

Parallel Genetic Algorithms for DVS Scheduling of Distributed Embedded Systems

Man Lin* and Chen Ding

Department of Mathematics, Statistics and Computer Science,
St. Francis Xavier University
{mlin,x2002rah}@stfx.ca

Abstract. Many of today's embedded systems, such as wireless and portable devices rely heavily on the limited power supply. Therefore, energy efficiency becomes one of the major design concerns for embedded systems. The technique of dynamic voltage scaling (DVS) can be exploited to reduce the power consumption of modern processors by slowing down the processor speed. The problem of static DVS scheduling in distributed systems such that the energy consumption of the processors is minimize while guaranteeing the timing constraints of the tasks is an NP hard problem. Previously, we have developed a heuristic search algorithm: Genetic Algorithm (GA) for the DVS scheduling problem. This paper describes a Parallel Genetic Algorithm (PGA) that improves over Genetic Algorithm (GA) for finding better schedules with less time by parallelizing the GA algorithms to run on a cluster. A hybrid parallel algorithm is also developed to further improve the search ability of PGA by combining PGA with the technique of Simulated Annealing (SA). Experiment results show that the energy consumption of the schedules found by the PGA can be significantly reduced comparing to those found by GA.

1 Introduction

With the growing mobile technology, nowadays our life is heavily relied on mobile and portable devices with built in CMOS processors. The biggest limitation of current electronic devices is the battery life. The technique of dynamic voltage scaling(DVS) has been developed for modern CMOS processors, which can effectively lower the power consumption and enable a quieter-running system while delivering performance-on-demand(DVS) [1]. Many of today's advanced processors, such as AMD and Intel, have this technology.

The CPU power consumed per cycle in a CMOS processor can be expressed as $P = C_L f V_{DD}^2$, where C_L is the total capacitance of wires and gates, V_{DD} is the supply voltage and f is the clock frequency. It is obvious that a lower voltage level leads to a lower power consumption. The price to pay for lowering the voltage level is that it also leads to a lower clock frequency and thus slows

* The author would like to thank NSERC (National Science Engineering Research Council, Canada) and CFI for supporting this research.

down the execution of a task. The relation between the clock frequency and the supply voltage is: $f = K * (V_{DD} - V_{TH})^2 / V_{DD}$. As a result, exploiting DVS may hurt the performance of a system (If we reduce the voltage by half, the energy consumed will be one-quarter and the execution time will be double.) When using DVS in a hard real-time system where tasks have deadlines, we can not lower the voltage levels of the processors too much as we also need to guarantee the deadlines of the tasks be met.

In the past few years, there have been a number of algorithms proposed for applying DVS to hard real-time systems. Many previous works of DVS-based scheduling either focus on single processor power conscious scheduling or consider independent tasks only [2,3]. In this paper, we focus on static DVS-based scheduling algorithm for distributed real-time systems consisting of dependent tasks. We consider discrete-voltage DVS processors instead of variable-voltage DVS as real DVS processors show only a limited number of supply voltage levels at which tasks can be executed.

The problem of optimally mapping and scheduling tasks to distributed systems has been shown, in general, to be NP-complete [4]. Because of the computational complexity issue, heuristic methods have been proposed to obtain optimal and suboptimal solutions to various scheduling problems. Genetic algorithms, inspired by Darwin's theory of evolution, have received much attention as searching algorithms for scheduling problems in distributed real-time systems [5,6,7]. The appeal of GAs comes from their simplicity and elegance as robust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems [8]. Our previous work has adopted Genetic Algorithms for DVS-based scheduling algorithm [9]. The method is different from other approaches of DVS scheduling of distributed systems [10,11,12,13,14] in that it does not just construct one schedule once, it constructs populations of schedules and iterates many generations to find one near optimal schedule. Different from another GA algorithm [15], our GA algorithm integrates task assignment (to which processor the tasks will be assigned), task scheduling (when the tasks will be executed) and voltage selection (at which voltage level the tasks will run) at the same phase.

When the task size becomes bigger, the search space becomes larger. In order to find better DVS schedules more efficiently, we design parallel genetic algorithms (PGA) to improve our previous GA. The PGA is implemented on a SUN cluster with 33 computation nodes. The PGA cuts down the computation time of genetic algorithm by reducing the population to smaller size in order to achieve reasonable speed up with good result. And it also expands the search space, increases the probability to obtain better result. A hybrid parallel algorithm which combines PGA and Simulating Annealing (SA) is also developed to further improve the search process.

The paper is organized as follows. First, the energy model and the task model are described in Section 2. The Parallel Genetic Algorithm (PGA) for DVS scheduling will be described in Section 3 and the experimental results will be shown in Section 4. Finally the conclusions are presented in section 5.

2 Task and Schedule Models

We consider the energy aware scheduling problem for a set of task T_1, T_2, \dots, T_N on a set of heterogeneous processors P_1, P_2, \dots, P_M where N is the number of tasks and M is the number of processors. Each processor has a number of voltage levels. The power dissipation for a processor p running at level l is denoted as $POW_{p,l}$.

There are precedence constraints among tasks which can be represented by a directed acyclic graph (DAG). If there is an edge from task T_i to T_j in the DAG, then T_j can only start after T_i finishes execution.

- Each task has a Worst Case Execution Time (WCET) on each processor for a given voltage level. The worst case execution time of task t on processor p at voltage level l is represented as $w_{t,p,l}$.
- Each task has a deadline. d_i is used to denote the deadline of task i .
- Suppose task t is assigned to processor p and run at voltage level l . Then the energy used to execute task t is given by $POW_{p,l} * w_{t,p,l}$.
- Assume T_i is mapped to processor $P(i)$ and runs at level $L(i)$. Then the total energy consumptions can be easily calculated as follows: $E_{total} = \sum_{i=1}^N (POW_{P(i),L(i)} * w_{T_i,P(i),L(i)})$.

To simplify the energy model, we assume that it takes no time to switch from one voltage level to another and therefore no energy consumed accordingly.

2.1 DVS Scheduling Example

Let's consider a very simple scheduling example with 3 tasks to be mapped into 2 processors as shown in Fig. 1(a).

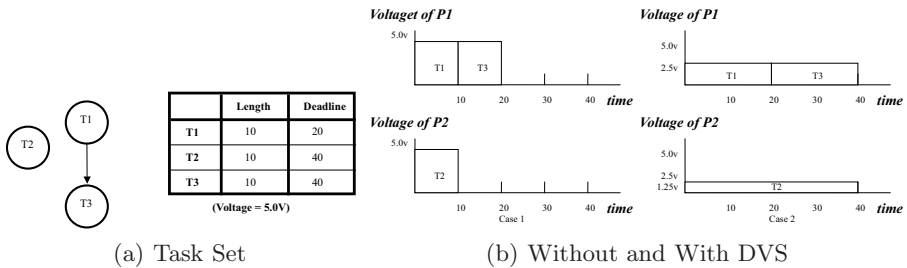


Fig. 1. An Example

Case 1 and case 2 in Fig. 1(b) shows the schedule without DVS and with DVS, respectively. With DVS, task T_1 and T_3 can be slowed down to run at 1/2 speed and consume only 1/4 energy. Task T_2 can save more energy consumption as it can slow down more.

3 Parallel Genetic Algorithms

GA approach can find better solutions for most scheduling problems in terms of the feasibility of the solutions and the energy saving [9]. However, when the problem size increases, it takes very long time for the solution to converge due to the bigger population size needed. Parallel computations reduce the computation time significantly when using multiple processors. To find better solution faster, we developed an Parallel Genetic algorithm (PGA) for the energy aware scheduling problem.

The chromosome representation and the genetic operator are adopted from the GA developed previously [9].

3.1 Chromosome Representation

Each chromosome is a schedule represented by an ordered list of genes and each gene contains three data items: Task (the task number), Proc (the processor number) and Level (the voltage level). The chromosome can be viewed as an $N * 3$ array where N is the total number of tasks. The first row of a chromosome indicates the tasks ordered from left to right. The second row indicates the corresponding processor that each task will be assigned to. And the third row is the voltage level selected for the corresponding processor for each task.

In the example shown in table 1, tasks t_1 , t_2 , t_3 and t_4 are to be scheduled onto processor p_1 and p_2 where both of the processors have two voltage levels. Task t_2 and t_3 are assigned to process p_2 and runs at voltage level 1 and 2 respectively. Task t_4 and t_1 are assigned to process p_1 and runs at level 2 and 1 respectively.

Table 1. A Schedule Example

Task	2	3	4	1
Proc	2	2	1	1
Level	1	2	2	1

A random order of tasks may result in infeasibility because the precedence constraints might be violated. To avoid such problem, we only allow chromosomes that satisfy topological order [7]. A topological ordered list is a list in which the elements satisfy the precedence constraints. To maintain all the individuals in the populations of any generation to be topological ordered, we adopted the techniques in [7].

- Generate only topological ordered individuals in the initial population;
- Use carefully chosen genetic operators (see section 3.2).

Note that the deadline constraint may still be violated even when a schedule satisfies a topological order.

3.2 Genetic Operators

Genetic Algorithms generate better solutions by exploring the search space by genetic operators. New individuals are produced by applying crossover operator or mutation operator to the individuals in the previous generation. The probability of the operators indicates how often the operator will be performed. We choose 0.6 as the probability for the crossover and 0.4 as the probability for the mutation.

The mutation operator creates a new individual with a small change to a single individual. In our approach, we use Processor and/or Level Assignment Mutation. To perform the mutation, we randomly select a range of genes from a chromosome and then randomly change the processor number and/or the level number within this range. Obviously, the mutation operator does not change the order of the tasks. Therefore, the new individual also satisfy the topological order.

The crossover operator creates new individuals by combining parts from two individuals. To perform crossover operation, we first randomly pick two chromosomes (as parents) from the current population. Then we randomly select a position where the crossover is going to occur. The first part of child 1 uses the schedule of parent 1 up to the chosen position. The second part of child 1 is constructed by selecting the rest tasks (the tasks not in the first part) from parent 2 in order. The same mechanism also applies to child 2. Below is an example of our crossover operator. Assume we have two individuals as shown in Fig. 2(a). Suppose the selected position is 2, then the children will be shown as in Fig. 2(b).

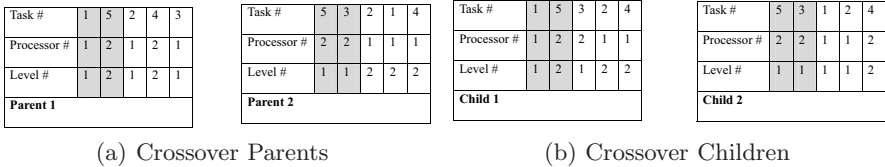


Fig. 2. Crossover operator

The crossover operator will produce two offsprings that are topological ordered if the parents are topological ordered. The detailed proof can be found in [7].

3.3 Parallel Environment

Our parallel hardware environment is a Sun Netra X1 Cluster Grid of 33 Sun servers (nodes). Each node has a Ultra SPARC II 64bit CPU, capable of 1 Gflops with 1 Gb main memory.

Based on the characteristic of coarse-grain PGA, a C language with MPI (Message Passing Interface) library has been chosen for our PGA programming.

We choose to implement a coarse-grain PGA to solve our problem because it is the most popular method and others have reported good results with this method in literature. First it cuts down the computation time of genetic algorithm by reducing the population to smaller size in order to achieve reasonable speed up with good result. Second improvement is that it expands the search space, increases the probability to obtain better result. And most it is obvious the large that amount of nodes used the better chance to get good result.

3.4 The Basic PGA Algorithm

In our PGA, subpopulation is employed in each node instead of the whole population as shown in Fig. 3. Assume the population of PGA is denoted as $POPSIZE$ and there are n computation nodes. Each node performs GA with $POPSIZE/n$ number of individuals in the subpopulation. Note that when a larger number of nodes being used, the subpopulation size may become too small. To avoid subpopulation exiguity problem, we set a minimal value for subpopulation. Typically in our case, 40 is the minimal population size. Each node randomly creates initial schedules that satisfy the partial order constraints.

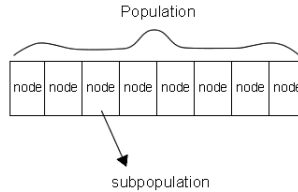


Fig. 3. Subpopulation Division

After initialization, PGA goes into a loop with many iterations, each iteration contains two steps: 1) basic GA operations in each node including selection, crossover and mutation and 2) communications among nodes. Step 1) in each iterations allows each node generate a new subpopulation. Each node then finds its own best chromosome out of the new subpopulation. In Step 2), one node sends its best solution to the rest of nodes. Note that the nodes take turns to send the best solution, one node in each iteration. Each node compares the best solution that it receives with its own best solution and determines whether to replace or discard it following the replacing policy.

The iterations will keep repeating until the stopping criteria are satisfied. Finally, the master node gathers all best chromosome from all the nodes and finds the best one among all.

The cost of communication between two MPI processors can be approximately divided into two parts: (i) Start-up time $T_{startup}$ and (ii) Transmission Time $T_t = (\text{Data size})/\text{Bandwidth}$. Start-up time is the duration needed for MPI to set up the communication links. Transmission time is the time needed to send

the message to its destination. The most commonly used model of the cost of communication is linear: $T_{comm} = T_{startup} + (Data\ size)/Bandwidth$.

When PGA broadcasts the best chromosome, a large amount of communication is required. Since the chromosome contains many pointers and MPI is not able to send non-continuous memory block at once, we defined a MPI communication data type called `ChromosomeType` that contains all the information in a chromosome. Therefore each MPI communication only need one start up time plus one transmission time to broadcast the best chromosome.

3.5 PGA Strategies

Initial Schedules Using EDF with TopologicalOrder. To increase the efficiency of our genetic algorithm, during the initialization of PGA, the generalization of schedules include two schedule that uses EDF scheduling policy while conforming to the topological order of tasks, where one has minimal energy level and the other one has maximal energy level.

Evaluation and Punishment of Infeasible Solutions. The aim of our optimization problem is to minimize the energy consumption of the tasks. The evaluation function is defined as the energy consumption of all the tasks.

However, the individual schedule with smaller E_{total} will not always be considered better. This is because we need to consider one more factor: the deadline constraints (precedence constraints are already encoded into the schedule enforced by the topological order.). Our algorithm give penalty to the individuals violating the deadline constraints so that they have less chance in getting into the next generation.

The Policy of Replacing the Best and Worst Solution. The best feature of parallel programming is that one computation node can communicate its result to other nodes to help them get close to optimal solution in the search process. So exchanging results becomes the most important issues in our PGA. The nodes send their best result sequentially to other nodes. When a node receive a solution from others nodes, it will determine whether to use it or discard it.

The replacing best policy works as follow: before communication, each node find both the best and worst chromosome indexes. After communication, the local best index will be compared to the received best solution. If the local best is better than the received one, the node abandons the received one, and replaces the local worst index with the received best. Otherwise, the node replaces both the local best and local worst solution with the receive best solution.

Cost Slack Stealing. In a schedule, it is possible that some tasks have slacks. Our cost slack stealing strategy is to determine whether the scheduling has slacks and try to reuse these slacks by extending the task execution time as much as possible by using the feasible lowermost energy level, so the total tasks execution time can be reduced. Considering the speed efficiency of PGA, this cost clack stealing method only performed on the final best result in each node to gain last enhancement of the PGA.

3.6 Hybrid Parallel Algorithm

This section, we present a hybrid parallel algorithm: parallel simulation annealing and genetic algorithm(PSAGA), to further improve the performance.

Simulated annealing (SA) is a generic probabilistic meta-algorithm for the global optimization problem, namely locating a good approximation to the global optimum of a given function in a large search space [16].

In analogy with the physical process of annealing in metallurgy, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the temperature), that is gradually decreased during the process. The allowance for "uphill" moves saves the method from becoming stuck at local minimum as would the greedier methods.

GA lacks of focus in the search due to the crossover operation. The combination of SA with GA allows finding a better solution in a particular search point and search area. PSAGA preforms an extra SA process besides those of PGA in each iteration. After performing replacing the best and worst solution, an SA will be started using the best schedule currently found. The SA tries to find the best neighbor in each iteration.

4 Experimental Results

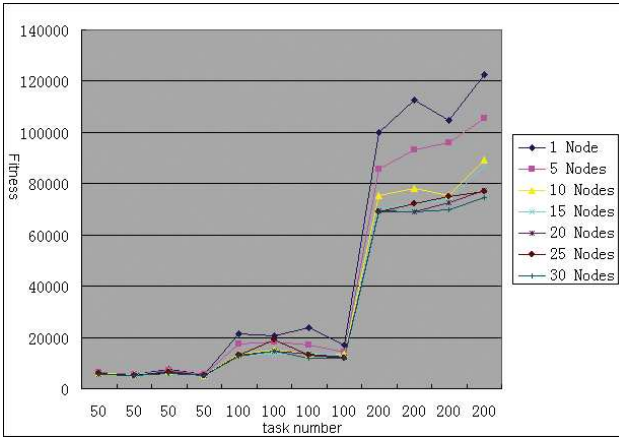
In this section, we describe the simulation experiments preformed and the results.

First we describe the features of the generated scheduling problems in the experiments. Fig. 4 shows the number of task and the number of constraints we have considered in our experiments. For each category, we compared GA, PGA, and PSAGA. The results you see later in the section are average result of the 5 problems in each category. Beside the task graph, each scheduling problem has one more variable: the number of processors. We have chosen 3, 5, and 10 processors for each task graph but we mainly compared 10 processors cases in this paper.

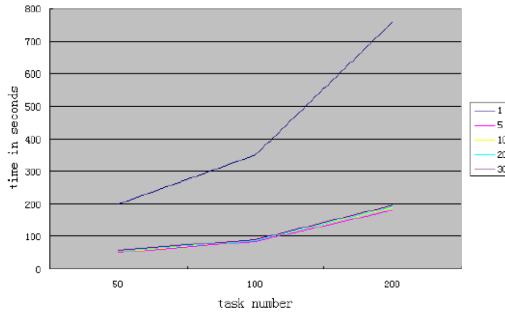
Number of Tasks	Number of Constraints
50	0, 10, 25
100	0, 10, 25, 50
200	0, 10, 25, 50, 100

Fig. 4. Test Parameters

We first demonstrate the efficiency of energy reduction and the corresponding runtime of the PGA algorithm. Fig. 5(a) shows the average energy consumption of the solutions found by GA and PGA using various computation nodes under different test categories. When the number of node equals one, it means that it



(a) Energy Consumption



(b) Computation Time

Fig. 5. The Parallelization Effect

is a serial genetic algorithm. That is, the GA program here is represented by a 1 node PGA program. For small task size 50, GA and PGAs almost have the same result, and it is hard to see any advantage of PGA in this case. PGA's results get better when the computation nodes used for PGA increase. The improvement is particularly obvious when the task size is above 100 where the energy consumption can be reduced more than 50% with 8 times less run time. With task number being 500, the difference seems to be more obvious. PGA has better solution as the size of nodes is increased and clearly the results show that the 30 node PGA has the best result overall. With more nodes PGA can find a schedule closer to the optimal solution. Fig. 5(b) shows the corresponding average computation time used for the GA and PGA, under different computation nodes. The run time speed up of PGA with 5 nodes is 4. As the node increases to more than 5, we use a fixed time for PGA as we aim to find better solutions and the computation time is tolerable.

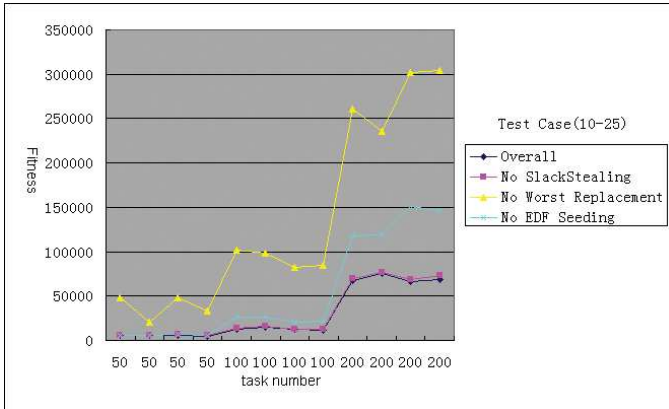


Fig. 6. PGA Strategies

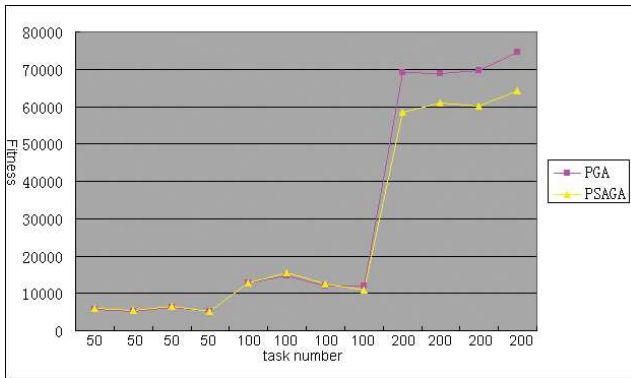


Fig. 7. PSAGA vs. PGA

Next, we show the effect of the typical strategies in PGA in Fig. 6. We found that policy of the best and worst replacement results in the best fitness than any other strategies. Including initial schedules using EDF becomes more efficient when the task number becomes larger. And slack stealing contributes to some improvement.

For the PSAGA, we only compare it with the PGA with 30 nodes as an example. From Fig. 7, for task size being 50 to 100, PSAGA and PGA have similar results. But when the task size increases to 200, the difference becomes obvious. PSAGA can save around 15% more energy than PGA. As the results are similar when using different number of nodes, we can expect that when the schedule problem size gets larger, the PSAGA performs better than PGA. When task size increases to 200, PSAGA can reduce 15% more energy than PGA.

5 Conclusion

We have addressed energy minimization problem in distributed embedded systems in this paper. A Parallel Genetic Algorithm (PGA) has been described for static DVS scheduling in such systems. The PGA improves over GA both on the optimization results and the computation speed. The PGA has been implemented in a SUN cluster. Various strategies of the PGA, such as the communication between the nodes in the cluster and the replacing strategies of best and worst individuals, have been discussed. A large number of experiments were conducted. The experiments show that PGA can find better solutions than GA for most scheduling problems in terms of the feasibility of the solutions and the energy saving. The improvement is particularly obvious when the task size is above 100 where the energy consumption can be reduced more than 50% with 8 times less run time. PSAGA (Parallel hybrid Simulating Annealing and Genetic Algorithm) is also employed to achieve the further improvement of PGA.

References

1. Burd, T.D., Pering, T.A., Stratakos, A.J., Brodersen, R.W.: A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuits* 35, 1571–1580 (2000)
2. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pp. 89–102. ACM Press, New York (2001)
3. Demers, A., Shenker, F.Y.: A scheduling model for reduced CPU energy. In: *Proceedings of the 36th Annual Symposium on Foundations of Computer Science*, pp. 374–382 (1995)
4. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman Publisher, San Francisco (1979)
5. Hou, E.S.H., Ansari, N., Ren, H.: A genetic algorithm for multiprocessor scheduling. *IEEE Transactions Parallel and Distributed Systems* 5, 113–120 (1994)
6. Lin, M., Yang, L.T.: Hybrid genetic algorithms for partially ordered tasks in a multi-processor environment. In: *Proceedings of the 6th International Conference on Real-Time Computing Systems and Applications (RTCSA-99)*, Hongkong, pp. 382–387 (1999)
7. Oh, J., Wu, C.: Genetic-algorithm-based real-time task scheduling with multiple goals. *Journal of Systems and Software*, 245–258 (2004)
8. Glover, F., Laguna, M.: *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications (1993)
9. Lin, M., Ng, S.: Energy Aware Scheduling for Heterogeneous Real-Time Embedded Systems Using Genetics Algorithms. In: *New Horizons of Parallel And Distributed Computing*, vol. ch. 8, pp. 113–127. Kluwer Academic Publisher, Dordrecht (2005)
10. Luo, J., Jha, N.K.: Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. In: *ASP-DAC/VLSI Design 2002*, Bangalore, India, pp. 719–726 (2002)
11. Zhang, Y., Hu, X., Chen, D.: Task scheduling and voltage selection for energy minimization. In: *DAC 2002*, New Orleans, Louisiana, USA, pp. 183–188 (2002)

12. Zhu, D., Melhem, R., Childers, B.: Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems. In: Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS-01), pp. 84–94. IEEE Computer Society Press, Los Alamitos (2001)
13. Mishra, R., Rastogi, N., Zhu, D., Mosse, D., Melhem, R.: Energy aware scheduling for distributed real-time systems. In: International Parallel and Distributed Processing Symposium (IPDPS'03), Nice, France (2003)
14. Ruan, X.J., Qin, X., Zong, Z.L., Bellam, K., Nijim, M.: An energy-efficient scheduling algorithm using dynamic voltage scaling for parallel applications on clusters. In: Proc. 16th IEEE International Conference on Computer Communication and Networks (ICCCN) (2007)
15. Schmitz, M., Al-Hashimi, B., Eles, P.: Energy-efficient mapping and scheduling for DVS enabled distributed embedded systems. In: Proceedings of 2002 Design, Automation and Test in Europe (DATE2002), pp. 514–521 (2002)
16. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* 4598, 671–680 (1983)