Energy-Constrained Scheduling of DAGs on Multi-core Processors

Ishfaq Ahmad[1], Roman Arora[1], Derek White[1], Vangelis Metsis[1], and Rebecca Ingram[2]

[1] University of Texas at Arlington, Computer Science and Engineering
iahmad@cse.uta.edu, roman.arora@mavs.uta.edu, derekwwhite@mavs.uta.edu, meci@uta.edu
[2] Trinity University, Computer Science Department
ringram@trinity.edu

Abstract. This paper proposes a technique to minimize the makespan of DAGs under energy constraints on multi-core processors that often need to operate under strict energy constraints. Most of the existing work aims to reduce energy subject to performance constraints. Thus, our work is in contrast to these techniques, and it is useful because one can encounter numerous energy-constraint scenarios in real life. The algorithm, named Incremental Static Voltage Adaptation (ISVA), uses the Dynamic Voltage Scaling technique and assigns differential voltages to each sub-task to minimize energy requirements of an application. Essentially, ISVA is a framework, rather than yet another DAG scheduling algorithm, in that it can work with any efficient algorithm to generate the initial schedule under no energy constraints. Given the initial schedule, ISVA efficiently identifies tasks' relative importance and their liabilities on energy. It then achieves the best possible new schedule by observing its energy budget. The algorithm marginally degrades the schedule length with extensive reduction in energy budgets.

Keywords: Parallel processing, multi-cores, scheduling, energy.

1 Introduction

Multi-core processors are becoming increasingly popular. By integrating several cores on a single chip and by running multiple threads in parallel on the same chip, considerable performance gains are expected. However, the lack of generally applicable software and tools for allocating tasks to cores remains a key challenge. With an increasing number of cores, multi-core processors are becoming progressively complex and heterogeneous in nature. Moreover, aggressive scalability of these architectures can lead to significant power and heat dissipation, making the adjustment of the voltage and frequency of cores essential considerations in scheduling. For these reasons, research is now being done to design energy-aware scheduling algorithms that rely on voltage scaling to reduce processors' power usage.

In this paper, we address the problem of power-aware scheduling/mapping of tasks onto homogeneous processor architectures. The objective is to minimize the energy

S. Ranka et al. (Eds.): IC3 2009, CCIS 40, pp. 592–603, 2009.

consumption and the makespan of complex computationally intensive scientific problems, subject to energy constraints. Most of the existing work aims to reduce energy subject to performance constraints. Thus, our work is in contrast to these techniques, and it is useful because one can encounter numerous energy-constraint scenarios in real life. Energy-constraints are often the result of limited battery supply in mobile environments where multi-core processors are being launched. Moreover, there are thermal constraints that can also place a limit on the maximum energy consumption.

Most energy minimization techniques are based on Dynamic Voltage Scaling (DVS) [3]. Specifically, we propose an algorithm by utilizing DVS and evaluate its effectiveness. The algorithm, named Incremental Static Voltage Adaptation (ISVA), uses the DVS technique and assigns differential voltages to each sub-task to minimize energy requirements of an application. Given the initial schedule, ISVA efficiently identifies tasks' relative importance and their liabilities on energy. It then achieves the best possible new schedule by observing its energy budget.

The rest of the paper is organized as follows. In Section 2, we present related work on scheduling and energy optimization techniques. In Section 3, we present task and energy models as well as existing scheduling algorithms. Section 4 describes the proposed energy constrained scheduling methodology. Section 5 discusses the testing methodology. In Section 6 we present our experimental results and give some concluding remarks in Section 7.

2 Related Work

Static scheduling of a parallel application on a multiprocessor system for minimizing the total completion time (or meeting deadlines) is a well-known problem that is known to be NP-complete. Researchers have devised a plethora of heuristics using a wide spectrum of techniques, including branch-and-bound, integer-programming, searching, graph-theory, randomization, genetic algorithms, and evolutionary methods [10]. Furthermore, there has been recent research on DVS scheduling algorithms for assigning tasks on parallel processors [1, 5, 16], but mostly for real-time systems where performance constraints are given. Recently, several DVS-based algorithms for slack allocation have been proposed for independent tasks and DAGs in a multiprocessor system [6, 10, 12, 14]. We have proposed static DVS scheduling algorithms for slack allocation on parallel machines and multi-core processors [7], outperforming other techniques, and in terms of time and memory requirements over Linear Programming based formulations for minimizing the energy presented in [15]. A few runtime approaches for slack allocation have

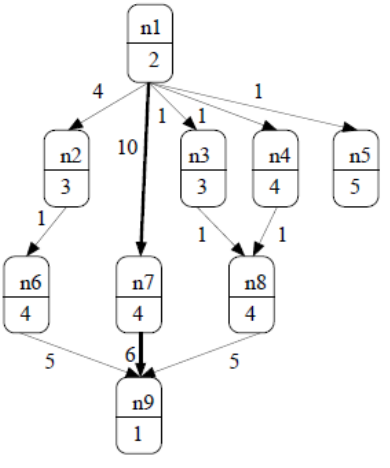

Fig. 1. Sample task graph with the critical path in bold

been studied in the literature for precedence-free tasks and for tasks with precedence relationships in a real-time system [1, 5, 10, 12, 16].

However, this work differs from existing research in that we approach the issue of energy efficiency from a different perspective. We consider an initial valid schedule and an energy constraint, and then optimize to reduce the increase in schedule length given that we have a limit on energy. Such a scenario is useful when one is working under energy budgets.

3 Task and Energy Model

The task model is a directed, acyclic graph (DAG) in which the nodes represent tasks, and the edges represent dependencies [9]. Tasks which have no predecessors are called entry nodes, and those which have no successors are called exit nodes. Additionally, each task has an associated computation cost, which is an estimate of the amount of time required to complete the task. There is also a communication cost for tasks with dependencies, which represents an estimate of the amount of time required for a task to send the results of its computation to a successor if the two tasks are scheduled to separate processors. In our model, we make the common assumption that when a task and its successor are scheduled to the same processor the communication cost becomes zero. Refer to Figure 1 for a sample task graph. Each task has a specific level in the graph, which is the maximum number of nodes from it to an entry node. Entry nodes have level 0. In the sample task graph in Figure 1, n4 is at level 1, and n9 is at level 3. The critical path of a task graph is the longest path from an entry node to an exit node, including communication costs. In Figure 1 the critical path (in bold) has length 23 and consists of nodes n1, n7, and n9.

The general problem of finding an optimal schedule is NP-complete, except for a few special cases [9]. Although several algorithms that create initial schedules based on task graphs have been proposed, this paper will focus on two of them: the traditional Level by Level algorithm (LBL) described later and the Dynamic Critical Path (DCP) algorithm described in [8]. A summary of these algorithms is presented below.

Dynamic voltage scaling (DVS) technique, available on most current and emerging processors, reduces the energy dissipation by dynamically scaling the supply voltage and the clock frequency of processing cores [3]. Mathematically, energy is simply a product of power and time. Power dissipation, P_d, is represented by $P_d = C_{ef} \cdot V_{dd}^2 \cdot f$, where C_{ef} is the switched capacitance, V_{dd} is the supply voltage, and f is the operating frequency. The relationship between the supply voltage and the frequency is represented by $f = k \cdot (V_{dd} - V_t)^2 / V_{dd}$, where k is a constant of the circuit and V_t is the threshold voltage. The energy consumed to execute task T_i, E_i, is expressed by $E_i = C_{ef} \cdot V_{dd}^2 \cdot c_i$, where c_i is the number of cycles to execute the task. The reduced supply voltage can decrease the processor speed in a linear manner, consequently reducing the energy requirement in a quadratic fashion [1].

3.1 LBL Algorithm

The LBL algorithm assigns tasks to processors level by level, so that all predecessors of a task have already been assigned to a processor when the task is being scheduled. If a

task has no predecessors, it will be scheduled to the processor that finishes first. If instead a task has one or more predecessors, then the *critical predecessor* is identified as the one with the maximum value of end time + communication cost, referred to later as the latest start time. If the critical predecessor is scheduled to the processor that finishes first, then the task is scheduled to that processor. Otherwise, two processors are considered for scheduling the task: the one that finishes first and the one containing the critical predecessor. The potential start time of the task is calculated for each of the two (taking into account end times and communication costs for all predecessors), and the task is scheduled to the processor which allows for an earlier start time.

The advantage of LBL is its simplicity and speed. It requires $O(v(v + p))$ time to complete, where v is the number of tasks, and p is the number of processors. This is because finding the critical predecessor has complexity $O(v)$ in a graph with v nodes, and locating the first ending processor has complexity $O(p)$, where p is the number of processors. In comparison, some similar algorithms described in [9] have complexities $O(v(v + e))$ or $O(e(v + e))$, and DCP has complexity $O(v^3)$. The disadvantage is that the schedules generated by LBL use a simple heuristic that yields suboptimal schedules. The pseudo-code for the LBL algorithm is shown in Figure 2.

```
for all levels in the graph do
  for all tasks t in the level do
    fep ← processor that finishes first
    if t has no predecessors then
      schedule t to fep
    else
      cpp ← processor containing the critical prede-
      cessor of t, the task with the largest value of
      end time + communication cost
      if cpp = fep then
        schedule t to fep
      else
        consider start time of t if it were scheduled
        to cpp or fep
        schedule t to the processor that would give
        it an earlier start time
      end if
    end if
  end for
end for
```

Fig. 2. Pseudo code for the LBL scheduling algorithm

3.2 DCP Algorithm

Rather than scheduling tasks level by level, DCP schedules tasks in their order of importance, which is determined by calculating the dynamic critical path. Furthermore, tasks' start times are not fixed until all tasks are scheduled, which allows for more flexibility in determining start times.

First, the absolute earliest start time (AEST) and the absolute latest start time (ALST) are calculated for each task. These values take into account the start and end times of

predecessors and successors and also the communication costs if tasks are scheduled to different processors. Tasks that are on the dynamic critical path have AEST = ALST. Such tasks are scheduled first, with priority going to those that have the smallest AESTs. AEST and ALST values are recomputed after each task is scheduled.

If a task is on the dynamic critical path, then the only processors considered as candidates for scheduling the task are those containing its predecessors and successors and one additional processor. Otherwise, if the task is not on the dynamic critical path, the only processors considered are those that already have tasks scheduled on them. Each processor in the list of candidates is considered to determine whether or not there is room for the task. If there is space, the task is tentatively scheduled there, and then the task's critical child is determined. This is the successor with the smallest difference between ALST and AEST (i.e., closest to the dynamic critical path). Then, a composite AEST is computed, which is the sum of the task's AEST on the candidate processor and the critical child's AEST on the same processor. The task will be scheduled to the processor which offers the best (i.e., the lowest) composite AEST. According to [8], the complexity of the DCP algorithm is $O(v^3)$. The advantage is that it produces much shorter schedules than many other scheduling algorithms. The corresponding pseudo-code is illustrated in Figure 3.

```
while not all nodes scheduled do
  n_i ← the highest node with the smallest difference
  between its ALST and AEST; break ties by choos-
  ing the one with the smaller AEST
  if n_i's ALST is equal to its AEST then
    call Select_Processor(n_i, On_DCP)
  else
    call Select_Processor(n_i, Not_On_DCP)
  end if
  Update AEST and ALST for all nodes
end while
Make all nodes' start time to be their respective
AESTs
```

Fig. 3. Pseudo code for DCP algorithm

4 ISVA Techniques

The Incremental Static Voltage Adaption algorithm (ISVA) is proposed as an energy-aware scheduling technique. The goal is to minimize as much as possible the schedule length degradation caused by having to execute tasks with a reduced energy budget. Dynamic voltage scaling is used to adjust the voltage levels at which individual tasks are executed.

The algorithm initially takes a DAG, the number of processors, the available voltage levels, and an energy budget. An initial schedule is created at the lowest voltage using a scheduling algorithm such as LBL or DCP. The energy consumed is computed and, if the energy budget has not been exceeded, the algorithm proceeds. The processor that finishes last in the generated schedule is identified, we call this processor the *critical processor*. A list L is constructed with the tasks in the critical processor. For each task

on list *L*, a list of parent tasks is constructed, which includes the task's predecessor. If a task has multiple predecessors, only the one with the latest end time is added to the list, we call this the *immediate predecessor* task. Once we have finished processing all tasks in list *L*, we move the contents of *L* to the beginning of list *E* and store the predecessor tasks onto list *L*. We repeat this process of processing tasks on list *L* until no tasks are available in *L*. This will happen when we reach the entry nodes of the DAG. From the list *E* of parent tasks, a task is selected which has the lowest voltage level and is as high as possible in the graph. If the selected task is not running at the maximum voltage level, then its voltage level is incremented. The task incremented is called the *candidate task*. If there is no candidate task for voltage adjustment, the algorithm will select the earliest starting task that is at the lowest voltage level from within all processors as the candidate task and adjust this one. The algorithm eventually stops when any task voltage adjustment would cause us to exceed the energy budget. The corresponding pseudo-code is illustrated in Figure 4.

```
for all tasks t_i ∈ T do
  Voltage(t_i) ←lowest voltage level
end for
Schedule tasks in T to processors in P
Recompute energy_consumed
if energy_consumed > available_energy then
  break // Not enough energy to run tasks
else
  while energy_consumed < available_energy do
    p_j ←Last_processor_to_finish(P) // Critical processor
    List L← All tasks in p_j
    List E ← φ
    while L ≠ φ do
      E ← L + E
      List M ← φ
      for all tasks t_k ∈ L do
        Task t_p ←Immediate_predecessor(t_k)
        Add t_p to M
      end for
      L ← M
    end while
    Task t_c ← Get_earliest_start_lowestVL_task(E)
    if t_c = null then
      t_c ← Get_earliest_start_lowestVL_task(T)
    end if
    if t_c ≠ null then
      Increment Voltage(t_c) by 1 level
      Schedule tasks in T to processors in P
    else
      break // No valid tasks for voltage adjustment remain
    end if
  end while
end if
```

Fig. 4. Pseudo code for ISVA algorithm

5 Testing Methodology

In order to test the ISVA algorithm, we propose a comparison of schedule length increase to energy budget used for a series of DAGs. The energy budgets used for the test ranged from 40% - 80% and the schedule of tasks to processors was generated by using the LBL and DCP algorithms. The ISVA algorithm was used in order to identify the order and tasks whose voltage levels would be adjusted. The method chosen to test this algorithm was to generate several task graphs based on a series of input parameters and to compare the results generated by using different scheduling algorithms, different voltage levels, and different energy budgets.

5.1 Workload Generation

In generating task graphs, there are two primary aspects to be considered, namely: the characteristics relevant to the graphs and the method for ensuring that graphs are acyclic. The first parameter is the number of nodes in the graph. It is important to test the algorithm on a wide range of graph sizes to ensure that it is scalable. Another important variable is the communication to computation ratio (CCR). Graphs which are communication-intensive will tend to have longer schedules because either processors will have longer idle time slots waiting for communication to complete or more tasks will be scheduled to a single processor in an attempt to avoid creating idle time slots.

The branching factor is the average number of successors per node. This value reflects the number of edges in the graph relative to the number of nodes. Graphs with large numbers of edges can be more limited in terms of the order in which nodes are scheduled, and this reduction in flexibility could result in an increase in the schedule length. The overall "shape" of the graph is described by α. For graphs with $\alpha = 1.0$, the shape will be approximately "square." That is, the graph will have $\sqrt{v}$ levels and $\sqrt{v}$ nodes per level, where v is the number of nodes. This concept of α is described in [8]. In general, the number of levels in the graph is equal to $\alpha\sqrt{v}$. For smaller values of α, the graph will have fewer levels, which will mean that more tasks may be executed in parallel, allowing for shorter schedules.

In addition to variables related to the graphs generated, there are certain parameters describing the hardware that could be used, and these parameters should also be tested. One of these is the number of processors (cores) used. Ideally, as the number of processors increases, the schedule length will decrease because more tasks will be executed simultaneously. The performance of a scheduling algorithm should scale with the number of processors. The final variable considered in these tests is the number of voltage levels available to the processors. The question is whether or not schedule length varies when processors have several voltage levels to choose from.

When one generates task graphs randomly, it is vital that graphs remain acyclic; otherwise, it will be impossible to honor the dependencies, and the algorithms will probably not complete as expected. In these tests, the graph generator was given four parameters: number of nodes, CCR, branching factor, and α. The number of levels was calculated first (using α), and a random number of nodes was chosen for each level, so that the total number of nodes was equal to that specified. Next, nodes were each assigned an identification number in ascending order by level. Thus, all nodes on the same level have consecutive IDs, and nodes with higher level numbers (closer to

the exit nodes) have larger IDs than nodes near the top of the graph (with lower level numbers). With this structure, maintaining an acyclic graph is simple: a node may not have a successor with ID less than its own ID or that is on the same level that it is.

5.2 Variable Values Used in Testing

In choosing the ranges of values to test, the goal is to try as wide a range as possible; however, time limitations constrained the addition of more values. A compromise has to be made between choosing several values and being able to complete the tests in a reasonable amount of time.

The ranges of values that are used for the graph variables are as follows: Number of nodes = {50, 100, 150}; Branching factor = {4}; CCR = {0.1, 1.0, 10.0}; α= {2.0}. The ranges of values that are used for the system variables are as follows: Number of voltage levels = {2, 3, 4, 5} and Number of processors = {Unbounded}.

The actual voltage levels ranged from 2 to 8V, and the value for each corresponding level was calculated based on how many voltage levels are being used. For example, with 2 voltage levels, we use {2V, 8V}, with 3 voltage levels, we use {2V, 5V, 8V}, and so on. The experiments were run on a 2.4 GHz PC.

6 Experimental Results

The results collected indicate that on average, running ISVA on DCP results in less schedule length degradation when compared to ISVA on LBL. Overall, using DCP as the scheduling algorithm resulted in 3-6% smaller schedules than using LBL. As expected, greater energy budgets results in shorter schedule length degradation when compared to lower energy budgets. Results show that operating at lower energy budgets has a small impact on schedule length degradation. For example, while an 80% energy budget produces a 20% schedule degradation, operating at half (40%) energy budget results in roughly an additional 10% in degradation increase. This can be explained from the linear relationship between energy and time, and the quadratic relationship between voltage level and energy.

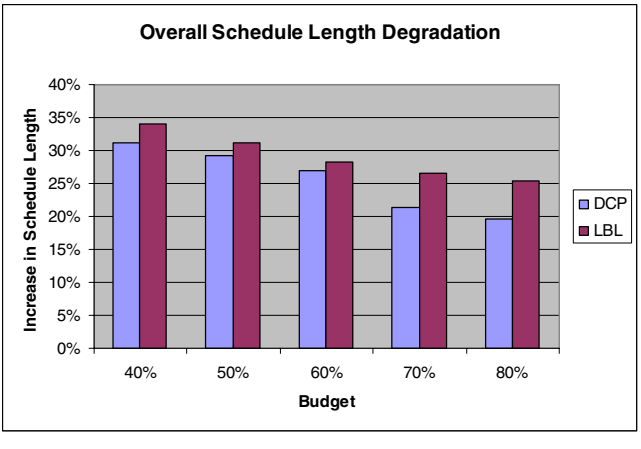

Fig. 5. Overall Schedule Length Degradation for ISVA on LBL and ISVA on DCP

Running at higher voltage levels increases power consumption quadratically but produces only a linear improvement in execution time. Results for various energy budgets are illustrated in Figure 5 with details of different energy budgets applied to task sizes of 50, 100, and 150 shown in Figure 6.

Furthermore, we observed no significant correlation between the number of voltage levels and the schedule length increase. However, operating with 4 voltage levels created significantly different results from the other configurations. Resulting schedule length increases using DCP and LBL variations of ISVA are shown with 2, 3, 4, and 5 available voltage levels in Figure 7.

As for the number of processors used by both algorithms, we noticed that ISVA on DCP used significantly less processors than ISVA on LBL and achieved better results. This is illustrated in Figure 8.

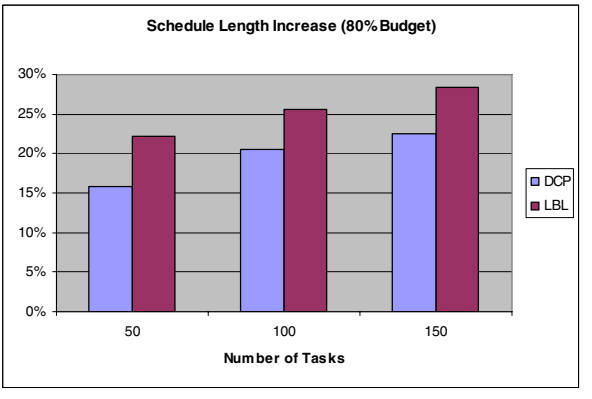

Fig. 6. Schedule length degradation of 50, 100, and 150 tasks with 80% energy

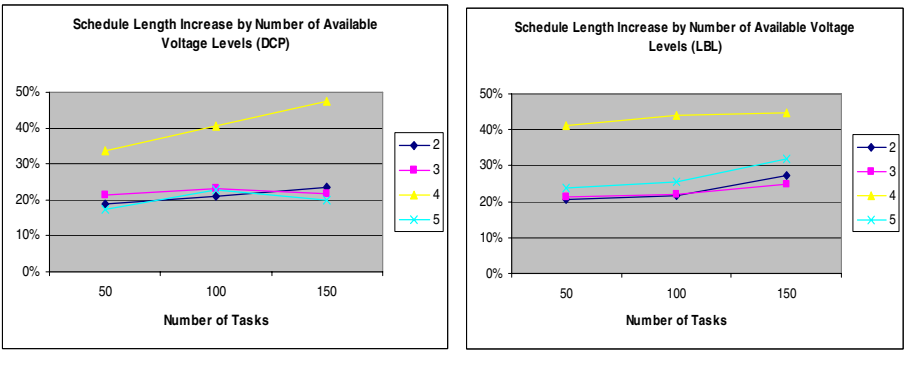

Fig. 7. Schedule length increase by number of available voltage levels

Finally, we analyzed the effect of CCR on schedule length degradation. As illustrated in Figure 9, the schedule length degradation was inversely proportional to the CCR value. The reason for this is that large CCR values tend to cause any schedule to

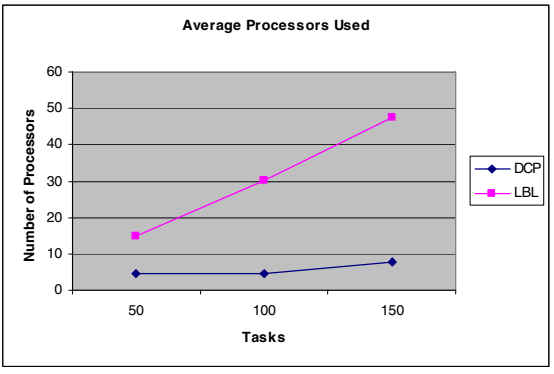

Fig. 8. Average number of processors used

be longer because there are large idle time slots for communication costs and larger numbers of tasks clustered on a single processor. This leaves more room for voltages to decrease, causing individual tasks to execute more slowly, without causing an overall increase in schedule length. In contrast, schedules that have lower communication costs are more easily parallelizable in the sense that tasks will require very little additional time to complete when they are on different processors than their predecessors. Such graphs will be more prone to increases in schedule length when task execution time increases, as tasks are assigned to lower voltage levels.

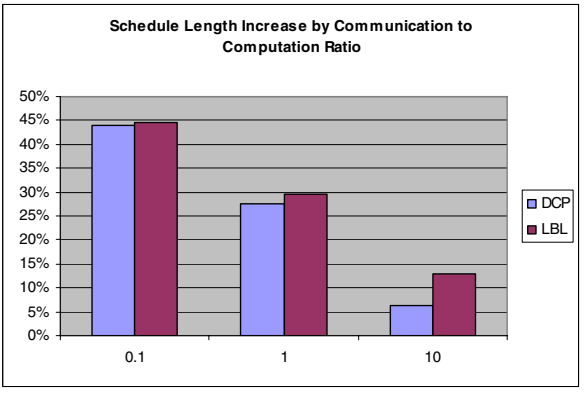

Fig. 9. Schedule length increase by communication to computation ratio

7 Future Research and Conclusions

Clearly, the ideal energy-aware algorithm would allow for reduced energy consumption while maintaining or only slightly increasing schedule length. Although this may seem like a goal unlikely to be achieved, there are some ways in which this algorithm might be improved. First, the algorithm works by choosing a task, incrementing its

voltage level, and then rescheduling all tasks. However, task scheduling tends to be a complex operation. Although it would not reduce the length of the schedule produced, a significant amount of computation time could be saved if several tasks were adjusted before rescheduling. Another possibility is to exploit idle time slots. Tasks whose execution is followed by an idle time slot may be able to run at lower voltage levels without increasing the total schedule length. This could occur if the task has no successor, or if its successor's start time is delayed waiting for results from another one of its predecessors.

The ISVA algorithm offers a spur for future research into energy-aware scheduling algorithms. Further testing and research needs to be conducted to design more efficient energy-aware scheduling algorithms that provide schedules of equal or only slightly greater length in comparison with non-energy-aware scheduling algorithms. Small decreases in voltage levels can have a larger impact on energy usage, and even small but consistent energy savings can accumulate over time. For this reason, it is important to continue searching for better energy-aware algorithms.

References

1. Ahmad, I., Khan, S.U., Ranka, S.: Using Game Theory for Scheduling Tasks on Multi-Core Processors for Simultaneous Optimization of Performance and Energy. In: Workshop on NSF Next Generation Software Program in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS 2008), Miami, FL (2008)
2. Aydin, H., Melhem, R., Moss, D., Meja-Alvarez, P.: Power-Aware Scheduling for Periodic Real-Time Tasks. IEEE Trans. on Computers 53(5), 584–600 (2004)
3. Burd, D., Pering, T.A., Stratakos, A.J., Brodersen, R.W.: Dynamic Voltage Scaled Microprocessor System. IEEE J. of Solid-State Circuits 35(11), 1571–1580, 473–484 (2000)
4. Gutnik, V., Chandrakasan, A.: An Efficient Controller for Variable Supply-Voltage Low Power Processing. In: IEEE Symposium on VLSI Circuits, pp. 158–159 (1996)
5. Jejurikar, R., Gupta, R.: Dynamic Slack Reclamation with Procrastination Scheduling in Real-Time Embedded Systems. In: Design Automation Conf., pp. 111–116 (2005)
6. Kang, J., Ranka, S.: Dynamic Algorithms for Energy Minimization on Parallel Machines. In: Euromicro Intl. Conf. on Parallel. Distributed and Network-based Processing (PDP 2008) (to appear, 2008)
7. Khan, S.U., Ahmad, I.: A Cooperative Game Theoretical Technique for Joint Optimization of Energy Consumption and Response Time in Computational Grids. IEEE Transactions on Parallel and Distributed Systems 20(3), 346–360 (2009)
8. Kwok, Y.-K., Ahmad, I.: Dynamic Critical-Path Scheduling: An effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Trans. Parallel Distributed. Systems 7(5) (1996)
9. Kwok, Y.-K., Ahmad, I.: Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors. ACM Computing Surveys 31(4), 406–471 (1999)
10. Luo, J., Jha, N.K.: Power-Conscious Joint Scheduling of Periodic Task Graphs and Aperiodic Tasks in Distributed Real-time Embedded Systems. In: Int. Conf. on Computer-Aided Design, pp. 357–364 (2000)
11. Luo, J., Jha, N.K.: Static and Dynamic Variable Voltage Scheduling Algorithms for Real-time Heterogeneous Distributed Embedded Systems. In: ASP-DAC 2002: Proceedings of the 2002 conference on Asia South Pacific Design Automation/VLSI Design, p. 719. IEEE Computer Society, Washington (2000)

12. Schmitz, M.T., Al-Hashimi, B.M.: Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. In: Int'l Sym. on System Synthesis, pp. 250–255 (2001)
13. Schmitz, M.T., Al-Hashimi, B.M.: Considering Power Variations of DVS Processing Elements for Energy Minimisation in Distributed Systems. In: Int'l Sym. on System Synthesis, pp. 250–255 (2001)
14. Yu, Y., Prasanna, V.K.: Power-Aware Resource Allocation for Independent Tasks in Heterogeneous Real-Time Systems. In: 9th IEEE International Conference on Parallel and Distributed Systems (2002)
15. Zhang, Y., Hu, X. (Sharon) ., Chen, D.Z.: Task Scheduling and Voltage Selection for Energy Minimization. In: Design Automation Conf., pp. 183–188 (2002)
16. Zhu, D., Melhem, R., Childers, B.R.: Scheduling with Dynamic Voltage/Speed Adjustment Using Slack Reclamation in Multiprocessor Real-Time Systems. IEEE Trans. on Parallel and Distributed Systems 14(7), 686–700 (2003)