Proceedings of the 18th IASTED International Conference
MODELLING AND SIMULATION
May 30–June 1, 2007, Montreal, Quebec, Canada
ISBN Hardcopy: 978-0-88986-663-8 / CD: 978-0-88986-664-5

A SIMULATION OF POWER-AWARE SCHEDULING OF TASK GRAPHS TO MULTIPLE PROCESSORS

Xiaojun Qi, Carson Jones, and Scott Cannon
Computer Science Department
Utah State University, Logan, UT, USA 84322-4205
xqi@cc.usu.edu, carsonj@cc.usu.edu, Scott.Cannon@usu.edu

ABSTRACT
Power-aware scheduling has been of great interest for systems whose energy consumption needs to be minimized. In this paper, we improve a voltage-scaling-based power-aware scheduling algorithm to reduce the task's energy consumption at the cost of a slower execution rate. The improved algorithm allows multiple scaling voltage levels of individual tasks in a task precedence graph and attempts to maximize the amount of energy saved while still meeting a deadline constraint. Five bounded number of processor scheduling algorithms are used as the basis for this improved power-aware scheduling. Other sophisticated scheduling algorithms can be easily embedded in our improved power-aware scheduling algorithm to reduce the energy consumption. We use the simulation program AnyLogic™ to implement an easy-to-use drag-and-drop interface for building large-scale task graphs and running various simulations. The simulation results demonstrate that our proposed scheduling can reduce the energy consumption and achieve more energy savings than the static voltage scaling step alone. In addition, our simulation tool also provides an efficient and effective means for building task graphs and viewing the scheduling results in the form of the Gantt chart. This simulation quickly facilitates the testing of the validity of a problem and its outcomes and greatly fosters learning.

KEY WORDS
Scheduling, task graph, power reduction, multiprocessors, and voltage scaling

1. Introduction

There has been much research in the area of multiprocessor scheduling for power constrained and real-time systems. The objective is to schedule all tasks in the system such that all precedence constraints are observed while ensuring the whole schedule will meet a specified deadline. If there is ample time between the end of the schedule and the deadline requirement, the voltage levels of selected tasks can be scaled down to save the energy consumed by the system. That is, the selected tasks will be scheduled to run at a slower execution speed. This tradeoff of processing power and execution speed has recently been extensively studied. In general, processor power consumption is proportional to the square of the voltage used [1, 2]. In addition, this voltage scaling ability of processors is becoming widely available and is proving to be a viable solution for the energy minimization problem [2].

Scheduling directed acyclic task graphs in even simple cases is NP-complete and a polynomial optimal scheduling algorithm exists for only a few simple cases [3]. As a result, many scheduling algorithms propose heuristics for finding reasonable time schedules, which is the class of algorithms our simulation addresses. In general, task graph scheduling algorithms can be divided into four major classes including Bounded Number of Processor (BNP), Unbounded Number of Clusters (UNC), Arbitrary Processor Network (APN), and Task Duplication Based (TDB) algorithms. BNP-based task graph scheduling algorithms are designed for a limited number of processors and are the simplest class. UNC algorithms try to maximize the parallelism of an arbitrary number of processors to minimize the schedule length. APN algorithms assume a network of processors and have more focus for communication. TDB algorithms duplicate tasks on multiple processors with an attempt to reduce communication costs, and may not be useful for energy minimization systems. A thorough analysis of each of these classes of algorithms and extensive background information are provided in [3]. In this research, we will build our power-aware scheduling upon BNP-based task scheduling algorithms due to the impracticality and complexity of the other systems.

More recently, task graph scheduling with respect to power-aware systems and heterogeneous processor systems have been studied. Roychowdhury et al. [4] propose a voltage scaling algorithm. Jiong and Niraj [5] consider the scheduling of periodic task graphs in conjunction with aperiodic tasks. Zhu et al. [2] propose a method for voltage scaling with slack reclamation for real-time systems. Topcuoglu et al. [6] introduce two effective static scheduling algorithms for heterogeneous processor systems. In this project, we extend the voltage scaling algorithm [4] to permit as many as eight levels and incorporate five BNP algorithms to find an order of the tasks on the processors before deploying the voltage

scheduling. These five BNP algorithms include Modified Critical Path (MCP) algorithm [7], Highest Level First with Estimated Times (HLFET) algorithm [8], Earliest Time First (ETF) algorithm [9], Dynamic Level Scheduling (DLS) algorithm [10], and Insertion Scheduling Heuristic (ISH) algorithm [11]. In addition, we present an efficient and effective simulation tool for building task graphs and viewing the results of five scheduling algorithms in the form of a Gantt chart. This simulation not only quickly facilitates the testing of the validity of a problem and its outcomes, but also greatly fosters learning. The implemented simulation system provides a powerful and convenient means for the user to study the performance of the voltage scaling algorithm. The remainder of the paper is organized as follows. Section 2 introduces our improved voltage-scaling-based power-aware scheduling method applied on five BNP scheduling algorithms. Section 3 demonstrates the simulation results. Section 4 draws conclusions and discusses the future work.

2. The Scheduling Algorithm

We build our simulation upon the voltage scheduling method [4] and extend the algorithm to allow as many as eight voltage levels to scale the tasks on five BNP scheduling algorithms. The order of the tasks can be represented as a directed acyclic task graph $G = \{V, E\}$ where V is a set of tasks and E is a set of edges. Fig. 1 shows a typical example of a task graph where each task is a set of instructions that must be executed on a single processor without preemption. Associated with each task is a computational cost, which defines the maximum time required for the task to finish its execution under the worst case conditions [4]. Each task finishes at or before its specified computation time. Each edge $e_i = (v_j, v_k)$ is a directed edge that communicates from task v_j to its successor task v_k, when v_j has finished its execution. The communication cost for each edge is assumed to be zero in our implementation. Tasks with no predecessors are entry tasks. Tasks with no successors are exit tasks. The topological ordering of tasks assures that any task v_k cannot begin its execution until all of its predecessors have communicated their completion.

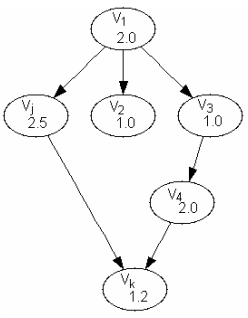

Fig 1: Example task graph

Our voltage-scaling-based power-aware scheduling algorithm is built upon a task ordering, which has already been established on each processor. That is, a scheduling algorithm has already been applied to the task graph for determining the execution order of each task. In general, any scheduling algorithm, which minimizes the schedule length, is suitable for keeping the energy consumption of the entire system low. In our system, we implemented five BNP scheduling algorithms to impose an order of the tasks on the processors. These five BNP algorithms are known as static scheduling algorithms because all task precedence and computation constraints are known as a priori, and the algorithms are applied before runtime [3]. The rationale to choose these five algorithms is that they use different heuristics to produce a schedule with a reasonable schedule length. Table 1 summarizes the heuristic steps, namely, prioritize tasks, select the best task for scheduling; and select the best processor for scheduling [3], used by each algorithm. Several terms are defined here to aid in understanding the prioritization method of each algorithm. The Static Level (SL) of a task n_i, is the longest path from n_i to an exit task. A Critical Path (CP) is the longest path, from an entry task to an exit task, in the task graph. A task's As-Late-As-Possible (ALAP) time, is defined as the critical path length minus the task's SL. The Earliest Start Time (EST) is the earliest time a task may execute, under the condition that all of its predecessor tasks have finished execution. A task's Dynamic Level (DL) is the difference between the task's SL and the EST of the task on a processor.

Table 1: Heuristics of the five BNP algorithms

Algorithm	Brief Outline of the Scheduling Method
MCP	Tasks prioritized by *As-Late-As-Possible Time* Priorities computed *once* Tasks scheduled to processors by *possibly* using insertion
HLFET	Tasks prioritized by *Static Level* Priorities computed *once* Tasks scheduled to processors *without* using insertion
ETF	Tasks prioritized by *Earliest Start Time* Priorities computed *at each scheduling step* Tasks scheduled to processors *without* using insertion
DLS	Tasks prioritized by *Dynamic Level* Priorities computed *at each scheduling step* Tasks scheduled to processors *without* using insertion
ISH	Tasks prioritized by *Static Level* Priorities computed *once* Tasks scheduled to processors by *possibly* using insertion

After a task ordering has been imposed by one of these five BNP algorithms, our improved voltage-scaling-based power-aware scheduling algorithm is applied to minimize the energy consumption. This algorithm has

two stages: a static voltage scheduling stage and a dynamic resource reclamation stage. The static voltage scheduling assigns each task a voltage setting based on its worst-case computation time. The dynamic resource reclamation improves energy savings by readjusting the voltages of the tasks based on whether the actual runtimes of the executed tasks are significantly better than their worst case times. This improvement results from the fact that each task may finish prior to its worst-case computation time.

Algorithm *StaticVoltageScheduling* (*G*, *S*, V)
Input: Task graph *G*, processor schedules *S*, set of voltage levels *V*
Output: Task graph G with updated voltage values and commit times for all tasks

$V \{ V_{lo}, V_2, V_3, V_4, V_5, V_6, V_7, V_{hi} \} \leftarrow V$;
Add precedence constraints to *G* for adjacent tasks on each processor in *S;*
for each task *e* in *G*
Set the voltage level of *e* to the lowest voltage, V_{lo};
endfor
MissedPaths ← Find all paths that miss the deadline under the current voltage assignments;
while *MissedPaths* is not empty
for each task *e* of path *p* in *MissedPaths*
increment *e*'s *weight* based on its appearance in *p* and *MissedPaths* (that is, the *weight* determines the priority for scaling the task)
endfor
taskID ← task in any of the *MissedPath*s with the maximum *weight;*
pathID ← path in *MissedPaths*, which contains *taskID*, missing the deadline by the greatest amount;
Scale the voltage level of *taskID* to the next highest voltage setting in *V;*
if *taskID*'s voltage is V_{hi}
set *taskID*'s *weight* to ∞ so it is never considered again;
endif
Remove any path from *MissedPaths* that now meets the deadline;
for each task *e* in *MissedPaths*
if *e*'s *weight* is less than ∞
e's *weight* ← 0;
endif
endfor
endwhile
Set *commit* times for all tasks in *G;*

Algorithm *DynamicResourceReclamation* (*G*, *T*, *V*)
Input: Task graph *G*, finished task *T*, set of voltage levels *V*

$V \leftarrow V\{ V_{lo}, V_2, V_3, V_4, V_5, V_6, V_7, V_{hi} \}$;
/**Note that precedence constraints from schedules are still applied **/
AheadTime ← *T*'s *commit* time – *T*'s actual finish time;
for each task *e* of *T*'s successors
scale *e* to the lowest voltage that allows for the execution time to be at most *AheadTime* + *e*'s *commit* time;
Update the *start* and *commit* time of *e*;
endfor

Fig. 2: Pseudocode for the voltage scaling algorithm

Fig. 2 summarizes the algorithmic view of our expanded power-aware scheduling algorithm. The *StaticVoltageScheduling* method is called after the schedule has been imposed on the processors. The *DynamicResourceReclamation* method is called as each task finishes. It is assumed that each task is assigned one voltage level.

In the following simulations, the runtime computational cost of each task is randomly computed. To ensure a fair comparison, all 5 BNP-based voltage-scaling-based power-aware algorithms use the same set of run-time values.

3. Simulation and Experimental Results

3.1 Simulation Environment

We implement our simulation within the Java™-based development environment of the Anylogic™ simulation toolkit by the XJ Technologies Company. We build a user-friendly interface to allow users to drag-and-drop the tasks and connect each task by a simple mouse click. In addition, a property window permits users to set the required parameters for simulation by either using the default configuration or typing a desired value to their interest. A sample property window is shown in Fig. 3.

Parameters:

Name	Value
nProcessors	**4**
algorithm	**Modified Critical Path...**
useDefaultAnimation	true
powerAwareScheduling	**true**
numVoltageLevels	**8**
Vlo	2.0
V2	2.0
V3	2.0
V4	2.0
V5	2.0
V6	2.0
V7	2.0
Vhi	3.3
deadline	**15**

Fig. 3: Property window for simulation

3.2 Simulation Results

In this section, we will illustrate a variety of simulation results from different perspectives. Fig. 4 demonstrates an example task graph created by our simulation tool where each circle n_i represents each task, the number within each circle represents the worst case computation cost prior to the execution, and the small square boxes on the left and right hands of the circle ensure the connection between two tasks and enforce precedence constraints (i.e., the right hand square box of a task, the predecessor, will be connected to the left hand square box of another task, the successor).

This task graph will be used as a common base to run different power-awareness scheduling algorithms, which use five previously mentioned scheduling algorithms, namely, MCP, HLFET, ETF, DLS, and ISH, with six

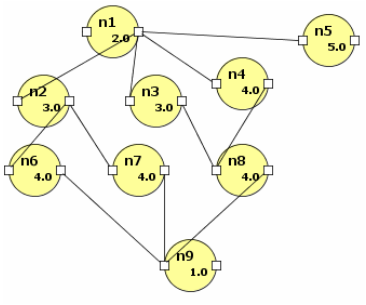

Fig. 4: The input graph

voltage levels. The values for each of the voltage levels are 1.8V, 2.1V, 2.4V, 2.7V, 3.0V, and 3.3V. For each simulation, we set a schedule deadline of 15.0 time units and use three processors for the scheduling. Fig. 5 shows the actual computation costs of each task at runtime, which will be used in the simulations for the dynamic resource reclamation component.

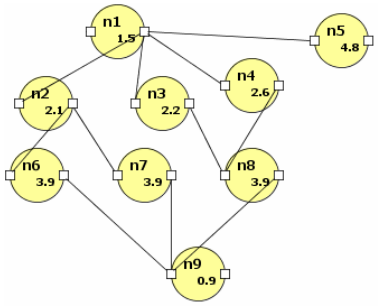

Fig. 5: The runtime graph

To compute task execution times at a specified voltage level, the following method is used. Tasks that are running at the highest possible voltage, V_{hi}, are assumed to take one unit of execution time for each time unit elapsed. Tasks that are running at a lower voltage, V_i, take $slow(V_i)$ units of execution time per time unit as defined by the following equation,

$$slow(V_i) = v_i/v_{hi}(v_{hi}/v_i)^2 \quad (1)$$

This equation determines the factor by which a task running at V_i is slowed down, compared to running at V_{hi} [4]. This is useful for simulation and interpretation purposes.

Final simulation results of our power-awareness scheduling algorithm applied with the five BNP scheduling algorithms are shown in Fig. 6 by Gantt chart. Two Gantt charts are associated with each algorithm. Specifically, the charts on the left-hand side represent the schedules after the static voltage scheduling, and the charts on the right-hand side represent the schedules after the dynamic resource reclamation. On the *y*-axis of each chart, PE*i* represents the schedule for processor numbered *i*. The *x*-axis shows the timeline for the tasks. Within each chart, the rectangular colored shapes represent each task as it resides in the schedule, with the task's name inside the rectangle. The color and height of each task represents the voltage level assigned to the task. The taller and darker-colored tasks represent tasks that have a higher assigned voltage. The shorter and lighter-colored tasks represent tasks that have a lower assigned voltage. The deadline constraint is drawn as a bold line over the schedule at time 15.0. A color legend which gives the specific voltage level for each task is shown on the right of each chart.

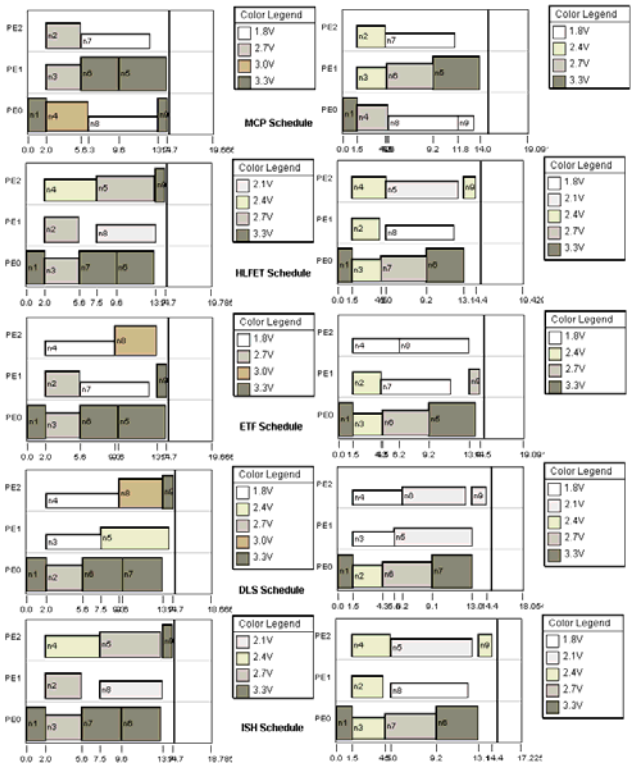

Fig. 6: Gantt chart outputs for each of the scheduling algorithms

The energy savings and schedule lengths of each of the schedules are shown in Table 2. Included in the table are the Schedule Length (SL) and Energy Savings (ES) for both the static and the dynamic components of the algorithm. **It should be noted that the values in the dynamic column represent the final values for the schedule length and the energy savings**. According to Table 2, the DLS algorithm outperforms the others with respect to energy savings. The ETF algorithm performs the second best followed by ISH, HLFET, and MCP.

Table 2: Simulation results for task graph shown in Figs. 4 and 5

Algorithm	Static Voltage Scaling		Dynamic Resource Reclamation	
	SL	ES (%)	SL	ES (%)
MCP	14.73	16.9	14.09	24.4
HLFET	14.79	15.1	14.42	24.6
ETF	14.73	16.9	14.51	26.2
DLS	14.73	18.1	14.46	26.9
ISH	14.78	15.1	14.29	24.6

To further our analysis, we also tested our power-awareness algorithm applied to these five algorithms on five additional task graphs with a varying number of

processors, voltage levels, and tasks. The average SL and ES are shown in Table 3. The average deadline for these graphs is 19.5.

Table 3: Average simulation results for 5 task graph experiments

Algorithm	Static Voltage Scaling		Dynamic Resource Reclamation	
	Ave. SL	Ave. ES (%)	Ave. SL	Ave. ES(%)
MCP	18.73	19.64	17.2	22.1
HLFET	19.48	20.56	17.5	24.48
ETF	19.1	20.56	17.4	24.06
DLS	19.2	20.58	17.5	24.42
ISH	19.2	20.88	17.7	24.06

It is observed that the HLFET and the DLS algorithms seem to be the best on average and MCP falls quite a bit behind the others. The reason for this may be because the MCP algorithm tends to densely schedule critical tasks on one processor and sparsely schedule noncritical tasks on other processors, when considering where to place a task. This makes it difficult to scale down the voltages and stretch out tasks that are on the densely scheduled processor. Consequently, it is difficult to reduce energy consumption without exceeding the deadline. On the other hand, the DLS and HLFET algorithms tend to schedule tasks evenly across all processors with large gaps, and therefore yield better energy savings. Overall, the proposed power-aware schedule algorithm did reduce the average SL and ES for all the five BNP-based scheduling algorithms. In addition, the two-step algorithm (i.e., static voltage scaling plus dynamic resource reclamation) achieves more improvement over the static voltage scaling step alone.

More extensive tests have been also performed to show the robustness of the proposal power-aware scheduling algorithm on more complicated task graphs with a varying number of processors, voltage levels, tasks, and deadlines. Table 4 and Table 5 list the SL and ES of each of the five schedules for static voltage scaling and dynamic resource reclamation on 10 more task graphs with different complexities and deadlines, respectively. Table 6 summarizes the average simulation results. In these three tables, 6 to 15 tasks as shown in the first column of each table are used in the experiments. These experiments further prove the proposed power-aware schedule algorithm can reduce the average SL and ES for all the five BNP-based scheduling algorithms. In addition, the two-step algorithm achieves more improvement over the static voltage scaling step alone.

Our simulation tool also provides three convenient and integrated visualization tools to display the schedule, simulate the task execution, and therefore interpret the results of the scheduling algorithm from different perspectives. Fig. 7 demonstrates these three visualization tools with an animation of the input graph showing the current execution of the tasks together with a detailed textual output on the left and a Gantt chart animation on the right. It shows an example of a simulation that doesn't use the power-aware scheduling. The green and orange tasks in the task graph animation (center) represent tasks currently executing on the respective processors in the Gantt chart (right), the gray colored tasks represent tasks having finished execution, and the manila colored tasks represent tasks having not yet been executed. The textual output (left) gives the specified start and end times of each task.

Table 4: Simulation results for 10 task graphs using static voltage scaling

	MCP		HLFET		ETF		DLS		ISH	
	SL	ES	SL	ES	SL	ES	SL	ES	SL	ES
6	10	9.0	10	9.0	10	9.0	10	9.0	10	9.0
7	10.3	22.3	10.3	18.0	11.9	10.9	10.6	18.0	10.6	18.0
8	13.4	22.3	14.6	22.0	14.6	22.0	14.6	22.0	14.6	22.0
9	10.6	23.3	12	25.2	12	25.2	12	25.2	12	25.2
10	15.4	30.5	16.4	29.7	16.4	29.7	16.4	29.7	16.4	29.7
11	11.3	29.7	11.8	26.9	11.9	26.2	12	28.5	12	28.5
12	11.9	23.3	11.7	20.5	11.9	21.9	11.9	22.2	11.9	20.4
13	16.9	20.4	16.6	19.1	16.6	20.4	16.6	20.4	16.6	20.4
14	11.9	21.6	11.9	18.4	12	18.8	11.9	22.3	11.9	18.4
15	12.9	27.4	12.9	27.5	12.8	24.4	13	27.7	13	25.0

Table 5: Simulation results for 10 task graphs using dynamic resource reclamation

	MCP		HLFET		ETF		DLS		ISH	
	SL	ES	SL	ES	SL	ES	SL	ES	SL	ES
6	7.7	10.3	7.8	10.1	8.8	7.3	7.7	10.3	7.8	10.1
7	10.4	22.1	10.7	19.5	10.8	11.6	9.5	17.1	10.4	17.1
8	12.9	22.2	13.7	30.5	13.4	25.2	13.6	25.6	14.3	23.9
9	10.4	23.9	11.4	27.4	11.4	27.4	10.7	26.5	10.7	26.5
10	15.9	31.6	15.7	30.1	17.1	31.5	15.7	30.1	15.7	30.1
11	11.3	31.3	11.2	31.9	11.1	29.5	11.7	32.5	11.4	31.0
12	10.9	25.9	10.9	23.6	10.5	25.7	11.2	24.8	10.9	25.4
13	16.2	22.0	15.5	21.1	15.3	22.2	15.2	23.1	15.3	23.1
14	10.9	24.3	11.2	23.7	11	22.7	10.6	24.3	11.2	21.9
15	12.1	31.3	11.7	29.3	12.0	26.2	11.6	29.3	11.3	27.6

Table 6: Average simulation results for 10 task graph experiments

Algorithm	Static Voltage Scaling		Dynamic Resource Reclamation	
	Ave. SL	Ave. ES (%)	Ave. SL	Ave. ES(%)
MCP	12.46	22.98	11.87	24.59
HLFET	12.82	21.63	11.98	24.72
ETF	13.01	20.85	12.14	22.93
DLS	12.9	22.5	11.75	24.36
ISH	12.9	21.66	11.9	23.67

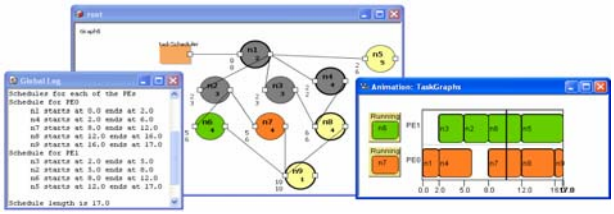

Fig. 7: Three visualization tools: Textual output (left), task graph animation (center), and Gantt chart animation (right).

4. Conclusion and Future Work

In this paper, we have shown the results of an improved voltage-scaling-based heuristic power-aware algorithm, in conjunction with five static scheduling algorithms, as they are applied to task precedence graphs. We implemented our simulation within the Java™-based development environment of the Anylogic™ simulation toolkit by the XJ Technologies Company. Our extensive simulation results show the proposed power-aware schedule algorithm is capable of reducing the average schedule length and energy savings for all the five BNP-based scheduling algorithms. In addition, the two-step algorithm (i.e., static voltage scaling plus dynamic resource reclamation) is able to achieve more improvement over the static voltage scaling step alone. Furthermore, our proposed power-aware algorithm can be easily plugged into any static scheduling algorithm to improve its energy savings and schedule length. Our Anylogic™ based simulation environment provides an efficient and effective tool for building task graphs and viewing the scheduling results in the form of a Gantt chart. This simulation tool not only quickly facilitates the testing of the validity of a problem and its outcomes, but also greatly fosters learning.

Some possibilities for future work include adding more voltage scheduling algorithms with which we could compare the current implementation. Also, a future avenue could also investigate simulating task graphs on heterogeneous processor systems. Or possibly a combination of scheduling on heterogeneous systems that are also power-aware.

Acknowledgement

This work is supported by the grant from the Air Force Research Laboratory.

References

[1] V. Kianzad, S. S. Bhattacharyya, & G. Qu, CASPER: an integrated energy-driven approach for task graph scheduling on distributed embedded systems, *Proc. Int. Conf. on Application-Specific Systems and Architecture Processors*, 2005, 191-197.

[2] D. Zhu, R. Melhem, & B. Childers, Scheduling with dynamic voltage/speed adjustment using slack reclamation in multi-processor real-time systems, *Real-Time Systems Symposium*, 2001.

[3] Y. K. Kwok & I. Ahmad, Static scheduling algorithms for allocating directed task graphs to multiprocessors, *ACM Computing Surveys*, 31(4), 1999, 406-471.

[4] D. Roychowdhury, I. Koren, C. Krishna, & Y. H. Lee, A voltage scheduling heuristic for real-time task graphs, *Proc. Int. Conf. on Dependable Systems and Networks*, 2003, 741-750.

[5] L. Jiong & J. Niraj, Power-conscious joint scheduling of periodic task graphs and aperiodic tasks in distributed real-time embedded systems, *Proc. Int. Conf. on Computer Aided Design*, 2000, 357-364.

[6] H. Topcuoglu, S. Hariri, & M. Y. Wu, Performance-effective and low-complexity task scheduling for heterogeneous computing, *IEEE Trans. on Parallel and Distributed Systems*, 13(3), 2002, 260-274.

[7] M. Y. Wu, MCP: modified critical path algorithm, UNM EECE Department, June 2006. <http://www.eece.unm.edu/~wu/mcp/mcp.pdf>

[8] T. L. Adam, K. M. Chandy, & J. R. Dickson, A comparison of list scheduling for parallel processing systems, *ACM communications*, 17(12), 1974, 685-690.

[9] J. J. Hwang, Y. C. Chow, F. D. Anger, & C. Y. Lee, Scheduling precedence graphs in systems with interprocessor communication times, *SIAM J. on Computing*, 18(4), 1989, 244-257.

[10] G. C. Sih & E.A. Lee, Declustering: a new multiprocessor scheduling technique, *IEEE Trans. on Parallel and Distributed Systems*, 4(6), 1993, 625-637.

[11] B. Kruatrachue, & T. G. Lewis, Duplication scheduling heuristic, a new precedence task scheduler for parallel systems, Technical Report 87-60-3, Oregon State Univ., 1987.