Performance Scalability of Multimedia Instruction Set Extensions

D. Cheresiz[1], B. Juurlink[2], S. Vassiliadis[2], and H. Wijshoff[1]

[1] LIACS, Leiden University, The Netherlands
[2] Computer Engineering Laboratory, Electrical Engineering Department, Delft University of Technology, The Netherlands

Abstract. Current media ISA extensions such as Sun's VIS consist of SIMD-like instructions that operate on short vector registers. In order to exploit more parallelism in a superscalar processor provided with such instructions, the issue width has to be increased. In the Complex Streamed Instruction (CSI) set exploiting more parallelism does not involve issuing more instructions. In this paper we study how the performance of superscalar processors extended with CSI or VIS scales with the amount of parallel execution hardware. Results show that the performance of the CSI-enhanced processor scales very well. For example, increasing the datapath width of the CSI execution unit from 16 to 32 bytes improves the kernel-level performance by a factor of 1.56 on average. The VIS-enhanced machine is unable to utilize large amounts of parallel execution hardware efficiently. Due to the huge number of instructions that need to be executed, the decode-issue logic constitutes a bottleneck.

1 Introduction

In the last years multimedia applications started to play an important role in the design of computing systems because they provide highly appealing and valuable services to the consumers, such as video-conferencing, digital content creation, speech recognition, virtual reality and many others. Applications like JPEG image compression/decompression, MPEG video and MP3 players are now common on most desktop systems. Multimedia applications impose high requirements on the performance of computing systems because they need to process huge amounts of data under stringent real-time constraints.

To meet these requirements, most processor vendors have extended their instruction set architectures (ISAs) with new instructions which allow key multimedia algorithms to be implemented efficiently. Examples of such extensions are Intel's *MMX* and *SSE* [12,14], Motorola's *AltiVec* [4] and Sun's *Visual Instruction Set (VIS)* [15]. All of them are, essentially, load-store vector architectures with short vector registers. These vector registers (also referred as multimedia registers) are usually 64 bits wide and contain vectors consisting either of eight 8-bit, four 16-bit, or two 32-bit elements. Multimedia instructions exploit SIMD parallelism by concurrently operating on all vector elements. More recent extensions (SSE, AltiVec) increase the register size to 128 bits allowing to process up to 16 8-bit values in parallel. Multimedia extensions have proven to provide significant performance benefits [13] by exploiting the data-level parallelism present in multimedia codes. However, these extensions have several features which are likely to hinder further performance improvements.

First, the fixed size of the multimedia registers limits the amount of parallelism that can be exploited by a single instruction to at most 8 (VIS, MMX) or 16 (SSE, AltiVec) parallel operations, while more parallelism is present in multimedia applications. In many important multimedia kernels, the same operation has to be performed on data streams containing tens to hundreds of elements. If existing ISA extensions are employed, such long data streams have to be split into sections which fit in the multimedia registers to process one section at a time. This results in a high number of instructions that need to be executed. Second, implementations of multimedia kernels with short-vector SIMD extensions require a significant amount of overhead for converting between different packed data types and for data alignment, increasing the instruction count even further. According to [13], for VIS up to 41% of the total instruction count constitutes overhead.

With the number of instructions being fixed, more instructions have to be fetched, decoded and executed in each cycle in order to increase performance. It is generally accepted, however, that increasing the issue width requires a substantial amount of hardware [5] and negatively affects the cycle time [11]. One way to achieve higher performance without increasing the issue width is to reduce the instruction traffic by increasing the size of the multimedia registers, but this approach implies a change of the ISA and, therefore, requires recompilation or even rewriting existing codes. Moreover, increasing the register size beyond 256 bits is not likely to bring much benefit, because many multimedia kernels process small 2-dimensional sub-matrices and only a limited number of elements, typically 8 or 16, are stored consecutively.

In order to overcome the limitations of short-vector SIMD extensions mentioned above, the *Complex Streamed Instruction* (CSI) set was proposed and has proven to provide significant speedups [8]. A single CSI instruction can process 2-dimensional data streams of arbitrary length, performing sectioning, aligning and conversion between different data formats internally in hardware.

In this paper we study how the performance of a superscalar processor enhanced with a short-vector SIMD extension (VIS) or with CSI scales with the amount of parallel execution hardware. This study is motivated by current trends in microprocessor technology, which exhibits growing levels of integration and transistor budgets. For example, in a contemporary 0.15-micron CMOS technology, a 32-bit adder requires less than 0.05 mm^2 of chip area [9], allowing designers to put dozens of such units on a single chip. The challenge is to feed these units with instructions and data. We explore the design space by varying key parameters of the processor, such as the issue width and instruction window size, in order to identify the performance bottlenecks.

This paper is organized as follows. In Section 2 we give a brief description of the CSI architecture and some related proposals. Section 3 describes the simulated benchmarks, the modeled processors and presents the experimental results. Conclusions and areas of future research are presented in Section 4.

2 Background

2.1 The CSI Architecture

In this section we briefly sketch the CSI multimedia ISA extension. Further details on the CSI architecture and a possible implementation can be found in [8]. CSI is a memory-to-memory architecture for two-dimensional streams. Usually, a CSI instruction fetches two input streams from memory, performs simple arithmetic operations on corresponding elements, and stores the resulting output stream back to memory. The streams follow a matrix access pattern, with a fixed *horizontal stride* (distance between consecutive elements of the same row) and a fixed *vertical stride* (distance between rows). The stream length is not architecturally fixed. The programmer visible state of CSI consists of several *sets of stream control registers (SCR-sets).* Each SCR-set consists of a several registers that hold the base address, the horizontal and vertical strides, the number of elements, etc., and completely specifies one stream.

The key advantages of the CSI architecture can be summarized as follows:

- **High parallelism**
  CSI increases the amount of parallelism that can be exploited by a single instruction. This is achieved by having no restriction on the stream length and by supporting 2-dimensional streams.
- **Reduced conversion overhead**
  CSI eliminates the overhead instructions required for converting between different packed data types by performing it internally in hardware and by overlapping it with useful computations.
- **Decoupling of data access and execution**
  Because the data streams in CSI are fully described by the SCR-sets, address generation, data access and execution can be decoupled. The CSI execution unit can generate addresses and send them to the memory hierarchy well before the data is used, thereby reducing the number of stalls due to late data arrival.
- **Compatibility**
  The same CSI code can be run without recompilation on a new implementation with a wider SIMD datapath and fully utilize it. This is possible because the number of elements that are actually processed in parallel is not part of the architecture.

2.2 Related Work

CSI instructions process two-dimensional data streams of arbitrary length stored in memory. Several early vector architectures such as the TI ASC and the Star-100 [6] were memory-to-memory as well. These architectures, however, suffered from a long startup time, which was mainly caused by execution of the overhead instructions needed for setting up the vector parameters and by long memory latency. The CSI implementation described in [8] does not suffer from these limitations for the following reasons. First, since CSI is implemented next to a fast superscalar core, the overhead instructions needed to set up the stream control registers take very little time. Second, the stream data is accessed through the L1 data cache. Because the L1 access time is short and the hit rates of the considered benchmarks are high, the time between the request for stream data and its arrival is short.

A related proposal that also exploits the 2-dimensional data-level parallelism available in multimedia applications is the Matrix Oriented Multimedia (MOM) ISA extension [2]. MOM instructions can be seen as vector versions of current SIMD media extensions. Two key features distinguish MOM from CSI. First, MOM is a register-to-register architecture (which implies sectioning when the streams do not fit into the MOM registers). Second, MOM requires overhead instructions for explicit data conversion.

Another related proposal is the *Imagine* processor [9], which has a load/store architecture for one-dimensional streams of data records. The SIMD processing hardware of Imagine consists of 8 arithmetic clusters. Each cluster is a VLIW engine with 6 functional units. Data records are distributed one per cluster and are processed in a SIMD fashion by executing the same sequence of VLIW instructions at each cluster. Contrary to CSI, which is an ISA extension and implemented in a CPU, Imagine is a stand-alone multimedia coprocessor.

3 Experimental Evaluation

The main goal of the experiments is to evaluate how the performance of a superscalar processor extended with VIS or CSI scales with the amount of SIMD execution hardware and to identify the performance bottlenecks. The benchmark set consists of MPEG-2 and JPEG codecs (*mpeg2encode, mpeg2decode, cjpeg, djpeg*) from the *Mediabench* suite [10] and of several image processing kernels (*add8, scale8, blend8, convolve3x3*) taken from the VIS Software Development Kit (VSDK) [16].

3.1 Tools and Simulation Methodology

In order to evaluate the performance of the VIS-enhanced and CSI-enhanced processors, we modified the `sim-outorder` simulator of the SimpleScalar toolset (version 3.0) [1]. This is a cycle-accurate execution-driven simulator of an out-of-order superscalar processor with a 5-stage pipeline based on the *Register Update Unit (RUU)*. A corrected version of the SimpleScalar memory model based on the SDRAM specifications given in [3] was used.

VIS- and CSI-executables of each *Mediabench* benchmark were obtained by manually rewriting the most time-consuming kernels in assembly. These are the *idct* and *Add_Block* routines in *mpeg2decode*, *dist1* in *mpeg2encode*, *idct* and *ycc_rgb_convert* in *djpeg*, and *rgb_ycc_convert* and *h2v2_downsample* in *cjpeg*. VIS- and CSI-executables of the VSDK kernels were obtained by completely rewriting them in assembly. Whenever available, we used the vendor-provided assembly codes for VIS.

We remark that out-of-order execution of CSI instructions is not allowed. Out-of-order execution of a CSI instruction with a potentially conflicting memory reference is also not allowed.

3.2 Modeled Processors

A wide range of superscalar processors was simulated by varying the issue width from 4 to 16 instructions per cycle and the instruction window size (i.e, the number of entries in the RUU unit) from 32 to 512. Table 1 summarizes the basic parameters of the processors and lists the functional unit latencies. Processors were made capable of processing VIS or CSI instructions by adding the VIS functional units or the CSI unit. The CSI unit is interfaced to L1 cache as described in [8].

Each time the issue width is doubled, the number of integer and floating-point units is scaled accordingly. The number of cache ports is fixed at 2, however, because multi-ported caches are expensive and hard to design (for example, currently there are no processors with a 4-ported cache). Note, that the CSI-enhanced processors do not require more ports to cache then the VIS-enhanced ones. The CSI-enhanced processors use two cache ports, sharing loads and stores generated by the CSI unit between them. The VIS- and the CSI-enhanced CPUs

Table 1. Processor parameters.

Clock frequency	666 MHz
Issue width	4/8/16
Instruction window size	16-512
Load-store queue size	8-128
Branch Prediction	
Bimodal predictor size	2K
Branch target buffer size	2K
Return-address stack size	8
Functional unit type and number	
Integer ALU	4/8/16
Integer MULT	1/2/4
Cache ports	2
Floating-point ALU	4/8/16
Floating-point MULT	1/2/4
VIS adder	2/4/8
VIS multiplier	2/4/8

FU latency/recovery (cycles)	
Integer ALU	1/1
Integer MUL	
multiply	3/1
divide	20/19
Cache port	1/1
FP ALU	2/2
FP MUL	
FP multiply	4/1
FP divide	12/12
sqrt	24/24
VIS adder	1/1
VIS multiplier	
multiply and pdist	3/1
other	1/1

Table 2. Memory parameters.

Instruction cache	ideal
Data caches	
L1 line size	32 bytes
L1 associativity	direct-mapped
L1 size	32 KB
L1 hit time	1 cycle
L2 line size	128 bytes
L2 associativity	2-way
L2 size	128 KB
L2 replacement	LRU
L2 hit time	6 cycles

Main memory	
type	SDRAM
row access time	2 bus cycles
row activate time	2 bus cycles
precharge time	2 bus cycles
bus frequency	166 MHz
bus width	64 bits

were simulated with different amounts of SIMD-processing hardware by varying the number of the VIS functional units and the width of the CSI unit's datapath, respectively.

The parameters of the cache and memory subsystems are summarized in Table 2. The memory latencies are expressed in memory clock cycles. A memory cycle is 6 ns, corresponding to a clock frequency of 166 MHz, which is typical for contemporary DRAM chips. The system bus (between the L2 cache and the memory controller) is also clocked at 166 MHz as in current PCs [7]. The ratio of CPU clock frequency to memory clock frequency was set to 4, corresponding to a CPU clock rate of 666 MHz.

3.3 Experimental Results

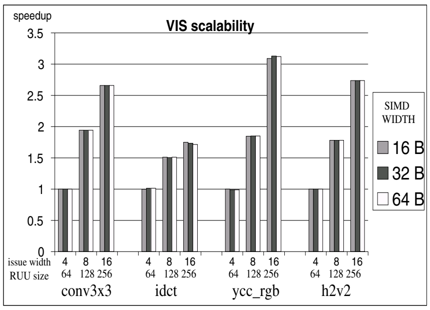
In this section we study the impact of the RUU size on the performance of the CSI- and VIS-enhanced processors. Then we analyze the performance behavior of these processors with respect to the number of SIMD functional units. After this, the performance bottleneck of the VIS-enhanced processors is identified. Finally, we present the speedups attained by CSI relative to VIS.

Increasing the Window Size. First, we study the influence of the instruction window (RUU) size on the performance. Because the RUU is a rather costly resource, it is important to minimize its size. The results (which are not presented here due to space limitations) show that for the studied kernels increasing the window size has a positive effect on the performance of a VIS-enhanced processor but does not influence the performance of a CSI-enhanced one. This is not surprising for the following reason. The kernels usually operate on long data streams, performing the same operation independently on all stream elements. In VIS, long data streams are split into sections which fit into VIS registers and a separate instruction is generated for each section. The instructions are independent which means that the VIS translates the parallelism present in the kernels into instruction-level parallelism (ILP). Larger instruction windows allow a VIS-enhanced superscalar CPU to expose and utilize larger amounts of ILP. On the other hand, in CSI the parallelism is exposed by a single CSI instruction which processes the whole stream. Therefore, a CSI-enhanced CPU does not need large instruction windows in order to exploit it. These results mean that the CSI-enhanced CPUs can be implemented with smaller (and cheaper) instruction windows then the VIS-enhanced ones.

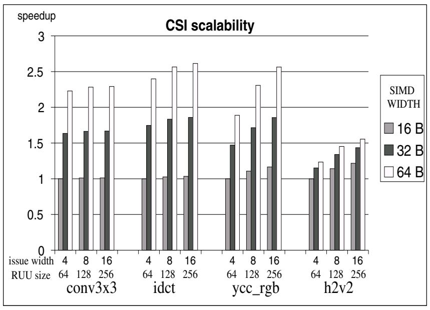
The simulation results also show that, for each issue width, increasing the RUU size beyond a certain limit yields diminishing returns. The speedup of the 4-way VIS-enhanced processor saturates when the RUU consists of 64 entries, the 8-way machine when the RUU size is 128, and the 16-way processor reaches its peak performance when the RUU size is 256. Therefore, we select these instruction window sizes both for VIS- and CSI-enhanced processors for all the experiments presented in the rest of the paper.

Increasing the Amount of Parallel Execution Hardware. The next question we consider is: how does the performance of the superscalar processors equipped with VIS or CSI scale with the amount of parallel execution hardware? This issue is important because of current trends in microprocessor technology. Larger scales of integration and, hence, growing transistor budgets allow to put dozens of functional units on a chip. The challenge for the designer is to keep these units utilized. The amount of SIMD processing hardware is characterized by the number of bytes which can be processed in parallel. For VIS this is determined by the number of VIS adders and multipliers. For example, to operate on 16 bytes in parallel, the machines are configured with 2 VIS adders and 2 VIS multipliers. For CSI, the amount of parallelism is determined by the width of the datapath of the CSI execution unit. So, in order to process 32 or 64 bytes in parallel the number of VIS units of the VIS-enhanced processor is doubled or quadrupled, whereas for CSI, the width of its datapath is increased appropriately.

As the baseline processors, we considered the 4-, 8-, and 16-way superscalar CPUs with instruction windows of 64, 128, and 256 entries, respectively. Each baseline processor was equipped with sufficient SIMD hardware so that it can process 16, 32, or 64 bytes in parallel. Let a VIS-extended processor with an issue width of x, a window size of y, and a SIMD processing width of z bytes be denoted by $VIS(x, y, z)$. A similar notation will be used for CSI-enhanced CPUs. Figure 1(a) depicts the speedups attained by all simulated VIS-enhanced processors relative to $VIS(4, 64, 16)$. The speedups achieved by all simulated CSI-enhanced CPUs relative to $CSI(4, 64, 16)$ are plotted in Figure 1(b). All VIS and CSI kernels exhibited similar behavior and, therefore, we only present the performance figures of four selected kernels. Figure 1(a) shows that, for a fixed issue width, increasing the number of the VIS functional units does not yield any benefit. Contrary to VIS, CSI is able to utilize additional SIMD execution hardware (see Figure 1(b)), and its performance scales almost linearly with the amount of SIMD resources. It can also be observed that the performance of the



(a)



(b)

Fig. 1. Scalability of the CSI and VIS performance w.r.t. available SIMD hardware.

CSI-extended processors is insensitive to the issue width. This is expected since CSI does not exploit instruction-level parallelism and, therefore, does not need to issue a lot of instructions in parallel.

Figure 2 presents the speedups attained by CSI-enhanced processors with respect to VIS-extended processors equipped with the same amount of SIMD execution hardware, i.e., the figure shows the speedups of $\mathit{CSI}(x, y, z)$ over $\mathit{VIS}(x, y, z)$ for various values of x, y, and z. It shows that CSI clearly outperforms VIS, especially for the 4-way and 8-way configurations. We remark that these processors are less costly and can probably achieve a higher clock frequency than a 16-way one.

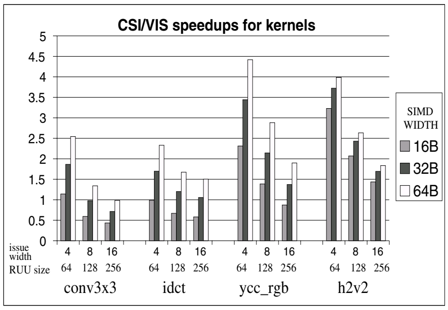


Fig. 2. Speedup of CSI over VIS for several kernels.

Identifying the Bottleneck. It is important to identify where the performance bottleneck of the VIS-enhanced machines lies. Figure 1(b) shows that the VIS-enhanced machines perform better when the issue width is increased. This suggests that the issue width is the limiting factor for VIS. To investigate this, we study the IPCs attained by the VIS-enhanced CPUs. For each issue width, Figure 3 depicts the attained IPCs normalized to the corresponding ideal IPCs (e.g., for a 4-way machine, the ideal IPC is 4). It shows that, for most kernels,

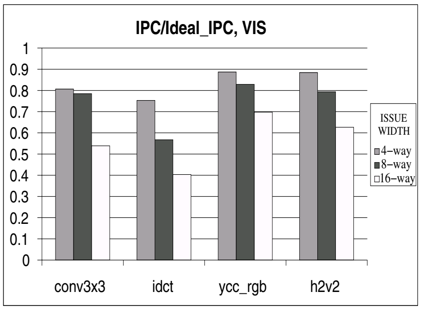
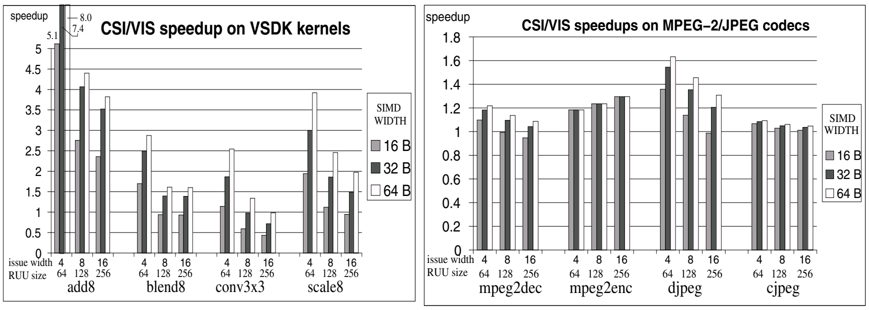


Fig. 3. Ratio of achieved IPC to ideal IPC for several VIS kernels and issue widths.



(a) (b)

Fig. 4. Speedup of CSI over VIS on VSDK kernels and MPEG/JPEG codecs.

the performance achieved by the 4-way and 8-way superscalar VIS-enhanced processors is close to ideal. These CPUs achieve IPCs which are within 78% to 90% of the ideal IPC for all kernels except *idct*. This means that they cannot be accelerated by more than 11-28% $(1/0.9 - 1/0.78)$ and, therefore, even when they achieve the ideal IPC their performance will not approach that of the corresponding CSI-enhanced CPUs. This shows that the issue width indeed limits the performance of the VIS-enhanced CPUs. The IPCs attained by the 16-way machine are far from the ideal and are within 55% to 70% of the ideal IPC, for all kernels except *idct*, meaning that the accelerations of 42-81% are possible. However, to achieve such increases in IPC a very accurate branch predictor is needed.

In conclusion, Figure 4 depicts the VSDK kernel and application speedups achieved by CSI over VIS. Of course, the speedups achieved by CSI on MPEG-2/JPEG codecs are smaller than those achieved on VSDK kernels due to Amdahl's law. Still rather impressive application-level speedups are achieved by CSI, especially for smaller (and more realistic) issue widths of 4 and 8. For example, when the issue width is 4 and 32 bytes can be processed in parallel, CSI achieves speedups ranging from to 1.08 (*cjpeg*) to 1.54 (*djpeg*) with an average of 1.24.

4 Conclusions

In this paper we have evaluated how the performance of the CSI and VIS multimedia ISA extensions scales with the the amount of SIMD parallel execution hardware. We have also performed experiments to identify the bottlenecks of the VIS-enhanced superscalar CPUs. Out results can be summarized as follows:

- The kernel-level performance of CSI increases almost linearly with the width of CSI datapath. It improves by a factor of 1.56 when the width of the CSI datapath doubled and by an additional factor of 1.27 when the width is quadrupled. Furthermore, the performance of the CSI-enhanced processor is not sensitive to the issue width.
- The VIS-enhanced CPUs do not perform substantially better when the number of VIS functional units is increased. The issue width is the limiting factor and the decode/issue logic, therefore, constitutes a bottleneck.

These results have the following implications for multimedia ISA and processor design. To increase the performance of a CPU extended with a short-vector media ISA there are two main options. The first one is to reduce the pressure on the decode-issue logic by increasing the size of the multimedia registers. The second one is to increase the issue width of the CPU. Both of them bear huge costs: software incompatibility for the first and expensive hardware for the second. CSI offers a solution which provides scalable performance without having to increase the issue width as well as software compatibility.

We plan to further investigate the CSI paradigm in several directions. For example, since the interface of the CSI execution unit with the memory subsystem is not fixed, a designer has freedom in its implementation. The CSI execution unit can be interfaced to the L1 cache, L2 cache or even main memory. We plan to evaluate such implementations.

References

1. D. Burger and T.M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, Univ. of Wisconsin-Madison, Comp. Sci. Dept., 1997.
2. Jesus Corbal, Mateo Valero, and Roger Espasa. Exploiting a New Level of DLP in Multimedia Applications. In *MICRO 32*, 1999.
3. M. Gries. The Impact of Recent DRAM Architectures on Embedded Systems Performance. In *EUROMICRO 26*, 2000.
4. L. Gwennap. AltiVec Vectorizes PowerPC. *Microprocessor Report*, 12(6), 1998.
5. J.L. Hennessy and D.A. Patterson. *Computer Architecture - A Quantitative Approach.* Morgan Kaufmann, second edition, 1996.
6. Kai Hwang and Faye A. Briggs. *Computer Architecture and Parallel Processing.* McGraw-Hill, second edition, 1984.
7. PC SDRAM Specification,Rev 1.7. Intel Corp., November 1999.
8. B. Juurlink, D. Tcheressiz, S. Vassiliadis, and H. Wijshoff. Implementation and Evaluation of the Complex Streamed Instruction Set. In *Int. Conf. on Parallel Architectures and Compilation Techniques (PACT)*, 2001.

9. B. Khailany, W.J. Dally, U.J. Kapasi, P. Mattson, J. Namkoong, J.D. Owens, B. Towles, A. Chang, and S. Rixner. Imagine: Media Processing With Streams. *IEEE Micro*, 21(2):35–47, 2001.
10. C. Lee, M. Potkonjak, and W.H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communication Systems. In *MICRO 30*, 1997.
11. S. Palacharla, N.P. Jouppi, and J.E. Smith. Complexity-Effective Superscalar Processors. In *ISCA'97*, 1997.
12. Alex Peleg, Sam Wilkie, and Uri Weiser. Intel MMX for Multimedia PCs. *Communications of the ACM*, 40(1):24–38, January 1997.
13. P. Ranganathan, S. Adve, and N.P. Jouppi. Performance of Image and Video Processing with General-Purpose Processors and Media ISA Extensions. In *ISCA 26*, pages 124–135, 1999.
14. Shreekant Thakkar and Tom Huff. The Internet Streaming SIMD Extensions. *Intel Technology Journal*, May 1999.
15. Marc Tremblay, J. Michael O'Conner, Venkatesh Narayanan, and Lian He. VIS Speeds New Media Processing. *IEEE Micro*, 16(4):10–20, August 1996.
16. VIS Software Developer's Kit.
    Available at `http://www.sun.com/processors/oem/vis/`.