adapted from [39]. The problem solved by this heuristic is defined in Section V, where the sets of configurations $C_{\alpha 1, \ .., \ } C_{\alpha p}$ is derived by the design-time exploration described in Section VI.1. Different steps of this heuristic are described below.

Step 1 removes in each active set C_α any configuration c_α whose execution time exceeds the application deadline, i.e. such that:

$$c_\alpha[\tau] > \tau_\alpha{}^{max}.$$

The minimization problem defined in Section V becomes the new maximization problem:

- Given p active applications $\alpha_1, \ldots, \alpha_p$ and set of available configurations $C_{\alpha 1, \ .., \ } C_{\alpha p}$
- Identify a set of configurations

  $$S = \{c_{\alpha 1} \in C_{\alpha 1}, \ldots, c_{\alpha p} \in C_{\alpha 1}\}$$

  in order to maximize the total value:

  $$\Sigma_{\alpha \in A} \, c_\alpha[\nu] \text{ (Eq. 1)}$$

  according to the resource constraint:

  $$\Sigma_{\alpha \in A} \, c_\alpha[\rho] \leq \rho^{max} \text{ (Eq. 2).}$$

Step 2 further filters each active set C_α updated from Step 1 as follows. All remaining configurations in C_α satisfy the application deadlines. Some of them are Pareto optimal due to their low execution time, at the expense of using more resources or consuming more power than other configurations in C_α. For our minimization problem, these configurations are undesirable and are removed from C_α too.

Concretely, the goal of Step 2 is, for each active set C_α and for each number of used processors ρ, to identify the combination of clock frequencies ϕ that minimizes the corresponding power consumption π, while satisfying the application deadline. Hence the resulting size of C_α is ρ^{max}.
Due to the sorting of C_α performed by Step 2 during the design-time pre-processing, this step only consists of a linear scan of C_α.

Step 3 sorts all configurations of all active sets C_α indifferently. These configurations are sorted in descending order according to their coefficient angular ($c_\alpha[\nu] / c_\alpha[\rho]$) in the two-dimension space (value versus resource usage). This implies that a high priority is given to configurations with high value and low resource usage. The sorted set of all configurations, resulting from Step 3, is denoted *C_sorted*.

1. ***cur_sol*** ← initial solution
2. **if** *feasible(cur_sol)*
3. *saved_sol* ← *cur_sol*
4. **for** each configuration c_α in *C_sorted*
5. *ExchangeConfiguration*(c_α)
6. **if** *feasible(cur_sol)* and *cur_val* > *saved_val*
7. *saved_sol* ← *cur_sol*
8. **end if**
9. **end for**
10. **else** no feasible solution exists

Algorithm 2: Greedy algorithm solving the knapsack problem

Step 4 is a greedy algorithm solving the knapsack problem (see Algorithm 2). In the pseudo-code of this algorithm, the following terminology is used:

- A *solution* is a set of configurations c_α, one per active application α. A *feasible solution* is a solution that satisfies the resource constraint ($\Sigma_{\alpha \in A} c_\alpha[\rho] \leq \rho^{max}$ (Eq. 2) in the definition of the maximization problem). An *optimal solution* is a feasible solution with a maximum total value ($\Sigma_{\alpha \in A} c_\alpha[\nu]$ (Eq. 1)).
- *cur_sol* (resp. *saved_sol*) denotes the current knapsack solution (resp. saved feasible solution) including the configurations c_cur_α (resp. c_saved_α).
- The Boolean variable *feasible(cur_sol)* indicates whether *cur_sol* is feasible or not.
- The Boolean variable *saved* becomes true once a feasible solution is found.
- *cur_val* (resp. *saved_val*) denotes the total value of *cur_sol* (resp. *saved_sol*).

- The initial solution includes the configuration from each set C_α first with the lowest resource usage and then with the lowest value. Starting with the lowest-resource-usage configurations guarantees that the algorithm always returns a feasible solution when one exists. Starting with the lowest-value configurations allows the greedy algorithm to perform better in finding a solution close to an optimal one.

- The procedure *ExchangeConfiguration*(c_α) updates *cur_sol* by exchanging c_cur_α against c_α and by updating both total resource it uses and total value. However this exchange is ignored if *cur_sol*, initially feasible, would become infeasible.

VI.3.2. Complexity analysis

The complexity analysis of each step is as follows:

- Both **Step 1** and **Step 2** can be combined. Their global complexity is $O(p\ N^{max})$, where p is the number of active applications and $N_\alpha \leq N^{max}$ is the number of configurations in C_α.
- **Step 3** is again a sorting. Hence From [12], its worst-case complexity is $O(p\ \rho^{max} \log(p\ \rho^{max}))$.
- **Step 4** consists of iterating over the whole *C_sorted* and has a linear complexity $O(p\ \rho^{max})$.

Hence the run-time selection heuristic has an overall worst-case complexity of only:

$$O(p\ N^{max} + p\ \rho^{max} \log(p\ \rho^{max})),$$

due to the adequate filtering and sorting performed before the greedy algorithm solving the knapsack problem.

VI.3.3. Deadline relaxation

Whenever no feasible solution exists in the heuristic of Section VI.3.1, the deadline of the active application with the lowest priority is relaxed, and the heuristic is performed again on the updated set of active applications.

The deadline relaxation is performed as follows:

- Let $\underline{\alpha}$ denotes the active application with the lowest priority.
- Look in $C_{\underline{\alpha}}$ for the configurations with a minimum ρ (i.e., with a minimum number of needed processors).
- Among these configurations, detect the minimum execution time τ.
- Relax the deadline $\tau_{\underline{\alpha}}^{max}$ with this the minimum execution time τ.

VII. Interface between design-time exploration and RRM

The interface between the design-time exploration and the RRM is performed with XML. It is similar to the ones explained in [3].

- For each application, the design-time exploration exports a Pareto set of optimal configurations, according to the used platform resources, the corresponding power consumption of the platform, and the execution time of the application.
- Each configuration is also characterized by a combination of clock frequencies, one for each needed processor.
- Whenever an application is loaded on the platform, the RRM reads the XML file describing all its configurations exported by the design-time exploration.

The content of such an XML files, relatively to one configuration, looks as follows:

```
<points xmlns="http://www.multicube.eu/" version="1.3">
	<point>
		<parameters>
			<parameter name="Clk1" value="20" />
			<parameter name="Clk2" value="20" />
			<parameter name="Clk3" value="20" />
			<parameter name="Clk4" value="20" />
			<parameter name="Clk5" value="20" />
			<parameter name="Clk6" value="20" />
			<parameter name="Clk7" value="20" />
			<parameter name="threads" value="2" />
		</parameters>
		<system_metrics>
			<system_metric name="Execution_Time" value="7.072556" />
			<system_metric name="Power_Consumption" value="11.0288"
/>
			<system_metric name="Resources" value="2.0000" />
		</system_metrics>
	</point>
	...
</points>
```

In this XML example, it is assumed that:

- A point represents an application configuration.
- The number of platform processors is 7.
- For this configuration, the application is parallelized into 2 threads, and hence needs 2 processors.
- The clock frequency of each processor is 20 MHz.

VIII. References

1. CoWare Virtual Platform Prototyping Tools. http://www.coware.com.
2. modeFRONTIER DSE Tool. http://www.esteco.com.
3. Multicube Explorer User Manual. http://home.dei.polimi.it/zaccaria/data/release_0_5/docs/pdf/user_guide.pdf, Chapter 6, pp. 15-20.
4. R. Baert, E. Brockmeyer, S. Wuytack and T.J. Ashby. Exploring Parallelizations of Application for MPSoC Platforms using MPA. *Proceedings of IEEE DATE,* Nice, France, April 2009.
5. T.P. Baker. An Analysis of EDF schedulability on a multiprocessor. *IEEE Transactions on Parallel and Distributed Systems*, 2005
6. L. Benini, R. Bogliolo and G. De Micheli. A Survey of Design Techniques for System-Level Dynamic Power Management. *IEEE Transactions on VLSI Systems*, 8, pp. 299-316, 2000.
7. A. Buchard. *Assigning Real-Time Tasks to Homogeneous Multiprocessor Systems.* Technical Report, University of Virginia, Charlottesville, VA, USA, 1994.
8. L. Cai and D. Gajski. Transaction Level Modeling: an Overview. *Proceedings of CODES+ISSS*, October 2003.
9. J. Castillo, V. Fernandez, H. Posadas, D. Quijano and E. E. Villar. SystemC Platform Modeling for Behavioral Simulation and Performance Estimation of Embedded Systems. *Behavioral Modeling for Embedded Systems and Technologies: Application for Design and Implementation*, IGI International ed., to be published.
10. H.L. Chan. Non-Migratory Online Deadline Scheduling on Multiprocessors. *Proceedings of SODA*, pp. 970-979, 2004.
11. J.J. Chen, C.Y. Yang, T.W. Kuo and C.S. Shih. Energy-Efficient Real-Time Task Scheduling in Multiprocessor DVS Systems. *Proceedings of IEEE ASP DAC*, pp. 342-349, Yokohama, Japan, January 2007.
12. T. Cormen, C. Leiserson, R. Rivest. *Introduction to Algorithms*. The MIT press, McGraw-Hill, 1990.
13. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, January 1979.
14. M. Gries. Methods of Evaluating and Covering the Design Space during Early Design Development. *Integration, the VLSI journal*, Elsevier, 38(2), pp. 131-183, December 2004.
15. K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pp. 849-858, 2000.
16. S. Gheorghita, M. Palkovic, J. Hamers, A.Vandecappelle, S. Mamagkakis, T. Basten and L. Eeckhout. *System-Scenario-Based Design of Dynamic Embedded Systems.* ACM Trans. On Design Automation of Electronic Systems, 14(1), 2009.
17. W. Hwu, K. Keutzer, and T. Mattons. The Concurrency Challenge. IEEE *Design and Test of Computers*, July/Augustus 2008.
18. C. Isci, A. Buyuktosunoglu, C. Cher, P. Bose and M. Martonosi. An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget. *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 347-358, 2006.

19. K. Keutzer, A.R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli. System-Level Design: Orthogonalization of Concerns and Platform-Based Design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 19(12), pp. 1523-1543, December 2000.
20. S. Lauzac. Comparison of Global and Partitioning Schemes for Scheduling Rate Monotonic Tasks on a Multiprocessor. *Proceedings of EUROMICRO*, pp. 188-195, 1998.
21. J. Li and J.F. Martinez. Power-Performance Implications of Thread-Level Parallelism on Chip Multiprocessors. *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 124-134, March 2005.
22. J. Li and J.F. Martinez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. *Proceedings of the International Symposium on High-Performance Computer Architecture*, pp. 77-87, February 2006.
23. C. Liu and J. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM*, 20(1), pp. 46-61, January 1973.
24. J. Luo and N.K. Jha. Static and Dynamic Variable Voltage Scheduling Algorithms for Real-Time Heterogeneous Distributed Embedded Systems. *Proceedings of IEEE ASP-DAC*, pp. 719, 2002.
25. P. Marchal, M. Jayapala, S.D. Souza, P. Yang, Fr. Catthoor and G. Deconinck. Matador: an Exploration Environment for System Design. *J. Circ. Sys. Comput.,* pp. 503-535.
26. G. Mariani, G. Palermo, C. Silvano and V. Zaccaria. A Design Space Exploration Methodology Supporting Run-Time Resource Management for Multi-Processor Systems-on-Chip. *Proceedings of the IEEE Symposium on Application Specific Processors,* San Francisco, USA, July 2009.
27. N. Miniskar, E. Hammari, S. Munaga, S. Mamagkakis, P. Kjeldsberg and Fr. Catthoor. Scenario Based Mapping of Dynamic Applications on MPSoC : A 3D Graphics Case Study. *Proceedings of the SAMOS Workshop*, pp. 48-57, July 2009.
28. N. Miniskar, S. Munaga, R. Wuyts and Fr. Catthoor. Pareto Based Run-Time Manager for Overlapped Resource Sharing. *Proceedings of the ECES Workshop*, Edegem, Belgium, September 2009.
29. S. Munaga and Fr. Catthoor. Proactive Reliability-Aware Energy Management in Hard Real-Time Systems - A Motivational Case Study. *Proceedings of the Workshop on Design for Reliability*, Cyprus, January 2009.
30. V.K. Prasanna. Power-Aware Resource Allocation for Independent Tasks in heterogeneous Real-Time Systems. *Proceedings of IEEE ICPADS*, PP. 341, 2002.
31. K. Ramamritham, G. Fohler and J.M. Adan. Issues in the Static Allocation and Scheduling of Complex Periodic Tasks. *IEEE Real-Time Systems Newsletter*, 9, pp. 11-16, 1993.
32. D. Shin and J. Kim. Power-Aware Scheduling of Conditional Task Graphs in Real-Time Multiprocessor Systems. *Proceedings of the ACM International Symposium on Low Power Electronics and Design*, PP. 408-413, Augustus 2003.
33. M. Schmitz. Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems. *Proceedings of IEEE DATE*, 2002.
34. H. Shojaei, A. Ghamarian, T. Basten, M. Geilen, S. Stuijk and R. Hoes. A Parameterized Compositional Multi-dDmensional Multiple-Choice Knapsack Heuristic for CMP Run-Time Management. *Proceedings of IEEE DAC,* New York, USA, 2009.
35. A. Sinha and A. Chandrakasan. Jouletrack – a Web Based Tool for Software Energy Profiling. *Proceedings of IEEE DAC*, 2001.

36. P. Yang, P. Marchal, C. Wong, S. Himpe, F. Catthoor, P. David, J. Vounckx and R. Lauwereins. *Multiprocessor Systems-on-Chip: Cost-Efficient Mapping of Dynamic Concurrent Tasks in Embedded Real-Time Multimedia Systems*. Morgan-Kaufmann, Eds W. Wolfs and A. Jerraya, pp. 46-58, 2004.
37. P. Yang, C. Wong, P. Marchal, F. Catthoor, D. Desmet, D. Verkest and R. Lauwereins. *Energy-Aware Runtime Scheduling for Embedded Multiprocessor SOCs*. IEEE Design and Test of Computers, 18(5), pp. 46-58, September 2001.
38. Ch. Ykman-Couvreur, Fr. Catthoor, J. Vounckx, A. Folens, F. Louagie. Energy-Aware Task Scheduling Applied to a Real-Time Multimedia Application on an Xscale Board. *Journal of Low Power Electronics*, 1(3), pp. 226-237, December 2005.
39. Ch. Ykman-Couvreur, V. Nollet, Fr. Catthoor, and H. Corporaal. Fast Multi-Dimension Multi-Choice Knapsack Heuristic for MP-SoC Run-Time Management. *Proceedings of the International Symposium on System-on-Chip*, pp. 195-198, Tampere, Finland, November 2006.
40. Ch. Ykman-Couvreur, V. Nollet, Th. Marescaux, E. Brockmeyer, Fr. Catthoor, and H. Corporaal. Design-Time Application Mapping and Platform Exploration for MP-SoC Customized Run-Time Management. *IET Comput. Digit. Tech*., 1(2), pp. 120-128, 2007.
41. Ch. Ykman-Couvreur, R. Obermaisser, C. El Salloum, M. Goedecke, R. Zafalon and L. Benini. Resource Management for Embedded Multi-Core Platforms. *Proceedings of DATE Workshop on Designing for Embedded Parallel Computing Platforms*, France, April 2009.
42. Y. Zhang, X. Hu and D. Chen. Task Scheduling and Voltage Selection for Energy Minimization. *Proceedings of IEEE DAC*, pp. 183-188, 2002.