ENERGY EFFICIENT SUPPORT FOR ALL LEVELS OF PARALLELISM FOR COMPLEX MEDIA APPLICATIONS

BY

RUCHIRA SASANKA

B.S., University of Moratuwa, 1998
M.S., University of Illinois at Urbana-Champaign, 2002

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2005

Urbana, Illinois

Abstract

Real-time complex media applications are becoming increasingly common on general-purpose systems such as desktop, laptop, and handheld computers. However, real-time execution of such complex media applications needs a considerable amount of processing power that often surpasses the capabilities of current superscalar processors. Further, high performance processors are often constrained by power and energy consumption, especially in the mobile systems where media applications have become popular.

The objective of this dissertation is to develop general-purpose processors that can meet the performance demands of future media applications in an energy-efficient way, while also continuing to work well on other common workloads for desktop, laptop, and handheld systems.

Fortunately, most media applications have a lot of *parallelism* that can be exploited for energy-efficient high-performance designs. Media applications exhibit multiple types of parallelism: thread-level parallelism (TLP), data-level parallelism (DLP), and instruction-level parallelism (ILP). In this work, we investigate exploiting all these three forms of parallelism to provide both high performance and energy efficiency.

This dissertation makes three broad contributions. First, we analyze the parallelism in complex media applications and make the case that contemporary media applications require efficient support for multiple types of parallelism, including ILP, TLP, and various forms of data-level parallelism such as sub-word SIMD, short vectors, and streams.

Second, to find the most energy efficient way of exploiting TLP, we perform a comparison between chip multi-processing (CMP) and simultaneous multi-threading (SMT). We perform this comparison for a large number of performance points derived using different processor architectures and frequencies/voltages. From this study, we find that, at equal performance, CMP is more energy efficient than SMT, especially when supporting four or more threads. We also find that the best

SMT and the best CMP configuration for a given performance target have different architecture and frequency/voltage. From our analysis, we find that the relative energy efficiency depends on a subtle interplay between various factors such as capacitance, voltage, IPC, frequency, and the level of clock gating, as well as workload features. Although CMP shows a clear energy advantage for four-thread (and higher) workloads, it comes at the cost of increased silicon area. We therefore investigate a hybrid solution where a CMP is built out of SMT cores, and find it to be an effective compromise. We use this hybrid architecture as the basis for providing ILP and TLP for our new architecture described next.

Finally, based on the results of the above two studies, we propose a complete architecture, called ALP, that effectively supports *a*ll *l*evels of *p*arallelism described above in an energy efficient way, using an *evolutionary programming model and hardware*. The most novel part of ALP is a DLP technique called *SIMD vectors and streams*, which is integrated within a conventional superscalar based CMP/SMT architecture with sub-word SIMD. This technique lies between sub-word SIMD and vectors, providing significant benefits over the former at a lower cost than the latter. Our evaluations show that each form of parallelism supported by ALP is important. Specifically, SIMD vectors and SIMD streams are effective – compared to a system with the other enhancements in ALP, they give speedups of 1.1X to 3.4X and energy-delay product improvements of 1.1X to 5.1X for applications with DLP.

More broadly, our results show that conventional architectures augmented with evolutionary mechanisms can provide high performance and energy savings for complex media applications without resorting to radically different architectures and programming paradigms.

Acknowledgements

I would like to express my heartfelt gratitude to Prof. Sarita Adve, who served as my thesis advisor for five years. Her insights and attention to detail were instrumental in achieving my academic goals. I would also like to thank members of my Ph.D. committee, Prof. Marc Snir, Prof. Josep Torrellas, Prof. Sanjay Patel, and Prof. Craig Zilles for the valuable feedback I received, especially at my preliminary exam.

I would also like to thank my mentors from Intel, Yen-Kuang Chen (YK) and Eric Debes who provided valuable insights into the applications we were studying. They provided the essential industrial perspective for all my research work.

I was fortunate to work with Chris Hughes, who guided my early days of research. He was both a friend and a mentor and his advice was invaluable in avoiding many pitfalls encountered in the first years as a graduate student.

I would like to thank the members of the RSIM group: Alex, Jayanth, Jerry, Rohit, and Vibhore. Alex was especially helpful with the ALP project and did a wonderful job as the application wizard.

I also extend my gratitude to Intel corporation for providing me with an Intel graduate fellowship in my fifth year of graduate studies. I would also like to acknowledge the following funding for this work: a gift from Intel corporation, an equipment donation from AMD Corp., and grants from the National Science Foundation (Grant No. CCR-0209198 and EIA-0224453).

Last but not least, I thank my parents, wife, and son, who helped me endure the hardships of being a graduate student.

Table of Contents

Chapter 1

Introduction

1.1 Motivation and Objective

Real-time media applications are becoming increasingly common on general-purpose systems such as desktops, laptops, tablet PCs, and handheld computers [17]. Users of such devices are using media applications for communication (e.g., video conferencing), entertainment (e.g., watching/creating movies, virtual reality, Internet, gaming), authentication (e.g., face recognition), and as a part of the human computer interface (e.g., speech/handwriting recognition). Although system designers have resorted to application specific processors (ASICs) to cater to media applications, general-purpose processors are becoming popular for these applications. This is mainly because of the growing realization that programmability is important for this application domain as well, due to a wide range of complex multimedia applications, multiple standards, and proprietary solutions. However, real-time execution of media applications on general-purpose processors faces two principal challenges:

- Performance: Real-time execution of complex media applications needs a considerable amount of processing power that often surpasses the capabilities of current superscalars. Due to their real-time nature, temporal correctness (i.e., meeting a given deadline) is as important as functional correctness (i.e., producing the correct output).

- Energy Efficiency: High performance processors are often constrained by power/energy consumption, especially in the mobile systems where media applications have become popular.

The objective of this dissertation is to develop general-purpose processors that can meet the performance demands of future media applications in an energy-efficient way, while also continuing to work well on other common workloads for desktop, laptop, and handheld systems.

To meet the above challenges and fulfill our objective, we resort to one key opportunity presented by these applications, viz., *parallelism*. Most media applications have a lot of parallelism in the form of Data Level Parallelism (DLP), Thread Level Parallelism (TLP), and Instruction Level Parallelism (ILP) that can be exploited for energy-efficient high-performance designs. This work investigates the nature of parallelism in complex media applications and how to exploit that parallelism in general-purpose architectures for better performance and energy efficiency.

1.1.1 Exploiting Parallelism for Performance and Energy Efficiency

All three forms of parallelism can be used for achieving both high performance and high energy efficiency. For instance, supporting DLP, TLP, and/or ILP allows more operations to be carried out in a cycle and hence increases performance. Similarly, encoding multiple operations with one instruction with DLP allows more energy efficient use of instructions. Further, the performance gain resulting from all three forms of parallelism can be used to increase the energy efficiency by employing lower frequencies/voltages and by reducing the duration of static (leakage) power consumption. However, current general purpose superscalar processors are mostly effective in exploiting ILP and, further, there is a large body of work concerning improving ILP on such processors. Consequently, this dissertation investigates and proposes ways of exploiting DLP and TLP in media applications for both high performance and energy efficiency.

State-of-the Art

There are several classes of machines that target media applications. Commercial general-purpose processors support sub-word SIMD instructions *(SIMD for short)* like Intel MMX/SSE, IBM AltiVec, and Sun VIS. These instructions are well suited to handling small grain DLP (i.e., short vectors) that require frequent data shuffling and reductions. However, such support is inadequate for large amounts of DLP such as long streams and long vectors.

The conventional wisdom has been that the parallelism in media applications (especially media

kernels) is in the form of large amounts of data-level parallelism (DLP). As a result, much of the recent effort for media applications has been on architectures that target large amounts of DLP in various ways. There are several architectures that target large grain DLP in streams and long vectors; e.g., VIRAM's [43] and CODE's [44] vector architecture, Imagine's streaming architecture [1]. Such architectures exploit the regularity of large grain DLP to perform computation more efficiently but do not target TLP. SCALE's vector-threading architecture [45], attempts to cater to irregular computation by combining threads with vectors. Most quantitative evaluations of these architectures, however, are largely based on small kernels, which are easily computed in real-time on today's general-purpose processors (e.g., speech codecs such as adpcm, color conversion such as rgb2cmyk, filters such as fir, and autocorrelation). Our analysis of *complete* applications (as opposed to kernels) shows that such complex applications exhibit many forms of parallelism including many granularities of DLP.

Chapter 7 further discusses the primary differences of our work from existing solutions for parallelism and media applications.

1.2 Contributions

This dissertation makes three high level contributions:

1. Analysis of parallelism in complex media applications. We characterize the parallelism and performance in complex media applications and make the case that such applications exhibit multiple types of parallelism including ILP, TLP, and multiple forms of DLP such as sub-word SIMD, short vectors, streams, and vectors and streams of sub-word SIMD. The last form exploits two dimensional parallelism; i.e., a vector of (short) vectors.

2. A comparison of alternatives for determining the most energy efficient way to exploit TLP in media workloads on general-purpose processors.

3. A complete architecture, called ALP, that effectively supports *a*ll *l*evels of *p*arallelism described above in an energy-efficient way, using an *evolutionary programming model and hardware*. This is the key contribution of this dissertation.

Overall, in this work, we exploit parallelism at multiple levels. We exploit ILP using standard out-of-order superscalar processors; we investigate the most energy-efficient ways to support TLP (Chapter 6) and propose new mechanisms for exploiting DLP for multimedia applications (Chapters 3 through Chapter 5).

1.2.1 Analysis of Parallelism

Our first contribution is a characterization of parallelism in complex media applications. In this study, we use a five complex media applications: MPEG-2 encode, MPEG-2 decode, speech recognition (Sphinx3), face recognition (CSU), and ray tracing (Tachyon). We believe such applications will be commonplace on future general-purpose systems such as desktops, laptops, and handheld computers. We also made these applications available to the public as a benchmark suite named ALPBench [50].

Chapter 5 presents a detailed analysis of these applications. The important characteristics found are listed below.

1. *Multiple forms of parallelism.* These applications usually exhibit multiple forms of parallelism; viz., DLP, TLP, and ILP. All these applications exhibit coarse-grain TLP. As reported by others, we also find DLP in the kernels of these applications. However, we find many large portions lack DLP and only exhibit ILP and TLP (e.g., Huffman coding in MPEG encode and ray-tracing).

2. *Small-grain DLP.* Many applications have small-grain DLP (short vectors) due to the use of packed (SIMD) data and new intelligent algorithms used to reduce computation. Packed data reduces the number of elements (words) to be processed. New intelligent algorithms introduce data-dependent control, again reducing the granularity of DLP. For example, older MPEG encoders performed a full motion search comparing each macroblock from a reference frame to all macroblocks within a surrounding region in a previous frame, exposing a large amount of DLP. Recent advanced algorithms significantly reduce the number of macroblock comparisons by predicting the "best" macroblocks to compare. This prediction is based on the results of prior searches, introducing data-dependent control between macroblock computations and reducing the granularity of DLP.

3. *Dense representation and regular access patterns within vectors.* Our applications use dense data structures such as arrays, which are traversed sequentially or with constant strides in most cases.

4. *Reductions.* DLP computations are often followed by reductions, which are less amenable to conventional DLP techniques (e.g., vectorization) but become significant with the reduced granularity of DLP. For example, when processing blocks (macroblocks) in MPEG using 16B packed words, reductions occur every 8 (16) words of DLP computation.

5. *High memory to computation ratio.* DLP loops are often short with little computation per memory access.

6. *Multiple forms of DLP.* Our applications exhibit DLP in the form of SIMD, short vectors, long streams, and vectors/streams of SIMD. The last form can be viewed as two dimensional DLP (i.e., as an array of arrays).

The above properties heavily influenced the design of ALP, especially the novel DLP support provided in ALP.

1.2.2 Comparison of Alternatives for Energy-Efficient Support for TLP

Our second contribution is a comparison of TLP mechanisms to determine the most energy-efficient form of exploiting TLP on general-purpose processors. To find the most energy-efficient way of exploiting TLP, we perform a comparison between chip multi-processing (CMP) and simultaneous multithreading (SMT). Since multi-media applications require meeting a given performance target, we compare the energy efficiency of CMP and SMT at a given performance target (i.e., at equal performance). Since the performance target can change depending on the user environment parameters like the load on the system, we do this comparison for a spectrum of performance points. Further, a given performance target can be obtained by changing both the frequency and the processor architecture. Therefore, we do the above comparison for each performance point using multiple processor architectures and frequencies/voltages.

From this study, we find that, at equal performance, CMP is more energy efficient than SMT, especially when supporting four or more threads. We also find that the best SMT and the best CMP

configuration for a given performance target have different architecture and frequency/voltage. From our analysis, we find that the relative energy efficiency depends on a subtle interplay between various factors such as capacitance, voltage, IPC, frequency, and the level of clock gating, as well as workload features.

Although CMP shows a clear energy advantage for four-thread (and higher) workloads, it comes at the cost of increased silicon area. We therefore investigate a hybrid solution where a CMP is built out of SMT cores, and find it to be an effective compromise. We use this architecture as the basis for providing ILP and TLP in ALP.

1.2.3 ALP

Based on our findings from the above two studies, we propose ALP, an architecture that can cater to multiple forms of parallelism found in our applications. ALP differs from the state-of-the art in one or both of the following important ways. First, as described above, we focus on more complex applications. Second, due to our focus on general-purpose processors (GPPs), we impose the following constraints/assumptions on our work:

(i) GPPs already exploit some DLP through sub-word SIMD instructions such as MMX/SSE.

(ii) GPPs already exploit ILP and TLP respectively through superscalar cores and through chip-multiprocessing (CMP) and simultaneous multithreading (SMT).

(iii) Radical changes in the hardware and programming model are not acceptable for well established GPPs.

Motivated by our application study and the above constraints/assumptions, we propose a complete architecture called ALP.

The properties of our applications summarized above in Section 1.2.1 motivate supporting multiple forms of parallelism including ILP, TLP, and various forms of DLP. For DLP, properties like short vector lengths, frequent reductions, and significant non-DLP parts imply that conventional solutions such as dedicated multi-lane vector units may be over-kill. For example, the Tarantula vector unit has the same area as its scalar core [19], but will likely be under-utilized for our

applications. We therefore take an alternative approach in this work that aims to improve upon SIMD, but without the addition of a dedicated vector unit.

Overall Architecture

To effectively support *a*ll *l*evels of *p*arallelism exhibited by our applications in the context of current GPP trends, ALP is based on a GPP with CMP, SMT, and SIMD. The most novel part of ALP is a technique called *SIMD vectors (SVectors)* and *SIMD streams (SStreams)* that support larger amounts of DLP than possible with SIMD. SVectors/SStreams use an evolutionary programming model and can be implemented with modest additional hardware support that is tightly integrated within a modern superscalar pipeline.

The programming model for SVectors lies between SIMD and conventional vectors. SVectors exploit the regular data access patterns that are the hallmark of DLP by providing support for conventional vector *memory* instructions. They differ from conventional vectors in that computation on vector data is performed by existing SIMD instructions. Each architectural SVector register (SVR) is associated with an internal hardware register that indicates the "current" element of the SVector. A SIMD instruction specifying an SVector register as an operand accesses and auto-increments the current element of that register. Thus, a loop containing a SIMD instruction accessing SVector register V0 marches through V0, much like a vector instruction. SStreams are similar to SVectors except that they may have unbounded length.

Our choice of supporting vector/stream *data* but not vector/stream *computation* exploits a significant part of the benefits of vectors/streams for our applications, but without need for dedicated vector/stream compute units. Specifically, ALP largely exploits existing storage and data paths in conventional superscalar systems and does not need any new special-purpose structures. ALP reconfigures part of the L1 data cache to provide a vector register file when needed (e.g., using reconfigurable cache techniques [2, 58]). Data paths between this reconfigured register file and SIMD units already exist, since they are needed to forward data from cache loads into the computation units. These attributes are important given our target is GPPs that have traditionally resisted application-specific special-purpose support.

Summary of Findings

Our evaluations show that our design decisions in ALP are effective. Relative to a single-thread superscalar without SIMD, for our application suite, ALP achieves aggregate speedups from 5X to 56X, energy reduction from 1.7X to 17.2X, and energy-delay product (EDP) reduction of 8.4X to 970.4X. These results include benefits from a 4-way CMP, 2-way SMT, SIMD, and SVectors/SStreams. Our detailed results show significant benefits from each of these mechanisms. Specifically, for applications with DLP, adding SVector/SStream support to a system with all the other enhancements in ALP achieves speedups of 1.1X to 3.4X, energy savings of 1.1X to 1.5X, and an EDP improvement of 1.1X to 5.1X (harmonic mean of 1.7X). These benefits are particularly significant given that the system compared already supports ILP, SIMD, and TLP; SVectors/SStreams require a relatively small amount of hardware; and the evaluations consider complete applications.

More broadly, our results show that conventional architectures augmented with evolutionary mechanisms can provide high performance and energy savings for complex media applications without resorting to radically different architectures and programming paradigms (e.g., Imagine, SCALE).

1.3 Thesis Organization

ALP is motivated by the properties of complex media applications. Therefore, we first describe a characterization of several such applications in the next chapter, Chapter 2. This chapter presents a high level description of each application and a characterization of parallelism and performance of these applications.

The primary contribution of this dissertation, ALP, is described next in Chapters 3 - 5. Specifically, Chapter 3 describes the programming model and Chapter 4 describes the implementation of ALP. Chapter 5 presents an evaluation of ALP.

Chapter 6 describes the study of energy-efficient support for TLP which influenced ALP; we discuss this work after presenting ALP since the details of this work are not necessary to appreciate ALP. Finally, Chapter 7 describes the related work and Chapter 8 concludes and provides future research directions.

Chapter 2

Complex Media Applications and Parallelism

This chapter studies five complex media applications used in this dissertation and characterizes the nature of parallelism found in these applications. Section 2.1 provides an overview of our key results - it describes the overall properties of our media applications and our specific observations about the nature of parallelism in these complex applications. Section 2.2 describes the five complex media applications we study. Section 2.3 describes our methodology for characterizing and quantifying various forms of parallelism and finally Section 2.4 characterizes the parallelism and performance of these applications.

2.1 Overview of Results

In this study, we use five complex media applications: MPEG-2 encode, MPEG-2 decode, speech recognition (Sphinx3), face recognition (CSU), and ray tracing (Tachyon). We believe such applications will be commonplace on future general-purpose systems such as desktops, laptops, and handheld computers.

Since complex media applications differ considerably from small media kernels and other desktop applications in many important respects, they need to be studied on their own. Specifically, most media applications have a lot of parallelism that can be exploited for energy-efficient high-performance designs. Table 2.1 describes the three main types of parallelism that are available in media (and other) applications.

Our attempts at characterizing and exploiting various forms of parallelism in these applications

Instruction Level Parallelism (ILP):	If the operations in a given set are both data and control independent of one another, such a set exhibits ILP. This is because each operation can be encoded as an individual instruction and can be executed independent of one another. ILP can be expressed in the programming model as with VLIW (Very Long Instruction Word) or EPIC (Explicitly Parallel Instruction Computing) instruction sets; alternatively ILP can be dynamically discovered by out-of-order superscalar processors.
Data Level Parallelism (DLP):	If the operations in a given set are (i) both data and control independent of one another and (ii) have the same op-code, then that set of operations exhibits DLP. Practical systems add one additional restriction to the above two; viz., data regularity. Data regularity requires that the locations (addresses/register numbers) of data items used by all operations in the set can be expressed using a simple formula so that it can be succinctly encoded in an instruction (for instance, data items can be sequential, or can have a constant stride between two consecutive items). Although it is possible to rearrange data to meet the data-regularity requirement (e.g., using scatter/gather operations in a vector architecture), such rearrangements usually have high costs. Some systems relax the requirement of control independence of operations by resorting to masking. However, masking reduces the effectiveness of DLP since masking causes unnecessary data to be loaded/processed. Further, some practical systems require a large number of operations within each set to be effective. Therefore, to obtain significant benefits using DLP, the number of operations should be relatively large, masking should be minimized, and the number of data rearrangements to satisfy data-regularity should be small. Sets with a large number of operations are said to exhibit *coarse-grain DLP* (as opposed to *fine- or small-grain DLP* with only a few operations per set). DLP is the most restrictive form of parallelism. DLP can be degenerated into both TLP and ILP. Vector instruction sets [15, 3, 19] and sub-word SIMD instructions like Intel MMX/SSE [34] are examples of DLP instruction sets.
Thread Level Parallelism (TLP):	TLP extends the idea of independence among multiple operations to *multiple sets of operations*, where operations within each set can be dependent or independent. Each such set of operations is called a thread. Trivially, two independent sets each with one operation can create two parallel threads. However, to be useful in practice, each thread has to contain a large number of dynamic operations and must be mostly independent of the other threads; i.e., there should be only a few, if any, points of interaction between threads. In other words, only a few operations should have direct inter-set dependencies. The operations that are *directly* dependent on operations from any other set require synchronization (e.g., locks, barriers) to avoid race conditions. Note that there is no restriction on the individual operations in any set; i.e., the individual operations within a set can be either data/control dependent or independent of the other operations in the same set. Further, DLP and/or ILP can be exploited within each set, if operations within a set satisfy the requirements described above for DLP and/or ILP. POSIX threads (Pthreads) are one popular thread standard on general-purpose processors.

Table 2.1: Three types of parallelism.

revealed several important properties about them:

1. *Multiple forms of parallelism.* These applications usually exhibit multiple forms of parallelism like thread/data/instruction-level parallelism (TLP/DLP/ILP). All these applications exhibit coarse-grain TLP. As reported by others, we also find DLP in the kernels of these applications. However, we find many large portions lack DLP and only exhibit ILP and TLP (e.g., Huffman

coding in MPEG encode and ray-tracing).

2. *Small-grain DLP.* Many applications have fine-grain DLP due to the use of packed (SIMD) data types and new intelligent algorithms used to reduce computation. Packed data reduces the number of elements (words) to be processed. New intelligent algorithms introduce data-dependent control, again reducing the granularity of DLP. For example, older MPEG encoders performed a full motion search comparing each macroblock from a reference frame to all macroblocks within a surrounding region in a previous frame, exposing a large amount of DLP. Recent advanced algorithms significantly reduce the number of macroblock comparisons by predicting the "best" macroblocks to compare. This prediction is based on the results of prior searches, introducing data-dependent control between macroblock computations and reducing the granularity of DLP.

3. *Dense representation and regular access patterns within vectors.* Our applications use dense data structures such as arrays, which are traversed sequentially or with constant strides in most cases.

4. *Reductions.* DLP computations are often followed by reductions, which are less amenable to conventional DLP techniques (e.g., vectorization) but become significant with the reduced granularity of DLP. For example, when processing blocks (macroblocks) in MPEG using 16B packed words, reductions occur every 8 (16) words of DLP computation.

5. *High memory to computation ratio.* DLP loops are often short with little computation per memory access.

6. *Multiple forms of DLP.* Our applications exhibit DLP in the form of SIMD, short vectors (small-grain DLP), long streams (coarse-grain DLP), and vectors/streams of SIMD (two dimensional DLP).

The next section describes the complex media applications used in this study.

2.2 Media Applications

This section describes our applications and the enhancements we made to them. To extract parallelism, we threaded the applications and inserted DLP instructions in the frequently used functions. For threading, we used POSIX threads (Pthreads). For most cases, straightforward parallelization was sufficient for the relatively small systems we consider (e.g., static scheduling of threads). For SIMD, we used Intel SSE2 and and a more aggressive simulated version called ALP SIMD, which is modeled after SSE2. SIMD hand-coding is prevalent practice for these applications and the maximum number of static assembly instructions inserted for any given application is about 400 (for MPGenc). In some cases, we made a few algorithmic modifications to the original applications to improve performance.

The following descriptions provide a summary of algorithmic modifications (where applicable), major data structures, application phases, thread support, and sub-word SIMD support, and SVector/SStream support The SVector/SStream support for each application is described along with the other modifications to the corresponding application only to improve readability. The details and evaluation of SVectors are described in Chapters 3 - 5. Recall from Section 1.2.3 that SVectors/SStreams use vector/stream loads to bring in data to SVector registers (SVRs) and use ALP SIMD instructions for computation.

2.2.1 MPEG 2 Encoder (MPGenc)

We use the MSSG MPEG-2 encoder [56]. MPGenc converts video frames into a compressed bitstream. A video encoder is an essential component in VCD/DVD/HDTV recording, video editing, and video conferencing applications. Many recent video encoders like MPEG-4/H.264 use similar algorithms.

A video sequence consists of a sequence of input pictures. Input images are in the YUV format; i.e., one luminance (Y) and two chrominance (U,V) components. Each encoded frame is characterized as an I, P, or B frame. I frames are temporal references for P and B frames and are only spatially compressed. On the other hand, P frames are predicted based on I frames, and B frames are predicted based on neighboring I and P frames.

Modifications: We made two algorithmic modifications to the original MSSG code: (1) we use

an intelligent three-step motion search algorithm [42] instead of the original full-search algorithm and (2) we use a fast integer discrete cosine transform (DCT) butterfly algorithm based on the Chen-Wang algorithm [79] instead of the original floating point matrix-based DCT.

Data Structures: Each frame consists of 16x16 pixel macroblocks. Each macroblock consists of four 8x8 luminance locks and two 8x8 chrominance blocks, one for U and one for V.

Phases: The phases in MPEG-2 include motion estimation (ME), quantization, discrete cosine transform (DCT), variable length coding (VLC), inverse quantization, and inverse DCT (IDCT).

The first frame is always encoded as an I-frame. For an I-frame, the compression starts with DCT. DCT transforms blocks from the spatial domain to the frequency domain. Following DCT is quantization that operates on a given 8x8 block, a quantization matrix, and a quantization value. The operations are performed on each pixel of the block independent of each other. After quantization, VLC is used to compress the bit stream. VLC uses both Huffman and run-length coding. This completes the compression.

For predictive (P and B) frames, the compression starts with motion estimation. In motion estimation, for each macroblock of the frame being currently encoded, we search for a "best-matching" macroblock within a search window in a previously encoded frame. The distance or "match" between two macroblocks is computed by calculating the sum of the differences between the pixels of the blocks. The original "full-search" algorithm performs this comparison for all macroblocks in the search window. Instead, we use a three-step search algorithm which breaks a macroblock search into three steps: (i) search at the center of the search window, (ii) search around the edges of the search window, and (iii) search around the center of the search window. A subsequent step is taken only if the previous step does not reveal a suitable match. Motion estimation is the longest (most compute intensive) phase for P and B frames. The rest of the compression for P and B frames is the same as that for an I-frame.

For processing subsequent frames, it is necessary to decode the encoded frame. For this purpose, inverse quantization and inverse DCT are applied to the encoded frame. These inverse operations have the same properties as their forward counterparts.

We removed the rate control logic from this application. The original implementation performs rate control after each macroblock is encoded, which imposes a serial bottleneck. For the threaded version, rate control at the end of a frame encoding would be more efficient but we did not implement this.

Threads: We create a given number of threads at the start of a frame and join them at the end of that frame. Within a frame, each thread encodes an independent set of contiguous macroblock rows in parallel. Each thread takes such a set through all the listed phases and writes the encoded stream to a private buffer. Thread 0 sequentially writes the private buffers to the output.

SIMD: Integer SIMD instructions are added to all the phases except VLC. 1-byte (char) sub-words are used in macroblocks; 2-byte (short) words are used to maintain running sums. The main SIMD computation in motion estimation is a calculation of sum of absolute difference (SAD) between two 128b packed words of two macroblocks. PSAD (packed SAD) instructions in SSE2 are used for this purpose. The result of the SAD is accumulated in a register. For half pixel motion estimation, it is necessary to find the average of two 128b records. This is achieved using PAVG (packed average) SSE2 instructions.

We obtained optimized SSE2 code for DCT and IDCT from [30] and [31], respectively. Sub-word sizes of 16b (short) are used for DCT/IDCT and multiply accumulate instructions are used for common multiply accumulate combinations in this code. Quantization is a truncation operation. We use packed minimum and packed maximum for performing the truncation [34].

Before DCT and after IDCT, the encoder performs a block subtraction and a block addition where a block of frequency deltas are added or subtracted from a block. We use packed saturated addition and subtraction for these operations.

SVectors: SVectors are used in all the phases where SIMD is used (i.e., in all the phases except VLC). Computation uses the same SIMD compute instructions used by the SIMD implementation described above. However, recall that SVectors use vector loads instead of SIMD loads. SVectors replace SIMD loads except in parts of DCT and IDCT. DCT and IDCT each consists of two

sub parts: row DCT/IDCT and column DCT/IDCT. Because of the non-sequential memory access patterns, column DCT/IDCT uses SIMD loads instead of SVector loads. Constant coefficient tables used in row DCT, row IDCT, and quantization can be loaded into SVRs at the start of a phase and can be held throughout that phase. In motion estimation, the macroblock being encoded is loaded once and held in an SVR while it is compared against many reference blocks.

2.2.2 MPEG-2 Decoder (MPGdec)

We use the MSSG MPEG-2 decoder [56]. MPGdec decompresses a compressed MPEG-2 bit-stream. Video decoders are used in VCD/DVD/HDTV playback, video editing, and video conferencing. Many recent video decoders, like MPEG-4/H.264, use similar algorithms.

Data Structures: Same as for MPGenc.

Phases: Major phases for MPGdec include variable length decoding (VLD), inverse quantization, IDCT, and motion compensation (MC),

The decoder applies the inverse operations performed by the encoder. First, it performs variable-length Huffman decoding. Second, it inverse quantizes the resulting data. Third, the frequency-domain data is transformed with IDCT to obtain spatial-domain data. Finally, the resulting blocks are motion-compensated to produce the original pictures.

Threads: At first glance, this application seems to be serial since frames have to be recovered by decoding blocks one by one from the encoded bit stream. However, a closer look at the application shows that the only limitation of exploiting TLP for this application is the serial reads from the bit stream and the rest can be readily parallelized.

In our implementation, thread 0 identifies the slices (contiguous rows of blocks) in the input encoded bit-stream. When a given number of slices is identified, those slices are assigned to a new thread for decoding. Due to this staggered nature of creating threads, different threads may start (and finish) at different times, thereby reducing the thread-level scalability of the application.

Each thread takes each block in a slice through all the phases listed above and then writes

each decoded block into a non-overlapping region of the output image buffer.

SIMD: Integer SIMD instructions are added to IDCT and motion compensation. IDCT uses the same SIMD code used in MPGenc. Motion compensation contains sub-functions like add-block (adding the reference block and error (frequency deltas)) and saturate. These operations are performed using packed addition with saturate on 16b words.

SVectors: Use of SVectors in motion compensation and IDCT is similar to that for MPGenc; i.e., SVectors are used in all the phases where SIMD is used. Computation uses the same SIMD compute instructions used by the SIMD implementation described above. SVectors replace SIMD loads except in a part of IDCT. IDCT consists of two sub-parts: row IDCT and column IDCT. Because of the random memory access patterns, column IDCT uses SIMD loads instead of SVector loads. Constant coefficient tables used in row IDCT can be loaded into SVRs at the start of the IDCT phase and can be held throughout that phase.

2.2.3 Ray Tracing (RayTrace)

We use the Tachyon ray-tracer [72]. A ray-tracer renders a scene using a scene description. Ray tracers are used to render scenes in games, 3-D modeling/visualization, virtual reality applications, etc.

The ray tracer takes in a scene description as input and outputs the corresponding scene. A scene description normally contains the location and viewing direction of the camera, the locations, shapes, and types of different objects in the scene, and the locations of the light sources.

Data Structures: The constructed scene is a grid of pixels. The pixels are colored based on the light sources and objects in the scene. The objects are maintained in a linked list. The color of each pixel is determined independently.

Phases: This application does not have distinct phases at a high level. At start, based on the camera location and the viewing direction specified, the viewing plane is created to represent the

grid of pixels to be projected from the scene to the resulting picture. To project the correct color for each pixel, a ray is shot from the camera through the viewing plane into the scene. The ray is then checked against the list of objects to find out the first object that the ray intersects. After that, the light sources are checked to see if any of the light rays reach that intersection. If so, the color to be reflected is calculated based on the color of the object and the color of the light source. The resulting color is assigned to the pixel at where the camera ray and the viewing plane intersect. Moreover, since objects can be reflective or transparent, the ray may not stop at the first object it intersects. Instead, the ray can be reflected or refracted to other directions until another object is intersected. In that case, the color of the corresponding pixel is determined by repeatedly reflecting/refracting the ray at each surface.

Threads: Each thread is given N independent rays to trace, where N is the total number of pixels in the viewing plane divided by the number of threads in the system. Rays are assigned to different threads using a round-robin algorithm to achieve better load balancing.

SIMD: No DLP support is added since the various computations done on each ray can be quite different from neighboring rays. This is because neighboring rays can intersect different objects leading to different computations (operations) with each ray. Further, there is no DLP within each ray since each ray performs control intensive operations.

2.2.4 Speech Recognition (SpeechRec)

We use the CMU SPHINX3.3 speech recognizer [61]. A speech recognizer converts speech into text. Speech recognizers are used with communication, authentication, and word processing software and are expected to become a primary component of the human-computer interface in the future.

Data Structures: The major data structures used include:

(1) 39-element feature vectors extracted from an input speech sample.

(2) Multiple lexical search trees built from the language model provided. Each tree node is a 3-state hidden Markov model (HMM) and describes a phoneme (sound element).

(3) Each senone (a set of acoustically similar HMM states) is modeled by a Gaussian model. Each Gaussian model contains two arrays of 39-element vectors (mean and variance) and one array of coefficients.

(4) A dictionary (hash table) of known words.

Phases: The application has three major phases: feature extraction, Gaussian scoring, and searching the language model/dictionary.

First, the feature extraction phase creates 39-element feature vectors from the speech sample. The Gaussian scoring phase then matches these feature vectors against the phonemes in a database. It evaluates each feature vector based on the Gaussian distribution in the acoustic model (Gaussian model) given by the user. In a regular workload, there are usually 6000+ Gaussian models. The goal of the evaluation is to find the best score among all the Gaussian models and to normalize other scores with the best one found. As this scoring is based on a probability distribution model, multiple candidates of phonemes are kept so that multiple words can be matched. The final phase is the search phase, which matches the candidate phonemes against the most probable sequence of words from the language model and the given dictionary. Similar to the scoring phase, multiple candidates of words (hypotheses) are kept so that the most probable sequence of words can be chosen.

The algorithm can be summarized as follows:

We make the root node of each lexical search tree active at start. The following steps are repeated until speech is identified. Step (i) is the feature extraction phase, (ii) is in the Gaussian scoring phase, and steps (iii) and (iv) are in the search phase.

(i) The feature extraction phase creates a feature vector from the speech sample.

(ii) A feature vector is compared against Gaussian models of most likely senones and a similarity score is computed for each senone.

(iii) For each active node in each lexical search tree, the best HMM score for it is calculated. Then the overall best HMM score among all nodes is calculated (call this S_{ob}).

(iv) All nodes with HMM scores below $S_{ob} - threshold$, where $threshold$ is a given threshold, are deactivated and the children of the still active nodes are also activated. If the node is a leaf

node with high enough score, the word is recognized and the dictionary is looked up to find the spelling.

For reporting results, the startup phase, where some data structures are initialized, is ignored since it is done only once for the entire session and it can be optimized by loading checkpointed data [55].

Threads: We parallelized both the Gaussian scoring and the search phase. We did not parallelize the feature extraction phase since it takes only about 2% of the execution time (with a single thread). A thread barrier is used for synchronization after each phase. To create threads for the Gaussian scoring phase, we divide the Gaussian models among threads to calculate senone scores.

In the search phase (steps (iii) and (iv) above), active nodes are divided evenly among threads. We use fine grain locking to synchronize updates to the existing hypotheses in step (iv). This locking makes this phase less scalable than the Gaussian scoring phase.

SIMD: We added floating point SIMD support to the Gaussian scoring phase. The SIMD computation in this phase consists of a short loop which performs multiplication and addition to calculate the score. Floating point SIMD instructions calculate the score of each feature vector and Gaussian model. Packed floating point multiplication and addition (MULPS, ADDPS) are used for this operation and 4B sub-words (floats) are used.

SVectors: As with SIMD, SVectors are also used with Gaussian scoring. Feature vectors are loaded only once and held in SVRs for many iterations of phase (i). Gaussian model vectors are loaded into SVRs. The same SIMD compute instructions described above are used with SVectors to calculate the Gaussian score between the feature vector and the Gaussian model vectors.

The original Sphinx code performs a pre-computation to reduce the number of Gaussian models that have to evaluated in the Gaussian scoring phase. This pre-computation reduces the total execution time of both non-SIMD and SIMD cases. Since SVectors can handle DLP more efficiently, for the SVector version, it is more beneficial to forego this pre-computation because this pre-computation is control intensive. Therefore, we do not use this pre-computation with the

SVector version but evaluate all the Gaussian models in the Gaussian scoring phase.

2.2.5 Face Recognition (FaceRec)

We use the CSU face recognizer [7]. Face recognizers recognize images of faces by matching a given input image with images in a given database. Face recognition is used in applications designed for authentication, security, and screening. Similar algorithms can be used in other image recognition applications that perform image searches and data-mining.

This application uses a large database (called subspace) that consists of multiple images. The objective of phase recognition is to find the image in subspace that matches best with a given input image. A match is determined by taking the "distance" or difference between two images.

Modifications: The CSU software tries to find the pairwise distances among all images in the database since the objective of CSU software is to find the effectiveness of distance finding algorithm. We modified the application so that a separate input image is compared with each image in the subspace to emulate a typical face recognition scenario (e.g., a face of a subject is searched in a database).

Data Structures: Each image is a single column vector with thousands of rows. The subspace is a huge matrix where each image is a column of the matrix.

Phases: This application is first trained with a collection of images in order to distinguish faces of different persons. Moreover, there are multiple images that belong to the same person so that the recognizer is able to match face images against different expressions and lighting conditions. Then, the training data is written to a file so that it can be used in the recognition phase. Since training is done offline we consider only the recognition phase for reporting results.

At the start of the recognition phase, the training data and the image database are loaded. The image database creates the subspace matrix.

The rest of the recognition phase has two sub-phases:

(i) Projection: When an input image is given, it is normalized and projected into the large subspace matrix that contains the other images. The normalization involves subtracting the subspace's mean from the input. Then that normalized image is "projected" on to the subspace by taking the cross product between the normalized image and the subspace.

(ii) Distance computation: Computes the difference between each image in the subspace and the given image by finding the similarity (distance).

Threads: In the projection sub-phase, each thread is given a set of columns from the subspace to multiply. In the distance-computation sub-phase, each thread is responsible for computing distances for a subset of images in the database.

SIMD: Floating point double precision (8B) SIMD instructions are used for matrix computations and for distance finding. Both these sub-phases contain short loops that perform multiplication and addition/subtraction. The SIMD instructions used include packed subtraction (SUBPD), packed multiplication (MULPD), and packed addition (ADDPD).

SVectors: SVectors/SStreams are used within both sub-phases where SIMD instructions are used (i.e., in matrix computations and distance finding). SStreams are used for loading columns of matrices for processing in both phases. The same SIMD compute instructions described above are used with SVectors.

2.3 Methodology

For this study, we primarily obtain results from a CMP simulator called AlpSim. AlpSim allows us to study the parallelism and scalability of systems under different conditions. To augment these results, where practically feasible, we also present data obtained on a real Pentium 4 system.

AlpSim is an execution-driven cycle-level simulator derived from RSIM [28], and models wrong path instructions and contention at all resources. AlpSim simulates all code in C libraries but only emulates operating system calls.

With AlpSim, we model a CMP system to study the parallelism in our applications. Each CMP

Parameter	Value PER PARTITION	# of Partitions
Phy Int Reg File (32b)	64 regs, 5R/4W	2
Phy FP/SIMD Reg File (128b)	32 regs, 4R/4W	2
Int Issue Queue		2
-# of Entries	24	
-# of R/W Ports	3R/4W	
-# of Tag R/W Ports	6R/3W	
-Max Issue Width	3	
FP/SIMD Issue Queue		2
-# of Entries	24	
-# of R/W Ports	3R/4W	
-# of Tag R/W Ports	5R/3W	
-Max Issue Width	3	
Load/Store Queue		2
-# of Entries	16	
-# of R/W Ports	2R/2W	
-Max Issue Width	2	
Branch Predictor (gselect)	2KB	2
Integer ALUs (32b)	2	2
FP SIMD Units (128b)	2	2
Int SIMD Units (128b)	2	2
Reorder Buffer	32 ent, 2R/2W	4
-Retire Width	2	
Rename Width	4 per thread	2
Max. Fetch/Decode Width	6 (max 4 per thread)	

Parameter	Value PER BANK	# Banks
L1 I-Cache	8K, 4 Way, 32B line, 1 Port	2
L1 D-Cache (Writethrough)	8K, 2 Way, 32B line, 1 Port	4
L2 Cache (Writeback, unified)	256K, 4 Way, 64B line, 1 Port	4

Bandwidth and Contentionless Latencies @ 4 GHz	
Parameter	**Value (cycles @ 4 GHz)**
ALU/Int SIMD Latency	8 (Div-32b), 2 (Mult-32b), 1 (Other)
FP/FP SIMD Latency	12 (Div), 4 (Other)
L1 I-Cache Hit Latency	2
L1 D-Cache/SVR Hit Latency	3
L2 Cache Latency	18 (hit), 256 (miss)
Memory Bandwidth	16 GB/s

Table 2.2: Base architecture parameters for AlpSim. Note that several parameter values are *per partition or bank.*

processor is an out-of-order superscalar processor and has separate private L1 data and instruction caches. All cores in the CMP share a unified L2 cache. Each thread is run on a separate CMP processor. The simulation parameters used are given in Table 2.2. Following the modern trend of

general purpose processor architectures, almost all processor resources are partitioned and caches are banked.

The ALP SIMD programming model used with AlpSim roughly emulates Intel's MMX/SSE2 with multiple 8-, 16-, 32-, or 64-bit sub-words within a 128-bit word. Most common opcodes are supported; e.g., packed addition, subtraction, multiplication, absolute difference, average, horizontal reduction, logical, and pack/unpack operations. SIMD operations use the FP register file and FP units.

AlpSim uses SPARC binaries for non-SIMD code. Pthreads-based C code is translated into binary using the Sun cc 4.2 compiler with options -xO4 -xunroll=4 -xarch=v8plusa. DLP code resides in a separate assembly file, organized as blocks of instructions and simulated using hooks placed in the binary. When such a hook is reached while simulating, the simulator switches to the proper block of SIMD instructions in the assembly file.

To complement the results obtained using AlpSim, we obtained data using a 3.06 GHz Pentium 4 system with SSE2 running the Linux 2.4 kernel (referred to later as **P4Sys**). The processor front-side bus operates at 533 MHz (quad-pumped) and the system has 2GB of PC2100 DDR memory. The applications for P4Sys were compiled using the Intel icc compiler with maximum optimization level O3 and options -march=pentium4 -mcpu=pentium4 (for Pentium 4). We aligned data arrays at 16B boundaries for best performance as suggested in [34]. On P4Sys, we used the Intel VTune performance analyzer and used the performance counter (sampling) mode to obtain results without any binary instrumentation. Only single-thread data were obtained using the P4Sys.

The following inputs were used for each application. For MPGenc and MPGdec, DVD resolution (704x480) input streams were used. For RayTrace, a 512x512 resolution picture (a scene of a room with 20 objects) is used. For SpeechRec, a dictionary/vocabulary of 130 words was used with the input speech sample containing the words "Erase T M A Z X two thousand five hundred and fifty four". For FaceRec, a database of 173 images (resolution 130x150) was used with an input image of the same resolution.

2.4 Results

This section provides quantitative results about the parallelism found in our applications. Primarily, we present results using AlpSim and ALP SIMD. To augment those results, we present results obtained on a Pentium 4 processor based system for ILP and SIMD (SSE2).

We categorize our results into several sections. First, we characterize each type of parallelism. Second, we analyze the effects of interaction between two types of parallelism (e.g., DLP and TLP). Since we observe that all types of parallelism investigated here are sensitive to the memory system parameters, in Section 2.4.5, we present data showing the size of working sets utilized by our applications, the effect of increasing memory latencies (i.e., frequency scaling), and the effect of supporting more threads on the memory bandwidth. Finally, in Section 2.4.6, we give the application-level real-time performance of our applications on the Pentium 4 system with SSE2.

Although the results we show are sensitive to the size of inputs, the overall parallelism should improve or remain the same with larger inputs for all applications.

2.4.1 TLP

Figures 2.1(a) and (b) show the speedup achieved with multiple threads on AlpSim with a 1 cycle ideal memory system and a non-ideal memory system, respectively. The threads do not use SIMD instructions. The ideal memory system results are obtained with perfect 1 cycle L1 caches to study the TLP scalability independent of the memory system parameters, especially those of the L2 cache. These applications can be executed on systems with very different L2 configurations, from shared L2 caches to private L2 caches. Similarly, the size and the associativity of L2 caches vary widely in commercial systems. When we use high memory latencies, the scalability becomes sensitive to the particular L2 configuration as described in Section 2.4.5. Therefore, Figure 2.1(a) shows inherent TLP in applications, independent of L2 parameters. However, since it is useful to see how these applications will behave on a practical machine, Figure 2.1(b) shows TLP scalability for the system described in Section 2.3 except for one change; these results use a 16-way, 16MB L2 cache with 64 GB/s memory bandwidth to support up to 16 threads.

As shown in Figure 2.1, MPGenc, MPGdec, FaceRec, and RayTrace scale well up to 16 threads with both ideal and realistic memory parameters since the threads are independent and hence

do not require extensive synchronization. For MPGenc, there are two limitations to obtaining ideal scalability characteristics: (i) serialization present at the end for writing private buffers, and (ii) imperfect load balancing due to different threads performing different amount of work. The scalability of MPGdec can be further improved by addressing the current limitations to its scalability, namely, (i) staggered thread creation, and (ii) load imbalance. With larger inputs (e.g., HDTV), the former has less effect (HDTV input improved the speedup of 16 threads by 14%-15% for both ideal and realistic memory parameters). The latter may be improved by dynamic slice assignment [26].

The thread scalability of SpeechRec is somewhat limited. Its scalability can be slightly improved by threading the feature extraction phase as well. However, the scalability of coarse-grained threads in SpeechRec is mainly limited by the fine grain synchronization (locking) used in the search phase [47]. However, we found that larger dictionaries increase the thread scalability. Note that multi-threaded versions of SpeechRec achieve slightly better speedups with realistic memory parameters. In that case, the execution time of the single thread version is dominated by the time stalled for memory. The multi-threaded version can reduce that stall time considerably due to memory parallelism offered by multiple threads. However, with ideal memory parameters, the multi-threaded version cannot reduce the memory access time any further. Therefore, synchronization has a larger negative effect on the multi-threaded versions with ideal memory parameters.

2.4.2 DLP

	MPGenc	**MPGdec**	**SpeechRec**	**FaceRec**
Sub-word size	1B,2B	1B,2B	4B (float)	8B (double)
% SIMD instr.	24	24	17	66
Computation/Memory	1.70	1.84	1.43	1.00
DLP Granularity (in 128b words)	1,8,16	1,4,8	10	9750 (stream)
% Reductions	36	10	28	50

Table 2.3: DLP characteristics of applications.

Table 2.3 summarizes the DLP characteristics of our applications, except for RayTrace, which has only TLP and ILP.

The first row summarizes the sub-word sizes used by each application. MPGenc and MPGdec

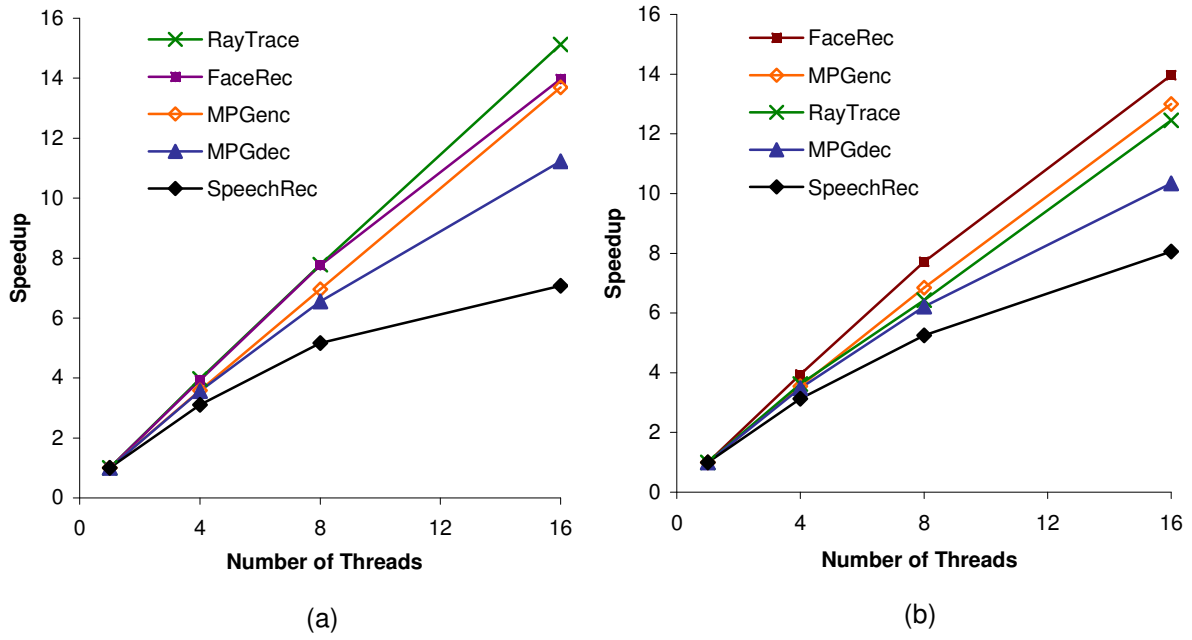

Figure 2.1: Scalability of TLP without SIMD instructions (a) with a perfect 1-cycle memory system, and (b) with realistic memory parameters.

use smaller integer sub-words whereas SpeechRec and FaceRec use larger FP sub-words.

The *% SIMD instr.* row gives the percentage of total dynamic instructions that are SIMD in the version of the code with ALP SIMD. The relatively small percentage shows the importance of supporting ILP and TLP for these complex applications.

The *Computation/Memory* row of Table 2.3 gives the ratio of SIMD computation to memory operations (and instructions) in the version of the code with ALP SIMD. All our applications show low computation to memory ratios. This indicates that efficient support for memory operations could lead to significant performance and energy improvements for these applications.

The *DLP Granularity* row shows the DLP granularity in 128b SIMD words (i.e., the iteration count of SIMD loops). All our applications except FaceRec exhibit small-grain DLP (i.e., small vector lengths), while FaceRec exhibits very large grain DLP (i.e., streams). It may (or may not) be possible to increase the vector lengths with much effort using substantially different algorithms. However, such changes were not obvious to us and, we believe, are beyond the scope of this work. (Some obvious changes substantially increase the amount of work to be done and significantly reduce performance; e.g., using a full-search instead of an intelligent search in MPGenc, using

matrix multiplication based DCT/IDCT in MPGenc/MPGdec instead of an optimized Chen-Wang butterfly algorithm [79].)

The *% Reductions* row shows the dynamic number of DLP operations that are part of a reduction, as a percentage of the total dynamic DLP compute operations in the version of the code with ALP SIMD. This relatively high percentage combined with the small DLP granularity underscores the importance of supporting efficient reductions for these applications. Some implementations are not efficient in supporting reductions when the DLP granularity is small (e.g., multi-laned vector units). This property indicates that such implementations may be a poor match for these applications.

Figure 2.2 gives the speedups achieved with SSE2 (on P4Sys) and ALP SIMD (on AlpSim) over the original non-SIMD single-threaded application. The results with SSE2 show the speedups achievable on existing general-purpose processors. The results with ALP SIMD indicate the speedups possible with a more general form of SIMD on a simulated 4 GHz processor. Overall, our applications (except RayTrace) achieve significant speedups with ALP SIMD and modest to significant speedups with SSE2.

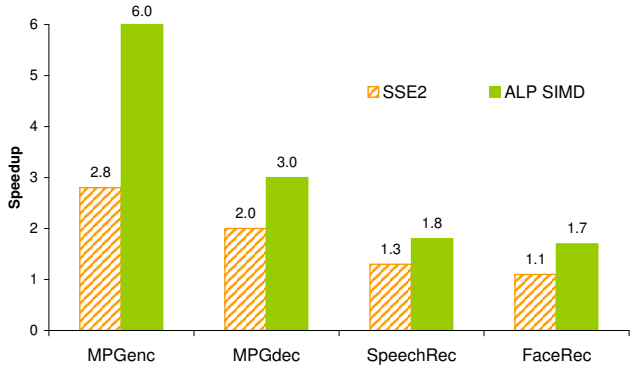

Figure 2.2: Speedup with SSE2 and ALP SIMD.

For all applications, the speedups with ALP SIMD are higher than the speedups with SSE2 due to several reasons. First, the latency of most SIMD instructions on AlpSim is 1 cycle whereas all SSE2 instructions have multi-cycle latencies. Further, the 128b SSE2 is implemented as two 64b operations on Pentium processors essentially halving the throughput. Specifically, FaceRec fails to achieve any significant speedup with SSE2 due to the lack of true 128b units because FaceRec uses double precision 64b operations. Second, the simulated processor has 4 SIMD units and a different

pipeline and hardware resources. Third, AlpSim supports SIMD opcodes that are more advanced than those in SSE2. For instance, horizontal sub-word reductions are available in AlpSim but not in SSE2 (although they are available with SSE3[1]).

SIMD Speedups of Individual Phases with SSE2

All phases and sub-phases of an application do not see the same level of performance improvement with SIMD support. Therefore, it is important to understand which parts of an application are more amenable to SIMD and which parts are not. Although some phases can show very high speedups, according to Amdahl's law, the overall speedup of the application is limited by phases with small or no speedups.

Table 2.4 shows the percentage of execution time and the SSE2 speedup of each phase in each application on P4Sys. The total SSE2 speedup for each application is also given. These results show which phases of the applications are more amenable to SIMD. The data for RayTrace is omitted since it does not have DLP instructions or multiple major phases. Note that the sampling error for small phases is more significant than for larger phases; small phases (i.e., phases with non-SSE2 execution time less than 2% or aggregates of such small phases) where the speedup cannot be measured reliably are marked as N/A in Table 2.4. It should also be noted that the phases in an application cannot be completely separated from the adjacent phases of the same application when run on an out-of-order processor where instructions from multiple phases can overlap. Further, the effects produced by one phase (e.g., branch histories, cache data) can affect other phases. As a result, Table 2.4 shows small slowdowns for some phases.

MPGenc and MPGdec see good overall speedups with SSE2. All phases of MPGenc and all but the VLD phase in MPGdec achieve speedups with SSE2. IDCT of MPGdec and DCT/IDCT phases of MPGenc achieve excellent speedups due to the use of optimized SSE2 code for these phases.

The motion estimation phase of MPGenc achieves very good speedups with SSE2 due to the use of byte operations and the elimination of data-dependent branches using PSAD (packed sum of absolute difference) instructions. Similarly, quantization achieves excellent speedups due to the

[1]We did not use SSE3 for our applications since it is fairly new and most existing systems do not support it.

	no-SSE2	with SSE2	
	% ExTime	% ExTime	Speedup
MPGenc			
Motion Estimation	64.3	66.3	2.69
DCT/IDCT	9.6	6.3	4.24
Form predictions	5.6	11.9	1.3
Quant/IQuant	18.8	9.2	5.69
VLC	1.3	3.8	N/A
Other	0.4	2.5	N/A
Total	100.0	100.0	**2.78**
MPGdec			
IDCT	36.6	13.8	5.38
Motion Compensation			
- Saturate	8.3	10.4	1.61
- Add Block	9.3	5.7	3.3
- Form predictions	21.5	20.4	2.14
- Clear Block	2.8	3.9	1.44
VLD	20.3	43.3	0.95
Other	1.3	2.4	N/A
Total	100	100	**2.03**
SpeechRec			
Feature Extraction	1.6	2.1	0.97
Gaussian Scoring			
- Vector Quantization	35.4	26.5	1.73
- Short-list Generation	10.5	13.7	0.99
- Gaussian Eval	35.7	34.3	1.34
- Others	7.4	10.8	N/A
Search	7.7	10.5	0.94
Other	1.7	2.1	N/A
Total	100.0	100.0	**1.29**
FaceRec			
Subspace projection	88.0	88.8	1.11
- Transform	87.3	87.4	1.12
Distance calculation	7.4	5.7	1.47
Other	5.3	6.9	N/A
Total	100.0	100.0	**1.12**

Table 2.4: Percentage execution time and SSE2 speedup for major phases of each application (except for RayTrace) on P4Sys. Small phases (i.e., phases with non-SSE2 execution time less than 2% or aggregates of such small phases) where the speedup cannot be measured reliably are marked as N/A.

elimination of branches by using PMAX and PMIN instructions to truncate.

In MPGdec, sub-phases of motion compensation phase like saturate, add block, and form prediction achieve good speedups with SSE2 since they contain straightforward DLP loops with (saturated) additions and subtractions. But VLD which is a significant portion of the total application does not see any speedup resulting in a lower overall speedup than MPGenc.

SpeechRec achieves reasonable speedup with SSE2 (due to its use of 32b single precision floats,

the peak possible speedup is roughly 2X on Gaussian scoring). As expected, SIMD instructions lead to significant speedups in the two most dominant sub-phases of the Gaussian scoring phase. However, the overall speedup is limited by phases without DLP.

FaceRec fails to achieve significant speedups with SSE2 due to FaceRec's use of double precision 64b operations as described above. However, it succeeds in recording a small overall speedup due to the elimination of overhead instructions.

2.4.3 ILP

	AlpSim		P4 Sys (Pentium 4)	
App	**Base**	**SIMD**	**Base**	**SIMD**
MPGenc	1.20	1.23 [4.24]	1.45 (1.87)	0.70 (1.03)
MPGdec	1.38	1.17 [3.31]	1.26 (1.73)	0.73 (1.14)
RayTrace	1.33	N/A	0.48 (0.73)	N/A
SpeechRec	0.35	0.39 [0.67]	0.38 (0.57)	0.34 (0.45)
FaceRec	0.32	0.30 [0.48]	0.51 (0.61)	0.43 (0.47)

Table 2.5: Instructions per cycle achieved on AlpSim and P4Sys for single-thread applications. For the ALP SIMD case, the number of sub-word operations retired per cycle is also given within square brackets. For P4Sys, x86 micro-instructions per cycle is given in parenthesis.

Table 2.5 gives instructions-per-cycle (operations per cycle) achieved on AlpSim and x86 instructions per cycle (micro-instructions per cycle) achieved on P4Sys. The IPC values for AlpSim and P4Sys cannot be directly compared because they do not use the same instruction set, processor or memory parameters. Rather, P4Sys data represents a real 3.06 GHz system and AlpSim data represents a simulated 4 GHz system.

FaceRec and SpeechRec fail to achieve large ILP due to their working sets not fitting in caches (Section 2.4.5). Other applications show reasonable ILP on AlpSim. However, the SIMD versions of MPGenc and MPGdec and the base version of RayTrace achieve lower IPC on P4Sys than on AlpSim. Specifically for the SIMD versions of MPGenc and MPGdec, as described in Section 2.4.2, the longer SSE2 latencies and the lack of true 128-bit functional units lower the IPC on P4Sys. For RayTrace, P4Sys sees lower IPC than AlpSim due to three main reasons. First, longer FP latencies and longer repetition intervals (lower throughput) of the FP units of P4 reduce performance. Second, the smaller 8K L1 cache of P4 further reduces the performance since RayTrace achieves lower hit rates with an 8K L1 data cache compared to a 32K L1 of AlpSim (see Figure 2.3 in Sec-

tion 2.4.5). Third, the much deeper pipeline of P4 reduces performance since branch misprediction rate is somewhat high (4%) and 10% of all instructions are branches. However, P4Sys sees higher IPC for some applications due its lower frequency, L2 hardware prefetcher, and differences in the ISA (e.g., x86 ISA uses far more register spill instructions than SPARC ISA used with AlpSim).

Although SpeechRec and FaceRec have low ILP due to the high memory latencies, we observed that their IPC values become higher (1.5 and 1.3 respectively, with SIMD as reported later in Table 5.3) when the memory latency is reduced to 42 cycles to simulate a 500 MHz processor or a 500 MHz frequency setting on a processor with frequency scaling. Further, we noticed that, reducing the fetch/retire width from 4 to 2 reduces the IPC of SIMD versions of MPGenc, MPGdec, and RayTrace by 35%, 33%, and 38% respectively. This again underscores the importance of ILP for these applications.

2.4.4 Interactions Between TLP, DLP, and ILP

Interaction Between TLP and DLP

Most DLP sections in our applications occur within the parallel portions of the code; i.e., we have only few DLP sections in the sequential parts of the code. When we exploit DLP and TLP together, DLP causes the parallel sections to execute faster. Therefore, according to Amdahl's law, the serial sections become more dominant due to the use of DLP. We quantify the above effect as follows. Let $NThrdSpeedup_{NoDLP}$ be

$$\frac{Execution\ time\ of\ non{-}DLP\ N{-}thread\ application}{Execution\ time\ of\ non{-}DLP\ 1{-}thread\ application}$$

and $NThrdSpeedup_{DLP}$ be

$$\frac{Execution\ time\ of\ DLP\ N{-}thread\ application}{Execution\ time\ of\ DLP\ 1{-}thread\ application}$$

Table 2.6 gives the ratio of $NThrdSpeedup_{NoDLP}/NThrdSpeedup_{DLP}$ for each application for $N = 1$, 4, 8, and 16 threads (obtained on AlpSim using 1 cycle perfect memory latencies). A ratio higher than 1.0 shows a reduction of TLP scalability due to the use of DLP (SIMD) instructions. Specifically, we see larger ratios for MPGdec and SpeechRec because they have relatively large

App	1T	4T	8T	16T
MPGenc	1.00	0.99	1.01	1.09
MPGdec	1.00	1.12	1.27	1.55
SpeechRec	1.00	1.09	1.23	1.49
FaceRec	1.00	0.97	0.96	0.96

Table 2.6: Ratio $NThrdSpeedup_{NoDLP}$ / $NThrdSpeedup_{DLP}$ for 1, 4, 8, and 16 threads for all applications with DLP.

serial sections that are devoid of DLP and significant portions of DLP within the thread-parallel sections. Further, note that the above ratio increases with the number of threads for MPGdec and SpeechRec limiting their TLP scalability in the presence of DLP. This can have a significant impact on the TLP scalability of emerging multi-core/multi-threaded commercial processors that already support SIMD instructions. The above ratio stays close to 1 for MPGenc and FaceRec since they do not have large serial sections.

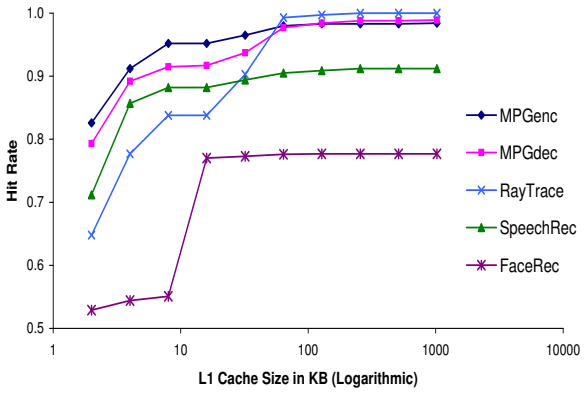

Figure 2.3: L1 cache hit rates. The hit rates are with SIMD for all applications except for RayTrace, for which it is without SIMD.

Interaction Between DLP and ILP

Exploiting DLP in a given piece of code should theoretically reduce the amount of ILP present in that piece of code since a DLP instruction can replace multiple independent non-DLP instructions. Table 2.5 shows this decrease for all applications except MPGenc and SpeechRec with AlpSim.

The exceptions occur for multiple reasons. First, on real processors, the DLP is usually exploited using separate resources. For instance, on Pentium processors and on AlpSim, the SIMD instructions are executed in the FP pipeline. Therefore, exploiting SIMD in integer code allows the

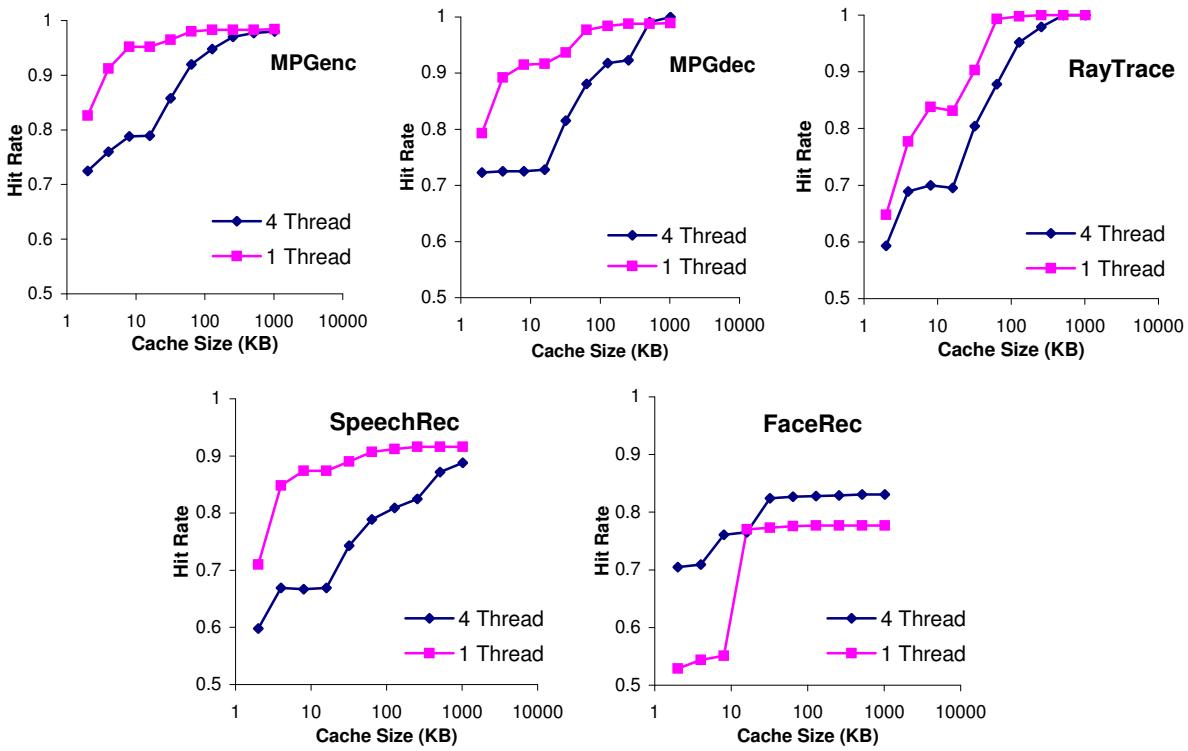

Figure 2.4: L2 cache hit rates. The rates are with SIMD for all applications except RayTrace, for which it is without SIMD.

otherwise idle FP pipeline to be utilized as well. This could increase the amount of ILP exploited in a cycle. Second, the SIMD instructions can reduce contention to the critical processor resources (e.g., load/store queue entries, cache ports) by combining several non-DLP instructions into one DLP instruction. The reduced contention potentially allows more ILP to be exploited from a piece of code. Third, SIMD code reduces the number of conditional branch instructions. This happens mainly because SIMD reduces the number of loop iterations and the branches associated with them. Further, some SIMD instructions like packed absolute difference, packed sum of absolute difference (PSAD), or packed maximum and minimum can reduce data dependent branches used in non-SIMD code.

Interaction Between TLP and ILP

The interaction between TLP and ILP is well known. On practical CMP systems, TLP can affect the amount of ILP available to a particular thread. The behavior of the caches could change due

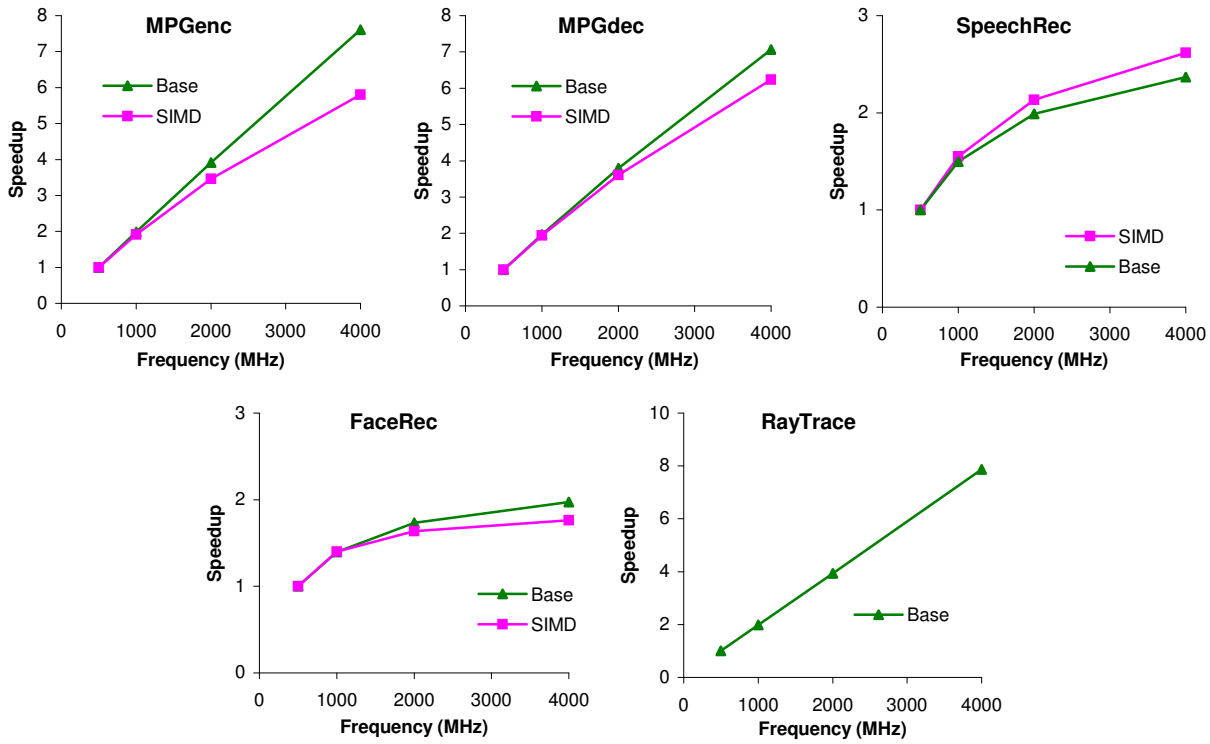

Figure 2.5: Frequency Scalability. The SIMD data are with ALP SIMD for all applications.

Application	MPGenc	MPGdec	RayTrace	SpeechRec	FaceRec
IPC Reduction	6.1%	8.8%	10.6%	7.8%	2.7%

Table 2.7: Percentage reduction of per thread IPC in the 16-thread CMP with respect to the IPC of 1-thread processor . The IPC does not include synchronization instructions or stall cycles caused by them.

to the presence of multiple threads and could affect the ILP of each thread. As discussed later (Figure 2.4 and Section 2.4.5), it is possible to have both positive and negative cache effects due to multiple threads. For instance, false sharing in caches could reduce the ILP whereas sharing of read-only data (or instructions) could increase the ILP of each thread. Table 2.7 gives the percentage reduction of per thread IPC in a 16-thread CMP with respect to the IPC of a single-thread processor. The IPC does not include the effect of synchronization instructions. We see that multiple threads cause a small to modest reduction in per-thread IPC due to the effects discussed above. This reduction is smallest in FaceRec due to some constructive data sharing among threads in the L2 cache (see Section 2.4.5).

2.4.5 Sensitivity to Memory Parameters

As expected, the parallelism of our applications is sensitive to the memory parameters. To understand how different memory parameters impact the parallelism, this section describes the cache hit ratios/working sets zof our applications, how our applications scale with increasing frequency (memory latencies), and the memory bandwidth requirement as the number of threads is increased.

Working Sets

Figure 2.3 gives the L1 data cache hit ratios obtained using AlpSim with SIMD for different L1 cache sizes (2K to 1024K). Note that the cache hit rates are sensitive to the sizes of inputs used. Using the concepts described in [80], all applications, except FaceRec, have first-level working sets about 8KB since the first knee of all hit-rate curves occurs around 8KB. FaceRec has the first-level working set of 16KB. Both RayTrace and MPGdec can further benefit significantly from a cache size up to 64KB. MPGenc sees a slight benefit if the cache size is further increased to 64KB. FaceRec and SpeechRec, however, do not benefit much from increasing the cache size after 8KB and 16KB, respectively (up to 1MB).

Figure 2.4 shows the shared L2 cache hit rates for different cache sizes (2K - 1024K) for both single-thread and 4-thread versions of each application with SIMD. The L1 caches were disabled for this experiment to study the effect of data sharing between multiple threads in L2. If the threads share a significant portion of data and the single thread version achieves a given hit rate with x KB, the 4-thread version should be able to achieve the same or a better hit rate with $4x$ KB of cache. Based on the data of Figure 2.4, we can see that only threads in FaceRec share a significant portion of data since the 4-thread version achieves better hit rates than the single-thread version for a given cache size. This is because the threads in FaceRec share parts of the large subspace matrix (database). Since we partition data among threads for all applications, the threads in the other applications do not exhibit constructive sharing. Even FaceRec, which exhibits some data sharing, does not share all the data in L2 since a significant portion of its memory accesses still have to go to memory.

To summarize, first we see that three of our applications have very good cache hit rates whereas two have low hit rates. Low cache hit rates reduce ILP when memory latencies are high. We also see

that increasing TLP demands larger caches to accommodate the working sets of our applications since threads do not share much data.

Sensitivity to Memory Latency or Processor Frequency

Figure 2.5 shows the speedup achieved by the base (non-SIMD) and SIMD versions of each single-thread application on AlpSim when the processor frequency is scaled from 4 GHz to 500 MHz. For RayTrace, the results for non-SIMD single-thread version are shown. The time to access the memory (and the memory bus) is decreased linearly from 240 cycles (at 4 GHz) to 30 cycles (at 500 MHz) (i.e., the L2 miss latency is the memory/bus access time plus 16 cycles for all frequencies). The other parameters given in Table 2.2 are not changed. Specifically, the parameters used at 4 GHz are identical to those given in Table 2.2. Speedups reported for non-SIMD (SIMD) are with respect to the non-SIMD (SIMD) single thread version of the application running at the lowest frequency (500 MHz). Such frequency scaling data is important since many systems, especially mobile systems running these media applications, run at lower frequencies. Further, many such systems employ dynamic frequency scaling to run at lower frequencies than the maximum frequency supported by the processor to reduce power and energy consumption. The following describes how each application scales when the frequency is *increased* from the lowest (500 MHz) to the highest (4 GHz).

Figure 2.5 shows that the base cases of RayTrace, MPGenc and MPGdec scale well with increasing frequency since most of their working sets fit in the caches. The base cases of FaceRec and SpeechRec show poor scalability after 1 or 2 GHz. This is mainly due to their larger working sets not fitting in caches (Figure 2.3).

Two factors affect the relative scalability between SIMD and non-SIMD versions of the same application. On the one hand, the SIMD version has a lower computation to memory ratio than the base case and hence is more sensitive to longer memory latencies. This is because the SIMD case reduces loop overhead and address calculation overhead instructions, which are compute instructions. This effect causes the SIMD versions of MPGenc, MPGdec, and FaceRec to show lower scalability. The effect is more prominent in MPGenc due to its use of small sub-words; in that case, SIMD can reduce the loop iteration and overhead significantly. On the other hand, the SIMD

version exposes more memory level parallelism to the out-of-order core – since the SIMD loops use fewer instructions per loop, the instruction window can contain a larger number of load instructions than possible in the non-SIMD case. This effect gives the SIMD version better scalability when the application has a significant L2 miss rate. Specifically, SpeechRec benefits from this effect. Although, the SIMD version of FaceRec should benefit from the same effect since it has high L2 miss rates, the increase in memory level parallelism for the SIMD version of FaceRec is low due to its use of double precision operations. That is, the number of additional loop iterations we can fit in the instruction window is not large as that of SpeechRec.

To summarize, three out of five of our applications scale well with frequency. We see that higher memory latencies affect applications with and without SIMD differently. SIMD versions of all our applications except SpeechRec show somewhat poorer scalability with increasing frequency than their non-SIMD counterparts.

Memory Bandwidth

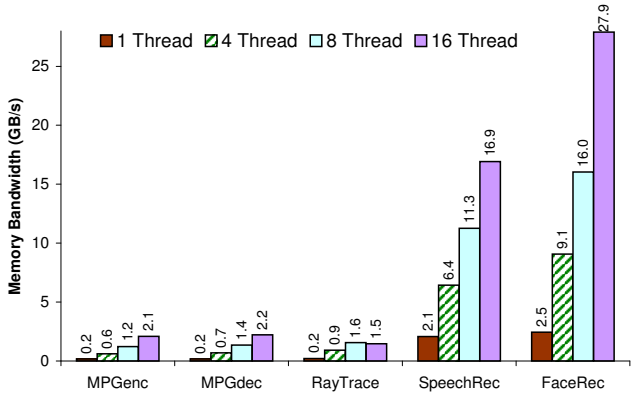

Figure 2.6: Memory bandwidth (in GB/s) at 4 GHz without SIMD.

Figure 2.6 shows how memory bandwidth demand increases for each application (non-SIMD) with the number of threads at 4 GHz. The results were obtained on AlpSim without ALP SIMD using the same parameters used for obtaining Figure 2.1(b). MPGenc, MPGdec, and RayTrace have relatively low bandwidth requirements since they have smaller working sets. However, FaceRec and SpeechRec demand much larger memory bandwidth since their working sets do not fit in the L2 cache. The increase in bandwidth generally follows the TLP speedup of applications, except for

Application	Performance
MPGenc	21.8 fps (704x480 DVD resolution)
MPGdec	166.3 fps (704x480 DVD resolution)
RayTrace	0.75 fps (512x512 resolution)
SpeechRec	9.0 words/sec.
FaceRec	152.1 130x150 images/sec.

Table 2.8: Application-level real-time performance obtained by single threaded versions of our applications on a 3.06 GHz Pentium-4 processor with SSE2.

RayTrace where the 16-thread version requires less memory bandwidth than the 8-thread version. This is because the *L1* hit ratio of the 16-thread case is far better than that for the 8-thread case due to the working set getting divided into 16 in the 16-thread case. For all applications that have DLP, SIMD versions will demand more bandwidth since they execute faster. These results show that the bandwidth of MPGenc, MPGdec, and RayTrace can be fulfilled by existing memory systems (assuming a maximum of 8.5 GB/s memory bandwidth on current personal computers with DDR2 memory). However, for SpeechRec and FaceRec, CMP systems that support 8 or more threads will have to support a higher memory bandwidth than supported today on many general-purpose systems [35].

2.4.6 Application-Level Real-time Performance

Table 2.8 shows the application-level real-time performance results for each application on P4Sys (Section 2.3). The results are for single-threaded applications with SSE2 (except for RayTrace). The approximate performance for systems with a higher number of threads and lower frequencies can be derived using the thread/frequency scaling results presented earlier. MPGdec already achieves the required real-time performance on current systems. The performance of RayTrace is far from real-time. Although MPGenc comes close to the required real-time performance of 30 frames per second, larger inputs (e.g., HDTV) will demand much higher performance. Similarly, although SpeechRec and FaceRec achieve reasonable performance with the given small input sets, much larger inputs anticipated in the future (e.g., much larger vocabularies and dictionaries for SpeechRec, higher resolution image/face recognition used with personal/database search, and much larger image databases) will demand much higher processing capabilities.

2.5 Summary

This chapter describes and analyzes our complex media applications with particular emphasis on the parallelism in them. Through our analysis, we find that these applications contain multiple forms of parallelism; thread-, data-, and instruction-level parallelism. As for TLP, we find that all these applications have coarse grain threads and most applications show very good thread scalability. As for DLP, we find that these applications use various granularities of DLP (e.g., sub-word SIMD, short vectors, long streams) and sub-word SIMD is capable of achieving significant speedups for applications with DLP. Further, we find that these applications contain frequent reductions and exhibit high memory to computation ratio. We also find that these applications use dense data structures that are usually traversed sequentially.

Overall, we observe that all three levels of parallelism should be exploited effectively to achieve the best performance. We also find that the effective exploitation of DLP can reduce the effectiveness TLP. Through our study investigating the effects of memory system on these applications, we find that individual applications have different working sets, bandwidth requirements, and memory latency tolerance.

Chapter 3

The ALP Programming Model

Based on our findings from the previous chapter, we propose ALP, an architecture that can cater to multiple forms of parallelism found in our applications. To support ILP, TLP, and sub-word SIMD, ALP is based on an out-of-order superscalar processor with CMP, SMT, and SIMD. The most novel part of ALP is a technique called *SIMD vectors (SVectors)* and *SIMD streams (SStreams)* that support larger granularities of DLP than possible with SIMD. This chapter describes the programming model of ALP including the programming model for SVectors and SStreams.

ALP supports conventional threads for TLP. ILP is not exposed to the programmer since ALP uses an out-of-order superscalar core as in current general-purpose processors. The SIMD programming model roughly emulates Intel's MMX/SSE2 with multiple 8-, 16-, 32-, or 64-bit sub-words within a 128-bit word and with eight logical SIMD registers. Most common opcodes are supported; e.g., packed addition, subtraction, multiplication, absolute difference, average, horizontal reduction, logical, and pack/unpack operations. SIMD operations use the FP register file and FP units. We next describe the novel SVectors and SStreams programming model.

3.1 SIMD Vectors (SVectors)

SVectors are built on three key enhancements to SIMD support:

1. **SIMD Vector Registers (SVRs)** hold a sequence of *records*, where each record itself is a sequence of (possibly strided) *packed-words* and each packed-word may contain multiple (contiguous) sub-words (see Figure 3.1). Unlike a conventional vector, the records of an SVR can be individually accessed with an index, called the *Current Record Pointer (CRP)*. An

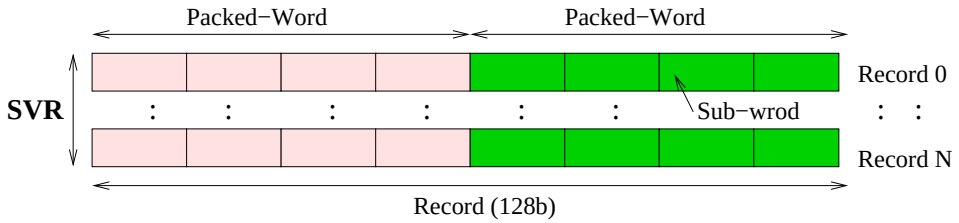

Figure 3.1: An SVR consists of records, a record consists of packed-words, and a packed-word consists of sub-words.

SVR is allocated on demand and can have a variable length up to a given maximum.

2. **SVector allocate (VALLOC) and SVector load (VLD) instructions.** VALLOC and VLD allocate an SVR. VLD additionally loads a (possibly strided) sequence of packed-words into the SVR from memory. A slight variation of VALLOC, called VALLOCst, allocates an SVR whose records are flushed to memory as they are written. All of these instructions reset the CRP of the SVR. These are the only special SVector instructions in the ALP ISA.

3. **SIMD instructions capable of accessing SVRs.** All computation on SVRs is performed using SIMD instructions which can directly access an individual record of an SVR. Such an instruction specifies an SVR as an operand, which implicitly accesses the record of the SVR pointed to by its CRP *and also increments the CRP*. Thus, a dynamic sequence of SIMD instructions specifying a given SVR will access successive records of the SVR.

The ALP ISA supports 8 *logical* SVRs, V0 to V7, with a record size of 128 bits and sub-word sizes of 8, 16, 32, and 64 bits. Associated with each logical SVR is an internal SVR descriptor register. This descriptor register stores pertinent information about the SVR, including the CRP. A VLD, VALLOC, or VALLOCst instruction must be used to explicitly allocate an SVR before any SIMD instruction can access it. These vector instructions specify the length of the SVector. The VLD and VALLOCst instructions also specify the organization of the SVector in memory, including the base memory address of the SVector, the stride between two packed-words in the SVector, and the number of packed-words per 128b record. All of this information is stored in the associated SVR descriptor.

```
(1) VLD addr:stride:length ==> V0
(2) VLD addr:stride:length ==> V1
(3) VALLOCst addr:stride:length ==> V2
    REPEAT for all records in SVector
(4)   simd_add V0, V1 ==> simd_reg0
(5)   simd_mul simd_reg0, simd_reg1 ==> simd_reg2
(6)   simd_sub simd_reg2, #16 ==> V2
```

Figure 3.2: SVector code for $V2 = k * (V0 + V1) - 16$

As an example, Figures 3.2 gives SVector code for the computation $V2 = k*(V0+V1)-16$, where V0, V1, and V2 are SVRs, and k is a constant, stored in simd_reg1. The first two instructions load the two source SVectors from memory. The next instruction, VALLOCst, allocates a new SVR for writing V2. All of these instructions implicitly reset the CRP of V0, V1, and V2. Next, a loop called a *SIMD loop* is used to traverse the records of the SVectors. This loop contains SIMD instructions that directly read/write the SVRs, accessing the record currently pointed by the corresponding CRP and incrementing this CRP. Each occurrence of instruction (4), therefore, reads from the next record of V0 and V1 and each occurrence of instruction (6) writes to the next record of V2 (and also stores that record to memory, since V2 is allocated with a VALLOCst).

ALP also provides an instruction, ClearCRP, to reset the CRP of the specified SVR, and an instruction, MoveRec, to read a specific SVR record into a SIMD register. ClearCRP is used if an SVR needs to be read again after it has already been traversed once with SIMD instructions; e.g., to reuse a quantization table in MPEG. MoveRec is used to provide random read access into records; e.g., MoveRec V0, #4 ⇒ simd_reg4 moves record V0[CRP+4] to simd_reg4.

ALP requires that an SVector/SStream be traversed sequentially. If a record needs to be skipped, it must be read and discarded to increment the CRP. Alternatively, it is possible to provide an instruction to increment the CRP by a given number of records; however, our applications do not exhibit such a requirement.

ALP does not support scatter/gather operations on SVectors since our applications do not exhibit memory access patterns that would benefit from such operations.

ALP imposes three implementation-driven ISA restrictions. The first two arise because ALP implements SVRs by reconfiguring part of the L1 data cache to allocate SVR space on demand (Section 4). First, the maximum SVector length allowed in ALP is related to the L1 size and the

number of SVRs supported. An SVector length of 32 records (512B) sufficed for our applications and fit comfortably in our L1 cache (except for FaceRec that uses SStreams).

Second, because SVRs are allocated on demand, clearly, an SVR cannot be read unless it is explicitly allocated using a VLD, VALLOC or VALLOCst. Third, the out-of-order ALP implementation uses conventional renaming to avoid stalls due to WAR and WAW hazards even for SVectors. A problem with this is that the renaming occurs at the granularity of the full SVR, at the vector load and allocate instructions. However, the SIMD writes occur at the granularity of individual records. We therefore impose a programming restriction that requires a VALLOC instruction before a vector record is overwritten by a SIMD instruction. This instruction indicates to the hardware that a new renamed copy of the vector must be allocated for subsequent SIMD writes of this logical vector. In our applications, writing to SVRs is infrequent, so this restriction has little impact.

3.2 SIMD Stream (SStreams)

An SStream is essentially a long SVector that (i) exceeds the maximum length of an SVR, and (ii) must be accessed strictly sequentially. Conventional vector processors would require strip-mining for loops containing such long vectors. Instead, we support two special stream load (SLD) and stream allocate (SALLOC) instructions. These instructions are similar to VLD and VALLOCst respectively in that they both allocate the specified SVR. Transparent to the programmer, however, the underlying hardware allocates an SVR size that is smaller than the stream size, and manages it like a FIFO queue – when the program reads a record from the head, it is discarded and a new record is automatically appended to the tail (Section 4.3). An exception incurred by the load of such a record is handled at the instruction that will consume the record (Section 4.3).

Like SVectors, computation on SStreams occurs with SIMD instructions. For instance, to perform the computation in Figure 3.2 on two streams and produce a resulting stream, we need only change VLD to SLD and VALLOCst to SALLOC. Unlike SVectors, ClearCRP and MoveRec are not supported for streams since streams are accessed sequentially (to simplify hardware management of the SVR space).

Note that it is possible to replace SStreams with SVectors by strip mining long loops. However,

SVectors may not be as effective as SStreams for hiding memory latency (Section 5.2). This is because SVector loads have to be explicitly scheduled for maximal latency hiding whereas hardware automatically schedules record loads well in advance for SStreams.

3.3 SVectors/SStreams vs. SIMD

This section qualitatively describes the performance and energy benefits of SVectors and SStreams over a pure SIMD ISA (e.g., MMX or SSE2). Differences from conventional vectors are discussed in Section 7. Not surprisingly, some of these benefits are similar to those from conventional vectors [3, 14, 25]. Section 7 elaborates on the differences between SVectors and conventional vectors.

3.3.1 Performance Benefits

1. *Reduced load/store and overhead instructions:* SVectors/SStreams reduce instruction count in two ways. First, VLD/SLD and VALLOCst/SALLOC reduce instruction count by replacing multiple loads and stores with one instruction and eliminating the corresponding address arithmetic overhead instructions.

   Second, SVRs reduce loads/stores and associated overhead instructions due to increased register locality. The SVRs increase the register space available that can be directly accessed by compute instructions, reducing loads/stores due to register spills. For instance, MPGenc and MPGdec repeatedly use quantization/coefficient tables – each table in DCT/IDCT has 32 records. A pure SIMD system repeatedly spills and loads entries of these tables from and into the small number of SIMD registers. With SVRs, these tables are loaded only once and then directly accessed by the SIMD compute instructions for as long as they are needed.

   A simple expansion of the SIMD register file is not as effective because (i) it would need a larger instruction width to encode the larger register space and (ii) a single large register file is energy inefficient and this price would be paid for *all* SIMD instructions. SVectors mitigate problem (i) by exploiting the regular nature of vector data to access them through an implicit index (the CRP) – this requires encoding only the SVR in the instruction since the CRP is implicit. They mitigate problem (ii) by splitting the register space into the smaller (and

so more energy efficient) SIMD register file and the larger (less energy efficient) SVR. The more efficient SIMD file stores temporary values from intermediate computation, making the most effective use of that space. The less efficient, larger SVR file primarily stores the large amounts of SVector data directly loaded from memory, reducing pollution of the SIMD file.

2. *Increased exposed parallelism and decreased contention from reduced instruction count:* The reduction of memory/overhead instruction count in frequently used loops allows more loop iterations to fit in the processor's instruction window. This allows hardware to extract more parallelism and hide latencies of compute instructions. In short loops, instruction count reduction can be as high as 50% allowing twice as many compute instructions in flight (e.g., in our face recognition application). Further, the reduction of memory/overhead instructions also reduces contention to critical resources like register files and issue ports.

3. *Load latency tolerance:* SVectors/SStreams allow more aggressive use of pipelined loads, without limits of register pressure. On an SVector/SStream load, the constituent loads of individual records are pipelined with each other and with the iterations of the corresponding SIMD computation loop. Further, SVector/SStream loads that can be predicted in advance can also be hoisted well before the corresponding SIMD computation loops.

   The above benefit from SVector/SStream loads is similar to that from using (hardware or software) prefetching, but is more effective than the latter for the following reasons. First, SVector/SStream loads eliminate many load instructions; prefetching does not have this benefit and software prefetching requires additional instructions for the prefetches and address calculation. Second, SVector/SStream loads only bring the data required, whereas prefetchers bring entire cache lines, potentially polluting the cache. Third, prefetching needs to be carefully scheduled; otherwise, it can evict useful data. Prefetches into separate buffers have been recommended to avoid this problem, but such buffers must be exposed to the cache coherence protocol. Finally, for short vectors such as 16x16 or 8x8 blocks seen in MPEG, there may not be enough time for a hardware prefetcher to effectively learn the pattern [26]. Section 5.2 discusses experimental results that show that ALP's benefits exceed well beyond those for prefetching.

4. *L1 cache space and bandwidth savings due to packed data:* SVRs contain packed and aligned data. In contrast, a cache line loaded to L1 using a SIMD load may contain useless data (e.g., when the stride between packed words is not one).

5. *Eliminating record alignment in L1:* In many cases, 16-byte SIMD records are not aligned at 16-byte boundaries in memory. SIMD instruction sets like MMX/SSE provide special unaligned load instructions to load SIMD data starting at unaligned addresses. Such instructions have higher latency than normal loads. This latency has to be paid each time data is loaded from L1. With SVectors/SStreams, the extra latency for alignment has to be paid only at the time of loading an SVector from L2. Accesses to SVRs do not require any alignment. Further, since this alignment is done in L2 as part of a vector load that is performed in parallel with the computation, it is often possible to remove this additional latency from the critical path.

3.3.2 Energy Benefits

SVectors/SStreams provide energy benefits over pure SIMD in the following ways. First, the reduction of load/store and overhead instructions provide direct energy savings. Second, the performance benefits above reduce execution time without a commensurate increase in power, thereby reducing energy. Third, an SVR access is more energy efficient than a usual cache access that it replaces. This is because a load/store requires accessing the TLB and all tag and data arrays in a bank. SVR accesses do not perform TLB and tag accesses at all and access only the cache way where the SVR resides. Finally, it is possible to use the performance benefits of SVectors/SStreams to save even more energy by running at a lower frequency and voltage, but we do not exploit this benefit here.

The combination of better performance and energy results in much better energy-delay product for SVectors and SStreams.

3.4 Summary

This chapter describes the programming model of ALP including the programming model for SVectors and SStreams and discusses the benefits of the SVectors/SStreams.

SVectors and SStreams are supported using three straightforward enhancements to the existing SIMD support: (i) SVector registers or SVRs, (ii) SVector load and SVector allocate instructions, and (iii) SIMD instructions capable of accessing SVRs.

Supporting SVectors/SStreams provides multiple performance and energy benefits. As for performance benefits, SVectors and SStreams reduce the load/store and overhead instructions, increase the exposed parallelism, improves load latency tolerance, save L1 cache space/bandwidth, and eliminate record alignment in L1. All the above performance enhancements also lead to direct energy benefits. Further, the reduction of execution time reduce static energy consumption and can be used to further reduce dynamic energy by running at a lower frequency and voltage.

Chapter 4

Implementation of ALP

This chapter describes the implementation of ALP including the support for SVectors and SStreams.

4.1 Support for ILP, TLP, and SIMD

ALP's support for ILP, TLP, and SIMD is conventional. As in Section 3, the SIMD implementation is roughly based on that of Intel's MMX/SSE2. Based on a previous study on the best combination of ILP and TLP for multimedia applications given in Chapter 6 and current trends of SMT and CMP general-purpose processors, ALP implements a CMP with four 4-wide out-of-order cores with two SMT threads per core. Each core has a private L1 instruction cache and a private writethrough L1 data cache. All cores logically share a unified writeback L2 cache. The L1 caches are kept coherent with a writethrough invalidate protocol.

To ensure that the baseline core is energy efficient, almost all processor resources are partitioned and caches are banked. Figure 4.1 illustrates the partitioning/banking for some resources. When both SMT threads run, each thread has exclusive access to half the partitions for most resources (e.g., reorder buffer/retirement logic, load/store queue). Notable exceptions are the caches, TLBs, and a few execution units (Figure 4.1) – these are physically partitioned, but logically shared among both threads as in a typical SMT design.

The L2 cache is logically shared among all four cores. It is physically divided into four banks (each with four sub-banks) connected with a crossbar. Each processor has one L2 bank closest to it called its *home bank*. There is a dedicated connection between a processor's L1 and its home L2 bank.

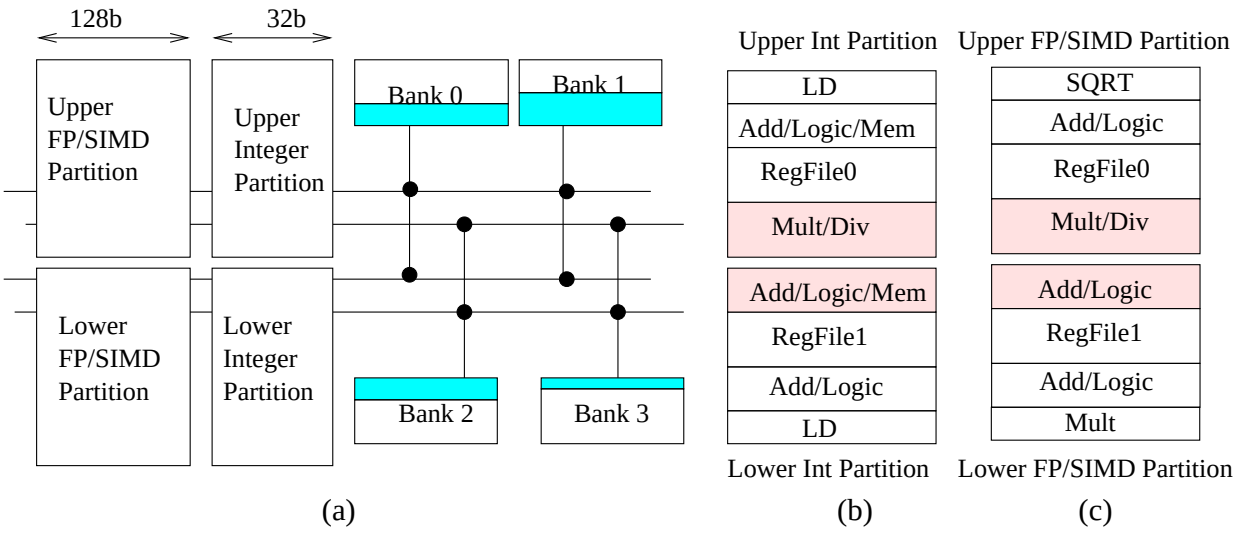

Figure 4.1: Integer, FP/SIMD, and L1 partitions/banks. (a) *Overview.* The integer and FP/SIMD execution units and register files consist of two partitions (upper and lower int or FP/SIMD execution partitions). The L1 cache consists of four banks. Each execution partition connects to all the four L1 banks – two 16B busses connect the upper partitions and another two connect the lower ones. This enables each of the two SMT threads to perform up to two memory operations per cycle. The shaded regions in the cache banks show SVRs. *(b) Integer execution partitions (32b wide).* Integer units in the upper (lower) partition can only read/write the upper (lower) partition of the register file, except for the shaded units which can access both partitions. *(c) FP/SIMD execution partitions (128b wide).* Similar to int, only the shaded units can access both register file partitions.

Table 4.1 provides the specific parameters used in our experiments. These choices were made to provide reasonable size/ports and reduced energy/cycle time for each structure. The processor and cache parameters used are the same parameters used in Chapter 2 except the processor frequency and the cache and memory latencies based on that frequency. Specifically, ALP uses a relatively low processor frequency of 500MHz (in 90nm technology) since ALP is targeted towards energy efficiency. Since ALP provides much higher performance with multiple threads and SVectors/SStreams, this frequency exceeds the real-time requirements of our applications. We can also interpret this frequency as a low frequency setting for a higher frequency processor with dynamic voltage/frequency scaling. We expect our qualitative results to hold with a wide range of parameter choices representative of modern superscalar processors. Section 5.2.1 reports sensitivity results for some important parameters.

Parameter	Value PER PARTITION	# of Partitions
Phy Int Reg File (32b)	64 regs, 5R/4W	2
Phy FP/SIMD Reg File (128b)	32 regs, 4R/4W	2
Int Issue Queue		2
-# of Entries	24	
-# of R/W Ports	3R/4W	
-# of Tag R/W Ports	6R/3W	
-Max Issue Width	3	
FP/SIMD Issue Queue		2
-# of Entries	24	
-# of R/W Ports	3R/4W	
-# of Tag R/W Ports	5R/3W	
-Max Issue Width	3	
Load/Store Queue		2
-# of Entries	16	
-# of R/W Ports	2R/2W	
-Max Issue Width	2	
Branch Predictor (gselect)	2KB	2
SVector Descriptors	12	2
Integer ALUs (32b)	see Fig. 4.1	2
FP SIMD Units (128b)	see Fig. 4.1	2
Int SIMD Units (128b)	see Fig. 4.1	2
Reorder Buffer	32 ent, 2R/2W	4
-Retire Width	2	
Rename Width	4 per thread	2
Max. Fetch/Decode Width	6 (max 4 per thread)	

Parameter	Value PER BANK	# Banks
L1 I-Cache	8K, 4 Way, 32B line, 1 Port	2
L1 D-Cache (Writethrough)	8K, 2 Way, 32B line, 1 Port	4
L2 Cache (Writeback, unified)	256K, 4 Way, 64B line, 1 Port	4

Bandwidth and Contentionless Latencies @ 500 MHz	
Parameter	**Value (cycles @ 500MHz)**
ALU/Int SIMD Latency	8 (Div-32b), 2 (Mult-32b), 1 (Other)
FP/FP SIMD Latency	12 (Div), 4 (Other)
L1 I-Cache Hit Latency	1
L1 D-Cache/SVR Hit Latency	1
L2 Cache Latency	10 (hit), 42 (miss)
Memory Bandwidth	16 GB/s

Table 4.1: Base architecture parameters. Note that several parameter values are *per partition or bank.* Section 5.2.1 reports sensitivity to some parameters.

4.2 Support for SIMD Vectors

This section describes modifications necessary for an out-of-order superscalar processor to support SVectors and SStreams. Section 4.2.1 describes the modifications to the caches and Section 4.2.2

describes the modifications to the rest of the core.

4.2.1 Modifications to the Caches

SVRs are allocated in the L1 data cache (Figure 4.1 and Figure 4.2). Thread 0 allocates even numbered SVectors in bank 0 and odd numbered SVectors in bank 2 of the L1. Thread 1 allocates odd and even SVectors in banks 1 and 3 respectively. This allows each thread to access one record from each of two SVectors in a cycle. Although each cache bank has multiple ways, we currently allocate SVRs only in way 0. Reconfiguring lines of a cache bank into an SVR is quite simple [2, 58]. One additional bit (SVR bit) per cache line is needed to indicate that it is part of an SVR. Since the L1 cache is writethrough, reconfiguration of a cache line into part of an SVR simply requires the above bit to be set; no cache scrubbing is required. An additional decoder to decode the SVR location is also not necessary since caches already have such a decoder to decode the cache line address. A multiplexer (or an additional input to an existing one) is necessary to drive the input of the cache line decoder since now there is one additional input (the CRP of a SIMD instruction). The SVector records traveling from an SVR to execution units use the existing forwarding paths used by usual SIMD loads. Thus, the L1 cache requires only minor modifications to support SVRs.

We note that since results of intermediate computations are stored in SIMD registers, SIMD instructions typically do not access SVectors for all three operands. This reduces the L1 cache bandwidth required to support multiple SIMD instructions per cycle. The two L1 busses per partition already provided (Figure 4.1) are sufficient to feed two SIMD instructions accessing two SVectors each in a cycle. It should be noted that the use of SIMD registers for temporaries makes it practically possible to allocate SVRs in the L1 cache. A traditional vector ISA requiring all vector instructions to use the vector register file will make it difficult to allocate the vector registers in the L1 cache due to the higher number of register ports required.

The L2 cache requires more support than the L1. SVector loads are sent to the requesting processor's home L2 cache bank. This bank then sends requests for the packed-words constituting the SVector to other banks as needed (recall that an SVector load may specify a stride between packed words). Each bank inserts such requests in its wait queue and services the requests in order (in parallel with the other banks). When the data is available, the bank sends it to the home bank

(across the crossbar). It should be possible for individual banks to access words starting at any byte location (i.e., to perform un-aligned loads). This capability is generally found in caches to access individual words for writing. The home bank assembles two 16B records into a 32B L1 cache line and sends these to the SVR in the L1. Each L2 bank contains separate buffers for packing records to cache lines. Note that entire L2 cache lines are not transmitted to L1 and only the records required are assembled and sent, thereby saving bandwidth and energy. The connection between the L1 and L2 can support one 32B cache line (two 16B records) per cycle.

4.2.2 Modifications to the Rest of the Core

We model an out-of-order pipeline with eight stages: fetch, decode, rename, issue, operand-read, execute, writeback, and retirement. Fetch and execute do not need any modification; decode needs small modifications to decode a handful of new instructions; and operand-read and retire stages need straightforward enhancements to read from/write to SVRs. The following discusses the modifications to rename, issue/scheduling, and retirement, and the handling of speculation, precise exceptions, and context switching.

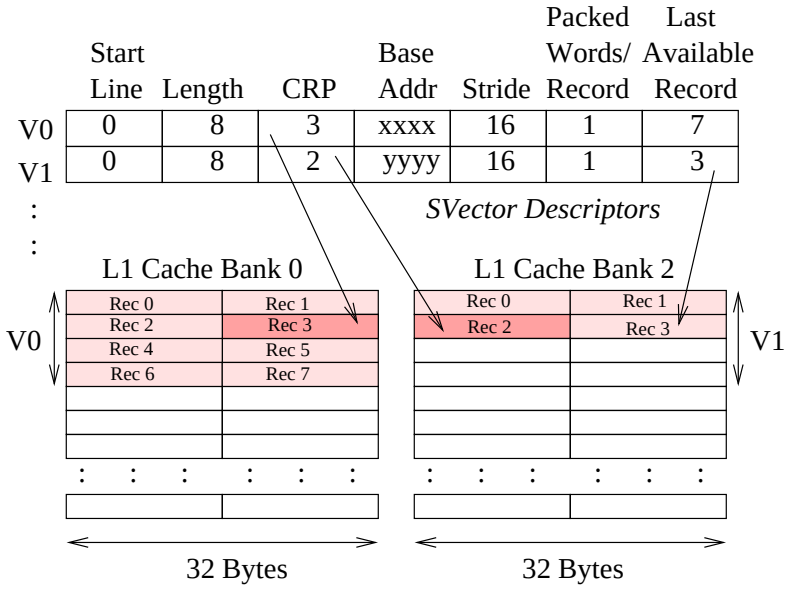

Figure 4.2: SVRs and SVector descriptor registers. Shaded cache lines contain SVRs whereas unshaded ones comprise normal cache data. Start Line, CRP, and Last Available Record are relative to the start of the cache.

Rename Stage

The rename stage of an SVector load or allocate instruction allocates an SVR and an SVector descriptor corresponding to the destination logical SVR. The logical SVector to physical descriptor mapping is stored in a rename table. The SVector descriptor register (see Figure 4.2) contains two fields (not programmer visible) in addition to the CRP and those initialized from the SVector instruction discussed in Section 3: (i) Start Line specifies the cache index address of the first record of the SVR, and (ii) Last Available Record specifies the absolute record number of the last record that has been produced (loaded/written) so far. The Last Available Record and CRP fields store the absolute record number relative to the start of the cache bank.

The SVector programming model dictates that the programmer allocate an SVector/SStream with VALLOC/VALLOCst before it is written. To facilitate the programmer detect the violation of this rule, each vector descriptor may also contain a read-only bit which is set when a vector descriptor is allocated with a vector/stream load or when a ClearCRP is executed for a vector. Any attempt to write to a vector with the read-only bit set can be flagged as an exception.

The allocation process for an SVR is analogous to that for a scalar register, requiring maintaining a free list of available space. However, an SVR requires allocation of a sequence of cache lines. One simple way to achieve this is to logically divide a cache bank into N equal sized segments, where N is determined by the number of physical registers that can be allocated in that bank. This fixes the number of SVRs, the StartLine of each SVR in a cache bank, and the maximum size of an SVR. Now a free bit can be maintained for each such logical segment to indicate whether it is free or allocated.

When a SIMD instruction with an SVector operand is renamed, the CRP field from the corresponding SVector descriptor is read to provide the location of the operand. The CRP is then incremented for the next SIMD instruction to the same SVector. Thus, the CRP is accessed and updated only in the in-order part of the pipeline, avoiding any RAW, WAW, or WAR hazards on it. Similarly, the ClearCRP instruction also performs its CRP update in the rename stage.

Issue and Scheduling

Only minor changes are necessary for the issue and scheduling logic. For a SIMD instruction that reads an SVR record, the availability of the record is marked in a ready bit as done for a normal register source operand. An SVR record is known to be available if the CRP of the SVector descriptor is less than or equal to the Last Available Record of the same descriptor.

If the required record is not yet available, the SIMD instruction awaits its availability in the issue queue just like other instructions waiting for their operands. When the required record arrives in the SVR, the cache sends a message to the rename stage to update the Last Available Record Field for that SVR. At the same time, the cache index plus bank number is passed as an 8-bit tag along a wakeup tag port of the issue queue (along the same tag ports used for passing an 8-bit register identifier when a normal load completes) and compared against the same information carried in the decoded instruction. On the tag match, the waiting SIMD instruction sets its operand ready bit, and is ready for issue if both its operands are ready. If an instruction reads from two vectors mapped to the same L1 bank, the instruction has to be stalled in the read-operand stage until both operands are read. However, this condition can be often avoided in the code itself by using vectors that map to different banks (i.e., odd and even vectors map to different banks).

For memory disambiguation and conflict resolution, the load/store queue receives VLD instructions and SIMD instructions that write to an SVector allocated with VALLOCst. Such an instruction may access several possibly strided packed-words – we conservatively assume that it accesses the entire memory range from the address of the first to the last packed-word for resolving conflicts. Support for detecting conflicts among accesses with different address ranges already exists; e.g., conflicts between regular SIMD loads/stores spanning 16 consecutive bytes and other FP/integer loads spanning fewer bytes.

Retirement

For SVector load and allocate instructions, the retirement stage frees SVRs similar to the freeing of renamed registers for ordinary instructions; i.e., the physical register that used to map to the destination logical register of the retired instruction is freed. Additionally, the SVR bits of the corresponding L1 cache lines are reset. Since the start lines and the maximum number of cache

lines for SVRs are predetermined, simple circuitry can be used to reset all SVR bits of an SVR in a cycle. An SVR is also freed when the thread that created it is killed. ALP also provides a special instruction to explicitly free all SVRs, which can be used when the subsequent code does not use SVectors. As for ordinary stores, storing records of a vector to memory also happens at retirement.

Speculation

To rollback modifications to SVR related resources by mispredicted instructions, ALP uses the conventional combination of renaming and checkpointing. Specifically, on a branch, ALP checkpoints the rename map table for SVectors and the CRP values (analogous to checkpointing integer/FP rename tables).

Precise Exceptions

Precise exceptions are largely handled through register renaming and in-order retirement as with current GPPs, with three additions. First, on an exception, the (currently allocated) SVRs need to be saved. Second, exceptions within VLD can be handled as in CODE by allowing the VLD to be restarted with partial completion [44]. Third, as discussed for retirement, for SIMD instructions that write to memory, the memory update is done only at retirement, after examining for exceptions. In case a memory write due to such an instruction needs to modify multiple packed words and there is a TLB miss/page fault on one of them, again, partial completion as in [44] can be used.

Context Switching

It is necessary to save the vector descriptors and allocated SVRs on a context switch. We can reduce the overhead of saving SVRS by resorting to lazy saving. In that case, the processor would not save an SVR until we run out of SVRs. To implement this scheme, each vector descriptor must have an additional protection bit which indicates whether the currently executing process, P, can access that vector descriptor. If this bit is set (i.e., the vector is protected), P cannot access that vector. On a context switch, the protection bit of already allocated vectors can be set after saving the current vector descriptors. When the processor runs out of SVRs, it will invoke a special trap to save all the SVRs that are not used by P (i.e., vectors whose protection bit is set). However,

lazy saving reduces the L1 cache space available for the next process.

4.3 Support for SIMD Streams

SStreams are implemented similar to SVectors, with the following additional support. For an SStream, an SVR is managed as a FIFO circular buffer with a head and a tail (LastAvailRec). When a SIMD instruction reading from (or writing to) the head of an SStream retires, the retirement logic checks if the record involved is at the end of an L1 cache line. In that case, the cache line is evicted from the SVR and a load request is sent to the L2 to bring in the next records that need to be appended to the tail (or a store request is sent to write the records at the head). Since an SVR cannot hold an entire SStream, an SStream load (SLD) is allowed to retire before the entire SStream is loaded. If the load of a subsequent record later incurs an exception, the record number is stored in a special exception field in the corresponding SVector descriptor. The exception is taken at the next instruction that refers to the record.

4.4 Summary

This chapter describes the implementation details of SVectors and SStreams, specifically the modifications that are necessary to support SVectors and SStreams on a modern out-of-order superscalar processor. SVectors and SStreams require relatively modest changes to the caches, rename, and retirement stages, especially compared with adding a separate vector unit to a core. The unique design point of SVectors that employs vector loads with SIMD computation allows SVectors to be be vector registered allocated in the L1 cache. The datapaths and the rest of the core require very few changes, if any. Overall, the additional hardware and energy overheads are modest.

Chapter 5

Evaluation of ALP

This chapter presents an evaluation of ALP. First it describes the experimental methodology in Section 5.1 followed by the main quantitative results in Section 5.2 comparing the base system, TLP support, SIMD support, and SVector/SStreams. Section 5.2 also presents a detailed analysis of the results and describes sensitivity to some key parameters. Finally, Section 5.3 describes several enhancements to ALP with the corresponding tradeoffs.

5.1 Experimental Methodology

We simulate the following systems based on Chapter 4:

- **1T:** The **base** system with only one thread and no SIMD or SVectors/SStreams (Figure 4.1 and Table 4.1).
- **1T+S:** 1T system with SIMD instructions.
- **1T+SV:** 1T+S system with SVectors/SStreams.
- **4T**, **4T+S**, **4T+SV:**. Analogous to the above three systems, respectively, but with four cores in each system (4 way CMP) and each core running one thread.
- **4x2T**, **4x2T+S**, **4x2T+SV**. Analogous to 4T, 4T+S, 4T+SV respectively, but each core is a 2-thread SMT.

We refer to the 1T system as the **base** and to the others as **enhanced**. ALP is 4x2T+SV. Note that we keep the total L2 cache size the same in 1T and 4T systems to ensure that the CMP benefits do not come simply from a larger cache size.

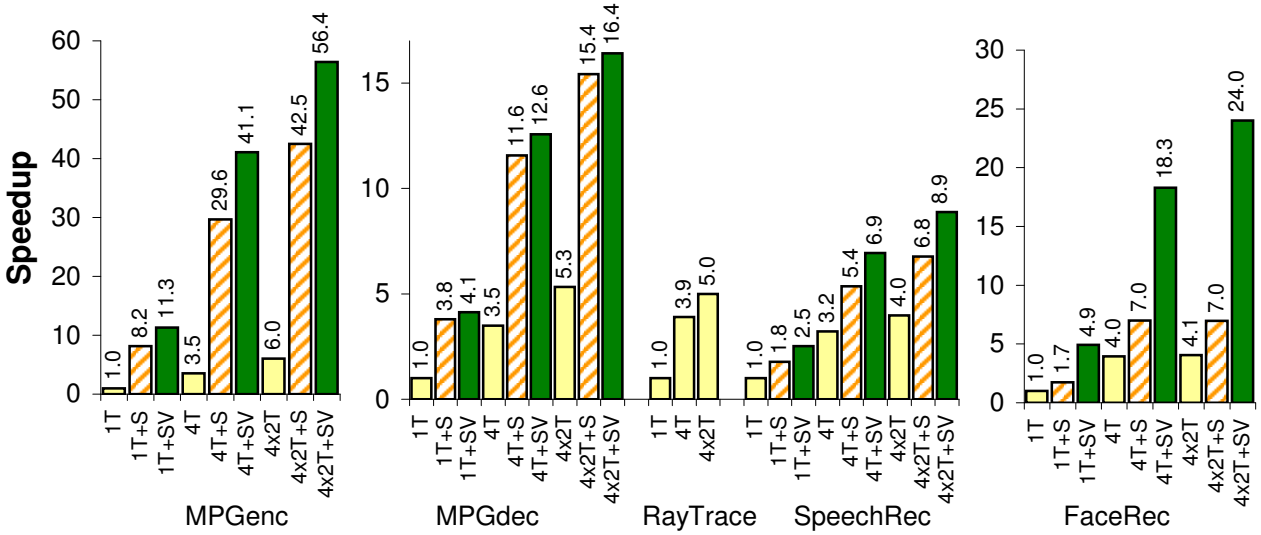

Figure 5.1: Speedup of enhanced systems over the base 1T system for each complete application (ExecutionTimeBase/ExecutionTimeEnhanced).

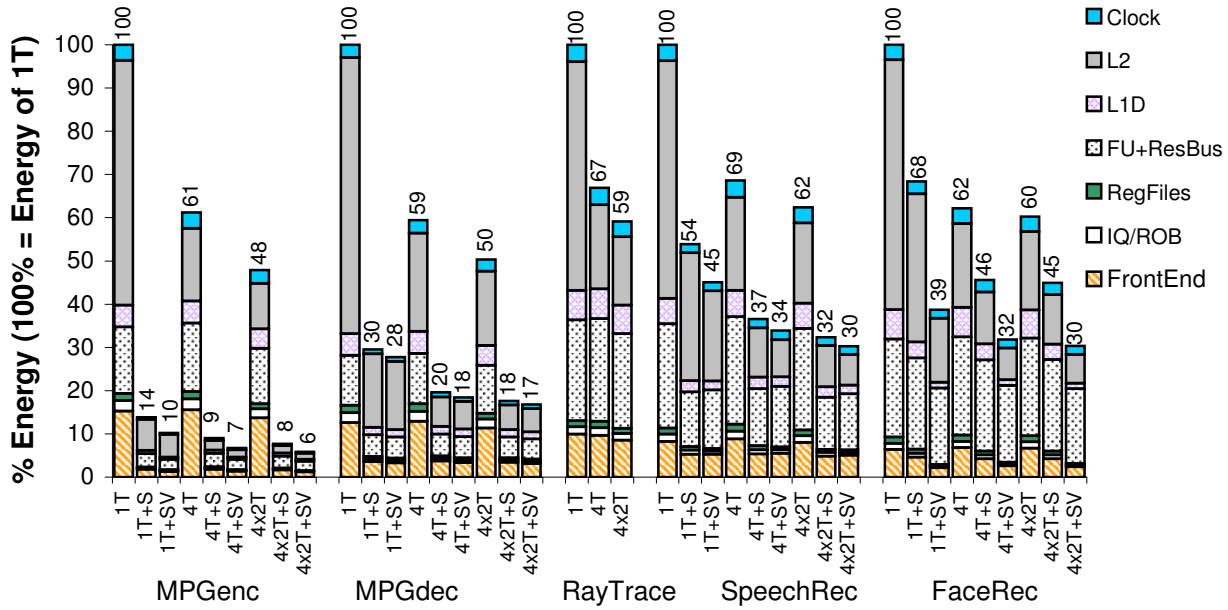

Figure 5.2: Energy consumption and distribution of enhanced systems as a percentage of the energy consumption of the base system.

To model the above systems, we use AlpSim (Section 2.3). We integrate Wattch [9] for dynamic power and HotLeakage [83] with temperature models from [69] for static power. We assume aggressive clock gating. Most components have 10% ungated circuitry [9]. To favor the base

system, we model only 2% ungated power for caches and functional units. Since the base system takes longer to execute a given application, having more ungated circuitry would make it consume more power. Since the exact amount of clock gating is highly implementation dependent, we made the choices to favor the base system. We model energy for supporting SVectors/SStreams (e.g., for SVRs, vector rename-map tables/descriptors, extra bits in caches).

5.2 Results

5.2.1 Overall Results

Figures 5.1 and 5.2 and Table 5.1 present our high-level results. For each application, they respectively provide the execution time *speedup* achieved over the base system (single-thread superscalar), the energy consumed normalized to the base, and the *improvement* in energy-delay product (EDP) over the base, for each system. Each energy bar also shows the distribution of energy among different components. Table 5.2 summarizes the above data by reporting the *harmonic* means of the speedup, energy *improvement*, and EDP improvement for key pairs of systems. For the DLP enhancements (+S and +SV), the means are computed across the applications that use those enhancements (i.e., all except RayTrace); for the others, the means are computed across all applications. For reference, Table 5.3 gives the instructions and operations per cycle (IPC and OPC) for each application and system.

Our data validates our claims that complex media applications demand support for a wide spectrum of parallelism and ALP effectively provides such support. Specifically, all the techniques in ALP are important and effective.

Across all our applications, compared with the base 1T system, ALP with all the enhancements (i.e., 4x2T+SV) shows a speedup of 5X to 56X (harmonic mean 9.2X), energy improvement of 1.7X to 17.2X (harmonic mean 2.8X), and EDP improvement of 8.4X to 970X (harmonic mean 22.7X). All comparisons are made at the same voltage/frequency for all systems – ALP's energy savings could be further improved by using dynamic voltage scaling, at the cost of some reduction in its performance speedups.

Table 5.2 clearly shows each technique in ALP contributes significantly to the above benefits,

especially when considering the relative complexity/area overhead of the technique. Specifically, comparing the effect of an enhancement over a system with all the other enhancements, we see that the mean improvement in EDP from adding SIMD instructions to 4x2T is 4.5X, from adding SVector/SStreams to 4x2T+S is 1.7X, from adding 4-way CMP to 1T+SV is 4.5X, and from adding SMT to 4T+SV is 1.4X. The means for the DLP enhancements (SIMD and SVector/SStream) are over the DLP applications (i.e., except for RayTrace).

We also performed experiments with the 4x2T+SV system restricted to 2-wide fetch/decode/retirement (but same issue width and functional units). Compared to this system, the 4-wide system reduced execution time from 5% to 22% (mean 12%), validating the need for the ILP support in ALP.

We also performed experiments with different L1/SVR hit latencies and the sensitivity to the L1/SVR latency is reported in Section 5.2.3. The speedup of SVectors/SStreams over SIMD remained within 6% for all cases. Similarly, in Section 5.2.3, we also report results with higher processor frequency (i.e., longer memory latencies) and generally found increased benefits for SVectors/SStreams over SIMD (since SVRs reduce the impact of the higher latency).

Since it is possible to emulate SStreams using SVectors by strip mining long loops, we also performed experiments using SVectors instead of SStreams for FaceRec, the only application that uses SStreams. However, SVectors are not as effective as SStreams for hiding memory latency. This is because SVector loads have to be explicitly scheduled for maximal latency hiding whereas hardware automatically schedules record loads well in advance for SStreams. As a result, we found that there is a 17% performance degradation with SVectors with respect to SStreams for FaceRec. This benefit from SStreams may appear modest but comes at a small additional hardware cost.

Finally, as an indication of application-level real-time performance, for each second, ALP supports MPEG2 encoding of 73 DVD resolution frames, MPEG2 decoding of 374 DVD frames, ray tracing of 5 512x512 frames (a scene of a room with 20 objects), recognizing 30 words using a 130 word vocabulary/dictionary (SpeechRec), and recognizing 1,451 130x150 images in a 173-image database (FaceRec). Although the above application performance may seem more than currently required in some cases, it is expected that these applications will be run with larger inputs in the future, requiring higher performance (except perhaps for MPEG decode which shows ample

performance even for future larger inputs).

App	1T+S	1T+SV	4T	4T+S	4T+SV	4x2T	4x2T+S	4x2T+SV
MPGenc	58.8	110.4	5.8	328.8	612.1	12.6	550.5	970.4
MPGdec	12.8	14.9	5.9	59.1	68.1	10.6	87.3	97.8
RayTrace	N/A	N/A	5.5	N/A	N/A	8.6	N/A	N/A
SpeechRec	3.3	5.6	4.7	14.7	20.5	6.4	20.9	29.4
FaceRec	2.6	12.7	6.4	15.3	57.5	6.7	15.5	79.2

Table 5.1: Energy Delay Product (EDP) improvement over the base (1T) system (EDP-base/EDPenhanced). Higher values are better.

Benefits of	SIMD			SVectors/SStreams			CMP			SMT			ALP
Systems compared	1T+S vs. 1T	4T+S vs. 4T	4x2T+S vs. 4x2T	1T+SV vs. 1T+S	4T+SV vs. 4T+S	4x2T+SV vs. 4x2T+S	4T vs. 1T	4T+S vs. 1T+S	4T+SV vs. 1T+SV	4x2T vs. 4T	4x2T+S vs. 4T+S	4x2T+SV vs. 4T+SV	4x2T+SV vs. 1T
Speedup	2.63	2.52	2.41	1.48	1.43	1.46	2.88	3.38	3.24	1.05	1.23	1.32	9.24
Energy	2.41	2.29	2.25	1.3	1.21	1.21	1.26	1.5	1.38	0.91	1.1	1.11	2.81
EDP	5.06	4.69	4.52	1.83	1.67	1.69	4.52	5.08	4.48	1.18	1.35	1.45	22.69

Table 5.2: Mean speedup, energy improvement, and EDP improvement. All means are *harmonic* means of the ratio of the less enhanced to the more enhanced system. The means for the DLP enhancements (SIMD and SVectors/SStreams) are over the DLP applications (i.e., all except RayTrace).

App	1T	1T+S	1T+SV	4T	4T+S	4T+SV	4x2T	4x2T+S	4x2T+SV
MPGenc	1.8 (1.8)	2.5 (8.5)	2.3 (10.3)	6.3 (6.3)	9 (30.9)	8.3 (37.5)	10.7 (10.7)	12.8 (44.4)	11.4 (51.5)
MPGdec	2.1 (2.1)	2.3 (6.4)	2.4 (6.6)	7.4 (7.4)	6.9 (19.5)	7.2 (20.1)	11.3 (11.3)	9.2 (25.9)	9.4 (26.2)
RayTrace	1.9 (1.9)	N/A	N/A	7.1 (7.1)	N/A	N/A	9.8 (9.8)	N/A	N/A
Sphinx	1.6 (1.6)	1.5 (2.2)	1.8 (2.9)	5.3 (5.3)	4.5 (6.8)	5.1 (8)	6.6 (6.6)	5.7 (8.6)	6.6 (10.3)
FaceRec	1.3 (1.3)	1.3 (2.2)	2.2 (3.4)	5.2 (5.2)	5.3 (8.7)	8.1 (12.8)	5.3 (5.3)	5.3 (8.7)	10.7 (16.9)

Table 5.3: Instructions-per-cycle (operations-per-cycle) achieved by all systems.

5.2.2 Analysis of SIMD Vectors/Streams

Section 3.3 qualitatively described the benefits of SVectors/SStreams over SIMD. We next relate our quantitative data to those benefits. We only consider the DLP applications here (i.e., all except RayTrace).

To aid our analysis, Table 5.4 gives the total instructions and operations retired for the 1T+S and 1T+SV systems as a percentage of the base 1T system. (The numbers for the other +S (+SV) systems are the same as for the 1T+S (1T+SV) systems.) Further, Figure 5.3 shows the different components of execution time in these systems, normalized to the total time of 1T+S. For an out-of-order processor, it is generally difficult to attribute execution time to different components. Following prior work [62], we follow a retirement-centric approach. Let r be the maximum number of instructions that can be retired in a cycle. For a cycle that retires a instructions, we attribute a/r fraction of that cycle as busy, attributing $1/r$ cycle of busy time to each retiring instruction. We charge the remaining 1 - a/r cycle as stalled, and charge this time to the instruction at the top of the reorder buffer (i.e., the first instruction that could not retire). This technique may appear simplistic, but it provides insight into the reasons for the benefits seen.

We categorize instructions as: Vector memory (VecMem) (only for 1T+SV), SIMD memory (SimdMem), SIMD ALU (SimdALU), and all others. In Figure 5.3, the lower part of the bars shows the busy time divided into the above categories, while the upper part shows the stall components. The busy time for a category is directly proportional to the number of instructions retired in that category. We also note that the "other" category includes overhead instructions for address generation for SimdMem instructions and SIMD loop branches; therefore, the time spent in the DLP part of the application exceeds that shown by the Simd category of instructions. The figure shows that the benefits of SVectors/SStreams arise from the following:

Reduction in busy time occurs due to the reduction in instruction count from SimdMem and related overhead (Other) instructions (Table 5.4 and benefit 1 in Section 3.3).

Eliminating SIMD loads should eliminate a significant fraction of the total SIMD and associated overhead instructions for our applications due to the low SIMD computation to memory ratio (Table 2.3). This effect can be clearly seen in MPGenc and FaceRec.

In SpeechRec, as a fraction of total instructions, the SIMD instructions are small. Nevertheless, the benefit of reducing SimdMem/overhead instructions (and associated stalls) is large enough that it allows skipping of a pre-computation phase.[1] This results in a further reduction of the "other" instructions.

[1]The original version of SpeechRec has a pre-computation phase to reduce the amount of work done in later phases. This pre-computation is omitted for +SV due to lack of any benefit.

For MPGdec, SVectors could not remove many of the SIMD loads because it uses an optimized IDCT algorithm with random memory access patterns (Section 2.2). Consequently, SVectors see a limited benefit from the reduction of instruction count. This in turn lowers the execution time benefit for SVectors.

In general, we do not expect the SimdALU instruction count to change since +SV performs the same computations. However, there is a slight increase in SpeechRec because skipping the pre-computation phase results in more SIMD computation.

Reduction in SimdMem stalls is given by the difference between SimdMem stalls in +S and VecMem stalls (plus SimdMem stalls, if any) in +SV. The benefit occurs because of the reduction in SimdMem instructions and increased load latency tolerance (benefits 1 and 3 in Section 3.3). However, the magnitude of this benefit is quite small for all applications. This is because (1) most memory accesses either hit in the L1 cache or the L2 cache and the out-of-order processor can tolerate these latencies, and (2) the L2 misses that do occur see a relatively low miss penalty since we model a low frequency processor.

Reduction in SimdALU stalls is significant specially in FaceRec because (1) a larger number of independent SimdALU instructions fit in the instruction window due to the elimination of intervening SimdMem and overhead instructions (benefit 2 of Section 3.3) and (2) better load latency tolerance results in the ALU instructions obtaining their operands sooner (benefit 3 in Section 3.3). FaceRec in particular has two dependent 4 cycle FP SimdALU instructions within a SIMD loop iteration, which feed into an FP reduction running through the loop iterations. This incurs a relatively large stall time in 1T+S. In 1T+SV, more of the (mostly independent) iterations fit in the reorder buffer (since about 50% of the instructions per iteration are eliminated), and so more parallelism can be exploited. SpeechRec sees a slight decrease in SimdALU stall time due to the same reasons. MPGenc also shows a small reduction in SimdALU but it has lower latency integer instructions; i.e., the stall time is much smaller to start with.

MPGdec does not have much of a SimdALU stall time to start with because it uses integer instructions and also has independent instructions within an iteration.

To confirm that not all of the benefits in SimdALU stall time came from load latency tolerance in FaceRec and SpeechRec, we also ran both 1T+S and 1T+SV versions of all applications with a

perfect cache where all memory accesses take 1 cycle. We continued to see benefits in SimdALU stalls (for FaceRec and SpeechRec) and total execution time from +SV (e.g., for FaceRec, 1T+SV showed a speedup of 1.8X over 1T+S with a perfect cache). These experiments also show that techniques such as prefetching cannot capture all the benefits of SVectors/SStreams.

It is possible to obtain more exposed parallelism for SIMD (+S) systems using larger resources. Although we already simulate an aggressive processor, we also conducted experiments where we doubled the sizes of the physical SIMD register file, FP/SIMD issue queue, and the reorder buffer. Of all our applications, FaceRec showed the largest speedup with increased resources - 1.67X for the SIMD version. However, SVectors/SStreams (+SV) continued to show significant benefits over SIMD even with larger resources (1.63X over SIMD for FaceRec) due to other sources such as the reduction of SIMD load instructions and overhead. Thus, SVectors/SStreams can be viewed as a way to achieve the benefits of much larger resources and more, without the higher power consumption and slower processor clock speeds associated with larger resources.

It may be possible to further improve SIMD performance by providing more SIMD logical registers. However, all the loop bodies in our applications, except the large tables, can comfortably fit in the logical SIMD registers provided. Fitting the larger tables would require a much larger register file (e.g., 32 additional SIMD registers for DCT/IDCT coefficient tables). We also note that our out-of-order core already performs dynamic unrolling effectively to use the much larger physical register file, and ALP SIMD already achieves much better performance compared with SSE2 (Section 2.4).

Reduction in other stalls results directly from the reduction in overhead instructions described above (most significantly in MPGenc and SpeechRec).

Energy benefits due to SVectors/SStreams come largely from the reduction of instruction count. Comparing corresponding +S and +SV systems in Figure 5.2, we can see energy reduction in almost every component.

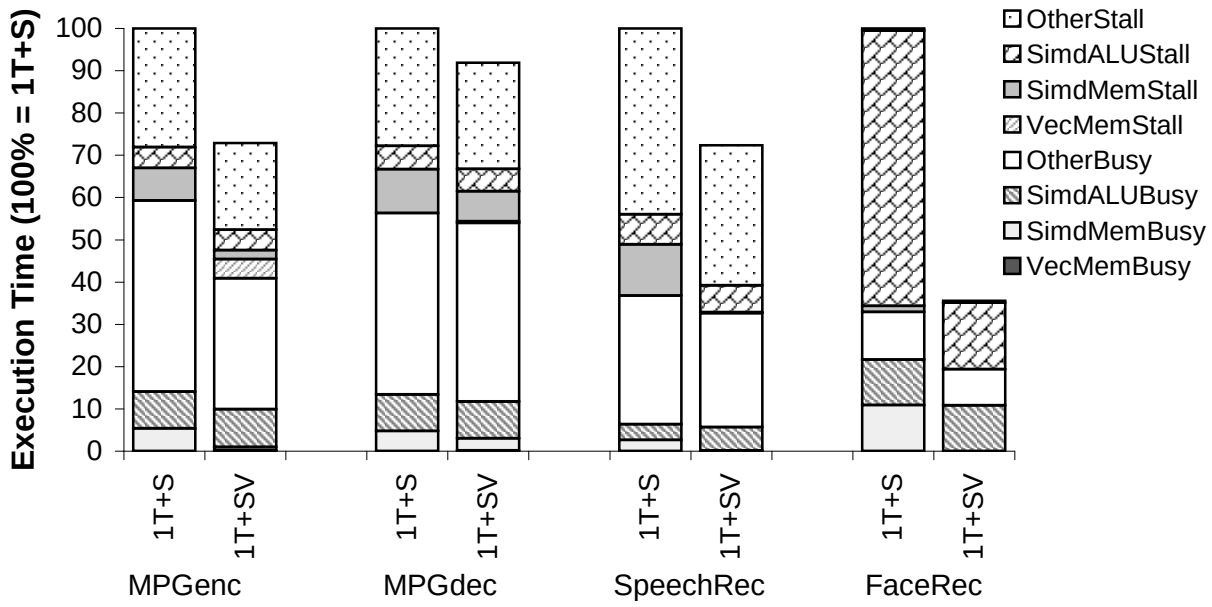

Figure 5.3: Execution time distribution for 1T+S and 1T+SV.

	MPGenc	MPGdec	RayTrace	SpeechRec	FaceRec
1T+S	17 (59)	28 (80)	N/A	51 (77)	58 (95)
1T+SV	11 (52)	27 (75)	N/A	45 (70)	34 (53)

Table 5.4: # of instructions (operations) retired for 1T+S and 1T+SV systems as a percentage of instructions (operations) retired by 1T. The numbers for other +S (+SV) systems are the same as for 1T+S (1T+SV).

5.2.3 Sensitivity to Memory and SVR Latency

Sensitivity to Memory Latency (Frequency Scalability)

Section 2.4.5 and Figure 2.5 discussed the frequency scalability (i.e., sensitivity to memory latency) of the non-SIMD and and the SIMD versions of our applications. Figure 5.4 augments Figure 2.5 by adding the frequency scalability of SVectors. We can see that generally SVectors achieve almost the same or better scalability with increasing frequency (and memory latency). The effect is significant with FaceRec and SpeechRec due to the ability of SVectors/SStreams to tolerate larger memory latencies.

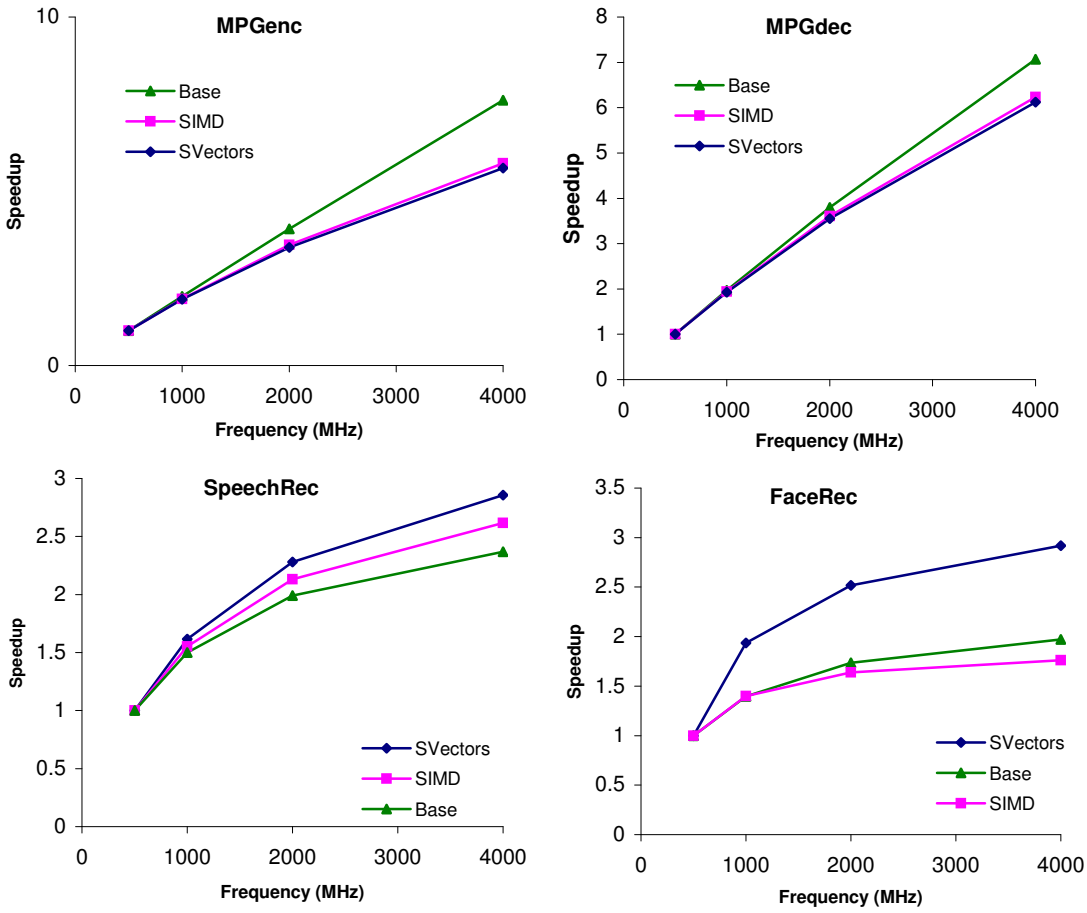

Figure 5.4: Frequency scalability of SVectors/SStreams.

Sensitivity to SVR/L1 Latency

Figure 5.5 shows the sensitivity of SVectors to SVR latency and the sensitivity of SIMD to L1 latency for single-threaded applications. For these experiments, the L1 and SVR hit latency is scaled from 1 cycle to 8 cycles to study the effect of varying L1/SVR latency on the speedup. The speedups for SVectors are relative to the SVector experiment with the SVR/L1 hit latency of 1 cycle. Similarly, the speedups for SIMD are relative to the SIMD experiment with the SVR/L1 hit latency of 1 cycle. Overall, SVectors and SIMD exhibit the same behavior with increasing L1/SVR latency. The increasing L1/SVR latency has very small effect on FaceRec since FaceRec mainly consumes data loaded from memory due to its large working set (See Figure 2.3). Applications which have critical loads that hit in L1 usually exhibit large performance degradations when the L1/SVR hit latency is increased. MPGdec shows the largest sensitivity to L1/SVR hit latency.

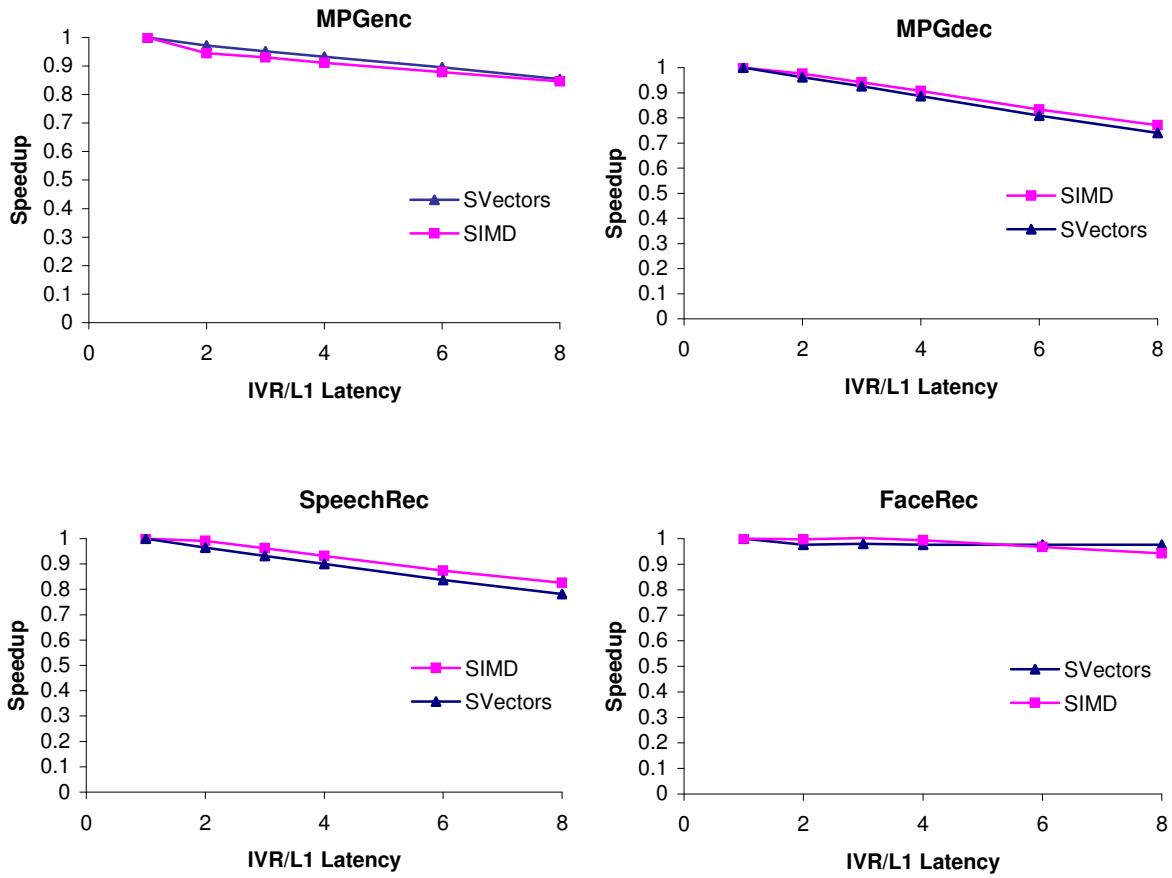

Figure 5.5: Sensitivity to SVR/L1 latency.

5.3 Discussion

This section describes some factors we should consider in supporting SVectors/SStreams. First, we consider how to decide whether an application with SIMD support would additionally benefit from SVectors/SStreams. Second, we discuss some enhancements and alternative implementation strategies for SVectors/SStreams and their tradeoffs.

5.3.1 SVector/SStream Design Tradeoffs and Enhancements

This section describes some enhancements and alternative implementation strategies for SVectors/SStreams and their tradeoffs.

REPEAT instructions

Most loops in the DLP sections of media applications have fixed bounds. Currently, SVectors use *for* loops to repeat a loop body a given number of times. However, the branch prediction may not be perfect for such loops. To achieve perfect branch prediction with such loops and to eliminate overhead instructions associated with loop control, we can use a REPEAT instruction as follows, where N specifies the number of iterations and M specifies the number of instructions in the loop body.

```
REPEAT N, M
  :    :   :
  loop body
  :    :   :
```

Some flavors of REPEAT instructions can be found in existing architectures like DSP processors. The x86 ISA also has a loop instruction. Such a REPEAT instruction is helpful in obtaining the full benefits offered by conventional vector ISAs since conventional vector code does not require a loop; instead the number of times an operation should be repeated is controlled with a vector length instruction.

CRP Fast Forward

Currently, the CRP of an SVR is incremented only when an instruction reads/writes to the associated vector. If it is necessary to skip records of a vector, we can provide an instruction to increment the CRP by a given number of records. However, we did not see any use for such an instruction in our applications.

Automatic CRP Incrementing

Automatic CRP incrementing is quite useful for all our applications. However, in some instances, a programmer could refer to the same record multiple times. In such cases, it is advantageous to have versions of instructions that do not increment the CRP. However, this requires an additional bit in

each static instruction to indicate whether that instruction automatically increments the CRP or not. This may not be desirable for some architectures.

Using SIMD Stores instead of VALLOCst

Since writing to vectors is relatively rare in our applications, an implementation can avoid implementing VALLOCst. In such a case, the programmer can use SIMD store instructions to store a vector. Since writing is rare and not often in the critical path, such an implementation could reduce some hardware complexity albeit at a possible performance loss since SIMD stores require address computation overhead instructions.

Unifying Streams and SVectors

On one hand, it may be sufficient to support only SVectors. Such an implementation could reduce the overhead of supporting SStreams. However, this requires long vectors that cannot fit in an SVR to be strip mined. As reported in Section 5.2, replacing SStreams with SVectors could lead to a modest slowdown in some cases.

On the other hand, it may be sufficient to support only streams in an implementation. In such an implementation, vectors are regarded as short streams. However, as already described in Section 4.3, there is one important difference between streams and vectors when it comes to the exception model. With vector loads, the vector load can be retired immediately after we know that it does not generate any exceptions, which happens usually after accessing the TLB for all records. Although this mechanism can be used for short streams that fit within a given number of pages, the same mechanism cannot be extended to very long streams that span a large number of pages. In such cases, we can use the exception model used for SStreams (Section 4.3). Recall that, with this model, the record at the head of a steam load (SLD) is allowed to retire before the entire SStream is loaded or the physical addresses of the entire stream are known. If the load of a subsequent record later incurs an exception, the record number is stored in a special exception field in the corresponding SVector descriptor. The exception is taken at the next instruction that refers to the record. In this sense, the stream load instruction functions as a stream specifier, or a stream handle analogous to a file handle in the operating system. A stream load specifies the properties

of the stream and individual instructions that refer to the stream will bring in (or produce) new elements.

There is one added advantage of representing vectors with streams; i.e., the architectural vector length does not have to be revealed to the programmer/compiler. This can be used to avoid vectors becoming a legacy issue in a given architecture. In the extreme case, we can support each stream with a vector register of just one record (i.e., a SIMD register).

5.4 Summary

This chapter evaluates ALP using five complex media applications. Our data validates our claims that complex media applications demand support for a wide spectrum of parallelism and ALP effectively provides such support. We found that all the techniques in ALP are important and effective. Specifically, SVectors/SStreams were effective and provided significant performance and energy benefits.

We also investigate the sensitivity of SVector/SStream speedup to the L1/SVR latency and the memory latency. Finally, we discuss some enhancements that can be made to the current SVector/SStream support and their tradeoffs.

Chapter 6

Energy Efficient TLP Support

As discussed in Chapter 2, multimedia applications exhibit coarse-grained thread-level parallelism (TLP). Further, in many environments like video conferencing, several multimedia applications are run together (e.g., an audio encoder, a video encoder, multiple audio decoders, and multiple video decoders). Such multimedia *workloads* inherently exhibit TLP. This chapter explores the design space for finding the most energy efficient form of supporting TLP. For that purpose, we explore a large design space which spans across multiple architecture styles, multiple core complexities, and a spectrum of frequencies. Although the evaluation is performed for multimedia workloads consisting of multiple single threaded applications without SIMD, we develop a mathematical model that can be used to generalize our findings to other configurations not explicitly simulated here, including multi-threaded applications we use with ALP. The insights gained from this study were the basis for the choice of TLP support in ALP.

6.1 Introduction

There are several mechanisms for supporting TLP on general purpose processors. This chapter compares three such techniques for energy efficiency:

- Chip multiprocessing (CMP) [24]
- Simultaneous multi-threading (SMT) [75]
- A hybrid (HYB) processor: a CMP processor made out of SMT cores.

SMT allows multiple application threads to be run at the same time, within the same processor, potentially increasing utilization of the processor resources. Specifically, current wide issue out-of-order processors are often unable to utilize the full supported fetch/decode/issue width for a single thread. SMT utilizes these otherwise wasted resources for other threads, potentially improving total throughput with little additional hardware. CMP, on the other hand, improves throughput by adding additional processors rather than improving their utilization.

At first glance, SMT may appear to be inherently more energy efficient than CMP since it potentially uses its resources more effectively – SMT can get more IPC (instructions per cycle) from less hardware. However, in reality, the comparison is more complex, both in the analysis to understand the experimental results and in the methodology to generate the right results.

6.1.1 Sources of Complexity and Our Solutions

For real-time multimedia applications, performance is a key constraint. A fair comparison of energy must therefore also consider performance. As a result, we compare the energy of SMT and CMP *at the same performance*, and perform this comparison for a wide range of performance points. [1] The complexity arises because each performance point can be obtained by CMP and SMT using several combinations of frequency and processor microarchitecture (referred to as *the core architecture*). For example, a narrow width core architecture at a high frequency or a wider width core architecture at a lower frequency can achieve the same performance but at different energy. A fair comparison must consider the combinations that provide the minimum energy for SMT and CMP at that performance point. This best combination for SMT could be different from that for CMP, both could differ for different workloads and different performance points, and both are difficult to determine a priori. Given that at the fairest point of comparison, the processor core architectures and frequencies employed by CMP and SMT may be different and are not known a priori, it is no longer clear which technique is most energy efficient.

The complexity of the problem becomes most evident when we try to analyze our results.

[1] A real system must also consider other metrics such as chip area, cost, power, temperature, and reliability, but considering all of these is outside the scope of this study. We further justify our focus on energy at equal performance because in the future, it is likely that chip power rather than transistor count or area will be the dominant constraint for general-purpose processors, and energy at a given performance is proportional to average power. Similarly, temperature and reliability are also related to power.

Multiple subtle interactions dictate which configuration would be most energy efficient for either CMP or SMT, and which of these two best configurations provides the lowest overall energy. The analysis involves consideration of the amount of clock gating employed, the increase in $Power/IPC^x$ (generally $2 \leq x \leq 3$) with increasing processor core complexity, and properties of the workload that make them amenable to performance speedups from CMP and SMT.

Methodologically, the complexity arises because, as implied by the above discussion, we must identify and explore a large design space, and carefully choose the specific pairs of SMT and CMP configurations that should be compared against each other. To bound the design space explored, this work focuses on out-of-order superscalar processors, based on contemporary general-purpose designs. Undoubtedly, there are other architectures that could be more energy efficient for these applications; however, we chose out-of-order superscalar processors since our focus is on general purpose processors which are becoming increasingly important for portable devices (e.g., Pentium-M). We simulate processor core complexities ranging from 2-wide to 8-wide fetch/decode width. To consider a full performance continuum, we evaluate the considered processor core architectures over a range of frequencies, from 600 MHz to 1.6 GHz (with corresponding voltages). For a given workload, we compare the energy efficiency of CMP and SMT by considering configurations that provide *the same performance*, and perform such comparisons for all performance points in the investigated design space. For each performance point, we identify the configuration (i.e., core architecture and frequency) that gives the minimum energy for SMT and CMP, and compare these minimum energy values.

We consider two- and four-thread workloads derived from combinations of 8 single-threaded multimedia benchmarks consisting of low and high bit rate video and speech codecs. (N-thread workloads run on an N-thread SMT or on an N-core CMP.)

6.1.2 Findings

Although SMT is known for its efficiency in utilizing resources, we find that CMP is consistently more energy efficient than SMT (comparable with two threads and significantly better with four threads). More specifically, our results show that for the design space explored, for each workload, at each performance point, (1) the least-energy CMP configuration showed lower energy than the

least-energy SMT configuration, (2) the energy difference was larger at the high-performance points and for four-thread workloads (for four-thread workloads, the average benefit of CMP over SMT was 44% for the highest performance points and 18% for the lowest performance points), and (3) the least-energy SMT configuration had moderately higher complexity and higher frequency/ voltage. To understand the reasons for our results and to extend our findings to other systems and workloads not simulated here, we perform a qualitative analysis and develop an analytic model that exposes a subtle interplay between various factors affecting the results.

6.1.3 Broader Implications

Our results have two broad implications beyond simply a comparison of SMT and CMP. First, our results clearly underscore the advantage of CMP for four-thread (and higher) workloads for our applications. However, a four-core CMP will have a much larger silicon area than an SMT with moderately higher core complexity. Thus, the energy advantage of CMP comes at an area cost. To get the best of both worlds, for four-thread workloads, we study a hybrid CMP/SMT architecture **(HYB)** where a CMP is built out of SMT cores (e.g., IBM Power5). We find such a two-core CMP with two-thread SMT cores has significantly higher energy efficiency than a pure SMT processor. The hybrid architecture with two SMT cores generally needs less silicon area than a 4-core CMP. Moreover, such an architecture will also provide better energy for workloads that would not scale well in performance across a four core CMP.

The second broad implication arises from our observation that although CMP configurations generally give the best energy efficiency, different configurations are optimal at different performance points (as is the case with SMT). This motivates the use of recently proposed adaptive architecture and frequency/voltage scaling techniques. We also find that applying these techniques independently on the different CMP processor cores to create a heterogeneous CMP provides even further energy savings for CMP. While it may be possible to apply some of these techniques to SMT, it is as yet unclear how this can be done.

6.2 Experimental Methodology

6.2.1 Systems Modeled

Design Space & Naming Convention

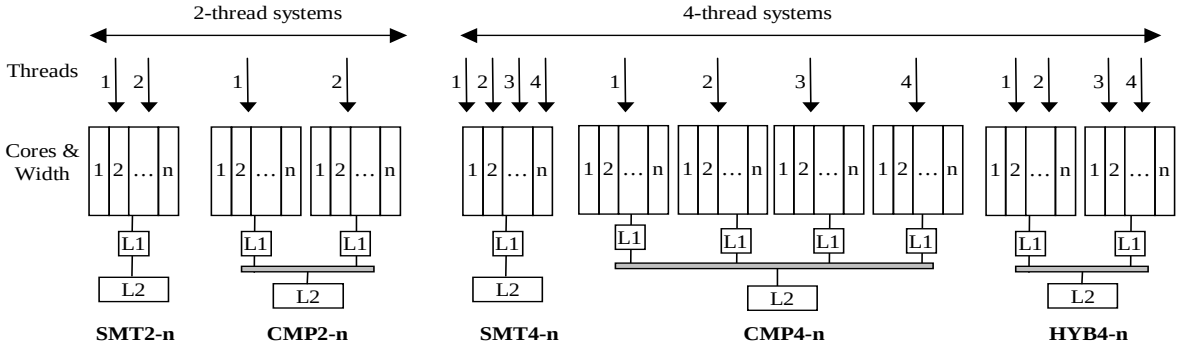

Figure 6.1: Systems modeled and naming convention. An SMT, CMP or HYB processor is named as NAME<# of threads>–<fetch/decode width of a core>, where NAME is SMT, CMP or HYB.

We model two **classes of systems** - one supporting two threads and the other supporting four threads as illustrated in Figure 6.1. For the two-thread systems, we model a single core SMT that supports two threads and a 2-core CMP (each core supports one thread). For the four-thread systems, we model a single core SMT that supports four threads, a 4-core CMP (each core supports one thread), and a hybrid system, which is a 2-core CMP with each core being an SMT supporting two threads.

To bound the design space explored, we focus on out-of-order processors. To adequately represent this design space, we model several *core architectures* for the out-of-order processor cores, ranging from a fetch/decode width of two to eight. For each fetch/decode width, we appropriately scale other resources (e.g., instruction window size and the number of functional units), as discussed in Section 6.2.1. For a given CMP, we assume all processor cores have the same configuration (except when we consider adaptive architectures in Section 6.3.4).

We adopt the following naming convention, as also shown in Figure 6.1. For a two thread system, we denote an SMT or CMP system with cores with a fetch/decode width of N by **SMT2-**N and **CMP2-**N respectively. Note that CMP2-N has two N-wide cores whereas the SMT2-N has only one N-wide core. Similarly, we use **SMT4-**N, **CMP4-**N, and **HYB4-**N to denote four-thread

SMT, CMP, and HYB systems (respectively) with cores that have a fetch/decode width of N.

Processor Core & Memory Parameters

Table 6.1 summarizes the processor core and memory hierarchy parameters used for the experiments reported here – *Width* refers to the fetch/decode width and *Threads* is the number of threads supported by a processor core. Given that there are many ways to reasonably scale processor core parameters with fetch/decode width and many ways to set memory hierarchy parameters, we explored a few alternate parameterizations (e.g., Instruction window size = $Width$ x 16) and found that the trends were the same as those presented here.

In all cases, the processor core is an out-of-order superscalar, modeled after the MIPS R10000 superscalar core. Specifically, the register file is separate from the reorder buffer as in modern implementations. For SMT, we assume that all concurrent threads share most resources in the processor, including the instruction window, functional units, L1 caches, and register files; however, each thread is given a separate branch prediction table and a return address stack. Further, 32 additional integer and floating point registers are assumed for each additional thread, to capture the architectural state. Sizes of the instruction TLB and data TLB are also increased to support additional threads on SMT processors as given in Table 6.1. We use the ICOUNT policy [75] to prioritize instruction fetch from different threads.

We model L1 instruction and data caches and a unified L2 cache. All threads of an SMT share the L1 data and instruction caches. For CMP, each processor core has its own L1 data and L1 instruction cache, and all cores share an L2 cache through a common bus. For this work, we studied unoptimized sequential versions of applications with SIMD instructions. Therefore, these applications have a relatively high computation to memory ratio than reported in Chapter 2. Further, the media applications studied have relatively small working sets. Therefore, relatively smaller L1 caches are sufficient to obtain very high hit rates.

Width Dependent Parameters	
Fetch/decode rate	$Width \in \{2, 3, 4, 5, 6, 7, 8\}$
Instruction window (reorder buffer) size	24 * $Width$
# of integer functional units	$Width$
# of floating point units	1 ($Width \leq 3$), 2 ($Width \leq 6$), 3 ($Width > 6$)
# L1 ports	2 ($Width \leq 6$), 3 ($Width > 6$)
# Address generation units	2 ($Width \leq 6$), 3 ($Width > 6$)
Retirement rate	# Int units + # FP units + # L1 ports
Load/Store buffer size	8 * $Width$
Thread Dependent Parameters	
L1 data cache	8K * $Threads$, Write Back
L1 data cache associativity	4 ($Threads \leq 2$), else 8
L1 instruction cache	16K * $Threads$, 4-way
iTLB & dTLB size	128 entries * $Threads$
Branch prediction	2K entries * $Threads$, bimodal agree
Return address stack size	16 * $Threads$
Integer register file size	32* $Threads$ + reorder buffer size
Float register file size	32* $Threads$ + (reorder buffer size)/2
Common Processor Parameters	
Processor core speed	600 MHz to 1.6 GHz (voltage scaled)
Process technology	0.13 micron
Integer FU latencies	1/4/12 add/mult/div (pipelined)
FP FU latencies	4 default, 12 div. (all but div. pipelined)
Branch penalty	16 cycles
Bit widths	64-bit data / 48-bit address
Common Memory Hierarchy Parameters	
L1 & L2 cache line size	64B
L2 cache (on chip)	1MB, 16-way associative, write back 64B line, 1 port
Main Memory	16B/cycle, 4-way interleaved
Common Contentionless Memory Latencies	
L1 instruction cache access time	1 cycle
L1 data cache access time	2 ($\leq$ 16K), 3 (32K)
CMP bus latency	2 cycles
L2 cache access time (on chip)	8 cycles
Main Memory	100 cycles (L2 miss to memory)

Table 6.1: Processor core and memory parameters. $Width$ refers to the fetch/decode width of a core and $Threads$ refers to the number of threads supported by a single core.

Both L1 instruction and data cache sizes were selected after a sensitivity analysis. For each application, we determined the minimum cache size necessary to obtain a hit ratio over or close to 99% for the data cache and a hit ratio over or close to 98% for the instruction cache. Across all applications, the largest such size was found to be 8K for data and 16K for instruction cache. Therefore, as summarized in Table 6.1, each CMP processor core was given an 8K L1 data cache and a 16K L1 I cache for all CMP systems. SMT cores were given the same amount of cache per thread supported (e.g., two-thread SMT has a 16K L1 data and 32K L1 instruction cache). Consequently, in a given system, both CMP and SMT processors have the same total amount of cache. (We also evaluated another set of cache parameters where the SMT cache size was smaller than the combined cache size on the corresponding CMP; e.g., for 4-thread systems, we used 16K data cache for SMT and 8K for each of the four CMP cores for a total of 32K. The overall trends in the results were the same as those reported here.) All caches are non-blocking and writeback. A relatively large cache line size of 64B was found to be beneficial for these applications due to their streaming nature.

We did not perform a detailed sensitivity analysis for the L2 cache because (a) it does not have much impact on performance since L1 hit ratios are very high due to the use of non-SIMD applications, (b) the average power for the L2 cache is a small fraction of the total even for the large 1MB L2 cache we model (see Figure 6.6), and (c) the L2 cache is common to all systems. For the same reasons, although we measure the energy of the L2 cache, we do not include it in the total energy reported in our comparisons.

We investigate all the core architectures at frequencies ranging from 600MHz to 1.6GHz with voltages scaled according to data for the Intel Pentium-M (Centrino) mobile processor which supports this frequency range [33]. The configurations of a core architecture at different frequencies may be interpreted as a single processor supporting dynamic frequency and voltage scaling (DVS) or as different fixed-frequency processor designs. Note that, compared to wider processors, it may be possible for narrower processors to support a higher frequency at a given voltage. We did not model this effect since it is difficult to do so accurately and would only further favor CMP processors, as easily confirmed by our analysis in Section 6.3.3.

Henceforth, we use the term ***core architecture*** to refer to the fetch/decode width and other

parts dependent on this width. We use the term ***configuration*** to refer to a combination of the core architecture and frequency. For a given number of threads t and core architecture c, we refer to CMPt-c (or SMTt-c or HYBt-c) as the ***system architecture*** and the combination of system architecture and frequency as the ***system configuration***.

6.2.2 Workloads

Benchmark	Type of codec	Input size (Frames)	Base IPC
GSMd	Speech	1000	3.4
GSMe		1000	3.5
G728d	Speech	1000	2.2
G728e		1000	1.9
H263d	Video	450	3.1
H263e		50	2.0
MPGd	Video	200	2.8
MPGe		50	1.5

(a)

2-thread workload	Total IPC	2-thread workload	Total IPC
MPGe_MPGe	2.9	MPGe_GSMd	4.9
MPGe_G728d	3.7	H263e_H263d	5.0
H263e_H263e	3.9	H263d_G728d	5.3
G728e_G728d	4.1	H263e_GSMe	5.4
MPGe_MPGd	4.2	GSMe_G728d	5.7
H263e_MPGd	4.7	H263d_GSMe	6.6
H263d_G728e	4.8	GSMe_GSMe	7.0

(b)

4-thread workload	Tot. IPC
MPGe_MPGe_MPGe_MPGe	5.8
MPGe_MPGe_G728e_G728d	7.0
H263d_G728e_H263d_G728e	9.6
H263e_H263d_H263e_GSMe	10.4
MPGe_MPGd_GSMe_GSMe	11.2
MPGd_GSMd_H263d_G728d	11.4
H263d_GSMe_H263d_GSMe	13.2
GSMe_GSMe_GSMe_GSMe	13.9

(c)

Table 6.2: (a) Single-thread benchmarks with IPC on most aggressive processor, (b) two-thread workloads, and (c) four-thread workloads. (b) and (c) are ordered by the sum of the IPCs of the constituent threads when run individually on the most aggressive core configuration.

We consider eight single-thread multimedia benchmarks covering high and low bit rate video and speech codecs. These benchmarks are summarized in Table 6.2(a) and described in more detail in previous work [27]. They form the core components of many high-level multimedia applications (e.g., video teleconferencing, DVD playback, video editing) and are representative of most widely used media benchmarks. A real system could run either the multi-threaded version of such an application or several of such applications together as a *workload*. For instance, a video encoding application may run a multi-threaded video encoder. In contrast, a video teleconferencing appli-

cation between N participants at N different sites would involve one video and speech encoder and N-1 video and speech decoders at each site. A participant may receive high or low bit rate streams depending on the computation power and bandwidth available at the other participating sites; therefore, a site may need to support different types of decoders and encoders within the same application.

Since we performed this study as a precursor to ALP for evaluating the effectiveness of TLP support, we had to pick one of the above two alternatives to make our study manageable. Since the applications we evaluated were not multi-threaded, we used the workload approach where several applications are run together as a single workload as in teleconferencing. Thus, for this study we use workloads consisting of a number of different copies of different combinations of the benchmarks in Table 6.2(a) with each copy working on its own data. However, our analysis and the mathematical model in Section 6.3.3 apply to both multi-threaded applications and multi-programmed workloads.

For the small-scale systems studied here (two or four thread CMP and SMT), we can assume that the total number of threads available for running in a realistic system will be larger than the number of simultaneous threads supported in the system. A real-time operating system (RTOS) must therefore choose which combination of threads to co-schedule at each time, with consideration for any synchronization among related threads (e.g., audio and video for the same stream). The co-scheduling algorithm can have an impact on the overall performance, but real-time co-scheduling algorithms for SMT are still an open area of research [37]. To eliminate dependence on the co-scheduling algorithm, we report results separately for different combinations of N threads for an N-thread system. Thus, for a two-thread system, we separately report results for different pairs of the eight benchmarks of Table 6.2(a). The actual performance of a full real-time application would depend on how often the RTOS co-schedules each of the specific combinations for its chosen co-scheduling policy.

Furthermore, a real RTOS scheduling policy may co-schedule different parts of N concurrent benchmarks at different times. For example, consider two benchmarks with very different execution times per frame. The shorter frame may be co-scheduled along with any part of a longer frame and it is possible the execution characteristics are different depending on when the two frames are co-scheduled. Again, to report results independent of the co-scheduling policy and to average out

such differences, we run several frames of the co-scheduled benchmarks to get the average behavior for that benchmark combination. The maximum number of frames we consider for each benchmark is reported in Table 6.2(a) along with the IPC on an 8-wide (superscalar) core for each application.

Studying all combinations of two and four out of our eight benchmarks would have resulted in an inordinately large number of workloads (e.g., 36 possibilities for 2-thread systems). We therefore selected a subset of these, summarized in Table 6.2, using the following methodology. We determined that our results were sensitive to the total IPC. A higher symbiosis means the SMT is well utilized. for the combination. For the two-thread case, we therefore divided all the possible benchmark pairs into four categories, based on the sum of the IPCs of the two benchmarks when run individually on the most aggressive processor (described in Section 6.2.1). From each category, we then selected at least two workloads to represent the range of symbiosis[2] values in that category. For the four-thread case, we considered all possible pairs of the two-thread workloads and used the same criteria as for the two thread workloads. For the four thread CMP/SMT architecture, we used the four-thread workloads – the constituent two-thread pairs were paired again for each SMT processor.

For workloads with threads from different applications, there is an inherent problem when attempting to compare the same amount of work on all system architectures. For the first application of a given workload, we run the same number of frames for all CMP and SMT system architectures. During that time, the amount of work done by the other threads can change slightly for different system architectures. This is an inherent property of SMT and CMP architectures. Since we assume that the RTOS has enough threads to schedule and we want to maximize the throughput of the system, we run the other threads until the first thread finishes its maximum number of frames. Thus, the first thread executes the same number of instructions on all core architectures. To overcome the problem of the other threads executing a slightly different number of instructions on different system architectures, when comparing energy efficiency, we use energy and execution time metrics that are normalized to the number of instructions executed – Energy Per Instruction (EPI) and Time Per Instruction (TPI).[3] Note that this problem does not arise when all threads

[2]Symbiosis is defined as *effectiveness with which multiple jobs achieve speedup when run on multithreaded machines* [71].

[3]We cannot use IPC since we vary the frequency as well. $TPI = 1/(\text{Frequency} \times \text{IPC})$.

in the workload are from the same application, since all threads finish almost at the same time due to the fairness of SMT's ICOUNT policy and the symmetry of CMP. We study several such workloads for all systems, and they follow the same overall patterns as the others. We also measured the discrepancy in instruction counts and found that it was $< 5\%$ for most and $< 12\%$ for all workloads.[4]

6.2.3 Simulation Environment and Methodology

We model the performance of the systems in Section 6.2.1 using a version of the RSIM simulator [28] modified to support both SMT and CMP. RSIM is an execution-driven, cycle level simulator that models the full impact of branch and address speculation (e.g., modeling wrong path instructions) and contention at all resources. All applications are compiled with the SPARC SC4.2 compiler with full optimization (O4). Previous work showed that for the benchmarks studied here, performance scales virtually linearly with processor frequency, since the amount of time spent on memory stalls is negligible [27]. We therefore run simulations at one base frequency, and use linear scaling to obtain execution time at other frequencies. We validated this by running actual simulations with frequencies at 100MHz intervals. (It was impractical to simulate all frequencies over the 1GHz range reported here.)

We use a combination of tools integrated with RSIM to model the dynamic and static energy of all systems. To model dynamic energy of processor cores and caches, we use the Wattch tool [10] integrated with RSIM. Wattch is enhanced to model duplicated resources, additional tags for thread IDs, etc. For CMP, we model the energy consumption of the bus between the L1 and L2 caches using models from the Orion project [78].[5] We assume a bus length of 5mm with two cores and 10mm with four cores. However, as shown in Figure 6.6, the bus energy is very small. We also model the static energy consumption of major structures like caches, register files, and the instruction window using the HotLeakage model [83]. For this purpose, we model the temperature of major structures on the processor using temperature models described in [69]. However, for the structures we model

[4]The only other reasonable alternative would be to consider a specific number of frames from the other threads. This has the drawback that some thread contexts/cores will remain idle when the shorter thread(s) finishes, and would likely favor CMP (since the idle processor could be deactivated with CMP). Nevertheless, we evaluated several workloads with this alternative and found the results to be similar.

[5]We thank Li-Shiuan Peh and Hang-Sheng Wang for quickly generating and providing us the bus models.

and for the 0.13 micron technology parameters we use, we find that the leakage power is less than 2% of the dynamic power.

We assume aggressive clock gating for all CMP and SMT cores, as in many current general-purpose processors [5]. Although it is relatively easy to disable clocking of unused ports of the multi-ported structures and unused functional units, it is practically not possible to achieve 100% clock gating. Some clock gating events are expensive to identify and some of the gating is foregone to avoid lengthening critical paths, race conditions, unequal clock distributions, extensive validation and di/dt effects [23]. We assume that all but 10% of the unused circuitry can be turned off with clock gating in typical industrial clock-gated circuits, as mentioned by Brooks et al. [10]. (For reference, for the Pentium 4 processor, the idle power consumption was found to be 15% to 20% of the power consumed when running MPEG [12].) We later discuss the sensitivity to this parameter and give results with 20% ungated circuitry and with no gating at all (Section 6.3.3).

6.2.4 Metrics and Representation of Collected Data

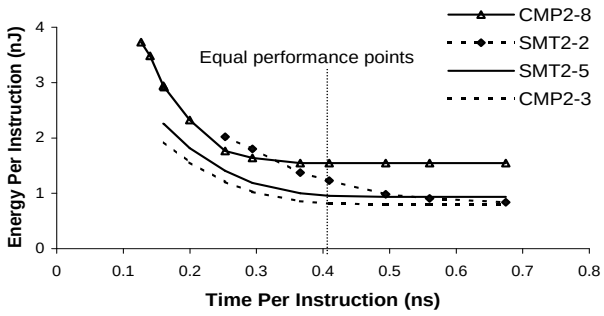

Figure 6.2: Example EPI vs. TPI graph.

When considering energy as a metric of comparison, one must also consider the performance obtained. One common metric used is the energy-delay product (i.e., energy/performance) [22]. However, this metric is unsatisfactory if the user desires a fixed amount of performance or is constrained by a fixed amount of energy (e.g., battery life). Specifically, for real-time applications such as those studied here, there is often a fixed desirable performance target (derived from the application deadlines and the rest of the load on the system). We therefore focus our effort here on understanding optimal energy configurations, given a fixed performance target (the data and

analysis for the optimal performance configuration for a fixed energy target is similar). We report results for the energy-delay product and other more conventional metrics in Section 6.3.5. Recall from Section 6.2.2 that we use the normalized energy per instruction (EPI) and time per instruction (TPI) to measure energy and performance respectively.

For a CMP or SMT with a given core architecture, varying the processor frequency provides a continuum of performance points. Any of these points could be achieved either as a fixed-frequency design or in a system with DVS support. We collect data for CMP and SMT systems using all combinations of core architectures and frequencies given in Section 6.2.1. For a given class of systems (2 or 4 thread), for each workload, for each performance point (i.e., TPI), we compare all the system configurations that provide that performance, to determine which system gives the least energy for that performance. In general, the lowest EPI system is different for different performance points.

Figure 6.2 illustrates how we represent the collected data to perform this comparison for 2-thread systems. It plots EPI versus TPI for the MPGe_MPGd workload. Each curve in the figure represents one core architecture for an SMT or CMP system and each point on a given curve represents a different frequency (from 1.6GHz on the left to 600MHz on the right). Only four systems are shown for clarity - CMP2-8, SMT2-2, SMT2-5 and CMP2-3. The points along a vertical line on this graph represent points of equal performance. The lowest point on the line represents the configuration that gives the least energy for that performance. For example, for a target TPI of 0.4ns, CMP2-3 provides the least energy. CMP2-3 also turns out to provide the least-energy for a large performance range for this workload.

In general, the least-energy system architecture may be different in different performance ranges for two reasons. First, not all system architectures may be able to provide all performance points on the extreme left and right sides of the TPI axis. For example, in Figure 6.2, CMP2-8 is least-energy for a few of the leftmost TPI values, because CMP2-3 is unable to provide that level of performance even at the maximum frequency. Second, in the middle performance ranges, the least-energy configuration could change if the curves of two architectures cross. This crossing can occur due to the non-linearities in the voltage vs. frequency curve.

6.3 Results

We performed all our analysis using graphs such as shown in Figure 6.2 – one graph per 2-thread or 4-thread workload, 14 curves per 2-thread graph (7 each for CMP and SMT), and 21 curves per 4-thread graph (7 each for CMP, SMT, and HYB). To distill the information from these graphs into a more readable form, Figures 6.3 and 6.4 show the EPI values of the *best* SMT and the *best* CMP architectures for the entire performance range. Figure 6.4 shows the *best* HYB architecture as well. To save space, only a few representative workloads are shown since other workloads follow similar trends. Since one system architecture is the best-energy architecture for a range of performance points (using different frequencies), we label the performance points at which the best system architecture changes (going from left to right). The best architectures are marked as sn, cn, and hn to indicate SMT, CMP, HYB respectively, with a core fetch/decode width of n.

Tables 6.3(a) and (b) supplement the above graphs by tabulating the magnitude of the EPI difference between the best SMT and CMP configurations, as a percentage of the SMT configuration. Since we cannot tabulate each of the infinite performance points and since an average over the entire space is not too meaningful, we divide the TPI axis on the EPI-TPI time graphs into three regions (high, medium, and low), based on the performance degradation relative to the highest performance configuration (which is always the 8-wide, 1.6GHz CMP for all workloads). For two-thread workloads, the performance degradation is from 1X to 1.5X for the first region (highest performance), 1.5X to 3X for the second region (medium performance), and $>$ 3X for the third region (lowest performance). For four-thread workloads, the performance degradation is from 1X to 2X, 2X to 4X, and $>$ 4X respectively for the three regions. For the highest performance region, we only include points where at least one SMT configuration can achieve that performance.

For each region, the tables give the average percentage improvement (in EPI) of the best CMP configuration over the best SMT configuration (and best hybrid configuration for four-threads) at different performance points in this region. The average is calculated by finding the area between the two curves for that region and dividing that by the TPI difference for that region. Note that for a given region in this table, the optimal CMP, SMT, and HYB architectures may be different at different points in the region.

The EPI vs. TPI graphs as shown in Figure 6.3 and Figure 6.4 also convey information regarding

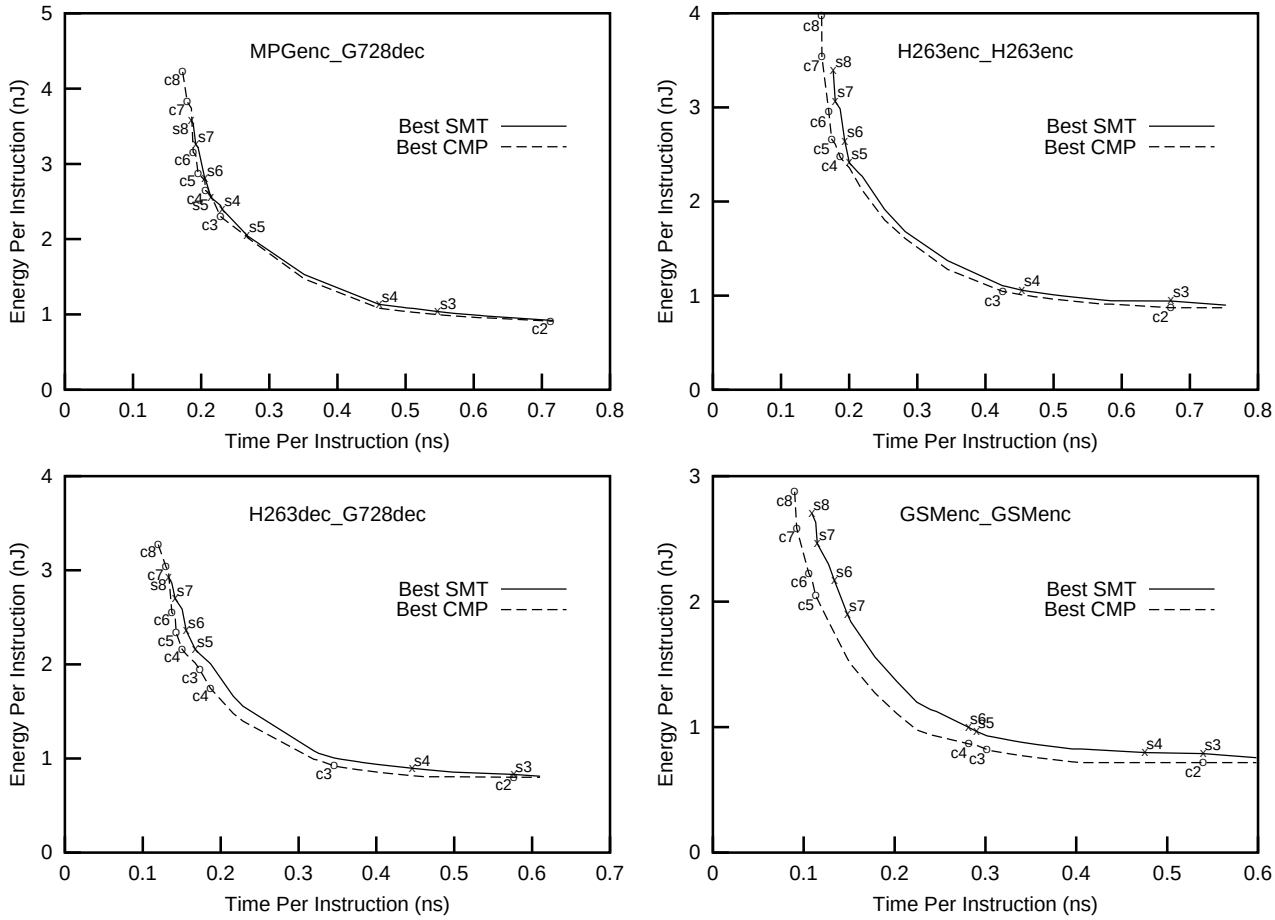

Figure 6.3: EPI for best-energy SMT, best-energy CMP and, best-energy HYB configuration at different performance points for two-thread workloads. The best system configurations are marked as sn or cn to represent SMT and CMP, respectively, with cores of fetch/decode width of n. They are marked only at the points where the best configuration changes, going from left to right.

absolute performance and *average power*. The performance of systems with 8-wide cores at 1.6GHz is given by the topmost points on each curve. Further, for a given TPI, EPI is proportional to the *average power*. Since the discussion below focuses on energy at equal performance (i.e., EPI for given TPI), it follows that the discussion also applies to average power (for different performance points).

6.3.1 Results Across All Configurations

Our data shows that for all our systems and workloads, for all performance regions, a CMP architecture gives the least EPI. Comparing CMP and SMT, for two thread workloads, the difference

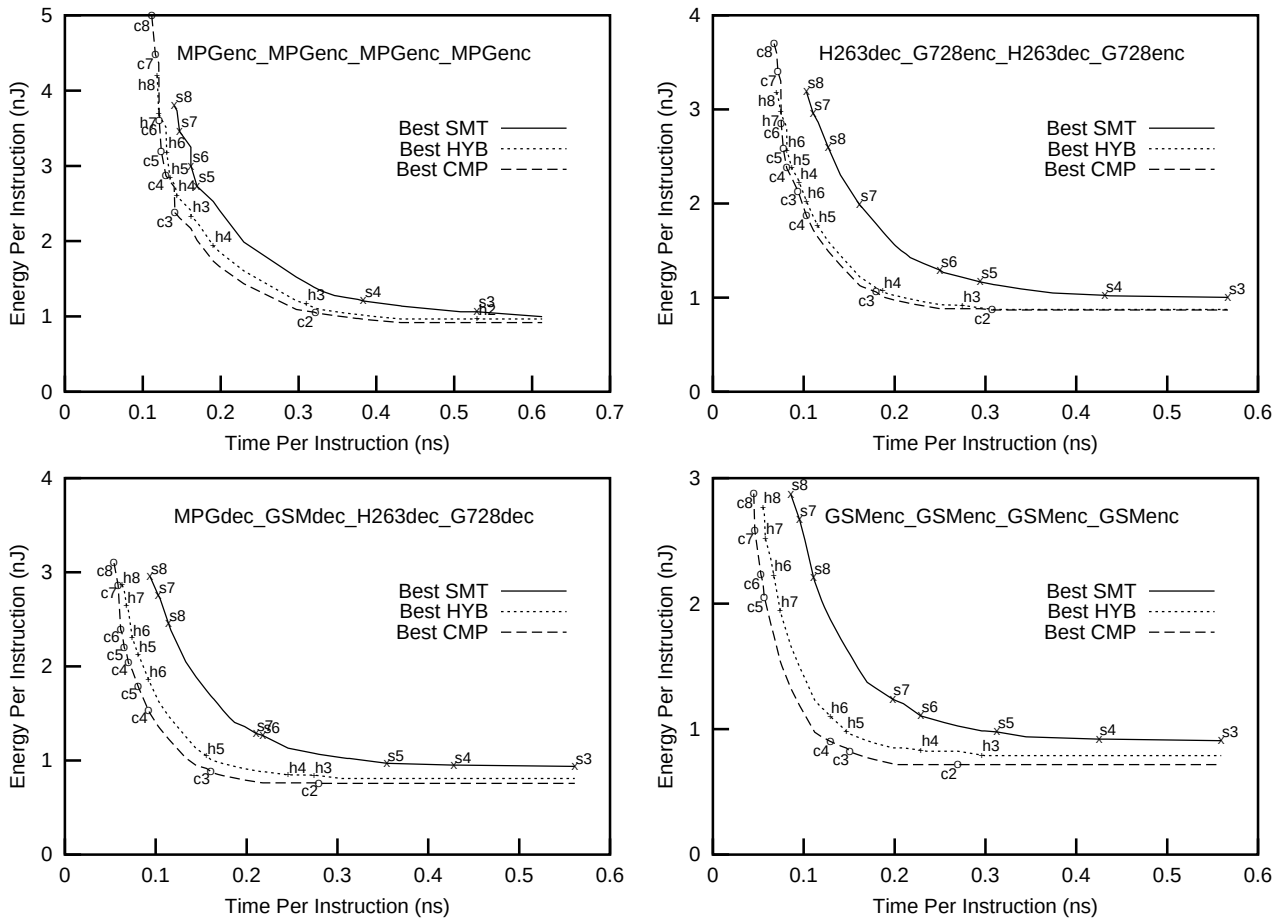

Figure 6.4: EPI for best-energy SMT, best-energy CMP and, best-energy HYB configuration at different performance points for four-thread workloads. The best system configurations are marked as sn, cn, or hn to represent SMT, CMP, and HYB, respectively, with cores of fetch/decode width of n. They are marked only at the points where the best configuration changes, going from left to right.

between them is mostly small (average 10%, 9%, and 4% respectively over the three regions). For four thread workloads, CMP is significantly better than SMT (average 44%, 39%, and 18% for the three regions). In both cases, the difference increases with increasing performance.

Focusing on HYB, our data shows that it is significantly more energy efficient than SMT and comes close to CMP for many performance points and workloads. On average, the difference between CMP and HYB is reduced to 13%, 11%, and 6% for the three regions, respectively.

Workload	High	Med	Low
MPGe_G728d	4	3	2
H263d_G728e	5	4	1
MPGe_MPGe	7	7	2
G728e_G728d	8	6	3
H263e_H263e	8	6	4
H263e_H263d	8	8	4
MPGe_MPGd	9	6	3
H263e_MPGd	10	8	3
H263e_GSMe	10	10	4
H263d_G728d	10	10	4
GSMe_G728d	11	10	5
MPGe_GSMd	13	10	6
H263d_GSMe	15	16	8
GSMe_GSMe	19	18	9
Average	10	9	4

(a)

Workload	CMP vs. SMT			CMP vs. HYB		
	Hi	Med	Lo	Hi	Med	Lo
MPGe_MPGe_MPGe_MPGe	30	23	10	10	7	5
MPGe_MPGe_G728e_G728d	30	24	10	10	8	5
H263d_G728e_H263d_G728e	42	37	16	6	5	1
H263e_H263d_H263e_GSMe	45	39	19	9	10	6
MPGe_MPGd_GSMe_GSMe	47	43	21	18	15	8
MPGd_GSMd_H263d_G728d	50	46	22	17	15	7
H263d_GSMe_H263d_GSMe	53	51	25	13	14	10
GSMe_GSMe_GSMe_GSMe	54	49	24	21	17	10
Average	44	39	18	13	11	6

(b)

Table 6.3: Range of % EPI savings of the best-energy CMP over the best-energy SMT and HYB for different performance regions (a) two-thread workloads and (b) with four-thread workloads for high, medium, and low performance regions.

6.3.2 The Best Core Architectures

Figures 6.3 and 6.4 show that for all the workloads, the best CMP uses a less or equally complex core architecture (i.e., lower or same fetch/decode width) than the best SMT and the best HYB at any given performance point. HYB is closer to CMP than SMT. (The total resources available to CMP, however, are larger because CMP has more processor cores.)

Further, different core architectures are the best in different performance regions. Systems with fixed (vs. adaptive) architectures need to pick one *overall best core architecture* to implement. We

define this to be the architecture that, when averaged across all performance points, has the least EPI difference from the best core at the same performance point. If we were to draw the curve for the overall best CMP (SMT) architecture in the graphs of Figures 6.3 and 6.4, it would be very close to the best CMP (SMT) curve given. Note, however, that the overall best architecture may not have performance points for the entire performance region covered by all core architectures. Further, this also assumes the presence of dynamic frequency/voltage scaling since an architecture is best only at the appropriate frequency.

Overall Best CMP core

For two-thread CMP, the CMP2-4 architecture is the overall best, across all performance points. Its EPI is better than or within 10% of the best CMP EPI for all but one 2-thread workloads (for which it is 14% worse). For four-thread workloads, the CMP4-4 configuration is best overall and its EPI is within 10% of the best CMP EPI for all but one workloads (for which it is 15% worse).

However, CMP2-4 and CMP4-4 do not exhibit the very highest and the very lowest performance points. At the highest frequency, CMP2-8 outperforms CMP2-4 by 26% to 48% for four of the two-thread workloads. For four-thread workloads, the performance difference is even more prominent. Similarly, at the lowest frequencies, less aggressive architectures can provide lower energy, albeit at lower performance (CMP4-4 cannot get down to these performance points because of the bound on the minimum frequency).

Overall Best SMT core

For SMT, the SMT2-5 and SMT4-6 architectures are the overall best for two and four-thread workloads, respectively. Across all performance points the EPI difference between the overall best and the actual best for a given performance point is at most 12%. However, as with the CMP case, SMT2-5 and SMT2-6 do not exhibit the very highest and the very lowest performance points. At the highest frequency, SMT2-8 outperforms SMT2-5 by 20% to 34% for eight two-thread workloads. For four-thread workloads, SMT4-8 outperforms SMT4-6 by 24% to 30% for four workloads.

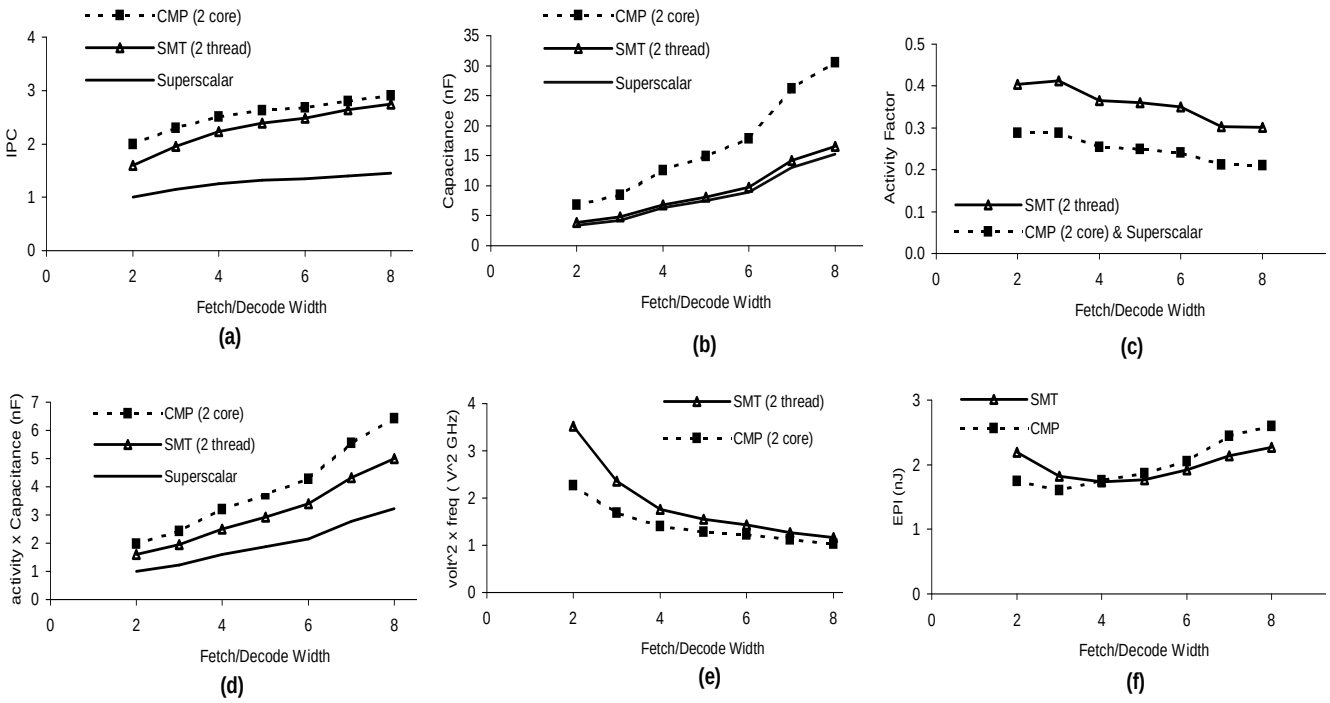

Figure 6.5: Factors affecting EPI vs. fetch/decode width of core, *for a given TPI*. (a) IPC, (b) total capacitance, C (proportional to maximum power), (c) activity factor, α, (d) αC (proportional to average power), (e) $V^2 f$, and (f) EPI.

Overall Best HYB core

For HYB, overall, the HYB4-5 architecture is the overall best configuration. Its EPI is better than or within 11% of the best HYB EPI for all workloads.

6.3.3 Analysis of the Results

Section 6.3.3 provides a qualitative analysis for the underlying reasons for the superior energy efficiency of CMP over SMT, and determines factors that could make SMT more energy efficient. Section 6.3.3 formalizes the intuition from Section 6.3.3 by providing a mathematical model. These sections are complex, but essential to understand our results and extrapolate to other system parameters and workloads.

Qualitative Understanding

We know that $EPI = Power \times TPI = \alpha CV^2 f \times TPI$, where α is the switching activity factor, C is the total capacitance, V is the supply voltage, and f is the frequency.[6] α and C depend on the core architecture. We refer to the reciprocal of $\alpha CV^2 f$ as the *energy efficiency*, or simply the *efficiency*, of the corresponding system configuration. It follows that for a given class of system (superscalar, CMPN, or SMTN, where N is the number of threads) and a given TPI, the minimum EPI occurs for the configuration that has the highest energy efficiency as defined above.

Figure 6.5 illustrates the effect of each factor in the energy efficiency term. *For a specific TPI* (selected from the middle part of the performance range), for CMP2 and SMT2 running the workload MPGe_MPGe, parts (a)-(f) of the figure respectively show the variation with core complexity (i.e., fetch/decode width) of IPC, C, α, αC, $V^2 f$, and EPI. Note that for the $V^2 f$ plot, the frequency value for a given core width is determined as the ratio of TPI and IPC with that core. We show the graph of the product αC due to its significance – αC is proportional to the average power at a given voltage/frequency. It is also interesting to see the contribution to average power from different structures. Figure 6.6 shows this information for four systems and the MPGe_MPGe workload.

To understand when the lowest EPI is obtained, the following discusses, *for a given TPI* (i) how each of the above terms varies with core width, and (ii) the relative values of each term for CMP and SMT *for the same core width.*

(a) IPC *(SMT $\leq$ CMP for our workloads):* For both SMT and CMP, we expect IPC to increase with core complexity at a diminishing rate, eventually leveling off, as seen in Figure 6.5(a). For our compute-bound workloads, at a given core complexity, CMP achieves almost perfect speedup in IPC, equal to the number of threads or processor cores. The IPC speedup achieved by SMT at that

[6]This only considers dynamic power. Static or leakage power was negligible for the simulated technology. Although static power is expected to become more important and CMP's higher area for equal performance may seem to imply higher static power, there are several factors that make it unclear whether static power will favor CMP or SMT. First, analogous to clock gating, static power can also be contained using power gating and various process technologies (e.g., SOI, multiple-threshold transistors, etc. [8]). Second, most of the chip area is typically consumed by the L2 cache which is common to both CMP and SMT. Finally, several factors favor CMP *when comparing at equal performance* – SMT's higher average dynamic power implies higher temperature which increases leakage, SMT's slightly more complex cores require higher area, CMP's lower resource utilization implies higher potential for power gating, and CMP's lower required frequency allows for slower but lower leakage transistors.

core complexity depends on the superscalar IPC of the individual applications and on the resource sharing interactions among the constituent applications. The higher the superscalar IPCs and the more negative the interaction, the more difficult it is for SMT to get a high IPC speedup. Thus, for our workloads, we expect that the SMT IPC will be $\leq$ the CMP IPC at a given complexity, with the relative difference depending on the above two factors.

(b) Capacitance *C (SMT < CMP)*: C increases rapidly with complexity, and depends only on the power model used. It is proportional to the *maximum power* of a processor. We see that at any core complexity, C of SMT is only a little higher than that of the superscalar, since an SMT processor adds very little hardware to the base superscalar. C for CMP, however, is a factor of N higher than that of the corresponding superscalar (and hence SMT), where N is the number of processor cores.

(c) Activity factor *α (SMT > CMP)*: Since α is the fraction of total transistors switched per cycle, informally, it depends on (i) the amount of clock gating or other power management techniques in the system, and (ii) on the fraction of total transistors that are "useful" to switch (this is roughly correlated to IPC/C, since more useful switching implies higher IPC and more total transistors imply higher C). Without clock gating or other power management techniques, the value of α is always roughly the same. As clock gating and other power management techniques become more aggressive, α becomes more sensitive to IPC/C.

With increasing complexity, α will either stay roughly constant (e.g., with no clock or power gating) or will change roughly correlated to IPC/C. At lower complexities, IPC/C could increase a little, but at higher complexities, this ratio will generally go down. Figure 6.5(c) shows this trend.

To compare α for SMT and CMP at a given core complexity, we note that α for CMP is the same as that for the corresponding superscalar. α for SMT is higher than for the superscalar (and hence for CMP) since SMT sees a higher resource utilization. Again, the factor by which α is higher for SMT depends on the amount of clock gating (and other power management) and on the IPC speedup from SMT (relative to C) over the corresponding superscalar. For example, with no clock gating, α stays the same as the superscalar (and CMP). More aggressive clock gating and higher IPC speedup of SMT will increase the value of α relative to CMP.

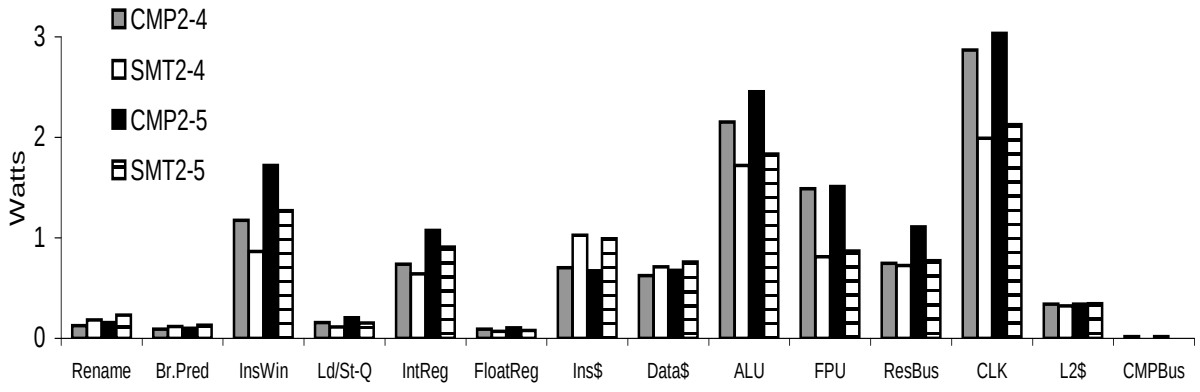

Figure 6.6: Average power of individual structures for CMP2-4, SMT2-4, CMP2-5, SMT2-5 for MPGe_MPGe at the maximum frequency.

(d) α**C** *(SMT < CMP)*: From the above discussion, we can deduce that αC increases with increasing width. Comparing CMP and SMT, αC is lower for SMT (at a given core width) due to its lower IPC and C, and high clock gating.

(e) $\mathbf{V^2f}$ *(SMT > CMP for our workloads)*: Since f is inversely proportional to IPC for a given TPI and since V also depends on f, the V^2f curve decreases rapidly with increasing complexity and IPC (following IPC^3 where $V \propto f$). At a given complexity, SMT will have a higher value of V^2f than CMP since the IPC of SMT is lower for our workloads.

(f) EPI *(SMT ? CMP)*: The EPI graph in part (f) is the product of the previous two curves (αC, V^2f). For both CMP and SMT, with increasing complexity, αC generally increases. At lower complexities, this increase is offset by the decrease in V^2f (due to increasing IPC), reducing EPI. As the IPC increase diminishes, however, the decrease in V^2f is insufficient, causing EPI to increase again. Thus, each EPI vs. core complexity curve has a minimum point, which is the highest energy efficiency point mentioned above.

Putting it together – Why is CMP EPI better than SMT EPI?

For SMTN to have a better EPI than CMPN (N is the number of threads), the most energy efficient SMT configuration (call this $C_{min}SMT$) must have a higher efficiency than the most efficient CMP configuration (call this $C_{min}CMP$). Let us start by considering the relative efficiency of SMT at the $C_{min}CMP$ configuration. Based on the above discussion, at the core complexity of

$C_{min}CMP$, SMT has an advantage over CMP from C, which is much lower than that for CMP. However, SMT has a disadvantage from α (based on clock gating and IPC speedup from SMT) and V^2f (based on IPC speedup from SMT and CMP). If SMT has a high enough IPC speedup (relative to CMP), then it is possible that this disadvantage is offset by the lower C, and SMT has a lower EPI than CMP at $C_{min}CMP$ (lower bounds for this IPC speedup are presented in the next section).

If the IPC speedup of SMT at $C_{min}CMP$ is not high enough, then it is still possible to see a lower EPI for SMT by changing its core complexity. By increasing the core complexity, SMT can increase its IPC and reduce V^2f.[7] However, this increases C, offsetting some of the benefit of reduced V^2f. C will increase by a larger amount as the slope of the C vs. complexity curve gets steeper. Increasing complexity could also increase α, depending on the amount of clock gating and the relative change in IPC to C. Depending on how much the IPC of SMT rises with respect to a rise in αC, the increase in IPC (i.e., reduced V^2f) may or may not be able to beat the EPI of CMP. Since the minimum EPI point for CMP is also the minimum for the superscalar, the superscalar IPC does not rise fast enough with increasing core complexity at this point. The rise in IPC for SMT must therefore come from a high speedup of SMT over the superscalar from running multiple threads together, not just from the inherent ILP of each thread. Thus, whether SMT or CMP will have lower EPI depends on a number of factors, as summarized below.

In summary, the following factors will hinder SMT from achieving a higher energy efficiency than CMP:

1. High level of clock gating (or other power management).
2. Steep capacitance vs. core complexity curve, requiring more IPC speedup from SMT to be more efficient.
3. Workload characteristics that make it harder for SMT to obtain IPC speedup over the corresponding superscalar; e.g., high superscalar IPC or negative resource sharing interactions among constituent applications.

[7]The analysis also applies to reducing complexity, but this is not likely to improve efficiency since we start from a point that is also most energy efficient for the superscalar.

4. Workload characteristics that give CMP a high IPC speedup over the corresponding superscalar (e.g., compute-bound workloads).

For the systems we study, all of the above factors are present, hindering the energy efficiency of SMT relative to CMP. The first factor is not likely to change towards favoring SMT in the near future. Nevertheless, we ran experiments increasing the non-clock-gated circuitry to 20% from 10%. This made SMT slightly better than reported here, but CMP is still better for most workloads. Although unrealistic, we also experimented with eliminating clock gating altogether. This made SMT significantly better for most 2-thread workloads and for the lower performance region of 4-thread workloads but CMP was still far better for most of the 4-thread workloads in the high and medium performance regions. The second factor above is also likely to hold for out-of-order cores. The third and fourth are workload dependent, and could possibly lead to different results with different workloads. For instance, FaceRec and SpeechRec described in Chapter 2, show low IPC when memory latencies are high since their working sets do not fit in the caches. Such applications could favor SMT when memory latencies are high. However, when the memory latencies are low even FaceRec and SpeechRec show relatively high IPC. Therefore, whether such applications would favor SMT or CMP on a given system depends on the single-thread IPC achieved on that particular system.

Mathematical Model and Validation

We further formalize the above qualitative analysis with a mathematical model, and use the model to derive quantitative bounds on the IPC speedups that are required from SMT for it to be more energy efficient. Since it is impractical to simulate all hardware configurations and workloads, our mathematical model plays an important role in increasing the applicability of our results to systems and workloads that we do not simulate. For simplicity, our model below assumes that $V \propto f$, but it can be modified for other relationships between V and f. The approximation gives EPI $= \alpha C f^3 \times TPI$. Substituting $f = \frac{1}{TPI \times IPC}$, we get $EPI = \frac{\alpha C}{IPC^3} \frac{1}{TPI^2}$. This expression is independent of frequency. It indicates that for a given system (i.e., superscalar, CMPN, or SMTN, where N is the number of threads) and a given TPI, there is one core architecture that provides the highest energy efficiency. This is the architecture that provides the lowest $\frac{\alpha C}{IPC^3}$ for that system, and

we refer to that architecture as *Amin*.[8] We call the reciprocal of $\frac{\alpha C}{IPC^3}$ as the energy efficiency, or simply *efficiency (Eff)*, of a *core architecture*. Note that in Section 6.3.3, we defined efficiency for a *system configuration*, which depends on frequency. The above efficiency is for the core architecture and is independent of frequency.

Denote the IPC speedup given by CMPN (SMTN) over the corresponding superscalar as IPCSpeedCMPN (IPCSpeedSMTN). Note that the following assumes a given TPI for all cases. When not clear from the context, we will use subscripts or postfixes to indicate the system and core architecture that a specific quantity applies to. Thus, $\alpha_{CMP2,AminCMP2}$ refers to α of a CMP2 system using the core that is the most efficient for CMP2 for the given TPI. We know that $C_{SMTN,AminSMTN} \approx \frac{C_{CMPN,AminSMTN}}{N}$. Let $\alpha_{SMTN,AminSMTN} = G \times \alpha_{CMPN,AminSMTN}$ (note that α for CMP is the same as that for the corresponding superscalar). Then SMTN EPI is better than CMPN EPI if

$$Eff_{\text{CMPN,AminCMPN}} < Eff_{\text{SMTN,AminSMTN}}$$

i.e., if $Eff_{\text{CMPN,AminCMPN}} <$

$$\frac{IPCSpeedSMTN_{AminSMTN}{}^3 \times \frac{IPC_{CMPN,AminSMTN}{}^3}{IPCSpeedCMPN_{AminSMTN}{}^3}}{G \times \alpha_{CMPN,AminSMTN} \times C_{CMPN,AminSMTN} \times \frac{1}{N}}$$

i.e., if $IPCSpeedSMTN_{AminSMTN}{}^3 >$

$$\frac{Eff_{\text{CMPN,AminCMPN}}}{Eff_{\text{CMPN,AminSMTN}}} \times \frac{G \times IPCSpeedCMPN_{AminSMTN}{}^3}{N} \quad (1)$$

Since for our workloads, CMPN sees an IPC speedup of N, it follows that SMT is more energy efficient if

$IPCSpeedSMTN_{AminSMTN}{}^3 >$

$$\frac{Eff_{\text{CMPN,AminCMPN}}}{Eff_{\text{CMPN,AminSMTN}}} \times G \times N^2 \quad (2)$$

[8]Choosing a core architecture fixes a frequency for a given TPI. If this frequency is not supported by the system for the architecture with the highest efficiency, then for Amin, we must choose the core architecture with the next highest efficiency for which the corresponding frequency is supported.

Equations (1) and (2) clearly quantify the impact of all the four factors identified in the previous section as hindrances for SMT. Higher clock gating is represented by a higher G. The impact of the steepness of the C vs. core complexity curve is quantified by the ratio of the efficiency of CMP (and equivalently the superscalar) at AminCMP and at AminSMT. A steep C vs. core complexity curve will yield a higher ratio (recall that in the earlier discussion, "steepness" was considered relative to the increase in IPC, which is quantified by the efficiency). Finally, the workload characteristics are represented by the SMT and CMP IPC speedup terms.

We can also use the above equation to yield a lower bound on the SMT IPC speedup for SMT to be more efficient. We know that the efficiency ratio in the above equation is ≥ 1, since CMP is most efficient at AminCMP. Similarly, $G \geq 1$. Then for our workloads, equation (2) implies that at the maximum SMT efficiency point, the speedup in IPC of SMTN must be $> N^{2/3}$. For N=2, this is 1.59 and for N=4, this is 2.52. More generally, from equation (1), SMT must see an IPC speedup of at least 80% of the CMP speedup for two threads and 63% for four-threads.

Validation: To increase our confidence in our analysis and experiments, we attempted to fit our results within the above equation. In particular, we determined the lowest value of G from our experiments for a given performance point and plugged it into equation (2) to get a tighter bound on the SMT speedup. All our results were within this bound. Specifically, for the highest performance region, the $IPCSpeedSMT_{AminSMT}$ averaged 1.6 to 1.8 for 2-thread workloads and 2.1 to 3.0 for 4-thread workloads. These speedups are high and were sufficient for SMT to be comparable with CMP for many 2-thread workloads, but still lower than the predicted bound for 4-thread workloads.

6.3.4 Implications of Results

A Case for a Hybrid CMP/SMT Architecture

Our results show that HYB is significantly more energy efficient than pure SMT and close to CMP. Moreover, HYB uses only slightly more complex cores than CMP at the least energy configuration. Consequently, the hybrid architecture with two SMT cores generally needs less silicon area than a 4-core CMP, for equal performance. Moreover, such an architecture will also provide better energy for workloads that would not scale well in performance across a four core CMP. Consequently, HYB

appears to be an attractive "best-of-all-worlds" solution for four-threads.

A Case for Adaptive Architectures and DVS

Our data shows that it is possible to pick one "overall best" core architecture for CMP and one for SMT to obtain close to optimal EPI for many cases. However, this core architecture is insufficient in the regions of maximum or minimum performance for many workloads, and in the middle performance regions for some workloads. If these cases are important, then the best design would involve an adaptive processor that can change the active resources and fetch/retire width depending on the workload and desired performance/energy target. Previous control algorithms for such adaptations (e.g., [66]) could be applied in a straightforward way to CMP, but need further investigation for SMT. Further, CMP shows better potential for such techniques due to its lower utilization of resources.

Similarly, our data also shows that the lowest EPIs are obtained across a range of frequencies for CMP and SMT, supporting the use of DVS for these systems.

Finally, we note that CMP provides unique methods of adaptation that are not readily available to SMT architectures. Specifically, CMP can apply DVS and architectural adaptations independently to each core, depending on the type and amount of work to be done in each co-scheduled thread. There is much current research on multivoltage and multifrequency chips that makes it appear that such a design would be viable (e.g., [67]). We find that 10 out of the 14 two-thread workloads can obtain 9%-15% energy savings over the currently optimal CMP configuration, in the *High* and *Medim* performance regions, if independent DVS is available to each core.

6.3.5 Energy-Delay Product (EDP) and Other Metrics

So far, we have focused on determining least-energy configurations for a given target performance. Here we discuss other energy related metrics.

Table 6.3.5 summarizes the energy efficiency of CMP, SMT and HYB using two other metrics: (1) Least Energy, which does not take performance into account at all and (2) Energy-Delay Product (EDP) [22]. For the two-thread system, the table gives the average energy savings of CMP over SMT and HYB (with the range in parenthesis), and the overall best CMP, SMT, and HYB systems

Metric	2-thread systems (avg. and range)		
	CMP vs. SMT	**BestCMP**	**BestSMT**
Least Energy	2% (0%-5%)	CMP2-2	SMT2-3
EDP	7% (1%-18%)	CMP2-4	SMT2-5

Metric	4-thread systems (avg. and range)				
	CMP vs. SMT	**CMP vs. HYB**	**BestCMP**	**BestSMT**	**BestHYB**
Least Energy	15% (7%-21%)	6% (0%-9%)	CMP4-2	SMT4-3	HYB4-3
EDP	42% (28%-53%)	12% (7%-20%)	CMP4-4	SMT4-7	HYB4-6

Table 6.4: Energy efficiency of CMP, SMT, and HYB using Least Energy and EDP as metrics, at the highest frequency.

at the highest frequency.

We find that even in terms of the more conventional energy metrics, CMP is superior to SMT especially for the four-thread workloads, and CMP is comparable to HYB for the four-thread workloads.

6.4 Summary

This chapter evaluates the mechanisms for supporting TLP on general-purpose processors. It provides the first comprehensive comparison of the energy efficiency of CMP and SMT for multimedia workloads on modern out-of-order general-purpose processors. SMT processors increase throughput by using resources more efficiently while CMP processors duplicate resources at the expense of low utilization. For the fairest comparison, it is important to compare energy at the same performance. Further, since different combinations of core architecture and frequency can provide a given performance but with different energy, it is important to explore a large design space and carefully pick the system pairs for comparison. We consider a wide range of architectures and frequencies to provide a large range of performance points. For each performance point, we compare the lowest energy SMT and CMP. We evaluate two-thread and four-thread multimedia workloads, derived from eight (sequential) multimedia benchmarks.

We find that across the performance spectrum, a CMP configuration is the most energy efficient for our systems, for all of our workloads used in this chapter. For two-threads, the difference between CMP and SMT is low, but for four-threads, it is significant. Our detailed analysis finds that several

factors influence this outcome, including (1) aggressive clock gating, (2) high CMP speedup, (3) the relatively steep slope of the power vs. complexity curve in modern out-of-order processors, and (4) the inability of SMT to achieve the extremely high speedups required for it to be more efficient than CMP. It is unlikely that a modest change of several processor or technology parameters would bring significantly different results. Our analysis shows that it is necessary for SMT to obtain very high speedups (80% of the CMP speedup for two-thread workloads and 63% of the CMP speedup for four-thread workloads), or reduce clock gating significantly for SMT to become considerably better. Since it is impractical to simulate all hardware configurations and workloads, we develop a mathematical model that can encompass all the above factors. This model plays an important role in increasing the applicability of our results to systems and workloads that we do not simulate, including explicitly parallel applications.

Although our results clearly underscore the advantage of CMP for four-thread workloads, this advantage comes at the cost of silicon area. A hybrid architecture consisting of two cores with each core supporting a two thread SMT is much more energy efficient than SMT and has a lower area than CMP for equal performance. This architecture is also likely to perform better for workloads that do not scale across four CMP cores. Thus, such a hybrid architecture appears to be an attractive middle ground solution.

Finally, we find that at most performance points, one core architecture provides the best overall energy efficiency. There are, however, other performance points where other core architectures are optimal (for CMP and SMT). This motivates adaptive architectures that can deactivate parts of the core that lead to energy inefficiencies. Similar observations also motivate DVS. We also find that exploiting heterogeneity in CMP cores could further improve the CMP energy efficiency. SMT processors are not currently easily amenable to such adaptations.

The analysis presented in this chapter for different multi-programmed workloads also applies to the multi-threaded applications presented in Chapter 2. MPGenc, MPGdec, and RayTrace in Chapter 2 show high cache hit ratios and relatively smaller working sets and the results and the analysis given here can be directly extended to such applications. However, FaceRec and SpeechRec described in Chapter 2 show low IPC when memory latencies are high since their working sets do not fit in the caches. Such applications could favor SMT when memory latencies are high. However,

as explained earlier, the relative advantage of SMT will be small if the amount of ungated circuitry is small.

Chapter 7

Related Work

This chapter describes the related work in detail. The related work for each of the three main contributions of this dissertation is discussed in a separate section.

7.1 Application Analysis

There have been many studies that characterize the individual applications used in this dissertation.

Several papers characterize MPEG-2. Chen et al [12, 26] characterize various phases of MPGdec on a real system and discuss and evaluate slice assignment policies, and data vs. functional partitioning for parallelization. However, they do not perform a thread-scaling or a frequency scaling study. Two widely used benchmark suites, MediaBench [48] and Berkeley multimedia workload [70] also include MPEG-2 encoder and decoder. Our applications differ from the above benchmarks since we expose *parallelism* in our applications using POSIX threads and SSE2 SIMD instructions. We also modified MPEG encoder to use an intelligent motion search algorithm and to use an optimized algorithm for discrete cosine transform. Iwata et al. [36] propose a number of coarse-grained parallel implementations of MPEG-2 decoding and encoding. They evaluate the performance of these implementations on a multiprocessor, compare the performance against a single and wide issue superscalar processor, and report results with multi-threading for 4 and 8 processors. They also find that thread scalability of MPGenc is better than that for MPGdec. However, they report 8 thread results only with a single issue processor. We do the TLP scalability study up to 16 threads using the same processor configuration. They also report 50% speedup with MIPS based SIMD instructions for MPGdec. We report SSE2 speedups for both MPGenc and MPGdec; we are

able to achieve much higher speedups (2X) with SSE2 for MPGdec. We also perform a frequency scalability study.

Turk et al. [76] discuss the theoretical underpinnings of the face recognition algorithms and Beveridge et al. [6] describe the CSU face identification software used with this study. Mathew et al. [54] characterize the features of Eigenfaces face recognition algorithm used in this study. They characterize the architectural features such as cache hit rates, IPC on several embedded architectures. In addition to these characterization, we analyze the thread and SIMD parallelism of this application. Vorbruggen [77] describes a similar face recognition algorithm used with SPEC CPU2000 but does perform an evaluation.

Ravishankar [60] describes the algorithms, data structures, inputs/outputs of Sphinx 3.3 used with this study. Mathew et al. [55] provide a detailed analysis of CMU's Sphinx speech recognizer; they identify the three distinct processing phases (Section 2.2.5), and quantify the architectural requirements for each phase. They also describe the large memory footprint and find the Gaussian and search phases to be the dominant ones. They also developed a parallel version of Sphinx that runs 3 major phases (i.e., feature recognition, Gaussian scoring, and search) using three threads and report a 1.67 speedup. Instead of this type of functional partitioning, we parallelize Sphinx3.3 using data partitioning (i.e., phases are divided into N symmetric threads). This method gives better speedups and is more scalable. They also develop a special-purpose accelerator for the dominant Gaussian scoring phase. The accelerator consists of specialized multipliers and adders to perform the specific multiply accumulate operation done in the inner loop of Gaussian scoring. To overcome the latency of FP multiply accumulate operations, they pipeline multiple independent iterations. Instead of using a separate co-processor, we use the SIMD units to exploit the DLP in the Gaussian scoring phase.

Baugh et al. characterize and parallelize Sphinx2 [4][1]. They divide Sphinx into multiple phases and use work queues in between phases. Then they use asymmetric threads to execute each phase. They also investigate using symmetric threads within each phase. In contrast, we use symmetric threads that span both Gaussian scoring and search phases and do not use work queues. They report speedups up to 2.7X using both asymmetric and symmetric threads (a total of 6 threads).

[1]Sphinx2 has somewhat different Gaussian models than those used in Sphinx3.3 [47].

They also show preliminary results where they achieve speedups up to 6.8X with 10 threads. They do not investigate exploiting DLP in this study.

Krishna et al. [46] analyze parameters affecting the performance of Sphinx2 speech recognition software with special emphasis on the memory system. They also find poor cache performance (Figure 2.3), poor memory reference predictability, and potential for using multiple threads albeit with higher demands on the memory system. Based on the insights from that work, they propose architectural SMT techniques to exploit the TLP in Sphinx [47]. They develop an architecture with multiple speech processing elements that are capable of generating their own threads and report good speedups (e.g., approximately 12X speedup with 16 speech processing elements and 4 thread contexts per processing element). They also perform partitioning of search tree nodes; for thread creation and synchronization, they use a fork/join model with a special barrier instruction (called EPOCH) and locks. We use a similar approach to exploit TLP but also investigate the DLP from sub-word SIMD.

Woo et al. [80] present a characterization of a different version of RayTrace with other applications introduced with the SPLASH-2 benchmark. They also report working set characteristics similar to those reported here and also report good TLP scalability. However, they do not study the scalability of RayTrace with frequency. Nguyen et al. [57] also characterize RayTrace provided with the SPLASH as a part of their work in evaluating multi-processor scheduling policies. They also study the TLP scalability and identify the sources of speedup loss. However, they do not study working sets or frequency scalability of this application.

In summary, this work is different from each of the above works in one or more of the following ways. First, we concentrate on studying the *parallelism* in these applications. Specifically, we characterize ILP, TLP, and DLP and also study the interaction between two forms of parallelism. Second, we look at the complex media applications as a benchmark suite and attempt to identify the features and parallelism common to all of them. Third, we do this study in the context of general-purpose CMP processors.

7.2 ALP

There is a vast amount of literature on conventional vector architectures and their recent uniprocessor and multiprocessor variations; e.g., VIRAM [43], CODE [44], Tarantula [19], T0 [3], out-of-order vectors [21], MOM [14], SMT Vectors [20], NEC SX [41], Cray X1 [15], and Hitachi SR [81]. Such systems require investment in a relatively large special-purpose dedicated vector unit (e.g., the Tarantula vector unit is the same size as the 4-thread scalar core [19]). In return, they provide excellent performance and energy efficiency for medium-grain regular DLP; e.g., through multi-laned implementations.

However, our applications mostly exhibit small-grain DLP interspersed with control and reductions, and have parts that do not exhibit any DLP at all (Section 2.2). We therefore chose to explore a lighter weight DLP mechanism, SVectors/SStreams, that could be tightly integrated into expected GPP designs that already have superscalar cores, CMP, SMT, and SIMD. Our results show that the resulting architecture, ALP, is effective in exploiting the different levels of parallelism in complex media applications, and SVectors/SStreams in particular show significant benefits over ALP's other enhancements. Further, ALP does so primarily using existing data paths and storage with only modest modifications to a conventional superscalar core. Thus, we believe that this dissertation has demonstrated a valuable design point between pure SIMD and conventional vectors.

Nevertheless, conventional vectors would have the advantage of reduced dynamic compute instructions (ALP uses SIMD compute instructions, which encode only 2-16 operations in each instruction). Further, it may (or may not) be possible that significant algorithmic changes to our applications can expose larger grain DLP for which conventional vectors would be well suited. We are currently performing a detailed quantitative comparison between ALP and a representative vector machine (Tarantula) – reporting the results of such a study is outside the scope of this dissertation.

The Imagine architecture [1] and its multiprocessor version, Merrimac [16] are also motivated by support for large amounts of DLP, specifically streams. ALP's focus on small-grain DLP and the constraint of integration within a GPP results in significant design differences. Specifically, (i) for computation, Imagine provides ALU clusters that work in lockstep while ALP uses independent

SIMD units to exploit ILP and TLP along with fine-grain DLP; (ii) Imagine is designed as a co-processor that depends on a scalar host for irregular scalar computation while ALP's DLP support is tightly integrated into the superscalar core; and (iii) unlike ALP, Imagine needs a substantially new programming model to manipulate streams. ALP and Imagine share similarities in the handling of data – ALP's combination of the SIMD register file and SVRs is analogous to Imagine's storage hierarchy with a local register file for intermediate computation and a stream register file for stream data. Results for Imagine report 138 frames per second throughput for MPEG2 encoding in 360x288 resolution at 200MHz [1]. However, these results do not include B frame encoding, Huffman variable length encoding (VLC) or half pixel motion search that are part of the full application. At 200 MHz, ALP encodes 98 352x288 resolution frames per second,[2] while processing B frames, Huffman VLC, and half-pixel motion estimation. B frames usually take at least twice the time of I frames and about 30% more time than P frames, and constitute of 2/3 of all frames. Half-pel motion search adds more than 30% execution time even to the 4x2T+SV system for ALP. Huffman VLC is serial code with no DLP instructions and can be sped up only with thread support. Considering all of the above, we conclude that the performance of ALP is competitive with that of Imagine. Regarding energy, at 200 MHz using 90nm parameters, we obtain much better energy than Imagine reports. However, due to variations in technology, gating, and assumptions in Wattch, we do not feel a fair energy comparison can be performed at this time.

A few architectures like SCALE [45], Pseudo Vector Machine (PVM) [49], conditional streams [39] of Imagine, and Titan [38] cater to fine-grain DLP. SCALE combines TLP and vectors in a concept called vector-thread architectures, which uses a control processor along with a vector of virtual processors. It can exploit DLP interspersed with control; however, it uses a new programming model while ALP extends the established GPP programming model. So far, SCALE has been evaluated primarily for kernels; a comparison with ALP on complex applications is therefore difficult.

PVM provides support for vector/stream-like processing of loops that are difficult to vectorize. Two source vectors are associated with two registers. A compute instruction accessing such a register implicitly accesses the next element of the associated vector. The PVM implementation does

[2]We could not find 360x288 resolution video streams - we do not expect the difference of 8 rows in the stream to be significant.

not support a cache hierarchy, and all vector data accessed by compute instructions is transferred from memory space. This shares similarity with SVectors/SStreams, but has some key differences. Our SVectors use vector load instructions to bring data into the SVR in a pipelined way, and enable preloading of data. Any data that is spilled from the SVRs is held in the L2 cache for some time. In contrast, PVM supports a fast scratchpad memory space, somewhat analogous to our SVR. However, there are no vector load instructions to bring data into this space; data can be moved to scratchpad only through functional units using explicit instructions.

Conditional streams provide limited fine grain DLP support for Imagine – they allow different operations on different records on a stream. However, conditional streams change the order of the resulting records.

Titan uses a different approach to cater to DLP interspersed with control. It uses successive *scalar* FP registers to store a vector allowing individual vector elements to be accessed. All compute instructions are vector instructions and scalar operations have a length of 1. It is difficult to map such a design on to current renamed/out-of-order cores.

At a high level, SVectors exploit two dimensional DLP as done by traditional SIMD array processors [29], MOM [14, 63], and CSI [13]. This is because SVectors are in turn composed of small vectors (SIMD). However, unlike ALP, MOM uses vector/matrix instructions for computation and uses a large matrix register file. Similarly, unlike ALP, CSI uses a memory to memory stream architecture with a seperate pipeline for streams.

Several architectures like Smart Memories [53], TRIPS [64], and RAW [73] support multiple forms of parallelism. Instead of supporting a DLP based programming model like vectors/streams in the ISA, these architectures support efficient mapping/scheduling of multiple instructions that work on independent data and schedule communication among them. For example, TRIPS' main support for DLP consists of rescheduling loop iterations for computation without requiring prefetching and other repeated front end overhead (called revitalization). RAW allows direct accessing of operands from the network, eliminating some explicit loads. SmartMemories can morph memories into many structures; ALP uses a more restricted type of morphing cache for SVRs. Unlike ALP, both Smart Memories and TRIPS require switching to a different mode to support DLP (resulting in mode changes between different parts of an application). Unlike ALP, both RAW and Smart Memories

expose underlying hardware details and communication to the programming model.

Several mechanisms enhance the memory system to support DLP. Impulse [82] augments the memory controller to map non-contiguous memory to contiguous locations. Processor in memory architectures like DIVA [18] increase memory bandwidth and decrease memory latency. Some DSP processors, as well as TRIPS, support software managed caches or scratchpad memories which usually need explicit loads/stores to be accessed. To reduce loads/stores, they support memory to memory addressing modes and DMA. SVRs achieve similar benefits without loads/stores.

To support regular computation, DSP and media processors include indexed addressing modes with auto-incrementing, loop repetition, and/or rotating registers. ALP achieves similar benefits with the unified mechanism of SVectors/SStreams.

Itanium [32], Cydra 5 [59], and Hitachi SR8000 [81] use rotating registers to hold data elements that are accessed sequentially. Rotating registers are used to provide different registers for different instances (in different loop iterations) of the same variable. In out-of-order processors, renaming provides the same functionality albeit at a higher hardware cost. Rotating registers, which are usually a part of the general purpose register file, are loaded with scalar load instructions. In contrast, SVectors use vector loads to bring a sequence of data records into the data arrays of reconfigured L1 cache. Further, rotating registers can hold variables that are accessed only within a given iteration. Therefore, unlike SVRs, such registers cannot store more permanent state (e.g., a table that is used many times or a variable used across iterations). SVectors do not have such limitations – i.e., SVectors can be loaded in advance and used repeatedly.

7.3 Comparison of CMP and SMT

Although there is significant prior work comparing the performance of CMP and SMT [74, 24], comparing the energy efficiency of SMT with a superscalar [68], and comparing the area and layout overheads of SMT and CMP [11], there is very little prior work on energy related comparisons of SMT and CMP. The only such work, to our knowledge, is by Kaxiras et al., also in the context of multimedia applications [40]. However, that work considers a VLIW processor core (which produces low IPC for the compiled codes studied) and primarily compares average power at a given frequency for CMP vs. SMT. It examines only two alternative core architectures and only one

workload. In contrast, we study out-of-order superscalar processors (which give higher IPCs) and compare energy at the same performance, for a wider range of workloads and core processor architectures. The primary conclusion from the work by Kaxiras et al. is that at equal frequency, CMP consumes more power than SMT. This does not contradict our results since we also find that CMP configurations have higher maximum and average power than corresponding SMT configurations at a fixed (highest) frequency (e.g., leftmost points of Figure 6.3 and Figure 6.4).

Kaxiras et al. perform one comparison at equal performance – the lowest frequency for SMT and CMP at which the real-time constraint of the workload is met. At this performance, they observe that the CMP can run at a lower frequency than SMT, again consistent with our observations. However, they find that SMT still consumes lower power than CMP except for some idealized conditions such as with 100% clock gating and no global clock network. While we al so show the sensitivity of CMP to clock gating and also report higher power in CMP's clock network relative to that of SMT, we find that CMP is more efficient even at the lowest performance points we studied (at reasonable clock gating levels and accounting for realistic clock network power). This is because we study a wider range of architectures, enabling the use of less aggressive architectures to improve the efficiency of CMP. Nevertheless, at the lowest performance points, the difference between CMP and SMT is relatively small.

There is recent work published after [65] comparing the energy and thermal efficiency of CMP and SMT processors for multi-programmed SPEC2000 workloads consisting of two applications [51, 52]. That work also finds CMP is more energy efficient for compute bound workloads. As predicted by our analysis, they also find SMT is more energy efficient for memory bound workloads when the working set does not fit in L2. They also investigate the heating patterns for CMP and SMT processors and find that CMP heats up mainly due to the global energy consumption whereas SMT exhibits more localized heating in several key resources.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The objective of this work is to provide energy efficient performance for contemporary complex media applications on general-purpose processors. We identify parallelism as the key opportunity presented by these applications and investigate means of exploiting multiple types of parallelism present in them to provide both high performance and energy efficiency.

This dissertation makes three broad contributions. First, we analyze the parallelism in complex media applications and make the case that contemporary media applications require efficient support for multiple types of parallelism. Second, we compare alternatives for supporting energy efficient TLP for media workloads on general-purpose processors. Finally, based on the results of the above two, we propose a complete architecture, called ALP, that effectively supports *a*ll *l*evels of *p*arallelism described above in an energy efficient way, using an *evolutionary programming model and hardware*.

From our study of parallelism, first, we observe that the complexity of contemporary media applications requires support for multiple forms of parallelism, including ILP, TLP, and various forms of data-level parallelism such as sub-word SIMD, short vectors, and streams. Second, we find that all our applications have coarse-grain TLP and fine-grain DLP. Third, we observe that the DLP computations are often followed by reductions. This could reduce the effectiveness of conventional multi-laned DLP architectures that are not optimized for reductions. Fourth, we observe that these applications have a low computation to memory ratio requiring to support memory operations effectively.

To find the most energy efficient way of exploiting TLP, we perform a comparison between CMP and SMT. We perform this comparison for a large number of performance points derived using different processor architectures and frequencies/voltages. From this study, we find that, at equal performance, CMP to be more energy efficient than SMT, especially when supporting four or more threads. We also find that the best SMT and the best CMP configuration for a given performance target have different architecture and frequency/voltage. From our analysis, we find that the relative energy efficiency depends on a subtle interplay between various factors such as capacitance, voltage, IPC, frequency, and the level of clock gating, as well as workload features. Although CMP shows a clear energy advantage for four-thread (and higher) workloads, it comes at the cost of increased silicon area. We therefore investigate a hybrid solution where a CMP is built out of SMT cores, and find it to be an effective compromise. We use this architecture as the basis for providing ILP and TLP in ALP.

Based on our findings from the above studies, we propose ALP, which can cater to multiple forms of parallelism effectively. The novel part of ALP is a DLP technique called *SIMD vectors and streams*, which is integrated within a conventional superscalar based CMP/SMT architecture with sub-word SIMD. This technique lies between sub-word SIMD and vectors, providing significant benefits over the former at a lower cost than the latter. Our evaluations show that each form of parallelism supported by ALP is important. Specifically, SIMD vectors and SIMD streams are effective – compared to a system with the other enhancements in ALP, they give speedups of 1.1X to 3.4X and energy-delay product improvements of 1.1X to 5.1X for applications with DLP.

The results of this thesis are applicable to the applications with properties described in Section 1.2.1. More broadly, our results show that conventional architectures augmented with evolutionary mechanisms can provide high performance and energy savings for our complex media applications without resorting to radically different architectures and programming paradigms.

8.2 Future Work

There are several directions for future work. The following sections give overviews of each.

8.2.1 Effect of Real-time Scheduling

Since temporal correctness is important for multimedia applications, they are often run on real-time operating systems with the help of a real-time scheduler. Real-time scheduling of multiple threads in a single application or multiple applications in a workload can affect both performance and energy efficiency of a given system, including ALP. Further, real-time scheduling can effect the relative energy efficiency of CMP and SMT processors. Primarily, real-time scheduling affects the completion time of an application. Which threads/processes are run concurrently can effect cache behavior, synchronization, and symbiosis (effectiveness of resource sharing) of CMP and SMT processors. Consequently, it is important to study how real-time scheduling will affect ALP and the relative performance/energy efficiency of CMP and SMT processors.

8.2.2 Adaptations

Adaptation of components in a system, including the processor, memory, operating system, and application itself can have an impact on the performance and energy efficiency of a system. In this dissertation, we did not investigate how such adaptations could affect ALP or the relative energy efficiency between CMP and SMT processors. ALP has new opportunities for adaptation due to its design. First, ALP can support dynamic frequency/voltage scaling. Second, ALP can support separate frequencies/voltages for each core. Third, ALP can make use of its clustered design to adapt for low IPC threads. Since adaptations can lead to substantial energy savings, it is useful to investigate how adaptations will affect ALP and the relative performance and energy efficiency of CMP and SMT processors.

8.2.3 Comparison of SVectors/SStreams with Conventional Vectors

In this work, we show that SVectors and SStreams lead to substantial performance and energy improvements over SIMD. However, how SVectors/SStreams would perform with respect to conventional vector implementations is not clear. Several of our applications exhibit properties like short vector lengths (small-grain DLP) and frequent reductions that are not beneficial to conventional multi-lane vector implementations. On the other hand, face recognition shows very long vector lengths and can benefit from vector implementations. Although, a separate vector adds

additional computation capability to a scalar core, the additional area and the energy overhead necessary could affect vector implementations negatively. Therefore, a thorough investigation is necessary to compare SVectors/SStreams with a conventional vector implementation.

8.2.4 Applications and Workloads

Existing systems and future systems will run both multi-threaded multimedia applications (e.g., video decoding) and heterogenous mixes of multimedia applications (e.g., video conferencing with multiple parties). Therefore, the evaluation of both multi-threaded applications and heterogenous workloads is important to assess the performance and energy consumption of real systems. Currently, we evaluate ALP using multi-threaded applications and perform the comparison of energy efficiency for CMP and SMT processors using heterogenous multimedia workloads. For the reasons given above, it is important to extend both studies to use multi-threaded applications and heterogenous multimedia workloads.

Further, to reinforce our findings, it is useful to evaluate other emerging multimedia applications on ALP. For instance, we can investigate media applications like radiosity, beam-forming, and multimedia search/mining, which may become popular in the future.

8.2.5 Design of Memory System

Since ALP can support quite different memory hierarchies from shared L2 caches to private L2 caches, a study exploring the design space for memory systems can result in memory hierarchies that are better suited for complex media applications. For instance, threads in some media applications and multiple applications in a heterogenous workload could share data among threads or processes, respectively. We could design cache hierarchies to facilitate such sharing. For instance, we could investigate policies for duplicating data and algorithms for data movement among banks or private L2 caches to lower the access latency.

References

[1] Jung Ho Ahn, William J. Dally, Brucek Khailany, Ujval J. Kapasi, and Abhishek Das. Evaluating the Imagine Stream Architecture. In *Proc. of the 31th Annual Intl. Symp. on Comp. Architecture*, 2004.

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

[3] Krste Asanovic. *Vector Microprocessors*. PhD thesis, Univ. of California at Berkeley, 1998.

[4] Lee Baugh, Jose Renau, James Tuck, and Josep Torrellas. Sphinx Parallelization. Technical Report UIUCDCS-R-2002-2606, Dept. of Computer Science, University of Illinois, 2002.

[5] Bob Bentley and Rand Gray. Validating The Intel Pentium 4 Processor: Power Reduction Validation. *Intel Technology Journal, Q1*, 2001.

[6] Ross Beveridge, David Bolme, Marcio Teixeira, and Bruce Draper. The CSU Face Identification Evaluation System User's Guide: Version 5.0. In *www.cs.colostate.edu/evalfacerec/algorithms/version5/faceIdUsersGuide.pdf*, 2003.

[7] Ross Beveridge and Bruce Draper. Evaluation of Face Recognition Algorithms. *http://www.cs.colostate.edu/evalfacerec/*, 2003.

[8] Shekhar Y. Borkar. Designing for power. *http://www.intel.com/labs/features/mi04031.htm?iid=labs+mi04031.htm*, 2003.

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

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

[11] James Burns and Jean-Luc Gaudiot. Area and System Clock Effects on SMT/CMP Processors. In *Proc. of the 2nd Parallel Architectures and Compilation Techniques*, 2000.

[12] Yen-Kuang Chen et al. Media Applications on Hyper-Threading Technology. *Intel Technology Journal, Vol.6, Issue 1*, 2002.

[13] Dmitry Cheresiz, Ben H. H. Juurlink, Stamatis Vassiliadis, and Harry A. G. Wijshoff. The CSI Multimedia Architecture. *IEEE Trans. VLSI Syst.*, 13(1), 2005.

[14] Jesus Corbal, Roger Espasa, and Mateo Valero. MOM: A Matrix SIMD Instruction Set Architecture for Multimedia Applications. In *Proc. of the 14th Intl. Conf. on Supercomputing*, 1999.

[15] Cray Inc. Cray X1 System Overview. *www.cray.com*, 2005.

[16] William J. Dally, P. Hanrahan, M. Erez, et al. Merrimac: Supercomputing with Streams. In *Proc. of 2003 ACM/IEEE conference on Supercomputing*, 2003.

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

[18] Jeff Draper, Jacqueline Chame, Mary Hall, et al. The architecture of the diva processing-in-memory chip. In *Proc. of the 17th Intl. Conf. on Supercomputing*, 2002.

[19] Roger Espasa, Federico Ardanaz, Joel Emer, et al. Tarantula: A Vector Extension to the Alpha Architecture. In *Proc. of the 29th Annual Intl. Symp. on Comp. Architecture*, 2002.

[20] Roger Espasa and Mateo Valero. Simultaneous multithreaded vector architecture. In *Proc. of the 3rd Intl. Symp. on High-Perf. Comp. Architecture*, 1997.

[21] Roger Espasa, Mateo Valero, and James E. Smith. Out-of-order vector architectures. In *Proc. of the 25th Annual Intl. Symp. on Comp. Architecture*, 1997.

[22] Ricardo Gonzalez and Mark Horowitz. Energy Dissipation In General Purpose Microprocessors. In *IEEE Journal of Solid-state Circuits*, September 1996.

[23] J. Griswell, B. Sinharoy, R. Eickemeyer, and W. El-Essawy. Using a Performance Model to Estimate Core Clock Gating Power Savings. In *Workshop on Complexity-Effective Design*, 2002.

[24] Lance Hammond, Basem A. Nayfeh, and Kunle Olukotun. A Single-Chip Multiprocessor. In *IEEE Computer Special Issue on Billion-Transistor Processors*, September 1997.

[25] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[26] M. Holliman and Y.-K. Chen. MPEG Decoding Workload Characterization. In *Proc. of Workshop on Computer Architecture Evaluation using Commercial Workloads*, 2003.

[27] Christopher J. Hughes, P. Kaul, Sarita V. Adve, et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. In *Proc. of the 28th Annual Intl. Symp. on Comp. Architecture*, 2001.

[28] Christopher J. Hughes, Vijay S. Pai, Parthasarathy Ranganathan, and Sarita V. Adve. RSIM: Simulating Shared-Memory Multiprocessors with ILP Processors. *IEEE Computer*, February 2002.

[29] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, programmability.* McGraw-Hill Inc., 1993.

[30] Intel Corporation. *Intel Application Notes AP-922*, 1999.

[31] Intel Corporation. *Intel Application Notes AP-945*, 1999.

[32] Intel Corporation. *Intel Itanium Architecture Software Developer's Manual*, 2001.

[33] Intel Corporation. *Intel Pentium-M Processor Datasheet*, 2003.

[34] Intel Corporation. *The IA-32 Intel Architecture Optimization Reference Manual*, 2004.

[35] Intel Corporation. *Intel 925XE Express Chipset, http://www.intel.com/products/chipsets/*, 2005.

[36] Eiji Iwata and Kunle Olukotun. Exploiting Coarse-Grain Parallelism in the MPEG-2 Algorithm. Technical Report CSL-TR-98-771, Computer Systems Lab, Stanford University, 1998.

[37] Rohit Jain, Chirstopher J. Hughes, and Sarita V. Adve. Soft Real-Time Scheduling on Simultaneous Multithreaded Processors. In *Proc. of the 23rd IEEE Intl. Real-Time Systems Symp.*, 2002.

[38] Norman P. Jouppi. A Unified Vector/Scalar Floating-Point Architecture. In *Proc. of the 8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 1989.

[39] Ujval J. Kapsai, William J. Dally, Scott Rixner, et al. Efficient Conditional Operations for Data-parallel Architectures. In *Proc. of the 36rd Annual Intl. Symp. on Microarchitecture*, 2003.

[40] Stefanos Kaxiras, Girija Narlikar, Alan D. Berenbaum, and Zhigang Hu. Comparing Power Consumption of an SMT and a CMP DSP for Mobile Phone Workloads. In *Proc. of the Intl. Conf. on Compilers, Architectures and Synthesis for Embedded Systems*, 2001.

[41] K. Kitagawa, S. Tagaya, Y. Hagihara, and Y. Kanoh. A Hardware Overview of SX-6 and SX-7 Supercomputer. *http://www.nec.co.jp/techrep/en/r_and_d/r03/r03-no1/rd02.pdf*, 2002.

[42] K. I. T. Koga, A. Hirano, Y. Iijima, and T. Ishiguro. Motion-Compensated Interframe Coding for Video Conferencing. In *Proc. of the 1981 National Telesystems Conference*, 1981.

[43] Christos Kozyrakis. *Scalable Vector Media Processors for Embedded Systems.* PhD thesis, Univ. of California at Berkeley, 2002.

[44] Christos Kozyrakis and David Patterson. Overcoming the Limitations of Conventional Vector Processors. In *Proc. of the 30th Annual Intl. Symp. on Comp. Architecture*, 2003.

[45] Ronny Krashinsky, Christopher Batten, Mark Hampton, et al. The Vector-Thread Architecture. In *Proc. of the 31th Annual Intl. Symp. on Comp. Architecture*, 2004.

[46] R. Krishna, S. Mahlke, and T. Austin. Insights Into the Memory Demands of Speech Recognition Algorithms. . In *Proc. of the 2nd Annual Workshop on Memory Performance Issues*, 2002.

[47] Rajeev Krishna, Scott Mahlke, and Todd Austin. Architectural Optimizations for Low-Power, Real-time Speech Recognition. In *Proc. of the Intl. Conf. on Compilers, Architectures and Synthesis for Embedded Systems.* ACM Press, 2003.

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

[49] Lea Hwang Lee. *Pseudo-Vector Machine for Embedded Applications.* PhD thesis, University of Michigan, 2000.

[50] Man-Lap Li, Ruchira Sasanka, Sarita V. Adve, Yen-Kuang Chen, and Eric Debes. The ALP-Bench Benchmark Suite for Multimedia Applications (Submitted for publication). Technical Report UIUCDCS-R-2005-2603, Dept. of Computer Science, University of Illinois, July 2005.

[51] Yingmin Li, David Brooks, Zhigang Hu, Kevin Skadron, and Pradip Bose. Understanding the Energy Efficiency of Simultaneous Multithreading. In *Proc. of the 2004 Intl. Symp. on Low Power Electronics and Design*, 2004.

[52] Yingmin Li, David Brooks, Zhigang Huy, and Kevin Skadron. Performance, Energy, and Thermal Considerations for SMT and CMP Architectures. In *Proc. of the 11th Intl. Symp. on High-Perf. Comp. Architecture*, 2005.

[53] Ken Mai, Tim Paaske, Nuwan Jayasena, Ron Ho, et al. Smart Memories: A Modular Reconfigurable Architecture. In *Proc. of the 27th Annual Intl. Symp. on Comp. Architecture*, 2000.

[54] Binu Mathew, Al Davis, and Robert Evans. A Characterization of Visual Feature Recognition. Technical Report UUCS-03-014, University of Utah, 2003.

[55] Binu Mathew, Al Davis, and Zhen Fang. A Low-Power Accelerator for the SPHINX 3 Speech Recognition System. In *Proc. of the Intl. Conf. on Compilers, Architecture and Synthesis for Embedded Systems*, 2003.

[56] MPEG Software Simulation Group. MSSG MPEG2 encoder and decoder. *http://www.mpeg.org/MPEG/MSSG/*, 1994.

[57] Thu D. Nguyen, Raj Vaswani, and John Zahorjan. Parallel Application Characterization for Multiprocessor Scheduling Policy Design. In *Job Scheduling Strategies for Parallel Processing, Volume 1162 of Lecture Notes in Computer Science, Springer-Verlag*, 1996.

[58] Parthasarathy Ranganathan, Sarita Adve, and Norman P. Jouppi. Reconfigurable Caches and their Application to Media Processing. In *Proc. of the 27th Annual Intl. Symp. on Comp. Architecture*, 2000.

[59] B. Ramakrishna Rau, David W. L. Yen, Wei Yen, and Ross A. Towie. The Cydra 5 Departmental Supercomputer: Design Philosophies, Decisions, and Trade-Offs. In *IEEE Computer*, 1989.

[60] Mosur K. Ravishankar. Sphinx-3 s3.X Decoder. *http://cmusphinx.sourceforge.net/sphinx3/*, 2004.

[61] Raj Reddy et al. CMU SPHINX. *http://www.speech.cs.cmu.edu/sphinx/*, 2001.

[62] Mendel Rosenblum, Edouard Bugnion, and Stephen Alan Herrod. Vector Vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks. In *Proc. of 20th ACM Symp. on Operanting Systems Principles*, 1995.

[63] Friman Sanchez, Mauricio Alvarez, Esther Salam, Alex Ramirez, and Mateo Valero. On the Scalability of 1- and 2-Dimensional SIMD Extensions for Multimedia Applications. In *Proc. of IEEE Intl. Symp. on Performance Analysis of Systems and Software*, 2005.

[64] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, et al. Exploiting ILP, TLP, and DLP with the Polymorphous TRIPS Architecture. In *Proc. of the 30th Annual Intl. Symp. on Comp. Architecture*, 2003.

[65] Ruchira Sasanka, Sarita V. Adve, Yen-Kuang Chen, and Eric Debes. The Energy Efficiency of CMP vs. SMT for Multimedia Workloads. In *Proc. of the 20th Intl. Conf. on Supercomputing*, 2004.

[66] Ruchira Sasanka, Christopher J. Hughes, and Sarita V. Adve. Joint Local and Global Hardware Adaptations for Energy. In *Proc. of the 8th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2002.

[67] Greg Semeraro et al. Energy-Efficient Processor Design Using Multiple Clock Domains with Dynamic Voltage and Frequency Scaling. In *Proc. of the 8th Intl. Symp. on High Performance Comp. Architecture*, 2002.

[68] John Seng, Dean Tullsen, and George Z. N. Cai. Power-Sensitive Multithreaded Architecture. In *Proc. of the 7th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 1996.

[69] Kevin Skadron, Tarek Abdelzaher, and Mircea R. Stan. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *Proc. of the 8th Intl. Symp. on High-Perf. Comp. Architecture*, 2002.

[70] Nathan T. Slingerland and Alan Jay Smith. Design and Characterization of the Berkeley Multimedia Workload. *Multimedia Syst.*, 8(4), 2002.

[71] A. Snavely and D. Tullsen. Symbiotic Job Scheduling for a Simultaneous Multithreading Architecture. In *Proc. of the 9th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2000.

[72] John E. Stone. Taychon Raytracer. *http://jedi.ks.uiuc.edu/ johns/raytracer/*, 2003.

[73] Michael Taylor, Walter Lee, Jason Miller, David Wentzlaff, et al. Evaluation of the RAW Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *Proc. of the 31th Annual Intl. Symp. on Comp. Architecture*, 2004.

[74] Dean Tullsen, Susan Eggers, Joel Emer, Henry Levy, Jack Lo, and Rebecca Stamm. Converting Thread-Level Parallelism Into Instruction-Level Parallelism via Simultaneous Multithreading. In *ACM Transactions on Computer Systems*, 1997.

[75] Dean Tullsen et al. Exploiting Choice: Instruction Fetch and Issue on an Implementable Simultaneous Multithreading Processor. In *Proc. of the 23th Annual Intl. Symp. on Comp. Architecture*, 1996.

[76] Matthew Turk and Alex Pentland. Face Recognition Using Eigenfaces. In *Journal of Cognitive Neuroscience, Vol. 3*, 1991.

[77] Jan C. Vorbruggen. 187.facerec: CFP2000 Benchmark Description. *http://www.spec.org/osg/cpu2000/CFP2000/*, 2000.

[78] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and Sharad Malik. Orion: A Power-Performance Simulator for Interconnection Networks. In *Proc. of the 35rd Annual Intl. Symp. on Microarchitecture*, 2002.

[79] Z. Wang. Fast Algorithms for the Discrete Cosine Transform and for the Discrete Fourier Transform. In *IEEE Transactions in Acoustics, Speech, and Signal Processing. Vol. ASSP-32*, 1984.

[80] Steven Cameron Woo et al. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proc. of the 22th Annual Intl. Symp. on Comp. Architecture*, 1995.

[81] Y.Tamaki, Naonobu Sukegawa, Masanao Ito, et al. Node Architecture and Performance Evaluation of the Hitachi Super Technical Server SR8000. In *Proc. of the 11th Intl. Conf. on Parallel and Distributed Systems*, 1999.

[82] Lixin Zhang, Zhen Fang, Mide Parker, Binu K. Mathew, et al. The Impulse Memory Controller. In *IEEE Transcations on Computers*, 2001.

[83] Y. Zhang, D. Parikh, K. Sankaranarayanan, et al. HotLeakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects. Technical Report CS-2003-05, Univ. of Virginia, 2003.

Author's Biography

Ruchira Sasanka was born in Kalutara, Sri Lanka. He attended high school at the Royal College, Colombo, Sri Lanka. He received his Bachelor of Science degree in Computer Science and Engineering from the University of Moratuwa, Sri Lanka in 1998 and the Master of Science degree in Computer Science from the University of Illinois at Urbana-Champaign in 2002.