Low-Power Dynamic Scheduling in Heterogeneous Systems

Saumya Uppaluri, Baback Izadi and Damu Radhakrishnan
Department of Electrical and Computer Engineering
State University of New York - New Paltz
75 South Manheim Blvd.
New Paltz, NY 12561, U.S.A.
uppalu52@newpaltz.edu, bai@engr.newpaltz.edu,
damu@engr.newpaltz.edu

Abstract: *This paper develops a matching and scheduling algorithm that accounts for both the execution time and the power consumption of the application. The power consumption of different processors on a variety of tasks is taken into account using a proposed incremental cost function, which is utilized in a low-power dynamic-level scheduling algorithm. This cost function is independent of the topology and can be adjusted to the desired level in such a way that the power consumption of the tasks would effect scheduling.*

Keywords**:** Heterogeneous computing, precedence constrained tasks, matching, scheduling, low power.

1. Introduction

A heterogeneous distributed computing system is a suite of diverse high-performance machines interconnected by a high-speed network. They have promising high-speed processing of computationally intensive applications with diverse computation needs. One of the challenges in heterogeneous computing is to develop matching and scheduling algorithms that assign the tasks of the application to the processors [1, 2]. Therefore, researchers have proposed many static, dynamic and even hybrid algorithms to minimize the execution time of applications running on a heterogeneous system [3, 4, 5, 6, 7, 8]. Another challenge facing distributed computing community has become power consumption of parallel machines [9]. A study by Argonne National Laboratory has indicated that a 2.5 petaflop supercomputer, made of over a hundred thousand CPUs, will be available by 2010. The study predicts that such a system will cost $16 million and would require 8 mega watts of power to operate at a cost of about $8 million per year. Hence, soaring energy prices and rising concern about the environmental impact of electronics systems highlight the importance of incorporating low power design schemes at all levels of such systems.

At the system level, some researchers have examined energy efficient scheduling algorithms to reduce power consumption [10, 11, 12, 13, 14]. Static techniques, such as synthesis and compilation for low power, are applied at design time [15]. In contrast, dynamic techniques use runtime behavior to reduce power when systems are serving light workloads or are idle [16]. For example, one such scheme is dynamic voltage scaling (DVS), where the supply voltage at runtime is changed as a method of power management. Reducing the supply voltage is an effective way of reducing the power consumption, but we pay a speed penalty for voltage reduction. The impact of reducing V_{dd} on the delay is shown in Figure 1 [17] for a variety of different logic circuits, ranging in size from 56 to 44,000 transistors and spanning a variety of functions. The figure illustrates that on a single processor, the circuit-delay degrades by a factor of two when the supply voltage is dropped from 5V to 3V. This reduction in supply voltage reduces the energy usage by a factor of 2.89 [17]. Figure 2 depicts power versus delay for the circuits in Figure 1. Figures 1 and 2 demonstrate that as we reduce voltage, the circuit's power and energy usage decrease and the circuit's delay increases. Moreover, Figure 2 shows that the required energy

for a circuit would be different depending on power or delay requirement. Therefore, one can specify the maximum tolerated power and determine the required delay within the circuit and vice versa.

At the system level, in a homogeneous system, different tasks would utilize a combination of different circuits and therefore exhibit similar characteristics to Figure 2. Hence, such a characteristic per task for a processor can be measured. In a heterogeneous system, each task exhibits different power-delay characteristics per processor.

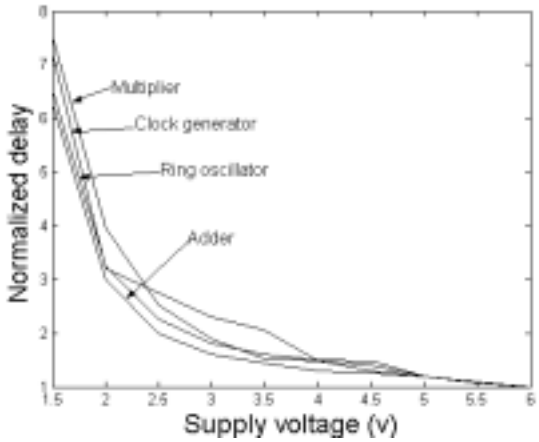

Figure 1: Normalized delay versus supply voltage for various experimental circuits.

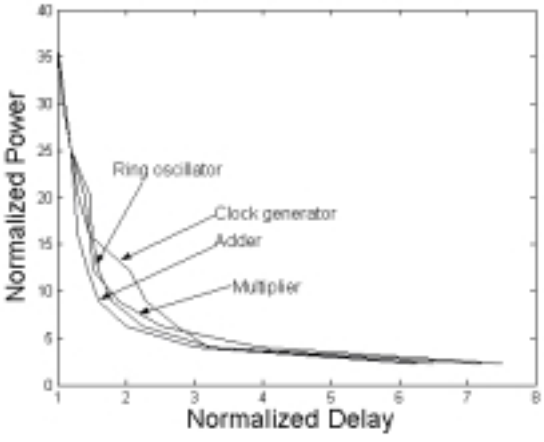

Figure 2: Normalized power versus normalized delay for various experimental circuits.

In this paper, we consider scheduling tasks in a heterogeneous distributed system, where processors have different speed and power characteristics. Our scheduling algorithm uses an efficient cost function, which, in addition to execution time, considers the power consumption of tasks per processor; we assume that power-delay characteristics are known for each task within a processor.

The rest of the paper is organized as follows. An overview of an existing Dynamic Level Scheduling algorithm is given in Section 2. The Low power Dynamic Level Scheduling algorithm is explained in Section 3. Section 4 evaluates the effects of the proposed low-power cost function on the scheduling heuristic. Finally, concluding remarks are given in Section 5.

2. DLS Algorithm

In this section, for completeness, we give a brief overview of the dynamic level scheduling (DLS) algorithm proposed by Sih and Lee in [18]. Let us consider M heterogeneous processors and N tasks of an application. The DLS algorithm is a compile time, static list scheduling heuristic. It allocates a Directed Acyclic Graph (DAG) structured application to a set of heterogeneous machines to minimize the execution time of the application. The algorithm also considers the interprocessor communication overheads when mapping precedence graphs onto multiple processor architectures. At each scheduling step, the DLS algorithm chooses the next task to schedule and the processor on which the task is to be executed. This is done by finding the ready task and processor pair that have the highest dynamic level. The dynamic level of a task-processor pair is given by the cost function specified by Equation 1. Each term in the cost function is expressed in time units. The terms collectively specify the priority of executing tasks to minimize the overall execution time as specified below. The dynamic level, denoted by $DL(N_i,M_j,\Sigma)$ reflects how well the task and processor are matched at state Σ. Σ indicates the state of the processing resources (previously scheduled tasks) and the state of the communication resources (previously scheduled data transfers). The static level of task, denoted by $SL(N_i)$, is the largest sum of execution times along any directed path from task N_i to an endtask of the graph, over all endtasks of the graph. The static level component indicates the importance of the task in the precedence hierarchy, giving higher priority to the task located further from the terminus. $DA(N_i,M_j,\Sigma)$ is defined as

the earliest time that all data required by task is available at processor at state Σ. $TF(M_j,\Sigma)$ represents the time that the last task assigned to the *j*th processor finishes execution. The maximization term $max[DA(N_i,M_j,\Sigma),TF(M_j,\Sigma)]$ reflects the availability of the processing and communication resources, penalizing the task-processor pairs that incur large communication costs. Hence, a task-processor pair with an earlier starting time will have higher scheduling priority. $\Delta(N_i,M_j)$ accounts for the processor speed differences, adding priority to the processors that execute the task quickly, and subtracting priority from processors that execute the task slowly. The higher the value of the cost function, $DL(N_i,M_j,\Sigma)$, the more beneficial the task processor pair is, and as such, the more likely it is that processor M_j should be used for executing task N_i.

$$DL(N_i,M_j,\Sigma)=SL(N_i)-\max[DA(N_i,M_j,\Sigma),TF(M_j,\Sigma)]+\Delta(N_i,M_j) \quad (1)$$

where the variables are defined as:

N_i	=	*i*th task
M_j	=	*j*th processor
Σ	=	State of time
$DL(N_i,M_j,\Sigma)$	=	Dynamic level of a task- processor
$SL(N_i)$	=	The static level of a task
$DA(N_i,M_j,\Sigma)$	=	Data ready time
$TF(M_j,\Sigma)$	=	Processor ready time
$\Delta(N_i,M_j)$	=	Accounts for the varying processor speeds

Example 1: Let us consider an application to be scheduled onto a two-processor heterogeneous system. The processors are assumed to be interconnected by a full-duplex link. Figure 3 shows the acyclic precedence expansion graph (APEG) of the application. The numbers over the arrows represent the communication time. Execution time and power consumption of each task within each processor is given in Table 1. Note that static level of each task is calculated based on median execution time of the task among different machines. For example SL(E) is calculated along the H→G→E path as 9+4.5+3.5=17.

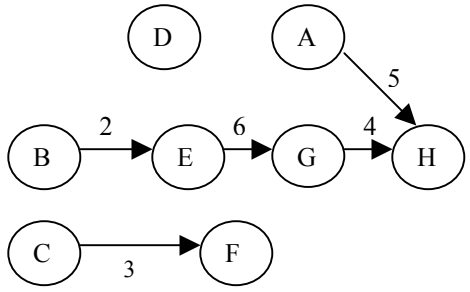

Figure 3. Acyclic precedence graph of an application.

The DLS algorithm selects a task-processor pair using the following steps. First, since tasks A, B, C and D are ready for execution on either processors M_0 or M_1, the dynamic levels of these tasks on M_0 and M_1 are evaluated using Equation 1 and the result is shown in Table 2. Since task B on

Table 1: Characteristics of the acyclic precedence graph

Task- N_i	M_0 Exec. Time- $T_0(i)$	M_1 Exec. Time - $T_1(i)$	Median Exec Time	Static Level $SL(N_i)$	Power Consumed on M_0 -$P_0(i)$	Power Consumed on M_1 -$P_1(i)$
A	2	3	2.5	11.5	4.25	2.5
B	2	3	2.5	19.5	5.25	2.6
C	4	6	5	8.5	2	1.5
D	5	4	4.5	4.5	1.5	2.4
E	3	4	3.5	17	6.2	2.7
F	3	4	3.5	3.5	4.5	2.6
G	4	5	4.5	13.5	2.1	1.75
H	10	8	9	9	1	1.2

processor M_0 has the highest dynamic level it is scheduled, as shown in Figure 4. Next, as task B is scheduled, task E can also be considered for scheduling. Hence, the algorithm evaluates the dynamic level of tasks A, E, C, and D, as shown in Table 3. Note that during these evaluations, $TF(M_0,\Sigma)=2$ and $DA(E,M_0,\Sigma)=2$ since both tasks E and C would reside on the same processor. Therefore, they require no communication overhead. On the other hand, $DA(E,M_1,\Sigma)=2+2=4$, since the result of task B from processor M_0 needs to be sent to processor M_1. Since task E has the highest dynamic level on processor M_0, it is scheduled on processor M_0.

Similarly, the algorithm makes other task-processor assignments. Figure 4 shows the resulting scheduling progressions using the DLS algorithm.

Table 2: Calculation of dynamic level in step1.

Processor M_0					
N	*SL*	*DA*	*TF*	Δ	*DL*
A	11.5	0	0	0.5	12
B	19.5	0	0	0.5	20
C	8.5	0	0	1	9.5
D	4.5	0	0	-0.5	4
Processor M_1					
A	11.5	0	0	-0.5	11
B	19.5	0	0	-0.5	19
C	8.5	0	0	-1	7.5
D	4.5	0	0	0.5	5

Table 3: Calculation of dynamic level in step2.

Processor M_0					
N	*SL*	*DA*	*TF*	Δ	*DL*
A	11.5	0	2	0.5	10
E	17	2	2	0.5	15.5
C	8.5	0	2	1	7.5
D	4.5	0	2	-0.5	2
Processor M_1					
A	11.5	0	0	-0.5	11
E	17	4	0	-0.5	12.5
C	8.5	0	0	-1	7.5
D	4.5	0	0	0.5	5

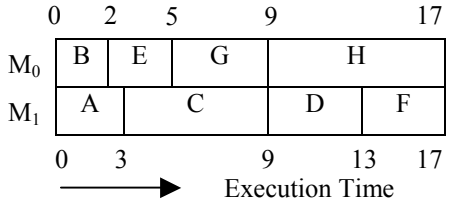

Figure 4. Scheduling using DLS Algorithm.

3. Low Power Scheduling

Similar to the DLS algorithm, the low power scheduling algorithm seeks a task-processor pair with the highest dynamic level. To account for power consumption, our algorithm uses the following cost function.

$$DL(N_i, M_j, \Sigma) = SL(N_i) - \max[DA(N_i, M_j, \Sigma), TF(M_j, \Sigma)] + \Delta(N_i, M_j) - E(N_i, M_j) \quad (2)$$

The first three terms of Equation 2 are taken directly from Equation 1. These terms promote the best-suited resources to tasks per the DLS algorithm. The term $E(N_i, M_j)$ is a measure of how much a given scheduling decision will contribute to the power consumption of tasks and vice versa. A larger value of $E(N_i, M_j)$ increases the scheduling priority for task N_i on processor M_j. Hence, we define $E(N_i, M_j)$ as

$$E(N_i, M_j) = \alpha[T_j(i) - T_{ave\_j}(i)] \quad (3)$$

α in equation 3 is a constant factor, which determines the relative importance of power in the scheduling process. $T_j(i)$ is the execution time of task N_i on processor M_j using $P_j(i)$, where $P_j(i)$) is the power used by task N_i on processor M_j. $T_{ave\_j}(i)$ is the execution time of task N_i on processor M_j using $P_{ave}(i)$, where $P_{ave}(i)$ is the average power used by task N_i on all processors. $T_{ave\_j}(i)$ is obtained from the power versus execution time characteristic graph. Figures 5 and 6 show examples of power-delay characteristics of a set of tasks on two processors.

3.1 LDLS Algorithm

Based on Equation 2, we have developed the following Low-power Dynamic Level Scheduling (LDLS) algorithm:

Step1. Calculate the median execution time of each task over all the processors.

Step2. Find the static level of each task in the APG, using the median execution times of the tasks.

Step3. Determine the dynamic level of every ready task for each processor using Equation 2.

Step4. Find the task-processor pair that has the highest dynamic level and assign the task on that processor.

Step5. Determine the new set of tasks that are ready for execution. If the set is not empty repeat steps three to five. ■

Example 2: Let us consider Example 1 using the LDLS algorithm. Figures 5 and 6 show the power-delay characteristics of the tasks on processors M_0 and M_1. Using Equation 3 and Figures 5 and 6,

$E(N_i, M_j)$ is obtained for each task-processor pair using α =1. The result is shown in Table 4. The

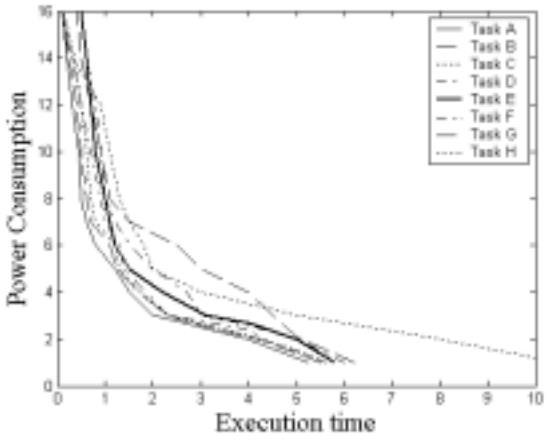

Figure 5: Power versus execution time of the application on processor M_0

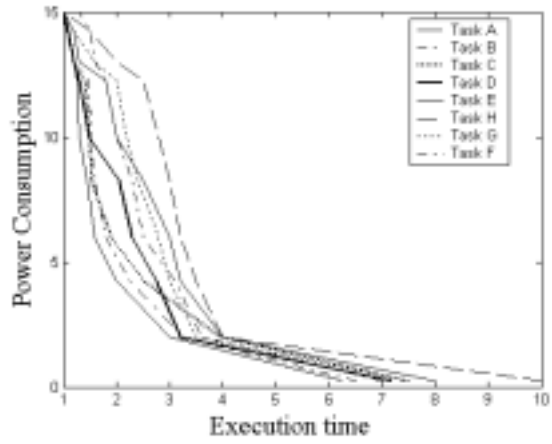

Figure 6: Power versus execution time of the application on processor M_1.

median execution time and static level of each task is the same as Example 1 and is shown in Table 1. Based on Figure 3, the available tasks are A,B,C and D. The dynamic levels of these tasks on processors M_0 and M_1 are evaluated using Equation 2 and the result is shown in Table 5. Accordingly, task B has the highest dynamic level on processor M_1. Therefore, it is scheduled on processor M_1.

Next, we consider the dynamic levels of tasks A, C, E and D as shown in Table 6. Note that $DA(E,M_0,\Sigma)$= 5 because of the communication delay between the processors. Moreover, $TF(M_0,\Sigma)$=3 since

Table 4: Evaluation of $E(N_i, M_j)$

N_i	$P_0(i)$	$P_1(i)$	$P_{ave}(i)$	$T_{ave-0}(i)$	$T_{ave-0}(i)$	$E(N_i, M_0)$	$E(N_i, M_0)$
A	4.25	2.5	3.37	2.4	1.7	-0.4	1.3
B	5.25	2.6	3.9	2.3	1.8	-0.3	1.2
C	2	1.5	1.75	4.5	5	-0.5	1
D	1.5	2.4	1.95	3.25	4.5	1.75	-0.5
E	6.2	2.7	4.45	3.25	2.4	-.25	1.6
F	4.5	2.6	3.55	3.4	2.5	-0.4	1.5
G	1.75	2.1	1.92	3.5	5.1	0.5	-0.1
H	1	1.2	1.1	8	10	0	0

Table 5: Calculation of dynamic levels in step 1 using LDLS algorithm.

Processor M_0						
N	*SL*	*DA*	*TF*	Δ	*E*	*DL*
A	11.5	0	0	0.5	-0.4	11.6
B	19.5	0	0	0.5	-0.3	19.7
C	8.5	0	0	1	-0.5	9
D	4.5	0	0	-0.5	1.75	5.75
Processor M_1						
A	11.5	0	0	-0.5	1.3	12.3
B	19.5	0	0	-0.5	1.2	20.2
C	8.5	0	0	-1	1	8.5
D	4.5	0	0	0.5	-0.5	4.5

Table 6: Calculation of dynamic levels in step 2 using LDLS algorithm.

Processor M_0						
N	*SL*	*DA*	*TF*	Δ	*E*	*DL*
A	11.5	0	0	0.5	-0.4	11.6
C	8.5	0	0	1	-0.5	9
D	4.5	0	0	-0.5	1.75	5.75
E	17	5	0	0.5	-0.25	12.25
Processor M_1						
A	11.5	0	3	-0.5	1.3	9.3
C	8.5	0	3	-1	1	5.5
D	4.5	0	3	0.5	-0.5	1.5
E	17	3	3	-0.5	1.6	15.1

processor M_1 has to first complete the execution of task B. Based on the results of Table 6, the LDLS algorithm assigns E on M_1.

Similarly, the algorithm evaluates the dynamic levels of the available tasks on the processors, and makes the scheduling assignment as shown in Figure 5.

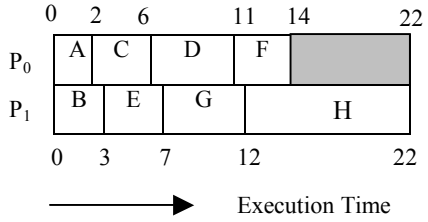

Figure 5. Scheduling using LDLS algorithm.

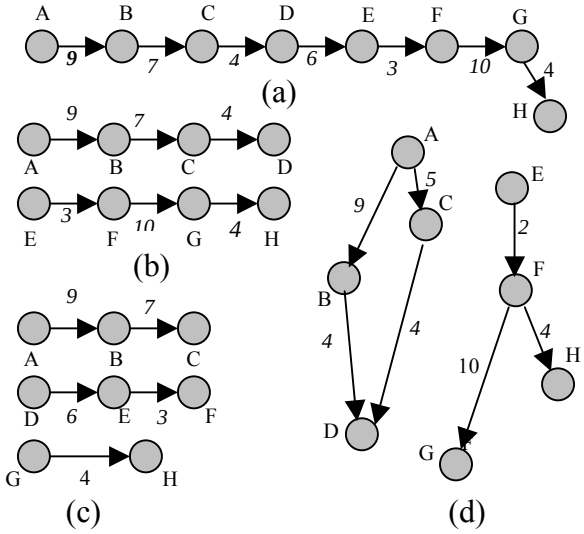

Figure 6. Acyclic Precedence Execution Graphs

4. Performance Results

In Example 1, the total power consumption is the sum of the power consumption of tasks B, E, G and H on M_0 and the power consumption of tasks A, C, D and F on M_1. Therefore, the total power consumption is

[(5.25+6.2+2.1+1)+(2.5+1.5+2.4+2.6)] =23.55

In Example 2, using LDLS algorithm, the total power consumption is the sum of the power consumption of tasks A, C, D and F on M_0 and the power consumption of tasks B, E, G and H on M_1. Hence, the total power consumption is

[(4.25+2+1.5+4.5)+(2.6+2.7+1.75+1.2)]= 20.5

The power saving is obtained at the cost of longer execution time. The tradeoff between execution time and power consumption can be controlled through α.

We next considered some other APEGs of the same tasks, as shown in Figure 6, and compared their scheduling using both LDLS and DLS algorithms. Their power-saving results are shown in Table 7. In all cases there is an improvement in power saving. The amount of improvement is dependent on a number of factors such as power usage of different processors, interdependencies of tasks, and the power delay characteristics of each task.

Table 6: Comparison of different APEGs.

APEG Figure	DLS		LDLS		Power improvement using LDLS (%)
	Power	Time	Power	Time	
3	23.55	17	20.5	22	12.95
8(a)	26.8	31	17.25	39	35.6
8(b)	22.8	18	21.25	23	6.79
8(c)	22.3	20	19.3	21	13.45
8(d)	22.8	18	21.25	23	6.79

5. Conclusion

In this paper, we have presented a compile-time scheduling strategy called Low-Power Dynamic Level scheduling algorithm. This algorithm takes into account the power consumption of an application. The algorithm is flexible and generates optimal scheduling time (with no power saving) when α is set to zero. In addition, desired power saving can be achieved by controlling α with a modest increase in the execution time.

References

[1] C. Reuter, M. Schwiegershausen and P. Pirsch. Heterogeneous Multiprocessor Scheduling and Allocation using Evolutionary Algorithms. In *Proceedings of the IEEE International Conference on Application-Specific Systems, Architecture and Processors*, pp. 294-303, July 1997.

[2] A. Khokhar, V. K. Prasanna, M. E. Shaaban and C. Wang. Heterogeneous Computing:

Challenges and Opportunities**.** *IEEE Transactions on Computers*, vol. 26, pp. 18-27, June 1993.

[3] H. Topcuoglu, S. Hariri and M.Y. Wu. Task Scheduling Algorithms for Heterogeneous Processors. In *Proceedings of The 8th Heterogeneous Computing Workshop***,** pp. 3-14, April 1999.

[4] L. Wang, H.J. Siegel, V.P. Rowchowdhury and A.A. Maciejewski. Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic- Algorithm-Based Approach. *Journal of Parallel and Distributed Computing*, vol. 47, pp. 8-22, November 1997.

[5] M.A. Iverson and F. Ozguner. Dynamic Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment. In *Proceedings of the 1998 Workshop on Heterogeneous Processing*, pp. 70-78, March 1998.

[6] A. Radulescu and A. J. C. Van Gemund. Fast and Effective Task Scheduling in Heterogeneous Systems. In *9th Heterogeneous Computing Workshop,* pp. 229-239, May 2000.

[7] Y.K. Kwok and A. Ishfaq. Link Contention-Constrained Scheduling and Mapping of Tasks and Messages to a Network of Heterogeneous Processors. In *Proceedings of 1999 International Conference on Parallel Processing*, pp. 551-558, September 1999.

[8] M. Tin, H.J. Seigel, J.K. Antonio and Y.A. Li. Minimizing the Application Execution Time Through Scheduling of Subtasks and Communication Traffic in a Heterogeneous Computing System. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 8, no. 8, pp. 857-870, August 1997.

[9] J.J. Dongarra and D.W. Walker. The Quest for Petascale Computing. In *IEEE Transactions on Computing in Science and Engineering*, pp. 32-39, May 2001.

[10] C.H. Gebotys and R.J. Gebotys. Power-Minimization in Heterogeneous Processing. In *Proceedings of 29th Hawaii International Conference on System Sciences* (HICSS'96), vol. 1, pp. 330-337, January 1996.

[11] R. Mishra, N. Rastogi, D. Zhu, D. Moss´e and R. Melhem. Energy Aware Scheduling for Distributed Real-Time Systems. In *Proceedings of International Parallel and Distributed Processing Symposium* (IDPS'03) (to appear).

[12] L. Benini and G. De Micheli. System-Level Power Optimization: Techniques and Tools. In *International Workshop on Low Power Electronics and Design*, 1999.

[13] W.T. Shiue and C. Chakrabarti**.** Low-Power Scheduling with Resources Operating at Multiple Voltages. In *IEEE Transactions on Circuits and Systems-2: Analog and Digital Signal Processing*, vol. 47, no. 6, pp. 536-543, June 2000.

[14] Y.H. Lu, L. Benini and G. De Micheli. Low-Power Task Scheduling for Multiple Devices. In *International Workshop on Hardware/Software Codesign 2000*, pp. 39-43, May 2000.

[15] D. Kirovski and M. Potkonjak. System-level Synthesis of Low-power Hard Real-time Systems. In *Proceedings of the 34th Annual Conference on Design Automation*, 1997.

[16] M. Srivastava, A. Chandrakasan, and R. Brodersen. Predictive System Shutdown and Other Architectural Techniques for Energy Efficient Programmable Computation. In *IEEE Transactions on VLSI Systems*, vol. 4, pp. 42-55, March 1996.

[17] J.M. Rabaey, "Digital Integrated Circuits," Prentice Hall, Inc., 1996.

[18] G.C. Sih and E.A. Lee. A Compile-Time Scheduling Heuristic for Interconnection-constrained Heterogeneous Processor Architectures. In *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 2, pp. 175-187, February 1993.