Network Flow Techniques for Dynamic Voltage Scaling in Hard Real-Time Systems

Vishnu Swaminathan and Krishnendu Chakrabarty, *Senior Member, IEEE*

***Abstract*—Energy consumption is an important performance parameter for portable and wireless embedded systems. However, energy consumption must be carefully balanced with real-time responsiveness in hard real-time systems. In this paper, we present two offline dynamic voltage scaling (DVS) schemes for dynamic power management in such systems. In the first method, we develop a generalized network flow (GNF) model for the uniprocessor DVS problem and solve it optimally using an efficient network flow algorithm. The proposed method outperforms existing DVS schemes for several popular embedded processors where the number of processor speeds is limited to a few values. The solutions for the GNF model provide theoretical lower bounds on energy consumption using DVS in hard real-time systems. We also describe a minimum-cost network flow model whose solutions are near-optimal. The minimum-cost models perform at par with competing methods for processor models with a large range of operating voltages, and better than them for processor models with a limited set of operating voltages.**

***Index Terms*—Deadlines, dynamic power management, embedded systems, low-energy, low-power, network flow models, real-time operating systems.**

I. Introduction

ENERGY consumption is an important performance parameter for battery-operated embedded systems. An effective approach to power reduction in embedded systems is based on *dynamic voltage scaling* (DVS), a runtime technique that exploits the quadratic dependence of power consumption of a CMOS processor on its operating voltage. However, a reduction in operating voltage results in a drop in the CPU operating frequency and an increase in the execution times of application tasks. Therefore, to ensure that no task deadlines are missed, DVS must be performed judiciously in hard real-time systems.

In this paper, we present two offline DVS schemes to minimize the energy consumed by a processor executing a set of jobs in a hard real-time system. In safety-critical applications, offline scheduling is often preferred over priority-based runtime scheduling to achieve high predictability [29]. In systems where offline scheduling is used, the problem of scheduling tasks for minimum energy can be readily addressed through the techniques presented here. We first model the DVS problem as a generalized network flow (GNF) graph and use the generalized network simplex algorithm [1] to solve it. We consider a scenario where a set of jobs execute on a processor that is capable of running at a limited number of speeds. Common examples of such processors are the Transmeta Crusoe [5] and the AMD K-6 [2]. We generate an offline job schedule and identify the speeds (and corresponding voltages) at which each job must be executed such that: 1) the total energy consumed by the set of jobs is minimized and 2) no deadlines are missed. We also prove that the solutions of the GNF models result in the theoretical lower bounds on energy consumption using DVS under the constraint that voltage switching can only be carried out at task boundaries. Network flow algorithms are extremely efficient; therefore, our models can potentially scale well to real-life task sets for a small number of processor speeds.

A limitation of the GNF model, however, lies in its inability to scale to large task sets using a processor model with a large number of operating voltages [28]. In this paper, we address this issue by developing a minimum-cost network flow model for the DVS problem. The minimum-cost models are solved using a fast polynomial-time cost-scaling algorithm to generate near-optimal solutions. We demonstrate the effectiveness of both network flow techniques by comparing them to some of the most efficient offline DVS schemes presented thus far in the literature [22], [30]. Our results also suggest that the number of available processor speeds is an important consideration for the design and evaluation of DVS algorithms.

The rest of the paper is organized as follows. In Section II, we review related prior work. In Section III, we describe DVS in greater detail, present the problem statement, and briefly review network flow concepts. In Section IV, we describe and analyze the GNF model assuming that the processor can operate at only two discrete voltages. In Section IV-B, we briefly describe a GNF model for more than two discrete speeds. In Section V, we describe and analyze the minimum-cost network flow graph for the DVS problem. In Section VI, we present our experimental results, and finally, in Section VII, we summarize the paper.

II. Related Prior Work on DVS

A significant body of research has been carried out on the problem of minimizing processor energy consumption in hard real-time systems. An offline, preemptive DVS scheme for minimum energy is described in [30]. An online, preemptive DVS algorithm is presented in [7]. Integer linear-programming models for statically assigning voltages to tasks are described in [11]. The authors also show that any processor frequency can be emulated using the two frequencies closest to it and on either

Manuscript received May 31, 2003; revised August 28, 2003 and December 24, 2003. This paper was supported in part by DARPA under Grant N66001-001-8946, in part by a graduate fellowship from the North Carolina Networking Initiative, and in part by DARPA and monitored by the Army Research Office under Emergent Surveillance Plexus MURI Award DAAD19-01-1-0504. A preliminary version of this paper appeared in a shortened form in the *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 21–25, San Jose, CA, November 2003. This paper was recommended by Associate Editor M. F. Jacome.

The authors are with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: krish@ee.duke.edu).

Digital Object Identifier 10.1109/TCAD.2004.833621

0278-0070/04$20.00 © 2004 IEEE

side of it. An online DVS technique based on the well-known rate-monotonic (fixed-priority) algorithm is presented in [24]. An offline near-optimal fixed-priority scheduling algorithm is described in [22]. Lagrange multipliers are used for low-energy task scheduling in [19]. DVS for real-time tasks with nonpreemptable sections has been studied in [31].

An important advantage of intertask DVS over intratask DVS lies in its simplicity of implementation. Although intratask DVS indeed results in greater energy savings than intertask DVS, a compiler-driven approach is necessary to fully utilize its energy-saving potential. Online intratask DVS methods, such as [17], result in increased scheduler complexity (owing to the numerous voltage-scaling checks that must be performed by the scheduler) and, therefore, increased scheduler overhead. As discussed in [17], the amount of overhead depends on the granularity of scheduler invocation within a single task.

On the other hand, compiler-driven intratask DVS methods such as in [3], [8], [9], [25], and [26] result in greater energy savings than online intratask DVS methods. Current-day practice consists of inserting "checkpoints" or voltage-switching code into the application program [3], [9], [26] at strategic points. These types of methods require OS modifications (when checkpoints are used [3]) and power-aware compilers [9], [26]. Thus, these methods have a significant impact on the portability of the application and cannot be used where power-aware compilers and/or software tools [26] are unavailable. These drawbacks have been recognized in prior work [3], [9], [26]. Therefore, in this paper, we have focused on intertask voltage scaling. Finally, while DVS algorithms for real-time multiprocessor systems have also been discussed in detail in [15], [32], we limit ourselves to uniprocessor systems in this work.

A drawback of many of the single-processor DVS methods listed above is that they either assume a processor model with a large range of operating frequencies and voltages or a specific scheduling strategy. For example, [22], [24] perform fixed-priority scheduling for a processor with a frequency range of 100 MHz to 8 MHz, adjustable in increments of 1 MHz, while the method described in [30] performs dynamic-priority scheduling with a continuously variable voltage spectrum. In this paper, we show that when such a fine-grained or continuous range of frequencies is unavailable, the efficiency of the above methods degrades significantly. Moreover, popular embedded processors such as the Transmeta Crusoe processor (frequency range of 300–800 MHz, adjustable in increments of 100 MHz [5]) and the AMD K-6 (300–500 MHz, adjustable in 50 MHz increments [2]) provide only a limited set of available speeds at which tasks can be executed. In this paper, we show that with this restriction on processor speeds, network flow techniques can be applied to solve the DVS problem optimally in reasonable periods of time. These network flow models are independent of the number of available processor speeds and scheduling strategy. Therefore, the DVS problem can be solved *optimally* using network flow techniques.

III. Background

A. DVS

DVS refers to the runtime variation of processor supply voltage to obtain quadratic reductions in energy. The dynamic power consumption in CMOS circuits is characterized by the following:

$$P_{\text{dynamic}} = KC_{\text{out}}V_{\text{dd}}^2 f \tag{1}$$

where P_{dynamic} is the dynamic power consumption of a logic gate, K is the average number of transitions made by the gate in a single clock cycle, C_{out} is the switching capacitance of the gate, V_{dd} is the value of the supply voltage, and f is the frequency of operation. Clearly, decreasing the values of any of these parameters results in reduced power consumption. Since P_{dynamic} varies quadratically with V_{dd}, supply voltage reduction is the most effective approach to decrease power consumption of a CMOS gate. However, a decrease in the supply voltage results in a corresponding increase in gate delay. This relationship between supply voltage and gate delay is reflected in the following:

$$T_d = \frac{C_{\text{out}}V_{\text{dd}}}{\eta(W/L)(V_{\text{dd}} - V_t)^{\alpha}} \tag{2}$$

where $\alpha(1 \leq \alpha \leq 2)$ is referred to as the velocity saturation index [23], T_d is the delay of the CMOS gate, W and L are the widths and lengths of the transistors, respectively, C_{out} is the output capacitance of the gate, and V_t is the threshold voltage of the transistors.

The execution time of an application task is proportional to the sum of the gate delays on the critical path in a CMOS processor. Since T_d varies (almost) linearly with V_{dd}, the execution time of a task increases with decreasing V_{dd}. Equation (2) therefore implies that a penalty in speed is inevitable when supply voltage is reduced and therefore, the dynamic power consumption of the circuit exhibits a cubic dependence on V_{dd} (since $f = 1/T_d$). However, the energy consumption of the CMOS circuit, which is the product of the power and the delay, i.e., $E = P_{\text{dynamic}}T_d$, exhibits a quadratic dependence on V_{dd}. Next, we define the problem $\mathcal{P}_{\text{cpu}}$ that is addressed in this paper.

B. Problem Description

$\mathcal{P}_{\text{cpu}}$: We are provided with a set $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$ of n jobs and a processor that is capable of operating at one of k speeds $S_1 > S_2 > \cdots > S_k$. These speeds correspond to k unique CPU operating voltages $V_1 > V_2 > \cdots > V_k$. Each job $j_i \in \mathcal{J}$ is characterized by: 1) a release time a_i; 2) a deadline d_i; and 3) worst-case execution times $c_{i1} < c_{i2} < \cdots < c_{ik}$ at the k different operating speeds.

The goal is to determine a voltage $v_i \in \{V_1, V_2, \ldots, V_k\}$ and a start time t_i for each job j_i such that

i) the total energy $\sum_{i=1}^{n} v_i^2 c_i, c_i \in \{c_{i1}, c_{i2}, \ldots, c_{ik}\}$, is minimized;

ii) $t_i \geq a_i$ and $t_i + c_i \leq d_i$.

Since we consider only jobs (and not tasks), our method can be used to schedule aperiodic as well as periodic task sets, since both can be represented using the job model described above through the application of the LCM method [16]. A periodic task consists of an infinite sequence of identical activities. Such tasks are characterized by a release time, a deadline, and a period, which defines the rate of activation of the task, and a worst-case

execution time. Aperiodic tasks also consist of an infinite sequence of identical activities, but the activation of these tasks is not regular. Each instance of a task is called a job. Jobs are characterized only by an arrival time, a deadline and the worst-case execution time. Our method can be used to generate optimal energy-solutions, and a postprocessing transformation ensures that the same energy-optimal schedule does not cause any missed deadlines with the use of a priority-based (fixed- or dynamic-) scheduling policy. We next briefly review important concepts from network flow theory.

C. Network Flow Models

The objective in a network flow model is to move some entity (electricity, water, time, etc.) from one point to another through an underlying network as efficiently as possible. A network flow model is represented as a directed network $G = (N, A)$ defined by a set N of n nodes and a set A of m directed arcs. Each arc $(i, j) \in A$ has an associated cost C_{ij} that denotes the cost per unit flow on the arc. The flow cost varies linearly with the amount of flow. Also associated with (i, j) is an upper bound u_{ij} on the arc capacity that denotes the maximum flow on (i, j) and a lower bound l_{ij} that denotes the minimum flow on the arc. In *GNF models*, a positive multiplier μ_{ij} exists for (i, j) of the network and if 1 unit of flow is sent from node i to node j along arc (i, j), then μ_{ij} units of flow arrive at node j. Each node $i \in N$ is also characterized by an integer $b(i)$ representing its supply/demand.

The decision variables in the GNF problem are the arc flows on (i, j) and are represented by x_{ij}. The GNF problem is an optimization problem that can be formulated as

$$\text{Minimize} \sum_{(i,j)\in A} C_{ij}x_{ij} \text{ subject to}$$

1) $\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} \mu_{ji}x_{ji} = b(i) \quad \forall i \in N;$
2) $l_{ij} \le x_{ij} \le u_{ij}.$

Constraint (1) is called the *mass balance constraint*. It states that the difference between the flow into a node j and the flow out of node j must equal its supply/demand of flow. However, in a circulation version of a network flow problem, such as the one used here, $b(i) = 0 \; \forall \; i \in N$, (in a circulation version, an additional arc exists between sink and source; all flow therefore circulates around the network) and so no flow can accumulate at any node $i \in N$. Constraint (2) is called the *flow constraint*. It states that the flow along any arc (i, j) must lie between its lower and upper capacity bounds. In any feasible solution, every arc flow x_{ij} satisfies both these constraints.

In GNF graphs, the arc costs vary linearly with the amount of flow through the arc and the arc flows in the objective function are separable, i.e., the different flow variables x_{ij} appear in separate $C_{ij}x_{ij}$ terms. Nonlinear optimization problems with separable convex objective functions are typically solved using network flow techniques through convex-cost network flow models. In such models, the separable terms are allowed to be *nonlinear functions* of the form $C_{ij}(x_{ij})$. Each nonlinear function is restricted to be purely convex, i.e., any linear interpolation between two points on the function always lies on or above it, as illustrated in Fig. 1. With this restriction

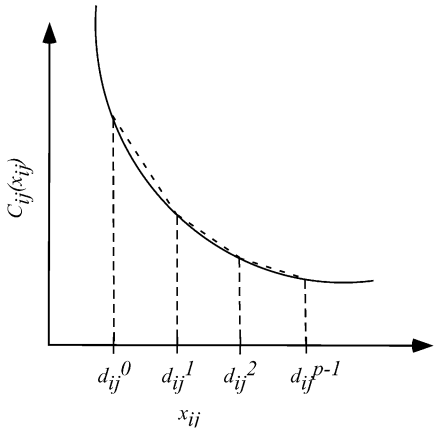

Fig. 1. Example of a convex cost function.

on the form of a nonlinear function, efficient techniques have been developed to solve nonlinear network flow models. $\mathcal{P}_{\text{cpu}}$ lends itself naturally to nonlinear (convex cost) network flow techniques through the convex dependence of energy on V_{dd}. Recall from Section III-A that the energy consumed by a CMOS processor is a quadratic (convex) function of the supply voltage, i.e., $E \propto v_i^2 c_i$.

The convex cost model is formally defined as

$$\text{Minimize} \sum_{(i,j)\in A} C_{ij}(x_{ij})$$

subject to

1) $\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \quad \forall i \in N;$
2) $l_{ij} \le x_{ij} \le u_{ij}.$

Note that the separable terms with linear cost coefficients in the objective function have now been replaced by separable convex cost functions. A convex cost model can be transformed into a minimum cost model which is defined as follows:

$$\text{Minimize} \sum_{(i,j)\in A} C_{ij}x_{ij}$$

subject to

1) $\sum_{j:(i,j)\in A} x_{ij} - \sum_{j:(j,i)\in A} x_{ji} = b(i) \quad \forall i \in N;$
2) $l_{ij} \le x_{ij} \le u_{ij}.$

The nonlinear functions $C_{ij}(x_{ij})$ in a convex cost model are represented in a piecewise linear representation fashion, where each arc cost has at most p linear segments. Let $0 = d_{ij}^0 < d_{ij}^1 < \cdots d_{ij}^{p-1}$ represent the different breakpoints of the cost function for arc (i, j). The cost varies linearly in the interval $[d_{ij}^{k-1}, d_{ij}^k]$. Therefore, a cost function is defined by the set of breakpoints and the slopes between successive breakpoints. This type of definition of arc costs sees an increase in the number of nodes in the model as the arc cost becomes more complex.

The piecewise linear representation of arc costs is illustrated in Fig. 2, which depicts an arc (i, j) in a convex cost network. The cost function here is represented as a convex function of arc flow, e.g., x_{ij}^2. The original arc shown in Fig. 2(a) is expanded into p parallel arcs from node i to node j (recall that each cost function has p linear segments). Each parallel arc $(i, j)^k, 1 \le k \le p$ has a cost equal to the slope between the breakpoints d_{ij}^k and d_{ij}^{k-1} and an upper bound equal to $d_{ij}^k - d_{ij}^{k-1}$.

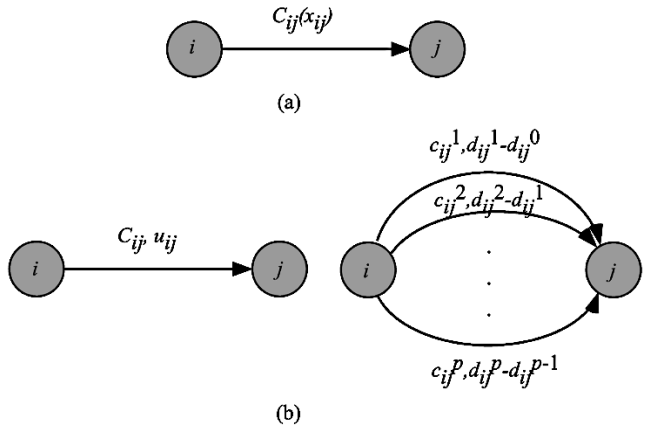

Fig. 2. Transformation of arcs in a convex cost model to arcs in a minimum cost model.

The arc capacities, costs and supplies/demands in the model assumed to be integers. Although this assumption results in some loss of generality, it is not truly a restriction because rational numbers can always be transformed to integers by multiplying them with suitably large numbers to obtain the required degree of accuracy. For example, if a solution more accurate than the integer optimal solution is required, each flow variable x_{ij} is replaced by My_{ij}, where M is sufficiently large, depending on the required accuracy. Then, if y_{ij}^* denotes an optimal solution to the transformed problem, $x_{ij}^* = y_{ij}^*/M$ is the optimal solution to a degree of accuracy $1/M$.

GNF models are solved using the generalized network simplex algorithm. The generalized network simplex algorithm maintains a feasible basis structure at every iteration and by performing pivot operations, it transforms a solution into a better one until the solution satisfies specific optimality criteria. The worst-case complexity of the generalized network simplex algorithm cannot be bounded by a polynomial function of the number of nodes n and number of arcs m. However, empirical studies have revealed that the running time of the algorithm is generally a low-order polynomial of n and m. The algorithm has been observed to be only two or three times slower than the network simplex algorithm used to solve the minimum cost flow problem [1].

Network flow models have been used in the past for task scheduling without energy considerations [18], [20], [27]. In [18], a maximum flow network model to identify a schedule for a set of jobs running at a single constant voltage is described. The problem of finding a preemptive schedule for a set of jobs with arrival times and deadlines running on parallel uniform machines has been studied in [20]. In [27], the author describes a network flow model for scheduling a job on multiple processors in order to minimize execution cost and interprocessor communication. In this paper, we consider the problem of scheduling a set of jobs on a single processor that is capable of operating at k speeds such that the total energy consumed by the set of jobs is minimized. A feasible solution for the models is a set of flows x_{ij} that satisfies all flow constraints and capacity constraints. The decision variables x_{ij} represent the execution time for each job. The identification of the execution times trivially leads to an assignment of speeds to the jobs. In the next section, we describe our GNF model for $\mathcal{P}_{\text{cpu}}$.

IV. GNF Model for $\mathcal{P}_{\text{cpu}}$

For the sake of simplicity, we first develop a GNF model for $\mathcal{P}_{\text{cpu}}$ with two speeds S_h and $S_l (S_h > S_l)$, with supply voltages V_h and $V_l (V_h > V_l)$, respectively. In Section IV-B, an extension to $k > 2$ speeds is described. Depending on the execution speed, a job j_i has an execution time of either c_{ih} or c_{il}.

We first sort the arrival times and deadlines of the n jobs in ascending order. This results in $2n$ elements in the set $P = \{p_0, p_1, p_2, \ldots, p_{2n-1}\}$, which divide the hyperperiod H into a sequence of at most $2n-1$ distinct subintervals. Let Δ_i represent the time interval $[p_i, p_{i-1}]$, i.e., $\Delta_i = p_i - p_{i-1}, 1 \leq i \leq 2n-1$.

The nodes and arcs in the GNF model for $\mathcal{P}_{\text{cpu}}$ are described next. In order to avoid tedium, each arc (i, j) is described by a four-tuple $[l_{ij}, u_{ij}, C_{ij}, \mu_{ij}]$ representing the lower bound, upper bound, cost and multiplier respectively.

Each interval node Δ_k represents a distinct time interval in $P(\Delta_k = [p_k, p_{k-1}])$. The GNF model for $\mathcal{P}_{\text{cpu}}$ with two speeds consists of a special source node s, a special sink node t, a set of nodes that represent the jobs, a set of nodes that represent execution speeds for the jobs, and a set of nodes that represent time intervals in which jobs can be scheduled. Each job node j_i has two associated speed nodes s_{ih} and s_{il} (see Fig. 3).

The arcs in the GNF model are best explained by classifying them based on the interpretation of their flows. They are described below:

1) **Speed identification**. These arcs are constructed from source s to each job vertex j_i. They are described by the four-tuple $[c_{ih}, c_{ih}+1, 0, 1]$. The lower bound represents the execution time of j_i at the high speed, and the upper bound is simply $c_{ih}+1$ (the reason for this is explained shortly). The lower bounds test the schedulability of the job set: recall that a feasible schedule is one where the arc flows satisfy their flow constraints and capacity constraints. Therefore, in any feasible solution for the network flow graph, at least c_{ih} units of flow is sent along each of these arcs. In other words, if at least c_{ih} units of flow do not appear along every arc (s, j_i), then job j_i cannot be scheduled between its arrival time and deadline and therefore, the set of jobs is not schedulable. Furthermore, the integer restriction on arc flows forces the flow value to one of two values, thereby ensuring that only one of the two allowable speeds is chosen for job execution.
2) **Speed selection**. These arcs are constructed from job node j_i to its associated speed nodes s_{ih} and s_{il}. The sum of the flows along this pair of arcs for each job node represents its assigned execution time. We describe each type of speed selection arc individually:

   Job execution at S_h. The flows along the arcs from j_i to s_{ih} represent the execution of job j_i at speed S_h. In any feasible schedule, the flows on these arcs is equal to c_{ih}. The associated costs are indicative of the power consumed by job j_i at the high speed, i.e., $C_{j_i,s_{ih}} = P_h$, where $P_h = V_h^2$. Since c_{ih} units of execution time flows along these arcs, a total energy of $P_h c_{ih}$ units is consumed.

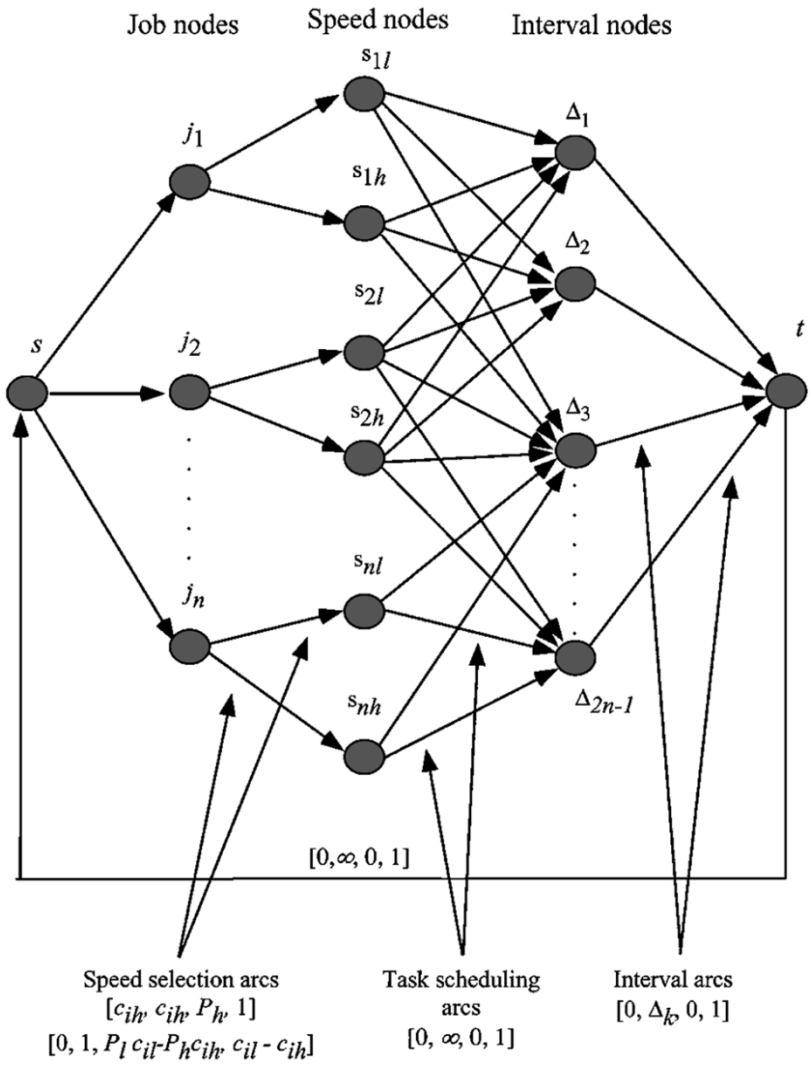

Fig. 3. GNF model for $\mathcal{P}_{\text{cpu}}$.

Job execution at S_l. A flow of 1 unit along the arcs from j_i to s_{il} indicates that j_i is executed at the low speed S_l. This requires further explanation. Recall that the flow into node j_i can only be c_{ih} or $c_{ih}+1$. Assume that c_{ih} units of flow enter node j_i. Constraint (2) from Section III-C states that no flow can accumulate at j_i and, therefore, this flow must leave node j_i. Since the lower bound on (j_i, s_{ih}) equals c_{ih}, all flow exits along this arc and constraint (2) is satisfied. Now, assume that $c_{ih}+1$ units of flow enter node j_i. Again, constraint (2) states that no flow can accumulate at j_i, and therefore, all flow entering j_i must leave through the (j_i, s_{ih}) and (j_i, s_{il}) arcs. c_{ih} units of flow exit through arc (j_i, s_{ih}) (the upper bound prohibits any additional flow on this arc) and 1 unit exits along arc (j_i, s_{il}), thereby satisfying the mass balance constraint. Now, a flow of c_{ih} appears at node s_{ih} and a flow of $c_{il} - c_{ih}$ appears at s_{il}, resulting in a total flow of c_{il} units that must now flow along the interval assignment arcs (described next). Therefore, a flow of $c_{ih}+1$ units entering node j_i represents j_i's execution at the low speed and a flow of c_{ih} units into node j_i represents its execution at the high speed.

3) **Interval assignment**. These arcs are constructed from speed nodes s_{ih} and s_{il} to each interval node Δ_k if $a_i \leq p_k$ and $d_i \geq p_{k+1}$. They are described by the four-tuple $[0, \infty, 0, 1]$, These arcs ensure that jobs can be scheduled only within certain intervals, i.e., between their arrival times and deadlines.
4) **Time intervals**. These arcs are constructed from each interval node Δ_k to sink node t with parameters $[0, \Delta_k, 0, 1]$. The capacities of these arcs represent the amount of time available each interval.

The solution of the GNF model results in a set of flows x_{ij} along the arcs. The flow along each arc (s, j_i) represents the execution time of job j_i. From this, the assigned execution speed for each job is determined. The job schedule is determined by inspecting the flows along the (s_{ih}, Δ_k) and (s_{il}, Δ_k) arcs. These flows represent the amount of CPU time that is allotted to job j_i in interval Δ_k.

A. Proof of Energy-Optimality

In order to show that the GNF model for $\mathcal{P}_{\text{cpu}}$ results in energy-optimal solutions, it is sufficient to show that the costs as-

sociated with the arcs accurately represent the energy consumptions of the jobs at the different speeds, and to show that the objective function and constraints in the GNF model are equivalent to an optimization problem formulation of $\mathcal{P}_{\text{cpu}}$.

We first show that the cost functions of the arcs accurately model the energy consumption of the jobs at different speeds. This is formalized through the following theorem.

Theorem 1: The costs associated with the speed selection arcs in the GNF model for $\mathcal{P}_{\text{cpu}}$ are representative of the energy consumption of the jobs at the different operating speeds.

Proof: Recall from Section III-C that the flow into a job node j_i can be either c_{ih} or $c_{ih}+1$. Depending on the value of the flow entering j_i, job j_i consumes different amounts of energy. A flow of c_{ih} into j_i corresponds to its execution at the high speed, while a flow of $c_{ih}+1$ represents its execution at the low speed. Since the mass balance constraint states that no flow can accumulate at j_i, any flow entering j_i must leave through the speed selection arcs. Consider the following two cases.

Case 1) c_{ih} units of flow entering j_i. In this case, j_i has been assigned the higher speed for execution. Since the lower bound for arc (j_i, s_{ih}) is equal to c_{ih}, all flow entering j_i exits through this arc. The cost function associated with arc (j_i, s_{ih}) is $P_h(=V_h^2)$. Recall that the cost represents the cost of transporting a single unit of flow along an arc. Therefore, the cost of transporting c_{ih} units of flow is $P_h c_{ih}$, which is the energy cost of j_i's execution at the high speed.

Case 2) $c_{ih}+1$ units entering j_i. This represents the assignment of the lower operating speed for job j_i. Recall that when $c_{ih}+1$ units of flow enter j_i, c_{ih} units exit through (j_i, s_{ih}) and 1 unit leaves through (j_i, s_{il}). The cost of the (j_i, s_{il}) arc is selected to be $P_l c_{il} - P_h c_{ih}$. Summing the total cost of all the flow leaving node j_i, we have

$$P_h c_{ih} + (P_l c_{il} - P_h c_{ih}) \cdot 1 = P_l c_{il} \tag{3}$$

which represents the energy cost of executing j_i at the lower speed S_l. This completes the proof of the theorem. □.

It is now easy to see that the separable terms in the objective function of the GNF model are representative of the energy costs of the jobs. Since the costs of all the arcs other than the speed selection arcs is zero, no other term in the objective function contributes to the overall flow cost in the objective function.

We next show that the objective function of the GNF model maps to an optimization problem statement for $\mathcal{P}_{\text{cpu}}$. This is formalized through the following theorem.

Theorem 2: The objective function and constraints that are modeled in the GNF model for $\mathcal{P}_{\text{cpu}}$ are equivalent to the optimization problem statement for $\mathcal{P}_{\text{cpu}}$.

Proof: Since we wish to minimize the total energy consumed by the set of jobs, and because $E_i \propto v_i^2 c_i$, the objective function can be written as

$$\text{Minimize} \sum_{i=1}^{n} v_i^2 c_i. \tag{4}$$

The execution time c_i and execution voltage v_i of job j_i can be represented in a "sum-of-products" form

$$c_i = a_{i1} c_{ih} + a_{i2} c_{il}$$
$$v_i^2 = a_{i1} V_h^2 + a_{i2} V_l^2$$

where a_{i1} and a_{i2} are 0–1 binary variables, i.e., $a_{i1} + a_{i2} = 1$.

Substituting for c_i and v_i in (4) and noting that after multiplication, the terms containing the product of a_{i1} and a_{i2} are equal to zero, we obtain the following simplified objective function:

$$\text{Minimize} \sum_{i=1}^{n} \left(a_{i1} c_{ih} V_h^2 + a_{i2} c_{il} V_l^2 \right). \tag{5}$$

We next show that the objective function of the GNF model simplifies to (5).

The objective function in the GNF model is

$$\text{Minimize} \sum_{i:(i,j)\in A} C_{ij} x_{ij}. \tag{6}$$

Since the only arcs with nonzero costs are (j_i, s_{ih}) and (j_i, s_{il}), (6) can be rewritten as

$$\text{Minimize} \sum_{i=1}^{n} \left(C_{j_i, s_{ih}} x_{j_i, s_{ih}} + C_{j_i, s_{il}} x_{j_i, s_{il}} \right). \tag{7}$$

Any feasible solution of the model always has a flow which is exactly equal to c_{ih} along arc (j_i, s_{ih}). Moreover, $C_{j_i, s_{ih}} = P_h = V_h^2$ and $C_{j_i, s_{il}} = V_l^2 c_{il} - V_h^2 c_{ih}$. Equation (7) can therefore be written as

$$\text{Minimize} \sum_{i=1}^{n} \left[V_h^2 c_{ih} + \left(V_l^2 c_{il} - V_h^2 c_{ih} \right) x_{j_i, s_{il}} \right]. \tag{8}$$

Simplification of this equation results in the following objective function:

$$\text{Minimize} \sum_{i=1}^{n} V_h^2 c_{ih} \left(1 - x_{j_i, s_{il}} \right) + V_l^2 c_{il} x_{j_i, s_{il}}. \tag{9}$$

It is now easy to observe the similarity between (9) and (5). Specifically, the variables $x_{j_i, s_{il}}$ are equivalent to the a_{i2} variables in the optimization problem formulation. Since $a_{i1} = 1 - a_{i2}$, the correspondence between the two objective functions is proved.

The constraints in the GNF model can be seen to map exactly to the constraints in the optimization problem statement by a simple inspection. These constraints are listed below.

1) An entire job is executed at a single speed.
2) For every execution speed, there is a corresponding execution time for a job.
3) Each job must start no earlier than its arrival time and must complete execution at its assigned speed no later than its deadline.

This completes the proof of the theorem. □.

We next describe a GNF model for $\mathcal{P}_{\text{cpu}}$ with more than two speeds.

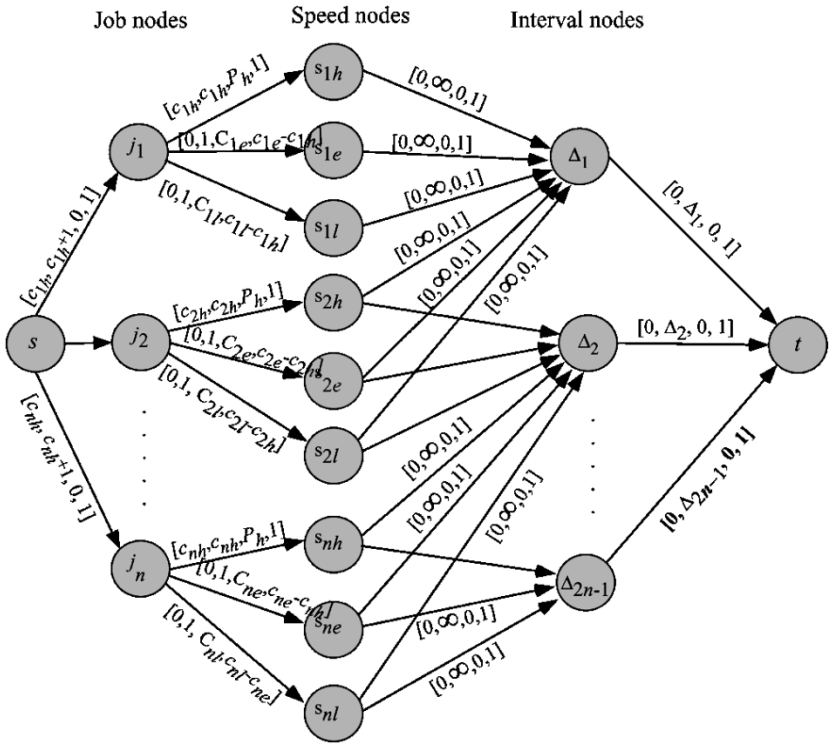

Fig. 4. GNF Model for three-speed $\mathcal{P}_{\rm cpu}$.

B. GNF Model for $\mathcal{P}_{\rm cpu}$ With More Than Two Speeds

In this section, we briefly describe a GNF model to handle the case where the CPU can operate at k discrete speeds, where $k > 2$. For the sake of illustration, we assume here that the CPU can operate at 3 speeds—$V_h > V_e > V_l$ represent three operating voltages. The extension to $k > 3$ is similar. The GNF model for this system consists of the following vertices and arcs (see Fig. 4):

1) the source vertex s and the sink vertex t;
2) a *job vertex* j_i for each job in the job set;
3) an *interval vertex* Δ_k for each interval $[p_k, p_{k+1}]$;
4) three *speed vertices* s_{ih}, s_{ie}, and s_{il} for each job vertex j_i;
5) an arc from source s to each of the job vertices j_i with parameters $[c_{ih}, c_{ih}+1, 0, 1]$;
6) an arc from each job node j_i to its corresponding speed node s_{ih} with parameters $[c_{ih}, c_{ih}, P_h, 1]$;
7) an arc from each job node j_i to its corresponding speed node s_{ie} with parameters $[0, 1, P_e c_{ie} - P_h c_{ih}, c_{ie} - c_{ih}]$;
8) an arc from each job node j_i to speed node s_{il} with parameters $[0, 1, P_l c_{il} - P_h c_{ih}, c_{il} - c_{ih}]$;
9) an arc from each speed node to each interval node Δ_k if $a_i \leq p_k$ and $d_i \geq p_{k+1}$ with parameters $[0, \infty, 0, 1]$;
10) an arc from each interval node Δ_k to sink node t with parameters $[0, \Delta_k, 0, 1]$.

C. Graph Complexity

In this section, we analyze the complexity of the network flow graph. Let n be the number of jobs in the job set, and k be the number of available processor speeds. The number of nodes in the network flow graph can be bounded from above through the following analysis: the network flow graph has n job nodes, and each job node has k corresponding speed nodes. This results in $n + nk$ vertices. The hyperperiod H is divided into $2n - 1$ distinct subintervals, thereby contributing at most $2n-1$ interval nodes Δ_k. Therefore, an upper bound on the number of nodes in the network flow graph is $(k + 3)n - 1$.

In a similar manner, an upper bound on the number of arcs can be derived through the following analysis. n arcs exist from the source node s to the job nodes j_i, and nk arcs from the job nodes to their corresponding speed nodes. In a worst-case scenario, arcs could exist from each of the nk speed nodes to each of the $2n-1$ interval nodes, resulting in $2n^2k-nk$ speed-interval arcs. Finally, $2n - 1$ arcs are constructed from the interval nodes to the sink t. An upper bound on the number of arcs in the network flow graph is therefore $2n^2k + 3n - 1$.

Thus far, we have focused on solving $\mathcal{P}_{\rm cpu}$ *optimally* using GNF graphs. A limitation of the GNF model for $\mathcal{P}_{\rm cpu}$ lies in its inability to scale to a large number if processor speeds. By utilizing the convex dependence of energy on CPU operating voltage, more efficient network flow techniques can be applied to $\mathcal{P}_{\rm cpu}$ for near-optimal solutions. These algorithms are known to be some of the fastest and most efficient algorithms in the field of operations research, and belong to the category of scaling-based algorithms. In the next section, we develop a minimum-cost network flow formulation for $\mathcal{P}_{\rm cpu}$. This model is scalable and generates efficient solutions in reasonable periods of time. A limitation of the model is the growth in the number of nodes in the network flow graph; however, our experimental results show that the model scales well even to real-life task sets. Unlike previously developed DVS algorithms, this model is not limited by

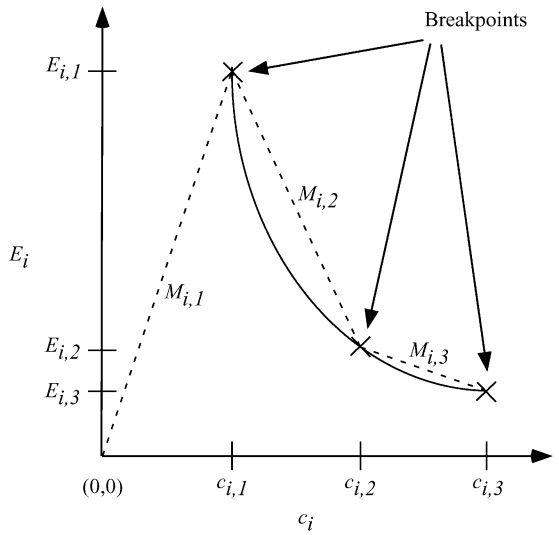

Fig. 5. Calculation of arc slopes for a three-speed processor.

the number of available processor frequencies (it performs efficiently with processors with a large range of frequencies as well as with processors with a small set of available frequencies).

V. Minimum-Cost Model for $\mathcal{P}_{\text{cpu}}$

A. Construction of the Model

In this section, we detail the construction of the minimum-cost network flow model for $\mathcal{P}_{\text{cpu}}$. Recall that each job can execute at one of k different speeds (voltages). Therefore, for every job j_i, the energy consumption at the k different voltages is plotted against the corresponding execution time, resulting in k discrete breakpoints on the "E_i-c_i" curve (see Fig. 5). The slopes of the line segments between successive breakpoints are calculated and used as linear arc costs in the minimum-cost network. Let $E_{i,j}$ and $c_{i,j}$ represent the energy consumption and execution time of job j_i executing at voltage V_j, respectively, and let $M_{i,j}$ represent the slope of the line segment between breakpoints $(c_{i,j}, E_{i,j})$ and $(c_{i,j-1}, E_{i,j-1})$. Note that for the highest voltage V_1, there is no preceding breakpoint. Therefore, $M_{i,1}$ is the slope of the line segment between the origin $(0,0)$ and $(c_{i,1}, E_{i,1})$.

Similar to the preprocessing performed for the GNF model in Section IV, the arrival times and deadlines of the jobs divide the hyperperiod H into a sequence of at most $2n-1$ distinct intervals. Assume that the set of arrival times and deadlines are sorted in ascending order and are given by $p_0, p_1, p_2, \ldots, p_{2n-1}$. Let Δ_i represent the time interval $[p_i, p_{i-1}]$, i.e., $\Delta_i = p_i - p_{i-1}$.

Each arc (i,j) is represented by a three-tuple $[l_{ij}, u_{ij}, C_{ij}]$ representing the lower bound, upper bound, and cost respectively. The minimum-cost model for $\mathcal{P}_{\text{cpu}}$ is shown in Fig. 6. It consists of the *source node* s and the *sink node* t, k *job-speed nodes* for each job j_i denoted by $j_{i,l}$, each of which represents job j_i executing at voltage V_l, and *interval nodes* Δ_k for each interval $[p_k, p_{k-1}]$.

We again categorize the arcs based on the interpretations of their flows. They are described below.

1) **Speed selection**. These arcs are constructed from source s to the job-speed nodes $j_{i,l}$. The arc flows are indicative of the speed at which j_i is executed. They are explained in greater detail next.

a) Job execution at the high speed S_1. A nonzero flow on $(s, j_{i,1})$ and zero flows on $(s, j_{i,l}), l > 1$ indicates that j_i is assigned the highest voltage V_1. These arcs are described by the three-tuple $[c_{i,1}, c_{i,1}, M_{i,1}]$. The lower and upper bounds on these arcs are equal to the execution time of job j_i at the highest speed. Therefore, no feasible schedule exists for the job set if the capacity constraints on these arcs are not satisfied. It is also easy to see that an energy cost of $c_{i,1}M_{i,1} (= E_{i,1})$ is incurred when a job is assigned voltage V_1, thereby accurately modeling the energy consumption of j_i at the V_1.

b) Job execution at reduced speed. These arcs are constructed from s to $j_{i,l}, l > 1$ with parameters $[0, c_{i,l} - c_{i,l-1}, M_{i,l}]$. A flow along $(s, j_{i,l})$ represents job j_i executing at voltage V_l. The arc cost for (s, j_{il}) is equal to the slope between the breakpoints $c_{i,l}$ and $c_{i,l-1}$. These arcs have the following interesting property: if a job is scheduled to run at a voltage V_p, the arc flows along all $(s, j_{i,l}), l \leq p$ arcs are equal to their upper bounds $c_{i,l} - c_{i,l-1}$. Furthermore, the costs associated with these arcs are negative and decrease with increasing values of l.

2) **Task scheduling**. Task scheduling arcs are constructed from $j_{i,l}$ to each interval node Δ_k if $a_i \leq g_k$ and $d_i \geq g_{k+1}$. They are described by $[0, \infty, 0]$. They model the constraint that jobs can be scheduled only within certain intervals, i.e., between their arrival times and deadlines.

3) **Interval arcs**. These arcs are constructed from each interval node Δ_k to sink node t with parameters $[0, \Delta_k, 0]$. The capacities of these arcs represent the amount of time available for scheduling jobs in each interval.

The solution of the minimum-cost model results in a set of flows x_{ij}. The flows along $(s, j_{i,l})$ represent the execution time of job j_i. Therefore, by inspecting the $x_{s,j_{i,l}}$ variables, the speed (voltage) at which each job is executed is identified. The job schedule is determined by the inspection of flows along $(j_{i,l}, \Delta_k)$. These flows represent the amount of time allotted to job j_i in interval Δ_k.

We next prove the equivalence of the minimum-cost network flow model to $\mathcal{P}_{\text{cpu}}$ by demonstrating the equivalence of the objective functions and constraints in two different formulations of $\mathcal{P}_{\text{cpu}}$: 1) the network flow formulation and 2) the optimization problem formulation.

B. Validation of the Minimum-Cost Model

The objective function in the optimization problem formulation of $\mathcal{P}_{\text{cpu}}$ from is given by

$$\text{Minimize} \sum_i c_i v_i^2. \tag{10}$$

Each job j_i can execute at any of k discrete speeds (voltages). Therefore, c_i and v_i can be represented using k binary variables $a_{i1}, a_{i2}, \ldots, a_{ik}$ as

$$c_i = a_{i1}c_{i1} + a_{i2}c_{i2} + \cdots + a_{ik}c_{ik}$$

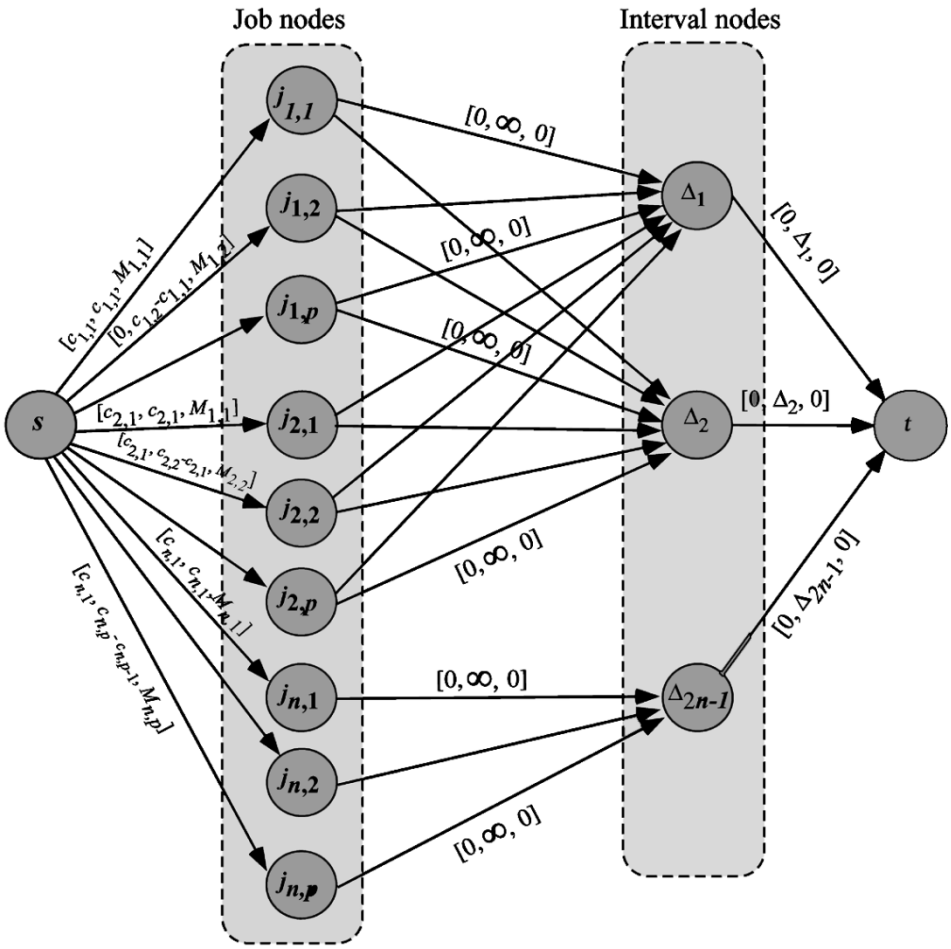

Fig. 6. Minimum-cost model for $\mathcal{P}_{\mathrm{cpu}}$.

and

$$v_i = a_{i1}V_1 + a_{i2}V_2 + \cdots + a_{ik}V_k$$

where

$$a_{i1} + a_{i2} + \cdots + a_{ik} = 1.$$

Substituting for c_i and v_i in (10) and noting that the product $a_{il}a_{im} = 0$ when $l \neq m$, we obtain the following simplified objective function:

$$\text{Minimize} \sum_i \left(a_{i1}c_{i1}V_1^2 + a_{i2}c_{i2}V_2^2 + \cdots + a_{ik}c_{ik}V_k^2\right). \tag{11}$$

On the other hand, the objective function for the network flow model is

$$\text{Minimize} \sum_i C_{s,(i,l)}x_{s,(i,l)}$$

where $x_{s,(i,l)}$ is the arc flow along $(s, j_{i,l})$ and $C_{s,(i,l)}$ is the cost associated with arc $(s, j_{i,l})$. The costs $C_{s,(i,l)}$ are the slopes $M_{i,l}$ of the different line segments between successive breakpoints. So, the objective function can be rewritten as

$$\text{Minimize} \sum \left(c_{i,1}V_1^2 + x_{s,(i,2)}M_{i,2} + \cdots + x_{s,(i,k)}M_{i,k}\right). \tag{12}$$

However, $M_{i,l} = (V_l^2 c_{i,l} - V_{l-1}^2 c_{i,l-1})/(c_{i,l} - c_{i,l-1})$. Further, let each decision variable $x_{s,(i,l)}$ be equal to $b_{i,l}x_{s,(i,l)}$, where $b_{i,l}$ is a binary variable. Due to the convexity of the cost function, if a nonzero flow exists along arc $(s, j_{i,l})$, then, for all arcs $(s, j_{i,p}), p \leq l, x_{s,(i,p)} = u_{s,(i,p)} = c_{i,p} - c_{i,p-1}$. This means that all binary variables $b_{i,p} = 1$ for all $p \leq l$. Using this characteristic of the costs, and substituting for $M_{i,l}$, (11) simplifies to

$$\text{Minimize} \sum_i \left(b_{i,1}c_{i,1}V_1^2 + b_{i,2}V_2^2 c_{i,2} - b_{i,2}c_{i,1}V_1^2 c_{i,1} + \cdots + b_{i,k}V_k^2 c_{i,k} - b_{i,k}V_{k-1}^2 c_{i,k-1}\right). \tag{13}$$

It is now easy to see the correspondence between the two objective functions. If a job j_i is chosen to execute at a voltage $V_l, b_{i,p} = 1$ for all $p \leq l$ and $b_{i,p} = 0$ for all $p > l$. The individual terms in (13) cancel out, leaving only $b_{i,l}V_l^2 c_{i,l}$ in the objective function. This exactly corresponds to the $a_{i,l}V_l^2 c_{i,l}$ term in (11), the objective function in the optimization problem formulation.

A drawback of this model is its inability to characterize the presence of slack in the low-energy schedule. For example, assume that job j_i can execute at a voltage V_l but not at voltage V_{l+1} because of insufficient time between its arrival and deadline. A nonzero flow along $(s, j_{i,l+1})$ implies the assignment of an inadmissible speed to job j_i. However, this can be avoided by raising the voltage of job j_i from V_{l+1} to V_l. Any inadmissible flow is simply the unavoidable slack in the system and can

Algorithm MCPPA(U, X, n, k)
U: set of upper bounds u_{ij} for each arc;
X: set of arc flows x_{ij};
n: number of jobs;
k: number of processor speeds;
1. **for** $i = 1$ **to** n
2. **for** $l = 1$ **to** k
3. **if** $x_{s,(i,l)} \neq 0$ **and** $x_{s,(i,l)} \neq u_{s,(i,l)}$
4. $v_i = V_{l-1}$
5. **end**

Fig. 7. Postprocessing algorithm for the correct assignment of speeds in the minimum-cost model.

be safely ignored. Additional postprocessing of the solution of the minimum-cost network flow model enables the identification of jobs that execute at inadmissible speeds, and allows us to raise the speeds to the next higher value and make the necessary adjustments to the total energy cost to reflect this change. A straightforward postprocessing algorithm is described in Fig. 7.

We have proved the correctness of the objective function of the minimum-cost model. We now explain the correctness of the constraints: 1) $t_i \geq a_i$and 2) $t_i + c_i \leq d_i$.

Recall that arcs are drawn from node $j_{i,l}$ to each interval node Δ_k only if $a_i \leq g_k$ and $d_i \geq g_{k+1}$. It is easy to see that the construction of these arcs accurately models the scheduling of job j_i only between its release and deadline.

We next analyze the complexity of the minimum-cost model for $\mathcal{P}_{\rm cpu}$ by deriving upper bounds on the number of nodes and arcs.

C. Complexity of the Minimum-Cost Model for $\mathcal{P}_{\rm cpu}$

Let n represent the number of jobs and k represent the number of processor speeds. An upper bound on the number of nodes in the network flow model can be derived as follows. Each of the n jobs has associated with it k job-speed nodes, resulting in nk job-speed nodes in the model. The $2n - 1$ intervals in the hyperperiod H contribute at most $2n - 1$ interval nodes. Therefore, the number of nodes in the minimum-cost model is bounded from above by $nk + 2n - 1$.

An upper bound on the number of arcs can be derived in a similar fashion. A total of nk arcs exist from source s to the job-speed nodes $j_{i,l}$. In a worst-case scenario, all jobs can be scheduled in all $2n-1$ intervals, resulting in $nk(2n-1)$ speed-interval arcs. Finally, $2n-1$ interval arcs exist from the interval nodes Δ_k to the sink node t. The total number of arcs in the model is therefore bounded from above by $2n^2k + 2n - 1$. In the next section, we present our experimental results.

VI. Experimental Results

We next evaluate the network flow models with several different job sets with varying utilization. We first compare the GNF method with a baseline case where all jobs are executed at the highest processor speed. We then compare our results to the offline fixed-priority DVS algorithm from [22], the offline dynamic priority scheme from [30], and the offline dynamic priority scheme from [14], which is designed specifically for processors with a discrete set of voltages. We refer to the algorithm described in [22] as VSLP, the one in [30] as LPEDF and the one in [14] as D-LPEDF. VSLP assumes a processor with a large range of operating frequencies from 8 MHz to 100 MHz that is adjustable in increments of 1 MHz, while LPEDF uses an infinitely continuous range of frequencies. To the best of our knowledge, the results presented in these papers are the best results published thus far in the literature. To demonstrate that the GNF method performs better than VSLP and LPEDF for processors with only a few speed settings, we also compare our method with modified versions of VSLP and LPEDF, which we call 5-VSLP and 5-LPEDF, that use only five speeds. In the first set of experiments, we use two different processor models. The first model is that of a Crusoe processor and our second model corresponds to an AMD K-6 processor. The voltage and frequency settings for these processors are listed in Table I.

TABLE I
Available Speed and Voltage Settings for the Five-Speed Processor Models Used in Our Experiments

Processor	Voltage (V)	Frequency (MHz)
Transmeta Crusoe	1.3	800
	1.2	667
	1.1	533
	1.0	400
	0.9	300
AMD K-6 IIIE	1.8	500
	1.7	450
	1.6	400
	1.5	350
	1.4	300

The job sets are constructed from pure periodic task sets, and the same task sets are used for all approaches. Each task set consists of five tasks, the periods and execution times of which are generated randomly. The arrival times of all tasks are set to 0. All times are assumed to be in units of milliseconds. The execution times in Table II represent the worst-case execution times of the tasks while executing at the maximum processor speed. The deadline of a task is assumed to be equal to its period. The properties of the evaluation task sets are shown in Table II. The GNF models are solved using a Microsoft Excel-based GNF solver [12] running on a Pentium III PC at 500 MHz with 256 MB of RAM.

The comparison of the GNF approach with the baseline case is shown in Table III.

For task sets of any utilization value, the GNF method results in significant energy savings.

The comparison of the GNF method with VSLP and 5-VSLP is presented in Table IV. Comparative results for a real-life benchmark, namely CNC [13], are also presented. The 5-GNF method performed at par with 5-VSLP due to the relatively low processor utilization of the CNC task set.

For task sets with low processor utilization, the results for 5-GNF and 5-VSLP are identical. This is because in these cases, the highest voltage used for scheduling jobs is less than the lowest voltage available in the processor. Therefore, the execution voltages of jobs that execute at voltage values less than 0.9 V (for the Crusoe processor) are raised to 0.9 V. However, at higher utilization values (0.65 and higher), 5-VSLP performs

TABLE II
DETAILS OF EVALUATION TASK SETS

Task set number	Processor utilization	Task parameters			
		Task	Execution time	Period	Deadline
$\mathcal{T}1$	0.27	1	0.25	14	14
		2	1	21	21
		3	1	21	21
		4	1	7	7
		5	0.25	14	14
$\mathcal{T}2$	0.43	1	3	30	30
		2	2	15	15
		3	0.5	10	10
		4	6	60	60
		5	0.5	10	10
$\mathcal{T}3$	0.54	1	0.5	14	14
		2	2	7	7
		3	0.5	14	14
		4	2	14	14
		5	1	21	21
$\mathcal{T}4$	0.59	1	1	12	12
		2	4	18	18
		3	3	24	24
		4	4	36	36
		5	2	36	36
$\mathcal{T}5$	0.65	1	0.5	7	7
		2	3	14	14
		3	0.5	7	7
		4	3	28	28
		5	4	21	21
$\mathcal{T}6$	0.72	1	4	30	30
		2	2	18	18
		3	3	10	10
		4	2	36	36
		5	2	15	15
$\mathcal{T}7$	0.75	1	1	7	7
		2	1	7	7
		3	3	21	21
		4	1	7	7
		5	5	28	28
$\mathcal{T}8$	0.85	1	5	12	12
		2	2	18	18
		3	1	20	20
		4	4	30	30
		5	5	36	36

worse than 5-GNF. This is because VSLP distributes slack uniformly to all jobs within a critical interval. In order to do this, it computes the processor utilization of all the jobs in the critical interval and then scales down the frequency of execution of the jobs in the critical interval such that each job sees an equal increase in execution time (yet no job misses its deadline). Thus, within each critical interval, all jobs run at a constant voltage. Now, consider the case where the CPU is allowed to assume only two frequencies, say, $V_{\max}$, and $V_{\max}/2$. If the constant minimum voltage required by VSLP to schedule a set of jobs in a critical interval is $0.57V_{\max}$, the set of jobs cannot be scheduled at $0.5V_{\max}$ because this exceeds the time available for job execution in the interval. Therefore, the next highest speed, which is $V_{\max}$, is chosen to schedule all the jobs within the critical interval. This causes a corresponding rise in the energy consumption using VSLP with a processor with a few speeds, and is the reason 5-VSLP performs worse than 5-GNF at high values of processor utilization.

TABLE III
COMPARISON OF GNF WITH EXECUTION AT A SINGLE PROCESSOR SPEED

Processor model	Processor utilization	Energy consumption ($\propto v_i^2 c_i$)		$\Delta E = \frac{E_{5\text{GNF}} - E_{\text{SS}}}{E_{\text{SS}}}$
		Single-speed	5-GNF	
Crusoe	0.27	38.8	26.91	-30.6%
	0.43	43.94	30.42	-30.6%
	0.54	77.74	53.82	-30.7%
	0.59	72.67	50.31	-25.9%
	0.65	92.90	64.35	-30.7%
	0.72	223.08	164.03	-26.4%
	0.75	106.47	80.05	-24.8%
	0.85	258.57	219.90	-14.9%
AMD	0.27	74.52	57.96	-21.9%
	0.43	84.24	65.52	-22.2%
	0.54	149.04	115.92	-22.2%
	0.59	139.32	108.36	-22.2%
	0.65	178.20	138.60	-22.2%
	0.72	427.68	332.64	-22.2%
	0.75	204.12	158.70	-22.2%
	0.85	495.72	421.78	-14.9%

In the GNF model, this equal allocation of slack does not take place. Jobs are allocated slack in a "lump-sum" manner, which appears to be more effective than an equal distribution of slack to all jobs when processor utilization is high. Note that the increased energy consumptions for the utilization values of 0.72 and 0.85 are due to the larger hyperperiods of the corresponding task sets.

A similar trend is also observed in the comparison of 5-GNF with LPEDF, shown in Table V. The GNF method for discrete speed processors performs better than 5-LPEDF at high utilization values. The reason for this observation is the same as that described in the comparison between 5-GNF and 5-VSLP. However, the value of processor utilization at which GNF outperforms LPEDF is much higher than the one for VSLP. Note also that LPEDF performs marginally better than VSLP (both algorithms are designed for continuous-speed processors). This is because the number of critical intervals generated by LPEDF is less than the number of critical intervals in VSLP. (This demonstrates that the scheduling strategy—either fixed-priority or dynamic priority—plays a role in energy consumption.)

Finally, we compare the GNF method to an algorithm designed specifically for discrete-speed processors [14], referred to here as D-LPEDF. This algorithm builds on LPEDF and discretizes a continuous-speed voltage schedule using the transformation described in [11]. The comparison of 5-GNF with D-LPEDF is shown in Table VI. D-LPEDF assumes that voltage transitions can occur within critical intervals, and can therefore potentially increase the number of preemptions. In the few cases where D-LPEDF slightly outperforms 5-GNF, it requires an additional preemption for the task set. If the cost of preemption is high, this may lead to undesirable results. In the GNF method, we restrict voltage switching to occur only at task boundaries.

The solutions of the GNF models are provably optimal lower bounds on energy consumption under the assumption that voltage switching can occur only at task boundaries. It therefore serves as a baseline against which other DVS schemes can be compared. However, with a large number of speeds, the GNF models do not run to completion. Hence, heuristic solution techniques such as VSLP are useful in such cases.

TABLE IV
COMPARISON OF GNF AND VSLP [22]

Processor model	Processor utilization	Energy consumption ($\propto v_i^2 c_i$)			$\Delta E = \frac{E_{5GNF} - E_{5VSLP}}{E_{5VSLP}}\%$	Execution time		
		VSLP [22]	5-VSLP	5-GNF		VSLP	5-VSLP	GNF
	0.27	11.22	26.91	26.91	0%	0.01s	0.01s	6s
	0.43	19.04	30.42	30.42	0%	0.01s	0.01s	1s
Crusoe	0.54	42.97	53.82	53.82	0%	0.01s	0.01s	9s
	0.59	43.44	50.31	50.31	0%	0.01s	0.01s	15s
	0.65	62.04	69.03	**64.35**	−6.7%	0.01s	0.01s	48s
	0.72	164.15	175.76	**164.03**	−6.6%	0.02s	0.02s	1m,3s
	0.75	82.20	86.71	**83.72**	−3.4%	0.01s	0.01s	1m,5s
	0.85	220.67	235.56	**219.90**	−6.6%	0.01s	0.01s	45s
	CNC (0.48)	50975.60	71358.30	71358.30	0%	0.5s	0.5s	1h,23m
	0.27	21.52	57.96	57.96	0%	0.01s	0.01s	6s
	0.43	36.5	65.52	65.52	0%	0.01s	0.01s	1s
AMD	0.54	82.38	115.92	115.92	0%	0.01s	0.01s	9s
	0.59	83.29	108.36	108.36	0%	0.01s	0.01s	15s
	0.65	118.95	142.92	**138.60**	−3%	0.01s	0.01s	25s
	0.72	314.72	338.40	332.64	−1.7%	0.02s	0.02s	1m,3s
	0.75	157.6	172.80	**158.70**	−8.1%	0.01s	0.01s	1m
	0.85	423.06	441.72	**421.78**	−4.5%	0.01s	0.01s	45s

TABLE V
COMPARISON OF GNF AND LPEDF [30]

Processor model	Processor utilization	Energy consumption ($\propto v_i^2 c_i$)			$\Delta E = \frac{E_{5GNF} - E_{5LPEDF}}{E_{5LPEDF}}\%$	Execution time		
		LPEDF [30]	5-LPEDF	5-GNF		LPEDF	5-LPEDF	GNF
	0.27	10.64	26.91	26.91	0%	0.01s	0.01s	6s
	0.43	19.04	30.41	30.41	0%	0.01s	0.01s	1s
Crusoe	0.54	42.57	53.82	53.82	0%	0.01s	0.01s	9s
	0.59	43.40	50.31	50.31	0%	0.01s	0.01s	15s
	0.65	60.86	64.34	**64.35**	0%	0.01s	0.01s	48s
	0.72	163.59	171.59	**164.03**	−4.4%	0.02s	0.02s	1m,3s
	0.75	79.85	81.89	83.72	2.2%	0.01s	0.01s	1m,5s
	0.85	219.78	238.67	**219.90**	−7.8%	0.01s	0.01s	45s
	CNC (0.48)	50975.6	71358.30	71358.30	0%	0.5s	0.5s	1h,23m
	0.27	20.40	57.95	57.96	0%	0.01s	0.01s	6s
	0.43	36.50	65.51	65.52	0%	0.01s	0.01s	1s
AMD	0.54	81.61	115.91	115.92	0%	0.01s	0.01s	9s
	0.59	83.20	108.36	108.36	0%	0.01s	0.01s	15s
	0.65	116.67	138.60	**138.60**	0%	0.01s	0.01s	25s
	0.72	313.63	332.63	332.64	0%	0.02s	0.02s	1m,3s
	0.75	153.08	158.76	**158.75**	0.003%	0.01s	0.01s	1m
	0.85	421.36	440.63	**421.78**	−4.2%	0.01s	0.01s	45s

Finally, we consider the DVS problem for processor models with less than five speeds. Examples of such embedded processors are the embedded SL enhanced Intel486 DX2 processor [10], which operates at 5.5 and 3.3 V, and the Motorola 6805 [21], which operates at 5.5, 3.3, and 2.2 V. This comparison highlights the importance of the number of available speeds as an important evaluation parameter for DVS algorithms. Our results for these processors are shown in Table VII. Here too, the GNF method outperforms VSLP for all task sets with high processor utilization.

We next evaluate the performance of the minimum-cost network flow model for $\mathcal{P}_{\text{cpu}}$. We compare the minimum-cost network flow method to VSLP and LPEDF. Since the minimum-cost models for $\mathcal{P}_{\text{cpu}}$ are scalable to processor models with a large number of speeds, we first consider these kinds of processors. Therefore, similar to the processor model used in [22], we assume that the maximum operating frequency of the CPU is 100 MHz at 3.3 V and the minimum frequency is 8 MHz at 0.264 V (note, however, that LPEDF uses an infinite range of frequencies). To solve our models, we used a scaling-based network flow solver [4], [6].

The three methods are evaluated with the same data sets, and the results are presented in Table VIII. The solutions of the minimum-cost models closely match the solutions generated by VSLP and LPEDF. However, unlike the GNF models, the execution times for solving the minimum-cost method for all task sets were comparable to the execution times of VSLP and LPEDF. Thus, this method results in efficient solutions in runtimes that are comparable to VSLP and LPEDF.

For processor models with a few speeds, the minimum-cost method results in better solutions than VSLP. We also generated minimum-cost models for the Crusoe processor (with five speeds) and compared the results with 5-VSLP. These results are presented in Table IX. For task sets of low utilization, the

TABLE VI
COMPARISON OF GNF AND D-LPEDF [14]

Processor model	Processor utilization	Energy consumption ($\propto v_i^2 c_i$)	
		D-LPEDF [14]	5-GNF
	0.27	26.91	26.91
	0.43	30.41	30.41
Crusoe	0.54	53.82	53.82
	0.59	50.31	50.31
	0.65	64.34	64.35
	0.72	164.03	164.03
	0.75	80.48	83.72
	0.85	219.86	219.90
	0.27	57.95	57.96
	0.43	65.51	65.52
AMD	0.54	115.91	115.92
	0.59	108.36	108.36
	0.65	138.59	138.59
	0.72	332.63	332.64
	0.75	158.75	158.75
	0.85	421.73	421.90

TABLE VII
RESULTS FOR THE INTEL 486DX2 AND MOTOROLA 6805 PROCESSORS

Processor utilization	Voltages used (V)	Energy ($\propto v_i^2 c_i$)		ΔE= $\frac{E_{\mathrm{GNF}} - E_{\mathrm{VSLP}}}{E_{\mathrm{VSLP}}}$
		VSLP	GNF	
	5.5,3.3 (Intel)	379.50	379.50	0%
0.27	5.5,3.3,2.2 (Motorola)	253.00	253.00	0%
	5.5, 3.3 (Intel)	961.95	**780.45**	-18.8%
0.59	5.5, 3.3, 2.2 (Motorola)	961.95	**777.40**	-19.18%
	5.5,3.3 (Intel)	1009.50	**907.50**	-10.1%
0.65	5.5,3.3,2.2 (Motorola)	1009.50	**902.25**	-10.6%
	5.5,3.3 (Intel)	3993.00	**3121.80**	-2.1%
0.72	5.5,3.3,2.2 (Motorola)	3993.00	**3121.80**	-2.1%
	5.5,3.3 (Intel)	1345.50	**1226.83**	-8.8%
0.75	5.5,3.3,2.2 (Motorola)	1345.50	**1226.83**	-8.8%
	5.5,3.3 (Intel)	4628.25	**4136.15**	-10.6%
0.85	5.5,3.3,2.2 (Motorola)	4628.15	**4136.15**	-10.6%

TABLE VIII
EXPERIMENTAL RESULTS FOR THE MINIMUM-COST MODEL FOR $\mathcal{P}_{\mathrm{cpu}}$ FOR TASK SETS OF VARYING UTILIZATION USING THE ARM PROCESSOR MODEL

Utilization	Energy consumption ($\propto v_i^2 c_i$)			Execution time		
	VSLP	LPEDF	Min-cost	VSLP	LPEDF	Min-cost
0.27	6.60	6.30	6.30	$<$ 1s	$<$ 1s	$<$ 1s
0.43	11.20	11.20	11.20	$<$ 1s	$<$ 1s	$<$ 1s
0.54	25.40	25.19	25.20	$<$ 1s	$<$ 1s	$<$ 1s
0.59	25.70	25.68	25.73	$<$ 1s	$<$ 1s	$<$ 1s
0.65	36.70	36.00	36.00	$<$ 1s	$<$ 1s	$<$ 1s
0.72	97.13	96.80	96.83	$<$ 1s	$<$ 1s	$<$ 1s
0.75	48.64	47.25	47.25	$<$ 1s	$<$ 1s	$<$ 1s
0.85	130.57	130.05	130.05	$<$ 1s	$<$ 1s	$<$ 1s
CNC (0.48)	30163.10	29975.30	29975.30	10s	3s	4s

minimum-cost model generates optimal-energy schedules. For task sets with higher processor utilization, the minimum-cost method begins to diverge from the optimal solution, but still performs slightly better than 5-VSLP.

VII. CONCLUSION

In this paper, we have solved the problem of minimizing the energy consumption of an embedded CMOS processor in a hard real-time system optimally using network flow techniques. We have developed a GNF model for the DVS problem when voltage switches are limited to task boundaries. The GNF problem can be solved efficiently when the CPU can operate only at a small number of admissible speeds. We have proved that the solutions generated using the GNF method are optimal and represent theoretical upper bounds on energy savings that can be achieved using DVS. The GNF method for discrete-speed processors performs significantly better than competing schemes. The models are also scalable to large real-life task sets. For processors with a large range of frequencies, we have shown how $\mathcal{P}_{\mathrm{cpu}}$ can be modeled as a minimum-cost network flow graph and solved using a fast solution algorithm to generate near-optimal task and voltage schedules in reasonable periods of time. This technique is as efficient as the best results published thus far in the literature with processors with a large range of frequencies, and performs better than competing methods when the processor speeds are limited to a small set of discrete frequencies.

As part of future work, we are investigating network flow models for intratask voltage scaling and network flow model that consider voltage transition time and energy.

TABLE IX
COMPARISON OF FIVE-SPEED VARIANTS

Utilization	Energy consumption ($\propto v_i^2 c_i$)			$\%\Delta\mathcal{E}$= $\frac{E_{\mathrm{5mc}} - E_{\mathrm{5vslp}}}{E_{\mathrm{5vslp}}}$
	5-VSLP	GNF	5-mincost	
0.27	26.90	26.90	26.90	0%
0.43	30.40	30.40	30.40	0%
0.54	53.80	53.80	53.80	0%
0.59	50.31	50.31	50.31	0%
0.65	69.00	64.30	64.30	-6.7%
0.72	164.15	163.59	165.23	0.6%
0.75	86.71	80.05	81.10	-6.4%
0.85	235.56	219.49	219.87	-6.6%
CNC	71358.90	71358.90	71358.90	0%

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[2] "AMD PowerNow! Technology Platform Design Guide for Embedded Processors,", AMD Document Number 24 267a, Dec. 2000.

[3] A. Azevedo, I. Issenin, R. Cornea, R. Gupta, N. Dutt, A. Veidenbaum, and A. Nicolau, "Profile-based dynamic voltage scheduling using program checkpoints in the COPPER framework," in *Proc. Design Automation Test Conf. Eur.*, 2002, pp. 168–176.

[4] CS2. An Implementation of a Scaling Push-Relabel Algorithm for the Minimum-Cost Flow/Transportation Problems [Online]. Available: http://www.avglab.com/andrew/soft.html

[5] Transmeta Crusoe TM5500 Data Sheet [Online]. Available: http://www.transmeta.com/developers/crusoe_docs.html

[6] A. V. Goldberg, "An efficient implementation of a scaling minimum-cost flow algorithm," *J. Algorithms*, vol. 22, pp. 1–29, 1997.

[7] I. Hong, M. Potkonjak, and M. B. Srivastava, "On-line scheduling of hard real-time tasks on variable-voltage processor," in *Proc. Int. Conf. Computer-Aided Design*, 1998, pp. 653–656.

[8] C.-H. Hsu, U. Kremer, and M. Hsiao, "Compiler-directed dynamic voltage/frequency scheduling for energy reduction in microprocessors," in *Proc. Int. Symp. Low-Power Electron. Design*, 2001, pp. 275–279.

[9] C.-H. Hsu and U. Kremer, "The design, implementation and evaluation of a compiler algorithm for CPU energy reduction," in *Proc. Conf. Program. Lang. Design Implementation*, 2003, pp. 38–48.

[10] Intel Embedded SL Enhanced 486DX2 Processor [Online]. Available: http://www.intel.com

[11] T. Ishihara and H. Yasuura, "Voltage scheduling problem for dynamically variable voltage processors," in *Proc. Int. Symp. Low-Power Electron. Design*, 1998, pp. 197–202.

[12] Operations Research Models and Methods, P. A. Jensen and J. F. Bard. [Online]. Available: www.ormm.net
[13] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin, "Visual assessment of a real-time system design: Case study on a CNC controller," in *Proc. Real-Time Syst. Symp.*, 1996, pp. 300–310.
[14] W.-C. Kwon and T. Kim, "Optimal voltage allocation techniques for dynamically variable voltage processors," in *Proc. Design Automation Conf.*, 2003, pp. 125–130.
[15] J. Luo and N. K. Jha, "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," in *Proc. Int. Conf. VLSI Design*, 2002, pp. 719–726.
[16] E. L. Lawler and C. U. Martel, "Scheduling periodically occurring tasks on multiple processors," *Inform. Process. Lett.*, vol. 12, no. 1, pp. 9–12, 1981.
[17] S. Lee and T. Sakurai, "Run-time voltage hopping for low-power real-time systems," in *Proc. Design Automation Conf.*, 2000, pp. 806–809.
[18] J. W. S. Liu, *Real-Time Systems*. Englewood Cliffs, NJ: Prentice-Hall, 2000.
[19] A. Manzak and C. Chakrabarti, "Variable voltage task scheduling algorithms for minimizing energy," in *Proc. Int. Symp. Low-Power Electron. Design*, 2001, pp. 279–282.
[20] C. Martel, "Preemptive scheduling with release times, deadlines, and due times," *J. ACM*, vol. 29, pp. 812–829, 1982.
[21] Motorola 6805 Processor [Online]. Available: http://www.motorola.com
[22] G. Quan and X. Hu, "Energy efficient fixed-priority scheduling for real-time systems on variable voltage processors," in *Proc Design Automation Conf.*, 2001, pp. 828–833.
[23] T. Sakurai and A. R. Newton, "Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas," *IEEE J. Solid-State Circuits*, vol. 25, pp. 584–594, Apr. 1990.
[24] Y. Shin and K. Choi, "Power conscious fixed priority scheduling for hard real-time systems," in *Proc. Design Automation Conf.*, 1999, pp. 134–139.
[25] D. Shin and J. Kim, "A profile-based energy-efficient intra-task voltage scheduling algorithm for hard real-time applications," in *Proc. Int. Symp. Low-Power Electron. Design*, 2001, pp. 271–274.
[26] D. Shin, J. Kim, and S. Lee, "Intra-task voltage scheduling for low-energy hard real-time applications," *IEEE Design Test Comput.*, pp. 20–30, 2001.
[27] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. Software Eng.*, vol. SE-3, pp. 85–93, Jan. 1977.
[28] V. Swaminathan and K. Chakrabarty, "Generalized network flow techniques for dynamic voltage scaling in hard real-time systems," in *Proc. Int. Conf. Computer-Aided Design*, 2003, pp. 21–25.
[29] J. Xu and D. L. Parnas, "Priority scheduling vs. pre-run-time scheduling," *Int. J. Time-Critical Comput. Syst.*, vol. 18, pp. 7–23, 2000.
[30] F. Yao, A. Demers, and S. Shenker, "A scheduling model for reduced CPU energy," in *Proc. IEEE Annu. Foundations Comput. Sci.*, 1995, pp. 374–382.
[31] F. Zhang and S. T. Chanson, "Processor voltage scheduling for real-time tasks with nonpreemptable sections," in *Proc. Real-Time Syst. Symp.*, 2002, pp. 235–245.
[32] Y. Zhang, X. Hu, and D. Chen, "Task scheduling and voltage selection for energy minimization," in *Proc. Design Automation Conf.*, 2002, pp. 183–188.

Vishnu Swaminathan received the B.E. degree in computer science and engineering from the University of Madras, Chennai, India, in 1996 and the M.S. and Ph.D. degrees in electrical and computer engineering from Duke University, Durham, NC, in 1999 and 2003, respectively.

He is currently a Researcher at Duke University. His research interests are in low-power system design, dynamic power management, and real-time operating systems.

Krishnendu Chakrabarty (S'92–M'96–SM'00) received the B.Tech. degree from the Indian Institute of Technology, Kharagpur, India, in 1990, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1992 and 1995, respectively, all in computer science and engineering.

He is currently an Associate Professor of Electrical and Computer Engineering at Duke University, Durham, NC. From 2000 to 2002, he was also a Mercator Visiting Professor at the University of Potsdam, Potsdam, Germany. His current research projects include: design and testing of system-on-a-chip integrated circuits; embedded real-time systems, distributed sensor networks; modeling, simulation, and optimization of microelectrofluidic systems; microfluidics-based chip cooling. He is a coauthor of two books: *Microelectrofluidic Systems: Modeling and Simulation* (Boca Raton, FL: CRC Press, 2002) and *Test Resource Partitioning for System-on-a-Chip* (Norwell, MA: Kluwer, 2002), and an editor of *SOC (System-on-a-Chip) Testing for Plug and Play Test Automation* (Norwell, MA: Kluwer, 2002). He has published over 160 papers in journals and refereed conference proceedings, and holds a US patent in built-in self-test.

Dr. Chakrabarty is a Member of ACM, ACM SIGDA, and Sigma Xi. He is a recipient of the National Science Foundation Early Faculty (CAREER) Award and the Office of Naval Research Young Investigator Award. He received a best paper award at the 2001 Design, Automation, and Test in Europe (DATE) Conference. He is also the recipient of the Humboldt Research Fellowship, awarded by the Alexander von Humboldt Foundation, Germany. He is an Associate Editor of IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, an Editor of the *Journal of Electronic Testing: Theory and Applications (JETTA)*, and a member of the editorial board for *Sensor Letters* and the *Journal of Embedded Computing*. He has also served as an Associate Editor of the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: ANALOG AND DIGITAL SIGNAL PROCESSING. He serves as Vice Chair of Technical Activities of the IEEE's Test Technology Technical Council and is a Member of the program committees of several IEEE/ACM conferences and workshops.