Workload Clustering for Increasing Energy Savings on Embedded MPSoCs

ABSTRACT

Voltage/frequency scaling and processor low-power modes (i.e., processor shut-down) are two important mechanisms used for reducing energy consumption in embedded MPSoCs. While a unified scheme that combines these two mechanisms can achieve significant savings in some cases, such an approach is limited by the code parallelization strategy employed. In this paper, we propose a novel, integer linear programming (ILP) based workload clustering strategy across parallel processors, oriented towards maximizing the number of idle processors without impacting original execution times. These idle processors can then be switched to a low power mode to maximize energy savings, whereas the remaining ones can make use of voltage/frequency scaling. In order to check whether this approach brings any energy benefits over the pure voltage scaling based, pure processor shut-down based, or a simple unified scheme, we implemented four different approaches and tested them using a set of eight array/loop-intensive embedded applications. Our simulation-based analysis reveals that the proposed ILP approach (1) is very effective in reducing energy consumption of the applications tested and (2) generates much better energy savings than all the alternate schemes tested (including a unified scheme that combines voltage/frequency scaling and processor shutdown). In this paper, we also compare this ILP-based approach to a heuristic scheme that clusters processor workloads.

1. INTRODUCTION

Embedded multi-processor system-on-a-chip architectures (MPSoCs) are becoming increasingly popular as they bring advantages of exploiting parallelism at a high level and easier validation/verification. However, since these architectures contain multiple processors on the same chip along with several types of memory components, the energy consumption of the chip can be a major concern. This observation has led to several recent efforts on reducing energy consumption of embedded MPSoCs.

We can roughly divide the efforts on energy savings in MPSoCs into two categories. In this first category are the studies that employ processor voltage/frequency scaling. The basic idea is to scale down voltage/frequency of a processor if its current workload is less than the workloads of other processors. In comparison, the studies in the second category shut down unused processors (i.e., put them into low-power states along with their private memory components) during the execution of the current computation. Both these techniques, i.e., voltage scaling and processor shut down, can be applied at the software level (e.g., directed by an optimizing compiler) or at the hardware-level (e.g., based on a past history-based workload/idleness detection algorithm). It is also conceivable to combine these two techniques under a unified optimizer.

Each of these techniques has its advantages and drawbacks. For example, a processor shut-down based scheme may not be applicable if there is no unused processor (note that this does not mean that the workloads of all the processors in the MPSoC are similar). Similarly, the effectiveness of a voltage scaling based scheme is limited by the number of voltage/frequency levels supported by the underlying hardware. In general, exploiting processor/memory shutdown saves more energy when it is applicable (as it reduces leakage energy significantly) or when we have only a couple of voltage/frequency levels to use. If this is not the case, then voltage scaling can be effective (and in some cases it is the only choice). Based on this discussion, one can expect a unified scheme to be successful. However, we want to re-iterate that if there is no unused (idle) processor in the current workload assignment, such a unified scheme simply reduces to a voltage scaling based approach.

Our goal in this paper is to explore a *workload (job) clustering* scheme that combines voltage scaling with processor shut-down[1]. The uniqueness of the proposed unified approach is that it maximizes the opportunities for processor shut-down by assigning workloads to processors carefully. It achieves this by clustering the original workloads of processors in as few processors as possible. In this paper, we discuss the technical details of this approach to energy saving in embedded MPSoCs. The proposed approach is ILP (integer linear programming) based; that is, it determines the optimal workload clusterings across the processors by formulating the problem using ILP and solving it using a publicly-available linear solver. In order to check whether this approach brings any energy benefits over the pure voltage scaling based, pure processor shut-down based, or a simple unified scheme, we implemented four different approaches within our linear solver and tested them using a set of eight array/loop-intensive embedded applications. Our simulation-based analysis reveals that the proposed ILP approach (1) is very effective in reducing energy consumption of the applications tested and (2) generates much better energy savings than all the alternate schemes tested (including one that combines voltage/frequency scaling and processor shutdown). In this paper, we also compare this ILP-based approach to a heuristic scheme that clusters processor workloads.

The remainder of this paper is structured as follows. The next section explains the embedded MPSoC architecture considered in this work and our application execution model under this architecture. This section also briefly discusses the related work. The details of our ILP-based solution to processor workload clustering are given in Section 3. Our experimental platform, characteristics of the applications used, and the results obtained are presented in Section 4. Finally, Section 5 concludes the paper.

2. EMBEDDED MPSOC ARCHITECTURE AND EXECUTION MODEL

The chip multiprocessor we consider in this work is a shared-memory architecture; that is, the entire address space is accessible by all processors. Each processor has a private L1 cache, and the shared memory is assumed to be off-chip. Optionally, we may include a (shared) L2 cache as well. Note that several architectures from academia and industry fit in this description [2, 13, 10, 11]. We keep the subsequent discussion simple by using a shared bus as the interconnect (though one could use fancier/higher bandwidth interconnects as well). We also use the MESI [17] protocol (the choice is orthogonal to the focus of this paper) to keep the caches coherent across the CPUs. We assume that voltage level and frequency of each processor in this architecture can be set independently of the others, and also processors can be placed into low power modes independently. This paper focuses on a single-issue, five-stage (instruction fetch (IF), instruction decode/operand fetch (ID), execution (EXE), memory access (MEM), and write-back (WB) stages) pipelined datapath for each on-chip processor.

Our application execution model in this embedded MPSoC can be summarized as follows. We focus on array-based embedded applications that are constructed from loop nests. Typically, each loop nest in such an application is small but executes a large number of iterations and accesses/manipulates large datasets (typically multidimensional arrays of signals). We employ a loop nest based application parallelization strategy. More specifically, each loop nest is parallelized independently of the others. In this context, parallelizing a loop nest means distributing its iterations across processors and allowing processors to execute their portions in parallel. For example, a loop with 1000 iterations can be parallelized across 10 processors by allocating 100 it-

[1]In this paper, we use the terms "processor show-down" and "low-power mode" interchangeably.

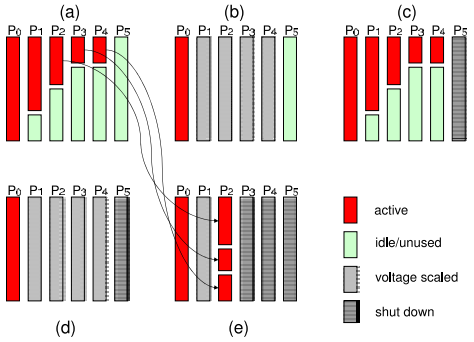

Figure 1: Comparison of different energy-saving approaches for a six processor architecture. Arrows indicate how the workloads (jobs) are clustered by our approach.

erations to each processor. We also assume that after each loop nest execution, all processors get synchronized before they start executing the next loop nest. Note that dropping this requirement would necessitate a sophisticated compiler analysis to identify the cases under which a processor that finishes its portion of iterations from the previous loop nest can go ahead and start executing its portion from the next loop nest without waiting for the others.

There are many proposals for power management of a dynamic voltage scaling-capable processor. Most of them are at the operating system level and are either task-based [12, 15] or interval-based [18, 5]. While some proposals aim at reducing energy without compromising performance, a recent study by Grunwald et al [6] observed noticeable performance loss for some interval-based algorithms using actual measurements. Most of the existing compiler based studies such as [7, 14] target single processor architectures. In comparison, our work targets at a chip multiprocessor based environment and combines voltage scaling and processor shutdown. [20] presents and analyzes a voltage/frequency scaling scheme but, they do not consider processor shutdown. [8] employs processor a shut-down based mechanism but does not consider voltage/frequency scaling. In our experimental evaluation, we compare our approach to pure voltage/frequency scaling and to pure processor shut-down as well.

3. OUR APPROACH

3.1 Overview

Figure 1 compares four different alternate schemes that saves energy in an embedded MPSoC architecture. It is assumed, for illustrative purposes, that the architecture has six processors. In Figure 1(a) shows the workloads of the processors (i.e., the jobs assigned to them) in a given loop nest. These are assumed to be the loads either estimated by the compiler or calculated through profiling and are for a single nest. Figures 1(b) and (c) show the scenarios with pure voltage/frequency scaling and pure processor shut-down based approach, respectively. In (b), four out of six processors take advantage of voltage scaling (note that P_5 is not used in the computation at all). In (c), on the other hand, we can place only one processors (P_5) into the low-power mode. A combination of these two approaches is depicted in Figure 1(d). Basically, this version combines the benefits of voltage/frequency scaling and processor shut-down. Finally, the result that can obtained by the ILP approach proposed in this paper is illustrated in Figure 1(e). Note that what our approach achieves is to *cluster* the total amount of computational load in as fewer processors as possible so that the number of unused processors is maximized. In this particular case, the original loads of three processors (P_2, P_3, and P_4) are combined and assigned to processor P_2. As a result, processors P_3 and P_4 can be also placed into the low-power mode (along with their private memory components) to maximize energy savings, in addition to P_5. The next subsection gives the technical details of this approach. When there are opportunities, our approach can also use voltage/frequency scaling for the clustered jobs.

However, we first need to clarify two important issues. Someone may ask at this point "why has the application (corresponding to the scenario in Figure 1(a)) not been parallelized at the first place as shown in Figure 1(e)?" There are several reasons for this. First, most current code parallelizers do not consider any energy optimizations. Therefore, there is really little reason for calculating the workloads of individual processors, and thus little opportunity for workload clustering. Second, the conventional parallelizing compilers try to use as many processors as possible for executing a given computation unless there exists a compelling reason to do otherwise (e.g., the excessive synchronization costs). Third, in many cases, trying to cluster computation in very few processors can have an impact on execution cycles. Since most parallelizing compilers do not predict or quantify this impact, they do not attempt such clusterings, being on the conservative side.

The second issue is that, it is possible that the scenario depicted in Figure 1(e) has poor data locality as compared to scenarios in Figures 1(b), (c), and (d). This is because conventional code parallelizers generally try to achieve good data locality, by ensuring that each processor mostly uses the same set of data elements as much as possible (i.e., high data reuse). As a result, the scenario in Figure 1(e) can lead to an increase in data cache misses, which in turn increases overall energy consumption. This overhead should also be factored in our clustering approach to ensure a fair comparison.

The main contribution of the ILP approach proposed in this paper is to obtain, for each loop nest in an application, the result shown in Figure 1(e), given the initial scenario (workload assignment) shown in Figure 1(a) and thus reduce energy consumption.

3.2 Technical Details and Problem Formulation

This section elaborates on the ILP model used to represent the problem. In our problem, there exists a set of jobs (workloads) that have to be executed on a set of available CPUs in the embedded MPSoC such that the total energy spent by the system is minimal and that the execution of the jobs completes within a specified time limit, T_{max}.[2] The processors can run different jobs at different voltage and frequency levels, which affects energy consumption. The energy expended by each processor is the sum of the dynamic energy as well as the leakage energy expended while running. The rest of this section describes the ILP model in detail.

3.2.1 System and Job Model

Jobs are members of the set J consisting of J_{max} elements and the processors belong to the set P in which there are P_{max} elements. The processors can run at V_{num} discrete set of voltage/frequency levels (as supported by the architecture). It is assumed that only one job can run on a processor at anytime and that once a job starts running on a processor, it runs uninterrupted to completion. However, a processor can run more that one job, as a result of workload clustering. The duration that the job occupies the processor is dependent on the supply voltage/frequency as well as the the frequency at which the processor is running that particular job. The time (latency) each job takes up at different voltage levels is specified in the array $Job\_Length(j, v)$. Similarly, the dynamic energy spent by each job at different voltage levels varies and is specified in $Job\_Dynamic(j, v)$.[3] $Total\_Energy$ is the sum of the energies spent by all jobs on all processors due to their running as well as the leakage energy consumed by the processors. This is the metric whose value we want to minimize.

3.2.2 Mathematical Programming Model

The constraints specified below give the mathematical representation of our model. We use 0-1 integer linear programming (ILP). This ILP formulation is executed for each loop nest separately. Table 1 gives the notation used in our formulation.

Job Assignment Constraints. The 0-1 variable $X(p, j, v)$ determines whether processor p runs job j at voltage/frequency level v. One

[2]In this paper, we do not assume a specific code (loop nest) parallelization strategy. Rather, we assume that each loop nest is parallelized using one of the known techniques. For each nest, T_{max} is determined by the processor with the largest workload. This is to ensure that our clustering does not have a negative impact on execution times.

[3]Here j represents a job (workload) and v represents a voltage (frequency) level. In our implementation, the entries of $Job\_Length(j, v)$ and $Job\_Dynamic(j, v)$ are filled using profiling. All energy estimations are performed using Wattch [1] under the 70nm process technology. The increase in data cache misses as a result of clustering is captured during our profiling.

Notation	Explanation
$Job\_Dynamic(j,v)$	Dynamic energy for running job (workload) j at voltage v
$Job\_Length(j,v)$	Time taken to run job j at voltage v
$X(p,j,v)$	Value is 1 if job j runs on processor p at voltage v (0-1 variable)
J	Set of jobs
P	Set of processors
T_max	Time deadline before which all jobs must finish
J_max	Total number of jobs to be executed
P_max	Total number of processors available
V_num	Total number of voltage (and frequency) levels available
$Total\_Energy$	Total energy consumption of the system (to be minimized)
$Leakage\_Value$	Leakage energy spent by a processor if it is not shut down

Table 1: Notation used in our model.

job runs completely on one processor and all jobs are scheduled to run only once. This is specified as follows:

$$\forall p \in P \quad \forall j \in J \quad \forall v \in V \quad X(p,j,v) \in \{0|1\} \tag{1}$$

$$\forall j \in J \sum_{p=0}^{P_{max}-1} \sum_{v=0}^{V_{num}-1} X(p,j,v) = 1 \tag{2}$$

Constraint (1) expresses the term $X(j,p,v)$ as a binary variable; a processor either runs the job or it does not. Constraint (2) states that each job can be run only on one processor and that all jobs are assigned to some processors (i.e., no job is left unassigned). Notice that we want to determine the value of $X(p,j,v)$ for all p, j, and v.

Deadline Constraints. Jobs are assigned to processors as long as they can meet the time deadline that is specified. Constraint (3) expresses this:

$$\forall p \in P \sum_{j=0}^{J_{max}-1} \sum_{v=0}^{V_{num}-1} X(p,j,v) * Job\_Length(j,v) \leq T_{max} \tag{3}$$

Note that T_{max} is determined, for each loop nest, by the longest (largest) workload.

Clustering and Processor Shut-Down Constraints. Multiple jobs are run on the same processor if the number of jobs, J_{max}, exceeds the number of processor, P_{max}, but also if such an arrangement reduces the overall energy spent by the system. In case a processor is not assigned any job, either because of clustering of jobs or because $J_{max} < P_{max}$ or because of both these reasons, then it is shut down. Such a processor does not consume any dynamic energy as it has no jobs running on it and it does not consume any leakage energy as it is shut down (except for some small amount of leakage in memory components). Constraint (4) is introduced to capture processor shutdown:

$$\forall p \in P, \forall j \in J, \forall v \in V \quad Busy(p) \geq X(p,j,v) \tag{4}$$

For a particular processor p, $Busy(p)$ is necessarily 1 if any of the values in $X(p,j,v)$ is 1. Through this constraint, the value of $Busy(P)$ is not explicitly expressed if all values in $X(p,j,v)$ are 0. However, a value of 1 in $Busy(p)$ adds leakage to the overall energy. As the objective of the ILP-based model is to reduce energy, $Busy(p)$ will be assigned to be 0 if all values in $X(p,j,v)$ are 0.[4]

Leakage and Dynamic Energy Calculation. The following expressions capture the leakage energy and dynamic energy spent by the system as the sum of the leakage and dynamic energies, respectively, spent by each processor. Dynamic energy spent by a processor is the sum of the dynamic energies spent for each job that is run on that processor. This is captured by expression (5):

$$D\_Energy = \sum_{p=0}^{P_{max}-1} \sum_{j=0}^{J_{max}-1} \sum_{v=0}^{V_{num}-1} X(p,j,v) * Job\_Energy(j,v) \tag{5}$$

[4]To preserve data in memory components, a shut-down processor consumes some leakage [4]. Our experiments are performed based on this principle. However, in our presentation of the ILP formulation, we assume no leakage consumption in the shut-down state for ease of presentation.

Expression (6) calculates the leakage energy spent. As mentioned earlier, if $Busy(p)$ is 1, then leakage is spent by processor p.

$$L\_Energy = Leakage\_Value * \sum_{p=0}^{P_{max}-1} * Busy(p) \tag{6}$$

Objective Function. The objective function which is the total energy spent by the system is the sum of the the leakage and dynamic energies. This is the function that our approach tries to minimize:

$$Total\_Energy = D\_Energy + L\_Energy \tag{7}$$

The constraints and expressions mentioned in this section are sufficient to express our problem within ILP. We next look at the additional constraints that can be used in order to handle two special cases.

Voltage/Frequency Scaling without Clustering. To model classical voltage/frequency scaling within our ILP formulation, an input value $Assign(j,p)$ should specify the processor on which each job runs. Further, by connecting this value to that of $X(j,p,v)$, all jobs are forced to run on the assigned processors alone. This connection can be specified by the following constraint:

$$\forall p \in P, \forall j \in J \sum_{v=0}^{V_{num}-1} X(p,j,v) = Assign(p,j) \tag{8}$$

Clustering without Voltage/Frequency Scaling. To model job clustering without voltage and frequency scaling, we need to constrain the choice of available voltage frequency levels to either each processor individually or all processors. In the case of constraining the voltage levels of all processors to one value, constraint (9) can be used to ensure that all no jobs are assigned voltage levels other than the one specified.

$$\forall p \in P, \forall j \in J, \forall v \in V - \{v'\} \quad X(p,j,v) = 0. \tag{9}$$

To constrain each individual processor to an independent voltage level, constraint (10) below can be used.

$$\forall p \in P, \forall j \in J, \forall v \in V - \{v'_p\} \quad X(p,j,v) = 0. \tag{10}$$

Here, v' and v'_p are the universal and individual (for processor p) voltage levels, respectively. These constraints simply limit the voltage levels to be used. In this case, the decision to cluster jobs together on a processor is made by our solver and depends on whether it results in a lowered overall energy consumption.

3.2.3 Explicit Expression of Overheads

Recall that, the degradation in cache behavior as a result of our workload clustering is captured in $Job\_Dynamic$ and Job_Length arrays during profiling of the application code. However, an alternate formulation of the problem that expresses the extra overheads due to clustering is also possible. This can be done as follows. $Assign(p,j)$, as defined above, represents the default assignment of jobs to CPUs (i.e., the assignment before clustering). If the jobs are not scheduled on their default processors, there is an energy penalty incurred. The extent of the penalty is dependent on the characteristics of the jobs. Array $Move\_Cost(j,p_1,p_2)$ is used to capture the energy spent in migrating (transferring for the sake of clustering) job j from processor p_1 to p_2. $Move(j,p_1,p_2)$ is a 0-1 variable, which is 1 if job j migrates from processor p_1 to processor p_2. Constraint (11) below calculates the total overhead energy spent due to clustering workloads and constraints (12) and (13) deal with the term $Move(j,p_1,p_2)$.

$$S\_Energy = \sum_{j=0}^{J_{max}-1} \sum_{p_1=0}^{P_{max}-1} \sum_{\substack{p_2=0 \\ p_1 \neq p_2}}^{P_{max}-1} \left(Move(j,p_1,p_2) * Move\_Cost(j,p_1,p_2)\right) \tag{11}$$

$$\forall j \in J, \forall p_1 \in P, \forall p_2 \in P : Move(j,p_1,p_2) \in \{0|1\} \tag{12}$$

Constant	Value
T_{max}	6 time units
J_{max}	4
P_{max}	4
V_{num}	5
$Leakage\_value$	5 energy units

Table 2: Constant parameters used in the example.

$X(p,j,v)$	Interpretation
$X(0,0,3)$	Processor 0 runs job 0 at voltage level 3
$X(0,2,2)$	Processor 0 runs job 2 at voltage level 2
$X(2,3,4)$	Processor 2 runs job 3 at voltage level 4
$X(3,1,3)$	Processor 3 runs job 1 at voltage level 3

Table 3: $X(p,j,v)$ values determined by the ILP approach.

$$\forall j \in J, \forall p_1 \in P, \forall p_2 \in P \text{ such that } p_1 \neq p_2 :$$
$$Move(j,p_1,p_2) = Assign(p_1,j) * \sum_{v=0}^{V_{num}-1} X(p_2,j,v) \quad (13)$$

We also need to alter the objective function to include S_Energy. The new objective function is:

$$Total\_Energy = D\_Energy + L\_Energy + S\_Energy. \quad (14)$$

3.2.4 Heuristic Approach

We now describe a heuristic algorithm that tries to perform processor shut-down through workload clustering as well as voltage/frequency scaling. The algorithm (given in Algorithm 1) has two phases. The primary phase of the algorithm closely resembles a bin-packing heuristic in which jobs are allocated to processors based on some given capacity limit so that the processors are filled as much as possible. This is essentially a greedy approach. In the second phase, one job is chosen among all the jobs allocated to a processor and this job is scaled (if there is a slack in that processor) so that the energy consumption is further reduced. The choice of the job is such that energy is reduced to the greatest extent, and this process is repeated for all processors with a slack (free time). Finally, the energy of the resulting clustering is calculated.

3.2.5 Example

This section presents an example and demonstrates how the ILP method and the heuristic method operate in practice. Table 2 shows the constant parameters for the system. There are 4 jobs (workloads) to be run on 4 processors. Each job can be run at 5 different voltage/frequency levels and the deadline for the completion of the jobs is 6 time units. These values are selected for illustrative purpose only.

Array $Job\_Dynamic(j,v)$ provides the dynamic energy spent in running each job at different voltage/frequency levels, and is assumed to be obtained (through profiling) as follows:

$$Job\_Dynamic = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 6 & 8 & 10 \\ 2 & 3 & 5 & 6 & 8 \\ 3 & 6 & 9 & 12 & 15 \end{pmatrix}.$$

Array $Job\_Length(j,v)$ provides the execution time (latency) of each job at different voltage/frequency levels and is assumed to be as follows:

$$Job\_Length = \begin{pmatrix} 6 & 5 & 4 & 3 & 2 \\ 12 & 10 & 8 & 6 & 4 \\ 9 & 7 & 3 & 2 & 1 \\ 24 & 15 & 12 & 9 & 6 \end{pmatrix}.$$

$X(p,j,v)$ values returned by our ILP solver are presented in Table 3. From this table, it can be gathered that there are two jobs executed on processor 0, one job each is executed on processors 2 and 3 and that no job is executed on processor 1. All jobs finish on or before the specified deadline. The total dynamic energy spent is 32 units, which is calculated as follows:

$$D\_Energy = X(0,0,3)*Job\_Dynamic(0,3) + X(0,2,2)*Job\_Dynamic(2,2) + X(0,1,3)*Job\_Dynamic(1,3) + X(0,3,4)*Job\_Dynamic(3,4) = 1*4 + 1*5 + 1*8 + 1*15 = 32.$$

Since three processors are used, 15 energy units are spent as leakage. This calculation is shown below.

$$L\_Energy = 3 * Leakage\_value = 3 * 5 = 15$$

As a result, the total energy spent is the sum of the dynamic and leakage energies spent by all processors. Therefore, we have:

$$Total\_Energy = D\_Energy + L\_Energy = 32 + 15 = 47$$

Our heuristic approach, on the other hand, proceeds as follows. In the primary phase, all jobs are assigned greedily to a processor in which they can complete within the time limit, T_{max} (6 unites). Job 0 is assigned to processor 0 at voltage level 4. Thus, it occupies 2 units of time on processor 0. Job 1 requires 4 time units to finish its execution. Hence, it is assigned to processor 0 since processor 0 has 4 time units free. Now, processor 0 is completely assigned, whereas processors 1, 2 and 3 are free. Job 2 takes 1 time unit to run and is assigned to processor 1. Job 3 takes 6 time units to execute. As processors 0 and 1 do

Algorithm 1 $Heuristic\_Job\_Allocator()$

```
1: // Initialization
2: D_Energy := 0; L_Energy := 0
3: Total_Energy := 0
4: for p = 0 to P_max -1 do
5:    Proc_Time_Left(p) := T_max
6:    for j = 0 to J_max -1 do
7:       X(p,j) := 0
8:    end for
9: end for
10: // Primary Phase
11: for j = 0 to J_max -1 do
12:    Least_So_Far := T_max; Least_So_Far_Holder := P_max -1
13:    for p = 0 to P_max -1 do
14:       if Proc_Time_Left[p] ≥ Job_Duration[j,V_num -1] and
          Proc_Time_Left ≤ Least_So_Far then
15:          Least_So_Far_Holder := p; Least_So_Far := Proc_Time_Left[p]
16:       end if
17:    end for
18:    X(Least_So_Far_Holder,j) := 1; Job_Level[j] :=0
19:    Proc_Time_Left[Least_So_Far_Holder]:= Proc_Time_Left[Least_So_Far_Holder] -
       Least_So_Far
20: end for
21: // Second Phase
22: for p = 0 to P_max -1 do
23:    Best_Job := -1; Best_Level := -1; Best_Value :=0
24:    for j = 0 to J_max -1 do
25:       if X[p,j] == 1 then
26:          v := 1
27:          while v ≤ (V_num -1) and Proc_Time_Left[p] ≥ (Job_Duration[j,v] -
             Job_Duration[j,0] ) do
28:             if Best_Value ≤ (Job_Energy[j,v] -Job_Energy[j,V_num -1] ) then
29:                Best_Job := j; Best_Level := v
30:                Best_Value := (Job_Energy[j,v] -Job_Energy[j,V_num -1] )
31:             end if
32:             v := v+ 1
33:          end while
34:       end if
35:    end for
36:    if Best_Job ≥ 0 then
37:       Job_Level[j] := v -1
38:    end if
39: end for
40: // Energy Calculation
41: for p = 0 to P_max -1 do
42:    Busy := 0
43:    for j = 0 to J_max -1 do
44:       Energy := X(p,j) * Job_Dynamic[j,Job_Level]; Busy := Busy + X(p,j,v)
45:    end for
46:    if Busy > 0 then
47:       L_Energy := L_Energy + Leakage_Value
48:    end if
49: end for
50: Total_Energy := D_Energy + L_Energy
```

$X(p,j,v)$	Interpretation	Scaled?
$X(0,0,4)$	Processor 0 runs job 0 at voltage level 4	No
$X(0,1,4)$	Processor 0 runs job 1 at voltage level 4	No
$X(1,2,2)$	Processor 1 runs job 2 at voltage level 2	Yes
$X(2,3,4)$	Processor 2 runs job 3 at voltage level 4	No

Table 4: $X(p,j,v)$ values determined by the heuristic approach.

not 6 units of available execution time, job 3 is assigned to Processor 2. This completes the first phase of the heuristic algorithm.

In the second phase, each processor is examined in turn and one job is chosen from each processor with a slack for voltage/frequency scaling. Processor 0 has no available free time; so, no job on it can be scaled. Processor 1 has only job 2 running on it. This job can be scaled from level 4 to level 2. This increases its execution time by 2 units, but reduces its energy consumption from 8 units to 5 units. Processor 2 has no slack and hence job 3 which is running on it cannot be scaled. $X(p,j,v)$ values returned by our heuristic approach are shown in Table 4. The dynamic energy spent with this heuristic approach is calculated as follows:

$$\begin{aligned} D\_Energy &:= X(0,0,4) * Job\_Dynamic(0,4) \\ &+X(0,1,4) * Job\_Dynamic(1,4) + X(0,2,2) * Job\_Dynamic(2,2) \\ &+X(0,3,4) * Job\_Dynamic(3,4) \\ &= 1*5 + 1*10 + 1*5 + 1*15 = 35. \end{aligned}$$

As three processors are used, 15 energy units are spent as leakage. This calculation is shown below.

$$L\_Energy = 3 * Leakage\_value = 3*5 = 15$$

As before, the total energy spent is the sum of the dynamic and leakage energies spent. This can be computed as follows:

$$Total\_Energy = D\_Energy + L\_Energy = 35 + 15 = 50$$

In this example, the ILP method saves 3 energy units over the heuristic method. This example also demonstrates that the ILP approach can be used as an upper bound to test the quality of the solutions returned by heuristics.

4. EXPERIMENTAL EVALUATION

We present only energy results in this section. The reason is that none of the techniques evaluated increases original execution cycles (i.e., we do not exceed T_{max} in any loop nest). Specifically, for each loop nest, the processor with the largest workload sets the limit for voltage/frequency scaling and processor shut-down. The ILP solver used in our experiments is lp_solve [9]. We observed that the ILP solution times with the application codes in our experimental suite varied between 56.7 seconds and 13.2 minutes. Considering the large energy savings, these solution times are within tolerable limits.

All the experimental results are obtained using the SIMICS simulation platform [16]. Specifically, we embedded in the SIMICS platform timing and energy models that help us simulate the behavior of the following four schemes: VS (pure voltage/frequency scaling based approach); SD (pure processor shut-down based approach); VS+SD (a unified approach that combines VS and SD); and CLUSTERING (the ILP-based approach proposed in this paper). The default simulation parameters used in our experiments are listed in Table 5. In the last three schemes, when a processor is unused in the current loop nest, it is shut-down and its L1 instruction and data caches are placed into the low-power mode. The specific low-power mode employed in this paper is from [4].

We used 8 array/loop-intensive applications for evaluating the four approaches mentioned above: 3D, DFE, LU, SPLAT, MGRID, WAVE5, SPARSE, and XSEL. 3D is an image-based modeling application that simplifies the task of building 3D models and scenes. DFE is a digital image filtering and enhancement code. LU is an LU decomposition program. SPLAT is a volume rendering application which is used in multi-resolution volume visualization through hierarchical wavelet splitting. MGRID and WAVE5 are C versions of two Spec95FP applications. SPARSE is an image processing code that performs sparse matrix operations, and finally, XSEL is an image rendering code. These C programs are written in such a fashion that they can operate on inputs

Simulation Parameter	Value
Processor Speed	400MHz
Number of Processors	8
Lowest/Highest Voltage Levels	0.8V/1.4V
Number of Voltage Levels	4
Instruction Cache	8KB 2-way associative 32 byte blocks
Data Cache	8KB 2-way associative 32 byte blocks
Memory	32MB (banked)
Off-Chip Memory Access Latency	100 cycles
Bus Arbitration Delay	5 cycles
Replacement Policy	Strict LRU
Cache Dynamic Energy Consumption	0.6 nJ
Memory Dynamic Energy Consumption	1.17 nJ
Leakage Energy Consumption for 32 bytes Normal Operation Shut-Down State	 4.49 pJ 0.92 pJ
Resynchronization Time for Shut-Down State	30 msec
Resynchronization Time for Voltage Scaling	5 msec

Table 5: The default simulation parameters.

of different sizes. We first ran these applications through our simulator without any voltage scaling or processor shut-down. This version of an application is referred to as the base version or base execution in the remainder of this paper. The energy consumptions (which include energies spent in processors, caches, interconnects, and off-chip memory) under the base execution are 272.1mJ, 388.3mJ, 197.9mJ, 208.4mJ, 571.0mJ, 466.2mJ, 292.2mJ, and 401.5mJ for 3D, DFE, LU, SPLAT, MGRID, WAVE5, SPARSE, and XSEL, respectively. The energy results presented in this section are given as *normalized* values with respect to this base execution.

To calculate the dynamic energy consumptions for caches and memory, we used the Cacti tool [19]. We approximated the leakage energy consumption by assuming that the leakage energy per cycle for 4KB SRAM is equal to the dynamic energy consumed per access to a 32 byte data from the same SRAM. Note that this assumption tries to capture the anticipated importance of leakage energy in the future as leakage becomes the dominant part of energy consumption for 0.10 micron (and below) technologies for the typical internal junction temperatures in a chip [3]. In the shut-down state, a processor and its caches consume only a small percentage of their original (per cycle) leakage energy. However, when a processor and its data and instruction caches in the shut-down state are needed, they need to be reactivated (resynchronized). This resynchronization costs extra execution cycles as well as extra energy consumption as noted in [4], and all these costs are captured in our simulations and included in all our results.

Our first set of results, the normalized energy consumptions with the different schemes, are presented in Figure 2. Each group of bars in this graph corresponds to an application, and the last group of bars gives the average results across all eight applications. The energy savings achieved by the VS scheme are not very large (6.55% on the average). There are two main reasons for this. The first one is the inherent characteristics of some applications. More specifically, when there are no long idle periods, VS is not applicable. The second reason is the limited number of voltage/frequency levels used in the default configuration (see Table 5). In comparison, the SD scheme behaves in a different manner. While it is not applicable in some cases (e.g., in applications DFE, MGRID, SPARSE, and XSEL), the energy savings it brings are significant in cases where it is applicable. VS+SD simply combines the benefits of the VS and SD schemes, reducing to VS when SD is not applicable. The average energy savings (across all eight applications) achieved by SD and VS+SD are 7.36% and 13.52%, respectively. The largest energy savings are obtained by our ILP-based approach, which is 22.65% on the average. These results clearly show the potential benefits of our ILP-based workload clustering approach.

To better illustrate where our energy benefits are coming from, we give in Figure 3 the percentage of time each processor spends in the active and idle states for procedure mx3-raw.c, one of the thirteen subprograms in application MGRID. We see from this graph that our ILP-

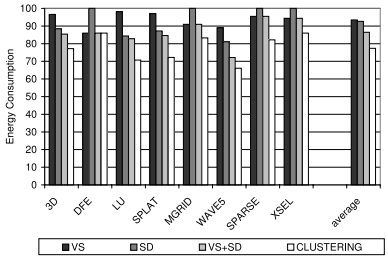

Figure 2: Normalized energy consumption.

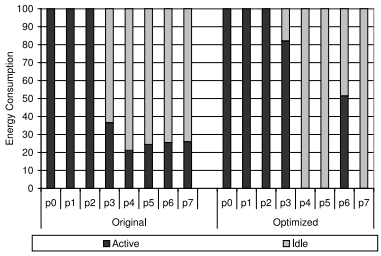

Figure 3: The active and idle periods of processors for the mx3-raw.c routine from MGRID.

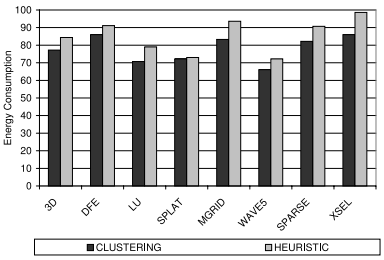

Figure 4: Comparison with the heuristic scheme.

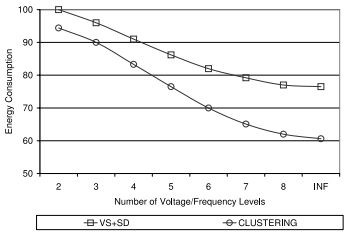

Figure 5: Normalized energy consumption with different voltage/frequency levels.

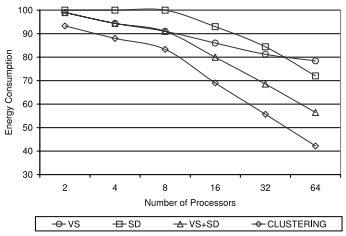

Figure 6: Normalized energy consumption with different processor counts.

based approach is able to increase the number of idle processors. We observed similar trends with most of other procedures in our applications. These results explain the energy benefits observed in Figure 2.

Our second set of results, given in Figure 5, look at the behavior of our approach and VS+SD when the number of available voltage/frequency levels is varied. Each point on the x-axis corresponds to a different number of voltage/frequency levels and INF means an infinite number of levels (i.e., mimicking a continuous voltage/frequency scaling scenario). All other simulation parameters are as shown in Table 5. We can make at least three observations from these results. First, both the approaches take advantage of increased number of voltage/frequency levels. This in a sense should be expected because more voltage/frequency levels means finer granular management of idle periods. Second, for all the voltage/frequency levels tried, our approach generates better results than the VS+SD scheme. This is a direct impact of workload clustering. Third, the gap between the case where we have 8 voltage/frequency levels and continuous scaling (INF) is not great, meaning that we may not need to go beyond 8 levels at all.

The next set of results investigates the influence of the number of processors on our energy savings. They are given in Figure 6. As before, the remaining simulation parameters are set to their default values given in Table 5. Our first observation is that the VS scheme does not scale very well. The main reason for this is the limited number of available voltage/frequency levels in our default configuration. In contrast, the SD version generates really good results as we increase the number of processors. This is due to the fact that the loop parallelizer is not able to take advantage of the increased number of processors, and consequently, many processors remain idle, thereby increasing the opportunities for SD. As expected, the VS+SD version combines the benefits of VS and SD. The largest savings are observed with our clustering-based approach since it is able to take advantage of additional processors by putting them into the low-power mode (if the loop parallelizer is not able to utilize them).

Our last set of experiments compares our ILP-based approach with the heuristic solution presented in Section 3.2.4. The energy consumption results are given in Figure 4. We see from these results that the ILP-based approach generates better results than the heuristic scheme for all the applications tested. The two schemes are very close in SPLAT. However, the energy savings when averaged over all application codes are 22.65% and 14.67%. Still, the heuristic approach generates competitive results with pure voltage scaling and pure processor shut-down.

5. CONCLUSIONS

This paper proposes a workload clustering scheme for embedded MPSoCs that combines voltage scaling with processor shut-down. The uniqueness of the proposed unified approach is that it maximizes the use of processor shut-down by clustering workloads (jobs) in as few processors as possible. We tested this approach along with three alternate schemes using a simulation-based platform and eight embedded applications. Our experiments show that this clustering approach is very effective in reducing energy consumption and generates better results than the three alternative schemes evaluated. Our results also show that the savings brought by this approach increases as the number of voltage/frequency levels or the number of processors is increased. We also designed and implemented a heuristic solution to workload clustering, and compared it to our ILP approach.

6. REFERENCES

[1] D. Brooks, V. Tiwari, and M. Martonosi, Wattch: A Framework for Architectural-Level Power Analysis and Optimizations, *Proceedings of ISCA'2000.*

[2] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. *Proceedings of ISCA'2000.*

[3] A. Chandrakasan, W. J. Bowhill, and F. Fox. *Design of High-Performance Microprocessor Circuits*. IEEE Press, 2001.

[4] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple techniques for reducing leakage power. *Proceedings of ISCA'2002.*

[5] K. Govil, E. Chan, and H. Wasserman. Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU. *Proceedings of the 1st ACM International Conference on Mobile Computing and Networking,* November 1995.

[6] D. Grunwald, P. Levis, K. Farkas, C. Morrey III, and M. Neufeld. Policies for Dynamic Clock Scheduling. *Proceedings OSDI'2000.*

[7] C.-H. Hsu and U. Kremer. Dynamic Voltage and Frequency Scaling for Scientific Applications. *Proceedings of the 14th Workshop on Languages and Compilers for Parallel Computing,* August 2001.

[8] I. Kadayif, M. Kandemir, and U. Sezer. An Integer Linear Programming Based Approach for Parallelizing Applications in On-Chip Multiprocessors. In *Proceedings of DAC'2002.*

[9] lp_solve. ftp://ftp.es.ele.tue.nl/pub/lp_solve/

[10] MAJC-5200. http://www.sun.com/microelectronics/ MAJC/5200wp.html

[11] MP98: A Mobile Processor. http://www.labs.nec.co.jp/MP98/top-e.htm.

[12] T. Okuma, T. Ishihara, and H. Yasuura. Real-Time Task Scheduling for a Variable Voltage Processor. *Proceedings of the 12th International Symposium on System Synthesis,* 1999.

[13] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The Case for a Single Chip Multiprocessor. *Proceedings of ASPLOS'1996.*

[14] H. Saputra, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, J. S. Hu, C-H. Hsu, and U. Kremer. Energy-Conscious Compilation Based on Voltage Scaling. *Proceedings of ACM SIGPLAN Joint Conference LCTES'02 and SCOPES'02,* Berlin , Germany, June, 2002.

[15] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. *Proceedings of the International Conference on Computer-Aided Design,* November 2000.

[16] SIMICS. http://www.virtutech.com/simics/simics.html.

[17] J. P. Singh and D. Culler. *Parallel Computer Architecture: A Hardware-Software Approach,* Morgan-Kaufmann, 1998.

[18] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for Reduced CPU Energy. *Proceedings of OSDI'1994.*

[19] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits,* May 1996.

[20] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal on-line methods for voltage/frequency control in multiple clock domain microprocessors. *Proceedings of ASPLOS'2004.*