Energy-Aware Task Allocation for Network-on-Chip Based Heterogeneous Multiprocessor Systems

Jia Huang, Christian Buckl, Andreas Raabe
fortiss GmbH
Guerickestr. 25, 80805 Munich, Germany
{*huang,raabe,buckl*}*@fortiss.org*

Alois Knoll
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
knoll@in.tum.de

***Abstract*—Energy-efficiency is becoming one of the most critical issues in embedded system design. In Network-on-Chip (NoC) based heterogeneous Multiprocessor Systems, the energy consumption is influenced dramatically by task allocation schemes. Although various approaches are proposed to allocate tasks in an energy-efficient way, existing work does not well explore the tradeoff between the two major power consumers, namely the processors and network links, resulting in suboptimal mappings from a system point of view. In this paper, we first extend the existing Integer Linear Programming (ILP) formulation to take both processing and communication energy into account. Thereafter, we propose a Simulated Annealing with Timing Adjustment (SA-TA) heuristic to accelerate the optimization process. While the SA-TA algorithm achieves performance very close to the global optimum, significant improvement in computation speed is observed.**

***Keywords*-Processor scheduling; Resource management**

I. Introduction

Multiprocessor System-on-Chip (MPSoC) platforms provide both high performance and high flexibility to allow efficient execution of a wide range of applications. Hence, MPSoCs are believed to be the major solution of future embedded systems [1], [2]. A major step in the MPSoC based development process is mapping and scheduling of applications onto the platform. Since the on-chip processing units typically have very distinct speed/power characteristics, it is a challenging task to find the optimal task mapping. Moreover, various constraints need to be considered, such as the hardware resource limitation and timing constraints.

Energy efficiency is one of the most critical design issues in embedded systems. For this reason, energy-aware task allocation becomes an active research topic [3], [4], [5], [6]. Existing studies that tackle this problem can be categorized into two main branches: software partitioning, which assigns each task to a particular processor statically, and dynamic scheduling, which relies on a global scheduler to select an appropriate processor at runtime to execute a ready task. Due to the simplicity in design and implementation, the partitioning approach is more common [7]. In this paper, we focus on the partitioning problem.

The dynamic energy consumption of a processor increases quadratically with its operating frequency. Hence, using a multiprocessor system, in which the execution can be distributed to a set of slower and low energy consuming cores, can lead to significant energy reduction. Nowadays, multiprocessor chips with more than hundred cores already exist in the market. We emphasize that, in such systems, the on-chip network consumes a substantial portion of the power budget. For example, in the Alpha 21364 processor, on-chip routers and links consume 20% of the system power [8]. Another example is the MIT Raw, in which communication power takes up to 36% of total power. Hence, to pursue the system-level optimal solution, it is essential to consider the task processing power and the communication power together. However, the optimal mapping that minimizes one part is often sub-optimal for the other part. A typical situation is that the designer must evaluate the tradeoff between processing and communication power. Moreover, the hardware resource limitations and Quality of Service (QoS) requirements of applications can not be ignored.

The goal of this study is to develop an algorithm for efficient computation of the system-wide energy-optimal task allocation for heterogeneous MPSoC systems. The main contribution of this paper is an ILP formulation extended from existing work and a novel Simulated Annealing with Timing Adjustment (*SA-TA*) heuristic. Experimental results show that the *SA-TA* heuristic achieves performance very close to global optimum and in the mean time much higher execution speed than ILP-based solutions.

The rest of this paper is organized as follows. Section II introduces the task model and architecture model used in this paper. The motivation of our work is explained in section III using an example. Section IV provides an ILP formulation of the problem and section V presents the details of the SA-TA heuristic. Experimental results are presented in section VI. A comparison of this work with related work is discussed in section VII and section VIII concludes this paper.

II. Task Model and Architecture Model

This work considers streaming applications represented using Task Graphs. A TG is a Directed Acyclic Graph (DAG), whose vertices V represent tasks to be executed and edges E capture data dependencies between tasks. Each

vertex v_i is annotated with three vectors: E_i contains the energy consumption of task v_i on each feasible processor, C_i contains the respective execution time and M_i contains the respective memory consumption. The execution time can be worst case execution time for hard real-time tasks or average execution time for soft real-time tasks. Each vertex may have a relative deadline d, which specifies the latest tolerable finishing time with respect to the start time of TG. For an edge $e_{i,j} \in E$ between source task t_i and target t_j, the size of the data (called token) to be transfered is annotated by $W_{i,j}$. The communication cost is assumed to be zero if both t_i and t_j are assigned to the same processor.

The hardware platforms we considered are NoC-based heterogeneous MPSoCs. Each node p_i in the model consists of a processor, a memory component and a communication facility, e.g. a router. Figure 1 shows a 2D-mesh NoC example with 3x3 nodes of two different types. The location of each node is determined by a pair of coordinates(x_i, y_i) in terms of *hops*, which is the distance between two adjacent nodes. A function Ω is provided in the model to compute the *distance* between two nodes. In case of 2D-mesh, the distance between p_i and p_j is the Manhattan distance according to the routing path:

$$D_{i,j} = abs(x_i - x_j) + abs(y_i - y_j)$$

While any topology can be involved in this architecture model (as long as Ω is available), we concentrate only on 2D-mesh structure in the paper, since it is the most common setup of MPSoC Platforms.

Minimal-path routing is assumed in this paper, which indicates that the distance reflects the effort of communication. A linear model used in [9] is adopted to estimate the communication cost in the mesh network. Let ϵ and τ denote the cost in terms of energy and time respectively for transferring a token with unit size between two nodes with unit distance (1 hop). Then the energy and time consumption to convey a token with W units between two nodes p_i and p_j is estimated by $\epsilon W D_{i,j}$ and $\tau W D_{i,j}$, respectively. As pointed out in [9], this model works well for today's MPSoCs with high-bandwidth on-chip networks.

III. Motivating Example

This section discusses our motivation using an example. We emphasize that traditional techniques, which either consider only one of the two major power consumers or consider both but ignore the tradeoff between them, can not guarantee global optimality.

Assume our job is to allocate a 4-node linear task graph ($t_1 \ldots t_4$ in figure 2b) to a 3x3 2D mesh MPSoC platform consisting of 5 types of processors (figure 2a). The indices of processors in figure 2a indicate the type and ID, e.g. $P_{1,x}$ labels the xth processor of type 1. The processing costs for executing each individual task on each type of processors are listed in 2c. As we can see, each task clearly has a

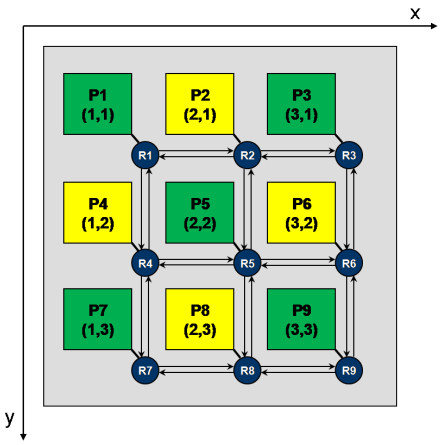


Figure 1. An example 2D-mesh NoC

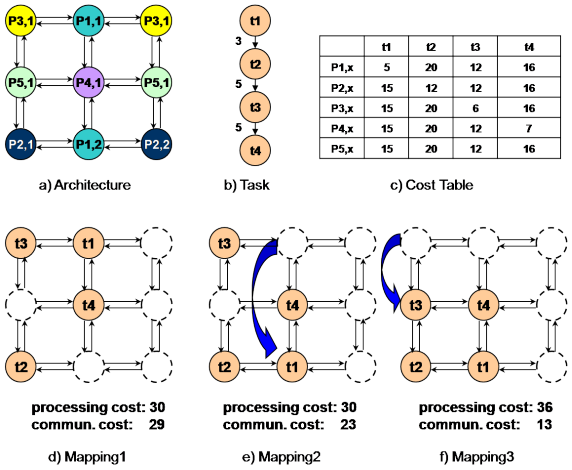
preferred processor, e.g. it is much more efficient to execute t_1 on $P_{1.x}$. The communication is assumed to have cost 1 per unit payload per hop.

Traditional approaches that minimize the processing power only may produce a task mapping as in figure 2d). As we can see, each task is mapped to its preferred type of processor. More sophisticated approaches try to reduce the communication cost without influencing the processing cost, i.e. explore different spatial allocation while fixing the task-to-processor-type mapping. For example, this strategy is used in [10]. Figure 2e shows the improved mapping (communication cost reduces from 29 to 23). However, if we further examine the mapping in 2e, it can be found that the *desirability* of mapping t_3 to its preferred processor $P_{3,x}$ is not very high, since the difference between execution cost of t_3 on $P_{3,x}$ and that on $P_{5,x}$ is only 6. It the same time, mapping t_3 to $P_{3,1}$ generates significant communication cost. Hence, it is more efficient to assign t_3 to $P_{5,1}$ such that 10 units power saving from communication is achieved with the tradeoff of 6 units lost from processing (figure 2f).

The lesson from this example is that it is particularly important to take the tradeoff between the processing and communication power into consideration while computing task allocation. However, due to the high complexity of the problem, most existing work such as [10] uses step-wise approaches and does not well explore this tradeoff. As to the motivating example, only mapping 2e can be found, which is however suboptimal from a system point of view. In contrast, the proposed approach always considers the system-level power consumption.

IV. ILP Formalism

In [6], the authors present an ILP formulation of the task allocation problem. However, only the task processing energy is considered. This section presents an extended version that takes communication power into account.



	t1	t2	t3	t4
P1,x	5	20	12	16
P2,x	15	12	12	16
P3,x	15	20	6	16
P4,x	15	20	12	7
P5,x	15	20	12	16

Figure 2. An Motivating Example

Let P denote the set of processors and T the set of tasks to be allocated. For each task $t \in T$, the binary variable $x_{i,m}$ is set to 1 iff t_i is assigned to processor p_m and 0 otherwise. The binary variable $p_{i,j}$ denotes the execution precedence between t_i and t_j. $p_{i,j}$ is 1 iff t_i executes before t_j and 0 otherwise. For each tasks t_i, two integers s_i and f_i are used to denote the starting and finishing time (relative to the start time of current iteration of the TG).

ILP Constraints. The ILP problem is incorporated with following constraints:

- Each task is assigned to exactly one processor, i.e. $\sum_{p_m \in P} x_{i,m} = 1, \forall t_i \in T$
- No more memory is allocated than available.
- Task dependencies. Let $src(e)$ and $tgt(e)$ denote respectively the source and destination of an edge $e \in E$ in task graph. Then the start time of $tgt(e)$ must be no earlier than the finishing time of $src(e)$ plus the token transfer latency. For tasks with data dependencies, the precedence variables are automatically determined, i.e. $p_{src(e),tgt(e)} = 1, p_{tgt(e),src(e)} = 0, \forall e \in E$
- If two tasks t_i and t_j are mapped to the same processor, they must be executed sequentially, i.e. either $p_{i,j}$ or $p_{j,i}$ is one and the execution time must not overlap.
- If task t_i has a deadline d_i, the finish time of t_i should be no later than the deadline.
- Intrinsic constraints: $p_{i,j}$ and $p_{j,i}$ can not be 1 at the same time. $p_{i,j} = p_{j,i} = 0$ means that t_i and t_j are executed simultaneously on two different processors.
- Prerequisites. Some tasks may have fixed mappings (e.g. to dedicated accelerators) or may be impossible to be executed on some processors.

Objective Function. As pointed out before, our goal is to minimize the system-level energy consumption. Hence the objective function is defined as:

$$Minimize: \ E_p + E_{com}$$

where E_p is the processing energy and E_{com} is the communication energy. Let $E_{i,m}$ denote the energy needed to execute task t_i on processor p_m, W_e be the token size in edge e and $D_{m,n}$ be the distance between processor p_m and p_n, the processing and communication energy can be computed as follows:

$$E_p = \sum_{t_i \in T} \sum_{p_m \in P} x_{i,m} \cdot E_{i,m} \quad (1)$$

$$E_{com} = \sum_{e \in E} \sum_{p_m \in P} \sum_{p_n \in P} \epsilon W_e D_{m,n} x_{src(e),m} \cdot x_{tgt(e),n} \quad (2)$$

where ϵ is the amount of energy needed to transfer a token with unit size along a path with unit distance.

Incorporating DVS-enabled processors. Dynamic Voltage Scaling (DVS) is a popular technique to reduce the task processing power. In our framework, different DVS modes of a DVS-enabled processor can also be incorporated. Each mode of a processor is defined as a separate processor type. In this case, multiple processors may reside in the same location in the architecture graph. Additional constraints are introduced to guarantee that different modes of the same processor are not enabled simultaneously. Assume that a processor has L DVS modes, the following constraints are introduced, saying that two tasks, which are mapped to different modes of the same processor, can not be executed simultaneously:

$$p_{i,j} + p_{j,i} = 1, \ if \ x_{i,m} = x_{j,n} = 1, \forall m, n \in L, t_i, t_j \in T$$

V. Heuristic Algorithm

Since the computational complexity of ILP grows rapidly as the problem becomes more complex, an efficient heuristic algorithm is necessary. Because of the high complexity and huge search space in the problem, we target on providing a scalable algorithm, where the user can configure the algorithm base to their need. The proposed approach is an extended Simulated Annealing (SA)[11] algorithm called *SA-TA* (SA with Timing Adjustment (TA)), the pseudo code of which is provided in algorithm 1. A mapping is denoted in algorithm 1 using $\pi : T \rightarrow P$.

Instead of starting the SA optimization from a random mapping, we first computes a **baseline mapping**, which is a rather intuitive yet very good mapping serving as the starting point of the SA optimization process. The hardware architecture is also taken into consideration right from beginning. The concept of *desirability* is adopted from [10], which is defined as the difference between the cheapest assignment (in terms of processing power only) for a specific task t_i and the second cheapest assignment for t_i. The idea of desirability is to consider the task with highest gain first, in order to prevent its preferred processor from being occupied by other tasks. Before computing the mapping, all tasks in the task graph are sorted in decreasing order of desirability. For the motivating example, the mapping

sequence is t_1, t_4, t_2, t_3. The task with highest desirability $t^\dagger$ is first assigned to a processor $p^\dagger$ with lowest processing energy (the preferred type). If multiple processors of the same type exist, the one that minimizes the communication cost of $t^\dagger$ is chosen, assuming that tasks communicating with $t^\dagger$ are mapped to their preferred type nearest to $p^\dagger$. Consider the motivating example, task t_1 is assigned to processor $p_{1,2}$ instead of $p_{1,1}$, since $p_{1,2}$ is closer to $p_{2,x}$, which is preferred by task t_2. The next task to be assigned (denoted by t_i) is allocated to the processor that has enough resources (e.g. memory) and minimizes the total energy considering the communication with tasks that are mapped previously. Assume Q_i is the set of already allocated tasks which have dependency with t_i, the processor minimizes the following term is chosen to execute t_i:

$$E = Ep + Ecom = E_{i,\pi(i)} + \sum_{j \in Q_i} \epsilon W_{\pi(i),\pi(j)} D_{\pi(i),\pi(j)} \quad (3)$$

This procedure is done until all tasks are allocated. For the motivating example, the optimal solution shown in figure 2f is directly found in baseline mapping.

Algorithm 1 *SA-TA* Heuristic Algorithm

```
//input: task graph TG, architecture graph AG
procedure TASKALLOCATION(TG, AG)
    step ← 0, T ← T_0
    π = baselineMapping()
    π_opt ← π, E_opt = computeEnergy(π)
    while i < N do
        π' = neighbor(π)
        E = computeEnergy(π)
        E' = computeEnergy(π')
        if E' < E then
            π ← π', E ← E'
            π_ta = timingAdjustment(π')
            E_ta = computeEnergy(π_ta)
            if isFeasible(π_ta) and E_ta < E_opt then
                π_opt ← π_ta, E_opt ← E_ta
            end if
        else
            if P(E, E', T) > random() then
                π ← π', E ← E'
            end if
        end if
        T ← λT, i ← i + 1
    end while
    return: π_opt, E_opt
end procedure
```

Starting from the baseline mapping, in each optimization iteration of *SA-TA*, a neighbor solution π' (a solution is in our case a task-to-processor mapping) of the current solution π is generated by the *neighbor()* function. In our algorithm, the *neighbor()* function removes the mapping of a randomly selected task from π and re-map it to a random processor. Then, the energy consumption of the newly generated mapping (E') and that of the original mapping (E) is computed. An *acceptance probability function* $P(E, E', T)$ is used to determine if the transition from π to π' is *accepted*, where T is a global parameter called *temperature*. The P function we used is:

$$P(E, E', T) = \begin{cases} 1 & \text{if } E' < E \\ e^{(E-E')/T} & \text{otherwise} \end{cases}$$

When the new mapping is more energy-efficient ($E' < E$), it is always accepted. When the new mapping is worse than the current mapping, probability still exists that it is accepted. This feature is essential for SA algorithm to prevent it from being stuck at a local optimum. This optimization process continues until it reaches the total number of iterations N configured by the user. The temperature T is decreased after each iteration by multiplying itself with a *cooling factor* λ ($\lambda < 1$), which is computed based on N:

$$\lambda = (T_n/T_0)^{(1/N)}$$

where T_0 is a very large and T_n is a very small number compared with E and E'. The temperature is initialized to T_0 and will reduce to T_n as the optimization completes. We extend the SA algorithm by introducing a timing adjustment phase, whose aim is to fine-tune the timing of an accepted mapping π' to meet the timing constraints. TA keeps the mapping unmodified if π' meets the deadline already. Otherwise, it examines mappings near π' to find a new solution that can improve the timing. The details of TA will be introduced shortly. The adjusted mapping π_{ta} is then checked for feasibility (i.e. if all timing/system constraints are met) and the feasible mappings are compared to find the best mapping so far. Note that TA has no impact on the searching path of the SA algorithm. The output of the entire algorithm is the global best mapping with the corresponding task execution schedule.

Timing Adjustment. After explaining the major steps in the heuristic, we now present the details of TA phase. To obtain the timing properties, we use a level-based list scheduling algorithm to schedule the task graph. Any other schedulers (e.g. the one used in [5]) can also be adopted. As mentioned before, the goal of timing adjustment is to explore the search space near an accepted mapping π' to find a feasible mapping π_{ta} that minimizes $E(\pi_{ta}) - E(\pi')$. This step is done as follows. Iterating over each of the tasks, we remove the assignment of a task t_i and check if a speedup can be obtained by re-mapping t_i to another processor. If the current mapping is already the fastest, we proceed with the next task. Otherwise, the re-mapping of t_i with the highest efficiency is remembered (but is not adopted directly). The efficiency η is defined as the ratio of the speedup gain and

the energy lost.

$$\eta = \frac{C_{i,\pi'(i)} - C_{i,\pi_{ta}(i)}}{E(\pi_{ta}) - E(\pi')}$$

Where C denotes the execution times. This procedure is done for all tasks. Then the re-mapping with the highest efficiency among all tasks is applied. Note that the efficiency defined above is possibly a negative quantity, which means we find a new mapping that achieves both speedup and energy reduction. Hence, negative efficiency is considered 'higher' than positive efficiency. This procedure continues until the deadline is met or no improvement can be found for all tasks. In the former case, the final mapping is the output of timing adjustment, and in the later case, the problem is reported as unfeasible. In extreme cases, timing adjustment phase can be applied directly upon baseline mapping to obtain fast results.

VI. Experiments

To evaluate the performance of our approach, we use *tgff* [12] to generate a set of synthetic task graphs with 20 ± 10 nodes (tasks). We also use a real-work task graph from automatic target recognition taken from [9]. The execution cycles of each task on each feasible processor are generated randomly using *tgff*. Execution time and energy consumption can then be calculated according to the frequency and power of the corresponding processor. For communication cost, the token sizes are randomly generated, and λ is defined in such a way that the total communication energy takes approximately 20% of system energy [8], [9]. Here the total communication cost is estimated by assuming the distance between each pair of communicating tasks to be the average distance in the architecture graph. The hardware model is the same as used in the motivating example, i.e. a 3x3 mesh MPSoC with 5 types of processors.

We use 10 task graphs in total and compare the following three cases: *ILP1* is the ILP formulation without considering communication energy, i.e. the objective function is minimizing E_p; *ILP2* is the ILP formulation that minimizes system energy $E_p + E_{com}$; the *Heuristic* is a simplified version of SA-TA that applies timing adjustment directly upon baseline mapping, i.e. the SA based optimization is completely bypassed. Figure 3 summarizes the experimental results. As can be seen, for all task graphs, the processing energy in *ILP2* is higher than in *ILP1*. Nevertheless, by sacrificing processing energy, a higher amount of communication energy is saved, resulting in a drop in system-wide energy consumption. Averaging over 10 TGs, 15% energy saving is achieved. The simplified heuristic algorithm also outperforms *ILP1*. The reason is that the communication energy is considered and the $E_p - E_{com}$ tradeoff is partially explored (recall that the tasks are iteratively mapped based on partial mapping using equation 3). On average, 11% energy is saved compared to *ILP1*.

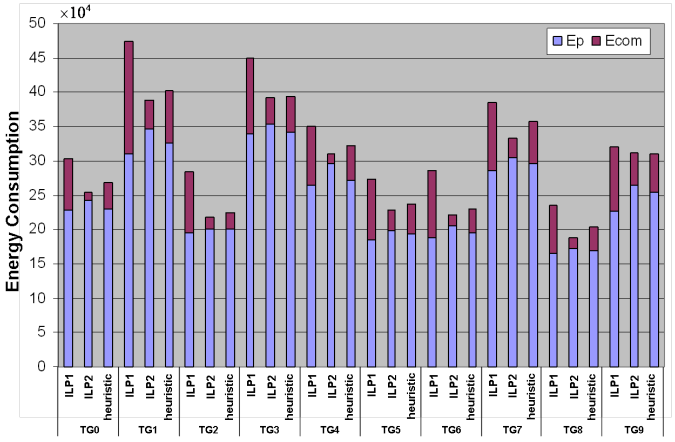


Figure 3. Experimental Results for Evaluating Tradeoff between E_p and E_{com}

In the next step, we focus on evaluation of the heuristic algorithm. As pointed out in section III, existing work that uses a *step-wise* approach ignoring the $E_p - E_{com}$ tradeoff may lead to sub-optimal solutions. For comparison, we implement a special step-wise version of the heuristic (denoted by *Algo A*), in which the tasks are first assigned to their preferred types of processors and the SA algorithm afterwards only search for better spatial alternatives. *Algo B* is the proposed SA-TA heuristic with a random mapping as starting point. *Algo C* is SA-TA with baseline mapping as the starting point.

For each task graph, the heuristic is executed 1000 times, from which the average result is taken. The energy consumption is normalized to the global optimum of respective task graph obtained from *ILP2* (denoted by E_{opt}). Figure 4 summarizes the results averaging over 10 task graphs, with *x* axis showing the number of optimization iterations and the *y* axis showing the normalized energy consumption. Comparing *Algo A* and *Algo B*, we observe that the unified approach outperforms the step-wise approach in almost all cases. The only case in which the step-wise approach shows a better performance is when the number of optimization steps are extremely small (100 steps). However, this use case is rare since the SA-based optimization is not expected to perform well due to limited iterations. The results also shows clearly the benefit of baseline mapping. For example, with a 100-iterations SA, an average result of $110\%E_{opt}$ is obtained from *Algo C* and the result is only $120\%E_{opt}$ for *Algo B*. As the refinement steps increase, both *Algo B* and *Algo C* show performance close to the global optimum (e.g. $102\%E_{opt}$ is achieved with 1000 iterations), whereas the performance of *Algo A* is far from that (only $111\%E_{opt}$).

The top-most curve in figure 4 (labeled by *Far Neighbor*) illustrates the results when two nodes in the task graph are re-mapped randomly in the *neighbor()* function. As it can be seen, the performance degrades dramatically compared with

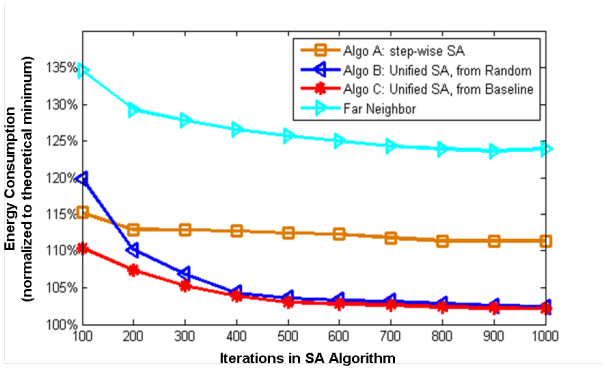


Figure 4. Performance Evaluation of Heuristic Algorithm without Timing Constraints

the the other cases, in which only one task is re-mapped. This effect complies with the general rule of SA algorithm that the *neighbor()* function should provide a new solution relatively close to the original solution [11].

In the experiments mentioned before, no timing constraint is introduced. Now we add deadline constraints to the sink tasks so that the complete algorithm can be evaluated. Besides the average energy consumption, another important metric is the *success rate* of the algorithm, which is the rate that at least one valid solution that meets timing constraints is found at the end of the algorithm. During the experiments, we found out that the baseline mapping is already a valid solution in most cases, resulting in a constant 100% success rate. Since the effectiveness of baseline mapping is already demonstrated in figure 4, we use *algo B* (SA starting from random mapping) in this experiment, so that we can focus on evaluating effectiveness of SA-TA. The **solid** curves show the energy consumption normalized to the global optimum under respective timing constraints. The **dashed** curves show the success rate in percentage. We compare the performance of two configurations: a *Pure SA* configuration, in which the accepted solutions that violate the timing constraints are abandoned, and a *SA-TA* setup, in which the accepted solutions are refined by TA step.

Figure 5a, 5b and 5c summarize the results under different deadline constraints: $d_1 = 1.3d_{min}$, $d_2 = 1.2d_{min}$ and $d_3 = 1.1d_{min}$, respectively, where d_{min} is the global minimum source-to-sink delay obtained from *ILP2*. Comparing these results, we observe that the performance of *pure SA* algorithm degrades rapidly as the timing constraints become tighter. Figure 6 compares the performance at medium effort (500 iterations in SA) with different deadlines. As can be seen, the average energy consumption increases from $106.7\%E_{opt}$ to $119.1\%E_{opt}$ as the deadline reduces from d_1 to d_3. In the mean time, the success rate drops dramatically from 94.4% to 45.2%. The reason lies in the fact that the *pure SA* concentrates on finding better (concerning energy) solutions, a lot of which are however abandoned because

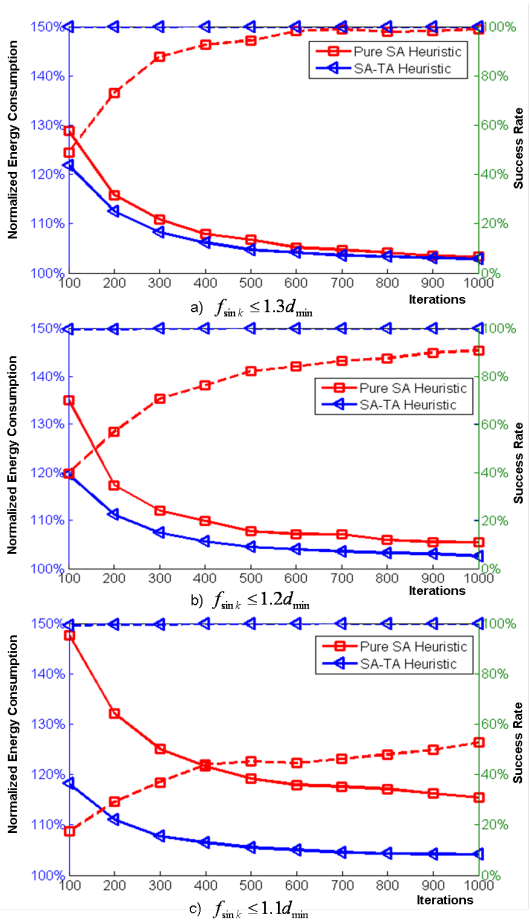


Figure 5. Performance Evaluation of Heuristic Under Timing Constraints

of violation of deadlines. The performance is significantly improved by introducing the TA phase. While the SA algorithm leads the way to the energy-efficient solution, the TA step actively looks around to find a near solution that meets timing constraints. It can be seen in figure 6 that the performance of *SA-TA* are $104.7\%E_{opt}$ in presence of d_1 and $105.5\%E_{opt}$ for d_3. With 1000 iterations, SA-TA achieves the results $102.8\%E_{opt}$, $102.6\%E_{opt}$ and $104.1\%E_{opt}$, under constraint d_1, d_2 and d_3 respectively.

We also performed experiments using SA-TA with baseline mapping. We observe that the performance both in terms of energy consumption and success rate can still be slightly improved. These experimental results verify that the performance of *SA-TA* algorithm is very close to the global optimum even under very stringent timing constraints. Further performance improvement can be achieved in two ways. First, the optimization iterations can be increased. Second, the heuristic can be executed multiple times for the same problem in order to discover a concrete optimal solution. The high execution speed of our algorithm makes it convenient to do so. While the typical solving time of ILP

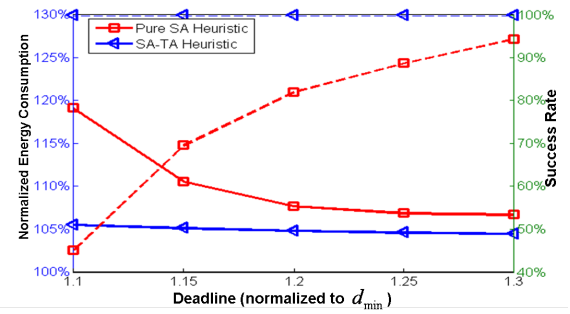


Figure 6. Performance Comparison over Different Deadline Constraints

Execution Time (ms) for TG3 (26 nodes, 28 edges)				
#Iterations	No TC, No TA	$1.3d_{min}$, with TA	$1.2d_{min}$, with TA	$1.1d_{min}$, with TA
100	7	8	10	11
500	26	58	85	77
1000	45	107	136	320

Table I
EXECUTION TIME OF *SA-TA* HEURISTIC

is multiple minutes up to several hours (in the complexity of our experimental setup), execution of *SA-TA* takes less than a second. The average execution time (in milliseconds) under several configurations (e.g. the number of iterations, with/without TA) is measured and presented in table I. Due to the space restriction, we present only the results for the most complex task graph used in the experiment (*TG3*). The execution time is measured on a work station with 2.2GHz CPU and 1GB memory. Since the TA procedure terminates when the deadline is met, the execution time under tighter deadline constraints may become longer. Comparing with recent work [13], in which the proposed heuristic takes seconds to allocate task graphs of much smaller size, our SA-TA algorithm achieves much higher execution speed, which is the key to enable efficient Design Space Exploration (DSE).

VII. RELATED WORK

Energy-aware task allocation for multiprocessor systems has been extensively studied in existing work. In this section, we review several relevant approaches and provide a comparison concerning three main factors, namely if the hardware platform is homogeneous or heterogeneous, whether or not task-dependencies are considered, and if the communication cost is considered (table II).

Xian et al studied the scheduling problem for a set of independent tasks on homogeneous MPSoCs [14]. Since the actual execution time of a real-time task may vary, e.g. depending on different inputs, the authors suggest taking the probabilistic distribution of the execution time into account to derive energy-efficient schedules. *Chen et al* presented a compiler-based application mapping framework in [4]. Their target applications are array-based loop-intensive applications and they focus on homogeneous MPSoCs with 2D-mesh topology. In [9], *Xu et al* proposed an algorithm for mapping task graphs onto chip multiprocessors. The authors exploited simultaneous application of pipelining and parallel execution to schedule task graphs in an energy-efficient manner under QoS constraints. *Baruah* and *Anderson* considered the problem of synthesizing multiple periodic tasks on identical multiprocessors [15].

Design techniques aiming at reducing the communication power have also been investigated. To name a few, *Kim* and *Horrowitz* applied the DVS policy on network links and suggested to use links with scalable voltage/frequency levels [16]. The idea is to dynamically adjust the supply voltage depending on the current network load. *Shin et al* considered the task allocation problem in the context of scalable links [5]. The task mapping is computed using a genetic algorithm based approach and only homogeneous platforms are considered. In this paper, we consider only systems with static network links. The case for systems with DVS-enabled links is left for future work.

While the work mentioned above considers only homogeneous multiprocessor systems, the case for heterogeneous systems has become an active research topic in recent years. *Yu et al* [17] studied the task allocation problem for a set of independent tasks. *Yang et al* [18] presented an approximation scheme to derive near-optimal solutions for the same problem. In [6], *Goraczko et al* addressed the problem with task dependencies and proposed an ILP formulation. Our work is different from [6] in following aspects: First, their target platforms are loosely coupled multiprocessor systems, whereas we focus on MPSoCs with on-chip network, in which the network structure plays a key role. Second, they ignored the communication latency/energy and we explicitly take that into account. In addition, we provide an efficient heuristic for fast computation of task mapping.

The authors in [10] consider also the case for heterogeneous systems with communication cost. One major difference is that [10] focuses on run-time task mapping whereas we focus on static allocation. In order to enable runtime allocation, the task implementation for each possible processor type must be available and stored in the system, which introduces considerable overhead and therefore limits the applicability of their approach. Also, the runtime scheduling overhead can not be underestimated. Another difference is that the task mapping is done stepwise in [10]. The task-to-processor-type mapping is firstly determined solely depending on processing energy is kept fixed in later steps. Our work uses a unified approach to find the task mapping based on system-wide energy. The advantage of using a unified approach has been demonstrated clearly in [13]. However, the heuristic proposed in [13] needs to solve multiple instances of modified ILP, leading to potentially very slow execution speed. The SA based heuristic presented

reference	Hetero.	task dep.	comm. cost
[14]	×	×	×
[4]	×	√	×
[9]	×	√	√
[5]	×	√	√
[17]	√	×	×
[18]	√	×	×
[6]	√	√	×
[10]	√	√	√
[13]	√	√	√
This Work	√	√	√

Table II
COMPARISON OF RELATED WORK

in this paper improves the execution speed.

VIII. CONCLUSION AND OUTLOOK

This paper presents algorithms for energy-aware task allocation on heterogeneous multiprocessor systems. An improvement to existing work is that we explicitly considered the tradeoff between processing and communication power and use an unified approach to pursue the system-level optimal task mapping and execution schedule. The proposed ILP formulation can compute the global optimal mapping but suffer from large execution time. The SA-TA algorithm achieves performance very close to the global optimum even under tight timing constraints (less than 5% above global optimum using 1000-iteration SA-TA). In the mean time, execution time is significantly reduced.

One limitation of the current work is the simplified energy model. To incorporate a more precise energy model, the leakage power should be considered and the power management features of the processor e.g clock gating should be taken into account. Another direction of future work is to consider the other side of the problem, that is, how to find the optimal platform configuration such as processor type and memory size for a certain task set. This is a typical design space exploration problem. The high execution speed of the *SA-TA* heuristic enables rapid DSE.

ACKNOWLEDGMENT

This work has been supported in part by the European research project ACROSS under the Grant Agreement ARTEMIS-2009-1-100208.

REFERENCES

[1] J. Ceng, J. Castrillón, W. Sheng, H. Scharwächter, R. Leupers, G. Ascheid, H. Meyr, T. Isshiki, and H. Kunieda, "MAPS: an integrated framework for MPSoC application parallelization," in *DAC*, 2008.

[2] W. Wolf, "The future of multiprocessor systems-on-chips." in *DAC*, 2004.

[3] J. Hu and R. Marculescu, "Energy- and performance-aware mapping for regular noc architectures," *IEEE Transactions on CAD of Integrated Circuits and Systems*, April 2005.

[4] G. Chen, F. Li, S. W. Son, and M. T. Kandemir, "Application mapping for chip multiprocessors," in *DAC*, 2008.

[5] D. Shin and J. Kim, "Power-aware communication optimization for networks-on-chips with voltage scalable links," in *CODES+ISSS*, 2004.

[6] M. Goraczko, J. Liu, D. Lymberopoulos, S. Matic, B. Priyantha, and F. Zhao, "Energy-optimal software partitioning in heterogeneous multiprocessor embedded systems," in *DAC*, 2008.

[7] T. A. AlEnawy and H. Aydin, "Energy-aware task allocation for rate monotonic scheduling," in *RTAS*, March, 2005.

[8] L. P. H. Wang, X. Zhu and S. Malik, "Orion: A power-performance simulator for interconnection networks," in *IEEE MICRO*, 2002.

[9] R. Xu, R. G. Melhem, and D. Mossé, "Energy-aware scheduling for streaming applications on chip multiprocessors," in *RTSS*, 2007.

[10] P. K. F. Hölzenspies, J. Hurink, J. Kuper, and G. J. M. Smit, "Run-time spatial mapping of streaming applications to a heterogeneous multi-processor system-on-chip (MPSOC)," in *DATE*, 2008.

[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, 1983.

[12] http://ziyang.eecs.umich.edu/ dickrp/tgff/, "TGFF."

[13] P. Ghosh, A. Sen, and A. Hall, "Energy efficient application mapping to noc processing elements operating at multiple voltage levels," in *NOCS*, 2009.

[14] C. Xian, Y.-H. Lu, and Z. Li, "Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time," in *DAC*, 2007.

[15] J. H. Anderson and S. K. Baruah, "Energy-efficient synthesis of periodic task systems upon identical multiprocessor platforms," in *ICDCS*, 2004.

[16] J. Kim and M. Horowitz, "Adaptive supply serial links with sub-1 v operation and per-pin clock recovery," in *IEEE International Solid-State Circuits Conference*, 2002.

[17] Y. Yu and V. Prasanna, "Power-aware resource allocation for independent tasks in heterogeneous real-time systems," in *ICPDS*, Dec. 2002.

[18] C.-Y. Yang, J.-J. Chen, T.-W. Kuo, and L. Thiele, "An approximation scheme for energy-efficient scheduling of real-time tasks in heterogeneous multiprocessor systems," in *DATE*, 2009.