



Temperature-Aware Task Allocation and Scheduling for Embedded Multiprocessor Systems-on-Chip (MPSoC) Design

YUAN XIE AND WEI-LUN HUNG

Department of Computer Science and Engineering, Pennsylvania State University, University Park,
PA 16802, USA

Received: 13 April 2006; Revised: 24 May 2006; Accepted: 26 May 2006

Abstract. Temperature affects not only the performance but also the power, reliability, and cost of the embedded system. This paper proposes a temperature-aware task allocation and scheduling algorithm for MPSoC embedded systems. Thermal-aware heuristics are developed, and a temperature-aware floorplanning tool is used to reduce the peak temperature and achieve a thermally even distribution while meeting real time constraints. The paper investigates both power-aware and thermal-aware approaches to the task allocation and scheduling. The experimental results show that the thermal-aware approach outperforms the power-aware schemes in terms of maximal and average temperature reductions. To the best of our knowledge, this is the first MPSoC task allocation and scheduling algorithm that takes temperature into consideration.

Keywords: embedded system design, thermal-aware design, scheduling, system-on-chip design

1. Introduction

Over the past several years, embedded system design has evolved from using simple architectures to adopting more complex MPSoC (Multiprocessor System-on-a-Chip) architectures. An MPSoC combines at least two embedded microprocessors, specialized digital hardware, and often mixed-signal circuits to provide a complete integrated system [17].

Normally, there are two ways to architect an MPSoC embedded system. The first one is the *platform-based design* approach. The designer picks one pre-defined MPSoC platform and maps the applications of the embedded system onto the platform. Some MPSoC platforms have homogeneous multiprocessors. For example, IBM PowerPC 440 platform-based SOC architecture contains two PowerPC 440 processor cores and several digital IP

cores [22]; a newly announced symmetric multiprocessing ARM [12] platform can support up to four ARMv6-compliant CPU cores. Other MPSoC platforms may have heterogeneous multiple embedded processors. For example, The Philips Nexperia Digital Video Platform contains one VLIW processor and one RISC processor [21]. Once the platform is decided, the designer has to partition the system specification among embedded processors and specialized cores. By using an existing platform as starting point for new MPSoC system implementation, platform-based design reduce design time and risk. The second way to architect an MPSoC system is called *Hardware/software co-synthesis* [11]. Given the system specification and the technology library (which contains different types of processing elements), co-synthesis process generates the customized MPSoC architecture that meets the performance

requirement while minimize the system cost. Compared to platform-based design, co-synthesis has more flexibility to explore design space and tailor the architecture to fit a particular application, at the expense of design time and higher risk.

No matter which approach is chosen to architect the MPSoC embedded system, the designer has to partition the system specification into hardware (implementing as specialized IP block) and software modules (running on embedded processors) to meet performance, power and cost goals [11]. The system allocation and scheduling routine [3] determines the mapping and execution schedules of the tasks on processing elements. To guarantee that all timing and resource constraints are met, usually a static allocation and scheduling strategy is used. An off-line scheduling strategy considers resource, precedence, and synchronization requirements of all tasks in the system and generates a feasible schedule that guarantees the embedded system performance/power constraint.

Traditional allocation and scheduling routines use performance or power as the design metric [3–5]. Process scaling and aggressive performance improvements have resulted in a dramatic power consumption increase. Power density directly translates into heat; as a result, the temperature in modern VLSI circuits increases dramatically due to smaller feature size, higher packing density, and rising power consumption. For example, the hotspot in a modern chip might have a temperature of more than 150°C (<http://www.isonics.com>). Temperature affects not only the reliability but also the performance, power, and cost of the embedded system. At sufficiently high temperatures, many failure mechanisms (such as electromigration and stress migration) are significantly accelerated, resulting in reduced system reliability [2]; interconnect delay increases and MOS current drive capability decreases as chip temperature increases. For example, interconnect (Elmore) delay increases approximately 5% and MOS current drive capability decreases approximately 4% for every 10°C temperature increase [1]. The leakage power increases exponentially with the temperature increase, which in turn causes further temperature increase and might incur the well-known thermal runaway problem; finally, the cost of cooling a hot chip increases as the hot spot temperature goes up. Therefore, it is very important to reduce hotspot temperature and have a thermal

balanced design through a thermal-aware design methodology.

Power-aware design alone is not able to address the temperature challenge, and many low-power techniques have insufficient impact on chip temperature because they do not directly target the *spatial and temporal* behavior of the operating temperature. Therefore, even though it is related to the power-aware design area, thermal-aware design itself is a distinct and important research area.

In this paper, we investigate both power-aware and thermal-aware approaches for task allocation and scheduling. The experimental results on both platform-based and co-synthesis based MPSoC designs show that thermal-aware approach outperforms the power-aware schemes in terms of maximal and average temperature reductions.

The paper is organized as follows: Section 2 discusses the related works, Section 3 gives a brief background on hardware/software co-synthesis and the problem formulation, Section 4 presents the task allocation and scheduling procedure, Section 5 presents the experimental result and finally we conclude the paper in Section 6.

2. Related Work

There have been extensive studies in the literature on the task allocation and scheduling of hardware/software co-synthesis. Precedence-constrained tasks allocation and scheduling is proved to be an NP-hard problem [19], thus the allocation and scheduling algorithms usually use various heuristics to quickly find a sub-optimal solution. The simulated annealing based scheduling was used in Dobioli's work [4] with an attempt to balance the latency and power consumption incurred by assigning tasks onto different PEs. In [5], Liu et al. targeted at a real mission-critical application to demonstrate the effectiveness of maintaining the power budget while satisfying the performance requirements. Shang and Jha [6] tackled the low power HW/SW co-synthesis problem for an embedded system with dynamically reconfigurable FPGA processors and other system resources.

Recently, several research studies investigated the task scheduling problem for DVS-enabled multiprocessor real-time embedded systems. For example, Zhang et al. [7] formulated the task scheduling

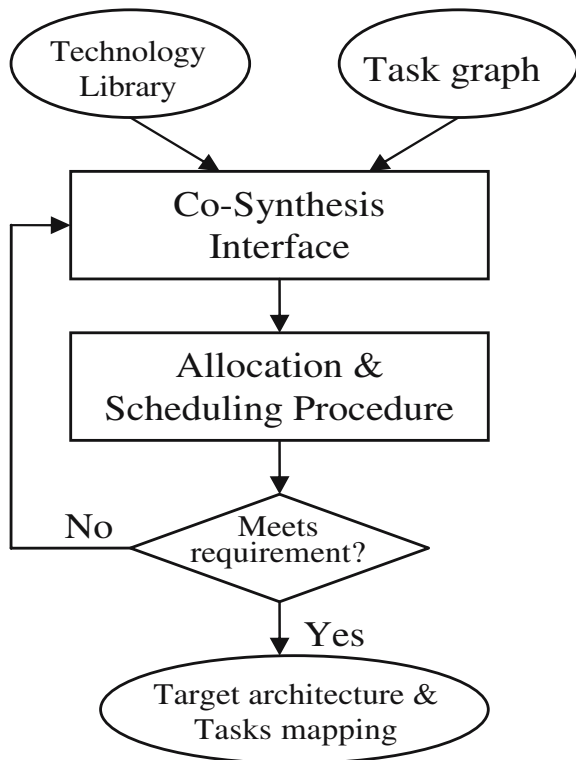


Figure 1. The flow of the traditional co-synthesis framework.

problem with voltage selection together as an integer linear programming (ILP) problem. Shin and Kim [8] and Luo and Jha [13] proposed the condition-aware DVS tasking scheduling algorithm, which can handle more complicated conditional task graphs. Schmitz et al. [9] used a two-step iterative synthesis approach for the same problem.

All previous studies tried to minimize the system cost or power consumption under the performance constraint, but the temperature issue has not been considered as an explicit design goal. Thermal-aware design techniques for electronic systems have mainly focused on the board level or package level. Recently, we have seen an increasing interest in the processor micro-architecture level thermal management, which employs dynamic thermal management (DTM) that use the runtime knowledge of the application behavior and current thermal status of different function units to adjust the instruction execution [15]. At the physical design level, thermal-aware design techniques have been investigated for several years. For example, Tsai and Kang [1]

proposed a thermal-aware standard cell placement tool. Hung et al. [18] proposed a thermal-aware IP placement algorithm for uniform-size Network-on-Chip IP cores.

Unlike the previous studies, our allocation and scheduling algorithm aims at reducing the hotspot temperature while achieving other design goals. To the best of our knowledge, this is the first thermal-aware task allocation and scheduling for MPSoC that takes temperature into account.

3. Problem Formulation

The traditional co-synthesis flow is shown in Fig. 1. The co-synthesis framework takes the technology library and the task graph as inputs. Typically, a co-synthesis framework consists of three steps: the construction of target architecture with a number of PEs from technology architecture, the allocation of tasks on PEs, and the scheduling of execution sequence of tasks on each PE. The co-synthesis is an iterative process that the underlying architecture is improved until the performance constraint is met while system cost and/or power are minimized.

The task graph model [11] is a common application model to describe the specification of a real time periodic model to describe the specification of a real time periodic application. Figure 2 shows an example of the task graph. Applications are partitioned into a task graph, which is a directed acyclic graph, as shown in Fig. 2. In a task graph, each node represents a task that may have moderate to large

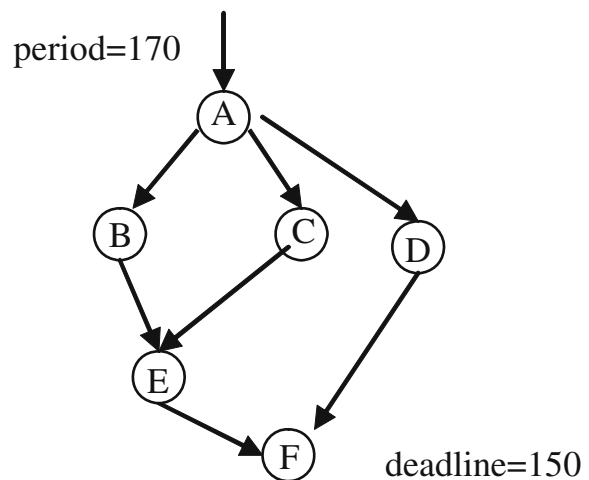


Figure 2. An example of a task graph.

granularity; the directed edges represent data dependencies between tasks. An edge, say $A \rightarrow D$, implies that task D cannot start execution until A is finished. Data dependency edges ensure the correct order of execution. Each edge is associated with a scalar describing the amount of data that must be transferred between the two connected tasks, which decides the communication time between the tasks if they are allocated onto different PE. We assume that if two tasks are allocated onto the same PE, the communication time is 0. Each node in the figure represents a task while the directed edge connects two nodes. The tasks come at a period and must finish before a deadline (real time constraint).

The target library stores the worst case power consumptions (WCPC) and worst case execution times (WCET) for a task executed on different PEs. Part of the technology library for the example in Fig. 2 is shown in Table 1.

Since some PEs are applications specific, they can only execute a certain types of tasks. For example, task D can never be partitioned on embedded processors because it can only be implemented as specialized ASIC to meet performance requirement.

The target architecture is normally a heterogeneous architecture which has a number of processing elements (PEs). A PE may be a CPU, DSP, ASIC or FPGA depending on the need of performance, cost, and/or power constraints. One example architecture is shown in Fig. 3. This architecture consists of two CPUs and two ASICs connected by a bus.

The co-synthesis problem tackled in this paper is defined as: given a task graph, and a technology library, the co-synthesis algorithm should generate a customized MPSoC architecture with minimum cost, and produce a feasible task mapping on the resultant

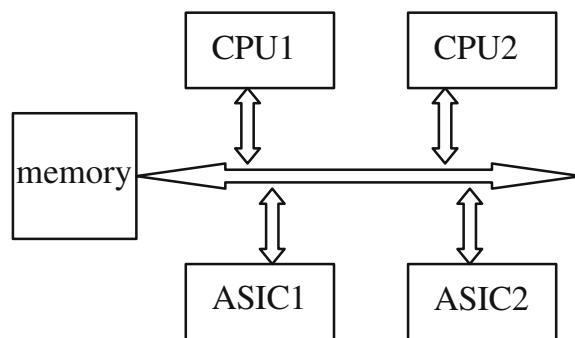


Figure 3. An example of the general MPSoC architecture.

architecture, which satisfies the deadline requirements of an application and reduces the hot spot and the average temperatures. The problem is the same for platform-based design except that the target architecture is pre-defined, such that only allocation and scheduling of tasks on the platform architecture is performed.

4. Tasks Allocation and Scheduling

For both platform-based MPSoC design and co-synthesis based customized MPSoC architecture, the task Allocation and Scheduling Procedures (ASP) is critical to generate good solutions. Task allocation and scheduling problem is proved to be an NP-hard problem [19], thus one usually uses various heuristics to quickly find a sub-optimal solution.

4.1. Allocation and Scheduling Procedure (ASP)

Our task allocation and scheduling procedure is similar to the one proposed by Xie and Wolf [3]. Figure 4 outlines the allocation and scheduling procedure (ASP). The procedure takes the task graph and architecture (either a pre-defined platform architecture or a customized architecture generated via co-synthesis) as input, and generates the task mapping and scheduling on the target architecture.

The static criticality (SC) for each task is calculated as the maximum distance from current task to the end task in a task graph. This is similar to the priority ordering in some list schedulers. For example, in Fig. 2, if we assume that the communication time for each edge is 1, and task D is partitioned to be implemented as ASIC. The static criticality (SC) of the tasks can be calculated and shown in Table 2.

Table 1. Part of a technology library.

Tasks	CPU1		CPU2		ASIC	
	WCET	WCPC	WCET	WCPC	WCET	WCPC
A	10	0.3	20	0.5	20	0.1
B	20	2.4	18	2	–	–
C	30	2	20	1.8	–	–
D	–	–	–	–	5	0.4
E	10	0.5	20	0.8	–	–
F	30	1.6	20	1.2	–	–

```

Allocation and Scheduling (task_graph, architecture)
begin
  for each task, calculate its static criticality
  while the ready list of tasks is not empty
  begin
    pick the first task in the ready list
    for each PE with respect to current task
      calculates the dynamic criticality upon
      current PE
    schedule the current task with the maximal
    dynamic criticality
    update the ready list by adding current task's
    successors to the ready list
  end
end

```

Figure 4. Allocation and Scheduling Procedure (ASP).

Note that the weight for each task that allocated on CPUs is calculated as the average WCET on CPUs, even though one can also specify to use the mediate WCET as the weight for the task.

The dynamic criticality (DC) calculation is based on three different factors and is defined as follows:

$$DC(task_i, PE_j) = SC(task_i) - WCET(task_i, PE_j) - \max(avl_PE_j, ready_task_i)$$

The dynamic criticality is related to the following factors:

1. Static urgency (SU). If a task's SU is high, it implies that this task is a critical task and should be given a high priority.
2. The worst case execution time (WCET) of this task on the PE.
3. The earliest start time of this task on the CPU, which is the maximum of the PE available time and the task ready time. Note that the ready_time(task) takes into account the communication time from its predecessor. We assume that the communication time between two tasks on the same CPU is 0.

Table 3 shows the first several steps to schedule the task graph in Fig. 2 on the target architecture with one CPU1 and one CPU2, while Task D is implemented as ASIC:

- Step 1: Only task A is in the ready list for scheduling. Since $DC(A, CPU1)=73$, $DC(A, CPU2)=63$, task A is scheduled on CPU1 from 0 to 10.

- Step 2: Task B, C, and D are all ready to be scheduled. Since D can only be implemented as ASIC, task D and the communication edge from A to D are scheduled. After that, C is allocated and scheduled on CPU2 since $DC(C, CPU2)=35$ is the largest one among all combination of B, C with CPU1 and CPU2.
- Step 3: Task B, E, and F are all ready to be scheduled, and B is scheduled to be on CPU1 because it has the largest DC (E and F are not shown in Table 3).

The allocation and scheduling algorithm is effective and fast on finding the task mapping and scheduling that satisfy the deadline requirement. However, it neglects the temperature impacts during the process and can potentially allocate tasks to PEs in such a way that some PEs have much higher temperature than other PEs due to uneven power density. To account for this problem, we introduce power/energy aware ASP and thermal-aware ASP.

4.2. Power-Aware Heuristics

Since the level of temperature heavily depends on the power density, in power-aware allocation and scheduling, the power/energy factor is involved in the process of calculating dynamic criticality. The intuition is that we want to reduce and balance the power consumption for each PE when we allocate and schedule a task. Therefore, the DC equation is modified as follows:

$$DC(task_i, PE_j) = SC(task_i) - WCET(task_i, PE_j) - \max(avl_PE_j, ready_task_i) - Pow$$

The last term (*Pow*) captures the effect of power/energy which can be interpreted by the following three heuristics.

Heuristic 1: minimize power consumption of current task This heuristic tries to minimize the power

Table 2. The static criticality for the example in Fig. 2.

Task	A	B	C	D	E	F
SU	83	61	67	31	41	25

consumption of the task to be allocated and scheduled. It makes use of fact that a task can have different power consumptions on different processing elements. When we try to assign a task to one specific PE, we first examine its power consumption on this PE. This number contributes the last term in the DC calculation. For example, in Fig. 5a, when considering scheduling task G with the assumption of the other terms staying equal in the equation, it will be placed on PE1 because of the lowest power consumption.

Heuristic 2: minimize cumulative average power of processing element This heuristic is to capture the historic cumulative average power consumption of each PE up to current execution time. If one PE has been used a lot, it will have a higher power density, and thus have higher chance to become a hot spot. The goal is trying to balance the average power consumption evenly among all available PEs. In Fig. 5b, the cumulative average power consumptions (including the power of task G) of PE1 are 0.8W, 0.76W for PE2, and 0.72W for PE3. The task G will be placed on a PE having the lowest cumulative average power, which is PE3.

Heuristic 3: minimize energy of current task This heuristic tries to minimize the energy of the task to be allocated and scheduled on the system. It makes use of fact that a task can have different energy consumptions on different processing elements. When we try to assign a task to one specific PE, we first examine its energy consumption on this PE. From the Fig. 5c, we can see that when the task G is placed on PE3, it will have the lowest energy value compared to those of the other two PEs.

4.3. Thermal-Aware Allocation and Scheduling

Although the above three heuristics include the power/energy effect, they address the temperature issue indirectly. The proposed thermal-aware ASP addresses this issue by taking the temperature into consideration.

4.3.1. Temperature Estimation. The temperature of each IP block depends on the power consumption, the physical dimension, and the position of the IP blocks. In order to facilitate the temperature estimation, a compact thermal model is needed to provide the temperature profile. Skadron et al. proposed a thermal modeling tool called HotSpot [10], which is

easy to use and computationally efficient for modeling thermal effects at the IP block level. HotSpot provides a simple compact model, where the heat dissipation within each functional block and the heat flow among blocks are accounted for. In Hotspot, the transfer thermal resistance R_{ij} of IP block PE i with respect to PE j is defined as the temperature rise at PE i due to one unit of power dissipated at PE j :

$$R_{ij} = \Delta T_{ij} / \Delta P_j$$

For any power distribution on the MPSoC architecture, each block's temperature can be calculated by applying the following equation:

$$\begin{bmatrix} T_1 \\ T_1 \\ \vdots \\ T_m \end{bmatrix} = \begin{bmatrix} R'_{11} & R'_{12} & \dots & R'_{1m} \\ R'_{21} & R'_{22} & \dots & R'_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ R'_{m1} & R'_{m2} & \dots & R'_{mm} \end{bmatrix} \begin{bmatrix} P_1 \\ P_1 \\ \vdots \\ P_m \end{bmatrix}$$

where P_i is the power consumed by IP block PE i and T_i is the temperature of the IP block PE i . The transfer thermal resistance matrix can be obtained from Hotspot, given the floorplanning of the MPSoC architecture.

4.3.2. Thermal-Aware Floorplanning Tool. Since the temperature of each IP block depends on not only the power consumption but also the physical dimension, and the position of the IP blocks, we designed a thermal-aware floorplanning tool [16], which is a genetic algorithm based floorplanning framework that

Table 3. The first several scheduling steps for Fig. 2.

SU	DC(task, CPU $_j$)			Schedule (T_{start} , T_{end} , PE)
	Step 1	Step 2	Step 3	
A	83 73 63	--	--	(0, 10, CPU1)
B	61	-- 31 31	31 11	(10,30, CPU1)
C	67	-- 27 35	--	(12,32, CPU2)
D	31	--	--	(11,16, ASIC)
A→D				(10,11, BUS)
A→C				(11,12, BUS)

aims at reducing hot spots and distributing temperature evenly across a chip while optimizing the traditional design metric, chip area. HotSpot is used as an inner loop during floorplanning to calculate temperature. It has been shown that our combined area and thermal optimization technique decreases the peak temperature sufficiently, and generates a floorplanning that is as

compact as the one generated using the traditional area-oriented techniques [16]. One example is shown in Fig. 6. The black areas depict the dead space while the numbered blocks represent different modules. The shaded blocks are the top four modules that have the largest power consumption. Figure 6a and b shows the floorplanning of the same example circuit. By

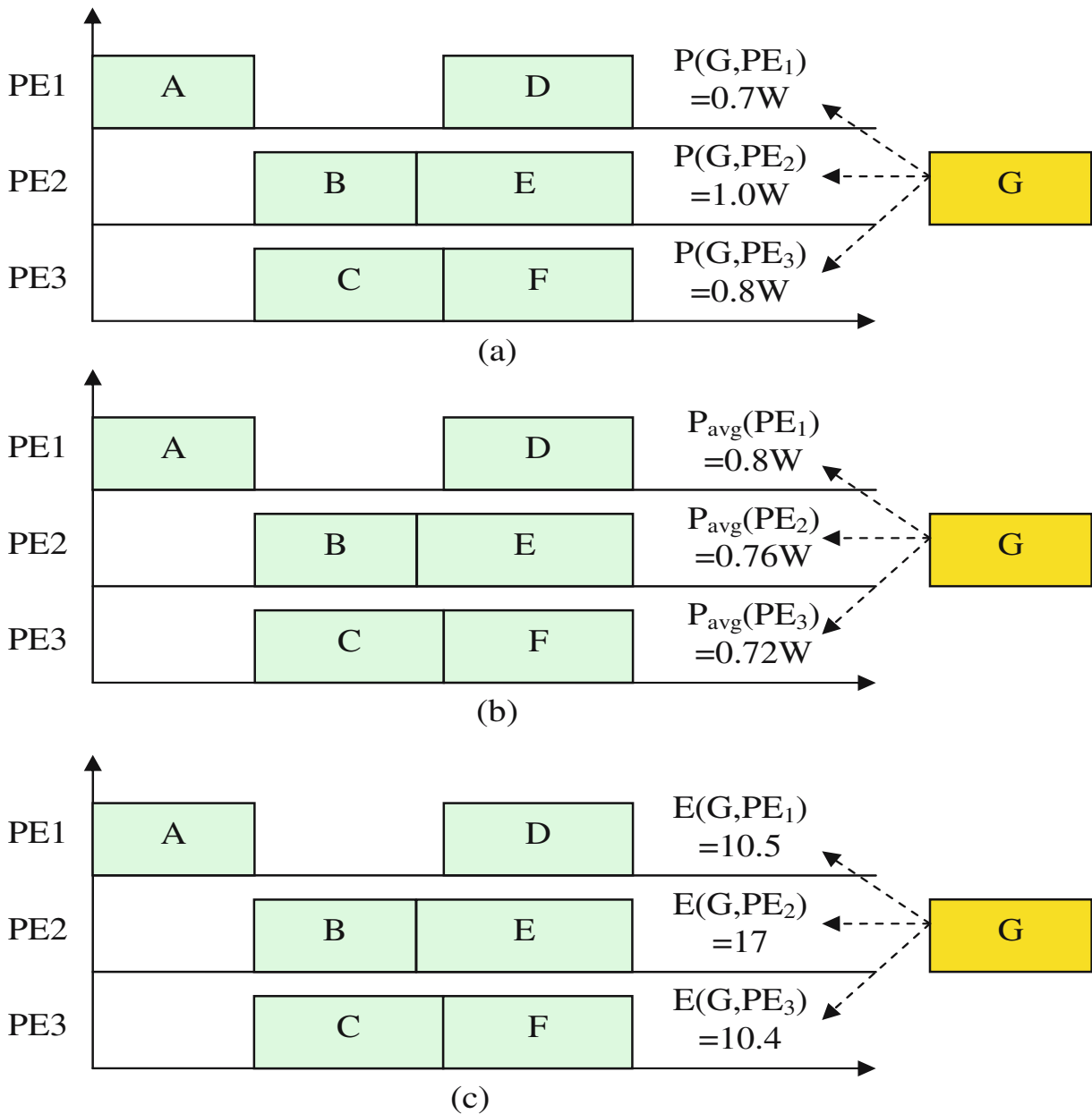


Figure 5. Example of three different power heuristics.

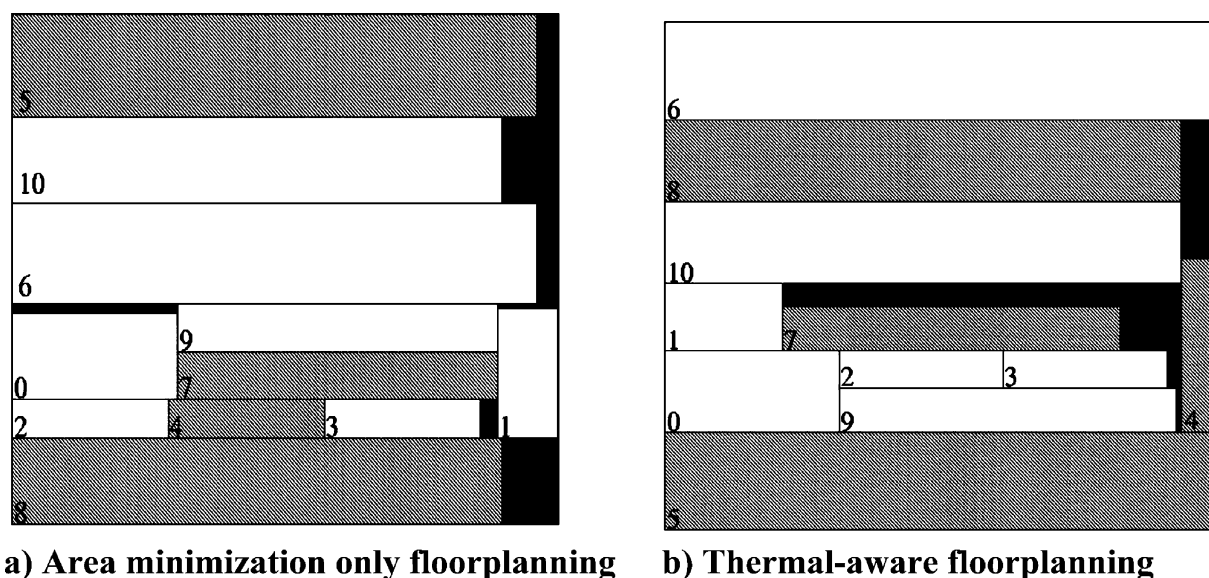


Figure 6. Comparison of two floorplanning: both have the same area but the *right* one has better thermal distribution.

using our thermal-aware floorplanning tool, the hot-spot temperature (maximum temperature) drops from 123 to 120°C, and the average temperature drops from 111 to 110°C, while the resulting area is the same. Comparing Fig. 5a and b, we can see that the thermal-aware floorplanning spreads out the modules with high power consumption, such that it can achieve lower hotspot temperature and thermal-balanced floorplanning.

4.3.3. Thermal-Aware Heuristics. To capture the thermal impact on allocation and scheduling, we develop two heuristics to capture the spatial and temporal behavior of the temperature effect.

Heuristic 1: The first heuristic is to capture the spatial behavior of the temperature effect, by modifying the calculation of the dynamic criticality. At each allocation and scheduling iteration, the temperature estimation (either average temperature or peak temperature, depending on the design goal) obtained from the HotSpot tool is used to calculate dynamic criticality, which is defined as below and the newly added Temp term sets the goal of minimization of the temperature:

$$\begin{aligned}
 DC(task_i, PE_j) = & SC(task_i) - WCET(task_i, PE_j) \\
 & - \max(avl.PE_j, ready_task_i) \\
 & - Temp
 \end{aligned}$$

If $task_i$ is allocated and scheduled onto PE_j and the resultant chip temperature is lower, the corresponding dynamic criticality will be larger. Note that the Temp in equation is not the temperature of PE_j . It is the global chip temperature (either the maximum temperature or the average temperature), because the allocation and scheduling of a task onto a particular PE not only affects that PE's temperature but also the neighboring PEs' temperature, since heat always flows from a PE with high temperature to the neighboring PEs with lower temperature.

Heuristic 2: The second heuristic is called **Activity Migration**, and it captures the *temporal* behavior of the operating temperature. Compared to Heuristic 1, which only generates a static schedule for a single period and repeat it periodically, this type of technique generates an allocation and schedule that moves tasks among processing elements periodically. Figure 7 shows one example: two homogenous processing elements (PE) execute five tasks arriving periodically; schedule (a) repeat the same schedule periodically, where PE1 is always fully utilized and potentially becomes a hotspot; schedule (b) takes the temporal thermal effect into account and migrates task A back and forth between two PEs periodically, such that both PEs are not fully utilized and a thermally balanced design is achieved. Note that the overall power consumption is the same for schedule

(a) and schedule (b), but the temperature profile is different, which demonstrates the difference of power-aware techniques and thermal-aware techniques. Note that this example only looks at the temporal behavior; by combining with thermal estimation from the thermal-aware floorplanning tool and the first heuristic, the activity migration is able to explore the spatial temperature profile as well (for example, putting more workload on a cooler processing element).

The heuristics are implemented in our thermal-aware allocation and scheduling procedure and the overall design framework is shown in Fig. 8. We use a co-synthesis tool called ASICosyn [23] as the co-synthesis interface. However, unlike the flow shown in Fig. 1, the allocation and scheduling procedure executes and then activates the thermal-aware floorplanning [16] when considering placing a task on one specific PE. The HotSpot tool interacts with the floorplanning procedure to provide information of temperature. The co-synthesis framework keeps executing until there is no improvement on current architecture configuration. For the platform-based thermal-aware design, the target architecture and the task graph are given, and the HotSpot is activated by the modified ASP with thermal inquiries. This flow is depicted in Fig. 8b.

5. Experimental Results

The co-synthesis work is modified to have the ability of handling the pre-defined platform architecture and the newly proposed scheduling procedures. The algorithm is implemented in C++ and the experiments are conducted on a dual Intel Xeon processors (3.2 GHz, 2 GB RAM) machine running Linux. The benchmarks with relative power consumptions are either generated by the modified TGFF tool [14] or from the Embedded System Synthesis Benchmarks Suite [20]. The runtime of our approach is time-efficient and is less than 250 s for all of the benchmark.

The baseline design is generated by using the traditional co-synthesis flow (as shown in Fig. 1) and the allocation and scheduling algorithm (as shown in Fig. 4) without considering either power balancing or thermal balancing. After the co-synthesis is done, the thermal-aware floorplanning tool is used to generate the layout and estimate the temperature.

The first experiment we conduct is to compare the temperature differences using different power heuristics, for both co-synthesis customized architecture and platform-based architecture. The experimental results are shown in Table 4. The two columns under the *co-synthesis customized architecture* are the results of using co-synthesis flow in Fig. 8a, while

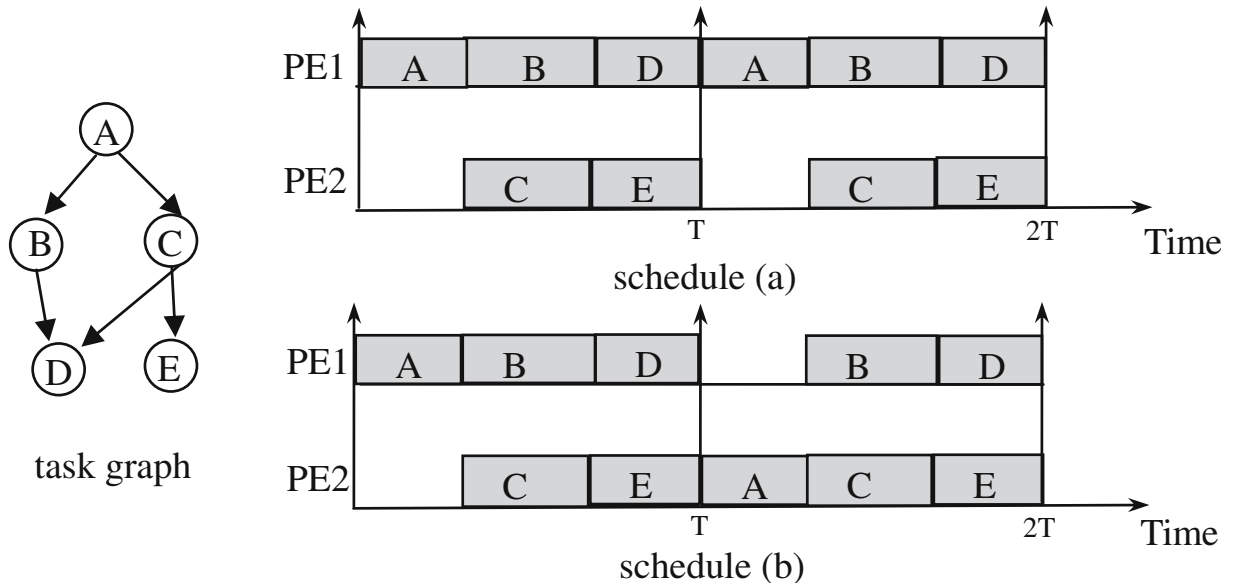


Figure 7. Task arrives at period T , both a and b meet timing requirement, but schedule b is a thermal-aware schedule using Activity Migration and results in lower temperature profile.

the other two columns represent the results from the platform-based target architecture in Fig. 8b.

The very first row out of four rows' groups indicates the characteristics of each benchmark and is the baseline case that does not take the power into consideration. The following three rows represent three power heuristics. As can be seen from the table, when considering power only, the third power heuristic outperforms the other two heuristics and the baseline approach. This result indicates that minimizing the energy of a task executed on one specific PE achieves the best temperature result among all three heuristics. Thus, the third power heuristic will be used in the following experiments.

Table 4 also shows that platform-based architecture has lower hotspot temperature than the customized architecture generated by co-synthesis. With the platform-based architecture, the tasks can be distributed among all PEs on the pre-defined platform. In contrast, the primary goal of co-synthesis is to minimize the cost

(using as fewer PEs as possible) while meeting the performance deadline. Thus the chance of having hotspot and un-even temperature distribution for platform-based design is lower. The customized architecture via co-synthesis tends to have higher power densities than the platform-based architecture, even though the number of PEs used in the architecture may be fewer than pre-defined platform architecture.

The second experiment is to demonstrate the effectiveness of our thermal-aware approach in terms of lowering the peak and the average temperatures. We take the best results of customized architecture and platform-based architecture from the first experiment for comparison. The power-aware and thermal-aware customized architecture comparison is shown in Table 5. As we mentioned in an earlier section, the peak temperature is reduced while trying to reach the lowest average temperature. From the results, the customized architecture with thermal-aware approach

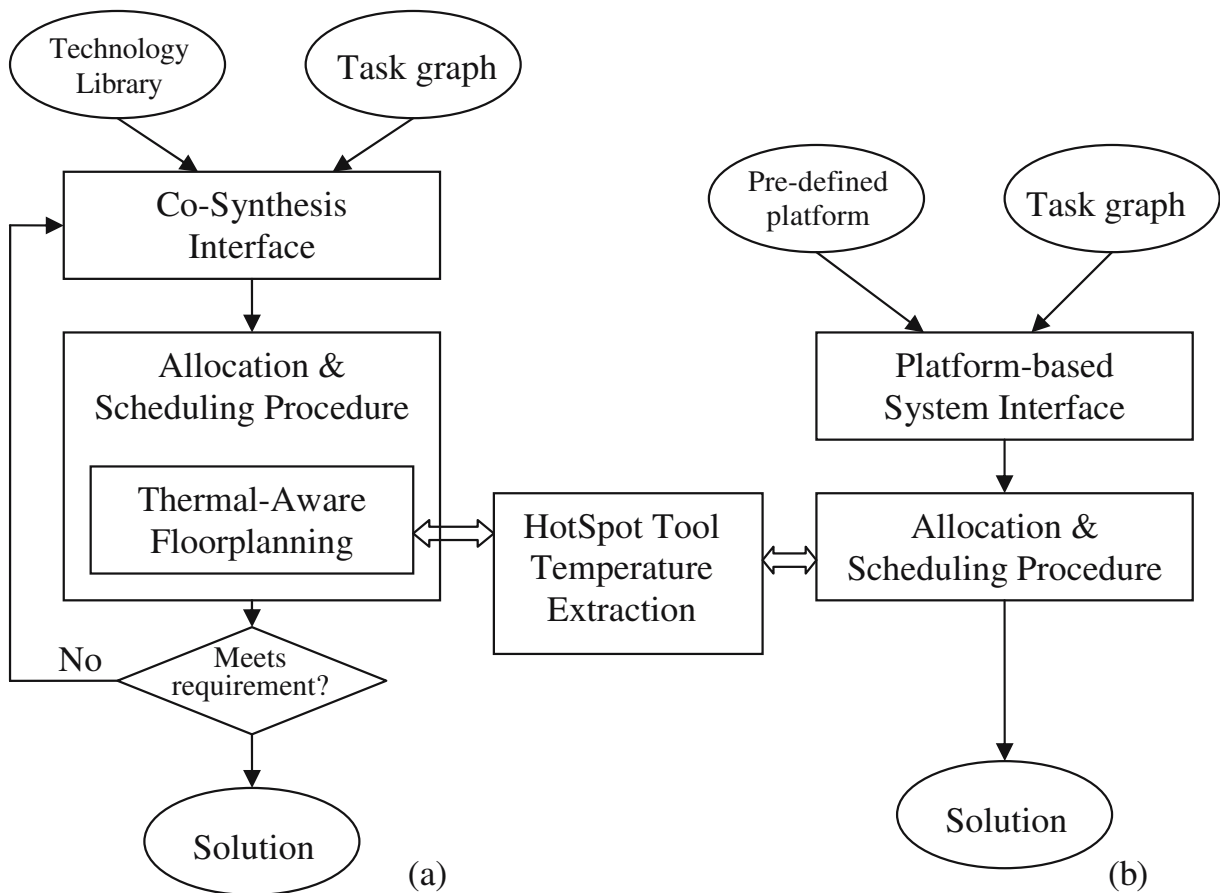


Figure 8. The flows of the thermal-aware co-synthesis framework and thermal-aware platform-based system design.

Table 4. The comparisons of different power heuristics for co-synthesis customized architecture and platform-based architecture.

Name/task/edge/deadline		Co-synthesis customized architecture		Platform-based architecture	
		Max. temp.	Avg. temp.	Max. temp.	Avg. temp.
Bm1/19/19/790	Baseline	118.18	106.32	100.59	81.03
	Heuristic 1	121.7	109.29	85.88	75.58
	Heuristic 2	118.18	106.32	107.16	82.78
	Heuristic 3	113.29	104.49	85.88	75.58
Bm2/35/40/1500	Baseline	121.44	110.22	114.33	101.04
	Heuristic 1	115.21	107.55	107.63	98.21
	Heuristic 2	121.44	110.22	113.31	99.96
	Heuristic 3	112.82	105.42	106.63	97.4
Bm3/34/39/1450	Baseline	122.63	111.84	111.81	100.87
	Heuristic 1	121.57	110.94	119.4	104.48
	Heuristic 2	122.63	111.84	114.35	100.40
	Heuristic 3	114.22	107.39	110.65	99.95
Bm4/39/43/1650	Baseline	113.58	101.76	113.81	98.47
	Heuristic 1	110.33	100.46	106.63	96.74
	Heuristic 2	110.49	100.6	113.81	98.47
	Heuristic 3	109.96	100.15	103.95	94.69
Bm5/51/60/2000	Baseline	122.09	111.14	106.54	97.05
	Heuristic 1	122.28	111.53	100.61	89.74
	Heuristic 2	117.86	111.13	106.54	91.62
	Heuristic 3	118.68	109.87	100.42	89.24
Bm6/54/63/2050	Baseline	121.47	103.52	116.35	99.31
	Heuristic 1	119.44	103.25	117.78	102.13
	Heuristic 2	114.52	103.09	113.58	101.39
	Heuristic 3	110.86	101.05	108.57	97.73

demonstrates that it can effectively reduce the divergence between hot spot and average temperatures and lower both of them. The total average temperature reduction is 10.9 and 6.95°C for the maximal and the average, respectively. This result indicates that

Table 5. The temperature comparisons of the power-aware and the thermal-aware approaches on co-synthesis architecture.

Benchmark	Power-aware co-synthesis		Thermal-aware co-synthesis	
	Max. temp.	Avg. temp.	Max. temp.	Avg. temp.
Bm1	113.29	104.49	87.11	86.13
Bm2	112.82	105.42	106.38	99.84
Bm3	114.22	107.39	108	103.2
Bm4	109.96	100.15	102.08	96.28
Bm5	118.68	109.87	106.32	102.48
Bm6	110.86	101.05	104.39	98.72

observing the average temperature of all using PEs while doing task scheduling is beneficial to control the temperature of an embedded system.

As for the platform-based architecture, the proposed thermal-aware approach outperforms the pow-

Table 6. The temperature comparisons of the power-aware and the thermal-aware approaches on platform-based architecture.

Benchmark	Power-aware platform arch.		Thermal-aware platform arch.	
	Max. temp.	Avg. temp.	Max. temp.	Avg. temp.
Bm1	85.88	75.58	65.71	61.16
Bm2	106.63	97.40	96.33	93.47
Bm3	110.65	99.95	96.98	93.63
Bm4	103.95	94.69	103.03	94.59
Bm5	100.42	89.24	94.85	85.76
Bm6	108.57	97.73	100.66	95.82

er-aware approach in both temperature attempts. From the results in Table 6, we can see that, under thermal-aware approach, both of the maximal and average temperatures are lower than those of in the corresponding power-aware approach and approximately by 9.75 and 5.02°C, respectively.

The results from Tables 5 and 6 show that with the platform-based architecture, the thermal ASP can balance the workloads of all PEs, and thus deliver a lower peak and average temperatures task mapping than that of in customized architecture.

6. Conclusion

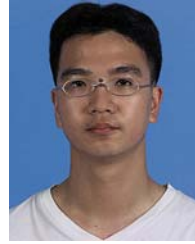
As technology scales, many embedded systems can be implemented as MPSoC due to smaller feature size, higher packing density. However, process scaling and aggressive performance improvements have resulted in a dramatic power density increase. Power density directly translates into heat; as a result, the temperature in modern VLSI circuits increases dramatically. Temperature affects not only the reliability but also the performance, power, and cost of the embedded system. To account for this problem, we propose the power-aware and thermal-aware task allocation and scheduling techniques, which try to reduce the hot spot and average temperatures by ordering tasks with respect to different PEs. The experimental results for both co-synthesis and platform-based design show that thermal-aware approach outperforms the power-aware schemes in terms of maximal and average temperature reductions.

References

1. C. Tsai and S. Kang, "Cell-Level Placement for Improving Substrate Thermal Distribution," *IEEE Trans. Comput.-Aided Des.*, vol. 19, no. 2, 2000, pp. 253–266, Feb.
2. J. Srinivasan, S. V. Adve, P. Bose, and J. Rivers, "The Impact of Technology Scaling on Lifetime Reliability," in *Proc. of International Conference on Dependable Systems and Networks*, June 2004.
3. Y. Xie and W. Wolf, "Allocation and Scheduling of Conditional Task Graph in Hardware/Software Co-Synthesis," in *Proc. of Design, Automation and Test in Europe Conference*, 2001, pp. 620–625.
4. A. Daboli, "Integrated Hardware-Software Co-Synthesis and High-Level Synthesis for Design of Embedded Systems under Power and Latency Constraints," in *Proc. of Design, Automation and Test in Europe Conference*, 2001.
5. J. Liu, P. Chou, N. Bagherzadeh, and F. Kurdahi, "Power-Aware Scheduling under Timing Constraints for Mission-Critical Embedded Systems," in *Proc. of the 38th Conference on Design Automation*, Las Vegas, Nevada, 2001, 840–845.
6. L. Shang and N. K. Jha, "Hardware-Software Co-Synthesis of Low Power Real-Time Distributed Embedded Systems with Dynamically Reconfigurable FPGAs," in *Proc. of the 2002 Conference on Asia South Pacific Design Automation/VLSI Design*, January 2002.
7. Y. Zhang, X. Hu, and D. Z. Chen, "Low-Power System Design: Task Scheduling and Voltage Selection for Energy Minimization," in *Proc. of the 39th Conference on Design Automation*, June 2002.
8. D. Shin and J. Kim, "System Level Issues: Power-Aware Scheduling of Conditional Task Graphs in Real-Time Multiprocessor Systems," in *Proc. of the 2003 International Symposium on Low Power Electronics and Design*, August 2003.
9. M. Schmitz, B. Al-Hashimi, and P. Eles, "Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems," in *Proc. of the Conference on Design, Automation and Test in Europe*, 2002, p. 514.
10. K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan, "Temperature-Aware Microarchitecture," in *Proc. of the 30th International Symposium on Computer Architecture*, pp. 2–13, June 2003.
11. W. Wolf and J. Staunstrup, "Hardware/Software co-design Principles and Practice," Kluwer, 1997.
12. *Microprocessor*, vol. 18, Archive 5, May 2004.
13. J. Luo and N. K. Jha, "Power-conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems," *Proc. of the 2000 IEEE/ACM International Conference on Computer-Aided Design*, November 2000.
14. R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task Graphs For Free," in *Proc. of the 6th International Workshop on Hardware/Software Codesign*, 1998.
15. D. Brooks and M. Martonosi, "Dynamic Thermal Management for High-Performance Microprocessors," in *Proc. of the Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 171–182.
16. W-L. Hung, Y. Xie, N. Vijaykrishnan, C. Addo-Quaye, T. Theocharides, and M. J. Irwin, "Thermal-Aware Floorplanning Using Genetic Algorithms," in *International Symposium on Quality Electronic Design (ISQED)*, 2005.
17. W. Wolf, "The future of Multiprocessor Systems-on-Chips," in *Proc. of Design Automation Conference*, 2004, pp. 681–685.
18. W. Hung et al., "Thermal-Aware IP Virtualization and Placement for Networks-on-Chip Architecture," in *Proc. of IEEE 22nd International Conference on Computer Design*, 2004.
19. K. Ramamritham, "Allocation and Scheduling of Precedence-Related Periodic Tasks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 6, no. 4, 1995, pp. 412–420, April.
20. Embedded System Synthesis Benchmarks Suite (E3S) <http://www.ece.northwestern.edu/~dickrp/e3s/>.
21. S. Dutta, R. Jensen, and A. Rieckmann, "Viper: A Multiprocessor SOC for Advanced Set-Top Box and Digital TV Systems," *IEEE Des. Test Comput.*, 2001, pp. 21–31, Sep/Oct.
22. <http://www-03.ibm.com/chips/products/asics/products/soc.html>
23. Y. Xie and W. Wolf, "Co-Synthesis with Custom ASICs," in *Proc. of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 129–134.



Yuan Xie is assistant professor of Computer Science Engineering Department at The Pennsylvania State University. Before joining Penn State in Fall 2003, he was with IBM Microelectronic Division's Worldwide Design Center. He received the B.S. degree in Electronic Engineering from Tsinghua University, Beijing, China, in 1997, and received the M.S. and Ph.D. degrees in Electrical Engineering from Princeton University in 1999 and 2002, respectively. His research interests include VLSI Design, computer architecture, embedded systems design, and electronics design automation. He is a member of IEEE and ACM.



Wei-Lun Hung is a Ph.D. student of Computer Science Engineering Department at The Pennsylvania State University. He received the B.S. degree in Information Management from National Taiwan University of Science and Technology, Taiwan, in 2000, and received the M.S. degree in Computer Science from National Tsinghua University, Taiwan in 2002. He is currently working toward his Ph.D. His research interests include VLSI Design, computer arithmetic, and electronics design automation. He is a member of IEEE.