To appear in Proceedings of the 28th International Symposium on Computer Architecture, June 2001

Variability in the Execution of Multimedia Applications and Implications for Architecture *

Christopher J. Hughes, Praful Kaul†, Sarita V. Adve, Rohit Jain, Chanik Park‡, and Jayanth Srinivasan

Dept. of Computer Science
University of Illinois
at Urbana-Champaign
rsim@cs.uiuc.edu

†Transmeta Corporation
pkaul@transmeta.com

‡Dept. of Computer
Science and Engineering
Seoul National University
park@iris.snu.ac.kr

Abstract

Multimedia applications are an increasingly important workload for general-purpose processors. This paper analyzes frame-level execution time variability for several multimedia applications on general-purpose architectures. There are two reasons for such an analysis. First, it has been conjectured that complex features of such architectures (e.g., out-of-order issue) result in unpredictable execution times, making them unsuitable for meeting real-time requirements of multimedia applications. Our analysis tests this conjecture. Second, such an analysis can be used to effectively employ recently proposed adaptive architectures.

We find that while execution time varies from frame to frame for many multimedia applications, the variability is mostly caused by the application algorithm and the media input. Aggressive architectural features induce little additional variability (and unpredictability) in execution time, in contrast to conventional wisdom.

The presence of frame-level execution time variability motivates frame-level architectural adaptation (e.g., to save energy). Additionally, our results show that execution time generally varies slowly, implying it is possible to dynamically predict the behavior of future frames on a variety of hardware configurations for effective adaptation.

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 telephones, laptop computers, and desktop systems [6, 7, 16, 17]. General-purpose processors (vs. specialized DSP processors or ASICs) are expected to be increasingly employed for such workloads [2, 6, 7]. This paper analyzes the variability in the execution time of several multimedia applications at the frame granularity on general-purpose architectures. Two broad sets of implications motivate such an analysis.

First, several articles have conjectured that current complex general-purpose architectural features such as out-of-order issue, branch prediction, and caches induce significant unpredictability in execution time, making them undesirable for multimedia applications [2, 6, 7, 8, 16]. Typical multimedia applications periodically process a set of data, commonly called a frame, and each frame must be completed in a certain amount of time. This real-time nature makes it important to be able to predict the execution time for multimedia applications. If execution time is unpredictable, then it is difficult to know how much processing power to schedule to guarantee a desired frame rate, or, conversely, what frame rate is sustainable with a given amount of processing power.

Although there is a widespread perception that current general-purpose architectures induce excessive execution time unpredictability [2, 6, 7, 8, 16], there is little quantitative data to support it. Furthermore, unpredictability itself is difficult to quantify. We use execution time variability at the frame granularity to quantify predictability and test the above perception.

The second motivation for our work concerns the use of adaptive architectures for multimedia applications. Researchers have recently begun to propose adaptive architectures that can dynamically reduce power and/or energy [1, 4, 10, 20]. Such techniques are especially relevant to a large class of systems running multimedia applications, where battery-life is a precious commodity. The applica-

*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. Chanik Park is supported by the BK21 SNU-UIUC program.

bility of adaptive architectures to multimedia applications depends on the amount of variability in these applications and the ability to predict such variability.

This paper performs a detailed quantitative analysis of execution time variability at the frame granularity for a number of multimedia applications. The results have significant implications for both predictability and adaptivity for general-purpose architectures. Specifically, we use an application suite of nine encoders and decoders for speech, video, and audio (music) media types (Section 2), and make the following contributions.

We find that several applications in our suite exhibit significant variability in execution time across different frames (Section 3). Most of this variability, however, arises from the input-dependent nature of the algorithms used, combined with the behavior of the inputs. Compared to this input-induced variability, the variability due to complex architectural features is negligible in almost all cases. This finding challenges the conventional wisdom that features of current general-purpose architectures induce significant unpredictability (Section 4).

The presence of frame-level execution time variability also suggests a potential for architectural adaptation [1, 4, 10, 20] at a frame granularity (Section 4). Our results also show that execution time varies slowly in most cases; therefore, it is possible to dynamically predict the behavior of future frames for a variety of hardware configurations for effective adaptation.

2. Methodology

2.1. Workload Description

Our workload consists of nine encoders and decoders (codecs) encompassing three media types – speech, video, and audio (music) – and is summarized in Table 1. We obtained codes for these applications from various public domain sources. The applications were chosen for their importance in real systems and (we believe) to be representative enough to make the inferences in this study.

We evaluated all our applications with four inputs, summarized in Table 2. For lack of space, we only report results from a single input for each application. We chose the input that gave the highest (normalized) standard deviation in per frame execution time on our base system. We call these inputs the *default inputs*, and list them in the second column of Table 1. Results with the other inputs are similar, both quantitatively and qualitatively.

Our analysis uses a frame (described in Table 1) as the basic unit of work for each application. The G728, H263, and MPG codecs statically distinguish multiple frame types. G728 uses an adaptive algorithm, where certain parameters are updated every four frames. The processing of each frame in a single four-frame cycle is different due to the calculation of these parameters. Thus, we treat these as different types of frames (numbered one through four). The H263 and MPG codecs use almost the same video compression scheme. A key difference is that MPG uses three different types of frames – I frames do not exploit inter-frame redundancy, P frames exploit inter-frame redundancy using a previous frame, and B frames exploit such redundancy using a previous and a later frame. Our H263 codecs do not use B frames. They use a single I frame at the beginning of the video and P frames for the rest. We do not include the I frame in our analysis.

It takes excessively long to simulate a frame with the MPG codecs using the frame sizes specified by the MPEG-2 standard (about 4 to 16 hours per frame for MPGenc on the system discussed in Section 2.2). We scaled down the frame size to 176x144 pixels so that we could simulate a reasonable number of frames to assess execution time variability. We ensured that the scaling did not affect the cache behavior by performing a working set analysis and running representative experiments with larger frame sizes and different cache sizes, as discussed further in Section 2.2. The chosen frame size conforms to the H.263 standard, so we used the same size for the H263 codecs for consistency. Also for consistency, we used the same set of four inputs for both MPG and H263 codecs. These inputs contain a great deal of motion to stress the applications. H263 was designed for low bit-rate applications such as video conferencing (which typically have less motion); therefore, our results from these inputs represent an upper bound on the expected variability for H263.

2.2. Architectures Studied and Experimental Methodology

The base architecture studied consists of an out-of-order processor similar to the MIPS R10000, and is summarized in Table 3. Since the applications studied have small instruction footprints, the instruction cache is assumed to be perfect and is not modeled. Several variations on the base architecture are also studied, and are described in the corresponding sections.

To ensure that the scaled inputs of the MPG codecs (Section 2.1) did not affect the cache behavior, we performed a working set analysis. We found that for the standard MPEG-2 frame size of 352x240 pixels, the first and second level working sets fit in the base L1 and L2 caches, respectively. Thus, the cache behavior of our scaled inputs is expected to be similar to this standard frame size. For the standard frame size of 704x480 pixels, we found that the first level working set still fits in the base L1 cache, but the second level working set does not fit in the base L2 cache. To account for the latter, we determined the performance

Application	Default Input (see Table 2)	Description	Frame Size, Frame/Sample Rate
Speech Codecs			
GSMenc	orignova	Low bit-rate speech coding based on the European GSM 06.10 provisional standard. Uses RPE/LTP (residual pulse excitation/long term prediction) coding at 13Kb/s. Compresses frames of 160 16-bit samples into 264 bits.	20ms (160 samples), 8KHz
GSMdec	homemsg		
G728enc	lpcqutfe	High bit-rate speech coding based on the G.728 standard. Uses low-delay CELP (code excited linear prediction) coding at 16Kb/s. Compresses frames of five 16-bit samples into 10 bits.	625μs, (5 samples), 8KHz
G728dec	homemsg		
Video Codecs			
H263enc	buggy	Low bit-rate video coding based on the H.263 standard. Primarily uses inter-frame coding (P frames). Widely used for bit-rates less than 64Kb/s	40ms, 25frames/s
H263dec	tens		
MPGenc	buggy	High bit-rate video coding based on the MPEG-2 video coding standard. Uses intra-frame (I) and inter-frame (P, B) coding. Typical bit rate is 1.5-6Mb/s.	33.3ms, 30 frames/s
MPGdec	flwr		
Audio (Music) Codecs			
MP3dec	filter	Audio decoding based on the MPEG Audio Layer-3 standard. Synthesizes an audio signal out of coded spectral components. Typical bit rate is 16-256Kb/s.	26.1ms (1151 samples), 44.1KHz

Table 1. Workload description.

Input	Description	Size in frames	Length in seconds
Speech		GSM/G728	Both
clinton	Speech by Clinton	922/29504	18.44
homemsg	An answering message	1000/32000	20
lpcqutfe	Sentence read by a boy	362/11572	7.23
orignova	Sentences read by 8 adults concatenated	1073/34328	21.45
Video		Both	H263/MPG
buggy	Buggy race	450	18/15
cact	Pan over still-life	450	18/15
flwr	Drive-by of houses	450	18/15
tens	Table tennis match	450	18/15
Audio			
beethoven	Classical piece	2500	65.25
cat_stevens	Soft rock song	2500	65.25
sting	Pop song	2500	65.25
filter	Rock song	2500	65.25

Table 2. Inputs used for the workload.

Base Processor Parameters	
Processor Speed	1GHz
Fetch/retire Rate	4 per cycle
Functional Units	2 Int, 2 FP, 2 Address generation
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	64 entries
Memory queue size	32 entries
Branch Prediction	2KB bimodal agree, 32 entry RAS
Base Memory Hierarchy Parameters	
L1 D-cache	64KB, 2-way associative, 64B line, 2 ports, 12 MSHRs
L2 D-cache	1MB, 4-way associative, 64B line, 1 port, 12 MSHRs
Main Memory	16B/cycle, 4-way interleaved
Base Contentionless Memory Latencies	
L1 hit time (on-chip)	2 cycles
L2 hit time (off-chip)	20 cycles
Main Memory (off-chip)	102 cycles

Table 3. Base (default) system parameters.

with the scaled inputs using a 32KB two-way associative L1 data cache and a 128KB four-way associative L2 data cache, which was at the knee of the working set curve. We found the difference in results from our base configuration to be negligible.

We use the RSIM simulator [23] for most of our experimental evaluation. RSIM is a user-level execution-driven simulator that models the processor and memory in detail, including contention for all resources. Operating system and I/O functionality is emulated, not simulated, so their effects are not reflected in our statistics.

For validation, we performed some experiments on a real machine. We used a Sun Microsystems Ultra 5 machine with an UltraSPARC 2i processor running at 400MHz and with 128MB of DRAM. The machine ran Solaris 7 and had no load other than background daemons and the operating system. We used perfmon, a tool that allows user-level code to access the hardware performance counters.

All applications were compiled with the SPARC SC4.2 compiler with the following options: *–xO4 –xtarget=ultra1/170 –xarch=v8plus*. The SPARC v9 ISA includes the visual instruction set (VIS) multimedia extensions. The compiler does not generate these instructions, so our base results are without these instructions. Section 3.7 describes results with VIS instructions inserted by hand.

We use a number of evaluation metrics in this study such as the range and standard deviation of execution times. These are described in the first section that uses them.

3. Results

This section analyzes the extent and the nature of the variability in the execution time of different frames in an

application. Section 3.1 determines the extent of the variability. Section 3.2 quantifies the part of the variability induced by the architecture. Section 3.3 ascertains the contribution of different types of instructions to the variability. Section 3.4 examines (local) variability in neighboring frames. Section 3.5 compares our results from simulations with results from the real machine. Section 3.6 provides the reasons for the above findings in terms of high-level application behavior. Section 3.7 describes the impact of VIS instructions.

3.1. Extent of Variability in Execution Time

We first discuss the extent of the execution time variability for different frames in an application. Figure 1 shows the execution time in cycles for each frame (referred to as the execution time profile) for all applications run on the base out-of-order architecture. For the applications with multiple frame types, G728 and MPG codecs, each frame type is displayed with a different marker. To quantify the execution time variability, Figure 2 presents the range[1] and standard deviations, both as a percentage of the mean, of execution time for each frame. For G728 and MPG codecs, the figure also shows these statistics for the individual frame types. For applications with an excessively large number of frames, profiles such as in Figure 1 show frames uniformly sampled from the entire run. However, all figures with aggregate statistics, such as Figure 2, reflect all frames.

All applications with the exception of GSMenc and GSMdec show significant execution time variability – range from 37% to 195% and standard deviation from 9% to 74%. G728 and MPG differ from the other applications in that they statically distinguish different types of frames. Examining the individual frame types in isolation, we find that G728enc shows little variability in execution time within each frame type while G728dec shows significant variability for one frame type (range of 49% and standard deviation of 16%). MPGenc and MPGdec show a significant range of execution time (15% to 43%) with the standard deviation being significant only for some of the encoder frames.

Overall, of the nine applications in our suite, five show significant variability (standard deviation of 9% or more and range more than 25%) in one or more of their component frame types. Seven of the nine applications show significant variability when all frame types are viewed as an aggregate.

3.2. Quantifying Variability Due to Architecture

Execution time variability can arise due to the architecture or a combination of the algorithm and input. We quantify the impact of the architecture in two different ways. The first method quantifies it in terms of the variability in IPC while the second method performs a comparison with simpler architectures. Both perspectives offer useful insights.

IPC vs. Instruction Count Variability.

The execution time for each frame is given by the expression $Instruction\,count \times \frac{1}{IPC} \times \frac{1}{Frequency}$. Thus, for a given clock frequency, the variability in execution time can be decomposed into the variability in dynamic instruction count and that in IPC. For a given instruction-set architecture, the instruction count depends solely on the application's algorithm and the input. The IPC depends on the algorithm, input, and the architecture. Thus, a high instruction count variability implies a high variability due to the algorithm and input while a low IPC variability implies low variability due to the architecture.

Figure 3 quantifies the variability of both instruction count and IPC by presenting their range and standard deviations (both as a percentage of the mean). It shows that for the applications and frame types that exhibit execution time variability, instruction count variability is larger than IPC variability. Thus, most of the variability in these applications arises from the algorithm and input. Overall, IPC variability is small for all applications and is negligible for most (standard deviation is 5% or less, range is less than 20% for all but two cases). Thus, the IPC for individual frame types remains roughly constant through the application run. The IPC profiles [15] (not shown here) are much flatter than the execution time (or instruction count) profiles, and the instruction count profiles [15] (also not shown) mostly follow the execution time profiles of Figure 1.

Comparison with Simpler Architectures.

We compare the execution time and IPC variability of our base architecture with that of simpler architectures. The difference in variability then is largely due to the complex architectural features in the base architecture. The simplest practical architecture chosen represents a DSP-like architecture. DSP architectures are the traditional alternative to general-purpose architectures for multimedia applications, and are also conjectured to have better execution time predictability [2, 6, 16]. Specifically, we replace four complex features of our base architecture with simpler ones to create the simplest architecture modeled: multiple-issue is replaced with single issue, dynamic branch prediction with a predict not-taken static branch prediction, caches with an infinite amount of single cycle access SRAM, and out-of-order with in-order issue. Further, in order to gauge which architectural features induce the most execution time variability, we study the range of architectures between the above and base architectures.

Figure 4 shows the standard deviation of execution time and IPC (as a percentage of the mean) for the above range of architectures. We also plot a hypothetical architecture with a perfectly predictable IPC of 1. We only show the applica-

[1] $\frac{(Maximum\ execution\ time - Minimum\ execution\ time)}{Mean\ execution\ time} \times 100$

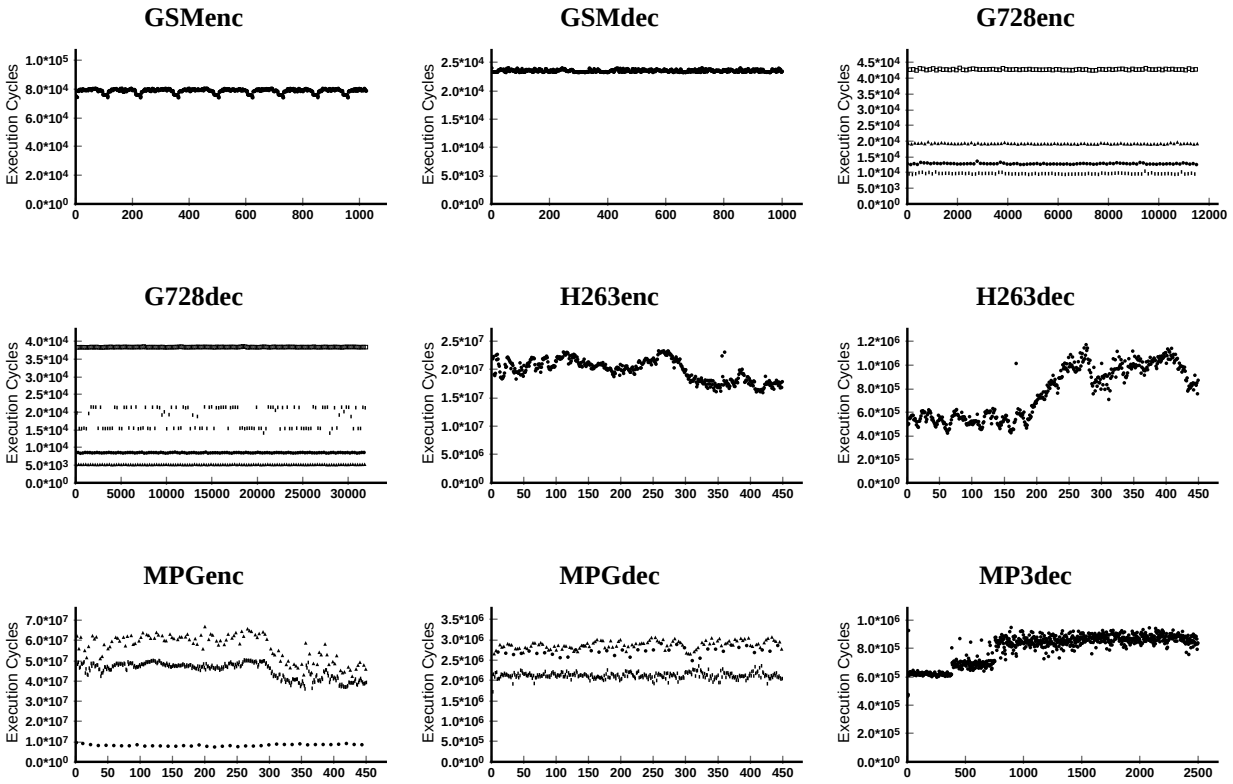

Figure 1. Execution time profiles for inputs that cause the most variability. The horizontal axis shows the frame numbers.

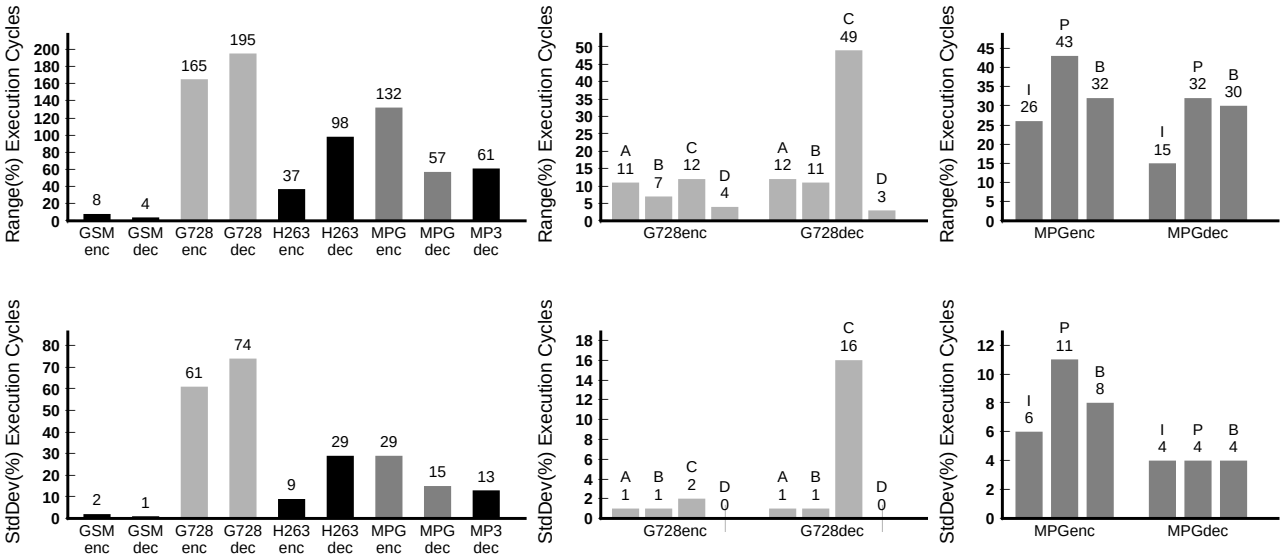

Figure 2. Execution time variability measured as range and standard deviation relative to the mean. The graphs on the right are for the individual frame types of G728 and MPG as indicated by the lighter shading. A, B, C, and D are for types 1, 2, 3, and 4 to avoid confusion.

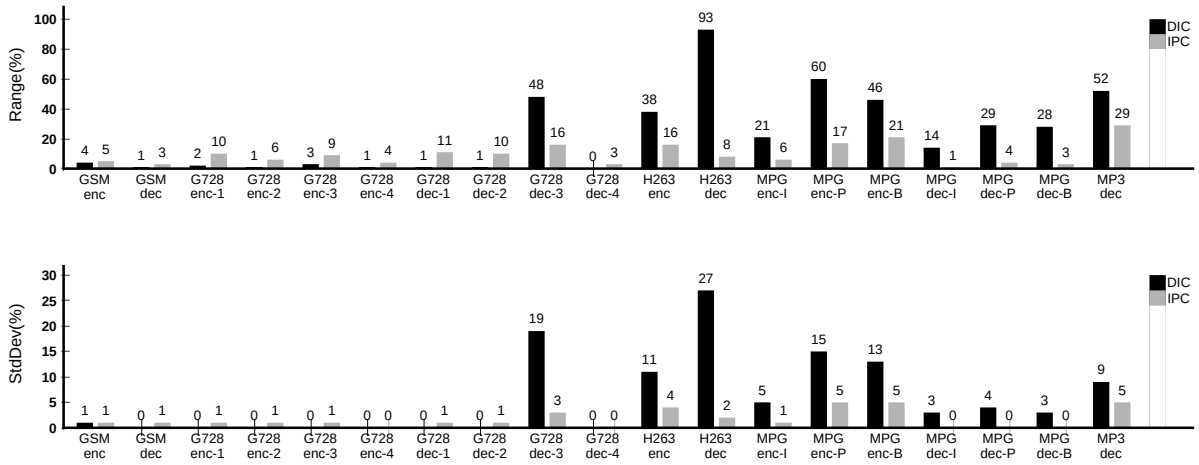

Figure 3. Dynamic instruction count (DIC) and IPC variability measured as range and standard deviation relative to the mean. For G728 and MPG, different frame types are shown separately.

tions and frame types for which there is significant variability in the base architecture.[2] The figure shows that, when significant, the standard deviations are remarkably close for the entire range of architectures for a given application. The largest change occurs for MP3dec (for reasons discussed in Section 3.6), but even there, the change is relatively small. This provides further evidence that aggressive architectural features do not significantly add to execution time variability for our application suite.

Since very little architecturally induced variability is present, it is difficult to conclude which features contribute most to it. The data for MP3dec in Figure 4 leads us to speculate that multiple-issue and out-of-order execution induce the most variability. An important note is that, contrary to the common perception, caches have negligible effect on execution time and IPC variability on these applications (for reasons discussed in subsequent sections).

3.3. Contribution of Different Instruction Components to Variability

This section explores the contribution of different instruction types to the execution time variability. Such information would be useful in ascertaining the cause of any variability (or lack thereof) and ascertaining how to adapt architectures to meet the varying requirements of the applications. We divide the instructions and execution time into different components for each frame. Figure 5 shows the percentage of ALU (integer and floating point), branch, and memory instructions in each frame. Figure 6 shows the percentage of execution time spent busy or stalled for the above instructions. Busy and stall times are calculated similar to previous work [22]. For each cycle, the ratio of instructions retired to the maximum retire rate is recorded as busy time. The remaining fraction of the cycle is charged as stall time to the first instruction in the instruction window unable to retire. Here, we present the graphs for only one application and frame type for each media type (speech, video, and audio). Where applicable, we chose the application and frame type that has the most variability.

First, we find that the distribution of the different instruction and execution time components stays roughly constant throughout the entire application. Thus, in applications that show execution time variability, the nature of the computation does not change (at the frame granularity). This explains the roughly constant IPC through most of the data.

Second, we note that little time is spent in memory stalls. Less than 10% of execution time is spent in memory stalls for all frame types of all applications. Most of the time the processor is busy or is stalled due to ALU instructions (as in [28]). This is because the codecs studied here perform significant computation per data item (as indicated by the composition of instruction count), and small caches are sufficient to hold the important working sets of these computations. As discussed in Section 2.2, we also performed some experiments on MPGenc and MPGdec with caches too small to hold the second level working set. The fraction of time spent in memory stalls increases by a small amount for all frames ($\leq$4% for MPGenc and $\leq$3% for MPGdec).

Also, although not shown here, the successful branch prediction rate and the L1 cache hit rate have little variability (standard deviations $\leq$5% and $<$1%, respectively, for all frame types for all applications).

[2] MPGenc and H263enc are run for frames 250 to 350 to save simulation time – these frames capture a part of the movie that shows high execution time variability.

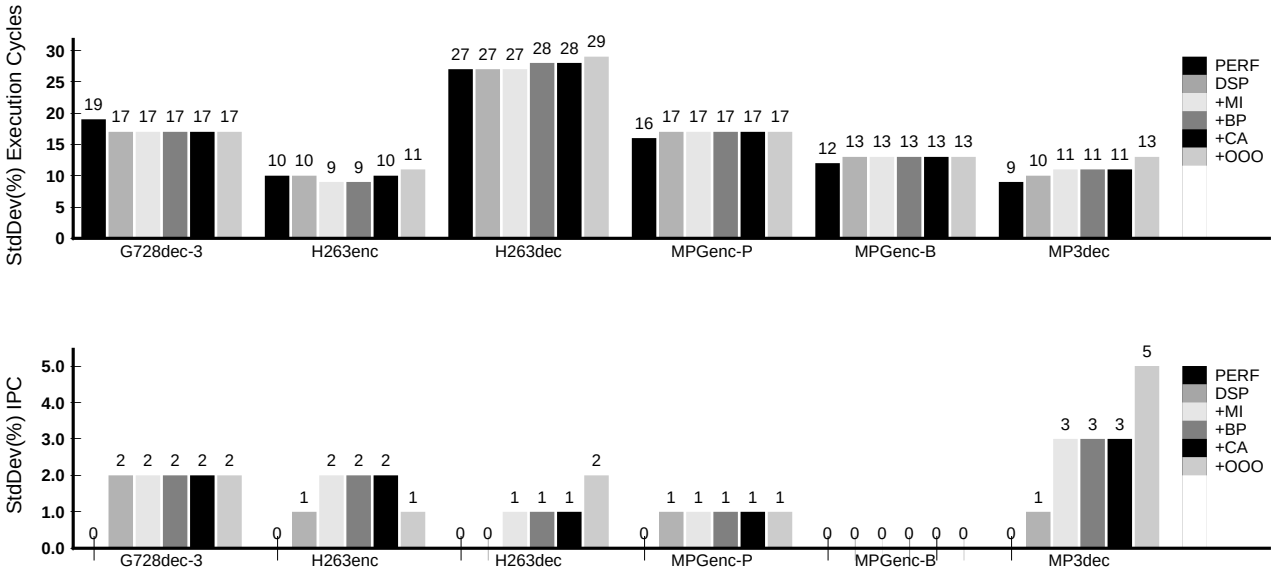

Figure 4. Comparison of base architecture with simpler architectures – standard deviation of execution time and IPC (relative to the mean). The architectures are: (PERF) Perfect (IPC=1), (DSP) DSP-like architecture (single-issue, static branch prediction with predict not-taken, DSP-like SRAM, and in-order issue), (+MI) previous with multiple issue, (+BP) previous with dynamic branch prediction, (+CA) previous with caches, (+OOO) previous with out-of-order issue.

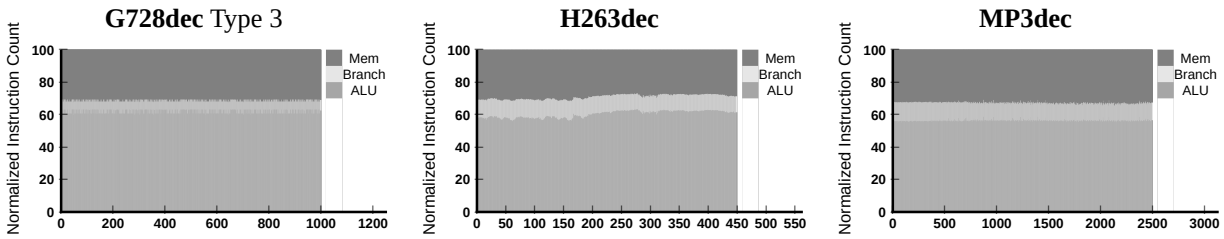

Figure 5. Components of dynamic instruction count.

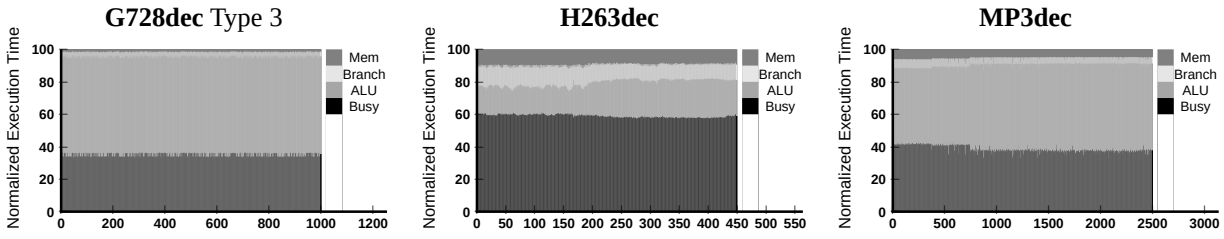

Figure 6. Components of execution time.

3.4. Local or Short-Term Variability

So far, we have focused on variability across the entire run of an application. This section analyzes short-term variability in the form of change in execution time in neighboring frames. Such an analysis can aid in designing dynamic execution time predictors for purposes such as scheduling or system adaptation (as discussed in Section 4). We plotted the change in execution time of a frame compared to the immediately preceding frame (as a percentage of the latter). For all but one frame type (G728dec Type 3), 90% of the frames saw execution times within 10% of the immediately preceding frame. The average change across all applications was 2%. Similar results hold for the dynamic instruction count. Thus, although execution time and instruction count show high variability in some applications, they vary slowly for all but one frame type.

3.5. Validation with a Real Machine

To verify our key results, we also performed some experiments using a real machine described in Section 2. We collected both per frame execution time and per frame IPC for each frame type for all applications. We compare these results to those for our in-order processor simulations (since the machine measured is in-order). The normalized standard deviations of execution time and IPC on the real machine are within 3% and 1%, respectively, of the results from simulation for all but GSMenc, G728enc, and G728dec. For the above applications, the differences are within 6% for both statistics. These applications have few instructions per frame. Therefore, file I/O may have a larger impact on the per frame execution time, resulting in higher variability. Nevertheless, the maximum standard deviation for IPC was 7%, even for these applications. The detailed results are in [15].

3.6. Relating Results to Application Behavior

This section uses high-level application behavior to explain the reasons for our findings so far. These findings are:

1. There is significant (per frame) execution time variability due to a combination of the algorithm and input for several applications.

2. The architecture induces very little variability and the IPC stays roughly constant at the frame granularity.

3. The composition of instructions and execution time stays roughly constant at the frame granularity.

4. The applications spend little time in memory stalls.

5. Execution time and instruction count change slowly from one frame to another in most cases.

The applications studied use optimizations that exploit the limitations of human perception. With media data, humans can often accept approximations without a perceptible change in quality. The applications exploit this characteristic to reduce the amount of work performed or to increase the compression rate. The use of these optimizations is by necessity input-dependent; therefore, any application using them would exhibit execution time variability with an appropriate input. This explains why several applications in our suite exhibited large execution time variability where a large part was due to the algorithm and input (finding 1).

The reason why the IPC and the composition of the instructions and execution time stay roughly constant at the frame granularity (findings 2 and 3) is as follows. The aforementioned approximations typically involve avoiding execution of a dominant piece of code for parts of the frame in favor of a simple approximation. Thus, often the variation occurs simply due to the number of times a dominant piece of code is invoked. Since the nature of the dominant computation does not change, the IPC remains constant. In some cases, the variation occurs because a different, but significant, piece of code was executed depending on the input. However, we see that the nature of computation is usually similar, and this input-induced change in IPC (and composition of execution time) is not much.

The following demonstrates the above reasoning for findings 1 through 3 more concretely for a representative application of each media type. Details on other applications are in [15]. **G728dec** frame type 3 code includes a conditional recomputation of a set of coefficients depending on whether the inputs for the frame meet a certain set of criteria, resulting in high input-induced variability. Further, since only the amount of computation changes and not its nature, the instruction and execution time composition and IPC remain stable throughout.

H263enc execution time is dominated by a technique known as motion estimation. Each frame is divided into blocks of 16x16 pixels. The encoder exploits temporal redundancy by attempting to find a match for each block in a previous frame. If a match is found, then a reference to the matching block is encoded. Otherwise, the block is run through a (relatively simple) transform computation that produces a small number of coefficients to represent the block. The amount of work in this phase is therefore input-dependent, and depends on the amount of motion in the video. For parts of the movie with a lot of motion, the motion estimator must do a lot of work and the execution time is high. If the amount of motion in a video varies, then the execution time will vary as well, again establishing the input-dependent nature of the execution time variability. Further, the amount of computation required changes, but

the nature of the dominant computation does not change; therefore, the IPC and composition of instructions and execution time remains roughly the same throughout.

MP3dec performs an optimization for audio samples that are very close to zero. It treats such samples as zeroes, avoiding executing a number of operations with the sample. This greatly reduces the number of computations performed on quiet frames. The default input is a song which starts off with a couple of instruments playing, has others come in after a delay, and finally has the remaining ones begin to play after another delay. This explains the execution time profile in Figure 1, as the steps in execution time correspond to the three phases of this portion of the song. The computation avoided for near-zero samples is less efficient than the rest of the computation, and avoiding it makes the initial part of the run see slightly higher busy time and IPC. Again, the variability is largely input-driven. Further, since the majority of the frames are not quiet, they show stable IPC behavior (and execution time composition).

The reason that the applications spend little time in memory stalls (finding 4) is that they perform significant computation and many loads per data item. Therefore, small caches are sufficient to hold the important working sets of these computations. Thus, the L1 cache hit rates are very high for all applications ($>$99%).

The reason that execution time and instruction count change slowly from one frame to another (finding 5) is that the properties of the inputs that affect these quantities tend to change smoothly rather than suddenly. For example, one such property of video inputs is the difference between consecutive frames. If the difference changes smoothly, so will the execution time and instruction count.

3.7. Impact of VIS Instructions

We hand-instrumented all applications except the G728 codecs with VIS instructions. The G728 codecs use floating-point, which is not supported by VIS. We were not able to determine if a fixed-point implementation of the G728 codecs is feasible. The GSM codecs see little benefit from VIS due to its limited (64-bit) datapath and the relatively high overhead in setting up SIMD multiplications with 32-bit products. Thus, only the video codecs and MP3dec see a significant benefit from using VIS and are discussed in more detail below.

In all cases, we applied VIS instructions to the dominant computations. For the video codecs, these are the DCT and IDCT, and also motion estimation for the encoders. For MP3dec, these are the inverse MDCT and an audio synthesizer. The IMDCT includes the optimization for near-zeroes discussed in Section 3.6. This optimization adds a control dependence and had to be removed to use VIS.

We find that execution time variability changes little in most cases with VIS, although some applications do cross our threshold of high variability (9% standard deviation) both ways. Specifically, the standard deviation of MPGenc I frames increases to 9% (from 6%), H263enc decreases to 8% (from 9%), and MP3dec decreases to 7% (from 13%). The change for MP3dec is mostly from the removal of the input-dependent optimization as discussed above. For MPGenc I frames and H263enc, the acceleration of the formerly dominant computation changed the variability. Depending upon whether the accelerated computation originally caused a little (MPGenc I frames) or a lot (H263enc) of variability, the application's variability increased or decreased, respectively. The effect also depends on an application's other critical computations. For example, H263enc has an input-dependent computation with slightly different characteristics than motion estimation. When the latter is less dominant, the effects of the former are more significant, slightly decreasing variability. Therefore, the application of VIS instructions can increase or decrease execution time variability, depending on the nature of the accelerated computation and the other important computations.

Our key result that the architecture induces little variability and IPC stays roughly constant across frames continues to hold with VIS. In all but one case, H263enc, the instruction count variability is significantly larger than the IPC variability. For H263enc, IPC variability is still small (7%) but is slightly larger than instruction count variability (5%).

Our other results on the low variability in the composition of instructions, the low time spent on memory stalls, and low local variability remain unchanged with VIS. The result on memory stalls bears further explanation. Since VIS instructions reduce computation and may require additional memory instructions to align the data, they could increase the fraction of time spent on memory stalls in some cases (seen in H263dec, MPGdec, and MP3dec). On the other hand, VIS instructions can also reduce this fraction by reducing the number of loads (seen in H263enc and MPGenc). In all cases, the changes are small and the overall fraction of time spent on memory stalls remains small ($\leq$11% for all frame types for all applications).

Overall, we find that even after applying VIS instructions, our primary findings (as enumerated in Section 3.6) continue to hold true.

4. Implications

Our results have two broad implications discussed below.

4.1. Predictability of General-Purpose Processors

Execution time predictability is an important attribute for architectures used for real-time applications. It has been

conjectured that current general-purpose processors are significantly lacking in this regard, relative to other alternatives [2, 6, 7, 8, 16].

Our results indicate that the high frame-level variability in execution time could indeed make it difficult to predict execution time on current general-purpose processors for several of our applications. The surprising result, however, is that most of this variability is due to the input and algorithm; the architecture introduces little additional variability. Therefore, predicting the execution time (in absence of knowledge of the input) would be difficult for *any* architecture. This includes a perfect hypothetical architecture with a fully predictable IPC of 1, whose execution time variability is found to be similar to that of more realistic architectures. This input-dependent nature is a result of input-specific approximations made in the application algorithms, and is likely to be a common feature of many media applications.

Hard real-time system designers typically handle input-dependent variability by making worst-case assumptions for execution time (e.g., [13, 26]). Our results on the normalized range of execution time and instruction counts show that this assumption would be quite conservative even for the perfect 1 IPC architecture. Many multimedia applications do not have absolutely hard real-time requirements. In practice, application designers and systems often use measurement to predict expected execution times (e.g., [5]). Further, as discussed in [3], many applications already make approximations and/or specify statistical error rates (e.g., wireless systems need to account for channel induced bit error rates). These soft real-time systems can afford to miss their frame processing deadlines once in a while. For such systems, it suffices to make statistical predictions that ensure that the real-time requirements would be met most of the time [3, 5]. The small architecture induced variability as seen in our results could easily be incorporated in such statistical predictions.

Thus, our results challenge the conventional wisdom that current general-purpose processors are unsuitable for real-time multimedia workloads due to their unpredictability.

4.2. Adaptive Architectures

Researchers have proposed architectural adaptivity in several forms to optimize metrics such as performance, power, and energy. Examples include speculation control [20], shutting off parts of the cache [1], and changing instruction window size [4], number of active functional units [21], and issue strategy from in-order to out-of-order [10]. Additionally, recent processors employ dynamic voltage and frequency scaling [12, 18, 24, 25, 27]. Adaptation can be particularly beneficial for multimedia applications since they require only that execution complete within a specified (often soft) deadline, and often allow trading off output quality with resource usage.

Two key questions that need to be addressed to employ adaptivity are (1) when to trigger an adaptation and (2) what adaptation to trigger (i.e., which of the possible hardware configurations to use next). Our variability analysis provides insights to address these questions.

When to adapt? The frame-level execution time variability exhibited by several of our applications implies benefits from triggering adaptation at the *frame granularity*. A full frame is a relatively long time; therefore, adaptations at the frame-granularity can tolerate relatively large overheads in switching between alternate configurations.[3]

What to adapt? To determine which hardware configuration to run for the next frame, the key ability needed is to predict the behavior of the application for the next frame. For example, an adaptive system that optimizes performance and energy must effectively predict the execution time and energy for each possible hardware configuration for the next frame. It can then select the lowest energy hardware configuration that can execute the next frame within the software-specified deadline. The following results from our variability analysis can be used to design efficient execution time predictors. We consider systems that employ both architectural adaptations such as those mentioned above and dynamic voltage/frequency scaling.

- IPC stays almost constant for each frame type at a given frequency.
- Since little time is spent in memory stalls, IPC is almost independent of processor clock frequency.
- For a given frame type, instruction count varies slowly from frame to frame.

The execution time for a frame for a particular hardware configuration (i.e., a particular architecture and voltage/frequency) depends on the IPC and the instruction count. Since IPC is roughly constant across different frames of the same type, it can be determined by measurement of a frame of that type at the beginning of the application. Since IPC is roughly independent of frequency, these measurements need be done for each architecture at only one of the possible frequencies. Since instruction count varies slowly, we can develop predictors that use the measured instruction counts for already executed frames to predict the count for the next frame. Since instruction count is independent of the hardware, measurements on any hardware suffice. A simple predictor uses the instructions of the current frame as the prediction. Using the IPC of the first frame and this simple predictor, execution time on our base hardware configuration is predicted with a mean error of 4% across all

[3] It is possible that intra-frame adaptivity is also useful, but an exploration of intra-frame variability is outside the scope of this paper.

applications and frame types (maximum is MP3dec with a mean error of 10%).

We have not performed energy or power simulations. However, since the nature of the execution is the same across all frames, it is likely that power dissipation will be similar across all frames. Thus, power for a frame could potentially be estimated by initial profiling, and frame energy could be estimated as the product of the power and predicted execution time. Using the analytic relationship between power and frequency, it would also be sufficient to profile all architectures at only one frequency. An evaluation of such an energy predictor requires detailed energy simulations, which we leave to future work.

The above prediction mechanisms can be combined to create an efficient algorithm to determine what to adapt for an architecture that optimizes performance and energy. After profiling for IPC and power in an initial phase, there is enough information for the algorithm to (1) order all hardware in increasing order of *energy per instruction* and (2) calculate the maximum instructions each hardware configuration can execute within the deadline. For the rest of the run, before executing a frame, the algorithm predicts the number of instructions in the frame and picks the configuration with the lowest energy per instruction that can also execute the predicted instructions within the deadline.

5. Related Work

We are aware of only one previous quantitative study related to the issue of unpredictability induced by aggressive features of general-purpose architectures. This study measured the performance of simple multimedia kernels (FIR and IIR filters and FFT) in the context of designing software radios on a Pentium PC [3]. It found very low execution time variability. It argued that this level of variability can be handled by communications systems which already have to deal with channel-induced errors in the incoming bit stream and can afford to use some buffering to smooth out the variability. Our conclusions are similar, but our study is much more comprehensive as it involves several full applications that do show significant execution time variability. We analyze this variability to show that most of it is input and algorithm induced and not architecture-induced. Further, we also discuss the application of our results to adaptive architectures for applications that do show variability.

Others have reported significant variability in the execution time for different frames of certain multimedia applications (e.g., [5]), but do not analyze the source of this variability or its relationship to the architecture.

There is a large body of work on statically predicting worst case execution time to determine compute requirements for real-time applications. Recent work has incorporated increasingly complex architectural features (e.g., [13, 19, 26]) as well as explored the use of measurement for such estimation [26]. The worst-case estimate is used to determine CPU requirements for the application. Using worst-case estimates, however, is inefficient if the application frequently executes a frame that is not worst-case. Our study shows that there is significant variability in execution time for many multimedia applications, and the range of execution times for many applications is quite large.

There is also a large body of work on dynamically estimating execution time of future frames of multimedia applications, both for CPU reservation (e.g., [5]) and for power and energy management (e.g., [9, 11, 12, 18, 24, 25, 27]). Govil et al. do a particularly thorough analysis of several predictors in the context of systems that dynamically scale voltage and frequency [11]. However, none of these studies examines predicting execution time for processors able to adapt their microarchitecture, or analyzes the reasons for the execution time variability.

To our knowledge, Huang et al. develop the only framework for choosing between multiple energy saving (and thermal management) techniques related to adaptive architectures and frequency scaling [14]. That work is driven by general applications and not focused on multimedia. It requires software to provide the maximum slowdown allowed; however, this is not fixed for multimedia applications with different amounts of work for each frame. Hardware to affect adaptation is invoked every few milliseconds and adjusts the hardware configuration based on measurements of IPC. Our results suggest triggering adaptations for multimedia applications at the frame granularity. At this granularity, our results indicate that the number of instructions, rather than the IPC, should be used to control adaptation, and in most cases it is sufficient to measure IPC once for each architectural configuration.

6. Conclusions

This paper analyzes execution time variability at the frame granularity for several multimedia applications running on a modern general-purpose processor. The results have two broad implications. First, they question the common perception that aggressive architectural techniques induce too much execution time unpredictability for use on multimedia workloads. Second, they suggest the use of adaptive architectures and a technique to use adaptation more effectively for multimedia applications.

More specifically, the paper shows that per frame execution time varies significantly for some multimedia applications. This variability, however, is mostly due to the input-dependent nature of the applications and is largely independent of the underlying architecture. In practice, statistical predictions already used by soft real-time system designers should be sufficient to account for the small increase in

variability from aggressive architectural features.

The presence of frame-level execution time variability for some of our applications motivates the use of adaptive architectures at a frame granularity. The findings of largely constant frame-level IPC, low memory stall time, and slow change in instruction count together suggest a technique to dynamically predict frame execution time and energy for a variety of hardware configurations. Adaptive hardware could use such predictions to choose the most energy-efficient configuration for each frame.

There are several directions for extending this work. First, the adaptation framework suggested by our analysis needs to be evaluated. Second, the interactions between such a framework and system software (e.g., the CPU scheduler) need to be explored. Third, it would be interesting to examine variability and adaptivity at a granularity finer than a frame. Finally, this study does not consider unpredictability due to the OS and I/O. Real-time systems reduce such unpredictability with several techniques (e.g., reservation based scheduling algorithms), but it is important to explore this issue in the future.

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

[3] V. Bose. *Design and Implementation of Software Radios Using a General Purpose Processor*. PhD thesis, Massachusetts Institute of Technology, 1999.

[4] A. Buyuktosunoglu et al. An Adaptive Issue Queue for Reduced Power at High Performance. In *Proc. of the Workshop on Power-Aware Computer Systems*, 2000.

[5] H.-H. Chu and K. Nahrstedt. CPU Service Classes for Multimedia Applications. In *Proceedings of IEEE Multimedia Computing and Systems*, 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] J. Eyre and J. Bier. DSP Processors Hit the Mainstream. *IEEE Computer*, pages 51–59, August 1998.

[9] J. Flinn and M. Satyanarayanan. Energy-Aware Adaptation for Mobile Applications. In *Proc. of the 18th ACM Symp. on Operating Systems Principles*, 1999.

[10] 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.

[11] 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.

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

[13] C. A. Healy et al. Bounding Pipeline and Instruction Cache Performance. *IEEE Trans. on Computers*, January 1999.

[14] 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.

[15] P. Kaul. Variability in the Execution of Multimedia Applications and Implications for Architecture. Master's thesis, University of Illinois at Urbana-Champaign, December 2000. URL: http://www.cs.uiuc.edu/rsim/Pubs/pkaulmsthesis.pdf.

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

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

[18] 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.

[19] T. Lundqvist and P. Stenstrom. Timing Anomalies in Dynamically Scheduled Microprocessors. In *Proc. of the 20th IEEE Real-Time Systems Symposium*, 1999.

[20] 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.

[21] 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.

[22] V. S. Pai, P. Ranganathan, H. Abdel-Shafi, and S. Adve. The Impact of Exploiting Instruction-Level Parallelism on Shared-Memory Multiprocessors. *IEEE Transactions on Computers, special issue on caches*, February 1999.

[23] 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.

[24] T. Pering and R. Brodersen. Energy Efficient Voltage Scheduling for Real-Time Operating Systems. In *4th IEEE Real-Time Technology and Application Symposium*, 1998.

[25] T. Pering, T. Burd, and R. Brodersen. The Simulation and Evaluation of Dynamic Voltage Scaling Algorithms. In *Proc. of Intl. Symp. on Low Power Electronics Design*, 1998.

[26] S. M. Petters and G. Farber. Making Worst Case Execution Time Analysis for Hard Real-Time Tasks on State of the Art Processors Feasible. In *Proc. of the 6th Intl. Conference on Real-Time Computing Systems and Applications*, 1999.

[27] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic Voltage Scaling on a Low-Power Microprocessor. Technical report, Delft University of Technology, 2000.

[28] P. Ranganathan, S. Adve, and N. P. Jouppi. Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions. In *Proc. of the 26th Annual Intl. Symp. on Comp. Architecture*, 1999.