To appear in Proceedings of the 34th Annual International Symposium on Microarchitecture, December 2001

Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications *

Christopher J. Hughes, Jayanth Srinivasan, and Sarita V. Adve
Department of Computer Science
University of Illinois at Urbana-Champaign
{cjhughes,srinivsn,sadve}@cs.uiuc.edu

Abstract

General-purpose processors are expected to be increasingly employed for multimedia workloads on systems where reducing energy consumption is an important goal. Researchers have proposed the use of two forms of hardware adaptation – architectural adaptation and dynamic voltage (and frequency) scaling or DVS – to reduce energy. This paper develops and evaluates an integrated algorithm to control both architectural adaptation and DVS targeted to multimedia applications. It also examines the interaction between the two forms of adaptation, identifying when each will perform better in isolation and when the addition of architectural adaptation will benefit DVS.

Our adaptation control algorithm is effective in saving energy and exploits most of the available potential. For the applications and systems studied, DVS is consistently better than architectural adaptation in isolation. The addition of architectural adaptation to DVS benefits some applications, but not all. Finally, in a seemingly counter-intuitive result, we find that while less aggressive architectures reduce energy for fixed frequency hardware, with DVS, more aggressive architectures are often more energy efficient.

1 Introduction

Multimedia applications are expected to form a large part of the workload on a growing number of systems, including future handheld computers, wireless phones, laptops, and desktop systems [6, 7, 18, 19]. General-purpose processors (as opposed to DSPs or ASICs) are expected to be increasingly employed for such workloads and systems [3, 6, 7]. Since most of these devices are powered by batteries, energy is a precious commodity. Therefore, reducing energy consumption of general-purpose processors for multimedia applications has become an important research goal.

Multimedia applications are required to process each unit of data, typically called a frame, within a time limit called a deadline. The processor may complete a frame's computation before the deadline, and may then remain idle until the end of the deadline. The existence of this idle time, called *slack*, implies that the processor can be slowed to save energy. The slowdown will not affect the perceived application performance as long as the frame is processed before the deadline. Further, this paper focuses on soft real-time applications, for which missing a small fraction of deadlines also does not perceptibly affect output quality.

This paper concerns hardware adaptations that slow down the processor to save energy for multimedia applications. Recall the execution time and energy equations:[1]

$$\begin{aligned} Execution\,time &= Instruction\,count \times \tfrac{1}{IPC} \times \tfrac{1}{f} \\ Energy &= Power \times Execution\,time \\ &= C\,V^2\,f\,\tfrac{Instruction\,count}{IPC \times f} \\ &= C\,V^2\,\tfrac{Instruction\,count}{IPC} \end{aligned}$$

Above, f is frequency, V is supply voltage, and C is the *effective* capacitance (including the effect of the number of gate transitions) [2]. These equations suggest at least two forms of hardware adaptation. The first is dynamic voltage scaling (DVS) [10, 11, 12, 20, 25, 26, 27, 29]. Reducing voltage reduces energy, but also requires reducing frequency, increasing execution time. The second technique adapts the architecture to reduce the effective capacitance [1, 8, 9, 17, 22, 23]. Since a lower effective capacitance often results in a lower IPC and vice versa, a net reduction in energy results only if the ratio of the two terms is reduced. Further, reducing IPC increases execution time.

*This work is supported in part by the National Science Foundation under Grant No. CCR-0096126 and funds from the University of Illinois at Urbana-Champaign. Sarita V. Adve is also supported by an Alfred P. Sloan Research Fellowship and Christopher J. Hughes is supported by a Richard T. Cheng Fellowship.

[1] The energy equation assumes dynamic power is dominant, as in current systems.

Examples of architectural adaptations are speculation control [22], changing instruction window size [8], changing the number of functional units [23], switching issue strategy between in-order and out-of-order [9], and shutting off parts of the cache [1].

Most studies have considered only one of these forms of adaptation (e.g., [1, 8, 9, 10, 11, 20, 22, 23, 25, 26, 27, 29]), and most studies of architectural adaptation target general applications without exploiting common characteristics of multimedia applications (e.g., [1, 8, 13, 17, 22, 23]).

This paper makes two contributions. First, we develop an integrated algorithm to control both architectural adaptation and DVS for saving processor energy for multimedia applications. The algorithm was partly outlined in our previous work [14]. Second, we use the algorithm to understand the interaction between the two forms of adaptation for multimedia applications. To our knowledge, this is the first such algorithm and study for multimedia applications.

Our adaptation control algorithm performs adaptations at the granularity of a frame. It begins with a short profiling phase that predicts the *energy per instruction (EPI)* of all possible hardware configurations (combinations of architecture and frequency), using profiles of a single frame for a subset of the configurations. Subsequently, before the execution of each frame, the algorithm predicts the number of instructions that will be executed for the frame. Using the instruction count and EPI estimates, the algorithm chooses the architecture and frequency combination that will consume the least energy and meet the deadline for the next frame. We also develop a simpler variation of the algorithm for a DVS system supporting a continuous range of frequencies [16]. The variation exploits the observation that with continuous DVS, a single architecture has the lowest EPI for most frequencies.

We evaluate the algorithm quantitatively, studying processors that have no adaptation, that perform only DVS, that perform only architectural adaptation, and that perform both DVS and architectural adaptation. We provide a qualitative analysis to explain the interaction between the two forms of adaptation. Our primary findings are as follows.

- Our adaptation control algorithm reduces slack effectively to provide significant energy savings with few missed deadlines. In cases where significant slack remains, most of it can be accounted for by the inherent limitations of the adaptive system and would be present regardless of the control algorithm.

- We identify the situations in which DVS alone or architectural adaptation alone saves the most energy, and when it is beneficial to add architectural adaptation to a system with DVS. For the applications, systems, and deadlines studied here, DVS alone gives significantly more benefits than architectural adaptation alone. Adding architectural adaptation to a system with DVS is significantly beneficial for some of our applications, but not for all.

- A seemingly counter-intuitive finding is that the energy efficiency of an architecture depends on whether the system supports DVS. As expected, for fixed frequency hardware, less aggressive (i.e., low IPC) architectures are usually useful for energy reduction. However, with DVS, more aggressive architectures are often more energy efficient. This behavior occurs because the higher IPCs of the more aggressive architectures often allow them to be run at lower frequencies. Thus, architects can use aggressive architectures at low frequency for energy savings, potentially side-stepping the high-frequency design problems that occur when using such architectures for high performance.

2 Controlling Hardware Adaptation

When employing adaptive hardware, two key questions to answer are: when to trigger an adaptation and what adaptation to trigger. This section uses the results and an algorithm outline from our previous work [14] to address these questions. For the first question (when to adapt), [14] recommended adaptation at the frame granularity. This is because (1) that work found variability in execution time for different frames, implying a potential for adaptation, and (2) a frame is usually associated with a completion deadline, giving a natural target for processor slowdown.

The rest of this section concerns the second question (what to adapt) for systems capable of changing both voltage/frequency and architecture. Section 2.1 gives some background and assumptions. Section 2.2 describes the algorithm for systems that change voltage in discrete steps, as in Transmeta's Crusoe processors. Section 2.3 describes a variation for systems that change voltage continuously, as in Intel's XScale processor. Sections 2.4, 2.5, and 2.6 discuss three remaining aspects of the algorithm – instruction count prediction, IPC changes, and overheads.

2.1 Background and Assumptions

Some of our applications statically distinguish multiple frame types (e.g., I, P, and B frames for MPEG). Our algorithm makes the same distinctions. The algorithm uses three results for multimedia applications from [14] as follows:

- IPC is almost constant for different frames of the same type at a given frequency. This is because while the amount of work per frame may vary, the nature of the work is the same for all frames of a given type.

- IPC of a frame is almost independent of clock frequency, since little time is spent in memory stalls.
- For a given frame type, instruction count varies slowly from frame to frame, due to mostly smooth changes in the amount of work per frame.

Based on the intuition behind the first result, we assume that average power dissipation is almost constant for all frames of the same type at a given frequency (validated in Section 3.2). We also assume hardware recognizes when a new frame begins, the type of the frame, and its deadline (in time units). It is conceptually straightforward to get this information from the software – either from the application or from the CPU scheduler. Real-time CPU schedulers typically require the application to provide deadline information and to identify the beginning of a new frame [5], and could also benefit from distinguishing between different frame types.

Below, we call the possible architectures *architectural configurations* and the possible combinations of architectures and voltage/frequency *hardware configurations*.

2.2 The Algorithm for Systems with Discrete DVS

As long as a frame of a multimedia application is processed within its deadline, its execution time is irrelevant to the perceived quality of the output but can impact energy. We therefore seek to use hardware that meets the deadline, but consumes minimal energy.

The algorithm proceeds in two phases – a *profiling phase* and an *adaptation phase*. During the profiling phase, the algorithm (1) determines the maximum instructions that a hardware configuration can execute within the deadline for each frame type, and (2) orders all hardware configurations in increasing order of *energy per instruction (EPI)*. In the adaptation phase, at the end of each frame the algorithm predicts the number of instructions that will be executed for the next frame of the same type. It then uses the profiling information to find the lowest energy hardware configuration that can execute the predicted number of instructions within the deadline. The following describes how the above goals are achieved in each phase. Figure 1 summarizes the discussion.

Profiling Phase

This phase is invoked at the start of the application. It measures the IPC and power dissipation for one frame of each type for each architectural configuration. This measurement is done at only one common frequency and voltage for all architectures. The IPC and power dissipation measurements are used as predictions for IPC and power dissipation of that architecture for all frames of that type (since both are expected to be roughly constant across all frames of a given type). Since IPC is expected to be almost constant across all frequencies, we use the same prediction for all frequencies. We scale power dissipation with the square of the voltage required for each frequency.

Given the IPC estimate for a hardware configuration, the algorithm calculates the maximum number of instructions that the hardware can execute within the deadline, $Imax$, as $deadline \times f \times IPC$.

For EPI calculation, consider a hardware configuration, H, with architecture, A, for some frame type. Denote the measured IPC and average power dissipation for architecture A and this frame type as IPC_A and P_A, respectively. The EPI for H and this frame type is:

$$EPI_H = \frac{Energy_H}{Instruction\,count} = \frac{C_A {V_H}^2}{IPC_A}$$

Since $P_A = C_A V^2 f$, where V and f are the voltage and frequency where power is measured and are constant for all architectures,

$$EPI_H \propto P_A \frac{{V_H}^2}{IPC_A} \quad (1)$$

The profiling phase uses this expression to order all hardware configurations by EPI. For each frame type, it builds an *energy-performance table* that gives the hardware configurations in increasing EPI order, along with their $Imax$.

The profiling phase need only last for a number of frames equal to Number of frame types $\times$ Number of architectural configurations. This is a negligible number compared to the total frames processed by our applications.

Adaptation Phase

This phase comprises the rest of the application run. It exploits the result that instruction count changes slowly from frame to frame. At the end of each frame, the hardware predicts the number of instructions to be executed for the next frame of the same type (as discussed in Section 2.4). The prediction is then matched against the $Imax$ values in the energy-performance table. The first entry with an $Imax$ as large as the prediction is predicted to be the hardware that will consume the least energy and still make the deadline.

Both phases can be implemented in either hardware or software. In software, the phases could be incorporated into the CPU scheduler, which would run the adaptive phase or part of the profiling phase each time the application completes a frame, when it yields control to the scheduler.

2.3 Variation for Systems with Continuous DVS

The algorithm described so far assumes a system with a finite number of frequency choices, when building the energy-performance tables. However, at least one commercial processor with DVS can vary frequency and voltage continuously [16]. Our algorithm can be applied to such a

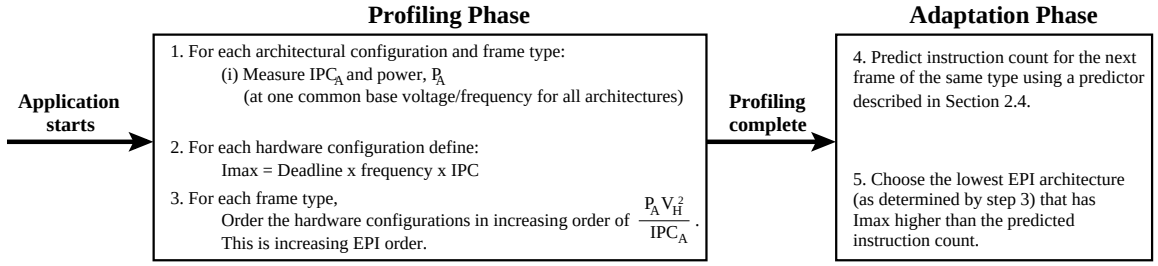


Figure 1. The algorithm for choosing hardware configurations for a system with discrete DVS.

system by considering only a finite number of frequencies, but may be sub-optimal. We develop a better variation of our algorithm for systems with continuous DVS as follows.

Re-examining equation 1, we note that voltage is related to frequency as $f_H = \frac{k \times (V_H - V_{threshold})^2}{V_H}$. Thus, for supply voltages sufficiently far from the threshold voltage, voltage is roughly proportional to frequency. Below, consider the range of the system when $V_H \gg V_{threshold}$, which can be expected to be a substantial part of the supported frequency range. This leads to $EPI_H \propto {f_H}^2 \frac{P_A}{IPC_A}$, and since $f_H = \frac{Imax_H}{Deadline \times IPC_A}$,

$$EPI_H \propto {Imax_H}^2 \frac{P_A}{{IPC_A}^3}$$

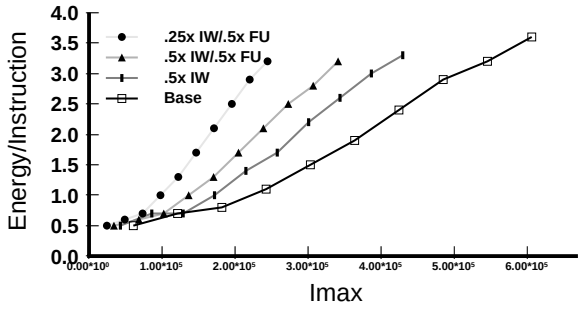


Figure 2. Example EPI versus $Imax$ curves for GSMenc. Each curve represents a different architecture from Table 3(a), and each point represents a different frequency from Table 3(b), ranging from 100MHz to 1GHz.

Since P_A and IPC_A are specific to an architecture, at each value of $Imax$, the lowest EPI hardware configuration has the same architecture. Thus, for each application and frame type, one architecture is most energy efficient for all values of $Imax$.[2] To illustrate this, Figure 2 shows EPI versus $Imax$ curves generated from the profiling phase of our control algorithm, using the experimental methodology in Section 3. The different curves represent different architectures studied in this paper (Table 3(a)). Each curve connects configurations with the same architecture, with frequency ranging from 100MHz to 1GHz. As expected, one line is lower than all others for all values of $Imax$ except at small values of $Imax$, where V_H is close to $V_{threshold}$.

The above observation motivates the following algorithm for a system with continuous DVS (summarized in Figure 3). The profiling phase finds the architecture with the smallest value of $\frac{P_A}{{IPC_A}^3}$. This has the lowest EPI for all values of $Imax$ except possibly at the lower frequencies. The adaptation phase computes the frequency necessary for that architecture to execute the predicted number of instructions within the deadline. If this frequency is within the range supported by the system, then the above architecture and this frequency are used for the next frame.

If the above frequency is outside the supported range, then the closest supported frequency (the minimum or maximum) is chosen. For the minimum frequency case, the architecture chosen above may not have the lowest EPI at that frequency. For the maximum frequency case, the architecture may not meet the deadline at that frequency. We adjust the algorithm to handle these cases as follows. The profiling phase creates two energy-performance tables, one each for the maximum and minimum frequencies. These tables are analogous to those for the discrete DVS algorithm, but each needs to contain only one entry per architectural configuration (at the corresponding frequency). If the minimum or maximum frequency is chosen, the architecture is determined by searching the corresponding table analogous to the discrete DVS algorithm. Thus, the first architecture in the table with $Imax \geq$ the predicted instruction count is chosen.

2.4 Instruction Count Predictor

Much work has been done on predicting execution times and processor utilizations for DVS (e.g., [10, 29]). This work can be applied to predicting instruction count of a frame as well. Typically, previous predictors use a function of the past application behavior to predict future behavior. We tested a wide range of predictors for the next frame's instruction count, including all of those tested in [10], as well as some others (e.g., stride predictors) [15]. Most previous studies have been concerned only with accuracy. However,

[2] This observation is not true for discrete DVS systems because not all frequencies are supported in the latter, as further explained in Section 4.4.

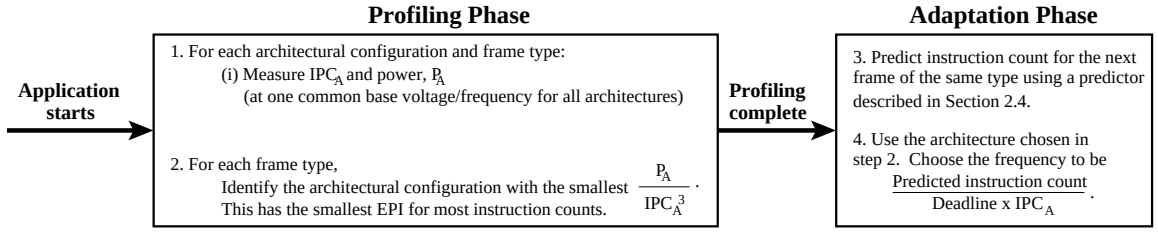


Figure 3. The algorithm for choosing hardware configurations for a system with continuous DVS. Two enhancements are applied to this algorithm to handle the minimum and maximum frequencies, as explained in the text.

since an under-prediction may cause a missed deadline, we also use the fraction of under-predictions as a metric. As one would expect, the predictors with the least error under-predict about half of the time, which is unacceptable. We set 5% as the limit for the fraction of under-predictions and introduce the following modifications to achieve this.

We add a margin of error, l, to each prediction, called the *leeway*. We multiply each prediction by l, where $l \geq 1$. The choice of l involves a tradeoff between the number of under-predictions and error. Since application behavior varies widely, we elect to dynamically change the leeway. We start l at 1.1 (10% greater than the prediction), and decay it by 0.25% per frame until a deadline is missed, in which case it is reset to 1.1.

Also, some applications have frames with much larger instruction counts than the surrounding frames (e.g., due to a scene change). We find it is best for the predictor to ignore these frames; therefore, we add some hysteresis to the predictor. If a frame's instruction count increases by $\geq$20% from the previous frame, the predictor ignores the frame.

With leeway and hysteresis, only one predictor meets our criteria of $\leq$5% under-predictions for all applications [15]. This predictor uses the maximum instruction count of the last five frames as the prediction for the next frame. This has a mean error of 9.2% (maximum 11.4%), and a mean fraction of under-predictions of 2.8% (maximum 4.8%). Therefore, we choose this predictor for our algorithm.

2.5 IPC Changes

Although we expect IPC to be nearly constant for all frames of a given type, small changes could result in sub-optimal energy savings or missed deadlines. The continuous DVS system is particularly sensitive to IPC reductions, which can lead to a slightly lower than optimal frequency choice causing a missed deadline. (The distance between supported frequencies in a discrete DVS system makes it relatively more robust). We therefore periodically correct for IPC changes in the adaptation phase as follows. For the discrete case, when a deadline is missed, the IPC and power values for the architecture just used are updated with the values from the frame just completed, and the energy-performance table is rebuilt. For the continuous case, after each frame, the IPC for the architecture used is updated with that of the frame just completed. We also add a 1% leeway to this IPC when computing the frequency at which to run the next frame.

2.6 Overheads

The algorithm does not incorporate adaptation overheads, but it can be modified to do so. For most of our applications, the deadline is much larger than the time to invoke either an architectural or DVS adaptation. Two exceptions are GSM and G728 (Table 4). GSM runs in a single hardware configuration in all our experiments, incurring only a one-time adaptation cost.[3] For G728, the time overhead for DVS is about 10% in the worst case (using data from [12]). The adaptation control algorithm will itself also incur overhead, but its simplicity makes us believe that those overheads will also generally be small relative to the frame size.

3 Experimental Methodology

3.1 Workload and Architectures Studied

Table 1 summarizes the nine applications and inputs used in this work. These were also used, and are described in more detail, in [14]. We also verified that the profiling phase run on a second set of inputs gives similar IPC and power dissipation values ($\leq$4% difference) for all applications on our base architecture (described below).

The base processor studied is similar to the MIPS R10000 and is summarized in Table 2. Several variations are also studied to model an adaptive processor (described in Table 3(a)). We allow adaptation of the issue width [17], the instruction window size [8], and the number of functional units [23]. All of these have a significant impact on both power and IPC for these applications (Table 3(a)), and so provide a range of interesting architectural configurations. We did not study an in-order issue variation because

[3] With multithreading, the cost depends on the thread that ran before the GSM frame, but such interactions are outside the scope of this paper.

App.	Input Size	Media	Frame Length
GSMenc	21.45s	Speech	20ms
GSMdec	20s		
G728enc	7.23s	Speech	625μs
G728dec	20s		
H263enc	4s	Video	40ms
H263dec	18s		
MPGenc	3.33s	Video	33.3ms
MPGdec	15s		
MP3dec	65.25s	Audio	26.1ms

Table 1. Workload description.

our energy simulator (described below) does not model in-order issue. For DVS, the voltages and frequencies considered for the discrete case are given in Table 3(b). For the continuous case, the same frequency range is considered. The voltages corresponding to the frequencies were derived from the information available for Intel's XScale processor [16]. This information gave us k and $V_{threshold}$ for the equation: $f = \frac{k \times (V_{supply} - V_{threshold})^2}{V_{supply}}$. We used the equation to extrapolate the rest of the desired points.

We evaluate six processors in this study. *NoAdapt* is the base processor. *Arch* supports only architectural adaptation as in Table 3(a). *DDVS* and *CDVS* support only discrete and continuous DVS, respectively. *Arch+DDVS* and *Arch+CDVS* support both architectural adaptation and DVS (discrete and continuous, respectively).

Base Processor Parameters	
Processor Speed	1GHz
Fetch/Retire Rate	8 per cycle
Functional Units	4 Int, 4 FP, 2 Address generators
Integer FU Latencies	1/7/12 add/multiply/divide (pipelined)
FP FU Latencies	4 default, 12 div. (all but div. pipelined)
Instruction window (reorder buffer) size	128 entries
Memory queue size	32 entries
Branch Prediction	2KB bimodal agree, 32 entry RAS
Base Memory Hierarchy Parameters	
L1 (Data)	64KB, 2-way associative, 64B line, 2 ports, 12 MSHRs
L1 (Instr)	16KB, direct mapped
L2 (Unified)	1MB, 4-way associative, 64B line, 1 port, 12 MSHRs
Main Memory	16B/cycle, 4-way interleaved
Base Contentionless Memory Latencies	
L1 (Data) hit time (on-chip)	2 cycles
L2 hit time (off-chip)	20 cycles
Main Memory (off-chip)	102 cycles

Table 2. Base (default) system parameters.

3.2 Performance and Energy Evaluation

We use the RSIM simulator [24] for performance evaluation, and the Wattch tool [4] integrated with RSIM for energy measurement. All applications were compiled with the SPARC SC4.2 compiler with the following options: *–xO4 –xtarget=ultra1/170 –xarch=v8plus*.

Name	Description (difference from base)	Mean IPC	Mean Power(W)
base	Base architecture	2.64	12.3
.5x IW	4-issue, 64 entry inst. win.	2.17	9.1
.5x IW/.5x FU	4-issue, 64 entry inst. win. 2 Int, 2 FP func. units	1.86	7.3
.25x IW/.5x FU	2-issue, 32 entry inst. win. 2 Int, 2 FP func. units	1.45	5.6

(a)

Freq. (MHz)	Voltage (V)	Freq. (MHz)	Voltage (V)
100	0.7	600	1.3
200	0.80	700	1.45
300	0.85	800	1.6
400	1.0	900	1.7
500	1.15	1000	1.8

(b)

Table 3. (a) Architectural configurations considered. IPC and power are averaged across all applications at 1GHz. (b) Voltage and frequency combinations for discrete DVS.

To model power for architectural adaptation, we generate separate power models for each possible architecture, as if each one were a separate processor. In particular, we assume that when an architecture other than the base is selected, the components not available in that architecture are powered down, consuming no energy. As discussed in Section 2.6, adaptation overheads are not modeled.

We verified our assumption that power dissipation is almost constant for all frames. The standard deviation normalized to the mean of the per frame power for each application and frame type on the base processor is always $\leq$3%.

3.3 Metrics and Deadlines

The primary metrics for our evaluation are energy consumed and missed deadlines. A missed deadline occurs when the execution time for a frame exceeds the deadline. We also examine the mean per frame execution time slack for each of the processors, not counting the frames with missed deadlines. The slack is the difference between the deadline and the execution time for the frame.

We consider two sets of deadlines, one loose and one relatively tight, to understand the dependence of our algorithm on this factor. The deadlines are shown in Table 4.

For the loose deadline, we use the application's frame length (from Table 1). This provides a bound on energy savings. The only exception is MPGenc, for which even *NoAdapt* cannot sustain the specified frame rate. Instead, we use twice this frame length (66.6ms) as the deadline for MPGenc.

We also study tighter deadlines. Real systems may run multiple applications simultaneously. In such a system, the admission control software typically provides a reservation for each application that is less than the loose deadline. We

	GSMenc	G728enc	H263enc	MPGenc	
Loose	20ms	625μs	40ms	66.6ms	
Tight	140μs	130μs	40ms	66.6ms	
	GSMdec	**G728dec**	**H263dec**	**MPGdec**	**MP3dec**
Loose	20ms	625μs	40ms	33.3ms	26.1ms
Tight	50μs	120μs	2.9ms	6.3ms	2.7ms

Table 4. Application deadlines.

	DDVS	*CDVS*	*Arch*	*Arch+DDVS*	*Arch+CDVS*
	Loose Deadlines				
Mean	0.4%	0.2%	0%	0.6%	0.2%
Maximum	2.3%	1.1%	0%	3.4%	1.1%
	Tighter Deadlines				
Mean	0.5%	2.2%	0%	0.8%	2.0%
Maximum	2.3%	4.3%	0%	3.4%	3.6%

Table 5. Missed deadlines.

study tighter deadlines to represent this scenario.[4]

We choose each application's tighter deadline to be three times the maximum frame processing time on *NoAdapt*. This rule ensures that the least aggressive architectures can meet the deadline for most frames at least at the higher frequencies, thereby providing some potential for adaptation. For the video encoders, we use the same deadlines as before because they were already relatively tight.

4 Results

We evaluated the six processors with both sets of deadlines. Section 4.1 presents the missed deadlines. Section 4.2 presents the energy consumption. Section 4.3 presents the hardware configurations selected by the algorithm. Section 4.4 presents a qualitative analysis of our algorithm and discusses interaction of architectural adaptation with DVS. Section 4.5 discusses the remaining potential.

4.1 Missed Deadlines

Table 5 shows the fraction of missed deadlines for the different processors, for each deadline. The mean fraction of missed deadlines averaged over all applications is very small ($\leq$ 2.2%) for all processors and deadlines. The maximum fraction for any application is also small ($\leq$ 4.3%).

4.2 Energy Consumption

Figure 4 shows the energy consumption for all processors normalized to *NoAdapt* for both sets of deadlines. Table 6 shows the mean relative energy reduction for different pairs of processors.

For both sets of deadlines, architectural adaptation and DVS combined gives the lowest energy. However, DVS alone gives most of this reduction. *DDVS* reduces energy

[4]There is clearly an interaction required between the admission control software and our algorithm. This is beyond the scope of this study.

Savings from	Relative to	Loose Deadline	Tighter Deadline
DDVS	*NoAdapt*	78%	68%
Arch	*NoAdapt*	22%	22%
CDVS	*DDVS*	1%	13%
Arch+DDVS	*DDVS*	17%	11%
Arch+CDVS	*CDVS*	16%	5%

Table 6. Mean relative energy savings for different processor pairs.

by averages of 78% and 68% for the loose and tighter deadlines, respectively. Relative to *DDVS*, *CDVS* shows significant reduction for some applications (up to 10% for the loose and 23% for the tighter deadlines). The average reduction is 13% for the tighter deadlines, but only 1% for the loose deadlines. For MPGenc, *CDVS* performs slightly worse than *DDVS* for reasons explained in Section 4.4.1.

Architectural adaptation alone shows significant savings (22% mean with both sets of deadlines), but these are much lower than DVS alone. When added to DVS, for the loose deadlines, architectural adaptation reduces energy by a mean of 17% over *DDVS* and 16% over *CDVS*. The average reduction from adding architectural adaptation to *DDVS* and *CDVS* with tighter deadlines is more modest (11% and 5%, respectively), but each case shows significant benefits ($\geq$ 20%) for two or more applications.

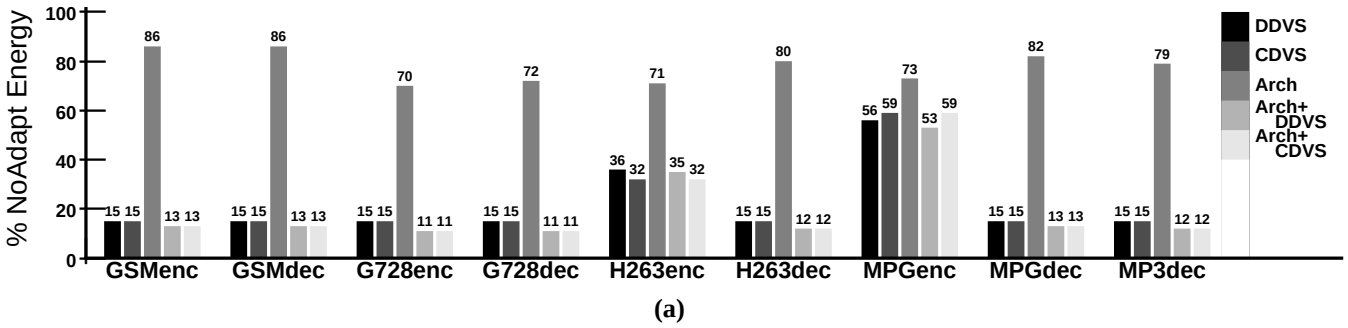
Thus, architectural adaptation gives lower (though substantial) benefits than DVS when used in isolation, and gives mixed benefits when used in addition to DVS.

4.3 Configurations Used

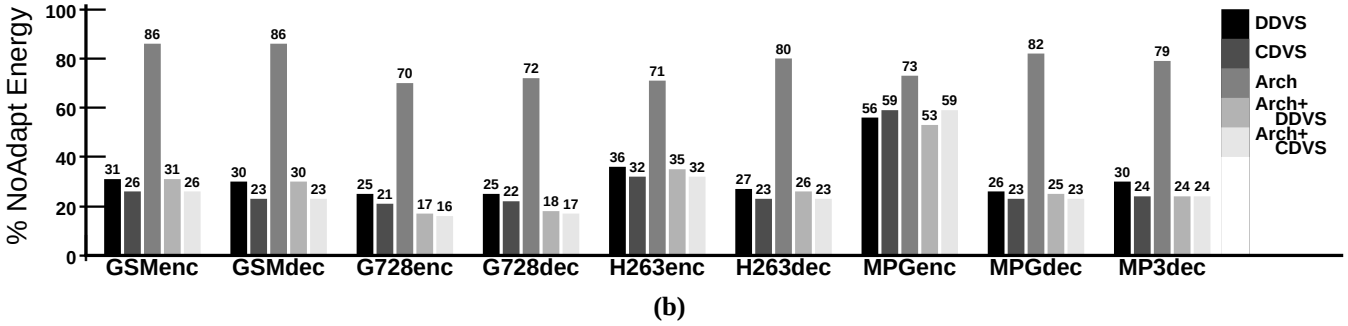
The frequencies exercised with DVS are as expected. With loose deadlines, processors with DVS use the minimum frequency for all applications except the video encoders, showing there is considerable slack. For tighter deadlines, more frequencies are exercised. The detailed data appears in [15], and is not shown here for lack of space.

Architectural adaptation shows more intriguing results. Figure 5 shows the architectural configurations exercised for *Arch*, *Arch+DDVS*, and *Arch+CDVS* for each application for the tighter deadlines. Each bar shows the percentage of frames that used a specific configuration. The figure shows that different architectures are exercised depending on whether there is DVS and whether it is discrete or continuous. Architectural adaptation alone exercises the two least aggressive architectures for all but a few frames. In contrast, architectural adaptation used with DVS exercises the more aggressive architectures quite frequently. In particular, continuous voltage scaling with tighter deadlines uses the most aggressive architecture exclusively for six out of nine applications. It is also interesting that the architectures exercised in the discrete and continuous cases have some differences.

These results lead to the non-intuitive conclusion that the most energy efficient architecture is different for systems



(a)



(b)

Figure 4. Energy consumption for (a) loose deadlines and (b) tighter deadlines.

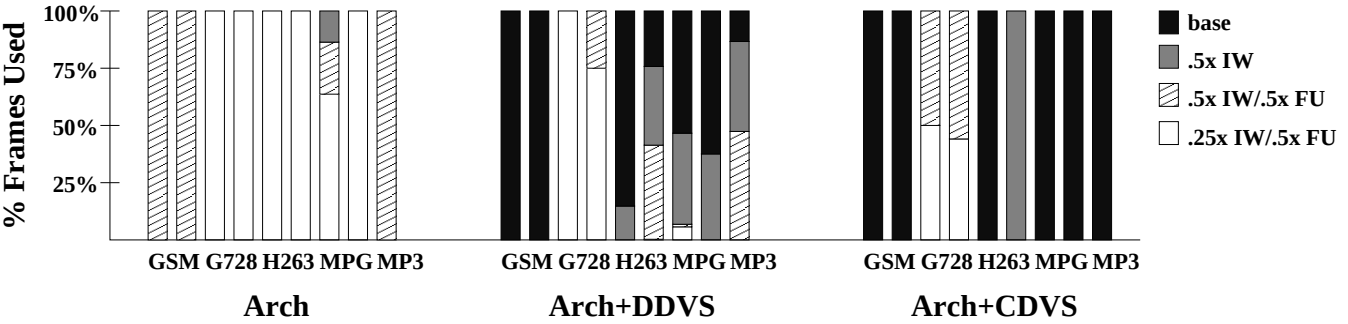


Figure 5. Architectural configurations used for tighter deadlines. Each application name (except MP3) refers to two bars, with the encoder on the left and the decoder on the right.

that do or do not use DVS. Without DVS, less aggressive architectures are more energy efficient, but with DVS, more aggressive architectures are often more efficient.

4.4 Qualitative Analysis

This section presents a qualitative analysis to explain the above results on the relative performance of the different processors and the configurations they exercise. For understanding, we use the EPI versus $Imax$ graphs as in Figure 2 in Section 2.3. Only *Arch+CDVS* has all points on this graph available for adaptation; the rest of the processors adapt between a subset of these points as discussed below.

For a predicted number of instructions (I) for a frame, the lowest energy configuration that will meet the deadline is the one with the lowest EPI and $Imax \geq I$. Our algorithm picks exactly this configuration with some exceptions for *Arch+CDVS* that are recalled below.

4.4.1 DVS Alone

Processors with DVS alone have points on only the base curve in the EPI graph available. *CDVS* can adapt between all points on the curve while *DDVS* can only adapt between the discrete points marked on this curve. Since *CDVS* supports more points, it is likely to perform better than *DDVS*. The primary exceptions to this are if there is too much or too little slack, in which case both *DDVS* and *CDVS* run at the minimum or maximum frequency. This occurs for the loose deadlines; therefore, *CDVS* shows no added benefit for any application except H263enc. MPGenc actually sees lower energy savings for *CDVS* because the IPC prediction used by *DDVS* is larger than the actual value for many frames. This results in *DDVS* choosing a lower frequency

than *CDVS* (thereby saving more energy), but it also results in more missed deadlines for *DDVS* (2.3% instead of 1.1%).

4.4.2 Adaptive Architecture Alone

For a processor with an adaptive architecture alone, the only points available on the EPI graph are the ones at a fixed (the maximum) frequency. The lowest of these varies from application to application, but we expect the less aggressive architectures (i.e., lower IPC) to have lower EPI at the same frequency. In our suite, the least aggressive architecture has the lowest EPI for six of the nine applications (the second least aggressive is lowest for the other three). The architecture with the lowest EPI will be chosen if the predicted instruction count does not exceed its $Imax$. For larger predicted instruction counts, a more aggressive architecture will be chosen to make the deadline. In our applications, the two least aggressive architectures can meet all deadlines except for some in MPGenc. Thus, architectural adaptation provides substantial benefits over no adaptation.

For our applications and deadlines, DVS is always better than architectural adaptation because DVS always has a point with a lower or same EPI with a large enough $Imax$. Nevertheless, it is possible for architectural adaptation to do better than DVS, as illustrated in Figure 6. With *CDVS*, this is possible only if the EPI for a less aggressive architecture at the maximum frequency (point A in Figure 6(a)) is below the base architecture curve. Then *Arch* will be better than *CDVS* for instruction counts that are (1) less than the $Imax$ of configuration A and (2) greater than an $Imax$ where the DVS curve (the base architecture) has the same EPI as A (point B in Figure 6(a)). In our experiments, G728enc and G728dec satisfy the first condition, but the instruction counts were too low to see higher gains from *Arch*.

Arch can do better than *DDVS* for the above case, and also because *DDVS* may not support the frequency at which the base architecture would give the optimal EPI for a given $Imax$ (e.g., point A in Figure 6(b)). Thus, *DDVS* will choose the next supported frequency with the base architecture (point B). In that case, if a less aggressive architecture (at maximum frequency) gives a better EPI than point B, and the architecture has a high enough $Imax$ (point C in Figure 6(b)), *Arch* will do better than *DDVS*.

4.4.3 Continuous DVS Combined with an Adaptive Architecture

Arch+CDVS has all points in the EPI graph available. Our algorithm typically picks the architecture with the lowest $\frac{P_A}{IPC_A{}^3}$, and uses the frequency that will execute the predicted instructions within the deadline. If the computed frequency is outside the supported range, the algorithm picks the closest supported frequency and the lowest EPI architecture at that frequency that can meet the deadline.

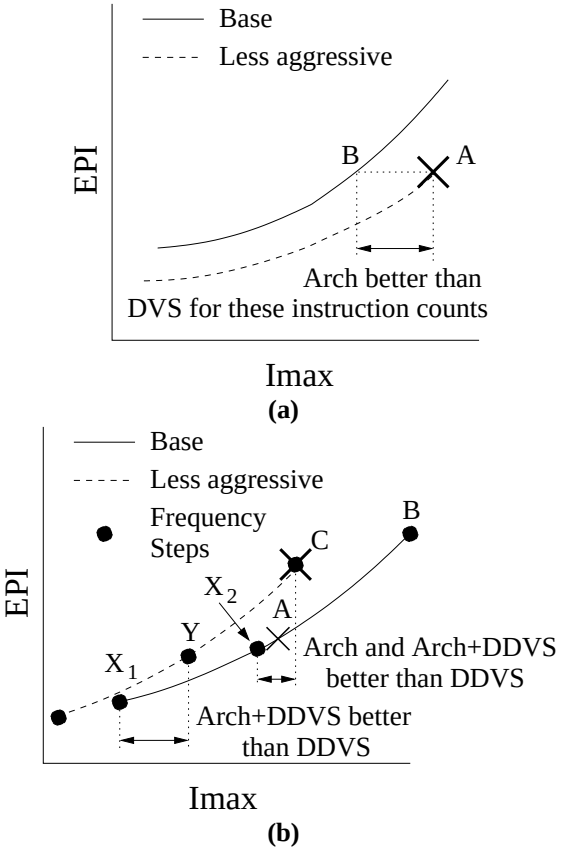


Figure 6. Situations for which architectural adaptation is better than or enhances DVS.

The architecture with the lowest EPI curve depends on the tradeoff between P_A and $IPC_A{}^3$. For most of our applications, this architecture is base. While it may seem counter-intuitive for a more aggressive architecture to be more energy-efficient than a less aggressive one, the higher IPC of the former means that it can be run at a lower frequency. The drop in EPI from the lower frequency may overcome the larger P_A due to the increased aggressiveness.

The above observation has significant implications for architects. When considering performance, it is important to target aggressive architectures at high frequencies. Such a design poses formidable challenges, often forcing a compromise between IPC and increased complexity (e.g., clustered microarchitectures). The above observation implies that, with DVS, it is sufficient to target aggressive architectures for the lower frequency ranges for the purpose of energy efficiency. Thus, adaptive processors could be designed so that the more aggressive architectures are only run at the lower frequencies for higher energy efficiency, resulting in relatively simpler designs for such architectures.

The base architecture has the smallest $\frac{P_A}{IPC_A{}^3}$ for six of the nine applications (all except G728enc, G728dec, and H263dec). However, for the loose deadlines, the predicted

instruction counts for all frames result in frequencies below the minimum supported for four of these six applications; therefore, they choose less aggressive architectures and show significant energy reductions from the addition of architectural adaptation to *CDVS*. For the tighter deadlines, the base architecture is used for all six of the above applications; therefore, they show no benefit from adding architectural adaptation. Of the other applications, G728enc and G728dec see the most improvement from adding architectural adaptation (28% and 27%, respectively).

Effectively, we will see benefits from architectural adaptation added to continuous DVS only if (1) the architecture with the lowest $\frac{P_A}{IPC_A{}^3}$ is not base, or (2) if the predicted instruction count requires a frequency lower than the minimum supported and another less aggressive architecture can meet that count with a lower EPI at the minimum frequency.

4.4.4 Discrete DVS Combined with an Adaptive Architecture

Arch+DDVS is similar to *Arch+CDVS* except that on the EPI graph, it has only the discrete points marked on all curves available. Thus, in this case, our algorithm will behave in a manner similar to *Arch+CDVS* with one primary exception: architectural adaptation may occur for instruction counts within the discrete $Imax$ ranges of the architecture with the smallest $\frac{P_A}{IPC_A{}^3}$. Figure 6(b) illustrates when this can happen. In our experiments, we see this case occur most for MP3dec, where architectural adaptation due to the frequency step gives an extra 21% and 20% savings for the loose and tighter deadlines, respectively.

4.5 Available Potential Exploited

We next seek to understand the extent to which our algorithm exploits the available potential for energy savings. We use the execution time slack for this purpose since the algorithm seeks to minimize this slack to save energy. Figure 7 shows the mean slack for the tighter deadlines.

The remaining slack in all cases has four possible sources: the minimum frequency of the adaptive system, the finite number of frequency choices with discrete DVS, variability in IPC, and over-estimatation (error) in instruction count prediction. The first two are inherent to the system and would be present regardless of the control algorithm.

Where significant slack remains, the hardware usually runs at the minimum frequency or suffers from the frequency step. Thus, most of this slack is apparently due to system limitations. For corroboration, we focus on *Arch+CDVS* with the tighter deadlines since it has the least system constraints (fewest frames run at lowest frequency and no frequency steps). For this processor, G728dec has the largest remaining slack (18%). Half of this application's frames run at the minimum frequency, and so much of its remaining slack is attributable to this system constraint.

The above observations lead us to conclude that our algorithm is effective in eliminating most of the execution time slack that is not inherent to the system.

5 Related Work

There has been a lot of work in the area of low power and low energy systems. For lack of space, we focus our discussion on systems supporting DVS (Section 5.1), architectural adaptation (Section 5.2), and both DVS and architectural adaptation (Section 5.3) to save energy from dynamic switching. Excellent coverage of circuit-level techniques to automatically detect idle circuits and switch them off to save power and energy appears in [2].

5.1 DVS

Significant work has been done on algorithms to control DVS. In general, these algorithms predict future work and set the voltage and frequency of the system based on this prediction. Most of this work is *interval* based, meaning the algorithm divides the execution time into fixed intervals and predicts processor utilization in an interval as some function of the utilization in the previous intervals [10, 11, 12, 20, 29]. The processor speed is set based on this prediction and some heuristics. A recent study on a real system by Grunwald et al. found that none of these heuristic policies gave significant energy savings [11].

Our work differs and can give better results than interval-based algorithms because we perform our scheduling on frame boundaries. Frame based intervals are a natural fit to multimedia applications. We additionally introduce leeway in our prediction strategy to explicitly minimize missed application deadlines.

More recent work on DVS has also considered frame boundaries. These studies show significant energy savings on a small set of applications or on synthetic benchmarks [25, 26, 27]. Our energy savings results confirm these results for a wider range of applications.

Recently, Šimunić et al. have proposed a stochastic algorithm for execution time prediction, but they do not seem to consider under-predictions [28]. Lorch et al. propose PACE, a framework that can work in conjunction with any existing DVS algorithm [21]. Working with the existing prediction algorithm, PACE proposes an optimal voltage/frequency schedule which leads to better results. It would be interesting to map these ideas to our algorithm, but this is beyond the scope of this study.

None of the above studies consider architectural adaptation, either by itself, or integrated with DVS.

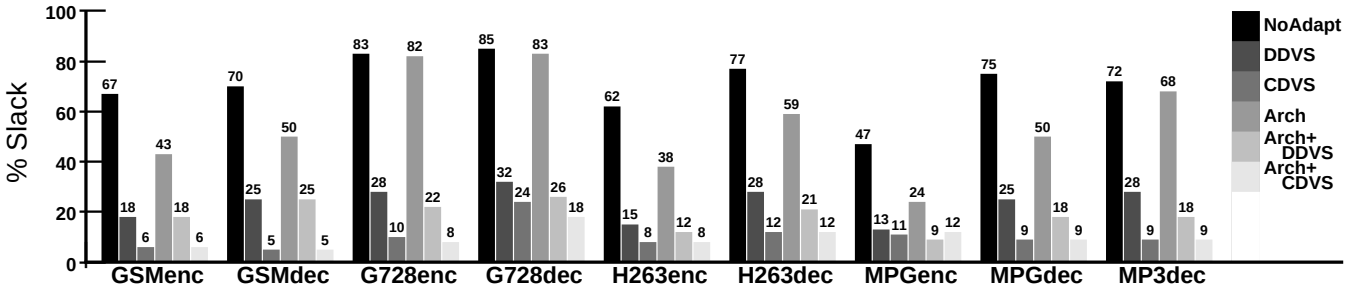


Figure 7. Slack for tighter deadlines.

5.2 Architectural Adaptation

Several researchers have proposed architectural adaptations for energy savings including speculation control [22], changing instruction window size [8, 17], changing the active functional units [23], changing issue width [17], and shutting off parts of the cache [1]. These techniques have been evaluated primarily for SPEC and technical benchmarks, and either ignore execution time degradation (e.g., [17]) or try to minimize it (e.g., [22]). In contrast, multimedia applications inherently have slack. Our algorithm exploits this slack to save energy by increasing execution time without impacting perceived performance.

Ghiasi et al. propose switching between in-order and out-of-order issue to exploit slack in multimedia applications [9]. They use IPC to determine when to switch. This study evaluates the technique only on MPGdec, and does not give energy savings results. Their algorithm also uses fixed instruction count intervals.

5.3 Combined Architectural Adaptation and DVS

To our knowledge, Huang et al. propose the only other framework capable of dynamically choosing multiple energy saving techniques related to architectural adaptation and DVS [13]. That work is driven by general applications. It requires software to provide the maximum slowdown allowed in terms of reduction in IPC over a base configuration, and a priority ordering of various energy saving techniques in order of their effectiveness. Hardware to effect adaptation is invoked every few milliseconds. An energy saving technique is chosen to be used if the reduction in the IPC due to that technique does not exceed the acceptable slowdown. Our framework is different because (1) we adapt at a frame granularity which is the typical granularity at which multimedia tasks are scheduled, and (2) we exploit characteristics of multimedia applications discovered in [14]. Specifically, we show that at the frame granularity, the number of instructions rather than the IPC should be used to control adaptation.

Our previous work studied execution time variability for multimedia applications, and used those results to outline an algorithm for controlling hardware adaptation to save energy [14]. The algorithm for discrete DVS presented here is based on that work. This paper advances the previous work by fully developing the discrete DVS algorithm, including identifying an appropriate instruction count predictor; by developing a simpler variation for continuous DVS, for which the previously outlined algorithm may be suboptimal; by evaluating the algorithm extensively; and by analyzing the interaction between architectural adaptation and DVS.

6 Conclusions

High energy consumption limits the use of general-purpose processors for the increasingly important workload of multimedia applications, but hardware adaptation is a possible solution to this. Two important forms of adaptation proposed previously are architectural and voltage/frequency adaptation (DVS). We have developed and evaluated an algorithm for controlling a processor with an adaptive architecture and DVS in an integrated way. To our knowledge, this is the only such algorithm targeted at multimedia applications and this is the first integrated evaluation of architectural adaptation and DVS for multimedia applications.

We have three sets of findings. First, our algorithm is effective at controlling adaptive architectures and DVS for multimedia applications. The algorithm effectively eliminates slack to save energy with few missed deadlines for the systems studied. In cases where significant slack remains, most of this remaining slack can be accounted for by the inherent limitations of the adaptive system.

Second, we examine the interaction of DVS and architectural adaptation. We identify the situations in which DVS alone or architectural adaptation alone would perform best, and when it is beneficial to add architectural adaptation to DVS. For the applications, systems, and deadlines studied here, we find that DVS alone gives most of the energy benefits; architectural adaptation is beneficial, both alone and with DVS, but helps less than DVS.

Third, in a seemingly counter-intuitive result, we find that more aggressive (i.e., higher IPC) architectures are

more energy efficient for many multimedia applications in the presence of DVS, since they can be run at lower frequency. This implies that for energy efficient adaptive processors with DVS, it is beneficial to target aggressive architectures only for low frequencies, avoiding the complexities of high frequency design for such architectures.

There are several directions for future work. First, we need to consider multiprogrammed workloads, and understand the interaction with the operating system's CPU scheduler. Second, we would like to explore intra-frame hardware adaptation. Third, it may be possible to improve our instruction count predictors using insights from previous work. Finally, we are exploring the design of high IPC, low frequency architectures for multimedia applications.

References

[1] D. H. Albonesi. Selective Cache Ways: On-Demand Cache Resource Allocation. In *Proc. of the 32nd Annual Intl. Symp. on Microarchitecture*, 1999.

[2] L. Benini and G. D. Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Acad. Publ., 1997.

[3] G. Blalock. Microprocessors Outperform DSPs 2:1. *Microprocessor Report*, December 1996.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proc. of the 27th Annual Intl. Symp. on Comp. Architecture*, 2000.

[5] H.-H. Chu. *CPU Service Classes: A Soft Real Time Framework for Multimedia Applications*. PhD thesis, University of Illinois at Urbana-Champaign, 1999.

[6] T. M. Conte et al. Challenges to Combining General-Purpose and Multimedia Processors. *IEEE Computer*, December 1997.

[7] K. Diefendorff and P. K. Dubey. How Multimedia Workloads Will Change Processor Design. *IEEE Computer*, September 1997.

[8] D. Folegnani and A. González. Energy-Efficient Issue Logic. In *Proc. of the 28th Annual Intl. Symp. on Comp. Architecture*, 2001.

[9] S. Ghiasi, J. Casmira, and D. Grunwald. Using IPC Variation in Workloads with Externally Specified Rates to Reduce Power Consumption. In *Proc. of the Workshop on Complexity-Effective Design*, 2000.

[10] K. Govil, E. Chan, and H. Wasserman. Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU. In *Proc. of the 1st Intl. Conf. on Mobile Computing and Networking*, 1995.

[11] D. Grunwald et al. Policies for Dynamic Clock Scheduling. In *Proc. of the 4th Symposium on Operating Systems Design and Implementation*, 2000.

[12] T. R. Halfhill. Transmeta Breaks x86 Low-Power Barrier. *Microprocessor Report*, February 2000.

[13] M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas. A Framework for Dynamic Energy Efficiency and Temperature Management. In *Proc. of the 33rd Annual Intl. Symp. on Microarchitecture*, 2000.

[14] C. J. Hughes et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *Proc. of the 28th Annual Intl. Symp. on Comp. Architecture*, 2001.

[15] C. J. Hughes, J. Srinivasan, and S. V. Adve. Supplemental Data for "Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications". URL: http://www.cs.uiuc.edu/rsim/Pubs/micro34-supplement.ps.

[16] Intel XScale Microarchitecture. http://developer.intel.com/design/intelxscale/benchmarks.htm.

[17] A. Iyer and D. Marculescu. Power Aware Microarchitecture Resource Scaling. In *Proc. of the Design, Automation and Test in Europe Conf.*, 2001.

[18] C. E. Kozyrakis and D. Patterson. A New Direction for Computer Architecture Research. *IEEE Computer*, November 1998.

[19] R. B. Lee and M. D. Smith. Media Processing: A New Design Target. *IEEE Micro*, August 1996.

[20] Y.-H. Lee and C. Krishna. Voltage-Clock Scaling for Low Power Energy Consumption in Real-Time Embedded Systems. In *Proc. of the 6th Intl. Conference on Real-Time Computing Systems and Applications*, 1999.

[21] J. R. Lorch and A. J. Smith. Improving Dynamic Voltage Scaling Algorithms with PACE. In *Proc. of ACM SIGMETRICS*, 2001.

[22] S. Manne, A. Klauser, and D. Grunwald. Pipeline Gating: Speculation Control for Energy Reduction. In *Proc. of the 25th Annual Intl. Symp. on Comp. Architecture*, 1998.

[23] R. Maro, Y. Bai, and R. Bahar. Dynamically Reconfiguring Processor Resources to Reduce Power Consumption in High-Performance Processors. In *Proc. of the Workshop on Power-Aware Computer Systems*, 2000.

[24] V. S. Pai, P. Ranganathan, and S. V. Adve. RSIM Reference Manual version 1.0. Technical Report 9705, Department of Electrical and Computer Engineering, Rice University, August 1997.

[25] T. Pering, T. Burd, and R. Brodersen. Voltage Scheduling in the lpARM Microprocessor System. In *Proc. of the Intl. Symposium on Low Power Electronics and Design*, 2000.

[26] P. Pillai and K. G. Shin. Real-Time Dynamic Voltage Scaling for Low-Power Embedded Operating Systems. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, 2001.

[27] J. Pouwelse, K. Langendoen, and H. Sips. Energy Priority Scheduling for Variable Voltage Processors. In *Intl. Symp. on Low-Power Electronics and Design*, 2001.

[28] T. Šimunić et al. Dynamic Power Management for Portable Systems. In *Intl. Conf. on Mobile Computing and Networking*, 2000.

[29] M. Weiser et al. Scheduling for Reduced CPU Energy. In *Proc. of the 1st Symposium on Operating Systems Design and Implementation*, 1994.