Energy-Efficient Scheduling for Parallel Applications Running on Heterogeneous Clusters

Ziliang Zong[†], Xiao Qin[†*], Xiaojun Ruan[†], Kiranmai Bellam[†],
Mais Nijim[‡], and Mohamed Alghamdi[§]
[†]*Department of Computer Science and Software Engineering*
Auburn University, Auburn, AL 36849
{ zzong, xqin, xruan, kbellam}@ eng.auburn.edu
[‡]*Department of Computer Science*
University of Southern Mississippi, Hattiesburg, MS 39406
mnijim@usm.edu
[§]*Department of Computer Science*
New Mexico Institute of Mining and Technology, Socorro, NM 87801

Abstract

High performance clusters have been widely used to provide amazing computing capability for both commercial and scientific applications. However, huge power consumption has prevented the further application of large-scale clusters. Designing energy-efficient scheduling algorithms for parallel applications running on clusters, especially on the high performance heterogeneous clusters, is highly desirable. In this regard, we propose a novel scheduling strategy called energy efficient task duplication schedule (EETDS for short), which can significantly conserve power by judiciously shrinking communication energy cost when allocating parallel tasks to heterogeneous computing nodes. We present the preliminary simulation results for Gaussian and FFT parallel task models to prove the efficiency of our algorithm.

1. Introduction

The combination of modern clusters and parallel computing technology exhibits powerful computing capabilities. Over the last decade, the rapid advancement of high performance microprocessors, high-speed networks, and standard middleware tools makes cluster computing platforms more powerful and convenient to use. Therefore, cluster computing technology has been extensively deployed and widely used to solve challenging and rigorous engineering problems in industry and scientific areas like molecular design, weather modeling, universe dark matter observations, and complex image rendering. However, the rapid growth of cluster computing centers introduces a serious problem: huge energy consumption. According to EUN (Energy User News) [1], the power requirements of today's cluster computing centers range from 75 W/ft2 to 150-200 W/ft2 and will increase to 200-300 W/ft2 in the nearest future. The new data center capacity projected for 2005 would require approximately 40TWh ($4B at $100 per MWh) per year to run 24x7 unless they become more efficient [2]. New data centers in the Seattle area are forecast to increase the city's power demands by 25% [3]. Therefore, it is highly desirable to design energy-aware scheduling algorithms for cluster systems.

However, designing energy-aware scheduling algorithms for heterogeneous clusters, is technically challenging because we have to take into account multiple design objectives, including performance (measured by throughput and schedule length), energy efficiency, and heterogeneities. Basically, processors, networks, disks, and cooling system are four major power consumers in a cluster computing system. Our approach aims to conserve network transmission energy by judiciously duplicating communication-intensive tasks. More specifically, we first assume the execution and communication times of tasks are already known in priori and apply a heuristic (a similar

[*] Corresponding author. http://www.eng.auburn.edu/~xqin

approach can be found in [5]) way to minimize schedule lengths by grouping the most related parallel tasks together. Next, we make use of a dynamic allocation method to obtain a suboptimal power consumption of a cluster computing system by comparing total energy consumption when grouped tasks are allocated to different computational nodes in the cluster.

The rest of the paper is organized as follows. In section 2, we will talk about the related work. Next, Section 3 defines mathematical models including a system model, a task model, and an energy consumption model for heterogeneous clusters. In Section 4 we present the detailed two-phases EETDS algorithm. Section 5 qualitatively compares EETDS with three existing approaches through extensive simulation results. Finally, section 6 provides the concluding remarks and future research directions.

2. Related Work

The issue of conserving energy consumption in high performance clusters has not attracted enough attention for a long period because researchers mainly concentrate on the performance, reliability, and security issues of clusters [5]. Recently, people start to realize that the energy consumption issue is also critical since energy demands of modern data centers have been steadily growing in a noticeable speed.

A handful of previous studies investigated energy-aware processor and memory design techniques to reduce energy consumption in processors and memory resources [6][7][8]. IBM researchers Elnozahy, Kistler, and Rajamony proposed the Request Batching Policy (RBP), in which servicing of incoming requests is delayed while a web server is kept in a low power state. Incoming requests are accumulated in memory until a request has been kept pending for longer than a specified batching timeout. RBP can save energy because while requests are being accumulated, the processor is placed in a lower power state such as deep sleep [9]. Dynamic power management is designed to achieve requested performance with minimum number of active components or a minimum load on such components [9][10]. Dynamic power management consists of a collection of energy-efficient techniques, which adaptively turn off system components or reduce their performance when the components are idle or partially unexploited. For example, based on the observation of past idle and busy periods, predictive shutdown policies can make power management decisions when a new idle period starts [12][13]. Shin and Choi proposed a scheme to slow down a processor when there is a single task eligible for execution [14]. Yao et al. developed a static off-line scheduling algorithm [15], whereas Hong et al. proposed on-line heuristics scheduling for aperiodic tasks [16]. Very recently, we developed a task allocation strategy aiming to minimize overall energy consumption while confining schedule lengths to an ideal range [17].

However, most of these prior works regarding energy-aware scheduling were mainly focused on energy consumed by processors and memories, which are not appropriate for parallel applications with intensive communication because the communication energy consumption was completely ignored. Many literatures have shown that energy dissipation caused by communication between is huge in large-scale distributed computing systems. For instance, network consumes 33 percent of the total energy in an Avici switch [18][19], and routers and links consume 37 percent of the total power budget in a Mellanox server blade [20]. The energy consumption in interconnects becomes even more critical for communication-intensive parallel applications, in which large number of data will be transferred among parallel tasks. Enlightened by duplication strategy, which has been proved to be a good solution to reduce communication overhead, we proposed two algorithms for homogeneous distributed system [21]. In this paper, we designed a two-phase energy-aware scheduling algorithm specifically for heterogeneous distributed parallel system. The algorithm, called EETDS, combines group-based scheduling with duplication-based scheduling strategies together and makes good tradeoffs between performance and energy savings.

3. System Architecture & Energy Model

3.1. System Architecture

Generally, a high performance heterogeneous cluster system can be divided into three main components, which are hardware components, network protocols and software components. More specifically, hardware components include PCs, workstations, high performance computers and underneath network equipments (e.g. routers, switches and network interface cards). High-quality network protocols are critical for heterogeneous system since the computing nodes communicate with each other by different type of networks. Therefore, the communication protocol should provide support for various networks. In addition, a standard message passing protocol (provide message passing scheme and manage messages) is definitely necessary. In order to support parallel applications, a special parallel computing layer should be designed to support parallel computing. Fig.1 shows the architecture of a heterogeneous cluster system.

If we define the heterogeneous cluster in a mathematical way, it consists of a set $P = \{p_1, sp_2,...,p_m\}$ of heterogeneous computing nodes (hereinafter referred to as nodes) connected by a high-speed interconnect like fast Ethernet, gigabit Ethernet, SCI, FDDI or Myrinet. All these loosely coupled nodes communicate with each other by passing messages. The whole platform can be represented by a graph, where computing nodes are vertices. There exists a weighted edge if a pair of corresponding nodes can communicate with each other. An $n \times m$ binary allocation matrix X is used to reflect a mapping of n tasks to m heterogeneous nodes. Thus, element x_{ij} in X is "1" if task t_i is assigned to node p_j and is "0", otherwise. Since our scheduling algorithm will be verified in a heterogeneous environment, it is imperative to define the following constraints for our heterogeneous distributed system. First, different nodes have different preference with respect to tasks, meaning that a node offering task t_i a shorter execution time does not necessarily run faster for another task t_j. Second, execution times of tasks on different nodes may various because the nodes may have various clock speed and processing capabilities. Third, the transmission rates of interconnections depend on underlying network types. Last, energy consumption rates of the nodes and interconnections may not necessarily be identical.

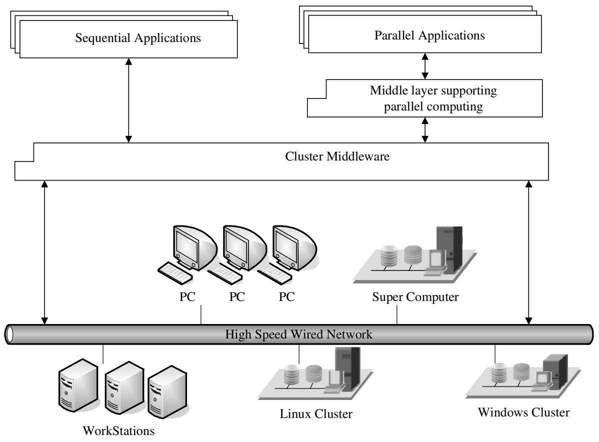

Fig.1 Architecture of Heterogeneous Clusters

3.2 Energy Consumption Model

If we do not consider the disk subsystem and cooling system, the energy consumption model of a heterogeneous cluster can be expressed as:

$$E = EN + EL \tag{1}$$

Where EN is the energy consumption of computing nodes, and EL is the energy consumption of communication. The energy consumption EN of computing node in Eq. (1) can be written as:

$$\begin{aligned} EN &= \sum_{i=1}^{|V|} en_i = \sum_{i=1}^{n} \left(PN \cdot t_i\right) \\ &= PN \sum_{i=1}^{n} t_i . \end{aligned} \tag{2}$$

where en_i is the energy consumption caused by node i, PN is the power consumption rate of the node, and t_i is the total execution time of tasks running on node i. The energy consumed by the interconnections is expressed as

$$\begin{aligned} EL &= \sum_{a=1}^{m} \sum_{b=1,b\neq a}^{m} EL^{ab} \\ &= \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \sum_{a=1}^{m} \sum_{b=1,b\neq a}^{m} \left(x_{ia} \cdot x_{jb} \cdot PL \cdot c_{ij}\right). \end{aligned} \tag{3}$$

where EL^{ab} is the energy consumption of the link between nodes p_a and p_b. PL in Eq. (2) is the power consumption rate of the link. Element x_{ia} is "1" if the ith task is assigned to node p_a and is "0", otherwise. c_{ij} is the communication cost.

4. EETDS Algorithm

In this section, we present the two-phases EETDS algorithm in detail. Basically, given parallel applications in form of DAGs, the objective of our scheduling algorithm is to allocate and schedule parallel tasks to computing nodes in a way to shorten schedule lengths while reducing energy consumption of heterogeneous clusters. The scheduling algorithm studied in this paper has been shown to be NP-hard problem by mapping it to a scheduling problem proven to be an NP-complete [22]. Therefore, the proposed scheduling algorithm is heuristic in the sense that it can produce suboptimal solutions in polynomial-times.

4.1 Grouping Phase

The grouping phase of EETDS is to associate the most relevant tasks (i.e. tasks in the same critical paths) into groups. Given a parallel application modeled as a task graph or DAG, the grouping phase yields a group-based graph of the DAG. Since all tasks in a group are allocated to the same computing node where there are no waiting times among the tasks within the group, we can significantly reduce communication overheads of highly dependent tasks with intensive communications. Additionally, a task-duplication strategy is applied in the process of grouping to further improve system performance by replicating tasks into multiple computing nodes if schedule lengths can be shortened. More specifically, the grouping phase can be finely divided into three sub-steps, namely, original task sequence generating, parameters calculating, and duplication task sequence generating. These sub-steps are detailed as follows.

4.1.1 Original task sequence generating

The first step in the grouping phase is to construct an original ordered task sequence using the concept of levels, which define how far it is from current task to the completion time of the exit task. Note that a similar approach proposed by Srinivasan can be found in [5]. We define the level *L(t$_i$)* of task t$_i$ as below

$$L(t_i)=\begin{cases} c_i, & \text{if successor}(t_i)=\Phi \\ \max\big(\underbrace{level(k)}_{k\in succ(i)}\big)+c_i, & \text{otherwise}\end{cases} \tag{4}$$

The level of an exit task is equal to its execution time. 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-hand side of Eq. (4) to calculate the levels of all the other tasks. Initially, all ungrouped tasks are marked NOT_GROUPED. The list of groups is initialized to an empty set. Next, all the tasks are sorted in an increasing order of the levels and then considered for grouping in that order.

Table 1. Important Notation and Parameters

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$_i$*

4.1.2 Parameters calculating

The second step in the grouping phase is to calculate some important parameters, which will be used in the third step (see Section 4.1.3) to generate duplication-based task sequences. The important notation and parameters are summarized in Table 1. Similar notations were first proposed by Darbha and Agrawal in 1997 [23].

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 as follows.

$$EST(v_i)=\begin{cases} 0, & \text{if predecessor(i)}=\Phi \\ \min\limits_{e_{ji}\in E}\left(\max\limits_{e_{ki}\in E, v_k\neq v_j}\left(ECT(v_j), ECT(v_k)+c_{ki}\right)\right), & \text{otherwise}\end{cases}$$

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

$$ECT(v_i)=EST(v_i)+t_i.$$

The favorite predecessor FP(v$_i$), *LACT(v$_i$), LAST(v$_i$)* are defined as below respectively

$$FP(v_i)=v_j, \text{where} \forall e_{ji}\in E, e_{ki}\in E, j\neq k \left| ECT(v_j)+c_{ji}\geq ECT(v_k)+c_{ki}\right..$$

$$LACT(v_i)=\begin{cases} ECT(v_i), & \text{if successor(i)}=\Phi \\ \min\left(\min\limits_{e_{ij}\in E, v_i\neq FP(v_j)}\left(LAST(v_j)-c_{ij}\right), \min\limits_{e_{ij}\in E, v_i=FP(v_j)}\left(LAST(v_j)\right)\right), & \text{otherwise}\end{cases}$$

$$LAST(v_i)=LACT(v_i)-t_i.$$

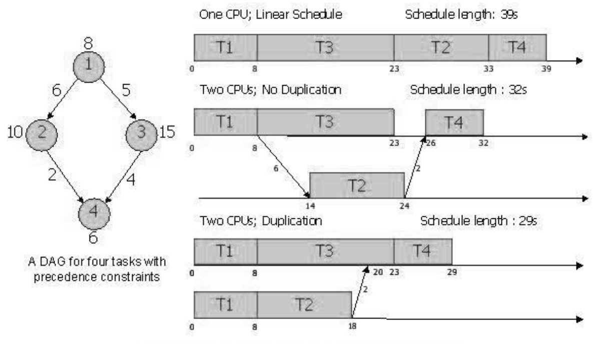

Fig. 2. An example of duplication scheduling strategy

4.1.3 Duplication task sequence generating

Once the original task sequence and important parameters are available, we are ready to apply the duplication strategy to complete the last step of the grouping phase. Fig. 2 illustrates the main idea of the duplication strategy using a simple example. The left part of Fig. 2 shows a DAG for four tasks with precedence constraints. The execution times of task T1, T2, T3, T4 are 8s, 10s, 15s, and 6s. The communication times among the tasks are 6s, 5s, 2s, and 4s, respectively. The right part of Fig. 2 provides three schedules made by the linear scheduling strategy, the non-duplication strategy, and the duplication strategy, respectively. The linear schedule has the longest schedule length because all the tasks allocated to one computing nodes have to be executed in a sequential order. The non-duplication schedule reduces the schedule length by allowing T2 and T3 running in parallel on two computing nodes. The duplication schedule further improves the performance by redundantly executing T1 on the second node. Thus, the duplication strategy trades CPU times for communication overheads.

At the very beginning of duplication process, no task is marked as "grouped" and the list of clusters is initialized to be empty. Next, EETDS will consider the first task and add it to a newly formed group called G1. Then in the following iterations, the algorithm goes through all the tasks along the favorite predecessor chain and attempts to add all the tasks in the critical path to the same group. Once a task is added to a group, it will be immediately marked as "grouped". If the task being processed is the entry task, the current iteration will end and a new iteration will start in the next loop by choosing the first ungrouped task from the original task sequence generated in step 1. During the process of walking through multiple critical paths, we may find some tasks have been added to a group. At this point, the duplication strategy is responsible to make the decision whether or not duplicate this task to multiple groups by calculating the schedule length. If duplicating a task can reduce schedule length, EETDS will go further step to check energy consumption caused by duplication. If the extra power cost is more than the energy threshold we have set, duplication process will be forbidden automatically. Therefore, any task will not be duplicated in either of the two cases: increase schedule length or consume too much power. At the end of the process, the task graph has been divided into groups. Finally, the group graph is generated by creating edges among all the groups communicating with each other. The algorithm then sets a weight for each edge to represent corresponding communication cost.

4.2 Energy-Efficient Group Allocation

After the grouping stage, the DAG has been partitioned into a number of groups, which will be allocated to heterogeneous computing nodes by the next step in an energy-efficient way. The main objective for this phase is to generate an allocation list with minimized energy dissipation. More specifically, the algorithm calculates energy consumption caused by the group running on the node. The estimation of the energy consumption can be carried out using the energy consumption model described in Section 3.2. The value of this energy consumption is saved in an energy cost array. The algorithm applies the same procedure to the next type of node. This procedure is repeatedly performed until all candidate nodes with respect to the group have been considered. Finally, the algorithm updates the allocation list with a node that leads to the minimized energy dissipation. After the group allocation phase is accomplished, the allocation list provides an allocation solution with minimized energy consumption of the heterogeneous cluster.

5. Performance Evaluation

Now we are positioned to evaluate the effectiveness of the proposed EETDS scheduling algorithm. To demonstratively show the strength of our novel scheduling scheme, we conducted extensive experiments using real-world applications including Gaussian Elimination and Fast Fourier Transform applications. In this section, we compare EETDS with two existing scheduling algorithms: the Non-Duplication Scheduling algorithm (NDS) [24] and the Task Duplication Scheduling algorithm (TDS) [25]. In addition, we compare EETDSS with a baseline algorithm: Energy-Efficient Non-Duplication Scheduling (EENDS). In order to reflect different applications, Communication-to-Computation Ratio (or CCR for short), is defined as the ratio between the average communication cost of |E| messages and the average computation cost of n parallel tasks with m heterogeneous computing nodes. Formally, the CCR of an application (T, E) is expressed by equation as below.

$$CCR(T,E) = \frac{\frac{1}{|E|}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{1}{m(m-1)}\sum_{u=1}^{m}\sum_{v=1,v\neq u}^{m}\frac{s_{ij}}{b_{uv}}\right)}{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{m}\sum_{j=1}^{m}c_i^j\right)} = \frac{\frac{1}{|E|\cdot(m-1)}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\sum_{u=1}^{m}\sum_{v=1,v\neq u}^{m}\frac{s_{ij}}{b_{uv}}\right)}{\frac{1}{n}\sum_{i=1}^{n}\left(\sum_{j=1}^{m}c_i^j\right)}$$

Generally speaking, communication-intensive applications have high CCRs, whereas CCRs of computation-intensive applications are low.

5.1 Simulation Setup

To simulate a heterogeneous distributed system running parallel tasks, we set three different environments using four different types of processors as follows.

AMD Athlon 64 X2 4600+ with 85W TDP (Type 1)

Energy Consumption Rate: Busy: 104w Idle: 15w
AMD Athlon 64 X2 4600+ with 65W TDP (Type 2)
Energy Consumption Rate: Busy: 75w Idle: 14w
AMD Athlon 64 X2 3800+ with 35W TDP (Type 3)
Energy Consumption Rate: Busy: 47w Idle: 11w
Intel Core 2 Duo E6300 processor (Type 4)
Energy Consumption Rate: Busy: 44w Idle: 26w

Environment1	Environment2	Environment3
# of Type 1: 4	# of Type 1: 6	# of Type 1: 6
# of Type 2: 4	# of Type 2: 2	# of Type 2: 3
# of Type 3: 4	# of Type 3: 2	# of Type 3: 3
# of Type 4: 4	# of Type 4: 6	# of Type 4: 5

Also, we simulate three networks with different energy consumption rate of (20W, 33.6W, 60W).

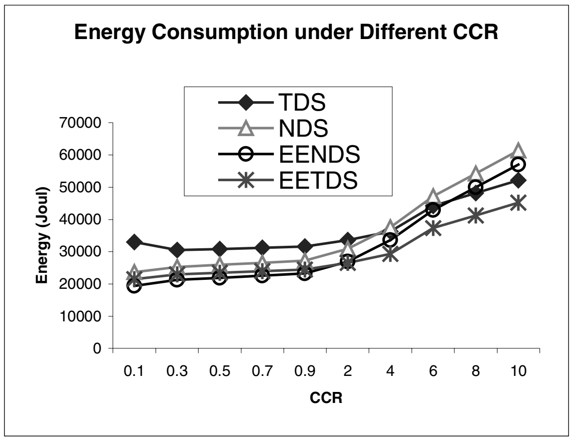

Fig.4 CCR Sensitivity of Gaussian Elimination (E1)

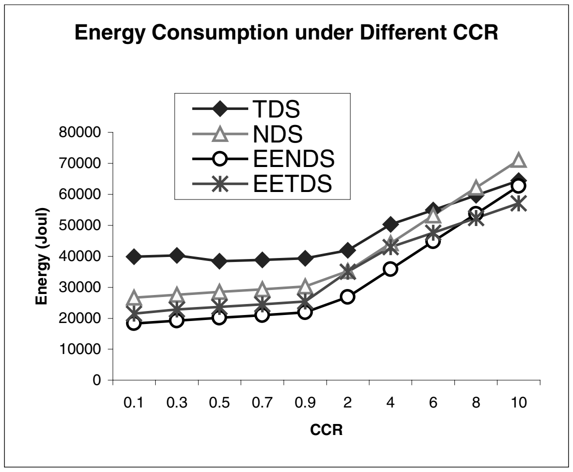

Fig.5 CCR Sensitivity of FFT (E2)

5.2 CCR Sensitivity

Figs. 4 and 5 show the impacts of CCR on energy dissipation of Gaussian Elimination application and FFT respectively. Four observations are evident from this group of experiments. First, the energy consumption of Gaussian Elimination under all the four algorithms is very sensitive to CCR. Second, NDS outperforms TDS with respect to energy conservation when the CCR values are small. However, TDS is superior to NDS when CCR becomes large (e.g., CCR is greater than or equal to 4). Third, EETDS works well for both Gaussian and FFT and it has overall better performance compared with the other three algorithms. Last, the energy savings exhibited by EETDS become more pronounced with the increasing values of CCR, which indicates that EETDS is more appropriate for communication-intensive applications as opposed to computation-intensive applications.

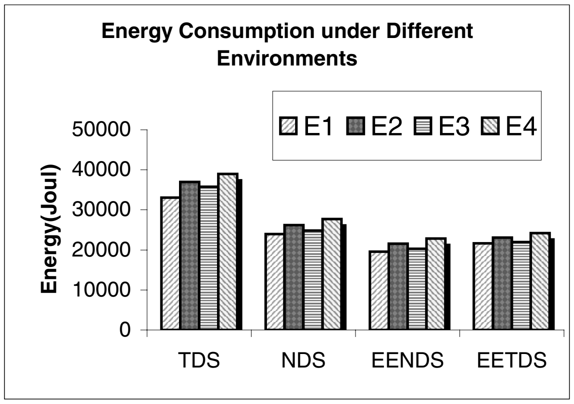

Fig.6 Energy consumption for Gaussian when Net_Energy=60 and CCR=0.1

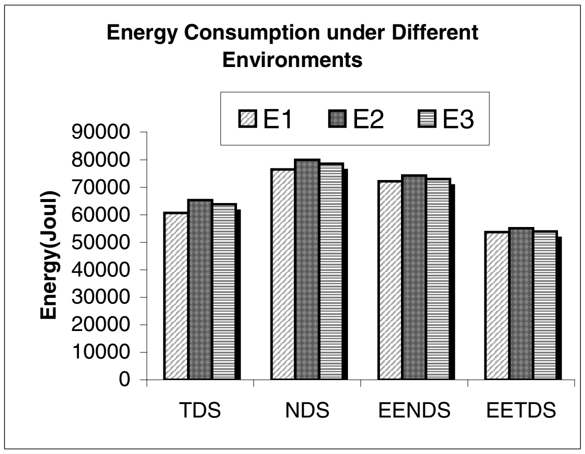

Fig.7 Energy consumption for Gaussian when Net_Energy=60 and CCR=8

5.3 Computing Nodes Heterogeneity

Figs. 6 and 7 illustrate the impacts of the computing heterogeneity. First, we observe that EETDS can conserve more energy in E1 and E3 compared with E2, from which we can draw the conclusion that less energy is consumed with more energy-efficient computing nodes. Second, the computing heterogeneity has significant impacts on the energy efficiency of EETDS. For example, when CCR equals to 0.1 in three different environments, EETDS reduces energy consumption (compared with TDS) by 36.4%, 47.1%, and 45.6%, respectively. These experimental results indicate that EETDS can conserve even more energy for heterogeneous clusters that are comprised of energy-consuming computing nodes. Third, Fig. 6&7 show a similar performance trend that EETDS significantly conserve energy in overall because TDS consumes huge energy when CCR is small and NDS consumes more energy when CCR is large due to the huge energy dissipation in the network interconnections.

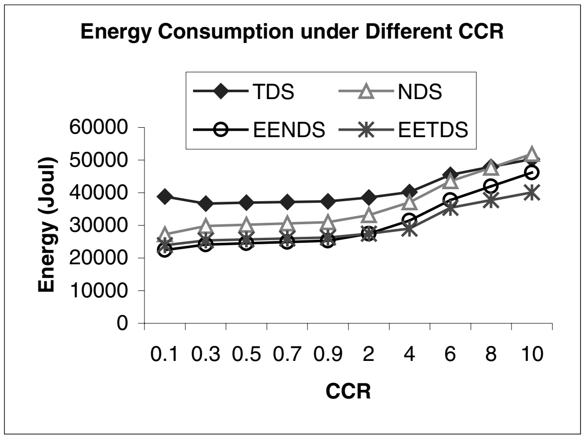

Fig.8 Energy consumption of Gaussian (Net = 20W)

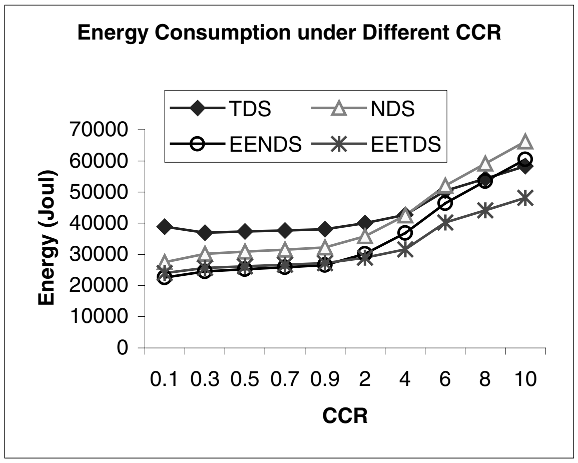

Fig.9 Energy consumption of Gaussian (Net = 33.6W)

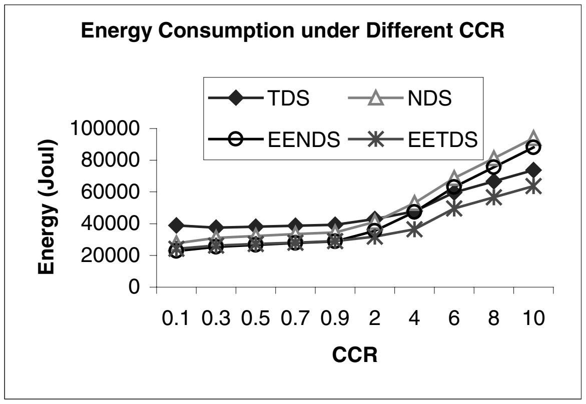

Fig.10 Energy consumption of Gaussian (Net = 60W)

5.4 Network Heterogeneity

After comparing Figs. 8, 9, and 10, we can quantify the impacts of network heterogeneity. For instance, given environment 1 for Gaussian application, EETDS can improve energy efficiency over TDS by 27.9%, 27.9%, 27.8% when network energy consumption rate is 20W, 33.6W, and 60W, respectively (CCR is set to 0.1). However, when CCR is large (e.g., 10), these improvements in energy efficiency are scaled down to 15.6%, 13.3% and 10.2%, respectively. In this set of experiments we confirm that the network energy consumption contributes a whole lot to total energy consumption when CCR is large. Last, we conclude that NDS is not suitable for communication-intensive parallel applications because NDS has schedule lengths significantly increased when communication overheads are high.

6. Conclusions

In this paper, we proposed a two-phase energy-efficient scheduling algorithm called EETDS, which aims to make the best tradeoffs between energy savings and performance for parallel applications running on heterogeneous clusters. EETDS is designed and implemented based on the duplication-based algorithm used to minimize communication overheads of parallel tasks with precedence constraints. The EETDS algorithm consists of two major phases. In the first phase, a grouping method is employed to minimize schedule lengths of parallel applications. The goal of phase two is to leverage power-consumption parameters to conserve energy. The experimental results show that compared with three existing algorithms, EETDS can significantly reduce energy dissipation for heterogeneous cluster system with only a marginal degradation in performance.

Acknowledgement

The work reported in this paper was supported by the US National Science Foundation under Grant No. CCF-0702781, Auburn University under a startup grant, New Mexico Institute of Mining and Technology under Grant No. 103295, the Intel Corporation under Grant No. 2005-04-070, and the Altera Corporation under an equipment grant.

References

[1] B. Moore. Taking the data centre power and cooling challenge. *Energy User News*, Aug. 2002.

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

[3] Robert Bryce. Power struggle. Interactive Week, December 2000. http://www.zdnet.com/intweek/, found under stories/news/0,4164,2666038,00.html.

[4] S. Darbha and D. P. Agrawal, "A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memory Systems", *J. Parallel and Distr. Comp.*, vol. 46, no. 1, pp. 15-27, Oct. 1997.

[5] 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.

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

[7] J. Rabaey and M. Pedram (Editors), Lower Power Design Methodologies, *Kluwer Academic Publisher*, Norwell, MA, 1998.

[8] A. Raghunathan, N. K. Jha, and S. Dey, High-Level Power Analysis and Optimization, *Kluwer Academic Publisher*, Norwell, MA, 1998.

[9] E. Elnozahy, M. Kistler, and R. Rajamony, "Energy-efficient server clusters," *Proc. Workshop Power-Aware Computing Systems*, Feb. 2002.

[10] 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.

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

[12] F. Douglis, P. Krishnan, B. Bershad, "Adaptive Disk Spin-down Policies for Mobile Computers," *USENIX Symp. Mobile and Location-Independent Computing*, pp. 121-137, 1995.

[13] M. Srivastava, A. Chandrakasan. R. Brodersen, "Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation," *IEEE Trans. VLSI Systems*, vol. 4, no. 1, pp. 42-55, March 1996.

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

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

[16] I. Hong, M. Potkonjak, and M. Srivastava, "On-line Scheduling of Hard Real-Time Tasks on Variable Voltage Processor," *Proc. Computer Aided Design*, pp. 653-656, 1998.

[17] 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.

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

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

[20] Mellanox Technologies Inc., "Mellanox Performance, Price, Power, Volumn Metric (PPPV)," http://www.mellanox.co/products/shared/PPPV.pdf, 2004.

[21] Z.-L. Zong, A. Manzanares, B. Stinar, and X. Qin, "Energy-Efficient Duplication Strategies for Scheduling Precedence Constrained Parallel Tasks on Clusters," *Proc. Int'l Conf. Cluster Computing* (Cluster'06), Sept. 2006.

[22] 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 Disc. Math*, pp.287-326, 1979.

[23] S. Darbha and D. P. Agrawal, "A Task Duplication Based Scalable Scheduling Algorithm for Distributed Memory Systems", *J. Parallel and Distr. Comp.*, vol. 46, no. 1, pp. 15-27, Oct. 1997.

[24] 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.

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