SESSION TA2:

Embedded Systems, Sensors and MEMS

Chair: Christopher Ryan, Vitesse
Co-chair: Kenneth Hsu, Rochester Institute of Technology
Workload Clustering for Increasing Energy Savings on Embedded MPSoCs

S.H.K. Narayanan, O. Ozturk, M. Kandemir
Department of Computer Science and Engineering
The Pennsylvania State University
{snarayan, ozturk, kandemir}@cse.psu.edu

M. Karakoy
180 Queen's Gate
Imperial College
mk22@doc.ic.ac.uk

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 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-based approach (1) is very effective in reducing the energy consumptions 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).

I. INTRODUCTION
We can roughly divide the efforts on energy savings in embedded multi-processor system-on-a-chip architectures (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 shutdown 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 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 based approach (1) is very effective in reducing the energy consumptions 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).

II. 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 architectures from academia and industry fit in this description [1, 10, 8, 9]. 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 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 multidimensionsal 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 itera-

\(^1\)In this paper, we use the terms "processor show-down" and "low-power mode" interchangeably.
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(c) 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(c) 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.

III. OUR APPROACH

III.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 approaches, respectively. In (b), four out of six processors take advantage of voltage scaling (note that P3 is not used in the computation at all). In (c), on the other hand, we can place only one processor (P2) 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 be obtained by the ILP approach proposed in this paper is illustrated in Figure 1(e). 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 loads of three processors (P2, P3, and P4) are combined and assigned to processor P2. As a result, processors P3 and P4 can also be placed into the low-power mode (along with their private memory components) to maximize energy savings, in addition to P3. 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. 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, a pure voltage/frequency scaling based approach 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 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,
job runs completely on one processor and all jobs are scheduled to run only once. This is specified as follows:

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

(1)

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

(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 meet the time deadline that is specified. Constraint (3) expresses this:

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

(3)

Note that $T_{max}$ is determined, for each loop nest, by the (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 processors, $P_{max}$, 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 since it is shut down (except for some small amount of leakage in memory components). Constraint (4) is introduced to capture processor shutdown:

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

(4)

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

**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 (5):

$$D.Energy = \sum_{p=0}^{P_{max}-1} \sum_{j=0}^{J_{max}-1} \sum_{v=0}^{V_{num}-1} X(p,j,v) \cdot \text{Job.Length}(j,v)$$

(5)

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

$$L.Energy = \text{Leakage.Value} \cdot \sum_{p=0}^{P_{max}-1} \cdot \text{Busy}(p)$$

(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 objective function that our approach tries to minimize:

$$\text{Total.Energy} = D.Energy + L.Energy$$

(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 $\text{Assign}(j,p)$ 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 captured by the following constraint:

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

(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 no jobs are assigned voltage levels other than the one specified.

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

(9)

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

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

(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.

**IV. 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 [7]. 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 [13]. 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 2. In the last three schemes, when a processor is unused in the current loop nest, it

\[ \text{To preserve data in memory components, a shut-down processor consumes some leakage} [3]. \text{ Our experiments are performed based on this principle. However, in our presentation of the ILP formulation, we assume no leakage consumption in the shut-down state for ease of presentation.} \]
There are characteristics of different simulations and included resynchronization below.) The energy consumption is derived from [3].

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-rawc, one of the thirteen 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 other procedures in our applications. These results explain the energy benefits observed in Figure 2.

V. 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 alternative schemes using a simulation-platform based framework 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.

VI. REFERENCES


