Power Minimization in QoS Sensitive Systems

Jennifer L. Wong, *Student Member, IEEE*, Gang Qu, and Miodrag Potkonjak, *Member, IEEE*

***Abstract*—The majority of modern multimedia and mobile systems have two common denominators: quality-of-service (QoS) requirements, such as latency and synchronization, and strict energy constraints. However, until now no synthesis techniques have been proposed for the design and efficient use of such systems. We have two main objectives: conceptual and synthesis. The conceptual objective is to develop a generic practical technique for the automatic development of online adaptive algorithms from efficient off-line algorithms using statistical techniques.**

The synthesis objective is to introduce the first design technique for QoS low-power synthesis. We introduce a system of provably-optimal techniques that minimize energy consumption of stream-oriented applications under two main QoS metrics: latency and synchronization. Specifically, we study how multiple voltages can be used to simultaneously satisfy hardware constraints and minimize power consumption while preserving the requested level of QoS. The purpose of the off-line algorithm is threefold. First, it is used as input to statistical software which is used to identify important and relevant parameters of the processes. Second, the algorithm provides buffer occupancy rate indicators. Lastly, it provides a way to combine buffer occupancy and QoS metrics to form a fast and efficient online algorithm. The effectiveness of the algorithms is demonstrated on a number of standard multimedia benchmarks.

***Index Terms*—Low power, quality of service (QoS), synchronization.**

I. Introduction

THE most popular mobile low-power applications, such as audio and video, are stream oriented. The nature of these applications impose a need for addressing the quality-of-service (QoS) requirements under energy constraints. Latency and synchronization are the most relevant QoS metrics in these types of applications. Our goal is to develop a spectrum of techniques and algorithms which minimize energy consumption under the most important QoS metrics. Specifically, we study how to use multiple voltage technologies to simultaneously satisfy hardware requirements and minimize power consumption, while preserving the requested level of QoS in terms of latency and synchronization. Our starting point is a provably optimal off-line algorithm for power minimization under QoS and buffer constraints. In addition to buffer occupancy, a crucial criteria for deciding which process to run at which voltage, we identified four key properties of streaming processes (latency slack, synchronization slack, relative burstiness, and number of tasks.

Manuscript received August 2, 2002; revised January 30, 2003. This work was supported in part by the National Science Foundation under Grant ANI-0085773.

J. L. Wong and M. Potkonjak are with the University of California, Los Angeles, CA 90005 USA (e-mail: jwong@cs.ucls.edu).

G. Qu is the Maryland Institute of Advanced Computer Studies, University of Maryland, College Park, MD 20742 USA.

Digital Object Identifier 10.1109/TVLSI.2004.827567

Our primary goal is to present competitive online algorithms for power minimization for streaming media applications for given hardware resource constraints: latency and synchronization, as well as context switching overhead. We aim to dynamically adjust the supply voltage in such a way that an incoming statistical stream of data does not overflow the buffer capacity of our processing system while expending the least amount of energy. By considering the long and short term statistics of the media streams and current buffer backlog, we decide which supply voltage to apply. Furthermore, by considering latency and synchronization constraints, we decide which task to schedule at the current moment. Finally, we use the new online algorithm to explore the tradeoff between buffer size (cost) and energy consumptions.

II. Related Work

Our research results can be viewed in the context of four related areas: low-power modeling and optimization, quality of service, online algorithms, and statistical techniques.

Mainly due to the demand for mobile applications, low-power research has attracted a great deal of attention in the last decade. A number of researchers proposed the use of multiple voltages in order to reduce power consumption [6], [11], [15], [18]. Furthermore, several variable voltage techniques have been reported [8], [20]. Numerous algorithms for dynamic priority real-time systems have been proposed including [10], [19]. Also, several industrial multiple voltage low-power designs have been reported [8] and prototypes [3], [13]. dynamic power management which aims to reduce the power consumption of electronic systems by selectively shutting down idle components [2]. From one point of view our work can be interpreted as a combination of these two techniques, multiple voltages and system-level power management.

The first QoS requirements, such as bounded delay, guaranteed resolution or synchronization have been addressed in the network and real-time operating systems (RTOS) communities. The most sound and practically relevant QoS model in the networking community was proposed by R. Cruz [7]. The main conceptual results in RTOS literature was presented by Rajkumar *et al.* [16]. They introduced an analytical approach for satisfying multiple QoS dimensions under a given set of resource constraints. They proved that the problem is NP-hard and developed an approximation polynomial algorithm for the problem by transforming it into a mixed integer programming problem [17]. A comprehensive survey of QoS research in these two areas is given in [1]. Recently, the first efforts in QoS, and in particular synchronization during the system design process, has been reported in design automation literature [14].

There are three main conceptual novelties in the presented research with the respect to the previous efforts. The first is that we consider QoS requirements for the streaming media task model. The second is that we have developed one or rare probably optimal algorithms for power minimization at the system level. The final novelty is that online algorithm is developed using statistical methods starting from the off-line algorithms and therefore they provide a generic paradigm for rapid development of effective online algorithms. Comprehensive versions of this paper can be found at [21]–[22].

III. Preliminaries

In this section, we outline the abstraction and models used for power consumption, and define latency, synchronization, and context switch overhead.

The dominating component of power consumption is the switching power. Switching power can be modeled as $P = \alpha \cdot C_L \cdot V_{\text{dd}}^2 \cdot f$, where $\alpha \cdot C_L$ is the effective switching capacitance. This results from the fact that greater throughput comes with the cost of higher voltage. Specifically, the gate delay of circuits is a function of applied voltage and can be calculated using the formula $T = k(V_{\text{dd}}/(V_{\text{dd}} - V_t)^2)$ where k is a constant [4].

We assume the design operates using multiple voltage supplies and that the voltages change instantaneously with no overhead. These changes in voltages are assumed to happen only at the beginning or the end of a time unit.

The demand-supply model for QoS was developed by Cruz *et al.* [7]. The model addresses the burstiness of QoS while handling resource allocation. This model assumes periodic segmentation of the time dimension. During each period, each process receives a task of generally varying complexity. The cumulative sum of tasks for a process can be depicted as a demand curve imposed on the system. The system serves the task sequentially by allocating resources during each time period to one of the processes. The cumulative sum of the processed data forms a supply curve.

The demand curve measures the burstiness of the service requirement. The service curve guides the resource allocation with QoS guarantees. Backlog is defined as the amount of demand that cannot be processed at that time point, and must then be carried over to the next time unit. The backlog in the QoS model is represented by the difference between the vertical positions of the demand curve and the service curve. Latency is the time between when the demand for a task arrives, and when it is processed. This is shown in the model by the horizontal difference between the demand and service curve at any given vertical position. A process is a program which assumes that it has independent use of the CPU. A process is long, in the sense that it consists of many tasks. These tasks are processed at periodic moments in time. With each task, we associate a processing time and a storage requirement.

Latency is defined as the difference between the time when the data is processed and the time the data arrived, i.e., $t_p - t_a$. We denote the time in which a particular sample (piece of date) arrives as t_a and the time when that piece of data is completely processed as t_p. At the intuitive level, synchronization indicates how well two or more processes are correlated in their execution.

We define synchronization in the following way. For the sake of simplicity, consider only two processes, p_1 and p_2. We denote the tasks of the processes p_1 by p_{1i} and p_2 by p_{2j}, where $i, j = \{1, \ldots, n\}$. Perfect synchronization constraints indicate which sample (task or piece of data) of process p_1, which is denoted by p_{1i}, has to be executed at the same time as piece of data p_{2j}. Synchronization tolerance (often for the sake of brevity is solely called "synchronization") indicates the maximal amount of time by which the execution of fully synchronized samples p_{1i} and p_{2j} can maximally differ.

A context switch is the time overhead which is incurred by a multitasking kernel when it decides to process different tasks. The amount of context switching time dramatically depends on the processor. Context switching time for a typical DSP processor is fairly low, around ten cycles, while for a RISC processor it is much higher, approximately 100 cycles. In our experimentation, we used ten cycles.

IV. Offline Optimal Algorithm

In this section, we formulate the off-line QoS low-power problem and present our optimal algorithm. We use a single processor that can operate at multiple supply voltages. The goal is to service multiple processes with minimal energy consumption and the minimal amount of memory while meeting various QoS requirements.

A process consists of a sequence of tasks. With each task t_i, we associate:

- a_i: The arrival time, the time when a task is generated from the process and makes the CPU request.
- p_i: The time needed to complete this task at the nominal voltage v_{ref}.
- s_i: The storage demand which is the minimal amount of memory to store this task on its arrival.

Each of the tasks may have QoS requirements such as latency and synchronization. Latency d_i is the time that task t_i has to be served after its arrival, that is, the actual finish time of task t_i must be earlier than $a_i + d_i$. Synchronization measures the interaction among tasks in different processes. We say that task t_i from one process and task t_j from another are k-synchronized if the difference of their finishing times is within k CPU units. We denote this by $\text{syn}(t_i, t_j) \leq k$.

The variable voltage processor has multiple supply voltages among which it can switch. The processor's processing speed varies as the voltage changes, so will the actual execution time for a task to receive its required amount of service. Suppose a task needs one CPU unit at the nominal voltage v_{ref}, then the execution time to accumulate the same amount of processing at voltage v_{dd} is given by [5]:

$$\frac{(v_{\text{ref}} - v_t)^2}{v_{\text{ref}}} \cdot \frac{v_{\text{dd}}}{(v_{\text{dd}} - v_t)^2} \tag{1}$$

where v_t is the threshold voltage.

Given n processes $\tau^1, \tau^2, \cdots, \tau^n$, each τ^k consists of a sequence of tasks $t_1^k, t_2^k, \cdots, t_n^k$. A *schedule* is a set consisting of the starting time, finishing time, and the voltage level for each task. A schedule is *feasible* if the processor starts each task after its arrival, finishes it before the latency constraint, and satisfies all synchronization requirements. The quality of a schedule is measured by its energy consumption and the memory requirement. Since these two metrics are noncomparable to each other, we introduce the concept of competitiveness. We say two schedules are *competitive* if neither outperforms the other in both energy consumption and memory requirement. We formulate the problem as:

On a processor with multiple voltages, for a given set of processes, find all the feasible competitive schedules.

We make the following assumptions.

- **Tasks from the same process have to be executed and completed in the first-in first-out (FIFO) fashion**. The common processes that we consider are audio, video, and streams generated by sensor networks. For this type of applications, there is a natural intrinsic order in which consecutive tasks have to be executed.
- **A task's processing demand, p_i, is proportional to its storage demand, s_i.** In general, of course, this assumption is not necessarily correct. However, in many situations, the amount of processing required for a specific set of data is proportional to the amount of data.
- **The memory occupied by a task can be partially freed, but only at the end of a CPU unit.**[1] This assumption is a direct consequence of the way how current operating systems function.
- **There is no context switching overhead.** Obviously, in almost all types of processors, there is context switching overhead. The key observation is that the operating system cycle is significantly longer than the overhead and therefore in the first approximation the overhead does not have to be considered.
- **The processor can instantaneously switch the supply voltage, but only at the beginning of each CPU unit.** Physical laws in current technology imply that the change of supply voltage can not be done instantaneously. Recently, there have been several efforts to take this time into account [12]. However, in modern technologies, context switching times are usually two to three orders of magnitude shorter than the operating system cycle. Therefore, this approximation is sound and impacts overall results only nominally.

A. Optimal Solution for Single Process

In this section, we show how to find all feasible competitive solutions for a single process. Suppose the reference voltage $v_{\rm ref}$ is 0.8 V, and there are two different voltage levels $v_{\rm hi} = 3.3$ V and $v_{\rm lo} = 1.8$ V. From (1), we approximate the processing speeds to be 3 and 10 at $v_{\rm lo}$ and $v_{\rm hi}$, respectively. Consider a process with six tasks, $t_0, t_1, \cdots, t_5$. For simplicity, we further assume that task t_i arrives at time i, and there are no deadline constraints. Finally, we assume that the processing and memory requirement are $\mathbf{4, 7, 12, 3, 5}$, and $\mathbf{1}$, respectively, for the six tasks. The goal is to determine the voltage to use for each unit time, such that the energy consumption is minimized. We developed a dynamic programming-based algorithm which achieves polynomial run time for this task.

[1]Memory can be partially freed means that, for instance, if half of the processing demand is fulfilled at the end of one CPU unit, then we are able to free half of the space used to store this task. Our proposed algorithm can be easily modified when this is not allowed.

TABLE I
MEMORY REQUIREMENTS FOR THE MOTIVATIONAL EXAMPLE

	0	1	2	3	4	5	6	7	8	9	10	11
0	4	8	17	17	19	17	14	11	8	5	2	0
1	7	12	12	14	10	7	4	1	0			
2	12	5	7	5	2	0						
3	5	5	1	0								
4	5	1	0									
5	1	0										
6	0											

Table I shows the memory requirement at the end of each unit time, which is the minimal amount of memory required to store all the arrived but unfinished tasks. The table uses the time (in terms of CPU units) that the processor is operating at $v_{\rm lo}$ and $v_{\rm hi}$ to label the horizontal and vertical axis respectively. For example, entry (i, j) is the minimal memory requirement after running at $v_{\rm hi}$ for i CPU units and at $v_{\rm lo}$ for j units.

Consider entry (1,1), whose content is the storage we need at the end of the second unit of time after we use $v_{\rm lo}$ for one unit of time and $v_{\rm hi}$ for one unit. We can either apply $v_{\rm hi}$ in the first unit and $v_{\rm lo}$ in the second unit, or start at $v_{\rm lo}$ and switch to $v_{\rm hi}$ after one CPU unit. In the first case, since task t_0's processing demand is 4 and we are able to process 10 at $v_{\rm hi}$, we will finish t_0, free the memory, and wait for t_1; then at $v_{\rm lo}$ in the next unit of time, we can finish 3 out of the 7 units of processing demand from t_1; now t_2 is arriving, therefore we need a total of $(7-3)+12 = 16$ units of memory to store t_1 and t_2. In the second case, $v_{\rm lo}$ then $v_{\rm hi}$, we can only finish 3 out of the 4 processing demand of task t_0 by the end of the first unit of time due to the slow processing speed at $v_{\rm lo}$; however, after raising the voltage to $v_{\rm hi}$ during the second unit, we are able to finish both the remaining of t_0 and entire t_1; the storage for tasks t_0 and t_1 are freed and therefore when t_2 arrives, we only need 12 units of storage to store this new task. Thus, we fill entry (1,1) with 12, the smaller storage requirement of the two different strategies.

Let $m(i, j)$ be the content of entry (i, j). We can reach this entry from entry $(i-1, j)$ by applying $v_{\rm hi}$ or from its left neighbor $(i, j-1)$ by applying $v_{\rm lo}$, hence, we have

$$m(i,j) = \min\left(s_{i+j} + \max\left(0, m(i-1,j) - sp_{\rm hi}\right)\right. \\ \left. s_{i+j} + \max\left(0, m(i,j-1) - sp_{\rm lo}\right)\right) \quad (2)$$

where $sp_{\rm lo}$ and $sp_{\rm hi}$ are the processing speed at $v_{\rm lo}$ and $v_{\rm hi}$, respectively. The inner max is introduced to enforce that excess processing resource cannot be used for future work. We build Table I based on (2), where every row ends with an entry of 0 meaning that there are no tasks left.

While $m(i, j)$ gives the minimal storage requirement at the instant $i + j$, we may have used more storage already before this time. We further denote $M(i,j)$ as the minimal amount of storage that has been used up to time $i + j$ after running i units of time at $v_{\rm hi}$ and j at $v_{\rm lo}$. Considering the voltage being used in the $(i + j)$th unit, we observe that if we use $v_{\rm hi}$, we

TABLE II
STORAGE REQUIREMENTS FOR THE MOTIVATIONAL EXAMPLE

	0	1	2	3	4	5	6	7	8	9	10	11
0	4	8	17	17	19	19	19	19	19	19	19	19
1	7	12	12	14	19	19	19	19	19			
2	12	12	12	14	14	14						
3	12	12	12	14								
4	12	12	12									
5	12	12										
6	12											

TABLE III
MEMORY AND ENERGY FOR DIFFERENT SCHEDULES

	(6,0)	(5,1)	(4,2)	(3,3)	(2,5)	(1,8)	(0,11)
M	12	12	12	14	14	19	19
E	6.0	5.1	4.2	3.3	2.5	1.8	1.1

can finish at most $\max(m(i-1,j), sp_{\text{hi}})$ and need a storage of $s_{i+j} + \max(0, m(i-1,j) - sp_{hi})$. Moreover, previously we have already required a storage at the amount of $M(i-1,j)$. This implies that

$$M(i,j) \geq \max\left(M(i-1,j), s_{i+j} + \max\left(0, m(i-1,j) - sp_{\text{hi}}\right)\right). \tag{3}$$

Similar inequality holds if we use v_{lo}, therefore, we have

$$\begin{aligned} M(i,j) = \min\big(&\max\left(M(i-1,j), s_{i+j}\right.\\ &\left.+\max\left(0, m(i-1,j) - sp_{\text{hi}}\right)\right),\\ &\max\left(M(i,j-1), s_{i+j}\right.\\ &\left.+\max\left(0, m(i,j-1) - sp_{\text{lo}}\right)\right)\big) \end{aligned} \tag{4}$$

Based on the recursive formulas (2) and (4), we calculate $M(i,j)$'s and store them in Table II, where the last entry of the ith row gives the minimal storage requirement to complete all the tasks by using v_{hi} for exactly i units.

The power consumption at $v_{\text{hi}} = 3.3$ V is 1, then the power consumption at $v_{\text{lo}} = 1.8$ V is 0.1 from our power model. Unlike the storage requirement, energy consumption is path independent. i.e., it depends on the total number of CPU units that we have used at v_{lo} and v_{hi}, not the voltage at every individual time unit.

Table III gives the memory requirements and total energy consumptions by different scheduling policies, where (i,j) in the first row indicates a schedule that uses v_{hi} for i units and v_{lo} for j units. Clearly from this table, we see that there exist three competitive optimal solutions, (4,2), (2,5), and (0,11). They consume different amounts of energy and require different amounts of memory. We can then choose the one that fits our preference of memory and energy, and retrieve the actual schedule (i.e., the voltage for each CPU unit) by using simple backtracking.

Fig. 1 shows the algorithm of finding all the competitive optimal solutions for multiple processes. A schedule in this case has to determine, for each CPU unit, which process to be executed and at which voltage level.

Assuming that there are m processes and k different voltages, we have $m \cdot k$ choices: running the ith process at voltage v_j $(1 \leq i \leq m, 1 \leq j \leq k)$. A *state* $S = (e_1, \cdots, e_m; u_1, \cdots, u_k)$ means that the ith process has been allocated e_i CPU units, and the processor has been working

```
Input: $m$ processes with their arrival time, processing load, and
       other timing requirement(deadline, synchronization, etc.);
       $k$ different supply voltages.
Output: All competitive pairs of memory requirement and
        energy consumption, and one schedule for each such pair.

Algorithm:
Phase I: Configuration for all states.
1.  Compute $Next(S_0)$ for the initial state $S_0$;
2.  for each $S \in Next(S_0)$
3.      { $Prev(S) = S_0$;
4.        $\mathcal{S} = \mathcal{S} \cup S$;}
5.  while ( $\mathcal{S} \neq \phi$ )
6.      { for each $S \in \mathcal{S}$
7.          { current_max_memory for state $S = \infty$;
8.            for each $S' \in Prev(S)$
9.                { calculate the max_memory requirement
                    if $S$ follows $S'$;
10.                 if ( max_memory $\leq$ current_max_memory )
11.                   { current_max_memory = max_memory;
12.                     current_previous_state = $S'$; }
                  }
13.           max_memory for state $S$ = current_max_memory;
14.           previous_state for $S$ = current_previous_state;
          }
15.       Compute $Next(S)$;
16.       $\mathcal{S} = \mathcal{S} \cup Next(S) - S$;
17.     for each $S' \in Next(S)$
18.         $Prev(S') = Prev(S') \cup S$; }
Phase II: calculation for energy consumption.
19. for each final state $S$
20.     calculate the energy_consumption for $S$;
21. compute all the competitive final states $\mathcal{F}$;
Phase III: determine one schedule for each competitive state.
22. for each competitive state $S = (e_1, \cdots, e_m; u_1, \cdots, u_k) \in \mathcal{F}$
23.     { index $= l = \sum_{j=1}^{k} u_j$;
24.       $S_{index} = S$;
25.       while ( index $\neq 0$)
26.         { $S'$ = previous_state for $S_{index}$;
27.           index = index - 1;
28.           $S_{index} = S'$; }
29.       report the schedule $(S_0, S_1, \cdots, S_l)$ for $S$; }
```

Fig. 1. Algorithm for all off-line competitive schedules.

at voltage v_j for u_j CPU units. Notice that $\sum_{i=1}^{m} e_i \leq \sum_{j=1}^{k} u_j$ and the equality holds if and only if at any time, there exists unfinished process(es). We say state $S = (e_1, \cdots, e_m; u_1, \cdots, u_k)$ *precedes* $S' = (e'_1, \cdots, e'_m; u'_1, \cdots, u'_k)$ if i) $e'_i \geq e_i$, ii) $u'_j \geq u_j$, iii) $\sum_{i=1}^{m} e'_i - e_i \leq 1$, and iv) $\sum_{j=1}^{k} u'_j - u_j = 1$. If S precedes S', we say that S' *follows* S. We define $\text{Prev}(S) = \{S' : S' \text{ precedes } S\}$, and $\text{Next}(S) = \{S' : S \in \text{Prev}(S')\}$. A state S is *reachable* if $\text{Prev}(S) \neq \phi$. A *final* state is a state when all processes' requests are satisfied. A schedule is a sequence of states $\{S_1, S_2, \cdots\}$ such that $S_l \in \text{Prev}(S_{l+1})$ and all processes' processing loads are satisfied at the final state.

The off-line optimal algorithm consists of three phases. First, we build an $(m \times k)$-dimensional table which stores the minimal memory requirements. For example, when there is only one process and two voltages, then we will have a table like Table II. Step 1 computes the Next set for the initial state S_0, if all m processes require CPU time at the beginning, then this set will have $m \times k$ elements. Steps 2–4 makes all the states in $\text{Next}(S_0)$ reachable, since each state is one move away from the initial state S_0. We denote the set of reachable states by $\mathcal{S}$. Steps 5–18 build the table recursively until there is no reachable state. We keep all the reachable states in a queue, we calculate the Next set for the head of the queue (state S) in Step 15, delete S from

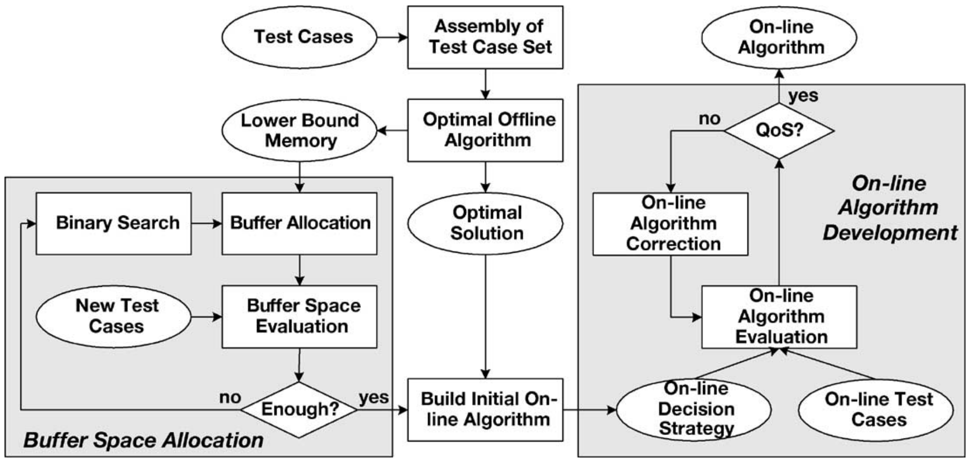
Test Cases
Assembly of Test Case Set
Lower Bound Memory
Optimal Offline Algorithm
Binary Search
Buffer Allocation
Optimal Solution
New Test Cases
Buffer Space Evaluation
no
Enough?
yes
Build Initial On-line Algorithm
Buffer Space Allocation
On-line Algorithm
yes
no
QoS?
On-line Algorithm Development
On-line Algorithm Correction
On-line Algorithm Evaluation
On-line Decision Strategy
On-line Test Cases

Fig. 2. Overall flow for the creation of the online algorithm.

the queue and put all elements of $\mathrm{Next}(S)$ into the queue in Step 16. When we compute $\mathrm{Next}(S)$, we consider all the timing requirements. For example, if process i has a deadline at the end of next CPU unit and its remaining process requirement can be fulfilled only when we use the highest voltage, then $\mathrm{Next}(S)$ will contain only one state, which assigns the current CPU unit to process i and applies the highest voltage. Because all other schedules will fail to meet process i's deadline. The memory requirement for each state is calculated using formulas similar to (2) and (4).

From the table built in the first phase, we can easily see the total memory requirement for each schedule, which is the value at its corresponding final state. In Phase II, we calculate their energy consumption. Recall that the energy consumption is path-independent. Let P_j be the power for voltage v_j, then for final state $S = (e_1, \cdots, e_m; u_1, \cdots, u_k)$, all schedulers with this final state will consume energy in the amount of $E = \sum_{j=1}^{k} P_j \cdot u_j$. So for each final state, we associate with the pair (M, E), the memory requirement and the energy consumption. Recall also that two final states S and S' are *competitive* if i) $M \leq M'$ and $E \geq E'$, or ii) $M \geq M'$ and $E \leq E'$.

In the third phase, we find a schedule for each competitive final state. We achieve this by using backtracking as shown in Steps 23–29. The existence of state S' in Step 26 is guaranteed by the way in which we build the memory requirement table in Phase I. Therefore, we have:

Theorem 4.1: The algorithm in Fig. 1 finds all the feasible competitive schedules.

We analyze the complexity of the algorithm, for a fixed processor that has k supply voltages to execute m processes, in terms of the total processing demands. Suppose that we need X CPU units to service all the processes at the reference voltage. In Phase I, we essentially fill in the entries of an $m \times k$ dimensional table.

The calculation of energy consumption in Phase II takes constant time for each final states. According to how many units we have run at the reference voltage, it is clear that we will have at most X different final states (c.f. Table II for an example). Thus, the cost here is $O(X)$. In the last phase, we determine a feasible schedule for each competitive final state by backtracking. In step 27, we move one entry closer to the starting point, and the total number of steps we need is also in the order of $O(X)$. Therefore, we have:

Theorem 4.2: If we need X CPU units to service all the processes at the reference voltage, the run-time of the proposed algorithm is $O(X^{\mathrm{mk}})$.

V. Online Heuristics

In this section, we present the online algorithm for power minimization under the QoS constraints, synchronization and latency.

We have multiple online streaming processes, with tasks which arrive at periodic time intervals. For each task of each process, we have memory and CPU requirements. Each of these tasks have a given latency constraint, and on some subset of these tasks additional synchronization constraints are imposed. We are given multiple supply voltage levels in which to execute these tasks. The goal of the online algorithm is to decide which task from the stream processes to execute at each time interval and at which voltage in such a way that all latency and synchronization constraints are satisfied. Additionally, at no point of time the requirements for storage should exceed the memory size (buffer space).

In order to solve the overall problem, we must answer the following three questions: (i) how much buffer space is needed; (ii) which task to execute; and (iii) which voltage to apply. The answer to each of these questions is determined by our synthesis and online scheduling approach which is presented in Fig. 2. The online approach uses the optimal off-line algorithm to determine its decision mechanism.

The online approach begins with the assembly of a diverse set of test cases. The off-line optimal algorithm provides a lower bound on the memory requirement for the system along with the optimal QoS solution for the test set. The lower bound memory requirement is used to determine the proper buffer allocation size for the online algorithm. In this phase, a binary search on the size of the buffer is conducted. Each iteration tests the new buffer size on a new set of test cases, until the buffer space allocated is sufficient to handle all considered cases.

Next, the buffer size and the optimal solutions are used to build the online algorithm. The initial online algorithm builds a statistical model from the optimal off-line solutions and creates an online decision strategy, which is used in order to select the proper task and voltage in which to execute in each situation. The decision strategy is then evaluated on a set of online test cases. If the decision strategy does not provide the level of QoS specified, then modification of not only the statistical model and the decision strategy, but also the allocated buffer space is conducted. We continue to make modifications until the desired level QoS is reached.

The initial online algorithm is created in five steps. In the first step, we identify the relevant properties for the QoS requirement. For example, in the case of latency and synchronization, we define properties such as average latency, maximum synchronization delay, and buffer occupancy. We evaluate the relevance of these properties in terms of the off-line optimal algorithm in the second step. We eliminate all properties which show little relevance to the outcome of the optimal off-line solutions. Following this step, both the optimal off-line solutions and the relevant properties are used to build the statistical model. We build a $n \times m$-dimensional space, where n is the number of properties and m is the number of processes. The resolution of each property is specified, and for each subspace we determine the statistical values for task selection. Each subspace contains the percentage of time the optimal off-line algorithm selected each of the tasks under the defined property conditions. In a similar way, the statistical values for all situations and each voltage level is calculated.

Before we evaluate the effectiveness of the model, we have to develop the online decision strategy. The strategy is responsible for making the decision as to which task and which voltage to select according to the particular combination of property values. The strategy is reliant on the context switch time or penalty. For each subspace, in the task selection statistical model, the decision strategy must decide with task to select based on the values in the subspace and the context switch penalty. If there was no penalty for context switching, then for each situation we would select the statistically strongest task from the statistical model. However, if the context switch penalty is high, we would like to continue to run the tasks of the currently selected process as long as possible. In the moderate case, the proper time to switch between processes needs to be defined, and therefore we propose different points in the statistical model to switch between processes. The final step is to evaluate each of these proposed points to determine the proper switching point in the model. Once the proper switching points has been defined, we compact the $n \times m$-dimensional table by combining subspaces with the same task/process selected to execute. Statistically, the subspaces should not be interleaved, and therefore we shall have continuous subspaces. For each decision the online algorithm determines which subspace the properties fall into, and select the assigned task/process to execute. The same process is applied to determining the voltage selection decision strategy. This online decision strategy is then passed on to the final stage of the overall online approach.

The online algorithm builds a statistical model and an online decision strategy based on the $n \times m$-dimensional space defined by the properties. The goal is to select properties which provide strong indication of which task should be run at which voltage. We have defined the following five properties.

- **Latency**. If the latency of multiple tasks of a process are close to their maximum allowed latency, this process should be selected. Additionally, a higher voltage should be run to ensure each of the tasks meet their latency requirements. However, if the latency for all tasks/processes are at lower levels, then the task of the current process should be executed to eliminate a context switching penalty.
- **Relative Burstiness**. The recent burstiness of a process, or rapid arrival of tasks for a process, can play an important role in voltage and task selection. If a task has shown recent burstiness, we should consider the execution of the task/process due to the likelihood that this task will continue to be bursty, therefore consuming more buffer space and extending the latency of each of the tasks if they are not run.
- **Number of Tasks**. The number of tasks which a process has waiting also plays a key in the task and voltage selection process. If the current selected process has more tasks than the other processes, the tasks of the current process should continue to be selected in order to eliminate context switching penalties.
- **Synchronization**. When the synchronization for any task/process is nearing the maximum allowed level for QoS this task should be selected.
- **Buffer Occupancy**. Buffer occupancy is an indiction of the current demand of the processes as a whole. This property looks at the percentage of the entire buffer in which each process occupies. If the buffer is near capacity, the processes with higher buffer occupancy should be selected.

VI. Experimental Results

In this section, we present the experimental results obtained using comprehensive simulation study. We first describe the used examples (multimedia applications). After that, we present our experimental setup and collected data. Finally, we conduct the analysis of the experimental results.

In order to evaluate the online heuristic, we adapt the following procedure. We use four CPU units for the latency constraints. For synchronization, we use eight CPU units. The goals of the our experimentation and results analysis was to answer the following questions: Are multiple voltages useful? How many voltages are needed? What is the relative quality of the online algorithm with comparison to the optimal off-line scheme? How much benefit one can obtain for online algorithms when the goal is to minimize design costs (buffer storage) under energy consumption constraints?

We used six streaming applications [9] to evaluate the effectiveness of the approach: IJG JPEG encoder and decoder, MSG MPEG encoder and decoder, CCITT G.721 encoder, and PGP encryption and description module.

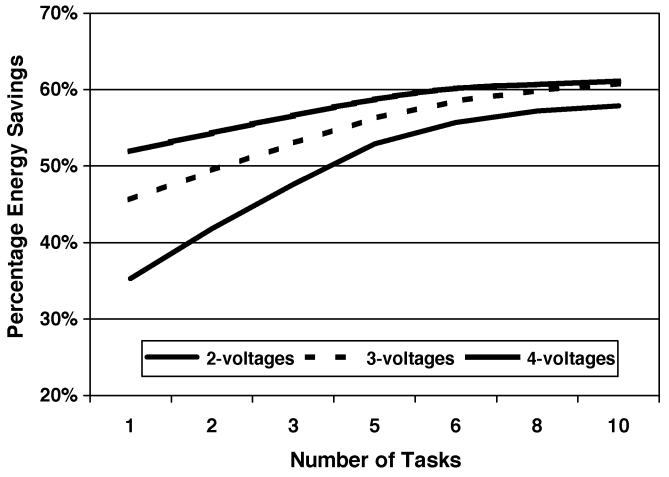

Fig. 3. Energy savings by off-line algorithm using multiple supply voltage assuming 3.3 V nominal voltage with the same amount of memory.

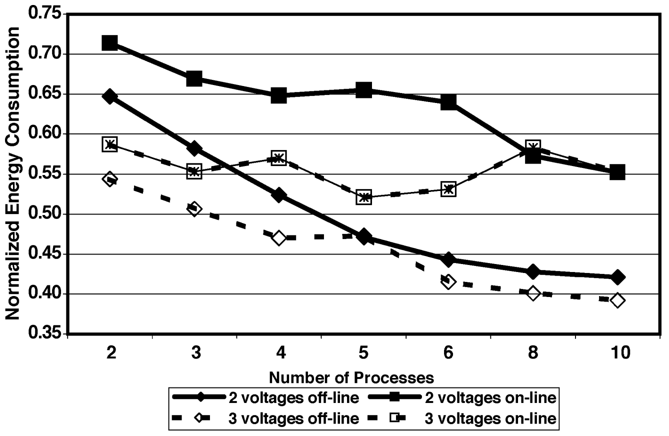

Fig. 4. Normalized energy consumption for 2 and 3 voltages. Normalization is performed with respect to the single voltage case.

In order to analyze the effectiveness of our approach, we started with preliminary testing to determine the appropriate number of voltage levels. We considered three cases, 2-voltages at 3.3 and 1.8 V, 3-voltages at 3.3, 1.8, and 1.0 V, and 4-voltages at 3.3, 2.4, 1.8, and 1.0 V. We used the off-line algorithm to determine the energy savings of the three cases on 1, 2, 3, 5, 6, 8, and 10 processes. All energy consumption values were normalized to the single voltage case at 3.3 V, and we used the memory requirement for the single voltage case as the basis for the other cases. We present the results in Fig. 3. The figure shows the percentage of energy savings versus the number of processes.

The first three questions are addressed using data from the experiments that are displayed in Fig. 4. The figure shows that for different number of processes the normalized energy requirements when the optimal off-line and online algorithms are applied under the same memory requirement. All the values are normalized to the single voltage (3.3 V) case found using the off-line algorithm. Since the off-line algorithm is optimal, we use the off-line values as the lower bound. The figure indicates the results after applying the optimal off-line algorithm and the online algorithm for both the 2-voltage and 3-voltage cases. The percentage by which the off-line and the online algorithm differ in savings compared to the number of processes is presented in Fig. 5.

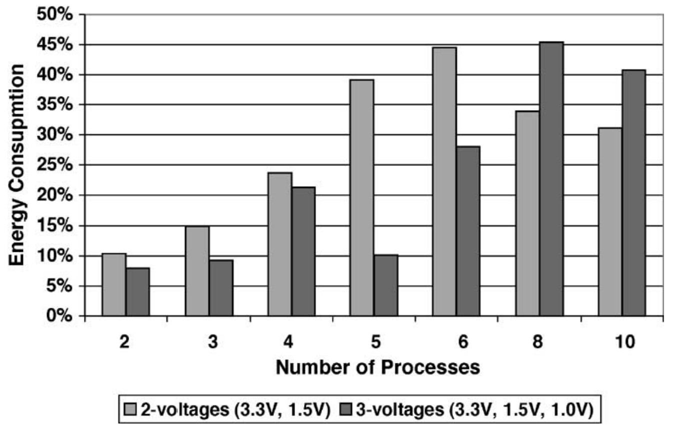

Fig. 5. Increase of energy consumption (in percentages) by the online heuristic over the off-line.

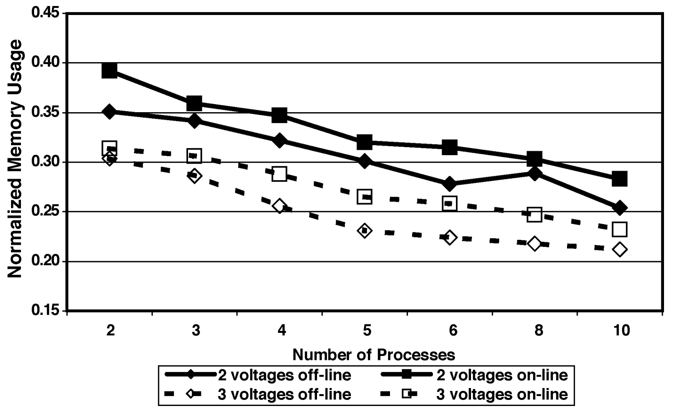

Fig. 6. Normalized memory consumption for 2-voltage and 3 voltages. Normalization is done with respect to the single voltage case.

Fig. 6 presents the results for the dual problem evaluated using Fig. 4. Here we evaluate how much the cost of the system, measured in terms of buffer space, can be reduced under the conditions that energy consumption is fixed. All results are normalized against the base case where storage requirements are first calculated for the set of tasks assuming that a single voltage is used. For the case when we use 2-voltages we compare to 2.5 V, and in the case for 3-voltages we use 1.8 V. Again, we present the normalized results for both the 2-voltage case, and the 3-voltage case in Fig. 6. Additionally, we present the percentage difference between the optimal off-line memory requirement and the online algorithm for both the 2-voltage case and the 3-voltage case in Fig. 7.

Lastly, we consider the relationship between the drop rate of the online and off-line algorithm relative to their performance. These results are presented in Table IV. For both the 2 and 3-voltage level cases, we varied the drop rate from 0.1 to 0.5 and considered and average and median percentage difference between the drop rates of the off-line and online approaches.

The first important question that we analyze is what is the optimal number of voltage levels required to obtain essentially all potential benefits from the use of multiple voltages. In Fig. 3, we analyze the potential benefit for the use of 2,3 and 4-voltage levels with our off-line algorithm. Our results show a energy savings of at least 35% when 2-voltages are used, and at least 45% improvement in the 3-voltage case. While the energy

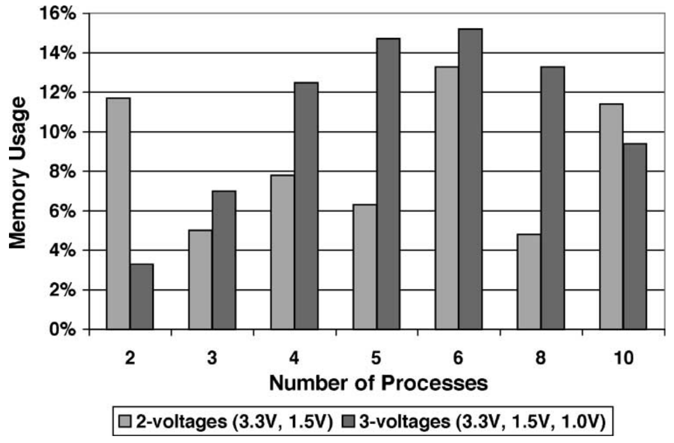

Fig. 7. Increase of memory consumption (in percentages) by the online heuristic over the off-line.

TABLE IV
RELATION BETWEEN DROP RATE AND RELATIVE PERFORMANCE OF THE ONLINE AND OFF-LINE ALGORITHMS

	2-voltages (3.3V, 1.8V)		3-voltages (3.3V, 1.8V, 1.0V)	
Drop Rate	average	median	average	median
0.1	29.8 %	34.5%	31.8%	36.1%
0.2	14.7%	15.6%	14.9%	17.6%
0.25	10.8%	11.2%	11.5%%	12.3%
0.3	9.1%	9.9%	10.1%	11.5%
0.4	4.8%	5.8%	6.2%	7.0%
0.5	2.8%	2.6%	2.8%	2.5%

saving increases with the number of voltage levels used, the benefit of using 4-voltages over 3-voltages is relatively small. Therefore, we see diminishing returns when using more than 3-voltage levels.

The comparison between the optimum off-line and the heuristic online algorithms with respect to storage requirements and energy savings indicates several important conclusions. Evaluation of the online and off-line memory consumption is shown in Fig. 5. On average the online algorithm indicates an overhead of 25%. However, it also saves energy over the single voltage case with 36.5% savings for 2-voltages, and 44.4% savings for 3-voltages.

From Fig. 7, we see that the online algorithm is not able to completely match the performance of the optimal off-line algorithm, the reduction for storage requirements are significantly larger than the energy savings. This is a consequence of the fact that energy consumption is dictated by the overall average effectiveness of online and off-line algorithms, while the storage requirements are primarily a function of how well these algorithms can use high voltages to reduce storage requirements during bursty periods of processes.

VII. CONCLUSION

We have developed an optimal polynomial-time algorithm for power minimization of popular streaming media applications, such as audio, video, and sensor network data under QoS requirements and hardware constraints using multiple voltages. Furthermore, we have developed an online adaptive policy for power minimization in the same scenario. The online approach leverages the insights from the off-line optimal algorithm.

REFERENCES

[1] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A survey of qos architectures," *Multimedia Syst.*, vol. 6, pp. 138–151, 1998.
[2] L. Benini, A. Bogliolo, G. A. Paleologo, and G. De Micheli, "Policy optimization for dynamic power management," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 813–833, June 1999.
[3] T. D. Burd, T. A. Pering, A. J. Stratakos, and R. W. Brodersen, "A dynamic voltage scaled microprocessor system," *IEEE J. Solid-State Circuits*, vol. 35, pp. 1571–1579, Nov. 2000.
[4] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design*, vol. 14, pp. 12–31, Jan. 1995.
[5] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low-power CMOS digital design," *IEEE J. Solid-State Circuits*, vol. 27, pp. 473–484, Apr. 1992.
[6] J. Chang and M. Pedram, "Energy minimization using multiple supply voltages," in *Proc. Int. Symp. Low-Power Electronics Design (ISLPED)*, 1996, pp. 157–162.
[7] R. L. Cruz, "Quality of service guarantees in virtual circuit switched networks," *J. Select. Areas Commun.*, vol. 13, pp. 1048–1056, Aug. 1995.
[8] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable voltage core-based systems," *IEEE Trans. Computer-Aided Design*, vol. 18, pp. 1702–1714, Dec. 1999.
[9] C. Lee *et al.*, "Mediabench: a tool for evaluating and synthesizing multimedia and communications systems," in *Proc. Int. Symp. Microarchitecture*, 1997, pp. 330–335.
[10] J. Luo and N. K. Jha, "Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems," in *Proc. Asia and South Pacific Design Automation Conf.*, 2002, pp. 719–726.
[11] A. Manzak and C. Chakrabarti, "A low power scheduling scheme with resources operating at multiple voltages," in *IEEE Int. Symp. Circuits Systems*, vol. 1, 1999, pp. 354–357.
[12] B. Mochocki, X. Hu, and G. Quan, "A realistic variable voltage scheduling model for real-time applications," in *IEEE Int. Conf. Computer Aided Design*, 2002, pp. 726–731.
[13] K. Nose, M. Hirabayashi, H. Kawaguchi, and L. Seongsoo *et al.*, "V/sub TH/-hopping scheme to reduce subthreshold leakage for low-power processors," *IEEE Custom Integrated Circuits*, pp. 93–96, 2001.
[14] G. Qu, M. Mesarina, and M. Potkonjak, "System synthesis of synchronous multimedia applications," in *Proc. Int. Symp. System Synthesis (ISSS)*, 1999, pp. 128–133.
[15] S. Raje and M. Sarrafzadeh, "Scheduling with multiple voltages," *Integration, The VLSI J.*, vol. 23, no. 1, pp. 37–59, 1997.
[16] R. Rajkumar, C. Lee, J. Lehoczky, and D. Siewiorek, "A resource allocation model for qos management," in *Proc. IEEE Real-Time Systems Symp.*, 1997, pp. 298–307.
[17] ——, "Practical solutions for qos-based resource allocation problems," in *Proc. IEEE Real-Time Systems Symp.*, 1998, pp. 296–306.
[18] V. Sandararajan and K. K. Parhi, "Synthesis of low power cmos vlsi circuits using dual supply voltages," in *Proc. Design Automation Conf.*, 1999, pp. 72–75.
[19] A. Sinha and A. P. Chandrakasan, "Energy efficient real-time scheduling microprocessors," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2001, pp. 458–463.
[20] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *USENIX Operating Syst. Design Implementation*, pp. 13–23, 1994.
[21] J. L. Wong, G. Qu, and M. Potkonjak, "An online approach for power minimization in QoS sensitive systems," in *Proc. Asia South Pacific Design Automation Conf.*, 2003.
[22] ——, "Power Minimization in QoS Sensitive Systems," Univ. California, Los Angeles, Los Angeles, CA, Tech. Rep. 030057, 2003.

Jennifer L. Wong received the B.S. degree in computer science and engineering and the M.S. degree in computer science from the University of California, Los Angeles, in 2000 and 2002, respectively, and is currently working toward the Ph.D. degree at the same university.

Her research interests include intellectual property protection, optimization for embedded systems, and mobility in ad hoc sensor networks.

Gang Qu received the B.S. and M.S. degrees in mathematics from the University of Science and Technology of China, in 1992 and 1994, respectively, and the Ph.D. degree in computer science from the University of California, Los Angeles in 2000.

He joined the Electrical and Computer Engineering Department, University of Maryland, College Park in 2000, and joined the Maryland Institute of Advanced Computer Studies, also at the University of Maryland, in 2001. His research interests include intellectual property reuse and protection, low-power system design, applied cryptography, and computer-aided synthesis.

Miodrag Potkonjak received the Ph.D. degree in electrical engineering and computer science from the University of California, Berkeley in 1991.

In 1991, he joined C&C Research Laboratories, NEC USA, Princeton, NJ. Since 1995, he has been with Computer Science Department, University of California, Los Angeles. His watermarking-based Intellectual Property Protection research formed a basis for the Virtual Socket Initiative Alliance standard. His research interests include system design, embedded systems, computational security, and intellectual property protection.

Dr. Potkonjak received the National Science Foundation CAREER Award, OKAWA Foundation Award, UCLA TRW SEAS Excellence in Teaching Award and a number of best paper awards.