

Copyright
by
Deependra Talla
2001

The Dissertation Committee for Deependra Talla
Certifies that this is the approved version of the following dissertation:

**Architectural Techniques to Accelerate Multimedia
Applications on General-Purpose Processors**

Committee:

Lizy K. John, Supervisor

James O. Bondi

Brian L. Evans

Ching-Yu Hung

Stephen W. Keckler

Earl E. Swartzlander, Jr.

**Architectural Techniques to Accelerate Multimedia
Applications on General-Purpose Processors**

by

Deependra Talla, B.E., M.S.E.E.

Dissertation

Presented to the Faculty of the Graduate School of

the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Doctor of Philosophy

The University of Texas at Austin

August 2001

This thesis is dedicated to my family and friends.

Acknowledgments

First, I would like to thank my advisor, Prof. Lizy John for her support, advice, guidance, and good wishes. Lizy has had a profound influence not only as my graduate advisor in Austin, but also on my life. Her availability at all times including weekends, dedication towards work and family, professional integrity, and pursuit of perfection helped me become a better individual. I am grateful to her for the freedom and flexibility she gave me during the three years I spent in Austin.

My gratitude goes to the committee members (in alphabetical order), Dr. James Bondi, Prof. Brian Evans, Dr. Ching-Yu Hung, Prof. Steve Keckler, and Prof. Earl Swartzlander, for their helpful comments, productive suggestions, and the time for reading the draft of my thesis. The mid-term evaluation helped me immensely to finish this work.

I would like to thank the students (past and current) at the Laboratory for Computer Architecture (LCA) – Ramesh, Srivats, Ravi, Juan, Tao, Madhavi, Jyotsna, Purnima, Vikram, Jason, Hrishi, Poorva, Lance, Shiwen, Yue, Byeong, Pattabi, and Anand. Juan Rubio was instrumental in jump-starting my research

by providing performance monitoring tools. Ravi Bhargava passed on valuable comments on drafts of my paper submissions during all stages of my research. Lance Karm spent considerable amount of time reading and correcting multiple drafts of my dissertation.

I would like to thank Dr. Raj Talluri for providing me with an opportunity to work with the digital still camera team at Texas Instruments, Dallas. I gained valuable experience working with the digital still camera team.

Thanks to Linda, Shirley, Debi, Melanie, and other administrative assistants who worked in Computer Engineering in the past 3 years.

Finally, I would like to thank my parents, sister, and friends who have had a tremendous influence on my life.

Architectural Techniques to Accelerate Multimedia Applications on General-Purpose Processors

Publication No. _____

Deependra Talla, Ph.D.

The University of Texas at Austin, 2001

Supervisor: Lizy Kurian John

General-purpose processors (GPPs) have been augmented with multimedia extensions to improve performance on multimedia-rich workloads. These extensions operate in a single instruction multiple data (SIMD) fashion to extract data level parallelism in multimedia and digital signal processing (DSP) applications. This dissertation consists of a comprehensive evaluation of the execution characteristics of multimedia applications on SIMD enhanced GPPs, detection of bottlenecks in the execution of multimedia applications on SIMD enhanced GPPs, and the design and implementation of architectural techniques

to eliminate and alleviate the impact of the various bottlenecks to accelerate multimedia applications.

This dissertation identifies several bottlenecks in the processing of SIMD enhanced multimedia and DSP applications on GPPs. It is found that approximately 75-85% of instructions in the dynamic instruction stream of media workloads are not performing useful computations but merely supporting the useful computations by performing address generation, address transformation/data reorganization, loads/stores, and loop branches. This leads to an underutilization of the SIMD computation units with only 1-12% of the peak SIMD throughput being achieved.

This dissertation proposes the use of hardware support to efficiently execute the overhead/supporting instructions by overlapping them with the useful computation instructions. A 2-way GPP with SIMD extensions augmented with the proposed MediaBreeze hardware significantly outperforms a 16-way SIMD GPP without MediaBreeze hardware on multimedia kernels. On multimedia applications, a 2-/4-way SIMD GPP augmented with MediaBreeze hardware is superior to a 4-/8-way SIMD GPP without MediaBreeze hardware. The performance improvements are achieved at an area cost that is less than 0.3% of current GPPs and power consumption that is less than 1% of the total processor power without elongating the critical path of the processor.

Table of Contents

List of Tables	xii
List of Figures	xiv
Chapter 1. Introduction	1
1.1 Approaches for multimedia processing	1
1.1.1 General-purpose processors with SIMD extensions	2
1.1.2 VLIW architectures for multimedia processing	8
1.1.3 ASICs	12
1.2 The problem	13
1.3 Objectives	14
1.4 Thesis statement	15
1.5 Contributions	15
1.6 Organization	17
1.7 Acronyms	19
Chapter 2. Related Work	22
2.1 Multimedia application characteristics	22
2.2 Benefits of SIMD extensions	26
2.3 Media architectural enhancements	28
2.4 Related GPP architectures	30
Chapter 3. Performance Evaluation Methodology	32
3.1 Tools	32
3.1.1 Performance monitoring counters.....	32
3.1.2 VTune	33
3.1.3 SimpleScalar simulator.....	34
3.1.4 TMS320C6x simulator	34
3.2 Benchmarks	35

3.3	Evaluation Environment	38
3.3.1	Compilers.....	39
3.3.2	Assembly libraries	39
3.3.3	Compiler intrinsics.....	41
3.4	Performance Measures.....	42
Chapter 4.	Execution Characteristics of Multimedia	
	Applications on GPPs	43
4.1	Detailed characterization of multimedia applications	44
4.1.1	Cycles per instruction	44
4.1.2	Resource and instruction-stream stalls	45
4.1.3	Branch statistics	47
4.1.4	Data memory references	50
4.1.5	Cache behavior	51
4.1.6	Floating-point operations.....	54
4.1.7	Multimedia extensions.....	55
4.1.8	Speculative execution factor and UOPS per instruction.....	57
4.2	A comparison of SIMD and VLIW approaches for media processing	59
4.2.1	TMS320C62xx DSP processor.....	59
4.2.2	Results.....	60
4.3	Summary.....	67
Chapter 5.	Bottlenecks in Multimedia Processing with	
	SIMD style Extensions	70
5.1	A scalability test.....	72
5.2	Identification of bottlenecks	75
5.2.1	Nested loops in multimedia applications.....	75
5.2.2	Overhead/supporting instructions.....	81
5.2.3	SIMD throughput and efficiency	86
5.2.4	Memory access and branch bottlenecks.....	90
5.3	Summary.....	91

Chapter 6. Hardware Support for Efficient SIMD Processing	94
6.1 The MediaBreeze architecture	95
6.1.1 Decoupling computation and overhead	95
6.1.2 Multicast: A technique to aid in data transformation	103
6.1.3 Example encoding using the Breeze instruction.....	106
6.2 Performance evaluation	110
6.3 Summary	114
Chapter 7. Hardware Cost of the MediaBreeze Architecture	116
7.1 Implementation methodology	119
7.2 Hardware implementation of the MediaBreeze units	121
7.2.1 Address generation	121
7.2.2 Looping.....	122
7.2.3 Breeze instruction decoder.....	124
7.2.4 Breeze instruction memory	124
7.2.5 Existing hardware units	125
7.3 Area, power, and timing results	125
7.4 Summary	129
Chapter 8. Conclusion	131
Appendix A. Performance Monitoring Events on the P6 Microarchitecture	135
Appendix B. Hardware Cost of the MediaBreeze Architecture across Different ASIC Technologies	136
Bibliography	139
Vita	150

List of Tables

1.1	GPP multimedia extensions	4
1.2	List of available media processors	10
3.1	Description of commercial media applications used as benchmarks in workload characterization	36
3.2	Summaries of benchmark kernels and applications.....	37
4.1	Data cache miss rates of multimedia applications	52
4.2	Execution clock cycles for SIMD and VLIW processors.....	61
5.1	Processor and memory configuration for the scalability test.....	73
5.2	Summary of key media algorithms and the required nested loops along with their primitive addressing sequences	80
5.3	Execution statistics and efficiency of media programs	88
5.4	Performance (IPC) with unit cycle memory accesses and perfect branch prediction	91
6.1	Speedup of the MediaBreeze architecture along with SIMD efficiency (as a %). The 2-way SIMD GPP is used as the baseline	112
6.2	Speedup of the MediaBreeze architecture with prefetching.....	114
6.3	Percentage reduction in dynamic instruction count of the MediaBreeze architecture in comparison to a conventional RISC ISA with SIMD extensions	114
7.1	Hardware functionality of various MediaBreeze hardware units ...	117

7.2	Area, power, and timing estimates of MediaBreeze units in a G12-p ASIC technology	126
7.3	Area of commercial SIMD and GPP implementations.....	127
A.1	P6 microarchitecture counters based performance measures	135
B.1	List of cell-based libraries used in the synthesis of MediaBreeze hardware units	136
B.2	Timing, area, and power estimates across different technologies ..	137

List of Figures

1.1	SIMD add instruction.....	5
1.2	SIMD multiply-add instruction.....	5
1.3	SIMD pack instruction.....	6
1.4	SIMD unpack instruction.....	6
1.5	SIMD permute instruction (1, 0, 0, 0 pattern)	7
1.6	SIMD permute instruction (mixing)	7
1.7	Simplified block diagram of a VLIW core	8
4.1	Cycles per instruction	45
4.2	Stalls per instruction	46
4.3	Branch statistics	48
4.4	Memory reference statistics.....	50
4.5	Cache statistics.....	53
4.6	Log plot of CPI versus L1 and L2 cache misses.....	54
4.7	Percentage of floating-point instructions.....	55
4.8	Percentage of MMX instructions.....	56
4.9	Packing and unpacking as a percentage of all MMX instructions..	57
4.10	Average number of UOPS per instruction and speculation execution factor	58

4.11	CPU core of the TMS320C62xx processor	60
4.12	Ratios of execution times of SIMD and VLIW processors	61
5.1	Results of the scalability test	74
5.2	A 2-D data structure in which sub-blocks of data are processed....	76
5.3	C-code for the 2D-DCT implementation	78
5.4	Typical access patterns in multimedia and DSP kernels	79
5.5	Pentium III optimized assembly code for the 1D-DCT routine.....	84
5.6	SimpleScalar optimized assembly code for the 1D-DCT routine ...	85
5.7	Breakdown of dynamic instructions into various classes	86
6.1	The MediaBreeze architecture	96
6.2	Structure of the Breeze instruction	99
6.3	Multicast technique versus traditional SIMD matrix multiply	105
6.4	Pseudo-code implementation of the MediaBreeze unit for looping	107
6.5	Pseudo-code implementation of the MediaBreeze unit for address generation.....	108
6.6	Pseudo-code implementation of the MediaBreeze unit for loads/stores.....	108
6.7	Pseudo-code implementation of the MediaBreeze unit for SIMD computation and data reorganization.....	108
6.8	Breeze instruction mapping of the 1D-DCT.....	109
6.9	Performance of the MediaBreeze (MB) versus SIMD	111

7.1	Block diagram of the address generation hardware.....	122
7.2	Block diagram of the five hardware loops.....	123
B.1	Percentage of interconnect area in the overall area	138
B.2	Breakdown of dynamic power into cell internal power and net switching power.....	138

Chapter 1

Introduction

Contemporary computer applications are multimedia-rich, involving significant amounts of audio and video compression, 2-D image processing, 3-D graphics, speech and character recognition, communications, and signal processing. With the proliferation of the World Wide Web and the Internet, future workloads are believed to be even more multimedia dominant. These applications run on a variety of systems ranging from the low power personal mobile computing environment to the high performance desktop, workstation, and server environment. This chapter describes the major approaches for processing multimedia applications and the objectives and contributions of this dissertation.

1.1 Approaches for Multimedia Processing

This section describes the common approaches for handling multimedia workloads, namely, general-purpose processors (GPP) with single instruction multiple data (SIMD) extensions, very long instruction word (VLIW) media processors and application specific integrated circuits (ASICs).

1.1.1 General-Purpose processors with SIMD extensions

Virtually every PC sold today is branded as multimedia capable. This has initiated a software revolution that has brought a wide range of audio- and video-based applications to the desktop. It is very common for desktop computers to run video editing or image processing applications (such as Adobe Photoshop) and 3-D games in addition to basic productivity applications (such as word processing, spreadsheet, and database applications). In addition, network multimedia applications leverage the existing network infrastructure to deliver video and audio to end users, such as video conferencing and video server applications. With these application types, video and audio streams are transferred over the network between peers or between clients and servers. With evolving standards and changing consumer needs, future general-purpose processors require good multimedia processing capabilities.

In order to provide the multimedia capability, GPP manufacturers have announced extensions to their instruction set architectures (ISA) that enhance the performance of multimedia applications [27][53][62][73]. These ISA extensions operate in a SIMD fashion to exploit data level parallelism (DLP) in multimedia applications. SIMD is one of the four paradigms for computer design as proposed by Flynn [31]. Multimedia and digital signal processing (DSP) applications typically use small data types (primarily 8- and 16-bits) and spend a significant portion of the execution time in loops that have a high degree of proc-

essing regularity. Packing several small data elements into the wider GPP data-path (typically 32- or 64-bits wide) enables simultaneous processing of separate data elements. Initial implementation of the SIMD extensions such as Intel's MMX, Sun's VIS, Compaq's MVI, MIPS's MDMX, and HP's MAX supported integer data types in the mid-1990's. Floating-point support in media extensions was introduced first in the 3DNow! from AMD and was followed by SSE and SSE2 from Intel. Motorola's AltiVec was introduced with both integer and floating-point capability simultaneously. Table 1.1 shows the list of GPP vendors that have announced/shipped SIMD extensions to their GPP core.

All of the initial SIMD implementations were based on 64-bit SIMD execution units providing 8-, 4-, or 2-way parallelism (8-, 16-, or 32-bit data respectively). The AltiVec and SSE2 are implemented with 128-bit SIMD execution units. SIMD instructions are available for several arithmetic and logical operations in addition to special media operations (such as sum-of-absolute differences). Instructions for data reorganization such as packing and unpacking, and permute are also included in the media extensions.

Figures 1.1-1.6 show examples of subword execution of common multimedia operations. In Figure 1.1, a purely data parallel add operation with four subwords in each register is accomplished. In Figure 1.2, multiplication of four subword pairs and addition of two sets of partial results is occurring leading to two result words. Figure 1.3 illustrates packing of two registers into one register.

Table. 1.1 GPP Multimedia Extensions

Vendor	Processor	ISA Extension	Description
Hewlett Packard	PA-RISC	Max-1 Max-2	Media acceleration extensions
Sun Microsystems	UltraSparc	VIS	Visual Instruction Set
Intel	x86	MMX SSE SSE2	MultiMedia eXtensions Streaming SIMD Extensions Streaming SIMD Extensions 2
AMD	x86	MMX 3DNow! SSE	MultiMedia eXtensions 3Dnow! Extensions Streaming SIMD Extensions
Cyrix	x86	MMX	MultiMedia eXtensions
MIPS	MIPS V	MDMX	MIPS Digital Media eXtensions
Compaq	Alpha	MVI	Motion Video Instructions
Motorola	PowerPC	AltiVec	AltiVec extensions

Figure 1.4 illustrates the complementary operation of unpacking. Some of the multimedia extensions also provide permute such as those depicted in Figures 1.5 and 1.6.

The number of SIMD instructions (and their functionality) has been seen to vary widely depending on the manufacturer. For example, the number of Compaq's MVI instructions is 13, Motorola's AltiVec has 162, and Intel

Pentium 4 has 270 SIMD instructions (MMX, SSE, and SSE2). Initial implementations of SIMD extensions shared the registers of the GPPs floating-point register file (for example, MMX and VIS). Recent extensions (SSE2, AltiVec) have dedicated register files to store temporary data.

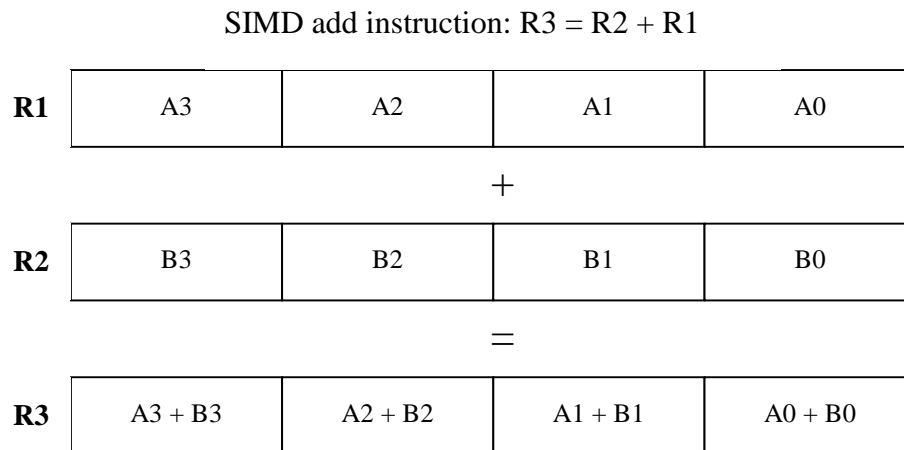


Figure. 1.1. SIMD add instruction

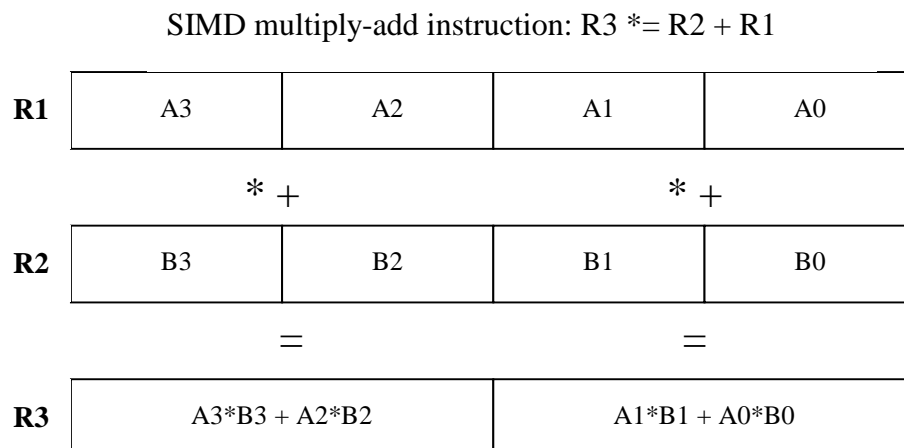


Figure. 1.2. SIMD multiply-add instruction

SIMD pack instruction: R3 = pack (R2 and R1)

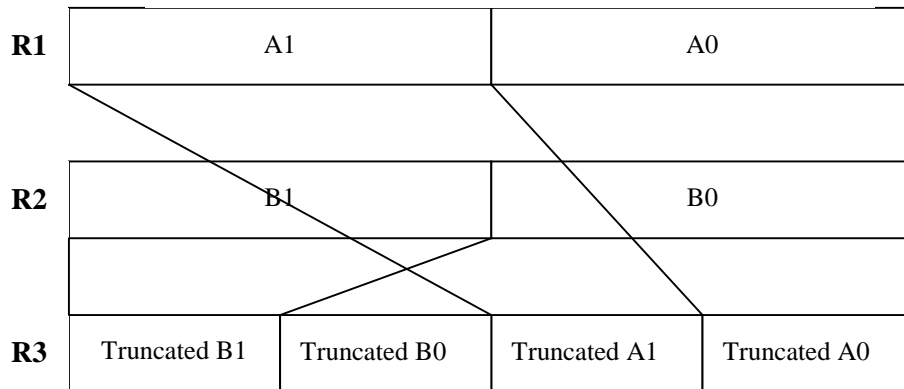


Figure. 1.3. SIMD pack instruction

SIMD unpack instruction: R2 = unpack (R1)

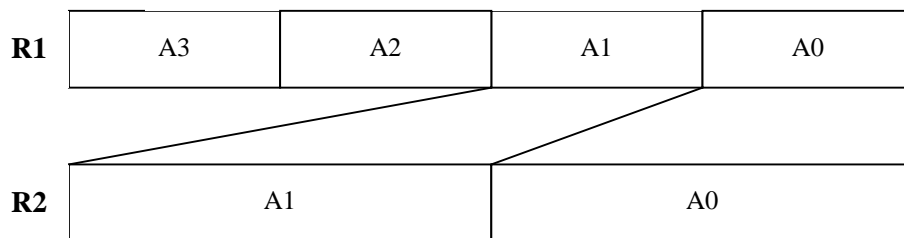


Figure. 1.4. SIMD unpack instruction

SIMD permute instruction: R2 = permute R1 (1, 0, 0, 0 pattern)

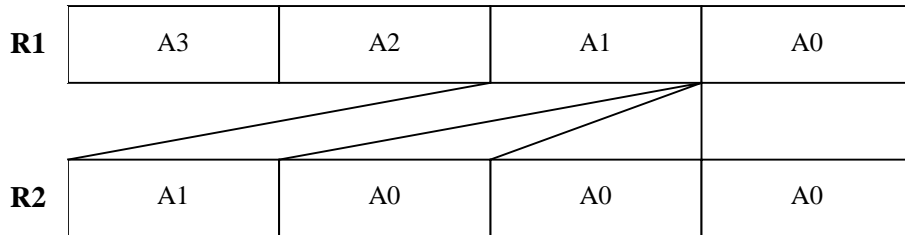


Figure. 1.5. SIMD permute instruction (1, 0, 0, 0 pattern)

SIMD permute instruction: R2 = permute R1 (mixing)

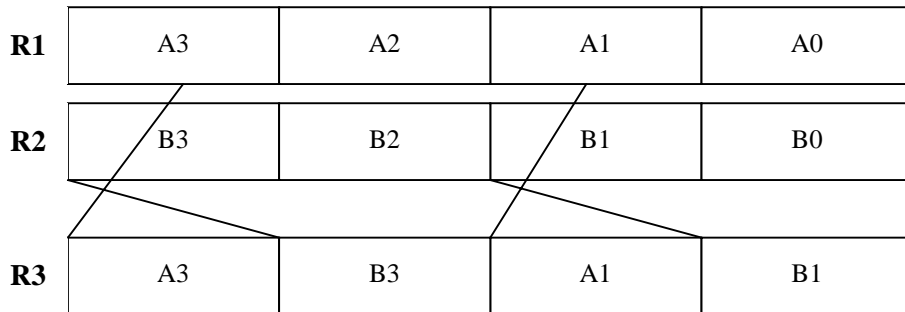


Figure. 1.6. SIMD permute instruction (mixing)

The cost of incorporating the media extension hardware is minimal (typically less than 3% of the overall processor die size). Compiler support for the multimedia extensions is still in its infancy. Current support for programmers involves use of compiler intrinsics [95] and assembly libraries provided by the vendors. However, these intrinsics and libraries vary between different vendor implementations and applications have to be coded separately for each processor platform. Some C/C++ compilers (e.g. Intel's compiler) can generate assembly code that utilizes MMX. Overall, multimedia extensions to GPPs have

been quite successful at providing additional performance for media applications in GPPs.

1.1.2 VLIW Architectures for Multimedia Processing

Due to the processing regularity of multimedia and DSP applications, statically scheduled processors such as VLIW processors are a viable option over dynamically scheduled processors, such as state-of-the-art superscalar GPPs. VLIW processors rely on software to identify parallelism and assemble wide instruction packets to issue multiple instructions per cycle Figure 1.7 shows the block diagram of a generic VLIW processor core.

In the past eight years, several IC vendors have touted processors, generally based on VLIW architectures, which can handle media processing chores for applications ranging from PC multimedia to high-definition digital TV. These VLIW processors are primarily appearing in the area of dedicated multimedia processors. Multimedia processors are defined as programmable proces-

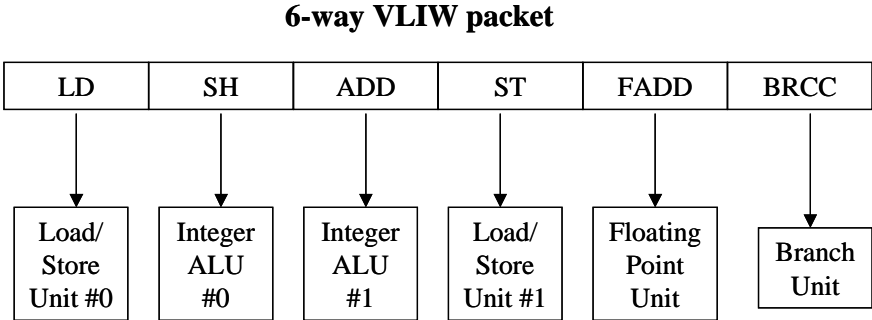


Figure. 1.7. Simplified block diagram of a VLIW core

sors dedicated to simultaneously accelerating operations on several multimedia data types. These processors are dedicated to processing multimedia data, in contrast with standard GPP host processors, so that their architectures can be specialized to processing these data types in the most cost-effective manner. The high bandwidth and fast integer performance allow media processors to simultaneously accelerate different multimedia data types. A single programmable device can replace numerous fixed discrete function devices. Media processors also possess features that differentiate them from DSPs. Standard DSPs typically do not include support for video and computer graphics.

Table 1.2 lists major vendors of media processors and their processors [7][30][32][56][67][68][80][103]. Chromatic Research's Mpack chip was designed to act as a co-processor to a SIMD enhanced general-purpose processor in addition to stand-alone operation in DVD players. All of the other media processors typically are marketed as low-cost, stand-alone processors (without a high-performance GPP in the system) for embedded multimedia systems rather than as a co-processor in a PC system. In addition to the VLIW core, most of the media processors are equipped with several co-processors uniquely targeted at specific functions. For example, Equator Technology's MAP1000A co-processors include a 16-bit microprocessor optimized for low-latency bit-serial processing, a programmable video scalar, and 9 kB of on-chip memory shared by the co-processors [7].

Table. 1.2 List of Available Media Processors

Manufacturer	Media processor	Target applications
Chromatic Research (LG Semiconductor, STMicroelectronics, and Toshiba)	Mpact 1 and 2: combine a two-instruction VLIW architecture with SIMD capabilities; the newer Mpact 2 executes 6 BOPS and includes a hardware 3-D graphics accelerator [67].	Mpact 2: PC multimedia co-processor capable of graphics, audio, MPEG-2 decoding, DVD-player, and modem tasks; Mpact 1: mainly as a DVD decoder
Equator Technologies	MAP1000A: VLIW processor (4-way) with SIMD vector units. Supports 3.2 GB/s of aggregate bandwidth. Also equipped with video co-processors for specific functions [7][67].	Camcorders to HDTV sets along with 3-D games
Fujitsu Microelectronics	Multi Media Assist (MMA): combines a two-instruction VLIW architecture with SIMD capabilities to reach more than 1 BOPS [67].	DVD players, set-top boxes, and printers
Matsushita Semiconductor	Media Core Processor (MCP): combines four-instruction VLIW architecture and a DSP execution unit to reach more than 3 BOPS [67].	DVD players, set-top boxes, and car navigation systems
Mitsubishi Electronics	D30V: combines two-instruction VLIW architecture and SIMD capabilities to reach 1 BOPS [67]	DVD players, set-top boxes, and videoconferencing systems; D10V precursor targets cell phones

Table. 1.2 List of Available Media Processors (continued)

Philips Semiconductors	TriMedia TM1000, 1100, and 2000: combine five-instruction VLIW architecture and SIMD capabilities to reach approximately 3 BOPS; includes dedicated MPEG-2 decoder and video scalar, and 1100 includes DVD encryption block [67][80].	DVD players, set-top boxes, and digital TV, including HDTV and video conferencing systems; offers robust digital-TV reference design and development platform
Sharp Digital	Data-Driven Media Processor (DDMP): clockless multiprocessor architecture with an integrated ARM RISC controller [67].	Color fax machines and printers to camcorders, DVD players, set-top boxes, and digital TV
Sony	PlayStation2's Emotion Engine: a CPU core with two vector processing engines [56].	3-D graphics
Texas Instruments	TMS320C6000 DSP (C62xx, C67xx, C64xx): eight-instruction VLIW DSP with the C64xx having SIMD capability [100][103]. C62xx/C67xx supports 2.4 GB/s and C64xx supports 4.8 GB/s of aggregate bandwidth.	Wireless communication base stations and remote-access servers for dial-up and ADSL lines
Analog Devices	TigerSharc DSP: four-way VLIW DSP with SIMD capability [32].	Wireless communication base stations and remote-access servers for dial-up and ADSL lines
VM labs	NUON processor: delivers over 1500 MIPS [67].	Consumer-electronics devices capable of DVD and 3-D games
Agere Systems	Starcore DSP: four-way VLIW DSP.	Servers and cellular infrastructure and third generation wireless systems

1.1.3 ASICs

Another alternative for processing multimedia streams is to use application specific integrated circuit (ASIC) chips. ASICs offer a fixed hardware solution for processing multimedia streams. Multimedia applications spend a significant amount of time within small numbers of processing routines (kernels). Designers of ASICs optimize the hardware for the most critical sections of the application to achieve high degrees of performance. An ASIC that is designed for a specific function typically delivers performance that is an order-of-magnitude greater than a GPP of the same raw processing capacity to run the same function. Because of the increasing transistor densities, several different functions can be implemented on a single-chip in addition to system logic, leading to a system-on-chip. For large volume applications, ASICs offer significant performance advantages in addition to tremendous savings in power consumption at a low-cost.

An example of a media processing ASIC is the recently announced Analog Devices Inc.'s ADV-JP2000 [3]. The ADV-JP2000 is a high performance image co-processor that implements the computationally intensive operations of the JPEG2000 image compression standard in hardware. The chip contains a full custom wavelet processor and entropy codec as well as associated interface and control functions. Another example of ASICs for multimedia processing is the

C-Cube family of products [16] for applications such as set-top boxes, broadcast, consumer video, and DVD.

The major drawback of using ASICs is that they provide limited if any flexibility because they are optimized to implement a specific function. For example, the ADV-JP2000 can only perform a 5/3-wavelet transform. Many multimedia technologies are fast-moving targets due to changing standards, evolving interfaces, and shifting consumer tastes. Programmable approaches (SIMD general-purpose processors and VLIW media processors) provide an advantage over ASICs.

1.2 The Problem

There are primarily two problems in media processing with general-purpose processors.

- The behavior of multimedia applications on general-purpose processors is not well understood.
- Media processing on general-purpose processors using SIMD style extensions contain several bottlenecks.

1.3 Objectives

The specific objectives of this dissertation are two-fold:

1. The first objective is to understand the characteristics of multimedia applications. This dissertation investigates the following issues:
 - How are the execution characteristics of multimedia applications different from that of other desktop applications? Execution characteristics such as memory and branch behavior, resource and instruction stream stalls, data memory references, and floating-point operations are examined.
 - How do characteristics of multimedia applications map onto SIMD enhanced general-purpose processors?
 - Are SIMD enhanced general-purpose processors capable of exploiting all the available data level parallelism in multimedia applications?
 - What percentage of the peak computation rate is achieved for the SIMD execution units in general-purpose processors?
 - If the computation rate is low, what are the reasons that prevent the SIMD execution units from achieving a good computation rate? What are the bottlenecks in media processing using SIMD style extensions on general-purpose processors?

2. The second objective is to design and implement cost effective hardware support for alleviating/eliminating the performance bottlenecks in SIMD enhanced general-purpose processors.

1.4 Thesis Statement

A dominant fraction of instructions in the multimedia instruction stream is not actually performing useful computations, but merely supporting the computations. Hardware to accelerate these supporting instructions can significantly improve the performance of media applications on SIMD enhanced general-purpose processors.

1.5 Contributions

This dissertation makes several contributions to the characterization of multimedia workloads, detection of bottlenecks, and explicit hardware support for accelerating media applications on SIMD GPPs. These are also described in more detail in [95], [96], [97], [98], and [99]. The summary of the contributions is listed below.

1. I perform a quantitative study of the execution characteristics of commercial multimedia applications on a state-of-the-art superscalar processor. Memory access behavior, cache and branch behavior, and resource

usage are studied. It is found that, contrary to popular belief that caches are ineffective for multimedia applications, multimedia applications exhibit better overall data cache performance than desktop applications. Also, the branch misprediction ratio of multimedia applications is higher than that of SPEC benchmarks. I also perform an evaluation of SIMD and VLIW techniques for multimedia and DSP applications.

2. I present a characterization of media workloads on SIMD GPPs from the perspective of support required for efficient SIMD processing rather than focusing on the computation part of the algorithms. This study shows that 75-85% of instructions in the media instruction stream are not performing useful (actual/true) computations, but merely supporting the computations. It is also observed that the SIMD computation units are computing at less than 12% of their peak computation rate.
3. I introduce the MediaBreeze architecture that significantly improves performance of media applications by decoupling media program execution into useful computations and overhead/supporting instructions. Explicit hardware support is provided for executing the supporting instructions. It is found that on a set of multimedia kernels, a 2-way SIMD GPP augmented with the MediaBreeze architecture is superior to a 16-way SIMD GPP. On a set of multimedia applications, a 2- and 4-way SIMD GPP

augmented with the MediaBreeze architecture outperforms a 4- and 8-way SIMD GPP respectively.

4. I implement the hardware units of the MediaBreeze architecture to analyze area, power, and timing tradeoffs. It is found that the added hardware consumes less than 0.3% of overall GPP chip area and less than 0.5W in power consumption at 1GHz. This is achieved without elongating the critical path of the GPP pipeline.

1.6 Organization

Chapter 2 describes existing work pertinent to this dissertation. Past research efforts on characterizing multimedia workloads are discussed first. Then studies evaluating the effectiveness of SIMD extensions, and architectural enhancements to improve performance of multimedia applications are discussed. General-purpose computer architecture schemes related to the proposed enhancements in this dissertation are also described.

Chapter 3 presents the performance evaluation methodology used in this dissertation. A detailed description of the tools, benchmarks, evaluation environment, and performance measures is presented.

Chapter 4 presents a quantitative study on the execution characteristics of commercial multimedia applications on a state-of-the-art SIMD enhanced

general-purpose processor. The similarities and differences between multimedia and other desktop workloads are highlighted. In addition, an evaluation and comparison of SIMD and VLIW paradigms for media and signal processing is presented. The Pentium II and TMS320C62xx processors are used as SIMD and VLIW representatives respectively.

Chapter 5 identifies bottlenecks in the execution of multimedia applications on SIMD GPPs. The supporting instructions that are necessary to feed the SIMD execution units are analyzed. The utilization rate of the SIMD execution units is measured.

Chapter 6 proposes the MediaBreeze architecture that is influenced by the characterization studies in Chapters 4 and 5. The focus of this architecture is on the instructions that support the core computations, rather than on the computations themselves. The performance of the MediaBreeze architecture is evaluated and compared with wide-issue SIMD GPPs.

Chapter 7 investigates the cost of incorporating the MediaBreeze architecture into a high-speed SIMD GPP. Tradeoffs in area, power, and timing are evaluated using a cell-based ASIC design methodology.

Chapter 8 concludes the dissertation by summarizing the contributions and suggesting future opportunities.

1.7 Acronyms

ADPCM	- Adaptive Differential Pulse Code Modulation
ALU	- Arithmetic Logic Unit
ASC	- Applied Scientific Computer
ASIC	- Application Specific Integrated Circuit
BOPS	- Billions of Operations per Second
BTB	- Branch Target Buffer
CAD	- Computer Aided Design
CFA	- Color Filter Interpolation
CISC	- Complex Instruction Set Computer
CMOS	- Complementary Metal Oxide Semiconductor
CPI	- Cycles per Instruction
CPU	- Central Processing Unit
DAE	- Decoupled Access Execute
DCT	- Discrete Cosine Transform
DLP	- Data Level Parallelism
DRAM	- Dynamic Random Access Memory
DSP	- Digital Signal Processing or Digital Signal Processor
DTLB	- Data Translation Look-aside Buffer
DWT	- Discrete Wavelet Transform

FFT	- Fast Fourier Transform
FIR	- Finite Impulse Response
FLOPS	- Floating Point Operations per Second
FP	- Floating-Point
GFLOPS	- Giga Floating-Point Operation per Second
GPP	- General Purpose Processor
IDEA	- International Data Encryption Algorithm
IIR	- Infinite Impulse Response
ILP	- Instruction Level Parallelism
IPC	- Instructions per Cycle
ISA	- Instruction Set Architecture
I-stream	- Instruction stream
ITLB	- Instruction Translation Look-aside Buffer
JPEG	- Joint Photographic Expert Group
L1 cache	- Level 1 cache
L2 cache	- Level 2 cache
LRU	- Least Recently Used
MAC	- Multiply and Accumulate
MAX	- Media Acceleration eXtensions
MDMX	- MIPS Digital Media eXtensions
MFLOPS	- Millions of Floating-Point Operations per Second

MIPS	- Million Instructions per Second
MMX	- MultiMedia eXtensions
MPEG	- Moving Picture Expert Group
MUX	- Multiplexer
MVI	- Motion Video Instructions
NOP	- NO Operation
NSP	- Native Signal Processing
RISC	- Reduced Instruction Set Computer
SDRAM	- Synchronous Dynamic Random Access Memory
SIMD	- Single Instruction Multiple Data
SMA	- Structured Memory-access Architecture
SPEC	- Standard Performance Evaluation Corporation
SPECint	- SPEC Integer benchmarks
SPECfp	- SPEC Floating Point benchmarks
SSE	- Streaming SIMD Extensions
TLB	- Translation Look-aside Buffer
UOPS	- Micro Operations
VIS	- Visual Instruction Set
VLIW	- Very Long Instruction Word

Chapter 2

Related Work

Multimedia workloads and processors have been researched extensively in the past few years. The related work can be divided into different categories: characterizing multimedia workloads, efforts quantifying the benefits of SIMD extensions, architectural enhancements proposed by other researchers to improve performance of multimedia applications, and general-purpose architecture research related to the enhancements proposed in this dissertation.

2.1 Multimedia Application Characteristics

This section describes past research that discusses characteristics of media workloads. Diefendorff and Dubey [26] mention several distinguishing characteristics of multimedia applications from general-purpose applications in a position paper.

- Real-time response: Multimedia applications such as video conferencing or electronic commerce often require a certain quality of service and real-time response.

- Continuous-media data types: The input data for multimedia applications often comprises a set of data elements derived from sampling some analog signal in a time domain – either video, audio, or other sensory perception. Media data types differ from other data types in that the width of typical data is 8 or 16 bits versus 32 or 64 bits.
- Significant fine-grained data parallelism: Data parallelism is inherent in almost all signal processing and graphics applications. Input data streams are frequently large collections of small data elements such as pixels, vertices, or frequency/amplitude values. This lends well to machines with SIMD hardware units executing in parallel.
- Significant coarse-grained parallelism: Most media applications and scenarios consist of more than one time-critical execution thread. For example, a typical video conferencing application consists of video encoding and decoding, audio encoding and decoding, and background task threads that are independent of each other.
- High instruction-reference locality to small loops: DSP and media processing applications often consist of small loops or kernels that dominate overall processing time.

- High memory bandwidth: The working data sets of media applications are huge implying that processors must provide high memory bandwidth and tolerate long memory latency.
- High network bandwidth: The processor must be able to accommodate for high network speeds of the future.
- Extensive data reorganization: Packing and unpacking of data is necessary for taking advantage of SIMD execution units.

Lee, *et al.* [59] introduce the MediaBench benchmark suite and compare media applications with SPECint95 workloads. They use a single-issue processor to perform several experiments. They observe that the MediaBench benchmarks have better instruction cache hit rates than SPECint95 benchmarks. They also find that data caches are more effective for reads on MediaBench than SPECint95, while they are less effective for writes. SPECint95 required almost three times more bus bandwidth than MediaBench. MediaBench applications were found to have higher IPC than SPECint95 workloads.

Fritts [34] extends the characterization of MediaBench workloads. He finds that nearly 70% of the instructions operate on data sizes of 8 and 16 bits. The average basic block size is found to be small, which leads him to conclude that parallelism in multimedia applications is not within basic blocks. Evaluation

of ILP revealed multimedia applications are similar to general-purpose applications.

An evaluation of parallelism, operation frequencies and memory performance on video signal processors (closely related to media processors) has been performed using trace-driven simulations in [63]. This study was performed with assumptions such as perfect branch prediction, perfect memory disambiguation, and an infinite-sized scheduling window. These trace-driven simulation results are best used to define an upper bound on potential performance. They found ILP ranging from 32.8 to over 1000 for their ideal machine model.

Sohoni, *et al.* [90] conducted a study of memory system performance of multimedia applications on the MediaBench suite. They observe that for L1 data caches, multimedia applications actually have lower cache miss rates than SPECint95 programs. In addition, they conclude that larger input data sizes do not necessarily result in a higher cache miss rate. Slingerland and Smith [87] analyze cache behavior of the Berkeley Multimedia Workload and find that multimedia applications actually exhibit lower instruction miss ratios and comparable data miss ratios when contrasted with other widely studied workloads. In addition, they find that longer data cache line sizes than are currently used would benefit multimedia processing.

Cucchiara, *et al.* [25] explore cache strategies for multimedia applications. They find that standard caching policies in GPPs exhibit poor performance

in exploiting the 2D spatial locality typical of programs handling and processing images. They introduce hardware prefetching by employing a 2D prefetch policy and observe better cache performance than one block lookahead policy.

Hughes, *et al.* [42] measure the variability in the execution of multimedia applications on GPPs. They 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. They conclude that aggressive architectural features induce little additional variability (and unpredictability) in execution time.

In this dissertation, I evaluate the execution characteristics of commercial multimedia applications on a state-of-the-art superscalar processor with SIMD extensions and compare them with existing characterizations of other desktop workloads.

2.2 Benefits of SIMD Extensions

Several research efforts have evaluated the benefits of SIMD extensions since their commercial introduction in 1994 [10][19][27][58][62][71][78][93]. Benchmarking of several applications on the UltraSparc processor using VIS [19] showed a performance speedup for some DSP applications over non-VIS versions. Applications with FIR filters showed the most improvement while IIR

filters and FFTs exhibited little or no performance increase. An evaluation of MMX on a Pentium processor on kernels and applications was presented by Bhargava, *et al.* in [10]. Performance of image and video processing with VIS extensions was analyzed by Ranganathan, *et al.* in [78] and benefits of VIS were reported. It was shown that conventional ILP techniques provided 2x to 4x performance improvements and media extensions provided an additional 1.1x to 4.2x performance improvement. Motorola's AltiVec technology is seen to result in a significant performance increase (1.6x to 11.7x) for DSP and multimedia kernels in [71].

Lappalainen [58] presented performance analysis of MMX technology for an H.263 video encoder and reported a performance improvement of 1.65x by using MMX over optimized scalar assembly code without MMX. A number of commercial general-purpose and DSP processors have been benchmarked by BDTI [11][13] on a suite of 11 kernels. However, only a single performance metric denoting the execution time is released in the public domain for all of the benchmarks together. The execution time is measured for kernels written in assembly that use only on-chip memory.

Sriram and Hung [93] presented an implementation of MPEG-2 video decoder on a C62xx DSP processor and compared the performance of the various components with MMX, HP MAX and VIS. The C62xx was found to be faster than the three SIMD implementations. Different DLP alternatives for the

embedded media domain are evaluated by Salami, *et al.* [84] and demonstrate the superiority of Matrix SIMD extensions (2D SIMD) over traditional SIMD extensions.

In this dissertation, I compare SIMD and VLIW approaches for multimedia and DSP applications using state-of-the-art commodity processors.

2.3 Media Architectural Enhancements

Research starting in the mid-to-late 1990's proposed several architectural enhancements to improve performance of multimedia workloads on GPPs. Rixner, *et al.* [81][82] developed the Imagine architecture for bandwidth-efficient media processing. This architecture is based on clusters of ALUs processing large data streams and is built as a co-processor for a high-end multimedia system. Three levels of memory hierarchy are provided – local register file for each cluster, a global stream register file and external SDRAM. Compared to a conventional scalar processor, they found that Imagine reduces the global register and memory bandwidth by factors of 13 and 21 respectively and is able to achieve a peak performance of 16.2 GFLOPS and a sustained performance of 8.5 GFLOPS on media processing kernels.

Goldstein, *et al.* [35] design the PipeRench co-processor for streaming multimedia acceleration. The PipeRench co-processor is a reconfigurable fabric

architecture achieving up to 190x performance improvement on media kernels over a RISC processor. A related effort is the Chimaera architecture from Ye, *et al.* [110]. Chimaera is a prototype system that integrates a small and fast reconfigurable functional unit into the pipeline of an aggressive dynamically scheduled superscalar processor. The authors demonstrate that for a 4-way out-of-order superscalar processor, Chimaera results in an average performance increase of 21%.

Quintana, *et al.* [76] have proposed adding a vector unit to a superscalar processor to improve performance of numeric and multimedia codes. Related work from Corbal, *et al.* [21] proposes to exploit DLP in two dimensions instead of one dimension processing as in MMX. A 20% performance improvement is shown using relevant multimedia applications over traditional SIMD extensions.

Lee and Stoodley [60] have proposed and evaluated the use of simple long vector microprocessors for multimedia applications. They show that instead of using an out-of-order superscalar processor or an out-of-order short vector (conventional SIMD extensions), a simple in-order long vector allows for potential saving in chip area and achieves better multimedia performance. However, it is important to have a general-purpose processor to achieve sustained performance on different domains of workloads.

Vassiliadis, *et al.* [50][106] have proposed the Complex Streamed Instruction Set (CSI) to enhance an existing out-of-order GPP. A stream computa-

tion instruction can capture two levels of loop nesting. Vermuelen, *et al.* [107] describe how DCT, Reed-Solomon code, and other similar media-oriented operations can be enhanced with a hardware accelerator that works in conjunction with a GPP. However, the accelerator has to be designed for each algorithm. Retargeting the accelerator to another algorithm incurs significant effort.

Ranganathan, *et al.* [79] propose reconfigurable caches and their applicability to media processing. They find IPC improvements ranging from 1.04x to 1.2x when applying instruction reuse for eight multimedia benchmarks.

In this dissertation, I accelerate multimedia applications by adding explicit hardware support to a SIMD GPP.

2.4 Related GPP Architectures

In this section, I describe GPP architectures relevant to the proposed enhancements in this dissertation. The proposed solution combines the advantages of SIMD, vector, DAE, and DSP processors. The DAE concept present in the IBM System 360/370, CDC 6600 [104], CDC7600, CRAY-1, CSPI MAP-200, SDP [85], PIPE [36], SMA [75], WM [109], and DS [112] demonstrated the potential of decoupling memory accesses and computations [88][89].

There also has been research in specialized access processors and address generation coprocessors [8][43]. The concept of embedding loops in

hardware was implemented commercially in the TI ASC [23] (do-loop in this case). The SMA architecture [75] provided similar flexibility in accessing matrices. This concept was seen to be successful in all these machines as well as many DSP processors [57]. The Burroughs scientific processor [55] was a pure SIMD array processor that had special-purpose hardware called alignment networks for packing and unpacking data. In addition, the processor has several powerful SIMD instructions of which many are being used in current SIMD extensions.

Chapter 3

Performance Evaluation Methodology

In this chapter, I present the methodology for characterizing media workloads and evaluating the performance of media enhancements proposed in this dissertation. A detailed description of the various tools and benchmarks used in this dissertation is presented. I also discuss the evaluation environment and performance measures.

3.1 Tools

3.1.1 Performance monitoring counters

Built-in processor performance counters on the Intel P6 microarchitecture (Pentium II and III processors) are used in the media workload characterization studies. Measurements of various statistics on P6 processors are performed using these counters. Hardware performance counters offer the advantage of measuring processor statistics in a non-obtrusive way and generating results in real-time. In addition, benchmark source code is not necessary for measuring execution statistics. The P6 microarchitecture implements two performance

counters [45][46], with each counter associated with an event select register that controls what is counted. The counters are accessed via the RDMSR and WRMSR instructions. To measure more than two events (performance counters can only measure two events for each run), several runs of each benchmark are necessary.

The performance monitoring utility on the P6 microarchitecture provides an option of reading only Ring3 events or both Ring0 and Ring3 events. Ring3 events correspond to the user level processes that are active at a particular time. Ring0 events correspond to the operating system processes. For this dissertation, the Ring0 events were masked to gather the execution characteristics of each multimedia application without intervention from operating system-related events. While evaluating each benchmark, no other user process was kept active to minimize the effects of pollution. A detailed listing of various performance monitoring events on the P6 microarchitecture is provided in appendix A.

3.1.2 VTune

VTune, an Intel performance analysis tool [44] was used to get the complete instruction mix (assembly instructions) of the code. This tool is designed for analyzing “hot spots” in the code and optimizing them. In addition, VTune provides time- and event-based system-based sampling and call graph profiling. VTune

was used to profile instruction mix when using processors with the P6 microarchitecture.

3.1.3 Simplescalar simulator

The out-of-order simulator from the Simplescalar tool suite [15] was used to study the performance of media workloads on superscalar processors. In addition, the Simplescalar simulator is modified to evaluate improvements achieved by the proposed methods. SIMD extensions are provided to the simulator by adding 64-bit SIMD execution units to the processor core.

The Simplescalar tool suite is widely used in computer architecture research involving superscalar processors. The simulator (sim-outorder) models the superscalar out-of-order pipeline in detail using execution driven simulation. It models several different ISAs. I use PISA, an ISA based on the MIPS architecture was used. The Simplescalar tool set provides the ability to add new instructions without altering the compiler via instruction annotations. Instruction annotations are used to model SIMD instructions.

3.1.4 TMS320C6x simulator

The C62xx simulator is used to analyze performance of DSP and multimedia applications on VLIW processors (Chapter 4). Texas Instruments provides a cycle accurate simulator for the C62xx VLIW DSP processor [100]. Execution cy-

cle counts of DSP and media benchmarks can be obtained from the stand-alone simulator. The “clock ()” function provided in the simulator returns the execution times of the benchmarks.

3.2 Benchmarks

Several multimedia benchmarks are used to understand the characteristics of media applications. Table 3.1 shows the commercial media applications used as benchmarks in this dissertation. These applications can be categorized into one of 3D graphics (*QuakeII* and *Unreal*), streaming video (*RealPlayer* and *QuickTime*), and streaming audio (*RealAudio* and *Winamp*). Commercial applications are excellent benchmarks to study the properties of multimedia applications. However, these applications are available only as binaries and source code is not available. Table 3.2 lists several media and signal processing kernels and applications that are used in this dissertation in addition to the benchmarks described in Table 3.1 for evaluating the proposed hardware support. The kernels in Table 3.2 form significant components of media applications. Most of the benchmark applications in Table 3.2 are from popular multimedia benchmark suites such as MediaBench [59].

Table. 3.1 Description of commercial media applications used as benchmarks in workload characterization

Application	Description
QuakeII	One of the most popular 3D games with excellent graphics, sounds, and smart combat enemies. Processor vendors and graphics accelerator manufacturers use this benchmark as a standard gaming benchmark. The game demo is run with a 1024x768 resolution on a 19-inch monitor. Executed over 17 billion instructions.
Unreal	A recent and feature-rich 3D game that is touted to heavily use the MMX instruction set. The graphics engine in Unreal is more advanced than in QuakeII and the audio engine in Unreal outperforms the QuakeII audio. The game demo is run with a 1024x768 resolution on a 19-inch monitor. Executed over 24 billion instructions.
RealVideo	Delivers high quality digital video at much lower bit-rates than other non-streaming solutions, such as compressed QuickTime, AVI, or MPEG. This technology allows Intranets to deliver video training, corporate communications and presentations to the desktop. A video clip of 4.5-inch by 3.5-inch was played. Executed 2.7 billion instructions.
QuickTime	QuickTime is a multimedia architecture developed by Apple to synchronize graphics, text, video, and sound. QuickTime is ideal for synchronizing picture and sound. QuickTime is an economical solution, in terms of bandwidth, for both music and video. An AVI video clip of 9-inch by 7-inch was played. Executed over 7 billion instructions.
Winamp	Winamp is a fast, flexible, high-fidelity music player for Windows 95/98/NT. Winamp supports MP3, MP2, CD, MOD, WAV and other audio formats, custom interfaces called <i>skins</i> and <i>audio visualization</i> and <i>audio effect plug-ins</i> . An MPEG audio stream was played. Executed 1.7 billion instructions.
RealAudio	RealAudio is a system designed to deliver streaming audio, both speech and music. The player does not cache downloaded files. Synchronization with video, flash, and a sequence of HTML files provides an excellent vehicle for multimedia presentation. A RealAudio audio stream was played. Executed 350 million instructions.

Table. 3.2 Summaries of Benchmark Kernels and Applications. Source code for each benchmark is available in [91]

Kernels	
Dot product <i>(dotp)</i>	Dot product of a randomly initialized 1024-element array repeated several times (16-bit data)
Autocorrelation <i>(auto)</i>	Autocorrelation of a 4096-element vector with a lag of 256 repeated several times (16-bit data)
Finite Impulse Response Filter <i>(fir)</i>	Low-pass filter of length 32 operating on a buffer of 256 elements repeated several times (16-bit data)
Color Filter Array <i>(cfa)</i>	Color filter array interpolation of a 2 million pixel image with a 5x5 filter (16-bit data)
Discrete Cosine Transform <i>(dct)</i>	2-D discrete cosine transform of a 2 million pixel image (16-bit data)
Motion Estimation <i>(motest)</i>	Motion estimation routine on a frame of 2 million pixels (8-bit data)
Image Scaling <i>(scale)</i>	Linear scaling of an image of 2 million pixels (8-bit data)
Applications	
Audio Effects <i>(aud)</i>	Adding successive echo signals, signal mixing, and filtering on 2 million data samples (16-bit data)
G.711 speech coding <i>(g711)</i>	A-law to μ -law conversion and vice versa as specified by ITU-T standard on 2 million data samples (8-bit data)
ADPCM speech compression <i>(adpcm)</i>	16-bit to 4-bit compression of a speech signal (obtained from Intel) on a 1024-element buffer repeated several times (16-bit data)
JPEG Image Compression <i>(jpeg)</i>	JPEG image compression on a 800-by-600 pixel image

<i>JPEG Image De-compression (jpeg)</i>	JPEG image de-compression resulting in a 800-by-600 pixel image
<i>IDEA Decryption (decrypt)</i>	IDEA decryption on 192,000 bytes of data

3.3 Evaluation Environment

Pentium II and Pentium III processor based systems running Windows NT 4.0 are used for experiments with the P6 microarchitecture (Chapters 4 and 5). Experiments for analyzing the performance of VLIW processors for media and DSP applications (Chapter 4) are performed on the C62xx simulator. Experiments with SIMD GPPs and the proposed enhancements along with bottleneck detection are performed using the SimpleScalar tool suite (Chapters 5 and 6).

Significant effort was put in generating code for each of the experiments in this dissertation. For the case of commodity processors (Pentium II, Pentium III, and TMS320C62xx DSP), assembly libraries and compiler intrinsics [95] are used to create either SIMD or VLIW versions of the code. For the case of SimpleScalar processor simulator, SIMD code was generated using hand coded assembly and instruction annotations. The rest of this section describes the compilers and code development using assembly libraries and compiler intrinsics.

3.3.1 Compilers

Several compilers are used for generating the media application code. Code generation for processors based on the P6 microarchitecture is performed using Intel C/C++ compiler [44]. Code generation for the C62xx VLIW DSP is performed using Texas Instruments' C62xx compiler [102]. A modified version of gcc (SimpleScalar gcc) is used for generating code for the SimpleScalar processor [15]. Hand coded assembly is used for creating the SIMD code for the SimpleScalar processor simulator. Code generation for the P6 microarchitecture and C62xx DSP processor is by using assembly libraries and compiler intrinsics (explained below). I use maximum optimizations provided by each of the compilers in my study.

3.3.2 Assembly libraries

Intel's assembly libraries [44] provide versions of many common signal processing, vector arithmetic, and image processing kernels that can be called as C functions. However, some signal processing library calls require library-specific data structures to be created and initialized before calling kernels such as *fir*. Using assembly libraries is thus restricted and I used Intel's libraries only for *dotp* and *auto* benchmarks (since only these two benchmarks have the same calling sequence for the C and library functions and the library versions do not use any extra data structures). Unless the code developer can replace a complete function

call in C with a call to the library function, the assembly libraries cannot be utilized completely.

For creating the SIMD versions (for Pentium II and III processors) of the benchmarks using assembly libraries, I replaced the function written in C with a call to the signal processing library that incorporates SIMD (MMX and SSE) instructions. There is no loss of accuracy by using SIMD because all versions of the benchmarks operate on 16-bit data or 8-bit data. Another issue with the use of Intel's libraries is that they are generally robust and intuitive, but employ a lot of error checking code to guarantee functional correctness that can potentially increase execution time. Also, the overhead of using SIMD instructions (misalignment-related instructions, and packing and unpacking data related instructions) should be less than the potential benefit of SIMD instructions.

TI provides optimized assembly code for the C62xx in [101]. These assembly libraries are C-code callable and also have the same calling sequence as the C-code counterpart. Several restrictions apply for using these C62xx optimized VLIW assembly codes. For example, the *fir* code requires that the number of filter coefficients must be a multiple of 4 and length of *auto* vector must be a multiple of 8.

3.3.3 Compiler intrinsics

Both Intel and TI libraries are the most useful when an entire function written in C can be replaced with an equivalent C-callable assembly function call. But in many applications such easily replaceable functions are difficult to find, especially for applications that do not use any of the kernels such as *g711* speech coding and *adpcm* benchmarks. The Intel C/C++ compiler and the C62xx compiler provide intrinsics that inline SIMD (MMX and SSE) and VLIW (C62xx) assembly instructions respectively. The compilers allow the use of C variables instead of hardware registers and also schedule the instructions to maximize performance.

For creating the SIMD versions (for the Pentium II and III processors) of the benchmarks, I profiled the benchmarks to identify key procedures that can incorporate SIMD instructions. The major computation was then replaced with an equivalent set of SIMD instructions with original functionality maintained. I unrolled the loops manually to isolate multiple iterations of the loop body and then replaced with equivalent intrinsics.

The C62xx compiler similarly provides intrinsics for inlining assembly instructions into the C code. Some of the compiler intrinsics provided are “multiply two numbers, shift, and saturate”, “approximate reciprocal square root”, and “subtract lower and upper halves of two registers”. All of the compiler intrinsics and their detailed descriptions can be obtained from [102].

3.4 Performance Measures

Several performance measures are used throughout the dissertation for evaluating multimedia applications. Some of the performance measures are:

- Execution time speedup – performance improvement obtained by a technique ‘x’ over a technique ‘y’ for a given benchmark is calculated as execution time of ‘y’ divided by execution time of ‘x’. Techniques ‘x’ and ‘y’ vary depending on the experiment.
- IPC – instructions retired per cycle indicates the processors ability to overlap multiple instructions.
- CPI – cycles per retired instructions.
- Cache hit-rates – L1 and L2 cache hit rates are computed as the ratio of number of cache hits to the total number of cache accesses.

Chapter 4

Execution Characteristics of Multimedia Applications

This chapter characterizes the performance of commercial multimedia applications (categorized as 3D graphics, streaming video, and streaming audio) on an x86 processor based system. Architectural data pertaining to the utilization of various hardware resources on the chip are collected using on-chip performance counters. Execution characteristics of multimedia workloads are compared with SPEC and other desktop applications.

The rest of the chapter is organized as follows. Section 4.1 presents the various execution characteristics of commercial multimedia applications (from Table 3.1) on a Pentium II processor with MMX technology. I compare them with existing SPEC and SYSmark/NT characteristics presented in [9]. Section 4.2 presents an evaluation of SIMD and VLIW techniques for media and signal processing using a Pentium II and C62xx as representative processors. Section 4.3 summarizes the chapter.

4.1 Detailed Characterization of Multimedia Applications

I use a Pentium II processor with MMX technology operating at 300 MHz running Windows NT 4.0 for this characterization. The Intel Pentium II processor is a three-way superscalar architecture (capable of retiring up to three micro-instructions per cycle). It implements dynamic execution using an out-of-order, speculative execution engine, with register renaming of integer, floating-point and flag variables, carefully controlled memory access reordering, and multi-processing bus support [45]. Two integer units, two floating-point units, and one memory-interface unit allow up to five micro-ops to be scheduled per clock cycle. In addition, it provides the MMX execution unit for SIMD processing. There are two 64-bit MMX ALUs and one 64-bit MMX multiplier. The Pentium II used in this evaluation has 16 kB of L1 instruction and data caches and 512 kB of L2 cache. The rest of the section presents the execution characteristics of multimedia applications.

4.1.1 Cycles per instruction

Figure 4.1(a) shows the cycles per instruction (CPI) for each of the six individual multimedia applications (from Table 3.1). The geometric mean of the multimedia, SPECint95, SPECfp95, and SYSmark/NT benchmarks are shown in Figure 4.1(b). The geometric mean of the CPI for the multimedia workloads is

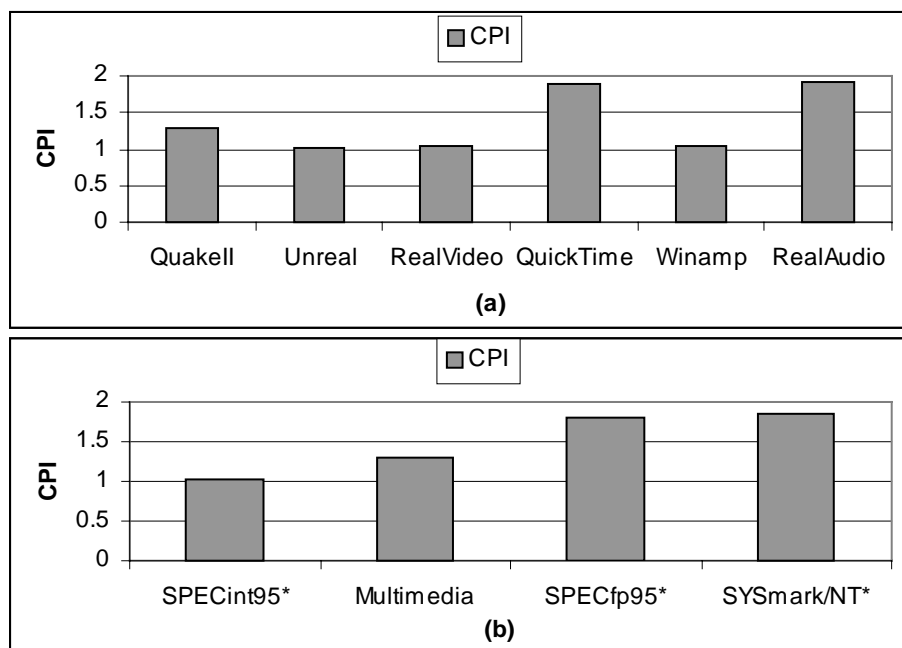


Figure 4.1. Cycles per instruction (a) for individual multimedia benchmarks and (b) comparison of media applications with other workloads [9]

1.31, which lies between the SPECint95 and the SPECfp95 benchmarks. Factors affecting CPI are discussed in detail in subsequent sections.

4.1.2 Resource and instruction-stream stalls

Figure 4.2 shows the I-stream stalls and resource stalls, measured in terms of the cycles in which the stall conditions occur. I-cache misses and ITLB misses cause I-stream stalls. Resource stalls show the number of cycles in which resources like register renaming or reorder buffer entries, memory entries, and

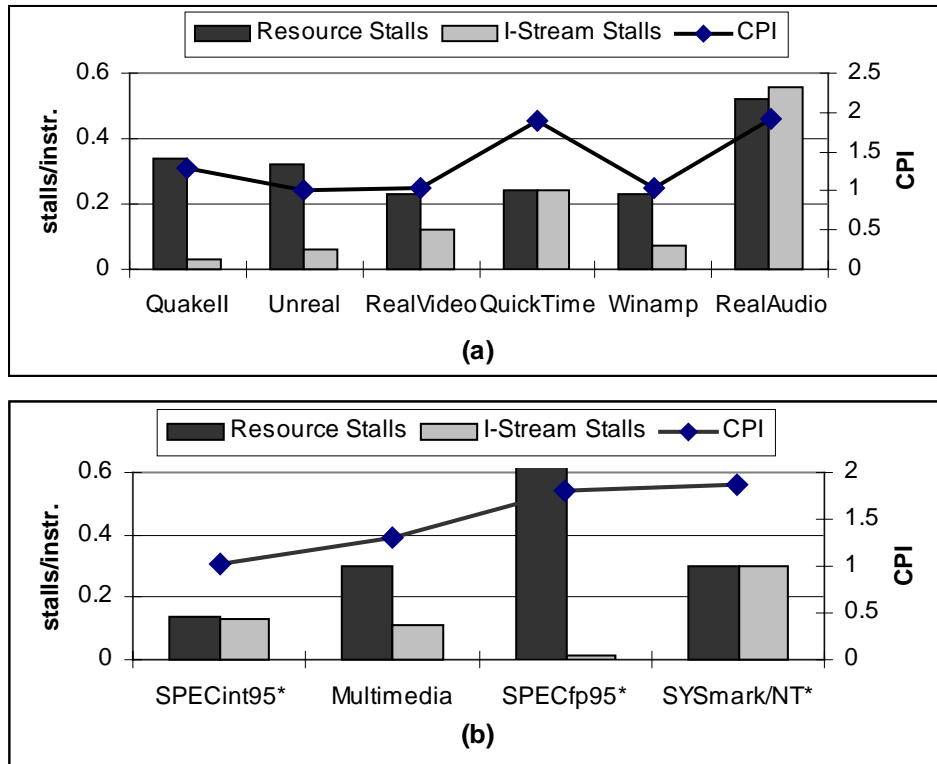


Figure. 4.2. Stalls per instruction (a) for individual multimedia benchmarks and (b) comparison of media applications with other workloads [9]

execution units are full; but these stalls may be overlapped with the execution latency of previously executing instructions.

The increase in CPI is directly proportional to the sum of I-stream and resource stalls as observed in Figure 4.2(a). *RealAudio* has the highest number of Resource and I-stream stalls and exhibits the largest CPI among the multimedia benchmarks. The geometric mean of the resource stalls for the multimedia workload is 0.30 and the I-stream stalls is 0.11. The number of resource stalls in

the case of the multimedia applications is over twice the number of stalls for the SPECint95 benchmarks. Resource stalls for the case of the SYSmark/NT is comparable to the multimedia benchmarks. SPECfp95 benchmarks incur significantly more resource stalls due to long dependency chains. Interestingly, the number of I-stream stalls per instruction of the multimedia benchmarks is similar to that of the SPECint95 and almost one-third of the SYSmark/NT benchmarks. The number of I-stream stalls for both 3D graphics applications is smaller than that of the audio and video applications.

The combined resource stall and I-stream stall ratios of the multimedia applications are between the SPECint95 and the SPECfp95 ratios correlating well with that the CPI of the multimedia applications, which lies in between the SPECint95 and SPECfp95 benchmark suites as was observed in the CPI ratios.

4.1.3 Branch statistics

Figure 4.3 shows the number of branches per instruction and the branch-mispredict ratio for each of the multimedia benchmarks. The multimedia programs have a branch per instruction ratio of 0.08. The ratio for SPECint95 is 0.17 and the ratio for SPECfp95 is 0.04. Branch statistics are not available for the SYSmark/NT, but Lee, *et al.* [61] report that desktop applications exhibit the same behavior as SPECint95 benchmarks with respect to average basic block size. While one out of every six instructions is a branch in the SPECint95

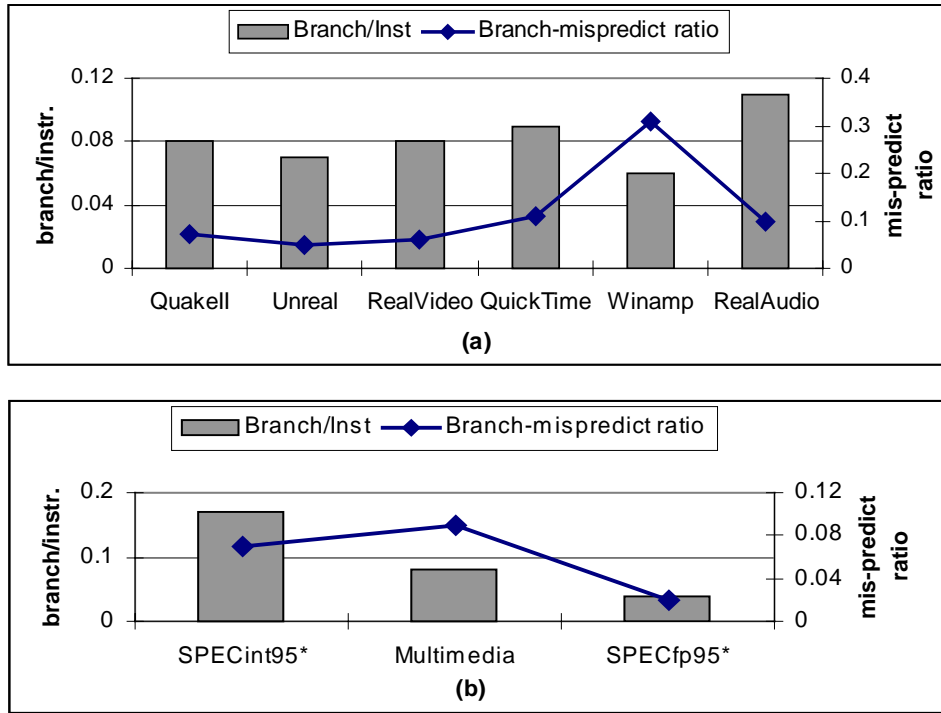


Figure. 4.3. Branch Statistics (a) for individual multimedia benchmarks and (b) comparison of media applications with other workloads [9]

benchmark suite, only one out of every 12.5 instructions is a branch in multimedia applications and one out of 25 instructions is a branch in the case of floating-point benchmarks. Thus the average available ILP of multimedia applications is potentially larger than the average ILP of SPECint95 programs. Moreover, in the case of these multimedia applications as will be explained later, MMX instructions operate on four data elements at the same time in a single instruction.

In spite of such a processing, the average basic block size of multimedia applications is over twice that of programs in the SPEC suite.

Fritts [34] reports that the average basic block size of multimedia applications in the MediaBench suite is similar to general-purpose integer applications. He observes that the average basic block size varies significantly from one media benchmark to the other.

Approximately 7% of all branches are mispredicted in SPECint95 and 2% in SPECfp95, while in multimedia applications 9% of all branches are mispredicted. The number of mispredicted branches ranges from about 2 to 40 per thousand instructions for the integer benchmarks, about 0.1 to 4 for the floating-point benchmarks and about 3.5 to 16 for the multimedia benchmarks. The multimedia benchmarks has a BTB miss ratio of 0.15. SPECint95 has a BTB miss ratio of 0.18 and SPECfp95 has a BTB miss ratio of 0.07.

In spite of having a better average basic block size, the CPI for multimedia applications is more than that of the integer benchmarks. The negative effect of having higher resource stalls is seemingly more than the positive effect of fewer branches per instruction. In the case of floating-point benchmarks, longer latencies of floating-point operations added with higher resource stalls increase the CPI considerably even when the number of branches is far less than any other types of benchmarks.

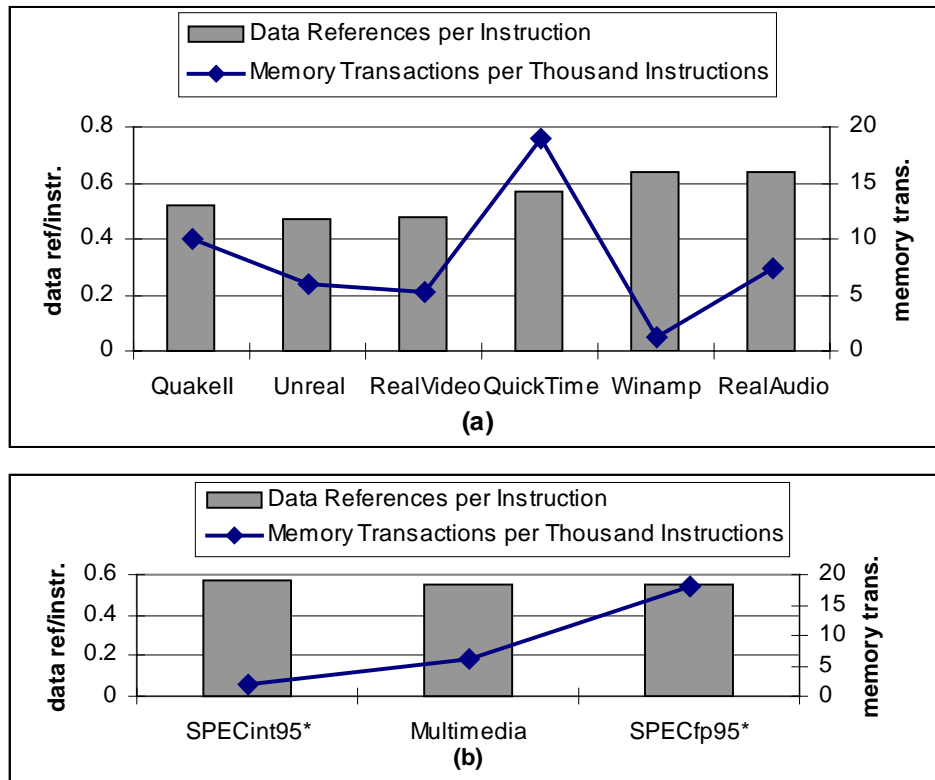


Figure 4.4. Memory Reference Statistics (a) for individual multimedia benchmarks and (b) comparison of media applications with other workloads [9]

4.1.4 Data memory references

Figure 4.4 shows the number of data references per instruction and the number of memory transactions per thousand instructions. On average, multimedia, SPECint95 and SPECfp95 benchmarks generate about 0.55 data references every instruction. Data memory reference statistics are not available for the SYSmark/NT, but Lee, *et al.* [61] report that desktop applications exhibit ratios similar to SPECint95 benchmarks. The IA-32 architecture results in more data

references than most RISC architectures because it has fewer registers (8 versus 32) [9].

Memory transactions arise from fetching of missed data/instructions and write-back of dirty blocks during replacement. The number of memory transactions per thousand instructions is higher in general if the miss rate of the L2 cache is higher (discussed below in Section 4.1.5). Multimedia applications have a higher number of memory transactions per thousand instructions than SPECint95, but lower than SPECfp95 benchmarks.

4.1.5 Cache behavior

The Pentium II processor consists of separate 16 kB four-way set associative L1 data and instruction caches with a cache line length of 32 bytes. The caches employ a write-back replacement policy and a pseudo-LRU replacement algorithm. The data cache consists of eight banks interleaved on four-byte boundaries. The data cache can be accessed simultaneously by a load instruction and a store instruction, as long as the references are to different cache banks. The latency for a load on an L1 cache hit is 3 cycles and four simultaneously outstanding misses can be handled. The Pentium II processor has a unified 512 kB four-way set associative cache. Table 4.1 shows the data cache miss rates of the multimedia applications.

Table 4.1 Data cache miss rates of multimedia applications
(16 kB 4-way separate L1, 512 kB 4-way unified L2 cache)

Benchmark	L1 miss rate	Local L2 miss rate	Global L2 miss rate
QuakeII	2.1 %	53.4 %	1.2 %
Unreal	1.5 %	25.8 %	0.5 %
RealVideo	2.0 %	19.9 %	0.6 %
QuickTime	1.8 %	15.1 %	0.6 %
Winamp	1.8 %	3.6 %	0.1 %
RealAudio	3.8 %	8.8 %	0.7 %

On the average, multimedia applications achieve a 98% L1 (16 kB) data cache hit rate (SPECint95 and desktop workloads have similar hit rates [9][61]) and only 0.5% of the processor data accesses miss in the L2 cache (global miss rate). The local miss rate in Table 4.1 corresponds to number of misses to the L2 cache divided by the number of L2 cache accesses. Figure 4.5(a) shows the L1 data and instruction and L2 cache misses per thousand instructions for the six multimedia benchmarks. The L1 (16 kB, 4-way) data and instruction cache misses per thousand instructions for the SPEC95 and SPEC2000 integer and floating-point suites are compared with the multimedia applications in Figure 4.5(b).

The streaming video and audio benchmarks incur more L1 instruction cache misses than the two 3D graphics applications. The L1 data and instruction

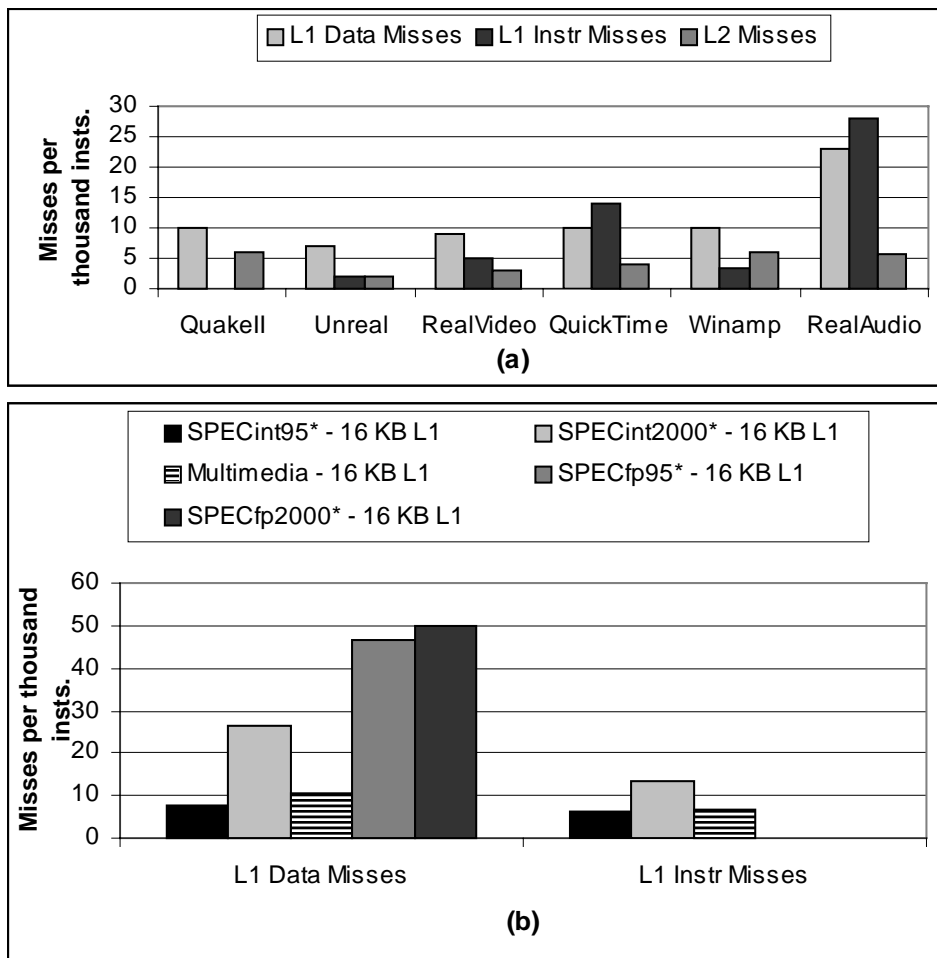


Figure 4.5. Cache Statistics (a) for individual multimedia benchmarks and (b) comparison of media applications with other workloads [9][83]

cache misses per thousand instructions of multimedia applications are slightly higher than that of SPECint95 benchmarks, but lower than that of SPECint2000 benchmarks. The SPECfp programs are dominated by loops, which results in a very predictable control flow, and have excellent L1 instruction cache perform-

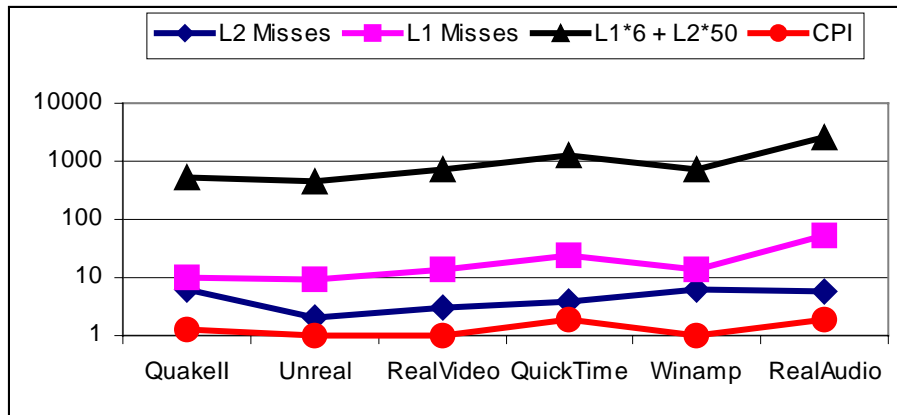


Figure 4.6. Log plot of CPI versus L1 and L2 cache misses

ance as reflected in the minimal cache misses per thousand instructions. However, the L1 data cache misses per thousand instructions for SPECfp programs are significantly worse than multimedia applications. My results indicate that data caches are used reasonably by multimedia applications as opposed to the popular notion that caches are ineffective for media applications due to their streaming nature. Figure 4.6 shows the correlation between the CPI and L1 (data and instruction) and L2 cache misses using a miss latency of 6 cycles for the L1 and 50 cycles for the L2 for the multimedia applications.

4.1.6 Floating-point operations

Figure 4.7 shows the amount of floating-point computation being performed in each of the six multimedia benchmarks. Except for *Winamp* and *RealAudio*, the rest of the benchmarks contain less than 5% floating-point related instructions.

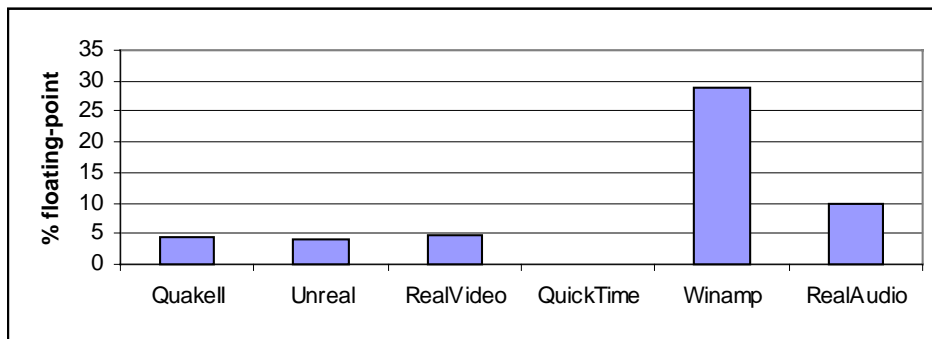


Figure. 4.7. Percentage of floating-point instructions

In fact, the two 3D graphics applications (*QuakeII* and *Unreal*) use integer computations as opposed to floating-point operations (most of the floating-point computation for 3D graphics applications is performed by graphics accelerator cards in desktop and workstations). Fritts [34] reports that multimedia applications in the MediaBench suite have few floating-point operations as well.

4.1.7 Multimedia extensions (MMX)

Multimedia applications can exploit available data parallelism by using SIMD extensions such as MMX technology. Unfortunately, not all media applications make use of MMX instructions because either they were developed before MMX technology was introduced or before compilers could take advantage of the MMX extensions. Compiler technology has yet to catch up with SIMD processing. The percentage of MMX instructions in each of the benchmarks is shown in Figure 4.8. MMX is especially suited for audio applications, and hence

I expected *RealAudio* and *Winamp* to take advantage of MMX instructions. Surprisingly, neither of them uses any MMX instructions. Moreover, *RealAudio* is a component of *RealPlayer*, which also has *RealVideo* as one of its components. While *RealVideo* uses MMX instructions, *RealAudio* fails to use any. *QuakeII* was developed before MMX was announced and hence it does not make use of MMX instructions. *Unreal* on the other hand is a recent game and uses MMX technology heavily. Nearly half of all the dynamic instructions in *Unreal* are MMX related.

The total number of MMX instructions can be sub-divided into 6 categories: packed multiply, packed shift, pack operations, unpack operations, packed logical operations, and packed arithmetic operations. The overhead involved in MMX computations is the packing and unpacking of instructions. Figure 4.9 shows the overhead percentage in each of the benchmarks. The overall overhead

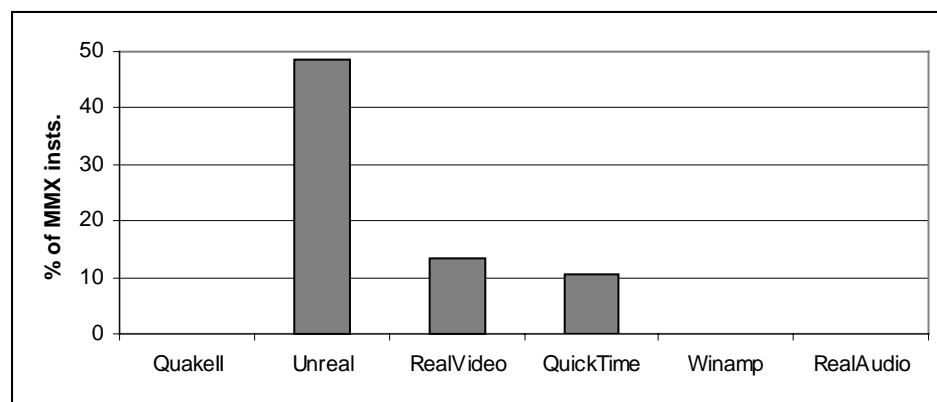


Figure. 4.8. Percentage of MMX related instructions

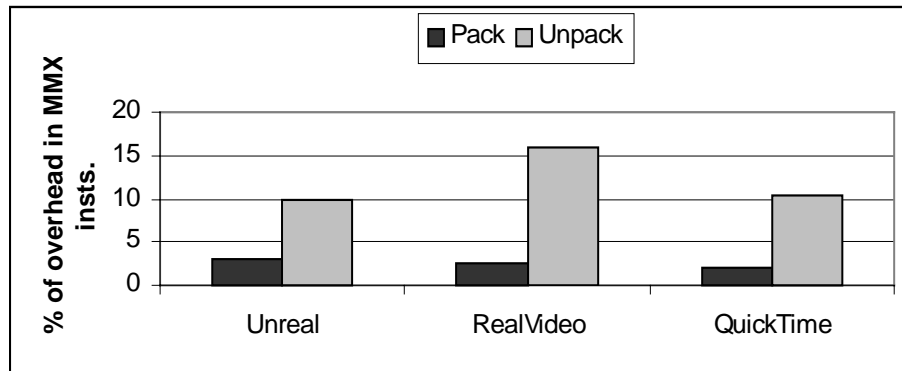


Figure. 4.9. Packing and unpacking instructions as a percentage of all MMX instructions

associated in MMX instructions is less than 20% (of MMX instructions) for *RealVideo* and less than 15% for *QuickTime* and *Unreal*. It is interesting to note that the unpacking overhead is several times the packing overhead. Nevertheless, the benefit of using MMX usually exceeds the overhead associated with packing and unpacking of instructions for MMX. *Unreal* has the option of disabling MMX instructions. It was observed that the number of frames per second when using MMX was 1.35 times greater than when not using MMX.

4.1.8 Speculative execution factor and UOPS per instruction

In the P6 microarchitecture, the instruction fetch unit fetches 16 bytes every clock cycle from the I-cache and delivers them to the instruction decoder. Three parallel decoders decode this stream of bytes and convert them into triadic UOPS. Most instructions are converted directly into single UOPS, some are de-

coded into one-to-four UOPS, and the complex instructions require microcode. Up to 5 UOPS can be issued every clock cycle to the various execution units, and up to 3 UOPS can be retired every cycle. If a branch is incorrectly predicted, then the speculated instructions down the mispredicted path are flushed. The speculative execution factor is defined as the number of instructions decoded, divided by the total number of instructions retired. Figure 4.10 shows the number of micro-operations per instruction and the speculative execution factor for each of the multimedia benchmarks and other workloads.

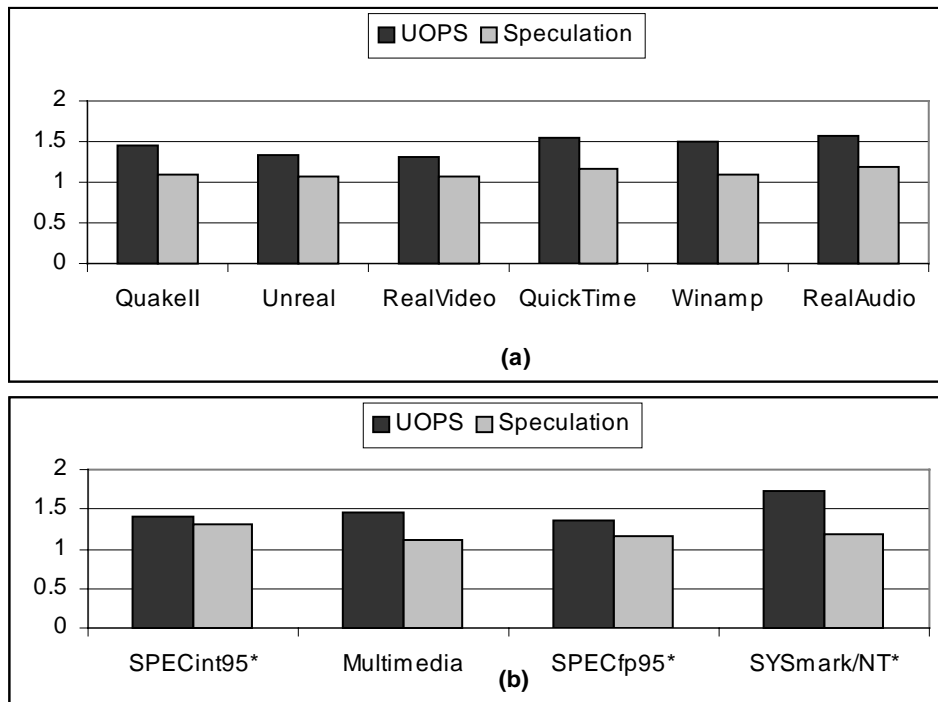


Figure. 4.10. Average number of UOPS per instruction and Speculation execution factor (a) for individual multimedia benchmarks and (b) comparison of media applications with other workloads [9]

In the multimedia applications one x86 instruction results in an average of 1.4 micro-ops, which is very similar to the behavior of SPECint95 and SPECfp95 programs. However, the SYSmark/NT applications have a high UOPS per instruction ratio. The speculation execution factor of multimedia applications is the lowest of all the four different workloads illustrating no significant mis-speculation ratio.

4.2 A Comparison of SIMD and VLIW Approaches for Media Processing

In this section, I evaluate the performance of the SIMD paradigm using Intel's Pentium II processor with MMX and the VLIW paradigm using Texas Instrument's TMS320C62xx processor on a subset of DSP and media benchmarks from Table 3.2. I briefly describe the C62xx processor followed by analysis of the results.

4.2.1 TMS320C62xx DSP processor

Texas Instruments TMS320C62xx, the first general-purpose VLIW DSP processor, is a 32-bit fixed-point chip. It is capable of executing up to eight 32-bit instructions per cycle. The C62xx processor has eight functional units that are grouped into two identical sets of four units each, and two register files, as

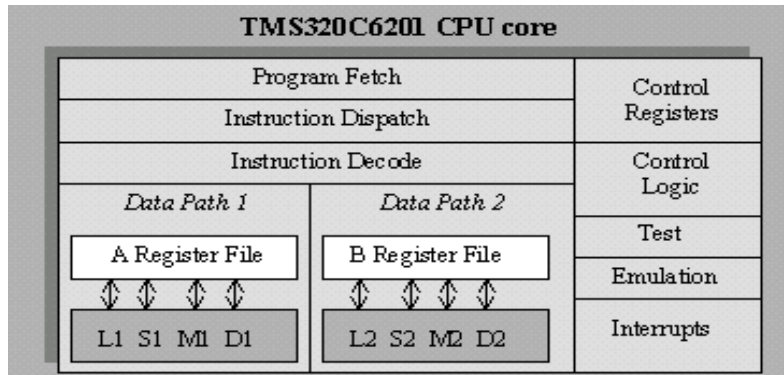


Figure. 4.11. CPU core of the TMS320C62xx processor (courtesy of TI)

shown in Figure 4.11. The functional units are the D unit for memory load/store and add/subtract operations; the M unit for multiplication; the L unit for addition/subtraction, logical and comparison operations; and the S unit for shifts in addition to add/subtract and logical operations. Each set of four functional units has its own register file, and a cross path is provided for accessing both register files by either set of functional units. The interested reader is referred to [100] for more architectural details of the C62xx processor.

4.2.2 Results

I profile a subset of six benchmarks for this evaluation (three kernels – *dotp*, *auto*, *fir* and three applications – *aud*, *g711*, *adpcm*). There are three versions of each benchmark (SIMD – Pentium II with MMX, VLIW – C62xx, and non-

SIMD – Pentium II without MMX). The baseline processor is a Pentium II processor without MMX (non-SIMD). Figure 4.12 illustrates the performance of SIMD and VLIW code over the non-SIMD version. The execution time is presented in Table 4.2. While interpreting the results, it should be remembered that the baseline (non-SIMD) performance is derived from a 3-way superscalar processor that performs dynamic scheduling to exploit ILP.

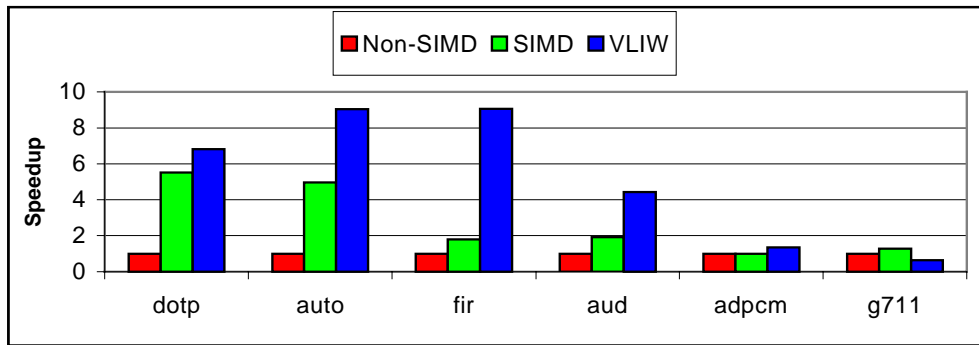


Figure. 4.12. Ratios of execution times of SIMD and VLIW processors

Table. 4.2 Execution Clock Cycles for SIMD and VLIW processors

Benchmark	Non-SIMD (cycles)	SIMD (cycles)	VLIW (cycles)
<i>dotp</i>	181242573	32804388	26600107
<i>auto</i>	222023315	44738100	24577801
<i>fir</i>	374628170	208238181	41370004
<i>aud</i>	2191761094	1148164486	494700006
<i>adpcm</i>	381143255	381143255	281980004
<i>g711</i>	109593602	85404734	173190004

The VLIW processor is capable of executing up to eight instructions per cycle and the SIMD unit is capable of executing four or eight operations per cycle for 16- or 8-bit data respectively. Significant speedup is achieved for both SIMD and VLIW versions over the non-SIMD code for the three kernels. The *dotp* kernel shows an improvement of approximately 5.5 times for the SIMD version over the non-SIMD version, despite using 16-bit data. Super-linear speedup is possible due to the presence of the pipelined multiply-accumulate instruction in MMX (throughput of 1 cycle and latency of 3 cycles). For the non-SIMD case, the integer multiply operation takes 4 cycles. Over 80% of the dynamic instructions in the case of the SIMD version have been found to be MMX-related instructions. The performance of the VLIW version of *dotp* is even better than the SIMD version, with a speedup close to 7 times. The VLIW code is capable of executing two data elements per clock cycle (in the case of a 1-way scalar processor it would take at least 5 clock cycles for each data element – two for loads, one multiply, one add and one store). Moreover, the C62xx code takes advantage of software pipelining to prefetch data three iterations before it is used.

The *auto* kernel also shows similar performance increase for both the SIMD and VLIW versions. As in the case of the *dotp*, *auto* uses several multiply and accumulates. For the SIMD case, 88% of the dynamic instructions are MMX-related instructions. In the case of the VLIW processor, over 90% of the

fetch packets have only one execute packet (indicating eight instructions are able to execute in parallel). A majority of the remaining 10% of the fetch packets has only two execute packets (indicating an average of four instructions in parallel).

The *fir* benchmark shows a modest performance increase (1.8 speedup) for the SIMD version over the non-SIMD code when compared to the other two kernels. The amount of MMX related instructions in the overall dynamic stream are far less than the other two kernels (29%). Also, the SIMD version needs four copies of filter coefficients to avoid data misalignment. The Intel library version of the *fir* filter actually exhibited a speedup of only 1.6. This was due to additional data structures that had to be defined and error checking code that can potentially decrease performance, which results from improved robustness. The VLIW version exhibits a stronger performance boost than the SIMD version. Again, as was in the case of *dotp* and *auto*, over 95% of the fetch packets had only one execute packet with all eight instructions executing in parallel. The VLIW kernel codes were hand optimized and presented as assembly libraries. Moreover, the VLIW code had constraints such as the number of filter coefficients should be a multiple of 4 and the size of the *auto* vector should be a multiple of 8.

The results of the VLIW versions of the applications are disappointing (when compared to performance improvements obtained in kernels). Both *g711*

and *adpcm* involve significant control dependent data dependencies, wherein execution is based either on table lookup or conditional branch statements based on immediately preceding computations. The *aud* application was the only one where any appreciable parallelism could be exploited by the VLIW environment. The VLIW version of the *aud* application exhibits a speedup close to 4.5 over the non-SIMD version. The echo effects and signal mixing components of the VLIW version were unrolled manually eight times. The speedup achieved by the VLIW version of the *aud* application is almost half that of the kernels. This is because the C62xx version was primarily developed in C code and only the filtering component utilized optimized assembly code. The compiler generates the echo effects and signal mixing components.

The “interlist” utility of the C62xx compiler provides the user with the ability to interweave original C source code with compiler-generated assembly code. The compiler-generated assembly code for the echo effects and signal mixing components indicates that the compiler is unable to fill all the pipeline slots (several execute packets in each fetch packet). The compiler was unable to software pipeline the echo effects component. This effectively introduced 3 NOPs after every load, which degraded performance. Moreover, even with a loop unrolling of 8, for each one of the eight computations the result was the same with 3 NOPs after every load. Since there is no out-of-order execution in

the VLIW processor, loop unrolling in this instance contributes to no performance increase in terms of speed but only increases code size.

The VLIW code for *adpcm* shows a speedup of 1.35 over the non-SIMD and SIMD cases. In this application, the C62xx compiler did not perform any loop unrolling or software pipelining. Since there is no parallelism to be exploited, unrolling will drastically increase the code size with little or any performance increase. Software pipelining was difficult because loads in this application depended on the execution of the conditional branch statements. Thus the compiler-generated assembly code is non-optimal with several branches that are followed by 5 NOPs and loads followed by four NOPs. Most of the fetch packets have eight execute packets (serial as opposed to the desired parallel execution).

The VLIW version of *g711* shows a slowdown (0.63) over the non-SIMD code. However, analysis showed that the base non-SIMD model, which is a 3-way dynamically scheduled superscalar processor, achieves an IPC of approximately 2.0. The C62xx code for *g711* has very few packets with more than one slot utilized. Branches are followed by NOPs for 5 cycles in the assembly code. There are also several loads due to the look-up table and NOPs for 4 cycles are inserted in the code. Because of static scheduling combined with no branch prediction, and the control nature of the application, no parallelism could

be exploited. Also, the *g711* operates on 8-bit data and the rest of the 24-bits (the C62xx data width is 32-bits internally) is being wasted.

Even the speedup achieved by the application benchmarks from SIMD technology is not appreciable. The *aud* application shows a moderate speedup of around 2.0 for the SIMD code over the non-SIMD code. About 28% of the dynamic instructions are MMX-related. Loop unrolling of 4 was used for each of the echo effects, filtering and signal mixing portions of this application. The *adpcm* benchmark does not have any MMX instructions because this algorithm is inherently sequential in that each computation on a data sample depends on the result of the immediately preceding sample. The *g711* SIMD version exhibited a speedup of 1.28 over the non-SIMD code. The number of MMX related instructions are only around 4% and the performance increase is partly due to manual loop unrolling.

Overall, it is found that both SIMD and VLIW processors are able to extract DLP and ILP respectively in multimedia and DSP programs. VLIW processors have the advantage of exploiting ILP where DLP does not exist (*adpcm* benchmark). On the other hand, VLIW processors without dynamic scheduling and branch prediction are seen to be heavily dependent on the capabilities of the compiler (*g711* and *adpcm*) while dynamically scheduled superscalar processors can exploit ILP irrespective of the quality of the compiled code.

4.3 Summary

In this chapter, the execution characteristics of several commercial multimedia applications were evaluated under different domains – 3D graphics, streaming video, and streaming audio. Using built-in hardware performance counters, statistics such as CPI, branch frequency and mis-prediction rate, instruction stream and resource stalls, and cache behavior are compared with corresponding characteristics of SPEC and desktop benchmarks. The major observations are summarized below:

- The number of resource stalls per instruction for multimedia applications is twice that of SPECint95 workloads, similar to desktop workloads, and one-third of SPECfp95 workloads. The number of instruction stream stalls per instruction for multimedia applications is similar to that of SPECint95 workloads and one-third of desktop workloads. SPECfp95 workloads have negligible instruction stalls per instruction. Overall, the combined resource and I-stream stall ratios for media benchmarks lie between SPECint95 and SPECfp95/desktop workloads. A similar trend is observed in the CPI reflecting the influence of resource and I-stream stalls on CPI.
- The branch frequency of multimedia workloads is one-half of the SPECint95 workloads and twice that of SPECfp95 workloads. However, the branch

mispredict ratio for multimedia applications is higher than both SPECint95 and SPECfp95 applications.

- The number of data memory references per instruction on the x86 architecture is observed to be approximately 0.55 irrespective of the workload (SPEC, desktop, or multimedia).
- Contrary to the popular belief that data caches are ineffective for multimedia applications due to their streaming nature, it is found that multimedia benchmarks achieve a 98% L1 cache hit-rate (16 kB, 4-way) and a 99.5% global L2 hit-rate (512 kB, 4-way). Multimedia applications achieve a better L1 data cache performance (16 kB, 4-way) than SPECint2000, SPECfp95, and SPECfp2000 and slightly worse than SPECint95 workloads. The L1 instruction cache miss-rate is similar to SPECint95, but one-half of SPECint2000 benchmarks (SPECfp95 and SPECfp2000 exhibit near 100% L1 I-cache hit-rates).
- Most of the multimedia applications do not use significant floating-point operations, except the streaming audio applications, *Winamp* (30% floating-point instructions) and *RealAudio* (10% floating-point instructions).
- Three of the six benchmarks evaluated used MMX instructions (10-50%) and less than 20% of all MMX instructions are used for packing and unpacking operations.

- In media applications, one x86 instruction results in an average of 1.4 UOPS, very similar to SPECint95 and SPECfp95 benchmarks (desktop workloads exhibit an average of 1.7).
- The speculative execution factor for multimedia applications is lower than SPECint95, SPECfp95, and desktop workloads (SPECint95 has the highest speculative execution factor).
- VLIW processors can exploit ILP in programs that do not have DLP. However, they are heavily dependent on the compiler and performance on applications is relatively poor when compared to performance on kernels.

Chapter 5

Bottlenecks in Multimedia

Processing with SIMD style Extensions

SIMD extensions accelerate multimedia applications by exploiting DLP. While the improvement in performance has been encouraging and exciting, I notice that performance does not scale with increasing SIMD execution resources. The primary contribution of this chapter is the characterization of media workloads from the perspective of support required for efficient SIMD processing. Typically, studies have focused on the SIMD computation part of the algorithms. In this chapter, I focus on the supporting instructions and show that significant additional performance opportunities exist for SIMD GPPs.

I embark on a study to understand the behavior of multimedia applications on SIMD GPPs and evaluate DLP in multimedia applications. In this chapter I attempt to answer the following:

- SIMD GPPs typically exploit the sub-word parallelism between independent loop iterations in the inner loops of multimedia programs. Where does DLP

in media applications reside? Does most of the DLP reside in the inner loops, or is there significant DLP in the outer loops?

- Nested loops are required for processing multimedia data streams and this necessitates the use of multiple indices while generating addresses. GPPs contain limited support to compute addresses of elements with multiple indices. How many levels of nesting are required in common media algorithms? Are the addressing sequences primarily sequential?
- While SIMD extensions are capable of performing multiple computations in the same cycle, it is essential to provide data to the SIMD computation unit in a timely fashion in order to make efficient use of the sub-word parallelism. Providing data in a timely fashion requires supporting instructions for address generation, address transformation (data reorganization such as packing, unpacking, and permute), processing multiple nested loop branches, and loads/stores. Are these supporting instructions a dominant part of the instruction stream?
- What percentage of the peak computation rate is achieved for the SIMD execution units in GPPs? If the computation rate is low, what are the reasons that prevent the SIMD execution units from achieving a good computation rate?

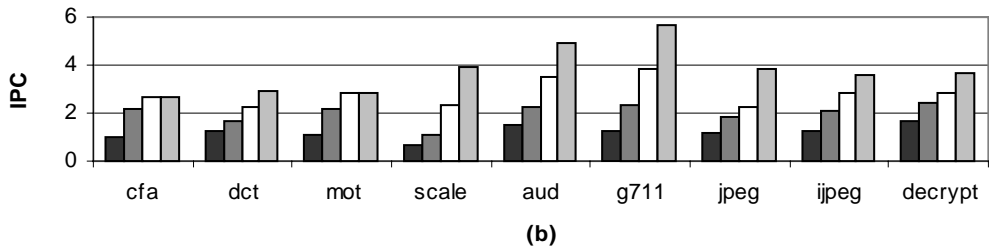
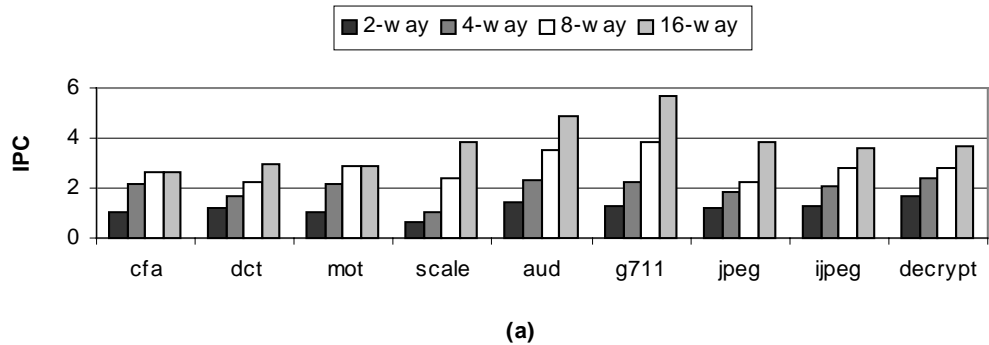
The rest of the chapter is organized as follows. Section 5.1 includes sensitivity experiments on the scalability of conventional ILP and DLP techniques. Section 5.2 describes studies to detect bottlenecks in the execution of SIMD programs. Section 5.2.1, describes the loop nesting and access patterns in multimedia applications and their mapping onto SIMD GPPs. Section 5.2.2 classifies dynamic instructions into two fundamental categories – the useful computation instructions and the overhead/supporting instructions and analyze their mix in media benchmarks. In Section 5.2.3, I measure the percent of peak computation rate achieved for the SIMD execution units in GPPs by conducting experiments on two different superscalar processors. Section 5.2.4 identifies additional bottlenecks in conventional ILP processors that limit the computation rate of the SIMD execution units. The chapter is summarized in Section 5.3.

5.1 A Scalability test

A logical approach to improve performance is to scale the processor resources to extract more parallelism. To understand the ability of wide out-of-order superscalar processors to increase performance of multimedia programs, I performed experiments scaling the various resources of the processor as in Table 5.1. Figure 5.1(a) shows the instructions per cycle (IPC) for different processor configurations for nine multimedia benchmarks.

Table. 5.1 Processor and memory configuration for the scalability test

Parameters	2-way	4-way	8-way	16-way
Fetch width, Decode width, Issue width, and Commit width	2	4	8	16
RUU Size	32	64	128	256
Load Store Queue	16	32	64	128
Integer ALUs (Latency/recovery = 1/1)	2	4	8	16
Integer Multipliers (Latency/recovery = 3/1)	1	2	4	8
Load/Store ports (Latency/recovery = 1/1)	2	4	8	16
L1 I-cache (Size in kB, hit time, Associativity, block size in bytes)	16, 1, 1, 32	16, 1, 1, 32	16, 1, 1, 32	32, 1, 1, 64
L1 D-cache (Size in kB, hit time, Associativity, block size in bytes)	16, 1, 4, 32	16, 1, 4, 32	16, 1, 4, 32	16, 1, 4, 32
L2 unified cache (Size in kB, hit time, Associativity, block size)	256, 6, 4, 64	256, 6, 4, 64	256, 6, 4, 64	256, 6, 4, 64
Main memory width	64 bits	128 bits	256 bits	256 bits
Main memory latency (First chunk, next chunk)	65, 4	65, 4	65, 4	65,4
Branch Predictor – bimodal (Size, BTB size)	2K, 2K	2K, 2K	2K, 2K	2K, 2K
SIMD ALUs	2	4	8	16
SIMD Multipliers	1	2	4	8



	cfa	dct	mot	scale	aud	g711	jpeg	ijpeg	decrypt
4-way	< 1 %	< 1 %	< 1 %	< 2 %	< 4 %	< 1 %	< 1 %	< 1 %	< 1 %
8-way	< 1 %	< 1 %	< 1 %	< 3 %	< 1 %	< 1 %	< 1 %	< 1 %	< 1 %
16-way	< 1 %	< 1 %	< 1 %	< 1 %	< 1 %	< 1 %	< 1 %	< 1 %	< 1 %

(c)

Figure. 5.1. Results of the scalability test. (a) IPC with both the SIMD and non-SIMD resources scaled, (b) IPC with non-SIMD resources scaled, but SIMD resources are constant (same as 2-way processor configuration) and (c) performance improvement of (a) over (b)

I, incidentally, also note that almost the same performance can be achieved even if the SIMD units were not scaled; i.e. the non-SIMD components are scaled up to the 16-way processor keeping the SIMD component constant as a 2-way processor (2 SIMD ALUs and 1 SIMD multiplier). The IPC for this case is depicted in Figure 5.1(b). The percentage increase in IPC when scaling

both the SIMD and non-SIMD resources over the case of scaling only the non-SIMD resources is shown in Figure 5.1(c). This experiment shows that there are several bottlenecks in multimedia processing using SIMD style extensions.

5.2 Identification of Bottlenecks

It is evident that there are several bottlenecks in SIMD style media processing and that it is not possible to achieve significant additional performance improvements by making the processor wider/bigger to extract more parallelism. I investigate characteristics of media programs that point towards the bottlenecks in current SIMD architectures.

5.2.1 Nested loops in multimedia applications

In this section the nature of multimedia loops is investigated to understand the levels of nesting, stride patterns, and the location of the parallelism. Desktop/workstation multimedia applications such as streaming video encoding/decoding (MPEG 1/2/4 and Motion JPEG), audio encoding/decoding (ADPCM, G.7xx, MP3, etc), video conferencing (H.323, H.261, etc), 3D games, and image processing (JPEG, filtering) typically operate on sub-blocks in a large 1- or 2-dimensional block of data. Audio applications operate on chunks of one-dimensional data samples at a time. For example, the MP3 codec operates on

“frames” which are smaller components of the complete audio signal that lasts a fraction of a second. Image and video applications operate on sub-blocks of two-dimensional data at a time. For example, the DCT algorithm operates on 8x8 pieces of data in a large image such as 1600x1200 pixels. Such a division of data into sub-blocks results in the data being accessed with different strides at various instances in the algorithm. Figure 5.2 depicts a 2-dimensional block of data that is accessed with four different strides – two in the vertical direction and two in the horizontal direction.

Source code for the aforementioned algorithms involves the usage of multiple nested loops (commonly ‘for’ loops in C language) to process the data streams. Much of the available parallelism in multimedia applications is DLP

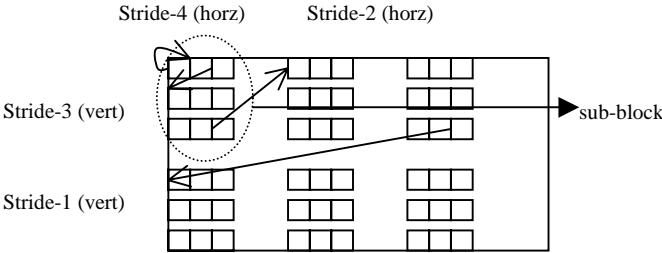


Figure. 5.2. A 2-D data structure in which sub-blocks of data are processed. Each sub-block requires two strides (one each along the rows and columns of the sub-block, namely stride-4 and stride-3). Additional two strides (stride-2 and stride-1) are required for accessing different sub-blocks in the horizontal and vertical direction.

that resides at the various levels of nesting. The dimensions of each sub-block for most multimedia algorithms are small (filtering typically uses 3x3 or 5x5 or 5x7 sub-blocks, DCT operates on 8x8 sub-blocks, and motion estimation operates on 16x16 sub-blocks) resulting in limited parallelism in the innermost loop. However, the number of sub-blocks themselves is large since the size of the data stream can be on the order of several MB. Consequently, a significant part of the DLP in multimedia applications resides outside the innermost loop, unless applications are coded differently.

Existing GPPs with SIMD extensions exploit DLP between independent loop iterations in the innermost loops leading to significant untapped available DLP in multimedia applications. Figure 5.3 shows the SIMD C-code implementation of the DCT (the DCT is a major component in JPEG image and MPEG video coding) which operates on 8x8 sub-blocks in an image of a given height and width. The second matrix is transposed before doing the computation because accessing the second matrix in column-major order results in a significant amount of overhead. This is particularly true when using SIMD instructions because a SIMD register needs to be packed with an element from different rows (and hence not contiguous). If a SIMD register holds eight elements, then all eight rows of a matrix need to be loaded into the cache and then elements belonging to the same column are packed into the register. It is possible to eliminate one of the transpose operations (either from row or column 1D-DCT) if a

```

void 2D_DCT(IMAGE[IMAGE_WIDTH][IMAGE_HEIGHT])
{
  for(i = 0; i < IMAGE_HEIGHT/8; i++)
    for(j = 0; j < IMAGE_WIDTH/8; j++)
    {
      /* perform 1D row and column DCT */

      /* output[8][8] = dct_coeff[8][8] * block[8][8] *
         dct_coeff[8][8]T */

      /*
      1D_ROW_DCT (dct_coeff [8][8], block [8][8]);
      1D_COL_DCT (block [8][8], dct_coeff [8][8]T);
      */
    }
}

```

```

void 1D_XXX_DCT(DCT_COEFF[8][8], BLOCK[8][8])
{
  Transpose (BLOCK [8][8]);
  for(k = 0; k < 8; k++)
  {
    for(l = 0; l < 8; l++)
    {
      temp = 0;
      for(m = 0; m < 8/SIMD_WIDTH; m++)

        temp +=
        SIMD_MUL (dct_coeff [k][m], block [l][m]);

      output[k][l] = SIMD_REDUCE (temp)
    }
  }
}

```

Figure 5.3. C-code for 2D-DCT implementation

transposed version of the DCT coefficients is available. In Figure 5.3, there are a total of five nested for-loops for the DCT routine. Current SIMD instructions

exploit DLP in the innermost for-loop (variable ‘m’). The number of iterations would be scaled down according to the width of the available SIMD datapath (currently 64 or 128 bits wide) and size of each element (8-bit, 16-bit, or 32-bit).

Next, the access patterns in media applications are studied. Analysis of media and DSP applications unveils invocation of several address patterns, often multiple simultaneous sequences [8]. Figure 5.4 shows the typical access patterns in media and DSP kernels. Table 5.2 lists several key multimedia and DSP kernels and the typical number of nested loops required along with their corresponding primitive addressing sequences.

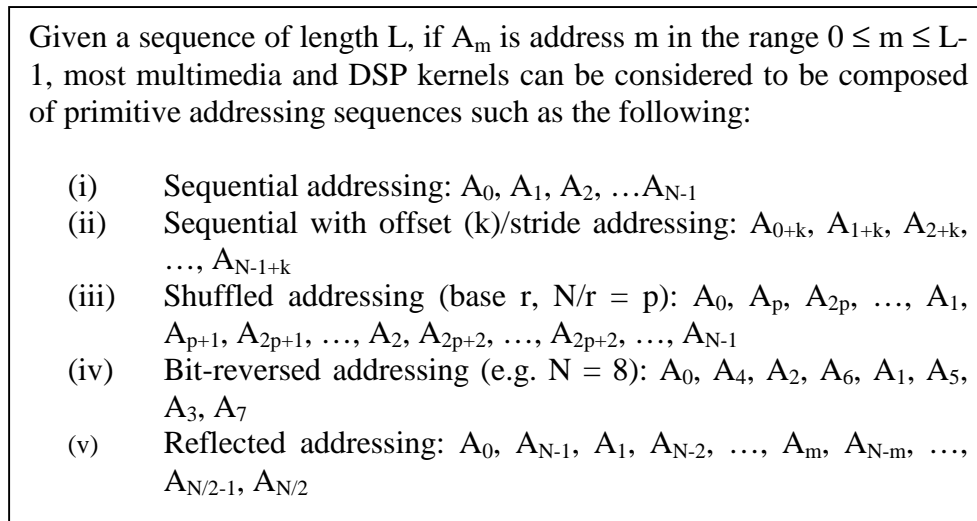


Figure. 5.4. Typical access patterns in multimedia and DSP kernels [8]

Table. 5.2 Summary of key media algorithms and the required nested loops along with their primitive addressing sequences

Multimedia/DSP algorithm	Nested loops	Addressing Sequences
Discrete Cosine Transform (JPEG & MPEG coding)	5	Sequential and sequential with multiple offsets/strides
Motion Est./Comp. (MPEG, H.263, etc)	5	Sequential and sequential with multiple offsets/strides
Wavelet Transform (JPEG2000)	> 5	Sequential and sequential with multiple offsets/strides
Color Space Conversion (JPEG, MPEG, 3D graphics)	> 4	Sequential, sequential with offsets, and shuffled
Scaling and matrix operations (image/video)	3	Sequential and sequential with multiple offsets/strides
Fast Fourier transform	> 3	Shuffled and bit-reversed
Color Filter Array, median filtering, correlation	2 – 5	Sequential and sequential with multiple offsets/strides
Convolution, FIR, and IIR filtering	3 – 4	Sequential, sequential with offsets, and reflected
Edge detection, alpha saturation (image/video)	2 – 5	Sequential and sequential with multiple offsets/strides
Up/Down sampling, 3-D transformation (graphics)	3 – 5	Sequential and sequential with multiple offsets/strides
Quantization (JPEG, MPEG)	2 – 4	Sequential and sequential with multiple offsets/strides
ADPCM, G.711 (speech)	2 – 3	Sequential and sequential with multiple offsets/strides

Hardware to generate multiple address sequences is not overly complicated, but supporting them using general-purpose instruction sets is not very ef-

ficient, as the available addressing modes are limited. Furthermore, there is not enough support for keeping track of multiple indices/strides efficiently in GPPs. Similarly, keeping track of multiple loop nests/bounds involves a combination of several addressing modes and instructions.

Thus, even though GPPs are enhanced with SIMD extensions to extract DLP in multimedia programs, there is a mismatch between the requirements of media applications (for address generation and nested loops) and the ability of GPPs with SIMD extensions. Simple ASICs can perform these tasks efficiently, however, loss of programmability and flexibility is a weakness of that approach.

5.2.2 Overhead/supporting instructions

The discussion in the previous section points to the need of several instructions to compute addresses and otherwise support the core SIMD computations. In this section, I analyze the media instruction stream by focusing on the two distinct sets of operations: the **useful computations** as required by the algorithm and the **supporting instructions** such as address generation, address transformation (data movement and data reorganization such as packing and unpacking), loads/stores, and loop branches. Consider the DCT code in Figure 5.3. The useful computation instructions for the DCT routine are the multiply (of DCT coefficients and data) and the accumulate operations (addition of multiplied values). This is shown in bold in Figure 5.3. All the other instructions are denoted

as **overhead**; their sole purpose is to aid in the execution of the useful computation instructions. Many of them arise due to the programming conventions of general-purpose processors, abstractions and control flow structures used in programming, and mismatch between how data is used in computations versus the sequence in which data is stored in memory. A similar kind of classification of instructions into access and execute instructions was performed in DAE processors [88][89]. In my classification, the overhead component includes loop branches and reduction operations [22] that are specific to multimedia applications (e.g. packing, unpacking, and permute) in addition to the memory access instructions. The instructions contributing to the overhead are:

- Address generation – considerable processing time is dedicated in performing the address calculations required to access the components of the data structures/arrays, which is sometimes called address arithmetic overhead.
- Address transformation – transforming the physical pattern of data into the logical access sequence (transposing the matrix in Figure 5.3, packing/unpacking data elements in SIMD computations, and reorganizing in other ways).
- Loads and stores – data is not always available in registers and has to be fetched from memory or stored to memory, the so-called access overhead.

- Branches – performing control transfer (for each of the nested for-loops).

Figures 5.5 and 5.6 show the assembly code classified into useful computation and overhead instructions (for the Pentium III and SimpleScalar based processors) for the 1D-DCT routine from Figure 5.3 (excluding the transpose function), i.e. the three inner level nested loop structure. Transposing the second matrix before multiplication will necessitate additional overhead instructions for address transformation.

From Figures 5.5 and 5.6, it can be seen that a significant number of overhead/supporting instructions are necessary to feed the SIMD computation units. In order to quantify the amount of overhead/supporting instructions in multimedia programs, the performance of six benchmarks (*cfa*, *dct*, *mot*, *scale*, *aud*, and *g711*) is evaluated. *Jpeg*, *ijpeg*, and *decrypt* are not used in this experiment because the source code for these three benchmarks includes initialization routines and file I/O. Five of the six benchmarks (except *g711*) were mapped in such a way that the SIMD execution units perform every useful computation. Figure 5.7 shows the breakdown of dynamic instructions into various classes (memory, branch, integer, SIMD overhead, and SIMD computation).

Pentium III – SIMD code		
lea	ebx, DWORD PTR [ebp+128]	load/address overhead
mov	DWORD PTR [esp+28], ebx	load/address overhead
\$B1\$2:		
xor	eax, eax	address overhead
mov	edx, ecx	address overhead
lea	edi, DWORD PTR [ecx+16]	load/address overhead
mov	DWORD PTR [esp+24], ecx	load/address overhead
\$B1\$3:		
movq	mm1, MMWORD PTR [ebp]	load overhead
pxor	mm0, mm0	initialization overhead
pmaddwd	mm1, MMWORD PTR [eax+esi]	Computation
movq	mm2, MMWORD PTR [ebp+8]	load overhead
pmaddwd	mm2, MMWORD PTR [eax+esi+8]	Computation
add	eax, 16	address overhead
paddw	mm1, mm0	Computation
paddw	mm2, mm1	Computation
movq	mm0, mm2	load related overhead
psrlq	mm2, 32	SIMD reduction overhead
movd	ecx, mm0	SIMD load overhead
movd	ebx, mm2	SIMD load overhead
add	ecx, ebx	SIMD conv. Overhead
mov	WORD PTR [edx], cx	store overhead
add	edx, 2	address overhead
cmp	edi, edx	branch related overhead
jg	\$B1\$3	loop branch overhead
\$B1\$4:		
mov	ecx, DWORD PTR [esp+24]	load/address overhead
add	ebp, 16	address overhead
add	ecx, 16	address overhead
mov	eax, DWORD PTR [esp+28]	load/address overhead
cmp	eax, ebp	branch related overhead
jg	\$B1\$2	loop branch overhead

Figure. 5.5. Pentium III optimized assembly code for the 1D-DCT routine shown in Figure 5.2 (excluding matrix transpose)

Simplescalar-SIMD – gcc code		
move	\$11,\$0	address overhead
l.d	\$f6,\$LC1	load overhead
\$L33:		
move	\$10,\$0	address overhead
move	\$9,\$5	address overhead
\$L37:		
mtc1	\$0,\$f4	initialization overhead
mtc1	\$0,\$f5	initialization overhead
move	\$8,\$0	address overhead
move	\$7,\$9	address overhead
move	\$3,\$4	address overhead
\$L41:		
l.simd	\$f0,0(\$3)	SIMD load overhead
l.simd	\$f2,0(\$7)	SIMD load overhead
mul.simd	\$f0,\$f0,\$f2	Computation
addu	\$8,\$8,1	address overhead
add.simd	\$f4,\$f4,\$f0	Computation
slt	\$2,\$8,2	branch related overhead
addu	\$7,\$7,8	address overhead
addu	\$3,\$3,8	address overhead
bne	\$2,\$0,\$L41	loop branch overhead
redu.simd	\$f4,\$f4,\$f6	SIMD reduction overhead
addu	\$9,\$9,16	address overhead
addu	\$10,\$10,1	address overhead
slt	\$2,\$10,8	branch related overhead
s.simd	\$f4,0(\$6)	SIMD store overhead
bne	\$2,\$0,\$L37	loop branch overhead
addu	\$6,\$6,16	address overhead
addu	\$4,\$4,16	address overhead
addu	\$11,\$11,1	address overhead
slt	\$2,\$11,8	branch related overhead
bne	\$2,\$0,\$L33	loop branch overhead

Figure. 5.6. Simplescalar optimized assembly code for the 1D-DCT routine shown in Figure 5.2 (excluding matrix transpose)

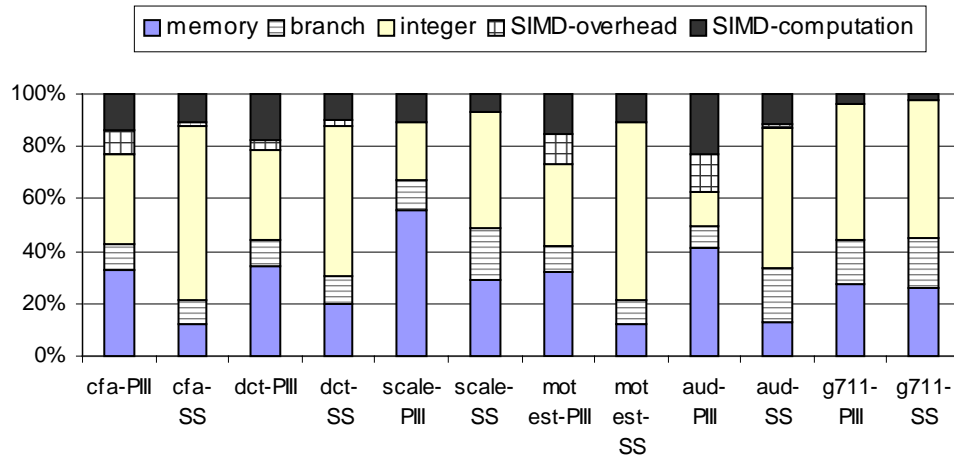


Figure. 5.7. Breakdown of dynamic instructions into various classes

The overhead/supporting instructions that are required to assist the SIMD computation (useful computations) instructions dominate the dynamic instruction stream (75-85%). A significant number of instructions are required for processing the loop branches and computing the strides for accessing the data organized in sub-blocks.

5.2.3 SIMD throughput and efficiency

In this section, the throughput of the SIMD units is evaluated to understand the impact of the overwhelming number of instructions needed to support the SIMD computations. I define SIMD efficiency as the ratio of the execution cycles **ideally** necessary for the useful computation instructions to the overall execution cycles **actually** consumed. In other words, SIMD efficiency indicates what frac-

tion of the peak throughput of the SIMD units is actually achieved. The actual execution cycles are obtained by measurement with processor performance counters or by simulation, while the ideal cycles are computed assuming that the overhead instructions can be perfectly overlapped with the useful computation instructions. In the ideal case, overhead instructions such as address generation, memory access, data reorganization, and loop branches do not consume additional processor cycles. The number of ideal execution cycles depends on the amount of SIMD resources in a machine. For example, consider a matrix multiplication algorithm of two $N \times N$ matrices, with computational complexity $O(N^3)$. Further, this assumes that the processor contains one multiplier, which is pipelined, and that the addition/accumulation can take place in parallel. Thus, an 8×8 matrix multiply should take 512 cycles on a machine with one multiplier (in the pure dataflow model), and take 128 cycles on a machine with 4 multipliers (assuming that there are at least 4 adders for the accumulation). If this algorithm were to take 2500 cycles on a real machine with one multiplier, then the efficiency of computation is 20% ($512/2500$). If efficiency achieved is low, then it suggests opportunities for further enhancement.

The SIMD efficiency is measured on two platforms, a Pentium III machine and a 2-way SimpleScalar simulator, for each of the six benchmarks. Table 5.3 shows the execution statistics and SIMD efficiency for each of the benchmarks. The ideal number of execution cycles is computed by identifying the

number of required useful computation operations and the available SIMD execution units (2 ALUs and 1 multiplier in both the processors).

Table. 5.3 Execution statistics and efficiency of media programs

Benchmark	Pentium III – MMX & SSE		
	Inst. Count	Actual Cycle count	Efficiency
<i>cfa</i>	404,290,544	231,616,932	5.16 %
<i>dct</i>	188,798,806	123,944,326	6.2 %
<i>scale</i>	2,170,274	20,756,929	2.31 %
<i>motest</i>	156,734,613	113,623,185	3.38 %
<i>aud</i>	220,320,505	150,386,375	11.97 %
<i>g711</i>	59,066,806	64,006,729	1.12 %
Benchmark	SimpleScalar - SIMD		
	Inst. Count	Actual Cycle count	Efficiency
<i>cfa</i>	349,447,420	338,685,938	3.53 %
<i>dct</i>	160,050,834	131,587,103	5.84 %
<i>scale</i>	3,129,815	4,626,696	10.36 %
<i>motest</i>	136,801,609	129,364,679	5.94 %
<i>aud</i>	283,199,976	191,516,819	9.40 %
<i>g711</i>	63,360,233	49,302,976	1.45 %

SIMD efficiency ranges from 1% to 12% and 1.5% to 10.5% for the Pentium III and SimpleScalar based SIMD processor, respectively. The SIMD efficiency is alarmingly low because the supporting instructions dominate the dynamic instruction stream. The execution time is also increased because of conventional architectural limitations such as cache misses, misalignment issues,

resource stalls, BTB misses, TLB misses, and branch mis-speculations. The efficiency of the Pentium III processor is slightly higher than the Simplescalar based processor in four of the six benchmarks because it is able to issue three micro-ops (equivalent to 2.7 x86 CISC instructions for the benchmarks above) while the Simplescalar processor issues two instructions per cycle. Two benchmarks (*scale* and *g711*) achieve a better efficiency for the Simplescalar configuration because they are more memory intensive than the other benchmarks (the L1 cache latency of the Pentium III processor is 3 cycles, while that of the Simplescalar configuration is 1 cycle).

I also measured similar statistics for the Pentium III and the Simplescalar based processor without SIMD extensions. It is found that the execution time is slower than SIMD enhanced processors, but the efficiency is higher for non-SIMD processors. This is because a 64-bit SIMD execution unit counts towards a peak rate of either 4 or 8 computations per cycle (16-bit or 8-bit data), whereas the scalar execution unit counts toward a single computation per cycle. While it is true that SIMD enhancements were not added to improve efficiency of processing but to speedup multimedia programs, this characterization highlights the gap between peak computation rate and achieved computation rate for SIMD programs and points to ample opportunities for performance improvement.

5.2.4 Memory access and branch bottlenecks

Supporting wide issue processors requires the ability to fetch across multiple branches. Also, memory latency prevents processors from fetching data in a timely fashion to achieve peak throughput. In this section, I investigate how memory latency and branch prediction impact the performance of these media kernels and applications. Table 5.4 shows the IPC with unit cycle memory access (i.e. a perfect L1 cache) and perfect branch prediction for the 2-, 4-, 8-, and 16-way processors with SIMD extensions.

It is seen that different programs vary in their sensitivity to memory latency and branch prediction. *Scale* and *g711* benchmarks are memory bound programs and improve significantly due to a unit cycle memory access but show negligible increase in IPC due to perfect branch prediction. *Cfa*, *dct*, and *mot* are benchmarks that operate on sub-blocks in a 2-D structure requiring five levels of loop nesting and benefit the most from perfect branch prediction and the ability to fetch across multiple branches in a single cycle. A unit cycle memory access has negligible performance impact on these three benchmarks. The remaining four benchmarks (*aud*, *jpeg*, *jpeg*, and *decrypt*) benefit equally from both perfect branch prediction and unit cycle memory access.

It is evident from this experiment that it is extremely important to provide low latency memory access and excellent branch prediction extending over multiple branches in order to achieve good performance.

Table. 5.4 Performance (IPC) with unit cycle memory accesses and perfect branch prediction

	<i>cfa</i>	<i>dct</i>	<i>mot</i>	<i>scale</i>	<i>aud</i>	<i>g711</i>	<i>jpeg</i>	<i>jpeg</i>	<i>decrypt</i>
Realistic IPC									
2-way	1.03	1.22	1.06	0.68	1.48	1.29	1.18	1.27	1.66
4-way	2.19	1.71	2.14	1.05	2.26	2.29	1.85	2.12	2.38
8-way	2.66	2.26	2.85	2.34	3.48	3.83	2.23	2.81	2.80
16-way	2.68	2.92	2.87	3.90	4.89	5.65	3.80	3.58	3.67
IPC with Unit cycle memory access									
2-way	1.04	1.26	1.06	1.43	1.57	1.59	1.33	1.34	1.75
4-way	2.19	1.78	2.14	2.84	2.50	3.10	2.00	2.30	2.52
8-way	2.71	2.30	2.85	5.56	3.66	5.22	2.37	3.21	2.95
16-way	2.71	2.95	2.86	9.54	5.27	7.76	5.10	4.07	3.89
IPC with perfect branch prediction									
2-way	1.75	1.60	1.79	0.68	1.62	1.29	1.24	1.42	1.70
4-way	3.44	3.09	3.59	1.05	2.69	2.29	1.92	2.60	2.40
8-way	6.47	5.91	7.03	2.35	4.35	3.79	2.46	3.99	2.86
16-way	10.49	11.19	11.61	3.91	6.37	5.55	5.45	6.66	3.79

5.3 Summary

It is often perceived that the characteristics of media applications are well understood. However, detailed analysis shows that there are several features in media workloads beyond the well-touted characteristics such as DLP and structured

computations. This chapter analyzes multimedia workloads focusing on instructions that support core computations rather than the computations themselves. Based on an investigation of loop structures and access patterns in multimedia algorithms, it is found that significant amount of parallelism lies outside the innermost loops, and it is difficult for SIMD units to exploit the parallelism. The characteristics preventing SIMD computation units from computing at their peak rate are analyzed. The major findings of the bottleneck analysis are:

- Approximately 75-85% percent of instructions in the dynamic instruction stream of media workloads are not performing useful computations. They are performing address generation, data rearrangement, packing/unpacking, loop branches, and loads/stores.
- The efficiency of the SIMD computation units is very low because of the overhead/supporting instructions. Measurements on a Pentium III processor with a variety of media kernels and applications illustrate SIMD efficiency ranging only from 1% to 12%.
- Increasing the number of SIMD execution units does not impact performance positively leading me to conclude that resources for overhead/supporting instructions need to be scaled. I observe that a significant increase in scalar resources is required to increase the SIMD efficiency using conventional ILP techniques. An 8-way or 16-way integer processor is nec-

essary to process the overhead instructions for the SIMD width in current processors.

Chapter 6

Hardware Support for Efficient SIMD Processing

Overhead or supporting instructions dominate the instruction stream of multimedia applications due to the programming conventions of GPPs. Overhead related instructions need to be either eliminated, alleviated, or overlapped with the useful computations for better performance, i.e. the higher the overlap of overhead related instructions, the higher the SIMD efficiency. In this chapter, I exploit the observed characteristics of the media programs and propose to augment SIMD GPPs with specialized hardware to efficiently overlap the overhead/supporting instructions.

SIMD instructions reduce the dynamic count of instructions because they operate on multiple data in a single instruction. Due to the repetitive operations required by media applications such a technique reduces the number of instruction fetches and decodes. However, SIMD instructions capture only the useful computation operation. In Chapter 5, I showed that overhead/supporting instructions that are necessary for feeding the SIMD execution units constitute 75% to 85% of the dynamic instructions.

Similar to the computation operations, parallelism exists in the overhead/supporting instructions. However, GPPs have limited support for generating addresses and keeping track of multiple loop nests/bounds. In this chapter, I present an architecture that incorporates explicit hardware support for efficiently executing the overhead/supporting instructions in a SIMD GPP. In addition to capturing the useful computation operations, all the associated overhead operations are captured in a single multidimensional vector instruction. This leads to a drastic reduction in dynamic instructions and reduces repeated (and unnecessary) fetch and decode of the same instructions.

The rest of the chapter is organized as follows. Section 6.1 describes the proposed architecture. Section 6.2 presents the performance evaluation. The chapter is summarized in Section 6.3.

6.1 The MediaBreeze Architecture

6.1.1 Decoupling computation and overhead

I exploit the characteristics of media applications that were observed in Chapter 5 by proposing the MediaBreeze architecture. Specialized hardware is incorporated in a SIMD GPP to efficiently overlap the overhead/supporting instructions. Figure 6.1 shows the block diagram of the MediaBreeze architecture.

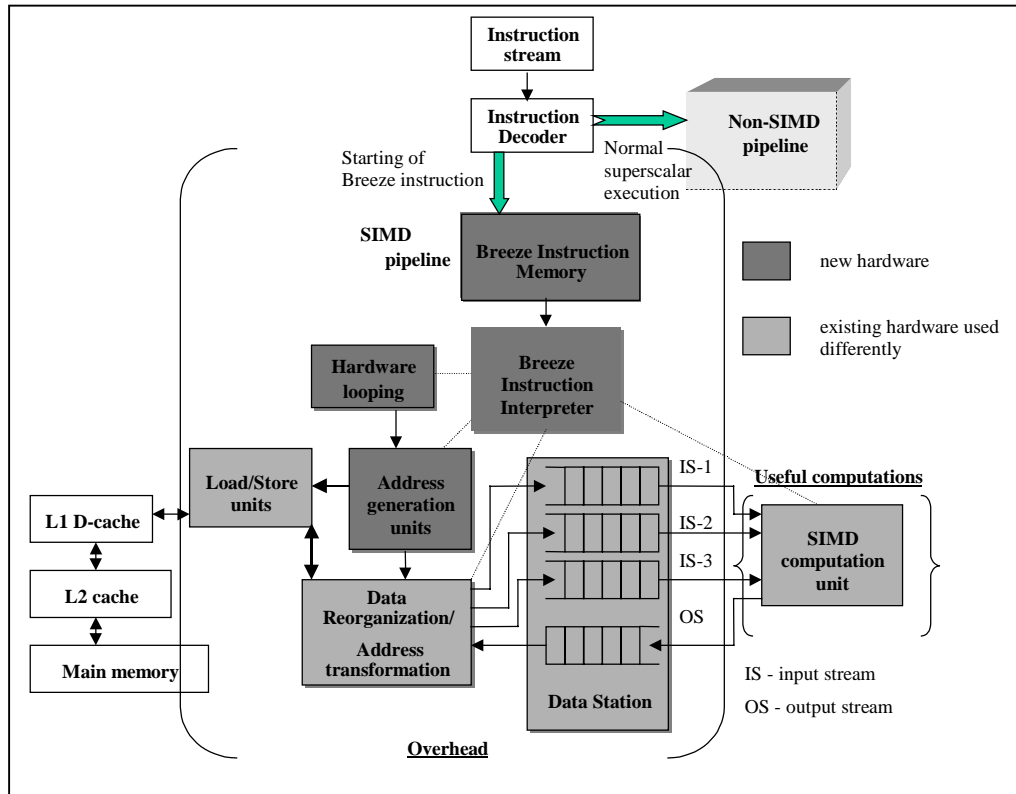


Figure. 6.1. The MediaBreeze Architecture

In order to perform the SIMD operations, the MediaBreeze architecture introduces new hardware units as well as reuses existing hardware units. The new hardware units (darkly shaded blocks in Figure 6.1) are the address generation units, hardware looping, and Breeze instruction memory & interpreter. The hardware units reused (lightly shaded blocks in Figure 6.1) are load/store units,

SIMD computation unit, data reorganization/address transformation, and the data station. The SIMD computation unit handles the useful computation part while the remaining units handle the overhead/supporting instructions. The hardware units that process the overhead related instructions are:

- Address calculation: address arithmetic functions are moved from the execution unit subsystem in current processors to a dedicated hardware unit where address arithmetic hardware generates all input and output address streams concurrently with the SIMD computations. Address calculations are performed explicitly by the CPU in current ILP processors. This involves some combination of extra instructions, parts of instructions, registers, memory accesses, and computation time. Dedicated address arithmetic hardware would allow for the SIMD computation unit to stream at the peak rate.
- Address transformation: In many algorithms, the logical access sequence of data is vastly different from the physical storage pattern. Various permute operations including pack, unpack instructions are used. For example, the first element in eight columns of a matrix must be packed into a single row (or SIMD register). Similarly a single element (16-bits wide) must be unpacked into all the four sub-words of a SIMD register (64-bits wide). MediaBreeze efficiently handles the task of reordering data with explicit hardware support.

- Loads and stores: The same load/store units present in conventional ILP processors are used for this purpose.
- Branch processing: To eliminate branch instruction overhead, MediaBreeze employs zero-overhead branch processing using dedicated hardware loop control and supports up to five levels of loop nesting. All branches related to loop increments (based on indices used for referencing data) are handled by this technique. This is done in many conventional DSP processors such as the Motorola 56000 and TMS320C5x from Texas Instruments [57]¹.
- Data Station: This is the register-file for the SIMD computation and is implemented as a queue. Dedicated register-files are present in conventional machines for SIMD either as a separate register file (as in AltiVec) or aliased to the floating-point register file (as in MMX).
- Breeze instruction memory and interpreter: In order to program/control the hardware units in the MediaBreeze architecture, a special instruction called the Breeze instruction is formulated. The Breeze instruction memory stores these instructions once they enter the processor. Figure 6.2 illustrates the structure of the Breeze instruction.

¹ Recent DSP processors such as the TMS320C62xx have eliminated hardware looping because this allows for smaller and simpler instruction sets. Furthermore, multiple loops cannot be encoded in a single 32-bit instruction.


Loop1-count	Loop2-count	Loop3-count	Loop4-count	Loop5-count	<p style="text-align: center;">Legend</p> <p>IS - input stream</p> <p>OS - output stream</p> <p>OPR - operation code</p> <p>RedOp - reduction operation</p> <p>LL - loop level to write results</p>
Starting Address of IS-1	Starting Address of IS-2	Starting Address of IS-3	Starting Address of OS	OPR / RedOp / Shift / LL	
Stride-1 IS-1	Stride-2 IS-1	Stride-3 IS-1	Stride-4 IS-1	Stride-5 IS-1	
Stride-1 IS-2	Stride-2 IS-2	Stride-3 IS-2	Stride-4 IS-2	Stride-5 IS-2	
Stride-1 IS-3	Stride-2 IS-3	Stride-3 IS-3	Stride-4 IS-3	Stride-5 IS-3	
Stride-1 OS	Stride-2 OS	Stride-3 OS	Stride-4 OS	Stride-5 OS	
Masks - IS-1 and IS-2	Masks - IS-3 and OS	Multicast and data types of each stream with remaining bits unused			
 32-bits					

Figure. 6.2. Structure of the Breeze Instruction

Five loop index counts (bounds) are indicated in the Breeze instruction to support five level nested loops (in hardware). None of the nine benchmarks required more than five nested loops. The MediaBreeze architecture allows for three input data structures/streams and produces one output structure. This was chosen because some media algorithms can benefit from this capability (current SIMD execution units sometimes operate on three input registers to produce one output value). For example, adaptive filtering algorithms use three input streams to generate an output stream. Similarly, certain signals are complex-valued requiring the third input stream. If an application does not use the third input stream, the third input can be used for prefetching either the first or the second input stream.

Each data structure/stream has its own dedicated address generation unit to compute the address every clock cycle with the base address specified in the Breeze instruction. Due to the sub-block access pattern in media programs, data is accessed with different strides at various points in the algorithm. The Breeze instruction facilitates multiple strides (one at each level of loop nesting, for a total of five strides) for each of the three input streams and one output stream. The strides indicate address increment/decrement values based on the loop-nest level. Depending on the mask values for each stream (indicated in the Breeze instruction) and the loop-nest level, one of the five possible strides is used to update the address pointer. If an application does not need five levels of nesting, non-constant strides can be generated with the extra levels of looping [75].

Data types of each stream/structure are also indicated in the Breeze Instruction. Depending on the size of each element in the data structures, a different amount of SIMD parallelism is achieved. If one data stream is 8-bit data (16-way parallelism for a 128-bit wide execution unit) and the other is 16-bit data (8-way parallelism), the SIMD processing achieves only 8-way parallelism. The maximum achievable SIMD parallelism is the minimum of all the data structures (all commercial SIMD extensions have this limitation). Current SIMD extensions provide data reorganization instructions such as packing, unpacking, and permute for solving the problem of having different element sizes across the data structures and introduce additional instruction overhead. By providing this

information in the Breeze Instruction, special hardware in the MediaBreeze will perform this function. Reduction operations to be performed by the MediaBreeze are also indicated in the Breeze Instruction. For example, multiple independent results in a single SIMD register are combined together in dot product, which require additional instructions in current DLP techniques. Support for signed/unsigned arithmetic, saturation, shifting/scaling of final results is also indicated in the Breeze Instruction. This eliminates additional instructions that are otherwise needed for conventional RISC processors.

With the support for multiple levels of looping and multiple strides, the Breeze Instruction is a complex instruction and decoding such an instruction is a complex process in current RISC processors. MediaBreeze instead handles the task of interpreting/decoding of the Breeze Instruction. MediaBreeze has its own instruction memory to hold a Breeze instruction. Two additional 32-bit instructions are also added to the ISA of the general-purpose processor for starting and interrupting the MediaBreeze. These 32-bit instructions are fetched and decoded by the traditional instruction issue logic and contain the length of the Breeze Instruction. Whenever a Breeze instruction is encountered in the dynamic instruction stream, the dynamic instructions prior to the Breeze instruction are allowed to finish after which the MediaBreeze instruction interpreter decodes the Breeze instruction. In the current implementation, the superscalar pipeline is halted until the execution of the Breeze instruction is completed because MediaBreeze re-

uses existing hardware units. Otherwise, arbitration of resources is necessary to allow for overlap of the Breeze instruction and other superscalar instructions. A 100-cycle penalty (a conservative estimate based on simulation) is associated between the detection and start of a Breeze instruction in the simulations.

Encoding all the overhead/supporting operations along with the SIMD useful computation instructions has the advantage that the Breeze instruction can potentially replace millions of dynamic RISC instructions that have to be fetched, decoded, and issued every cycle in a normal superscalar processor. This results in giving the MediaBreeze architecture advantages similar to ASIC-based acceleration in [107].

It is possible that an exception or interrupt occurs while a Breeze instruction is in progress. The state of all five loops, their current counts, and loop bounds are saved and restored when the instruction returns. This is similar to the handling of exceptions during move instructions with REP (Repeat Prefix) in x86. MediaBreeze has registers to hold the loop parameters for all the loops. Code development for the MediaBreeze architecture is currently done by hand. Similar to developing code for SIMD extensions, compiler intrinsics may be employed to utilize the MediaBreeze architecture. I do underestimate the challenge of compiling for the MediaBreeze architecture; however, the effort is comparable to that of compiling for current SIMD extensions.

6.1.2 Multicast: A technique to aid in data transformation

The MediaBreeze uses a technique called *Multicast* to eliminate the need for transposing data structures, to allow reordering of the computations, and to increase reuse of data items soon after fetch. Multicasting means copying one/many data items into several registers or buffers at the same item. For example, a data value A may be copied into 8 registers (or 8 sections of a big SIMD register) resulting in a pattern A,A,A,A,A,A,A,A or two items A and B may be copied to 8 registers in the pattern A,A,B,B,A,A,B,B or A,B,A,B,A,B,A,B or another such pattern. The usefulness of multicasting can be illustrated by the well-understood matrix-multiply routine. In a matrix-multiply routine, usually the first matrix is traversed in row-order and the second matrix in column-order. Spatial locality can be exploited in the first matrix due to multiple data elements in each cache block, while the second matrix incurs a compulsory miss on each column the first time; assuming that two consecutive rows do not fit in a cache-block. In a machine with no SIMD execution units, during each iteration for the second matrix, a new cache-line has to be loaded as data belongs to the same column but different cache-line. However, for the case of SIMD processing, multiple cache-lines need to be loaded and data belonging to the required column needs to be reorganized from a vertical to a horizontal direction (packing). This involves substantial overhead and usually, the second

matrix is transposed prior to the computation to eliminate the column-access pattern.

The transposing overhead can be eliminated using the Multicast technique. Instead of using column-access pattern, row-order access pattern is used for matrix B, while for matrix A, a single element is multicast to all eight sub-element locations in the SIMD register. Then instead of doing the eight multiplications to generate the first element $C_{1,1}$ of the result matrix, all eight multiplications using $A_{1,1}$ (i.e. the first partial product of each of the result terms in the first row) are performed. The sequence of multiplications in a normal SIMD matrix multiply and a Multicast matrix multiply are illustrated in Figure 6.3.

After 64 multiplications, all eight result terms of the first row of the result matrix will be simultaneously generated. The algorithm using the multicast technique is always operating on multiple independent output values, while traditional techniques compute one result term at a time. This eliminates the need for transposing the second matrix. It also increases the reuse of items that were loaded, thus improving the cache behavior of the code. The MediaBreeze architecture provides hardware support for multicasting. This allows the use of cache-friendly algorithms to perform many media algorithms. In this example, broadcast rather than multicast was employed, because one element is transmitted to all eight registers. However, in several applications such as horizontal/vertical downsampling/upsampling, and filtering, several elements are multicast into the

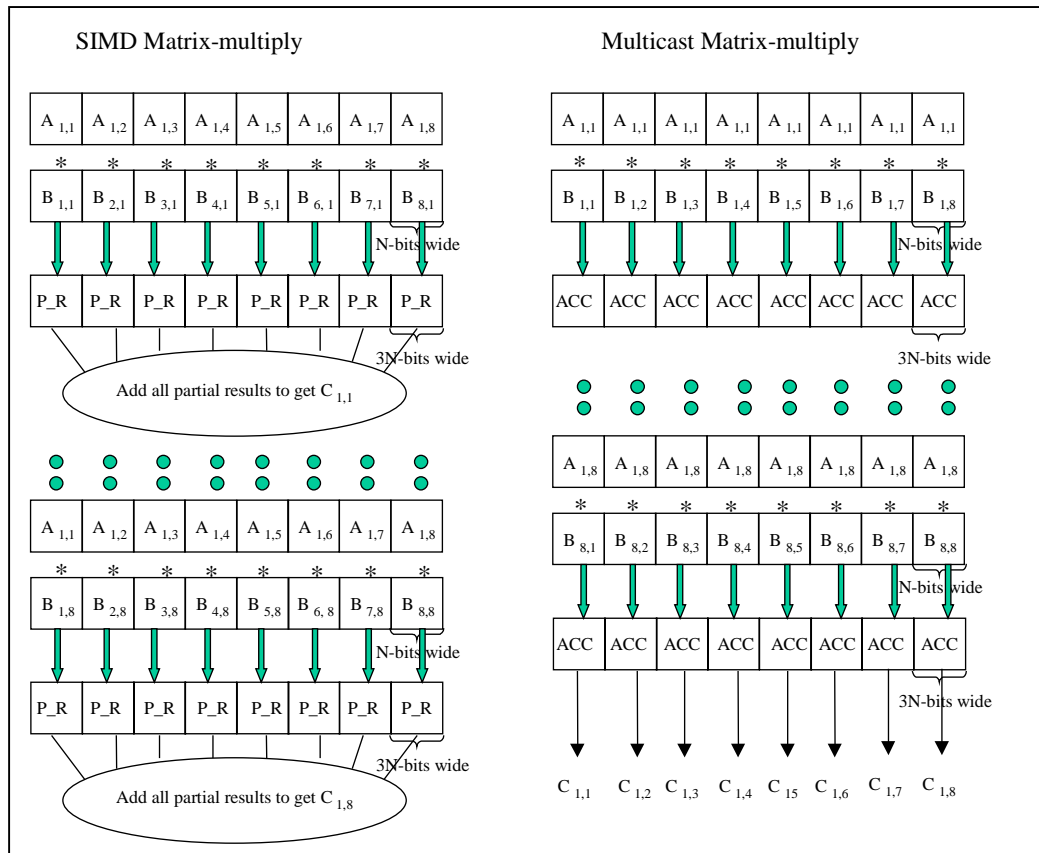


Figure. 6.3. Multicast technique versus traditional SIMD matrix multiply

sub-element locations, many-to-many mapping as opposed to one-to-many mapping and hence the name multicast.

If the dimension of the matrices to be multiplied is large, then the multicast method needs temporary registers or an accumulator to store the accumulated results. However, multimedia applications operate on sub-blocks in huge matrices as opposed to processing the entire matrix as a whole. A SIMD parallelism of 8 or 16 is quite adequate to capture most media sub-block matrices.

Another common operation where multicast is extremely useful is in 1-D and 2-D filtering, and convolution. For example, when using MMX for implementing a finite impulse response (FIR) filter, multiple copies of the filter coefficients are needed (equal to the SIMD parallelism) to reduce considerable overhead due to misalignment of coefficient data. Multicasting eliminates need for data reorganization due to misalignment issues. In fact, the MediaBreeze architecture has hardware for handling misalignment issues even if multicasting is not used.

6.1.3 Example encoding using the Breeze instruction

The Breeze instruction is a densely encoded instruction and hence most media algorithms can be processed in just a few Breeze instructions. Common kernels such as the DCT, color space conversion, motion estimation, and filtering can be mapped to either one or two Breeze instructions. Figures 6.4-6.7 show the pseudo-code for the implementation of the Breeze instruction. Figure 6.8 illustrates the Breeze instruction mapping of the 1-D DCT routine assuming an 8-way SIMD for 16-bit data. For the 1-D DCT routine, only four of the five possible loop nests are needed with the loop boundaries indicated in the Breeze instruction. The starting address of each stream is represented by the starting address of each of the arrays. The third input stream is not used for this algorithm. The value of the strides is computed based on the loop indices and the value of

the address pointer in the previous cycle. The address pointer is updated each clock cycle choosing one stride depending on the nesting level of the loops.

In a scenario in which all the loop nests and data streams are processed, MediaBreeze executes (in hardware) the following equivalent number of dynamic software instructions (in conventional ILP processors) during each cycle -

- five branches
- three loads and one store
- four address value generation (one on each stream with each address generation representing multiple RISC instructions)
- one SIMD operation (2-way to 16-way parallelism depending on each data element size)
- one accumulation of SIMD result and one SIMD reduction operation
- four SIMD data reorganization (pack/unpack, permute, etc) operations
- shifting & saturation of SIMD results

```
looping {  
  for (i_1 = 0; i_1 < loop1_count; i_1++) {  
    for (i_2 = 0; i_2 < loop2_count; i_2++) {  
      for (i_3 = 0; i_3 < loop3_count; i_3++) {  
        for (i_4 = 0; i_4 < loop4_count; i_4++) {  
          for (i_5 = 0; i_5 < loop5_count; i_5++) {  
            }  
          }  
        }  
      }  
    }  
  }  
}
```

Figure. 6.4. Pseudo-code implementation of the MediaBreeze unit for hardware looping

```

IS1 = start_address_IS1; IS2 = start_address_IS2;
IS3 = start_address_IS3; OS1 = start_address_OS;

increment_address (level) {
  if (mask_IS1 [level] )   IS1 += stride_IS1[level];
  if (mask_IS2 [level] )   IS2 += stride_IS2[level];
  if (mask_IS3 [level] )   IS3 += stride_IS3[level];
  if (mask_OS [level] )    OS += stride_OS[level];
}

      if      (( i_5 + 1) = loop1_count) increment_address(4);
      elseif  (( i_4 + 1) = loop2_count) increment_address(3);
      elseif  (( i_3 + 1) = loop3_count) increment_address(2);
      elseif  (( i_2 + 1) = loop4_count) increment_address(1);
      else    increment_address(5);

```

Figure. 6.5. Pseudo-code implementation of the MediaBreeze unit for address generation

```

Load (IS1, R1);
Load (IS2, R2);
Load (IS3, R3);
Store (R4, OS);

```

Figure. 6.6. Pseudo-code implementation of the MediaBreeze unit for loads/stores

```

SIMD_data_reorganization (R1, R2, R3)
SIMD_compute (operation, R1, R2, R3, R4);
SIMD_data_reorganization (R4)

```

Figure. 6.7. Pseudo-code implementation of the MediaBreeze unit for SIMD computation and data reorganization

```

1D_DCT( image[1200][1600], dct_coef[8][8], output[8][8] )
{
for ( i = 0; i < 1200/8; i++)
  for ( j = 0; j < 1600/8; j++)
    for ( k = 0; k < 8; k++) {
      temp_simd_vector = 0;
      for ( l = 0; l < 8; l++)

/* Since there is 8-way SIMD parallelism, the innermost loop folds into one iteration
and is not required */

      temp_simd_vector += multicast(dct_coef[ k ][ l ] * image[ i*8+k ][ j*8+l ]);
      output[ i*8 ][ k*8 ] = temp_simd_vector >> s_bits;
    } }

```

0	1200/8	1600/8	8	8
Starting Address of image	Starting Address of dct coeff	----- NONE -----	Starting Address of output	OPR = MAC Shift = s_bits LL = 4
NONE	16 bytes	-22384 bytes	-22400 bytes	3200 bytes
NONE	-126 bytes	-126 bytes	2 bytes	2 bytes
NONE	NONE	NONE	NONE	NONE
NONE	-22384 bytes	3200 bytes	NONE	NONE
IS-1 = 01111 IS-2 = 01111	IS-3 = 00000 OS = 01100	Multicast is used for dct coefficients data types of each stream is set to 16-bit data		

Figure. 6.8. Breeze instruction mapping of 1D-DCT

6.2 Performance Evaluation

To measure the impact of the MediaBreeze architecture, the PISA version of SimpleScalar-3.0 (sim-outorder) was modified to simulate Breeze instructions using instruction annotations. The same SIMD execution unit configurations as in a Pentium III processor (two 64-bit SIMD ALUs and one 64-bit SIMD multiplier) are used. The memory system for the MediaBreeze architecture is modified to allow for cache miss stalls and memory conflicts (i.e., the SIMD pipeline stalls in the event of a cache miss) since the MediaBreeze operates in an in-order fashion. The MediaBreeze hardware is incorporated into a 2-way and 4-way SIMD GPP. Figure 6.9 shows the speedup obtained for each of the benchmarks using the MediaBreeze architecture with a 2-way processor as the baseline.

The speedup of the 2-way MediaBreeze architecture over a 2-way SIMD enhanced processor ranges from 1.0x to over 16x. In four of the nine benchmarks (*cfa*, *dct*, *mot*, *scale* – which are kernels) all of the benchmark code translates into one or two Breeze instructions with no other superscalar instructions necessary. The remaining five benchmarks (*aud*, *g711*, *jpeg*, *jpeg*, and *decrypt* – which are applications) require scalar superscalar instructions along with Breeze instructions. G711 and decrypt are applications that have the least amount of SIMD instructions as the superscalar pipeline accounts for a bulk of the execution time rather than the MediaBreeze pipeline. On these applications, a 2-way

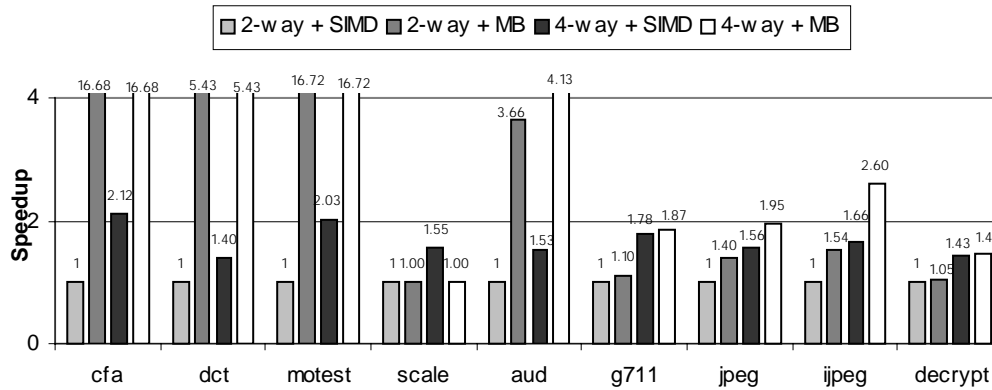


Figure. 6.9. Performance of MediaBreeze (MB) versus SIMD

MediaBreeze architecture is only slightly faster than a 2-way SIMD processor. On the other hand for the remaining three benchmarks (*aud*, *jpeg*, and *jpeg*), a 2-way MediaBreeze architecture is significantly faster than a 2-way SIMD processor. Table 6.1 shows the speedup and SIMD efficiency achieved by the 2-way and 4-way MediaBreeze enhanced processors along with 2-, 4-, 8-, and 16-way superscalar out-of-order SIMD processors. The SIMD efficiency for *jpeg*, *jpeg*, and *decrypt* could not be computed because of several initialization and file I/O routine in the source code of the benchmarks.

Table. 6.1 Speedup of the MediaBreeze architecture along with SIMD efficiency (as a %). The 2-way SIMD GPP is used as the baseline.

	<i>cfa</i>	<i>dct</i>	<i>mot</i>	<i>scale</i>	<i>aud</i>	<i>g711</i>	<i>jpeg</i>	<i>jpeg</i>	<i>decrypt</i>
2-way + SIMD	1 3.53 %	1 5.84 %	1 2.97 %	1 10.4 %	1 9.4 %	1 1.45 %	1	1	1
2-way + MB	16.7 58.9 %	5.43 31.7 %	16.7 99 %	1 10.4 %	3.66 34.4 %	1.1 1.6 %	1.40	1.50	1.05
4-way + SIMD	2.12 7.48 %	1.40 8.17 %	2.03 6.03 %	1.55 16.1 %	1.53 14.4 %	1.78 2.58 %	1.56	1.66	1.43
4-way + MB	16.7 58.9 %	5.43 31.7 %	16.7 99 %	1 10.4 %	4.13 38.8%	1.87 2.7 %	1.95	2.60	1.46
8-way + SIMD	2.58 9.11 %	1.85 10.8 %	2.69 8.0 %	3.45 35.7 %	2.36 22.2 %	2.98 4.32 %	1.88	2.20	1.69
16-way + SIMD	2.59 9.14 %	2.40 14.0 %	2.69 8.0 %	5.76 59.7 %	3.31 31.1 %	4.40 6.38 %	3.22	2.81	2.21

The MediaBreeze pipeline is susceptible to memory latencies because it operates in-order. Thus MediaBreeze is unable to achieve maximum SIMD efficiency on three of the four kernels (*cfa*, *dct*, and *scale*) in spite of them being mapped completely to one or two Breeze instructions. To reduce the impact of memory latencies on the MediaBreeze architecture, a prefetch engine was introduced to load future data into the L1 cache. Since the access pattern of each data stream is known in advance based on the strides, the prefetch engine does not load any data that is not going to be used. The regularity of the media access patterns prevents the risk of superfluous fetch very commonly encountered in many prefetching environments. The prefetch engine ‘slips’ ahead of the loads

for computation and the computation itself to gather data into the L1 cache. Table 6.2 shows the speedup of the MediaBreeze architecture with prefetching for the 2-way and 4-way configurations. It is observed that prefetching in the MediaBreeze architecture achieves unit cycle memory access performance in the Breeze instruction portion of the program. The speedup is most noticeable in *cfa*, *dct*, *scale*, and *aud*.

Table 6.3 shows the percentage reduction in dynamic instructions by the MediaBreeze architecture. This leads to a significant reduction in fetch, decode, and issue logic power consumption in a GPP. The instruction fetch and issue logic are expected to consume greater than 50% of the total execution power (not including clock power) in future speculative processors [108]. Implementation cost of adding the MediaBreeze hardware to a SIMD GPP is further evaluated in Chapter 7.

The geometric mean of the speedup of the 2-way MediaBreeze processor over a 2-way SIMD processor for the five applications (not including the kernels - *cfa*, *dct*, *mot*, and *scale*) is 1.73 while that of a 4-way SIMD processor over a 2-way SIMD processor is 1.59. Therefore, on applications, a 2-way MediaBreeze architecture achieves a performance slightly better than a 4-way superscalar SIMD processor. A similar trend is observed for the case of a 4-way MediaBreeze processor being slightly superior to an 8-way superscalar SIMD processor.

Table. 6.2 Speedup of the MediaBreeze architecture with prefetching

	<i>cfa</i>	<i>dct</i>	<i>mot</i>	<i>scale</i>	<i>aud</i>	<i>g711</i>	<i>jpeg</i>	<i>jpeg</i>	<i>decrypt</i>
2-way	1 3.53 %	1 5.84 %	1 2.97 %	1 10.4 %	1 9.4 %	1 1.45 %	1	1	1
2-way + MB prefetching	27.92 98.3 %	16.52 96.5 %	16.84 99.5 %	4.54 47.2 %	6.87 64.6 %	1.21 1.76 %	1.44	1.61	1.05
4-way	2.12 7.48 %	1.4 8.17 %	2.03 6.03 %	1.55 16.1 %	1.53 14.4 %	1.78 2.58 %	1.56	1.66	1.43
4-way + MB prefetching	27.92 98.3 %	16.52 96.5 %	16.84 99.5 %	4.54 47.2 %	8.74 82.2 %	2.22 3.22 %	2.02	2.82	1.46

Table. 6.3 Percentage reduction in dynamic instruction count of the MediaBreeze architecture in comparison to a conventional RISC ISA with SIMD extensions

	<i>cfa</i>	<i>dct</i>	<i>mot</i>	<i>scale</i>	<i>aud</i>	<i>g711</i>	<i>jpeg</i>	<i>jpeg</i>	<i>decrypt</i>
MB	99 %	99 %	99 %	99 %	91 %	11 %	43 %	42 %	0.2 %

6.3 Summary

Many enhancements such as increasing the number of SIMD execution units target exploiting additional parallelism in the useful computation while the MediaBreeze architecture proposed in this chapter focuses on the overhead instructions and the ability of the hardware to eliminate, alleviate, and overlap the overhead. MediaBreeze exploits the regularity and predictability of the overhead

instructions to devise simple hardware by combining the advantages of SIMD, vector, DAE, and DSP processors. The major findings are:

- Eliminating and reducing the overhead using specialized hardware that works in conjunction with state-of-the-art superscalar processor and SIMD extensions can dramatically improve the performance of media workloads without deteriorating the performance of general-purpose workloads. On kernels, a 2-way processor with SIMD extensions augmented with MediaBreeze hardware significantly outperforms a 16-way processor with SIMD extensions.
- On applications, a 2-way processor with SIMD extensions with the supporting MediaBreeze hardware outperforms a 4-way superscalar processor with SIMD extensions. Similarly, a 4-way processor with SIMD extensions added with MediaBreeze hardware is superior to an 8-way superscalar with SIMD extensions.

Chapter 7

Hardware Cost of the MediaBreeze Architecture

Adding hardware support to execute the overhead/supporting instructions increases the computation throughput of the SIMD execution units. In Chapter 5, I found that the SIMD execution units in a GPP are under-utilized and bottlenecks are concealed elsewhere in the overhead/supporting instructions. Using specialized hardware to overlap the overhead related instructions with the useful computation instructions allows the SIMD execution units to achieve a higher throughput. The MediaBreeze architecture presented in Chapter 6 incorporates explicit hardware support for efficient looping and address generation to process the overhead/supporting instructions. A 2-way SIMD GPP enhanced with the MediaBreeze architecture outperforms a 4-way SIMD GPP on multimedia applications. Similarly, a 4-way SIMD GPP augmented with the MediaBreeze hardware was superior to an 8-way SIMD GPP. This chapter investigates the associated cost of adding the MediaBreeze units to a high-speed SIMD GPP. Using a cell-based methodology targeting a 0.18-micron ASIC library, I estimate area, power, and timing requirements of the added hardware.

Table 7.1 summarizes the hardware units (divided into two parts) necessary for implementing the MediaBreeze components in an out-of-order GPP. The first four entries (address generation, looping, Breeze instruction decoder, and Breeze instruction memory) relate to hardware units that are augmented to a SIMD GPP. The remaining four entries (SIMD computation unit, data reorganization unit, load/store units, and data station) relate to hardware units that are already existent in current commodity SIMD GPPs. In this chapter, I evaluate the cost of adding the hardware units that are not present in current SIMD GPPs.

Table. 7.1 Hardware functionality of various MediaBreeze hardware units

New Hardware Units	Functionality
Address generation	Address generation is moved from the execution unit subsystem to four address generation units that are added to the GPP core (one for each data stream).
Looping	Using five levels of nesting allows for capturing parallelism in the outer loops. Moreover, dedicated hardware looping allows for zero-overhead loop branch processing.
Breeze instruction decoder	The Breeze instruction is introduced to capture all the overhead/supporting instructions along with the core SIMD computation instructions. MediaBreeze handles the task of decoding the Breeze instruction and controlling the various hardware units.
Breeze instruction memory	The Breeze instruction memory stores a Breeze instruction once it enters the processor.

Table. 7.1 Hardware functionality of various MediaBreeze hardware units (continued)

Existing Hardware Units	Functionality
SIMD computation unit	All arithmetic and logical SIMD computations along with multiplication and special media operations such as sum-of-absolute-differences are executed in this unit. Current GPPs typically have two SIMD ALUs and one SIMD multiplier in their SIMD datapath.
Data reorganization	SIMD processing mandates several data reorganization mechanisms such as packing, unpacking, permute, etc. Reduction operations, scaling, and shifting of the results are also required for SIMD processing. Current commodity SIMD GPPs have data reorganization hardware in their SIMD datapath.
Load/store units	The same load/store units present in the GPP are used for the MediaBreeze architecture.
Data station	The data station acts as a register file for the SIMD computation. Current SIMD GPPs either have dedicated SIMD register files (AltiVec) or share the floating-point register file (MMX).

The rest of the chapter is organized as follows. Section 7.1 describes the implementation methodology and tools that are used to estimate the hardware cost. In section 7.2, a detailed implementation of the new MediaBreeze hardware units is presented. Section 7.3 evaluates the cost associated with implementing each mechanism in the MediaBreeze architecture and their applicability to mainstream commodity GPP pipelines. Finally, Section 7.4 summarizes the chapter.

7.1 Implementation Methodology

To estimate the area, power, and timing requirements of the MediaBreeze architecture, I developed VHDL models for the various components. Using Synopsys synthesis tools [94], I used a cell-based methodology to target the VHDL models to a 0.18-micron ASIC cell-library (G-12p) from LSI Logic [64][65]. The G-12p technology library operates at 1.8V and supports up to six layers of metal.

The Synopsys synthesis tools estimate area, power, and timing of circuits based on the ASIC technology library. The ASIC technology library provides four kinds of information.

- Structural information. This describes each cell's connectivity to the outside world, including cell, bus, and pin descriptions.
- Functional information. This describes the logical function of every output pin of every cell so that the synthesis tool can map the logic of a design to the actual ASIC technology.
- Timing information. This describes the parameters for pin-to-pin timing relationships and delay calculation for each cell in the library.
- Environmental information. This describes the manufacturing process, operating temperature, supply voltage variations, and design layout. The design layout includes wire load models that estimate the effect of wire length on

design performance. Wire load modeling estimates the effect of wire length and fanout on resistance, capacitance, and area of nets.

I use the default wire load models provided by LSI Logic's G12-p technology. The Synopsys synthesis tools compute timing information based on the cells in the design and their corresponding parameters defined in the ASIC technology library. The area information provided by the synthesis tools is prior to layout and is computed based on the wire load models of the associated cells in the design. Average power consumption is measured based on the switching activity of the nets in the design. In my experiments, the switching activity factor originates from the RTL models as the tool gathers this information from simulation. The area, power, and timing estimates are obtained after performing maximum optimizations for performance in the synthesis tools. The results obtained in this chapter reflect a first order approximation based on the accuracy of the synthesis tools and cell-based libraries. The interested reader is referred to [94] for further information regarding the capabilities and limitations of the synthesis tools.

7.2 Hardware Implementation of MediaBreeze Units

7.2.1 Address generation

The MediaBreeze architecture supports three input and one output data structures/streams. Each of the four data streams has a dedicated address generation hardware unit. Address arithmetic on each stream is performed based on the strides and mask values indicated in the Breeze instruction. For each clock cycle, depending on the mask bits and loop index counts, one of the five possible strides is selected. The new address value is then computed based on the selected stride and the previous address value. Figure 7.1 depicts the block diagram of the address generation circuitry for a single data stream/structure.

The *last_val comparators* determine which of the four inner level loop counters have reached their upper bound. The outermost loop comparison is not necessary because the Breeze instruction finishes execution at the instant when the outermost loop counter reaches its upper bound. The *inc-cond* and *inc-combine* blocks generate *flag* signals based on the output from the *last_val comparators* and mask values from the Breeze instruction. If none of the *flag* signals are true, then *stride-5* is used to update the *prev-address*; otherwise, the appropriate *stride- (1-4)* is selected depending on *flag- (1-4)*. The *address-generate* block uses a 32-bit adder to add the selected stride to the previous address. On either an exception or a stall, only the *prev-address* value needs to be stored as

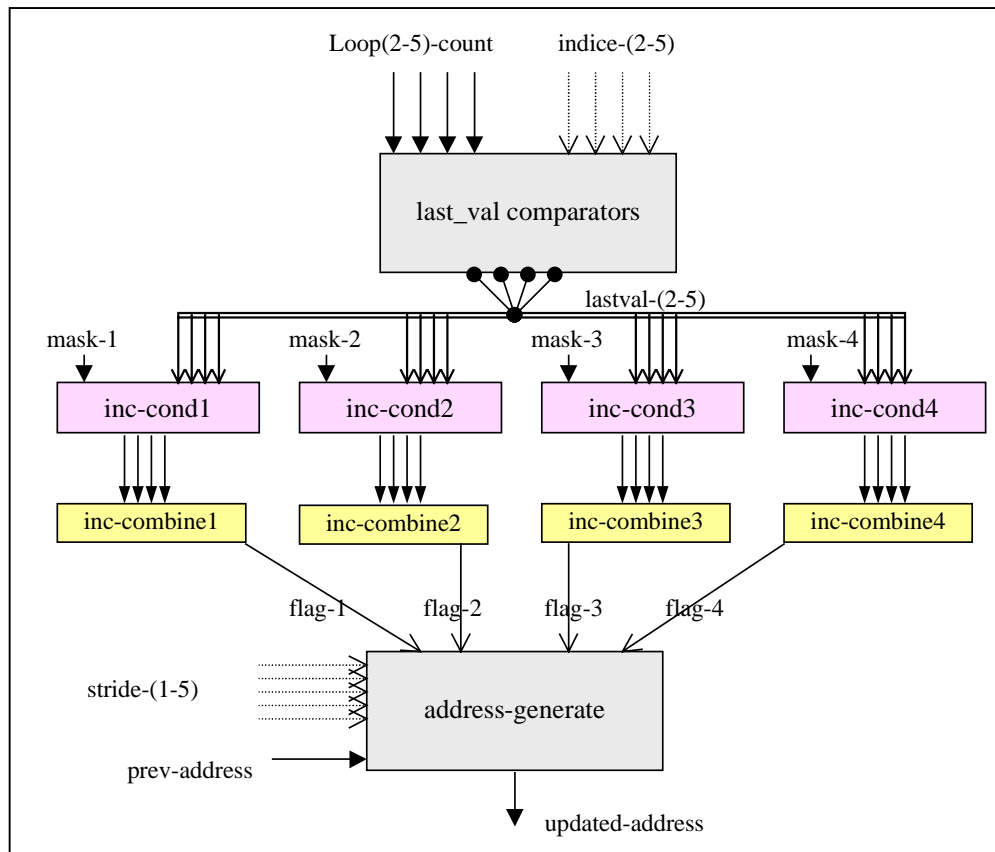


Figure. 7.1. Block diagram of address generation hardware (per data stream)

the loop counters are stored by the hardware looping circuitry. For each of the four data structures/streams, the *last_val comparators* portion of the logic is shared, but the remaining hardware needs to be replicated.

7.2.2 Looping

The MediaBreeze architecture incorporates five levels of loop nesting in hardware to eliminate branch instruction overhead for loop increments. A similar

mechanism was commercially implemented in the TI ASC [23] (two levels of do-loop nesting in addition to a self-increment loop). Conventional DSP processors such as the Motorola 56000 and the TMS320C5x from TI also use such a technique for one or more levels of loop nesting [57]. Figure 7.2 shows the block diagram of the looping hardware.

Loop index values are produced every clock cycle based on the loop bound for each level of nesting (bounds for each of the five loops are specified in the Breeze instruction). The value of a loop index varies from 1 (lower bound) to the corresponding loop bound (upper bound), and resets to its lower bound once the upper bound is reached in the previous cycle. The execution of the Breeze instruction ends when the outermost loop (loop1 in Figure. 7.2) reaches its upper bound. On encountering either an exception or a stall, the loop indices are stored and the increment logic is halted; the counting process is

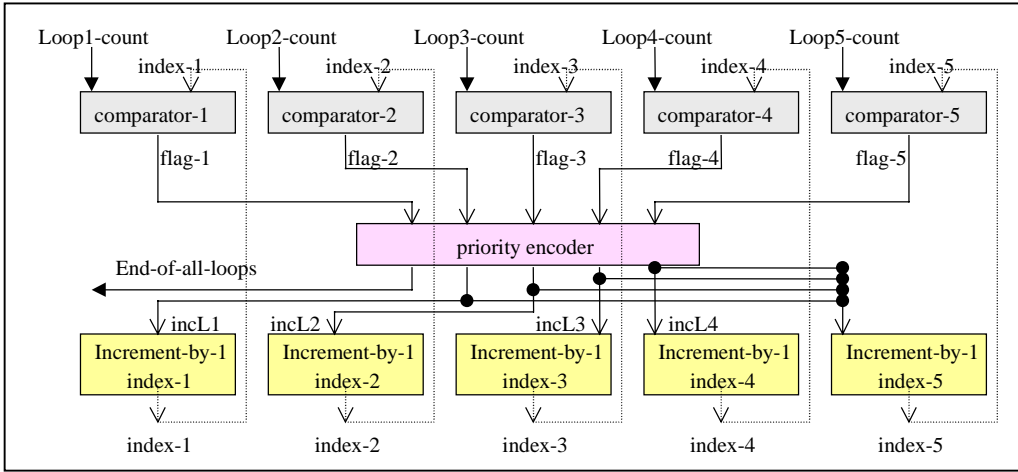


Figure. 7.2. Block diagram of the five hardware loops

started once the exception/stall is serviced. Each of the five *comparators* (32-bit wide) operates in parallel to generate *flag* (1-bit wide) signals that are *priority encoded* to determine which one of the five loop counters to increment. When a loop counter is *incremented-by-1* (circuit for incrementing a 32-bit value by 1), all the loop counters belonging to its inner level are reset (for example, if loop3 is *incremented-by-1*, then loop4 and loop5 are reset to their lower bound).

7.2.3 Breeze instruction decoder

A stand-alone instruction decoder for the Breeze instructions eliminates the need to modify the conventional instruction decoder of current GPPs. A Breeze instruction needs to be decoded only once since various control parameters are stored in hardware registers after the decoding process. The implementation of the Breeze instruction decoder was merged into the address generation and looping circuitry.

7.2.4 Breeze instruction memory

The Breeze instruction memory stores the Breeze instruction once it enters the processor. I do not estimate the cost of this storage because the G12-p ASIC library is not targeted for memory cells. However, the area, power, and timing estimates of the Breeze instruction memory are similar to an SRAM structure. The size of a Breeze instruction is typically 120 bytes.

7.2.5 Existing hardware units

The remaining hardware units that are required for the operation of the MediaBreeze architecture are the SIMD computation unit, data reorganization, load/store units, and data station. These hardware units are already present in commodity SIMD GPPs. However, the Breeze instruction decoder controls the operation of these units as opposed to the conventional control path. This mandates an extra multiplexer to differentiate between control from the conventional control path and the Breeze instruction decoder. I do not model any of the existing hardware units.

7.3 Area, Power, and Timing Results

Table 7.2 shows the composite estimates of area, power and timing for the address generation and looping circuitry when implemented using the ASIC cell-based methodology described in Section 7.1. The results for the Breeze instruction decoder are merged into the address generation and looping hardware. The power and area estimates in Table 7.2 correspond to a clock frequency of 1 GHz.

Table. 7.2 Area, Power, and Timing estimates of MediaBreeze units in a G12-p ASIC technology

Address generation (per stream)			Looping (five levels)		
Area (μm^2)	Power (mW)	Timing (ns)	Area (μm^2)	Power (mW)	Timing (ns)
57398 μm^2 (0.06 mm^2)	85.16 mW	1.74 ns	72830 μm^2 (0.07 mm^2)	88.57 mW	1.00 ns
Overall area = 0.31 mm^2			Overall power = 430 mW		

Area - The overall chip area required for implementing the address generation (all four streams), looping, and the Breeze instruction decoder is approximately 0.31 mm^2 . Table 7.3 shows the hardware area cost of commercial SIMD GPP implementations for comparison. In a 0.29-micron process, the increase in chip area for implementing VIS hardware into the Sparc processor family was 4 mm^2 , MMX into the Pentium family was 15 mm^2 , and AltiVec into the PowerPC family was 30 mm^2 [39]. In a 0.25-micron process, the AltiVec hardware was expected to occupy 15 mm^2 [39]. In a 0.18-micron technology, the die size of a Pentium III processor was 106 mm^2 with the MMX and SSE execution units requiring approximately 3.6 mm^2 [74]. Thus, the increase in area due to the MediaBreeze units for SIMD-related hardware is less than 10% and the overall increase in chip area is less than 0.3%.

Table. 7.3 Area of commercial SIMD and GPP implementations

VIS – 4 mm ² in a 0.29-micron process
MMX – 15 mm ² in a 0.29-micron process
AltiVec – 15 mm ² in a 0.25-micron process
Pentium III processor – 106 mm ² in a 0.18-micron process
MMX + SSE in a Pentium III processor – 3.6 mm ² in a 0.18-micron process

Power – The power consumed by the address generation, looping and the Breeze instruction decoder is approximately 430 mW at 1 GHz. GPPs with speeds in the GHz range typically consume a power ranging from 50W to 150W. Thus the increase in power consumption due to the added hardware is less than 1% of the overall processor power. The overall energy consumption of the MediaBreeze architecture would be less than that of a SIMD GPP because the Breeze instruction reduces the total dynamic instruction count (0.2 to 40% in media applications not including kernels). Since the Breeze instruction is densely encoded, very few Breeze instructions are needed for any media-processing algorithm. The number of dynamic instructions that need to be fetched and decoded reduces tremendously, which leads to a minimal use of the instruction fetch, decode, and issue logic in a superscalar processor. The instruction fetch and issue logic are expected to consume greater than 50% of the total execution power (not including the clock power) in future speculative processors

[108]. Once the Breeze instruction is decoded, the fetch, decode, and issue logic in the superscalar processor can be shutdown to save power.

Timing - Pipelining the hardware looping logic into two stages would allow for incorporating it into current high-speed superscalar out-of-order processors with over 1 GHz clock frequency. Similarly the address generation stage needs to be divided into three pipe stages to achieve frequencies greater than 1 GHz. The timing results show that incorporating the MediaBreeze hardware into a high-speed processor does not elongate the critical path of the processor (after pipelining). The Breeze instruction decoder multiplexers that control the hardware units introduce an extra gate delay in the pipeline. However, using a cell-based methodology gives a conservative estimate while custom design (typically used in commercial general-purpose processors) would allow for greater clock frequencies for the added MediaBreeze hardware. In spite of adding five pipeline stages, the overall pipeline depth of a processor is not affected because the looping and address generation stages bypass the conventional fetch, decode and issue pipeline stages.

Estimates of area, power, and timing across different technologies and optimizations are also described in Appendix B.

7.4 Summary

In this chapter, I estimated the cost of incorporating explicit hardware support into a SIMD GPP to execute the supporting instructions. Using an ASIC cell-based methodology targeting a 0.18-micron technology, I obtained area, power, and timing information for the MediaBreeze architecture components. The major findings of this chapter are:

- The area cost is less than 10% of the SIMD execution unit's area (such as MMX and SSE). When compared to the overall processor chip area, the increase is less only 0.3%.
- Power consumption of the added units is less than 1% of the total processor power.
- The MediaBreeze hardware units do not increase the effective pipeline depth of a high-speed GPP. With appropriate pipelining in the SIMD regions, a SIMD GPP with MediaBreeze hardware can be operated well over 1 GHz in a 0.18-micron technology.

I expect the overall energy consumption to be potentially lower when using a MediaBreeze augmented processor because the Breeze instruction is heav-

ily encoded and encompasses multiple operations. Power consumption is decreased due to the reduced use of traditional instruction fetch, decode, and issue logic for the duration of the Breeze instruction execution.

Chapter 8

Conclusion

As certain workloads become dominant, GPP architectures have added hardware support to execute them efficiently. SIMD extensions have been integrated into the processor by all GPP vendors to accelerate multimedia applications. In this dissertation, I show that SIMD support alone is not enough to extract parallelism in multimedia applications. Providing explicit hardware support to assist the SIMD computations is one way to bridge the existing performance gap between realistic versus ideal utilization of SIMD execution units. The major findings of this dissertation are summarized below.

- A comprehensive study of execution characteristics of commercial multimedia applications revealed that, contrary to popular belief, data caches are effective for multimedia programs. It is found that multimedia benchmarks achieve a 98% L1 data cache hit-rate (16 kB, 4-way) and a 99.5% global L2 hit-rate (512 kB, 4-way). When compared to SPEC benchmarks, the data cache performance is superior for multimedia ap-

plications. Another major finding of this study is that multimedia applications have a higher branch misprediction ratio than SPEC benchmarks.

- Analysis of bottlenecks in the execution of multimedia programs on SIMD GPPs revealed that there is a mismatch between the requirements of multimedia applications and the capabilities of SIMD GPPs. It is found that SIMD GPPs exploit DLP in the inner loops while significant DLP exists in the outer level nested loops of media applications.
- Experiments with SIMD GPPs revealed that a majority of dynamic instructions in the instruction stream of multimedia applications are not performing the useful computations, but merely supporting the computations. It is found that 75% to 85% of the dynamic instructions are supporting the useful computations and performing address generation, address transformation, loop branches, and loads/stores.
- Measuring the utilization of SIMD execution units revealed that they are vastly underutilized. Experiments on SIMD GPPs with a variety of media kernels and applications illustrate SIMD efficiency ranging only from 1% to 12%.
- Scalability tests with SIMD GPPs revealed that increasing the number of SIMD execution units does not improve performance. It is observed that

significant increase in scalar resources is required to increase the utilization of SIMD execution units using conventional ILP techniques.

- Providing explicit hardware support, that works in conjunction with a state-of-the-art SIMD GPP, for eliminating and reducing the overhead dramatically accelerates media workloads without deteriorating the performance of general-purpose workloads. The Breeze instruction was introduced to capture the useful and supporting instructions simultaneously. The Breeze instruction is similar to multidimensional vector instructions. It is found that a 2-way SIMD GPP augmented with MediaBreeze hardware support significantly outperforms a conventional 16-way SIMD GPP on multimedia kernels. On media applications, a 2- and 4-way SIMD GPP enhanced with MediaBreeze hardware support is superior to a 4- and 8-way conventional SIMD GPP.
- The cost of adding the MediaBreeze hardware to a SIMD GPP is negligible compared to the performance improvements. It is found that the MediaBreeze hardware units occupy less than 0.3% of the overall processor chip area, consumes less than 1% of the total processor power, and does not increase the effective pipeline depth of a high-speed GPP.

In summary, while SIMD extensions accelerate media applications, several bottlenecks exist in SIMD GPPs that prevent even higher performance im-

provements. In this dissertation, I propose a cost-effective solution to address the supporting instructions rather than focusing on the SIMD computations themselves. If any media processor designer decides to exploit more parallelism by just scaling the current architectures, they should scale the non-SIMD part much more aggressively than the SIMD part. Other future opportunities in improving media application performance on GPPs involves improving compiler abilities to extract DLP and ILP in media applications.

Appendix A

Performance Monitoring Events on the P6 Microarchitecture

Table A.1 lists the P6 microarchitecture counter based performance measures. For a complete list of performance events that can be measured using the counters, the interested reader is referred to [45][46].

Table A.1 P6 microarchitecture counters based performance measures

Performance Measure	Numerator Event	Denominator Event
Data references per instruction	DATA_MEM_REFS	INST_RETIRED
L1 Dcache misses per instruction	DCU_LINES_IN	INST_RETIRED
L1 Icache misses per instruction	L2_IFETCH	INST_RETIRED
ITLB misses per instruction	ITLB_MISS	INST_RETIRED
Installs cycles per instruction	IFU_MEM_STALL	INST_RETIRED
L1 cache misses per instruction	L2_RQSTS	INST_RETIRED
L2 cache misses per instruction	L2_LINES_IN	INST_RETIRED
L2 Miss ratio	L2_LINES_IN	L2_RQSTS
Memory transactions per instruction	BUS_TRAN_MEM	INST_RETIRED
FLOPS per instruction	FLOPS	INST_RETIRED
UOPS per instruction	UOPS_RETIRED	INST_RETIRED
Speculative execution factor	INST_DECODED	INST_RETIRED
Branch frequency	BR_INST_RETIRED	INST_RETIRED
Branch mispredict ratio	BR_MISS_PRED_RETIRED	BR_INST_RETIRED
Branch taken ratio	BR_TAKEN_RETIRED	BR_INST_RETIRED
BTB miss ratio	BTB_MISSES	BR_INST_DECODED
Branch Speculation factor	BR_INST_DECODED	BR_INST_RETIRED
Resource stalls per instruction	RESOURCE_STALLS	INST_RETIRED
Cycles per instruction	CPU_CLK_UNHALTED	INST_RETIRED

Appendix B

Hardware Cost of MediaBreeze Architecture across Different ASIC Technologies

Table B.1 lists the different cell-based libraries for evaluating area, power, and timing tradeoffs in MediaBreeze hardware. The G12-p technology discussed in chapter 7 is also included in this appendix.

Table. B.1 List of cell-based libraries used in the synthesis of MediaBreeze hardware units

Library name	Description
lcbg12-p (G12-p)	A 0.18-micron L-drawn (0.13-micron L-effective) CMOS process. Highest performance solution at 1.8 V with high drive cells optimized for long interconnects associated with large designs.
lcbg12-d (G12-d)	A 0.18-micron L-drawn (0.13-micron L-effective) CMOS process. Optimized for performance, density, and power for most applications at 1.8 V.
lcbg12-l (G12-l)	A 0.18-micron L-drawn (0.13-micron L-effective) CMOS process. Ultra-low power and high-density solution with a low dynamic and standby leakage current at 1.8 V.
lcbg11-p (G11-p)	A 0.25-micron L-drawn (0.18-micron L-effective) CMOS process. Highest performance solution at 2.5 V.
lcbg11-v (G11-v)	A 0.25-micron L-drawn (0.18-micron L-effective) CMOS process. Ultra-low power and cost sensitive solution at 1.8 V.
lcbg10-p (G10-p)	A 0.35-micron L-drawn (0.25-micron L-effective) CMOS process. Optimized for high performance at 3.3 V.

Table B.2 shows the composite estimates of timing, area, and power consumption for the hardware looping and address generation circuitry when implemented using the cell-based methodology. The power and area estimates in Table B.2 correspond to a clock frequency of 1 GHz. Power consumption at 250 MHz (corresponding to the slowest technology and circuit) is also shown in parenthesis.

Table B.2 Timing, area, and power estimates across different technologies

	Hardware Looping (5 loops)			Address Generation (per stream)		
	Time (ns)	Area (μm^2)	Power (mW)	Time (ns)	Area (μm^2)	Power (mW)
G12-p	1.00 ns	72830 μm^2	88.57 mW (22.1 mW)	1.74 ns	57398 μm^2	85.16 mW (21.2 mW)
G12-d	1.16 ns	64666 μm^2	62.40 mW (15.4 mW)	1.91 ns	41053 μm^2	46.18 mW (11.4 mW)
G12-l	1.30 ns	65714 μm^2	55.44 mW (13.8 mW)	2.22 ns	41144 μm^2	42.34 mW (10.5 mW)
G11-p	1.49 ns	273249 μm^2	249.30 mW (61.6 mW)	2.60 ns	165099 μm^2	193.20 mW (46.2 mW)
G11-v	1.90 ns	500864 μm^2	166.00 mW (41.3 mW)	3.29 ns	204603 μm^2	82.93 mW (20.6 mW)
G10-p	2.01 ns	-	846.90 mW (210 mW)	3.76 ns	-	554.30 mw (138 mw)

Figure B.1 shows the percentage of interconnect area in the MediaBreeze hardware. Figure B.2 shows the power consumption split into cell internal power and net switching power.

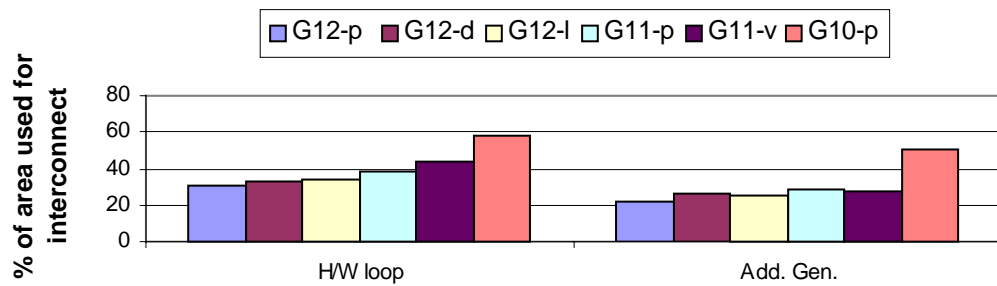


Figure. B.1. Percentage of interconnect area in the overall area

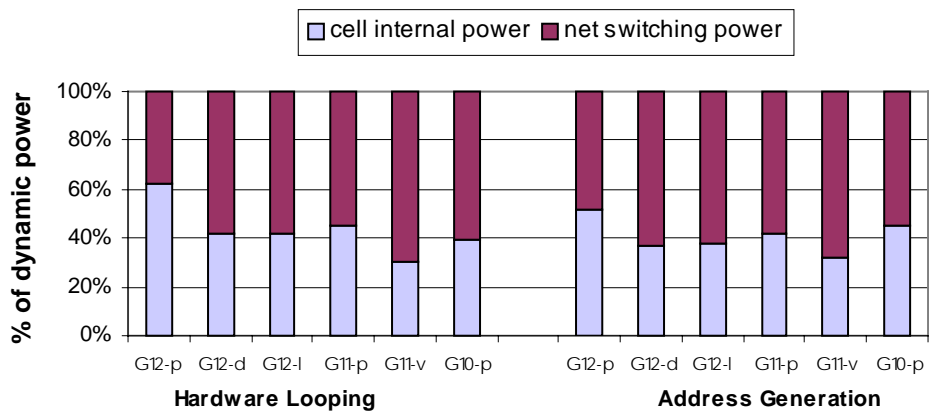


Figure. B.2. Breakdown of dynamic power into cell internal power and net switching power

Bibliography

- [1] D. H. Allen, S. H. Dhong, H. P. Hofstee, J. Leenstra, K. J. Nowka, D. L. Stasiak, D. F. Wendel, "Custom circuit design as a driver of microprocessor performance," *IBM Journal of Research and Development*, vol. 44, no. 6, Nov. 2000. Available at:
<http://www.research.ibm.com/journal/rd/446/allen.html>
- [2] AMD 3DNow! Website. Available at:
<http://www.amd.com/products/cpg/3dnow/index.html>.
- [3] Analog Devices JPEG2000 co-processor. Available at:
http://www.analog.com/pdf/preview/ADV-JP2000_pra.gif
- [4] K. Asanovic, "Vector microprocessors," Ph.D. Thesis, Computer Science Division, University of California at Berkeley, May 1998.
- [5] T. Austin and G. Sohi, "Tetra: Evaluation of serial program performance on fine-grain parallel processors," Technical Report, University of Wisconsin; Also "Dynamic dependency analysis of ordinary programs," *Proc. of the 19th Int. Sym. on Computer Architecture*, pp. 342-351, May 1992.
- [6] R. D. Barnes, R. Chaiken, and D. M. Gillies, "Feedback-directed data cache optimizations for the x86," *Proc. of 2nd ACM Workshop on Feedback-Directed Optimization (FDO) in conjunction with the 32nd Int. Sym. on Microarchitecture*, Nov. 1999.
- [7] C. Basoglu, W. Lee, and J. S. O'Donnell, "The Map1000A VLIW media processor," *IEEE Micro*, vol. 20, no. 2, pp.48-59. Mar/Apr. 2000.
- [8] A. S. Berrached, P. T. Hulina, and L. D. Coraor, "Specification of a co-processor for efficient access of data structures," *Proc. Hawaii Int. Conf. on System Sciences*, pp. 496-505, Jan. 1992.
- [9] D. Bhandarkar and J. Ding, "Performance characterization of the Pentium Pro processor," *Proc. High Performance Computer Architecture*, pp. 288-297, Feb. 1997.

- [10] R. Bhargava, L. John, B. Evans, and R. Radhakrishnan, "Evaluating MMX technology using DSP and multimedia applications," *Proc. of IEEE/ACM Sym. on Microarchitecture*, pp. 37-46, Dec. 1998.
- [11] J. Bier and J. Eyre, "Independent DSP benchmarking: Methodologies and latest results," *Proc. Int. Conf. on Signal Processing Applications and Technology*, Sep. 1998.
- [12] D. Bistry, C. Dulong, M. Gutman, M. Julier, and M. Keith, *The complete guide to MMX technology*, McGraw-Hill, 1997.
- [13] G. Blalock, "The BDTIMark: A measure of DSP execution speed," 1997. White Paper from Berkeley Design Technology, Inc. Available at: <http://www.bdti.com/articles/wtpaper.htm>.
- [14] G. Blalock, "Microprocessors Outperform DSPs 2:1," *MicroProcessor Report*, vol. 10, no. 17, pp. 1-4, Dec. 1995.
- [15] D. Burger, and T. M. Austin, "The SimpleScalar tool set," Version 2.0. *Technical Report 1342*, Univ. of Wisconsin-Madison, Comp. Sci. Dept. 1997.
- [16] C-Cube products. Available at: <http://www.c-cube.com/products.cfm>
- [17] A. Chang, W. J. Dally, S. W. Keckler, N. P. Carter, and W. S. Lee, "The effects of explicitly parallel mechanisms on the Multi-ALU processor cluster pipeline," *Proc. IEEE Conf. on Computer Design*, pp. 474-481, Oct. 1998.
- [18] T. F. Chen and J. L. Baer, "A performance study of software and hardware data prefetching schemes," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 223-232, Apr. 1994.
- [19] W. Chen, H.J. Reekie, S. Bhave and E.A. Lee, "Native signal processing on the UltraSparc in the Ptolemy environment," *Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers*, pp. 1368-1372, Nov. 1996.
- [20] T. M. Conte, P. K. Dubey, M. D. Jennings, R. B. Lee, A. Peleg, S. Rathnam, M. Schlansker, P. Song, and A. Wolfe, "Challenges to combining general-purpose and multimedia processors," *IEEE Computer Magazine*, vol. 30, no. 12, pp. 33-37, Dec. 1997.

- [21] J. Corbal, M. Valero, and R. Espasa, "Exploiting a new level of DLP in multimedia applications," *Proc. IEEE/ACM Sym. on Microarchitecture*, pp. 72-79, Nov. 1999.
- [22] J. Corbal, R. Espasa, and M. Valero, "On the efficiency of reductions in micro-SIMD media extensions," *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, Sep. 2001, to appear.
- [23] H. G. Cragon, and W. J. Watson, "The TI advanced scientific computer." *IEEE Computer Magazine*, pp. 55-64, Jan. 1989.
- [24] P. Crowley and J. Baer, "Trace sampling for desktop applications on Windows NT," *Workshop on Workload Characterization held in conjunction with Micro-31*, Nov 1998. Appears in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE Computer Society Press, 1999.
- [25] R. Cucchiara, M. Piccardi, and A. Prati, "Exploiting cache in multimedia," *Proc. IEEE Int. Conf. on Multimedia Computing and Systems*, pp. 345-350, vol. 1, Jun. 1999.
- [26] K. Diefendorff and P.K. Dubey, "How multimedia workloads will change processor design," *IEEE Computer Magazine*, vol. 30, no. 9, pp. 43-45, Sep. 1997.
- [27] K. Diefendorff, P. K. Dubey, R. Hochsprung, and H. Scales, "AltiVec extension to PowerPC accelerates media processing," *IEEE Micro*, vol. 20, no. 2, pp. 85-95, Mar/Apr 2000.
- [28] EDN Embedded Microprocessor Benchmark Consortium, Available at: <http://www.eembc.org>.
- [29] P.M. Embree, "*C Algorithms for Real-Time DSP*," NJ: Prentice Hall, 1995.
- [30] P. Faraboschi, G. Desoli, and J A. Fisher, "The Latest Word in Digital and Media Processing," *IEEE Signal Processing Magazine*, vol. 15, no. 2, pp. 59-85, Mar. 1998.
- [31] M. J. Flynn, "Very high-speed computing systems," *Proc. of the IEEE*, vol. 54, no.12, pp.1901-1909, 1966.

- [32] J. Fridman and Z. Greenfield, "The TigerSHARC DSP architecture," *IEEE Micro*, vol. 20, no. 1, pp. 66-76, Jan/Feb. 2000.
- [33] J. Fritts, and W. Wolf, "Dynamic parallel media processing using speculative broadcast loop (SBL)," *Proc. Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia (held in conjunction with IPDPS'01)*, Apr. 2001.
- [34] J. Fritts, "Architecture and compiler design issues in programmable media processors," Ph.D. Thesis, Dept. of Electrical Engineering, Princeton University, 2000.
- [35] S. C. Goldstein, H. Schmit, M. Moe, M. Nudiu, S. Cadambi, R. R. Taylor, and R. Laufer, "PipeRench: A coprocessor for streaming multimedia acceleration," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 28-39, May 1999.
- [36] J. R. Goodman, T. J. Hsieh, K. Liou, A. R. Pleszkun, P. B. Schechter, and H. C. Young, "PIPE: A VLSI decoupled architecture," *Proc. IEEE Sym. on Computer Architecture*, pp. 20-27, Jun. 1985.
- [37] R. Gonzalez and M. Horowitz, "Energy dissipation in general purpose microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 9, pp. 1277-1284, Sep. 1996.
- [38] L. Gwennap, "Intel's MMX speeds multimedia," *Microprocessor Report*, vol. 10, no. 3, p. 1, 1996.
- [39] L. Gwennap, "AltiVec vectorizes PowerPC," *Microprocessor Report*, vol. 12, no. 6, May 11, 1998.
- [40] J. C. Gyllenhaal, W. Hwu, and B. R. Rau, "IMPACT Technical Report," *IMPACT-96-03*, University of Illinois, Urbana, IL, Mar. 1996.
- [41] D. Hansson, "Reducing power in a RISC/DSP core," *Electronic Engineering Times*, Aug. 7, 2000. Available at: <http://www.eetimes.com/story/OEG20000807S0034>.
- [42] C. J. Hughes, P. Kaul, S. V. Adve, R. Jain, C. Park, and J. Srinivasan, "Variability in the execution of multimedia applications and implications

- for architecture,” *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 254-265, Jul. 2001.
- [43] P. T. Hulina, L. D. Coraor, L. Kurian, and E. John, “Design and VLSI implementation of an address generation coprocessor,” *IEE Proc. on Computers and Digital Techniques*, vol. 142, no. 2, pp. 145-151, Mar. 1995.
- [44] Intel, “Performance Library Suite”. Available at:
<http://developer.intel.com/software/products/perflib/index.htm>.
- [45] Intel Literature, P6 architecture developer’s manuals. Available at:
<http://developer.intel.com/design/processor/>.
- [46] Intel Architecture Optimization Reference Manual. Available at:
<http://developer.intel.com/design/pentiumii/manuals/245127.htm>.
- [47] Intel XScale Microarchitecture. Available at:
<http://developer.intel.com/design/intelxscale/ixm.htm>.
- [48] N. P. Jouppi and D. W. Wall, “Available instruction-level parallelism for superscalar and superpipelined machines,” *Proc. of Int. Sym. on Architectural Support for Programming Languages and Operating Systems*, pp. 272-282, Apr. 1989.
- [49] N. Jouppi, “Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers,” *Proc. IEEE Sym. on Computer Architecture*, pp. 364-373, May 1990.
- [50] B. Juurlink, D. Tcheressiz, S. Vassiliadis, and H. Wijshoff, "Implementation and evaluation of the complex streamed instruction set," *Proc. Int. Conf. on Parallel Architectures and Compilation Techniques*, Sep. 2001, to appear.
- [51] J. Kalamatianos, R. Chaiken and D. Kaeli, “Parameter value characterization of Windows NT-based applications,” *Workshop on Workload Characterization held in conjunction with Micro-31*, Nov 1998. Appears in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE Computer Society Press, 1999.

- [52] U. J. Kapasi, W. J. Dally, S. Rixner, P. R. Mattson, J. D. Owens, and B. Khailany, "Efficient conditional operations for data-parallel architectures," *Proc. IEEE/ACM Sym. on Microarchitecture*, pp. 159-170, Dec. 2000.
- [53] L. Kohn, G. Maturana, M. Tremblay, A. Prabhu, and G. Zyner, "The Visual Instruction Set (VIS) in UltraSPARC", *COMPCON Digest of Papers*, pp. 462-469, Mar. 1995.
- [54] C. E. Kozyrakis and D. A. Patterson, "A new direction for computer architecture research," *IEEE Computer Magazine*, vol. 31, no. 11, pp. 24-32, Nov. 1998.
- [55] D. J. Kuck, and R. A. Stokes, "The Burroughs scientific processor (BSP)," *IEEE Trans. on Computers*, vol. 31, no. 5, pp. 363-376, 1982.
- [56] A. Kunimatsu, N. Ide, T. Sato, Y. Endo, H. Murakami, T. Kamei, M. Hirano, F. Ishihara, H. Tago, M. Oka, A. Ohba, T. Yutaka, T. Okada, and M. Suzuoki, "Vector unit architecture for emotion synthesis," *IEEE Micro*, vol. 20, no. 2, pp.40-47. Mar/Apr. 2000.
- [57] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee. *DSP processor fundamentals: architectures and features*, Chapter 8, IEEE Press series on Signal Processing, ISBN 0-7803-3405-1, 1997.
- [58] V. Lappalainen, "Performance analysis of Intel MMX technology for an H.263 video encoder," *Proc. ACM Int. Conf. on Multimedia*, pp. 309-314, Sep. 1998.
- [59] C. Lee, M. Potkonjak and W.H. Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications Systems," *Proc. IEEE/ACM Sym. on Microarchitecture*, pp. 330-335, Dec 1997.
- [60] C. G. Lee, and M. G. Stoodley, "Simple vector microprocessors for multimedia applications," *Proc. IEEE/ACM Sym. on Microarchitecture*, pp. 25-36, Dec. 1998.
- [61] D.C. Lee, P.J. Crowley, J. Baer, T. Anderson, and B. Bershad, "Execution characteristics of desktop applications on Windows NT," *Proc. IEEE Int. Sym. on Computer Architecture*, pp. 27-38, Jun. 1998.

- [62] R. B. Lee, "Multimedia extensions for general-purpose processors," *Proc. IEEE Workshop on Signal Processing Systems*, pp. 9-23, Nov. 1997.
- [63] H. Liao and A. Wolfe, "Available parallelism in video applications," *Proc. IEEE/ACM Int. Sym. on Microarchitecture*, pp. 321-329, Dec. 1997.
- [64] LSI Logic ASIC technologies. Available online at:
<http://www.lsilogic.com/products/asic/technologies/index.html>
- [65] "LSI Logic ASKK Documentation System".
- [66] S. A. Mckee, "Maximizing memory bandwidth for streamed computations," Ph.D. Thesis, School of Engineering and Applied Science, University of Virginia, May 1995.
- [67] Media Processors "Software-Driven Multimedia", A white paper by Chromatics Research Inc. Available at: <http://www.vxm.com/21R.84.html>
- [68] "Media processors target digital video roles". *EDN Magazine*, Sep. 1998. Available at:
<http://www.ednmag.com/ednmag/reg/1998/090198/18df1.htm>
- [69] Microprocessor Architecture for Java Computing from Sun Microsystems. Available at:
[http://www.sun.com/microelectron-ics/MAJC/documentation/docs/majctutorial.pdf](http://www.sun.com/microelectronics/MAJC/documentation/docs/majctutorial.pdf)
- [70] Motorola, "AltiVec Technology", Available at:
<http://www.mot.com/SPS/PowerPC/AltiVec/index.html>.
- [71] H. V. Nguyen, and L. K. John, "Exploiting SIMD parallelism in DSP and multimedia algorithms using the AltiVec technology," *Proc. ACM Int. Conf. on Supercomputing*, pp. 11-20, Jun. 1999.
- [72] S. Palacharla and R. E. Kessler, "Evaluating stream buffers as a secondary cache replacement," *Proc. IEEE Int. Sym. on Computer Architecture*, pp. 24-33, Apr. 1994.

- [73] A. Peleg and U. Weiser, "The MMX technology extension to the Intel architecture," *IEEE Micro*, vol. 16, no. 4, pp. 42-50, Aug. 1996.
- [74] Pentium III implementation (IA-32). Available at:
<http://www.sandpile.org/impl/p3.htm>.
- [75] A. R. Pleszkun, and E. S. Davidson, "Structured memory access architecture," *Proc. IEEE Int. Conf. on Parallel Processing*, pp. 461-471, Aug. 1983.
- [76] F. Quintana, J. Corbal, R. Espasa, and M. Valero, "Adding a vector unit to a superscalar processor," *Proc. ACM Int. Conf. on Supercomputing*, pp. 1-10, Jun. 1999.
- [77] R. Radhakrishnan and F. Rawson, "Characterizing the behavior of windows NT web server workloads using processor performance counters," *Workshop on Workload Characterization held in conjunction with Micro-31*, Nov 1998. Appears in *Workload Characterization: Methodology and Case Studies*, edited by John and Maynard, IEEE Computer Society Press, 1999.
- [78] P. Ranganathan, S. Adve, and N. Jouppi, "Performance of image and video processing with general-purpose processors and media ISA extensions," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 124-135, May 1999.
- [79] P. Ranganathan, S. Adve, and N. Jouppi, "Reconfigurable caches and their application to media processing," *Proc. 27th IEEE/ACM Sym. on Computer Architecture*, pp. 214-224, Jun. 2000.
- [80] A. K. Riemens, K. A. Vissers, R. J. Schutten, F. W. Sijstermans, G. J. Hekstra, and G. D. La Hei, "TriMedia CPU64 Application Domain and Benchmark Suite," *Proc. Int. Conf. on Computer Design*, pp. 580-585, Oct. 1999.
- [81] S. Rixner, W. J. Dally, U. J. Kapasi, B. Khailany, A. Lopez-Lagunas, P. R. Mattson, and J. D. Owens, "A bandwidth-efficient architecture for media processing," *Proc. IEEE/ACM Sym. on Microarchitecture*, pp. 3-13, Dec, 1998.

- [82] S. Rixner, W. J. Dally, U. J. Kapasi, P. Mattson, and J. D. Owens, "Memory access scheduling," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 128-138, Jun. 2000.
- [83] S. Sair and M. Charney, "Memory behavior of the SPEC2000 benchmark suite," *IBM Research Report*, Oct. 2000.
- [84] E. Salami, J. Corbal, M. Valero, and R. Espasa, "An evaluation of different DLP alternatives for the embedded domain," *Proc. Workshop on Media Processors and DSPs in conjunction with Micro-32*, Nov. 1999.
- [85] R. R. Shively, "Architecture of a programmable digital signal processor," *IEEE Trans. on Computers*, vol. 31, no. 1, pp. 16-22, Jan. 1978.
- [86] Siglib version 2.4, Numerix Co Ltd. Available at:
<http://www.numerix.co.uk>.
- [87] N. Slingerland and A. J. Smith, "Cache performance for multimedia applications," *Proc. ACM Intl. Conf. on Supercomputing*, pp. 204-217, Jun. 2001.
- [88] J. E. Smith, "Decoupled access/execute computer architectures," *ACM Trans. on Computer Systems*, vol. 2, no. 4, pp. 289-308, Nov. 1984.
- [89] J. E. Smith, S. Weiss, and N. Y. Pang, "A simulation study of decoupled architecture computers," *IEEE Trans. on Computers*, vol. 35, no. 8, pp. 692-701, Aug. 1986.
- [90] S. Sohoni, Z. Xu, R. Min, and Y. Hu, "A study of memory system performance of multimedia applications," *Proc. ACM Sigmetrics*, pp. 206-215, Jun. 2001.
- [91] Source code for the benchmarks. Available at:
http://www.geocities.com/microarch_34/Benchmarks
- [92] Speech Coding Resource. Available at:
http://www-mobile.ecs.soton.ac.uk/speech_codecs/.
- [93] S. Sriram and C. Hung, "MPEG-2 video Decoding on the TMS320C6x DSP Architecture," *Proc. IEEE Asilomar Conf. on Signals, Systems, and Comp.*, pp. Pacific Grove, CA.

- [94] “Synopsys online Sold Documentation system,” version 2000-0.5-1.
- [95] D. Talla, L. K. John, V. Lapinskii and B. L. Evans “Evaluating signal processing and multimedia applications on SIMD, VLIW and superscalar architectures,” *Proc. IEEE Int. Conf. on Computer Design*, pp. 163-172, Sep. 2000.
- [96] D. Talla and L. K. John, “Execution characteristics of multimedia applications on a Pentium II processor,” *Proc. IEEE Int. Performance, Computing, and Communications Conference*, pp. 516-524, Feb. 2000.
- [97] D. Talla and L. K. John, “Performance evaluation and benchmarking of native signal processing”, *Proc. European Conf. on Parallel Processing, Lecture Notes in Computer Science #1685*, pp. 266-270, Sep. 1999.
- [98] D. Talla and L. K. John, “Quantifying the effectiveness of MMX in Native Signal Processing”, *Proc. IEEE Mid-West Symposium on Circuits and Systems*, pp. 18-21, Aug. 1999.
- [99] D. Talla and L. K. John, “Cost-effective hardware acceleration of multimedia applications,” *Proc. IEEE Int. Conf. on Computer Design*, Sep. 2001, to appear.
- [100] Texas Instruments, “TMS320C6000 CPU and instruction set reference guide”, *Lit. Num. SPRU189D*.
- [101] Texas Instruments, “TMS320C6000 benchmarks”, Available at: <http://www.ti.com/sc/docs/products/dsp/c6000/62bench.htm>.
- [102] Texas Instruments, “TMS320C6x Optimizing C Compiler User’s Guide”, *Lit. Num. SPRU187B*.
- [103] Texas Instruments, “TMS320C64x DSP Technical Brief”. Available at: <http://www.ti.com/sc/docs/products/dsp/c6000/c64xmptb.pdf>.
- [104] J. E. Thornton, “Parallel operation in the Control Data 6600,” *Fall Joint Computer Conference*, vol. 26, pp. 33-40, 1961.
- [105] J. T. J. van Eijndhoven, F. W. Sijstermans, K. A. Vissers, E. J. D. Pol, M. J. A. Tromp, P. Struik, R. H. J. Bloks, P. van der Wolf, A. D. Pimen-

- tel, and H. P. E. Vranken, "TriMedia CPU64 Architecture," *Proc. Int. Conf. on Computer Design*, pp. 586-592, Oct. 1999.
- [106] S. Vassiliadis, B. Juurlink, and E. A. Hakkennes, "Complex streamed instructions: introduction and initial evaluation," *Proc. IEEE Euromicro Conf.*, vol. 1, pp. 400-408, Sep. 2000.
- [107] F. Vermeulen, L. Nachtergaele, F. Catthoor, D. Verkest, and H. De Man, "Flexible hardware acceleration for multimedia oriented microprocessors," *Proc. IEEE/ACM Sym. on Microarchitecture*, pp. 171-177, Dec. 2000.
- [108] K. Wilcox and S. Manne, "Alpha processors: A history of power issues and a look to the future," *Cool Chips Tutorial in conjunction with IEEE/ACM Sym. on Microarchitecture*, Nov. 1999.
- [109] Wm. A. Wolf, "Evaluation of the WM architecture," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 382-390, May 1992.
- [110] Z. A. Ye, A. Moshovos, S. Hauck, and P. Banerjee, "CHIMAERA: A high-performance architecture with a tightly-coupled reconfigurable functional unit," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 225-235, Jun. 2000.
- [111] G. K. Yeap, *Practical Low Power Digital VLSI Design*, Kluwer Academic Publishers, 1998.
- [112] Y. Zhang, and G. B. Adams, "Performance modeling and code partitioning for the DS architecture," *Proc. IEEE/ACM Sym. on Computer Architecture*, pp. 293-304, Jun. 1998.
- [113] V. Zivojnovic, J. Martinez, C. Schlager and H. Meyr, "DSPstone: A DSP-Oriented benchmarking methodology," *Proc. Int. Conf. on Signal Proc. Applications and Technology*, Oct. 1994.
- [114] D. F. Zucker, "Architecture and arithmetic for multimedia enhanced processors," Ph.D. Thesis, Dept. of Electrical Engineering, Stanford University, Jun. 1997.

Vita

Deependra Talla was born in Visakhapatnam, India, on April 15, 1975, as the son of Dr. Madhavi Venkata Talla and Dr. Seshi Reddy Talla. After completing his high school education at Timpany School in Visakhapatnam, India, he entered the College of Engineering, Andhra University in Visakhapatnam, India in August 1992. He received the degree of Bachelor of Engineering in Electronics and Communication Engineering from Andhra University in June 1996. He joined the graduate program for Electrical and Computer Engineering at Villanova University, Villanova, PA in August 1996 and obtained the degree of Master of Science in Electrical Engineering in August 1998. In August 1998, he entered the Ph.D. program in Computer Engineering at The University of Texas at Austin. During the summers of 1999 and 2000, he interned at Texas Instruments Inc. Dallas working on the architecture of digital still cameras. He is a student member of IEEE, IEEE Computer Society, ACM, and ACM Sigarch.

Permanent Address: Navodaya Nursing Home

MVP colony

Visakhapatnam, India 530 017

This dissertation was typed by the author.