Energy Optimization on OpenMP Loop Scheduling

Yong Dong
National University of Defense Technology, Changsha, 410073,China
Email: yongdong@nudt.edu.cn

Juan Chen
National University of Defense Technology, Changsha, 410073,China
Email: juanchen@nudt.edu.cn

***Abstract*—Energy consumption has become a serious problem in high-performance computing (HPC) systems. Parallel loops often occupy a significant part of the execution time of overall parallel programs. Thus, reducing their energy consumption is the key to the reduction in energy consumption of the program. This paper discusses energy optimization in OpenMP loop scheduling, which is a method of optimizing energy consumption of parallel loops within a certain performance loss. OpenMP programming includes Uniform-Sized scheduling unit (US)-based Energy-saving Optimal Static Scheduling algorithm (EOSS), UnUniform-Sized scheduling unit (UUS)-based Energy-oriented Static Scheduling algorithm (ESS). EOSS obtains the maximum energy savings under the ideal condition (US) by combining loop rescheduling and voltage/frequency scaling. ESS optimizes energy consumption under the non-ideal condition (UUS). We evaluate these energy-oriented loop scheduling algorithms through simulations on a multiprocessor system. Taking 160 processors as example, five NPB programs with EOSS can reduce 43.5% energy consumption on the average, which is 2.1% more energy saved than the DVFS-only method. ESS can achieve 54% energy savings with the example code we use.**

***Index Terms*—OpenMP loop scheduling, dynamic voltage frequency scaling, processor shutdown, energy optimization**

I. INTRODUCTION

As high performance computing (HPC) systems become more powerful, their energy consumption continues to increase as well. For instance, the average power consumption of the Top 10 systems in the TOP500 list is nearly 2000 KW [1] among which the power consumption of the Jaguar system is nearly 7000 KW. The power of HPC system is expected to further grow to 100 MW [2] in the next few years. High energy consumption results in many problems to HPC systems, such as low reliability, bad stability, and high costs. Thus, energy saving in parallel computing has become an important issue. For a large number of parallel scientific programs, parallel loops often account for majority of the entire execution time. As energy is the product of power and time, reducing the energy of parallel loops is the key to reducing the energy of the whole program as well.

OpenMP [3] is a popular shared-memory parallel programming interface. With the increasing number of cores in a processor chip, the natural parallelism of OpenMP makes it more widely used in HPC systems, and makes it one of the most indispensable multithread programming models. Thus, optimizing the energy consumption of OpenMP loops is important. Our optimization goal is to implement user-transparent OpenMP loop scheduling with energy optimization processor dynamic voltage/frequency scaling (DVFS). There is no doubt that such energy-oriented OpenMP loop scheduling algorithm is very attractive for users. In this paper, we focus on the obtainment of as much energy saving as possible in theory, or what is termed as the upper bound of energy saving.

OpenMP loop scheduling is different from ordinary parallel nested loop scheduling in many aspects. Loop-carried dependences generally exist in parallel nested loops, which increase the complexity of loop parallelization. Thus, estimating statically the execution time of each processor is difficult, which in turn results in problems in the use of energy optimization methods such as dynamic voltage/frequency scaling. Profiling can effectively obtain the run time characteristic of a program, but its usability is limited. In comparison, in OpenMP loop scheduling, the dependence among loop iterations does need not to be considered. Programmers guarantee the correctness of loop scheduling. Scheduling strategy and chunk size are the key factors that influence loop performance. An oversized chunk leads to load imbalance, whereas a smaller chunk reduces performance and destroys data locality. These two factors are also important for the optimization of energy consumption.

DVFS is an effective low-power technique that clearly balances energy savings and performance loss of the processor. It has been applied in some high-performance processors, such as Intel Xeon processor (Enhanced Intel SpeedStep® Technology) [4] and AMD Opteron processor (Enhanced AMD PowerNow!™ Technology) [5]. Such DVFS-capable processors provide hardware support to research on low-power software.

In massively parallel systems with multiprocessors, the speed of processors assigned with less workload can be decreased by DVFS. Therefore, their execution time of a task remains the same with the processors assigned with the maximum workload. As a result, the energy consumption of all processors is reduced. This DVFS policy can also be used in OpenMP loop scheduling.

© 2012 ACADEMY PUBLISHER
doi:10.4304/jsw.7.8.1694-1705

However, is DVFS the only way to save energy? Should more opportunities be explored to maximize energy saving, such as changing the loop scheduling algorithm itself?

To answer the above questions, we examine the optimal energy saving of OpenMP static loop scheduling. We focus on the impact of different scheduling policies on DVFS, and take both scheduling policies and DVFS into consideration. We also use the processor shutdown technique to optimize the energy consumption of OpenMP dynamic loop scheduling.

In this paper, *energy consumption* refers to the energy of processors needed to complete a parallel loop, excluding the energy required of other components such as storage system, network, and so on. *Energy optimization* is the improvement in energy consumption under a certain performance loss. *Performance* is the execution time of OpenMP loop. *Performance loss* is the delay arising from DVFS or processor shutdown. Generally, the more performance loss is allowed, the more opportunities are given to DVFS, and the greater the amount of energy savings that can be achieved. With a given performance loss, energy saving has its upper bound in theory. We want to build a model to maximize the energy saving of OpenMP loop scheduling under some assumptions, which is briefly expressed as *energy optimization with the given performance constraint*.

In this paper, the energy optimization problem is divided into two types according to whether the execution time of each outermost loop is the same or not. If the execution time of each outermost loop is the same (called US), a *US-based **E**nergy-saving **O**ptimal **S**tatic **S**cheduling algorithm* (**EOSS**) is derived. **EOSS** combines scheduling chunk size scaling and DVFS for maximum energy saving, and its optimality is proven. When the execution time of each outermost loop is not the same (called UUS), an energy-oriented algorithm—the *UUS-based **E**nergy-oriented **S**tatic **S**cheduling algorithm* (**ESS**)—is proposed.

The optimal solution only exists under some given hypothetical conditions. The optimal solution of **EOSS** is the upper bound in theory, and it is not exactly the same as the actual solution. When other factors like data locality and synchronization are taken into account, the solution proposed by **EOSS** should be reconsidered. Although some given hypothetical conditions in this article sound ideal and are better than actual cases, they are helpful for research on optimality. Thus, such kind of optimistic hypothesizing is acceptable. Prospective research on energy-saving optimality provides a theoretical upper bound as well as the basis for energy optimization in the future. Therefore, from this point of view, our theoretical research has much practical significance.

We extend the previous work [19] about **EOSS** algorithm, and propose two energy optimization algorithms for OpenMP loop scheduling:

- *US-based **E**nergy-saving **O**ptimal **S**tatic **S**cheduling algorithm* (**EOSS**)
- *UUS-based **E**nergy-oriented **S**tatic **S**cheduling algorithm* (**ESS**)

The rest of this paper is organized as follows. Section 2 reviews the related literature. Section 3 gives the detailed description of the two algorithms. Section 4 gives the experimental results and the evaluation of the proposed algorithms. Section 5 concludes this paper.

II. Related Work

The power consumption of processors encompasses dynamic power, short circuit power, and leakage power [6], among which dynamic power accounts for a major fraction. Energy is considered the accumulation of power over time. In this article, we focus on reducing the dynamic energy consumption of processors. Related studies, on the other hand, are limited to optimizing dynamic energy consumption.

In multiprocessor systems, the dynamic energy consumption of processors is influenced primarily by the number of running processors and the power state of each processor. A large number of related studies on effectively optimizing the energy consumption of multiprocessors exist. Adaptively adjusting the number of processor cores as the program runs is an important energy optimization policy. Kadayif et al. [7] proposed an adaptive parallel loop policy for on-chip multiprocessor (CMP) architectures. During the running process of nested loops, using a portion of processor cores is more efficient than employing all of them because of loop-carried dependencies. Too many processor cores result in an increase in communication time generated by them and growth of barrier overhead among processor cores. Furthermore, an excessive number of processor cores also cause wastage from unnecessary energy consumption. The adaptive parallel loop policy allows each nested loop to choose an individual number of processor cores while shutting down unused ones during program runs. Kadayif et al. found that the number of processor cores that yields the best results is usually much smaller than the total number. Furthermore, the authors [8] derived an ILP formulation to describe the foregoing problem, and obtained the optimal number of processor cores by solving this ILP formulation.

Aside from adjusting the number of processor cores, scaling down the voltage/frequency of processors is another common method for reducing the energy consumption of processors. In practical applications, maximizing the computing ability of processors is sometimes unnecessary. Allotting more computing power than what the program requires leads to energy wastage. Using multi-level voltage/frequency scaling as basis, Kadayif et al. [9] developed a tradeoff between performance and energy consumption, and set each processor with the appropriate voltage/frequency—instead of the highest—to save energy. They adopted compiler analysis techniques to exploit the load imbalance between processor cores. The differences in energy optimization between nested loops and OpenMP loop scheduling lie in different parallelization problems and different low-power methods. Kadayif et al. were interested in the parallel nested loops of data-intensive applications in on-chip multiprocessors. For these kinds of programs, data locality is sensitive to loop iteration assignment. Nested loops usually include loop-carried

© 2012 ACADEMY PUBLISHER

dependencies. During the process of assigning loop iterations to multiprocessors, the effect of loop-carried dependencies must be considered. We allow the scaling down of each processor's voltage/frequency according to its individual busy/idle state, and we cannot easily estimate the delay because of dependencies and other factors. At this point, adopting an actual test to obtain the appropriate voltage/frequency for each processor is a better choice. For OpenMP loop scheduling, however, considering loop-carried dependence presents no problems. Programmers are in charge of eliminating loop-carried dependencies. When it comes to different low-power methods, Kadayif only use DVFS to save energy. However, our interest is focused not only on the DVFS technique, but also on scheduling the algorithm itself, especially the scheduling chunk size and scheduling policy. We study the effect of OpenMP scheduling chunk size on DVFS of processors. To the best of our knowledge, there are few related studies on energy-oriented OpenMP loop scheduling.

The processor shutdown technique and the DVFS approach can be used in combination for more efficient energy consumption. Li et al. [10] proposed a two-dimensional optimization method, in which one dimension involves changing the active number of processors, and the other entails scaling the voltage/frequency of processors. Their idea is similar to that in our previous work [11], but differs from the present work in a number of aspects. Their platform is CMP architecture, whereas ours is large-scale shared memory multi-core architecture. Although both our efforts focus on the energy optimization of parallel regions, an obvious difference is that the granularity adopted by Li et al. is a loop region, which ignores the discrepancy of processors. They set a uniform voltage level for each parallel region and chose an optimal number of processors. The granularity that we chose is loop iteration, a finer-grained type.

Teodorescu et al. [12] proposed variation-aware scheduling algorithms to save power or improve throughput. They also proposed variation-aware power management. They proposed an algorithm called LinOpt, which uses linear programming to find the best voltage and frequency level for each core in the CMP.

Bhattacharjee et al. [13] proposed and evaluated thread criticality predictors for parallel applications, which can guide Intel Threading Building Blocks (TBB) task stealing decisions to improve performance. It can also guide dynamic energy optimizations in barrier-based applications.

Wang et al. [14] proposed a chip-level power control algorithm that is systematically designed based on optimal control theory. Rangan et al. [15], on the other hand, proposed thread motion (TM), a fine-grained power-management scheme for CMPs. TM enables rapid movement of threads to facilitate adaptation of the time-varying computing needs of running applications to a mixture of cores with fixed but different power or performance levels.

MPI is a widely used parallel programming interface. Representative studies on low-power MPI optimization are as follows. Kappiah et al. [16] presented a system called Jitter, which exploits node slack time to scale down node frequency for energy conservation. Freeh et al. [17] focused on low-power and high-performance clusters, divided programs into phases, and chose a suitable frequency for each phase. Lim et al. [18] identified communication phases and reduced the voltage/frequency of processors in communication phases. MPI-based DVFS algorithms cannot be directly applied to OpenMP programs because they are distinctly different in implementation schemes. OpenMP loop scheduling utilizes compiler-directed methods to achieve energy optimization. For MPI programs, however, energy optimization is implemented by MPI libraries. The processes of MPI send and receive messages to and from others, and the computation of each process are decided by programmers. There is no scheduling problem among them. The energy optimization of the MPI program is primarily focused on energy reduction of processors when processes are changing messages.

In our previous work [19], we presented some studies on energy-saving optimal static scheduling (**EOSS**) algorithm. However, **EOSS** is limited in that it assumes that ideal conditions are satisfied. The same scheduling chunk, instead of ununiform-sized scheduling chunk, is assumed. In other words, the more general cases are ignored. In this paper, we provide a more detailed illustration of **EOSS,** including the determination of the chunk size of two-phase scheduling. More important, we extend the assumption and discuss the energy consumption optimization of OpenMP loop scheduling with ununiform-sized scheduling chunks. We propose **e**nergy-oriented **s**tatic **s**cheduling algorithm (**ESS**), which enriches earlier studies.

III ENERGY OPTIMIZATION ALGORITHMS FOR OPENMP LOOP SCHEDULING

A. Introduction

```
DOALL   i=1, 37
    { block }
END DOALL
```

Figure 1. A simple DOALL loop

In this paper, we assume that the number of outermost iterations in one OpenMP loop is N, and the number of processors for executing OpenMP loop is p. We take the DOALL loop shown in Figure. 1 as an example (N =37, p =5).

Taking each outermost iteration as a scheduling unit, OpenMP static scheduling distributes all loop iterations to available processors. The given chunk of iterations is defined as *scheduling chunk size* (*S)*, which refers to the number of outermost loop iterations in one schedule. *S* is statically determined by an OpenMP compiler. Static scheduling has very little scheduling overhead, but load imbalance occurs among multiprocessors. Based on varying chunk sizes, static scheduling is divided into three types: block assignment, block cyclic assignment, and cyclic assignment. As shown in Figure. 2(a), block assignment divides the workload into blocks of iterations

© 2012 ACADEMY PUBLISHER

that are as equal as possible. Chunk size is obtained through the number of iterations divided by the number of processors. However, most of the time, the number of iterations is not divided exactly by the number of processors. Each processor is assigned with a group of consecutive loop iterations and scheduled only once. In the example shown in Figure. 2(a), the scheduling chunk size is eight. Block cyclic assignment allows loop assignments to be interleaved using some stride, assigning iterations by round-robin to each processor. Each processor can be scheduled multiple times. Figure. 2(b) shows a scheduling chunk size of three. Figure. 2(c) describes cyclic assignment, which is a special case with a chunk size of one. Each processor is scheduled numerous times. On the same processor, data in two consecutive schedules are discontinuous, leading to poor data locality.

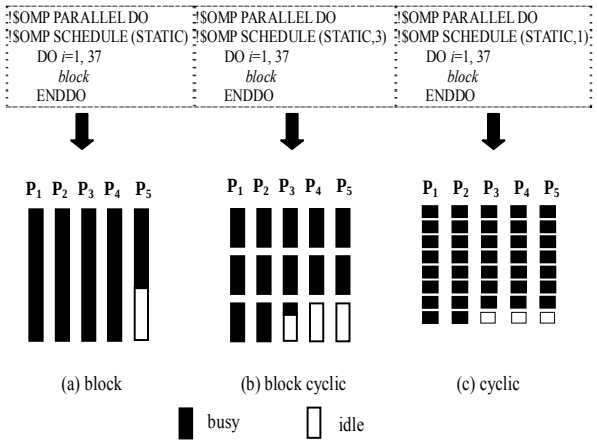


Figure 2. Results of OpenMP static loop scheduling

We disregard synchronization overhead and data locality discrepancy because of the different chunk sizes, and measure load balancing only in terms of the number of iterations assigned by each processor. Thereafter, we conclude that block assignment most easily causes load imbalance. If $p \nmid N$, the last processor is assigned with the remaining iterations.

OpenMP loop scheduling can be divided into two types in terms of the characteristics of the scheduling unit (outermost iteration).

1) *Uniform-sized scheduling unit (US)*: The execution time of each outermost loop iteration is always the same. **EOSS** is based on this assumption.

2) *Ununiform-sized scheduling unit (UUS)*: The execution time of each the outermost loop iteration is not always the same. **ESS** is based on this assumption.

Before discussing the algorithms in detail, we provide basic definitions of processor power and energy consumption.

According to CMOS power equations,

$$Power = Dynamic\ Power + Short\ Circuit\ Power + Leakage\ Power \quad (1)$$

Among the sources of power consumption, we focus on dynamic power because it accounts for a big fraction of total power. Dynamic power P is calculated according to the following equation [6]:

$$P = \alpha C V^2 f \quad (2)$$

where α refers to the switching activity factor, $0<\alpha<1$, C is the total capacitance driven by all the gate output, V is the operating voltage, and f denotes the clock frequency. P is proportional to the square of V and f. Furthermore, voltage and frequency must be adjusted together while satisfying the following relationship:

$$f \propto \frac{(V - V_{th})^{\tau}}{V} \quad (3)$$

where V_{th} is the threshold voltage and is a constant, τ is a proportional factor, and $1\le\tau\le2$. When $\tau=2$, f is proportional to V, so that (2) can be replaced by

$$P \propto f^3 \quad (4)$$

Hence, we can conclude that scaling the voltage/frequency of the processor can result in cubic power savings. Delay is inversely proportional to frequency; thus, energy is proportional to the square of the voltage.

$$E \propto f^2 \quad (5)$$

Equation (3) shows that voltage and frequency have to be scaled simultaneously, and satisfy a certain relationship. We can only use the term frequency scaling instead of voltage/frequency scaling. Hence, the value of the voltage can be calculated in terms of the value of the frequency.

B. EOSS

Table 1 shows some symbol descriptions used in this article. In OpenMP scheduling, the workload and frequency of each processor form a pair (a_i,f_i). Scheduling Γ is composed by p pairs of (a_i,f_i), where p represents the number of processors. Γ-based λ scheduling is a new scheduling derived from Γ in terms of certain rules. As much as possible, Γ-based λ scheduling is obtained by scaling down the frequency of each processor under certain performance conditions. The detailed rules that Γ-based λ scheduling follows are: (1) the frequency of each processor in Γ-based λ scheduling must be lower than that in Γ; and (2) the parallel time of Γ-based λ scheduling is given by users, equal to T, which is no less than the time of Γ. Γ-based λ scheduling is used in the following proof of optimality.

First, as Figure. 3 shows, we can directly apply DVFS to multiprocessors where light-load processors can run at lower speeds to save energy; meanwhile, the deadline of the loop is satisfied. Figure. 3(a) is identical to Figure. 2(b), in which the parallel time is determined by processors P_1 and P_2. In Figure. 3(b), we prolong the execution time of P_3–P_5 to reach the parallel time by scaling down the voltage/frequency of P_3–P_5. As a result, DVFS saves energy without performance loss.

© 2012 ACADEMY PUBLISHER

TABLE I.
SYMBOL DESCRIPTIONS

Symbol	Meaning
$\Gamma = \{(a_1, f_1), (a_2, f_2), ..., (a_p, f_p)\}$	Static scheduling sequence Γ. Each pair of (a_i, f_i) represents the number of iterations a_i and the frequency f_i for processor P_i
$T(\Gamma)$	Parallel execution time of Γ
$E(\Gamma)$	The energy consumption of Γ
$T_i^w(a, f)$	Busy time of processor P_i under (a, f)
$\lambda(\Gamma, T) = \{(a_1, \lambda(f_1)), ..., (a_p, \lambda(f_p))\}$	Γ-based λ scheduling. It is based on $\Gamma = \{(a_1, f_1), ..., (a_p, f_p)\}$. Frequency $\lambda(f_i)$ for P_i meets: $\forall i \in \{1, ..., p\}, \lambda(f_i) \le f_i$ and $T_i^w(a_i, \lambda(f_i)) = \begin{cases} T, & a_i > 0 \\ 0, & a_i = 0 \end{cases}$ where $T \ge T(\Gamma)$.

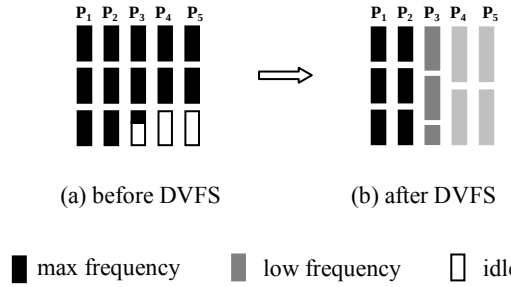


Figure 3. DVFS of OpenMP static loop scheduling

The DVFS policy in Figure. 3 is straightforward. It does not change scheduling policy; rather, it only scales down the frequency of each processor according to the load. If we modify scheduling policy by adjusting chunk size, is there any possibility of more energy savings?

The basic idea of the improved algorithm is described as follows. Before DVFS is carried out, we change the loop scheduling policy. The key to the algorithm is rescheduling the loops with the principle that the workload of all the processors should, as much as possible, stay balanced. Loop rescheduling combined with DVFS results in more energy savings than DVFS alone (Figure. 4). Therefore, we propose *US-based* **EOSS** algorithm.

Figure. 4 describes the two steps of **EOSS**. The first step is loop rescheduling intended to obtain more load balance, and the second step is DVFS. Figure. 4(a) is the initial OpenMP static scheduling paradigm, and it can be any of the static schedulings shown in Figure. 2. We assume that the chunk size of initial scheduling Γ_0 is S, and the parallel time of the initial scheduling is $T(\Gamma_0)$. These values are known before **EOSS** is applied. The optimization of **EOSS** is based on fixed scheduling chunks, and choosing different chunk sizes leads to different **EOSS** energy optimization results. $T(\Gamma_0)$ determines the parallel time of **EOSS**. Under the performance loss of β, the parallel execution time after **EOSS** optimization is $T(\Gamma_0) \times (1+\beta)$. In the initial scheduling, processors P_3–P_5 are idle in the last round of scheduling, while P_1 and P_2 are constantly busy. According to (5), scaling down the frequency of processors P_3–P_5 brings forth energy savings of square stage.

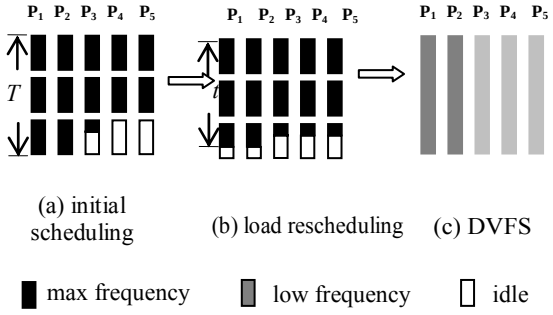


Figure 4. Two-phase illustration of EOSS

To obtain more opportunities to save energy, loads on multiprocessors must be as balanced as possible. Even if the loads of each processor cannot be exactly the same, the discrepancy between multiprocessors must be minimized. Block cyclic assignment with a chunk size of one has the best load balance, but it leads to poor data locality because of discontinuous iterations in two adjacent scheduling rounds on the same processor. To compensate for this defect and minimize the load imbalance, we propose two-phase scheduling, which keeps the most rounds of iteration assignment unchanged, and only changes the chunk size in the last round of the iteration assignment. Consequently, load differences between any two processors are no more than one. Let a_i^* represent the number of iterations assigned to processor P_i, $i, j = 1, 2, ..., p$. That is,

$$\forall i, j, \quad |a_i^* - a_j^*| \le 1 \tag{6}$$

In the first phase of **EOSS**, the chunk size is unchanged and equal to S. Each processor is scheduled $\left\lfloor \frac{N}{p \times S} \right\rfloor$ times. In the second phase, each processor is scheduled once. To satisfy (6), the chunk size is changed to S_i from S; S_i refers to the chunk size of processor P_i during the second phase, and it is calculated by

© 2012 ACADEMY PUBLISHER

$$S_i = \begin{cases} \left\lfloor \frac{N}{p} \right\rfloor + 1 & i = 1,2,...,(N \bmod p) \\ \left\lfloor \frac{N}{p} \right\rfloor & i = (N \bmod p)+1,...,p \end{cases} \quad (7)$$

where N is the total number of all iterations scheduled in the second phase, which satisfies (8).

$$N = N - \left\lfloor \frac{N}{p \times S} \right\rfloor \times p \times S \quad (8)$$

If $(p \times S) | N$ and $N = 0$, each processor has equal workload, and (6) is satisfied. At this time, two-phase scheduling is degraded to the DVFS method (Figure. 3).

In the second phase, taking $T(\Gamma_0)\times(1+\beta)$ as the deadline, we scale down each processor's voltage/frequency, which extends the busy time of each processor from $t_0 \cdot a_i^*$ to $T(\Gamma_0)\times(1+\beta)$, where t_0 refers to the execution time of one iteration at the maximum frequency. The frequency of processor P_i is changed from $f_{\max}$ to f_i^*, expressed as

$$f_i^* = \frac{t_0 \times a_i^*}{T(\Gamma_0)\times(1+\beta)} \quad (9)$$

where a_i^* can be calculated by

$$a_i^* = \left\lfloor \frac{N}{p \times S} \right\rfloor \times S + S_i \quad (10)$$

We use the example shown in Figure. 4(a) to explain that **EOSS** in Figures. 4(b)–(c) can generate more energy savings than DVFS in Figure. 3(b). We assume that the frequency of processor P_i in Figure. 3(b) is scaled to f_i, and the frequency of processor P_i in Figure. 4(c) is scaled to f_i^*; then we have $f_1 = f_2 = f_{\max}$, $f_3 = \frac{7}{9} f_{\max}$, $f_4 = f_5 = \frac{2}{3} f_{\max}$, $f_1^* = f_2^* = \frac{8}{9} f_{\max}$, and $f_3^* = f_4^* = f_5^* = \frac{7}{9} f_{\max}$. Thus, we have $\sum_{i=1}^{p} (f_i^*)^2 < \sum_{i=1}^{p} f_i^2$. Figure. 5 shows the detailed description of **EOSS**.

In this section, we prove that **EOSS** can obtain maximum energy savings.

The energy-saving optimal problem is described as follows: Given p processors, a DOALL loop L has N outermost loop iterations, initial OpenMP static loop scheduling Γ_0 has a chunk size of S, and β is performance loss. Under the condition of *US scheduling unit*, we find energy-saving optimal scheduling Γ^* to reach the maximum energy savings while satisfying $T(\Gamma^*)=T(\Gamma_0)\times(1+\beta)$.

The following lemma and theorems prove that **EOSS** can yield maximum energy savings. The proof is detailed in the Appendix. Lemma 1 and Theorem 1 are the bases of Theorem 2. Lemma 1 proves that the energy of Γ-based λ scheduling is no more than that of Γ. In other words, Γ-based λ scheduling is better than Γ from the angle of energy. This conclusion is easy to understand because Γ-based λ scheduling does not change the load and merely scales down the frequency of each processor, so that the energy of Γ-based λ scheduling is no higher than that of Γ scheduling.

Algorithm: ***E**nergy-saving **O**ptimal **S**tatic **S**cheduling algorithm* (**EOSS**)
Input: p processors, one DOALL nested loop L, N outermost loop iterations, initial OpenMP static loop scheduling with chunk size of S, performance loss with β.
Output: energy-optimal scheduling Γ^*, which satisfying the maximum energy saving under performance loss with β.

1. Set chunk size of each processor in the first phase to be S;
2. Set chunk size of processor P_i in the second phase to be S_i, $S_i = \begin{cases} \left\lfloor \frac{N}{p} \right\rfloor + 1 & i = 1,2,...,(N \bmod p) \\ \left\lfloor \frac{N}{p} \right\rfloor & i = (N \bmod p)+1,...,p \end{cases}$ where $N = N - \left\lfloor \frac{N}{p \times S} \right\rfloor \times p \times S$.
3. Calculate workload of each processor. $a_i^* = \left\lfloor \frac{N}{p \times S} \right\rfloor \times S + S_i$.
4. Set processor P_i new frequency $f_i^* = \frac{t_0 \times a_i^*}{T(\Gamma_0)\times(1+\beta)}$, $i = 1,...,p$;
5. New frequency sequence for this nested loop is $(f_1^*, f_2^*, ..., f_p^*)$;
6. Output $\Gamma^* = \{(a_1^*, f_1^*), (a_2^*, f_2^*), ... (a_p^*, f_p^*)\}$;
7. End.

Figure 5. Description of EOSS

Lemma 1. *Given* $\Gamma = \{(a_1, f_1), (a_2, f_2), ..., (a_p, f_p)\}$, *then* $E(\lambda(\Gamma, T)) \le E(\Gamma)$, *where* $T \ge T(\Gamma)$.

We assume that the sum of all the loads of the processors (the outermost loop iterations) is N. Each assignment $(x_1, x_2, ..., x_p)$ satisfies $\sum_{i=1}^{p} x_i = N$. Set W comprises all the assignments. Theorem 1 proves that one of the assignments ($w^* \in W$) exists, which drives $\sum_{i=1}^{p} x_i^3$ to reach the minimum value. In w^*, the load difference between any two processors is no more than one.

Theorem 1. *Given a function* $f(x_1, x_2, ..., x_p) = \sum_{i=1}^{p} x_i^3$, *in which* $\forall x_i \in \mathbb{Z}^+$, $\sum_{i=1}^{p} x_i = N$. *Here, N is a constant. Given assignment set* $W = \{(x_1, x_2, ..., x_p)\}$. *Let* $\bar{x} = \lfloor N / p \rfloor$. *Assume that* $w^* = (x_1^*, x_2^*, ..., x_p^*) \in W$. *If* $p \mid N$, *then* $x_i^* = \bar{x}, 1 \le i \le p$; *otherwise,*

© 2012 ACADEMY PUBLISHER

$\forall i, x_i^* = \bar{x}$ *or* $x_i^* = \bar{x} + 1$. *Hence,*

$\min f(x_1, x_2, \ldots, x_p) = f(w^*)$.

Based on Lemma 1 and Theorem 1, Theorem 2 proves that **EOSS** generates maximum energy savings. The energy consumption of any scheduling is not less than that of **EOSS**. Optimal scheduling Γ^* in **EOSS** is obtained in two steps. Step one is rescheduling the initial scheduling Γ to Γ' as shown in Figure. 4(b). Step two is DVFS, which is completed by generating Γ'-based λ scheduling. We need to prove that for any scheduling Γ^{Δ}, we can obtain $E(\Gamma^*) \le E(\Gamma^{\Delta})$.

Theorem 2. *Given* $\Gamma = \{(a_1, f_{\max}), (a_2, f_{\max}), \ldots, (a_p, f_{\max})\}$ *in which* $\sum_{i=1}^{p} a_i = N$, *assume* $N \ge p$, *denote* $b_0 = \left\lfloor \frac{N}{p} \right\rfloor$, *and let* $\Gamma' = \{(b_1, f_{\max}), \ldots, (b_2, f_{\max}), (b_p, f_{\max})\}$ *where* $\sum_{i=1}^{p} b_i = \sum_{i=1}^{p} a_i$. *If* $p \mid N$, *then* $b_i = b_0$, $1 \le i \le p$. *Otherwise,* $b_i = \begin{cases} b_0 + 1, & 1 \le i \le \Delta b \\ b_0, & \Delta b + 1 \le i \le p \end{cases}$ *where* $\Delta b = N \bmod p$. *Let* $\Gamma^* = \lambda(\Gamma', (1+\beta) \cdot T(\Gamma))$, *in which* β *is the performance loss. The randomly given* $\Gamma^{\Delta} = \{(c_1, f_{\max}), (c_2, f_{\max}), \ldots, (c_p, f_{\max})\}$ *satisfies* $T(\Gamma^{\Delta}) = (1+\beta) \cdot T(\Gamma)$ *and* $\sum_{i=1}^{p} c_i = \sum_{i=1}^{p} a_i$. *Let* $\Gamma^{\gamma} = \lambda(\Gamma^{\Delta}, T(\Gamma^{\Delta}))$, *and we have* $E(\Gamma^{\gamma}) \le E(\Gamma^{\Delta})$ *in terms of Lemma 1. We prove* $E(\Gamma^*) \le (1+\beta) \cdot E(\Gamma)$ *and* $E(\Gamma^*) \le E(\Gamma^{\gamma})$, *that is,* $E(\Gamma^*) \le E(\Gamma^{\Delta})$.

C. *ESS*

EOSS shows the process of obtaining the maximum energy savings in the case of *US*. However, when the execution time of the outermost loop iterations is not the same, does the maximum energy-saving solution exist? If not, how can we optimize the energy consumption? To address this problem, we propose a *UUS-based* **ESS** algorithm.

Figure. 6 shows four kinds of parallel loop scheduling. In Figure. 6(b), the outermost loop iterations have different execution time because of the branches in the inner loops. In Figures. 6(c) and (d), the bound of inner index *j* is decided by outer index *i*, which also leads to the outermost loop iterations having different execution times. Evenly assigning the outermost loops still causes heavy load imbalance. The shapes of four subFigureures in Figure. 6 are (a) rectangular, (b) trapezoidal, (c) right up-triangular, and (d) left up-triangular.

The examples in Figure. 6 can be reduced to four categories as shown in Figure. 7[20]. Horizontal axis *i* of each subfigure represents the index of the outermost loop. Vertical coordinates *L(i)* represent the workload of each outermost loop. Figure. 7(a) shows the case of *US*. Figures. 7(b)–(d) show the case of *UUS*, which can be divided into two types: irregular varying case as Figure. 7(b) shows and regular varying case as Figures. 7(c)–(d) show. The regular varying case includes monotonic increasing and decreasing scheduling units. In this article, we consider optimizing the energy only with a monotonic increasing scheduling unit. Meanwhile, the same method can be used for monotonically decreasing. For the irregular varying case, presenting an effective energy optimization method is difficult and goes beyond the scope of this paper. Because of varying scheduling units, providing an energy-saving optimal algorithm is impossible. We combine loop rescheduling and DVFS to discuss **ESS** algorithm for the *UUS* case.

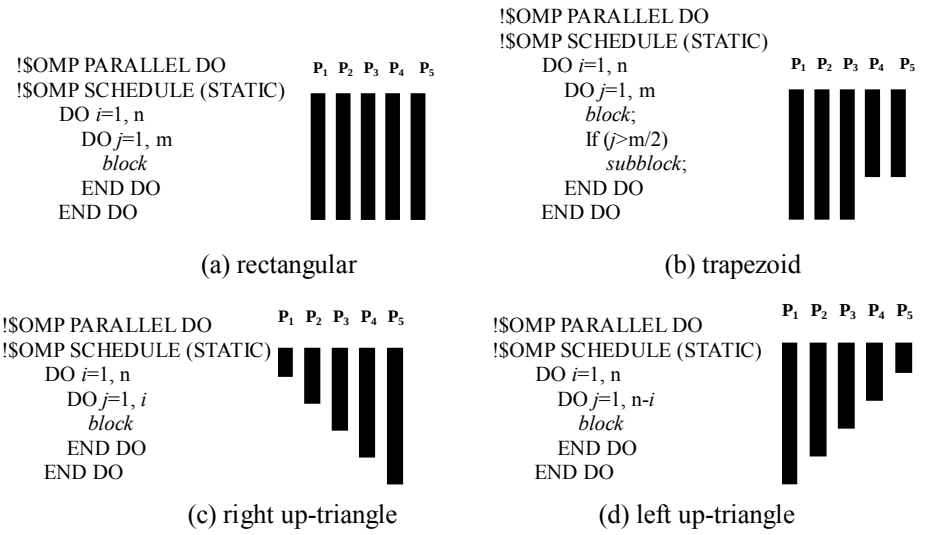


Figure 6. Four kinds of examples

© 2012 ACADEMY PUBLISHER

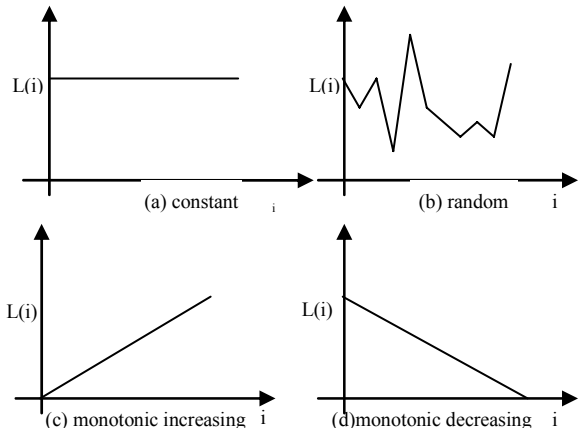


Figure 7. Four kinds of chunk shape

We assume that the chunk size of initial scheduling Γ_0 is S, and the permitted performance loss is β. Taking Figure. 8(a) as an example, we illustrate the idea of **ESS**. Here, we assume that the number of outermost iterations is 10, the number of processors is 5, and the block size is 2. Only DVFS brings about the result shown in Figure. 8(b). **ESS** improves the workload assignment. The process of **ESS** consists of three steps as follows:

STEP 1: **Adopting a round diverse cyclic scheduling method**. Based on cyclic assignment, we propose a *round diverse cyclic scheduling method* that changes Γ_0 to Γ^{Δ}. It keeps the processor frequency unchanged (f_{max}) and only scales the number of iterations assigned to each processor. The detailed method involves scaling chunk size from S (equal to two) to one. According to increasing or decreasing sequence, we alternately change the order of processor assignment for each round scheduling. For all processors, completing one task assignment is called *one round scheduling*. In Figure. 8(a), only one round scheduling is applied in terms of a chunk size of two. After applying the round diverse cyclic scheduling method, the chunk size is changed to one, and two round schedulings are applied. The sequences followed by the processors in the first and second rounds of scheduling are $\{P_1, P_2, P_3, P_4, P_5\}$ and $\{P_5, P_4, P_3, P_2, P_1\}$, respectively. As Figure. 9 (b) shows, the iteration numbers assigned to processors P_1–P_5 are(i=1, i=10),(i=2, i=9), (i=3, i=8), (i=4, i=7) and (i=5, i=6), respectively.

STEP 2: **Determining scheduling policy**. If $\frac{T(\Gamma^{\Delta})-T(\Gamma_0)}{T(\Gamma_0)} > \beta$ then $\Gamma^{*}=\Gamma_0$; otherwise, $\Gamma^{*}=\Gamma^{\Delta}$.

STEP 3: **Applying DVFS to** Γ^{*}. We scale down each processor's voltage/frequency, which extends the busy time of each processor to $T(\Gamma_0)\times(1+\beta)$ as Figure. 9(c) shows.

Figure. 10 presents the detailed description of **ESS.**

IV. EXPERIMENTAL EVALUATIONS

In this section, we evaluate our energy-oriented OpenMP loop scheduling algorithms. Section 4.1 introduces the experimental environment, including the simulation platform, methodology, and benchmark. Section 4.2 presents the experimental results and detailed analyses of each algorithm.

A. Experimental Environment

1) Simulation Platform and Methodology

To conduct our experiments, we extended Simplescalar [21] to simulate shared-memory multiprocessor architecture. Each processor has the same configuration, including cache and memory. Memory access latency includes local memory access latency and remote memory access latency. Power simulation adopts Wattch [22], which models the dynamic power dissipation of each processor. We added the module implementation of DVFS based on the Wattch power model. The range of clock frequency scaling is 300 MHz–1 GHz. This simulation platform provides a frequency scaling interface for DVFS implementation, in which the relationship of frequency and voltage satisfies (3). When applying processor shutdown technology, the power of processors in the shutdown state consumes 10% of the power in the active state. To simplify the computation of time and energy overhead, we adopted fixed voltage/frequency switching overhead instead of (11) and (12). We also adopted fixed reactivate overhead. Table 2 lists the relevant parameters.

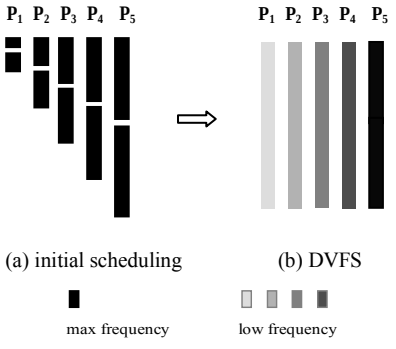


Figure 8. Results of DVFS

© 2012 ACADEMY PUBLISHER

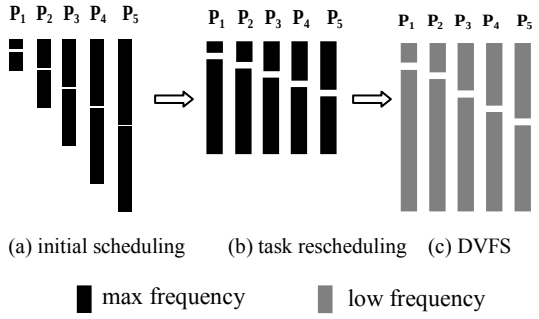


Figure 9. Illustration of ESS

Algorithm: UUS-based ***E**nergy-oriented **S**tatic **S**cheduling algorithm* (**ESS**)
Input: p processors, one DOALL nested loop L, N outermost loop iterations, initial OpenMP static loop scheduling with chunk size of S, performance loss with β.
Output: energy-oriented scheduling Γ^*, which satisfying performance loss with β.

1. Obtaining new scheduling Γ^{∇} through round diverse cyclic scheduling method;
2. if $\frac{T(\Gamma^{\nabla})-T(\Gamma_0)}{T(\Gamma_0)} > \beta$
3. then $\Gamma^* = \Gamma^0$ where $\Gamma^* \leftarrow \left\{\left(a_1^*, f_{\max}\right),\left(a_2^*, f_{\max}\right),\ldots\left(a_p^*, f_{\max}\right)\right\}$;
4. else $\Gamma^* = \Gamma^{\nabla}$ where $\Gamma^* \leftarrow \left\{\left(a_1^*, f_{\max}\right),\left(a_2^*, f_{\max}\right),\ldots\left(a_p^*, f_{\max}\right)\right\}$;
5. Set processor P_i new frequency $f_i^* = \frac{t_0 \times a_i^*}{T(\Gamma_0)\times(1+\beta)}$, $i = 1,\ldots, p$;
6. New frequency sequence for this nested loop is $\left(f_1^*, f_2^*, \ldots, f_p^*\right)$;
7. Output $\Gamma^* = \left\{\left(a_1^*, f_1^*\right),\left(a_2^*, f_2^*\right),\ldots\left(a_p^*, f_p^*\right)\right\}$;
8. End.

Figure 10. Description of ESS

The loop scheduling results are based on the CCRG OpenMP parallel compiler[23], which obeys OpenMP API 2.0 standard and supports Fortran 77/90/95 C language. As Figure. 11 shows, the CCRG OpenMP parallel compiler consists of the OpenMP translator, OpenMP run-time library, native compiler, and linker. The OpenMP translator does the following work: (1) identifying and processing OpenMP directives; and (2) optimizing the OpenMP program, covering data locality optimization, voltage scaling instruction insertion, and so on.

Our experiment was accomplished through the following steps: 1) the original program was simulated on Simplescalar + Wattch environment, and performance and energy results were obtained; 2) the program traces generated by CCRG OpenMP were analyzed; 3) frequency scaling clause was incorporated by hand; 4) the program traces with "marks" were generated as input of the simulation environment; and 5) the new binary code was simulated again, new performance and energy results were obtained and compared with those in 1).

2) Benchmark

To evaluate the effectiveness of our energy-oriented loop scheduling algorithms, we used five NPB3.2-OMP [24] kernel programs: IS, EP, FT, CG, and MG. NPB is a representative parallel benchmark, and NPB3.2-OMP is the OpenMP implementation of NPB3.2. There are six problem sizes for each program (i.e., S, W, A, B, C, and D) from which we choose class C. In addition, we use a code segment to verify the effectiveness of **ESS,** which is shown in Figure. 12. In this code, A and B are *double* type constants. The number of outermost loop is defined as M, which is 10240.

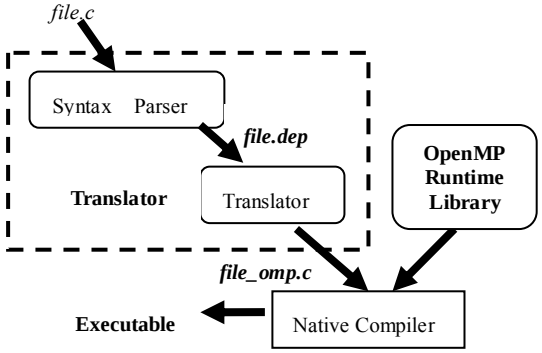


Figure 11. CCRG OpenMP parallel compiler

TABLE II. DEFAULT SIMULATION PARAMETERS IN THE EXPERIMENTS

Parameter	Default Value
Highest Frequency/Voltage	1 GHz / 1.9 V
Lowest Frequency	300 MHz
Voltage/Frequency Switching Penalty	2 us /2 uJ
Reactivate Switching Penalty	50 us / 50 uJ
L1 latency	1 cycle
L2 latency	4 cycles
Local memory access latency	100 cycles
Remote memory access latency	500 cycles

```
#define M 10240
double a[M][M];
double b[M][M];
double c[M][M];
for(i=0;i<M;i++)
        for(j=0;j<i;j++){
                c[i][j] = a[i][j]/A + b[i][j]/B;
                d[i][j] = a[i][j]/B + b[i][j]/A;
                e[i][j] = c[i][j]/B + d[i][j]/A;
        }
```

Figure 12. experiment code for ESS

© 2012 ACADEMY PUBLISHER

TABLE III. PROFILE DATA FOR EACH PROGRAM

Program	The number of parallel regions	The max number of iterations	The min number of iterations
IS	24	268435456	134217728
EP	1	65536	65536
FT	38	1024	512
CG	2918	149907	149800
MG	770	139264	512

Table 3 provides all the profile data of the parallel loop regions of each program, including the number of parallel loop regions, as well as the maximum and minimum number of outermost iterations for all the parallel regions.

B. Results

The energy results in this section pertain only to the energy consumption of the parallel loop regions.

1) Results of *EOSS*

This section presents the experimental results of **EOSS**, which are compared with the DVFS method.

First, we set the chunk size of each benchmark. Chunk size refers to the number of outermost loop iterations in one scheduling. IS, EP, FT, CG, and MG have chunk sizes of 300000, 100, 3, 100, and 200, respectively. FT chooses the least chunk size when considering both the number of outermost loop iterations and number of processors.Figure. 13 shows the experimental results of **EOSS**. In this figure, the number of processors ranges from 20–160, and performance loss β=5%. For these five programs, when the numbers of processors are 20, 40, 60, 80, 100, 120, 140, and 160, the average energy savings are 5.9%, 17.1%, 8.5%, 28.3%, 21.5%, 28.9%, 35.7%, and 43.5%, respectively. From the perspective of energy savings, a large-scale system can save more energy, and energy optimization is more significant for larger computing systems. We can also say that the larger the system, the more energy is wasted.

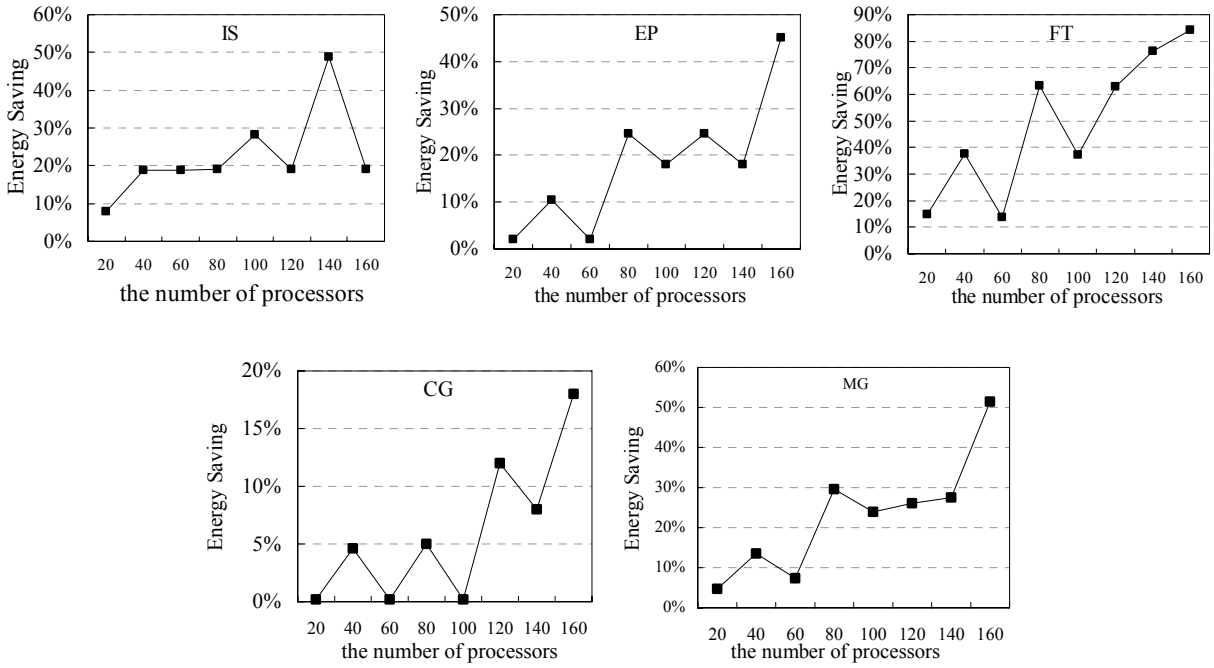


Figure 13. Energy saving of EOSS

The **EOSS** uses DVFS as the major energy saving method, but DVFS is not the major characteristic of this algorithm. The most important characteristic of **EOSS** is loop rescheduling that helps DVFS. Combining loop rescheduling with DVFS can obtain more energy savings than using DVFS method alone. This is the important contribution of this article. To evaluate the advantage of our algorithm over the DVFS method on energy savings, we compare the results of **EOSS** to those of the DVFS method. Here, the DVFS method refers to the method shown in Figures. 3 and 8. Each subfigure in Figure. 14 is obtained by subtracting the energy saving percentage of DVFS only from that of **EOSS**. From this figure, we can find that **EOSS** results in more energy savings than the DVFS method alone. For example, in FT under 140 processors, the energy saved in **EOSS** is 12% larger than that in DVFS.

2) Results of *ESS*

ESS determines different scheduling with a performance loss of β 5%. For the code in Figure. 12, we choose a chunk size of 20 for Γ_0. The energy savings in the new scheduling compared with Γ_0 is shown in Figure. 15.

From this figure, we see that when the chunk size is fixed, ESS achieves significant energy savings. This result also confirms that with the increase in the number of processors, the difference in each processor's workload of such a kind of loop becomes larger, which yields more opportunities for saving energy. For example, when the number of processors is 160, **ESS** has more

© 2012 ACADEMY PUBLISHER

than 50% energy savings compared with that of the original scheduling.

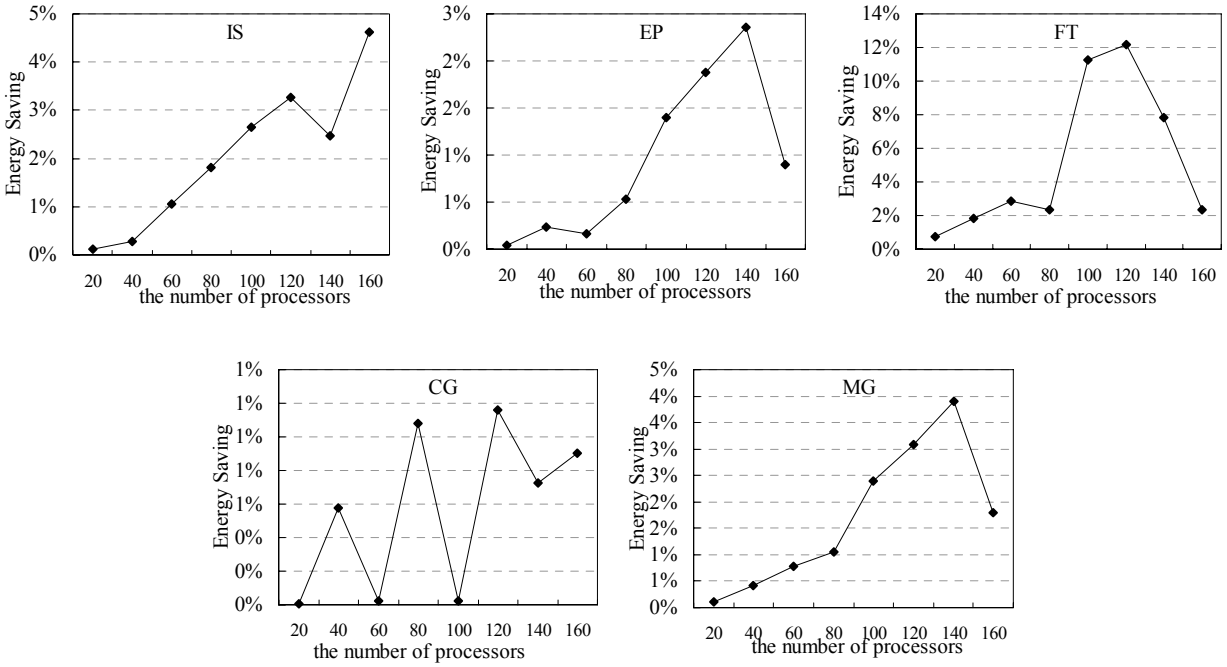


Figure 14. Advantage of EOSS compared with DVFS method

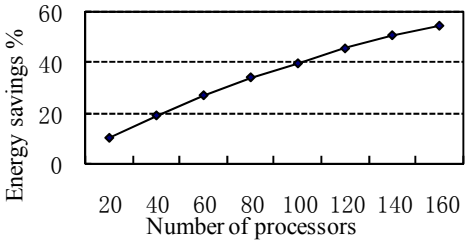


Figure 15. Energy savings of ESS

V. CONCLUSIONS

In this paper, we studied the energy optimization of OpenMP static and dynamic scheduling, and proposed two energy optimization algorithms for OpenMP static and dynamic scheduling: *US-based* **EOSS**, *UUS-based* **ESS**. **EOSS** obtains optimal energy saving effectiveness under given assumptions.

We built a simulation platform to evaluate the effectiveness of our energy-oriented scheduling algorithms. Experimental results show that our algorithms can achieve energy savings under different scheduling conditions. In the future, more practical conditions, such as communication overhead and data locality, will be considered to achieve further energy optimization of OpenMP loop scheduling.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of china under grant No. 60903044.

REFERENCES

[1] Top 500 list. 2010. http://www.top500.org/list/2010/06/10 0.

[2] Katherine Yelick. Ten ways to waste a parallel computer. In Proceedings of the the 36th annual international symposium on Computer architecture. Austin, TX, USA: ACM, 2009. 1-1.

[3] The OpenMP API specification for parallel programming. http://openmp.org.

[4] intel website. www.intel.com.

[5] AMD website. www.amd.com.

[6] T. Burd, R. Brodersen. Energy efficient CMOS microprocessor design. In Proceedings of the 28th Hawaii International Conference on System Science (HICSS-95)1995.

[7] I. Kadayif, M. Kandemir, M. Karakoy. An Energy Saving Strategy Based on Adaptive Loop Parallelization. In Proceedings of Design Automation Conference (DAC'02). New Orleans, Louisiana, USA: ACM, 2002. 195-200.

[8] I. Kadayif, M. Kandemir, U. Sezer. An Integer Linear Programming Based Approach for Parallelizing Applications in On-Chip Multiprocessors. In Proceedings of the 39th IEEE/ACM Design Automation Conference (DAC'02). New Orleans, LA, USA: ACM, 2002. 703-708.

[9] I. Kadayif, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, I. Kolcu. Exploiting Processor Workload Heterogeneity for Reducing Energy Consumption in Chip Multiprocessors. In Proceedings of Design, Automation and Test in Europe (DATE'04). Paris, France: IEEE Computer Society, 2004. 1158-1163.

[10] Jian Li, Jose F. Martinez. Dynamic Power-Performance Adaptation of Parallel Computation on Chip Multiprocessors. In Proceedings of the International

© 2012 ACADEMY PUBLISHER

Symposium on High Performance Computer Architecture(HPCA'06). Austin, Texas: IEEE Computer Society, 2006. 77-87.

[11] Juan Chen, Yong Dong, Xuejun Yang, Dan Wu. A Compiler-Directed Energy Saving Strategy for Parallelizing Applications in On-chip Multiprocessors. In Proceedings of the Fourth International Symposium on Parallel and Distributed Computing (ISPDC-05). Lille, France: IEEE Computer Society, 2005. p.147-154.

[12] Radu Teodorescu, Josep Torrellas. Variation-Aware Application Scheduling and Power Management for Chip Multiprocessors. In Proceedings of the 35th International Symposium on Computer Architecture (ISCA 2008). Beijing, China: IEEE, 2008. 363-374.

[13] Abhishek Bhattacharjee, Margaret Martonosi. Thread criticality predictors for dynamic performance, power,and resource management in chip multiprocessors. In Proceedings of the 36th International Symposium on Computer Architecture (ISCA 2009). Austin, TX, USA: ACM, 2009. 290-301.

[14] Yefu Wang, Kai Ma, Xiaorui Wang. Temperature-constrained power control for chip multiprocessors with online model estimation. In Proceedings of the 36th International Symposium on Computer Architecture (ISCA 2009). Austin, TX, USA: ACM, 2009. 314-324.

[15] Krishna K. Rangan, Gu-Yeon Wei, David Brooks. Thread motion: fine-grained power management for multi-core systems. In Proceedings of the 36th International Symposium on Computer Architecture (ISCA 2009). Austin, TX, USA: ACM, 2009. 302-313.

[16] Nandini Kappiah, Vincent W. Freeh, David K. Lowenthal. Just In Time Dynamic Voltage Scaling: Exploiting Inter-Node Slack to Save Energy in MPI Programs. In Proceedings of the ACM/IEEE SC2005 Conference on High Performance Networking and Computing, SC2005. Seattle, WA, USA: IEEE Computer Society, 2005.

[17] Vincent W. Freeh, David K. Lowenthal. Using multiple energy gears in MPI programs on a power-scalable cluster. In Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2005. Chicago, IL, USA: ACM, 2005. 164-173.

[18] Min Yeol Lim, Vincent W. Freeh, David K. Lowenthal. MPI and communication - Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs. In Proceedings of the ACM/IEEE SC2006 Conference on High Performance Networking and Computing, SC 2006. Tampa, FL, USA: ACM, 2006.

[19] Yong Dong, Juan Chen, Xuejun Yang, Lin Deng, Xuemeng Zhang. Energy-Oriented OpenMP Parallel Loop Scheduling. In Proceedings of the IEEE International Symposium on Parallel and Distributed Processing with Applications, ISPA 2008. Sydney, NSW, Australia: IEEE, 2008. 162-169.

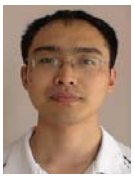
[20] Teebu Philip. Increasing Chunk Size Loop Scheduling Algorithems for Data Independent Loops. Master. The Pennsylvania State University. Department of Computer Science and Engineering. 1995.

[21] Doug Burger, Todd M. Austin. The SimpleScalar tool set, Version 2.0. Technical Report:CS-TR-1342. University of Wisconsin-Madison, July 1997.

[22] D. Brooks, V. Tiwari, M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In Proceedings of 27th International Symposium on Computer Architecture (ISCA)2000. 83-94.

[23] Chun Huang, Xuejun Yang. CCRG OpenMP: Experiments and Improvements. In Proceedings of the 1st International Workshop on OpenMP. Eugene, Oregon USA: Lecture Notes in Computer Science 2690, 2005. 514-521.

[24] NAS Parallel Benchmarks. http://www.nas.nasa.gov/Resources/Software.



Yong Dong Shandong Province, China. Birthdata: May 1980. He is now working in Computer School, Nation University of Defense Technology. His research interests including: parallel computing, massive storage system and energy optimization of HPC system.

Juan Chen Jiangxi Province, China. Birthdate: Feb, 1980. is Computer Science Ph.D, graduated from Computer School, Nation University of Defense Technology. Her research interests including: compiling technique and energy optimization of HPC system

© 2012 ACADEMY PUBLISHER