A Rate Matching-based Approach to Dynamic Voltage Scaling

David Biermann, Emin Gün Sirer, Rajit Manohar
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853, U.S.A.

Abstract

We present a simple rate matching-based mechanism for voltage adaptation in a microprocessor running a multiprogrammed workload. The mechanism incorporates a set of architecture and operating system extensions through which applications can communicate their actual and desired progress to the operating system. Using this feedback, the operating system uses a modified scheduling algorithm to run all applications at a single, globally-optimal voltage. We demonstrate that significant energy savings are possible with a simple, practical set of extensions to the architecture and operating system.

1 Introduction

Power conservation is critical in many computational settings. It is well known that improvements in battery capacity have not tracked the increased power requirements in modern processors. Energy efficiency is critical in mobile and ubiquitous computing environments, including sensor networks and hand-held devices, where form factor constrains the total battery capacity. Voltage scaling is a promising mechanism for reducing the power consumption, thereby extending the battery life of such devices. Energy efficiency is also important in high-end processors, where thermal limits constrain the maximum power consumption of a processor. Voltage scaling in this scenario would be useful to prevent the proecessor from exceedings its thermal budget.

There is a wealth of research on voltage scaling algorithms [20, 3, 7, 8, 16, 23, 27]. This work has mostly focused on operating system (OS) techniques for selecting a globally optimal voltage setting. The primary driving factor in this class of selection algorithms has been the total system idle time. The operating system typically scales the voltage (and frequency) down in response to idle periods and increases it during bursts of activity to try and find the lowest possible voltage setting that eliminates idleness. Such schemes are compelling because they only require minor changes to the operating system scheduler and no application-level modifications. However, heuristic-based, operating system driven algorithms tend not to exhibit stable behavior, nor do they robustly converge to a single optimal operating point. This has led to a recent experimental study that thoroughly evaluated many previous voltage scaling schemes to conclude that "**No heuristic policy that we examined achieved [the optimal voltage and frequency]**."[1] [8] Part of the reason why these heuristic approaches are limited is because they are driven solely by system idle time and have no application-specific information.

Other work has examined how to select the optimal voltage given complete information about application start times, deadlines, and computation needs [12, 20]. Given complete application knowledge, these omniscient schemes can optimally pick the operating voltage to minimize energy requirements while meeting application deadlines. However, while such schemes can provide lower bounds on energy requirements, they are hard to use in practice because they require complete application information. Due to data-dependent execution and hardware effects such as cache misses, estimating future execution time for an application is a daunting task.

We contend that the problem with these two extremes is the lack of application-specific information. OS-directed schemes do not take any application-specific deadline information into account, while omniscient schemes assume an impractical level of application knowledge. The problem stems from the lack of an interface by which application writers can inform the hardware of relevant information for making energy-optimal decisions.

In this paper, we propose a new interface through which applications can independently express their power needs to dynamically select the globally optimal operating voltage. An interface for power-aware voltage adaptation should exhibit the following properties:

- **Simplicity.** It should be practical and intuitive to use. In particular, their use should not be predicated on detailed knowledge of future application behavior.
- **Efficiency.** It should provide sufficient information for the hardware to make optimal or near-optimal voltage scheduling decisions with minimal run-time overhead.

[1] Emphasis in the original.

- **Protection.** It should allow the operating system to make per-process voltage scheduling decisions. Applications should not be able to override energy limits imposed on them by the operating system.

- **Flexibility.** It should enable applications to implement any voltage selection algorithm. Variations in application execution bursts necessitate differing voltage adaptation schemes.

- **Compatibility.** It should not preclude legacy applications from being executed without modification.

We propose a set of simple extensions that achieve these goals via fine-grained *rate-matching*. Our approach relies on extracting explicit progress information from the application. The hardware can then use this information to tune its voltage and frequency to match application needs. Applications provide their target execution rate on initialization, and include progress indicators in critical locations in the code. The progress indicators specify the actual application execution rate, which is then adjusted to match the target rate by scaling the operating voltage.

This paper makes the following contributions. First, it introduces a simple, efficient, and flexible interface for application-directed voltage control. The interface is easy and intuitive to use; it typically requires the addition of a small amount of code to initialize the system, and a single call in the main application loop. Second, we show that this interface when used independently by competing applications leads to a globally optimal dynamic voltage level. Our interface provides a way for each application to pursue the best voltage for its own application-specific goals. In aggregate, the global system converges to the optimal voltage without the operating system having to explicitly derive it from application characteristics or needs. Finally, we show that this interface permits an efficient implementation that achieves significant power savings. For a demanding multiprogrammed workload, our implementation in the Linux operating system is within **3.4**% of optimal and achieves **42**% reduction in energy on average.

The rest of the paper is organized as follows. Section 2 relates our contributions to previous work. Section 3 describes our API for voltage management, in the context of a single application. Section 4 shows how the API can be used to achieve globally optimal voltage selection in a multiprogrammed environment. Section 5 describes our Linux implementation, and shows that achieves performance that is close to the optimal.

2 Related Work

For any dynamic voltage adaptation scheme to be feasible, the hardware must support operation at multiple voltage levels. Current commercial processors support a small number of discrete voltage/frequency adjustment options. Intel's Mobile Pentium III with SpeedStep has two levels of operation [10] and AMD's Mobile Athlon 4 with Power NOW! [1] has five levels. This small range of voltage adjustment can only support very course grain voltage adjustments, such as lowering the voltage when a system switches to battery power. Transmeta's Crusoe processor is one of the few commercial processors to support fine-grain voltage/frequency adjustments. However, unlike our voltage-adaptive scheme the Crusoe's voltage/frequency adjustments are not directly driven by the application. Crusoe's power management software monitors power consumption by sampling CPU sleep states and using heuristics to adjust the voltage and frequency of the processor [6]. Some low-power embedded processors, such as the Intel 80200 Processor [11], also support fine-grain voltage/frequency adjustments. The existence of these capabilities in modern processors has spurred researchers to examine algorithms for voltage/frequency management.

Previous work on voltage adaptation has focused on operating system techniques that choose an operating voltage which minimizes idle time. Early work by Weiser et al. showed the potential benefits of voltage scaling [27]. They looked at two schemes, `FUTURE` and `PAST`, that examine idle time in scheduling windows to determine the voltage setting for the next epoch and compared these to `OPT`, the optimal strategy. This work was further extended by Govil et al. which examined many other candidate strategies for voltage scheduling using the same framework as Weiser [7]. They showed that when evaluating a range of applications with a single scheduling policy, simple strategies achieved energy savings that were comparable to those obtained by more sophisticated strategies. Martin examined the effect of non-ideal battery behavior and memory performance, and uses this to formulate a more sophisticated model of the effect of voltage/frequency scaling on system lifetime [18]. Grunwald et al. experimentally evaluate different voltage scaling policies on Itsy, a prototype hand-held computer [8]. They conclude that none of the policies proposed to date work well in the general case. These papers share a set of common characteristics: (i) They investigate single system heuristics that have to operate well across a wide range of applications; (ii) They are all coarse grained and interval based. The system re-evaluates the voltage setting only when the scheduler is invoked. The choice of the interval is determined by the scheduler, independent

of application needs; (iii) They are driven entirely by system idle time, which is not directly related to application needs. Inferring computation needs from idle time measurements is complicated by phenomena like deceptive idle times [13], where applications might remain idle due to outstanding I/O requests. These three characteristics have limited the efficiency of such schemes. In contrast, our rate-matching based approach enables application-controlled voltage adaptation, facilitates scheduling at finer granularity, and is directly driven by application progress.

There has been much work (both theoretical and simulation based) on optimal voltage scheduling policies based on complete information about application deadlines, arrival times, and computation workload. Hong et al. look at the voltage scheduling problem given full information about periodic tasks [9]. Pering et al. describe the design of a low-power microprocessor system that incorporates dynamic voltage scaling [20]. They build on a real-time OS infrastructure, and assume that application deadlines and computation needs are available to their scheduler. Ishihara et al. make the same assumptions and approach the optimal voltage scheduling problem through linear programming [12]. Manzak et al. provide techniques to compute the optimal task voltages for a number of tasks that have a common, global deadline [16]. More recent work makes similar assumptions when implementing their real-time dynamic voltage scaling algorithms [19, 28]. This body of work relies on explicit application deadline information, *and* total application execution time for at least one reference voltage. In practice, these two metrics, especially the latter, may be difficult to obtain. Furthermore, having the application compute an accurate estimate of future workload is likely to incur an unacceptable performance penalty. In contrast, our rate-matching based approach does not require that the application make any estimates of its future behavior.

Simunic et al. present techniques that use a change-point detection algorithm to detect the difference in arrival and service rates for an MPEG player and MP3 player [23]. They assume the presence of a power manager that can monitor these rates and the number of frames decoded by the two players. Their technique also uses an off-line calculation to determine thresholds for the change-point detection algorithm. Our rate-matching based, approach while similar in spirit, relies only on run-time information and does not require an off-line calculation phase. Further, it is not limited to applications with input and output queues that can be monitored by dedicated hardware.

3 Voltage Scaling API

Our API achieves application-driven adaptivity by rate-matching. We call this API RMVS, for rate-matching voltage scaling. We begin with the minimal set of operations that provide a mechanism by which the application can inform the hardware about its progress. The hardware can then pick the optimal voltage level to reduce energy consumption given a target progress guarantee.

3.1 API Description

The voltage control system is centered about a counter, `VCNT`, which captures the application's progress. This counter is periodically incremented by the application via the `PROGRESS` operation, and decremented by the system at a rate specified by the `VRATE` register. Equilibrium is achieved when the increment and decrement rates balance and keep the counter at a near-constant value.

If the program runs too slowly, then the application-controlled increments will occur less often than the system-controlled decrements and the counter will eventually underflow. Likewise, if the application runs too quickly, then the counter will eventually overflow. These conditions are signals to adjust the voltage up or down respectively.

Such conditions are reflected to the application through exceptions. Throughout this paper, we will refer to these exceptions as *counter exceptions*. They are handled by an exception handler that picks the new operating voltage for the application. The exception handler does so by writing the `VLT` register, which in turn updates the actual supply voltage level.

For protection reasons, the exception handler should not be able to adjust `VLT` to an arbitrary value. Otherwise, malicious applications could consume more power than their allotment and circumvent OS resource management decisions. The OS has the ability to set hard bounds on `VLT` via the registers `VMIN` and `VMAX`. These bounds could be the physical limits at which the processor can operate, or the OS may wish to restrict the voltage level further. For example, to extend battery life when a laptop is not plugged in to a wall socket, the OS may set `VMAX` to something far lower than it would if there is more power available.

If the counter hits zero while an application is running and `VLT` is already equal to `VMAX`, then the application cannot meet its current performance goal. Depending on the nature of the application, it may then want to exit, continue running at the highest allowable voltage level, or modify its performance requirement, possibly

Reg	Description
CMAX	Maximum counter level
VINCS	Increments since last voltage exception
VINS	Instructions since last voltage exception
VLT	Supply voltage level
VMAX	Maximum allowable supply voltage
VMIN	Minimum allowable supply voltage
VRATE	Counter decrement rate (Hz)
VCNT	Counter

Table 1: New registers introduced by the voltage-adaptive ISA extensions.

Instruction	Description
PROGRESS	Increments counter by one
HALT	Stops processor until interrupt

Table 2: New instructions introduced by the voltage-adaptive ISA extensions.

performing a quality-of-service adjustment. For example, an scalable video-decoder that cannot meet its frame rate goal, even at `VMAX`, may choose to use fewer colors, lower resolution, or a slower frame rate. Likewise, if a program is running too quickly while already at `VMIN`, it may want to do nothing and continue running at this level, or change its quality-of-service by switching to *more* colors or higher resolution.

The amount of hysteresis in the system can be controlled by limiting the range of the `VCNT` register. To accomplish this, we provide a new register `CMAX` that bounds the maximum possible counter value. Note that a `CMIN` register is unnecessary, as the amount of hysteresis only depends on the range of values the counter can take, not the absolute value of the counter itself. With this modification, exceptions are reported when the counter hits zero or `CMAX`. To keep the hysteresis symmetric, `VCNT` is typically initialized to `CMAX`/2.

To more quickly arrive at the equilibrium voltage level (i.e. the level at which the rate of increments equals the rate of decrements), a counter exception handler needs to know the number of instructions and the total number of counter increments since the last exception. These values are stored in `VINS` and `VINCS`, respectively. After the exception handler has changed the voltage, it typically resets the counter to `CMAX`/2.

By default, an application begins with `VRATE` set to zero. This has the effect of turning off the voltage control mechanism if the application does not contain any `PROGRESS` instructions. This allows legacy applications to execute with no modification.

Finally, there are also times when the processor is truly idle and just needs to wait until it receives an interrupt (from a timer, for instance). In this case, the `HALT` instruction causes the processor to wait until an external

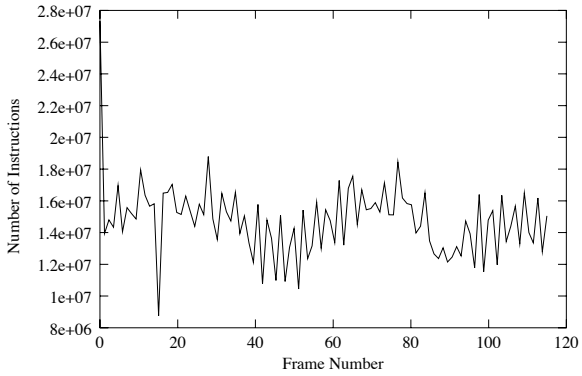

Figure 1: Number of instructions required to decode frames in an MPEG video sequence.

interrupt occurs. The `HALT` instruction has already been adopted and implemented in many modern ISAs, and is simply included here for completeness.

3.2 Example Application

In this section, we illustrate how to integrate our extensions into real applications. We focus on a video decoder with a fixed, average-case throughput goal. An MPEG video decoder with its periodic structure provides a good example of how our extensions can be used to control the average-case behavior of an application. Modifying applications involves two steps: inserting `PROGRESS` instructions appropriately into a program, and determining application-specific values for `VRATE` and `CMAX`.

Even with a regular benchmark like MPEG, it is difficult to specify its future computation requirements based on its immediate inputs. Each frame may have drastically different computational requirements for decoding, with some using as many as three times as many dynamic instructions as others [2]. A graph of the number of instructions needed to decode frames for a 116-frame video clip is shown in Figure 1. This figure shows that even a regular benchmark like MPEG exhibits irregular and time-varying computational needs that are hard to capture.

To provide an indication of progress to the hardware, the application writer needs to place `PROGRESS` operations at appropriate locations. A natural choice for MPEG is to place one operation at the end of the code that decodes a single frame. We found this to be a straightforward modification for the benchmarks we examined in this paper. Generally, the `PROGRESS` operations were placed in the main dispatch loop of event-driven applications.

The desired rate of progress is determined by the value in `VRATE`. For MPEG, this corresponds precisely with the designed frame rate. Therefore, `VRATE` is set to 30Hz for a 30 frames/sec (fps) target.

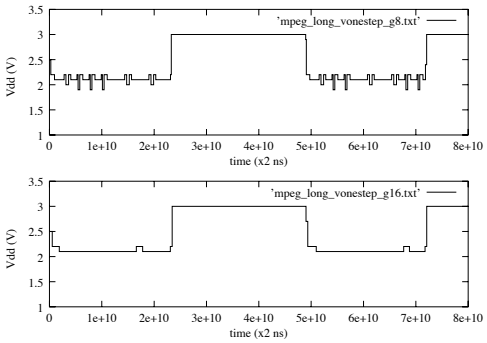

Figure 2: Voltage level for an MPEG video decoder with (top) `CMAX` $= 8$ and (bottom) `CMAX` $= 16$.

`CMAX` determines how quickly the rate of `PROGRESS` operations approaches the equilibrium `VRATE`. `CMAX` can be thought of as the size of the window over which the `PROGRESS`-rate is averaged, and hence the sensitivity of the mechanism. Note that if `CMAX` $= 2$, then any counter increment or decrement will immediately cause an exception. On the other hand, if `CMAX` is large (say 2000), then *at least* a thousand decrements (≥ 33 seconds) must occur before a counter exception could allow the system to adjust its throughput. For the MPEG video decoder, the desired sensitivity depends on the amount of buffering available. In general, we would like to make the mechanism as insensitive as possible within the constraints of the application, since changing voltage levels adds to the energy and time overhead cost. If the number of buffered frames is b, then we should ensure that `CMAX`$/2 < b$. Otherwise, the buffer may run out of frames before an exception occurs! In practice, one might further restrict this bound to avoid having the buffer becoming nearly depleted.

A small value of `CMAX` makes the voltage scaling scheme very sensitive to variations in the arrivals of `PROGRESS` operations. Figure 2 shows the variation in voltage while looping a video clip between a computationally intensive sequence and very still sequence, requiring about half as much computation per frame. The system with larger `CMAX` experiences far less voltage adjustment without significant increase in the convergence time, as compared to the system with the smaller value of `CMAX`.

This example demonstrate the flexibility and versatility of our API. By selecting appropriate settings for the registers that control voltage adaptation, a variety of adaptation schemes can be selected to suit application needs.

3.3 Implementation Considerations

The API described above can be implemented using instruction set architecture (ISA) extensions, or by using a system call interface with little/no architectural support.

Extending the ISA. The overall impact of our ISA extensions on the main processor execution path is modest, especially when integrating such an ISA extension with a MIPS-like ISA that supports coprocessors [14]. The ISA extension along with its accompanying registers is implemented as a voltage management coprocessor (VU), and the primary decode need only classify the proposed extensions as VU instructions to be routed to the voltage management coprocessor. The only interaction between the primary processor datapath and the VU is through move instructions that transfer data to and from VU registers. Protection can be achieved by only allowing user-mode code access the progress operation and the supply voltage level register. We implemented this modification to the RTL-level description of an asynchronous MIPS processor, and it had no noticeable impact on the performance of the processor.

Voltage Calculation. When the counter value exceeds `CMAX` or drops below 0, application-level code must be executed to adjust the operating voltage. The simplest strategy we adopt is called `INCDEC`, where the voltage is decreased by a fixed amount if the counter crosses the low threshold, and increased by a fixed amount if the counter exceeds `CMAX`. In general, an application could adopt more sophisticated strategies when determining its new voltage in response to a counter exception.

Voltage Adaptation. The granularity of our voltage-adaptive scheme is determined by the granularity of the adjustable voltage regulator. The regulator can be designed in a number of ways. One simple solution is to use off-chip voltage regulators that are readily available for high-performance microprocessors. The maximum voltage switching latency for typical off-chip regulators is approximately $250\mu s$ [22] and we will assume this regulator characteristic in the simulations presented in the latter sections of this paper.[2] Other regulator options include custom off-chip regulators or integrated on-chip regulators which will have smaller voltage latencies on the order of $70\mu s$ [3, 15].

If the processor implementation uses asynchronous circuits, then scaling the voltage automatically scales the performance of the system [17] and no further support is necessary. If the processor uses clocked circuits, then the underlying hardware must scale the operating frequency of the processor along with its voltage.

Counter Updates. The reference clock provides a periodic timing signal to the processor. This is required to produce an absolute time reference so that the voltage-adaptive logic can keep track of how fast the processor

[2]This latency strongly depends both on the capacitance between Vdd and GND, and the direction and amount of the voltage change.

is running, regardless of its current operating voltage. In our implementation, the reference clock controls how often the voltage counter is decremented. The `VRATE` register controls the rate the counter is decremented by dividing down the reference clock frequency. The reference clock may either be integrated on-chip with the rest of the processor (using an additional fixed supply voltage) or can be an external oscillator.

4 From Local to Global Voltage Selection

The preceding section described how applications express their energy needs in isolation. In this section, we show how this can be combined with scheduling decisions in the operating system to achieve globally optimal voltage selection.

4.1 Optimal Voltage for a Single Application

Our model for the effect of voltage scaling on the performance and energy requirements of a design uses the conventional first-order approximation for energy and delay. Let k be the number of units of work that a processor operating at voltage V can complete in t units of time using E units of energy. We assume that these quantities are related by the following equations:

$$k = \frac{Vt}{V_0} \tag{1}$$

$$E = CV^2 \times k = \frac{CV^3 t}{V_0} \tag{2}$$

where the quantities V_0 and C are constants that depend on the design.[3] For the purpose of analysis, a task corresponds to a given amount of work that must be completed by a certain time. Give a set of such tasks, we can determine the voltage schedule that minimizes the total energy required by the system. Some properties this schedule obeys are:

Property 1. The optimal voltage schedule is piecewise constant, with changes occuring when tasks are completed. The proof of this follows from (1) and (2). This can also be established for more complex functional forms by posing the following variational problem: Given a fixed amount of work to be completed in a fixed amount of time, what is the optimal $V(t)$ that minimizes the total energy? Since both k and E are functions of only $V(t)$ and other constants, the optimal solution is $V(t)$ is a constant.

Property 2. In an optimal voltage schedule, the voltage can increase only when new tasks enter the system; otherwise, the voltage is a non-increasing function with changes in voltage occurring only when tasks leave the system. This can be proved in the same way as Property 1.

These two properties provide us with some intuition regarding the behavior of the optimal voltage curve.

4.2 Globally Optimal Voltage Selection

Consider a simple scheduler with two tasks A and B. For simplicity, we assume that task A has k_A units of work that need to be completed in t time slices, and B has k_B units of work that need to be completed in the same number of time slices t.

A priority scheduler schedules task A with probability p and B with probability $(1-p)$. Therefore, pt slices are allocated to task A, and $(1-p)t$ slices are allocated to task B. In this situation, we would like to use the following voltages for A and B:

$$\begin{aligned} V_A &= \frac{V_0 k_A}{pt} \\ V_B &= \frac{V_0 k_B}{(1-p)t} \end{aligned}$$

where V_0 is the voltage where a process can complete one unit of work in one time slice.

Therefore, the energy used after t slices (normalized by the average capacitance per unit of work) is given by:

$$\begin{aligned} E_A &= k_A V_A^2 = V_0^2 \frac{k_A^3}{p^2 t^2} \\ E_B &= k_B V_B^2 = V_0^2 \frac{k_B^3}{(1-p)^2 t^2} \\ E &= \frac{V_0^2}{t^2}\left(\frac{k_A^3}{p^2} + \frac{k_B^3}{(1-p)^2}\right) \end{aligned}$$

We can complete this joint task using the minimum amount of energy when:

$$\begin{aligned} \frac{\partial E}{\partial p} &= 0 \\ \text{i.e.,}\quad p &= \frac{k_A}{k_A + k_B} \\ \text{i.e.,}\quad V_A &= V_B = V_0 \frac{k_A + k_B}{t} \end{aligned}$$

[3] This first order model neglects threshold voltage effects, leakage, and short-circuit current terms in the equation for energy and delay.

In other words, the optimal scheduling strategy is to treat the combined job "$A + B$" as one job running at a single voltage that is chosen precisely so that both tasks can meet the joint deadline. This result easily generalizes to the case of n processes, where the optimal scheduling policy once again equalizes all the voltages, and the scheduling probability is proportional to the amount of work to be performed.

In general, if we have n tasks with periodic deadlines that have slice allocations s_1, s_2, ..., s_n operating at steady state voltages of V_1, ..., V_n, each task completing work given by k_1, ..., k_n units per deadline. We know that

$$V_i = \frac{V_0 k_i}{s_i}$$

where V_0 is the voltage where a task can complete one unit of work per slice. As the operating system scheduler, we can observe the values V_i and s_i. Each task should receive a priority that is proportional to its work k_i. Therefore, the operating system scheduler can compute the new slice values

$$\begin{aligned} s_i' &= \frac{k_i}{\sum_{j=1}^{N} k_j} = \frac{V_i s_i / V_0}{\sum_{j=1}^{N} V_j s_j / V_0} \\ \text{i.e.,} \quad s_i' &= \frac{V_i s_i}{\sum_{j=1}^{N} V_j s_j} \end{aligned} \tag{3}$$

Therefore, to minimize energy, the operating system should dynamically adjust its scheduling policy using equation (3). Note that the quantities V_i and s_i are both known to the operating system: V_i is maintained in our ISA as a register in the voltage unit; s_i is the proportion of time that is allocated to the process by the scheduler.

We modified the Linux scheduler according to equation (3) to use per-process voltage information to adjust scheduling priorities of the processes that participated in dynamic voltage scheduling. Once the scheduler priorities have been correctly adjusted, each application with a non-zero `VRATE` will automatically pick its own local voltage to be the same as the globally optimal voltage choice.

Note that as opposed to previous work on idle time minimization, the voltage level is our system is determined by a combination of application information and operating system scheduling. While previous work has examined the problem of finding a single global voltage that satisfies all applications, we enable each application to determine its own voltage level that is appropriate for its own progress metric. The operating system uses the voltage information per application to change scheduling priorities, and this combination results in a globally optimal choice of voltage.

The voltage scheduling that results is quite different from what one would observe with a Weiser-style idle time scheduler. For instance, consider a single, cpu-bound application. An idle time scheduler would always run this application at a high voltage, because each scheduling interval has no idle time. The insertion of `PROGRESS` instructions in the application gives the hardware additional information about the actual needs of the application which may not always be reflected in the idle time. This enables us to save energy without loss in application performance.

An interesting result of this scheme is that the overall average voltage-level of the processor will scale with the load on the system. If there are m copies of the previous video application using the CPU and if the processor's throughput scales linearly with the voltage level, the overall system voltage will also increase linearly with m.

5 Results

Due to the absence of hardware that provides an implementation of our API, we evaluated its effectiveness with a simulation-based study. The simulator we use is based on Bochs, an open-source, x86 simulator that includes models for the network, disk, and other devices and can boot the Linux operating system. We adopted a full system approach to simulation because our proposed extensions include both operating system and architecture modifications.

The core simulator was modified to support the API as an ISA extension. The effect of voltage scaling only impacts the energy and performance of the processor, and Bochs was modified to correctly account for a selective slowdown/speedup of the processor. We augmented Bochs to also record the energy consumption of the processor.

Name	Description
austin	clip from Austin Powers
bach	clip from a Bach composition
godfather	clip from The Godfather
hal2001	clip from 2001
jesse	Jesse by Joshua Kadison
kennedy	clip of John F. Kennedy speaking
lastresort	clip from Last Resort by Papa Roach
mozart	Ein Kleine Nacht Musik by Mozart
pachebel	Canon in D by Pachebel
rebecca	Rebecca by Pat McGee Band
fear	clip of Roosevelt's nothing to fear speech
infamy	clip of Roosevelt's Pearl Harbor speech

Table 3: Audio files used as input for benchmarks.

The simulator supports voltage levels ranging from 1.5V to 0.3V in 0.1V steps. Our simulator also takes the non-linear dependence between voltage and throughput into account, as well as accounting for those times that do not scale with voltage (cache misses, disk access, etc). The simulator also calculates the optimal energy that an application could operate at if it had perfect knowledge of future behavior based on the arrival times of tasks using the analysis from Section 4. All the reported results correspond to applications that run on our modified Linux being simulated on a modified version of Bochs.

We modified a Linux 2.4 kernel to include all the necessary operating system support for our API. The thread structure was modified to include all the values for the registers introduced by the API. In addition, the context switch routine was modified to save and restore the hardware registers that correspond to our API. Processes that operate at constant voltages have `VRATE` set to zero. The voltage state is shared by child processes so that the time taken by sub-tasks spawned by a parent is correctly accounted for.

Benchmarks. We used a set of six benchmark programs (shown in Table 4), attempting to use a range of different application types to illustrate the applicability of the proposed API. `toast` and `untoast` are audio codecs, and `mpg123` is an MP3 player. `go` is a game-tree search, and we modified it to limit the search it performs by regulating the amount of work per move. This adaptively prunes the search tree based on computational requirements. `ehgml` is a ray-tracer that we modified so as it would render scenes at a fixed frame rate. Finally `abyss` is a web-server that was modified to respond to web traffic at a fixed rate. The multimedia benchmarks used a variety of input data sets that are described in Table 3. Modifying each benchmark was a simple task, and it took us less than an hour per benchmark to perform the necessary modifications. Unlike other related work where the quality of service was varied to meet performance requirements [20], our benchmarks provide a fixed quality of service—i.e. each run of a benchmark corresponds to the same amount of work.

5.1 Calibration

We calibrated the time and energy reported by Bochs against the time and energy we measured from a 400 MHz Pentium II-based system with 128MB memory, 4GB disk, and the Intel 440BX chipset. The time taken by each application was measured using the Unix `time` command on Bochs as well as on the real machine. The runtimes of the applications ranged from 4.43 secs (low) to 24 secs (high). For calibration purposes, we used four of the audio data sets (jesse, mozart, pachebel, rebecca). For energy calibration, we charged a different energy cost for each instruction type (integer, floating-point, memory). The results of the energy reported by the simulator were compared against measurements from the Pentium II system. We measured the current being used by the processor by attaching a probe to the voltage regulator on the motherboard.

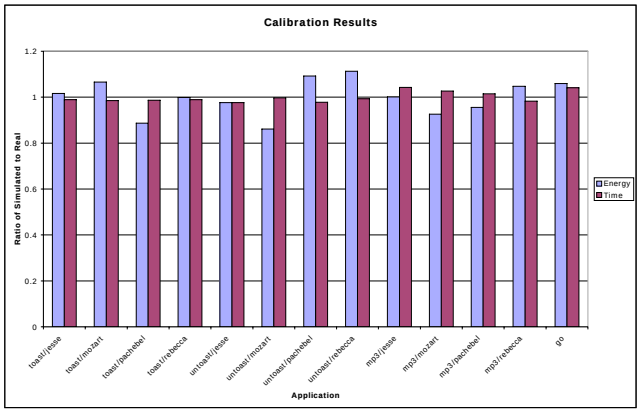

Figure 3: Results of calibration against measured data.

Figure 3 shows the results of our calibration runs. The y-axis shows the ratio of the metric reported by the simulator to the measured metric. Both energy and time calibration is reported per benchmark. The largest error in energy measurements we observed was an underestimate by 13.9%, and the largest error we observed in timing measurements was an overestimate by 4.2%. The average of the absolute values of the error percentages was 6.1% for the energy and 1.9% for the time.

5.2 Application Performance

Figure 4 shows the results of RMVS on the six benchmarks from Table 4. For benchmarks with multiple input sets, we took the sum of the energy per input set which corresponds to a workload that executes each clip from Table 3 once. For each benchmark, we normalize the reported energy against the energy required by the application when no voltage scaling is performed (i.e. the normal energy requirement for each application would be 1.0 in Figure 4). Each application has two bars: one cor-

Benchmark	Description
`toast`	GSM encoder
`untoast`	GSM decoder
`mpg123`	mp3 player
`go`	simulation of the game of go
`ehgml`	ray tracer
`abyss`	web server

Table 4: List of benchmarks

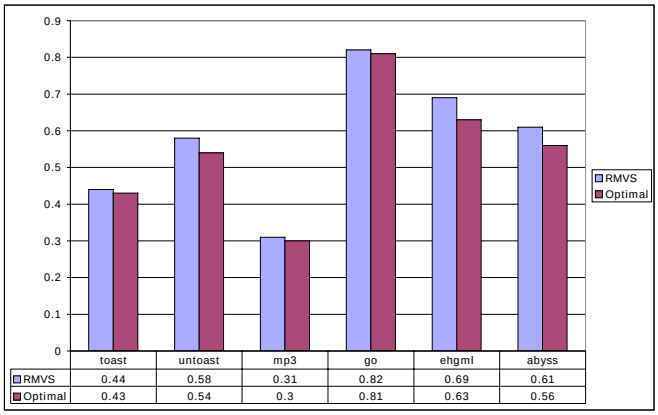

Figure 4: RMVS voltage adaptation with single application.

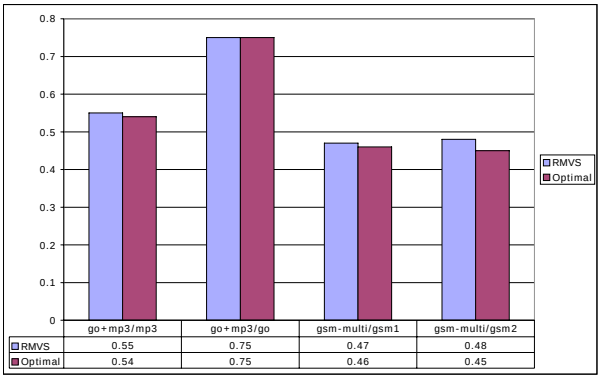

Figure 5: RMVS with two applications running simultaneously.

responding to using RMVS, and the other corresponding to optimal voltage scheduling according to Section 4.

RMVS uses 10% more energy than the optimal voltage scaling strategy in the worst case (abyss), and 5.3% more energy on average. Compared to not applying any form of voltage scaling, RMVS saves 43% of the total energy required on average.

5.3 Multiprogrammed Workloads

We now examine the effects of running multiple applications simultaneously. In each case, we keep track of the per-application energy as well as the optimal per-application energy. We report results from RMVS runs for three different multiprogrammed workloads. Workload `go+mp3` corresponds to running the `go` benchmark and `mpg123` benchmark simultaneously. Workload `gsm-multi` corresponds to two runs of `untoast`, the GSM decoder. Finally `3app` corresponds to `go`, `mpg123`, and `untoast` running simultaneously.

A summary of the results for the two application workloads is provided in Figure 5. Results from the three

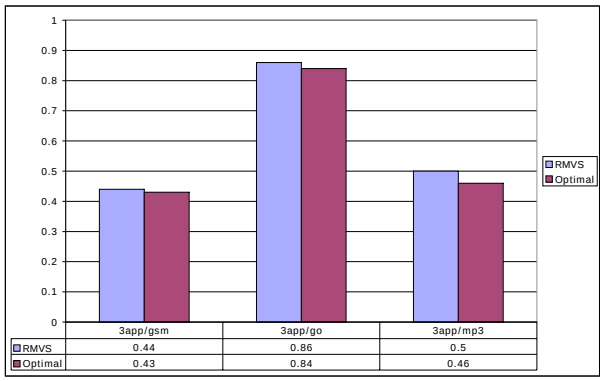

Figure 6: RMVS with three applications running simultaneously.

application workload is shown in Figure 6. For each workload, the per application energy consumption using RMVS is compared against the energy used by the application under the optimal voltage schedule for the workload. Our implementation of RMVS performs to within 3.4% of the optimal, and reduces the energy requirements of the workload by 42% on average.

Figure 7 shows the voltage as a function of time for `mpg123` with the "Bach" dataset. The voltage curve shows the effect of using an incremental adjustment in the voltage. For this particular run, the optimal voltage lies between 1.0V and $1.1V$. The discrete nature of the voltage adaptation causes the voltage to periodically increase by 0.1V before stabilizing at 1.0V for a further interval.

Figure 8 shows the voltage as a function of time when both go and mpg123 are executing. The mp3 player is playing a clip that ends at 12 seconds. The combination of the two applications causes the processor to operate at 1.5V until the mp3 player completes. Notice how both applications independently chose the same voltage to operate at due to the modified scheduler. Once the mp3 player completes, go is allowed to use a larger fraction of the processor—immediately lowering its operating voltage and proceeding along the same phases as before.

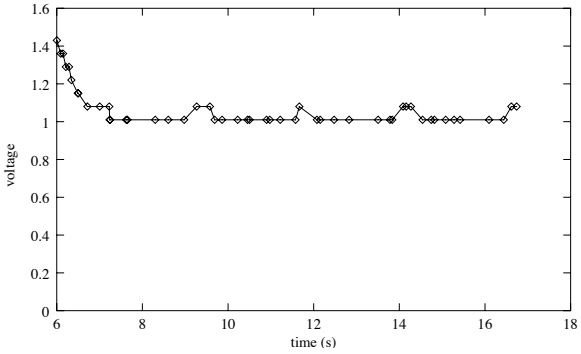

Figure 7: mpg123/Bach

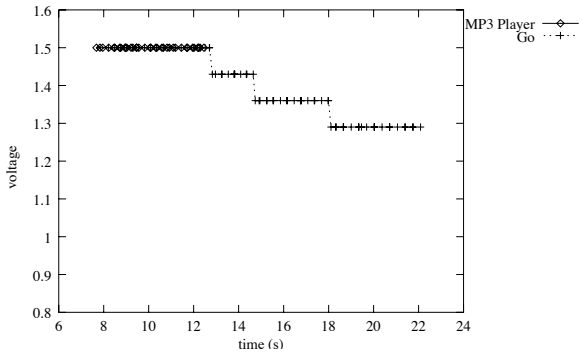

Figure 8: go and mpg123

6 Summary

This paper presented a simple API for rate matching-based dynamic voltage scaling. The API provides a mechanism for applications to choose their own voltages, and competing applications automatically adjust their voltages to the same, globally optimal value due to interactions with our proposed operating system scheduler. We evaluated our strategy on a calibrated, full system simulator using a number of applications running under both single application and multiprogrammed workloads. Our results show that RMVS achieves an single application energy reductions that are within 5.3% of optimal on average, and multiprocessor reductions that are within 3.4% of the optimal.

References

[1] AMD, Mobile AMD Athlon 4 Processor Model 6 CPGA Data Sheet, Sep 2001.

[2] A. C. Bavier, A. B. Montz, L. L. Peterson. Predicting MPEG Execution Times. *Proceedings of SIGMETRICS*, pp. 131-140, 1998.

[3] T. D. Burd, T. A. Pering, A. J. Stratakos and R. Brodersen. Dynamic Voltage Scaled Microprocessor System. *IEEE J. Solid-State Circuits*, vol. 35, pp. 1571-1580, Nov. 2000.

[4] T. D. Burd and R. Brodersen. Design issues for dynamic voltage scaling. *Proc. 2000 international symposium on Low Power Electronics and Design*, pp. 9–14, 2000.

[5] J. Douceur and W. Bolosky. Progress-based Regulation of Lowimportance Processes. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*, pages 247–258, December 1999.

[6] M. Fleischmann. LongRun Power Management - Dynamic Power Management for Crusoe Processors. Transmeta Corporation, 2001.

[7] K. Govil, E. Chan, and H. Wassermann. Comparing algorithms for dynamic speed-setting of a low-power cpu. *Proceedings of the 1st Conference on Mobile Computing and Networking*, Mar 1995.

[8] Dirk Grunwand, Phillips Levis, Keith Farkas, Charles B. Morrey III, and Michael Neufeld. Policies for Dynamic Clock Scheduling. *Proc. Operating Systems Design and Implementation*, 2000.

[9] I. Hong, M. Potkonjak, and M. Srivastava. On-line scheduling of hard real-time tasks. *Proc. International Conference on Computer-Aided Design*, November 1998.

[10] Intel. Intel SpeedStep Technology, Jan 2000.

[11] Intel. Intel 80200 Processor based on Intel XScale Microarchitecture, Nov 2000.

[12] T. Ishihara and H. Yasuura. Voltage Scheduling Problem for Dynamically Variable Voltage Processors. *Proc. of International Symposium on Low Power Electronics and Design*, August 1998.

[13] Sitaram Iyer and Peter Druschel. Anticipatory scheduling: A disk scheduling framework to overcome deceptive idleness in synchronous I/O. *Proc. 18th ACM Symposium on Operating Systems Principles*, 2001.

[14] G. Kane and J. Heinrich. MIPS RISC Architecture. Prentice-Hall, 1992.

[15] Kuroda et al. Variable supply-voltage scheme for low-power high-speed CMOS. *IEEE J. Solid-State Circuits*, vol. 33, pp 454-462, March 1998.

[16] A. Manzak and C. Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy. *International Symposium on Low Power Electronics and Design*, 2001.

[17] A. J. Martin, A. Lines, R. Manohar, M. Nyström, P. Penzes, R. Southworth, U. V. Cummings, and T.K. Lee. The Design of an Asynchronous MIPS R3000. *Proceedings of the 17th Conference on Advanced Research in VLSI*, September 1997.

[18] Thomas Martin. *Balancing Batteries, Power and Performance: System Issues in CPU Speed-Setting for Mobile Computing*. Ph.D. thesis, Carnegie Mellon University, 1999.

[19] P. Pillai and K.G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. *SOSP 2001*

[20] T. Pering and R. Brodersen. Energy Efficient Voltage Scheduling for Real-Time Operating Systems. *4th IEEE Real-Time Technology and Applications Symposium*, 1998, Works In Progress Session.

[21] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. *Proceedings of the 7th Int. Conference on Mobile Computing and Networking*, July 2001.

[22] Semtech. Power Supply Controller for Portable Pentium II & III SpeedStep Processors, Aug 2000.

[23] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. De Micheli. Dynamic voltage scaling and power management for portable systems. *Proceedings of the 38th Design Automation Conference*, 2001.

[24] Marshall McKusick, Keith Bostic, Michael Karels, John Quarterman. The Design and Implementation of the 4.4 BSD Operating System. Addison-Wesley, 1996.

[25] G. Qu and M. Potkonjak. Energy minimization with guaranteed quality of service. *Proc. 2000 international symposium on Low Power Electronics and Design*, pp. 43–49, 2000.

[26] R. Wahbe, S. Lucco, T. E. Anderson, and S. L. Graham. Efficient software-based fault isolation. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 203–216, December 1993.

[27] Mark Weiser, Brent Welch, Alan J. Demers, Scott Shenker. Scheduling for Reduced CPU Energy. *Proc. of Operating System Design and Implementation*, pp. 13–23, 1994.

[28] Y. Zhu and F. Mueller. Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling. *Proc. Real-Time and Embedded Technology and Applications Symposium, May 2004*.