Low Energy and High Performance Scheduling on Scalable Computing Systems

Pecero Sánchez, J.E., Bouvry P.
CSC Research Unit, University of Luxembourg, L-1359 Luxembourg
johnatan.pecero@uni.lu, pascal.bouvry@uni.lu

Barrios Hernández, C.J.
Industrial University of Santander, Bucaramanga, Colombia
carlosjaimebh@computer.org

Abstract

With the fast development of supercomputers, energy consumption by large scale computer systems has become a major concern. How to reduce energy consumption is now a critical issue in designing high-performance computing systems. Moreover, reducing energy consumption for high-performance computing can bring various benefits such as, reduce monetary operating costs, increase system reliability, and reduction of environmental impacts. Therefore, in this paper we address the problem of scheduling precedence-constrained parallel applications on heterogeneous scalable computing systems with the objectives of minimizing finish time and reduce energy consumption. We provide a scheduling algorithm based on the best-effort idea that adopts dynamic voltage scaling (DVS) to reduce energy consumption. That is, the algorithm firstly looks for near-optimal solutions employing a list-based scheduling algorithm to find the minimum finish time (best-effort). Then, a fast random local search algorithm that exploits voltage scaling is used to reduce the energy consumption of the generated schedule without any performance degradation. Simulation results on structured graphs representing real-world applications emphasize the interest of the proposed approach.

Keywords: *Scalable computing systems; Heterogeneous computing; Distributed Systems; Green Computing; Energy Optimization; Task Scheduling;*

1 Introduction

Task scheduling problem for workflow applications has been one of the fundamental issues in parallel and distributed computing. Task scheduling on homogeneous and heterogeneous computing systems has been studied extensively in recent years addressed to different efforts. Traditionally, to minimize the application completion time and to treat time complexity. Nowadays, it exist an important interest to reduce energy consumption in computing systems due to performance, environment and monetary issues.

In environmental terms, since the idea to build environmentally sustainable computing or Green Computing [5] [3], the idea to minimize energy consumption or maximize energy efficiency designing algorithms and systems for efficiency-related computer technologies, scheduling strategies is an active field of research. One of the main concern is to provide high performance with low energy consumption, in other words to provide energy efficiency[1].

Several works are based in the use of Dynamic Voltage Scaling Algorithms (DVS). DVS exploits hardware characteristics of processors to reduce energy dissipation by lowering the supply voltage and the operating frequency. The DVS algorithms are shown to make dramatic energy savings while providing the necessary peak computation power in general purpose systems. On the other hand, DVS tries to address the tradeoff between performance and energy efficiency by taking into account two important characteristics: the peak computing rate needed is much higher than the average throughput that must be sustained and the processors are based on CMOS Logic. In this paper we adopt this technique. DVS enables processors to dynamically adjust voltage supply lev-

[1] Energy efficiency can be measured in terms of performance per watt, depending on the definition; reasonable measures of performance are FLOPS, MIPS, or the score for any performance benchmark (i.e. FLOPS per watt, MIPS per watt)

els aiming to reduce power consumption at the expense of performance degradation.

Using DVS algorithms, the most part of the studies on scheduling are addressed on homogeneous systems. Works as [1], [2], and [7] propose strategies to efficient energy consumption of HPC Clusters or VLSI systems considering deadlines of applications. In all cases the interval of energy savings that can be achieved is between 20% and 40% for different types of utilization, taking in account not only interactive workloads but also scientific workload caused by parallel scientific applications. However, few works treat the problem of heterogeneous systems and moreover of large scale systems such as Grid computing platforms. Works as [7] proposes algorithms implementable on both homogeneous and heterogeneous systems using task scheduling and exploiting DVS capabilities of processing elements to reduce energy consumption. The authors consider scientific applications represented by a Directed Acyclic Graph (DAG). The target architecture is a distributed computing system which consists of DVS enabled processors with varying processing capabilities and network links with varying bandwidths.

In this paper, the goal is to minimize the energy consumption of a scalable computing architecture when executing applications while a given performance is preserved. Since the resource manager is the component of a system responsible for deciding which tasks of a parallel application run on the processors simultaneously, scheduling is crucial for performance and energy efficiency. The main contributions of the paper are two-fold: problem modeling as a constrained multi-objective optimization. Design a scheduling algorithm based on the best-effort idea that optimize both makespan and energy objectives. That is, the algorithm firstly looks for near-optimal solutions employing a list-based scheduling algorithm to find the minimum makespan (best-effort). Then, we fix the makespan objective as constraint and a random local search algorithm that promotes voltage scaling is used to reduce the energy consumption of the generated schedule without incurring into the increase of the fixed criterion (i.e., the makespan).

The rest of the paper is organized as follows: Section 2 describes the models used in this paper. Section 3 presents the best-effort list scheduling and random local search and voltage scaling algorithms. Some simulation experiment results are discussed in Section 4. In Section 5, concludes remarks and discusses the future work.

2 Models

In this section, we describe the system, application, energy and scheduling models used in this work. Most of them are based on the models from [7].

2.1 System model

The target system used in this work is a scalable computing system that consists of a set M of m heterogeneous processors/machines that are fully interconnected. The processors have different processing speed or provide different processing performance in term of MIPS (Million Instruction Per Second). Each processor $m_j \in M$ is DVS-enabled; that is, it can be operated on a set of supply voltages V. Hence different speed performance.

In this work, we assume that for each processor $m_j \in M$, a set V_k of v voltages is random and uniformly distributed among six different sets of supply voltages (see **Table 2.1**).

We assume that when a processor is idle a lowest voltage is supplied. For sake of simplicity and without any loss of generality, we assume that all the processing elements are fully connected through network resource. That is, every processor has a direct link to any other processor. The studied network is considered to be nonblocking; that is, the communication of two processors does not block the connection of any other two processors in the network [8]. The inter-processor communications are assumed to perform with the same speed on all links without contentions. It is also assumed that a message can be transmitted from one processor to another while a task is executed on the recipient processor which is possible in many systems. Finally, communications between tasks executed onto the same processor are neglected. This computational model corresponds to the classical delay model [9].

2.2 Application model

Often, parallel programs are represented by a directed acyclic graph (DAG). A DAG, $G = (T, E)$, consists of a set T of n nodes corresponding to the tasks t_i of the parallel program and a set E of e edges. A DAG is also called a task graph or macro-dataflow graph. In general, the nodes represent tasks partitioned from an application; each edge $(t_i, t_j) \in E$ represents precedence constraints such that task t_j cannot start until t_i finishes its execution. The edge $(t_i, t_j) \in E$ between tasks t_i and t_j also represents inter-task communication. It means that, the output

Table 1. Voltage-Relative Speed Pairs [7, 10]

Level	Pair 1		Pair 2		Pair 3		Pair 4		Pair 5		Pair 6	
	Voltage V_k	Relative Speed (%)	Voltage V_k	Relative Speed (%)	Voltage V_k	Relative Speed (%)	Voltage V_k	Relative Speed (%)	Voltage V_k	Relative Speed (%)	Voltage V_k	Relative Speed (%)
0	1.50	100	2.20	100	1.50	100	1.75	100	1.20	100	1.35	100
1	1.20	80	1.90	85	1.40	90	1.40	80	1.15	90	1.25	85.7
2	0.90	50	1.60	65	1.30	80	1.20	60	1.10	80	1.20	71.5
3			1.30	50	1.20	70	0.90	40	1.05	70	1.10	57.1
4			1.00	35	1.10	60			1.00	60	0.9	32.2
5					1.00	50			0.90	50		
6					0.90	40						

of task t_i has to be transmitted to task t_j in order for task t_j start its execution. A task with no predecessors is called an entry task, t_{entry}, whereas an exit task, t_{exit}, is one that does not have any successors. Among the predecessors of a task t_i, the predecessor which completes the communication at the latest time is called the most influential parent (MIP) of the task denoted as $MIP(t_i)$. The longest path of a graph is called the critical path (CP).

To every task t_i, there is an associated value p_{ij} representing the computation cost of the task t_i on a processor m_j at a maximum speed and voltage (i.e., it corresponds to the Level 0 in Table 2.1), and its average computation cost is denoted as $\overline{p_i}$. The relative execution time p'_{ij} at Level x greater than Level 0 is given by the ratio (Eq.)

$$p'_{ij} = \frac{p_{ij}}{RelativeSpeed} \tag{1}$$

The weight on any edge stands for the communication requirement among the connected tasks. Thus, to every edge $(t_i, t_j) \in E$ there is an associated value c_{ij} representing the time needed to transfer data from t_i to t_j. However, a communication cost is only required when two tasks are assigned to different processors, as stated early. In other words, the communication cost when tasks are assigned to the same processor can be neglected.

The Earliest Start Time (EST) of, and the Earliest Finish Time (EFT) of, a task t_i on a processor m_j are defined as (Eq. 2 and 3, respectively):

$$EST(t_i, m_j) = \begin{cases} 0 \quad \text{if } t_i = t_{entry} \\ EFT(MIP(t_i), m_k) + c_{MIP(t_i),i} \\ \quad \text{otherwise} \end{cases} \tag{2}$$

$$EFT(t_i, m_j) = EST(t_i, m_j) + p'_{ij}, \tag{3}$$

where m_k is the processor on which $MIP(t_i)$ is scheduled.

Note that the Actual Start Time and Actual Finish Time of a task t_i on a processor m_j, denoted as $AST(t_i, m_j)$ and $AFT(t_i, m_j)$ can be different from its earliest start and finish times, $EST(t_i, m_j)$ and $EFT(t_i, m_j)$, if the actual finish time of another task scheduled on the same processor is later than $EST(t_i, m_j)$ [7].

In the case of adopting task insertion the task can be scheduled in the idle time slot between two consecutive tasks already assigned to the processor as long as no violation of precedence constraints is made. This insertion scheme would contribute in particular to increasing processor utilization for a communication intensive task graph with fine-grain tasks [6].

A simple task graph is shown in Figure 1 with its details in Tables 2 and 3. The values presented in Table 3 are computed using two frequently used task prioritization methods, *t-level* (top level) and *b-level* (bottom level). Note that, both computation and communication costs are averaged over all nodes and links. The t-level of a task t_j is defined as the summation of the computation and communication costs along the longest path of the node from the t_{entry} task in the task graph to t_j (excluding t_j). The $t-level(t_j)$ is computed recursively by traversing the DAG downward starting from the entry task t_{entry}.

In contrast, the b-level of a task t_i is computed by adding the computation and communication costs along the longest path of the task from the exit task t_{exit} in the task graph (including the task t_i). The b-level is used in this study. The $b-level(t_i)$ is computed recursively by traversing the DAG upward starting from the exit task t_{exit}.

The communication to computation ratio (CCR) is

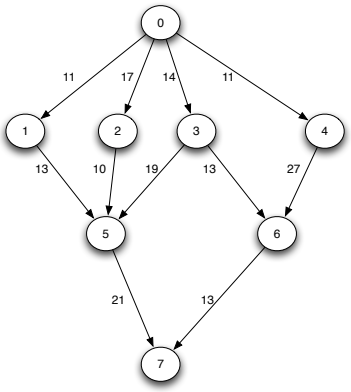

Figure 1. A simple precedence task graph

Table 2. Computation cost with maximum voltage

task	m_0	m_1	m_2	$\overline{p_i}$
0	11	13	9	11
1	10	15	11	12
2	9	12	14	12
3	12	16	10	12
4	15	11	19	15
5	13	9	5	9
6	11	15	13	12
7	11	15	10	12

a measure that indicates wether a task graph is communication intensive, computation intensive or moderate. For a given task graph, it is computed by the average communication cost divided by the average computation cost on a target system.

2.3 Energy model

The energy model used in this paper is derived from the power consumption model in complementary metal-oxide semiconductor (CMOS) logic circuits as defined in [7]. The power consumption of a CMOS-based microprocessor is defined to be the summation of capacitive, short-circuit and leakage power (static power dissipation).

The capacitive power (P_c) (dynamic power dissipation) is the most significant factor of the power consumption. It is directly related to frequency and supply voltage, and it is defined as:

$$P_c = AC_{ef}V^2f, \tag{4}$$

Table 3. Task priorities

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

where A is the number of switches per clock cycle, C_{ef} is the total capacitance load, V is the supply voltage, and f is the operating frequency [11, 12]. The relationship between circuit delay (T_d) and the supply voltage is approximated by (Eq. 5):

$$T_d \propto \frac{C_{ef}V}{(V - V_{th})^\alpha}, \tag{5}$$

where C_L is the load capacitance, V_{th} is the threshold voltage, and α is the velocity saturation index which varies between one and two (α=2). Because the clock frequency is proportional to the inverse of the circuit delay, the reduction of the supply voltage results in reduction of the clock frequency. It would be not beneficial to reduce the CPU frequency without also reducing the supply voltage, because in this case the energy per operation would be constant.

Equation 4 clearly indicates that the supply voltage is the dominant factor; therefore, its reduction would be most influential to lower power consumption. The energy consumption of the execution of any application (i.e., represented by a DAG) used in this work is defined as

$$E_c = \sum_{i=1}^{n} AC_{ef}V_i^2 f.p_i^* = \sum_{i=1}^{n} KV_i^2 p_i^*, \tag{6}$$

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

2.4 Scheduling model

In this work the scheduling model is defined as follows. The scheduling problem is the process of allocating a set N of n tasks to a set M of m processors (without violating precedence constraints) aiming to minimize makespan with energy consumption as low as possible. The makespan is defined as $C_{max} = max\{AFT(n_{exit})\}$ after the scheduling of n

tasks in a task graph G is completed. We do not consider deadlines of tasks as in real-time systems. Furthermore, tasks are not allowed to be preempted.

3 Best-effort and Random Local Search for Energy Optimization

As noted, the problem of scheduling precedence-constrained task graphs with minimum makespan and energy is a trade-off of schedule length and energy consumption. The reduction in energy consumption is often made by lowering supply voltage and this result in increasing the tasks execution, hence the schedule length. These two objectives are conflicting [7]. Different solutions produce trade-offs between the two objectives, which means there is no single optimum solution. In this case, the energy-aware task scheduling problem is modeled as a bi-criteria optimization problem and our objective becomes finding Pareto optimal schedules (i.e., non-dominated schedules), such that no schedule can be better and use less energy. A common approach to bicriteria problems is to fix one of the parameters. This approach corresponds to the ϵ-constrained method [13]. In this paper, we fix the schedule length and then we optimize the energy consumption on the scalable computing system. We consider the following scenario and the corresponding objective: *Reduce energy consumption without performance degradation*. That is, assume we know the makespan of a parallel application that minimizes the execution time in the scalable computing system. Determine a new schedule that tries to minimize the energy consumption on the computing system assuming no allowable increase in the makespan.

For the first step of the proposed approach, we look for solutions with optimal or near-optimal makespan. We consider a list scheduling heuristic as a best-effort algorithm. List scheduling heuristic is a two phase scheduling algorithm that maintain a list of all tasks of a given graph according to their priorities [14]. In the first phase of the algorithm a ready task is selected from the list based on its priority. The task with highest priority is selected. This process corresponds to the *task prioritizing* or *task selection* phase. Then, a suitable processor that minimizes a predefined cost function is selected (i.e., the *processor selection* phase). The best-effort algorithm we use is the task list scheduling heuristic proposed by Topcuoglu *et al.* in [6], Heterogeneous Earliest Finish Time (HEFT). We used HEFT because it is simple, well-known and was shown to be competitive in [6]. An extension to parallel task scheduling on cluster-based computing architectures was done in [14]. The pseudo-code of HEFT is presented in Algorithm 1.

Let us remark that HEFT uses the maximum voltage to obtain the makespan. Once HEFT computes the schedule for the application, we apply a random local search to reduce the energy consumption. The aim is not only to improve the quality of the makespan, but also to reduce energy.

Local search was one of the early techniques for combinatorial optimization [15]. The principle is to refine a given initial solution point in the solution space by searching through a neighborhood of the solution point. We use a simple random search in which the local search direction is randomly selected. If the initial solution point is improved, it moves to the refined solution point. Otherwise, another search direction is randomly selected. A simple neighborhood point of a schedule in the solution space is another schedule which is obtained by transferring a task from a processor to another processor. In our solution, we also define a neighborhood by changing a voltage and speed level to another operating point (i.e. level) of the processor. That is, the algorithm can modify only one parameter (i.e., to move one task to another location or to change voltage and speed to another operating point) at a time, or both parameters simultaneously. In both cases we define a new neighborhood.

The constant MAXSTEPS in Algorithm 2 is defined to limit the number of steps so that only MAXSTEPS tasks are examined. At each iteration of the random local search we compute a new sched-

Algorithm 1 Pseudo-code for HEFT.

1: Calculate the priority of each task according to the *b-level*
2: Sort the tasks in a scheduling list by descending order of tasks priorities
3: **while** *there are unscheduled tasks in the list* **do**
4: Remove the first task, n_i, from the list for scheduling
5: **for** each processor m_j in the processor set **do**
6: Compute EFT(t_i, m_j) value using the insertion-based scheduling policy
7: **end for**
8: Assign t_i on the processor m_j that minimizes the EFT of task t_i
9: **end while**

Algorithm 2 Pseudo-code for random local search and voltage scale.

1: searchstep = 0
2: **while** searchstep < MAXSTEPS **do**
3: Pick a task t_i randomly
4: Pick a processor m_j randomly
5: Pick a voltage v_k from the corresponding set of voltage of m_j randomly
6: Assign t_i on the processor m_j with the operating voltage v_k that minimizes both Energy and EFT of task t_i or minimizes Energy without increasing EFT
7: **end while**

ule. Step 6 indicates that a new solution is accepted if the movements improve the Energy and makespan or only the Energy consumption without any performance degradation. In other case the movements are not allowed and task n_i and voltage v_k are moved back to their original processor and operating point.

4 Performance Evaluation and Experiments

Simulation studies have been used to compare the energy saving capability and the performance of the proposed approach with the previously proposed HEFT. Before to present simulation results, we illustrate with an example the qualitative schedules in terms of makespan and Energy that the proposed approach could generate.

4.1 Performance evaluation

Figure 2 shows the schedules generated by HEFT and the proposed approach (HEFT + random local search and voltage scaling). The schedules were obtained taking into account three processors under three different pairs (i.e. voltage set and relative speed). Processor 1 (m_0) operates under Pair 4, processor 2 (m_1) uses Pair 3 and processor 3 (m_2) utilizes under Pair 2 as described in Table 2.1. In the left of Figure 2 the schedule generated by HEFT using maximum voltage (i.e., Level 0) is depicted. The schedule length is equal than 89 units of time and this schedule is obtained using 380 units of Energy. In the right shows the schedule computed by the proposed approach using as input the schedule generated by HEFT. The quality of the solution is improved by 16% while saving 28% of Energy. The makespan is equal to 74 units of time and 272 units of Energy are used to generate such as schedule.

4.2 Simulation results

In this section, we present the comparative evaluation of the proposed approach and HEFT by simulations. For this purpose, we consider some structured applications as the workload for testing the algorithms. The applications represent some real-world applications. The four real-world applications are a subset of tasks of the Laser Interferometer Gravitational Wave Observatory (LIGO) application [16, 17], a molecular dynamics code (MD-Code) application [6], the sparse matrix solver and fpppp applications from the *Standard Task Graph Set* (STG) [18]. Table 4 describes the main characteristics for these applications. The computational costs of the tasks in each application graph were generated as described in [6]. We fixed the parameter β to 1. Parameter β is basically the heterogeneity factor for processor speeds. A high percentage value (i.e., a percentage of 1) causes a significant difference in a task's computation cost among the processors. Additionally, for each graph we have varied the *communication to computation cost ratio* (CCR). It is the ratio of the average communication cost to the average computation costs. A very low CCR means that the application is considered as a computation-intensive. We have generated five CCRs (0.1, 0.5, 1, 2, 5) for each graph. The execution of the applications is performed on 8, 16 and 32 processors. We compare the algorithms in terms of the schedule length and Energy.

Table 4. Employed Benchmark and Their Main Characteristics

Application	# of Tasks	# of Edges
fpppp	334	1196
Sparse matrix	96	128
LIGO	76	131
MDCode	41	70

Figure 3 depicts the results of the simulations with respect to the five CCR values. We present only average results. The results indicate that the proposed approach outperforms HEFT in terms of Energy. Our algorithm utilizes less Energy than HEFT without degrading performance. The provided approach have generated schedules with makespan smaller than 3% on average than the schedules computed by HEFT. As in previous studies, HEFT has been proven to perform very competitively [6] and it has been fre-

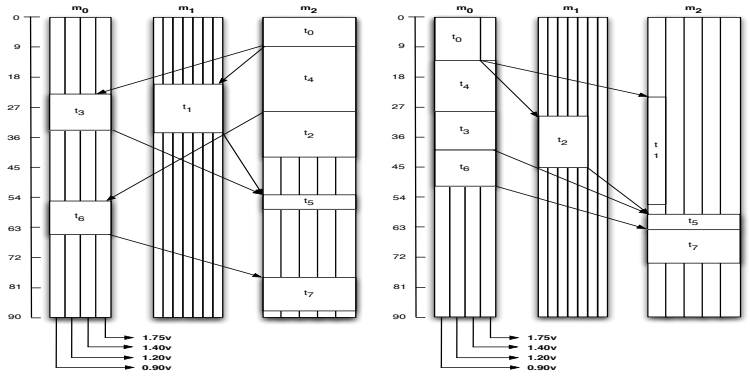

Figure 2. In left HEFT without DVSF (Makespan = 89, Energy = 380). In the right HEFT + random local search and voltage scaling (Makespan = 74, Energy = 272)

quently extended as for example in [14]; It implies that the average makespan of the proposed approach with even three percent margin is convincing.

In terms of Energy, our algorithm can achieve up to 16% energy saving on average in the experiments. From Figure 3 we can observe that in most of cases the energy saved increases as the CCR values increase. In some cases our approach can produce schedules with energy less than 30% than the schedules computed by HEFT, which is the case for $fpppp$ and sparse applications and for a CCR equal to five (i.e. in this case the application is communication-intensive).

5 Conclusion and Future Work

We have presented a best-effort scheduling algorithm with random local search and voltage scaling to minimize energy consumption in scalable computing systems. We investigated the problem of minimizing energy for precedence task graph execution. We modeled the problem as a constrained optimization problem. The main idea was to look for schedules with minimum execution time (best-effort), hence the makespan was fixed as a constraint for energy optimization. A random local search combined with voltage scaling was developed for energy optimization.

Although the results may be controversial, values observed in several applications have this behavior. Differences with other models suggest take into account infrastructure features and propose more experimental testbeds. Precisely, future work includes to investigate the proposed approach on applications with arbitrary structure and large number of tasks. On the other hand, it exists the possibility to study how implement the algorithm in real time monitors and in structure management systems (using automatic tools), to establish efficient mechanisms to minimize energy consumption.

Acknowledgment

This work is supported by the National Research Fund (FNR) of Luxembourg through project Green-IT no. C09/IS/05 and the Luxembourg research grant scheme (AFR) through the grant no. PDR-08-010.

References

[1] Chen, J. J., and Kuo, T. W. *Multiprocessor Energy-Efficient Scheduling for Real-Time Tasks with Different Power Characteristics*, Proc of Int Conf on Parallel Processing, Norway, 2005.

[2] Ge, R., Feng, X. and Cameron, K. W. *Performance constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters*, Proc of the ACM/IEEE Conf on Supercomputing, USA, 2005.

[3] The Green500 List Site http://www.green500.org

[4] Kim, K. H., Buyya, R. and Kim, J. *Power Aware Scheduling of Bag-of-Tasks Applications with Deadline Constraints on DVS-enabled Clusters*, Proc of the IEEE Int Symp on Cluster Computing and the Grid, CCGRID'07, Brazil, 2007.

[5] Murugesan, S., *Harnessing Green IT: Principles and Practices*, IT Professional, Vol. 10, Issue 1,

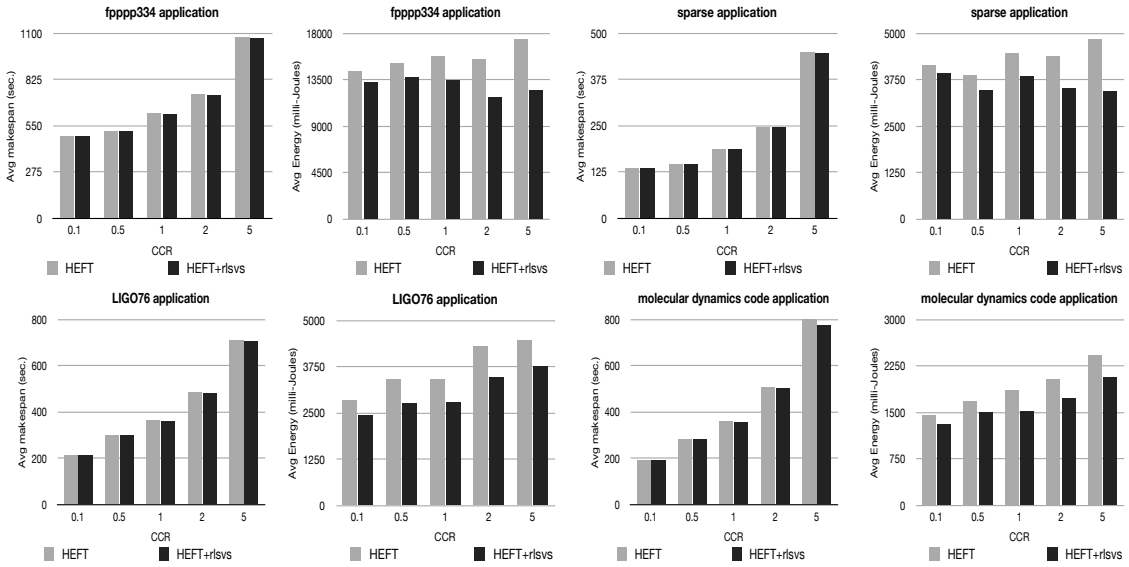

Figure 3. Makespan and Energy for all the applications with respect to CCR values

Pages 24-33. IEEE Educational Activities Department Piscataway, NJ, USA, 2008.

[6] Topcuoglu H., Hariri S. and Wu M. Y., Performance-effective and Low Complexity Task Scheduling for Heterogeneous Computing, *IEEE Trans. Parallel Distrib. Syst.*, Vol. 13, Issue 3, Pages 260-274, 2002.

[7] Lee, Y. C. and Zomaya A. Y. *Minimizing Energy Consumption for Precedence-Constrained Applications Using Dynamic Voltage Scaling*, Proceedings of the IEEE/ACM Int Symp on Cluster Computing and the Grid, CCGRID'09, China, 2009.

[8] Sinnen O. *Task Scheduling for Parallel Systems*. Hoboken, NJ: Wiley-Interscience, 2007.

[9] Papadimitriou C. H., and Yannakakis M. Towards and architecture independent analysis of parallel algorithms, *SIAM Journal on Computing*, vol. 19, no. 2, pp. 322–328, April 1990.

[10] Rizvandi N.B., Taheri J., Zomaya A. Y. and Lee Y. C. *Linear Combinations of DVFS-Enabled Processor Frequencies to Modify the Energy-Aware Scheduling Algorithms*, Proc of the IEEE/ACM Int Symp on Cluster Computing and the Grid, CCGrid'10, Pages 388-397, Melbourne, VIC, Australia, 2010

[11] Burd T. D., Pering T. A., Stratakos A. J. and Brodersen R. W. A Dynamic Voltage Scaled Microprocessor System, *IEEE J. Solid-State Circuits*, Vol. 35, No. 11, Pages 1571–1580, 2000.

[12] Brodersen R. W., Chandrakasan A. P. and Sheng S. Low-power CMOS Digital Design, *IEEE J. Solid-State Circuits*, Vol. 27, No. 4, Pages 473–484, 1992.

[13] Chankong V. and Haimes Y. Multiobjective Decision Making Theory and Methodology, *Elsevier Science*, New York, USA, 1983.

[14] Suter, F., Desprez, F. and Casanova, H. *From Heterogeneous Task Scheduling to Heterogeneous Mixed Parallel Scheduling*, LNCS, Special Edition for Europar 2004. Pages 230-237. Springer Ed. Germany, 2004.

[15] Gu J. Local Search for Satisfiability (SAT) Problem, *IEEE Trans. Systems, Man, and Cybernetics*, Vol. 23, No. 4, Pages 1108-1129, 1993.

[16] Barish B. C. and Weiss R. LIGO and the Detection of Gravitational Waves, *Physics Today*, Vol. 52, Pages 44–50, 1999.

[17] Brown D. A., Brady P. R., Dietz A., Cao J., Johnson B. and McNabb J. *A case study on the use of workflow technologies for scientific analysis: Gravitational Wave Data Analysis*, In Taylor, I., D. Gannon, and M. Shields (eds.), Workflows for e-Science Scientific Workflows for Grids, Springer, Pages 39–59, 2007.

[18] Tobita T. and Kasahara H. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms, *J Scheduling*, Vol. 5, Issue 5, Pages 379–394, 2002.