Compiler-Directed Energy Reduction Using Dynamic Voltage Scaling and Voltage Islands for Embedded Systems

Ozcan Ozturk, *Member*, *IEEE*, Mahmut Kandemir, *Member*, *IEEE*, and Guangyu Chen, *Member*, *IEEE*

Abstract—Addressing power and energy consumption related issues early in the system design flow ensures good design and minimizes iterations for faster turnaround time. In particular, optimizations at software level, e.g., those supported by compilers, are very important for minimizing energy consumption of embedded applications. Recent research demonstrates that voltage islands provide the flexibility to reduce power by selectively shutting down the different regions of the chip and/or running the select parts of the chip at different voltage/frequency levels. As against most of the prior work on voltage islands that mainly focused on the architecture design and IP placement related issues, this paper studies the necessary software compiler support for voltage islands. Specifically, we focus on an embedded multiprocessor architecture that supports both voltage islands and control domains within these islands, and determine how an optimizing compiler can automatically map an embedded application onto this architecture. Such an automated support is critical since it is unrealistic to expect an application programmer to reach a good mapping correlating multiple factors such as performance and energy at the same time. Our experiments with the proposed compiler support show that our approach is very effective in reducing energy consumption. The experiments also show that the energy savings we achieve are consistent across a wide range of values of our major simulation parameters.

Index Terms—Voltage islands, compiler optimizations, energy consumption, voltage scaling, compiler-based parallelization

1 Introduction

Power and energy related issues in deep submicron embedded designs may limit functionality, reliability, and performance and severely affect yield and manufacturability. It is well known that higher power dissipation increases junction temperatures, which in turn slows down transistors and increases interconnect resistance. Therefore, power consumption needs to be considered as one of the primary metrics in embedded system design, and any optimization approach targeted at improving performance may therefore fall short if power is not also taken into account.

Recent years have witnessed several efforts aimed at reducing power consumption from both hardware and software perspectives. One such hardware approach is voltage islands, which are areas (logic and/or memory) supplied through separate, dedicated power feed. The prior work on voltage islands so far generally focused on the design and placement issues, and will be discussed in detail in Section 2. Our goal in this paper is to study the necessary software compiler support for voltage islands. Specifically, we focus on an embedded multiprocessor architecture that supports both voltage islands and control domains within these islands and determine how an optimizing compiler can map an embedded application onto this architecture. The specific types of applications this paper considers are embedded multimedia codes that are built from multidimensional arrays of signals and multiloop nests that operate on these arrays. One of the nice characteristics of these applications is that an optimizing compiler can analyze their data access patterns at compile time and restructure them based on the target optimization in mind (e.g., enhancing iteration-level parallelism or improving data locality).

We first give, in Section 3, a characterization of a set of embedded applications that illustrates the potential benefits that could be obtained from a voltage island based embedded multiprocessor architecture. Based on this characterization, we then present, in Section 4, a compiler-directed code parallelization scheme, which is the main contribution of this paper. A unique characteristic of this scheme is that it minimizes power consumption (both dynamic and leakage) by exploiting both task and data parallelism. We tested the impact of this approach using a suite of eight embedded multimedia applications and a simulation environment. Our experiments, discussed in Section 5, reveal that the proposed parallelization strategy is very effective in reducing power (40.7 percent energy savings on the average) as well as execution cycles (14.6 percent performance improvement on the average). The experiments also show that the power savings we achieve are consistent across a wide range of values of our simulation parameters. For example, we found that our approach scales very well as we increase the number of processor cores in the architecture and the number of

- *O. Ozturk is with the Department of Computer Engineering, Bilkent University, Ankara, Turkey. E-mail: ozturk@cs.bilkent.edu.tr.*
- *M. Kandemir is with the Computer Science and Engineering Department, The Pennsylvania State University, 111 IST Building, University Park, PA 16802. E-mail: kandemir@cse.psu.edu.*
- *G. Chen is with Facebook, 156 University Ave., Palo Alto, CA.*

Manuscript received 28 Aug. 2010; revised 20 Oct. 2011; accepted 26 Oct. 2011; published online 29 Nov. 2011.
Recommended for acceptance by A. Zomaya.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2010-08-0479.
Digital Object Identifier no. 10.1109/TC.2011.229.

voltage islands. Our results also indicate that, for the best energy savings, both data and task parallelism need to be employed together and application mapping should be performed very carefully. Overall, our results show that automated compiler support can be very effective in exploiting unique features of a voltage island based multiprocessor architecture.

2 Related Work

As power consumption and heat dissipation are becoming increasingly important issues in chip design, major chip manufacturers, such as IBM [3] and Intel [4], are adopting voltage islands in their current and future products [2]. For example, voltage islands will be used in IBM's new CU-08 manufacturing process for application-specific integrated processors (ASIPs) [3]. The chip design tools that support voltage islands are also starting to appear in the market (e.g., [1]).

Different approaches for adapting and using voltage islands have been explored [43], [23], [34]. Specifically, Lackey et al. [19] discuss the methods and design tools that are being used today to design voltage island based architectures. Hu et al. [14] present an algorithm for simultaneous voltage island partitioning, voltage level assignment, and physical-level floor planning. In [27], authors discuss the problem of energy optimal local speed and voltage selection in frequency/voltage island based systems under given performance constraints. Liu et al. [22] propose a method to reduce the total power under timing constraints and to implement voltage islands with minimal overheads. Wu et al. [42] implement a methodology to exploit nontrivial voltage island boundaries. They evaluate the optimal power versus design cost tradeoff under performance requirements. In [7], authors explore a semi-custom voltage-island approach based on internal regulation and selective custom design. Giefers and Rettberg [12] propose a technique that partitions the design into different frequency/voltage islands during the scheduling phase of the High-Level Synthesis (HLS). Our approach is different from all these prior efforts on voltage islands as we focus on automated compiler support for such architectures, with the goal of reducing energy consumption.

An important advantage of chip multiprocessors is that it is able to reduce the cost from both performance and power perspectives. The prior work [5], [6], [11], [17], [18], [26], [28], [30], [39] discusses several advantages of these architectures over complex single-processor-based designs. Besides voltage island based systems, there are many prior efforts that target at reducing energy consumption of MPSoC-based architectures and chip multiprocessors [15]. For example, Ozturk et al. [29] propose an energy-efficient on-chip memory design for embedded chip multiprocessor systems. Manolache et al. [25] present a fault and energy-aware communication mapping strategy for applications implemented on NoCs. Soteriou and Peh [38] explore the design space for communication channel turn-on/off based a dynamic power management technique for both on-chip and off-chip interconnections. Yang et al. [44] present an approximate algorithm for energy efficient scheduling. Rae and Parameswaran [31] study voltage reduction for power minimization. Shang et al. [35] propose applying dynamic voltage scaling to communication channels.

There has been various compiler-based approaches to voltage scaling. Chen et al. [10] propose a compiler-directed approach where the compiler decides the appropriate voltage/frequency levels to be used for each communication channel in the NoC. Their approach builds and operates on a graph-based representation of a parallel program. In [20], authors propose a compiler-based communication link voltage management technique. They specifically extract the data communication pattern among parallel processors along with network topology to set the voltages accordingly. In [36], authors propose a real-time loop scheduling technique using dynamic voltage scaling. They implement two different scheduling algorithm based on a directed acyclic graph (DAG) using voltage scaling. Shi et al. [37] present a framework for embedded processors using a dynamic compiler. Their compiler-based approach specifically utilizes the OS-level information and hardware status. Rangasamy et al. [32] propose a petri net-based performance model where compiler is used to set the frequencies. Jejurikar and Gupta [16] present a DVS technique that focuses on minimizing the entire system energy consumption. In addition to the leakage energy, authors also consider the energy consumption of the components like memory and network interfaces. Hsu et al. [13] present a compiler-based system to identify memory-bound loops. Authors reduce the voltage level for such loops since the memory subsystem is much slower than the processor. Our work is different from these compiler-based studies since our scheme minimizes dynamic and leakage energy consumption for voltage islands by exploiting both task and data parallelism at the loop level.

3 Load Imbalance in Multimedia Applications

In this section, we focus on a set of multimedia applications and study the opportunities for saving energy through voltage scaling. Fig. 1 shows load imbalances across eight processors for the first five loop nests of some of our multimedia applications (we will discuss the important characteristics of these applications later). These results were obtained by parallelizing the loop nests of the applications on a uniform multiprocessor architecture (simulated using the SIMICS tool-set [24]), i.e., all the processors are the same and operate under the highest clock frequency and voltage levels available. Each bar in this figure corresponds to the normalized completion time of the workload of that processor in the corresponding loop nest, assuming that the time of the latest processor, i.e., the one finishes last, is set to 100 for ease of observing the load imbalance trends across the applications. We see that, for all the applications and all their loop nests, there is a significant load imbalance among the parallel processors. There are several factors that contribute to this load imbalance, which we explain next.

First, sometimes, the upper bound or lower bound of an inner loop in a nest depends on the index of the outer loop. This situation is illustrated by the following loop nest written in a pseudolanguage:

```
for $I := 1, N$
for $J := I, N$
{...}.
```

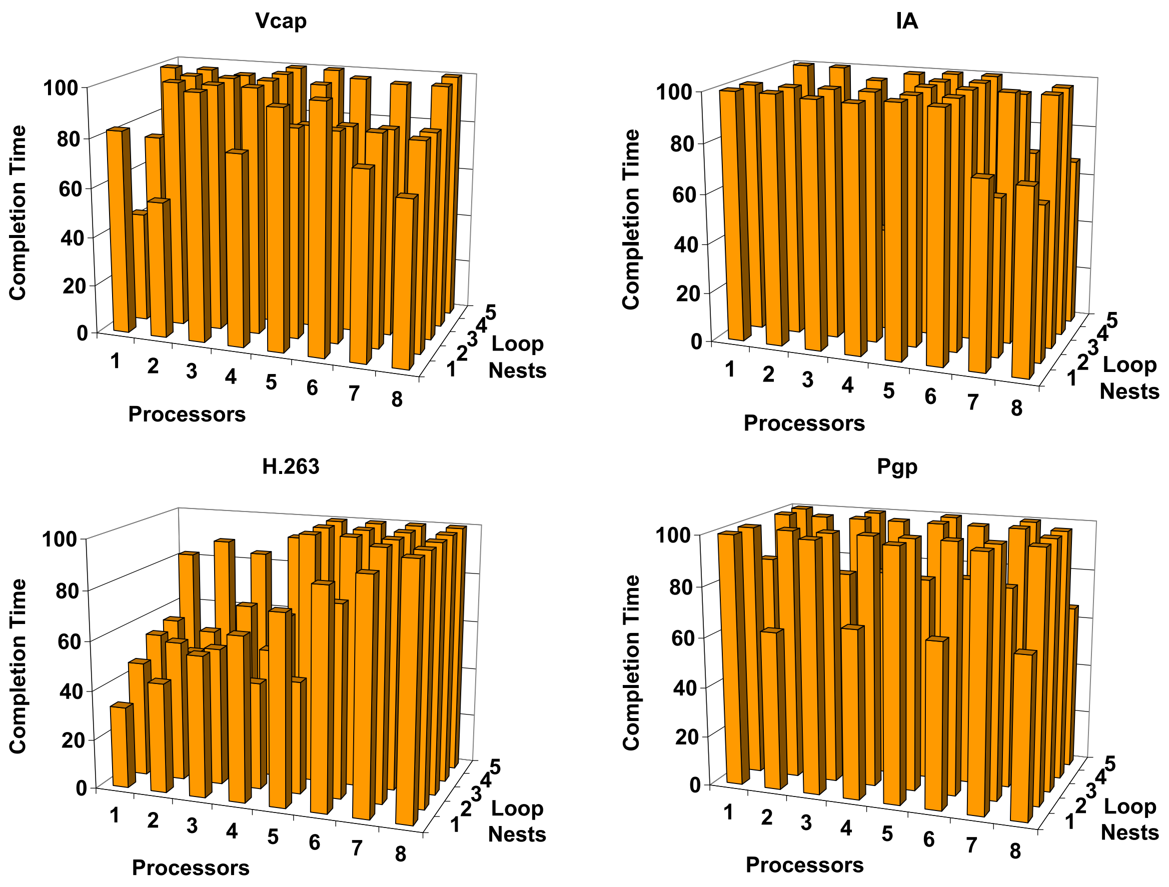


Fig. 1. Load imbalance across eight processors for the first five loop nests of some of our multimedia applications.

In this loop nest, assuming that the outer loop is parallelized over multiple processors and the inner loop is run sequentially, it is easy to see that each processor will execute a different number of loop iterations. This is because the number of loop iterations that will be executed by the inner loop (J) of each processor is different since the lower bound of J depends on I. The second possible reason for the load imbalance among the processors is the different data cache behavior of the different processors. Since each processor can experience a different number of data cache hits and misses, this can cause load imbalance across them. A third possible reason for load imbalance is the conditional constructs such as IF statements within the bodies of the parallelized loop nests. If two different processors happen to take the different branches of an IF statement in the loop iterations they execute, their execution times (i.e., the time it takes for a processor to complete its workload in that loop nest) can be different from each other. Fig. 2 gives the contribution of these three factors to the overall load imbalance for each of our applications. The segment marked "Other" in each bar shown in this figure represents the remaining load imbalances in the corresponding application whose source we could not identify. We see from these results that the loop bound based imbalance, the first reason discussed above, dominates the remaining factors for all eight applications. This, in a sense, is good news from the compiler's perspective, as this is the only type of load imbalance, among those mentioned above, that we can identify at compile time and try to eliminate as much as possible; the remaining causes of load imbalance are very difficult to capture and characterize at compile time. The next section discusses such a compiler approach to exploit the load imbalances across processors in the context of a voltage island based embedded multiprocessor architecture.

4 Compiler-Directed Application Code Mapping

4.1 Architecture Abstraction

A high-level view of the embedded architecture considered in this paper is shown in Fig. 3. In this architecture, the chip area is divided into multiple voltage islands, each of which is

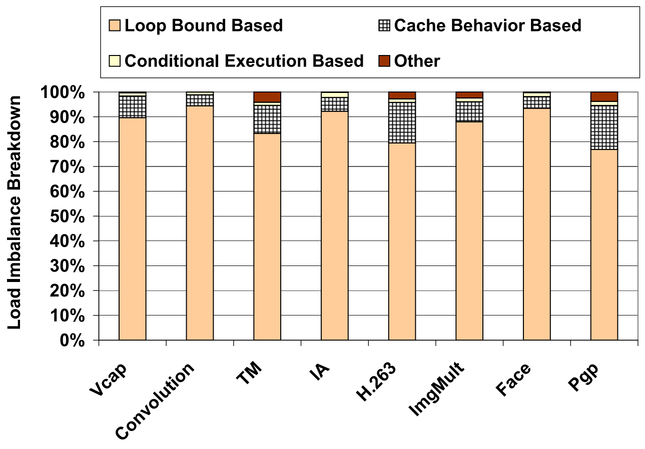


Fig. 2. Breakdown of the load imbalances in our applications.

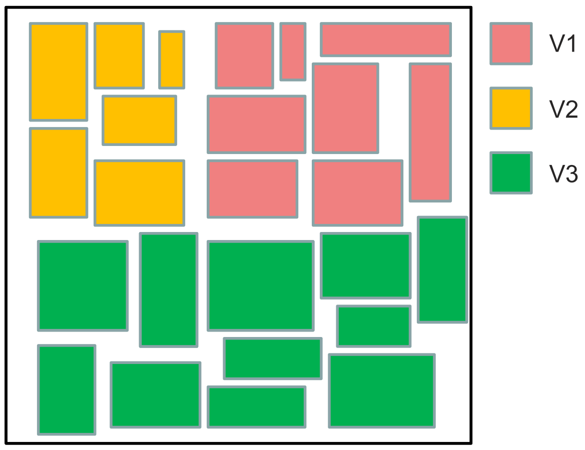


Fig. 3. An example voltage island based architecture, with three islands. We assume that there exists a large off-chip memory, shared by all processors.

controlled by a separate power feed and operates under a different voltage level/frequency. We further assume that each voltage island is divided into multiple power domains. All the domains within an island are fed by same V_{dd} source but independently controlled through intraisland switches. To implement power domains, a power isolation logic ensures that all inputs to the active power domain are clamped to a stable value. The important point to note is that this island-based architecture can help save both dynamic and leakage power. Specifically, it can save dynamic energy by employing different voltage levels for the different islands, and leakage energy by shutting down the power domains that are not needed by the current computation. The difficult task however is to decide how a given embedded multimedia application needs to be mapped to this multiprocessor architecture, i.e., code parallelization in this island-based architecture, which is discussed in the rest of our paper.

4.2 Mapping Algorithm

In order to map a given multimedia code to the architecture shown in Fig. 3, our approach uses both *data parallelism* and *task parallelism*. Data parallelism involves performing a similar computation on many data objects simultaneously. In our approach, this corresponds to a group of processors executing a given loop nest in parallel. All the processors execute a similar code (i.e., the same loop body) but work on the different parts of array data, i.e., they execute different iterations of the loop. Task parallelism, in comparison, involves performing different tasks in parallel, where a task is an arbitrary sequence of computations. In our approach, this type of parallelism represents executing different loop nests in different processors at the same time. Our compiler uses a structure called the *Loop Dependence Graph* (LDG) to represent the application code being optimized. Each node, N_i, of this graph corresponds to a loop nest in the application and there is a directed edge from node N_i to N_j if the loop nest represented by the latter is dependent on the loop nest represented by the former. The proposed compiler support maps this application onto our voltage island based architecture. In the rest of this section, we describe the details of three different voltage island and power domain aware code mapping (parallelization) schemes that map a given LDG onto our architecture.

4.2.1 The EA_DP Scheme

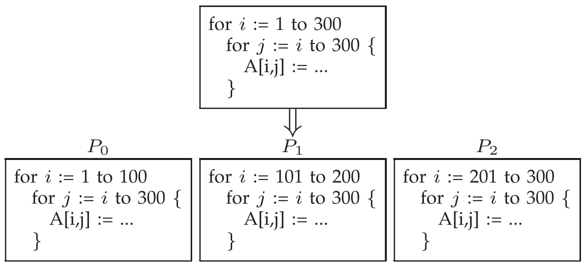
The first scheme that we describe, referred to as EA_DP, exploits only data parallelism. It proceeds in three steps, as suggested by its pseudocode given in Algorithm 1. The first step is the parallelization step. In this step, the compiler parallelizes an application in a loop nest basis. That is, each loop nest of the given LDG is parallelized independently considering the intrinsic data dependencies it has. Since we are targeting a chip multiprocessor, our parallelization strategy tries to achieve for each nest the outer loop parallelism to the extent allowed by the data dependencies exhibited by the loop nests [21]. The second step is the processor workload estimation step. In this step, the compiler estimates the load of each processor in each nest. To do this, it performs two calculations: 1) iteration count estimation and 2) per-iteration cost estimation. Since in most array-based applications bounds of loops are known before execution starts or they can be estimated through profiling, estimating the iteration count for each loop nest is not very difficult. The challenge is in determining the cost, in terms of execution cycles, of a single iteration of a given loop nest. Note that, various Worst Case Execution Time (WCET) calculation methods have been explored in literature [40]. Since the processors employed in our chip multiprocessor are simple single-issue embedded cores, the cost computation is closely dependent on the number and types of the assembly instructions that will be generated for the loop body. Specifically, we associate a base execution cost with each type of assembly instruction. In addition, we also estimate the number of cache misses. Since loop-based embedded applications exhibit very good instruction locality, we focus on data cache only and estimate data cache misses using the method proposed by Carr et al. [8]. An important issue is to estimate, at the source level, what assembly instructions will be generated for the loop body in question. We address this problem as follows. The constructs that are vital to the studied group of codes, that is, array-based multimedia applications, include a typical loop, a nested loop, assignment statements, array references, and scalar variable references within and outside loops. Our objective is to estimate the number of assembly instructions of each type associated with the actual execution of these constructs. To achieve this, the assembly equivalents of several codes were obtained using our back-end compiler (a variant of gcc) with the O3-level optimization flag. Next, the portions of the assembly code were correlated with corresponding high-level constructs to extract the number and type of each instruction associated with the construct.

Algorithm 1. EA_DP
N_L : Number of loop nests
N_P : Number of processors
L_i : Loop nest i
P_i : Processor i
SP_i : Keeps the IDs of processors (based on a sorted workload)
W_i : Workload of processor i
T_i : Execution time of processor i
V_{max} : Highest voltage level
T_{max} : Execution time of SP_1 (maximum workload)

```
1:  i = 1
2:  while $i \leq N_L$ do
3:      Parallelize loop nest $L_i$ among the $N_P$ processors
4:      i++
5:  end while
6:  i = 1
7:  while $i \leq N_P$ do
8:      $IterCount_i$ = Estimate the number of iterations
        executed by $P_i$
9:      $IterCost_i$ = Estimate the average cost per iteration
        executed by $P_i$
10:     Estimated Workload for $P_i$ is, $W_i = IterCount_i \times IterCost_i$
11:     i++
12: end while
13: Sort the processors in non-increasing order of their
    workloads ($SP_1 \ldots SP_{N_P}$)
14: Assign the highest voltage level $V_{max}$ to $SP_1$
15: $T_{max}$ = Time to execute the workload of $SP_1$
16: i = 2
17: while $i \leq N_P$ do
18:     Select lowest voltage level $V_L$ for $SP_i$ such that
        $T_i \leq T_{max}$
19:     i++
20: end while
```

To illustrate our parameter extraction process in more detail, let us focus on some specifics of the following sample constructs. First, let us focus on a loop construct. Each loop construct is modeled to have a one-time overhead to load the loop index variable into a register and initialize it. Each loop also has an index comparison and an index increment (or decrement) overhead, whose costs are proportional to the number of loop iterations (called trip count or trip). From correlating the high-level loop construct to the corresponding assembly code, each loop initialization code is estimated to execute one load (lw) and one add (add) instruction (in general). Similarly, an estimate of trip+1 load (lw), store-if-less-than (stl), and branch (bra) instructions is associated with the index variable comparison. For index variable increment (resp. decrement), $2\times$ trip addition (resp. subtraction) and trip load, store, and jump instructions are estimated to be performed. We next consider extracting the number of instructions associated with array accesses. First, the number and types of instructions required to compute the address of the element are identified. This requires the evaluation of the base address of the array and the offset provided by the subscript(s). Our current implementation considers the dimensionality of the array in question, and computes the necessary instructions for obtaining each subscript value. Computation of the subscript operations is modeled using multiple shift and addition/subtraction instructions, instead of multiplications, as this is the way our back-end compiler generates code when invoked with the O3 optimization flag. Finally, an additional load/store instruction was associated with read/write the corresponding array element.

Based on the process outlined above, the compiler estimates the iteration count for each processor and per-iteration cost. Then, by multiplying these two, it calculates the estimated workload for each processor. While this workload estimation may not be extremely accurate, it allows the compiler to rank processors according to their workloads and assign suitable voltage levels and frequencies to them as will be described in the next item. As an example, consider the code fragment shown in Fig. 4, parallelized using three processors. Assuming that our estimator estimates the cost of loop body as L instructions, the loads of processors P_0, P_1, and P_2 are $25050L$, $15050L$, and $5050L$, respectively.



```
for i := 1 to 300
  for j := i to 300 {
    A[i,j] := ...
  }
```

P_0

```
for i := 1 to 100
  for j := i to 300 {
    A[i,j] := ...
  }
```

P_1

```
for i := 101 to 200
  for j := i to 300 {
    A[i,j] := ...
  }
```

P_2

```
for i := 201 to 300
  for j := i to 300 {
    A[i,j] := ...
  }
```

Fig. 4. Data parallelization of a loop nest.

The last step that implements EA_DP is voltage and frequency assignment. In this step, the compiler first orders the processors according to their nonincreasing workloads. After that, the highest voltage is assigned to the processor with the largest workload (the objective being not to affect the execution time to the greatest extent possible). Then, the processor with the second highest workload gets assigned to the minimum voltage level V_k supported by the architecture that does not cause its execution time to exceed that of the processor with the largest workload. In this way, each processor gets the minimum voltage level to save maximum amount of power without increasing the overall parallel execution time of the nest, which is determined by the processor with the largest workload. The unused processors (and their caches) are turned off to save leakage. Note that, in EA_DP, the loop nests of the application are handled one by one. That is, observing the dependencies between the nodes of the given LDG, we process a single nest at a time. Also, this scheme uses at most one processor from each island (assuming that no two islands have the same voltage).

4.2.2 The EA_TP Scheme

We now describe how our second voltage island aware parallelization scheme, called EA_TP, operates. This scheme implements only task parallelism. In fact, it implements an algorithm similar to list scheduling on the LDG. Specifically, it can execute multiple nests in parallel if the dependencies captured in LDG allow such an execution. Suppose that we have Q loop nests that can execute in parallel. We first estimate the workload of each loop nest, using a similar procedure to the one explained in detail above. Then, each loop nest is assigned to an island and executed in a single processor there. This assignment is done considering the workloads of the processors as well as the voltage/frequency levels of the islands. It needs to be emphasized that, since in this scheme the iterations of the same loop are not executed in parallel, we exploit only task parallelism. Algorithm 2 gives the pseudocode for this scheme.

Algorithm 2. EA_TP

```
$N_L$ : Number of loop nests
$N_P$ : Number of processors
$N_P$ : Number of groups
$L_i$ : Loop nest $i$
$P_i$ : Processor $i$
$G_i$ : A group of loop nests (that can run parallel)
$WL_i$ : Workload of loop nest $i$

1: i = 1
2: while $i \leq N_L$ do
3:     $IterCount_i =$ Estimate the number of iterations executed by $L_i$
4:     $IterCost_i =$ Estimate the average cost per iteration executed by $L_i$
5:     Estimated Workload for $L_i$ is, $WL_i = IterCount_i \times IterCost_i$
6:     i++
7: end while
8: Group the loop nests using the LDG (loop dependence graph) such that loop nests in a group can be executed in parallel
9: $G = G_1 \ldots G_k$, such that $G_{i+1}$ is dependent on $G_i$
10: while $i \leq N_G$ do
11:     Assign each loop nest $L_t \in G_i$ to an island and execute in a single processor
12:     i++
13: end while
```

Let us now explain how EA_TP works using the code fragment in Fig. 5a. Fig. 5b shows the LDG for this code fragment. For illustration purposes, we assume that the number of execution cycles for loop nests L_1, L_2, L_3, and L_4 are 9000, 4000, 2000, and 1000, respectively. These four loop nests can be divided into three groups

$$G_1 = \{L_1\},$$
$$G_2 = \{L_2, L_3\},$$
$$G_3 = \{L_4\}.$$

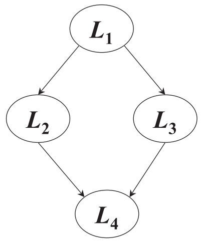
The loop nests of the same group can be executed in parallel. Since G_1 contains only one loop nest, L_1, we assign this loop nest to a processor with the highest voltage. G_2 contains loop nests L_2 and L_3, and L_2 requires twice as many execution cycles as L_3. We assign L_2 to the same processor as L_1, and L_3 to a processor with the lowest voltage such that the execution time of L_3 does not exceed that of L_2. Like G_1, G_3 contains only one loop nest (L_4), and we assign it to the same processor as L_1.

4.2.3 The EA_TDP Scheme

Our third scheme, referred to as EA_TDP, combines the task and data parallelism based approaches explained above. More specifically, like EA_TP, it first identifies, using LDG, the loop nests that can be executed in parallel, i.e., the nests that do not have dependencies among them. However, unlike EA_TP, it next calculates (estimates) the workloads at the processor granularity. It then assigns voltages to each processor, i.e., it determines where each workload is executed based on the approach explained when we discuss

```
L1: for i := 1 to 300
      for j := i to 300 {
          A[i,j] := ...
      }
L2: for i := 1 to 300
      for j := i to 300 {
          B[i,j] := ...A[i,j]...
      }
L3: for i := 1 to 300
      for j := i to 300 {
          C[i,j] := ...A[i,j]...
      }
L4: for i := 1 to 300
      for j := i to 300 {
          D[i,j] := ...B[i,j]...C[i,j]...
      }
```

(a)



(b)

Fig. 5. (a) A code fragment and (b) its LDG.

EA_DP above. Note that this approach exploits both task and data parallelism. Task parallelism is exploited since the iterations from the different loop nests are executed at the same time, that is, different loop bodies are concurrently executed. Data parallelism is executed because the iterations of the same loop are run in parallel, which is not the case in EA_TP. The pseudocode for the EA_TDP scheme is shown in Algorithm 3.

Algorithm 3. EA_TDP

```
$N_L$ : Number of loop nests
$N_P$ : Number of processors
$N_P$ : Number of groups
$L_i$ : Loop nest $i$
$P_i$ : Processor $i$
$G_i$: A group of loop nests (that can run parallel)
$SP_i$ : Sorted processor $i$ based on workload
$W_i$ : Workload of processor $i$
$T_i$ : Execution time of processor $i$
$V_{max}$ : Highest voltage level
$T_{max}$ : Execution time of $SP_1$ (maximum workload)

1: i = 1
2: Group the loop nests using the LDG (loop dependence graph) such that loop nests in a group can be executed in parallel
3: $G = G_1 \ldots G_k$, such that $G_{i+1}$ is dependent on $G_i$
4: i = 1
5: while $i \leq N_P$ do
6:     $IterCount_i =$ Estimate the number of iterations executed by $P_i$
7:     $IterCost_i =$ Estimate the average cost per iteration executed by $P_i$
8:     Estimated Workload for $P_i$ is, $W_i = IterCount_i \times IterCost_i$
9:     i++
10: end while
11: Sort (non-increasingly) the processors based on their workloads $SP_1 \ldots SP_{N_P}$
12: Assign the highest voltage level $V_{max}$ to $SP_1$
13: $T_{max} =$ Time to execute the workload of $SP_1$
```

TABLE 1
Default Simulation Configuration

Total Number of Processors	12
Number of Islands	3
Number of Power Domains/Island	2
Number of Processors/Power Domain	2
Voltage Levels for Islands	1.2V; 1V; 0.8V
Cache Capacity	8KB, 2-way associative
Processor Type	Single Issue, 5-Stage Pipeline

```
14: i = 2
15: while $i \leq N_P$ do
16:     Select lowest voltage level $V_L$ for $SP_i$ such that
        $T_i \leq T_{max}$
17: i++
18: end while
```

We now use the code fragment shown in Fig. 5a as an example to show how EA_TDP works in practice. Let us assume that we have three processors. Similar to EA_TP, we first divide the loop nests into groups $G_1 = \{L_1\}$, $G_2 = \{L_2, L_3\}$, and $G_3 = \{L_4\}$ such that the loop nests of the same group can be executed in parallel to each other. Since we have three processors and group G_1 contains only L_1, we can apply data parallelism, as shown in Fig. 4, to L_1 such that we can use three processors to execute this loop nest. For group G_2, however, we split loop nest L_2 into two loop nests, L_2' and L_2''. After that, we assign loop nests L_2', L_2'', and L_3 to the three available processors with different voltages such that the execution time of these loop nests are approximately equal to each other. Finally, we apply data parallelism to loop nest L_4 by splitting it into three parallel loop nests and assigning them to the three processors.

Note that, all our voltage island aware schemes (EA_DP, EA_TP, EA_TDP) are also power domain aware in that the unused power domains in any voltage island are turned off to save leakage energy.

5 Experiments

5.1 Setup

To test the success of our code parallelization approach for voltage island based architectures, we performed experiments with five different schemes, which can be summarized as follows. DP is a scheme that exploits only data parallelism without taking into account the voltage islands in the architecture. TP is similar to DP in that it does not consider voltage islands; however, it exploits task parallelism instead of data parallelism. Both of these approaches assign the highest voltage to the processors to prevent possible performance penalties. Note that, this is the highest voltage available in the voltage island that processor belongs to. EA_DP and EA_TP are similar to DP and TP, respectively, except that they are voltage island aware. That is, they perform code parallelization taking into account the voltage islands in the architecture. The details of these two schemes have been discussed earlier in Section 4. Finally, EA_TDP, which is also discussed in Section 4, is the scheme that employs both task and data parallelism in a voltage island aware manner. Note that the main difference between DP (resp. TP) and EA_DP (resp. EA_TP) is that, while the latter considers workloads to select the best processor, the former does not care about the different voltage/frequency levels of the islands. However, even in the schemes DP and TP, the unused power domains are turned off to save leakage.

We implemented these five schemes within an experimental compiler infrastructure [41], and performed experiments using the SIMICS infrastructure [24]. SIMICS is a simulation toolset that allows execution of unmodified binaries and can simulate a multicore architecture. We enhanced the basic SIMICS architecture to model voltage islands, using an execution-driven, cycle-accurate energy model. More specifically, for estimating energy consumption of the memory components, we used the CACTI [33] tool; for the CPU data path and interconnects, we used SimplePower [45]. SimplePower is an architectural level cycle-accurate simulator where a single-issue five-stage pipelined processor is being simulated according to the voltage levels. We also have results with out of order cores; the results were similar to those with in order cores. Performance of individual instructions are affected with the voltage level being used. The instruction set architecture used in the simulator is a subset of the instruction set of SimpleScalar. On the other hand, CACTI is an integrated cache simulator which measures access time, area, leakage, and dynamic power. Collected cache access traces are used to calculate the cache energy consumption. While it is possible to collect results for different technologies, sizes, or associativity, we use a 65 nm, 8 KB, 2-way set associative cache. The default simulation configuration is given in Table 1. We used eight benchmark codes to evaluate the five schemes explained above. The important characteristics of these benchmarks are listed in Table 3. The third column gives the total number of execution cycles of each application and the fourth column gives the energy consumption, both under the case when an application is executed only on a single processor under the highest clocking frequency and voltage supported by the architecture. We need to mention that the values in the last column include both dynamic and leakage energies, and capture the energy consumed in processors, caches, interconnects, and off-chip memory. Processor characteristics with varying voltage levels are shown in Table 2.

TABLE 2
Processor Characteristics with Different Voltage Levels

Voltage	Frequency	Dynamic Energy/Cycle	Leakage Energy/Cycle
0.8V	1.4GHz	4.6e-10	1.9e-10
1.0V	2.0GHz	5.7e-10	2.0e-10
1.2V	2.5GHz	7.2e-10	2.1e-10

TABLE 3
Our Applications

Application	Functionality	Execution Cycles ($\times 10^6$)	Energy Consmp. (mJ)
Vcap	Video Capture and Processing	357.12	24.25
Convolution	Convolution Filter Implementation	196.71	19.07
TM	Image Conversion	382.15	22.93
IA	Target Detection and Classification	494.30	29.07
H.263	H.263 Decoder Implementation	776.36	49.36
ImgMult	Image Processing	252.08	24.73
Face	Face Recognition	607.42	42.90
Pgp	Encryption	484.18	26.11

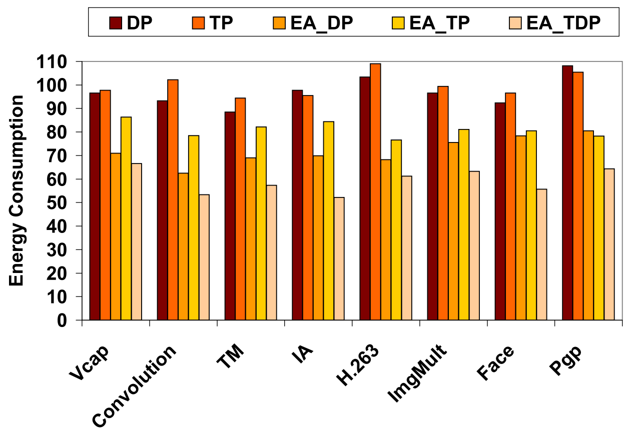


Fig. 6. Energy consumption with the default configuration.

5.2 Energy Consumption

Fig. 6 gives the energy consumption results with the default values of our simulation parameters, shown in Table 1. These energy results are normalized against the last column of Table 3. We see from these results that, on the average, the DP scheme saves 2.9 percent energy, which is not significant at all. This means that it does not make much sense to parallelize an application in this architecture without taking the voltage islands into account. The results with the TP scheme are even worse; this scheme generates similar results to the single processor case, and in fact, it increases energy consumption slightly on the average (0.3 percent). It needs to be mentioned that, the schemes DP and TP bring energy benefits in some applications. This is due to two main factors. First, leakage energy can be reduced when the execution cycles are cut. Second, some of the workloads or tasks can possibly be mapped to a processor in a low-frequency voltage island which has potential to incur less dynamic energy. Note that, as explained in Section 4, the chip area is divided into multiple voltage islands, each of which operates under a different voltage level/frequency. However, the voltage island oblivious mapping employed by these two methods can cause critical computations to be mapped to processors with low frequency and this can have a negative impact on both leakage and dynamic energy. When we look at the remaining three versions, we see that all three of them save significant amount of energy. These reductions come from both the dynamic and leakage components. The average energy reductions with the EA_DP and EA_TP schemes are 28.1 and 19.0 percent, respectively. However, the best energy savings are obtained when both data and task parallelism are exploited together (i.e., the EA_TDP scheme). This scheme achieves 40.7 percent energy saving on the average. These results clearly emphasize the importance of exploiting both task and data parallelism in a voltage island aware manner. That is, code parallelization employed by a compiler that targets an architecture with voltage islands should consider the workloads of processors and perform computation mapping considering the voltage/frequency levels of the processors in the system. Although we do not present the execution cycle results in detail, the schemes EA_DP, EA_TP, and EA_TPD also reduce execution cycles by 11.1, 8.6, and 14.6 percent, respectively.

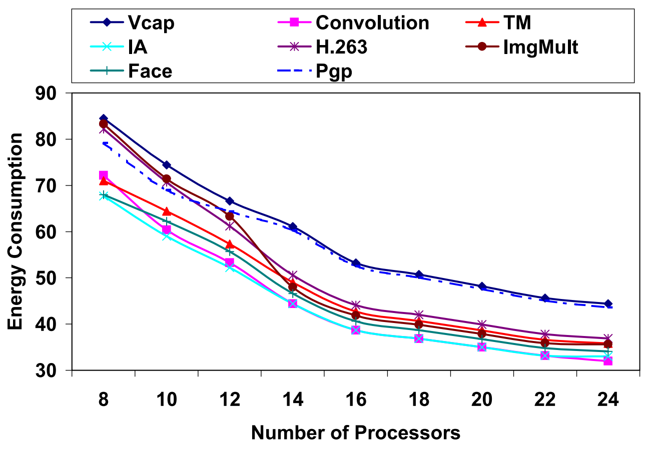


Fig. 7. Energy results of the EA_TDP scheme with the different processor counts.

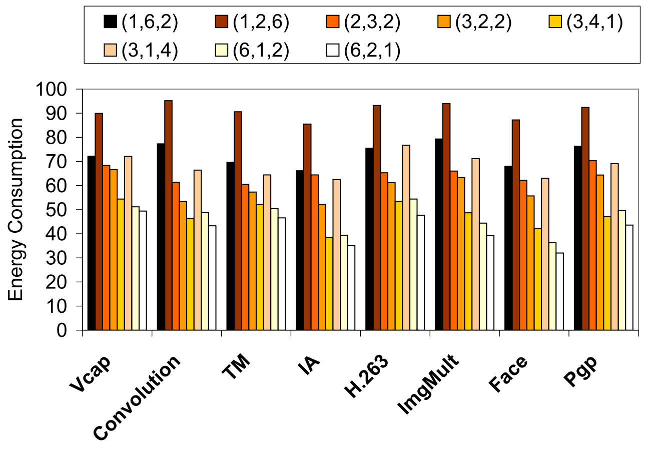


Fig. 8. Impact of the number of voltage islands.

In the rest of our experiments, we present the results obtained by varying our default system configuration. Since we already established that EA_TDP is superior to both EA_DP and EA_TP, in our experiments with different configurations, we concentrate only on the EA_TDP scheme.

We first change the total number of processors while keeping the number of power domains per island and the number of processors per power domain the same as in the default configuration. This means that we increase the number of islands in the architecture, which also means that we operate with a larger number of voltage/frequency levels. Fig. 7 shows the energy results (of the EA_TDP scheme) with the different processor counts. We see that our approach is able to take advantage of the increase in the number of processors. The main reason for this is that, an increased number of islands (voltage/frequency) gives more flexibility to our approach in performing computation mapping and this in turn increases our energy savings.

5.3 Sensitivity Analysis

In next last set of experiments, we studied the impact of the number of islands, the number of power domains per island, and the number of processor cores per power domain, while keeping the total number of processors in the system fixed. Each point on the x-axis of Fig. 8 represents a different configuration (number of islands, number of domains/island, number of processors/domain). Note that the total number of processors in the chip is fixed at 12 (our default count). Note also that the default configuration used so far is captured by (3, 2, 2). Several observations can be made from this graph. First, increasing the number of voltage islands generally brings more energy savings. This is because a larger number of islands mean a larger number of voltage/frequency levels, and this in turn impacts energy gains positively. This is why we observe more savings, as compared to the default configuration, with configurations such as (3,*,*) and (6,*,*) except (3,1,4). The reason that (3, 1, 4) generates worse results than (3, 2, 2) is that the former has only a single power domain per island, which means we can either turn off all the processors in an island or none of them. This restriction limits the potential leakage savings. On the other hand, the configuration (1, 2, 6) does not generate very good results either, mainly because we have only single voltage island (i.e., all benefits in this case can come from the leakage savings). Overall, these results show that the best energy savings are obtained when we have sufficiently large number of voltage islands and power domains. However, it needs to be clarified that voltage island architecture exploration (i.e., deciding the best combination of islands, domains, and processors) is not the topic addressed in this paper. This paper proposes a compiler-directed voltage island aware scheme to increase energy benefits. The reason that we made experiments with the different configurations and reported results in Fig. 8 is to show that the proposed approach works well across wide range of system configurations (not to defend one configuration over the others). When averaged across all configurations and applications used in our experiments, we calculate that our approach (EA_TDP) reduces energy consumption by 38.4 percent. Although not presented here in detail, the corresponding savings with the EA_DP and EA_TP versions were 26.8 and 18.2 percent. Again, these results reiterate the importance of exploiting both task and data parallelism in a voltage island based architecture for the best energy savings.

5.4 Comparison with Optimal Scheme

Our last set of results compare our best approach (EA_TPD) against an hypothetical scheme that performs code parallelization in an optimal manner. To collect the results with such a scheme, we first profiled each application and, by trying all possible parallelization schemes and voltage assignments, we distributed loop iterations across the available processors. We implemented an Integer Linear Programming (ILP)-based approach to measure the latency of such an optimal scheme compared to our approach. ILP execution time exponentially increases with the problem size, thereby making it harder to use for bigger problems with multidimensional problem spaces. In our implementation, there are three main variables effecting the optimality: 1) parallelization schemes used, 2) the number of voltage levels, 3) loop iteration distribution on available processors. In our experiments, the optimal approach took between 3-4 hours even when some of the variables are excluded or simplified, whereas our approach generates results in 47 seconds on average. Note that, we report the optimal scheme results based on either completion or the best results generated within 24 hours. These comparison results are presented in Fig. 9 under the default values of our simulation parameters. We observe from these results that our approach comes close to optimal for many of our embedded applications. In fact, the average energy saving values with our scheme and the optimal scheme are 40.7 and 45.6 percent, respectively. But, we also see some relatively larger difference in some benchmarks (e.g., H.263) due to the heuristic nature of our scheme.

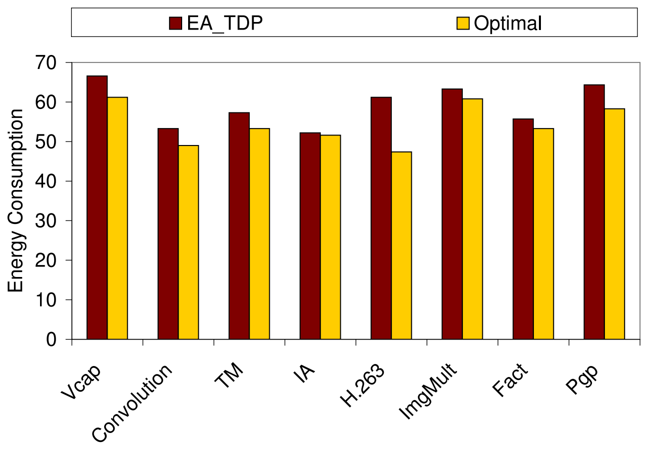


Fig. 9. Comparison with an optimal scheme.

6 Conclusions

Advances in semiconductor technology are enabling designs with several hundred million transistors. Since building sophisticated single processor based systems is a complex process from the design, verification, and software development perspectives, the use of chip multiprocessing is inevitable in future embedded systems. Another important architectural trend that we observe in embedded systems, namely, multivoltage processors, is driven by the need of reducing energy consumption during program execution. Considering these two trends, chip multiprocessing and voltage/frequency scaling, this paper presents an optimization strategy for a voltage island based embedded architecture that makes use of both chip parallelism and voltage scaling. In this approach, the compiler takes advantage of heterogeneity in parallel execution among the loads of different processors and assigns different voltages/frequencies to different processors if doing so reduces energy consumption without increasing overall execution cycles. Our experiments with a set of applications show that this optimization can bring large energy benefits in practice. To our knowledge, this is the first compiler-based study that targets a voltage island based embedded architecture.

Acknowledgments

A preliminary version of this paper appears in the SOCC'06 Proceedings [9]. This paper extends the paper by 1) presenting a detailed analysis and discussion of sources for load imbalance, 2) explaining our schemes in more detail, and 3) presenting an experimental analysis comparing our approach to an optimal scheme. This work is supported in part by TUBITAK grant 108E233, by a grant from IBM, and by a Marie Curie International Reintegration Grant within the Seventh European Community Framework Programme.

REFERENCES

[1] "Blast Power," http://www.magma-da.com, 2011.

[2] "Chip Designers Voyage to Voltage Island," http://news.com.com/Chip+designers+voyage+to+voltage+island/2100-1001_3-934355.html, 2011.

[3] "Solving the Power Management Dilemma in System-on-Chip Asic Design," http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/31C99333210244D 787256C1A0068B733/$file/VoltageIslands.PDF, 2011.

[4] "Voltage Guidelines for Pentium Processors with Mmx Technology," http://www.intel.com/design/intarch/applnots/24318603.pdf, 2011.

[5] "Mp98: A Mobile Processor," http://www.labs.nec.co.jp/MP98/top-e.htm,http://www.labs.nec.co.jp/MP98/top-e.htm, 2011.

[6] "Majc-5200," http://www.sun.com/microelectronics/MAJC/5200wp.html, 2011.

[7] J.-A. Carballo, J.L. Burns, S.-M. Yoo, I. Vo, and V.R. Norman, "A Semi-Custom Voltage-Island Technique and Its Application to High-Speed Serial Links," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '03)*, pp. 60-65, 2003.

[8] S. Carr, K.S. McKinley, and C.-W. Tseng, "Compiler Optimizations for Improving Data Locality," *SIGPLAN Notices*, vol. 29, no. 11, pp. 252-262, 1994.

[9] G. Chen, M. Kandemir, and M. Karakoy, "Compiler Support for Voltage Islands," *Proc. IEEE Int'l SOC Conf. (SOCC '06)*, Sept. 2006.

[10] G. Chen, F. Li, M. Kandemir, and M.J. Irwin, "Reducing Noc Energy Consumption through Compiler-Directed Channel Voltage Scaling," *Proc. Symp. Programming Language Design and Implementation (PLDI '06)*, vol. 41, no. 6, pp. 193-203, June 2006.

[11] G. Chen, G. Chen, O. Ozturk, and M. Kandemir, "Exploiting Inter-Processor Data Sharing for Improving Behavior of Multi-Processor Socs," *Proc. IEEE CS Ann. Symp. VLSI: New Frontiers in VLSI Design (ISVLSI '05)*, pp. 90-95, 2005.

[12] H. Giefers and A. Rettberg, "Energy Aware Multiple Clock Domain Scheduling for a Bit-Serial, Self-Timed Architecture," *Proc. 19th Ann. Symp. Integrated Circuits and Systems Design (SBCCI '06)*, pp. 113-118, 2006.

[13] C.-H. Hsu, U. Kremer, and M. Hsiao, "Compiler-directed Dynamic Frequency and Voltage Scheduling," *Proc. First Int'l Workshop Power-Aware Computer Systems-Revised Papers, (PACS '00)*, pp. 65-81, 2001.

[14] J. Hu, Y. Shin, N. Dhanwada, and R. Marculescu, "Architecting Voltage Islands in Core-Based System-on-a-Chip Designs," *Proc. Int'l Symp. Low Power Electronics and Design*, pp. 180-185, 2004.

[15] M.J. Irwin, L. Benini, N. Vijaykrishnan, and M. Kandemir, *Multiprocessor Systems-on-Chips.* Morgan Kaufmann, 2003.

[16] R. Jejurikar and R. Gupta, "Dynamic Voltage Scaling for System-wide Energy Minimization in Real-Time Embedded Systems," *Proc. Int'l Symp. Low Power Electronics and Design (ISLPED '04)*, pp. 78-81, 2004.

[17] I. Kadayif, M. Kandemir, G. Chen, O. Ozturk, and U. Sezer, "Optimizing Array-Intensive Applications for On-Chip Multiprocessors," *IEEE Trans. Parallel Distributed Systems*, vol. 16, no. 5, pp. 396-411, May 2005.

[18] V. Krishnan and J. Torrellas, "A Chip-Multiprocessor Architecture with Speculative Multithreading," *IEEE Trans. Computers*, vol. 48, no. 9, pp. 866-880, Sept. 1999.

[19] D.E. Lackey, P.S. Zuchowski, T.R. Bednar, D.W. Stout, S.W. Gould, and J.M. Cohn, "Managing Power and Performance for System-on-Chip Designs Using Voltage Islands," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design*, pp. 195-202, 2002.

[20] F. Li, G. Chen, and M. Kandemir, "Compiler-Directed Voltage Scaling on Communication Links for Reducing Power Consumption," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '05)*, pp. 456-460, Nov. 2005.

[21] A.W. Lim and M.S. Lam, "Maximizing Parallelism and Minimizing Synchronization with Affine Transforms," *Proc. 24th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL '97)*, pp. 201-214, 1997.

[22] B. Liu, Y. Cai, Q. Zhou, and X. Hong, "Power Driven Placement with Layout Aware Supply Voltage Assignment for Voltage Island Generation in Dual-vdd Designs," *Proc. Conf. Asia South Pacific Design Automation (ASP-DAC '06)*, pp. 582-587, 2006.

[23] Q. Ma and E.F.Y. Young, "Voltage Island-Driven Floorplanning," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '07)*, pp. 644-649, 2007.

[24] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50-58, Feb. 2002.

[25] S. Manolache, P. Eles, and Z. Peng, "Fault and Energy-Aware Communication Mapping with Guaranteed Latency for Applications Implemented on Noc," *Proc. 42nd Ann. Conf. Design Automation*, pp. 266-269, 2005.

[26] S.H.K. Narayan, O. Ozturk, M. Kandemir, and M. Karakoy, "Workload Clustering for Increasing Energy Savings on Embedded Mpsocs," *Proc. IEEE Int'l SOC Conf.*, 2005.

[27] K. Niyogi and D. Marculescu, "Speed and Voltage Selection for Gals Systems Based on Voltage/Frequency Islands," *Proc. Conf. Asia South Pacific Design Automation (ASP-DAC '05)*, pp. 292-297, 2005.

[28] K. Olukotun, B.A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The Case for a Single-Chip Multiprocessor," *SIGPLAN Notices*, vol. 31, no. 9, pp. 2-11, 1996.

[29] O. Ozturk, M. Kandemir, G. Chen, and M.J. Irwin, "Customized On-chip Memories for Embedded Chip Multiprocessors," *Proc. Conf. Asia South Pacific Design Automation*, vol. 2, pp. 743-748, http://www.gigascale.org/pubs/638.html, Jan. 2005.

[30] O. Ozturk, M. Kandemir, M.J. Irwin, and I. Kolcu, "Tuning Data Replication for Improving Behavior of Mpsoc Applications," *Proc. 14th ACM Great Lakes Symp. Very Large Scale Integration (GLSVLSI '04)*, pp. 170-173, 2004.

[31] A. Rae and S. Parameswaran, "Voltage Reduction of Application Specific Heterogeneous Multiprocessor Systems for Power Minimization," *Proc. Asia and South Pacific Design Automation Conf.*, 2000.

[32] A. Rangasamy, R. Nagpal, and Y. Srikant, "Compiler-directed Frequency and Voltage Scaling for a Multiple Clock Domain Microarchitecture," *Proc. Fifth Conf. Computing Frontiers*, pp. 209-218, 2008.

[33] G. Reinman and N.P. Jouppi, "Cacti 2.0: An Integrated Cache Timing and Power Model," technical Report, Compaq, Feb. 2000.

[34] C. Seiculescu, S. Murali, L. Benini, and G. De Micheli, "Comparative Analysis of Nocs for Two-dimensional Versus Three-Dimensional Socs Supporting Multiple Voltage and Frequency Islands," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 57, no. 5, pp. 364-368, May 2010.

[35] L. Shang, L.-S. Peh, and N.K. Jha, "Dynamic Voltage Scaling with Links for Power Optimization of Interconnection Networks," *Proc. Ninth Int'l Symp. High-Performance Computer Architecture (HPCA '03)*, p. 91, 2003.

[36] Z. Shao, M. Wang, Y. Chen, C. Xue, M. Qiu, L.T. Yang, and E.H.M. Sha, "Real-Time Dynamic Voltage Loop Scheduling for Multi-Core Embedded Systems," *IEEE Trans. Circuits and Systems II: Express Briefs*, vol. 54, no. 5, pp. 445-449, May 2007.

[37] Q. Shi, T. Chen, X. Liang, and J. Huang, "Dynamic Compilation Framework with Dvs for Reducing Energy Consumption in Embedded Processors," *Proc. Int'l Conf. Embedded Software and Systems (ICESS '08)*, pp. 464-470, July 2008.

[38] V. Soteriou and L.S. Peh, "Dynamic Power Management for Power Optimization of Interconnection Networks Using On/Off Links," *Proc. 11th Symp. High Performance Interconnects (Hot-I)*, 2003.

[39] S. Tosun, N. Mansouri, M. Kandemir, and O. Ozturk, "Constraint-Based Code Mapping for Heterogeneous Chip Multiprocessors," *Proc. IEEE Int'l SOC Conf.*, 2005.

[40] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, "The Determination of Worst-Case Execution Times—Overview of the Methods and Survey of Tools," *ACM Trans. Embedded Computing Systems*, vol. 7, no. 3, article 36, 2008.

[41] R.P. Wilson, R.S. French, C.S. Wilson, S.P. Amarasinghe, J.M. Anderson, S.W.K. Tjiang, S.-W. Liao, C.-W. Tseng, M.W. Hall, M.S. Lam, and J.L. Hennessy, "Suif: An Infrastructure for Research on Parallelizing and Optimizing Compilers," *SIGPLAN Notices*, vol. 29, no. 12, pp. 31-37, 1994.

[42] H. Wu, I.-M. Liu, M.D.F. Wong, and Y. Wang, "Post-Placement Voltage Island Generation under Performance Requirement," *Proc. IEEE/ACM Int'l Conf. Computer-Aided Design (ICCAD '05)*, pp. 309-316, 2005.

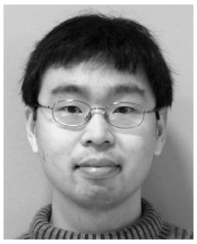
[43] H. Wu and M.D.F. Wong, "Incremental Improvement of Voltage Assignment," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 28, no. 2, pp. 217-230, Feb. 2009.

[44] C.Y. Yang, J.J. Chen, and T.W. Kuo, "An Approximation Algorithm for Energy Efficient Scheduling on a Chip Multiprocessor," *Proc. ACM/IEEE Design Automation and Test in Europe,* Mar. 2005.

[45] W. Ye, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin, "The Design and Use of Simplepower: A Cycle-Accurate Energy Estimation Tool," *Proc. 37th Conf. Design Automation,* pp. 340-345, 2000.

Ozcan Ozturk received the bachelor's degree from Bogazici University, Istanbul, Turkey, in computer engineering in 2000, the MSc degree in computer engineering in 2002, from the University of Florida, Gainesville, and the PhD degree in computer engineering in 2007, from The Pennsylvania State University, University Park. He is currently an assistant professor in the Department of Computer Engineering at Bilkent University, Ankara, Turkey. Prior to joining Bilkent University, he worked as a software optimization engineer in Cellular and Handheld Group at Intel (Marvell). He also held visiting researcher positions at NEC Labs America in Princeton, New Jersey, ALCHEMY group in INRIA, Paris, France, and at Processor Architecture Laboratory (LAP), Swiss Federal Institute of Technology of Lausanne (EPFL), Lausanne, Switzerland. His research interests are in the areas of multicore and manycore architectures, cloud computing, power-aware architectures, and compiler optimizations. He is currently serving as an editor and a reviewer on leading IEEE, ACM, and other journals. He is a recipient of 2006 International Conference on Parallel and Distributed Systems (ICPADS) Best Paper Award, 2009 IBM Faculty Award, and 2009 Marie Curie Fellowship from the European Commission. He is a member of the IEEE, ACM, GSRC, and HiPEAC.

Mahmut Kandemir is an associate professor in the Computer Science and Engineering Department at the Pennsylvania State University, State College. He is a member of the Microsystems Design Lab. His research interests are in optimizing compilers, runtime systems, embedded systems, I/O and high performance storage, and power-aware computing. He is the author of more than 300 papers in these areas. His research is funded by the US National Science Foundation (NSF), US Defense Advanced Research Projects Agency (DARPA), and SRC. He is a recipient of the NSF Career Award and the Penn State Engineering Society Outstanding Research Award. He is a member of the ACM and IEEE.



Guangyu Chen received the BSc degree from the University of Science and Technology of China, Hefei, China in 1997 and the MEng degree from Institute of Software, Chinese Academia of Science, Beijing, China, 2000. He also received the PhD degree from the Computer Science and Engineering Department, Pennsylvania State University in 2006. He worked as a software engineer at Microsoft Corporation from 2006 to 2009. Since 2009, he has been with Facebook as a software engineer. His main research interests are Java technology, Just-In-Time compilers, and power-aware computing. He is a member of the IEEE.