This paper appeared in the *Proceedings of the 8th IEEE International Conference on Cluster Computing* (Cluster 2006), Sept. 2006.

Energy-Aware Duplication Strategies for Scheduling Precedence-Constrained Parallel Tasks on Clusters

Ziliang Zong, Adam Manzanares, Brian Stinar, and Xiao Qin*
Department of Computer Science, New Mexico Institute of Mining and Technology
zzong @ nmt.edu, {amanzana, bstinar, xqin}@cs.nmt.edu

Abstract

Optimizing energy consumption has become a major concern in designing economical clusters. Scheduling precedence-constrained parallel tasks on clusters is challenging because of high communication overhead. Although duplication-based strategies are applied to minimize communication overhead, most of them merely consider schedule lengths, completely ignoring energy consumption of clusters. In this regard, we propose two energy-aware duplication scheduling algorithms, called EADUS and TEBUS, to schedule precedence-constrained parallel tasks. Unlike existing duplication-based scheduling algorithms that replicate all possible predecessors of each task, the proposed algorithms judiciously replicate predecessors only if the duplication can help in conserving energy. Our energy-aware scheduling strategies are conducive to balancing the scheduling length and energy consumption of precedence-constrained parallel tasks. Extensive experimental results based on real-world applications demonstrate the effectiveness and practicality of the proposed scheduling strategies.

1. Introduction

With the advances in VLSI technology and high-speed networks, cluster systems have become very poplar for a diverse set of parallel applications like molecular design, weather modelling, database systems, and complex image rendering. Energy demands of cluster computing systems have been growing rapidly. For example, a large-scale cluster commonly requires 40TWh per year, costing over $4B per year at the price of $100 per MWh [6]. Designing economically attractive and environmentally friendly clusters is highly challenging, because we have to take into account multiple design objectives, including performance (throughput, execution time), energy efficiency, quality-of-service (QOS), and scalability.

Although much attention has been paid to processor and memory energy conservation in clusters, saving energy in cluster interconnects for parallel applications remains an open problem. Reducing energy dissipation in cluster interconnects is of critical importance as a significant amount of the total energy consumption in a cluster is due to the interconnect fabric. For example, it is observed that interconnect consumes 33% of the total energy in an Avici switch [8] [12], whereas routers and links consume 37 percent of the total power budget in a Mellanox server blade [12]. The energy consumption in interconnects becomes more critical for communication-intensive parallel applications, which extensively make use of cluster interconnects to transfer data. Examples of this type of parallel applications include 3-D perspective rendering, molecular dynamics simulation, and 2-D fluid flow using the vortex method [7]. Lack of energy conservation technology for cluster interconnects becomes a severe problem because, without it, reducing the energy consumption of cluster systems is most unlikely.

Task partitioning and scheduling strategies play an important role in achieving high performance for parallel applications on clusters. A partitioning algorithm can be employed to partition a parallel application into a set of precedence-constrained tasks represented in the form of a directed acyclic graph (DAG), whereas a scheduling algorithm can be used to schedule the DAG onto the computational nodes of a cluster. Duplication heuristics proved to be feasible strategies to schedule parallel tasks to minimize communication overhead. However, almost all the duplication-based scheduling algorithms only take schedule lengths into consideration. One of the most important factors, energy consumption, was completely ignored. To remedy this deficiency, in this paper we design two energy-aware duplication scheduling algorithms to schedule a set of precedence-constrained parallel tasks in a judicious way to improve performance (shorten schedule length) while optimizing energy consumption in cluster interconnects.

In this research, we first build an energy consumption model used to estimate power dissipation

* Corresponding author. *http://www.cs.nmt.edu/~xqin*

in CPUs and network links. Second, we proposed two duplication-based scheduling algorithms, called EADUS and TEBUS, to provide energy savings in network links by duplicating tasks on more than one computational node to reduce network traffic. EADUS is designed to aggressively provide the greatest energy savings by using task replicas to eliminate energy-consuming messages, whereas TEBUS aims at making the best tradeoff between energy conservation and performance. Hence, the two algorithms are named Energy-Aware Duplication Scheduling algorithm (or EADUS) and Time-Energy Balanced Duplication Scheduling algorithm (or TEBUS). Finally, to demonstrate the effectiveness of the proposed scheduling strategies, we designed and implemented a simulated cluster computing system with Myrinet-style [11] cluster interconnects and compare our approaches against existing duplication-based scheduling scheme using real-world applications.

The rest of the paper is organized as follows. In section 2, we present a brief description of related work. Next, Section 3 introduces an energy consumption model. The novel scheduling strategies are proposed in Section 4. In Section 5, the experimental environment and simulation results are analysed. Finally, the concluding remarks are provided in Section 6.

2. Related Work

Since energy demands of clusters have been steadily growing in the last five years, the issue of energy conservation has increasingly received much attention. A large body of research has been conducted into power-aware scheduling [1] [2] [14] [24] [26]. For example, dynamic power management [5] [4] [17] and variable voltage scheduling schemes [15] [18] were proposed to achieve minimum energy consumption. Yao et al. developed a static off-line scheduling algorithm [25], whereas Hong et al. proposed on-line heuristics scheduling for aperiodic tasks [15]. Shin and Choi proposed a scheme to slow down a processor when there is a single task eligible for execution [22]. Very recently, we proposed a task allocation strategy, which can minimize overall energy consumption while confining schedule lengths to an ideal range [24]. We also studied a power-aware message scheduling algorithm in the context of real-time wireless networks [1]. Most of the prior work in the area of energy-aware scheduling has focused on energy consumed by processors. Growing evidence indicates that the interconnection fabric is one of the most energy-consuming components in clusters [8] [19]. In this study, we aim at developing energy conservation techniques for both processors and interconnect of a cluster. As such, our approaches are in sharp contrast to the existing scheduling algorithms for processor energy conservation.

Scheduling strategies deployed in clusters have a large impact on overall system performance. Three extensive categories for scheduling schemes include priority based scheduling, cluster based scheduling, and task duplication based scheduling [3][21]. Duplication based scheduling strategies address the problem that inter-processor communication in parallel and distributed systems accounts for a major portion of total system overhead. Many researchers have demonstrated that various strategies regarding task duplication are extremely applicable for reducing total execution time within a system employing static scheduling [3][21][9][10]. However, most of the existing duplication-based scheduling algorithms are developed with consideration of schedule lengths, completely ignoring energy consumption of clusters. Our algorithms are fundamentally different from existing duplication-based scheduling approaches in that ours are the first two duplication-based scheduling strategies designed to conserve energy consumption in clusters.

3. Mathematical Models

In this section, we build mathematical models used to represent a cluster system, precedence-constrained parallel tasks, and energy consumption in CPUs and interconnects.

3.1 System model

A cluster system in this study is characterized by a set $P = \{p_1, p_2, ..., p_m\}$ of computational nodes connected by a Myrinet-style cluster interconnects. It is assumed that the computational nodes are homogeneous in nature, meaning that all the computing nodes are identical in their capabilities. Similarly, the underlying interconnection is assumed to be homogeneous and, thus, communication overhead of a message with fixed data size between any pair of nodes is considered to be the same. In our system model, computation and communication can take place simultaneously. This assumption is reasonable because each computational node in a modern cluster has a communication coprocessor that can be used to free the processor of the node from communication tasks [16].

To simplify the system model without loss of generality, we assume that the cluster system is fault

free and the page fault service time of each task is integrated into its execution time. With respect to energy conservation, energy consumption rate of each node in the system is measured by Joule per unit time. Each interconnection link is modelled by its energy consumption rate that heavily relies on data size and the transmission rate of the link.

3.2 The task model

A parallel application with a set of precedence-constrained tasks is represented in form of a Directed Acyclic Graph (DAG), which throughout this paper is modeled as a pair (V, E). $V = \{v_1, v_2, ..., v_n\}$ represents a set of precedence-constrained parallel tasks, and t_i is the ith task's computation requirement showing the number of time units to compute v_i, $1 \le i \le n$. It is assumed that all the tasks in V are nonpreemptive and indivisible work units, and a similar assumption can be found in related studies [9] [20]. $(v_i, v_j) \in E$ is a message transmitted from task v_i to v_j, and c_{ij} is the communication cost of the message $(v_i, v_j) \in E$. We assume in this study that there is one entry task and one exit task for an application with a set of tasks. The assumption is reasonable because in case multiple entry or exit tasks exist, the multiple tasks can always be connected through a dummy task with zero computation cost and zero communication cost messages.

The communication-to-computation ratio or CCR of a parallel application is defined as the ratio between the average communication cost of the application and the average computation cost on a given cluster. Formally, the CCR of an application (V, E) is given by Eq. (1):

$$CCR(V,E) = \frac{\frac{1}{|E|}\sum_{(v_i,v_j)\in E} c_{ij}}{\frac{1}{|V|}\sum_{i=1}^{|V|} t_i} \tag{1}$$

A task allocation matrix (e.g., X) is an n×m binary matrix reflecting a mapping of n precedence-constrained parallel tasks to m computational nodes in a cluster. Element x_{ij} in X is "1" if task v_i is assigned to node p_j and is "0", otherwise.

3.3 Energy consumption model

We use a bottom-up approach to derive energy dissipation experienced by a parallel application running on a cluster. In this subsection, we first model energy consumption exhibited by computational nodes in the cluster. Next, we calculate energy dissipation in the interconnection network of the cluster.

Let en_i be the energy consumption caused by task v_i running on a computational node, of which the energy consumption rate is PN_{active}, and the energy dissipation of task v_i can be expressed as Eq. (2)

$$en_i = PN_{active} \times t_i \tag{2}$$

Given a parallel application with a task set *V* and allocation matrix *X*, we can calculate the total energy consumed (TECN) by all the tasks in *V* using Eq. (3).

$$\begin{aligned} EN_{active} &= \sum_{i=1}^{|V|} en_i = \sum_{i=1}^{n} \left(PN_{active} \cdot t_i\right) \\ &= PN_{active} \sum_{i=1}^{n} t_i . \end{aligned} \tag{3}$$

We denote el_{ij} as the energy consumed by the transmission of message $(t_i, t_j) \in E$. We can compute the energy consumption of the message as below

$$el_{ij} = PL_{active} \times c_{ij} \tag{4}$$

where PL_{active} is the power of the link when it is active. The cluster interconnect in this study is homogeneous, which implies that all messages are transmitted over the interconnection network at the same transmission rate. The energy consumed by a network link between p_a and p_b is a cumulative energy consumption caused by all messages transmitted over the link. Then, the link's energy consumption is obtained as follows, where L_{ab} is a set of messages delivered on the link, and L_{ab} can be expressed as

$$L_{ab} = \left\{ \forall (v_i, v_j) \in E, 1 \le a, b \le m \,\middle|\, x_{iu} = 1 \wedge x_{jv} = 1 \right\}$$

$$\begin{aligned} EL_{active}^{ab} &= \sum_{(v_i,v_j)\in L_{ab}} el_{ij} = \sum_{(v_i,v_j)\in L_{ab}} \left(PL_{active} \cdot c_{ij}\right) \\ &= \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \left(x_{ia} \cdot x_{jb} \cdot PL_{active} \cdot c_{ij}\right), \end{aligned} \tag{5}$$

The energy consumption of the whole interconnection network is derived from Eq. (5) as the summation of all the links' energy consumption. Thus, we have

$$EL_{active} = \sum_{a=1}^{m} \sum_{b=1, b\neq a}^{m} EL_{active}^{ab}$$

$$= \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} \sum_{a=1}^{m} \sum_{b=1, b\neq a}^{m} \left(x_{ia} \cdot x_{jb} \cdot PL_{active} \cdot c_{ij} \right). \quad (6)$$

It is observed from our experiments that energy consumption caused by idle computational nodes (PN_{idle}) and interconnection network (EL_{idle}) is negligible and, therefore, here we ignore the definition of energy consumption model by idle resources.

Thus, we can compute the energy dissipation experienced by the parallel application on the cluster using Eqs. (3) and (6). Hence, we can express the total energy consumption of the cluster executing the application as

$$E \approx EN_{active} + EL_{active} \quad (7)$$

4. Energy-Aware Duplication Strategies

In this section, we present two energy-aware duplication strategies, called EADUS and TEBUS. The objective of the two scheduling strategies is to optimize energy consumption of clusters. The scheduling problem studied in this paper can be shown to be NP-hard by mapping it to the scheduling problem proven to be an NP-complete [13]. Therefore, the proposed two scheduling algorithms are heuristic in the sense that they can produce suboptimal solutions in polynomial-time. The EADUS and TEBUS algorithms consist of three major steps delineated in Sections 4.1-4.3.

4.1 Generate a task sequence

Precedence-constraints of a set of parallel tasks have to be guaranteed by executing predecessor tasks before successor tasks. To achieve this goal, the first step in our algorithms is to construct an ordered task sequence using the concept of level, which of each task is defined as the length in computation time of the longest path from the task to the exit task. In this study, we use a similar approach as proposed by Srinivasan and Jha [27] to define the level $L(v_i)$ of task v_i as below

$$L(v_i) = \begin{cases} t_i, & \text{if } \forall 1 \le j \le n: (t_i, t_j) \notin E \\ t_i + \max\limits_{(v_i, v_j) \in E} \left(L(v_j) + c_{ij} \right), & \text{otherwise} \end{cases} \quad (8)$$

The levels of other tasks can be obtained in a bottom-up fashion by specifying the level of the exit task as its execution time and then recursively applying the second term on the right side of Eq. (8) to calculate the levels of all the other tasks. Next, all the tasks are placed in a queue in the decreasing order of levels.

4.2 Calculate important parameters

The second phase in the EADUS and TEBUS algorithms is to calculate some important parameters, which the algorithms rely on. The important notation and parameters are listed in Table 1.

Table 1. Important notations and parameters

Notation	Definition
$EST(v_i)$	Earliest start time of task v_i
$ECT(v_i)$	Earliest completion time of task v_i
$FP(v_i)$	Favorite predecessor of task v_i
$LACT(v_i)$	Latest allowable completion time of task v_i
$LAST(v_i)$	Latest allowable start time of task v_i

Note that similar notation was used by Darbha and Agrawal in [9]. The earliest start time of the entry task is 0. The earliest start times of all the other tasks can be calculated in a top-down manner by recursively applying the second term on the right side of Eq. (9).

$$EST(v_i) = \begin{cases} 0, & \text{if } \forall 1 \le j \le n: (v_j, v_i) \notin E \\ \min\limits_{(v_j, v_i) \in E} \left(\max\limits_{(v_k, v_i) \in E, v_k \neq v_j} \left(ECT(v_j), ECT(v_k) + c_{ki} \right) \right), & \text{otherwise} \end{cases} \quad (9)$$

The earliest completion time of task v_i is expressed as the summation of its earliest start time and execution time. Thus, we have

$$ECT(v_i) = EST(v_i) + t_i. \quad (10)$$

Allocating task v_i and its favorite predecessor $FP(v_i)$ on the same computational node can lead to a shorter schedule length. As such, the favourite processor $FP(v_i)$ is defined as Eq. (11)

$$FP(v_i) = v_j, j \neq k \mid ECT(v_j) + c_{ji} \ge ECT(v_k) + c_{ki}. \quad (11)$$

As shown by the first term on the right side of Eq. (12), the latest allowable completion time of the exit task equals to its earliest completion time. The latest allowable completion times $LACT(v_i)$ of all the other tasks are calculated in a top-down manner by recursively applying the following Expression.

$$\begin{cases} ECT(v_i), & \text{if } \forall 1 \le j \le n: (v_i, v_j) \notin E \\ \min\left(\min\limits_{(v_i, v_j) \in E, v_i \neq FP(v_j)} \left(LAST(v_j) - c_{ij} \right), \min\limits_{(v_i, v_j) \in E, v_i = FP(v_j)} \left(LAST(v_j) \right) \right), & \\ \text{otherwise} & \end{cases} \quad (12)$$

The latest allowable start time of task v_i is derived from its latest allowable completion time and execution time. Hence, the $LAST(v_i)$ can be written as

$$LAST(v_i) = LACT(v_i) - t_i. \quad (13)$$

1. v = first waiting task of scheduling queue;
2. i = 0; assign v to P_i;
3. **while** (not all tasks are allocated) **do**
4. u = FP(v);
5. **if** (u has already been assigned to another processor) **then**
6. **if** (LAST(v) - LACT(u) ≥ c_{vu}) **then** /* if duplicate u, we can shorten the execution time*/
7. moreenergy = en_u - el_{vu}; /*energy increase*/
8. **if** (moreenergy ≤ threshold h) **then** /* increased energy less than our threshold*/
9. assign u to P_i; /*duplicate u*/
10. **if** v has another predecessor $z \neq u$ has not yet been allocated to any node **then**
11. $u = z$;
12. **else**
13. **if** u is entry task **then**
14. u = the next task that has not yet been assigned to a node;
15. i++;
16. **else**
17. for another predecessor z of v, $z \neq u$,
18. **if** (ECT(u)+cc_{uv} = ECT(z) + cc_{zv}) and z hasn't been allocated) **then**
19. $u = z$; /* do not duplicate*/
20. **else**
21. for another predecessor z of x, z≠ u,
22. **if** (ECT(u)+cc_{uv} = ECT(z) + cc_{zv}) and z hasn't been allocated) **then**
23. $u = z$; /* do not duplicate*/
24. **else** allocate u to P_i;
25. v = u;
26. **if** v is entry task **then**
27. v = the next task that has not yet been allocated to a computational node;
28. i++;
29. assign v to P_i;
30. **return** schedule list, schedule length and energy consumption;

Fig. 1. Pseudocode of phase 3 in the EADUS algorithm

4.3 Duplication phase

4.3.1 The EADUS algorithm. Given a parallel application presented in form of DAG, the EADUS algorithm in this phase allocates each parallel task to a computational node in a way to aggressively shorten the schedule length of the DAG while conserving energy consumption. The pseudocode in Fig. 1. shows the details of this phase in the EADUS algorithm. Most existing duplication-based scheduling schemes merely optimize schedule lengths without addressing the issue of energy conservation. As such, the existing duplication-based approaches tend to yield minimized schedule lengths at the cost of energy consumption. To make the best trade-off between energy-saving and schedule lengths, we design the EADUS algorithm in which a task-duplication is strictly forbidden if the duplication does not exhibit energy conservation (see Steps 8-9). In other words, duplication is infeasible if it results in a significant increase in energy consumption (e.g., the increase exceeds a threshold) and, is avoided in EADUS.

Before this phase starts, phase 1 sorts all the tasks in a waiting queue, followed by phase 2 to calculate the important parameters. In phase 3 EADUS strives to group communication-intensive parallel tasks together and have them allocated to the same computational node. Once multiple task groups are constructed, each group of tasks is assigned to a different node in the cluster. The process of grouping tasks is repeated from the first task in the queue by performing a depth-first style search, which traces the path from the first task to the entry task. Steps 4 and 5 choose a favorite predecessor if it has not been allocated a computational node. Otherwise, EADUS may or may not replicate the favorite predecessor on the current node. For example, we assume that v_j is the favorite predecessor of the current task v_i, and v_j has been allocated to another node. If duplicating v_j on the current node to which v_i is allocated can improve performance without sacrificing energy conservation, Step 11 makes a duplication of v_j. More formally, the following property must be satisfied before any duplication is generated. Note that en_u - el_{vu} are computed by Eqs (2) and (4), respectively.

Property 1. Let v_j be the favorite predecessor of the current task v_i, and v_j has been allocated to another node. A duplication of v_j is made on v_i's current node if the following two conditions are satisfied:

- $LAST(v_i) - LACT(v_j) < c_{ji}$,
- $en_u - el_{vu}$ < threshold h, and
- $$\neg\left(\begin{array}{l}\exists(v_k, v_i) \in E, k \neq j, v_k \text{ has not been allocated:} \\ ECT(v_k) + c_{ki} = ECT(v_j) + c_{ji}\end{array}\right)$$

Here the first condition in Property 1 ensures that v_j is a critical predecessor of v_i, the second condition signifies that the increase in energy due to the duplication must be maintained at a low level. The third condition is used to identify if v_i's other unallocated predecessors

can initially be the favorite predecessors. In case that such an initial favorite predecessor (e.g., v_k) exists, the path to the entry task will be traversed through v_k.

The generation of a task group terminates once the path reaches the entry task. The next task group starts from the first unassigned task in the queue. If all the tasks are assigned to the computation nodes, then the algorithm terminates.

4.3.2 The TEBUS algorithm. The third phase of the TEBUS algorithm is similar as that of EADUS except that TEBUS seamlessly integrate the approach to minimizing schedule lengths with the process of energy optimization, which means that we will change the rule of deciding duplication. Here we define three variables called *moreenergy*, *lesstime*, *ratio* and replace the corresponding lines (see line 8-10 in Fig. 1) with the following segment,

1. moreenergy = $en_u - el_{vu}$; /* energy increase */
2. lesstime = LAST(v) – LACT(u) – c_{vu}; /* schedule length is reduced */
3. cost ratio = moreenergy/lesstime; /* calculate the value of cost ratio */
4. **if** (ratio ≤ threshold h) **then** … /* continued */

Unlike EADUS, the development of TEBUS is motivated by the needs of making the right tradeoff between performance and energy conservation. Thus, the TEBUS algorithm is geared to efficiently reduce schedule lengths while providing the greatest energy savings. Energy consumption incurred by duplicating a task involves judging whether the duplication is feasible or not. To facilitate the construction of TEBUS, we introduce a concept of cost ratio of a duplication, which is defined as the ratio between the energy saving and schedule length reduction. While the energy increasing of the duplication is obtained in Step 1, the reduction in schedule length is computed in Step 2. The TEBUS algorithm is, of course, conducive to maintaining cost ratios at a low level, thereby efficiently shortening schedule lengths with low energy consumption. This feature is accomplished by Steps 3-4, which duplicate a task in case the cost ratio of such duplication is smaller than a given threshold.

5. Performance Evaluation

To demonstratively show the strength of our novel scheduling schemes, we conducted extensive experiments using real-world applications like Gaussian elimination and Fast Fourier Transform applications. Furthermore, we compare EADUS and TEBUS with two existing scheduling algorithms: the non-duplication-based scheduling heuristic (NDS) [23], and the task duplication-based scheduling algorithm (TDS) [9].

5.1 Simulation setup

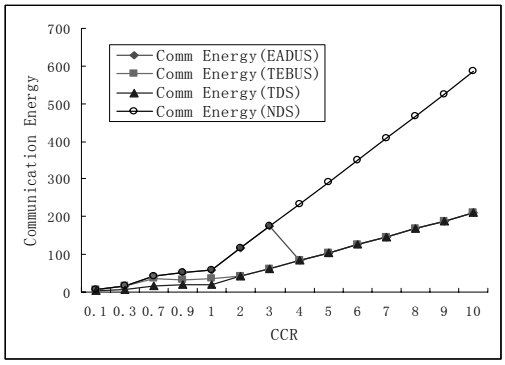
In this subsection we present the experimental setup. Table 2 summarizes the configuration parameters of simulated cluster systems used in our experiments. On the right hand side of each row in Table 2, parameters in the first part are fixed while parameters in the second part are varied or randomly generated using uniform distributions. For instance, the threshold values of EADUS and TEBUS are respectively fixed to 0.5 and 2 in one experiment and the threshold values are varied from 0.02 to 500 in another experiment (see the last row of Table 2).

Table 2. Characteristics of System Parameters

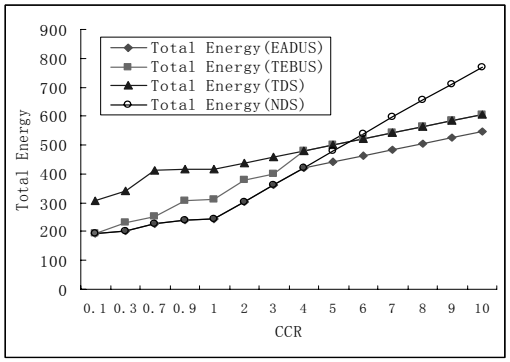
Parameters	Value (Fixed) - (Varied)
Different Trees	Gaussian elimination, Fast Fourier Transform
Execution time of Gaussian Tree	{5, 4, 1, 1, 1, 1, 10, 2, 3, 3, 3, 7, 8, 6, 6, 20, 30, 30 }-(random)
Execution time of FFT Tree	{15, 10, 10, 8, 8, 1, 1, 20, 20, 40, 40, 5, 5, 3, 3 }-(random)
Node energy consumption rate	6.0 mW
Comm_energy consumption rate	1.5 mW
CCR set	{0.1, 0.2, 0.3, 0.5, 0.7, 0.9,1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Threshold h	{0.5, 2}-{0.02, 0.1, 0.2, 0.3, 0.4, 0.5, 0.8, 1, 5, 10, 20, 30, 100, 500}

5.2 Experimental Results

To compare the performance of the EADUS and TEBUS algorithms with NDS and TDS, we apply them to allocate parallel tasks of two real-world applications, namely, the Gaussian Elimination and Fast Fourier Transform applications. We are focusing on the energy consumption for each application under various CCRs and thresholds. The experimental results for the energy consumption of the Gaussian Elimination application are shown in Fig. 2.



(a) Communication Energy Consumption

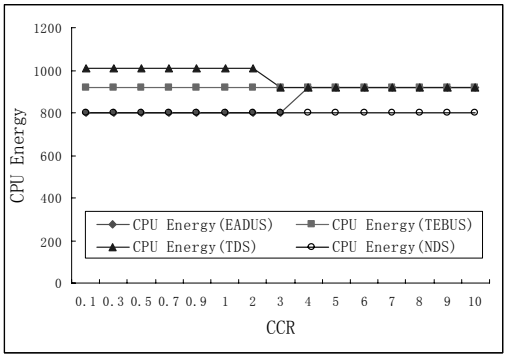


(b) Total Energy Consumption

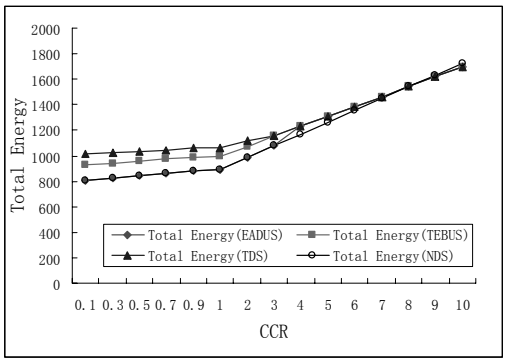
Fig. 2. CCR Sensitivity for Gaussian Elimination

Four observations are evident from this group of experiment. First, the energy consumption of Gaussian Elimination under all the four scheduling schemes is very sensitive to CCR. Second, when CCR is greater than 6, energy consumption under NDS is consistently higher than that under the other three algorithms. However, NDS provides the greatest energy savings if CCR is less than 4. This is because energy cost in the interconnection network is extremely low with a small CCR value. Third, with respect to energy conservation, EADUS performs as well as NDS with small CCRs. However, EADUS is superior to NDS when CCR is large. These results demonstrate that regardless of the CCR value, EADUS is the best energy-efficient duplication scheduling algorithm among the four examined schemes. Last, and generally speaking, the energy performance of TEBUS is somewhere between those of EADUS and TDS.

Fast Fourier Transform is a very well known algorithm used to implement a three-dimensional Fast Fourier transform. We are focused on the energy sensitivity of the Fast Fourier Transform application to CCR. Fig. 3 plots CPU and total energy consumption of Fast Fourier Transform under an array of CCR values from 0.1 to 10. Fig. 3 shows that the total energy consumption of Fast Fourier Transform becomes more sensitive to CCR when CCR is less than 1. Comparing energy consumption results plotted in Figs. 2 and 3, we observe that Fast Fourier Transform is less sensitive to CCR than Gaussian Elimination. The implication behind this observation is that Gaussian Elimination can take more energy-saving advantages of EADUS and TEBUS than Fast Fourier Transform.



(a) CPU Energy Consumption



(b) Total Energy Consumption

Fig. 3 CCR Sensitivity for Fast Fourier Transform

6. Conclusions

In this paper, EADUS and TEBUS are designed and implemented to provide energy savings in clusters by duplicating tasks on more than one computational node. We also proposed mathematical energy consumption models to facilitate the presentation of EADUS and TEBUS. We conducted extensive experiments based on real-world applications running on a simulated cluster. The experimental results show that EADUS and TEBUS significantly improve the

performance in terms of energy dissipation over two existing allocation schemes called NDS and TDS. Compared with TDS, EADUS achieves energy-performance improvement for Gaussian Elimination on average of 16.08% with only 5.7% increase in schedule length. Likewise, TEBUS improves energy conservation on average of 8.1% with merely 2.2% increase in schedule length.

Acknowledgements

The work reported in this paper was supported in part by the New Mexico Institute of Mining and Technology under Grant 103295 and by Intel Corporation under Grant 2005-04-070.

References

[1] M. Alghamdi, T. Xie, X. Qin, "PARM: A Power-Aware Message Scheduling Algorithm for Real-Time Wireless Networks," *Proc. ACM Workshop Wireless Multimedia Networking and Performance Modeling, Montreal*, Oct. 2005.

[2] H. Aydin, R. Melhem, D. Mossé, and P. M. Alvarez, "Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics," *Proc. EuroMicro Conf. Real-Time Systems*, Delft, Netherlands, June 2001.

[3] S. Bansal, P. Kumar, and K. Singh, "An Improved Duplication Strategy for Schedulng Precedence Constrained Graphs in Multiprocessor Systems," *IEEE Trans. Parallel and Distributed Systems*, Vol. 14, No. 6, pp. 533-544, June 2003.

[4] L. Benini and G. De Micheli, Dynamic Power Management: Design Techniques and CAD Tools, *Kluwer*, 1998

[5] L. Benini, A. Bogliolo, G. D. Micheli, "A Survey of Design Techniques for System-Level Dynamic Power Management," *IEEE Trans. Very Large Scale Integration Systems*, Vol. 8, No. 3, pp.299-316, June 2000.

[6] J. Chase and Ron Doyle, "Energy Management for Server Clusters," *Proc. the 8th Workshop Hot Topics in Operating Systems* (HotOS-VIII), pp. 165, May 2001

[7] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, "Architectural Requirements of Parallel Scientific Applications With Explicit Communication," *Proc. 20th Int'l Symp. Computer Architecture*, 1993

[8] W. Dally, P. Carvey, and L. Dennison, "The Avici Terabit Switch/Rounter," *Proc. Hot Interconnects 6*, pp. 41-50, Aug. 1998.

[9] S. Darbha, 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] S. Darbha and D. P. Agrawal, "A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memory Systems", *J. Parallel and Distributed Computing*, Vol. 46, No. 1, pp. 15-27, Oct. 1997.

[11] C. Dubnicki, A. Bilas, Y. Chen, S.N. Damianakis, and K. Li, "Myrinet Communication," *IEEE Micro*, Vol. 18, No. 1, pp. 50-52, Jan.-Feb. 1998.

[12] E.N. M. Elnozahy, M. Kistler, and R. Rajamony, "Energy-Efficient Server Clusters," *Proc. Int'l Workshop Power-Aware Computer Systems*, Feb. 2002.

[13] R.L. Graham, L.E. Lawler, J.K. Lenstra, and A.H. Kan, "Optimizing and Approximation in Deterministic Sequencing and Scheduling: A Survey," Annals of Discrete Math, pp.287-326, 1979.

[14] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. Srivastava, "Power Optimization of Variable Voltage Core-Based Systems," *Proc. Design Automation Conf.*, 1998.

[15] I. Hong, M. Potkonjak, and M. Srivastava, "Synthesis Techniques for Low-Power Hard Real-Time Systems on Variable Voltage Processors," *Proc. IEEE Real-Time System Symp.*, Dec. 1998.

[16] J. Kuskin et al., "The Stanford FLASH Multiprocessor," *Proc. 21st Int'l Symp. Computer Architecture*, 1994.

[17] J. Lorch and A. Smith, "Software Strategies for Portable Computer Energy Management," *IEEE Personal Commun.*, Vol. 5, pp.60-73, June 1998.

[18] J. R. Lorch and A. J. Smith, "Improving Dynamic Voltage Scaling Algorithm with PACE," *Proc. ACM SIGMETRICS Conf.*, Cambridge, MA, June 2001

[19] Mellanox Technologies Inc., "Mellanox Performance, Price, Power, Volumn Metric (PPPV)," 2004.

[20] X. Qin and H. Jiang, "A Dynamic and Reliability-driven Scheduling Algorithm for Parallel Real-time Jobs on Heterogeneous Clusters," *J. Parallel and Distributed Computing*, Vol. 65, No. 8, pp.885-900, Aug. 2005.

[21] S. Ranaweera, and D.P. Agrawal, "A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems," *Proc. Parallel and Distributed Processing Symp.*, pp.445-450, May 2000.

[22] Y. Shin and K. Choi, "Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems," *Proc. Design Automation Conf.*, 1999.

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

[24] T. Xie, X. Qin, and M. Nijim, "Solving Energy-Latency Dilemma: Task Allocation for Parallel Applications in Heterogeneous Embedded Systems," *Proc. 35th Int'l Conf. Parallel Processing*, Columbus, Ohio, Aug. 2006.

[25] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," *Proc. IEEE Annual foundations of Computer Science*, pp. 374-382, 1995.

[26] Y. Yu and V.K. Prasanna, "Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks," Mobile Networks and Applications, Vol.10, No.1-2, pp.115-131, Feb. 2005.

[27] S. Srinivasan and N.K. Jha, "Safety and Reliability Driven Task Allocation in Distributed Systems," *IEEE Trans. Parallel and Distributed Systems*, Vol. 10, No. 3, pp. 238-251, March 1999.