International Journal of Grid Distribution Computing
Vol. 8, No.5, (2015), pp.205-214
http://dx.doi.org/10.14257/ijgdc.2015.8.5.20

Energy Aware Scheduling based on Two-phase Frequency Scaling for Parallel Tasks in Cluster

Aihua Liang, Jun Liang and Jiazheng Yuan

Beijing Key Laboratory of Information Service Engineering, Beijing Union University, Beijing, China
ldtliangah@163.com

Abstract

Improving the energy efficiency of high performance clusters has become important research issue. We proposed a new algorithm that reduces energy consumption of precedence constrained parallel tasks in power-scalable clusters. To reduce energy consumption without increasing the schedule length, our algorithm reclaims both static and dynamic slack time and employs different frequency adjusting techniques in different slack time. The optimal frequency is obtained through analyzing the precedence constraints of parallel tasks. We conducted extensive experiments to compare the proposed algorithm with two other existing algorithms, simulation results show that the proposed algorithm can get better energy efficiency without increasing the make span.

Keywords*: energy aware, cluster, make span, DVFS, DAG*

1. Introduction

As the performance of modern processor has increased, however, high energy consumption of high performance computer system has become an important and urgent problem [1]. Nowadays, performance and energy efficiency are two key criterions of modern clusters. Designing energy efficient and environmental friendly clusters is highly desirable. Many recent high performance microprocessors are equipped with the Dynamic Voltage and Frequency Scaling (DVFS) technique, which allows processors to be operated at multiple frequencies under different supply voltages at run time, thereby saving energy by spreading run cycles into idle time. Parallel applications can be represented as a Directed Acyclic Graph (DAG), called a task graph, where nodes denote the tasks with precedence constraints and the edges denote the communications between tasks. A parallel program may have some slack time due to their precedence constraints and synchronizations between the tasks.

Our research is devoted to developing the scheduling algorithm which reduces energy consumption of parallel task execution by using the DVFS mechanism at slack time. We identify the slack time of tasks in different stages and scale their supply voltages to the corresponding level thus reducing the jobs energy consumption.

The rest of the paper is organized as follows: in Section 2, related work has been described. Section 3 introduces computational models including the task model and the energy consumption model. In Section 4, we present the two-phase frequency scaling strategy. Simulation results are demonstrated in Section 5. Finally, Section 6 provides the concluding remarks and future research directions.

2. Related Work

Increasing attention has been directed toward energy efficiency research for high performance clusters [2–3]. Among many energy saving techniques, scheduling is an efficient approach to reducing energy consumption on clusters. Nowadays, there has been a lot of research on energy efficient scheduling based on DVFS [4–11].

ISSN: 2005-4262 IJGDC

DVFS is a run-time power reduction technique, which has been proven to be a feasible solution to reduce processor power consumption [4]. Tasks are run at reduced voltages and clock frequencies to fill idle periods and reduce energy consumption, while providing required performance. DVFS fills the slack time by elongating computation time.

Some work applies DVFS during the communication phases of high performance computing, for example MPI [5-6]. Kimura et al. [7] adopted DVFS at slack time of tasks. They also designed a toolkit called Power Watch that can monitor the power and provide the control library. Ruan et al. [8] proposed an energy-efficient scheduling algorithm, named TDVAS, for clusters. Dynamic Voltage Scaling (DVS) technique is employed to parallel tasks followed by idle processor times to conserve energy consumption without increase schedule lengths of parallel applications. Zhu et al. [9] presented an adaptive energy-efficient scheduling for aperiodic and independent real time tasks on heterogeneous clusters with DVS, which can adjust voltage levels according to the workload variation. Wang et al. [10] considered the Green Service Level Agreement and studied the slack time for non-critical jobs, extends their execution time using DVFS. Kim et al. [11] proposed the DVS scheduling algorithms for time-shared and space-shared resource sharing policies.

DVFS exploits low frequency and voltage at the slack time. Slack time includes the static slack time of non-critical tasks and dynamic slack time due to task communication and synchronization. Existing research mainly focused on the static slack time. In this paper, we not only pay attention to static slack time, but also consider the dynamic slack time. Aiming at improving energy efficiency from two respects, our scheduling method adopted different frequency scaling strategies in different stages.

3. Computational Model

3.1. Task Model

Subsection text here. Parallel application with precedence constrained tasks can be represented as DAG with weight. In this paper, a parallel application G is modeled as a vector *(V, E, C, T)*, where $V= \{v_1, v_2, \ldots, v_n\}$ represents a set of parallel tasks, and $E = \{e_{ij} = (v_i, v_j) | 1 \le i, j \le n\}$ denotes a set of communication messages among parallel tasks. In all tasks, v_1 denotes the entry node and v_n is the exit node. C is the set of edge communication costs and T is the set of computation costs. The value $t_i \in T$ is defined as the required computing time of v_i. $C_{ij} \in C$ is defined as the communication cost incurred at the edge e_{ij}.

A task is non-preemptive and indivisible work unit, which may be an assignment statement, a subroutine or even an entire program. For each edge, if v_i and v_j are allocated to the same processor, the communication cost e_{ij} is set to zero. $PRED(v_i)$ is the set of immediate predecessors of v_i and $SUCC(v_i)$ is the set of immediate successors of v_i. A task allocation matrix, named *U*, is a $n \times m$ binary matrix denoting a mapping of n parallel tasks to m computational nodes in a cluster. Element u_{ij} in *U* is "1" if task v_i is assigned to processor v_j and "0", otherwise.

3.2. Energy Model

A cluster is represented as a set $P = \{p_1, p_2, \ldots, p_m\}$, where p_i denotes processor i. Energy consumption model is represented as the sum of processors energy and interconnections energy. Let EP be the energy consumption caused by processors. The energy consumption caused by interconnections is denoted as EC. So the total energy consumption of task set can be represented as

$$EN = EP + EC \quad (1)$$

Let *Rp* be the energy consumption rate of a processor. The execution time of task *i* is denoted as t_i. The energy consumption rate R_p can be expressed as

$$R_p = C \times V^2 \times f \quad (2)$$

Where *C* is the capacity of the circuit, v is the supply voltage, and f is the frequency. The capacity *C* is a constant parameter when processor is working. Energy consumption of a processor is written as a product of energy consumption rate *Rp* and execution time of tasks. Thus, we have

$$EP = \sum_{i=1}^{n} R_p t_i = \sum_{i=1}^{n} (\lambda \times f^3) t_i \quad (3)$$

To calculate the energy consumption of interconnects, let el_{ij} be energy consumed by the transmission of message e_{ij}. The energy consumption of the message can be computed as a product of communicating rate R_c and the communication cost c_{ij} as

$$el_{ij} = k \cdot R_c \cdot C(e_{ij}) \quad (4)$$

k is the constant parameter. The energy consumption of a network link is a cumulative energy consumption caused by all messages delivered over the link. The communication energy EC can be expressed as below.

$$EC = \sum_{i=1}^{n} \sum_{j=1}^{n} el_{ij} = \sum_{i=1}^{n} \sum_{j=1}^{n} k \cdot R_c \cdot c_{ij} = k \cdot R_c \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \quad (5)$$

Because our research focuses on the DVFS technique, processor energy consumption would be reduced during the tasks execution due to processor frequency adjustment.

4. Two-Phase Frequency Scaling

The key idea of DVFS is to dynamically scale the supply voltage and frequency of CPU while meeting total computation time. The parallel application has the slack time due to the precedence constraints and synchronization or communication of tasks.

The slack time can be classified into two categories [12]: Worst Slack Time (WST) and Workload-variation Slack Time (WVST). WST belongs to static slack time, which results from low processor utilization due to precedence constraints between tasks. It can be computed before task execution. WVST occurs due to execution time variation caused by data dependent computation. It is dynamically generated. Thus, it can be known only after execution.

A novel energy aware scheduling algorithm (called NEASA) based on DVFS technique is proposed in this section. According to two kinds of slack time, our energy aware scheduling method includes two stages.

(1) In the first stage, the optimum frequency of processor is calculated based on the DAG of parallel application before tasks execution. In the WST, processor frequency would be reduced to optimum frequency. The make span of overall parallel tasks would not be affected in adjusted frequencies.

(2) In the second stage, after every task finished, the execution time of the task is checked and confirms that if the real execution time is less than the original t_i. Then the WVST would be calculated in the run time. In WVST, the minimum allowed frequency would be adopted to reduce the energy consumption at the most extent. The objective of the scheduling is to reduce the overall energy consumption through adopting DVFS technique in both WST and WVST according to the critical path of task graph and dynamic execution variation at run time.

Table 1. Important Notations and Parameters

Notation	Description
$EST(v_i)$	The earliest start time of v_i
$ECT(v_i)$	The earliest complete time of v_i
$LAST(v_i)$	The latest allowed start time of v_i
$LACT(v_i)$	The latest allowed complete time of v_i
$RCT(v_i)$	The real complete time of v_i
$WST(v_i)$	Worst slack time time of v_i
$WVST(v_i)$	Workload-variation slack time of v_i
$PRED(v_i)$	The immediate predecessors of v_i
$SUCC(v_i)$	The immediate successors of v_i

4.1. First Phase Frequency Scaling

Parallel task scheduling should consist of grouping and allocating. Grouping is to divide the tasks of a DAG into several groups. Allocating refers to a mapping the groups on the processor. The tasks of the same group will execute on the same processor. In the grouping and allocating of proposed algorithm, the liner clustering method is employed. The key issue is the frequency setting during execution.

The important parameters are listed in Table 1. The similar notations of some parameters are used by Zong in [13]. Top Level and bottom level of v_i can be calculated based on the vector *(V, E, C, T)* of DAG.

EST of an entry task is defined as 0. The EST of all the other tasks can be calculated in a top-down manner by recursively applying the following term on the right side.

$$EST(v_i) = \begin{cases} 0 & i = 1 \\ \max\limits_{1 \le j \le n-1, e_{ji} \in E} (EST(v_j) + C(e_{ji})) & 1 \le i \le n \end{cases} \quad (6)$$

ECT of all tasks can be calculated as the summation of its EST and execution time.

$$EST(v_i) = EST(v_i) + T(v_i) \quad (7)$$

Latest Allowable Completion time (LACT): the latest allowable completion time of task can be calculated in a top-down manner by applying the following term.

$$LACT(v_i) = \begin{cases} \mathrm{ECT}(v_i) & i = n \\ \min\limits_{1 \le j \le n-1, e_{ji} \in E} (LACT(v_j) - C(e_{ij})) & 1 \le i \le n-1 \end{cases} \quad (8)$$

Latest Allowable Start Time (LAST): the latest allowable start time of task can be derived from LACT of task.

$$LAST(v_i) = LACT(v_i) - T(v_i) \quad (9)$$

The worst slack time of task v_i can be calculated using the following Equation.

$$WST(v_i) = LACT(v_i) - ECT(v_i) \quad (10)$$

The critical path is longest path form an entry node to an exit node. It can decide the scheduling make span. If LACT (v_i) = ECT (v_i), then v_i is the critical task. All the critical tasks can form the critical path. Since the ECT and LACT of critical tasks are equal, there is no slack time to scale the processor frequency. Thus, only non-critical tasks would be adopted the dynamic frequency scaling. The shortest execution time of task i can be calculated from the following equation.

$$\min_{v_i} = t_i = ECT(v_i) - EST(v_i) \quad (11)$$

Task v_i will attain the shortest execution time if the processor of its computing node runs with the highest frequency. In contrast, the longest execution time of task v_i can be calculated from the following equation.

$$\max_{v_i} = \min_{v_i} + WST(v_i) = LACT(v_i) - EST(v_i) \quad (12)$$

If the task v_i finished within the longest execution time, the makespan would not be affected. Therefore, the optimal frequency of non-critical tasks on the fly can be derived from the aforementioned parameters using the following equation.

$$f_{optimum} = \frac{\min_{v_i}}{\max_{v_i}} f_{\max} \quad (13)$$

4.2. Second Phase Frequency Scaling

Optimal frequency in WST can be calculated before the parallel tasks start. However, WV ST dynamically exists and is unknown before the tasks running. Only a task finished, the WV ST can be calculated.

$$WVST_{v_i} = LACT(v_i) - RCT(v_i) \quad (14)$$

Due to some factors, the real execution time of a task is not always t_i of DAG. Therefore, RCT is not equal to LACT. When RCT is less than LACT, the task completed ahead of schedule and has slack time. WVST can be calculated according to following equation. The processor frequency of computing node in WVST would be scaled to lowest. When RCT is greater than LACT, the task has no slack time and may extend the successor tasks. To minimize the task subsequent task execution time delay, processor frequency of SUCC (v_i) would be scaled to f_{max}.

4.3. Energy Aware Scheduling Algorithm

The proposed energy aware scheduling algorithm NEASA differentiates the static slack time and dynamic slack time. There are two phases: Firstly, the important parameters are calculated based on DAG. The optimal frequency is derived from the important parameters. Secondly, at run time, once a task finished, the dynamic workload variation slack time can be calculated. If WVST is greater than zero, the lowest allowable frequency would be adopted. Otherwise, the maximum frequency would be adopted by the successor tasks. The algorithm description is given as Figure 1.

The WST and WVST are calculated through traversing all tasks. The Implementation complexity is proportional to the number of tasks. That is O(n).

5. Experimental Evaluation

The proposed dynamic clustering algorithm is realized in C++, in which interconnect adopted gigabit connection and CPU frequency is 2.4GHz. The impact of RCT and overall energy efficiency are evaluated. The standard task sets [14] are adopted as Table 2 showed.

Table 2. Standard Task Sets Used

Parallel application	characteristics	Task number
Random	Integrated	150
Robot	Communication intensive	88
SPEC fpppp	Computation intensive	334

5.1. Impact of ECR and LCR

ECR (Early Complete Rate) and LCR (Late Complete Rate) are defined. Since ETF algorithm does not consider slack time. We define the energy and makespan of

ETF is 1. The proposed algorithm (NEASA) is compared with TDVAS from reduced energy and makes span in random task set.

Algorithm NEASA

Input:
Vector: (V, E, C, T)

Output:
Makespan, Energy

```
1: //define array Flag for n tasks
2: int Flag[n]={0};
3: //Compute the optimal frequency for each non-critical task in WST
4: For each task v_i ∈ V
5:    Calculate the EST, ECT, LAST, LACT;
6:    Calculate the WST;
7:    Calculate the optimal frequency for v_i;
8: End for
9: //Compute the WVST for each task and scale the frequency at run time
10: For each task v of scheduling queue
11:    Assigned v to P_i;
12:    Scale the frequency of p_i to optimal value;
13: End for
14: Calculate Energy;
15: While (not all tasks finished)
16:    If Task v_i has finished then
17:       If Flag[i]=1 then
18:          Scale the frequency to highest value;
19:          Update Energy;
20:       Else
21:          Calculate the WVST(v_i);
22:          //if there exists WVST(v_i)
23:          If (WVST(v_i) > 0) then
24:             Scale the frequency to lowest value;
25:             Update Energy;
26:          Else
27:             j=SUCC(v_i);
28:             Flag[j]=1;
29:          End if
30:       End if
31:    End if
32: End while
33: Calculate the Makespan;
34: Return Makespan, Energy;
```

Figure 1. Algorithm Description

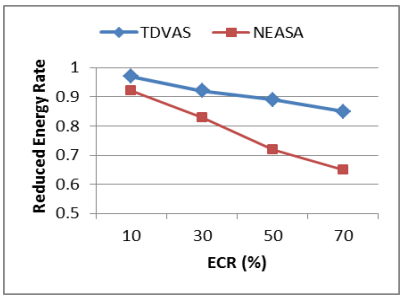

Figure 2. Reduced Energy Rate trend in different ECR

Figure 2 shows the reduced energy rate change trend when ECR is 10%, 30%, 50%, 70% respectively. Both TDVAS and NEASA reduced energy consumption. NEASA has more advantages over TDVAS. As ECR increases, reduced energy consumption of NEASA is more obvious. Main reason is that TDVAS only consider static slack time. NEASA consider both static slack time and dynamic slack time. While ECR increases, WVST also increases. Therefore, NEASA can reduce more energy compared with TDVAS. For makespan, NEASA, TDVAS and ETF have no difference.

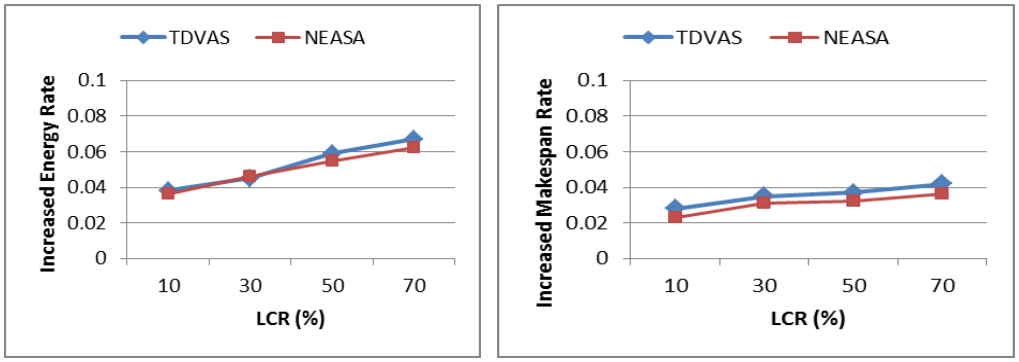

Figure 3. Increased Energy and Make Span Rate in Different LCR

Figure 3 show the impact of LCR on energy and makes pan. Both TDVAS and NEASA increased energy while LCR increases compared with ETF. For make span, the make span also slowly increases. But NEASA has less increase proportion compared with TDVAS. When LCR is large, NEASA has advantage over TDVAS. Main reason is that when some tasks complete late, NEASA would scale the frequency of successor tasks to maximum. So the successor tasks can execute rapidly.

5.2. Overall Energy Efficiency

CCR is the ratio of communication time and computation time. It is an important parameter. Computation intensive application has less CCR. Otherwise, communication application has greater CCR. NEASA mainly focuses on the computation energy optimization. Therefore, CCR is scaled from 0 to 1.

ECR and LCR range from 10% to 70%. They have

$$ECR + LCR = 1 \quad (15)$$

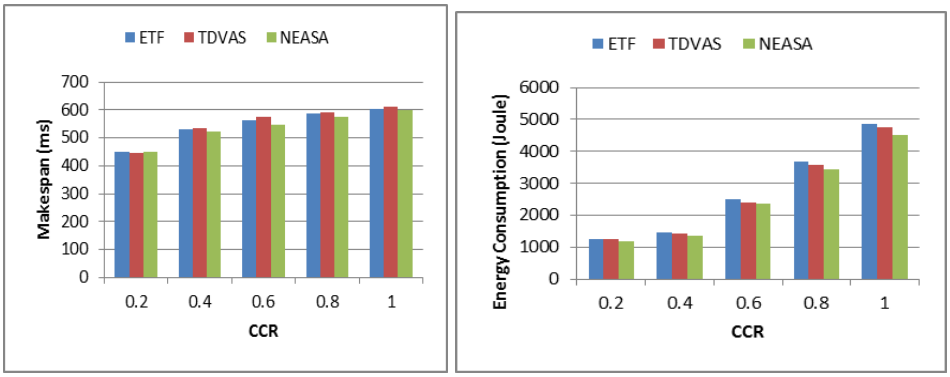

Figure 4. Make Span and Energy Consumption in Different CCR

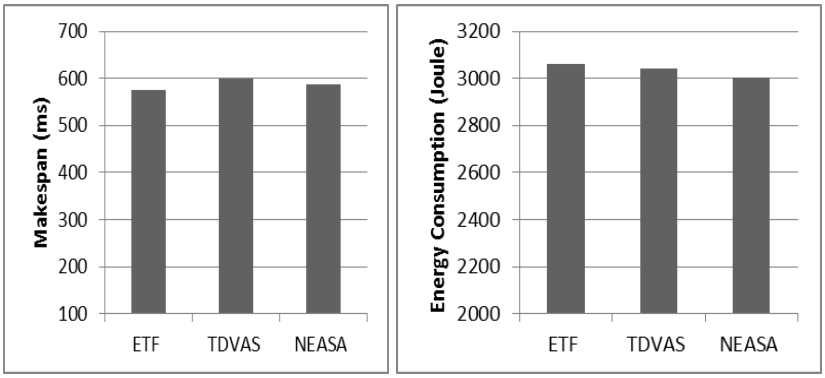

Figure 5. Make Span and Energy Comparison for Sparse Matrix

Figure 4 shows the make span and energy comparison for Random task sets with different CCR. NEASA can reduce the energy consumption compared with other two algorithms by 4% on average. For make span, NEASA has similar results compared with other two. Figure 5 shows the comparison results for Sparse Matrix. Both in make span and energy consumption, NEASA have no obvious advantage over other two algorithms. Main reason is that Sparse Matrix belongs to communication intensive application. The proposed algorithm focuses on the computation energy reduction. Therefore, NEASA does not apply to communication intensive applications.

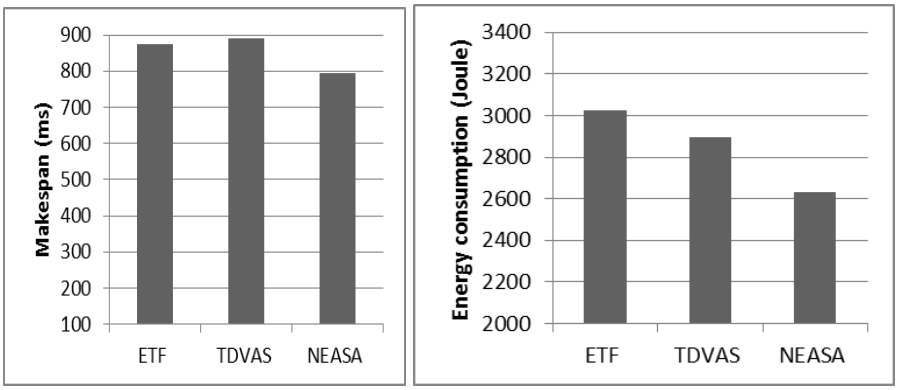

Figure 6. Make Span and Energy Comparison for SPEC fpppp

Figure 6 shows the make span and energy comparison for SPEC fpppp. SPEC fpppp belongs to computation intensive applications. NEASA can significantly reduce the energy compared with other two algorithms. Make span has also been reduced to a great extent. Energy consumption is reduced by 12% and 9.1% respectively on average compared with ETF and TDVAS. NEASA can effectively reduce the make span by 10.8% compared with TDVAS mainly because NEASA scales the processor frequency dynamically at run time. The proposed algorithm has great advantage for computation intensive applications.

To evaluate the proposed algorithm for synthetic applications, the make span and energy comparison results are showed in Figure 7. NEASA can reduce the energy by 8.3% and 5.6% respectively on average compared with ETF and TDVAS. Make span is reduced by 3.2% on average compared with TDVAS.

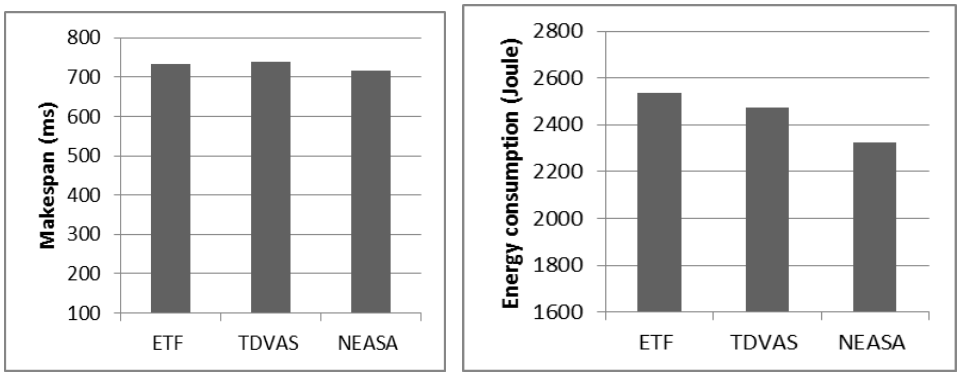

Figure 7. Make Span and Energy comparison for Random

6. Conclusion and Future Work

To reduce the energy consumption for precedence constrained parallel tasks, energy aware algorithm (named NEASA) based on two-phase frequency scaling is proposed, which reclaims both static and dynamic slack time and employs different frequency

adjusting techniques in different slack time. The extensive experiments are conducted. NEASA can reduce energy by 10.8% on average for computation intensive applications. Energy can be reduced by 5.6% on average for random applications. For communication intensive applications, the proposed algorithm has no obvious advantage.

Future research will investigate the scheduling method to reduce the communication energy on heterogeneous clusters where computational nodes have different processing capabilities and network interconnection may have various performances.

Acknowledgement

This study is supported by Beijing Educational Committee (KM201511417003), Beijing Higher Education Young Elite Teacher Project (YETP1772), National Sci-Tech Support Plan (2014BAK08B02), The Project of Construction of Innovative Teams and Teacher Career Development for Universities and Colleges under Beijing Municipality (IDHT20140508).

References

[1] R. Ge, X. Feng and K. W. Cameron, "Performance-constrained Distributed DVS Scheduling for Scientism Applications on Power-aware Clusters", Proceedings of the International Conference for High Performance Computing, Networking and Storage, **(2005)** November 12-18, Seattle, WA, USA.
[2] G. Laszewski, L. Wang, A. J. Younge and X. He, "Power-aware scheduling of virtual machines in DVFS-enabled clusters", Proceedings of IEEE International Conference on Cluster Computing, **(2009)** August 31 - September 4, New Orleans, USA.
[3] M. Marzollaa and R. Mirandolab, "Dynamic power management for QoS-aware applications. Sustainable Computing", Informatics and Systems, vol. 3, no. 4, **(2013)**.
[4] C. Hsu and W. Feng, "A power-aware run-time system for high performance computing", Proceedings of the ACM/IEEE conference on Supercomputing, **(2005)** Washington, DC, USA.
[5] J. Peraza, A. Tiwari, M. Laurenzano, L. Carrington and A. Snavely, "PMaC's green queue, a framework for selecting energy optimal DVFS configurations in large scale MPI applications", Concurrency and Computation: Practice and Experience, **(2013)**.
[6] M. Y. Lim, V. W. Freeh and D. K. Lowenthal, "Adaptive, transparent frequency and voltage scaling of communication phases in MPI programs", Proceedings of the ACM/IEEE conference on Supercomputing, **(2006)** New York, NY, USA.
[7] H. Kimura, M. Sato, Y. Hotta, T. Boku and D. Takahashi, "Empirical study on reducing energy of parallel programs using slack reclamation by DVFS in a power-scalable high performance cluster", Proceedings of IEEE International Conference on Cluster Computing, **(2006)**, Barcelona, Spain.
[8] X. Ruan, X. Qin, Z. Zong, K. Bellam and M. Nijim, "An Energy-Efficient Scheduling Algorithm Using Dynamic Voltage Scaling for Parallel Applications on Clusters", Proceedings of the 16th IEEE International Conference on Computer Communications and Networks, **(2007)** August Honolulu, Hawaii, USA.
[9] X. Zhu, C. He, K. Li and X. Qin, "Adaptive energy-efficient scheduling for real-time tasks on DVS-enabled heterogeneous clusters", Journal of Parallel and Distribute Computing, vol. 72, no. 6, **(2012)**, pp. 751-763.
[10] L. Wang, G. V. Laszewski, J. Dayal and F. Wang, "Towards energy aware scheduling for precedence constrained parallel tasks in a cluster with DVFS, In Proceedings of the 10th IEEE/ACM International Conference on Cluster", Cloud and Grid Computing **(2010)**, Melbourne, Australia.
[11] K. H. Kim, R. Buyya and J. Kim, "Power aware scheduling of bag-of tasks applications with deadline constraints on DVS-enabled clusters", Proceedings of the 7th IEEE international symposium on cluster computing and the grid, **(2007)**, Rio de Janeiro, Brazil.
[12] S. Baskiyar and R. Abdel-Kader, "Energy aware DAG scheduling on heterogeneous systems", Cluster computing, vol. 13, **(2010)**, pp. 373-383.
[13] Z. Zong, A. Manzanares, X. Ruan and X. Qin, "EAD and PEBD, Two Energy-Aware Duplication Scheduling Algorithms for Parallel Tasks on Homogeneous Clusters", IEEE Transactions on Computers, vol. 60, no. 3, **(2011)**, pp. 360-374.
[14] "Standard Task Graph Set", http://www.kasahara.elec.waseda.ac.jp/schedule/, **(2014)**.

Authors

Aihua Liang, received her Ph. D degree in School of Computer and Engineering from Beihang University. Currently she is working as an Associate Professor of Institute of Computer Technology at Beijing Union University. Her main research areas include computer system software, high performance computing, pattern recognition, image processing.