CHAPTER 19

WORKLOAD CLUSTERING FOR INCREASING ENERGY SAVINGS ON EMBEDDED MPSoCs

OZCAN OZTURK, MAHMUT KANDEMIR, and SRI HARI KRISHNA NARAYANAN

19.1 INTRODUCTION

We can roughly divide the efforts on energy savings in embedded multiprocessor system-on-a-chip (MPSoC) architecture into two categories. In the 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 workload 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, that is, voltage scaling and processor shutdown, can be applied at the software level (e.g., directed by an optimizing compiler) or 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 shutdown-based scheme may not be applicable if there is no unused processor (note that this does not mean that the workload 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

Energy-Efficient Distributed Computing Systems, First Edition.
Edited by Albert Y. Zomaya and Young Choon Lee.

only choice). On the basis of this discussion, one can expect a unified scheme to be successful. However, we want to reiterate 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 chapter is to explore a workload (job) clustering scheme that combines voltage scaling with processor shutdown.[1] The uniqueness of the proposed unified approach is that it maximizes the opportunities for processor shutdown by carefully assigning workload to processors. It achieves this by clustering the original workload of processors in as few processors as possible. In this chapter, we discuss the technical details of this approach to energy saving in embedded MPSoCs. The proposed approach is based on ILP (integer linear programing); that is, it determines the optimal workload clustering across the processors by formulating the problem using ILP and solving it using a linear solver. In order to check whether this approach brings any energy benefits over pure voltage scaling, pure processor shutdown, 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-based approach (i) is very effective in reducing the energy consumptions of the applications tested and (ii) generates much better energy savings than all the alternate schemes tested (including one that combines voltage/frequency scaling and processor shutdown).

19.2 EMBEDDED MPSoC ARCHITECTURE, EXECUTION MODEL, AND RELATED WORK

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 architecture from academia and industry fit in with this description [1–4]. We keep the subsequent discussion simple using a shared bus as the interconnect (although one could use fancier/higher bandwidth interconnects as well). We also use the MESI protocol (the choice is orthogonal to the focus of this chapter) to keep the caches coherent across the CPUs. We assume that voltage level and frequency of each processor in this architecture can be set independent of the others and also that processors can be placed into low power modes independently. This chapter 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

[1] In this chapter, we use the terms *processor showdown* and *low power mode* interchangeably.

executes a large number of iterations and accesses/manipulates large data sets (typically multidimensional arrays of signals). We employ a loop-nest-based application parallelization strategy. More specifically, each loop nest is parallelized independent of the others. In this context, parallelizing a loop nest means distributing its iterations across processors and allowing processors to execute their portions parallelly. For example, a loop with 1000 iterations can be parallelized across 10 processors by allocating 100 iterations to each processor.

There are many proposals for power management of a processor capable of dynamic voltage scaling. Most of them are at the operating system level and are either task based [5] or interval based [6]. While some proposals aim at reducing energy without compromising performance, a recent study by Grunwald et al [7] observed noticeable performance loss for some interval-based algorithms using actual measurements. Most of the existing compiler-based studies, such as Reference 8, target single-processor architecture. In comparison, our work targets at a chip multiprocessor-based environment and combines voltage scaling and processor shutdown. Wu et al. [9] present and analyze a voltage/frequency scaling scheme, but they do not consider processor shutdown. Kadayif et al. [10] employs processor a shutdown-based mechanism but does not consider voltage/frequency scaling. In our experimental evaluation, we compare our approach to pure voltage/frequency scaling and also to pure processor shutdown.

19.3 OUR APPROACH

19.3.1 Overview

Figure 19.1 compares four different alternate schemes that save energy in an embedded MPSoC architecture. It is assumed, for illustrative purposes, that the architecture has six processors. Figure 19.1a 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. Figure 19.1b and 19.1c show the scenarios with approaches based on pure voltage/frequency scaling and pure processor shutdown, respectively. In (b), four out of our 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 processor (P_5) in the low power mode. A combination of these two approaches is depicted in Figure 19.1d. Basically, this version combines the benefits of voltage/frequency scaling and processor shutdown. Finally, the result that can be obtained by the ILP approach proposed in this chapter is illustrated in Figure 19.1e. Note that what our approach essentially does 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 load of three processors (P_2, P_3, and P_4) is combined and assigned to processor P_2. As a result, processors P_3 and P_4 can be also placed in the low power mode (along with their private memory components) to maximize energy savings, in addition to P_5. The next section gives the technical details

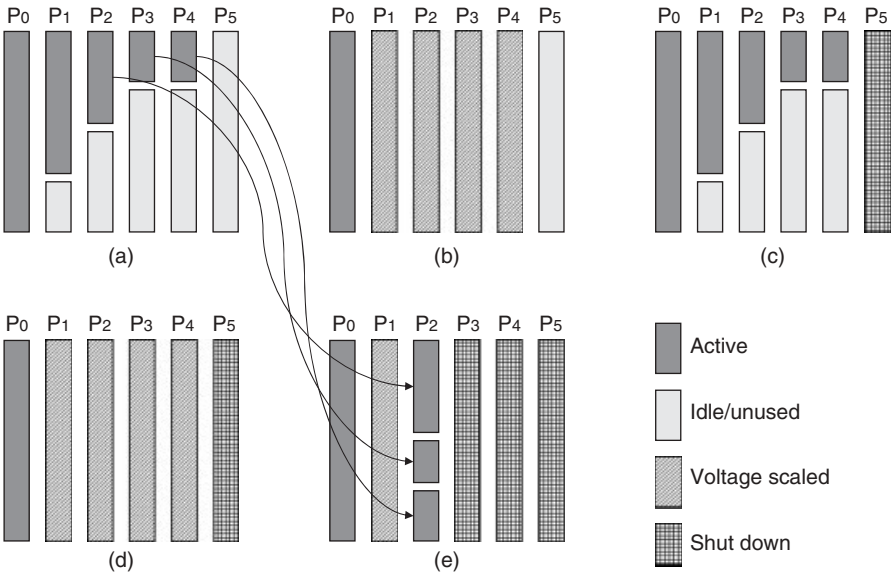

Figure 19.1 Comparison of different energy-saving approaches for a six-processor architecture. Arrows indicate how the workloads (jobs) are clustered by our approach.

of this approach. When there are opportunities, our approach can also use voltage/frequency scaling for the clustered jobs. It is important to point out that the benefits from our approach can be expected to be even more significant when the number of voltage/frequency levels is small. In such a case, an approach based on pure voltage/frequency scaling cannot stretch the execution time of a processor to fill the available slack completely.

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 19.1a) not been parallelized at the first place as shown in Figure 19.1e?" 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 workload 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 clustering, being on the conservative side.

The second issue is that it is possible that the scenario depicted in Figure 19.1e has poor data locality as compared to scenarios in Figure 19.1b, 19.1c, and 19.1d. 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 19.1e 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 chapter is to obtain, for each loop nest in an application, the result shown in Figure 19.1e, given the initial scenario (workload assignment) shown in Figure 19.1a, and thus reduce energy consumption.

19.3.2 Technical Details and Problem Formulation

This section elaborates on the ILP model used to represent the problem. In our problem, there exist a set of jobs (workloads) that have to be executed on a set of available processors in the embedded MPSoC such that the total energy spent by the system is minimal and the execution of the jobs is completed 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.

19.3.2.1 System and job model. We assume that the 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 be assigned to run more than one job, as a result of workload clustering. The duration for which the job occupies the processor is dependent on the supply voltage/frequency as well as 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 captured by 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.

---

[2]In this chapter, 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 loop nest, $T_{\max}$ is determined by the processor with the largest workload. This is to ensure that our workload 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 ofJob_Length(j, v) and Job_Dynamic(j, v) are filled using profiling. All energy estimations are performed using Wattch [11] under the 70-nm process technology. The increase in data cache misses as a result of clustering is captured during our profiling.

TABLE 19.1 Notation Used in Our Model

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
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

19.3.2.2 Mathematical programing model. The constraints specified below give the mathematical representation of our model. We use 0-1 ILP. This ILP formulation is executed for each loop nest separately. Table 19.1 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 job runs completely on one processor, and all jobs are scheduled to run only once. This is specified as follows:

$$\forall\, p \in P \quad \forall j \in J \quad \forall v \in V \quad X(p, j, v) \in \{0|1\}, \tag{19.1}$$

$$\forall\, j \in J \sum_{p=0}^{P_{\max}-1} \sum_{v=0}^{V_{\text{num}}-1} X(p, j, v) = 1. \tag{19.2}$$

Constraint 19.1 expresses the term $X(j, p, v)$ as a binary variable; a processor either runs the job or it does not. Constraint 19.2 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 19.3 expresses this:

$$\forall p \in P \sum_{j=0}^{J_{\max}-1} \sum_{v=0}^{V_{\text{num}}-1} X(p, j, v) * \text{Job\_Length}(j, v) \leq T_{\max}. \tag{19.3}$$

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

Clustering and Processor Shutdown Constraints. Multiple jobs are run on the same processor not only 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, because of clustering of jobs, 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 since it is shut down (except for some small amount of leakage in memory components). Constraint 19.4 is introduced to capture processor shutdown:

$$\forall p \in P, \forall j \in J, \forall v \in V \quad \text{Busy}(p) \geq X(p, j, v). \tag{19.4}$$

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. The total amount of 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 19.5:

$$\text{D\_Energy} = \sum_{p=0}^{P_{\max}-1} \sum_{j=0}^{J_{\max}-1} \sum_{v=0}^{V_{\text{num}}-1} X(p, j, v) * \text{Job\_Energy}(j, v). \tag{19.5}$$

Expression (19.6) calculates the leakage energy spent. As mentioned earlier, if Busy(p) is 1, then leakage is spent by processor p.

$$\text{L\_Energy} = \text{Leakage\_Value} * \sum_{p=0}^{P_{\max}-1} * \text{Busy}(p). \tag{19.6}$$

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

$$\text{TotaL\_Energy} = \text{D\_Energy} + \text{L\_Energy}. \tag{19.7}$$

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.

[4]To preserve data in memory components, a shutdown processor consumes some leakage [12]. Our experiments are performed based on this principle. However, in our presentation of the ILP formulation, we assume no leakage consumption in the shutdown state for ease of presentation.

Additional Constraints. If two or more jobs run on the same processor, the order in which they are executed may be important and this can be found out by the following constraints. The term $\text{Seq}(p, j_1, j_2)$ is defined as being 1 if j_1 and j_2 both run on processor p and j_1 precedes j_2 in execution. Constraint 19.8 specifies $\text{Seq}(p, j_1, j_2)$ as being binary, and constraint 19.9 specifies that two jobs cannot both precede each other:

$$\forall p \in P, \forall j_1 \in J, \forall j_2 \in J | j_1 \neq j_2 \\ \text{Seq}(p, j_1, j_2) \in \{0|1\}, \tag{19.8}$$

$$\forall p \in P, \forall j_1 \in J, \forall j_2 \in J | j_1 \neq j_2 \text{Seq}(p, j_1, j_2) + \\ \text{Seq}(p, j_2, j_1) \leq . \tag{19.9}$$

Constraint 19.10 links $X(j, p, v)$ and $\text{Seq}(p, j_1, j_2)$ by stating that two processors need be sequenced only if they are executed on the same processor:

$$\forall\ p \in P, \forall j_1 \in J, \forall j_2 \in J | j_1 \neq j_2 \text{Seq}(p, j_1, j_2) \geq \\ [1 - \text{Seq}(p, j_2, j_1)] * \left(\sum_{v1=0}^{V_{\text{num}}-1} X(j_1, p, v1) \right. \\ \left. + \sum_{v2=0}^{V_{\text{num}}-1} X(j_2, p, v2) - 1 \right). \tag{19.10}$$

Finally, constraint 19.11 states the transitive nature of sequenced jobs. That is, if job j_1 precedes j_2 and j_2 precedes j_3 then job j_1 necessarily precedes job j_3.

$$\forall p \in P, \forall j_1 \in J, \forall j_2 \in J \forall j_3 \in J | j_1 \neq j_2 \neq j_3, \\ \text{Seq}(p, j_1, j_3) \geq \text{Seq}(p, j_1, j_2) + \text{Seq}(p, j_2, j_3) - 1. \tag{19.11}$$

Voltage/Frequency Scaling without Clustering. To model classical voltage/frequency scaling within our ILP formulation, an input value $\text{Assign}(j, p)$ should specify the processor on which each job runs. Furthermore, 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 captured by the following constraint:

$$\forall p \in P, \forall j \in J \sum_{v=0}^{V_{\text{num}}-1} X(p, j, v) = \text{Assign}(p, j). \tag{19.12}$$

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 19.13 can be used to ensure that no jobs are assigned voltage levels other than the one specified.

$$\forall p \in P, \forall j \in J, \forall v \in V - \{v'\} \quad X(p, j, v) = 0. \tag{19.13}$$

To constrain each individual processor to an independent voltage level, constraint 19.14 can be used:

$$\forall p \in P, \forall j \in J, \forall v \in V - \{v'_p\} \quad X(p, j, v) = 0. \tag{19.14}$$

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.

19.3.2.3 Example. This section presents an example and demonstrates how the ILP method and the heuristic method operate in practice. Table 19.2 shows the constant parameters for the system. There are four jobs (workloads) to be run on four processors. Each job can be run at five 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:

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

TABLE 19.2 Constant Parameters Used in the Example

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

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:

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

$X(p, j, v)$ values returned by our ILP solver are presented in Table 19.3. 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 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:

$$\begin{aligned} \text{D\_Energy} &= X(0,0,3) * \text{Job\_Dynamic}(0,3) + X(0,2,2) * \text{Job\_Dynamic}(2,2) \\ &\quad + X(0,1,3) * \text{Job\_Dynamic}(1,3) + X(0,3,4) * \text{Job\_Dynamic}(3,4) \\ &= 1 * 4 + 1 * 5 + 1 * 8 + 1 * 15 = 32. \end{aligned}$$

Since three processors are used, 15 energy units are spent as leakage. This calculation can be given by

$$\text{L\_Energy} = 3 * \text{Leakage\_Value} = 3 * 5 = 15.$$

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

$$\text{Total\_Energy} = \text{D\_Energy} + \text{L\_Energy} = 32 + 15 = 47.$$

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 units). 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

TABLE 19.3 $X(p, j, v)$ Values Determined by the ILP Approach

$X(p, j, v)$	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

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 not require 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 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 19.4. The dynamic energy spent with this heuristic approach is calculated as follows:

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

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

$$\text{L\_Energy} = 3 * \text{Leakage\_Value} = 3 * 5 = 15.$$

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

$$\text{Total\_Energy} = \text{D\_Energy} + \text{L\_Energy} = 35 + 15 = 50.$$

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.

TABLE 19.4 $X(p, j, v)$ Values Determined by the Heuristic Approach

$X(p, j, v)$	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

19.4 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 shutdown. The ILP solver used in our experiments is lp_solve [13]. We observed that the ILP solution times with the application codes in our experimental suite varied between 56.7 s and 13.2 min. Considering the large energy savings, these solution times are within tolerable limits.

All the experimental results are obtained using the SIMICS simulation platform [14]. 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 shutdown-based approach), VS+SD (a unified approach that combines VS and SD), and CLUSTERING (the ILP-based approach proposed in this chapter). The default simulation parameters used in our experiments are listed in Table 19.5. 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 in the low power mode. The specific low power mode used in this chapter is from Reference 12.

TABLE 19.5 The Default Simulation Parameters

Simulation Parameter	Value
Processor speed	400 MHz
Number of processors	8
Lowest/highest voltage levels	0.8 V/1.4 V
Number of voltage levels	4
Instruction cache	8 KB Two-way associative 32-B blocks
Data cache	8 KB Two-way associative 32-B blocks
Memory	32 MB (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 B	
Normal operation	4.49 pJ
Shutdown state	0.92 pJ
Resynchronization time for shutdown state	30 ms
Resynchronization time for voltage scaling	5 ms

We used eight array/loop-intensive applications for evaluating the four approaches mentioned earlier: 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 that is used in multiresolution 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 of different sizes. We first ran these applications through our simulator without any voltage scaling or processor shutdown. This version of an application is referred to as the base version or the base execution in the remainder of this chapter. The energy consumptions (which include energies spent in processors, caches, interconnects, and off-chip memory) under the base execution are 272.1, 388.3, 197.9, 208.4, 571.0, 466.2, 292.2, and 401.5 mJ 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 [15]. We approximated the leakage energy consumption by assuming that the leakage energy per cycle for 4-KB Static Random Access Memory (SRAM) is equal to the dynamic energy consumed per access to a 32-B 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-? (and below) technologies for the typical internal junction temperatures in a chip. In the shutdown 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 shutdown state are needed, they need to be reactivated (resynchronized). This resynchronization costs extra execution cycles as well as extra energy consumption as noted in Reference 12, 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 19.2. Each group of bars in this graph correspond 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 is not very large (6.55% on 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 (Table 19.5). 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 is 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

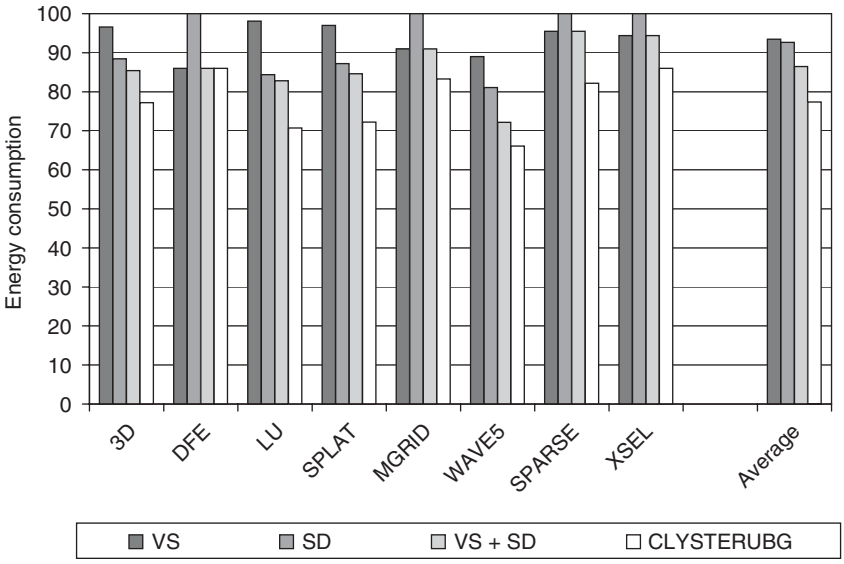

Figure 19.2 Normalized energy consumptions.

all eight applications) achieved by SD and VS + SD are 7.36% and 13.52%, respectively. The highest energy savings is obtained by our ILP-based approach, which is 22.65% on 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 19.3 the percentage of time each processor spends in the active and idle states for procedure mx3-raw.c, one of the 13 subprograms in application MGRID. We see from this graph that our ILP-based approach is able to increase the number of idle processors. We observed similar trends with most of the other procedures in our applications. These results explain the energy benefits observed in Figure 19.2.

Our second set of results, given in Figure 19.4, looks 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 19.5. One can make 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 eight voltage/frequency levels and continuous scaling (INF) is not great, meaning that we may not need to go beyond eight levels at all.

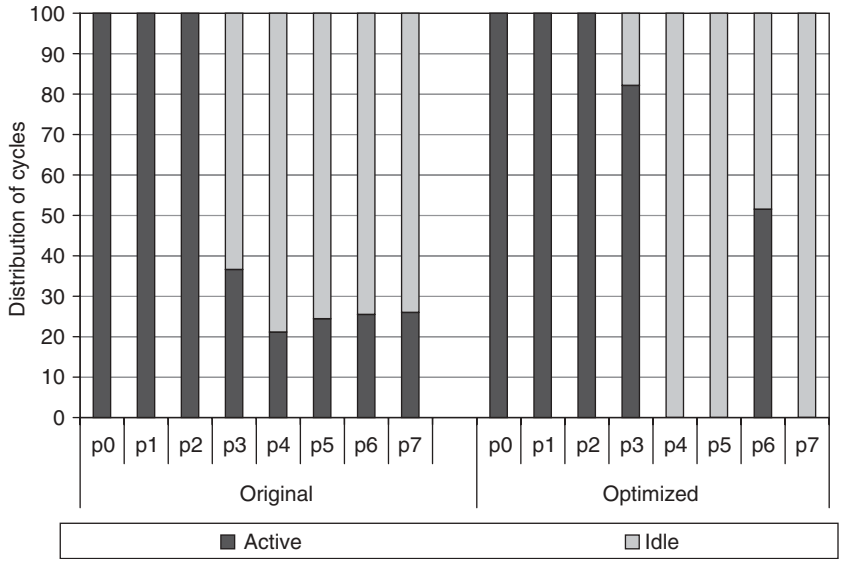

Figure 19.3 The active and idle periods of processors in the mx3-raw.c routine from MGRID.

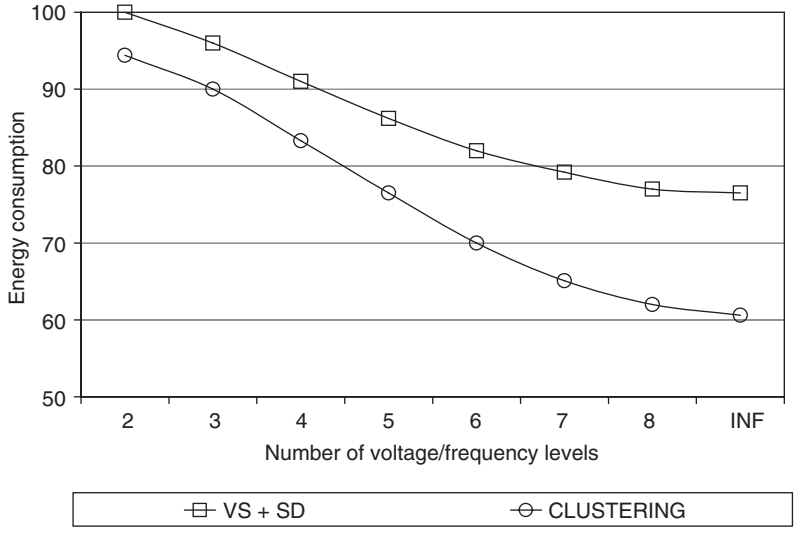

Figure 19.4 Normalized energy consumption with different voltage/frequency levels.

Our last set of results investigates the influence of the number of processors on our energy savings. They are given in Figure 19.5. As before, the remaining simulation parameters are set to their default values given in Table 19.5. Our first observation is that the VS scheme does not scale very well. The main reason for

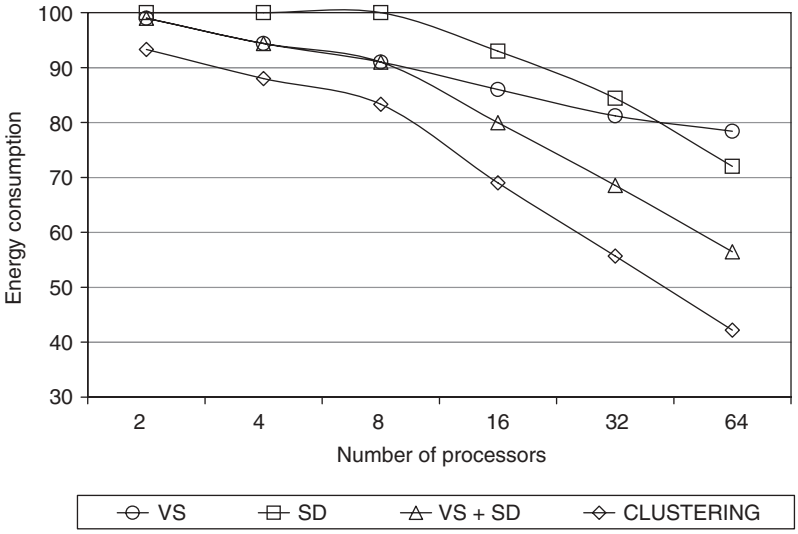

Figure 19.5 Normalized energy consumptions with different processor counts.

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 highest savings is observed with our clustering-based approach since it is able to take advantage of additional processors by putting them in the low power mode (if the loop parallelizer is not able to utilize them).

19.5 CONCLUSIONS

This chapter proposes a workload clustering scheme for embedded MPSoCs that combines voltage scaling with processor shutdown. The uniqueness of the proposed unified approach is that it maximizes the use of processor shutdown 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.

REFERENCES

1. Barroso LA, Gharachorloo K, McNamara R, Nowatzyk A, Qadeer S, Sano B, Smith S, Stets R, Verghese B. Piranha: a scalable architecture based on single-chip multiprocessing. In: Proceedings of ISCA'2000; Vancouver BC, Canada.
2. Olukotun K, Nayfeh BA, Hammond L, Wilson K, Chang K. The case for a single chip multiprocessor. In: Proceedings of ASPLOS'1996; Cambridge, Massachusetts.
3. Sudharsanan S. MAJC-5200: A High Performance Microprocessor for Multimedia Computing, Parallel and Distributed Image Processing, Video Processing, and Multimedia (PDIVM) Workshop, Cancun, Mexico, 2000.
4. Edahiro M, Matsushita S, Yamashina M, Nishi N. A Single-Chip Multiprocessor for Smart Terminals, IEEE Micro, volume 20, pp 12–20, 2000.
5. Shin Y, Choi K, Sakurai T. Power optimization of real-time embedded systems on variable speed processors. In: Proceedings of the International Conference on Computer-Aided Design; California, USA; 2000 Nov.
6. Govil K, Chan E, Wasserman H. Comparing algorithms for dynamic speed-setting of a low-power CPU. In: Proceedings of the 1st ACM International Conference on Mobile Computing and Networking; California, USA; 1995 Nov.
7. Grunwald D, Levis P, Farkas K, Morrey C III, Neufeld M. Policies for dynamic clock scheduling. In: Proceedings OSDI'2000; California, USA.
8. Saputra H, Kandemir M, Vijaykrishan N, Irwin M, Hu J, Kremer U. Energy-conscious compilation based on voltage scaling. In: Proceedings of ACM SIGPLAN Joint Conference LCTES'02 and SCOPES'02; 2002 Jun; Berlin, Germany.
9. Wu Q, Juang P, Martonosi M, Clark DW. Formal on-line methods for voltage/frequency control in multiple clock domain microprocessors. In: Proceedings of ASPLOS'2004; Massachusetts, USA.
10. Kadayif I, Kandemir M, Sezer U. An integer linear programming based approach for parallelizing applications in on-chip multiprocessors. In: Proceedings of DAC'2002; New Orleans, Louisiana.
11. Brooks D, Tiwari V, Martonosi M. Wattch: a framework for architectural-level power analysis and optimizations In: Proceedings of ISCA; 2000; Canada.
12. Flautner K, Kim N, Martin S, Blaauw D, Mudge T. Drowsy caches: simple techniques for reducing leakage power. In: Proceedings of ISCA; 2002.
13. Notebaert P. lp _solve. Available at ftp://ftp.es.ele.tue.nl/pub/lp_solve/; 2012.
14. Tian T. Improving the Embedded Intel Architecture Design Process with Simics, Intel White Paper, November 2011. http://download.intel.com/design/intarch/papers/326341.pdf.
15. Wilton S, Jouppi N. Cacti: an enhanced cache access and cycle time model. IEEE J Solid State Circ 1996;31(5):677–688.