Energy Conscious Scheduling for Distributed Computing Systems under Different Operating Conditions

Young Choon Lee, *Member*, *IEEE*, and Albert Y. Zomaya, *Fellow*, *IEEE*

Abstract—Traditionally, the primary performance goal of computer systems has focused on reducing the execution time of applications while increasing throughput. This performance goal has been mostly achieved by the development of high-density computer systems. As witnessed recently, these systems provide very powerful processing capability and capacity. They often consist of tens or hundreds of thousands of processors and other resource-hungry devices. The energy consumption of these systems has become a major concern. In this paper, we address the problem of scheduling precedence-constrained parallel applications on multiprocessor computer systems and present two energy-conscious scheduling algorithms using dynamic voltage scaling (DVS). A number of recent commodity processors are capable of DVS, which enables processors to operate at 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. To effectively balance these two performance goals, we have devised a novel objective function and a variant from that. The main difference between the two algorithms is in their measurement of energy consumption. The extensive comparative evaluations conducted as part of this work show that the performance of our algorithms is very compelling in terms of both application completion time and energy consumption.

Index Terms—Computer systems organization, energy-aware systems, scheduling and task partitioning, simulation of multiprocessor systems, multicomputer systems, performance of systems.

1 Introduction

Stiff increases in the volume of both computation and data over the past few decades have spurred computer architecture researchers and designers to focus on high performance; which led to the development of resource-intensive technologies, such as multicore microprocessors, high-capacity storage devices, and super-speed communications devices. The efficient use of these powerful resources has always been a crucial issue, particularly for multiprocessor or multicomputer systems (MCSs), such as grids and clouds, in which there are a multitude of processors often as many as hundreds of thousands of them.

Most MCSs operate with some form of resource management facility (such as PBS, LSF, etc.) for achieving performance goals like high performance, high throughput and/or high availability [1]. However, most existing resource management facilities generally lack energy-saving capability. Therefore, it is not very hard to imagine the size of adverse environmental footprint that MCSs leave; this issue has gained a particular interest, in recent times, with the growing advocacy for green computing systems. Until recently energy issues have been mostly dealt with by advancements in hardware technologies [2], such as low-power microprocessors (CPUs), solid state drives, and energy-efficient computer monitors.

Energy-aware scheduling and resource allocation have emerged as promising approaches for sustainable/green computing. Although many algorithms and strategies have been developed, their scope is quite restricted to battery-powered embedded systems [3], homogeneous computing systems [4], [5], [6], [7] or single-processor systems [8]. In addition to system homogeneity, tasks are often homogeneous or independent.

In this paper, we address the energy-aware scheduling of precedence-constrained parallel applications for MCSs consisting of heterogeneous resources. Since these applications in scientific and engineering fields are the most typical application model, the problem of scheduling these applications (task scheduling) has been studied extensively over the past few decades (e.g., [9], [10], [11], [12]). 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 objective 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 for MCSs [4], [5], [7], [13].

The energy consumption issue in these MCSs raises various monetary, environmental and system performance concerns. A recent study on power consumption by servers [14] 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

• *The authors are with Center for Distributed and High Performance Computing, School of Information Technologies, University of Sydney, NSW 2006, Australia. E-mail: {yclee, zomaya}@it.usyd.edu.au.*

Manuscript received 8 Oct. 2009; revised 4 June 2010; accepted 7 June 2010; published online 21 Dec. 2010.
Recommended for acceptance by K. Li.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-10-0497.
Digital Object Identifier no. 10.1109/TPDS.2010.208.

1045-9219/11/$26.00 © 2011 IEEE Published by the IEEE Computer Society

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 to 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 [15]; 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 scaling (DVS), resource hibernation, and memory optimizations have been investigated and developed [2]. DVS among these has been proven to be a very promising technique with its demonstrated capability for energy savings (e.g., [4], [6], [7], [13]). For this reason, we adopt this technique and it is of particular interest to this study. DVS enables processors to dynamically adjust voltage supply levels (VSLs) aiming to reduce power consumption; however, this reduction is achieved at the expense of sacrificing clock frequencies.

This paper presents two energy-conscious scheduling heuristics (*ECS* [16] and $ECS{+}idle$) that take into account not only makespan, but also energy consumption. Both algorithms are devised with the incorporation of DVS to reduce energy consumption; this implies that there is a trade-off between the quality of schedules (makespans) and energy consumption. The 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 $ECS{+}idle$. During MCER, the current schedule which was generated during the scheduling phase is scrutinized to identify whether any changes to the schedule can reduce further energy consumption without an increase in makespan. The main difference between the two algorithms lies in their measurement of energy consumption of schedules. Specifically, while *ECS* only takes into account energy consumed for running tasks, $ECS{+}idle$ explicitly accounts for the total energy consumption for a given schedule including energy consumed during idling periods of processors. Both the main objective function and MCER used in *ECS* have been revised in order for $ECS{+}idle$ to deal more comprehensively with energy consumption of schedules.

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. The *ECS* and $ECS{+}idle$ algorithms are presented in Section 4. The results of our comparative evaluation study are presented in Section 5. Section 6 concludes the paper.

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 DVS 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). Processors consume energy while idling; that is, when a processor is idling it is assumed that the lowest voltage is supplied. Since clock frequency transition overheads take a negligible amount of time (e.g., 10 μs-150 μs [17], [18]), these overheads are not considered in our study. Interprocessor 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 which is possible in many systems.

TABLE 1
Voltage-Relative Speed Pairs

level	pair 1		pair 2		pair 3		pair 4	
	voltage (v_k)	relative speed (%)	voltage (v_k)	relative speed (%)	voltage (v_k)	relative speed (%)	voltage (v_k)	relative 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 intertask 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_j ni, 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., zero. The earliest start and earliest finish times of, a task n_i on a processor p_j are defined as:

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

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

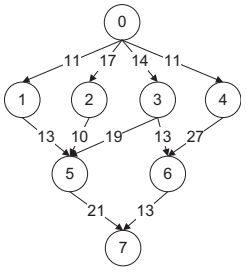

Fig. 1. A simple task graph.

where p_k is the processor on which the MIP of task n_i is scheduled.

Note that, the actual start and finish times of a task n_i on a processor p_j, are 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 $EFT(n_i, p_j)$.

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*. Note that, both computation and communication costs are averaged over all nodes and links. 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.

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:

$$P_c = ACV^2 f \tag{3}$$

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.

Since processors consume a certain amount of energy while idling, the total energy consumption of the execution for a precedence-constrained parallel application in this study is comprised of direct and indirect energy consumption. The direct energy consumption is defined as:

$$E_d = \sum_{i=1}^{n} ACV_i^2 f \cdot w_i^* = \sum_{i=1}^{n} \alpha V_i^2 w_i^*, \tag{4}$$

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. On the other hand, the indirect energy consumption is defined as:

$$E_i = \sum_{j=1}^{p} \sum_{idle_{j,k} \in IDLE_j} \alpha V_{j,low}^2 t_{j,k}, \tag{5}$$

where $IDLE_j$ is the set of idling slots on processor p_j, $V_{j,low}$ is the lowest supply voltage on p_j, and $t_{j,k}$ is the amount of idling time for $idle_{j,k}$. Then, the total energy consumption is defined as:

$$E_t = E_d + E_i. \tag{6}$$

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. Since the two objectives (minimization of

TABLE 2
Task Priorities

n_i	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

n_i	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

makespan and energy consumption) in our scheduling model conflict with each other, scheduling decisions should be made accounting for the impact of each of those objectives on the quality of schedule.

3 Related Work

In this section, we present two noteworthy works in task scheduling, particularly for MCSs, and then scheduling algorithms with power/energy consciousness.

3.1 Scheduling in MCSs

Due to the NP-hard nature of the task scheduling problem in general cases [19], heuristics are the most popular scheduling model adopted by many researchers, and they deliver good solutions in less than polynomial time.

The *HEFT* algorithm [11] is a well-known list-scheduling heuristic. It consists 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 *DBUS* algorithm [20] is a duplication-based scheduling heuristic that performs a CP-based listing for tasks and schedules them with task duplication and insertion.

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.

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.

3.2 Scheduling with Energy Consciousness

DVS is a promising energy saving technique that can be incorporated into scheduling; and many scheduling algorithms (e.g., [6], [21]) using DVS 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 [4], [6], [7], [21]. Slack management/reclamation is a frequently adopted technique with DVS.

CPU slacks during the runtime of a given application are an obvious source of energy wastage. They occur due primarily to communication events, for example with memory/IO devices or between tasks (intertask data dependencies). In [22], [23], [24], power-performance tradeoffs are studied for (sequential) applications with non/soft-real-time deadlines. Since these applications should be completed within certain time frames and their execution can be at most partitioned into a series of on-chip and off-chip tasks, energy savings can only be possible at the expense of performance. Our work differs in that it focuses on exploiting slacks (CPU idle time slots)—inevitably occurring due to "multidimensional" intertask communication—with no or little effect on the overall application completion time.

In [6], several different scheduling algorithms using the concept of *slack sharing* among DVS-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 [6] adopt global scheduling in which all tasks wait in a global queue and are dispatched based on their priorities. The work in [6] has been extended in [25] with AND/OR model applications. Since the target system for both works is shared-memory multiprocessor systems, communication between tasks is not considered.

Rountree et al. in [13] 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 pregenerated schedule with a makespan constraint as in most existing algorithms.

4 Energy-Conscious Scheduling Heuristics

In this section, we present two energy-conscious scheduling heuristics (*ECS* and $ECS{+}idle$). Their scheduling decisions are made using the relative superiority metric (RS) devised as a novel objective function. In addition to RS, the makespan-conservative energy reduction technique is incorporated into these heuristics to exploit indirect energy consumption. The main objective of *ECS* and $ECS{+}idle$ is to optimize the quality of output schedules in terms of energy consumption relative to makespan with a low time complexity, $O(nlogn + 2((e+n)pv))$.

4.1 ECS

The incorporation of energy consumption into task scheduling adds another layer of complexity to this 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. The RS

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 and $E_d(n_i, p_j, v_{j,k}) < E_d(n_i, p', v')$ **then**
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.

metric devised and incorporated into *ECS* (Fig. 2) effectively deals with this trade-off. Specifically, the RS value of a scheduling alternative (task-processor-VSL combination) is of a measure to identify the degree of energy efficiency relative to task execution time. For each scheduling combination in consideration, its RS value is computed in addition to that of the best combination seen up to that point of decision making; that is, the latter is recomputed with the current combination being considered. The actual RS value computation starts from the second combination due to the involvement of two combinations in each computation. A positive RS value indicates the finding of a new best scheduling alternative. For a given task n_i, the RS value of a scheduling combination of processor p_j and VSL $v_{j,k}$ with the best combination of p' and v' is defined as:

$$RS(n_i, p_j, v_{j,k}, p', v') = - \left(\left(\frac{E_d(n_i, p_j, v_{j,k}) - E_d(n_i, p', v')}{E_d(n_i, p_j, v_{j,k})} \right) + (\{EFT(n_i, p_j, v_{j,k}) - EFT(n_i, p', v')\} / \{EFT(n_i, p_j, v_{j,k}) - \min(EST(n_i, p_j, v_{j,k}), EST(n_i, p', v'))\}) \right), \quad (7)$$

where $E_d(n_i, p_j, v_{j,k})$ and $E_d(n_i, p\prime, v\prime)$ 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 $EFT(n_i, p', v')$. For a given ready task, its RS value with each pair of processor and VSL is computed using the current best combination of processor and VSL (p' and v') for that task, and then the best combination—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.

4.2 ECS+idle

For a given schedule, it is normally the case that a shorter makespan yields less energy consumption primarily due to the energy consumption associated with idling slots of processors within the schedule. This observation leads us to make modifications to the original RS objective function and the MCER technique in order to incorporate indirect energy consumption.

The original RS objective function is revised to make the second term more effective in terms of reduction in indirect energy consumption. This change enforces (for two task-processor combinations in comparison) the processor-VSL match that delivers a shorter task completion time to impact more on the final RS value.

$$RS(n_i, p_j, v_{j,k}, p', v') = \begin{cases} -\left(\left(\frac{E_d(n_i,p_j,v_{j,k}) - E_d(n_i,p',v')}{E_d(n_i,p_j,v_{j,k})} \right) + \left(\frac{EFT(n_i,p_j,v_{j,k}) - EFT(n_i,p',v')}{EFT(n_i,p_j,v_{j,k}) - EST(n_i,p_j,v_{j,k})} \right) \right) \\ \qquad if\ EFT(n_i, p_j, v_{j,k}) < EFT(n_i, p', v') \\ -\left(\left(\frac{E_d(n_i,p_j,v_{j,k}) - E_d(n_i,p',v')}{E_d(n_i,p_j,v_{j,k})} \right) + \left(\frac{EFT(n_i,p',v') - EFT(n_i,p_j,v_{j,k})}{EFT(n_i,p',v') - EST(n_i,p',v')} \right) \right) \\ \qquad otherwise \end{cases} . \quad (8)$$

The change made to the original MCER technique in *ECS* is in its energy function. That is, reassignment decisions are made based on the actual energy consumption metric (E_a) instead of the direct energy consumption metric (E_d). The actual energy consumption of a task n_i on a processor p_j with a supply voltage of $v_{j,k}$ is defined as:

$$E_a(n_i, p_j, v_{j,k}) = E_d(n_i, p_j, v_{j,k}) - E_i(n_i, p_j, v_{j,k}). \quad (9)$$

The modified MCER technique seems to make reductions in energy consumption in a similar way to its original counterpart; however, there are some cases in which reductions are made only if the modified MCER technique is used. For instance, if a task on a processor with a supply voltage of 1.50 has its computation cost being 10, and it is considered for a lower VSL of 1.10v with which the computation cost of that task is 20, the original MCER technique will not take this alternative allocation since the (direct) energy consumption for the latter allocation is higher (i.e., 22.5 versus 24.2). If the VSL during idling is 0.80v, however, the modified MCER technique considering

TABLE 4
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 actual energy consumption identifies the latter allocation ($E_a = 11.4$; 16.1 versus 11.4) more energy efficient.

For the working of $ECS{+}idle$, Fig. 2 can be referred with the two differences—in the RS objective function and the MCER technique—described earlier in this section.

5 Experiments and Results

This section presents the results obtained from our extensive comparative evaluation study between *ECS* and $ECS{+}idle$, and two previously proposed heuristics (*HEFT* and *DBUS*) introduced in Section 3.1. Although the scheduling of these previous algorithms is energy "unconscious," they were proven to perform well for the task scheduling problem; in addition, none of the existing scheduling algorithms is directly applicable to such a problem. This comparison between *ECS* and $ECS{+}idle$, and *HEFT* and *DBUS* clearly demonstrates the energy saving capability of our algorithms.

5.1 Experimental Settings

The performance of *ECS* and $ECS{+}idle$ was thoroughly evaluated 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 [26], the LU-decomposition [27] and Fast Fourier Transformation [29]. A large number of variations were made on these task graphs for more comprehensive experiments. In addition to task graphs, various different characteristics of processors were applied to simulations. Table 4 summarizes the parameters used in our experiments.

The number of experiments conducted with four different algorithms (*HEFT*, *DBUS*, *ECS*, $ECS{+}idle$) 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 variant 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 real-world applications was 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.

TABLE 5
Comparative Results

algo / DAG set	*HEFT over ECS*		*DBUS* over *ECS*		*HEFT* over *ECS+idle*		*DBUS* over *ECS+idle*		*ECS* over *ECS+idle*	
	make-span	energy	make-span	energy	make-span	energy	make-span	energy	make-span	energy
random	2%	19%	5%	49%	3%	29%	7%	56%	3%	7%
FFT	< 1%	17%	16%	43%	2%	23%	18%	49%	2%	3%
Laplace	1%	11%	22%	45%	4%	15%	19%	54%	<1%	11%
LU	-3%	16%	2%	33%	0%	21%	5%	44%	2%	5%
average	0%	16%	11%	43%	2%	22%	13%	51%	2%	7%

5.2 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_t of a schedule generated for a task graph G by a scheduling algorithm are defined as:

$$SLR = \frac{M}{\sum_{n_i \in CP} \min_{p_j \in P}\{w_{i,j}\}} \tag{10}$$

$$ECR = \frac{E_t}{\sum_{n_i \in CP} \min_{p_j \in P}\{w_{i,j}\} \times \max_{v_{j,k} \in V_j}\{v_{j,k}\}^2}, \tag{11}$$

where CP is a set of CP tasks of G.

5.3 Results

The overall comparative results from our evaluation study are summarized in Table 5, followed by the results for each different DAG set (Figs. 3 and 4). Table 5 clearly signifies the superior performance of our algorithms over *DBUS* and *HEFT*, irrespective of different DAG types. Specifically, schedules generated by *DBUS* and *HEFT* consumed on average 43 percent and 16 percent, and 51 percent and 22 percent more energy than *ECS* and $ECS{+}idle$, respectively. In addition, our proposed algorithms mostly outperformed those two previous algorithms with various different CCRs as shown in Figs. 3 and 4.

Obviously, the primary purpose of task duplication is the minimization of communication overheads—which eventually reduces makespan—by redundantly scheduling

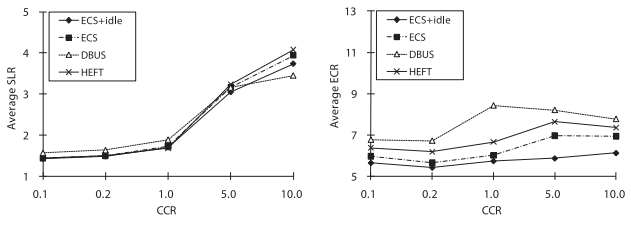

Fig. 3. Average SLR and ECR for random DAGs.

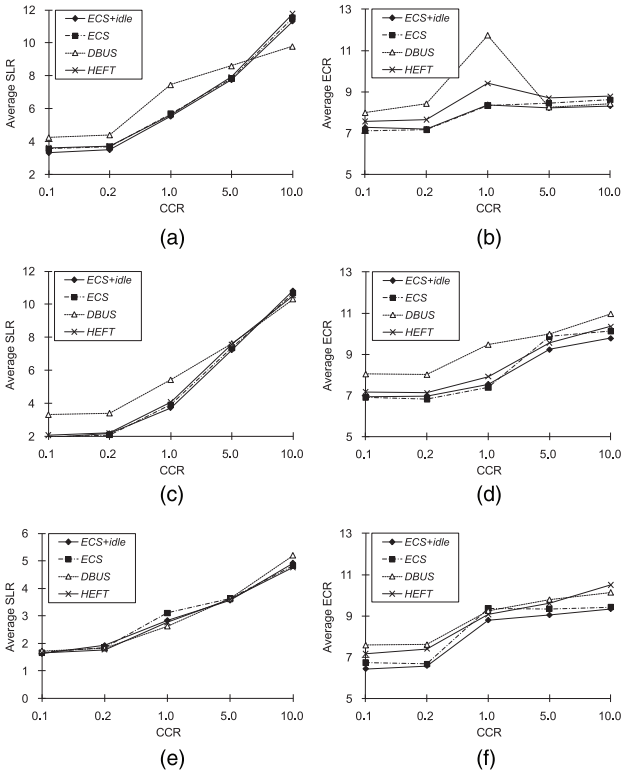

Fig. 4. Average SLR and ECR for real-world application DAGs. (a) FFT. (b) Laplace. (c) LU.

some tasks. Task duplication might be an effective technique for communication-intensive applications; however, the incorporation of such a technique into scheduling algorithms should be avoided, or at least carefully considered, when energy consumption comes into play. Since *DBUS* is a duplication-based scheduling algorithm, its energy consumption is far greater than that of *ECS* and $ECS{+}idle$. Note that when task graphs are communication intensive (CCRs of 5.0 and 10.0), the energy consumption of schedules generated by *DBUS* is comparable; this is due to good makespans resulted from task duplication.

The comparison between *ECS* and $ECS{+}idle$, and *HEFT* reconfirmed the favorable performance of our algorithms particularly in terms of energy consumption. Note that, in many previous studies (e.g., [29]), *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 $ECS{+}idle$ with even a one percent margin (Table 5) is compelling.

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, further 3.4 percent and 3.9 percent improvements (on average) in energy consumption—for schedules after the main scheduling phase of *ECS* and $ECS{+}idle$—were made by the MCER technique. Once again, these reductions are achieved in both direct and indirect energy consumption.

Since recent Intel and AMD processors allow better power management particularly for idle times (e.g., deeper C states with only a fraction of energy use compared with energy consumption of conventional idle states), the

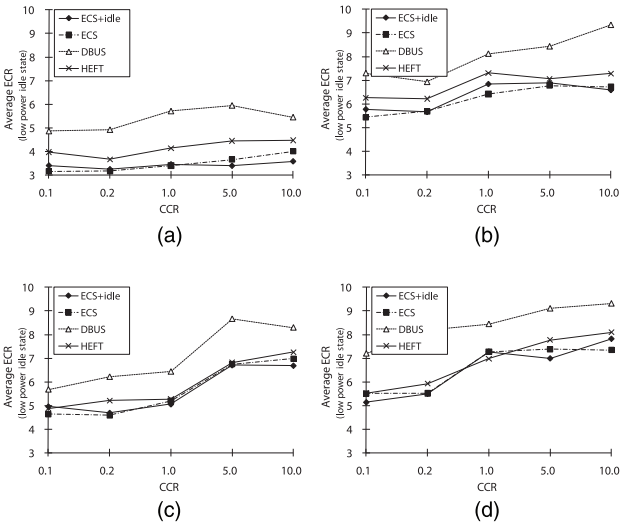

Fig. 5. Average ECRs for different types of DAGs. (a) random. (b) FFT. (c) Laplace. (d) LU.

energy saving capability of our algorithms using DVS needs to be further investigated. For this reason, we run another 72,000 simulations for each algorithm with a different energy consumption rate for idle time slots. We have set an energy consumption rate during idle times to 10 percent of energy consumption (low power idle state) used in the first set of our experiments. Results of these additional experiments (Fig. 5) mostly repeated those patterns identified in results obtained from the first set with an exception that the energy savings of *ECS* seems to be increased relative to that of $ECS{+}idle$. This improvement is primarily achieved due to the low power idle state used in the second set of experiments. Besides, the explicit consideration of energy consumption during idle time slots in $ECS{+}idle$ tends to reduce idle slots more than *ECS* does; this, therefore, contributes to such an improvement.

6 Conclusion

Since most traditional task scheduling algorithms lack the consideration of energy saving, we investigated the energy issue in task scheduling, particularly on MCSs, and presented *ECS* and $ECS{+}idle$. Our algorithms with the incorporation of RS and MCER greatly contribute to reducing energy consumption. In essence, the energy saving of *ECS* and $ECS{+}idle$ is enabled by the exploitation of the DVS technique—a recent advance in processor design. Our study provides promising results showing the significance and potential of DVS in the reduction of energy consumption. We have evaluated *ECS* and $ECS{+}idle$ with an extensive set of simulations. They were also compared with two previous algorithms. The experimental results from our comparative evaluation study confirm the superior performance of *ECS* and $ECS{+}idle$ over the other two, particularly in energy saving.

Acknowledgments

This work is supported by an Australian Research Grant DP1097110.

REFERENCES

[1] Y.C. Lee and A.Y. Zomaya, *Scheduling in Grid Environments, Handbook of Parallel Computing: Models, Algorithms and Applications,* S. Rajasekaran and J. Reif, eds, pp. 21.1-21.19. CRC Press, 2008.

[2] V. Venkatachalam and M. Franz, "Power Reduction Techniques for Microprocessor Systems," *ACM Computing Survey,* vol. 37, no. 3, pp. 195-237, 2005.

[3] Y. Tian, J. Boangoat, E. Ekici, and F. Ozguner, "Real-Time Task Mapping and Scheduling for Collaborative In-Network Processing in DVS-Enabled Wireless Sensor Networks," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS '06),* 2006.

[4] K.H. Kim, R. Buyya, and J. Kim, "Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-Enabled Clusters," *Proc. Seventh IEEE Int'l Symp. Cluster Computing and the Grid (CCGrid '07),* May 2007.

[5] D.P. Bunde, "Power-Aware Scheduling for Makespan and Flow," *Proc. 18th Ann. ACM Symp. Parallelism in Algorithms and Architectures,* July 2006.

[6] D. Zhu, R. Melhem, and B.R. Childers, "Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Realtime Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 14, no. 7, pp. 686-700, July 2003.

[7] R. Ge, X. Feng, and K.W. Cameron, "Performance-Constrained Distributed DVS Scheduling for Scientific Applications on Power-Aware Clusters," *Proc. ACM/IEEE Conf. Supercomputing (SC '05),* pp. 34-44, Nov. 2005.

[8] X. Zhong and C.-Z. Xu, "Energy-Aware Modeling and Scheduling for Dynamic Voltage Scaling with Statistical Realtime Guarantee," *IEEE Trans. Computers,* vol. 56, no. 3, pp. 358-372, Mar. 2007.

[9] S. Darbha and D.P. Agrawal, "Optimal Scheduling Algorithm for Distributed-Memory Machines," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 1, pp. 87-95, Jan. 1998.

[10] A.Y. Zomaya, C. Ward, and B.S. Macey, "Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 8, pp. 795-812, Aug. 1999.

[11] H. Topcuouglu, S. Hariri, and M.-Y. Wu, "Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing," *IEEE Trans. Parallel and Distributed Systems,* vol. 13, no. 3, pp. 260-274, Mar. 2002.

[12] Y.C. Lee and A.Y. Zomaya, "A Novel State Transition Method for Metaheuristic-Based Scheduling in Heterogeneous Computing Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 19, no. 9, pp. 1215-1223, Sept. 2008.

[13] B. Rountree, D.K. Lowenthal, S. Funk, V.W. Freeh, B.R. de Supinski, and M. Schulz, "Bounding Energy Consumption in Large-Scale MPI Programs," *Proc. ACM/IEEE Conf. Supercomputing,* Nov. 2007.

[14] J.G. Koomey, "Estimating Total Power Consumption by Servers in the U.S. and the World," 2009.

[15] G. Koch, "Discovering Multi-Core: Extending the Benefits of Moore's Law," *Technology@Intel Magazine,* http://www.intel.com/technology/magazine/computing/multi-core-0705.pdf, July 2005.

[16] Y.C. Lee and A.Y. Zomaya, "Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling," *Proc. Int'l Symp. Cluster Computing and the Grid (CCGRID '09),* pp. 92-99, May 2009.

[17] Intel, *Intel Pentium M Processor Datasheet,* 2004.

[18] R. Min, T. Furrer, and A. Chandrakasan, "Dynamic Voltage Scaling Techniques for Distributed Microsensor Networks," *Proc. IEEE Workshop Very Large Scale Integration,* pp. 43-46, Apr. 2000.

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

[20] 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. (IPDPS '05),* Apr. 2005.

[21] J.J. Chen and T.W. Kuo, "Multiprocessor Energy-Efficient Scheduling for RealTime Tasks with Different Power Characteristics," *Proc. Int'l Conf. Parallel Processing (ICPP '05),* pp. 13-20, 2005.

[22] K. Choi, R. Soma, and M. Pedram, "Fine-Grained Dynamic Voltage and Frequency Scaling for Precise Energy and Performance Tradeoff Based on the Ratio of Off-Chip Access to On-Chip Computation Times," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* vol. 24, no. 1, pp. 18-28, Jan. 2005.

[23] A. Weissel and F. Bellosa, "Process Cruise Control," *Proc. Conf. Compilers, Architecture and Synthesis for Embedded Systems,* pp. 238-246, 2002.

[24] A. Miyoshi, C. Lefurgy, E.V. Hensbergen, R. Rajamony, and R. Rajkumar, "Critical Power Slope: Understanding the Runtime Effects of Frequency Scaling," *Proc. 16th Int'l Conf. Supercomputing,* pp. 35-44, 2002.

[25] D. Zhu, D. Mosse, and R. Melhem, "Power-Aware Scheduling for AND/OR Graphs in Realtime Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 9, pp. 849-864, Sept. 2004.

[26] 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.

[27] 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, Jan. 1983.

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

[29] S.C. Kim, S. Lee, and J. Hahm, "Push-Pull: Deterministic Search-Based DAG Scheduling for Heterogeneous Cluster Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 11, pp. 1489-1502, Nov. 2007.

Young Choon Lee received the BSc (hons) degree in computer science in 2003 and the PhD degree from the School of Information Technologies at the University of Sydney in 2008. He is currently with the Center 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 is currently chair professor of High Performance Computing and Networking and an Australian Research Council Professorial Fellow in the School of Information Technologies, The University of Sydney. He is also the director for the newly established Sydney University Center 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/coauthor of seven books, more than 350 publications in technical journals and conferences, the editor of eight books, and eight conference volumes. He is currently an associate editor for 19 journals, the founding editor of the *Wiley Book Series on Parallel and Distributed Computing* and a founding coeditor of the *Wiley Book Series on Bioinformatics*. He was the chair of *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. He 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 (United kingdom), and a distinguished engineer of the ACM. His research interests are in the areas of high performance computing, parallel algorithms, mobile computing, and bioinformatics.