Scheduling DAGs For Minimum Finish Time and Power Consumption on Heterogeneous Processors

Except where reference is made to the work of others, the work described in this thesis is my own or was done in collaboration with my advisory committee. This thesis does not include proprietary or classified information.

Kiran Kumar Palli

Certificate of Approval:

Prathima Agrawal, Co-Chair
Professor
Department of Electrical and Computer Engineering

Sanjeev Baskiyar, Chair
Assistant Professor
Department of Computer Science and Software Engineering

Levent Yilmaz
Assistant Professor
Department of Computer Science and Software Engineering

Stephen L. McFarland
Acting Dean, Graduate School

Scheduling DAGs For Minimum Finish Time and Power Consumption on Heterogeneous Processors

Kiran Kumar Palli

A Thesis

Submitted to

the Graduate Faculty of

Auburn University

in Partial Fulfillment of the

Requirements for the

Degree of

Master of Science

Auburn, Alabama
August 8, 2005

Scheduling DAGs For Minimum Finish Time and Power Consumption on Heterogeneous Processors

Kiran Kumar Palli

_____
Signature of Author

_____
Date

Copy sent to:

Name Date

Vita

Kiran Kumar Palli, son of Prakash Palli and Sita Laxmi Palli, was born in Manthani, a small village in the district of Karimnagar, Andhra Pradesh, India. He graduated from Gautami Junior college, Hyderabad and then earned his Bachelor of Engineering in Electronics and Communication Engineering from Osmania University, Hyderabad, India in 2002.

Thesis Abstract

Scheduling DAGs For Minimum Finish Time and Power Consumption on Heterogeneous Processors

Kiran Kumar Palli

Master of Science, August 8, 2005
(B.E., Osmania University–Hyderabad,India 2002)

50 Typed Pages

Directed by Sanjeev Baskiyar

In the past years, scheduling has always been a hot research area, and with the advancement of processor technologies, there has been an increasing focus on the strategies that allow managing resource allocation in an optimal way. There is a large amount of increase on battery operated devices, and hence low power design is taking a main role in manufacturing most of today's electronics.

In a distributed environment, an application is usually decomposed into several independent and/or interdependent sets of co-operating tasks and assigned to a set of available processors for execution. These sets of tasks can be represented by a directed a-cyclic graph (DAG). The set of processors can be homogenous or heterogeneous. Heterogenous processors vary in factors like processor's speed, available memory, operating system, power supply, etc. The role of a good scheduling algorithm is to efficiently assign each task to a processor depending on the resources needed such that the communication overhead between related tasks is reduced and the precedence relations among tasks are satisfied. This will minimize the total finish time and total power consumption.

The scheduling algorithm developed in this work to minimizes both total execution time (makespan) and total power consumption of a DAG structured application on heterogeneous processors. This algorithm combines the techniques of heterogeneous earliest finish time (HEFT)[1] and voltage scaling[2] to obtain both minimum makespan and low power consumption. The processors used are considered to be continuously voltage scalable in range of operation. After the initial scheduling is done using HEFT for minimum makespan, the processors are voltage scaled down and/or frequency scaled down, i.e., slowed down, to reduce the power consumption whenever there is an idle time. This voltage scaling is performed without violating the precedence relationships among tasks. The simulation results show power saving of 22.5% over simple HEFT and same makespan as of simple HEFT.

Acknowledgments

I would like to acknowledge the support and guidance from Dr. Sanjeev Baskiyar, whose suggestions and directions have a major influence on all aspects of this thesis. It was a great pleasure to undergo this learning experience. I thank Dr. Prathima Agrawal and Dr. Levent Yilmaz for being on my committee and providing me valuable inputs. I thank Dr. Fa Foster Dai and Dr. Richard Jaeger who supported me during the first year of my study at Auburn. I am thankful to all my friends at Auburn for being the surrogate family during my Masters.

I cannot end without thanking my parents, brother, and sister on whose constant encouragement and love I have relied throughout my life. And to my loving wife for always being there but never asking why it was taking so long. To them I dedicate this thesis.

Style manual or journal used Journal of Approximation Theory (together with the style known as "aums"). Bibliography follows van Leunen's *A Handbook for Scholars*.

Computer software used The document preparation package TEX (specifically LATEX) together with the departmental style-file `aums.sty`.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Heterogeneous computing uses a diverse set of resources connected over a high speed network to support computationally intensive applications. These applications involve distributed and parallel processing tasks. These tasks often require to be scheduled in an optimum way so that the computation is done in minimum time, consuming minimum resources. Task scheduling finds extensive use in ubiquitous computing, which operates in resource constrained environment.

1.1 Problem Statement

There has been a lot of research in scheduling the tasks either for minimum finish time or for high parallelism. There are many publications describing static, dynamic or hybrid scheduling algorithms for these purposes. Tasks are scheduled either on super computers or on resource constrained processors. Any real-time environment will be imposed with some constraints like speed, power, memory, etc. Hence it is very important to address these issues in scheduling tasks. There is little work done which addresses both makespan and power consumption. Most of the work done to reduce the power consumption in a distributed system neglects the finish time of such a system. The power saving is obtained at the cost of longer execution time.

There are many applications which require both minimum finish time and minimum power consumption. Power consumption is a major issue in many real time distributed systems such as real time communication in satellites, as most applications running on a

power-limited systems inherently impose temporal constraints on finish time. A new area of interest is multi-hop radio networks used for sensor kind data traffic. New wireless communication systems are expected to evolve using this system. These networks are distributed networks operating on power constraints, also called power-aware distributed systems (PADS).

Hence there is a need for a scheduling algorithm which would effectively reduce the overall power consumption and yet attain the best possible makespan.

1.2 Prior Work on Scheduling for Low Power

The Classification of Low Power Research as described in [4] is as follows:

- Power Estimation Techniques (Power Model): A literature survey on existing power estimation methodologies at various levels of abstraction namely, instruction-level, architectural level and gate-levels, is presented here.
  - Instruction-level: Nikolaidis et al.[5] presents a power consumption measurement configuration for embedded processing systems. This work takes an assembly or machine level program as input and gives an estimate of the energy consumption of the specific program in the specific processing systems. This provides an accurate estimation of power consumption even in the presence of instantaneous power supply variations.
  - Architecture-level:[6] describes a method for architecture-level estimation of power consumption. It provides cycle-by-cycle power consumption data of the architecture on the basis of the instruction/data flow stream. At the

architecture level, Landman et al.[7] presented a technique for the characterization of module library using signal statistics. Landman et al.[8] presented a methodology for low-power design-space exploration at the architectural level of abstraction. Black-box power models for the architecture-level components were generated [9] and used to estimate power while preserving the accuracy of the gate or circuit level estimation.

– Gate-level Power: [10] provides a method for gate-level measurement of power consumption. This work compares all the different kinds of power estimation models and proves that gate-level estimation of power consumption is the most accurate measurement.

The power estimation techniques at the gate level and lower levels of abstraction can be broadly classified into

1. Simulation based techniques
2. probabilistic techniques; and
3. statistical techniques

One of the earliest techniques proposed were simulation based techniques [11], [12] , where the average power is calculated by monitoring both the supply voltage and current waveforms. These are too slow to handle very large circuits. Other simulation based techniques [13], [14] assume that the power supply voltages and ground voltages are constant, estimating only the supply current waveform.

In the probabilistic techniques [15], user supplied input signal probabilities are propagated into the circuit. To achieve this, special models for the components have to be developed and stored in the module library. Various other probabilistic techniques [16], [17] were proposed but all of these approaches are applicable only to combinational circuits.

Statistical techniques [18], [19] do not require any specialized models for the components. The idea is to simulate the circuit with randomly generated input vectors until power converges to the average power. The convergence is tested by statistical mean estimation techniques.

- Power Optimization Techniques: Competition is driving the requirement for power optimization and shorter design cycles. Until now, low power design methodology has consisted of power estimation, utilizing tools that report power consumption of a design at various stages of the design cycle. Today's designs require power optimization tools that address power consumption early in the design cycle.
  - Hardware Optimization
    - Behavior-level: Transformations, Scheduling, Resource Allocation, etc.
    - Architecture-level: Low power flip flop, Low power adder, etc.
    - Circuit-level: Low power circuit, etc.
  - Software Optimization
    - Instruction-level: Low power compiling, Low power instruction scheduling, etc.
    - System-level: Dynamic power management, Low power memory management, etc.

There has been a lot of work to minimize the power consumption at all the levels. This work tries to minimize the power consumption using voltage and frequency scaling.

1.3 Contributions

This scheduling algorithm effectively addresses both the issues of minimizing the makespan and reducing the power consumption. This algorithm uses the HEFT algorithm as proposed by Topcuoglu et al.[1] without modifying it at all. After the initial scheduling is done using HEFT for minimum finish time, this algorithm performs voltage scaling and frequency scaling to reduce the power consumption without decreasing the finish time obtained by HEFT. Frequency scaling is done whenever there is an idle time on the processor. Also, care is taken to meet the precedence relationships.

1.4 Organization of the report

The remainder of the thesis is organized as follows. In the next section, we describe the necessary background information on scheduling and related terminology. Different types of scheduling and differences between scheduling on homogeneous processors and heterogeneous processors are also discussed. Prior work on scheduling for minimum finish time and prior work on low power scheduling algorithms are discussed.

Chapter 3 introduces the concepts of HEFT and discusses its algorithm in detail.

Chapter 4 discusses how voltage scaling can be used to reduce the power consumption of a processor.

Chapter 5 combines the concept of HEFT and voltage scaling to create a new algorithm called low power heterogeneous earliest finish time (LPHEFT).

Results and graphs are discussed in chapter 6.

Chapter 7 ends the thesis with conclusions and future work.

Chapter 2

Background

Definition of a DAG: In a distributed environment, an application can be decomposed into a set of computational tasks. These tasks may have data dependencies among them, thereby creating a partial order of precedence in which the tasks may be executed. DAGs are widely used to represent dependancy graphs in scheduling of both homogeneous and heterogeneous processors. DAGs are an important class of graphs and have many applications such as those involving precedence among events.

Basic terminology:

- **node:** a task or sub-task being executed
- **link(edge):** represents data dependencies between nodes
- **root:** a node with no incoming edges
- **leaf:** a node with no outgoing edges

An example of a task DAG is shown in Figure 2.1.

In this thesis, a DAG is represented by the tuple *G=(V,E,P,T,C,W)* where V is the set of v nodes, E is the set of e edges between the nodes, and P is a set of p processors. $E(v_i, v_j)$ is an edge between nodes v_i and v_j. T is the set of costs $T(v_i, p_j)$, which represent the computation times of tasks v_i on processor p_j. C is the set of costs $C(v_i, v_j)$, which represent the communication cost associated with the edges $E(v_i, v_j)$. Since intra-processor communication is insignificant compared to inter-processor communication, $C(v_i, v_j)$ is considered to be zero if v_i and v_j execute on the same processor. W is the set

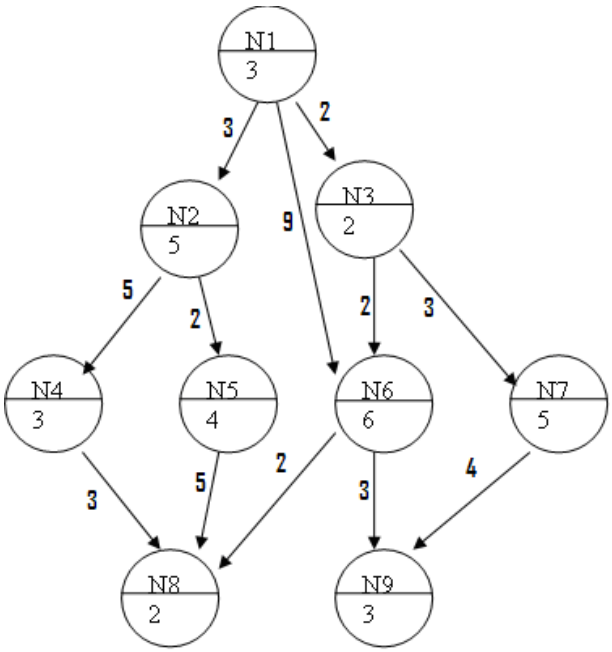

Figure 2.1: A Simple Task DAG Graph

of costs $W(v_i, v_j)$, which represent the power consumption costs of tasks v_i on processor p_j. The length of a path is defined as the sum of node and edge weights in that path.

Node v_p is a predecessor of node v_i if there is a directed edge originating from v_p and ending at v_i. Likewise, node v_i is a successor of node v_i if there is a directed edge originating from v_i and ending at v_i. We can further define $pred(v_i)$ as the set of all predecessors of v_i and $succ(v_i)$ as the set of all successors of v_i. An ancestor of node v_i is any node v_p that is contained in $pred(v_i)$, or any node v_a that is also an ancestor of any node v_p contained in $pred(vp)$.

The earliest execution start time of node v_i on processor p_j is represented as $EST(v_i, p_j)$. Likewise the earliest execution finish time of node v_i on processor p_j is represented as $EFT(v_i, p_j)$. $EST(v_i)$ and $EFT(v_i)$ represent the earliest start time upon any processor and the earliest finish time upon any processor, respectively. $T_avail[v_i, p_j]$ is defined as the earliest time that processor p_j will be available to begin executing task v_i. Hence

$$EST(v_i, p_j) = maxT_avail[v_i, p_j], max_{v_p \epsilon pred(v_i)}(EFT(v_p, p_k) + C(v_p, v_i)) \tag{2.1}$$

$$EFT(v_i, p_j) = T(v_i, p_j) + EST(v_i, p_j) \tag{2.2}$$

The maximum clause finds the latest time that a predecessor's data will arrive at p_j. The goal of low power HEFT algorithm is to minimize both makespan and total power consumption (makepower) which are defined as follows:

$$makespan = maxEFT(v_i), \tag{2.3}$$

where v_i is the exit node of the graph.

$$makepower = \sum W(v_i, p_j) \tag{2.4}$$

Schedule length (makespan) is a major metric to measure the performance of a scheduling algorithm on a graph. Since a large set of graphs is used, it is necessary to normalize the schedule length to a lower bound called schedule length ratio (SLR). The SLR of an algorithm on a graph is defined by

$$SLR = \frac{makespan}{\Sigma_{n_i \epsilon CP_{MIN}} min_{p_j \epsilon Q} W_{i,j}} \tag{2.5}$$

The denominator is the summation of the minimum computation costs of tasks on CP_{MIN}. The SLR of a graph cannot be less than one since the denominator is the lower bound.

2.1 Derivation of formula for power calculation

The processor clock frequency, *f*, as explained in [21] can be expressed in terms of the supply voltage, V_{dd} , and the threshold voltage, V_t, as follows:

$$f = k(V_{dd} - V_t)^2 / V_{dd} \tag{2.6}$$

where k is a constant.

From the above equation, we can derive V_{dd} as a function of *f*, *F(f)*,

$$V_{dd} = F(f) = (V_t + \frac{f}{2k}) + \sqrt{(V_t + \frac{f}{2k})^2 - (V_t)^2} \tag{2.7}$$

The processor power, p, can be expressed in terms of the frequency, f, switched capacitance, N and the supply voltage, V_{dd}, as:

$$p = \frac{1}{2} f N (V_{dd})^2 = \frac{1}{2} f N F(f)^2 \tag{2.8}$$

Given the no. of clock cycles, η_i, for executing task i, its energy consumption, E_i, under supply voltage V_i and clock frequency, f_i, is given by

$$E_i = (\frac{\eta_i}{f_i}) * p(f_i) \tag{2.9}$$

2.2 Scheduling of DAGs

A lot of work has been done in scheduling of DAG based applications to minimize finish time. But it is a well known fact that scheduling of DAGs is NP-complete. In other words, for a given arbitrary task, it is unlikely that a polynomial time algorithm exists for an optimal schedule [13]. Scheduling can be preemptive or non-preemptive. In this work we deal with non-preemptive scheduling.

The task scheduling algorithms for a distributed system can be classified broadly into two categories viz., (1) Static scheduling and (2) Dynamic scheduling In static scheduling, the allocation of tasks to processors is decided in advance and the allocation is done only when all resources are available. The logical allocation of jobs takes place at compile-time and it does not have any runtime overhead.

- **Advantages:** Simple, low run-time overhead
- **Disadvantages:** Low resources utilization, lack of adaptation to changing situations

On the other hand, in dynamic scheduling, the allocation of jobs to the processors is done as soon as the resources are available and is decided in advance. Therefore, logical allocation of jobs is done at run-time. But it suffers from the drawback of runtime overhead as the allocation chosen may not be the best.

- **Advantages:** Optimal schedules in terms of CPU utilization
- **Disadvantages:** High run-time cost, no assurance of schedulability.

There are some hybrid scheduling algorithms which integrate both static and dynamic scheduling heuristics.

2.3 Scheduling on heterogeneous processors

In a real-time distributed environment, the availability of computing resources varies a lot which results in both temporal as well as spatial heterogenity. Heuristic-based and guided-random-search based algorithms are two principal approaches of scheduling DAGs. Heuristic algorithms are again classified as list scheduling, cluster scheduling and task duplication based scheduling. Guided random search based algorithms can be classified as Genetic Algorithms, Simulated Annealing and Local Search Technique. List scheduling heuristic tries to minimize a predefined cost function by first making a list of all tasks based on their priorities and then selecting the appropriate processor based on the heuristic. List scheduling algorithms are more practical and give a better finish time

than others. Examples of list scheduling are HEFT [1], Heterogeneous N-predecessor Decisive Path (HNPD)[3], Modified Critical Path (MCP) [18], Dynamic Critical Path [19], Dynamic Level Scheduling[20] and Mapping Heuristic [21]. Among the algorithms to schedule DAGs onto heterogeneous processors, HNPD, HEFT and STDS are shown to be the best.

Task duplication can minimize interprocessor communication and hence results in shorter finish times. HNPD[3] is shown to give better makespan than HEFT. It combines the techniques of HEFT and Scalable Task Duplication Scheduling (STDS)[28]. HNPD uses task duplication to minimize finish time, however this approach increases power consumption. We therefore chose to perform voltage scaling on HEFT.

Chapter 3

HEFT

HEFT is an insertion-based algorithm, i.e it tries to schedule a task between two already scheduled tasks. This chapter first introduces the graph attributes used for setting the task priorities. Then it discusses the HEFT algorithm.

3.1 Graph Attributes

Tasks are ordered with respect to their upward and downward ranking. The upward rank of a task n_i is recursively defined as follows.

$$rank_u(n_i) = \overline{W_i} + max_{n_j \epsilon succ(n_i)}(\overline{C_{i,j}} + rank_u(n_j)) \tag{3.1}$$

where $succ(n_i)$ is the set of immediate successors of task n_i, $\overline{C_{i,j}}$ is the average communication cost of edge(i,j) over all processor pairs, and $\overline{W_i}$ is the average the set of computation cost of task n_i. Since the rank is computed recursively by traversing the graph upward, starting from the exit task, it is called upward rank. For the exit task n_{exit}, the upward rank value is equal to

$$rank_u(n_{exit}) = \overline{W_{exit}} \tag{3.2}$$

Basically, $rank_u(n_i)$ is the length of the critical path from task n_i to the exit task, including the communication cost of task n_i. Similarly, the downward rank of a task n_i is recursively defined by

$$rank_d(n_i) = max_{n_j \epsilon pred(n_i)} rank_d(n_j) + \overline{w_j} + \overline{C_(j,i)} \tag{3.3}$$

where $pred(n_i)$ is the set of immediate predecessors of task n_i. The downward ranks are computed recursively by traversing the task graph downward starting from the entry task of the graph. Basically, $rank_d(n_i)$ is the longest distance from the task to task n_i, excluding the computation cost of the task itself.

3.2 HEFT Algorithm

The HEFT algorithm can be divided into two phases, the task prioritizing phase and processor selection phase. In the task prioritizing phase, the priority of each task is set to its upward rank. Next, the task list is sorted in decreasing order of upward rank. HEFT algorithm has an insertion based policy in which it tries to insert a task in the earliest idle time-slot between two already scheduled tasks on a processor. The idle time-slot should be large enough to accommodate the task to be inserted. Also, insertion of the task should not violate any precedence relationships. The HEFT algorithm has an $O(exq)$ time complexity for e edges and q processors. For a dense graph when the number of edges is proportional to $O(v^2)$ (v is the number of tasks), the time complexity is on the order of $O(v^2xp)$.

Chapter 4

Voltage Scaling

Power consumption is the limiting factor for the functionality of devices operating on batteries with rapidly increasing computing and communication costs. Hence it is very important to utilize the energy resources as efficiently as possible.

4.1 Definition

Voltage scaling is a technique in which the core power supply voltage of a system is varied depending on the processing load, to decrease the total power consumption. But reducing the power supply voltage also reduces the speed of execution. In some instances it is observed that with the reduction in supply voltage from 5.0V to 3.3V, there is about 56% reduction in power consumption[26].

4.2 Voltage Scaling Design Challenges

The embedded system or the circuit board should have an operating system that has a power management section and a hardware which scales the frequency and voltage dynamically. The operating system commands the voltage and frequency scaling circuit to change the operating frequency and processor core voltage depending on the processing load. The voltage scaling circuit controls the core power supply voltage. The power supply usually requires milliseconds to switch from one voltage to another. This delay between the logic control signal and the actual voltage change is called voltage scaling latency. CMOS devices typically require higher voltage to operate at higher frequencies.

Stable operation is not guaranteed during power supply voltage transition. Frequency scaling circuitry should stop the clock during this voltage transition. In order to maintain the timing integrity of a synchronous system, there should be no runt clock cycles. A runt clock cycle is a clock cycle which ends before its expected clock period. For example, when a clock speed is switched from 300MHz to 33MHz, it should do so only after the current 300MHz cycle. It may be necessary to implement an additional handshaking mechanism between the processor and the frequency scaling circuitry.

4.3 Voltage Scaling Algorithm

Consider the example in Figure. 4.1 (a) and (b), where to perform task T6 we need to first perform task T2 and T3. Assume T3 takes 20 seconds to complete, while T2 takes only 15 seconds on their respective processors. So, the processor executing task T2 has an idle time of 5 seconds which means it can operate at a lower speed, i.e., at a low supply voltage. Suppose, that T4 takes 15 seconds or less to complete. Now, the processor executing task T4 has some idle time and hence can operate at low power. Next, tasks T2 and T4 can be merged into a group so that they both can be dynamically voltage scaled at the same time as shown in Figure 4.1 (c). Such merging decreases the number of times one must dynamically switch the voltage. Again T2 and T4 can be merged if they don't have an off chip data transfer. Thus given a voltage scaled schedule, tasks are merged wherever possible and then merged tasks which can operate at lower frequencies are determined.

Consider Figure 4.2 which graphs energy consumption against execution time at different power supply voltages and clock frequencies. This clearly shows the advantage of voltage and frequency scaling. When a task is executed at 5V power supply and

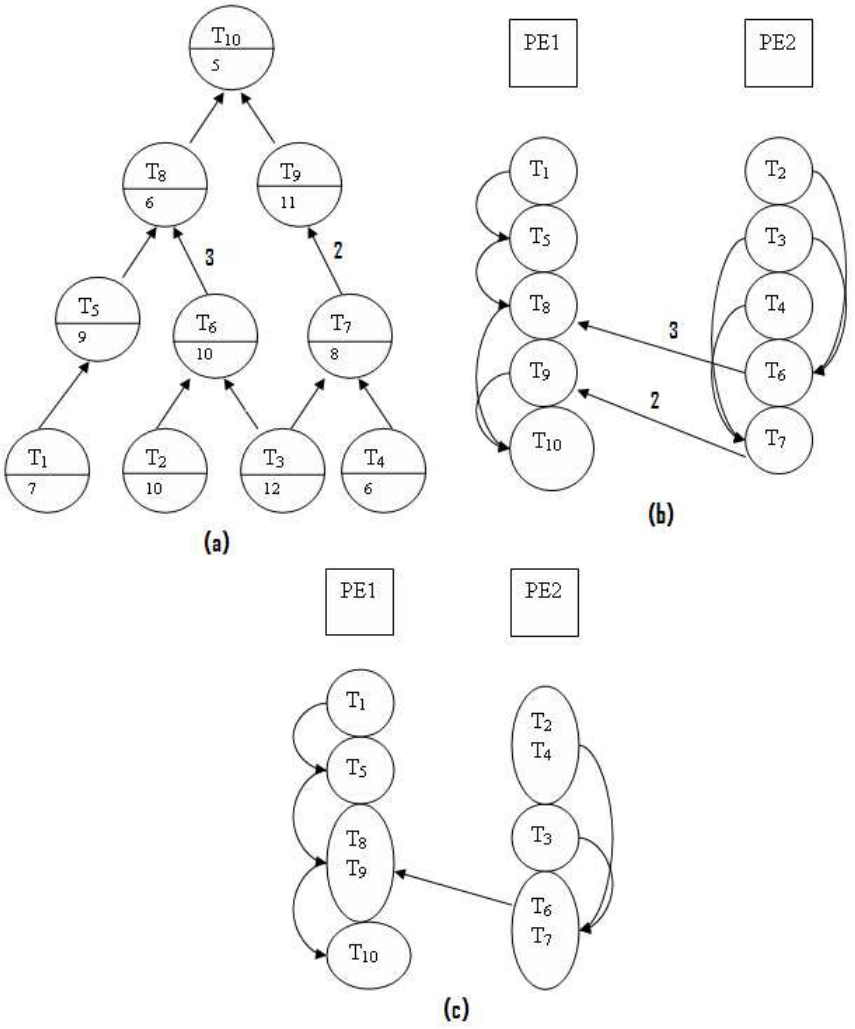

Figure 4.1: Voltage Scaling

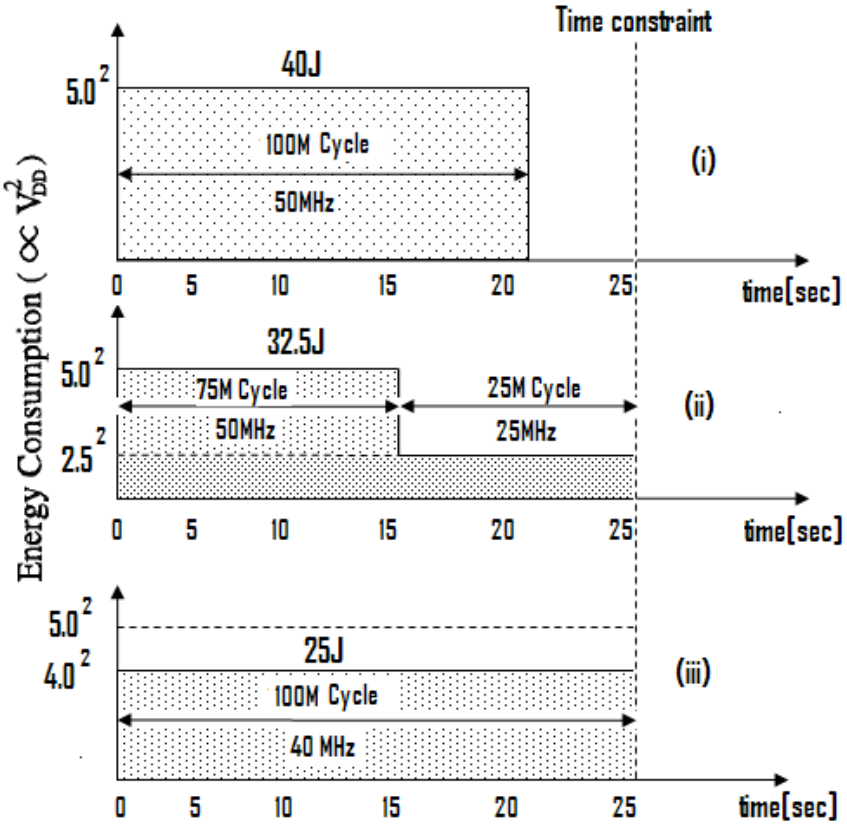

Figure 4.2: An Example of Power-Delay Optimization [2]

50MHz frequency, it consumes 40J of energy. But if the power supply in the last quarter of the execution time is scaled down to 2.5V and frequency to 25MHz, it meets the deadline and consumes only 32.5J of energy. If the power supply is scaled down 4V and frequency to 40MHz from the start, it still meets the deadline requirements and consumes only 25J of energy.

Chapter 5

Low Power HEFT Algorithm

5.1 Problem Redefined

There are many time critical applications which operate on low power and require parallel processing of their tasks such as wireless embedded sensor networks, real time communication in satellites and other image processing applications. There has been little work done which addresses both low power design and minimum finish time.

5.2 LPHEFT for minimum finish time and minimum power consumption

This algorithm, which is a low power version of HEFT, gives us the same makespan and yet consumes considerably less power than the HEFT algorithm. In this algorithm, the processor is voltage scaled during idle time. Idle time is obtained when one processor is waiting to execute a task which is dependant on another task being executed by some other processor. The idle time is eliminated by decreasing the processor's speed thereby increasing the time taken for the processor to execute the previous task. Care has been taken that all precedence relationships are satisfied.

The Algorithm

1	Set the computation costs of tasks and communication costs of edges with mean values. Also set the power consumption of tasks for each processor.
2	Calculate the $rank_u$ for all tasks by traversing graph upward starting from the exit task.
3	Sort the tasks in a scheduling list by non-increasing order of $rank_u$ values.
4	**while** there are unscheduled tasks in the list
5	Select the first task n_i, from the list for scheduling
6	**for** each processor P_j in the processor-set ($P_j \epsilon Q$)
7	compute EFT(n_i, P_j) value using the insertion based scheduling policy
8	Assign task n_i to the processor P_j that minimizes EFT of task n_i
	End for
	End while
9	**For** all k, m, i, j
	do
10	**if**$(EFT(n_k, P_i) + C(P_i, P_j)) > EFT(n_m, P_j)$ $EFT(n_m, P_j) = EFT(n_k, P_j) + C(P_i, P_j)$
	Continued on next page

Table 5.1 – continued from previous page

11	**While** ($n_k isparentofanodewhichisachildofn_m$)
	END For
12	Calculate the total power consumption which is the sum of power consumption of all tasks using the formula $p = \frac{fNV2_{dd}}{2} = \frac{fNF(f)2}{2}$ where $V_{dd} = F(f) = (V_t + \frac{f}{2k}) + \sqrt{((V_t + \frac{f}{2k}) - V_t^2)}$

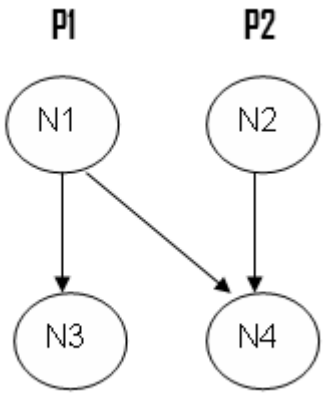

Figure 5.1: Low Power Algorithm

By using the First algorithm, $if(EFT(N1, P1) + C(P1, P2)) > EFT(N2, P2)$ we would increase EFT(N2,P2) because P2 has to however wait for N1 to execute N4.

If searched for idle time on P2, then we would find some time $ifEFT(N2, P2) < EST(N4, P2)$

This is in fact the same as looking for any available idle time on all the processors. This leads to our alternate algorithm which is more effective.

Alternate Algorithm

1	Set the computation costs of tasks and communication costs of edges with mean values. Also set the power consumption of tasks for each processor.
2	Calculate the $rank_u$ for all tasks by traversing graph upward, starting from the exit task.
3	Sort the tasks in a scheduling list by non-increasing order of $rank_u$ values.
4	**while** there are unscheduled tasks in the list **do**
5	Select the first task n_i, from the list for scheduling
6	**for** each processor P_j in the processor-set ($P_j \epsilon Q$)
7	compute $EFT(n_i, P_j)$ value using the insertion based scheduling policy
8	Assign task n_i to the processor P_j that minimizes EFT of task n_i
	End for
	End while
9	**For** all k, m, j
	Continued on next page

Table 5.2 – continued from previous page

	do
10	**if**$((EFT(n_k, P_j) < EST(n_m, P_j)))$ $EFT(n_k, P_j) = EST(n_m, P_j)$
11	**While**(n_k is a node which is scheduled immediately before n_m on P_j)
	END For.
12	Calculate the total power consumption which is the sum of power consumption of all tasks using the formula $p = \frac{fNV2_{dd}}{2} = \frac{fNF(f)2}{2}$ where $V_{dd} = F(f) = (V_t + \frac{f}{2k}) + \sqrt{((V_t + \frac{f}{2k}) - V_t^2)}$

5.3 Random DAGs

Random weighted directed-acyclic-graphs are generated each time computing the results. It requires the following input parameters.

- Number of nodes in the graph (ν),
- Shape parameter of the graph (α),
- Mean out degree of a node, ($out - degree$)
- Communication to computation ratio, (CCR)
- Computation range (β),

In each experiment, these values are varied over a set of values given below.

SET(ν) = 10, 20, 40, 60, 80

SET(α) = 0.5, 1.0, 2.0

SET($out - degree$) = 1, 2, 3, 4, 5

SET(CCR) = 0.1, 0.5, 1.0, 5.0, 10.0

SET(β) = 0.1, 0.25, 0.5, 0.75, 1.0

These combinations give 2,250 different DAG types. Assigning a number of input parameters and selecting each input parameter from a set of values will generate diverse DAGs with wide variety of characteristics.

Chapter 6

Results and Discussions

The performance of the algorithm was compared with respect to various graph characteristics. Figure. 6.1 gives the net percentage reduction in power consumption with respect to varying CCR. The percentage of power reduction is increasing as CCR is increasing. This is because, as CCR increases, the communication cost of tasks increases and thus more tasks are scheduled on same processor rather than choosing a new processor. This increases idle time between tasks among processors and thus more power reduction.

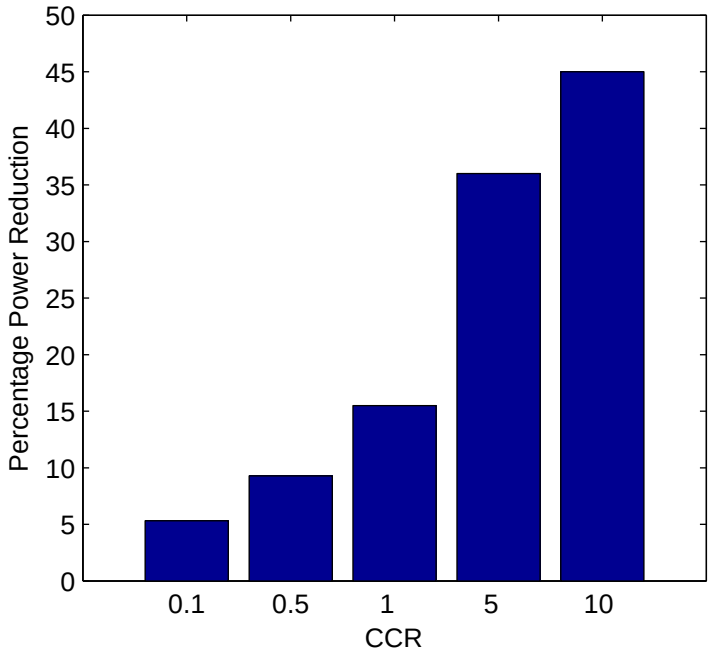

Figure 6.1: Percentage power reduction w.r.t. CCR

CCR	Power Consumption	
	Without Voltage Scaling	With Voltage Scaling
0.1	2882	2737
0.5	3722	3386
1	2940	2533
5	4091	2591
10	4788	2476

Table 6.1: Percentage power reduction w.r.t. CCR

Figure. 6.2 graph shows the variation of percentage reduction in power consumption with respect to processor availability factor (PAF). As the number of processors increase, initially there is an increase in power reduction. Afterwards, there is not much difference in power reduction as not all processors are efficiently utilized. In the case of PAF=1, each processor is assigned only one task. All processors are assumed to be in sleep mode before starting their execution and after finishing their execution. Hence there is not a lot of difference in power reduction as PAF increases after certain point. If low power algorithm alone assumes sleep mode, a much higher power reduction can be reported.

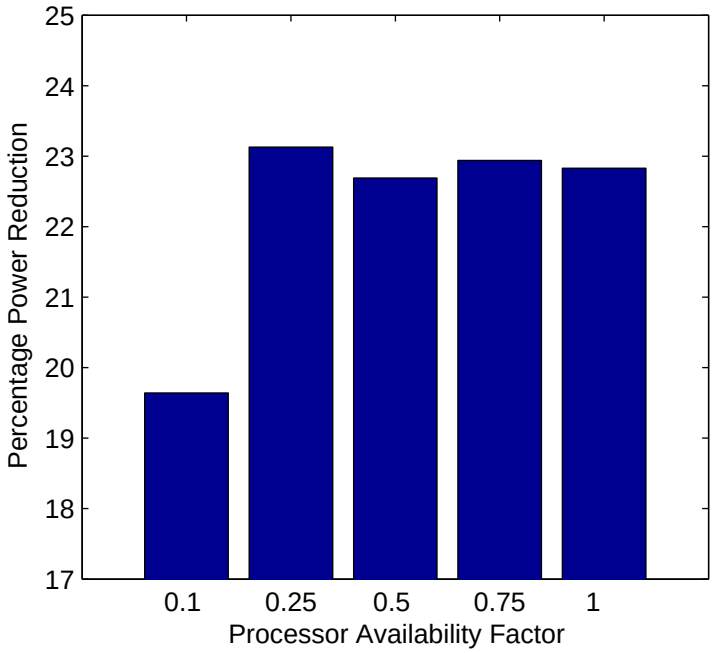

Figure 6.2: Percentage power reduction w.r.t. PAF

PAF	Power Consumption	
	Without Voltage Scaling	With Voltage Scaling
0.1	3186	2408
0.25	3445	2411
0.5	3766	2734
0.75	3892	2998
1	4133	3171

Table 6.2: Percentage power reduction w.r.t. PAF

Figure. 6.3 shows the graph of varying percentage of power reduction with respect to shape parameter. If $SP < 1$, it means a longer graph with less parallelism and if $SP > 1$ it means a short graph with high parallelism. For graphs with small parallelism, tasks are more dependent and processors have to wait more, thus more power reduction is possible using voltage scaling.

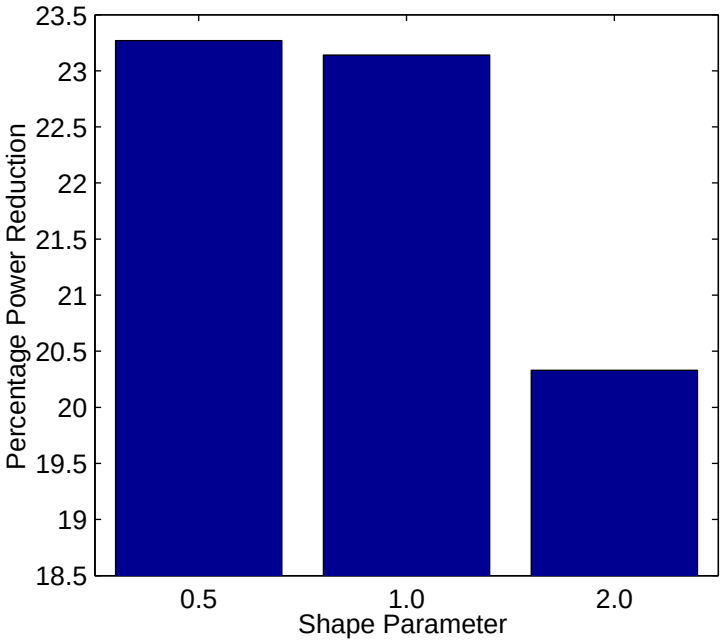

Figure 6.3: Percentage power reduction w.r.t. shape parameter

Shape Parameter	Power Consumption	
	Without Voltage Scaling	With Voltage Scaling
0.5	3311	2257
1	3895	2887
2	3848	3090

Table 6.3: Percentage power reduction w.r.t. shape parameter

Figure. 6.4 shows the graph of percentage power reduction with respect to computation range. As computation range increases, heterogeneity among processors increases. Hence different processors execute similar tasks at different speeds. Hence we can find more idle time between tasks. Hence we see a slight increase in power reduction as computation range increases.

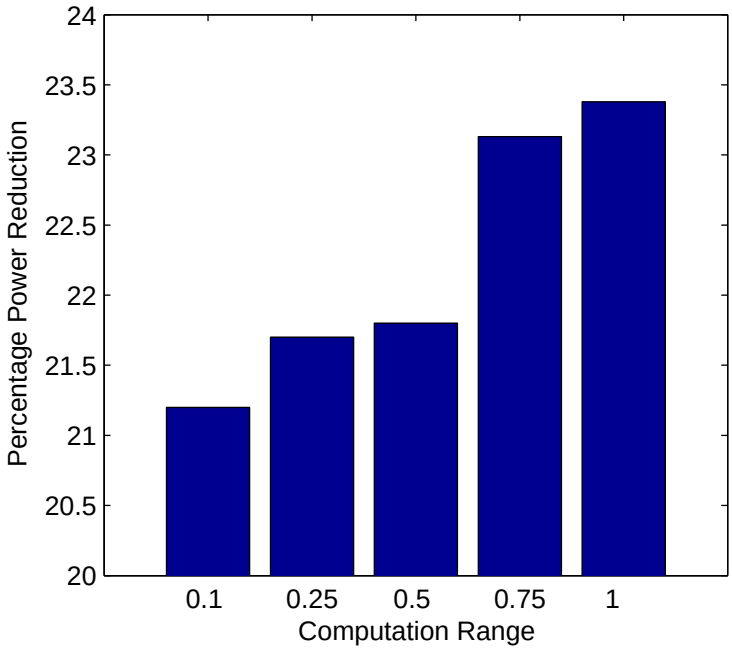

Figure 6.4: Percentage power reduction w.r.t. computation range

Computation Range	Power Consumption	
	Without Voltage Scaling	With Voltage Scaling
0.1	4364	3291
0.25	4146	3122
0.5	3780	2845
0.75	3292	2400
1	2840	2066

Table 6.4: Percentage power reduction w.r.t. computation range

Thus we observe a considerable power reduction when compared to the base algorithm HEFT. An average of 22.5% reduction in power consumption is observed.

Chapter 7

Conclusions and Future Work

This work reports a new scheduling algorithm which tries to minimize both makespan and power consumption. The LPHEFT algorithm has been shown to obtain the least finish time with an average power reduction of about 22.5% when compared to the base algorithm HEFT. The results are based on a experimental study using a large set of randomly generated graphs with various characteristics.

This work can be improved to obtain more reduction in power consumption by slowing down all the tasks which do not interfere the critical path. Also, power consumption should be calculated by slowing down all the tasks which comes at the expense of slightly higher finish time. This study can be used to find the optimum frequency of operation for the best possible minimum finish time and minimum power consumption.

Bibliography

[1] H. Topcuoglu, S. Hariri and Min-You-wu, "Peformance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, Vol 13, Issue 3, pp. 260-274, March 2002.

[2] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," *Proceedings of 1998 International Symposium on Low Power Electronics and Design*, pp. 197-202, August 1998.

[3] S. Baskiyar and C. Dickinson, "Scheduling directed a-cyclic task graphs on a bounded set of heterogeneous processors, using task duplication," *Elsevier (to appear).*

[4] H. Edwin and M. Sha, "Tutorial Scheduling for DSP and Embedded Systems with minimum code-size and low-power," Department of Computer Science, University of Texas at Dallas.

[5] S. Nikolaidis and Th. Laopoulos, "Instruction-level power consumption estimation of embedded processors for low-power applications," *Journal of Computing Standards and Interfaces*, Vol. 24, Issue 2, pp. 133-137, June 2002.

[6] Rita Yu Chen, M. J. Irwin and R. S. Bajwa, "Architecture-level power estimation and design experiments," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 6, Issue 1, pp. 50-66, January 2001.

[7] P. Landman and J. Rabaey, "Power estimation for higher level synthesis," *Proceedings of EDAC-EUROASIC*, pp. 361-366, February 1993.

[8] P. Landman, "Low power architectural design methodologies," *Ph. d Thesis, Memorandum no. UCB/ERL M94/62*, August 1994.

[9] P. Landman and J. Rabaey, "Black-box capacitance models for architectural power analysis," *Proceedings of 1994 International Workshop on Low Power Design*, pp. 165-170, April 1994.

[10] T. Ishihara and H. Yasuura, "Basic experimentation on accuracy of power estimation for CMOS VLSI circuits," *Proceedings of the 1996 International Symposium on Low Power Electronics and Design*, pp. 117-120, 1996.

[11] S. M. Kang, "Accurate simulation of power dissipation in VLSI circuits," *IEEE Journal of Solid-State Circuits*, Vol. 21, Issue no. 5, pp. 889-891, october 1986.

[12] G. Y. Yocoub and W. H. Ku, "An accurate simulation technique for short-circuit power dissipation based on current component isolation," *IEEE International Symposium on Circuits and Systems*, pp. 1157-1161, 1989.

[13] A. C. Deng, Y-C. Shiau and K-H. Loh, "Time domain current waveform simulation of CMOS circuits," *IEEE International Conference on Computer-Aided Design*, pp. 208-211, Nov 1988.

[14] R. Tjarnstrom, "Power dissipation estimate by switch level simulation," *IEEE International Symposium on Circuits and Systems*, pp. 881-884, May 1989.

[15] M. A. Cirit, "Estimating dynamic power consumption of CMOS circuits," *Proceedings of International Conference on Computer-Aided Design*, pp. 534-537, November 1987.

[16] F. N. Najm, "Transition Density: A new measure of activity in digital circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 12, Issue no. 2, pp. 310-323, February 1993.

[17] A. Ghosh, S. Devdas, K. Keutzer and J. White, "Estimation of average switching activity in combinational and sequential circuits," *Proceedings of 29th Design Automation Conference*, pp. 253-259, 1992.

[18] C. M. Huizer, "Power dissipation analysis of CMOS VLSI circuits by means of switch level simulation," *IEEE European Solid State Circuits Conference, Grenoble, France*, pp. 61-64, 1990.

[19] M. Xakellis and F. Najm, "Statistical estimation of the switching activity in digital circuits," *31st ACM/IEEE Design Automation Conference*, pp. 728-733, 1994.

[20] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, R. W. Brodersen, "Optimizing power using transformations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, Issue 1, pp. 12-31, January 1995.

[21] Jiong Luo and N. Jha, "Static and dynamic Variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," Proceedings of the 15th International Conference on VLSI Design, pp. 719-726, January 2002.

[22] Dongkun Shin and Jihong Kim, "Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," *Proceedings of the 2003 IEEE International Symposium on Low Power Electronics and Design*, pp. 408-413, August 2003.

[23] Jiong Luo and N. Jha, "Power-profile driven variable voltage scaling for heterogeneous distributed real-time embedded systems," *Proceedings of the 16th International conference on VLSI Design*, pp. 369-375, January 2003.

[24] C. H. Gebotys and R. J. Gebotys, "Power minimization in heterogeneous processing," *Proceedings of the 29th Annual Hawaii International Conference on System Sciences*,Vol. 1, pp. 330-337, January 1996.

[25] Yang yu and V. K. Prasanna, "Power-aware resource allocation for independent tasks in heterogeneous real-time systems," *Proceedings of the 9th International Conference on Parallel and Distributed Systems*, pp.341-348, December 2002.

[26] J. Pouwelse, K. Langandoen and H. Sips, "Dynamic voltage scaling on a lowpower microprocessor,"

[27] Shyam Chandra, "Reducing dynamic power consumption through voltage and frequency scaling," March 31st, 2005.

[28] S. Ranaweera, D. P. Agrawal, "A task duplication based scheduling algorithm for heterogeneous systems," *Proceedings of 14th International Symposium on Parallel and Distributed Processing*, pp. 445-450, May 2000.

[29] C. D. Polychronopoulos, "On program restructuring scheduling and communication for parallel processing systems," *CSRD-595*, University of Illinois at Urbana Champaign, August 1986.

[30] S. Uppaluri, B. Izadi, D. Radhakrishnan, "Low-power dynamic scheduling in heterogeneous systems," *Proceedings of the International Conference on Embedded Systems and Applications*, pp. 261-273, June 2003.

[31] Wen-Tsong Shine, and C. Chakrabarti, "Low-power scheduling with resources operating at multiple voltages," *IEEE Transactions on Circuits and Systems II:Analog and Digital Signal Processing*, Vol. 47, Issue 6, pp. 536-543, June 2000.

[32] A. Manzak, C. Chakrabarti, "Variable voltage task scheduling algorithms for minimizing energy/power," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, Vol. 11, Issue 2, pp. 270-276, April 2003.

[33] D. Zhu, R. Melhem, and B. R. Childers, "Scheduling with dynamic voltage/speed adjustment using slack reclamation in multiprocessor real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 14, Issue 7, pp. 686-700, July 2003.

[34] I. Ahmad, M. K. Dhodhi, R. Ul-Mustafa, "DPS: dynamic priority scheduling heuristic for heterogeneous computing systems," *IEEE Proceedings on Computers and Digital Techniques*, Vol. 145, Issue 6, pp. 411-418, November 1998.

[35] M. Wu and D. Gajski, "Hypertool: A programming aid for message passing systems," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, pp. 330-343, July 1990.

[36] Y. Kwok and I. Ahmad, "Dynamic Critical-Path Scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, Issue 5, pp. 506-521, May 1996.

[37] G. C. Sih, and E. A. Lee, "A compile-time scheduling heuristic for interconnection-constrained heterogenous processing architectures," *IEEE Transactions on Parallel and Distributed*, Vol. 4, Issue 2, pp. 175-187, February 1993.

[38] H. El-Rewini and T.G. Lewis, "Scheduling parallel program tasks onto arbitrary target machines," *Journal of Parallel and Distributed Computing*, Vol. 9, pp. 138-153, 1990.