Performance of the Complex Streamed Instruction Set on Image Processing Kernels

Dmitri Tcheressiz[1], Ben Juurlink[2], Stamatis Vassiliadis[2], and Harry Wijshoff[1]

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

Abstract. The Complex Streamed Instruction (CSI) set is an architectural paradigm designed to accelerate multimedia applications. These applications are characterized by streaming operations on small-width data elements such as 8-bit pixels or 16-bit audio samples. CSI instructions operate on two-dimensional data streams in a SIMD fashion and are able to process streams of arbitrary length. In this paper we evaluate the performance of the CSI architecture on a set of important image processing kernels. These kernels are characterized by little data reuse which results in poor cache performance. Simulation results show that CSI provides a speedup by a factor of up to 3.98 (2.60 on average) compared to Sun's media ISA extension VIS. We also analyze the scalability of VIS and CSI with respect to memory bandwidth. The results show that CSI scales much better than VIS with increasing bandwidth.

1 Introduction

Multimedia applications have begun to play an important role in our lives and are expected to do so even more in the future [4,12]. In reaction to this trend, many vendors have extended their instruction set architecture (ISA) with instructions targeted to multimedia applications. Examples are MMX [16], VIS [18], and MDMX [13]. These instructions exploit SIMD parallelism at the subword level, i.e., they operate concurrently on, e.g., 8 bytes packed in one 64-bit register.

It has been shown that these multimedia ISA extensions can improve the performance of many multimedia codes (see, e.g., [1,17]). Nevertheless, they have several limitations which can be summarized as follows:

- The multimedia register size is architecturally visible and fixed. So, in order to exploit more data parallelism, there are two options. One is to add more multimedia functional units and increase the issue width. However, it is generally accepted that this requires a substantial amount of hardware and may negatively affect the cycle time [7,15]. A second option is to increase the multimedia register size, thereby changing the ISA. However, this implies that existing codes have to be recompiled or rewritten.
- Another drawback of increasing the multimedia register size is that it may not be beneficial, because multimedia kernels often operate on sub-matrices and the vector length in both directions is rather small.

R. Sakellariou et al. (Eds.): Euro-Par 2001, LNCS 2150, pp. 678–686, 2001.

– Codes employing SIMD-like instructions typically incur a lot of overhead for converting between different packed data types and alignment-related instructions.

Previously, we proposed the *Complex Streamed Instruction* set that avoids these limitations [19,10]. However, the considered benchmarks exhibited very high cache hit rates (98-99%) and were compute-bound. The purpose of this paper is to evaluate the performance of the CSI extension on memory-bound benchmarks. In particular, we are interested in the scalability of the CSI architecture when the memory bandwidth improves while the latency does not. This interest is motivated by the observation that new memory chip generations such as SDRAM [9] and RAMBUS [5] yield higher bandwidths but not much shorter access latencies. To evaluate the efficiency of CSI in such systems, we selected several image processing kernels that exhibit low hit rates, and simulated their execution on a superscalar processor, on the same processor extended with VIS [18], and on the same processor augmented with CSI.

This paper is organized as follows. In Section 2 the CSI architecture is briefly described. Section 3 describes the benchmarks, the modeled processors and presents the experimental results. Related work is discussed in Section 4, and conclusions and topics for future research are given in Section 5.

2 The CSI Architecture

In this section we briefly describe the CSI multimedia ISA extension and list some of its benefits. More details can be found in [10].

CSI is a memory-to-memory architecture for two-dimensional streams. Most CSI instructions load two data input streams from memory, operate on them element-wise, and store the resulting stream back to memory. The streams can be of any length and follow a matrix access pattern. A stream is specified by a *stream control register set* (SCR-set) that signifies the base address, the stream length, the horizontal and vertical strides, and the number of stream elements in a row. Each SCR-set also contains a control register that specifies the size and sign of the stream elements, if saturation or modular arithmetic should be performed, and the position of the binary point in fixed-point calculations.

The main advantages of the CSI instruction set can be listed as follows:

– It increases the amount of data-level parallelism that can be exploited. This cannot be achieved by enlarging the multimedia registers, since the vector length in both directions is typically 8 or 16 bytes.
– Since streams can be of any length, the number of elements that are processed in parallel (the *section size*) does not appear explicitly in the code. Instead, the hardware is responsible for dividing streams into sections. This implies that existing codes do not have to be recompiled or rewritten when the width of the SIMD datapath is increased.
– Conversion between different packed data types (if required) is performed internally in hardware.

– It eliminates loop control and address generation overheads, since one CSI instruction can replace two embedded loops.

3 Experimental Evaluation

To evaluate the performance of CSI on image processing kernels, we simulated their execution on three different processors: a 4-way superscalar processor, the same processor extended with VIS, and the same processor augmented with CSI. The following benchmarks taken from the VIS Software Development Kit (VSDK) were selected: `add8` (adding two images using mean of corresponding pixels), `blend8` (alpha-blending of two images), `scale8` (linear scaling) and `convolve3x3` (convolution with a 3x3 kernel). As input, we used 332x345 images in Sun rasterfile format with 3 color components. These benchmarks are characterized by little data reuse which results in poor cache performance (the average L1 hit rate is 79%). The memory bandwidth, therefore, is a decisive factor for overall system performance.

3.1 Simulation Methodology and Tools

We used the `sim-outorder` simulator of the SimpleScalar toolset (version 2.0) [2]. This execution-driven simulator emulates an out-of-order superscalar processor. Its architecture (Portable ISA or PISA) is derived from the MIPS-IV ISA. CSI and VIS instructions were synthesized by using annotations to instructions in the assembly files.

Three different executables of each kernel were created: a baseline PISA version, a VIS version, and a CSI version. The baseline PISA versions were obtained by compiling the C code taken from the VSDK using the `gcc` compiler with option `-O4`. For VIS and CSI we manually rewrote the assembly files, using for VIS a 1-1 translation of the VIS codes provided in the VSDK.

Since VIS instructions are register-to-register and operate on the floating-point register file, they do not interfere with the existing processor pipeline. CSI instructions are memory-to-memory and require extra care: one must ensure that the execution of a CSI instruction does not overlap with scalar memory instructions. We, therefore, took the following conservative approach. When a CSI instruction is detected, the pipeline is stalled. The processor then waits until all memory instructions have committed, after which it issues the CSI instruction. Fetching and decoding resumes after the CSI instruction has finished.

3.2 Modeled Processors

The base system is a 4-way superscalar processor with out-of-order issue and execution running at 500 MHz. Its main parameters are listed in Table 1.

The VIS-enhanced processor is modeled after the UltraSPARC [18], except that we assumed two 64-bit VIS multipliers whereas the UltraSPARC has only

Table 1. Processor parameters

Clock frequency	500 MHz
Issue width	4-way
Reorder buffer size	16
Load-store queue size	8
Branch Prediction	
Bimodal predictor size	2K
Branch target buffer size	2K
Return-address stack size	8
Functional unit type and number	
Integer ALU	4
Integer MULT	1
Cache ports	2
Floating-point ALU	4
Floating-point MULT	1
VIS adder	2
VIS multiplier	2

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	page-mode
page size	4 KB
first page access	30 cycles
next page access	10 cycle
bus clock frequency	100 MHz
bus width	8/16/32 bytes

one. We chose this configuration because CSI instructions are assumed to process two 128-bit packed data types in parallel. Any speedup of CSI over VIS, therefore, does *not* result from exploiting more data-level parallelism.

The cache and memory parameters are summarized in Table 2. The memory latencies are expressed in CPU clock cycles. Converted to absolute time, they correspond to access latencies of 20-60ns, which is close to those of contemporary DRAM chips. The frontside memory bus (between the L2 cache and the memory controller) is clocked at 100 MHz as in current PCs [9]. In order to study the effect of memory bandwidth on the performance of the modeled processors, we vary the bus width from 8 bytes (current PC standard) to 16 and 32 bytes, which corresponds to bandwidths ranging from 0.8 to 3.2 GB/sec. Contemporary PCs have a memory bandwidth of 0.8 GB/s using a 64-bit wide bus and *DDR SDRAM* [14] and *Direct Rambus* [5] already provide 1.6 GB/s of bandwidth.

3.3 Experimental Results

In this section the performance improvements attained by the CSI and VIS extensions are presented and discussed. We also analyze the different components of the execution time in order to identify the bottlenecks.

Figure 1 depicts the execution times of the baseline, VIS-enhanced and CSI-enhanced processor on each benchmark. The bars labeled '8b', '16b', '32b' depict the execution time when the bus width is 8 bytes, 16 bytes, or 32 bytes, respectively. The execution time is normalized w.r.t. the time taken by the baseline superscalar system with an 8-byte wide bus.

Figure 1 shows that the baseline system is compute-bound and hardly benefits from higher memory bandwidths. This is evidenced by the fact that the `add8` kernel (which requires a small amount of computation per pixel) scales better than `convolve3x3` (which is computationally more expensive). The VIS ISA extension significantly improves performance across all benchmarks. Nevertheless, the processor core becomes the bottleneck when the bandwidth is increased from 1.6 GB/s to 3.2 GB/s, since the performance improvements are smaller than when going from 0.8 GB/s to 1.6 GB/s. The CSI-enhanced processor provides additional performance gains ranging from a factor of 1.2 to 3.9. This can be seen more clearly from Figure 2, which depicts the speedups of CSI over VIS. Observe that the speedup increases with the bus width.

In order to further identify the performance bottlenecks, we analyze the different components of the execution time. Every cycle is classified either as *non-*

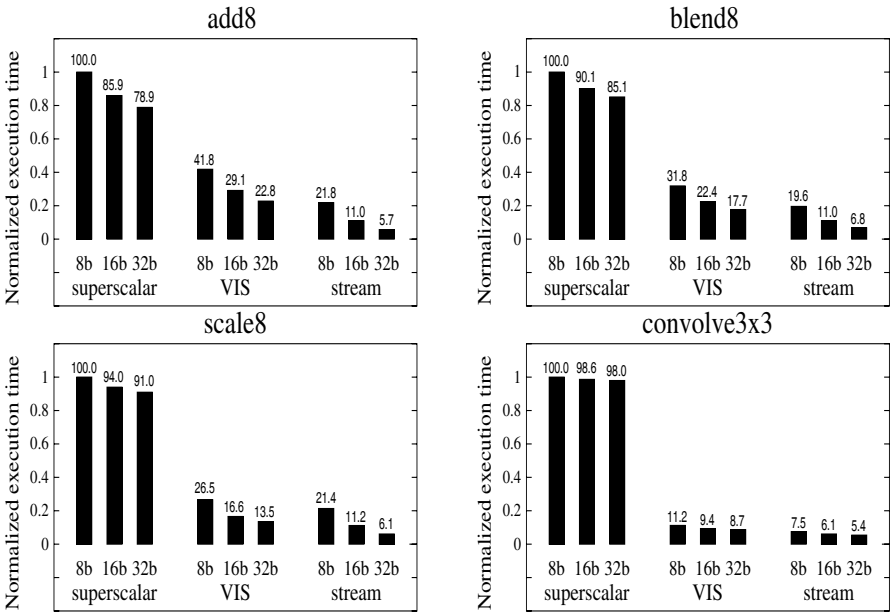

Fig. 1. Normalized execution time of the image processing kernels

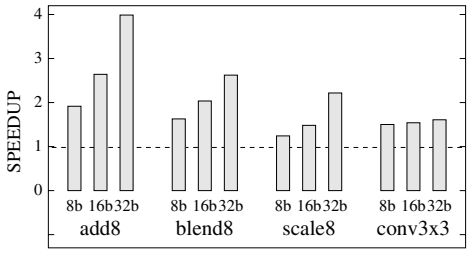

Fig. 2. Speedup of CSI over VIS

stall, *cpu stall* or *memory stall*. A cycle is classified as non-stall if the number of instructions retired that cycle is equal to the issue rate (4). If not, we look at the first instruction that could not retire. If it is a memory access instruction, it is classified as memory stall. Otherwise, it is classified as cpu stall. Since instructions retire in order, the first instruction that cannot retire stalls all instructions behind it. We used this metric instead of busy time used in [17] because it clarifies why less than the maximum number of instructions retired.

For the CSI architecture, a cycle is classified as non-stall if a SIMD operation was initiated at that cycle. Otherwise, it is classified as memory stall, meaning that data has not yet been delivered by the memory. The CSI-enhanced processor does not experience cpu stall cycles because there are no dependencies between stream elements, and there are enough resources to initiate a SIMD operation every cycle.

Figure 3 breaks down the execution times in non-stall, cpu stall and memory stall cycles. It shows that neither the superscalar nor the VIS-enhanced processor can fully utilize the increased memory bandwidth. The superscalar system gets saturated already when the bus width is 8 bytes, since the CPU component of the execution time (non-stall plus cpu stall) increases only marginally. The VIS-enhanced system utilizes the available bandwidth much better, but the growth of its CPU component is mostly due to the cpu stall component. Increasing the memory bandwidth of the CSI-enhanced system, on the other hand, leads to a significant reduction in memory stall cycles and a corresponding increase in non-stall cycles.

We remark that some execution may occur during cpu stall cycles. The fact that a given cycle is cpu stall means that a certain instruction was not able to retire (i.e., its result was not ready) although it was allowed. There can be multiple reasons why an instruction did not finish, the main being the following. Either the instruction was issued and started execution too late because no functional unit was available, or the instruction was decoded and placed in the reorder buffer too late because there were no free entries.

These observations suggest that the performance of the superscalar and VIS-enhanced processors may be improved by increasing the number of functional units and/or by increasing the size of the reorder buffer. We conducted experiments using these techniques. Adding 2 VIS adders and 2 VIS multipliers pro-

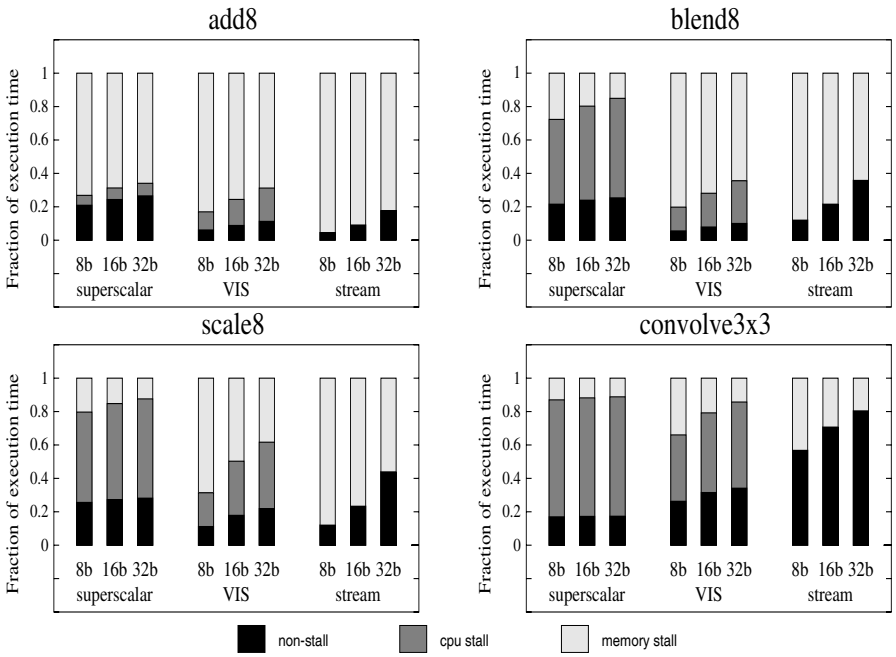

Fig. 3. Execution time partitioned in non-stall, cpu stall and memory stall cycles

duced no effect. Doubling the size of the reorder buffer and the load/store queue indeed improved performance (for the VIS-enhanced system by a factor of up to 1.20). However, this improvement is still significantly less than that achieved by CSI (cf. Figure 2). This shows that the performance of the superscalar and VIS architectures is limited by the execution engine and cannot be improved much by low-cost techniques such as resource duplication. So, it seems that to improve performance one has to pack more parallel operations in a single instruction, but this may result in binary incompatibility.

4 Related Work

The CSI architectural paradigm was introduced in [19]. A detailed description of the CSI instruction set, its implementation and performance on video and image codecs can be found in [10]. Here we extend that work by studying a new application domain with different memory behavior and by evaluating the effect of memory bandwidth.

CSI is a memory-to-memory architecture for two-dimensional streams. Early vector architectures such as the TI ASC and the Star-100 [8] were also memory-to-memory and could also process vectors of arbitrary length. However, these architectures suffered from long startup times, which was mainly due to over-

head instructions needed for setting up the parameters and to long memory latency. Since CSI is implemented next to a superscalar core, the overhead needed for setting the parameters is small. Furthermore, because the streams are two-dimensional, they are longer than one-dimensional vectors, allowing the memory latency to be overlapped with execution.

The performance of image processing benchmarks on superscalar processors with and without the VIS extension was also studied by Ranganathan et al. [17]. They studied the effect of varying the L1 and L2 cache size and of prefetching, but not the effect of memory bandwidth.

A related proposal aimed at exploiting more data-level parallelism in multimedia applications is the Matrix Oriented Multimedia (MOM) extension [3]. MOM instructions can be seen as vector versions of current SIMD media extensions. Two main differences between MOM and CSI are that MOM has architectured registers (and, hence, sectioning) and requires explicit data conversion instructions.

Another related proposal is the Imagine processor [11], which has a load/store architecture for one-dimensional streams of data records. It is centered around a large, 128KB stream register file, and consists of 48 functional units grouped in 8 arithmetic clusters. Each cluster must execute the same VLIW operation and is controlled by a microcoded controller. Imagine is suited for applications performing many arithmetic operations on each element of a long, one-dimensional stream. It seems less suited when only a few operations on each record are performed and when the vector length is small.

5 Conclusions

In this paper we have evaluated the performance of the CSI multimedia ISA extension on a set of image processing kernels characterized by little data reuse. We have also studied the scalability of the proposed architecture with respect to the memory bandwidth. The results show:

- On the high-bandwidth system (3.2 GB/s), the CSI extension improves performance by a factor of 1.6 to 3.9 (2.6 on average) compared to the VIS-enhanced processor.
- On the low-bandwidth system (0.8 GB/s), the speedups range from 1.2 to 1.9 with an average of 1.56.
- Neither the superscalar nor the VIS-enhanced processors are able to fully utilize the high memory bandwidth provided by current high-performance and future generations DRAM architectures. We identified the CPU core as the bottleneck and observed that increasing the size of the reorder buffer and the number of functional units does not alleviate the problem.

We plan to continue our investigation of CSI in the following directions. First, we want to further improve the memory system performance for CSI instructions by exploiting the information available in the stream control registers. This information could be communicated to an intelligent DRAM controller such as

proposed in [6]. Second, we would like to apply our approach to another important application domain characterized by abundant data-level parallelism and high computational demands, namely 3D graphics.

References

1. R. Bhargava, L. K. John, B. L. Evans, and R. Radhakrishnan. Evaluating MMX Technology Using DSP and Multimedia Applications. In *MICRO 31*, 1998. 678
2. D. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report 1342, Univ. of Wisconsin-Madison, Comp. Sci. Dept., 1997. 680
3. J. Corbal, M. Valero, and R. Espasa. Exploiting a New Level of DLP in Multimedia Applications. In *MICRO 32*, 1999. 685
4. K. Diefendorff and P. K. Dubey. How Multimedia Workloads Will Change Processor Design. *IEEE Computer*, 30(9):43–45, September 1997. 678
5. Direct RDRAM 64/72-Mbit data sheet. Document available via `http://www.rambus.com/docs/64dDDS.pdf`, 1998. 679, 681
6. S.McKee et al. Design and Evaluation of Dynamic Access Ordering Hardware. In *Proc. of the 10th ICS*, 1996. 686
7. J. L. Hennessy and D. A. Patterson. *Computer Architecture - A Quantitative Approach*. Morgan Kaufmann, second edition, 1996. 678
8. K. Hwang and F. A. Briggs. *Computer Architecture and Parallel Processing*. McGraw-Hill, second edition, 1984. 684
9. PC SDRAM Specification,Rev 1.7. Intel Corp., November 1999. 679, 681
10. B. Juurlink, D. Tcheressiz, S. Vassiliadis, and H. Wijshoff. Implementation and Evaluation of the Complex Streamed Instruction Set. In *PACT-2001*, 2001. To appear. 679, 684
11. 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, March/April 2001. 685
12. R. B. Lee and M. D. Smith. Media Processing: A New Design Target. *IEEE Micro*, 16(4):6–9, August 1996. 678
13. MIPS Extension for Digital Media with 3D. Document available via `http://www.mips.com/Documentation/isa5_tech_brf.pdf`, 1996. 678
14. 256Mbit DDR SDRAM part specification. Document available via `http://samsungelectronics.com/semiconductors/DRAM/DDR_SDRAM/256M_bit/K4H560438B/256mB_R03.PDF`, 2000. 681
15. S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-Effective Superscalar Processors. In *ISCA 24*, 1997. 678
16. A. Peleg, S. Wilkie, and U. Weiser. Intel MMX for Multimedia PCs. *Comm. of the ACM*, 40(1):24–38, January 1997. 678
17. 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*, 1999. 678, 683, 685
18. M. Tremblay, J. Michael O'Conner, V. Narayanan, and L. He. VIS Speeds New Media Processing. *IEEE Micro*, 16(4):10–20, August 1996. 678, 679, 680
19. S. Vassiliadis, B. Juurlink, and E. Hakkennes. Complex Streamed Instructions: Introduction and Initial Evaluation. In *EUROMICRO 26*, pages 400–408, 2000. 679, 684