VARIABILITY IN THE EXECUTION OF MULTIMEDIA APPLICATIONS AND IMPLICATIONS FOR ARCHITECTURE

BY

PRAFUL KAUL

B.Tech., Indian Institute Of Technology, Delhi, 1998

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2001

Urbana, Illinois

Abstract

Multimedia applications are an increasingly important workload for a large range of systems including handheld computers, laptops, and desktops. It is expected that general-purpose processors will be increasingly employed for such workloads; however, there are still some shortcomings that they need to overcome. One shortcoming of general-purpose architectures results from their use of complex techniques (e.g., out-of-order issue, dynamic branch prediction, and caches) for high performance. It is conjectured that these complex features result in highly unpredictable execution times, making these architectures unsuitable for meeting the real-time requirements of multimedia applications. There is, however, little quantitative data to support or disprove this conjecture.

This thesis performs a detailed quantitative analysis of execution time variability at the frame granularity for a number of multimedia applications processing speech, video, and audio data types. We find that while execution time varies from frame to frame for many applications, this variability is mostly caused by the algorithms used by the applications and the media input itself. Aggressive architectural features induce little additional variability (and unpredictability) in execution time, in contrast to conventional wisdom. Our finding of frame-level execution time variability also motivates frame-level architectural adaptivity to better meet the varying requirements of the application.

Acknowledgments

I would firstly like to thank my parents. They have always been a source of inspiration for me and the force behind all my endeavors. I am grateful to my research advisor, Sarita Adve, for giving me the opportunity to work on such an exciting research project. She gave me the freedom and guidance that helped shape this work. Finally, I would like to thank members of my research group and friends for their support. Special thanks are due to Partha, Vijay, and Chris for their contribution to this work. Their help and guidance during the course of my graduate studies has been of immense value to me.

Contents

List of Tables

Table

List of Figures

Figure

1 Introduction

As we move to the post-PC era, multimedia applications are expected to form a large part of the workload on a growing number of systems [11, 5, 13, 4]. These systems include future handheld computers, wireless telephones, and laptop computers as well as desktop systems. Several articles have argued that general-purpose processors (vs. specialized DSP processors or ASICs) will be increasingly employed for such workloads [5, 1, 4]. On the other hand, several articles have pointed out serious shortcomings of current general-purpose architectures, many of which arise due to their use of complex features (e.g., out-of-order issue, dynamic branch prediction, and large caches) to improve performance [11, 1, 5, 6, 4].

One shortcoming relates to execution time predictability. Multimedia applications periodically process a set of data, typically 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. The conventional wisdom is that complex features such as out-of-order issue, branch prediction, and caches induce significant unpredictability in the execution time, making them undesirable for multimedia applications [11, 1, 5, 6, 4]. There is, however, little quantitative data on the unpredictability induced by current architectural features. Bose has done some measurement-based studies on a Pentium workstation to quantify execution time variability, but only for simple filter (FIR and IIR) and FFT kernels [2]. He did not find much execution time variability for these kernels, and suggested a statistical real-time system for such applications. Others have reported significant variability in the execution time for different frames of certain multimedia applications (e.g., [3]), but do not analyze the source of this variability or its relationship to the architecture.

This thesis performs a detailed quantitative analysis of execution time variability at the frame granularity for a number of multimedia applications. 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 (and unpredictability) due to complex architectural features is negligible in almost all cases. This finding challenges the conventional wisdom that current general-purpose architecture features induce significant unpredictability making their use for real-time appli-

cations untenable. The presence of execution time variability also suggests a potential for architectural adaptation at a frame granularity – better performance or energy consumption may be achieved by adapting the architecture to the varying needs of the application at the frame granularity. We also perform a preliminary analysis of variability at a granularity finer than a frame.

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. 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. We next give a brief background of the applications in our benchmark suite.

2.1.1 Speech Encoding and Decoding

The first set of applications concern speech compression (encoding) and decompression (decoding). While speech codecs do not have particularly high bandwidth requirements, some important applications (such as wireless communications) require very low bit-rates. Also, speech is easily compressible due to its inherent redundancy. Speech can be compressed in a number of different ways, and a large number of speech compression standards exist.

The applications studied here use the same basic principle for speech compression. The speech signal is processed by the encoder to produce synthesis filter parameters. These compactly represent the vocal tract of the speaker. The encoder also derives a vocal tract excitation signal from the speech input. The decoder passes the excitation signal through a vocal tract model generated from the synthesis filter parameters. This produces an approximation of the original speech signal.

This technique is lossy because the model of the vocal tract and the excitation signal are approximate. This implies that the reconstructed output will not be identical to the original input. The ultimate aim of lossy coding schemes is to optimize decoded data quality for a target bit rate.

Speech data typically consists of 16-bit samples recorded at a sampling rate of 8KHz. We looked at two different speech codecs in this study, one for low bit-rates and one for higher bit-rates. These are the GSM and G.728 codecs, respectively.

The full-rate GSM speech codec is based on the European GSM (Global System for Mobile Communications) 06.10 provisional standard for use in mobile communications. It operates at

Application	Default Input	Description	Frame Size, Frame/Sample Rate
Speech Codecs			
GSMenc GSMdec	orignova homemsg	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 13Kbits/sec. Compresses frames of 160 16-bit samples into 264 bits.	20ms (160 samples), 8KHz
G728enc G728dec	lpcqutfe homemsg	High bit-rate speech coding based on the G.728 standard. It uses low-delay CELP (code excited linear prediction) coding at 16Kbits/sec. Compresses frames of five 16-bit samples into 10 bits.	625μs, (5 samples), 8KHZ
Video Codecs			
H263enc H263dec	buggy tens	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 64Kbits/sec.	40ms, 25fps
MPGenc MPGdec	buggy flwr	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 fps
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.

Media Type	Input	Description	Size in frames		Length in seconds	
Speech			GSM	G.728	GSM & G.728	
	clinton	Speech by Clinton	922	29504	18.44	
	homemsg	Recording of an answering message	1000	32000	20	
	lpcqutfe	Sentence read out by a boy	362	11572	7.23	
	orignova	Different sentences read by 8 different adults concatenated	1073	34328	21.45	
Video			H.263 & MPEG		H.263	MPEG
	buggy	Buggy race	450		18	15
	cact	Pan over still-life	450		18	15
	flwr	Drive-by of neighborhood	450		18	15
	tens	Table tennis match	450		18	15
Audio	lopez	Rock song	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.

13Kbits/sec and uses regular pulse excited (RPE) speech coding scheme. The GSM encoder splits input speech into 20ms long frames. For each frame the encoded representation contains the synthesis filter parameters and the excitation signal. A 20ms long frame containing 160 16-bit PCM samples is compressed into 264 bits after coding in the current software implementation.

G.728 was developed at the AT&T Bell Labs and standardized in 1992. It operates at 16Kbits/sec rather than 13Kbits/sec like GSM, and is based on the code excited linear prediction (CELP) principle. The higher bit-rate results in much better quality reconstructed speech for G.728. The

improved quality of reconstructed speech requires significantly higher compute power as well. G.728 finds use in applications such as video conferencing. This codec differs from the GSM codec in how it determines the excitation signal. G.728 uses frames of only 625μs, or five 16-bit PCM samples. It compresses these to ten bits in our implementation.

Our analysis uses a frame (described in Table 1) as the basic unit of work for each application. G728 statically distinguishes multiple frame types. It 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 four speech inputs we use cover a diverse background of human voices. There is a mix of adults' and childrens' voices, male and female voices, and speech transitions in a single input. The lengths of the speech inputs range from 7.23 to 21.45 seconds. These speech files have been downloaded from research group websites [20, 21].

2.1.2 Video Encoding and Decoding

The need for video compression is motivated by video processing's extraordinarily high bandwidth requirements. For example, one second of 352x240 24-bit video at 30 frames/sec is over 60Mbits of data. This makes it difficult if not impossible to store or transmit raw video data. Video compression schemes do not vary overly much in their basic principles, although a large number of standards exist (for different frame sizes and bit-rates).

Video compression schemes focus on exploiting both spatial and temporal redundancies in input data to reduce the amount of information needed to represent the original data. Spatial redundancy is exploited within a frame and temporal redundancy is exploited between frames. The compression schemes studied here are lossy in nature in order to further increase the compression ratios.

The video codecs studied here exploit spatial and temporal redundancy in the following way. Certain frames are categorized as intra-frames (I frames), and the codecs use only spatial redundancy for these. Those for which temporal redundancy is removed are known as inter-frames (P or B frames). P frames exploit temporal redundancy in a preceding I or P frame, while B frames exploit temporal redundancy in a preceding and a following I or P frame (i.e., both directions in time).

Each frame is broken into small squares or blocks. For inter-frames the encoder attempts to find a match for each square in one or more other frames (i.e., it tries to find temporal redundancy). If a match is found then a reference to the matching square is encoded. Otherwise (and for all squares

in intra-frames), the square is run through a transform where a small number of coefficients which represent the square is produced. Note that the execution time for encoding and decoding I, P, and B frames can be different due to the level of redundancy removed from the data and, specifically for the encoder, the tolerance of the inter-frame matches.

Original video input is represented as frames containing 24-bit pixels, meant to be displayed at a rate of at least 24 frames per second. We chose to examine the H.263 and MPEG-2 standards for this study. They are codecs for low bit-rate and high bit-rate video, respectively.

H.263 is a video compression standard for lower bit rate communications, and is frequently used for bit-rates less than 64Kbits per second. H.263 supports five different picture formats - SQCIF, QCIF, CIF, 4CIF, and 16CIF. H.263 supports video compression for video-conferencing and video-telephony applications. The H.263 implementation used in our study generates a single I frame at the beginning of the movie and P frames for every other frame in the movie. This simulates common H.263 applications. The number of encoded bits per frame is variable.

Motion Picture Experts Group (MPEG) is a family of audio-visual coding standards. We studied the MPEG-2 video coding standard implementation available from the MPEG software simulation group [15]. The MPEG-2 standard aims to provide video quality not lower than NTSC/PAL and up to CCIR601 quality with bit rates in the 1.5 to 6Mbits/sec range. Typical inputs used in MPEG-2 video encoding includes NTSC (704x488 frame size at 30 frames/sec), PAL (704x576 frame size at 25 frames/sec), CCIR 601(720x480 frame size at 30 frames/sec), and SIF (352x240 frame size at 30 frames/sec). The MPEG-2 encoder targets digital cable TV, digital VTR applications, and satellite and terrestrial digital broadcasting distribution. It is also frequently used for digital distribution of video over the Internet.

MPEG encoding operates on I, P, and B frames. Our implementation does so in a deterministic way, creating groups of pictures (GOPs). A GOP contains a single I frame for reference and then a regular pattern of P and B frames. The P and B frames allow for better compression, while the I frames allow a user to fast forward to certain fixed points in the movie with ease (since they can be decoded in isolation). The number of frames and sequencing of the three different frames relative to each other in one GOP can vary from one movie to the other. The number of encoded bits per frame is variable.

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 inputs for both MPG and H263 codecs.

We used four videos as inputs for the video codecs. The movie selection has been based on the content of the movie. The four input movies try to cover different real life situations with different levels of motion across inputs as well as within inputs. 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.

2.1.3 Audio Decoding

High-quality audio applications can require very high bandwidths. Storing raw digitized audio on a compact disc or DVD works well because of their high capacity. However, the amount of data per track is too large to allow easy portability. For example, without data compression 1.4Mbit of data is required for one second of stereo music in CD quality. A variety of audio codec standards exist to handle different quality and bit-rate audio data.

The standard studied here makes extensive use of the limitations of the human auditory system. Our limited audible frequency range and effects such as masking (where two simultaneous sounds close in frequency cannot be distinguished) are exploited in order to obtain large compression ratios without sacrificing quality. However, these techniques are approximate and thus the scheme is lossy.

Original digital audio signals typically consist of 16-bit samples recorded at a sampling rate in excess of twice the audio bandwidth. For example, for compact discs it is 44.1KHz. We chose to study the MP3 audio decoder due to its immense popularity.

The MPEG Audio Layer-3 (MP3) standard is defined by the Motion Picture Experts Group. MPEG Layer-3 achieves the lowest bit rate in the current MPEG audio coding family. It supports bit-rates in the 16–256Kbits/sec range. MP3 encodes and decodes frames of 1152 16-bit samples (26.1ms for 44.1KHz audio). The compression factor achieved by MP3 encoding is in the 10-12x range and can be as high as 24x.

We have used 4 MP3 encoded music files for the simulations. These represent a variety of musical genres, including pop, rock and soft rock. The simulations were run for the first 2500 frames (65.25

seconds) of each song.

2.2 Architectures Studied and Experimental Methodology

The base architecture studied is an out-of-order processor similar to the Compaq Alpha 21264, and is summarized in Table 3. The functional unit latencies were chosen to match those of the Alpha 21264 processor. All functional units are fully pipelined except for the floating point divide unit. All models use a 64KB two-way associative first-level (L1) data cache and a 1MB four-way associative second-level (L2) data cache. Both caches have twelve miss status holding registers (MSHRs) to reserve space for outstanding cache misses, and up to eight concurrent requests are allowed for the same line. 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 again the first level working set 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 with the scaled inputs using a 32KB two-way associative L1 data cache and a 128KB four-way associative L2 data cache. We found the difference in results from our base configuration to be negligible.

We use the RSIM simulator [17] for our experimental evaluation. RSIM is a user-level execution-driven simulator that models the processor and memory in great 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 was running Solaris 7 and had no load other than background daemons and the operating system. Perfmon tool is used to make the measurements on the real machine. Perfmon is a tool that allows user-level code to access the performance counters present in the Ultra-series workstations and servers produced by Sun Microsystems. This is accomplished by a loadable driver that re-programs devices with performance counters so that user-level code can access these counters.

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

Table 3 Base (default) system parameters.

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 and we did not insert any by hand. A previous study characterizes the impact of VIS for MPGenc and MPGdec (and other applications) [19]. Based on that study, we do not expect VIS to qualitatively affect application characteristics relevant to this study. (For example, the study shows that the memory stall time remains small for the MPG codes with and without VIS, and the difference between in-order and out-of-order processors is qualitatively similar for both cases.)

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

3 Extent and Nature of Execution Time Variability

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 presents the measurement results on a real machine and compares them with the simulator results. Section 3.5 provides the reasons for the above findings in terms of high-level application behavior. Section 3.6 summarizes the implications of the above findings. Section 3.7 presents results for variability at a granularity finer than a frame.

We only report results from the applications run with the default input (the input with the maximum standard deviation for execution time). In most cases, the results with the other inputs are remarkably similar both qualitatively and quantitatively. We report any important differences in behavior when the applications are run with the non-default inputs.

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 relative to the mean, of execution time for each frame. For G728 and MPG codecs, the figure also shows these statistics for the individual frame types (shown as type A, B, C, and D corresponding to types 1, 2, 3, and 4 respectively for G728 and type I, P, and B for MPG). For applications with an excessively large number of frames, profiles such as in Figure 1 show frames uniformly sampled from the entire run. All figures with aggregate statistics such as Figure 2, however, reflect all frames. The numbers in Figure 2 are representative of all inputs except in the case of H.263. For H.263 encoder the default input's range is almost twice as large as the other inputs and its standard deviation is more than twice as large as the rest of the inputs. For H.263 decoder the default input's range is close to the other inputs, but its standard deviation is almost twice as large as the other inputs.

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

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

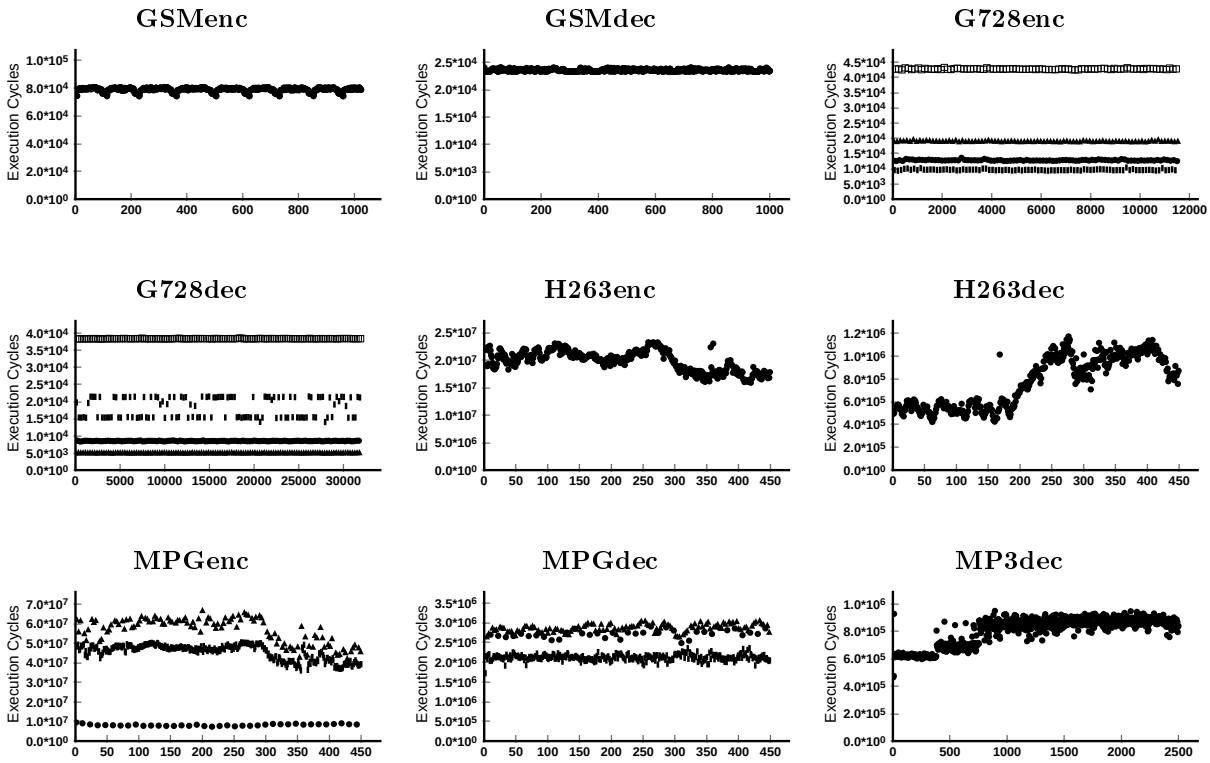

Figure 1 Execution time profiles for inputs that cause the most variability.

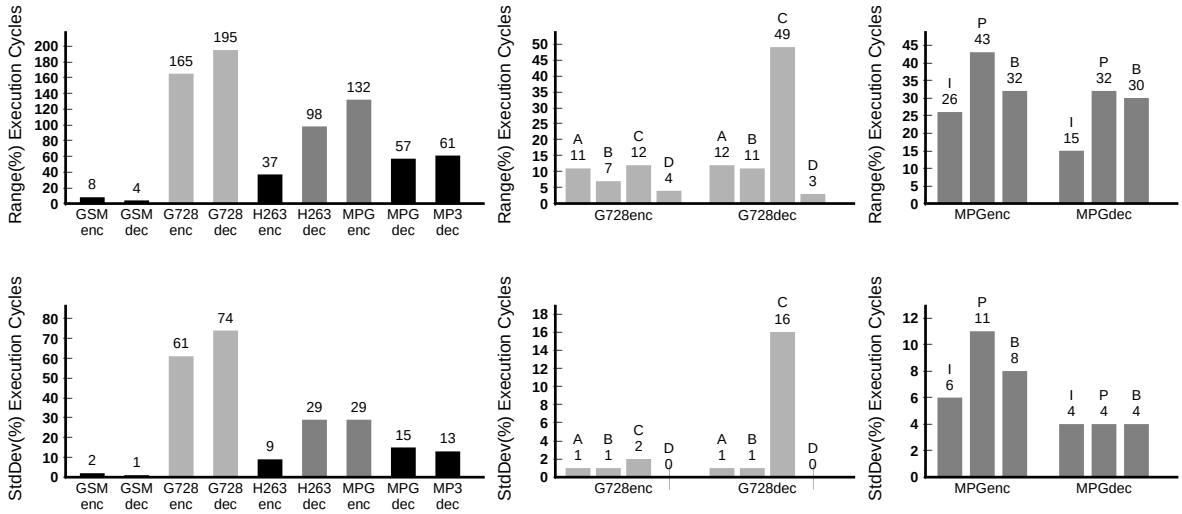

Figure 2 Execution time variability measured as range and standard deviation

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 and combination of the algorithm and input. The execution time variability numbers reported in Section 3.1 do not reflect the contribution of each of these two factors individually to the variability. We present two methods to quantify the impact of the architecture. 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 dynamic instruction count depends solely on the application's algorithm and the input. Variability due to algorithm and the input alone can therefore be measured as variability in dynamic instruction count. The IPC depends on the algorithm, input, and the microarchitecture. 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 shows the dynamic instruction count profile for each frame analogous to the execution time profiles in Figure 1. The instruction count profile for each application matches the corresponding execution time profile in Figure 1 very closely.

Figure 4 shows the profiles for the mean IPC for each frame. The IPC profiles are much flatter than the execution time (or instruction count) profiles. The IPC across similar frame types appears relatively stable for all the applications.

Figure 5 quantifies the variability of both instruction count and IPC by presenting their range

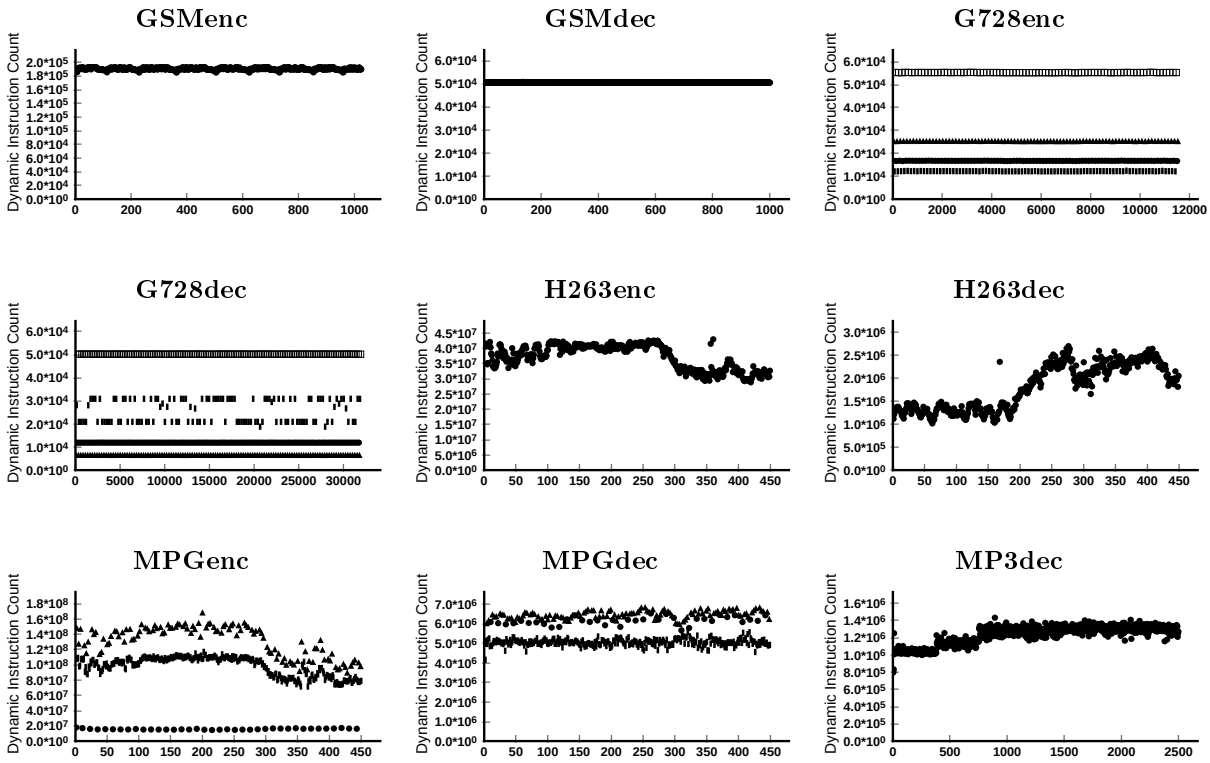

Figure 3 Dynamic instruction count profiles for inputs that cause the most variability.

and standard deviations (both relative to the mean). For applications showing significant execution time variability, the instruction count range for each application is between 14% to 93% and standard deviation is from 3% to 27%. 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.

Overall, Figure 5 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.

Comparison with Simpler Architectures.

We compare the execution time and IPC variability of our base architecture with an ideal architecture and simpler architectures. An ideal architecture is defined as the architecture that adds no variability to the execution time. An architecture with constant IPC of 1 qualifies as an ideal architecture. The execution time in this case is equal to the number of instructions and its variabil-

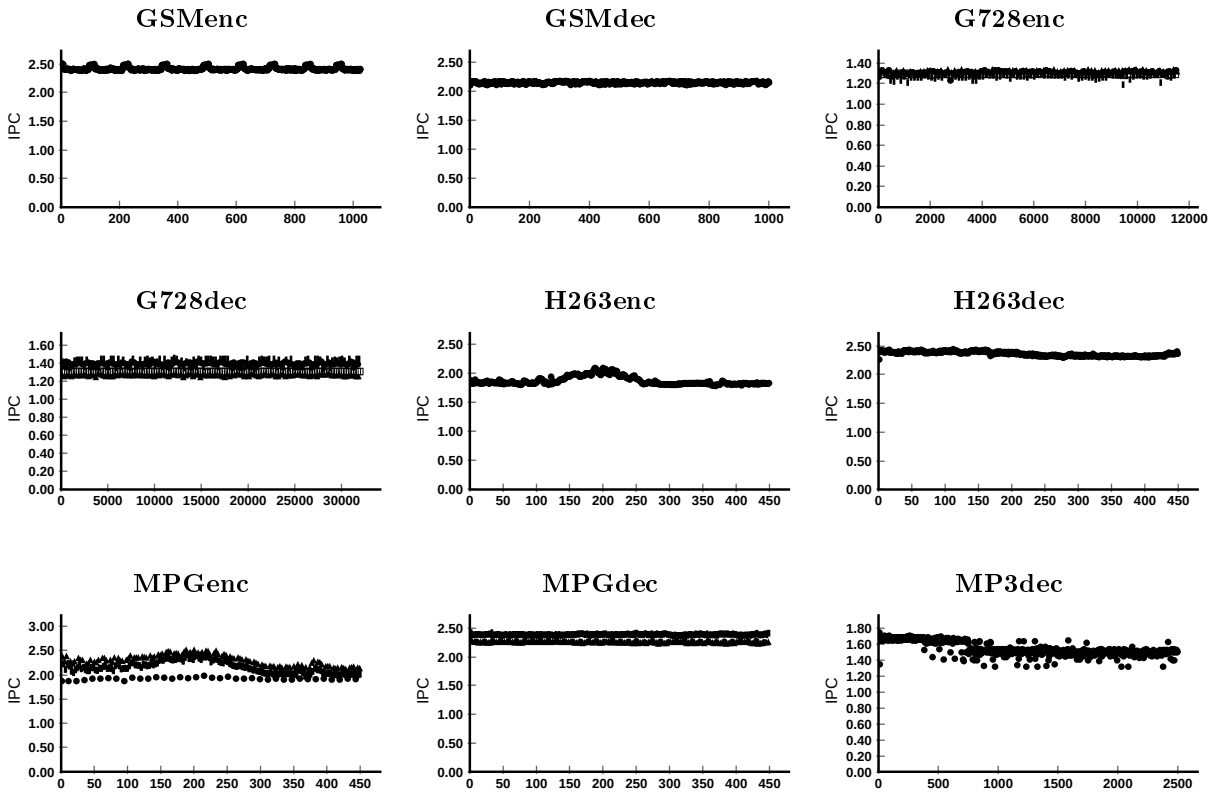

Figure 4 IPC profiles for inputs that cause the most variability.

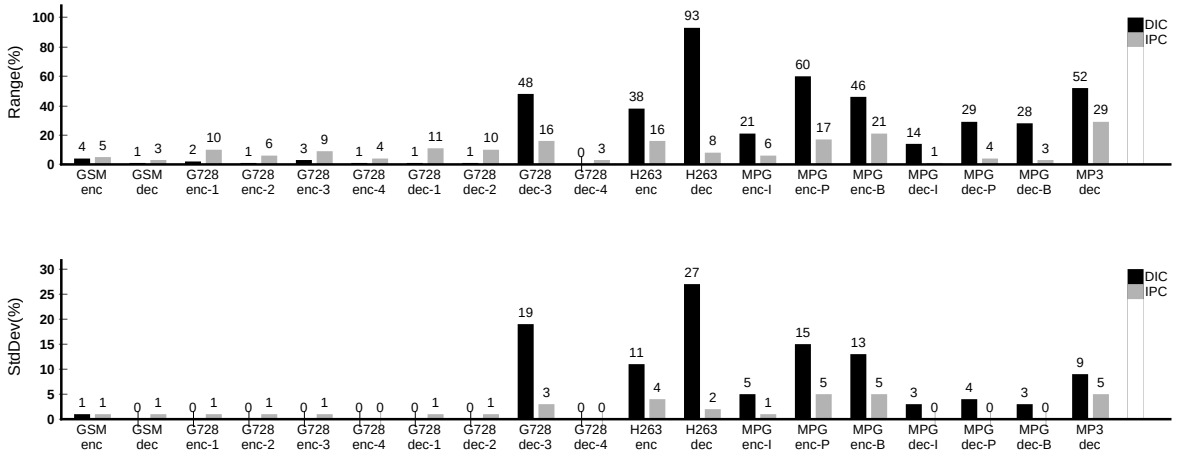

Figure 5 Dynamic instruction count and IPC variability.

ity is the same as the instruction count variability. While ideal architecture is a hypothetical case, simpler architectures are based on realistic architecture implementations.

In comparing variability between base architecture and simpler architectures, the difference in variability is largely due to the complex architectural features in the base architecture. The simplest 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 [1, 4, 11]. 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 (similar to that in most DSPs), 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 in between the simple one and our base architecture.

Figure 6 shows the standard deviation of execution time and IPC (relative to the mean) for the above range of architectures. [2] The figure shows that, when significant, the standard deviations in IPC are remarkably close for the entire range of architectures for a given application. The largest change occurs for MP3dec (for reasons discussed in Section 3.5), but even there, the change is relatively small. This provides further evidence that aggressive architectural features do not 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 6 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 the variability 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 7 shows the percentage of ALU

[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.

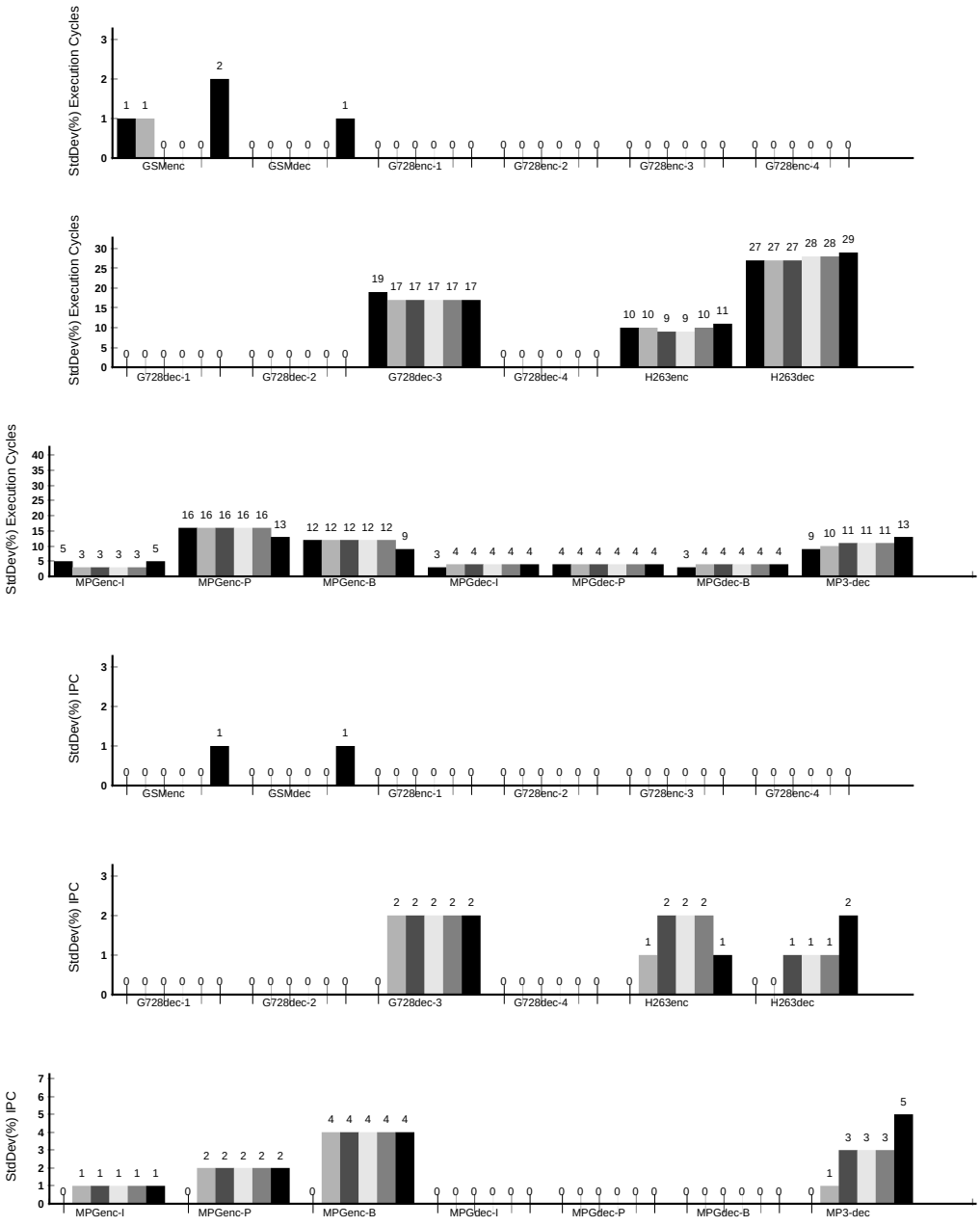

Figure 6 Comparison of base architecture with simpler architectures – standard deviation of execution time and IPC (relative to the mean). The architectures are (from left to right): (1) ideal (IPC=1) (2) simple DSP-like architecture (single-issue, static branch prediction with predict not-taken, DSP-like SRAM, and in-order issue), (3) previous with multiple issue, (4) previous with dynamic branch prediction, (5) previous with caches, (6) previous with out-of-order issue.

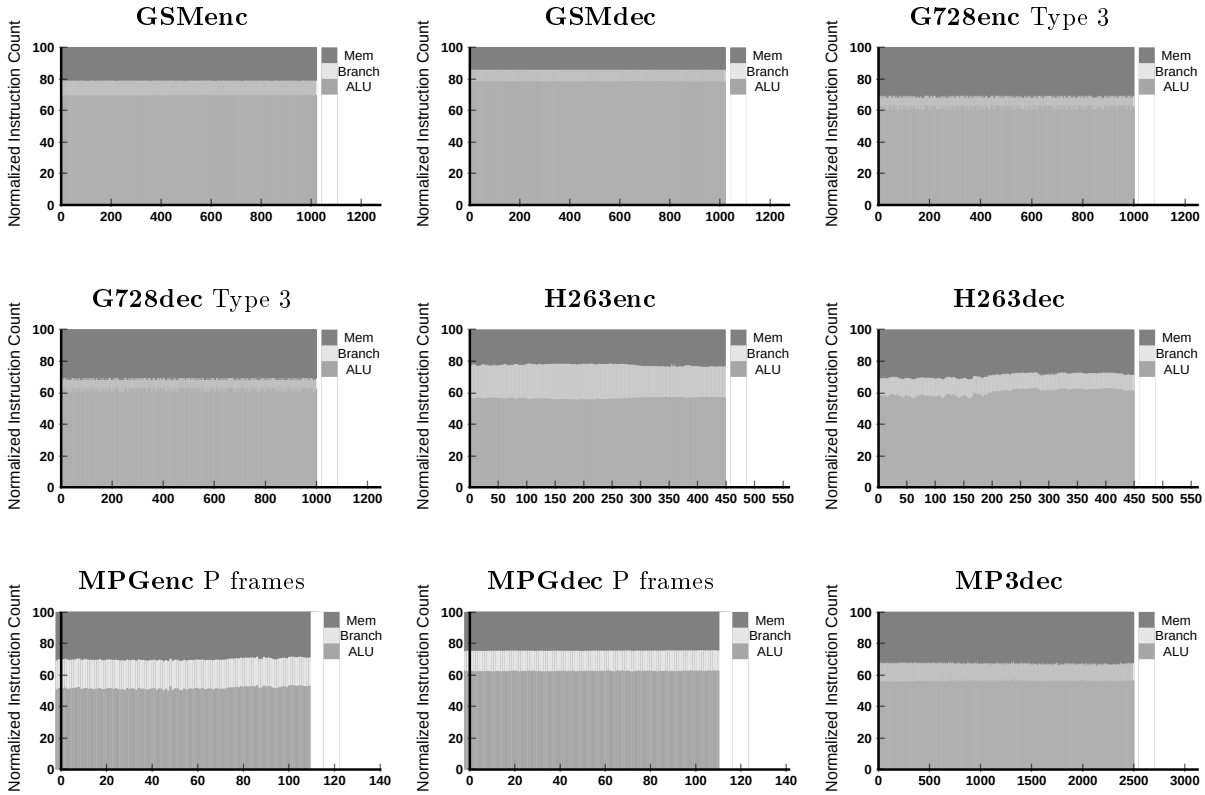

Figure 7 Components of dynamic instruction count.

(integer and floating point), branch, and memory instructions in each frame. Figure 8 shows the percentage of execution time spent busy or stalled for ALU (integer and floating point), branch, and memory instructions.[3] Here, we present one graph for applications with multiple frame types. The frame type showing most variability in execution time is chosen.

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. Some fluctuations in the instruction count and execution time composition are present. In particular, G.728 encoder and decoder, H.263 encoder and decoder, MPEG encoder, and MP3 decoder all exhibit some non-negligible (but small)

[3]Busy and stall times are calculated similar to previous work [16]. 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.

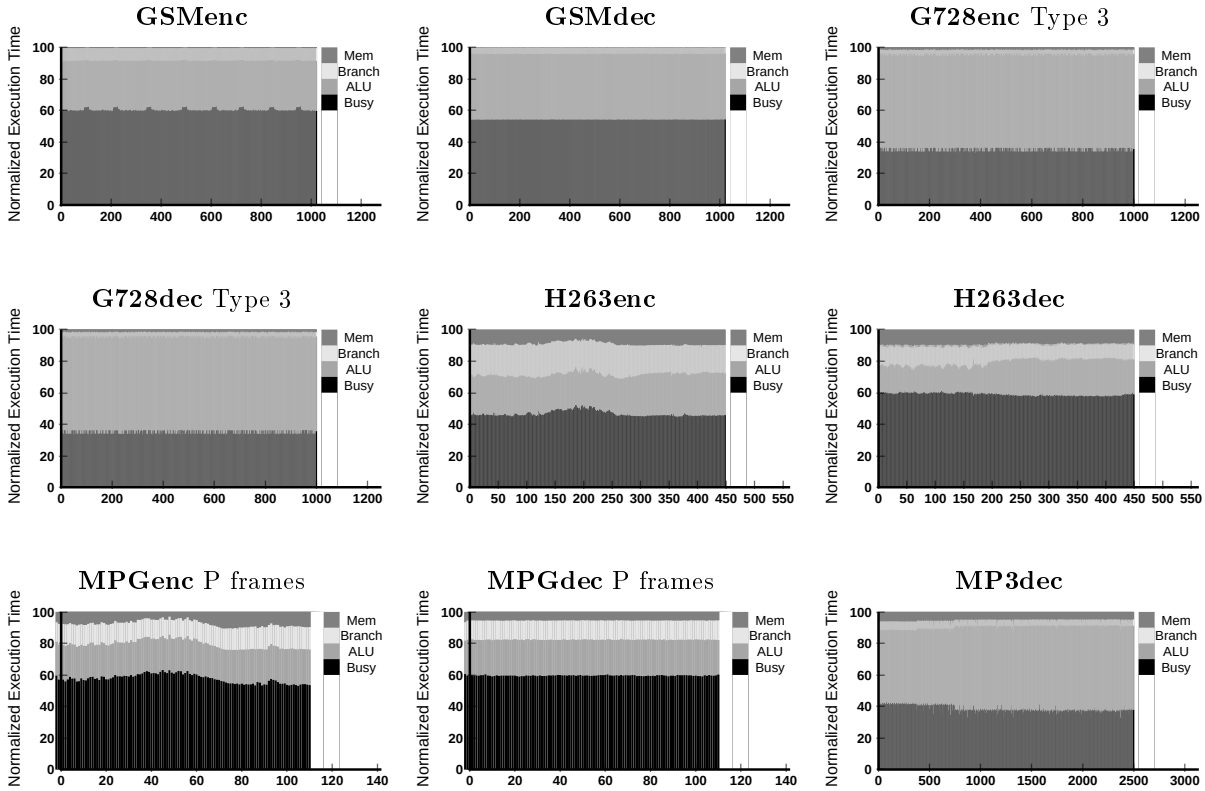

Figure 8 Components of execution time.

changes. These are the applications that have variability. The constant IPC effect (and the reasons for the small changes seen in some applications) are discussed in Section 3.5.

Second, we note that very little time is spent in memory stalls. Most of the time the processor is busy or is stalled due to ALU instructions (as also reported in other work characterizing media applications [19]). 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.

3.4 Comparing Real Machine and Simulator Results

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 all applications. We compare these results to the results we obtained for our in-order processor simulations (since the

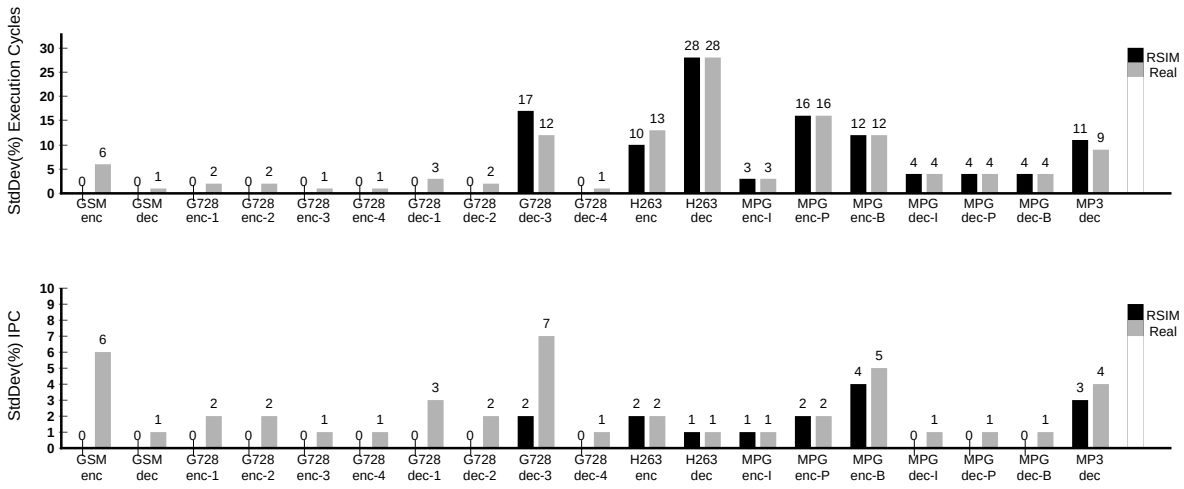

Figure 9 Comparison between measured variability on real machine and simulator.

machine measured is in-order) in Figure 9.

We see that for all applications, the normalized standard deviations for execution time and IPC measured on the real machine are almost similar to those measured from the simulator. The results match perfectly for applications spending a large number of execution cycles per frame (MPG, H263, MP3dec). However, there is some difference between the results on the real machine and the simulator for GSM and G728 – these applications have a small execution time per frame. In particular, the variability estimate is higher for these applications. This could be due to higher impact of file I/O on execution time per frame for such applications which can result in higher variability.

3.5 Using Application Behavior to Explain Findings

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

- There is significant (per frame) execution time variability due to a combination of the algorithm and input for several applications.
- The architecture induces very little variability and the IPC stays roughly constant at the frame granularity.

- The composition of instructions and execution time stays roughly constant at the frame granularity.
- The applications spend very little time in memory stalls.

Different media types – speech, video, and audio – have excessively large storage and communication bandwidth requirements (for speech, the bandwidth requirements are relatively high for the wireless communication domain). The encoder applications transform the original data into an alternate representation for compression. The basic principle behind compression schemes is to either exploit limitations of human perception (used in GSM, G728, MP3) or redundancy in original representation or a combination of both (H263, MPEG). With media data, humans can often accept approximations without a perceptible change in quality. The use of these compression principles is by necessity input-dependent; therefore, any application using them would exhibit execution time variability depending on the input. Further, the applications can also control the amount of compression depending on the target bit rate parameter which can also impact the execution time. This explains why several applications in our suite exhibited large execution time variability where a large part was due to the algorithm and input.

IPC stays roughly constant at the frame granularity because of the similar composition of the instructions across different frames. The reason behind similar instruction breakdown for applications with no execution time variability is because the same code gets executed for each frame. The reason for applications with execution time variability 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. Since the nature of the dominant computation does not change even if the amount changes, the IPC remains constant. The small variability seen in some applications is because a different, but significant, piece of code was executed depending on the input. However, the input-induced change in IPC (and composition of execution time) is not much. We chose variability in IPC as a metric to measure architecture-induced variability to execution time. Since IPC variability is negligible, we infer that complex architectural features do not contribute to execution time variability.

The reason that the applications spend little time in memory stalls is because they perform significant computation per data item and small caches are sufficient to hold the important working sets of these computations. Thus, the L1 cache hit rates are very high for all these applications.

The following demonstrates the above reasoning more concretely for all applications in our benchmark suite.

Speech applications

GSM and G728 codecs show no execution time variability with the exception of G728dec frame type 3. GSM and G728 applications exploit limitations of human perception by approximately modelling the vocal tract for compression. They do not exploit any inherent redundancies in original input data. We ran simulations with speech input in which all samples were the same. The execution time measured is the same as the execution time for speech inputs with significant variability. The nature of the compression algorithm therefore results in input-independent execution time. Absence of execution time variability explains the constant IPC and instruction breakdown across different frames.

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. This explains the bimodal nature of execution times of type 3 frames (Figure 1) and the high variability in execution time and instruction count. It validates that this variability is largely due to the input. Further, since only the amount of computation changes and not its nature, the instruction and execution time composition and IPC remain stable throughout.

Video applications

H263 and MPEG video codecs exploit a combination human perception limitation and redundancy compression principle. Both inter and intra frame redundancies are exploited for compression (discussed in Section 2.1). The amount of redundancy in input has significant impact on execution time. This reflects in the high execution time variability seen for video applications.

H263enc and MPGenc execution time is dominated by a technique known as motion estimation. Each frame is divided into blocks of 8x8 or 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 again, 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.

The execution time variability seen in MPGdec and H263dec is also linked to the amount of motion in the movie. The arguments presented for video encoders apply for the video decoders as well. H263dec shows significantly higher variability compared to MPGdec. This can be attributed to the different implementation of the two decoders. H263 is a lower bit rate video codec. For higher compression ratios, no information is transmitted for macroblocks that find a match in the reference frame (a match does not mean all the pixels are identical). Inverse discrete cosine transform (iDCT) is not invoked for such macroblocks. Since iDCT forms the major component of execution time in the decoders, the amount of execution time variability is almost same as variability in iDCT. MPGdec transmits the difference between the macroblock and the reference even if a match is found. The iDCT is therefore always invoked. This explains the difference in variability between MPGdec and H263dec.

Audio application

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, 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).

3.6 Implications

The findings of this section have at least two broad implications. The first concerns execution time predictability with general-purpose architectures and the second concerns the use of adaptive architectures for multimedia applications.

Predictability.

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 [11, 1, 5, 6, 4].

Our results indicate that the high frame-level variability in execution time may 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; therefore, predicting the execution time (in absence of knowledge of the input) would be difficult for *any* architecture. Specifically, consider the hypothetical ideal architecture with a predictable IPC of 1 from Section 3.2, which induces no architectural unpredictability. The variability seen is similar to the execution time variability of the more realistic architectures. This high input-dependent nature is a result of input-specific approximations made in the application algorithm, 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., [9, 18]). Our results on the normalized range of instruction counts show that this assumption would already be quite conservative even for the ideal 1 IPC architecture. However, many multimedia applications do not have absolutely hard real-time requirements. In practice, application designers often use measurement to predict expected execution times. Further, 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, 2]. Our results show that the additional execution time variability induced by complex architectures is low enough to be easily incorporated in such statistical predictions.

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

Adaptive architectures.

The frame-level execution time variability exhibited by several of our applications implies that better performance or energy consumption may be achieved by adapting the architecture to the varying needs of the application at the frame granularity. A full frame is a relatively long time (625μs to 40ms for our applications); therefore, adaptations at the frame-granularity can tolerate relatively large overheads in switching between alternate configurations. For example, the overhead involved in scaling frequency and voltage is about 20 microseconds in Transmeta's Crusoe processor [8], but would be acceptable at the frame granularity for most applications. The findings of this thesis are used to develop an adaptation framework in [10].

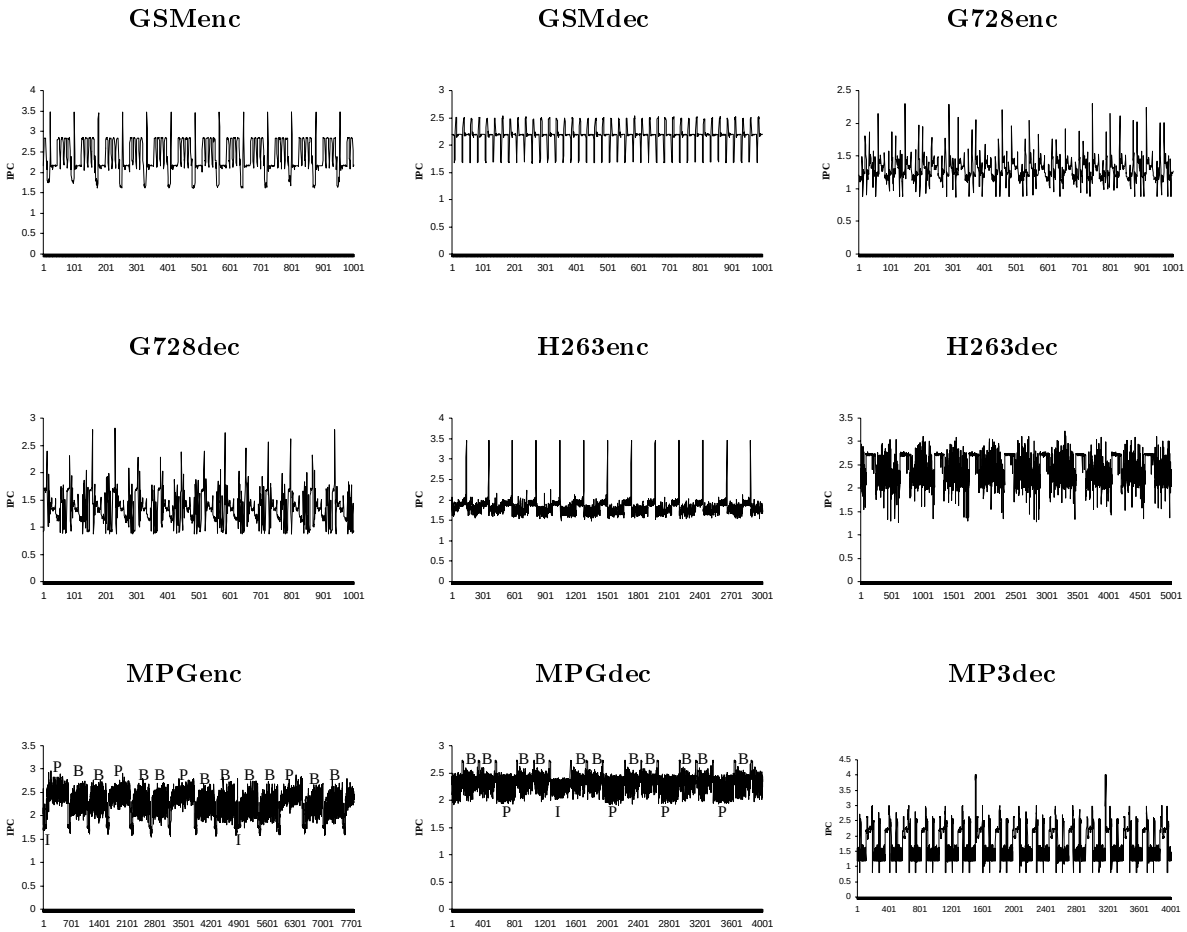

Figure 10 Finer granularity IPC profiles.

It is possible that our applications also exhibit variability in execution time, instruction count, and IPC within each frame. Such finer-grain adaptivity could potentially also be helpful for energy efficiency and is discussed in next section.

3.7 Finer Granularity Analysis

Execution time variability results and their implication on architecture have been discussed in the thesis at a frame level. We also did some experiments to collect and analyze data at a finer granularity. We chose to do a preliminary analysis at 1000 cycle granularity for all applications except for MPGenc and H263enc which are evaluated at 100,000 cycle granularity.

Figure 10 presents the IPC profiles for all applications. For MPGenc and MPGdec, the sections of profile corresponding to different frame types have been marked with a label. This could not be done for G728 because of smaller number of 1000 cycle phases per frame. From the profiles, we find there is significant variability in IPC within a frame. This is in contrast to the relatively constant IPC's seen at the frame level. The range of IPC is between 25% and 100%. The IPC jumps are not gradual but instant. A key observation that can be made from these profiles is that the IPC profile appears to have a periodic behavior for all applications (except for G728). This periodicity is observed at a frame granularity. IPC profile for a particular frame type appears to be repeating in the subsequent frames of the same type. The periodic behavior of IPC profile can be exploited in adaptive systems for speed and energy benefits.

4 Related Work

Many characterizations of multimedia applications exist, such as [12, 7, 19]. However, ours is the first study to quantify execution time variability or predictability for a wide range of media processing applications.

Several articles have qualitatively conjectured about the unpredictability induced by general-purpose architectures [11, 1, 5, 6, 4]. They examine complex IPC enhancing techniques such as out-of-order execution, caches, branch prediction, and speculative execution. However, none of these does an indepth analysis on this topic, nor do they present a quantitative analysis of unpredictability on general-purpose processors and the reasons behind it.

As mentioned in Section 1, 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 [2]. 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.

There is a large body of work on statically predicting worst case execution time to determine the compute requirements for real-time applications. Recent work in this area has incorporated increasingly complex architectural features (e.g., [9, 18, 14]) as well as explored the use of measurement for such estimation [18]. 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.

5 Conclusions

General-purpose processors are expected to be widely used for multimedia workloads. However, one serious perceived concern about the suitability of current architectures for this purpose is regarding their unpredictability. It is conjectured that complex performance-enhancing features used by current architectures induce significant execution time unpredictability, making them unsuitable for the realtime nature of multimedia applications.

This thesis performs the first detailed quantitative analysis of execution time variability at the frame granularity for a number of multimedia applications processing speech, video, and audio data types. We find that while execution time varies from frame to frame for many applications, this variability is mostly caused by the algorithm and input and is therefore also present in simpler architectures. Aggressive architectural features induce little additional variability (and unpredictability) in execution time, in contrast to widespread perception. System designers of real-time systems with absolutely hard deadlines deal with input-dependent variability by making worst-case execution time assumptions. However, these assumptions are often overly conservative and many multimedia applications do not have hard real-time requirements and/or already deal with approximations and data errors at the application level. Such soft real-time applications can afford to miss their deadlines once in a while and it suffices to make statistical predictions that ensure that the real-time requirements would be met most of the time. In practice, system designers often use measurement to determine statistical behavior of execution time and use this to ensure that realtime requirements are met most of the time. Our results show that the additional execution time variability induced by complex architectures is low enough to be easily incorporated in such statistical predictions for such systems. The presence of frame-level execution time variability also motivates frame-level architectural adaptivity.

There are several directions in which we can extend this work. First, we can use our analysis to develop an adaptivity framework to dynamically choose an energy-efficient architecture [10]. Second, it will be interesting to further explore variability and adaptivity at a granularity finer than the frame level. Third, in this work, we did not study the impact of the operating system and I/O. Realtime systems employ several techniques to reduce the unpredictability due to these aspects (e.g., reservation based scheduling algorithms) and a comprehensive study was outside the scope of this work – it is important to explore this issue in the future.

References

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

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

[3] Hao Hua Chu and Klara Nahrstedt. CPU Service Classes for Multimedia Applications. In *Proceedings of IEEE Multimedia Computing and Systems*, 1999.

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

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

[6] Jennifer Eyre and Jeff Bier. Dsp processors hit the mainstream. *IEEE Computer*, pages 51–59, August 1998.

[7] Jason Fritts, Wayne Wolf, and Bede Liu. Understanding Multimedia Application Characteristics for Designing Programmable Media Processors. In *SPIE Photonics West, Media Processors '99*, 1999.

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

[9] C. A. Healy et al. Bounding pipeline and instruction cache performance. *IEEE Trans. on Computers*, January 1999.

[10] Chris Hughes et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. Submitted for publication.

[11] Christoforos E. Kozyrakis and David Patterson. A New Direction for Computer Architecture Research. *IEEE Computer*, November 1998.

[12] Chunho Lee, Miodrag Potkonjak, and W.H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimeddia and Communications Systems. In *Proc. of the 29th Annual Intl. Symp. on Microarchitecture*, 1997.

[13] Ruby B. Lee and Michael D. Smith. Media Processing: A New Design Target. *IEEE Micro*, August 1996.

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

[15] MPEG Software Simulation Group. URL: http://www.mpeg.org/MPEG/MSSG/.

[16] 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.

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

[18] Stefan M. Petters and Georg 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.

[19] Parthasarathy Ranganathan, Sarita Adve, and Norman 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.

[20] MediaBench Home. URL: http://www.cs.ucla.edu/ leec/mediabench/.

[21] ELDA: Speech Resources Specifications. URL: http://www.elda.fr/cata/spee_det.html.