An Iterative Multi-Application Mapping Algorithm for Heterogeneous MPSoCs

Wei Quan[†,‡]

[†]Informatics Institute
University of Amsterdam
The Netherlands
{w.quan,a.d.pimentel}@uva.nl

Andy D. Pimentel[†]

[‡]School of Computer Science
National University of Defense Technology
Hunan, China
quanwei02@gmail.com

***Abstract*—Task mapping plays a crucial role in achieving high performance and energy savings in heterogeneous multiprocessor platforms. The problem of optimally mapping tasks onto a set of given heterogeneous processors for maximal throughput/minimal overall energy consumption has been known, in general, to be NP-complete. This problem is exacerbated when mapping multiple applications onto the target platform. To address this problem, this paper proposes an iterative multi-application mapping algorithm that operates at run time. Based on statically derived optimal (or near optimal) mappings for each separate application, this algorithm will quickly find a near optimal mapping under the objectives of high performance and low energy consumption for the simultaneously running applications on heterogeneous platforms. We have evaluated our algorithm using a heterogeneous MPSoC system with three real applications. Experimental results reveal the effectiveness of our proposed algorithm by comparing derived solutions to the ones obtained from other well-known algorithms.**

I. Introduction

Modern embedded systems, which are more and more based on heterogeneous Multi-Processor System-on-Chip (MPSoC) architectures, often require supporting an increasing number of applications and standards. In these systems, multiple applications can run concurrently and are thus simultaneously contending for system resources. As heterogeneous architectures are capable of providing better performance and energy trade-offs than their homogeneous counterparts [14], the process of application task mapping plays a crucial role in exploiting the system properties such that applications can meet their, often diverse, demands on performance and energy efficiency [25].

The problem of optimally mapping tasks onto a given set of heterogeneous processors for maximal throughput (performance) or minimal overall energy consumption has been known, in general, to be NP-complete [13]. When considering mapping multiple applications onto a target architecture, this problem is exacerbated as the resource contention between applications should be carefully considered in this case. State-of-the-art methods for solving this problem can be divided into two categories: static and dynamic task mapping algorithms which, respectively, work at design time and run time. Traditionally, the task mapping problem is solved statically at design time for which there are many known task mapping algorithms targeting different application domains and different hardware architectures (e.g. [19], [7], [11]). These algorithms typically use computationally intensive search methods to find the optimal mapping or near optimal mapping for the applications that may run on the system. Dynamic task mapping techniques, on the other hand, cannot be computationally intensive as they have to efficiently make task mapping decisions at run time. Therefore, these techniques typically use heuristics to find good task mappings. Evidently, static task mapping techniques usually obtain mappings of higher quality compared to those derived from dynamic algorithms as the former allow for exploring a larger design space for the underlying architecture. This, of course, at the cost of consuming more time. Another drawback of static mapping techniques is that they cannot cope with dynamic application behavior in which different combinations of applications can be executing concurrently over time that are contending for system resources. In this paper, we propose an Energy-aware Iterative multi-application Mapping (EIM) algorithm for heterogeneous multimedia MPSoCs that tries to exploit the advantages from both static and dynamic algorithms.

The proposed approach can be divided into two stages. Firstly, the design-time stage will perform design space exploration (DSE) to find and store three optimal mappings for each application with the objectives of maximizing the throughput, minimizing the energy consumption and maximizing the throughput under a predefined energy budget respectively. Secondly, the run-time stage dynamically optimizes the mapping of multiple simultaneously running applications with the objective of maximizing the throughput under the predefined energy budget or minimizing the system energy consumption based on the optimal mappings of corresponding applications explored in the first stage. Combining these two steps, the proposed approach can dynamically find a near optimal mapping for multiple executing applications, while it is also capable of running single applications under the optimal mapping (derived from design-time DSE) with respect to different optimization objectives.

To support the dynamism of applications, we use the concept of *scenarios* [18], [6]. Here, one can distinguish two forms of scenarios to capture dynamic application behavior: inter-application scenarios describe the simultaneously running applications in the system, while intra-application scenarios define the different execution modes within each application. The combination of these inter- and intra-application scenarios

are called *workload scenarios*, and specify the application workload in terms of the different applications that are concurrently executing and the mode of each application. At design time, a system designer could aim at finding the optimal mapping of application tasks to MPSoC processing resources for each inter- and intra-application scenario with different objectives (performance/energy). However, when the number of applications and application modes increase, the total number of workload scenarios will explode exponentially. Considering, e.g., 10 applications with 5 execution modes for each application, there will be 60 million workload scenarios. If it takes one second to find the optimal mapping for each scenario at design time, then one would need nearly two years to obtain all the optimal mappings. Moreover, storing all these optimal mappings such that they can be used at run time by the system to remap tasks when a new scenario is detected would also be unrealistic as this would take up too much memory storage.

In this paper, we solve this problem by splitting the handling of intra-application scenarios and inter-application scenarios according to the two stages mentioned above. The design-time phase takes charge of exploring three optimal mappings for each intra-application scenario of each application with respect to three different objectives: maximizing the throughput, minimizing the energy and maximizing the throughput under the predefined energy budget. The run-time phase subsequently finds a mapping for the workload scenario that has emerged in the system by considering the active inter-application scenario at run time. By using this approach, the number of mappings that need to be determined at design time will be greatly reduced. Considering the above example, only 150 mappings need to be found (and stored) at design time. This overcomes the drawbacks of static mapping algorithms (time and memory usage for exploring and storing mappings for every workload scenario) but also takes advantage of the capability of run-time algorithms to support dynamic application behavior.

The remainder of this paper is organized as follows. Section II gives some prerequisites and the problem definition for this paper. Section III provides a detailed description of our iterative multi-application mapping algorithm. Section IV introduces the experimental environment and presents the results of our experiments. Section V discusses related work, after which Section VI concludes the paper.

II. Prerequisites and Problem Definition

In this section, we explain the necessary prerequisites for this work and provide a detailed problem definition.

A. Application Model

In this paper, we target the multimedia application domain. For this reason, we use the Kahn Process Network (KPN) model of computation [12] to specify application behaviour since this model of computation fits well to the streaming behaviour of multimedia applications. In a KPN, an application is described as a network of concurrent processes that are interconnected via FIFO channels. This means that an application can be represented as a directed graph $KPN = (P,F)$ where P is set of processes (tasks) p_i in the application and $f_{ij} \in F$ represents the FIFO channel between two processes p_i and p_j. Figure 1 shows the KPN of a Motion-JPEG (MJPEG) decoder application.

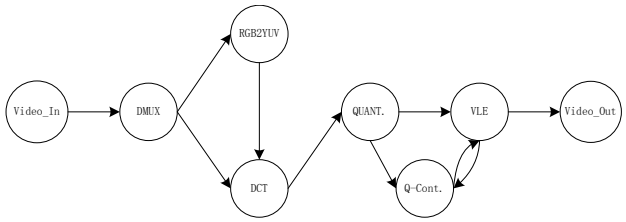


Figure 1: KPN for MJPEG.

B. Architecture Model

In this work, we restrict ourselves to heterogeneous MPSoC architectures with shared memory. An architecture can be modeled as a graph $MPSoC = (PE,C)$, where PE is the set of processing elements used in the architecture and C is a multiset of pairs $c_{ij} = (pe_i, pe_j) \in PE \times PE$ representing a (FIFO) communication channel between processors pe_i and pe_j. Combining the definition of application and architecture models, the computation cost of task (process) p_i on processing element pe_j is expressed as T_i^j and the communication cost between tasks p_i and p_j via channel c_{xy} that connects pe_x and pe_y is $C_{ij}^{c_{xy}}$. With respect to power consumption, SP_i and DP_i refer to the static and dynamic power consumption for pe_i. Besides processing elements, another main component of energy consumption in our target system is the shared memory. For this component, we denote the static and average dynamic power consumption (for read/write transactions) as *SM* and *DM* respectively.

C. Task Mapping

The task mapping defines the corresponding relationship between the tasks in a KPN application and the underlying architecture resources. For a single application, given the *KPN* of this application and a target MPSoC, a correct mapping is a pair of unique assignments ($\mu : P \to PE$, $\eta : F \to C$) such that it satisfies $\forall f \in F, src(\eta(f)) = \mu(src(f)) \wedge dst(\eta(f)) = \mu(dst(f))$. When tasks are mapped onto the underlying architecture, the usage U_k of each pe_k can be calculated by equation 1, where $p_i \mapsto pe_k$ and $p_j \mapsto pe_y$ mean that tasks p_i, p_j are mapped onto processors pe_k and pe_y respectively.

$$U_k = \sum_{p_i \mapsto pe_k, p_j \mapsto pe_y} (T_i^k + C_{ij}^{c_{ky}}) \quad (1)$$

In the case of a multi-application workload, the state of simultaneously running applications that are distinguished as inter- and intra-application scenarios should be considered in the task mapping. Let $A = \{app_0, app_1, ..., app_m\}$ be the set of all applications that can run on the system, and $M^i = \{md_0^i, md_1^i, ..., md_n^i\}$ be the set of possible execution modes for $app_i \in A$. Then, $SE = \{se_0, se_1, ..., se_{n_{inter}}\}$, with $se_i = \{app_0 = 0/1, ..., app_m = 0/1\}$ and $app_i \in A$, is the set of all inter-application scenarios. And $sa_j^i = \{app_0 = md_{j_0}^0, ..., app_m = md_{j_m}^m\}$, with $app_i \in A \wedge app_i = 1 \in se_i$ and $md_{j_x}^i \in$

M^i, represents the j-th intra-application scenario in inter-application scenario $se_i \in SE$. The set of all workload scenarios can then be defined as the disjoint union $S = \sqcup_{i \in SE} SA^i$, with $SA^i = \{sa^i_1, sa^i_2, ..., sa^i_{n^i_{intra}}\}$.

As already explained in the previous section, we propose to perform the task mapping of applications in two stages. In the first stage, which is performed at design time, we perform DSE for each intra-application scenario of each application (denoted by scenario s_i in the whole workload scenario space S) to find three mappings that show the maximal throughput, minimal energy consumption and maximal throughput under a certain energy budget b_i respectively. Here, b_i is a user defined energy budget for workload scenario s_i. The mappings derived from design-time DSE are stored so they can be used by the second stage to get a final mapping – by directly using the stored mappings or by deriving a new one from the stored mappings – for the current system objective when a new workload scenario is detected. Here, we can distinguish two system objectives: maximal throughput and minimal energy consumption for each workload scenario, denoted as O_t and O_e respectively. These two objectives will be used for run-time mapping optimization. For the convenience of exploring the pareto front of objectives at design time, we change the objective of maximal throughput into a minimal objective $O_p = 1/O_t$, namely the scenario execution time. With regard to the run-time behavior, we assume that our hardware platform can run under *two modes*: energy-aware high performance mode (using a certain energy budget) and energy saving mode. Consequently, the run-time system objectives for each workload scenario are O_{pb}, which means minimal scenario execution time (or maximal throughput) under the energy budget, and O_e. Users can choose the running mode of the system, or the system itself can adaptively adjust the mode based on e.g. the battery usage.

Under these definitions and given the $KPN = (P,F)$ for each application and an $MPSoC = (PE,C)$, our goal is to find the optimal or near optimal mapping at run time for each detected workload scenario $s_i \in S$ with the objective to minimize O_{pb} or O_e based on the system execution mode.

III. Iterative Multi-Application Mapping Optimizing

Our EIM algorithm, which is outlined in Algorithm 1, can be divided into a static part and a dynamic part. The static part is used to capture the intra-application dynamism in those inter-application scenarios with only a single active application. For these inter-application scenarios $se_i \in SE$, we have determined – using design-time DSE – optimal or near optimal mappings (optimized for O_{pb} and O_e) for each intra-application scenario $sa^i_j \in SA^i$ of se_i. To this end, we have deployed a so-called scenario-based DSE approach [26], which is based on the well-known NSGA-II genetic algorithm. As this design-time DSE stage is not the main focus of this paper, we refer the interested reader to [26] for further details.

The mappings derived from this design-time DSE are used by the static part of our EIM algorithm as shown in lines 1-3 of Algorithm 1. When the system detects a new workload scenario, the algorithm will first choose the corresponding optimal mapping – as derived from the design-time DSE stage and stored in a so-called *scenario database* – for each application active in the detected workload scenario as the initial mapping. This process is implemented in the function of line 1 of Algorithm 1 which will be explained in detail in the next paragraph. As the database only stores mappings for the intra-application scenarios of each single application, its size typically is relatively small. However, if its size becomes too large, then the size can be controlled by clustering intra-application scenarios [6], [20] and choosing a proper granularity of scenario clusters.

If there is only a single application active in the workload scenario, then the initial mapping will be chosen from one of the following two statically derived mappings, based on the system execution mode: the mapping with the maximal throughput under a given energy budget (the mapping optimized for O_{pb}) or the mapping with the minimal energy consumption (the mapping optimized for O_e). Hereafter, as shown in lines 2-3 of Algorithm 1, the algorithm will directly return the initial mapping as a final mapping decision. Otherwise, if there are multiple applications active simultaneously, then the mapping with maximal throughput (the mapping optimized for O_p) or minimal energy consumption (based on the system mode) for each active application will be chosen as initial mappings. These initial per-application mappings will then simply be merged together to form the initial mapping for the complete workload scenario. Here, there are two reasons for not choosing the mapping with maximal throughput under a certain energy budget as the initial mapping in the energy-aware high performance mode. First, the communication locality behavior of the mapping with maximal throughput under an energy budget typically is not as good as the one with maximal throughput without an energy budget. Our run-time algorithm exploits this locality incorporated in the initial per-application mappings for further improvement of the workload scenario mapping. Second, we will consider the energy constraints during the mapping optimization process at run time, so we do not yet have to consider an energy budget for the initial mapping in the case of an active multi-application workload scenario.

The dynamic part of our EIM algorithm is only used for those workload scenarios that contain multiple simultaneously active applications and is outlined in lines 4-16 of Algorithm 1. It aims at further optimizing the initial mapping found during the static part of the EIM algorithm, as described above. To this end, it distinguishes the system execution mode to take different strategies for mapping optimization. As described in the previous section, our target heterogeneous MPSoC system can run under energy-aware high performance and energy saving modes. We will use different strategies in these two modes targeting different optimization objectives. These two strategies in the dynamic part of the EIM algorithm are described below. For the propose of a better understanding of our algorithm, the metrics used in algorithms 2 and 3 are shown in Table I.

A. Performance Optimization

When the MPSoC system is running under the energy-aware high performance mode, our algorithm will optimize the mapping for the active multi-application scenario with the objective to minimize the system metric O_{pb}. Consequently,

Algorithm 1 EIM algorithm

Input: $KPN_{app_{active}}$, $MPSoC$, scenario_id(s_i), sys_mode
Output: (μ, η)
1: (μ, η) = getInitMapping(s_i, sys_mode);
2: if singleAppActive(s_i) == true:
3: return (μ, η);
4: else:
5: switch(sys_mode):
6: case EA-HIGHPERF:
7: U = peUsage($KPN_{app_{active}}$, $MPSoC$, μ, η);
8: M_p = maxPUsage(U);
9: V_p = varPUsage(U);
10: b_i = eBudget(s_i);
11: return iterativePOpt(μ, η, M_p, V_p, b_i);
12: case ENERGYSAVING:
13: econs = energyCons($KPN_{app_{active}}$, $MPSoC$, μ, η);
14: return iterativeEOpt(μ, η, econs);
15: default:
16: return (μ, η);

the optimal mapping for each scenario is the one that has the minimal O_{pb} among all the possible mappings under energy budget of b_i for workload scenario s_i. It is, however, extremely hard to find the optimal mapping for each workload scenario at run time because of the following reasons. Firstly, as one cannot obtain the true value of O_p before actually executing the application on the target platform, an estimated $O_p^{'}$ needs to be used to guide the algorithm to find the optimal mapping. Here, there exists of course a clear accuracy/overhead trade-off between different estimation techniques. Efficient but less accurate run-time mapping-performance estimation techniques may lead to sub-optimal mappings, while the high overhead of more accurate techniques may neutralize the performance benefits of the mapping optimization itself. Secondly, the mapping problem is NP-complete, as was mentioned before. It is unrealistic for a run-time mapping algorithm to explore the entire searching space to determine the optimal mapping for a scenario. An alternative method is using heuristics to search a part of the mapping space which may contain the optimal or a near optimal mapping.

To solve the above problems, we change the objective of performance into two other metrics: M_p and V_p that represent the maximal usage and usage variation in $U_k \in U$, where $pe_k \in PE$ and U is an array of processor usage with a total number of $|PE|$ elements. These two metrics will be used to optimize the bottleneck of application pipeline and balance the system workload. In this case, we do not need to use the metric O_p as the optimization objective, thereby addressing the first of the two above problems. Regarding the second problem, by using an optimization heuristic based on the metrics M_p and V_p, we aim at finding an optimal or near optimal mapping in a computationally efficient fashion. The rationale behind this heuristic is that a better mapping for the objective of high performance usually has smaller M_p and V_p values. For the purpose of restricting the energy consumption of the resulting mapping, we use the estimated energy consumption of a mapping (μ_j, η_j) for workload scenario s_i given by equation 2 and the energy budget b_i calculated by equation 3. Here, the index e in E_{ie} represents the energy-optimized mapping stored in memory for app_k, to control the searching space of possible mappings. The details of equation 2 will be explained in the next subsection. In equation 3, the first part α is a user defined constant scaling factor set for the energy budget and the second part represents the estimated minimal energy consumption for a workload scenario.

$$E_{ij} = E_p^{'} + E_m^{'} \tag{2a}$$

$$E_p^{'} = \sum_{active\_pe_k} (DP_k * U_k + SP_k * argmax(U_k)) \tag{2b}$$

$$E_m^{'} = DM * \sum_{\substack{c_{xy}=mem \\ f_{rt} \mapsto c_{xy} \in \eta_j}} (C_{rt}^{c_{xy}}) + SM * argmax(U_k) \tag{2c}$$

$$b_i = \alpha * \sum_{active\_app_k \in s_i} E_{ie} \tag{3}$$

The mapping algorithm for the energy-aware high performance mode is outlined in Algorithm 2, which will be executed in an iterative fashion. The starting mapping used in this algorithm is the one derived from Algorithm 1. In each iteration, it first proposes a new mapping for each active application as shown in line 2 of Algorithm 2. In this process, the algorithm searches the mapping space using the following greedy pattern: it checks the processors in U_k in descending order to determine whether the KPN application in question has a task or a bundle of adjacent, communicating tasks[1] resident on this processor. If so, then the algorithm finds a possible substitute processor for the task/adjacent tasks that satisfies the following conditions:

1) The $M_p^{'}$ of the new mapping is smaller than the M_p of the old mapping
2) If the previous condition cannot be satisfied, then the algorithm tries to find a substitute processor for which the resulting $M_p^{'}$ is equal to M_p and $V_p^{'}$ is smaller than V_p. If the first condition was satisfied, then this condition will never be used in this particular iteration
3) The estimated energy consumption of the new mapping should be smaller than the energy budget b_i.

The above process proposes new mappings for those applications that satisfy the conditions (for the other applications, the mapping remains unaltered). These newly proposed mappings are either a mapping that has a minimal $M_p^{'}$ (if condition 1 has been satisfied) or a mapping with minimal $V_p^{'}$. However, in the above process, it can also be the case that there are multiple new mappings proposed for an application, e.g. when there are multiple tasks (or task bundles) that can be remapped and for which the above conditions hold. In these cases, we use another metric, L, to decide on the final proposed mapping, where the value of L needs to be minimized. The metric L tries to capture the performance loss of a task remapping for

[1]Mapping such a task bundle to a single processor is the outcome of the design-time mapping optimization to reduce communication overhead.

Table I: Metrics used in Algorithms 2 and 3

Metrics	Description
(μ_j, η_j)	the mapping proposed by app_j
M_p^j	the maximal usage in U_k under the mapping of (μ_j, η_j)
V_p^j	the maximal usage variation in U_k under the mapping of (μ_j, η_j)
L^j	the performance loss of a remapping from (μ, η) to (μ_j, η_j)
M_p^k	the minimal M_p among M_p^j
(μ_k, η_k)	the mapping with M_p^k
V_p^k	the V_p of the mapping (μ_k, η_k)
W_p^t	the minimal $V_p + L$ among $V_p^j + L^j$
(μ_t, η_t)	the mapping with W_p^t
V_p^t	the V_p of the mapping (μ_t, η_t)
E_w	the minimal mapping energy consumption among E_{ij}
(μ_w, η_w)	the mapping with E_w

Algorithm 2 IPO algorithm

```
//performance optimization for workload scenario s_i
iterativePOpt(μ, η, M_p, V_p, b_i):
1: for each active app_j:
2:     (μ_j, η_j) = getPSubstitute(μ, η);
3:     if (μ_j, η_j) != (μ, η):
4:         U = peUsage(KPN_app_active, MPSoC, μ_j, η_j);
5:         M_p^j = maxPUsage(U);
6:         V_p^j = varPUsage(U);
7:         L^j = perfLoss(app_j, μ, η, μ_j, η_j);
8: M_p^k = argmin(M_p^j);
9: if M_p^k < M_p:
10:    (μ*, η*) = (μ_k, η_k);
11:    iterativePOpt(μ*, η*, M_p^k, V_p^k, b_i);
12:else:
13:    W_p^t = argmin(V_p^j + L^j);
14:    (μ*, η*) = (μ_t, η_t);
15:    if (μ*, η*) == (μ, η):
16:        return (μ, η);
17:    else:
18:        iterativePOpt(μ*, η*, M_p^t, V_p^t, b_i);
```

the application in question[2] and is calculated using equation 4.

$$L = \sum_{p_k \in B_i^j} (T_k^j - T_k^i) + (C_{kt}^{cjl} - C_{kt}^{cil}) \quad (4)$$

Here, we mark the task/task bundle that needs to be remapped from pe_i to pe_j as B_i^j.

After the algorithm has proposed a new mapping for each application, the next step is to select the *most effective* among these remapping proposals to be used for the next optimization iteration of the algorithm based on the metrics of each new mapping calculated in lines 4-7 of Algorithm 2 or return a mapping as the final one. This whole process is shown in lines 8-18 of Algorithm 2. If no new mapping has been proposed for any of the applications in the workload scenario in the previous step, then the input mapping will be returned as the final optimized result. Otherwise, we use the following conditions to select the most effective remapping for the next iteration of the algorithm:

1) If there is one and only one proposed mapping that has the minimal $M_p^{'}$ and this $M_p^{'}$ is smaller than the M_p of the original mapping, then this mapping will be passed to the next mapping optimization iteration (lines 9-11 in Algorithm 2).
2) If the first condition has not been satisfied, then the proposed mapping with $argmin(V_p^{'} + L)$ will be taken as the input mapping for the next iteration (lines 12-18 in Algorithm 2). The rationale behind this is that the algorithm tries to gradually optimize the mapping for the entire workload scenario while keeping the performance loss for a single application due to task remappings as small as possible (i.e., taking into account the processor affinity of the tasks proposed to be remapped).

The time complexity of our EIM algorithm is highly dependent on the diversity (or locality) of each pre-optimized (i.e., statically derived) mapping stored in system memory, especially considering the iteration count of our EIM algorithm. The diversity $|D_i|$ of an application (app_i) mapping is defined as the number of pipeline segments in this mapping. Under this definition, $|D_i| = 1$ and $|D_i| = |P_i|$ mean that all the tasks in app_i are mapped onto a same processor and different processors respectively. In the function on line 2 in Algorithm 2, the maximal number of possible new mappings is $|PE| * |D_i| * |PE|$. For each possible new mapping, the time consumed for computing the values of M_p and V_p is $O(|PE||P| + |F||C| + 2|PE|)$. Consequently, the time complexity of each active application in each iteration is $O(|PE|^2|D_i|(|PE||P| + |F||C| + 2|PE|))$. The approximate time complexity of each iteration then is $O(|PE|^2|D|(|PE||P| + |F||C| + 2|PE|))$. As the algorithm searches the mapping space to minimize M_p and V_p simultaneously, the maximal iteration count of Algorithm 2 is $argmax(|D|, |D||PE|)$, where the first and the second argument represent the maximal iteration count needed for searching each of the above metrics. Then, the overall time complexity of Algorithm 2 is $O(|PE|^3|D|^2(|PE||P| + |F||C| + 2|PE|))$, where $|PE|$, $|D|$, $|P|$, $|F|$ and $|C|$ respectively represent the total number of processor elements, the sum of $|D_i|$ of each active application app_i, the total number of active tasks, the total number of active FIFO channels and the total number of communication channels.

B. Energy Optimization

The algorithm used in the energy saving system mode is shown in Algorithm 3, which is similar to the algorithm for the high performance mode. It will iteratively optimize the mapping with the objective O_e for a unit of input workload (e.g., frame in the domain of multi-media applications). For the purpose of energy savings, we need not only to consider the dynamic energy consumption but also the static energy consumption. The energy consumption E_{ij} of a mapping (μ_j, η_j) for workload scenario s_i is calculated by equation 2, where $E_p^{'}$ is the dynamic and static energy consumed by all active processors and $E_m^{'}$ represents the dynamic and static energy consumption of the shared memory. This relatively simple energy model is built on several assumptions of the

[2]We note that L can be negative, implying that the task/task bundle has a higher affinity with the processor it is proposed to be mapped on.

Algorithm 3 IEO algorithm

//energy optimization for workload scenario s_i
iterativeEOpt(μ, η, E):
1: for each active app_j:
2: (μ_j, η_j) = getESubstitute(μ, η);
3: E_{ij} = energyCons($KPN_{app_{active}}$, $MPSoC$, μ_j, η_j);
4: $E^w = argmin(E_{ij})$;
5: if $E^w \geq E$:
6: return (μ, η);
7: else:
8: $(\mu^*, \eta^*) = (\mu_w, \eta_w)$;
9: iterativeEOpt(μ^*, η^*, E^w);

target architecture: 1) the power model used for the shared memory in the system already includes the power consumption of the bus connected to it; 2) for simplicity, we ignore the energy consumption caused by resource contention and communication delays. Consequently, the system active time for a specific workload scenario is simply assumed to be $argmax(U_k)$, which is subsequently used to calculate the static energy consumption. Note that the application of techniques such as dynamic power management (DPM) and dynamic voltage scaling (DVS) are beyond the scope of this paper.

The mechanism for searching the mapping space to find the energy optimized mapping is implemented in the function listed on line 2 of Algorithm 3. In each iteration, it greedily finds the mapping with minimal energy consumption for each active application in the workload scenario by just remapping a single B_i^j. Similar to Algorithm 2, Algorithm 3 first proposes a new mapping for each of the active applications, after which the mapping with minimal energy consumption among the proposed mappings will be used in the next optimization iteration. However, if the condition on line 5 of Algorithm 3 is satisfied, then the input mapping will be returned as the final optimization result.

Similar to the complexity of Algorithm 2, for each possible new mapping in Algorithm 3, the time consumed for computing the value of E_{ij} is $O(|PE||P| + |F||C|)$. The approximate time complexity of each iteration in Algorithm 3 is $O(|PE|^2|D|(|PE||P| + |F||C|))$ and the maximal iteration count is $|D|$. So, the overall time complexity of this algorithm is $O(|PE|^2|D|^2(|PE||P| + |F||C|))$.

IV. Experiments

A. Experimental Framework

To evaluate the efficiency of our EIM algorithm and the mappings found at run time by this algorithm, we deploy the open-source Sesame system-level MPSoC simulator [19]. To this end, we have extended this simulator with our run-time resource scheduling framework, as illustrated in Figure 2. Our extension includes the Scenario DataBase (SDB), a Run-time System Monitor (RSM) and a Run-time Resource Scheduler (RRS). The SDB is used to store the mappings for intra-application scenarios of each application as derived from design-time DSE. The RSM is in charge of detecting and identifying the active workload scenario. The RRS uses the EIM algorithm and the identified workload scenario by the RSM to generate a mapping for this scenario, as explained in the previous section.

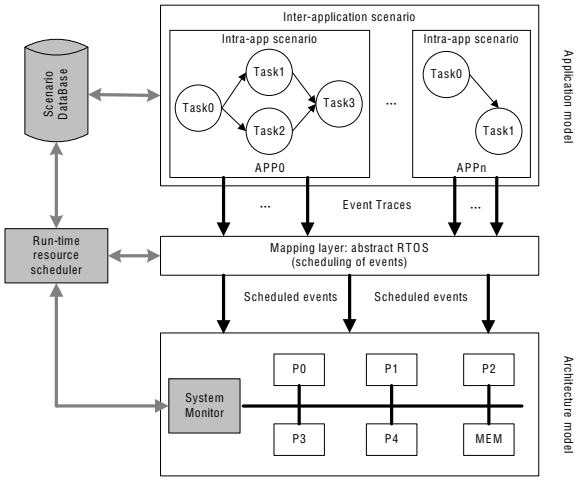


Figure 2: Extended Sesame framework.

B. Experimental Results

In this subsection, we present a number of experimental results in which we investigate various aspects of our EIM algorithm. More specifically, in each system execution mode, we compare the algorithm to three different run-time mapping algorithms using the optimization objective of the system mode. For the high performance mode, we compare our EIM algorithm to the following algorithms: Task Processor Affinity (TPA) which uses the affinity between tasks and processors to greedily determine a mapping without considering resource contention, and Output-Rate Balancing (ORB) [4] which aims at balancing the computation and communication load of each processor. For the energy saving mode, we compare our algorithm to TPA and Iterative Energy-Aware Task Mapping (IEATM) [22], [10]. Moreover, we also compare the run-time mapping results to the results of optimal mappings for each workload scenario. These optimal mappings have been statically determined by means of design-time DSE using a NSGA-II genetic algorithm.

For our experiments, we use three typical multi-media applications: a Motion-JPEG (MJPEG) encoder, an MP3 decoder, and a Sobel filter for edge detection in images which are denoted as A1, A2 and A3 respectively in Table II and Figures 3 and 5. The KPN of the MJPEG application contains 8 processes and 18 FIFO channels, Sobel contains 6 processes and 6 FIFO channels, and MP3 contains 27 processes and 52 FIFO channels. Moreover, MJPEG has 11 intra-application scenarios, MP3 has 3 intra-application scenarios, whereas Sobel only has 1 intra-application scenario. This results in a total of 95 different workload scenarios. At design time, we have determined the optimal mapping for each intra-application scenario in each application targeting the different optimization objectives as explained in Section III. That means that we need to store 45 optimal mappings in system memory (i.e., the scenario database).

With respect to the target architecture, we target a heterogeneous MPSoC containing 5 different processors with different computational and energy characteristics, connected

Table II: Studied application workload scenarios.

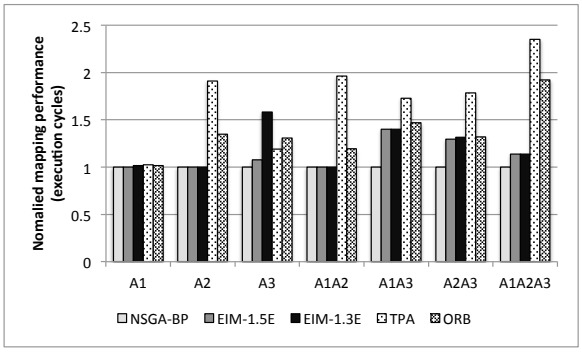
Inter-app scenario	Workload scenario
A1	mjpeg 7
A2	sobel 0
A3	mp3 2
A1A2	mjpeg 7, sobel 0
A1A3	mjpeg 7, mp3 2
A2A3	sobel 0, mp3 2
A1A2A3	mjpeg 7, sobel 0, mp3 2

to a shared bus and memory. The model also includes the required components for our run-time scheduling framework.

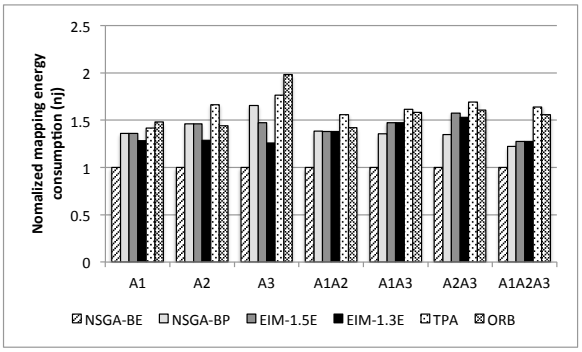
1) Performance Optimization Experiments: The experiments in this subsection concern the evaluation of our run-time mapping algorithm in high performance mode, considering different inter- and intra-application workload scenarios.

In the first experiment, we study the run-time mapping behaviour in the occurrence of different inter-application scenarios. To this end, we focus on the subset of workload scenarios that have the heaviest computational demands in each inter-application scenario. These workload scenarios are listed in Table II, where the first column specifies the encoded name (in terms of A1, A2 and A3) for each inter-application scenario and the second column specifies the intra-application scenarios (labeled by the integer following the application name) used to form the workload scenario. For the scaling factor α of the energy budget in our EIM algorithm (see equation 3) we use the values 1.5 and 1.3 in our experiments. For design-time DSE using the NSGA-II genetic algorithm, we have set the parameters for the population size, offspring size and generation count all to 256, which is large enough for obtaining a high-quality mapping for each workload scenario. In the initial population, we have added the mappings found by the run-time algorithms used in our experiment as the initial individuals in order to help the NSGA-II algorithm to find even better mappings.

The experimental results are shown in Figure 3. In Figure 3(a), we compare the performance of the mappings resulting from the EIM, TPA, and ORB algorithms as well as from NSGA-II-based design-time DSE. The energy consumption of these mappings is shown in Figure 3(b). In these two figures, the bars of NSGA-BP and NSGA-BE respectively represent the mappings with best performance and minimal energy consumption found by the NSGA-II-based design-time DSE. These are used as a baseline for comparison. From Figure 3(a), we can see that our EIM algorithm in most cases produces a better mapping for the tested workload scenarios than the TPA and ORB algorithms. For the workload scenarios in which only a single application is active (i.e., bars for A1, A2 and A3) our EIM algorithm directly uses the mapping from design-time DSE, which results in a mapping performance that is very close or even equivalent to the optimal mapping. However, although the mappings have similar performance, they could still have a different energy consumption behavior. In the case of our EIM algorithm, we use the energy budget in the search for an efficient mapping to limit the energy consumption of the resulting mapping. Consequently, and as shown in Figure 3(b), the EIM algorithm can yield mappings for single-application workload scenarios that are more energy efficient than the ones



(a) Performance of mappings from different algorithms



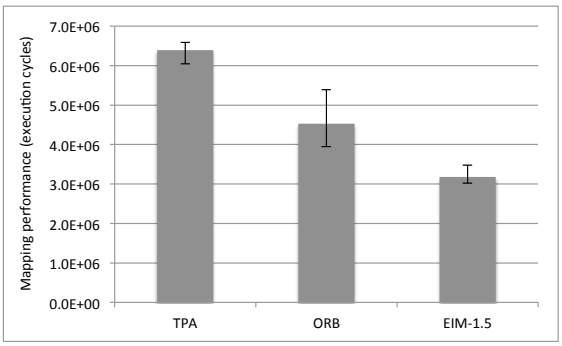
(b) Energy consumption of mappings from different algorithms

Figure 3: Algorithm comparison under high performance mode (inter-application scenarios).

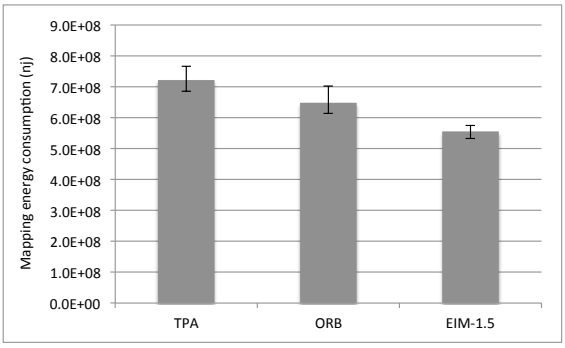
obtained by NSGA-BP.

In the workload scenarios with multiple simultaneously active applications, we can see that the EIM algorithm yields clear performance improvements compared to the other three run-time task mapping algorithms, especially in the case of workload scenario A1A2A3. By setting the parameter α of our EIM algorithm to different values, we can notice that in some workload scenarios, like A1, A3 and A2A3, the mapping performance with a higher energy budget is better than the one with a lower energy budget. However, in other workload scenarios, there is no such behavior. This can be explained by the fact that for the latter workload scenarios the energy budget is big enough for the algorithm with a lower energy budget to find a mapping that is as good as the one found by EIM with a higher energy budget. In Figure 3(b), we can see that even if we have an energy budget in our EIM algorithm, the actual energy consumption of the final mapping may still exceed the energy budget: like for EIM-1.5E in the A2A3 workload scenario and for EIM-1.3E in a few other workload scenarios. This is caused by estimation inaccuracies of the energy model used in our algorithm. Even if the estimated energy consumption of a new mapping is under the predefined energy budget, the actual resulting system energy consumption after the remapping has taken place may still not fully satisfy our desired energy budget.

In the second experiment, all the intra-application scenarios for one particular inter-application scenario, namely A1A2A3, are considered as the experimental workload. This means that there are 33 workload scenarios in total. We compare the



(a) Performance of mappings from different algorithms



(b) Energy consumption of mappings from different algorithms

Figure 4: Algorithm comparison under high performance mode (intra-application scenarios).

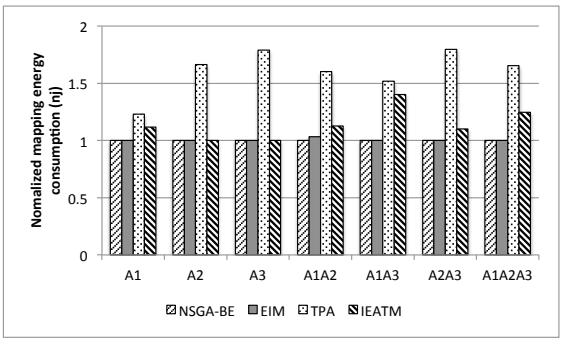


Figure 5: Algorithm comparison under energy saving mode (inter-application scenarios).

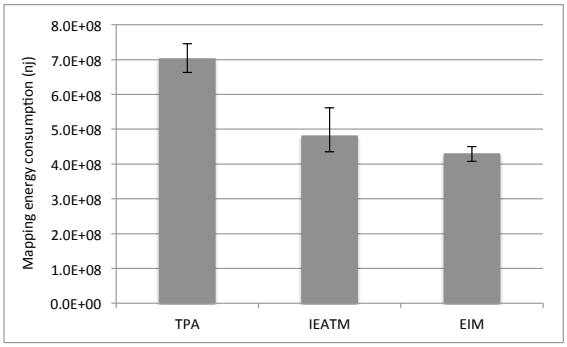


Figure 6: Algorithm comparison under energy saving mode (intra-application scenarios).

average performance and energy consumption of the optimized mappings as obtained by the different algorithms. The results are shown in Figure 4(a) and Figure 4(b) respectively. The error bars in the graphs show the variability of the results. From Figure 4(a), we can see that the mappings from our EIM algorithm with a scaling factor $\alpha = 1.5$ achieve the best average performance among the investigated four algorithms. Even the worst mapping performance among all the 33 workload scenarios of our EIM algorithm is still better than the best one in any of the other algorithms. Comparing the performance of each final mapping obtained by our EIM algorithm with the ones from TPA and ORB in all our tested 33 workload scenarios, we measure ranges of 56.3%-66.6% and 11.5%-42.3% of performance improvement respectively. Figure 4(b) shows the average energy consumption of the final mappings used in Figure 4(a). The results in this figure illustrate that the mappings from our EIM algorithm have the lowest average energy consumption. Considering the energy consumption of each final mapping, our EIM algorithm achieves, respectively, a 20.7%-29.6% and 6.1%-19.0% improvement for energy savings compared with TPA and ORB.

2) Energy Optimization Experiments: Considering the energy saving system mode, we also investigate our run-time mapping algorithm considering different inter- and intra-application workload scenarios. The results of the different mapping algorithms when the primary objective is energy optimization and when using the subset of inter-application scenarios from Table II are shown in Figure 5. From this experiment, we can see that our EIM algorithm can efficiently produce near optimal (in terms of energy consumption) mappings. Comparing the four run-time mapping algorithms, our EIM algorithm overall shows the best results, with the IEATM algorithm ranked second. In the single-application workload scenarios, the EIM simply use the mapping optimized at design time. For the multi-application scenarios, we can notice that our EIM algorithm clearly improves on the other algorithms, finding mappings that are almost as good as the ones obtained by NSGA-BE.

Figure 6 shows the results of average energy consumption of mappings optimized for energy consumption by the different mapping algorithms when considering all 33 workload scenarios of inter-application scenario A1A2A3. The results in this figure illustrate that the EIM algorithm performs much better than the other algorithms. For each single workload scenario, the EIM algorithm achieves energy improvements of in between 50.5%-54.2% and 5.5%-20.2% as compared with TPA and IEATM respectively.

3) Run-time Cost: Here, we would like to give an intuition of the run-time cost of our approach in terms of the number of tasks that need to be migrated and the computation cost of the algorithm. In this experiment, the intra-application scenarios in inter-application scenario A1A2A3 with only application MP3 changing its execution mode will be considered as the targeting scenarios. This means that the execution mode of each application is: MJPEG (7), Sobel (0) and MP3 (0/1/2). These three scenarios will be executed in sequence to find out

Table III: Run-time cost of different algorithms for performance optimization.

Algorithms	Total task migration number	Normalized total algorithm computation time
TPA	12	1.0
ORB	36	22.5
EIM1.5	25	651.2

Table IV: Run-time cost of different algorithms for energy optimization.

Algorithms	Total task migration number	Normalized total algorithm computation time
TPA	8	1.0
IEATM	9	64.0
EIM	16	62.2

the total task migration number and total algorithm computation time by applying different approach. The total number of tasks in each scenario is 41. Table III shows the cost of different approaches for mapping performance optimization, where the total algorithm computation time of each approach is normalized to the one of TPA. From the results, we can see that the baseline approach TPA has the minimal run-time cost in both migration cost and algorithm computation time. Our proposed approach EIM has the highest algorithmic computation time. This is mainly caused by the large diversity ($|D|$) of the pre-optimized mappings used in this experiment as described in the algorithm complexity analysis in Section 3.1. However, we believe that the computation cost of our EIM algorithm is still acceptable as it just need a few milliseconds (on an CPU with 2.7GHZ) to optimize the mapping for each workload scenario in our test case. Here, we would like to note that we have not yet performed any effort to optimize our EIM algorithm to reduce its computational cost. The run-time cost of the approaches for mapping energy optimization is listed in Table IV, where our approach shows the highest total task migration cost while the algorithmic computation time is relatively low as the diversity of the pre-optimized mappings is small.

Compared to the algorithmic computation cost, the run time task migration overhead typically is more substantial for MPSoC systems. From both experimental results, we can see that the task migration cost of our EIM is relatively heavy. For this reason we make the assumption that each workload scenario will execute for a long enough time so that the system is able to benefit from our EIM algorithm. Further research is needed to exactly determine at which switching granularity of workload scenarios our algorithm could benefit from remapping.

V. Related Research

In recent years, much research has been performed in the area of run-time task mapping for embedded systems. Recently, Singh et al. [24] gave a nice survey of current and emerging trends for the task mapping problem on multi/many-core systems. In the context of performance optimization, the authors of [5] propose a run-time mapping strategy that incorporates user behavior information in the resource allocation process. An agent based distributed application mapping approach for large MPSoCs is presented in [1]. The work of [8] proposes a run-time spatial mapping technique to map streaming applications onto MPSoCs. In [3], dynamic task allocation strategies based on bin-packing algorithms for soft real-time applications are presented. A runtime task allocator is presented in [9] that uses an adaptive task allocation algorithm and adaptive clustering approach for efficient reduction of the communication load. Considering the hybrid task mapping approaches, Mariani et al. [15] proposed a run-time management framework in which Pareto-fronts with system configuration points for different applications are determined during design-time DSE, after which heuristics are used to dynamically select a proper system configuration at run time. In [28], a fast and light-weight priority based heuristic is used to select near-optimal configurations explored at design time for the active applications according to the available platform resources. [23] proposes DSE strategies that perform exploration in view of optimizing throughput and energy consumption by considering a generic platform. The design points derived from the DSE will be selected efficiently at run time. Compared with these algorithms, our performance optimization algorithm in EIM takes an application scenario-based approach, and takes computational and communication behavior embodied in design-time optimized mappings into account when making run-time mapping decisions. Recently, Schor et al. [21] and Quan et al. [20] also proposed scenario-based run-time mapping approaches in which mappings derived from design-time DSE are stored for run-time mapping decisions. However, [21] does not address the reduction of mapping storage (all workload scenarios are stored) and does not dynamically optimize the mappings at run time. In [20], an approach is proposed in which mappings for inter-application scenarios are stored and used as a basis for run-time mapping decisions, after which an run-time algorithm aims at gradually further optimizing these mappings. A major drawback of this method is that it needs to search for optimal mappings for *inter-application* scenarios at design time, which implies that it should already been known at design time which applications can execute on the target platform. For example, extending the system with a new application would require to redo the entire design-time DSE for all inter-application scenarios. In our approach, this problem is avoided by taking intra-applications as the basis for doing design-time DSE (i.e., performing DSE on applications in isolation).

With regard to energy optimization, dynamic mapping methodologies have also been studied. These studies basically split into two directions. Some tackle the problem by defining efficient heuristics to assign new arriving tasks onto processing units at run time, e.g., [16]. Others analyze applications offline and compute schedules and allocations that are then stored on the system, e.g., [2], [17], [27], [22]. In [22], Schranzhofer et al. proposed static and dynamic task mapping approaches for probabilistic applications based on static and dynamic power components. Statically pre-computed template mappings for each execution probability are stored on the system and applied at run time, allowing the system to adapt to changing environment conditions. Based on this work, [10] presents an extension that considers only the static mapping and takes into account the communication and reconfiguration energy component. However, comparing these two efforts to our approach,

we capture dynamism of applications running on the system using the concept of realistic workload scenarios instead of execution probabilities. Moreover, we use pre-computed, per-application mappings as a basis for optimizing the mapping for a new workload scenario rather than using template mappings.

VI. Conclusion

We have proposed a run-time mapping algorithm, called EIM, for MPSoC-based embedded systems to improve their performance and energy consumption by capturing the dynamism of the application workloads executing on the system. This algorithm is based on the idea of application scenarios and consists of a design-time and run-time phase. The design-time phase produces mappings for intra-application scenarios targeting different optimization objectives after which the run-time phase aims to continuously monitor the changes in workload scenarios on the underlying system and trying to perform iterative mapping optimization to improve the system performance and/or energy consumption. In various experiments, we have evaluated our algorithm and compared it with other run-time mapping algorithms. The results clearly confirm the effectiveness of our algorithm.

References

[1] M. A. Al Faruque, R. Krist, and J. Henkel. Adam: run-time agent-based distributed application mapping for on-chip communication. In *Proc. of DAC'08*, pages 760–765, 2008.

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

[3] E. W. Brião, D. Barcelos, and F. R. Wagner. Dynamic task allocation strategies in mpsoc for soft real-time applications. In *Proc. of DATE'08*, pages 1386–1389, 2008.

[4] J. Castrillon, R. Leupers, and G. Ascheid. Maps: Mapping concurrent dataflow applications to heterogeneous mpsocs. *IEEE Trans.on Industrial Informatics*, PP(99):1, 2011.

[5] C.-L. Chou and R. Marculescu. User-aware dynamic task allocation in networks-on-chip. In *Proc. of DATE'08*, pages 1232–1237, 2008.

[6] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere. System-scenario-based design of dynamic embedded systems. *ACM Trans. Design Autom. Electr. Syst.*, 14(1), 2009.

[7] M. Gries. Methods for evaluating and covering the design space during early design development. *Integr. VLSI J.*, 38(2):131–183, Dec. 2004.

[8] P. K. Hölzenspies, J. L. Hurink, J. Kuper, and G. J. Smit. Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (mpsoc). In *Proc. of DATE'08*, pages 212–217, March 2008.

[9] J. Huang, A. Raabe, C. Buckl, and A. Knoll. A workflow for runtime adaptive task allocation on heterogeneous mpsocs. In *Proc. of DATE'11*, pages 1119–1134, 2011.

[10] A. Hussien, A. Eltawil, R. Amin, and J. Martin. Energy aware task mapping algorithm for heterogeneous mpsoc based architectures. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 449–450, 2011.

[11] Z. J. Jia, A. Pimentel, M. Thompson, T. Bautista, and A. Nunez. Nasa: A generic infrastructure for system-level mp-soc design space exploration. In *Embedded Systems for Real-Time Multimedia (ESTIMedia), 2010 8th IEEE Workshop on*, pages 41–50, 2010.

[12] G. Kahn. The semantics of a simple language for parallel programming. In *Information processing*, pages 471–475. North Holland, Amsterdam, Aug 1974.

[13] J.-K. Kim, S. Shivle, H. Siegel, A. Maciejewski, T. Braun, M. Schneider, S. Tideman, R. Chitta, R. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. Yellampalli. Dynamic mapping in a heterogeneous environment with tasks having priorities and multiple deadlines. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, pages 15 pp.–, 2003.

[14] R. Kumar, D. M. Tullsen, P. Ranganathan, N. P. Jouppi, and K. I. Farkas. Single-isa heterogeneous multi-core architectures for multithreaded workload performance. In *Proceedings of the 31st annual international symposium on Computer architecture*, ISCA '04, pages 64–75, Washington, DC, USA, 2004. IEEE Computer Society.

[15] G. Mariani, P. Avasare, G. Vanmeerbeeck, C. Ykman-Couvreur, G. Palermo, C. Silvano, and V. Zaccaria. An industrial design space exploration framework for supporting run-time resource management on multi-core systems. In *Proc. of DATE'10*, pages 196 –201, march 2010.

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

[17] S. Murali, M. Coenen, A. Radulescu, K. Goossens, and G. De Micheli. Mapping and configuration methods for multi-use-case networks on chips. In *Design Automation, 2006. Asia and South Pacific Conference on*, pages 146–151, 2006.

[18] J. M. Paul, D. E. Thomas, and A. Bobrek. Scenario-oriented design for single-chip heterogeneous multiprocessors. *IEEE Trans. VLSI Syst.*, 14(8):868–880, 2006.

[19] A. D. Pimentel, C. Erbas, and S. Polstra. A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Trans. Computers*, 55(2):99–112, 2006.

[20] W. Quan and A. D. Pimentel. A scenario-based run-time task mapping algorithm for mpsocs. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 131:1–131:6, New York, NY, USA, 2013. ACM.

[21] L. Schor, I. Bacivarov, D. Rai, H. Yang, S.-H. Kang, and L. Thiele. Scenario-based design flow for mapping streaming applications onto on-chip many-core systems. In *Proc. of CASES'12*, pages 71–80, 2012.

[22] A. Schranzhofer, J.-J. Chen, and L. Thiele. Dynamic power-aware mapping of applications onto heterogeneous mpsoc platforms. *Industrial Informatics, IEEE Transactions on*, 6(4):692–707, 2010.

[23] A. K. Singh, A. Kumar, and T. Srikanthan. Accelerating throughput-aware runtime mapping for heterogeneous mpsocs. *ACM Trans. Des. Autom. Electron. Syst.*, 18(1):9:1–9:29, Jan. 2013.

[24] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, DAC '13, pages 1:1–1:10, New York, NY, USA, 2013. ACM.

[25] W. Sun and T. Sugawara. Heuristics and evaluations of energy-aware task mapping on heterogeneous multiprocessors. In *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*, pages 599–607, 2011.

[26] P. van Stralen and A. D. Pimentel. Scenario-based design space exploration of mpsocs. In *Proc. of IEEE ICCD'10*, pages 305–312, October 2010.

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

[28] C. Ykman-Couvreur, P. Avasare, G. Mariani, G. Palermo, C. Silvano, and V. Zaccaria. Linking run-time resource management of embedded multi-core platforms with automated design-time exploration. *Computers Digital Techniques, IET*, 5(2):123–135, 2011.