

Energy Efficient Multiprocessor Task Scheduling under Input-dependent Variation

Jason Cong and Karthik Gururaj

Department of Computer Science, University of California, Los Angeles, CA 90095, USA

Email: {cong, karthikg}@cs.ucla.edu

Abstract — In this paper, we propose a novel, energy aware scheduling algorithm for applications running on DVS-enabled multiprocessor systems, which exploits variation in execution times of individual tasks. In particular, our algorithm takes into account latency and resource constraints, precedence constraints among tasks and input-dependent variation in execution times of tasks to produce a scheduling solution and voltage assignment such that the average energy consumption is minimized. Our algorithm is based on a mathematical programming formulation of the scheduling and voltage assignment problem and runs in polynomial time.

Experiments with randomly generated task graphs show that up to 30% savings in energy can be obtained by using our algorithm over existing techniques. We perform experiments on two real-world applications – MPEG-4 decoder and MJPEG encoder. Simulations show that the scheduling solution generated by our algorithm can provide up to 25% reduction in energy consumption over greedy dynamic slack reclamation algorithms.

Index Terms—DVS, scheduling, average energy consumption, precedence constraints, convex optimization

I. INTRODUCTION

ENERGY consumption is an important issue in designing battery operated embedded systems. The goal of designers is to create a system which consumes minimal energy and satisfies performance constraints at the same time. Multi-core chips are becoming increasingly popular for embedded systems. Examples include the ARM Cortex A-9 MPCore [23] and the MIPS 1004K [24]. One of the most effective design techniques for minimizing energy consumption of processors is dynamic voltage frequency scaling (DVFS), in which processor frequency and voltage can be adjusted depending on the workload of the processor. We first briefly examine techniques that have been proposed to exploit variation in execution time to reduce energy.

A. Uniprocessor Systems

The work in [7][9] [12][14][28] models the execution time of a task as a random variable and minimizes expected energy consumption on a single processor system. A heuristic is provided in [7] for obtaining a low-energy schedule. In [9][12], exact solutions are provided using convex optimization techniques; however, many of their assumptions, such as the ability to change the voltage to any arbitrary value at any point during the execution of a task, are not valid for practical systems. Many of these issues are addressed in [14] for uniprocessor systems. In [28], a mathematical formulation is presented to optimize the expected energy consumption (both dynamic and leakage) by using DVFS and Adaptive Body Biasing (ABB). However, a simple extension of the proposed method for multiprocessor systems leads to an exponential increase in complexity.

B. Multiprocessor Systems

Dynamic slack reclamation based techniques: In [13][8], the authors propose techniques by which the dynamic slack is distributed among the remaining tasks. While the work in [8] does not consider precedence constraints among tasks, a list scheduling heuristic is used in [13] for tasks with precedence

constraints. Such online techniques are constrained to be relatively simple and fast.

Schedule Table based: The idea of using heuristic to build a schedule table at design time was proposed in [5][3] for scheduling and voltage assignment for *conditional task graphs (CTG)*. However, the proposed techniques are restricted to CTGs.

Expected energy minimization: A highly complex, non-linear integer programming based method is proposed in [1] for task mapping and scheduling in multiprocessor systems. In [2], the authors attempt to balance the expected energy consumption across processors by partitioning a set of independent tasks. In [29], a dynamic programming based method is used to minimize expected energy consumption. However, the proposed method is exponential in complexity for multiprocessor systems.

The primary contributions of this paper can be stated as follows.

1) We propose a mathematical programming formulation based technique for scheduling tasks on *DVFS capable multiprocessor systems*, which takes into account *input-dependent* variation in execution time of tasks to reduce *average energy consumption* subject to a specified latency constraint. Our technique is capable of handling *precedence constraints* among tasks.

2) Our technique runs in *polynomial time* for multiprocessor systems; the solution is optimal for tree like task graphs. This is achieved by a novel pruning method during the formulation phase that avoids the enumeration done in [28] [29].

Our algorithm constructs a schedule table at design time to provide multiple scheduling options for each task. While complex algorithms can be used to build the schedule table at design time, the only extra processing that a system needs to perform at run-time is a table look-up.

The rest of the paper is organized as follows: Section II describes our task graph and processor models. Section II.C provides the problem statement. In Section III, our formulation is presented by which variation in execution-time can be exploited; Experiments performed on randomly generated task-graphs as well as two real-world applications are described in Section V.

II. PRELIMINARIES AND PROBLEM STATEMENT

A. Processor Model

We assume that the number of processors is restricted to be no more than a certain number P and that the voltage of every processor in the system can be set independently to any value within a given range $[V_{lower}, V_{upper}]$ at run-time. The overhead for switching between voltages is assumed to be negligible compared to the execution time of individual tasks. To model the relation between energy, voltage and clock frequency, we use well known equations for CMOS logic [15]:

$$f = C_1 \frac{(V - V_{th})^\alpha}{V} \quad (1)$$

$$E = C_2 W V^2 \quad (2)$$

where f is the clock frequency, V is the supply voltage, W is the number of cycles taken by a task (or the workload). C_1 and C_2 are constants. α is a constant between 1 and 2. In this paper, we approximate the frequency to be a linear function of supply voltage [6][11]. Thus, we can see that energy can be modeled as a non-increasing convex function of the clock period as shown in Equation (3). From this point onwards in the paper, we model the energy consumption as a function of clock period cp , instead of the clock frequency. Our method is applicable to any convex, non-increasing function of clock period.

$$E = C_3 W f^2 = \frac{C_3 W}{(cp)^2} \quad (3)$$

B. Application Model

We assume that an application is represented as a directed acyclic graph (DAG) $G(V, E)$ called the task graph. Each node $v \in V$ represents a task that has to be executed on a single processor *without preemption*. Moreover, every task is restricted to run at a *single voltage*. A directed edge $(u \rightarrow v) \in E$ represents a precedence constraint between the tasks represented by nodes u and v . Each precedence constraint is also associated with a certain *communication delay* between the two tasks. In our model, we assume that the execution time of the source task (of an edge) includes the time for communication of data to its successors. A latency constraint on G is a timing constraint which specifies the allotted time L within which the execution of G should be completed. We assume that G will be re-executed every L time units.

We now define a set of parameters associated with each task v :

- **WorkLoad(v, I)** of a task v is defined as the number of clock cycles taken by v to complete execution. Note that the workload is not the execution time because the execution time for a fixed workload varies with the clock frequency. Also, the *WorkLoad* depends on input I .
- **WCW(v)** (*Worst-Case Workload*) of a task v is defined as the maximum workload of the task v .

Motivating example

Consider the task graph G shown in Figure 1. All four tasks are assumed to be identical and the *WorkLoad* vector for the tasks is shown in Figure 1. The worst-case workload, $WCW(v)$, is assumed to be 100 cycles for all the tasks. The probability that the workload of a task is no greater than 75 cycles is 0.7 and that the workload is between 75 and 100 cycles is 0.3.

Suppose we are given a latency constraint of 450ns for this example on a 2 processor system – P_1 and P_2 . Using the model described in Section II.A, we can determine that when all the tasks run for 75 cycles, the worst-case scheduling produces a solution which consumes 44% more energy than the optimal solution.

- **Start Cycles Elapsed ($SCE(v)$)** for a task v is the number of clock cycles elapsed when all predecessors of v have completed. For tasks with only primary inputs, $SCE(v)$ is 0. For other tasks,

$$SCE(v) = \max_u (SCE(u) + WorkLoad(u)) \text{ where } u \in \text{predecessors}(v).$$

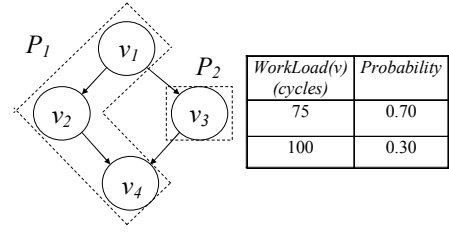


Figure 1: Example task graph and workloads

End Cycles Elapsed ($ECE(v)$) for a task v is the number of clock cycles elapsed when v finishes.

$$ECE(v) = SCE(v) + WorkLoad(v)$$

Note $SCE(v) = \max_u (ECE(u))$ where $u \in \text{predecessors}(v)$.

In the above example, if v_1 , v_2 and v_3 take 75 cycles each, $SCE(v_4)$ is 150 cycles and $ECE(v_4)$ is 225 cycles.

- A **schedule table** is a table that has for each task v , a vector of tuples $\langle s_{ce_{v,i}}, cp_{v,i}, s_{v,i} \rangle$ where $(s_{ce_{v,1}}, s_{ce_{v,2}}, \dots, s_{ce_{v,K}})$ is the list of possible values of $SCE(v)$ sorted in increasing order, $cp_{v,i}$ is the clock period at which the task v is run and $s_{v,i}$ is the start time of v when the value of $SCE(v)$ is $ce_{v,i}$.

Table 1: Sample schedule table

Task	Entry 1	Entry 2	Entry 3
v_1	$\langle 0, 1.5, 0 \rangle$	-	-
v_2	$\langle 75, 1.68, 112.5 \rangle$	$\langle 100, 1.5, 150 \rangle$	-
v_3	$\langle 75, 1.68, 112.5 \rangle$	$\langle 100, 1.5, 150 \rangle$	-
v_4	$\langle 150, 2.1, 238.5 \rangle$	$\langle 175, 1.69, 280.5 \rangle$	$\langle 200, 1.5, 300 \rangle$

Continuing to use our example in Figure 1, we show a sample schedule table in Table 1 when the latency constraint is set to 450ns. Suppose at run-time, $SCE(v_4)$ is computed to be 175 cycles, then the schedule table matches this value to the entry $\langle 175, 1.69, 280.5 \rangle$ (Row 4, Column 2 in Table 1). Thus, the task v_4 is scheduled to start at time 280.5ns and run with a clock period of 1.69 ns. This implies that task v_4 will complete at most by time 450ns because the worst case workload of v_4 is 100 cycles.

C. Problem Statement

At run-time, each task u propagates the value of $ECE(u)$ to its successors. Every task v computes the value of $SCE(v)$ and performs a table look-up to determine the start time of v and the clock period to use. This immediately implies that considering different values of $SCE(v)$ provides us with a way of exploiting workload variation to generate different scheduling solutions and clock period assignments for task v (seen in Table 1). With this in mind, we formally state the problem:

Given a task-graph $G(V, E)$, the *WorkLoad* distribution associated with each task $v \in V$, and a latency constraint L , the goal of the scheduling algorithm is to construct a schedule table T such that the average energy consumption is minimized and the latency and precedence constraints are satisfied for any combination of workloads of the tasks of G .

III. VAR-TB - VARIATION AWARE TIME BUDGETING

Our algorithm is divided into two phases – the first phase assigns tasks to processors and the second phase determines the start time and voltage assignment for each task.

A. Task assignment heuristic

The problem of resource-constrained energy minimization subject to latency constraints has been proved to be NP-Hard [8]. We use a priority based heuristic to assign the tasks to a set of P processors. The highest priority task is scheduled to run on the first processor that is ready to accept a new task. In our experiments, we use the difference between the ALAP and ASAP time to decide the task priorities. These times are computed by assuming that all processors run at their highest frequency and all tasks run at their worst-case. After a task is assigned, the ASAP and ALAP times for the remaining tasks are re-computed. We insert fake edges between consecutive tasks running on the same processor to maintain task order.

B. Task scheduling and voltage assignment

We first present the variable organization in our mathematical formulation for the scheduling and voltage assignment problem. We contrast our approach with 2 existing works and explain how our formulation avoids enumeration of a large number of task workloads. In Section IV.B, we prove why our approach can run in polynomial time and yet provide optimal solutions for certain kinds of task graphs.

C. Mathematical formulation of VAR-TB

For each task v , we introduce a start time variable $s_{v,i}$ and clock period variable $cp_{v,i}$ for every value of $SCE(v)$ (given by $sce_{v,i}$). For every predecessor u of v , we introduce a finish time variable $-f_{u,j}$ for every value of $ECE(u)$ (given by $ece_{u,j}$). Our formulation consists of the following constraints:

$$s_{v,i} \geq 0 \quad (1)$$

$$s_{v,i} + WorkLoad_k(v) * cp_{v,i} \leq L \quad (2)$$

$$s_{v,i} \geq f_{u,j} \quad \text{iff} \quad ece_{u,j} \leq sce_{v,i} \quad (3)$$

$$cp_{lower} \leq cp_{v,i} \leq cp_{upper} \quad (4)$$

The first 2 constraints impose non-negativity and latency constraints. The third constraint imposes precedence between u and v . Note that the only variables in the above problem are the start times, finish times and clock periods. Constraint 2 is repeated for every value of $WorkLoad(v)$. Constraint 4 imposes bounds on the clock period variable.

We explain precedence constraints with the example from Figure 1. Task v_1 can complete after 75 cycles or 100 cycles ($ECE(v_1)$). Thus, task v_1 has 2 finish time variables $-f_1$ and $-f_2$ associated with 75 and 100 cycles respectively. Task v_2 can start after 75 cycles or 100 cycles ($SCE(v_2)$). Hence, v_2 has 2 start time variables $-s_1$ and $-s_2$. Hence, the precedence constraints are given by:

$$s_1 \geq f_1 \quad s_2 \geq f_2$$

Note that *no constraints exist* between s_1 and f_2 because task v_2 starts at time s_1 only when task v_1 has completed within 75 cycles. Thus, task v_1 would have completed by time f_1 . For task v_2 , s_2 is the worst-case start time. If v_1 completes by 75 cycles, v_2 can start at time s_1 (which is possibly less than s_2), thus allowing our method to exploit variation in $WorkLoad(v_1)$.

We make the following observations about our method:

1. *Feasible schedule*: The proposed constraints are safe because for a task v , for every value of $SCE(v)$, there always exists a start time that is greater than the finish time of its predecessors and

satisfies latency constraint (assuming a feasible schedule exists).
 2. *Exploits variation* in workload of tasks
 3. *Avoids enumeration of all workloads of all tasks*: Note that in our method, the only variables that are considered while specifying the precedence constraints are the finish time variables associated with the predecessor tasks. The approaches in [28][29] run an optimization pass for every possible combination of start times of all tasks with multiple predecessors. As an example, consider a task graph with 10 tasks each of which can have 4 different start times. The methods in [28][29] will require 4^{10} optimization passes each involving approximately 10 variables and m constraints (where m is the number of edges). Our method on the other hand requires a *single* optimization pass with around 40 variables and $(4^2)*m$ constraints. Note that [28][29] work well for single processor systems because the start time of only a single task needs to be fixed for each optimization phase.

Objective Function

The objective function represents the average energy consumption and is given by Equation (5). $prob_{v,i}$ is the probability, $cp_{v,i}$ is the clock period for task v when $SCE(v)$ is $ece_{v,i}$ and C_v is a constant.

$$\sum_{v \in V} \sum_{i=1}^K \sum_{n=1}^M C_v * prob_{v,i} * WorkLoad_n(v) / (cp_{v,i}^2) \quad (5)$$

We observe that the only variable in the objective function is the clock period $cp_{v,i}$. Also, the objective function is convex separable and non-increasing. The probability information is obtained by application profiling.

IV. IMPROVING THE SCHEDULING ALGORITHM

A. Restricting the number of $SCE(v)$ entries per task

1) The number of values of $SCE(v)$ can be large even for small task graphs. To avoid this, we limit the number of values of $SCE(v)$ per task to be less than a constant K . In our approach, we select the K values so that the area under the probability distribution curve for $SCE(v)$ is split into K equal regions.

B. Time complexity

Since the number of values of $SCE(v)$ per task is always less than a constant K , we ensure that the number of variables associated with each task is no more than $O(K)$ and the number of constraints associated with each precedence edge is no more than $O(K^2)$. Thus, the number of variables is linear with respect to the number of tasks ($O(Kn)$) and the number of constraints is linear with respect to the number of edges ($O(K^2m)$) (where m is the number of precedence edges). In our experiments, we set K to 16 and obtain good energy savings for graphs with as many as 95 tasks.

THEOREM 1: Since all the constraints are linear and the objective function is separable convex, algorithm *VAR-TB* produces a schedule table to minimize average energy consumption subject to latency and precedence constraints in polynomial time [16].

We state two theorems which prove why our method is optimal for certain kinds of task graphs. Proofs are omitted due to the page limit. The main assumption is that dynamic energy is a convex, non-increasing function of clock period.

THEOREM 2: Given a chain of tasks ($T_1, \dots, T_i$) to be executed sequentially and a latency constraint L , the value of the optimal energy consumption for a single run is *dependent only on the total number of cycles consumed during the run and independent of the cycles consumed by the individual tasks*.

Theorem 2 also holds for task graphs that are trees but with a small modification.

THEOREM 3: Given a task T_i in a tree G , the optimum energy consumption for the sub-tree consisting of T_i and predecessors depends only on the number of cycles consumed on length of each path to T_i from the primary inputs in G .

For DAGs, such a claim does not hold true because of reconvergent fan-outs. However, *VAR-TB* is still an effective technique for DAGs.

C. Online algorithm

We introduce a simple online phase to reclaim slack. When a task starts, it consumes all dynamic slack available from its predecessors and completes within its assigned finish time. Thus, the complexity of the online algorithm is $O(1)$.

V. EXPERIMENTAL RESULTS

We compare the results of our algorithm *VAR-TB* with a DVS algorithm which considers the worst-case only (*WC-DVS*), a worst-case DVS algorithm which performs dynamic slack reclamation [27] (*WC-Reclaim*) and a hypothetical method that can accurately determine the workloads of each task before hand and performs optimal scheduling (*Oracle*). *WC-Reclaim* allocates dynamic slack proportional to worst-case *WorkLoad(v)*.

A. Random task-graphs

We run our scheduling algorithm on several random task graphs generated from TGFF [4] with a resource constraint of 4 processors. We add the probability distribution of the task workloads as task attributes in the TGFF description. We obtain the probability distribution of *SCE(v)* every task by performing a number of Monte-Carlo simulations (10,000 in our experiments). After the optimization step, we use Monte-Carlo simulations to compute the energy consumption and determine the average energy consumption.

We compute the energy savings obtained by each algorithm and plot the savings as a percentage of energy consumption of the worst-case algorithm in Figure 2. The plot depicts the energy savings over algorithm *WC-DVS* for different task graphs. As can be seen, the simple algorithm *WC-Reclaim* performs much better than *WC-DVS* suggesting that the task graphs have significant variation in workloads to exploit. Our algorithm *VAR-TB* performs significantly better than algorithm *WC-Reclaim*; on an average the solutions provided by *VAR-TB* are 20-25% better than algorithm *WC-Reclaim*. Finally, we analyze the scheduling solution obtained by algorithm *Oracle*. As we can see, *VAR-TB* performs well compared to *Oracle* – with the maximum deviation being 20% and the average deviation being 15%. Moreover, the performance of *VAR-TB* does not degrade with increase in the number of tasks in the task graph. Additionally, *VAR-TB* completes within 90 seconds for all task graphs, when executed on an Intel Xeon 2 GHz processor.

We have not plotted the results of algorithm *DynOpt* [29]. We discovered that the pruning step for *DynOpt* proposed in [29]

causes scheduling solutions that are inferior locally but optimal globally to be discarded because of which *DynOpt* performs worse than *WC-Reclaim* in some cases. Moreover, *DynOpt* and [28] will require up to 5^{20} optimization passes for the medium sized task graphs.

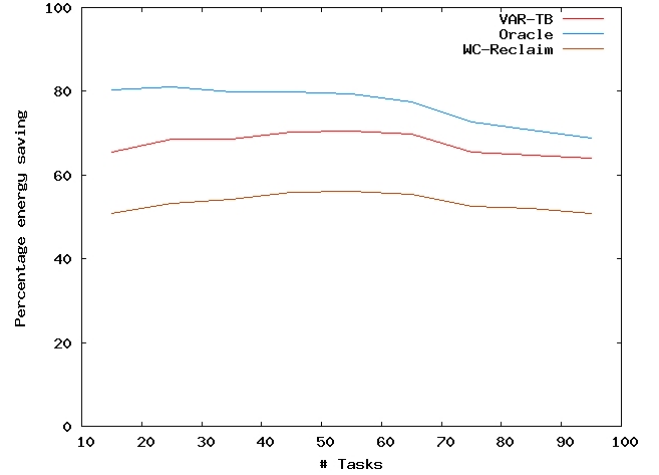


Figure 2: Energy savings over *WC-DVS*

B. Real-world Benchmarks

We perform experiments on two real-world applications – MPEG-4 decoder and Motion-JPEG (MJPEG) encoder. For these benchmarks, we apply dynamic slack reclamation after every algorithm. We evaluate two different schemes – *W-Aware* in which the workload of a task can be predicted from its input values and *W-Unaware* where the workload of the task cannot be predicted from its input values.

Processor Architecture

The processor cores in our experiments are modeled after the Intel XScale processor has 7 voltage levels as given in [26]. All processors share a 1 MB on-chip L2 cache through a common bus and implement a MESI cache-coherence protocol. Table 2 lists the relevant parameters. We use SESC [17] to simulate our multi-processor system and obtain profiling information (probabilities and *WorkLoad* values).

For measuring dynamic energy consumption of on-chip components, we use Wattch [19] (integrated with SESC). Energy values for read-write operations for caches and SRAM-based array structures (TLB, ROB) are obtained from Cacti [18] for 90nm technology. For other processor components (such as ALU, decoder etc), energy numbers are obtained from Wattch for 180nm technology and scaling factors are applied as in [20]. Inter-processor communication is carried out through blocking FIFOs with bandwidth of 300 MB/s, which are similar to the Fast-Simplex Link (FSL) [21] provided by Xilinx.

Table 2: Processor characteristics

Processor	Single issue, 5 stage pipeline, FPU
Registers	64 (32-bit)
Technology	90nm
L1 cache	16K I and D caches – 4 way set-associative
L2 cache	1MB L2 cache – 16 way set associative

1) MPEG-4 decoder

A simplified task graph for the MPEG-4 decoder provided by Xilinx [22] is shown in

Figure 3(a). The main components of the decoder are the Parser (*P*), Copy-Controller (*CC*), Inverse-DCT (*IDCT*), Motion

Compensation (*MC*) and Texture Update (*TU*). While *IDCT* does not show significant variation across different runs, the *P*, *CC*, *MC* and *TU* modules exhibit significant variation (Figure 4). By unrolling the loop for one macroblock and performing loop pipelining (Figure 3(b)), a parallel version of the decoder was implemented on a 7 processor system. A performance requirement of 20 frames/second was imposed on the decoder leading to a latency constraint of 500us per macroblock.

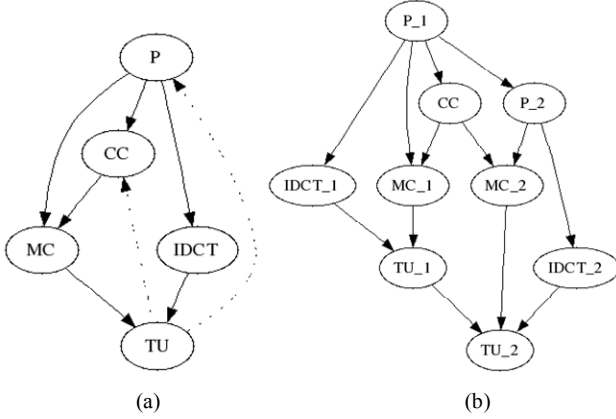


Figure 3: MPEG-4 decoder (a) Single iteration (b) Unrolled task graph for two iterations

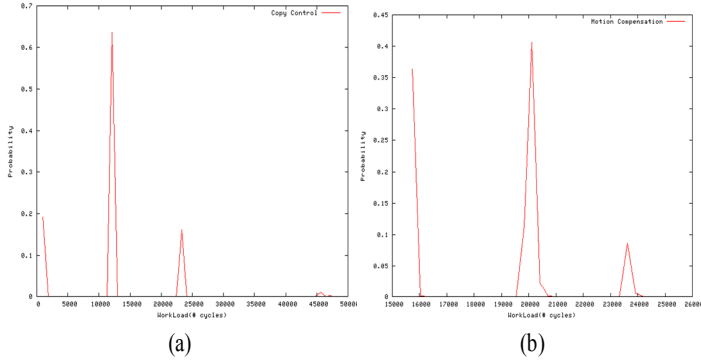


Figure 4: Probability distribution of Workload (a) Copy Control (b) Motion Compensation

We measure the energy reduction that our algorithm provides over the *WC-Reclaim* algorithm. Moreover, we measure how the quality of the solution is affected by the number of *SCE(v)* values per task. The results are summarized in Figure 5. The two curves represent the energy consumption of the schedule produced by the two schemes – *W-Aware* (red curve) and *W-Unaware* (blue curve). From the plot, it is clear that our algorithm can achieve significant savings over the *WC-Reclaim* – up to 40% for *W-Aware* and up to 22% for *W-Unaware*.

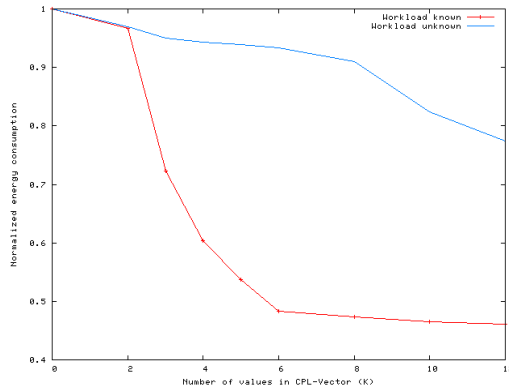


Figure 5: Normalized energy consumption for *W-Aware* and *W-Unaware* schemes for MPEG-4 decoder.

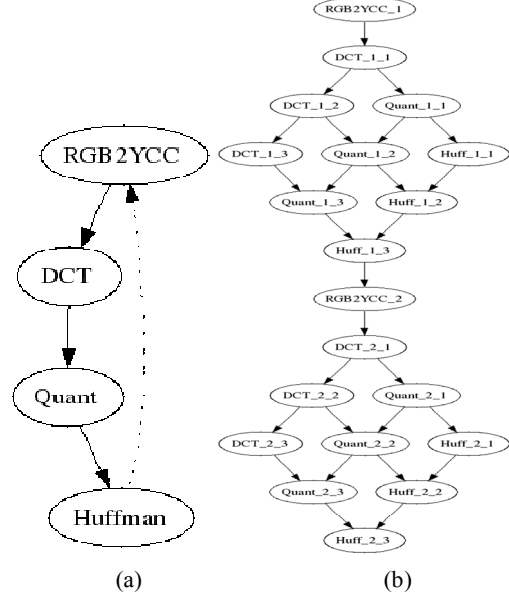


Figure 6: MJPEG encoder task-graph (a) Single block (b) Unrolled task graph for processing two successive MCUs

2) MJPEG encoder

We apply our algorithm to the MJPEG encoder [25] for which the task graph is shown in Figure 6(a). To process each Minimum Coded Unit in the incoming stream, we implement a pipelined version of the encoder using a four processor system where each task in Figure 6(a) is assigned to a processor. Of the four tasks, only the Huffman encoding task shows significant variation (Figure 7). We perform loop unrolling to obtain the graph in Figure 6(b), on which we apply our method.

We compare the energy savings obtained from our algorithm against the *WC-Reclaim* algorithm. As explained before, we consider two cases – *W-Aware* and *W-Unaware* and vary the number of *SCE(v)* entries. We show the energy consumption of the two schemes normalized to the energy consumption obtained by the *WC-Reclaim* algorithm in Figure 8. For the *W-Aware* scheme (red curve), we see that we can obtain up to 14% savings in energy whereas for the *W-Unaware* scheme (blue curve), the maximum savings we obtain is 4%. The small savings is because of the low variation seen in this benchmark.

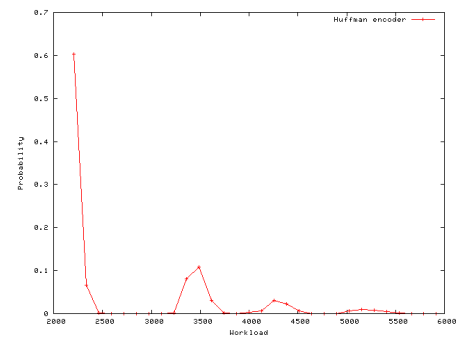


Figure 7: Workload variation of Huffman encoding module

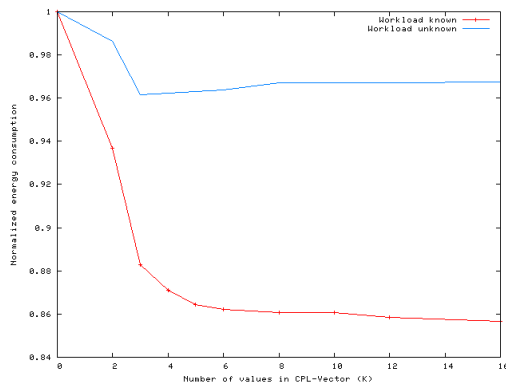


Figure 8: Normalized energy consumption for *W-Aware* and *W-Unaware* schemes for MJPEG encoder.

VI. CONCLUSIONS

We present a mathematical formulation which can exploit variation in workloads of tasks in applications to provide a low-energy scheduling solution. Our algorithm is capable of handling precedence constraints and multiple processors. We show that the schedule table can be generated in polynomial time and is optimal for special cases. Finally, our experiments show that significant energy savings can be obtained by our scheduling algorithm over dynamic slack reclamation methods.

VII. ACKNOWLEDGEMENT

This research is partially supported by MARCO Gigascale Systems Research Center (GSRC), the NSF grant CNS-0647442 and the NSF grant CNS-0725354.

REFERENCES

- [1] M. Lombardi and M. Milano, "Stochastic allocation and scheduling for conditional task graphs in MPSoCs," In Technical Report 77 LIA-003-06.
- [2] Changjiu Xian, Yung-Hsiang Lu and Li Zhiyuan, "Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time," in *Proc of 44th ACM/IEEE Design Automation Conference, 2007*, pp.664-669.
- [3] D. Shin and J. Kim, "Power-aware scheduling of conditional task graphs in real-time multiprocessor systems," in *Proceedings of the 2003 international Symposium on Low Power Electronics and Design*, pp 408-413, 2003.
- [4] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proc. of Int. Workshop Hardware/Software Codesign*, pp. 97-101, March 1998.
- [5] D. Wu, B.M Al-Hashimi and P. Eles, "Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems," in *IEEE Proceedings of Computers and Digital Techniques*, vol.150, no.5, pp. 262-273, 2003.
- [6] F. Gruian, "Hard real-time scheduling for low-energy using stochastic data and DVS processors," In *Proceedings of the 2001 international Symposium on Low Power Electronics and Design* pp 46-51, 2001.
- [7] F. Gruian and K. Kuchcinski, "Uncertainty-based scheduling: energy-efficient ordering for tasks with variable execution time," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pp. 465-468, 2003.
- [8] J. J. Chen, H. Hsu, K. Chuang, C. Yang, A. Pang and T. Kuo, "Multiprocessor energy-efficient scheduling with task migration considerations," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, pp 101- 108, 2004.
- [9] L. Leung, C. Tsui and X. S. Hu, "Exploiting dynamic workload variation in low energy preemptive task scheduling," in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 634- 639, 2005
- [10] A. Wächter and L. T. Biegler, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming*, 106(1), pp. 25-57, 2006.
- [11] P. Chowdhury and C. Chakrabarti, "Static task-scheduling algorithms for battery-powered DVS systems," in *IEEE Transactions on Very Large Scale*

- Integration (VLSI) Systems*, vol.13, no.2, pp.226-237, 2005
- [12] R. Xu, D. Mossé, and R. Melhem, "Minimizing expected energy in real-time embedded systems," in *Proceedings of the 5th ACM international Conference on Embedded Software*, pp. 251-254, 2005.
- [13] D. Zhu, R. Melhem, and B. Childers, "Scheduling with dynamic voltage /speed adjustment using slack reclamation in multi-processor realtime systems", *Real-Time Systems Symposium*, pp 84-94, 2001.
- [14] Ruibin Xu, Rami Melhem and Daniel Mossé, "A Unified Practical Approach to Stochastic DVS Scheduling," in the *7th ACM International Conference on Embedded Software*, pp 37-46, 2007.
- [15] T. Mudge, "Power: a first-class architectural design constraint," *Computer*, vol.34, no.4, pp.52-58, Apr 2001.
- [16] J. Cong, W. Jiang, M. Potkonjak and Z. Zhang, "Scheduling with Integer Delay Budgeting for Low-Power Optimization", in *Proceedings of ASP DAC*, pp 22-27 2008.
- [17] SESC simulator, <http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/>
- [18] P. Shivakumar and P.N. Jouppi, "CACTI 3.0: An integrated cache timing, power and area model," HP Labs Technical Report, WRL-2001-2, 2001.
- [19] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A framework for architectural-level power analysis and optimizations," In *International Symposium on Computer Architecture*, pp 83-94, 2000.
- [20] Hangsheng Wang, Li-Shiuan Peh, Sharad Malik. "A Technology-Aware and Energy-Oriented Topology Exploration for On-Chip Networks". In *Proceedings of Design, Automation and Test in Europe*, pp 1238-1243, 2005.
- [21] H.P. Rosinger, "Connecting Customized IP to the MicroBlaze Soft Processor using the Fast Simplex Link (FSL) Channel," www.xilinx.com/support/documentation/application_notes/xapp529.pdf
- [22] P. Schumacher and W. Chung, "FPGA-based MPEG-4 codec," In *DSP Magazine*, pp. 8-9, 2005.
- [23] ARM MPCore www.arm.com/products/CPUs/ARMCortex-A9_MPCore.html
- [24] MIPS 1004K <http://www.mips.com/products/cores/32-bit-cores/mips32-1004k/#>.
- [25] Abhijit Davare, Jike Chong, Qi Zhu, Douglas Michael Densmore and Alberto L. Sangiovanni-Vincentelli. "Classification, Customization, and Characterization: Using MILP for Task Allocation and Scheduling". *Technical report, University of California, Berkeley, UCB/EECS-2006-166*, December, 2006.
- [26] R. S. Kihwan Choi and M. Pedram, "Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times," in *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, January 2005.
- [27] N. AbouGhazaleh, D. Mosse, B. Childers, and R. Melhem, "Toward the placement of power management points in real time applications," in *Proceedings of the Workshop on Compilers and Operating Systems for Low Power*, 2001.
- [28] A. Andrei, M. T. Schmitz, P. Eles, Z. Peng, and B. M. Hashimi, "Quasi-static voltage scaling for energy minimization with time constraints," In *DATE*, pp. 514-519, 2005.
- [29] M. Qiu, C. Xue, Z. Shao and H. M. Sha, "Energy minimization with soft real-time and DVS for uniprocessor and multiprocessor embedded systems," In *DATE*, pp. 1-6, 2007.