Chapter 5
Design Space Exploration Supporting Run-Time Resource Management

Prabhat Avasare, Chantal Ykman-Couvreur, Geert Vanmeerbeeck, Giovanni Mariani, Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria

Abstract Running multiple applications optimally in terms of Quality of Service (e.g. performance and power consumption) on embedded multi-core platforms is a huge challenge. Moreover, current applications exhibit unpredictable changes of the environment and workload conditions which makes the task of running them optimally even more difficult. This dynamic trend in application runs will grow even more in future applications.

This Chapter presents an automated tool flow which tackles this challenge by a two-step approach: first at design-time, a *Design Space Exploration* (DSE) tool is coupled with a platform simulator(s) to get optimum operating points for the set of target applications. Secondly, at run-time, a lightweight *Run-time Resource Manager* (RRM) leverages the design-time DSE results for deciding an operating configuration to be loaded at run-time for each application. This decision is performed dynamically, by taking into consideration available platform resources and the QoS requirements of the specific use-case. To keep RRM execution and resource overhead at minimum, a very fast optimisation heuristic is integrated.

Application of this tool-flow on a real-life multimedia use case (described in Chap. 9) will demonstrate a significant speedup in optimisation process while maintaining desired Quality of Service.

5.1 Introduction

The future of embedded computing is shifting to multi-core designs due to handling of simultaneous multiple applications and at the same time to boost performance given the unacceptable power consumption and operating temperature increase of fast single-core processors. This introduces new big challenges for application and platform designers. The first challenge is the support for a variety of applications:

P. Avasare (✉)
IMEC, Leuven, Belgium
e-mail: avasare@imec.be

C. Silvano (eds.), *Multi-objective Design Space Exploration of Multiprocessor SoC Architectures*,
DOI 10.1007/978-1-4419-8837-9_5,

mobile communications, networking, automotive and avionic applications, multimedia. These applications may run concurrently, start, and stop at any time. There is lot of dynamism in these applications which makes it challenging to run them optimally. Each application may have multiple configurations, with different constraints imposed by the external world or the user (deadlines and quality requirements, such as audio and video quality, output accuracy), different usages of various types of platform resources (processing elements, memories and communication), and different costs (performance, power consumption, bandwidth). 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 resource requirement for each application may vary over time (by the time applications are put in the market for sell) 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. The third challenge is time to market, which makes software development productivity of paramount importance. Designers are getting much less time to design their application and platforms whereas the complexity of the design and mapping is increasing. This can make it hard to perform exhaustive design space explorations. At the same time, due to the increasing complexity of the architecture, it is almost impossible for traditional design techniques to identify an optimal static configuration matching the available architecture resources with the dynamic requirements (i.e., performance and power consumption).

Given such tough challenges, one can design an embedded system by making conservative worst-case assumptions, but that will not lead to an efficient design. Especially in embedded system design, where there are very tight constraints (e.g. cost of a device), it is not acceptable to have a device with large inefficiency. To overcome these challenges, we have developed a methodology to enable efficient running of multiple applications on multi-core platforms. Our approach uses a run-time manager to control different possible application configurations [39]. Thus, concerning all the parameters that can be changed at run-time (e.g. the number of cores used to run an application and their operating frequencies), a run-time manager layer is built to achieve desirable *Quality of Service* (QoS) for the applications.

Our run-time manager layer consists of a Run-time Resource Manager (RRM) with following features:

- It supports a holistic view of the resources. This is needed for global resource allocation decisions, arbitrating between all applications, and minimizing the total costs.
- It transparently optimizes the resource usage and the application mapping on the platform. This is needed to facilitate the application development and mapping from diverse application domains.
- It dynamically adapts to changing environment. This is needed to enable the best usage of resources and to achieve a high efficiency under changing environment and requirements. To that end, dynamic resource allocation and dynamic reconfiguration of applications must be supported. Also, quality requirements

and resources must be adapted dynamically (e.g. by adjusting the processor clock frequency, or by switching off some processors) in order to control platform performance (i.e., the power consumption and the heat dissipation of the platform).
- Such a resource management problem is a Multi-Objective Optimization (MOO) problem (known as Multi-dimension Multiple-choice Knapsack Problem or MMKP) falling into a complexity category of NP-Hard problems [11]. Since our RRM is intended for embedded platforms, it is implemented as a lightweight heuristic implementation which consumes during its execution as little platform resources as possible and does not impact heavily execution time of the running application.

With respect to discussions in this Chapter, run-time manager consists of only RRM. Hence we have used term run-time manager and RRM interchangeably.

To alleviate run-time decision making (i.e., to reduce computational complexity at run-time) and to avoid conservative worst-case assumptions, our run-time management methodology consists of two phases:

- First, a design-time Design Space Exploration (DSE) per application derives a multi-dimensional Pareto set of optimal configurations on a given multi-core platform. During this phase, optimization and modeling techniques presented in Chaps. 3 and 4 are adopted.
- Second, a low-complexity Run-time Resource Manager (RRM) is incorporated on top of the basic services of the platform Operating System (OS) and is acting as an exception handler at run-time to optimize the usage of resources.

In the first phase, at design time, each configuration is characterized by a code version together with an optimal combination of constraints, used 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. To enable the design-time DSE phase, the methodology presented in this Chapter considers only applications within a set defined at design-time. System-wide approaches, for resource management at coarse grain, considering software applications for which the platform was not directly designed (as the approach proposed in [4]), are discussed in Chap. 6.

In the second phase of run-time management presented in this Chapter, whenever the environment is changing dynamically (e.g. when a new application or use case starts, or when the user requirements change), RRM reacts as follows:

- It selects a configuration from the Pareto set of each active application, according to the available resources, in order to minimize the costs, while satisfying the constraints.
- Second, it reconfigures and maps the application 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 executions according to the newly selected configurations.

The remainder of this Chapter is organized as follows. Section 5.2 overviews the state-of-the-art on RRM for embedded multi-core platforms and details RRM problem tackled in this Chapter. Section 5.3 formulates the RRM problem. Section 5.4 presents the exploration tool flow to solve this RRM problem. Conclusions are drawn in Sect. 5.5.

5.2 Run-Time Resource Management in Embedded Systems

This section introduces state-of-the-art techniques on RRM and places our methodology in comparison to previous literature.

In the context of Run-time Resource Management (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:

- First, some of them are applicable only for single-core platforms [32], or for homogeneous multi-core platforms [42], but not for heterogeneous multi-core platforms.
- Second, none of the existing approaches proposes a complete framework. They 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 [28]. Some others are based only on slowing or shutting down the platform resources [5] and on Dynamic Voltage and Frequency Scaling (DVFS) [9, 14, 19, 27].
- Third, the objective of the majority of these approaches is performance optimization [3, 6, 8, 15, 18], and not together with power consumption optimization.
- Finally, design-time approaches involve slow heuristics [27, 29, 30] using Integer Linear Programming (ILP) and cannot be used at run time. On the other hand, to reach a lightweight implementation, run-time approaches hide the specification of the internal application tasks, and they do not 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 [34–36], explores the energy-performance trade-offs 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 [20, 39].
- Scenario-based approaches [12, 23], which are based on the concept of application scenarios identified at design time, operate 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 at design time.

- The task scheduling techniques [34, 35] 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 in [31, 37] 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 [24], which allows overlapped sharing of the platform resources. The 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 [16, 17] also exploits the number of available cores, in addition to voltage and frequency. Different parallelized versions of a single application are used to trade-off the available platform resources with the performance and the power consumption.
- The addition of the parallelism to the set of 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 a Multi-dimension Multiple-choice Knapsack Problem (MMKP) and solving it through dedicated heuristics [31, 37]. Another approach [22] 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-45 nm 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 [26], 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 RRM framework has been developed in [40], with the most relevant generic services.

This Chapter describes a tool flow (shown in Fig. 5.1), combining a design-time exploration with a lightweight run-time manager for embedded multi-core platforms [21]. The run-time manager leverages a set of pre-determined run-time configurations (or *operating points*) identified at design-time (see Fig. 5.1) by analyzing and exploring the architecture run-time parameters impact on the QoS through an architecture simulator. The operating points consist of information/knowledge about parameters (e.g. the power consumption, the throughput) that designers wish to optimize and resource usage associated with each configuration of the run-time parameters of the hardware/software infrastructure. The overall goal of the run-time manager is to make a reasonable assignment of the run-time parameters to optimize

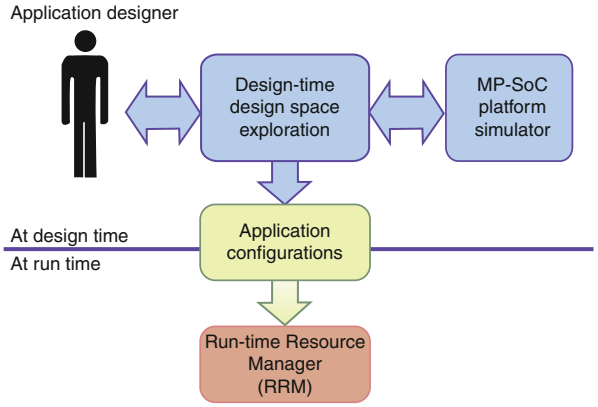

Fig. 5.1 Exploration tool-flow for Run-time Resource Management

desired output parameters (in our case execution time and power consumption) while meeting output constraints (in our case throughput QoS).

For each application, a multi-dimension Pareto set of optimal configurations is identified at design time, by analyzing and exploring several parallelizations and impacts on the constraints, the platform resource usage, and the costs. This is automated through a Design Space Exploration tool coupled with a platform simulator(s):

- The used design space explorers can be: either modeFRONTIER [**?**], commercially available, or Multicube Explorer [41] (being open-source, and being the target of this Chapter). Typically in design space exploration for multi-core systems, the number of possible application configurations can be huge. Hence these exploration tools with sophisticated analytical and modeling techniques are needed within the embedded system design cycle. And as previously mentioned, they also allow alleviating the run-time decision making.
- As pointed in [13], platform simulations can be done at many abstraction levels: e.g. functional-level simulation, timed simulation, cycle-accurate simulation. But the key problem 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. In this Chapter, used platform simulators are available at two abstraction levels [1]. In our approach, we combine two simulators during the DSE as follows: first, an extensive DSE is performed using the fast high-level timed and functional simulator HLSim [2] and derives a large set of optimal application configurations, including execution time and power consumption estimations reported by HLSim; then, only the interesting configurations are first verified and then explored further using the more accurate simulator. TLMsim is a SystemC-based cycle-accurate Transaction-Level Model (TLM) [7], built using the Synopsys virtual platform prototyping tools [33]. This simulator consumes more time, but it reports more accurate estimations. Recently, simulations based on a cycle-accurate TLM have gained importance due to standardization efforts.

In this Chapter we target both the number of cores and their frequency as a run-time configurable parameter of an application. We thus assume that:

- The task-level parallelism of each application running on the system can be changed through code *versioning*.
- The frequency associated with each core can be changed (or *scaled*) dynamically.

The overall goal of RRM is to make a reasonable assignment of the run-time parameters to reach pre-determined operating goals. Operating goals can be just one (e.g. execution time) or multiple (e.g. power consumption, throughput QoS), as in our case. In our tool flow, the RRM is incorporated on top of the basic services of the platform OS and acts as an exception handler. Its implementation conforms to the framework proposed in [40]. Its optimization strategy to globally select optimal configurations for the active applications extends the fast MMKP heuristic for multi-core run-time management [37]. In case of soft real-time applications, and when the optimization strategy can find no solution, RRM also takes the application priorities into account to relax the constraints and to reach a solution after all [21]. Also to reduce execution overhead of the run-time selection heuristic, our methodology performs initial filtering of optimal run-time parameter configurations. This allows further alleviating the run-time selection heuristic.

5.3 Run-time Resource Management Problem Definition

As described above, the goal of the proposed methodology (Fig. 5.1) for our case is to achieve, at deployment time, a desired QoS while minimizing power consumption and maximizing the usage of multiple cores. Before describing our methodology (Sect. 5.4), we formally define RRM problem. This section first describes terminologies used in this Chapter and then formally describes the RRM problem.

5.3.1 Terminologies

In this Chapter we will assume that there are p active applications; we identify each application with an identifier $\alpha \in A = \{\alpha_1 \ldots \alpha_p\}$. During its lifetime, each application can be described as having a specific **operating point** which consists of its actual **cost**, used **resources** and achieved **QoS**. A list of suitable operating points is essential for the correct behavior of the run-time manager since it represents both the *goals* to be optimized (cost, QoS and resources) as well as the independent *control parameters* (the resources and their associated properties) with which the optimization goal can be achieved.

More formally, for our optimisation problem, we can define the operating point of application α with the following tuple:

$$\boldsymbol{c}_\alpha = \langle \rho, \boldsymbol{\phi}, \pi, \tau \rangle \tag{5.1}$$

where:

- ρ is a scalar representing the **resources** associated with the operating point $\boldsymbol{c}_\alpha$. We assume that there exists an application binary version which has been parallelized over ρ cores. In our case, $\rho \in R = \{1 \ldots \rho_{max}\}$, where ρ_{max} = maximum number of available cores. Given the features of the interconnection bus, we assume homogeneity among the cores and a fixed mapping for each of the threads.
- $\boldsymbol{\phi}$ corresponds to the frequency configuration for each of the ρ cores: $\langle \phi_1 \ldots \phi_\rho \rangle,\ \phi_i \in \Phi \wedge 1 \leq i \leq \rho$.
- π is the actual **cost** associated with the current operating point $\boldsymbol{c}_\alpha$. Here, we consider the cost as the average power consumption of application α.
- τ is the average execution time needed by α for a single period[1] of an application (e.g. encoding a single frame in a multi-media application) when on the current operating point $\boldsymbol{c}_\alpha$.

In the following sections, we will use the notation $\boldsymbol{c}_\alpha[X]$ to access the element X of the tuple defined in Eq. 5.1.

5.3.2 RRM Problem

We assume that for each application α, there is an available set of operating points $\boldsymbol{C}_\alpha$ whose size is N_α. The run-time manager has to select exactly one point from each active set $\boldsymbol{C}_\alpha$, according to the available platform resources, in order to minimize the total power consumption of the platform, while respecting all application deadlines. Given a set of required application deadlines τ^α_{max}, our problem definition is to identify, at run-time, a comprehensive set of operating points:

$$\boldsymbol{\gamma} = \langle \boldsymbol{c}_{\alpha_1} \ldots \boldsymbol{c}_{\alpha_p} \rangle,\ \ \boldsymbol{c}_{\alpha_j} \in \boldsymbol{C}_{\alpha_j} \wedge 1 \leq j \leq p \tag{5.2}$$

such that the following measure of power consumption is minimized:

$$\sum_{\alpha \in A} \boldsymbol{c}_\alpha[\pi] \tag{5.3}$$

[1] Note that in domains of multi-media and wireless applications, usually the applications are periodic i.e., they do same task again and again but on different input data e.g. processing video frames or wireless packets

subject to the following QoS and resource constraints:

$$\boldsymbol{c}_\alpha[\tau] \leq \tau^\alpha_{max},\ \forall \alpha \in A \tag{5.4}$$

$$\sum_{\alpha \in A} \boldsymbol{c}_\alpha[\rho] \leq \rho_{max} \tag{5.5}$$

where ρ_{max} is the maximum number of resources (or cores) in the system, p is number of active applications and A is a set of all active applications.

According to [38], the previous problem is a Multi-dimension Multiple-choice Knapsack Problem (MMKP) whose complexity resides in the NP-hard space with respect to p, N_α and ρ_{max}. Moreover, depending on how tight τ^α_{max} constraints have been set, there may not be *feasible* solutions $\boldsymbol{\gamma}$. However, in our case of embedded systems for multi-media domain, usually applications do not have *hard* real-time constraints. Instead, we address the design of a **soft real-time system** in which deadlines can be missed with the lowest penalty possible and/or the lowest probability. We manage this possibility by introducing a priority $\omega(\alpha)$ measure to be used by the run-time manager to relax some τ^α_{max} and reach a feasible solution.

In the following section we describe our proposed toolflow which consists of two parts: one design-time analysis and other run-time heuristic.

5.4 Proposed Tool-Flow for RRM

This section describes our proposed tool-flow to solve Run-time Resource Management (RRM) problem described above. Application of this tool-flow on a real-life multimedia use case is described in Chap. 9. Our tool-flow solves RRM problem in two steps:

1. A **design-time heuristic methodology** for reducing the average size N_α for each α.
2. A **run-time management layer** consisting of a filtering algorithm for each $\boldsymbol{C}_\alpha$ and a greedy, *prioritized* heuristic for solving the MMKP.

5.4.1 Design-Time Heuristic Methodology

Our design-time methodology is shown in Fig. 5.1. At design time, we identify an ordered list $\boldsymbol{C}_\alpha$ of operating points:

$$\boldsymbol{C}_\alpha = \langle \boldsymbol{c}^1_\alpha \ldots \boldsymbol{c}^{N_\alpha}_\alpha \rangle \tag{5.6}$$

$\boldsymbol{C}_\alpha$ is generated at design-time by analyzing and exploring the impact of the architecture run-time parameters on the QoS through an architecture simulator(s). Optimal $\boldsymbol{C}_\alpha$ is derived from these design-time analysis by help of sophisticated optimization

1: **for** each $\alpha \in A$ **do**
2: **for** each $\rho \in R$ **do**
3: Φ_p = arg min $\langle P_{\alpha,\rho}(\phi), T_{\alpha,\rho}(\phi)\rangle$ (ϕ is the vector of frequencies associated with the current ρ).
4: **for** each $\phi \in \Phi_p$ **do**
5: $c_\alpha = \langle \rho, \phi, P_{\alpha,\rho}(\phi), T_{\alpha,\rho}(\phi)\rangle$
6: $C_\alpha = C_\alpha \cup \{c_\alpha\}$.
7: **end for**
8: **end for**
9: **end for**
10: Pareto filter C_α on the objectives ρ, π and τ
11: Sort C_α configurations by ascending $c_\alpha[\rho]$ and (second) $c_\alpha[\pi]$.

Fig. 5.2 Design-time methodology for identifying the initial $\boldsymbol{C}_\alpha$, for each application α

and modeling techniques (see Chaps. 3 and 4) supported by DSE tool. The following algorithm describes sequence of steps taken during our design-time methodology.

Algorithm in Fig. 5.2 shows the structure of adopted design-time methodology. The algorithm mainly performs the minimization of the execution time and power consumption for each application version (i.e., for each application α and each available version of α which is parallelized over ρ resources).

Once all results are collected, these are sorted for obtaining a further speedup at run-time. A more detailed step-by-step description follows:

- *Steps 1, 2 and 3*: The algorithm loops over the set of available applications A (Step 1) and the set of resources R (Step 2) by solving a multi-objective minimization problem with respect to both $P_{\alpha,\rho}(\boldsymbol{\phi})$ and $T_{\alpha,\rho}(\boldsymbol{\phi})$ (Step 3). In our case, $P_{\alpha,\rho}(\boldsymbol{\phi})$ and $T_{\alpha,\rho}(\boldsymbol{\phi})$ are, respectively, the average power consumption and the execution time delay returned by the architectural simulator for a frequency configuration vector $\boldsymbol{\phi}$. Note that the independent variable to be optimized is $\boldsymbol{\phi}$ since ρ is set as a constraint in the loop (Step 2).
  Since the size of the design space increases rapidly with ρ, while for $\rho \leq 3$ a full search exploration is performed, for $\rho \geq 4$ the problem is solved by using the NSGA-II multi-objective genetic algorithm [10]. The genetic algorithm is run with a population size $|\Phi| \times \rho$ for 50 generations. The set Φ_p contains all the Pareto configurations associated with the solution of the problem.
- *Steps 5 and 6*: The set Φ_p is used to construct the operating points and insert them into the associated $\boldsymbol{C}_\alpha$.
- *Step 10*: It prunes $\boldsymbol{C}_\alpha$ from all the configurations that do not represent an efficient trade-off in terms of resources, energy and execution time.
- *Step 11*: It sorts the resulting $\boldsymbol{C}_\alpha$ to improve the performance of the run-time algorithm while finding an operating point with a lower resource usage while achieving τ^{α}_{max}.

The proposed heuristic methodology is able to save a considerable number of simulations. The exact amount of simulation time saving depends on the specific application scenario as well as from the selected optimization algorithm used for solving the

MOO problem in the Step 3. In particular, for the multimedia scenario proposed in Chap. 9 the design-time DSE flow is able to solve the optimization problem simulating less than the 1% of the overall run-time candidate configurations.

5.4.2 The Run-Time Management Methodology

We assume that a series of events change the application deadlines τ^{α}_{max} and trigger the RRM to identify a new $\boldsymbol{c}_{\alpha}$ for every active application α. The RRM layer is invoked as an exception handler within the OS layer and elaborates the following input information:

- Optimal application configurations $\boldsymbol{C}_{\alpha}$, for each α as determined by the design-time methodology. This information is stored within the OS layer once at the startup so there is no execution time overhead.
- QoS requirements τ^{α}_{max} for each α. In our case related to multi-media applications, this is set to average *time-per-frame*.
- A priority $\omega(\alpha)$ function which ranks each α (low $\omega(\alpha)$ corresponds to low priority).

We define, for each application α, the following input tuple:

$$\boldsymbol{\xi}_{\alpha} = \langle \boldsymbol{C}_{\alpha}, \tau^{\alpha}_{max}, \omega(\alpha) \rangle \tag{5.7}$$

The output of the run-time algorithm is a set of working operating points ($\boldsymbol{\gamma}$) achieving QoS ($\boldsymbol{c}_{\alpha}[\tau] \leq \tau^{\alpha}_{max}$) while meeting overall constraints: in our case resource ($\sum_{\alpha} \boldsymbol{c}_{\alpha}[\rho] \leq \rho_{max}$) and power consumption ($\sum_{\alpha} \boldsymbol{c}_{\alpha}[\pi]$). The operating points are then set by loading an appropriate application binary version (if the number of resources was varied) or modifying the frequency of each core as dictated by the frequency vector $\boldsymbol{c}_{\alpha}[\boldsymbol{\phi}]$.

The overall run-time management algorithm is shown in Fig. 5.3. Mainly the algorithm starts trying to find a solution which satisfies all run-time QoS constraints and then, if no solutions have been found, it iteratively relax the QoS constraint of the lowest priority application until a feasible solution is identified. In particular the algorithm consists of the following steps:

Require: run-time-manager($\xi_{\alpha_1} \ldots \xi_{\alpha_p}$)
1: γ = allocate($\xi_{\alpha_1} \ldots \xi_{\alpha_p}$)
2: $\Omega = \{\}$ /* relaxed application set */
3: **while** not feasible(γ) $\wedge$ ($A - \Omega \neq \emptyset$) **do**
4: $\overline{\alpha}$ = arg min $\omega(\alpha)$, $\alpha \in (A - \Omega)$
5: $\rho_{\overline{\alpha}}$ = min $c[\rho]$, $c \in C_{\overline{\alpha}}$
6: $\tau^{\overline{\alpha}}_{max}$ = min $c[\tau]$, $c \in C_{\overline{\alpha}} \wedge c[\rho] = \rho_{\overline{\alpha}}$
7: γ = allocate($\xi_{\alpha_1} \ldots \xi_{\alpha_p}$)
8: $\Omega = \Omega + \{\overline{\alpha}\}$
9: **end while**
10: **return** γ

Fig. 5.3 Priority based QoS-aware run-time management

- *Step 1*: The algorithm invokes an *allocate* function which solves the actual MMKP problem with the given constraints to produce a solution $\boldsymbol{\gamma}$.
- *Step 2 and 3*: The invocation is performed iteratively until $\boldsymbol{\gamma}$ satisfies the given constraints on the resources or all the application constraints have been relaxed ($\Omega = A$).
- *Steps 4, 5 and 6*: Whenever the solution is unfeasible in terms of overall resources, the deadline τ^{α}_{max} of the lowest priority application $\overline{\alpha}$ is relaxed (Step 6) reducing it to the minimum possible among a reduced set of resources $\rho_{\overline{\alpha}}$.
- *Step 7*: *allocate()* algorithm is invoked.
- *Step 8*: The selected $\overline{\alpha}$ is then put into a *relaxed application set* Ω.

The *allocate* function (see Fig. 5.4) actually solves the MMKP problem with a light-weight algorithm as presented in [38].

- *Step 2*: $\boldsymbol{C}_{\alpha}$ is pruned in order to have a single $\boldsymbol{\phi}$ assignment for each ρ. This is done by selecting only those configurations which meet τ_{max} but have the lowest power consumption for a unique ρ. The resulting set of operating points is put in $\boldsymbol{\chi}_{\alpha}$ and actually reduces the amount of data to be elaborated by the MMKP solver considerably. Note that the pre-filtering Step 2 has a linear complexity with respect to the cardinality of $\boldsymbol{C}_{\alpha}$ due to the sorting performed in Fig. 5.2, Step 11.
- *Step 4*: A *unified knapsack* vector $\boldsymbol{\Psi}$ is created; this is a vector of $\boldsymbol{c}$ sorted by prioritizing the *improvement* per single resource (*normalized user value*) $u(\boldsymbol{c}_{\alpha})$:

$$u(\boldsymbol{c}_{\alpha}) = \frac{v(\boldsymbol{c}_{\alpha})}{\boldsymbol{c}_{\alpha}[\rho]},\ v(\boldsymbol{c}_{\alpha}) = \max_{\forall \boldsymbol{c} \in \boldsymbol{\chi}_{\alpha}} \boldsymbol{c}[\pi] - \boldsymbol{c}_{\alpha}[\pi] \tag{5.8}$$

  In other words, $v(\boldsymbol{c}_{\alpha})$ is the improvement in terms of power consumption with respect to the maximum power consuming operating point in $\boldsymbol{\chi}_{\alpha}$. $u(\boldsymbol{c}_{\alpha})$ is $v(\boldsymbol{c}_{\alpha})$ per unitary resource.
- *Step 5*: A greedy linear-scan algorithm [38] is invoked on $\boldsymbol{\Psi}$ for identifying the final set of operating points $\boldsymbol{\gamma}$. Given the sorting performed previously, the algorithm has a worst case complexity of $O(pN\log(pN))$, where N is the average size of the operating point set per application.

It can be shown that the total worst-case complexity of our RRM scheme is $O(pN_{max} + pN\log(pN))$, where p is number of active applications, N is a set of all active applications and N_{max} is maximum number of configurations. This reduction in complexity is achieved due to the adequate filtering and sorting done

Require: allocate($\xi_{\alpha_1} \ldots \xi_{\alpha_p}$)
1: **for** each α **do**
2: χ_{α} = pre-filter($C_{\alpha}, \tau^{\alpha}_{max}$)
3: **end for**
4: Ψ = merge-and-sort ($\chi_{\alpha_1} \ldots \chi_{\alpha_p}$)
5: γ = solve-knapsack(Ψ)
6: **return** γ

Fig. 5.4 Allocate function

before the greedy algorithm solving the knapsack problem. Running such an algorithm on a host processor will not cause too much overhead in terms of resources and execution time (Chap. 9).

5.5 Conclusions

In this Chapter, we presented an automated tool flow which tackles Run-time Resource Management challenge for multi-core embedded systems by efficiently combining a design space exploration tool coupled with platform simulator(s). This tool flow consisted of two steps: first a design-time heuristic methodology was described which reduces number of configurations that an application can be run optimally. This first design-time phase leverages optimization algorithms described in Chap. 3 for minimizing the amount of simulations to be performed. Second step is a light-weight run-time manager which selects an optimal configuration for each running application depending on demanded QoS requirement. Hence, at run-time, run-time manager leverages the design-time DSE results for deciding an operating configuration to be loaded for each application. This operation is performed dynamically, by following the QoS requirements of the specific use-case. Application of this tool-flow on a real-life multimedia use case is described in Chap. 9 while other frameworks operating at the Operating System (OS) level are presented in Chap. 6.

References

1. Avasare, P., Vanmeerbeeck, G., Kavka, C., Mariani, G.: Practical approach for design space explorations using simulators at multiple abstraction levels. In: Design Automation Conference (DAC) Users' Track (2010)
2. Baert, R., Brockmeyer, E., Wuytack, S., Ashby, T.: Exploring parallelizations of application for mpsoc platforms using mpa. In: Proceedings of IEEE Design, Automation Test in Europe Conference Exhibition (DATE), pp. 1148–1153. France (2009)
3. Baker, T.P.: An analysis of edf schedulability on a multiprocessor. IEEE Transactions on Parallel and Distributed Systems **16**, 760–768 (2005)
4. Ballesi, P., Fornaciari, W., Siorpaes, D.: A hierarchical distributed control for power and performances optimization of embedded systems. In: Conference on Architecture of Computing Systems (ARCS), pp. 37–48. Hannover (2010)
5. Benini, L., Bogliolo, R., De Micheli, G.: A survey of design techniques for system-level dynamic power management. IEEE Transactions on VLSI Systems **8**, 299–316 (2000)
6. Buchard, A.: Assigning real-time tasks to homogeneous multiprocessor systems. Technical Report, University of Virginia (1994)
7. Cai, L., Gajski, D.: Transaction level modeling: an overview. In: Proceedings of CODES+ISSS (2003)
8. Chan, H.L.: Non-migratory online deadline scheduling on multiprocessors. In: Proceedings of SODA, pp. 970–979 (2004)
9. Chen, J.J., Yang, C.Y., Kuo, T.W., Shih, C.: Energy-efficient real-time task scheduling in multiprocessor dvs systems. In: Design Automation Conference, 2007. ASP-DAC '07. Asia and South Pacific, pp. 342–349 (2007)

10. Deb, K., Agrawal, S., Pratab, A., Meyarivan, T.: A Fast and Elitist Multi-Objective Genetic Algorithm: NSGA-II. Proceedings of the Parallel Problem Solving from Nature VI Conference pp. 849–858 (2000). URL citeseer.ist.psu.edu/article/deb00fast.html
11. Garey, M.R., Johnson, D.S.: Computers and Intractability : A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences). W. H. Freeman (1979)
12. Gheorghita, S., Palkovic, M., Hamers, J., Vandecappelle, A., Mamagkakis, S., Basten, T., Eeckhout, L.: System-scenario-based design of dynamic embedded systems. ACM Trans. on Design Automation of Electronic Systems **14**, 1–45 (2009)
13. Gries, M.: Methods of evaluating and covering the design space during early design development. Integration, the VLSI journal **38**(2), 131–183 (2004)
14. Isci, C., Buyuktosunoglu, A., Cher, C., Bose, P., Martonosi, M.: An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 347–358 (2006)
15. Lauzac, s.: Comparison of global and partitioning schemes for scheduling rate monotonic tasks on a multiprocessor. In: Proceedings of EUROMICRO, pp. 188–195 (1998)
16. Li, J., Marinez, J.: Power-performance implications of thread-level parallelism on chip multiprocessors. In: Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software, pp. 124–134 (2005)
17. Li, J., Marinez, J.: Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In: Proceedings of the International Symposium on High-Performance Computer Architecture, pp. 77–87 (2006)
18. Liu, C., Layland, J.: Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of the ACM **20**(1), 46–61 (1973)
19. Luo, J., Jha, N.K.: Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In: Proceedings of IEEE ASP DAC, pp. 719–726 (2002)
20. Marchal, P., Jayapala, M., Souza, S., Yang, P., Catthoor, F., Deconinck, G.: Matador: an exploration environment for system design. Journal of Circuits, Systems, and Computers **11**(5), 503–535 (2002)
21. Mariani, G., Avasare, P., Vanmeerbeeck, G., Ykman-Couvreur, C., Palermo, G., Silvano, C., Zaccaria, V.: An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In: DATE 2010 - International Conference on Design, Automation and Test in Europe., pp. 196 –201. Dresden, Germany (2010)
22. Mariani, G., Palermo, G., Silvano, C., Zaccaria, V.: A design space exploration methodology supporting run-time resource management for multi-processor systems-on-chip. In: Proceedings of the IEEE Symposium on Application Specific Processors, pp. 21–28. San Fransisco, USA (2009)
23. Miniskar, N., Hammari, E., Munaga, S., Mamagkakis, S., Kjeldsberg, P., Catthoor, F.: Scenario based mapping of dynamic applications on mpsoc : A 3d graphics case study. In: Proceedings of SAMOS Workshop, pp. 48–57 (2009)
24. Miniskar, N., Munaga, S., Wuyts, R., Catthoor, F.: Pareto based run-time manager for overlapped resource sharing. In: Proceedings of the ECES Workshop. Belgium (2009)
25. modeFRONTIER DSE tool, http://www.esteco.com
26. Munaga, S., Catthoor, F.: Proactive reliability-aware energy management in hard real-time systems - a motivational case study. In: Proceedings of the Workshop on Design for Reliability, pp. 195–200. Cyprus (2009)
27. Prasanna, V.K., Yu, Y.: Power-aware resource allocation for independent tasks in heterogeneous real-time systems. In: Parallel and Distributed Systems, 2002. Proceedings. Ninth International Conference on, pp. 341–348 (2002)
28. Ramamritham, K., Fohler, G., Adan, J.M.: Issues in the static allocation and scheduling of complex periodic tasks. IEEE Real-Time Systems Newsletter **9**, 11–16 (1993)
29. Schmitz, M., Al Hasimi, B., Eles, P.: Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems. In: Proceedings of IEEE Design, Automation Test in Europe Conference Exhibition (DATE), pp. 514–521 (2002)

30. Shin, D., Kim, J.: Power-aware scheduling of conditional task graphs in real-time multiprocessor systems. In: Proceedings of ACM International Symposium on Low Power Electronics and Design, pp. 408–413 (2003)
31. Shojaei, H., Ghamarian, A., Basten, T., Geilen, M., Stuijk, S., Hoes, R.: A parameterized compositional multi-dimensional multiple-choice knapsack heuristic for cmp run-time management. In: Design Automation Conference (DAC), pp. 917–922 (2009)
32. Sinha, A., Chandrakasan, A.: Jouletrack - a web based tool for software energy profiling. In: Design Automation Conference (DAC), pp. 220–225 (2001)
33. Synopsys virtual platform prototyping tools. http://www.synopsys.com
34. Yang, P., Marchal, P., Wong, C., Himpe, S., Catthoor, F., David, P., Vounckx, J., Lauwereins, R.: Multiprocessor Systems-on-Chip: Cost-Efficient Mapping of Dynamic Concurrent Tasks in Embedded Real-Time Multimedia Systems, Eds W. Wolf and A. Jerraya. Morgan-Kaufman (2004)
35. Yang, P., Wong, C., Marchal, P., Catthoor, F., Desmet, D., Verkest, D., Lauwereins, R.: Energy-aware runtime scheduling for embedded multiprocessor socs. IEEE Design and Test of Computers **18**(5), 46–58 (2001)
36. Ykman-Couvreur, C., Catthoor, F., Vounckx, J., Folens, A., Louagie, F.: Energy-aware task scheduling applied to a real-time multimedia application on an xscale board. Journal of Low Power Electronics **1**(3), 226–237 (2005)
37. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast multi-dimension multi-choice knapsack heuristic for mp-soc run-time management. In: Proceedings of the International Symposium on System-on-Chip, pp. 195–198. Tampere, Finland (2006)
38. Ykman-Couvreur, C., Nollet, V., Catthoor, F., Corporaal, H.: Fast multi-dimension multi-choice knapsack heuristic for MP-SoC run-time management. In: Proceedings of International Symposium on System-on-Chip, pp. 1–4 (2006).
39. Ykman-Couvreur, C., Nollet, V., Marescaux, T., Brockmeyer, E., Catthoor, F., Corporaal, H.: Design-time application mapping and platform exploration for mp-soc customized run-time management. IET Computers and Digital Techniques **1**(2), 120–128 (2007)
40. Ykman-Couvreur, C., Obermaisser, R., El Salloum, C., Goedecke, M., Zafalon, R., Benini, L.: Resource management for embedded multi-core platforms. In: Proceedings of DATE Workshop on Designing for Embedded Parallel Computing Platforms. France (2009)
41. Zaccaria, V., Palermo, G., Mariani, G.: Multicube explorer (2008). http://www.multicube.eu
42. Zhang, Y., Hu, X., Chen, D.: Task scheduling and voltage selection for energy minimization. In: Design Automation Conference (DAC), pp. 183–188 (2002)