Dynamic and Adaptive Allocation of Applications on MPSoC Platforms *

Andreas Schranzhofer* Jian-Jia Chen* Luca Santinelli† Lothar Thiele*
*Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
† Scuola Superiore Sant'Anna (SSSUP), Pisa, Italy
*{schranzhofer,jchen,thiele}@tik.ee.ethz.ch, †santinelli@sssup.it

ABSTRACT

— Multi-Processor Systems-on-Chip (MPSoC) are an increasingly important design paradigm not only for mobile embedded systems but also for industrial applications such as automotive and avionic systems. Such systems typically execute multiple concurrent applications, with different execution modes. Modes define differences in functionality and computational resource demands and are assigned with an execution probability. We propose a dynamic mapping approach to maintain low power consumption over the system lifetime. Mapping *templates* for different application modes and execution probabilities are computed offline and stored on the system. At runtime a manager monitors the system and chooses an appropriate pre-computed template. Experiments show that our approach outperforms global static mapping approaches up to 45%.

Keywords: Probabilistic Applications, MPSoC, Dynamic Power-Aware Allocation, Runtime Adaptive Allocation.

I. INTRODUCTION

Multi-processor systems are becoming increasingly important in consumer electronics as well as in industrial applications, such as automotive software. These systems, known as Multi-Processor Systems-on-Chip (MPSoC), typically contain multiple heterogeneous processing units. In multimedia applications, such as mobile phones and Software-Defined-Radio (SDR) systems, it is uncommon to assume a fixed task set, since these applications (a) are usually composed of multiple execution modes and (b) have heavily varying execution patterns over their lifespan.

Power management and energy awareness are important design issues for embedded systems as well as for server system. Power consumption not only influences the battery lifetime of mobile devices or the cost of operating a server farm, but also influences the lifespan of systems, due to increased heat build-up. Power consumption is caused by a dynamic and a static part [5; 2]. In nano-meter manufacturing, leakage current contributes to the static power consumption significantly and cannot be neglected. Dynamic power consumption is related to a processing units' utilization.

*The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement no. 216008 project Predator and the European Network of Excellence on Embedded System Design ARTISTDesign.

In this paper we study how to map applications to processing units. Applications are defined as sets of modes and a mode is assigned with an execution probability, instead of concrete activation and active times. *Scenarios* define the possible combinations of modes that execute concurrently. Consequently, a mode change of an application results in a transition to another scenario. As a result, sequences of scenario transitions can be identified and a particular scenario can be attained by a sequence of mode changes. The underlying hardware platform is given as a library of available processing unit types. Deriving the actual hardware platform by instantiating processing units is part of the mapping problem.

The major contributions of this paper are:

- We propose a dynamic mapping process that is described by an offline and an online part. The offline part computes a mapping for each feasible sequence and stores them on the system as *templates*. In the online part, a manager observes mode changes and chooses an appropriate pre-computed template.
- Adaptivity to altering execution probabilities is introduced. Templates for different execution probabilities are computed and stored on the system.
- Experimental results show that the dynamic mapping process outperforms existing global static mapping approaches in [11] by up to 45%.

Related Work: Recent studies pick up similar application structures and tackle the problem by computing static global task allocations, e.g., [6; 11; 10]. In [11] application groups are constituted by applications that can only execute mutually exclusively. Application groups and applications therein are assigned with probabilities of execution. A static mapping heuristic is proposed, that considers static and dynamic power consumption. The effectiveness of the resulting system is limited by two factors. First, the structure of the applications, i.e., their modes and the resulting scenarios, are not exploited. A global static mapping can only define a single task allocation and therefore needs to consider the worst-case scenario. Second, the execution probabilities of applications need to be both, known a priori and remain constant. The system cannot adopt to alternating execution probabilities. Kim et al. propose a task allocation process in [6] that considers only dynamic power consumption. They propose to increase

the number of processing units up to an area constraint, which results in a reduced average utilization. In [10] Schmitz et al. propose a genetic algorithm to compute a task allocation for multi-mode applications. They focus on developing different mutation strategies.

Xu et al. [12] and Chen et al. [2] explore how to execute tasks and turn off processors in homogeneous multi-processor systems. These works focus on developing schedules for a fixed set of tasks, e.g., a single application with known activation and execution time or a set of periodically executing tasks, e.g. [3]. For power management in a server farm, [4] et al. explore how to integrate different power management policies.

Dynamic mapping methodologies are studied more recently. These studies basically split in two directions. Some tackle the problem by defining efficient heuristics to assign new arriving tasks onto processing units at runtime, e.g., [8; 7]. Online heuristics cannot guarantee schedulablity, e.g., Moreira et al. evaluate their approaches by computing the mapping success rate in [8] and [7]. Others analyze applications offline and compute schedules and allocations that are then stored on the system, e.g., [1; 9; 13]. In [1] Benini et al. propose to compute system configurations and derive task allocations and schedules for each of them. At run-time, transitions between allocations are assigned a migration cost. This work assumes that tasks can be migrated from one processing unit to another, once the system configuration changes. The decision whether tasks are migrated or not depends on pre-computed migration costs. We assume tasks to be resident, i.e. task migration is prohibited. This increases the complexity of the problem, since we have to consider possible future scenario transitions when we assign a task to a processing unit. Execution probabilities are neglected in [1], which might lead to adverse system configurations. Migration costs might be low compared to the increased dynamic power dissipation that results from not reallocating tasks that execute very frequently.

The rest of the paper is organized as follows: Section II introduces the application and hardware model and shows how scenarios are derived. We give a detailed problem description in Section III and continue with the proposed approach in Section IV. We show the effectiveness of our approach in Section V and finally conclude in Section VI.

II. System Model

This section introduces the hardware and application model, the terminology, and the assumptions that are used throughout the paper.

A. Hardware Model and Task Model

We define a heterogeneous multi-processor hardware platform by specifying a library $\mathcal{P}$ of available processing unit (PU) types. Each PU type $p_j \in \mathcal{P}$ is characterized by its computational resources per time unit (e.g., operations per time unit), denoted λ_j, and its power dissipation. The power dissipation of a PU type p_j is divided into a static and a dynamic part, σ_j and δ_j respectively. Static (or leakage) power is consumed once a PU is switched on. Dynamic power consumption depends on a processing unit's utilization and is consumed in addition to its static power dissipation.

A task t_i in a given task set $\mathcal{T}$ is defined by its computation demand (e.g., operations per time unit) $\gamma_{i,j}$ on a processing unit type $p_j \in \mathcal{P}$, where the corresponding utilization $u_{i,j}$ is $\frac{\gamma_{i,j}}{\lambda_j}$. Therefore, for a given task set $\hat{\mathcal{T}}_j$ assigned on a processing unit of type p_j, the utilization of the processing unit is then defined as $\sum_{t_i \in \hat{\mathcal{T}}_j} u_{i,j}$, and the power dissipation is defined as the sum of static and dynamic power dissipation, which is $\sigma_j + \delta_j \sum_{t_i \in \hat{\mathcal{T}}_j} u_{i,j}$. By definition, a processing unit cannot be overloaded, i.e., the utilization of tasks assigned on it must be less than 100%. We consider the library of available PU types given and furthermore assume that any number $k \in \mathbb{N}$ of instances is allowed for each type. Deriving a concrete hardware platform from the library of processing units is also considered in this paper.

B. Applications and Scenarios

$\mathcal{A}$ represents the set of concurrent applications executing on the hardware platform. An application $A_\ell \in \mathcal{A}$ is described by the given set of tasks $\mathcal{T}$ representing the applications functional and computational properties.

Additionally, an application $\mathcal{A}_\ell$ is constituted by a set $\mathcal{M}_\ell$ of modes. Each mode defines a use-case of an application, and, hence, has its own functionality. In other words, for an application $A_\ell \in \mathcal{A}$, a mode $\mu_k \in \mathcal{M}_\ell$ defines the active tasks, see Fig. 1 for an example. Applications have sets of initial and final modes. Initial modes define the start of an application. Once a mode change happens, final modes transition to initial modes, i.e., the application is restarted. In Fig. 2a, applications are composed of 3 modes. Modes `I1` and `I2` are the initial and modes `R1` and `R2` are the final modes of applications A_1 and A_2 respectively.

Tasks can be active in multiple modes of an application. Conclusively only one mode of an application can be active at a time. As an example consider an application implementing a radio standard, such as Wireless Local Area Network (WLAN). The application might be described by modes such as *synchronize* or *receive* and there might be a functional dependency, namely a task performing Fast-Fourier-Transform (FFT) which is active in both modes.

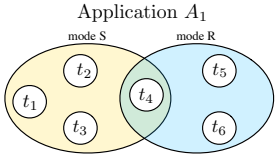


Fig. 1. Application with 6 tasks and 2 modes. t_4 is active in modes S and R.

A directed graph $G_\ell = (E_\ell, V_\ell)$ describes the modes of an application A_ℓ, where each node $v_i \in V_\ell$ of the graph describes a mode and each edge $e_i \in E_\ell$ describes a valid mode change, see Fig. 2a for example.

Applications can change their modes anytime without affecting the other applications. Thus, there are various combinations of active modes, as each mode of an application can execute concurrently with any other mode of other applications. A combination of the active modes for applications in $\mathcal{A}$ is defined as a *scenario* in this paper. We define the cross product of the applications' representative graphs as $G_\pi = (E_\pi, \mathcal{S})$. Therefore, $\mathcal{S}$ is the set of scenarios and a node S_m in $\mathcal{S}$ is a scenario. A directed edge in graph G_π represents a possible transition from one scenario to another.

As an example, suppose that $\mathcal{A}$ consists of two applications A_1 and A_2 represented by graphs $G_{A_1} = (E_{A_1}, V_{A_1})$ and $G_{A_2} = (E_{A_2}, V_{A_2})$ respectively. The resulting cross product $G_\pi = (E_\pi, \mathcal{S})$ is $G_{A_1} \times G_{A_2}$ as shown in Fig. 2b. The initial and final modes of applications define the initial and final modes of graph G_π. The number of nodes in graph G_π is $|\mathcal{S}| = |V_{A_1}| \cdot |V_{A_2}|$.

A path through the graph $G_\pi(E_\pi, \mathcal{S})$ is a sequence of transitions, starting at node $v_i \in \mathcal{S}$ and leading to another node $v_j \in \mathcal{S} \setminus \{v_i\}$, following the edges in E_π. Thus, a path describes a sequence of scenarios and the set of all feasible paths is denoted $\mathcal{C}$. Under the condition that paths are loop free, this set is finite.

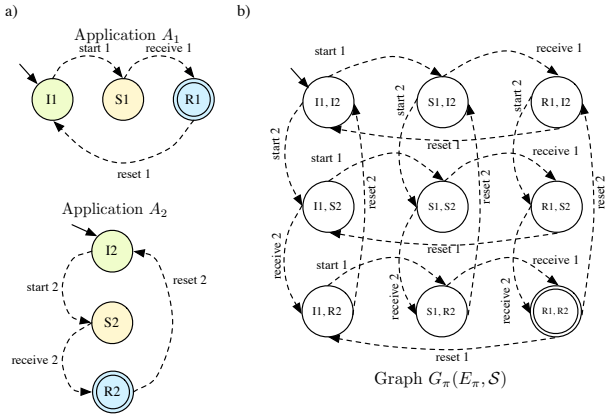


Fig. 2. a) Graphs representing applications A_1 and A_2
b) Cross product $G_\pi = G_{A_1} \times G_{A_2}$ representing all possible scenarios.

Modes of an application are denoted with an execution probability, such that for application $A_\ell \in \mathcal{A}$ the share of time a mode $\mu_k \in \mathcal{M}_\ell$ is active is denoted as $\hat{\chi}_k$. Modes of different applications are stochastically independent, and therefore the probability of a scenario $S_m \in \mathcal{S}$ is derived as the product of its constituting modes:

$$\chi_m = \prod_{\mu_k \in S_m} \hat{\chi}_k. \tag{1}$$

Conclusively, the execution probability ψ_i for a task $t_i \in \mathcal{T}$ can be computed, where

$$\psi_i = \sum_{S_m : t_i \text{ is active in scenario } S_m} \chi_m. \tag{2}$$

III. Problem Definition and Motivations

We explore how to derive a task to processing unit allocation, such that the power dissipation is minimized. Given the set $\mathcal{A}$ of applications, the task set $\mathcal{T}$, and a set $\mathcal{P}$ of available processing unit types, we compute (a) the number of allocated instances k_j of PU type $p_j \in \mathcal{P}$ and (b) the mapping of tasks onto the allocated processing units. The objective of the studied problem is to minimize the power dissipation, while the utilization constraints on processing units have to be satisfied.

It has been shown in [11] that the studied problem is $\mathcal{NP}$-hard in a strong sense and that it is not easy to have constant approximation factors with worst-case guarantees. The multi-step heuristic approach in [11] tries to compute a static mapping in polynomial time.

This static mapping considers a set of applications with a probability distribution and computes the set of scenarios from that. All tasks that are active in these scenarios are considered for computing a static task to processing unit allocation. First, consider the probability distribution to be known a priori. Diverging probability distributions in the actual system might result in inappropriate power dissipation. Second, since modes of an application can only execute in mutual exclusion, not all tasks can be active at a time. Considering all tasks for the allocation process limits the degree of freedom and therefore the performance of the system.

We propose a dynamic approach, which takes advantage of the applications structure. Instead of computing a global static mapping, we compute a mapping for each feasible sequence of scenarios $c_i \in \mathcal{C}$. Since probability distributions of applications are typically neither known nor static, we compute different mappings for a set of representative probability distributions. These mappings are called *templates* and are stored in a table. A manager chooses the appropriate template from this table at run-time.

As an example, consider a radio application. Sometimes synchronization is performed very frequently, due to bad signal reception. At times, data transmission is active more often. Thus, the modes' probability distributions change and only a subset of tasks is active at a time. Static task allocation can only cover one of the previously shown use-cases and might result in an increased power dissipation for the other.

IV. Proposed Mapping Methodology

This section describes our proposed dynamic approach. Based on the graph G_π representing the scenarios and their valid transitions, the offline part computes the set of paths through the graph, i.e., the scenario sequences $\mathcal{C}$. A static mapping for each scenario sequence is computed and stored as a template. The online part monitors the scenario transitions and, once a new scenario becomes active chooses an appropriate mapping from the pre-computed templates, see Fig. 3. We show how to derive a finite set of paths and how to adapt a static mapping approach for our purposes.

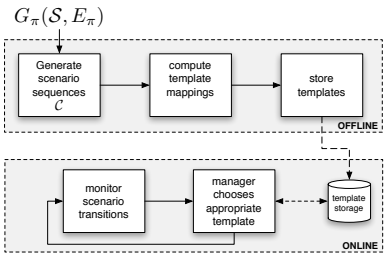


Fig. 3. Dynamic Mapping Approach - Overview of offline and online steps

A. Scenario Sequence Generation

Consider the cross product $G_\pi(E_\pi, \mathcal{S})$ in Fig. 2b as an example. We want to construct all possible paths from the initial to the final state. The initial and final modes of applications A_1 and A_2 in Fig. 2a are known, thus the initial and final states of graph G_π are `(I1,I2)` and `(R1,R2)` respectively. Considering loop free paths only, i.e., paths that traverse a state at most once, results in a finite set of paths.

Deriving the set of paths $\mathcal{C}$ can be formulated by a recursive algorithm as shown in Alg. 1. Consider $\mathcal{V}_{init}$ as the set of initial states and $\mathcal{V}_{end}$ as the set of final states. The set of paths from all $v_{init,i} \in \mathcal{V}_{init}$ to all $v_{end,j} \in \mathcal{V}_{end}$ is the set of scenario sequences $\mathcal{C}$ and $c_i \in \mathcal{C}$ is one unique sequence of scenarios. We apply Alg. 1 to any combination of initial and final states in $\mathcal{V}_{init}$ and $\mathcal{V}_{end}$ respectively. In our example in Fig. 2b, there is only one initial and one final state. Therefore, let Alg. 1 be initialized with $v_{init,1}$ = `(I1,I2)`, the initial sequence $c = v_i$, $v_{end,1}$ =`(R1,R2)` and the global variable $\mathcal{C} = \emptyset$. Each iteration adds the current state v_i to sequence c and computes its successor states. Successor states v_j, that lead to loop free paths are considered for another iteration of the algorithm. Once all the sequences are known, we can compute template mappings for each of them.

Algorithm 1 `compute_paths`

Global: $\mathcal{C}$
Input: $G_\pi(\mathcal{S}, E_\pi), v_{init}, v_{end}, v_i, c$;
1: **if** $v_i = v_{end}$ **then**
2: $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$
3: return;
4: **end if**
5: **for** each successor v_j of $v_i \in \mathcal{S}$ **do**
6: **if** $v_j \notin c$ **then**
7: compute_paths($G_\pi(\mathcal{S}, E_\pi), v_{init}, v_{end}, v_j, c \cup \{v_j\}$);
8: **end if**
9: **end for**

Upper bound on the number of scenario sequences Consider the graphs $G_{A_1}(V_{A_1}, E_{A_1})$ and $G_{A_2}(V_{A_2}, E_{A_2})$ representing applications A_1 and A_2 respectively. Assume that in graphs G_{A_1} and G_{A_2} each state can be reached by any other state in one step and that n is the upper bound of outgoing edges at each node of graph $G_\pi = G_{A_1} \times G_{A_2}$. Then, for general cases, the number of loop free paths can be bounded by $|\mathcal{C}| \leq \binom{\prod_{A_i \in \mathcal{A}} |V_{A_i}|}{n}$.

The graphs used to represent the feasible mode changes of our applications in Fig. 2a are much more restrictive, as they assume a sequential order of mode changes. The number of feasible paths for such graphs, assuming that each node can reach its successor as well as its predecessor in one step can be bounded by:

$$|\mathcal{C}| \leq \binom{\sum_{A_i \in \mathcal{A}} |V_{A_i}|}{n}. \tag{3}$$

B. Deriving templates and the hardware platform

The template mappings for our dynamic approach are computed by applying a static mapping procedure to each scenario sequence $c_i \in \mathcal{C}$. We define the static mapping, following [11]. The binary variable $M_{i,j,k} = 1$ indicates that a task t_i is allocated on instance k of processing unit type $p_j \in \mathcal{P}$, otherwise $M_{i,j,k} = 0$. The binary variable $Z_{m,j,k} = 1$ indicates that scenario $S_m \in \mathcal{S}$ is active on processing unit type p_j and instance k, and $Z_{m,j,k} = 0$ otherwise. Once a scenario is active on a processing unit, the static power σ_j is consumed on that processing unit. The probability χ_m of a scenario S_m defines the share of time static power consumption has to be consumed. The dynamic power dissipation is related to a unit's utilization and the tasks execution probabilities. The objective of the static mapping approach can be written as:

$$\sum_{S_m \in \mathcal{S}} \sum_{p_j \in \mathcal{P}} \sum_{\forall k} \sigma_j \chi_m Z_{m,j,k} + \sum_{t_i \in \mathcal{T}} \sum_{p_j \in \mathcal{P}} \sum_{\forall k} \delta_j \psi_i u_{i,j} M_{i,j,k} \tag{4}$$

, where the first term represents static power consumption and the second term represents dynamic power consumption. By considering the probability ψ_i of a task t_i, the approach optimizes for the most likely scenarios.

Feasibility of a mapping is guaranteed by a utilization bound. The binary variable $Z_{m,j,k}$ defines this utilization bound. Once a scenario is active on a processing unit and consumes static power, the utilization bound is 100% ($Z_{m,j,k} = 1$). Otherwise, the utilization bound is 0% ($Z_{m,j,k} = 0$). The approach minimizes the systems *expected average power dissipation* by an optimal (ILP), which is formulated in Eq. 5 and solved by an ILP solver.

Consider a scenario sequence $c_i \in \mathcal{C}$. It describes the transitions from an initial scenario S_{init} to another scenario S_{end}. Each scenario of that sequence activates and deactivates some tasks. This results in a set of tasks $\mathcal{T}_{c_i}$ that has to be considered for allocation. The tasks in $\mathcal{T}_{c_i}$ all belong to scenarios in the scenario sequence c_i, but not all scenarios that activate a task $t_i \in \mathcal{T}_{c_i}$ are traversed by the sequence c_i. The execution probability ψ_i of task $t_i \in \mathcal{T}_{c_i}$ has to be recomputed, such that only those scenarios are considered that are actually traversed by c_i, see Eq. 2. The mapping process is applied and a task allocation for each sequence $c_i \in \mathcal{C}$ is derived.

The static mapping approach is altered, such that only those scenarios $S_m \in \mathcal{S}$ are considered, that are traversed in sequence c_i. The number of instances k for each PU type $p_j \in \mathcal{P}$ is limited to be F_j. F_j can at most be $|\mathcal{T}|$, representing the case, when every task is executed on a single processing unit.

For larger problem instances a multi-step heuristic is used. First, an initial solution based on linear programming relax-

ation is derived, i.e., constraints in Eq. 5e and Eq. 5f can be any fractional variable. This may result in tasks being distributed over a large amount of low utilized PUs. Second, an iterative process remaps tasks to other processing unit instances, aiming to reduce static power consumption. Please refer to [11] for a detailed description.

min. (5a)

$$\sum_{S_m \in c_i} \sum_{p_j \in \mathcal{P}} \sum_{k=1}^{F_j} \chi_m \sigma_j Z_{m,j,k} + \sum_{t_i \in \mathcal{T}_m} \sum_{p_j \in \mathcal{P}} \sum_{k=1}^{F_j} u_{i,j} \delta_j \psi_i M_{i,j,k} \tag{5b}$$

s.t.

$$\sum_{t_i \in \mathcal{T}_{new}} M_{i,j,k} u_{i,j} \leq Z_{m,j,k}, \quad \begin{matrix} \forall p_j \in \mathcal{P}, \\ \forall k = 1, 2, \ldots, F_j \end{matrix} \tag{5c}$$

$$\sum_{p_j \in P} \sum_{k=1}^{F_j} M_{i,j,k} = 1, \; \forall t_i \in \mathcal{T}_m, \tag{5d}$$

$$M_{i,j,k} = \{0,1\} \, \forall t_i \in \mathcal{T}_{c_i}, p_j \in \mathcal{P}, k = 1, 2, \ldots, F_j, \tag{5e}$$

$$Z_{m,j,k} = \{0,1\}, \; p_j \in \mathcal{P}, k = 1, 2, \ldots, F_j. \tag{5f}$$

The process of deriving all templates for all scenario sequences can be formulated as in Alg. 2.

Algorithm 2 `compute_templates`

Input: $\mathcal{P}, G_\pi(\mathcal{S}, E_\pi), \mathcal{T}, \mathcal{V}_{init}, \mathcal{V}_{end}$;
Global: $\mathcal{C}$
Output: templates for $\mathcal{C}$

1: **for** all combinations of $v_{init,i} \in \mathcal{V}_{init}$ and $v_{end,j} \in \mathcal{V}_{end}$ **do**
2: $c \leftarrow v_{init,i}$
3: $\mathcal{C} =$ `compute_paths`$(G_\pi(\mathcal{S}, E_\pi), v_{init,i}, v_{end,j}, v_{init,i}, c)$
4: **end for**
5: **for** each $c_j \in \mathcal{C}$ **do**
6: initialize Z, M
7: $\mathcal{T}_{c_j} \leftarrow$ active tasks in sequence c_j
8: recompute ψ_i for each task $t_i \in \mathcal{T}_{c_j}$
9: $M, Z \leftarrow$static_mapping$(c_j, \mathcal{T}_{c_j}, \mathcal{P})$;
10: store mapping in M, Z as template for c_j
11: **end for**

Deriving a hardware platform Each sequence $c_i \in \mathcal{C}$ results in a distinct mapping and therefore a distinct PU allocation. The resulting hardware platform is made up of processing unit types $p_j \in \mathcal{P}$ and their corresponding number of instances $k_j \forall p_j$. Assuming the maximum number of instances $k_j \forall p_j \in \mathcal{P}$ from all computed mappings results in a single hardware platform that guarantees feasibility for any scenario sequence $c_i \in \mathcal{C}$.

C. Online mapping

Templates are stored on the system and a manager observes scenario transitions. Based on this observation a pre-computed template is chosen. Adaptivity to altering execution probabilities is achieved by providing templates for a set of execution probabilities.

There is one mapping for each scenario sequence and the number of such sequences is bound by Eq. 3. Assuming two outgoing transitions at each scenario, as shown in our example in Fig. 2, the number of templates is bound by $\binom{\sum_{A_i \in \mathcal{A}} |V_i|}{2}$. As an example, consider two applications, constituted by a total of 34 tasks and an assumed hardware platform of 20 processing units. Then the template mappings require 1kb of memory, or 12 times the amount required by a global static mapping.

V. Experiments

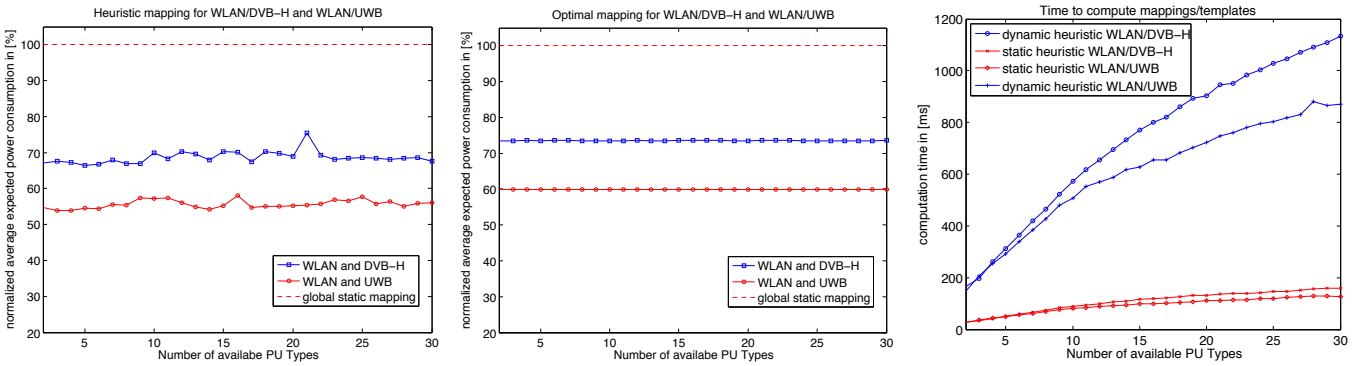
Experiments are done with two sets of concurrent applications, both using SDR applications. The first experiment exploits Wireless Local Area Network (WLAN) and Digital Video Broadcast - Handhelds (DVB-H) applications. The WLAN application consists of 23 tasks, where 10 are active in multiple modes. The DVB-H application consists of 11 tasks, and none is active in multiple nodes. The second set of experiments considers WLAN and Ultra-Wideband (UWB) applications. The UWB application consists of 8 tasks, where one is active in multiple modes. Each application is composed by 3 modes, see Fig. 2a, and parameters are generated as random variables, following the definitions given in [11]. The PU type library $\mathcal{P}$ size varies in the experiments, from a minimum of 2 to a maximum of 30 available PU Types. For fair comparison, we reuse the publicly available data by Schranzhofer et al. in [11].

For each library size and each realization thereof, the templates for our proposed dynamic mapping approach are computed. We compute a template for each sequence $c_i \in \mathcal{C}$ and the corresponding average expected power consumption. For our particular experimental applications, there are 12 scenario sequences. The overall power dissipation can be computed by the sum of the sequences' power dissipation and their respective probability. Providing templates for different probability distribution increases the computation time and the memory requirement linearly with the number of distributions. Eventually, a global static mapping is computed by applying the approaches in [11] as a reference value.

In Fig. 4(a) we compare the average expected power consumption of our dynamic approach to the global static mapping. We report the respective applications' average expected power consumption normalized to the power consumption a global static mapping would result in. We use the multi-step heuristic approach to compute the templates and the global static mapping. Fig. 4(b) shows the same comparison by using the optimal (ILP) approach to compute the respective templates and the global static mapping. In Fig. 4(c) the time required to compute the templates and static mappings is presented.

A. Results

Fig. 4(a) and Fig. 4(b) show the reduction of power dissipation we achieve with our proposed dynamic mapping approach in comparison to the global static mapping. Fig. 4(a) and Fig. 4(b) reveal that for the experiments using the WLAN and the DVB-H application a reduction of power consumption of up to 32% can be achieved. The multi-step heuristic approach in Fig. 4(a) shows a slightly increased power reduction compared to the optimal approach in Fig. 4(b). The multi-step heuristic approach benefits from the fact that the dynamic



(a) multi-step heuristic mappings/templates (b) optimal mappings/templates (c) Time complexity

Fig. 4. Experimental Results.

mapping approach computes a number of templates instead of a global static mapping. This reduces the number of tasks and scenarios that have to be considered, from which the multi-step heuristic can benefit more than the optimal approach.

The second experiment uses the WLAN and the UWB applications. Fig. 4(a) and Fig. 4(b) reveal a reduction of average expected power consumption of 40 - 45 %. In this experiment the total number of tasks is smaller, but there are more tasks that are active in multiple modes. This experiment clearly shows the strengths of our proposed dynamic approach

Fig. 4(c) presents the time required to compute the templates and the static mapping using a multi-step heuristic. Instead of a single global mapping, we have to compute 12 templates. Therefore the required computation time increases compared to the global static mapping approach. However, there are 12 templates to compute but the required computation time is only 9-fold, while the absolute amount of time to compute the templates rises to only 1.2 seconds for large processing unit libraries.

Conclusively, computing a single mapping is less challenging than computing a global static mapping. In [11] it is shown that the optimal global static mapping fails to deliver results in up to 6% due to complexity reasons. Our approach is able to compute optimal templates for all instances and all PU type library sizes in the experiments.

VI. Conclusion

This paper studies a dynamic mapping strategy based on pre-computed template mappings. We introduce an application model that defines applications as sets of modes and assigns execution probabilities to each mode. Multiple concurrently executing applications result in a set of scenarios and each scenario can be attained by sequences of mode changes. We show that there exists a finite set of sequences to attain each scenario. We compute a static mapping for each sequence and store all the mappings as templates on the system. A manager observes mode changes at runtime and chooses an appropriate pre-computed template to assign newly arriving tasks to processing units. Compared to the static-mapping approaches in [11], experimental results reveal that we can achieve a reduction on average expected power consumption of 40 - 45%, while keeping the introduced overhead to store the template mappings as low as 1kb. The ability to adapt to altering execution probabilities is provided by templates.

References

[1] L. Benini, D. Bertozzi, and M. Milano. Resource management policy handling multiple use-cases in mpsoc platforms using constraint programming. In *ICLP '08: Proceedings of the 24th International Conference on Logic Programming*, pages 470–484, Berlin, Heidelberg, 2008. Springer-Verlag.

[2] J.-J. Chen, H.-R. Hsu, and T.-W. Kuo. Leakage-aware energy-efficient scheduling of real-time tasks in multiprocessor systems. *Real-Time and Embedded Technology and Applications Symposium, IEEE*, 0:408–417, 2006.

[3] J.-J. Chen, A. Schranzhofer, and L. Thiele. Energy minimization for periodic real-time tasks on heterogeneous processing units. In *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–12, May 2009.

[4] J. Heo, D. Henriksson, X. Liu, and T. Abdelzaher. Integrating adaptive components: An emerging challenge in performance-adaptive systems and a server farm case-study. In *Proceedings of the 28th IEEE Real-Time Systems Symposium (RTSS)*, pages 227–238, 2007.

[5] R. Jejurikar, C. Pereira, and R. Gupta. Leakage aware dynamic voltage scaling for real-time embedded systems. In *DAC '04: Proceedings of the 41st annual Design Automation Conference*, pages 275–280, New York, NY, USA, 2004. ACM.

[6] M. Kim, S. Banerjee, N. Dutt, and N. Venkatasubramanian. Energy-aware cosynthesis of real-time multimedia applications on MPSoCs using heterogeneous scheduling policies. *ACM Trans. Embed. Comput. Syst.*, 7(2):1–19, 2008.

[7] O. Moreira, J.-D. Mol, M. Bekooij, and J. v. Meerbergen. Multiprocessor resource allocation for hard-real-time streaming with a dynamic job-mix. In *RTAS '05: Proceedings of the 11th IEEE Real Time on Embedded Technology and Applications Symposium*, pages 332–341, Washington, DC, USA, 2005. IEEE Computer Society.

[8] O. Moreira, J. J.-D. Mol, and M. Bekooij. Online resource management in a multiprocessor with a network-on-chip. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, pages 1557–1564, New York, NY, USA, 2007. ACM.

[9] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli. Mapping and configuration methods for multi-use-case networks on chips. In *ASP-DAC '06: Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, pages 146–151, Piscataway, NJ, USA, 2006. IEEE Press.

[10] M. T. Schmitz, B. M. Al-Hashimi, and P. Eles. Cosynthesis of energy-efficient multimode embedded systems with consideration of mode-execution probabilities. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 24(2):153–169, Feb. 2005.

[11] A. Schranzhofer, J.-J. Chen, and L. Thiele. Power-aware mapping of probabilistic applications onto heterogeneous MPSoC platforms. In *RTAS '09: Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 151–160, Washington, DC, USA, 2009. IEEE Computer Society.

[12] R. Xu, D. Zhu, C. Rusu, R. Melhem, and D. Mossé. Energy-efficient policies for embedded clusters. In *LCTES '05: Proceedings of the 2005 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 1–10, New York, NY, USA, 2005. ACM.

[13] C. Yang and A. Orailoglu. Towards no-cost adaptive MPSoC static schedules through exploitation of logical-to-physical core mapping latitude. *Design, Automation & Test in Europe Conference & Exhibition, 2009. DATE '09.*, pages 63 – 68, Mar 2009.