

# Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions

Parthasarathy Ranganathan\*, Sarita Adve\*, and Norman P. Jouppi†

\*Electrical and Computer Engineering  
Rice University  
{parthas,sarita}@rice.edu

†Western Research Laboratory  
Compaq Computer Corporation  
jouppi@pa.dec.com

## Abstract

*This paper aims to provide a quantitative understanding of the performance of image and video processing applications on general-purpose processors, without and with media ISA extensions. We use detailed simulation of 12 benchmarks to study the effectiveness of current architectural features and identify future challenges for these workloads.*

*Our results show that conventional techniques in current processors to enhance instruction-level parallelism (ILP) provide a factor of 2.3X to 4.2X performance improvement. The Sun VIS media ISA extensions provide an additional 1.1X to 4.2X performance improvement. The ILP features and media ISA extensions significantly reduce the CPU component of execution time, making 5 of the image processing benchmarks memory-bound.*

*The memory behavior of our benchmarks is characterized by large working sets and streaming data accesses. Increasing the cache size has no impact on 8 of the benchmarks. The remaining benchmarks require relatively large cache sizes (dependent on the display sizes) to exploit data reuse, but derive less than 1.2X performance benefits with the larger caches. Software prefetching provides 1.4X to 2.5X performance improvement in the image processing benchmarks where memory is a significant problem. With the addition of software prefetching, all our benchmarks revert to being compute-bound.*

## 1 Introduction

In the near future, *media processing* is expected to become one of the dominant computing workloads [6, 13].

\*This work is supported in part by an IBM Partnership award, Intel Corporation, the National Science Foundation under Grant No. CCR-9410457, CCR-9502500, CDA-9502791, and CDA-9617383, and the Texas Advanced Technology Program under Grant No. 003604-025. Sarita Adve is also supported by an Alfred P. Sloan Research Fellowship.

Media processing refers to the computing required for the creation, encoding/decoding, processing, display, and communication of digital multimedia information such as images, audio, video, and graphics. The last few years have seen significant advances in this area, but the true promise of media processing will be seen only when applications such as collaborative conferencing, distance learning, and high-quality media-rich content channels appear in ubiquitously available commodity systems. Further out, advanced human-computer interfaces, telepresence, and immersive and interactive virtual environments hold even greater promise.

One obstacle in achieving this promise is the high computational demands imposed by these applications. These requirements arise from the computationally expensive nature of the algorithms, the stringent real-time constraints, and the need to run many such tightly synchronized applications at the same time on the same system. For example, a video teleconferencing system may need to run video processing including encoding/decoding, audio processing, and a software modem simultaneously. As a result, such applications currently display images of only a few square inches at a few frames per second when running on general-purpose processors. Full-screen images at 20-30 frames per second could require more than two orders of magnitude more performance.

To meet the high computational requirements of emerging media applications, current systems use a combination of general-purpose processors accelerated with DSP (or media) processors and ASICs performing specialized computations. However, benefits offered by general-purpose processors in terms of ease of programming, higher performance growth, easier upgrade paths between generations, and cost considerations argue for increasing use of general-purpose processors for media processing applications [6, 13]. The most visible evidence of this trend has been the SIMD-style media instruction-set architecture (ISA) ex-

tensions announced for most high-performance general-purpose processors (e.g., 3DNow! [15], AltiVec [19], MAX [12], MDMX and MIPSV [9], MMX [18], MVI [4], VIS [23]).

Unfortunately, in spite of the large amount of recent attention given to media processing [5, 6, 13], there is very little quantitative understanding of the performance of such applications on general-purpose systems. A major challenge for such studies has been the large number of application classes in this domain (e.g., image, video, audio, speech, communication, graphics, etc.), and the absence of any standardized representative benchmark sets. Consequently, in contrast to the much-researched SPEC, SPLASH, and (more recently) TPC benchmarks, a number of fundamental questions still remain unanswered for media processing workloads. For example, is computation or memory the primary bottleneck in these applications? How effective are current architectural designs and media ISA extensions? What are the future challenges for these workloads? Given the lack of understanding of such issues, it is not surprising that the media instruction set extensions announced by different processor vendors vary widely – from 13 instructions in MVI for Alpha [4] to 162 instructions in AltiVec for PowerPC [19].

This paper is a first step in understanding the above issues to determine if and how we need to change the way we design general-purpose systems to support media processing applications. We focus on image and video workloads, an important class of media processing workloads, and attempt to cover the spectrum of the key tasks in this class. Our benchmark suite consists of 12 kernels and applications covering image processing, image source coding, and video source coding. We use detailed simulation to study a variety of general-purpose-processor architectural configurations, both with and without the use of Sun's visual instruction set (VIS) media ISA extensions. VIS shares a number of fundamental similarities with the media ISA extensions proposed for other processors, and is representative of the benefits and limitations of current media ISA extensions.

We start with a base single-issue in-order processor. In this system, all the benchmarks are primarily compute-bound. We find that conventional techniques in current processors to enhance instruction-level parallelism or ILP (multiple issue and out-of-order issue) provide a factor of 2.3X to 4.2X performance improvement for the benchmarks studied. The VIS media ISA extensions provide an additional 1.1X to 4.2X performance improvement. Our detailed analysis indicates the sources and limitations of the performance benefits due to VIS. The conventional ILP techniques and the VIS extensions together significantly reduce the CPU component of execution time, making five of the image processing benchmarks memory-bound.

The memory behavior of these workloads is character-

ized by large working sets and streaming data accesses. Increasing the cache size has no impact on 8 of the benchmarks. The remaining reuse data, but require relatively large cache sizes (dependent on the display sizes) to exploit the reuse and derive a performance benefit of less than 1.2X. Software-inserted prefetching provides 1.4X to 2.5X performance improvement in the image processing benchmarks where memory stall time is significant. With the addition of software prefetching, all of our benchmarks revert to being compute-bound.

The rest of the paper is organized as follows. Section 2 describes our workloads, the architectures modeled, and the simulation methodology. Section 3 presents our results on the impact of ILP features and VIS media extensions. Section 4 studies the performance of the cache system and the impact of software prefetching. Section 5 discusses related work. Section 6 concludes the paper.

## 2 Methodology

### 2.1 Workloads

We attempt to cover the spectrum of key tasks in image and video processing workloads. The kernels and applications in our benchmark suite form significant components of many current and future real-world workloads such as collaborative teleconferencing, scene-visualization, distance learning, streaming video across the internet, digital broadcasting, real-time flight imaging and radar sensing, content-based storage and retrieval, online video cataloging, and medical tomography [8]. Future standards such as JPEG2000 and MPEG4 are likely to build on a number of components of our benchmark suite.

Table 1 summarizes the 12 benchmarks that we use in this paper, and is divided into image processing (Section 2.1.1), image source coding (Section 2.1.2), and video source coding (Section 2.1.3). These benchmarks are similar to some of the benchmarks used in the image and video parts of the Intel Media Benchmark (described at the Intel web site) and the UCLA MediaBench [11].<sup>1</sup>

All the image benchmarks were run with 1024x640 pixel 3-band (i.e., channel) input images obtained from the Intel Media Benchmark. The video benchmarks were run with the *mei16v2* test bit stream from the MPEG Software Simulation Group that operates on 352x240 sized 3-band images. We did not study larger (full-screen) sizes because they were not readily available and would have required impractical simulation time.

---

<sup>1</sup>We did not use the Intel Media Benchmark or the UCLA MediaBench directly because the former does not provide source code and the latter does not include image processing applications.

### Image processing

|                 |   |
|-----------------|---|
| <i>Addition</i> | Addition of two images ( <i>sf16.ppm</i> , <i>rose16.ppm</i> ) using mean of two pixel values   |
| <i>Blend</i>    | Alpha blending of two images ( <i>sf16.ppm</i> , <i>rose16.ppm</i> ) with another alpha image ( <i>winter16.ppm</i> ); the operation performed is $dst = alpha \times src1 + (255 - alpha) \times src2$ .   |
| <i>Conv</i>     | General 3x3 image convolution of an image ( <i>sf16.ppm</i> ). The operation performed includes a saturation summation of 9 product terms. Each term corresponds to multiplying the pixel values in a moving 3x3 window across the image dimensions with the values of a 3x3 kernel matrix. |
| <i>Dotprod</i>  | 16x16 dot product of a randomly-initialized 1048576-element linear array  |
| <i>Scaling</i>  | Linear image scaling of an image ( <i>sf16.ppm</i> )  |
| <i>Thresh</i>   | Double-limit thresholding of an image ( <i>sf16.ppm</i> ). If the pixel band value falls within the low and high values for that band, the destination is set to the map value for that band; otherwise, the destination is set to be the same as the source pixel value.                   |

### Image source coding

|                 |   |
|-----------------|---|
| <i>Cjpeg</i>    | JPEG progressive encoding ( <i>rose16.ppm</i> )     |
| <i>Djpeg</i>    | JPEG progressive decoding ( <i>rose16.jpg</i> )     |
| <i>Cjpeg-np</i> | JPEG non-progressive encoding ( <i>rose16.ppm</i> ) |
| <i>Djpeg-np</i> | JPEG non-progressive decoding ( <i>rose16.jpg</i> ) |

### Video source coding

|                 |   |
|-----------------|---|
| <i>Mpeg-enc</i> | MPEG2 encoding of 4 frames (I-B-B-P frames) of the <i>mei16v2rec</i> bit stream. Properties of the bit stream include frame rate of 30fps, bit rate of 5Mbps at the Main profile@Main level configuration. The image is 352x240 pixels in the 4:2:0 YUV chroma format, and is scaled to a 704x480 display. The quantization tables and the motion estimation search parameters are set to the default parameters specified by the MPEG group. |
| <i>Mpeg-dec</i> | MPEG2 decoding of the <i>mei16v2rec</i> video bit stream into separate YUV components.  |

**Table 1. Summary of the benchmarks used in this study.**

#### 2.1.1 Image Processing

Our image processing benchmarks are taken from the Sun VIS Software Development Kit (VSDK), which includes 14 image processing kernels. These kernels include common image processing tasks such as one-band and three-band (i.e., channel) alpha blending (used in image compositing), single-limit and double-limit thresholding (used in chroma-keying, image masking, and blue screening), and functions such as general and separable convolution, copying, inversion, addition, dot product, and scaling (used in the core of many image processing codes like blurring, sharpening, edge detection, embossing, etc.). We study all 14 of the VSDK kernels, but due to space constraints, we report results for only 6 representative benchmarks (*addition*, *blend*, *conv*, *dotprod*, *scaling*, and *thresh*).

#### 2.1.2 Image Source Coding

We focus on the Joint Photography Experts Group (JPEG) standard and study the performance of the Release 6a codec (encoder/decoder) from the Independent JPEG Group. We study two different commonly used codecs specified in the standard, a progressive JPEG codec (*cjpeg* encoder and *djpeg* decoder), and a non-progressive JPEG codec (*cjpeg-np* encoder and *djpeg-np* decoder).

The JPEG encoding process consists of a number of phases many of which exploit properties of the human visual system to reduce the number of bits required to spec-

ify the image. First, the *color conversion* and *chroma-decimation* phases convert the source image from a 24-bit RGB representation domain to a 12 bit 4:2:0 YUV representation. Next, a linear *DCT image transform* phase converts the image into the frequency domain. The *quantization* phase then scales the frequency domain values by a quantization value (either constant or variable). The *zig-zag scanning* and *variable-length (Huffman) coding* phases then reorder the resulting data into streams of bits and encode them as a stream of variable-length symbols based on statistical analysis of the frequency of symbols.

*Progressive image compression* uses a compression algorithm that performs multiple Huffman coding passes on the image to encode it as multiple scans of increasing picture quality (leading to the perception of gradual focusing of images seen on many web pages).

The decoding process performs the inverse of the operations for the encoding process in the reverse order to obtain the original image from the compressed image.

#### 2.1.3 Video Source Coding

We focus on the Motion Picture Experts Group-2 (MPEG2) video coding standard, and study the performance of the version 1.1 codec from the MPEG Software Simulation Group.

The first part of the video compression process consists of spatial compression similar to that described for JPEG

|   |        |
|---|--------|
| Processor speed                           | 1 GHz  |
| Issue width                               | 4-way  |
| Instruction window size                   | 64     |
| Memory queue size                         | 32     |
| <i>Branch prediction</i>                  |        |
| Bimodal agree predictor size              | 2K     |
| Return-address stack size                 | 32     |
| Taken branches per cycle                  | 1      |
| Simultaneous speculated branches          | 16     |
| <i>Functional unit counts</i>             |        |
| Integer arithmetic units                  | 2      |
| Floating-point units                      | 2      |
| Address generation units                  | 2      |
| VIS multipliers                           | 1      |
| VIS adders                                | 1      |
| <i>Functional unit latencies (cycles)</i> |        |
| Default integer/address generation        | 1/1    |
| Integer multiply/divide                   | 7/12   |
| Default floating point                    | 4      |
| FP moves/converts/divides                 | 4/4/12 |
| Default VIS                               | 1      |
| VIS 8-bit loads/multiply/pdist            | 1/3/3  |

**Table 2. Default processor parameters.**

|                                    |          |
|------------------------------------|----------|
| Cache line size                    | 64 bytes |
| L1 data cache size (on-chip)       | 64 KB    |
| L1 data cache associativity        | 2-way    |
| L1 data cache request ports        | 2        |
| L1 data cache hit time             | 2 ns     |
| Number of L1 MSHRs                 | 12       |
| L2 cache size (off-chip)           | 128K     |
| L2 cache associativity             | 4-way    |
| L2 request ports                   | 1        |
| L2 hit time (pipelined)            | 20 ns    |
| Number of L2 MSHRs                 | 12       |
| Max. outstanding misses per MSHR   | 8        |
| Total memory latency for L2 misses | 100 ns   |
| Memory interleaving                | 4-way    |

**Table 3. Default memory system parameters.**

and includes the color conversion, chroma decimation, frequency transformation, quantization, zig-zag coding, and run-length coding phases. Additionally, MPEG2 has an inter-frame predictive-compression *motion-estimation* phase that uses difference vectors to encode temporal redundancy between macroblocks in a frame and macroblocks in the following and preceding frames. Motion estimation is the most compute-intensive part of *mpeg-encode*.

The video decompression process performs the inverse of the various encode operations in reverse order to get the decoded bit stream from the input compressed video. The *mei16v2* bit stream is already in the YUV format, and consequently, our MPEG simulations do not go through the color conversion phase discussed in Section 2.1.2.

## 2.2 Architectures Modeled

### 2.2.1 Processor and Memory System

We study two processor models – an in-order processor model (similar to the Compaq Alpha 21164, Intel Pentium, and Sun UltraSPARC-II processors) and an out-of-order processor model (similar to the Compaq Alpha 21264, HP PA8000, IBM PowerPC, Intel Pentium Pro, and MIPS R10000 processors). Both the processor models support non-blocking loads and stores.

For the experiments with software prefetching, the processor models provide support for software-controlled non-binding prefetches into the first-level cache.

The base system uses a 64KB two-way associative first-level (L1) write-back cache and a 128KB 4-way associative second-level (L2) write-back cache. Section 4.1 discusses the impact of varying the cache sizes. All the caches are non-blocking and allow support for multiple outstanding misses. At each cache, 12 miss status holding registers (MSHRs) reserve space for outstanding cache misses and combine a maximum of 8 multiple requests to the same cache line.

Tables 2 and 3 summarize the parameters used for the processor and memory subsystems. When studying the performance of a 1-way issue processor, we scale the number of functional units to 1 of each type. The functional unit latencies were chosen based on the Alpha 21264 processor. All functional units are fully pipelined except the floating-point divide (non-pipelined).

### 2.2.2 VIS Media ISA Extensions

The VIS media ISA extensions to the SPARC V9 architecture are a set of instructions targeted at accelerating media processing [10, 23]. Both our in-order and out-of-order processor models include support for VIS.

The VIS extensions define the packed byte, packed word and packed double data types which allow concurrent operations on eight bytes, four words (16-bits each) or two double words of fixed-point data in a 64-bit register. These data types allow VIS instructions to exploit single-instruction-multiple-data (SIMD) parallelism at the subword level. Most of the VIS instructions operate on packed words or packed doubles; loads, stores, and *pdist* instructions operate on packed bytes. Many of the VIS instructions make implicit assumptions about rounding and the number of significant bits in the fixed-point data. Hence, their use require ensuring that they do not lead to incorrect outputs. We next provide a short overview of the VIS instructions (summarized in Table 4).

**Packed arithmetic and logical operations.** The packed arithmetic VIS instructions allow SIMD-style parallelism to be exploited for add, subtract, and multiply instructions. To

|   |
|---|
| <i>Packed arithmetic and logical operations</i> |
| Packed addition                                 |
| Packed subtraction                              |
| Packed multiplication                           |
| Logical operations                              |
| <i>Subword rearrangement and realignment</i>    |
| Data packing and expansion                      |
| Data merging                                    |
| Data alignment                                  |
| <i>Partitioned compares and edge operations</i> |
| Partitioned compares                            |
| Mask generation for edge effects                |
| <i>Memory-related operations</i>                |
| Partial stores                                  |
| Short loads and stores                          |
| Blocked loads and stores                        |
| <i>Special-purpose operations</i>               |
| Pixel distance computation                      |
| Array address conversion for data reuse         |
| Access to the graphics status register          |

**Table 4. Classification of VIS instructions**

minimize implementation complexity, VIS uses a pipelined series of two 8x16 multiplies and one add instruction to emulate packed 16x16-bit multiplication. The VIS logical instructions allow logical operations on the floating-point data path.

**Subword rearrangement and alignment.** To facilitate conversion between different data types, VIS supports subword rearrangement and alignment using pack, expand, merge (interleave), and align instructions. The subword rearrangement instructions also include support for implicitly handling saturation arithmetic (limiting data values to the minimum or maximum instead of the default wrap-around).

**Partitioned compares and edge operations.** For branches, VIS supports a partitioned compare that performs four 16-bit or two 32-bit compares in parallel to produce a mask that can be used in subsequent instructions. VIS also supports the edge instruction to generate masks for partial stores that can eliminate special branch code to handle boundary conditions in media processing applications.

**Memory-related operations.** For memory instructions, VIS supports partial stores that selectively write to parts of the 64-bit output based on an input mask. Short loads and stores transfer 1 or 2 bytes of memory to the register file. Blocked loads and stores transfer 64 bytes of data between memory and a group of eight consecutive VIS registers without causing allocations in the cache.

**Special-purpose operations.** The pixel distance computation (*pdist*) instruction is primarily targeted at motion estimation and computes the sum of the absolute differences between corresponding 8-bit components in two packed bytes. The *array* instruction is mainly targeted at 3D graphics rendering applications and converts 3D fixed-point coordinates into a blocked byte address that allows for greater

cache reuse. VIS also defines instructions to manipulate the graphics status register, a special-purpose register that stores additional data for various media instructions.

Overall, the functionality discussed above for VIS is similar to that of fixed-point media ISA extensions in other general purpose processors (e.g., MAX [12], MMX [18], MVI [4], MDMX [9], AltiVec [19]). The various ISA extensions mainly differ in the number, types, and latencies of the individual instructions (e.g., MMX implements direct support for 16x16 multiply), whether they are implemented in the integer or floating-point data path, and in the width of the data path. The most different ISA extension, the proposed PowerPC AltiVec ISA, adds support for a separate 128-bit vector multimedia unit in the processor.

Our VIS implementation is closely modeled after the UltraSPARC-II implementation and operates on the floating-point register file with latencies comparable to the UltraSPARC-II [23] (Table 2).<sup>2</sup> The increase in chip area associated with the VIS instructions was estimated to be less than 3% for the UltraSPARC-II [10].

## 2.3 Methodology

### 2.3.1 Simulation Environment

We use the RSIM simulator [16] to simulate the in-order and out-of-order processors described in Section 2.2. RSIM is a user-level execution-driven simulator that models the processor pipeline and memory hierarchy in detail including contention for all resources. To assess the impact of not modeling system level code, we profiled the benchmarks on an UltraSPARC-II-based Sun Enterprise server. We found that the time spent on operating system kernel calls is less than 2% on all the benchmarks. The time spent on I/O is less than 15% on all the benchmarks except *mpeg-dec*. This benchmark experiences an inflated I/O component (45%) because of its high frequency of file writes. In a typical system, however, these writes would be handled by a graphics accelerator, significantly reducing this component. Since our applications have small instruction footprints, our simulations assume all instructions hit in the instruction cache.

All the applications<sup>3</sup> were compiled with the SPARC SC4.2 compiler with the *-xO4 -xtarget=ultra1/170 -xarch=v8plusa -dalign* options to produce optimized code for the in-order UltraSPARC processor.

<sup>2</sup>Our VIS multiplier has a lower latency compared to the integer (64-bit) multiplier because it operates on 16-bit data.

<sup>3</sup>We changed the 14 image processing kernels from the Sun VSDK to skew the starting addresses of concurrent array accesses and unroll small innermost loops. This reduced cache conflicts and branch mispredictions leading to 1.2X to 6.7X performance benefits. To facilitate modifying the applications for VIS, we replaced some of the key routines in the JPEG and MPEG applications with equivalent routines from the Sun *MediaLib* library.

### 2.3.2 VIS Usage Methodology

We are not aware of any compiler that automatically modifies media processing applications to use media ISA extensions. For our experiments studying the impact of VIS, we manually modified our benchmarks to use VIS instructions based on the methodology detailed below.

We profiled the applications to identify key procedures and manually examined these procedures for loops that satisfied the following three conditions: (1) The loop body should have no loop-carried dependences or control dependences that cannot be converted to data dependences (other than the loop branch). (2) The key computation in the loop body must be replaceable with a set of equivalent fixed-point VIS instructions. The loss in accuracy in this stage, if any, should be visually imperceptible. (3) The potential benefit from VIS should be more than the overhead of adding VIS; VIS overhead can result from subword rearrangement instructions to convert between packed data types or from alignment-related instructions.

For loops that satisfied the above criteria, we strip-mined or unrolled the loop to isolate multiple iterations of the loop body that we then replaced with equivalent VIS instructions. We used the inline assembly-code macros from the Sun VSDK for the VIS instructions; this minimizes code perturbation and allows the use of regular compiler optimizations. Wherever possible, we tried to use procedures available from the Sun VSDK Kit and the SUN MediaLib library routines that were already optimized for VIS.

Our benchmarks use all the VIS instructions except for the array and blocked load/store instructions. Array instructions are targeted at 3D array accesses in graphics loops, and are not applicable for our applications. Blocked loads and stores are primarily targeted at transfers of large blocks of data between buffers without affecting the cache (e.g., in operating system buffer management, networking, memory-mapped I/O). We did not use these instructions since the Sun VSDK does not provide inline assembly-code macros to support them. The alternative of hand-coded assembly could result in lower performance since it is hard to emulate the compiler optimizations associated with modern superscalar processors by hand [22]. Note that both the array and blocked load/store instructions are unique to VIS and are not supported by other general-purpose ISA extensions.

### 2.3.3 Software Prefetching Algorithm

We studied the applicability of software prefetching for the benchmarks where the cache miss stall time is a significant component of the total execution time (>20%). We identified the memory accesses that dominate the cache miss stall time, and inserted prefetches by hand for these accesses. We followed the well known software prefetching compiler

algorithm developed by Mowry et al. [14].

### 2.3.4 Performance Metrics

We use the execution time of the system as the primary metric to evaluate the performance of the system, while also reporting the individual components of execution time. With out-of-order processors, an instruction can potentially be overlapped with instructions preceding and following it. We therefore use the following convention to identify the different components of execution time. At every cycle, the fraction of instructions retired that cycle to the maximum retire rate is attributed to the busy time; the remaining fraction is attributed as stall time to the first instruction that could not be retired that cycle. We also study other metrics such as dynamic instruction counts, branch misprediction rates, cache miss rates, MSHR occupancies, and prefetch counts for further insights into the system behavior.

## 3 Improving Processor Performance

For each benchmark, Figure 1 presents execution times for three variations of our base architecture, each without VIS (the first set of three bars) and with VIS (the second set of three bars). The three architecture variations are (i) in-order and single issue, (ii) in-order and 4-way issue, and (iii) out-of-order and 4-way issue. On the VIS-enhanced architecture, we use the VIS-enhanced version of the application as mentioned in Section 2. The execution times are normalized to the time with the in-order single-issue processor. For all the benchmarks, the execution time is divided into the busy component, the functional unit stall (FU stall) component, and the memory component. The memory component is shown divided into the L1 miss and L1 hit components.

### 3.1 Impact of Conventional ILP Features

This section focuses on the system without the VIS media ISA extensions (the left three bars for each benchmark in Figure 1).

**Overall results.** Both multiple issue and out-of-order issue provide substantial reductions in execution time for most of our benchmarks. Compared to a single-issue in-order processor, on the average, multiple issue improves performance by a factor of 1.2X (range of 1.1X to 1.4X), while the combination of multiple issue and out-of-order issue improves performance by a factor of 3.1X (range of 2.3X-4.2X).

**Analysis.** Compared to the single issue processor, we find that multiple issue achieves most of its benefits by reducing the busy CPU component of execution time. Data, control, and structural dependences prevent the CPU component from attaining an ideal speedup of 4 from a 4-way is-

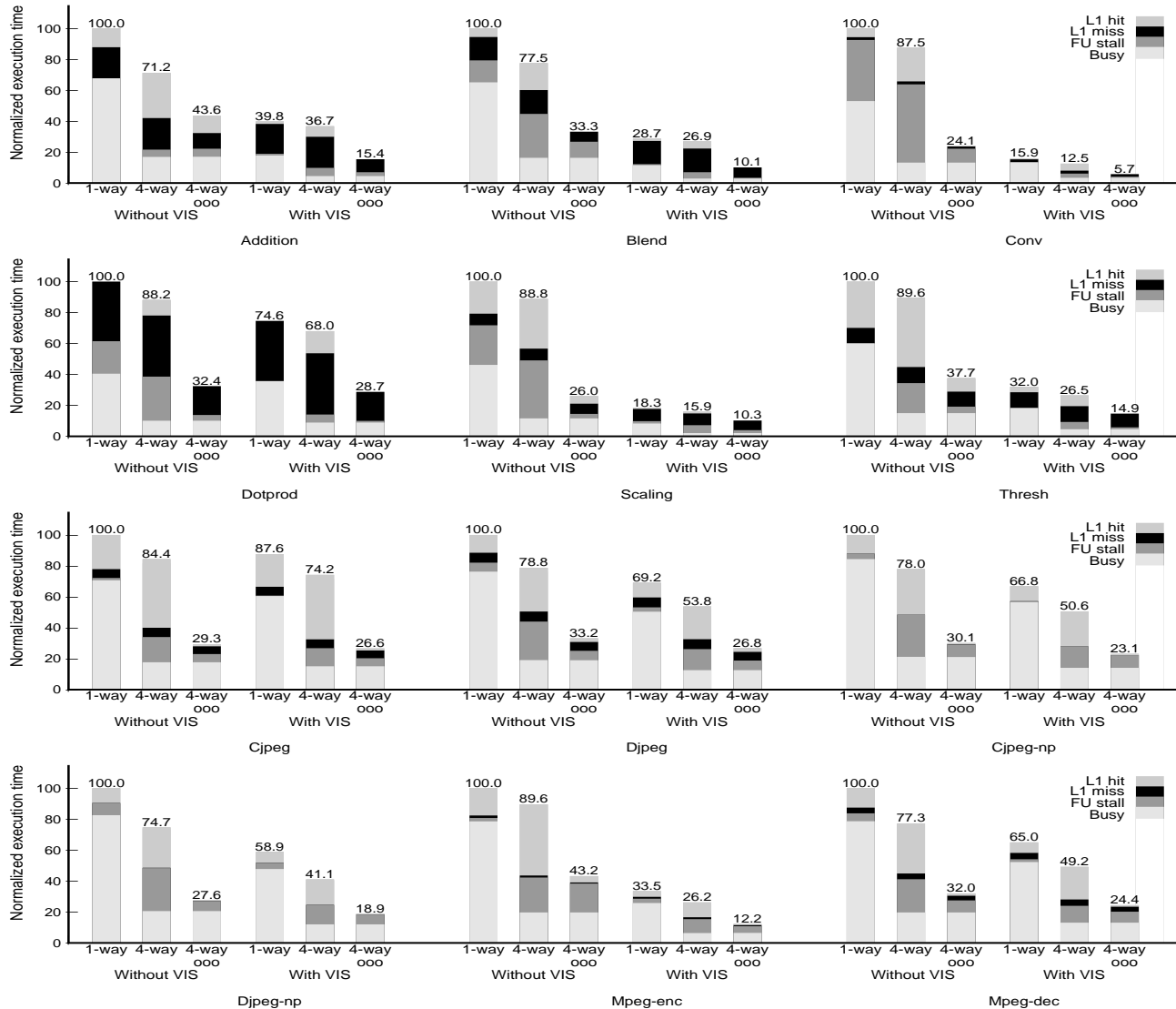


Figure 1. Performance of image and video benchmarks.

sue processor, reflected in the increased functional unit and L1 hit memory stall time.

Some of the benchmarks see additional memory latencies when the number of outstanding misses to one cache line (MSHR) increases beyond the maximum allowed limit of 8. This is caused by the higher use of small data types associated with media applications which leads to a high frequency of accesses for each cache line (e.g., 64 pixel writes in a 64-byte line). Since the processors do not stall on writes, in benchmarks with small loop bodies (e.g., *addition*, *cjpeg*, *djpeg*), this leads to a backup of multiple writes. This backup leads to contention for the MSHR that eventually prevents other accesses from being serviced at the cache.

Out-of-order issue, on the other hand, improves performance by reducing both functional unit stall time and mem-

ory stall time. A large fraction of the stall times due to data, control, and structural dependences, as well as MSHR contention, is now overlapped with other useful work. This is seen in the reduction in the FU stall and L1 hit components of execution time. Additionally, out-of-order issue can better exploit the non-blocking loads feature of the system by allowing the latency of multiple long-latency load misses to be overlapped with one another. Our results examining the MSHR occupancies at the cache indicate that while there is increased load miss overlap in all the 12 benchmarks, only 2 to 3 misses are overlapped in most cases. The total capacity of 12 MSHRs is never fully utilized for load misses in all our benchmarks.

Overall, the impact of the various ILP features is qualitatively consistent with that described in previous studies for scientific and database workloads. Quantitatively, these ILP

features are substantially more effective for the image and video benchmarks than for previously reported online transaction processing (OLTP) workloads [21], and comparable in benefit to previously reported scientific and decision support system (DSS) workloads [1, 17, 21].

It must be noted that the performance of the in-order issue processor is dependent on the quality of the compiler used to schedule the code. Our experiments use the commercial SPARC SC4.2 compiler with maximum optimizations turned on for the in-order UltraSPARC processor. To try to isolate compiler scheduling effects, we studied two other processor configurations with single-cycle functional unit latencies and functional unit latencies comparable to the UltraSPARC processor. In both these configurations, our results continued to be qualitatively similar; the out-of-order processor continues to significantly outperform the in-order processor. The impact of future, more advanced, compiler optimizations, however, is still an open question.

Interestingly, a recent position paper [6] on the impact of multimedia workloads on general-purpose processors conjectures that complex out-of-order issue techniques developed for scientific and engineering workloads (e.g., SPEC) may not be needed for multimedia workloads. Our results show that, on the contrary, out-of-order issue can provide significant performance benefits for the image and video workloads.

## 3.2 Impact of VIS Media ISA Extensions

This section discusses the performance impact of the VIS ISA extensions (comparing the left-hand three bars and right-hand three bars for each benchmark in Figure 1).

### 3.2.1 Overall Results

The VIS media ISA extensions provide significant performance improvements for all the benchmarks (factors of 1.1X to 4.0X for the out-of-order system, 1.1X to 7X across all configurations). On average, the addition of VIS improves the performance of the single-issue in-order system by a factor of 2.0X, the performance of the 4-way-issue in-order system by a factor of 2.1X, and the performance of the 4-way issue out-of-order system by a factor of 1.8X.

Multiple issue and out-of-order issue are beneficial even with VIS. On average, with VIS, compared to a single-issue in-order processor, multiple issue achieves a factor of 1.2X performance improvement, while the combination of multiple issue and out-of-order issue achieves a factor of 2.7X performance improvement. The reasons for these performance benefits from ILP features are the same as for the systems without VIS.

### 3.2.2 Benefits from VIS

Figure 2 presents some additional data showing the distribution of the dynamic (retired) instructions for the 4-way out-of-order processor without and with VIS, normalized to the former. Each bar divides the instructions into the Functional unit (FU, combines ALU and FPU), Branch, Memory, and VIS categories. The use of VIS instructions provides a significant reduction in the dynamic instruction count for all the benchmarks. The reductions in the dynamic instruction count correlate well with the performance benefits from VIS. We next discuss the sources for the reductions in the dynamic instructions.

**Reductions in FU instructions.** The VIS packed arithmetic and logical instructions allow multiple (typically four) arithmetic instructions to be replaced with one VIS instruction. Consequently, all the benchmarks see significant reductions in the FU instructions with corresponding, smaller, increases in the VIS instruction count. Additionally, the SIMD VIS instructions replace multiple iterations in the original loop with one equivalent VIS iteration. This reduces iteration-specific loop-overhead instructions that increment index and address values and compute branch conditions, further reducing the FU instruction count.

**Reductions in branch instructions.** All the benchmarks use the edge masking and partial store instructions to eliminate testing for edge boundaries and selective writes. They also use loop unrolling when replacing multiple iterations with one equivalent VIS iteration. These lead to a reduction in the branch instruction count for all the benchmarks. For some applications, branch instruction counts are also reduced because of the elimination of the code to explicitly perform saturation arithmetic (mainly in *conv*<sup>4</sup> and the JPEG applications), and the use of partitioned SIMD compares (mainly in *thresh*).

Many of the branches eliminated are hard-to-predict branches (e.g., saturation, thresholding, selective writes), leading to significant improvements in the hardware branch misprediction rates for some of our benchmarks (branch misprediction rate decreases from 10% to 0% for *conv* and from 6% to 0% for *thresh1*).

**Reductions in memory instructions.** With VIS, memory accesses operate on packed data as opposed to individual media data. Consequently, most of the benchmarks see significant reductions in the number of memory instructions (and associated cache accesses). This reduces the MSHR contention discussed in Section 3.1.

Most of the memory instructions eliminated are cache hits, without a proportional decrease in the number of

---

<sup>4</sup>The original source code from the Sun VSDK checks for saturation only in the *conv* code. The *add*, *blend*, and *dotprod* kernels are written in the non-saturation mode. These could, however, potentially be rewritten to check for saturation in which case they would also see similar benefits.

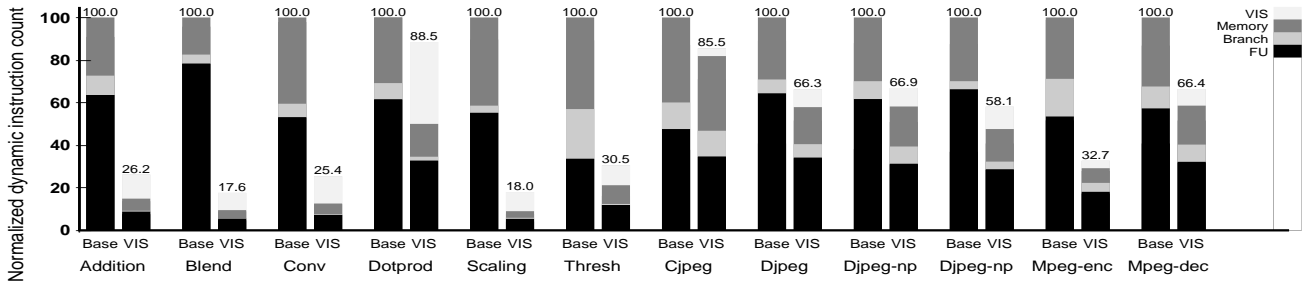


Figure 2. Impact of VIS on dynamic (retired) instruction count.

misses, causing higher L1 cache miss rates with VIS. The higher miss rate and the lower instruction count allows additional load misses to appear together within the instruction window and be overlapped with each other. However, the system still rarely sees more than 3 load misses overlapped concurrently.

**Pixel distance computation for *mpeg-enc*.** *mpeg-enc* achieves additional benefits from the special-purpose pixel distance computation (`pdist`) instruction in the motion estimation phase. The `pdist` instruction allows a sequence of 48 instructions to be reduced to one instruction [23], significantly reducing the FU, branch, and memory instruction counts. The elimination of hard-to-predict branches to perform comparisons and saturation improves the branch misprediction rate from 27% to 10%.

### 3.2.3 Limitations of VIS

Examining the variation of performance benefits across the benchmarks, we observe that the JPEG applications, *mpeg-dec*, and *dotprod* exhibit relatively lower performance benefits (factors of 1.1X to 1.5X) compared to the remaining benchmarks (factors of 2.8X to 4.2X). We next discuss factors limiting the benefits from VIS.

**Inapplicability of VIS.** The VIS media ISA extensions are not applicable to a number of key procedures in the JPEG applications and *mpeg-dec*. For example, the JPEG applications (especially the progressive versions) spend a large fraction of their time in the variable-length Huffman coding phase. This phase is inherently sequential and operates on variable-length data types, and consequently cannot be optimized using VIS instructions.<sup>5</sup> Other examples of code segments in the JPEG and MPEG applications where VIS could not be applied include bit-level input/output stream manipulation, scatter-gather addressing, quantization, and saturation arithmetic operations not embedded in a loop.

**VIS overhead.** All our benchmarks use subword rearrangement and alignment instructions to get the data in a form that the VIS instructions can operate (e.g., pack-

ing/unpacking between packed-byte pixels and packed-word operands). This results in extra overhead that limits the performance benefits from VIS (on the average, for our benchmarks, 41% of the VIS instructions are for subword rearrangement and alignment). Overhead is also increased when multiple VIS instructions are used to emulate one operation (e.g., 16x16 multiply in *dotprod*) or when the data need to be reordered to exploit SIMD (e.g., byte reordering in the color conversion phase in JPEG).

**Limited parallelism and scheduling constraints.** Most of the packed arithmetic instructions operate only on packed words or packed double words. This ensures enough bits to maintain the precision of intermediate values. However, this limits the maximum parallelism to 4 (on 16-bit data types), even in cases when the operations are known to be performed on smaller data types (e.g., 8-bit pixels). The limit on the SIMD parallel path,<sup>6</sup> in combination with contention for VIS functional units, limits the benefits from VIS on some of our benchmarks (most significantly in *mpeg-enc*).

**Cache miss stall time.** As discussed earlier, the reductions in memory instructions mainly occur for cache hits. The VIS instructions do not directly target cache misses though there are indirect benefits associated with increased load miss overlap due to instruction count reduction (discussed in Section 3.2.2).

### 3.3 Combination of ILP and VIS

The combination of conventional ILP features and VIS extensions achieves an average factor of 5.5X performance improvement (range of 3.5X to 18X) over the base single-issue in-order processor. The benefits from VIS are achieved with a much smaller increase in the die area compared to the ILP features.

On the base single-issue in-order processor, all the benchmarks are primarily compute-bound. With ILP features and VIS extensions, *cjpeg*, *djpeg*, and *mpeg-enc* continue to spend most of their execution time in the proces-

<sup>5</sup>It is instructive to note that many media processors (e.g., the Mitsubishi VLIW Media Processor and Samsung Media Signal Processor) have a special-purpose hardware unit to handle the variable-length coding.

<sup>6</sup>The MIPS MDMX provides support for a larger size accumulator that allows greater parallelism without losing the precision of the intermediate result [9]. The PowerPC AltiVec supports a larger 128-bit data path to increase the parallelism[19].

sor sub-system (87% to 97%). Five of the image processing kernels, however, now spend 55% to 66% of their total time in memory stalls. The strong compute-centric performance benefits from ILP features and VIS extensions shift the bottleneck to the memory sub-system for these benchmarks. The remaining 4 applications (*conv*, *cjpeg*, *djpeg*, and *mpeg-dec*) spend between 20% to 30% of their total time on memory stalls.

## 4 Improving Memory System Performance

This section studies memory system performance for the benchmarks. Section 4.1 discusses the effectiveness of caches, and Section 4.2 discusses the impact of software prefetching.

### 4.1 Impact of Caches

**Impact of varying L2 cache size.** We varied the L2 cache size from 128K to 2M, keeping the L1 cache fixed at 64K. Our results (not shown here due to lack of space) showed that increasing the size of the L2 cache has no impact on the performance of the 6 image processing kernels and the *cjpeg-np* and *djpeg-np* applications. The remaining four applications, *cjpeg*, *djpeg*, *mpeg-enc*, and *mpeg-dec*, reuse data; but the cache size needed to exploit the reuse depends on the size of the display. For our input image sizes, L2 cache sizes of 2M capture the entire working sets for all the four benchmarks, and provide 1.1X to 1.2X performance improvement over the default 128K L2 cache. With the 2M cache sizes, memory stall time is between 7-9% on all the applications and is dominated by L1 hit time (due to MSHR contention) and L2 hit time.

The image processing kernels have streaming data accesses to a large image buffer with no reuse and low computation per cache miss. Consequently, they exhibit high memory stall times unaffected by larger caches. *cjpeg-np* and *djpeg-np* do not see any variation in performance with larger caches because of their negligible memory components. These applications implement a blocked pipeline algorithm that performs all the computation phases on 8x8-sized blocks at a time, reducing the bandwidth requirements and increasing the computation per miss. The *cjpeg* and *djpeg* applications differ from their non-progressive counterparts in the progressive coding phase where they perform a multi-pass traversal of the buffer storing the DCT coefficients. The low computation per miss in this phase combined with the reuse of the image-sized (1024x640 pixels) buffer results in a 1.2X performance benefit from increasing the cache size to 2M. Larger images would increase the working set requiring larger caches. For example, a 1024x1024 image would require a 4M cache size. *mpeg-enc* and *mpeg-dec* perform inter-block distance vector op-

erations and therefore need to operate on multiple image-sized buffers (as opposed to blocks-sized buffers in *cjpeg-np* and *djpeg-np*). The reuse of these 352x240 buffers across the frames in the video leads to 1.1X performance benefits with 512K (*mpeg-dec*) and 1M (*mpeg-enc*) cache sizes. Larger image sizes would require larger caches; for example a 1024x1024 image would require almost a factor of 12X increase.

**Impact of varying L1 caches.** We also performed experiments varying the size of the L1 cache from 1K to 64K while keeping the L2 cache fixed at 128K. Our results showed that the L1 cache size had no impact on five of the image processing kernels. On the remaining benchmarks, a 64K L1 configuration outperforms the 1K L1 configuration by factors of 1.1X to 1.3X; 4K-16K L1 caches achieve within 3% of the performance of the 64K L1 cache configuration. Small data structures other than the main data, such as tables for convolution, quantization, color conversion, and saturation clipping, are responsible for these small first-level working sets. At 64K L1 caches, memory stall time is mainly due to L1 hits (mainly related to MSHR contention) or to L2 misses.

### 4.2 Impact of Software Prefetching

Figure 3 summarizes the execution time reductions from software prefetching relative to the base system with VIS (with 64K L1 and 128K L2 caches). We do not report results for *cjpeg-np*, *djpeg-np* and *mpeg-enc* since these benchmarks spend less than 6% of their total time on L1 cache misses. Our results show that software prefetching achieves high performance improvements for the six image processing benchmarks (an average of 1.9X and a range of 1.4X to 2.5X). The *cjpeg*, *djpeg*, and *mpeg-dec* benchmarks exhibit relatively small performance improvements. Overall, after applying software prefetching, all our benchmarks revert to being compute bound.

For the image processing kernels, a significant fraction of the prefetches are useful in completely or partially hiding the latency of the cache miss with computation or with other misses. The addition of software prefetching also increases the utilization of cache MSHRs; in many of the image processing kernels, more than 5 MSHRs are used for a large fraction of the time. The remaining memory stall time is mainly due to late prefetches and resource contention. Late prefetches (prefetches that arrive after the demand access) arise mainly because of inadequate computation in the loop bodies to overlap the miss latencies. Contention for resources occurs when multiple prefetches are outstanding at a time. These effects are similar to those discussed in previous studies with scientific applications for ILP-based processors [20].

The other benchmarks (*cjpeg*, *djpeg*, and *mpeg-dec*) see

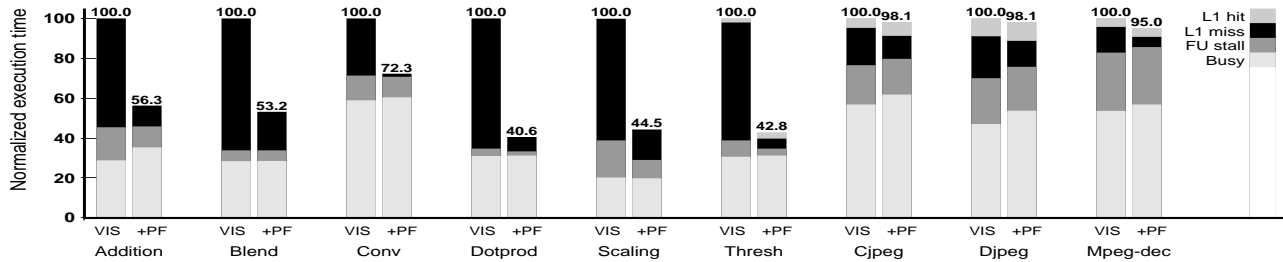


Figure 3. Effect of software-inserted prefetching.

lower performance benefits primarily because the fraction of memory stall time is relatively low and includes an L1 hit component (mainly due to MSHR contention). Software prefetches do not address the L1 component. Second, in *cjpeg* and *djpeg*, the prefetches are to memory locations that are indirectly addressed (of the form  $A[B[i]]$ ). Consequently, the prefetching algorithm is unable to distinguish between hits and misses and is constrained to issue prefetches for all accesses. The resulting overhead due to address calculation and cache contention limits performance (seen as increased Busy and FU stall components<sup>7</sup>). Finally, as before, late prefetches and resource contention also contribute to the lower benefits from prefetching.

## 5 Related Work

Most of the papers discussing instruction-set extensions for multimedia processing have focused on detailed descriptions of the additional instructions and examples of their use [4, 9, 12, 15, 18, 19, 23]. The performance characterization in these papers is usually limited to a few sample code segments and/or a brief mention of the benefits anticipated on larger applications. Eyre studies the applicability of general-purpose processors for DSP applications; however, the study only reports high-level metrics such as MIPS, power efficiency, and cost [7].

Daniel Rice presents a detailed description of VIS and 8 image processing applications without and with VIS [22]. The study reports speedups of 2.7X to 10.5X on an actual UltraSPARC-based system, but does not analyze the cause of performance benefits, the remaining bottlenecks, or the impact of alternative architectures.

Yang et al. look at the benefits of packed floating-point formats and instructions for graphics but assume a perfect memory system [24]. Bhargava et al. study some MMX-enhanced applications based on Pentium-based systems; but again, no detailed characterization of performance bottlenecks or the impact of other architectures is done [2]. Zucker et al. study MPEG video decode applications and

<sup>7</sup>Some of the image processing kernels see a reduction in the CPU component because of the reduction in instructions and better scheduling when loops are unrolled for the prefetching algorithm [14].

show the benefits from I/O prefetching, software restructuring to use SIMD without hardware support, and profile-driven software prefetching [25, 26]. However, the studies assume a simplistic processor model with blocking loads and do not study the effect of media ISA extensions. Bilas et al. develop two parallel versions of the MPEG decoder and present results for multiprocessor speedup, memory requirements, load balance, synchronization, and locality [3]. Similar to our results, they also find that the miss rates for 352x240 images on “realistic” cache sizes are negligible.

## 6 Conclusions

Media processing is a workload of increasing importance for desktop processors. This paper focuses on image and video processing, an important class of media processing, and aims to provide a quantitative understanding of the performance of these workloads on general-purpose processors. We use detailed simulation to study 12 representative benchmarks on a variety of architectural configurations, both with and without the use of Sun’s visual instruction set (VIS) media ISA extensions.

Our results show that conventional techniques in current processors to enhance ILP (multiple issue and out-of-order issue) provide a factor of 2.3X to 4.2X performance improvement for the image and video benchmarks. The Sun VIS media ISA extensions provide an additional 1.1X to 4.2X performance improvement. The benefits from VIS are achieved with a much smaller increase in the die area compared to the ILP features.

Our detailed analysis indicates the sources and limitations of the performance benefits due to VIS. VIS is very effective in exploiting SIMD parallelism using packed data types, and can eliminate a number of potentially hard-to-predict branches using instructions targeted towards saturation arithmetic, boundary detection, and partial writes. Special-purpose instructions such as *pdist* achieve high benefits on the targeted application, but are too specialized to use in other cases. Routines that are sequential and operate on variable data types, VIS instruction overhead, cache miss stall times, and the fixed parallelism in the packed arithmetic instructions limit the benefits on the benchmarks.

On our base single-issue in-order processor, all the benchmarks are primarily compute-bound. Conventional ILP features and the VIS instructions together significantly reduce the CPU component of execution time, making 5 of our image processing benchmarks memory-bound. The memory behavior of these workloads is characterized by large working sets and streaming data accesses. Increasing the cache size has no impact on the image processing kernels and the non-progressive JPEG applications. This is particularly interesting considering current trends towards large on-chip and off-chip caches. The remaining benchmarks require relatively large cache sizes (dependent on the display sizes) to exploit data reuse, but derive less than 1.2X performance benefits with the larger caches. Software-inserted prefetching achieves 1.4X to 2.5X performance benefits on the image processing kernels where memory stall time is significant.

With the addition of software prefetching, all our benchmarks revert to being compute-bound. Architectural optimizations that improve computation time (e.g., multiprocessing) may be useful to exploit greater parallelism. Such efforts are likely to expose the memory system bottleneck yet again, possibly requiring additional novel memory system techniques beyond conventional software prefetching. In the future, we plan to explore new architectural techniques for general-purpose processors to support media processing workloads. We also plan to expand our study to include other media processing applications such as speech, audio, communication, and natural language interaction.

## 7 Acknowledgments

We would like to thank Behnaam Aazhang, Mohit Aron, Rich Baraniuk, Joe Cavallaro, Tim Dorney, Aria Nostratinia, and Jan Odegard for numerous discussions on the behavior of media processing workloads. We would also like to thank Partha Tirumalai, Ahmad Zandi and Tony Zhang from Sun for useful pointers on enhancing the applications with VIS. We also thank Vijay Pai, Barton Sano, Chaitali Sengupta, and the anonymous reviewers for their valuable comments on earlier drafts of the paper.

## References

- [1] D. Bhandarkar and J. Ding. Performance characterization of the pentium pro processor. In *HPCA97*, pages 288–297, Feb 1997.
- [2] R. Bhargava et al. Evaluating MMX Technology Using DSP and Multimedia Applications. In *MICRO-31*, Dec 1998.
- [3] A. Bilas et al. Real-time Parallel MPEG-2 Decoding in Software. In *IPPS-11*, April 1997.
- [4] D. A. Carlson et al. Multimedia Extensions for a 550MHz RISC Microprocessor. In *IEEE Journal of Solid-State Circuits*, 1997.
- [5] T. M. Conte et al. Challenges to Combining General-Purpose and Multimedia Processors. In *IEEE Computer*, pages 33–37, Dec 1997.
- [6] K. Diefendorff and P. K. Dubey. How Multimedia Workloads Will Change Processor Design. In *IEEE Micro*, pages 43–45, Sep 1997.
- [7] J. Eyre. Assessing General-Purpose Processors for DSP Applications. Berkeley Design Technology Inc. presentation, 1998.
- [8] International Organisation for Standardisation – ISO/IEC JTC1/SC29/WG11MPEG 98/N2457. *MPEG-4 Applications Document*, 1998.
- [9] E. Killian. MIPS Extension for Digital Media with 3D. Slides presented at Microprocessor Forum, October 1996.
- [10] L. Kohn et al. The Visual Instruction Set (VIS) in UltraSPARC. In *COMPCON Digest of Papers*, March 1995.
- [11] C. Lee et al. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *MICRO-30*, 1997.
- [12] R. B. Lee. Subword Parallelism with MAX-2. In *IEEE Micro*, volume 16(4), pages 51–59, August 1996.
- [13] R. B. Lee and M. D. Smith. Media Processing: A New Design Target. In *IEEE MICRO*, pages 6–9, Aug 1996.
- [14] T. Mowry. *Tolerating Latency through Software-controlled data prefetching*. PhD thesis, Stanford University, 1994.
- [15] S. Oberman et al. AMD 3DNow! Technology and the K6-2 Microprocessor. In *HOTCHIPS10*, 1998.
- [16] V. S. Pai et al. RSIM: A Simulator for Shared-Memory Multiprocessor and Uniprocessor Systems that Exploit ILP. In *Proc. 3rd Workshop on Computer Architecture Education*, 1997.
- [17] V. S. Pai et al. The Impact of Instruction Level Parallelism on Multiprocessor Performance and Simulation Methodology. In *HPCA-3*, pages 72–83, 1997.
- [18] A. Peleg and U. Weiser. MMX Technology Extension to the Intel Architecture. In *IEEE Micro*, volume 16(4), pages 51–59, Aug 1996.
- [19] M. Phillip et al. AltiVec Technology: Accelerating Media Processing Across the Spectrum. In *HOTCHIPS10*, Aug 1998.
- [20] P. Ranganathan et al. The Interaction of Software Prefetching with ILP Processors in Shared-Memory Systems. In *ISCA24*, pages 144–156, 1997.
- [21] P. Ranganathan et al. Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors. In *ASPLOS8*, pages 307–318, 1998.
- [22] D. S. Rice. High-Performance Image Processing Using Special-Purpose CPU Instructions: The UltraSPARC Visual Instruction Set. Master's thesis, Stanford University, 1996.
- [23] M. Tremblay et al. VIS Speeds New Media Processing. In *IEEE Micro*, volume 16(4), pages 51–59, Aug 1996.
- [24] C.-L. Yang et al. Exploiting Instruction-Level Parallelism in Geometry Processing for Three Dimensional Graphics Applications. In *Micro31*, 1998.
- [25] D. F. Zucker. *Architecture and Arithmetic for Multimedia Enhanced Processors*. PhD thesis, Department of Electrical Engineering, Stanford University, June 1997.
- [26] D. F. Zucker et al. An Automated Method for Software Controlled Cache Prefetching. In *Proc. of the 31st Hawaii Intl. Conf. on System Sciences*, Jan 1998.