An ILP Formulation for Task Scheduling on Heterogeneous Chip Multiprocessors

Suleyman Tosun[1], Nazanin Mansouri[2], Mahmut Kandemir[3], and Ozcan Ozturk[3]

[1] Computer Engineering Department, Selcuk University, Konya, Turkey
stosun@selcuk.edu.tr
[2] EECS Department, Syracuse University, Syracuse, NY, USA
namansou@ecs.syr.edu
[3] Computer Science and Engineering Department, The Pennsylvania State University, University Park, PA, USA
{kandemir, ozturk}@cse.psu.edu

Abstract. One of the main difficuties to map an embedded application onto a multiprocessor architecture is that there are multiple ways of this mapping due to several constraints. In this paper, we present an Integer Linear Programming based framework that maps a given application (represented as a task graph) onto a Heterogeneous Chip Multiprocessor architecture. Our framework can be used with several objective functions such as energy, performance, and fallibility (opposite of reliability). We use Dynamic Voltage Scaling (DVS) for reducing energy consumption while we employ task duplication to minimize fallibility. Our experimental results show that over 50% improvements on energy consumption are possible by using DVS, and the fully task duplicated schedules can be achieved under tight performance and energy bounds.

Keywords: Reliability, duplication, energy minimization, DVS, heterogeneous chip multiprocessors.

1 Introduction

Increasing complexity of embedded applications and their large dataset sizes make it imperative to consider novel embedded architectures that are efficient from both performance and power angles. Heterogeneous Chip Multiprocessors (HCM) are one such example where multiple processor cores are placed into the same die. While it is possible to put together multiple processor cores, interconnect and necessary memory components and build an HCM architecture, programming it in an effective manner is an entirely different matter.

The main difficulty comes from the fact that there are typically multiple ways of mapping a given embedded application to HCM. Favoring one way over the other is not easy, as it depends strongly on the constraints to be met and the objective function to be optimized. In particular, a typical real-time execution environment may demand proper balancing among different metrics such performance, power/energy,

A. Levi et al. (Eds.): ISCIS 2006, LNCS 4263, pp. 267 – 276, 2006.

code/ memory size, and reliability/fault tolerance. Therefore, software optimizers and other automated tools can be of great help in such situations since they can optimize the input code automatically under multiple criteria and a given objective function. Unfortunately, with a couple of exceptions, most of the previously proposed optimization frameworks in the literature exclusively target at minimizing the execution cycles of the application code. The power, code/data size, and reliability aspects of doing so have received relatively less attention.

In this paper, our goal is to map a given real-time embedded application (represented as a task graph) onto an HCM architecture. This mapping is implemented within a publicly available ILP (integer linear programming) tool [1] and operates under an objective function and multiple criteria (constraints). The objective function to be optimized and constraints to be satisfied can be functions of energy, performance or fallibility (opposite of reliability). Therefore, our framework can be used for three different purposes: (1) maximizing performance under energy and fallibility constraints; (2) minimizing energy under performance and fallibility constraints; and (3) minimizing fallibility under performance and energy constraints.

What makes this problem even more interesting is the fact that each processor in the HCM architecture can have different characteristics from the others in terms of maximum clock speed, available voltage/frequency levels (that could be used for reducing energy consumption), attached memory capacity, etc.. We use Dynamic Voltage Scaling (DVS) for reducing energy consumption while we employ task duplication to minimize fallibility. Our experimental evaluation shows that, with a small increase in energy consumption or a small decrease in performance, we can have designs with zero fallibility (i.e., all tasks are duplicated). In addition, by employing voltage scaling, we reduce the energy consumption over 50% as compared to the case where no voltage scaling is used.

The remaining of this paper is organized as follows. In the next section, we review related work. In Section 3, we explain performance, energy, and fallibility metrics and describe our task graph model and our target architecture. In Section 3, we give our ILP formulation and objective functions. We discuss the experimental results in Section 4. Finally, we conclude the paper in Section 5.

2 Related Work

Dynamic Voltage Scaling (DVS) has been a promising energy reduction technique since Weiser et al. [2] introduced the idea to the community. Exploiting the availability of the processors that run on multiple voltages, most of the prior work focused on using voltage scaling as the primary energy reduction technique. While some researchers used ILP for voltage selection [3], others employed heuristic approaches to find the solutions faster [4]. However, such studies do not take reliability related issues into account. There are also several research efforts at the system level focusing on reliability-aware high-level synthesis [5] and hardware/ software co-design [6], [7]. In [7], for example, Xie et al. employs duplication to improve the reliability of the design. However, they do not consider the energy consumption introduced by adding new tasks to the system. We reported the

preliminary version of this work in [8]. Our work differs from the prior studies in at least two aspects. First, we embed the energy consumption and the fallibility (reliability) metrics at the same time into the scheduling problem unlike above mentioned efforts. Second, our approach is flexible in that it can optimize any of several objective metrics, such as the fallibility, energy consumption, and/or performance, based on a set of constraints. To the best of our knowledge, this is the first study that integrates all these three important parameters in a framework that targets heterogeneous chip multiprocessors.

3 Preliminaries

Dynamic voltage scaling (DVS) is one of the most commonly used techniques to reduce the energy consumption of the tasks in the design. Its attraction comes from the fact that tasks consume less energy (proportional to the square of the supply voltage) when they run under low supply voltages. However, the drawback is that they also take more time (proportional to the supply voltage) to execute. When a task executes on a lower voltage, its WCET increases at a rate which is proportional to the following expression:

$$t = \frac{C_L . v}{k.(v - v_t)^{\alpha}}, \tag{1}$$

where C_L is the output load capacitance of the circuit, k is a constant that depends on the process technology employed and gate size, v_t is the threshold voltage, and α is a variable that ranges between 1.2 and 2. In this paper, we assume α is 2 and v_t is 0.6V for our numerical evaluations. Consequently, if the WCET and energy consumption of a task when using a high voltage v_h is known, its corresponding WCET and energy consumption on a lower voltage v_j can be determined by the following expressions:

$$t_{v_l} = t_{v_h} . \frac{v_l}{v_h} . \left(\frac{v_h - v_t}{v_l - v_t} \right)^2 \quad \text{and} \quad E_{v_l} = E_{v_h} . \left(\frac{v_l}{v_h} \right)^2 \tag{2}$$

In this work, we assume that we know the worst case execution times and energy consumptions of the tasks when using the highest voltage/frequency level available for each processor in our HCM.

Our next concern in this paper is to minimize the fallibility (maximize the reliability) of the design. To do this, we introduce duplication to the task graph. The duplicated tasks in our designs can be the exact copy of the primary tasks or the scaled down versions of them (as a result of voltage scaling). When a task is duplicated, we also add a checker circuitry to check the correctness of the computation. As our reliability metric, we use the percentage of duplicated tasks in the design. The fallibility of the design is the opposite of the reliability value, which is the number of tasks that are not duplicated.

We use a *task graph* to represent the given embedded application. Task graph is a directed-acyclic graph where vertices (nodes) represent the tasks and edges represent

the dependencies among these tasks. An example task graph is given in Figure 1(a). For a given task graph, we add source (t_0) and sink (t_{n-1}) tasks to make our scheduling formulations simpler. In Figure 1(a), tasks 0 and 4 are the source and sink tasks, respectively. The deadline is the maximum allowed time that a task should finish its execution. While each task in the graph may have its own deadline, in this work, we use a single deadline for the entire task graph, though our approach can be extended to handle the case where multiple nodes have their own deadlines. A fully duplicated task graph example is shown in Figure 1(b).

The target architecture has an important role in our framework. Its configuration affects the resulting design in many aspects such as performance, cost, and energy consumption. In our target architecture, each processor has its own memory and the communication between processors (tasks) is conducted via shared bus. In fact, the mapping of each task onto a processor may result in different memory area consumption and communication overhead. These problems (minimizing the memory cost and communication overhead) have been studied extensively in the literature, and they are beyond the scope of this work. In Figure 1(c), we illustrate a possible mapping of fully duplicated task graph (shown in Figure 1(b)) onto our target architecture. In this figure, we assume that we have only two processors, each differing in clock speed/frequency, energy/power consumption, and memory consumption (cost). As can be seen from this figure, we may map the primary task and its duplicate onto the same processor or onto the different processors to minimize our objective function. However, mapping the primary task and its duplicate onto the same processor may cause a problem if that processor (specifically, its data path) is faulty. This is because; the results of a task and its duplicate on a faulty processor will be the same and faulty. However, these faulty results will be passed as correct results, since both the faulty results will be the same as far as the checker circuitry is concerned. This can be avoided by forcing the duplicate of a task to be executed on a different processor, which brings extra energy consumption since the duplicate of the task will not be mapped to its best alternative.

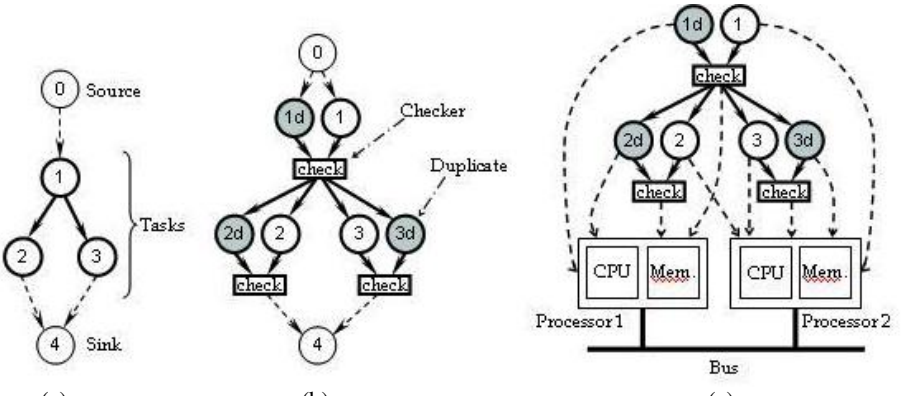

Fig. 1. (a) An example task graph, (b) its fully duplicated version, (c) a possible mapping of fully duplicated task graph onto the target architecture with two processors

4 Task Scheduling

In this section, we give our ILP formulation for the task scheduling problem. First, we define a binary variable $b_{i,j,v}$, which indicates whether task i is assigned to processor j that runs under voltage level v. The duplicates may not necessarily be assigned to a processor since we may not know if a node is duplicated or not. The duplication criterion depends on the energy, fallibility, and/or performance constraints. Expressions (3) and (4) capture this constraint. In these expressions, n, m, and o represent the number of tasks, processors, and voltage levels, respectively.

$$\sum_{j=1}^{m}\sum_{v=1}^{o} b_{i,j,v} = 1 \text{ , } \forall i : 0 \le i < n \tag{3}$$

$$\sum_{j=1}^{m}\sum_{v=1}^{o} b_{i,j,v} \le 1 \text{ , } \forall i : n \le i < 2n-2 \tag{4}$$

Next, we need to assign the most appropriate execution time d_i and energy consumption E_i values to the tasks and their duplicates. For each task, we have several alternatives for both the execution time and energy consumption values since these values can vary depending on the processor used. Additionally, these values will be different for each voltage level on a processor. We then assign these time and energy values to each task using the expressions below. In these expressions, $t_{i,j,v}$ and $E_{i,j,v}$ are the WCET and energy consumption, respectively, of task i on processor j that runs under voltage level v.

$$\forall i : \quad d_i = \sum_{j=1}^{m}\sum_{v=1}^{o} t_{i,j,v}.b_{i,j,v} \quad \text{and} \quad E_i = \sum_{j=1}^{m}\sum_{v=1}^{o} E_{i,j,v}.b_{i,j,v} \tag{5}$$

Each task i and its duplicate must be assigned a start time s_i, an integer variable that we define as:

$$\forall i : 0 \le s_i \le l - d_i \text{ ,} \tag{6}$$

where l is the deadline and d_i is the WCET of task i.

In order to ensure the correct execution order for the tasks, the sequencing constraints must be satisfied. This means that a task can start its execution only after its predecessors have finished their executions (i.e., no task can start its execution unless all its input data are available). To express the sequencing constraints, we need to know when a task finishes its execution. As we mentioned earlier, some of the tasks may be duplicated. After each task and its duplicate, we need to insert a checker circuitry (an additional node in the task graph), which introduces some extra delay to the design. As a result, we should add the delay of the checker after the task and its duplicate complete their executions. To do this, we need to know which one finishes its execution later so that we add the delay of the checker circuitry to this task. Consequently, we define a binary variable $c_{i,k}$, which is set 1 when task i finishes its execution later than its duplicate k (k=i+n-1). Expressions (7), (8), and (9) below are

used to capture this constraint, where et_i is the finish time of a task, disregarding the checker circuitry:

$$\forall i:\ et_i = s_i + d_i\ , \tag{7}$$

$$et_i - et_k \le (l+1).c_{i,k} - 1\ , \quad \forall i: 0 \le i < n \tag{8}$$

$$et_k - et_i \le l - (l+1).c_{i,k}\ , \quad \forall i: 0 \le i < n \tag{9}$$

We then define a binary variable r_i, which is set to one if task i is duplicated:

$$r_i = \sum_{j=1}^{m}\sum_{v=1}^{o} b_{i,j,v}\ , \forall i: n \le i < 2n-2 \tag{10}$$

We combine binary variables $c_{i,k}$ and r_i from Expressions (8), (9), and (10) to obtain binary variable ch_i, given in Expression (11), which is set to 1 if task i is duplicated and it finishes its execution later than its duplicate. We then add the checker delay t_{ch} to the end time et_i of the task i to find the exact end time e_i of the task after checker insertion, if the task is duplicated.

$$ch_i \ge c_{i,k} + r_i - 1\ ,\ \forall i: 0 \le i < n \tag{11}$$

$$e_i = et_i + t_{ch}.ch_i\ ,\ \forall i: 0 \le i < n \tag{12}$$

$$e_i = et_i + t_{ch}.(1 - c_{i-n+1,i})\ ,\ \forall i: n \le i < 2n-2 \tag{13}$$

Expressions (14) and (15) below ensure that the tasks and their duplicates start execution after their predecessor and predecessors' duplicates complete. In these expressions, *Edge* represents the edge set, where the existence of an edge (i, k) in this set means task k is a direct successor of task i.

$$s_k \ge e_i \wedge s_k \ge e_{i+n-1}\ , \tag{14}$$

$$s_{k+n-1} + l - l.r_k \ge e_i \wedge s_{k+n-1} + l - l.r_k \ge e_{i+n-1}\ , \quad \forall i,k: 0 < i,k < n; (i,k) \in Edge \tag{15}$$

An additional timing constraint states that there cannot be two tasks on the same processor with overlapping time frames. We formulate this constraint by adopting an approach similar to that used in [9]. Specifically, we define a binary variable $a_{i,k}$, which is set to one if tasks i and k are scheduled on the same processor. We use the following expression:

$$\forall i,k,j \text{ such that } i > j\ ; \quad a_{i,k} \ge \sum_{v=1}^{o} b_{i,j,v} + \sum_{v=1}^{o} b_{k,j,v} - 1\ . \tag{16}$$

If task i starts its execution earlier than task k, then binary variable $f_{i,k}$ is set to 1:

$$\forall i,k,j \text{ such that } i > j\ ; \quad s_k - s_i \le (l+1).f_{i,k} - 1\ , \tag{17}$$

$$\forall i,k,j \text{ such that } i > j\ ; \quad s_i - s_k \le l - (l+1).f_{i,k}\ , \tag{18}$$

Finally, we adopt the following expressions from [9], which ensure that there will be no two tasks that are mapped onto the same processor and their execution times overlap with each other.

$$\forall i,k,j \text{ such that } i > j \qquad s_k \geq et_i - (3 - f_{i,k} - 2a_{i,k}).l\,, \tag{19}$$

$$\forall i,k,j \text{ such that } i > j \qquad s_i \geq et_k - (2 - f_{i,k} + 2a_{i,k}).l\,, \tag{20}$$

As we explained in Section 2, the duplicate of a task may be forced to be mapped onto a different processor. The following expression captures this constraint.

$$\forall i,j: \quad \sum_{v=1}^{o} (b_{i,j,v} + b_{n-1+i,j,v}) \leq 1 \tag{21}$$

Expressions (22), (23), and (24) below give the fallibility, performance, and energy consumption constraints of the design respectively.

$$F_s \geq n - 2 - \sum_{i=n}^{2n-2} \sum_{j=1}^{m} \sum_{v=1}^{o} b_{i,j,v} \tag{22}$$

$$s_{n-1} \leq l \tag{23}$$

$$TE = \sum_{i=1}^{2n} E_i \leq E\max \tag{24}$$

In this paper, we focus on three different objective functions:

Minimize energy under fallibility (F_s) and performance (l) constraints.

$$MIN\text{: } TE = \sum_{i=1}^{2n} E_i \text{ under Expressions (22) and (23).}$$

Minimize fallibility (i.e., maximize the number of duplicated tasks) under performance (l) and energy ($Emax$) constraints.

$$MIN\text{: } n - 2 - \sum_{i=n}^{2n-2} \sum_{j=1}^{m} \sum_{v=1}^{o} b_{i,j,v} \text{ under Expressions (23) and (24).}$$

Minimize execution time of the task graph (i.e., maximize performance) under energy and fallibility constraints.

$$MIN\text{: } s_{n-1} \text{ under Expressions (22) and (23).}$$

5 Experimental Results

In this section, we test the impact of our approach through two sets of experiments, each having a different objective function and a different set of constraints to be

satisfied. In our experiments, we use task graphs extracted from embedded appli cations; namely, G721decode and G721encode from Mediabench, Mismatch_test from Trimaran GUI, and a custom task graph tg01. Since a uniprocessor system is a special case of an HCM, we consider HCMs from two to five heterogeneous processors in our experiments. For each processor, we employ five different levels of supply voltages. These voltage levels range from 2.0V to 3.3V (specifically, 2.0V, 2.4V, 2.7V, 3.0V, and 3.3V). We tested our approach on randomly generated task graphs as well. However, due to lack of space, we report only one of the experiments (Experiment 2) here.

Experiment 1: In our first set of experiments, we study the impact of voltage scaling on energy reduction. We also show how fallibility affects the energy improvement brought by voltage scaling. To do this, we scheduled three benchmarks (G721encode, G721decode, and Mismatch_test) under a fixed performance constraint. We assume that we have three processors and three voltage levels (2.0V, 2.7V, and 3.3V) in our target architecture. Then, we change the percentage of duplicated tasks from 0% to 100%. Note that under given performance constraints, we may not be able to fully duplicate the given task graph.

In Figures 2(a), 2(b), and 2(c), we give the comparisons of energy consumptions with and without dynamic voltage scaling (DVS) under a fixed performance cons traint for G721encode, G721decode, and Mismatch_test benchmarks, respectively. In Figure 2(d), we summarize the percentage energy improvements brought by our approach over a scheme that does not use DVS. As can be seen from this bar-chart, when we increase the number of duplicated tasks in the design the energy improvement reduces. This is because when we add more duplicated task to the design, we may not scale down all the tasks to meet the performance bound, resulting in less energy improvement (i.e., our energy consumption will be closer to the energy consumption of the scheme without DVS when we decrease fallibility).

Experiment 2: In this set of experiments, we schedule the task graph tg01 under different performance and energy bounds, with the objective of minimizing fallibility. For the target architecture, we assume that we have three processors that can use three different supply voltages. In Figure 3, we show how the percentage of duplicated tasks changes as a function of the different energy and performance bounds. As can be seen from this bar-chart, a very small energy and/or performance bound relaxation allows us to duplicate more tasks. Beyond certain performance and energy bounds, however, increasing the bounds further does not bring any additional improvements on fallibility since the application reaches its maximum energy savings under the given deadline, and the fallibility is minimized to zero (i.e., the task graph is 100% duplicated).

Our experiments show how the proposed ILP-based approach can be used effectively for scheduling a real-time embedded application on an HCM under several constraints, and for optimizing the desired cost function. It can be observed that our approach is flexible in the sense that it can be employed for several scheduling problems from simple schedules that do not consider fault-tolerance and/or power to very complex schedules that involve all three metrics in some fashion.

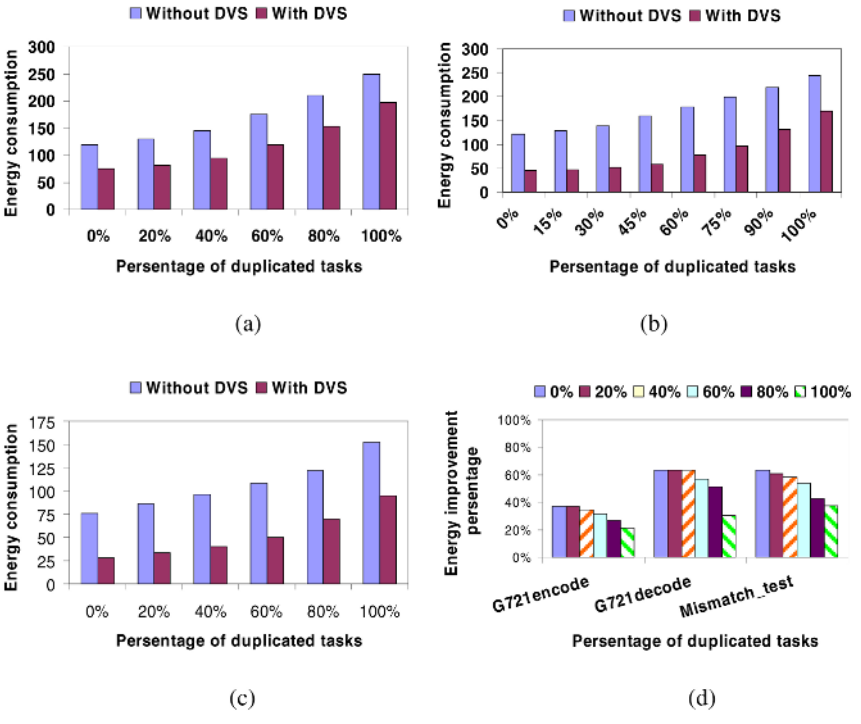

Fig. 2. Energy consumptions under different percentage of duplicated tasks with and without DVS for (a) G721encode, (b) G721decode, and (c) Mismatch_test benchmarks, and (d) the percentage energy improvements brought by our approach

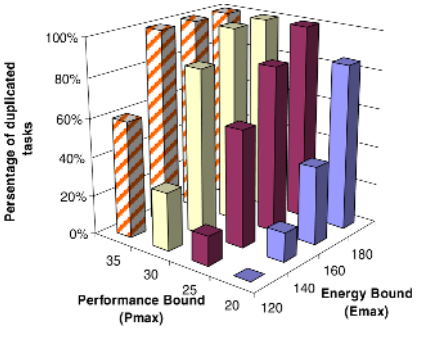

Fig. 3. Maximum percentage values of duplicated tasks with respect to energy and performance bounds for tg01

One of the concerns for ILP based approaches is the solution time required for the given problem, depending highly on its complexity. When the complexity increases, producing an output using an ILP can become too time-consuming, sometimes even impossible in a reasonable amount of time. In our experiments, the CPU time of the

solutions ranges from one (1) second to 140 minutes. However, most of the solution times were less than 10 minutes. In fact, we calculated the average CPU time for all the experiments as 15 minutes. We believe that these solution times are not excessive. Moreover, in our experiments, we use a UNIX machine with a sparcv9 processor operates at 360 MHz, and its memory size is 128 MegaBytes. Using a faster processor and a better solver can further reduce these solution times.

6 Conclusions

In this paper, we presented an ILP (Integer Linear Programming) based framework that maps a given application (represented as a task graph) onto an HCM (Heterogeneous Chip Multiprocessor) architecture. We showed how our framework can be used in conjunction with several objective functions, namely, energy, performance, and fallibility. We illustrated the effectiveness of our approach on several benchmarks.

References

1. M. Berkelaar, K. Eikland, and P. Notebaert, “lp_solve: Open source (Mixed-Integer) Linear Programming system”, Version 5.0.0.0. dated 1 May 2004.
2. M. Weiser, B. Welch, A. Demers, and S. Shenker, “Scheduling for reduced CPU energy”, In Proceedings of the 1st Symposium on Operating Systems Design and Implementation, 1994.
3. T. Ishihara and H. Yasuura, “Voltage scheduling problem for dynamically variable voltage processors”, Proc. of the 1998 International Symposium on Low Power Electronics and Design, 1998.
4. J. Luo and N. K. Jha, “Power-Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-Time Embedded Systems”, ICCAD, 2000.
5. S. Tosun, N. Mansouri, E. Arvas, M. Kandemir, and Y. Xie, “Reliability-Centric High-Level Synthesis”, Proceedings of the Design, Automation and Test in Europe (DATE'05), 2005.
6. C. Bolchini, L.Pomante, F.Salice and D.Sciuto, “A System Level Approach in Design Dual-Duplex Fault Tolerant Embedded Systems,” Online Testing Workshop, 2002.
7. Y. Xie, L. Li, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Reliability-Aware Co-synthesis for Embedded Systems", in Proceedings of IEEE 15th International Conference on Application-specific Systems, Architectures and Processors (ASAP'04), pp. 41-50, September 2004.
8. S. Tosun, N. Mansouri, M. Kandemir, and O. Ozturk, “Constraint-Based Code Mapping for Heterogeneous Chip Multiprocessors”, *IEEE International SOC Conference (SOCC 2005)*, Washington, D.C., September 2005.
9. S. Prakash and A. C. Parker, “SOS: Synthesis of Application-Specific Heterogeneous Multiprocessor Systems", Journal of Parallel and Distributed Computing, December 1992, Vol. 16, pp. 338-351.