Invited Paper

DOI : 10.3745/JIPS.2009.5.4.175

Journal of Information Processing Systems, Vol.5, No.4, December 2009 175

On Effective Slack Reclamation in Task Scheduling for Energy Reduction

Young Choon Lee* and Albert Y. Zomaya*

Abstract: Power consumed by modern computer systems, particularly servers in data centers has almost reached an unacceptable level. However, their energy consumption is often not justifiable when their utilization is considered; that is, they tend to consume more energy than needed for their computing related jobs. Task scheduling in distributed computing systems (DCSs) can play a crucial role in increasing utilization; this will lead to the reduction in energy consumption. In this paper, we address the problem of scheduling precedence-constrained parallel applications in DCSs, and present two energy-conscious scheduling algorithms. Our scheduling algorithms adopt dynamic voltage and frequency scaling (DVFS) to minimize energy consumption. DVFS, as an efficient power management technology, has been increasingly integrated into many recent commodity processors. DVFS enables these processors to operate with different voltage supply levels at the expense of sacrificing clock frequencies. In the context of scheduling, this multiple voltage facility implies that there is a trade-off between the quality of schedules and energy consumption. Our algorithms effectively balance these two performance goals using a novel objective function and its variant, which take into account both goals; this claim is verified by the results obtained from our extensive comparative evaluation study.

Keywords: *Scheduling, Energy Awareness, Green Computing, Dynamic Voltage and Frequency Scaling, Data Centers*

1. Introduction

Until recently energy issues have been mostly dealt with by advancements in hardware technologies [1], such as low-power CPUs, solid state drives, and energy-efficient computer monitors. Energy-aware resource management has emerged as a promising approach for sustainable/green computing. Although many algorithms and strategies have been developed, their application is quite restricted for example, to systems such as battery-powered devices, homogeneous computing systems [2-5] or single-processor systems [6]. In addition to system homogeneity, tasks are often homogeneous or independent.

The energy consumption issue in distributed computing systems (DCSs) raises various monetary, environmental and system performance concerns. A recent study on power consumption by servers [7] shows that electricity use for servers worldwide—including their associated cooling and auxiliary equipment—in 2005 cost 7.2 billion US dollars. The study also indicates that electricity consumption in that year had doubled compared with consumption in 2000. Clearly, there are environmental issues with the generation of electricity. The number of transistors integrated into today's Intel Itanium 2 processor reaches nearly 1 billion. If this rate continues, the heat (per square centimeter) produced by future Intel processors would exceed that of the surface of the sun [8]; this implies the possibility of worsening system reliability, eventually resulting in poor system performance.

Due to the importance of energy consumption, various techniques including dynamic voltage and frequency scaling (DVFS), resource hibernation, and memory optimizations have been investigated and developed [1]. DVFS among these has been proven to be a very promising technique with its demonstrated capability for energy savings (e.g., [3,4,9]). For this reason, we adopt this technique and it is of particular interest to this study. DVFS enables processors to dynamically adjust voltage supply levels (VSLs) aiming to reduce power consumption; however, this reduction is achieved at the expense of clock frequencies.

Since precedence-constrained parallel applications in scientific and engineering fields are the most typical application model, the problem of scheduling these applications (task scheduling) both on homogeneous and heterogeneous computing systems has been studied extensively over the past few decades, e.g., [10-13]. However, most efforts in task scheduling have focused on two issues, the minimization of application completion time (makespan/schedule length) and time complexity; in other words, the main ob-

Manuscript received November 19, 2009; accepted December 3, 2009.
Corresponding Author: Albert Y. Zomaya
* Centre for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, Australia (yclee, zomaya@it.usyd.edu.au)

jective of a task scheduling algorithm is the generation of the optimal schedule for a given application with the minimal amount of scheduling time. It is only recently that much attention has been paid to energy consumption in scheduling, particularly on DCSs.

In this paper, we address the task scheduling problem on DCSs comprised of heterogeneous processors and present the *ECS* algorithm with its extension ($ECS^{makespan}$). A prior work of *ECS* can be found in [14]. While the extended version still takes into account both makespan and energy consumption, its focus is more on the reduction in makespan; hence, the name $ECS^{makespan}$. This design focus may lead to better energy efficiency as low-power processors are becoming more prevalent. The heuristics can easily be applied to loosely coupled DCSs (e.g., grids) using advance reservation and various sets of frequency-voltage pairs. *ECS* and $ECS^{makespan}$ are devised with the incorporation of DVFS to reduce energy consumption; this implies that there is a trade-off between the quality of schedules (makespans) and energy consumption. A novel objective function used in the main scheduling phase of each of our algorithms effectively deals with this trade-off balancing these two performance considerations. In addition, the energy reduction phase using the makespan-conservative energy reduction technique (MCER) is incorporated into *ECS* and $ECS^{makespan}$. In this phase, the current schedule generated in the scheduling phase is scrutinized to identify whether any changes to the schedule further reduce energy consumption without an increase in makespan. The low time complexity of our algorithms should also be noted. The results obtained from our extensive comparative evaluation study clearly show that *ECS* and $ECS^{makespan}$ outperform previous scheduling algorithms in terms of energy consumption by a noticeable margin. Their schedules are also shorter in makespan than those of other algorithms.

The remainder of the paper is organized as follows. Section 2 describes the application, system, energy and scheduling models used in this paper. Section 3 discusses the related work. *ECS* and $ECS^{makespan}$ algorithms are presented in Section 4 followed by a discussion of its performance in Section 5. The results of our comparative evaluation study are presented in Section 6. In Section 7, we make our conclusion.

2. Models

In this section, we describe the system, application, energy and scheduling models used in our study.

2.1 System model

The target system used in this work consists of a set P of p heterogeneous processors/machines that are fully interconnected. Each processor $p_j \in P$ is DVFS-enabled; in other words, it can operate in different VSLs (i.e., different clock frequencies). For each processor $p_j \in P$, a set V_j of v VSLs is random and uniformly distributed among four different sets of VSLs (Table 1); this ensures that our system model is realistic. Since clock frequency transition overheads take a negligible amount of time (e.g., 10μs-150μs [15], [16]), they are not considered in our study. The inter-processor communications are assumed to perform with the same speed on all links without contentions. It is also assumed that a message can be transmitted from one processor to another while a task is being executed on the recipient processor that is possible in many systems.

Table 1. Voltage-relative speed pairs

level	Pair 1		Pair 2		Pair 3		Pair 4	
	v_k	rel. speed (%)	v_k	rel. speed (%)	v_k	rel. speed (%)	v_k	rel. speed (%)
0	1.75	100	1.50	100	2.20	100	1.50	100
1	1.40	80	1.40	90	1.90	85	1.20	80
2	1.20	60	1.30	80	1.60	65	0.90	50
3	0.90	40	1.20	70	1.30	50		
4			1.10	60	1.00	35		
5			1.00	50				
6			0.90	40				

2.2 Application model

Parallel programs, in general, can be represented by a directed acyclic graph (DAG). A DAG, $G = (N, E)$, consists of a set N of n nodes and a set E of e edges. A DAG is also called a task graph or macro-dataflow graph. In general, the nodes represent tasks partitioned from an application; the edges represent precedence constraints. An edge $(i, j) \in E$ between task n_i and task n_j also represents inter-task communication. In other words, the output of task n_i has to be transmitted to task n_j in order for task n_j to start its execution. A task with no predecessors is called an *entry* task, n_{entry}, whereas an *exit* task, n_{exit}, is one that does not have any successors. Among the predecessors of a task n_i, the predecessor which completes the communication at the latest time is called the most influential parent (MIP) of the

task denoted as $MIP(n_i)$. The longest path of a task graph is the critical path (CP).

The weight on a task n_i denoted as w_i represents the computation cost of the task. In addition, the computation cost of the task on a processor p_j, is denoted as $w_{i,j}$ and its average computation cost is denoted as $\overline{w_i}$.

The weight on an edge, denoted as $c_{i,j}$ represents the communication cost between two tasks, n_i and n_j. However, a communication cost is only required when two tasks are assigned to different processors. In other words, the communication cost when tasks are assigned to the same processor can be ignored, i.e., 0.

The earliest start time and the earliest finish time of a task n_i on a processor p_j is defined as

$$EST(n_i, p_j) = \begin{cases} 0 & \text{if } n_i = n_{entry} \\ EFT(MIP(n_i), p_k) + c_{MIP(n_i),i} & \text{Otherwise} \end{cases} \quad (1)$$

$$EFT(n_i, p_j) = EST(n_i, p_j) + w_{i,j} \quad (2)$$

Note that the actual start and finish times of a task n_i on a processor p_j, denoted as $AST(n_i, p_j)$ and $AFT(n_i, p_j)$ can be different from its earliest start and finish times, $EST(n_i, p_j)$ and $EFT(n_i, p_j)$, if the actual finish time of another task scheduled on the same processor is later than $EST(n_i, p_j)$.

In the case of adopting task insertion the task can be scheduled in the idle time slot between two consecutive tasks already assigned to the processor as long as no violation of precedence constraints is made. This insertion scheme would contribute in particular to increasing processor utilization for a communication intensive task graph with fine-grain tasks.

A simple task graph is shown in Fig. 1 with its details in Tables 2 and 3. The values presented in Table 2 are computed using two frequently used task prioritization methods, *t-level* and *b-level*. The t-level of a task is defined as the summation of the computation and communication costs along the longest path of the node from the entry task in the task graph. The task itself is excluded from the computation. In contrast, the b-level of a task is computed by adding the computation and communication costs along the longest path of the task from the exit task in the task graph (including the task). The b-level is used in this study.

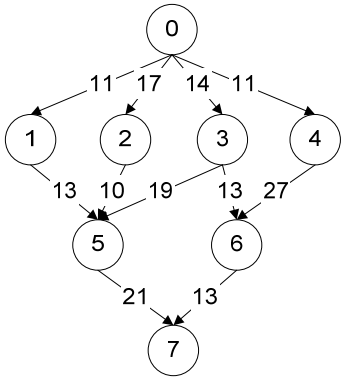

Fig. 1. A simple task graph

Table 2. Task Priorities

Task	b-level	t-level
0	101.33	0.00
1	66.67	22.00
2	63.33	28.00
3	73.00	25.00
4	79.33	22.00
5	41.67	56.33
6	37.33	64.00
7	12.00	89.33

Table 3. Computation cost with VSL 0

Task	p_0	p_1	p_2
0	11	13	9
1	10	15	11
2	9	12	14
3	11	16	10
4	15	11	19
5	12	9	5
6	10	14	13
7	11	15	10

The communication to computation ratio (CCR) is a measure that indicates whether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target system.

2.3 Energy model

Our energy model is derived from the power consumption model in complementary metal-oxide semiconductor (CMOS) logic circuits. The power consumption of a CMOS-based microprocessor is defined to be the summation of capacitive, short-circuit and leakage power. The capacitive power (dynamic power dissipation) is the most significant factor of the power consumption. The capacitive power (P_c) is defined as

$$P_c = ACV^2f \quad (3)$$

where A is the number of switches per clock cycle, C is the total capacitance load, V is the supply voltage, and f is the frequency. Equation 3 clearly indicates that the supply voltage is the dominant factor; therefore, its reduction

would be most influential to lower power consumption. The energy consumption of the execution of a precedence-constrained parallel application used in this study is defined as

$$E = \sum_{i=1}^{n} ACV_i^2 f \cdot w_i^* = \sum_{i=1}^{n} \alpha V_i^2 w_i^* \quad (4)$$

where V_i is the supply voltage of the processor on which task n_i executed, and w_i^* is the computation cost of task n_i (the amount of time taken for n_i's execution) on the scheduled processor.

2.4 Scheduling model

The task scheduling problem in this study is the process of allocating a set N of n tasks to a set P of p processors—without violating precedence constraints—aiming to minimize makespan with energy consumption as low as possible. The makespan is defined as $M=max\{AFT(n_{exit})\}$ after the scheduling of n tasks in a task graph G is completed. Although the minimization of makespan is crucial, tasks of a DAG in our study are not associated with deadlines as in real-time systems.

3. Related Work

In this section, we present two noteworthy works in task scheduling, particularly for DCSs, and then scheduling algorithms with power/energy consciousness. Since there are only few existing algorithms comparable to our study in terms of the explicit consideration of the energy issue in the task scheduling problem on DCSs, we discuss the closest possible algorithms—developed for homogeneous systems and/or with independent tasks.

3.1 Scheduling in HCSs

Due to the NP-hard nature of the task scheduling problem in general cases [17], heuristics are the most popularly adopted scheduling model, and they deliver good solutions in less than polynomial time. Heuristics are characterized by their essentially deterministic operation: the choice of solutions to a scheduling problem is not stochastic. Among the different heuristic techniques, list scheduling, clustering-based scheduling and guided random search are the three most prevalent approaches. List scheduling heuristics are the dominant heuristic model. This is because empirically, list scheduling algorithms tend to produce competitive solutions with lower time complexity compared to algorithms in the other categories [18].

The *HEFT* algorithm [12] is highly competitive in that it generates a schedule length comparable to other scheduling algorithms, with a low time complexity. It is a list-scheduling heuristic consisting of the two typical phases of list scheduling (i.e., task prioritization and processor selection) with task insertion.

Before scheduling begins, the b-level values of all tasks in a task graph are computed and arranged in a scheduling list in decreasing order of their b-level values. Each task is then scheduled, starting from the first task in the scheduling list. In the processor selection phase, the processor, p_j, on which the finish time of a task n_i, $EFT(n_i, p_j)$ is minimized, is selected using an insertion-based policy. In other words, a task can be inserted into the earliest time slot between two already-scheduled tasks on a processor if the precedence constraint of that task is not violated and the slot is large enough to accommodate the task. The time complexity of *HEFT* is on the order of $O(n \log n+(e+n)p)$.

The *DBUS* algorithm [19] is a duplication-based scheduling heuristic that first performs a CP-based listing for tasks and schedules them with both task duplication and insertion. The experimental results in [19] show its attractive performance, especially for communication-intensive task graphs.

As its name implies, *DBUS* schedules tasks in a task graph, traversing it in a bottom-up fashion. In the listing phase, it first computes the b-level, t-level and st-level values of the tasks and identifies the CP tasks. The CP tasks are stored in a list in decreasing t-level order along with the child tasks of each of these CP tasks, such that the child tasks of a CP task precede the CP task. These child tasks are stored in decreasing st-level order. The only distinction between the t- and st-levels is that communication costs are not considered in the st-level. The order of the tasks in the list determines the scheduling order.

In the scheduling phase, each task in the list is scheduled and duplicated as many times as either the number of its child tasks already scheduled or the number of processors—whichever is less. The processor to which a child task is assigned is regarded as a processor that should be covered. For each processor to be covered, a copy of the task to be scheduled is assigned to a particular processor on which its completion time is minimized, and the child task on the former processor can then start as it was originally scheduled. This process repeats until all processors to be covered are actually covered. It is possible that a single task assignment can cover more than one processor. One drawback of this duplication scheme is that there might be a significant increase in schedule length if the number of

processors is very small compared to the number of tasks; this is because, although redundant duplications of a task might be effective for the task itself, its replicas can cause a 'cascade effect', in which the replicas invoke too many subsequent duplications. The time complexity of *DBUS* is in the order of $O(n^2p^2)$.

3.2 Scheduling in HCSsaScheduling with energy consciousness

Since CPUs are the major source of power consumption in DCSs [4], many microprocessor manufacturers including Intel, AMD, Motorolla and Transmeta have put a lot of effort into low-power processor design focusing on DVFS [15,20-22]. DVFS is a promising energy saving technique that can be incorporated into scheduling; and many scheduling algorithms (e.g., [5,23]) using DVFS have been proposed for different problems. However, the majority of previous studies on scheduling that take into consideration energy consumption are conducted on homogeneous computing systems [2-5,23] or single-processor systems [6]. In addition to system homogeneity, tasks are generally homogeneous and independent. Slack management/reclamation is a frequently adopted technique with DVFS.

In [3], several different scheduling algorithms using the concept of *slack sharing* among DVFS-enabled processors were proposed. The target system model and the task model are homogeneous multiprocessor systems, and heterogeneous independent and dependent real-time tasks with hard deadline, respectively. The rationale behind the algorithms is to utilize idle (slack) time slots of processors lowering voltage supply (frequency/speed). This technique is known as slack reclamation. These slack time slots occur, due to earlier completion (than worst case execution time) and/or dependencies of tasks. The scheduling algorithms for both independent and dependent tasks in [3] adopt global scheduling in which all tasks wait in a global queue and are dispatched based on their priorities. The work in [3] has been extended in [23] with AND/OR model applications. Since the target system for both works is shared-memory multiprocessor systems, communication between dependent tasks is not considered.

In [4], two voltage scaling algorithms for periodic, sporadic, and aperiodic tasks on a dynamic priority single-processor system were proposed. They are more practical compared with many existing DVFS algorithms in that a priori information on incoming tasks is not assumed to be available until the tasks are actually released.

Rountree et al. in [24] developed a system based on linear programming (LP) that exploits slack using DVS (i.e., slack reclamation). Their LP system aims to deliver near-optimal schedules that tightly bound optimal solutions. It incorporated allowable time delays, communication slack, and memory pressure into its scheduling. The LP system mainly deals with energy reduction for a given pre-generated schedule with a makespan constraint as in most existing algorithms.

Another two scheduling algorithms for bag-of-tasks (BoT) applications on clusters were proposed in [2]. Tasks in a BoT application are typically independent and homogeneous, yet run with different input parameters/files. In [2], deadline constraints are associated with tasks for the purpose of quality control. The two algorithms differ in terms of whether processors in a given computer cluster are time-shared or space-shared. Computer clusters in this paper are composed of homogeneous DVFS-enabled processors.

4. Task scheduling with energy consciousness

In this section, we begin by discussing design focuses of our algorithms formulating an objective function for each of our algorithms, and describe algorithmic details.

4.1 Characterization of design objectives

As in most multi-objective optimization problems, the goal in our task scheduling problem is to find Pareto-optimal solutions since the performance objectives of the problem are most likely to be in conflict with each other. In other words, for a given task graph, the heuristics presented in this study aim to generate a schedule that minimizes both the makespan and energy consumption; however, the reduction in energy consumption is often made by lowering supply voltage and this results in an increase in makespan. The incorporation of energy consumption into task scheduling adds another layer of complexity to an already intricate problem.

Unlike real-time systems, applications in our study are not deadline-constrained; this indicates that evaluation of the quality of schedules is not straightforward, rather the quality of schedules should be measured explicitly considering both makespan and energy consumption. For this reason, each of our algorithms (*ECS* and $ECS^{makespan}$) is devised with relative superiority (*RS*) as a novel objective function, which takes into account these two performance considerations. While *ECS* accounts for energy consumption and makespan equally, $ECS^{makespan}$ considers makespan as a preferred performance metric. The latter design preference can be explained such that recent and upcoming processors tend to be operated with lower power; and this trend

$$RS(n_i, p_j, v_{j,k}, p', v') = -\left(\left(\frac{E(n_i, p_j, v_{j,k}) - E(n_i, p', v')}{E(n_i, p_j, v_{j,k})}\right) + \left(\frac{EFT(n_i, p_j, v_{j,k}) - EFT(n_i, p', v')}{EFT(n_i, p_j, v_{j,k}) - \min(EST(n_i, p_j, v_{j,k}), EST(n_i, p', v'))}\right)\right) \quad (5)$$

$$RS'(n_i, p_j, v_{j,k}, p', v') = \begin{cases} -\left(\left(\frac{E(n_i, p_j, v_{j,k}) - E(n_i, p', v')}{E(n_i, p_j, v_{j,k})}\right) + \left(\frac{EFT(n_i, p_j, v_{j,k}) - EFT(n_i, p', v')}{EFT(n_i, p_j, v_{j,k}) - EST(n_i, p_j, v_{j,k})}\right)\right) & \text{if } EFT(n_i, p_j, v_{j,k}) < EFT(n_i, p', v') \\ -\left(\left(\frac{E(n_i, p_j, v_{j,k}) - E(n_i, p', v')}{E(n_i, p_j, v_{j,k})}\right) + \left(\frac{EFT(n_i, p', v') - EFT(n_i, p_j, v_{j,k})}{EFT(n_i, p', v') - EST(n_i, p', v')}\right)\right) & \text{if } EFT(n_i, p_j, v_{j,k}) \geq EFT(n_i, p', v') \end{cases} \quad (6)$$

is anticipated to be continuing. For example, a schedule (for a particular task graph) with lower makespan may consume less energy compared with another schedule with longer makespan, if the difference between the maximum VSL and the minimum VSL is small. While RS values for a task n_i on a processor p_j with a VSL $v_{j,k}$ in *ECS* and *ECS*^{makespan} are defined as

where $E(n_i, p_j, v_{j,k})$ and $E(n_i, p', v')$ are the energy consumption of n_i on p_j with $v_{j,k}$ and that of n_i on p' with v', respectively, and similarly the earliest start/finish times of the two task-processor allocations are denoted as $EST(n_i, p_j, v_{j,k})$ and $EST(n_i, p', v')$, and $EFT(n_i, p_j, v_{j,k})$ and $EFT(n_i, p', v')$.

4.2 Energy-conscious scheduling heuristics

Our algorithms (*ECS* and *ECS*^{makespan}) can be described as multi-pass (two-pass) algorithms—the main scheduling pass and the makespan-conservative energy reduction pass. The main scheduling pass plays as an initial schedule generator using the RS objective function; and schedules generated are further refined primarily based on their energy consumption. Both of our algorithms perform their scheduling nearly the same way with an exception being their RS objective function. Thus, we describe the common procedure involved in both algorithms in this sub section and in Figure 2.

For a given ready task, its *RS* value on each processor is computed using the current best combination of processor and VSL (p' and v') for that task, and then the processor—from which the maximum RS value is obtained—is selected (Steps 3-15).

Since each scheduling decision that *ECS* makes tends to be confined to a local optimum, another energy reduction technique (MCER) is incorporated with the energy reduction phase of *ECS* without sacrificing time complexity (Steps 17-31). It is an effective technique in lowering energy consumption, although the technique may not help schedules escape from local optima. MCER is makespan conservative in that changes it makes (to the schedule generated in the scheduling phase) are only validated if they do not increase the makespan of the schedule. For each task in a DAG, MCER considers all of the other combinations of task, host and VSL to check whether any of these combinations reduces the energy consumption of the task without increasing the current makespan.

The workings of *ECS* and an example of its scheduling are presented in Figures 2 and 4, respectively.

Algorithm *ECS*
Input: A DAG $G(N, E)$ and a set P of DVS-enabled processors
Output: A schedule S of G onto P
1. Compute b-level of $\forall n_i \in N$
2. Sort N in decreasing order by b-level value
3. **for** $\forall n_i \in N$ **do**
4. Let $p' = p_0$
5. Let $v' = v_{0,0}$
6. **for** $\forall p_j \in P$ **do**
7. **for** $\forall v_{j,k} \in V_j$ **do**
8. Compute $RS(n_i, p_j, v_{j,k}, p', v')$ value with p' and v'
9. **if** $RS(n_i, p_j, v_{j,k}, p', v') > RS(n_i, p', v', p_j, v_{j,k})$ **then**
10. Replace p' and v' with p_j and $v_{j,k}$
11. **end if**
12. **end for**
13. **end for**
14. Assign n_i on p' with v'
15. **end for**
16. Let S = the current schedule
17. **for** $\forall n_i \in N$ **do**
18. Remove n_i in S
19. Let p' = the processor on which n_i is currently scheduled
20. Let v' = the VSL selected for p'
21. **for** $\forall p_j \in P$ **do**
22. **for** $\forall v_{j,k} \in V_j$ **do**
23. Compute $E_d(n_i, p_j, v_{j,k})$
24. Recompute makespan
25. **if** no increase in makespan && $E_d(n_i, p_j, v_{j,k}) < E_d(n_i, p', v')$ **the**
26. Replace p' and v' with p_j and $v_{j,k}$
27. **end if**
28. **end for**
29. **end for**
30. Reassign n_i on p' with v'
31. **end for**

Fig. 2. The *ECS* algorithm

5. Performance evaluation

In this section, we discuss the qualitative implications of schedules that *ECS* and *ECS*^{makespan} generate. The discussion is led with an illustration of their scheduling of *ECS* (Figures 4 and 5).

We first present two schedules (Figure 3) generated by *HEFT* and *DBUS* for the task graph in Figure 1 to show our algorithms' capability of energy reduction. For fair comparisons a slack reclamation technique is used for *HEFT* and *DBUS* (Figure 3), although they originally do not incorporate DVFS or any other energy saving techniques into their scheduling.

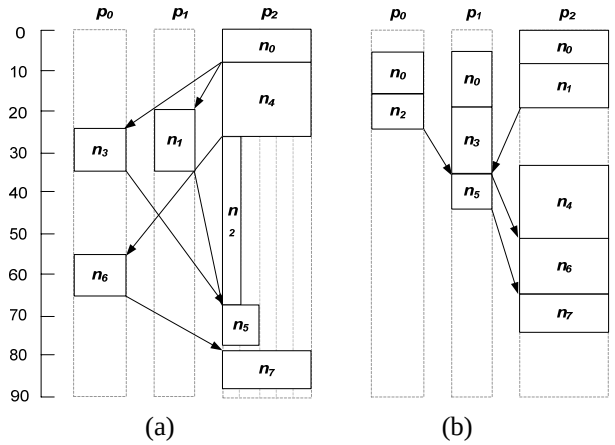

Fig. 3. Schedules of task graph in Figure 1 with slack reclamation. (a) *HEFT* (makespan=89). (b) *DBUS* (makespan=73)

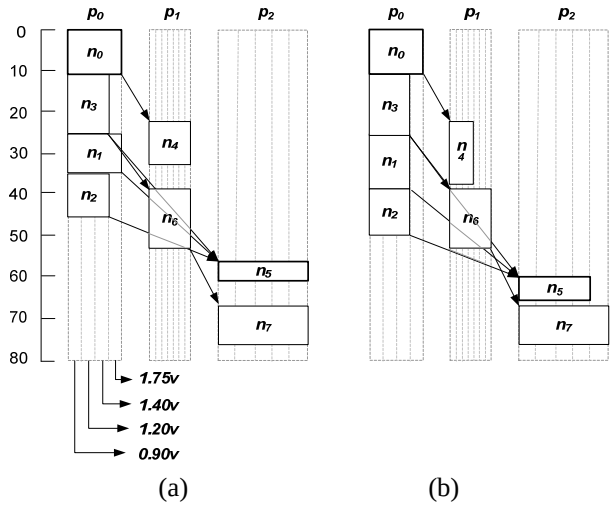

Fig. 4. Schedules of task graph in Figure 1. (a) ECS without MCER (makespan=77). (b) ECS with MCER (makespan=77)

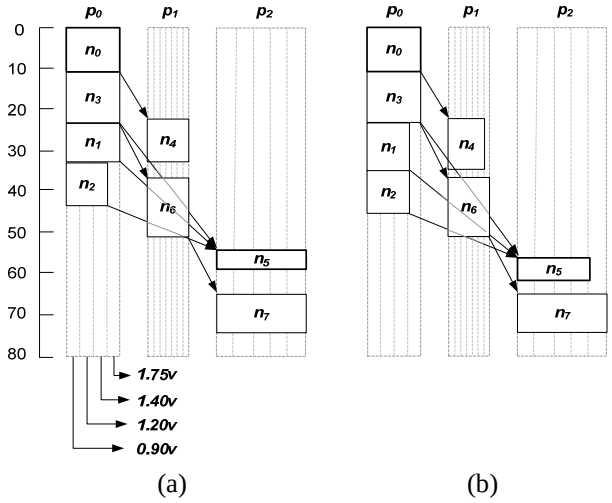

Fig. 5. Schedules of task graph in Figure 1. (a) *ECS*^{makespan} without MCER (makespan=74). (b) *ECS*^{makespan} with MCER (makespan=74)

The schedules in Figure 4 and 5 are generated with an α of 1 and the first three sets of voltage-relative speed pairs in Table 1 (i.e., pairs 1, 2 and 3). The schedules in Figures 4a and 5a are intermediate schedules generated by *ECS* and *ECS*^{makespan} before the MCER technique is applied; these schedules are already superior to those in Figure 3 in terms of both makespan and energy consumption (Table 4). This high quality of schedule is achieved using *RS*. Although the scheduling of *ECS* and *ECS*^{makespan} based on *RS* is very effective and efficient, it is observed that there are often cases in which schedules can be improved even more, particularly in terms of energy consumption. MCER plays a crucial role in this respect primarily exploiting idling time slots identified in the schedule generated in the main scheduling phase. As shown in Figures 4b and 5b, MCER further improves the energy savings of the schedule using DVFS. Both *ECS* and *ECS*^{makespan} made three changes to VSLs of tasks 1, 4 and 5 in the initial schedule exploiting two slacks identified between tasks 4 and 6, and tasks 5 and 7. However, due to the difference in their RS metrics, schedules vary to a certain degree. Those schedules in Figures 4 and 5 are well reflected difference design focuses of our algorithms. Specifically, the schedule *ECS* generated is the best in terms of energy consumption (236.98), whereas that *ECS*^{makespan} generated is the most compelling in terms of makespan (74).

The time complexity of both *ECS* and *ECS*^{makespan} is in the order of $O(nlogn+2((e+n)pv))$, which still remains at a reasonably low level.

Table 4. Energy consumption of schedules generated by different algorithms

algo / task	*HEFT*		*DBUS*		ECS^{-MCER}		ECS^{+MCER}		$ECS^{makespan-MCER}$		$ECS^{makespan+MCER}$	
	time	energy	time	energy	time	energy	time	energy	time	energy	time	energy
n_0	9	43.56	11 13 9	29.25 33.69 43.56	11	33.69	11	33.69	11	33.69	11	33.69
n_1	15	33.75	11	53.24	10	30.63	13	25.48	10	30.63	13	25.48
n_2	14	40.00	40	27.56	11	21.56	11	21.56	11	21.56	11	21.56
n_3	12	36.75	17	38.25	15	29.40	15	29.40	12	36.75	12	36.75
n_4	19	91.96	19	91.96	11	24.75	16	23.04	10	24.75	11	23.52
n_5	5	16.90	10	20.25	5	24.20	6	21.66	5	24.20	6	21.66
n_6	11	33.69	13	62.92	15	33.75	15	33.75	15	33.75	15	33.75
n_7	10	48.40	10	48.40	10	48.40	10	48.40	10	48.40	10	48.40
total		**345.01**		**449.08**		**246.38**		**236.98**		**253.73**		**244.81**

6. Experiments and results

A large number of experiments were conducted using our simulator developed using C/C++. Our evaluation study was primarily carried out based on comparisons between *ECS* and $ECS^{makespan}$, and two existing heuristics (*HEFT* and *DBUS*). The latter two algorithms were modified with the incorporation of a slack reclamation technique in order for fair comparisons. The slack reclamation technique adopted essentially attempts to identify that any VSL changes (i.e., lowering VSLs) are possible without an increase in makespan.

6.1 Experimental settings

To ensure experiments being at a reasonable level in terms of practicality and reality, we have carefully configured simulations with a diverse set of simulation parameters (Table 5) and with two extensive sets of task graphs: randomly generated, and real-world application. The three real-world parallel applications used for our experiments were the Laplace equation solver [25], the LU-decomposition [26] and Fast Fourier Transformation [27]. A large number of variations were made on these task graphs for more comprehensive experiments.

Table 5. Experimental parameters

Parameter	Value
The number of tasks	U(10, 600)
CCR	{0.1, 0.2, 1.0, 5.0, 10.0}
The number of processors	{2, 4, 8, 16, 32, 64}
Processor heterogeneity	{100, 200, random}

The total number of experiments conducted with different task graphs on the six different numbers of processors is 288,000 (i.e., 72,000 for each algorithm). Specifically, the random task graph set consisted of 150 base task graphs generated with combinations of 10 graph sizes, five CCRs and three processor heterogeneity settings. For each combination, 20 task graphs were randomly generated, retaining the characteristics of the base task graph. These 3,000 graphs were investigated with six different numbers of processors. Each of the three applications were investigated using the same number of task graphs (i.e., 18,000); hence the figure 72,000.

The computational and communication costs of the tasks in each task graph were randomly selected from a uniform distribution, with the mean equal to the chosen average computation and communication costs. A processor heterogeneity value of 100 was defined to be the percentage of the speed difference between the fastest processor and the slowest processor in a given system. For the real-world application task graphs, the matrix sizes and the number of input points were varied, so that the number of tasks can range from about 10 to 600.

6.2 Comparison metrics

Typically, the makespan of a task graph generated by a scheduling algorithm is used as the main performance measure; however, in this study, we consider energy consumption as another equally important performance measure. For a given task graph, we normalize both its makespan and energy consumption to lower bounds—the makespan and energy consumption of the tasks along the

CP (i.e., CP tasks) without considering communication costs. Specifically, the 'schedule length ratio' (SLR) and 'energy consumption ratio' (ECR) were used as the primary performance metrics for our comparison. Formally, the SLR and ECR values of the makespan M and energy consumption E of a schedule generated for a task graph G by a scheduling algorithm are defined as

$$SLR = \frac{M}{\sum_{i=1}^{|CP|} n_i \in CP \min_{p_j \in P}\{w_{i,j}\}} \quad (6)$$

$$ECR = \frac{E}{\sum_{i=1}^{|CP|} n_i \in CP \min_{p_j \in P}\{w_{i,j}\} \times \max_{v_{j,k} \in V_j}\{v_{j,k}\}^2} \quad (7)$$

where CP is a set of CP tasks of G.

6.3 Results

As we try to balance two conflicting objectives (makespan and energy consumption), our experimental results in this section are presented based on these two objectives using *SLR* and *ECR*. The performance of *HEFT* and *DBUS* in terms of energy consumption is noticeably improved compared to that of their original design; this can be identified by results reported in [14]. However, overall their performance is still not comparable to the performance of our algorithms.

The overall comparative results between our algorithms and the two previous algorithms are presented in Table 6 followed by the results for each different DAG set (Figures 6 and 7). Table 6 clearly signifies the superior performance of our algorithms over *HEFT* and *DBUS*, irrespective of different DAG types. In addition, *ECS* and *ECS*^{makespan} outperformed these two previous algorithms consistently with various different CCRs as shown in Figures 6 and 7. Note that makespans of schedules generated by *HEFT* and *DBUS* using a slack reclamation technique are the same as those generated by their original counterparts since the slack reclamation technique only exploits slack times identified in schedules generated by original *HEFT* and *DBUS*.

Once again, *HEFT* and *DBUS* are well known heuristics with their scheduling quality (i.e., makespan); however, their makespan-centric design fails (or is at least incompetent) to account for the energy consumption of schedules they generate. Particularly, in the case of *DBUS* excessive energy consumption nearly prohibits its use in the current computing environment. *HEFT* has been proven to perform very competitively with a low time complexity, and it has been frequently adopted and extended; this implies that the average SLR of *ECS* and *ECS*^{makespan} with even a one percent margin (Table 6) is compelling. Note that *ECS*^{makespan} has improved the quality of schedules over its original

Table 6. Comparative results

Algorithm / DAG set	*HEFT over ECS*		*DBUS* over *ECS*		*HEFT* over *ECS*^{makespan}		*DBUS* over *ECS*^{makespan}	
	makespan	energy	makespan	energy	makespan	energy	makespan	energy
Random	2%	6%	5%	38%	3%	6%	6%	35%
FFT	< 1%	6%	16%	39%	1%	5%	17%	39%
Laplace	1%	7%	22%	43%	2%	7%	21%	39%
LU	-3%	8%	2%	21%	1%	6%	4%	20%
Average	0%	7%	11%	35%	2%	6%	12%	33%

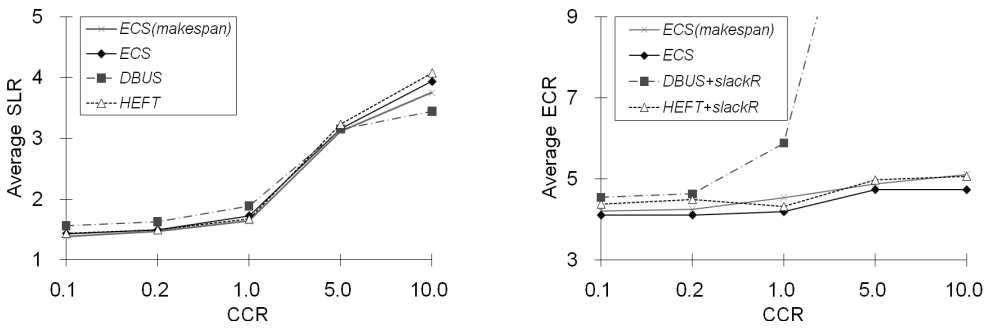

Fig. 6. Average SLR and ECR for random DAGs

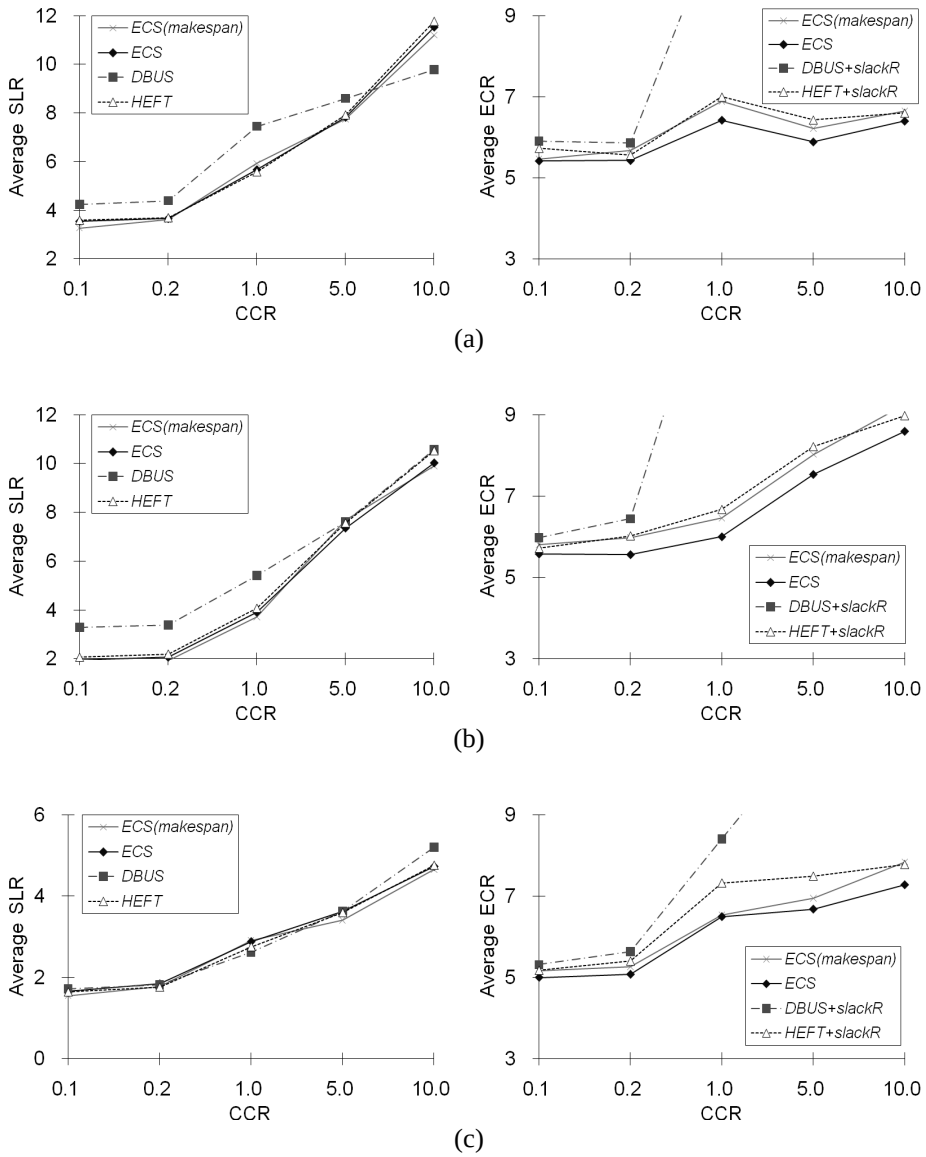

Fig. 7. Average SLR and ECR for real-world application DAGs. (a) FFT. (b) Laplace. (c) LU

counterpart (*ECS*) in terms of makespan.

The source of the main performance gain of our algorithms is the use of the RS objective function, which contributes to reducing both makespan and energy consumption. In our experiments, on average a further 3.2 percent and 2.9 percent improvement in energy consumption—for schedules after the main scheduling phase of *ECS* and $ECS^{makespan}$—was made by the MCER technique.

7. Conclusion

As energy consumption is increasingly getting attention in DCSs due to their operational and environment implications, resource management practices (particularly scheduling practices) in these systems is revisited and revised to improve energy efficiency. In this paper we have presented an energy-conscious scheduling algorithm as an extension of our previous work. Unlike most existing scheduling algorithms regardless of the incorporation of energy consumption in their scheduling, our algorithms explicitly take into account both makespan and energy consumption. Since these two performance metrics (makespan and energy consumption) are closely correlated, the RS objective function devised as part of our study plays a key role in balancing these two performance objectives. This claim and the superior performance of our algorithms are confirmed with results presented in this paper.

References

[1] Venkatachalam, V. and Franz, M. "Power reduction techniques for microprocessor systems", *ACM Computing Surveys*, 37(3), pp.195-237, 2005.

[2] Kim, K. H. et al. "Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters", Proc. of the 7th IEEE International Symposium on Cluster Computing and the Grid, pp.541-548, 2007.

[3] Zhu, D., et al. "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems", *IEEE Trans. Parallel and Distributed Systems*, 14(7), pp.686-700, 2003.

[4] Ge, R., et al. "Performance-constrained distributed DVS scheduling for scientific applications on power-aware clusters", Proc. of the ACM/IEEE Conference on Supercomputing, pp.34-44, 2005.

[5] Chen, J. J. and Kuo, T. W. "Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics", Proc. of International Conference on Parallel Processing, pp.13-20, 2005.

[6] Zhong, X. and Xu, C.-Z. "Energy-aware modeling and scheduling for dynamic voltage scaling with statistical real-time guarantee", *IEEE Trans. Computers*, 56(3), pp.358-372, 2007.

[7] J. G. Koomey, Estimating total power consumption by servers in the U.S. and the world.

[8] G. Koch, Discovering multi-core: extending the benefits of Moore's law, Technology@Intel Magazine, July 2005 (http://www.intel.com/technology/magazine/computing/multi-core-0705.pdf).

[9] D. P. Bunde, Power-aware scheduling for makespan and flow, Proc. the eighteenth annual ACM symposium on Parallelism in algorithms and architectures, July, 2006.

[10] S. Darbha and D. P. Agrawal, Optimal scheduling algorithm for distributed-memory machines, IEEE Trans. Parallel and Distributed System, Vol.9 , No.1, 1998, pp.87-95.

[11] A. Y. Zomaya, C. Ward, and B. S. Macey, Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues, IEEE Trans. Parallel Distrib. Syst., Vol.10, No.8, pp.795-812, 1999.

[12] H. Topcuouglu, S. Hariri, and M.-Y. Wu, Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing, IEEE Trans. Parallel Distrib. Syst., Vol.13, No.3, pp.260-274, 2002.

[13] Y. C. Lee and A. Y. Zomaya, A Novel State Transition Method for Metaheuristic-Based Scheduling in Heterogeneous Computing Systems, IEEE Trans. Parallel Distrib. Syst., Vol.19, No.9, pp.1215-1223, 2008.

[14] Y. C. Lee, and A. Y. Zomaya, Minimizing Energy Consumption for Precedence-constrained Applications Using Dynamic Voltage Scaling, Proceedings of the International Symposium on Cluster Computing and the Grid (CCGRID), May, 18-21, pp.92-99, 2009.

[15] Intel, Intel Pentium M Processor datasheet, 2004.

[16] R. Min, T. Furrer, and A. Chandrakasan, Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks, Proc. IEEE Workshop on VLSI, pp.43-46, April, 2000.

[17] M.R. Garey and D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman and Co., pp.238-239, 1979.

[18] Y. K. Kwok and I. Ahmad, Benchmarking the Task Graph Scheduling Algorithms, *Proc. First Merged Int'l Parallel Symposium/Symposium on Parallel and Distributed Processing (IPPS/SPDP '98)*, pp.531-537, 1998.

[19] D. Bozdag, U. Catalyurek and F. Ozguner, A task duplication based bottom-up scheduling algorithm for heterogeneous environments, *Proc. Int'l Parallel and Distributed Processing Symp.*, April, 2005.

[20] AMD, AMD Athlon™ 64 Processor Power and Thermal Data Sheet, 2006.

[21] C. Pyron, M. Alexander, J. Golab, G. Joos, B. Long, R. Molyneaux, R. Raina, and N. Tendolkar, DFT advances in the Motorola's MPC7400, a PowerPC™ microprocessor, Proc. Int'l Test Conference, pp.137-146, 1999.

[22] D. R. Ditzel and the Transmeta LongRun2 team, Power Reduction using LongRun2 in Transmeta's Efficeon Processor, Spring processor forum, 2006.

[23] D. Zhu, D. Mosse, and R. Melhem, Power-aware scheduling for AND/OR graphs in real-time systems, IEEE trans. Parallel and distributed Systems, Vol.15, No.9, pp.849-864, 2004.

[24] B. Rountree, D. K. Lowenthal, S. Funk, V. W. Freeh, B. R. de Supinski, M. Schulz, Bounding energy consumption in large-scale MPI programs, Proc. the ACM/IEEE conference on Supercomputing, November, 2007.

[25] M.-Y. Wu and D.D. Gajski, Hypertool: A Programming Aid for Message-Passing Systems, IEEE Trans. Parallel and Distributed Systems, Vol.1, No.3, pp.330-343, July, 1990.

[26] R.E. Lord, J.S. Kowalik, and S.P. Kumar, Solving Linear Algebraic Equations on an MIMD Computer, J. ACM, Vol.30, No.1, pp.103-117, January, 1983.

[27] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, Introduction to Algorithms, MIT Press, 1990.

Young Choon Lee
He received the BSc (hons) in computer science in 2003 and the Ph.D. degree from the School of Information Technologies at the University of Sydney in 2008. He is currently with the Centre for Distributed and High Performance Computing, School of Information Technologies. His current research interests include scheduling strategies for heterogeneous computing systems including clouds, nature-inspired techniques, and parallel and distributed algorithms. He is a member of the IEEE and the IEEE Computer Society.

Albert Y. Zomaya
He is currently the Chair Professor of High Performance Computing and Networking in the School of Information Technologies, The University of Sydney. He is also the Director for the newly established Sydney University Centre for Distributed and High Performance Computing. Prior to joining Sydney University he was a full professor in the Electrical and Electronic Engineering Department at the University of Western Australia, where he also led the Parallel Computing Research Laboratory during the period 1990-2002. He is the author/co-author of seven books, more than 350 publications in technical journals and conferences, and the editor of eight books and eight conference volumes. He is currently an associate editor for 16 journals, the Founding Editor of the Wiley Book Series on Parallel and Distributed Computing and a Founding Co-Editor of the Wiley Book Series on Bioinformatics. Professor Zomaya was the Chair of the IEEE Technical Committee on Parallel Processing (1999-2003) and currently serves on its executive committee. He also serves on the Advisory Board of the IEEE Technical Committee on Scalable Computing and IEEE Systems, Man, and Cybernetics Society Technical Committee on Self-Organization and Cybernetics for Informatics, and is a Scientific Council Member of the Institute for Computer Sciences, Social-Informatics, and Telecommunications Engineering (in Brussels). He received the 1997 Edgeworth David Medal from the Royal Society of New South Wales for outstanding contributions to Australian Science. Professor Zomaya is also the recipient of the Meritorious Service Award (in 2000) and the Golden Core Recognition (in 2006), both from the IEEE Computer Society. He is a Chartered Engineer (CEng), a Fellow of the American Association for the Advancement of Science, the IEEE, the Institution of Electrical Engineers (UK), and a Distinguished Engineer of the ACM. His research interests are in the areas of high performance computing, parallel algorithms, mobile computing, and bioinformatics.