Performance Benefits of Special-Purpose Instructions in the CSI Architecture

Dmitry Cheresiz, Ben Juurlink, Stamatis Vassiliadis
Computer Engineering Laboratory,
Electrical Engineering Department,
Delft University of Technology,
Delft, The Netherlands
{cheresiz, benj, stamatis}@dutepp0.et.TUDelft.NL

Abstract— **The Complex Streamed Instruction Set (CSI) architecture was proposed in order to overcome the limitations of existing multimedia-oriented ISA extensions, such as Intel's *MMX* and *SSE*. One of the main limitations is the large amount of instructions which has to be executed. In CSI, instructions operate on data streams of arbitrary-length, which allows to dramatically reduce the instruction counts for the kernels with sufficient amount of data-level parallelism. Previously, we have shown that CSI provides impressive performance improvements for several important multimedia kernels and applications. In these experiments the kernels were coded using elementary arithmetic CSI instructions such as addition, multiplication, etc. Many kernels, however, perform more complex operations and, thus, need to be translated to multiple elementary CSI instructions. For some kernels, CSI provides special-purpose instructions that collapse several elementary operations in a single complex one and thereby achieve an additional reduction of the number of executed instructions. In this paper we study the performance provided by an example of such an instruction. Using the *SimpleScalar* simulator, we evaluate performance of the superscalar CPU augmented with the CSI execution unit on the *Paeth prediction* kernel of the image encoder/decoder for the *PNG* image compression standard. Simulation results show that the kernel-level performance of the 4-way superscalar CPU augmented with the CSI execution unit improves by the factor of 12.9x when this kernel is implemented using the special-purpose CSI instruction instead of the elementary ones.**

Keywords— **Processor Architecture, Multimedia ISA Extensions, Data-level parallelism**

I. Introduction

The growing importance of multimedia applications for the desktop market motivated major processor vendors to extend the instruction set architectures (ISAs) with instructions that can be used to implement key multimedia algorithms efficiently. Examples of such extensions are the *Visual Instruction Set (VIS)* for the *UltraSPARC* architecture and *MMX* for the *x86* architecture [10], [12]. These extensions are, essentially, load-store vector architectures with short vector registers, which are called the multimedia registers. In each 64-bit vector register a vector consisting of eight 8-bit, four 16-bit, or two 32-bit elements can be stored. The instructions provided in VIS and MMX exploit the data-level parallelism present in multimedia codes by operating on all vector elements in parallel.

The ISA extensions mentioned above have proven to provide significant performance benefits [11]. These benefits, however, are limited because of several characteristics of short-vector SIMD extensions. First, the fixed size of the registers limits the number of parallel operations performed by a single VIS or MMX instruction to at most 8. Much higher amounts of parallelism are present in many multimedia kernels where the same operation often has to be performed on data streams consisting of tens to hundreds of elements. To implement such kernels using VIS or MMX, long streams have to be split into sections that fit into the 64-bit registers. This process, called *sectioning*, results in a large number of instructions that have to be executed. Second, VIS and MMX implementations of multimedia kernels require a significant number of overhead instructions needed for aligning the data and converting it between different packed data types, increasing the instruction count even further. According to [11], up to 41% of the total instruction count for VIS constitutes overhead. High dynamic instruction counts increase the pressure on the decode-issue logic of a superscalar processor. For example, in [3] we showed that, provided enough functional units are available in the processor, for the 4-way and 8-way issue CPUs, the processor issue width limits the performance on the MPEG-2 and JPEG codecs.

The *Complex Streamed Instruction (CSI) Set* was proposed in order to overcome the limitations of short-vector SIMD extensions mentioned above. Two-dimensional streams of arbitrary length can be processed by a single CSI instruction which performs the actual computation, as well as memory accesses, sectioning, aligning and conversion between different data formats. Previous studies

of the superscalar processors enhanced with a CSI execution unit show that, for a wide range of media applications, such processors exhibit dramatic reduction of the instruction counts and execution time, comparing to the same processors enhanced with existing commercial media extensions, such as *VIS* [8], [4], [3]. In these studies, the compute-intensive kernels were coded using elementary arithmetic CSI instructions such as addition, multiplication, etc. Many important kernels, however, perform more complex operations and, therefore, are translated into multiple elementary CSI instructions. We remark that an interesting scenario could occur if the multiple operations required for the main computation could be grouped and compounded into a single complex operation which doesn't require more cycles than the simple ones. In such a case, a new instruction that specifies this complex operation can be introduced. This single complex instruction can then be used to substitute the multiple simple instructions which are used to implement the main operation. It was shown that the main computations which are performed by a number of important media kernels, such as the *Sum of Absolute Differences (SAD)* of the MPEG-2 encoder, the *Paeth Prediction* of the *PNG* image codec, and others, can be implemented at acceptable cost in dedicated hardware and, therefore, lend themselves to such a technique [7], [13]. In this paper we extend CSI with the special-purpose instruction which computes the Paeth prediction. We use the corresponding kernel to study the performance benefits provided by this new CSI instruction in order to decide if the addition of dedicated hardware required for it can be justified. The paper is structured as follows. In Section II we present a brief overview of the CSI architecture and extend CSI with the special-purpose instruction for Paeth prediction. Section III sketches an example implementation the CSI execution unit. In Section IV we present the experimental results. The conclusions are drawn in Section V.

II. CSI Architecture

In this section we briefly review the CSI architecture and then extend it with the special-purpose instruction for acceleration of the Paeth prediction scheme.

CSI Architecture Overview. CSI is a memory-to-memory vector-like architecture. CSI instructions process arbitrary-length streams of data located in memory. A single instruction is responsible for loading source stream data, converting it (if necessary) from storage to computation format, performing the main computation and storing the results. For example, the `csi_add` instruction loads two input streams from memory, adds their corresponding elements, and writes the resulting stream back to memory. Stream elements follow a two-dimensional access pattern, as shown in Figure 1 Each stream consists of an arbitrary number of rows, and the row elements are stored at a fixed stride which will be referred to as the *horizontal stride*. There is also a fixed stride between consecutive rows, which will be referred to as the *vertical stride*. Such an access pattern is typical for media kernels. All the elements of a stream have the same data type, which should be one of the following 7 standard arithmetic types: 8-, 16-, or 32-bit binary integer (signed or unsigned), or the 32-bit single precision floating-point format (IEEE 754 format). The strides, together with the address of the first element (called the *Base*), and some additional parameters, which describe the format of the stream elements, completely specify the stream. All these parameters of a stream are stored in a *set of stream control registers* (SCR-set), which consists of eight 32-bit registers. Programmer-visible state of CSI contains 16 SCR-sets. CSI instructions address their stream operands by addressing the corresponding SCR-sets. For details of the CSI architecture, such as format and functions of individual registers within each SCR-set, the reader is referred to [8].

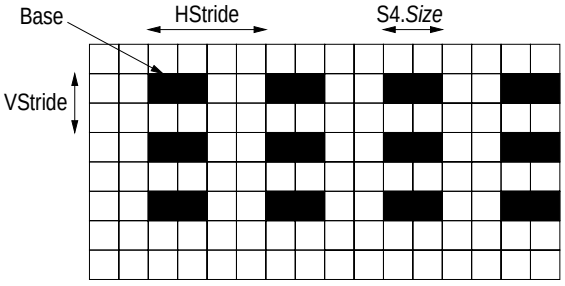

Fig. 1. Two-dimensional stream format. Each box represents a byte. Filled boxes are stream elements.

Extending CSI with the Paeth instruction. Figure 2 presents the C-code of the *Paeth prediction* kernel which is extracted from an implementation of the *Portable Network Graphics (PNG)* image compression standard. This code fragment implements an important stage of the PNG coding process. It computes the *Paeth prediction* for each pixel d of the current row, starting from the second pixel. The Paeth prediction scheme selects from the 3 neighboring pixels a, b, and c, that surround d as depicted in Figure 3, the pixel that differs the least from the value $p = a + b - c$ (which is called the initial prediction). The selected pixel is called the *Paeth prediction* for d. If the pixel rows contained $length + 1$ elements, $length$ prediction values are produced. The CSI instruction set, as it was introduced in [8], contained only standard arithmetic instructions, such as add and multiply. Althoug these instructions are sufficient for a CSI implementation of the

```
void Paeth_predict_row(char *prev_row, char *curr_row,
char *predict_row, int length)
{ char *bptr, *dptr, *predptr;
char a, b, c, d;
short p, pa, pb, pc;

bptr = prev_row+1;
dptr = curr_row+1;
predptr= predict_row+1;

for(i=1; i < length; i++)
{ c = *(bptr-1); b = *bptr;
a = *(dptr-1); p = a + b - c; pa = abs(p - a); pb = abs(p - b);
pc = abs(p - c); if ((pa ≤ pb)&&(pa≤pc) *predptr = a;
else if (pb≤pc) *predptr = b;
else *predptr = c;

bptr++; dptr++; predptr++;
}
}
```

Fig. 2. The Paeth prediction routine according to the PNG specification [1].

-	-	-	-
-	c	b	-
-	a	d	·
·	·	·	·

Fig. 3. Definition of a, b, c, and d according to PNG specification

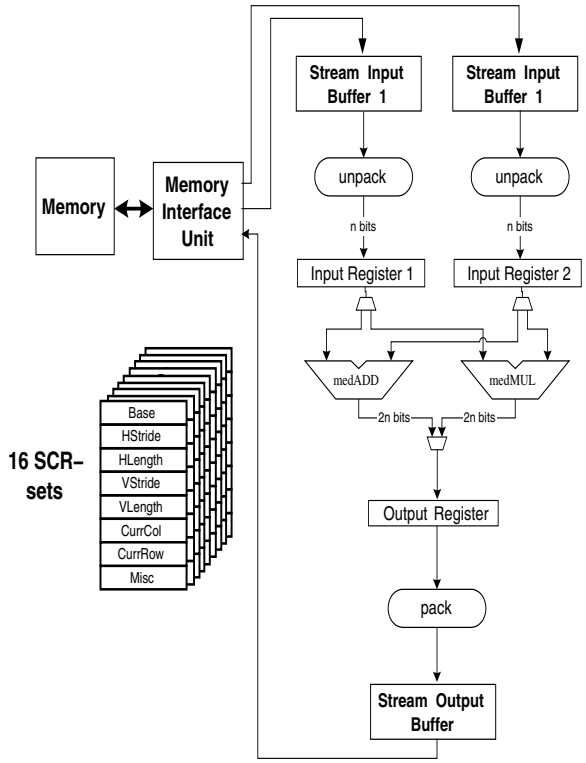

Fig. 4. Datapath of the Stream Unit

presented kernel, in order to achieve additional speedups, we extend the CSI instruction set with the special-purpose instruction `csi_paeth`. For each triple a, b, and c of corresponding elements of the source operand streams, which are described by the SCR-sets `SCRSi`, `SCRSj`, and `SCRSk`,the instruction computes the Paeth prediction value and stores it into the destination stream, which is described by `SCRSl`. Because elementary operations required for computing of the Paeth prediction are collapsed in a single complex operation specified by the proposed instruction, the whole loop of the kernel can be translated into a single `csi_paeth` instruction.

III. Example Implementation

CSI instructions (even those performing simple arithmetic operations) are inherently complex because a single instruction performs the main operation as well as all the miscellaneous tasks, such as address-generation, memory access, etc. The proposed `csi_paeth` instruction is even more complex because of the complexity of the arithmetic operation it performs. Therefore, it is important to verify if all the CSI instructions are implementable. This issue was addressed in [5], [7]. The general organization of the CSI execution unit was described in [5]. It is depicted in Figure 4. We recall that a typical CSI instruction, such as `csi_add`, has two input and one output arithmetic streams, and performs the following operations. The instruction loads source stream data from memory, unpacks (if necessary) the stream elements from storage to computational format, performs a certain operation on corresponding elements, packs (again if necessary) the results, and stores the resulting data stream back to memory. Since these operations are independent, they can be executed in a pipelined fashion, Therefore, the CSI execution unit is organized as a pipeline in which stream data flows through a sequence of stages which perform these operations. Data flows through the datapath in the way, which is shown in the picture: the source stream data is loaded by the memory-interface unit and placed into the input stream buffers. From there it flows through the unpack units to the SIMD functional units *medADD* and *medMUL* that perform the main operation specified by an instruction. The *medADD* unit performs performs the Paeth prediction and usual addition, subtraction and bitwise logical operations. The latency of the Paeth prediction is assumed to be two cycles, while the latency of the addition, sub-

traction and logical operations is assumed to be one cycle. A cycle is comparable to a general-purpose ALU cycle. The *medMUL* execution unit performs the packed multiply operation. It could also incorporate multiply related operations, complex media hardwired functions (see, for example, [7]), sum of absolute differences [14]. The latency of the *medMUL* is at least two cycles and the unit is assumed to be fully pipelined. From the SIMD functional units, the results are then passed through the pack unit and placed into the output stream buffer, from where they are taken by the MIU and stored back to memory.

As described above, the SIMD execution unit *medADD* is assumed to be capable of performing the Paeth prediction with the latency of 2 cycles. In [7] it was shown, that such a unit is implementable and the cost of the implementation was estimated.

IV. Experimental Results

In order to evaluate the performance provided by the proposed new instruction, we simulated superscalar processors extended with only standard CSI instructions and the same processors extended with standard CSI instructions and with `csi_paeth`. For illustration purposes we also simulated the processors without any media extension. We studied the `pngtest` image coding benchmark, which is provided together with `libpng` library, a public domain implementation of the PNG image compression standard [9].

A. Simulation Methodology and Tools

The simulator we used is the `sim-outorder` simulator of the SimpleScalar toolset (release 3.0) [2]. This cycle-accurate simulator simulates an out-of-order multiple-issue processor. A corrected version of the SimpleScalar memory model based on SDRAM specifications given in [6] was used. To simulate the benchmark on the standard superscalar processors, we created the executable `pngtest_scalar`, which was obtained by compiling the sources using the `gcc` compiler. For simulation of the superscalar CPUs extended just with standard CSI instructions and the same CPUs additionally extended with the complex `csi_paeth` instruction, two other executables, `pngtest_csi_simple` and `pngtest_csi_complex` were created. These programs were obtained by manually rewriting the Paeth prediction kernel in assembly and then compiling it together with all the other sources. In the first case, the kernel was rewritten using only the standard CSI instructions. Since the main loop of the `paeth_predict` kernel contains data-dependent control, the masked versions of the arithmetic CSI instructions were used [4]. In the second case, the new `csi_paeth` instruction was employed to implement the loop. The test image `pngtest.png`, which has 132 pixels per row, was used as input for all three versions of the `pngtest` program.

TABLE I
Processor configuration.

Clock rate	1660 MHz	
Issue width	4	
Register update unit size	64	
Load-store queue size	32	
Branch Prediction		
Bimodal predictor size	2K	
Branch target buffer size	2K	
Return-address stack size	8	
Functional unit types, number and cycles (latency, recovery)		
Integer ALU	4	(1/1)
Integer MULT	4	
multiply		(3/1)
divide		(20/19)
Cache ports	2	(1/1)
CSI unit	1	
datapath width	16/32 bytes	
Pack/Unpack unit	3	(1/1)
medADD unit	1	
ALU operations		(1/1)
Paeth prediction		(2/1)
medMUL unit	1	
multiply		(3/1)
divide		(20/19)

B. Modeled Processors

The base system is a 1660 MHz 4-way superscalar processor with out-of-order issue and execution based on the Register Update Unit (RUU). The processor parameters are listed in Table I. The `pngtest` benchmark exhibits a very high instruction cache hit rates. Therefore, and in order to reduce simulation time, a perfect instruction cache is assumed. The processor is configured with the 32 KB 4-way associative L1 data cache which has the cache line size of 64 bytes and with the 1 MB 2-way associative L2 data cache having 128 byte lines. The access times for the caches are 1 and 6 CPU cycles, respectively. The main memory is the standard 166 MHz SDRAM memory which has the row access, the row activate and the precharge times of 2 (memory) cycles. The memory bus is 64 bytes wide and is clocked at 166 MHz as well. Two types of the CSI-enhanced processors have been simulated: CPUs with the CSI unit which can execute only standard CSI instructions and CPUs with the CSI unit which, in addition to these instructions, can execute the new complex

Processor Type	Average instruction count
standard superscalar	4838.63
superscalar+simple CSI	239.92
superscalar+complex CSI	44.00

TABLE II
AVERAGE NUMBER OF EXECUTED INSTRUCTIONS FOR THE PAETH PREDICTION KERNEL

Processor Type	CSI datapath width		
	non-CSI	16 bytes	32 bytes
std	1924.55		
std +simple CSI		555.96	410.04
std +complex CSI		43.54	36.83

TABLE III
AVERAGE EXECUTION TIME (IN CYCLES) FOR THE PAETH PREDICTION KERNEL

`csi_paeth` instruction. For CSI instructions, the latencies of their main arithmetic operations are the same as the latencies of the corresponding scalar instructions. The latency of the main operation for `csi_paeth` is assumed to be 2 cycles. Latencies of the miscellaneous operations performed by the CSI unit are presented in Table I. For each of two types of the CSI units, two configurations were studied: one capable of processing 16 and another capable of processing 32 bytes in parallel.

The CSI unit is interfaced to L1 cache and a whole cache block is brought to the unit on a single access, as described in [5]. Similar to the standard superscalar processor, the CSI-enhanced processor uses two cache ports which are time-multiplexed between load and store accesses for source, mask and destination streams.

C. Results

The goal of the experiments we have carried out for this study is to evaluate the performance provided by the special-purpose `csi_paeth` instruction and to compare it with the performance provided by the standard arithmetic CSI instructions. Such comparison is necessary in order to decide whether augmenting the CSI execution unit with the specialized hardware required for implementation of `csi_paeth` can be justified.

One of the goals for which the `csi_paeth` instruction was introduced, is to reduce the instruction counts. Therefore, prior to presenting the performance figures we study the number of instructions executed by all three types of processors: plain superscalar CPUs, the same CPU extended with standard CSI instructions, and the CPUs extended with standard and the special-purpose CSI instructions. Table II presents the average dynamic instruction counts exhibited by the mentioned processors (which are referred as *standard superscalar*, *superscalar+simple CSI* and *superscalar+complex CSI*) on the Paeth prediction kernel of the `pngtest` program. This table shows that employing simple CSI instructions provides a superscalar CPU with the 20.2x reduction of the number of executed instructions. If, instead of the simple CSI instructions, the special-purpose CSI instruction `csi_paeth` is employed, an additional 5.43x reduction can be achieved. We remark here that although the whole loop of the paeth prediction kernel is reduced to a single instruction `csi_paeth`, the total instruction count of the kernel is 44 (instead of just 1). This happens due to the auxiliary CSI instructions which are needed to initialize the individual control registers of the SCR-sets with parametersof the operand streams.

We now present performance figures for the studied processors. Table III depicts the average number of cycles required for the execution of the paeth prediction kernel. The three types of the processors are referred as *std* (standard superscalar), *std+simple CSI* (superscalar extended with simple arithmetic CSI instructions), and *std+complex CSI* (superscalar extended with simple CSI instructions and with the complex `csi_paeth` instruction). The results show that the CSI execution unit, which supports only simple CSI instruction and is capable of processing 16 bytes in parallel, improves the kernel performance by a factor of 3.46x. Enhancing such execution unit with the specialized hardware for the `csi_paeth` instruction provides an additional speedup of 12.9x. The speedup of such CSI-enhanced superscalar CPU over the baseline CPU is equal to 44.2x. These results show that the additional costs associated with the specialized hardware required for `csi_paeth` are justified by a dramatic performance improvement. Table III also shows that the performance of the both types of CSI-enhanced CPUs improves when the amount of the parallel execution hardware is increased. This result is important because of the current trends in microprocessor technology, which allow a designer to put dozens or even hundreds of functional units on a single chip. The challenge is to feed these units with instructions and data in order to keep them utilized. CSI offers a possible solution to this challenge.

V. CONCLUSIONS

In this paper we extended the Complex Streamed Instruction set (CSI) with a special-purpose instruction `csi_paeth`, which computes the *Paeth prediction* for

three streams of input pixels. Two dofferent CSI implementations of the Paeth prediction kernel of the PNG image codec are studied: first one uses only standard CSI instructions, while the second employs the new `csi_paeth` instruction. The comparison of these two implementations shows the following. The second implementation executes 5.43 times less instructions than the first one. Furthermore, when executed on a 4-way issue CSI-enhanced superscalar processor with CSI unit capable of processing 16 bytes in parallel, the second implementation outperforms the first one by the factor of 12.9x. These results show that the costs of the specialized hardware needed to implement such a complex instruction, are justified by impressive performance improvements.

REFERENCES

[1] T. Boutell and T. Lane. Portable Network Graphics Specification, version 1.0. Document available via ftp://ftp.uu.net/graphics/png/documents/png-1.0-w3c.ps.gz.

[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.

[3] D. Cheresiz, B. Juurlink, S. Vassiliadis, and H. Wijshoff. Performance Scalability of the Multimedia Instruction Set Extensions. In *Euro-Par'02*, 2001.

[4] D. Cheresiz, B. Juurlink, S. Vassiliadis, and H. Wijshoff. Architectural Support for 3D Graphics in the Complex Streamed Instruction Set. In *Int. Conf. on Parallel and Distributed Computing and Systems (PDCS 14)*, 2002.

[5] D. Cheresiz, B. Juurlink, S. Vassiliadis, and H. Wijshoff. Implementation of a Streaming Execution Unit. In *EUROMICRO 28*, 2002.

[6] M. Gries. The Impact of Recent DRAM Architectures on Embedded Systems Performance. In *EUROMICRO 26*, 2000.

[7] Edwin Hakkennes and Stamatis Vassiliadis. Multimedia Execution Hardware Accelerator. *Journal of VLSI Signal Processing*, 28, July 2001.

[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] Libpng: the official PNG reference library. Document available via http://www.libpng.org/pub/png/libpng.html.

[10] Alex Peleg, Sam Wilkie, and Uri Weiser. Intel MMX for Multimedia PCs. *Communications of the ACM*, 40(1):24–38, 1997.

[11] 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.

[12] Shreekant Thakkar and Tom Huff. The Internet Streaming SIMD Extensions. *Intel Technology Journal*, May 1999.

[13] S. Vassiliadis, B. Blaner, and R. Eickemeyer. SCISM: A Scalable Compound Instruction Set Machine Architecture. *IBM Journal of Research and Development*, 38(1), January 1994.

[14] S. Vassiliadis, E.A. Hakkennes, J.S.S.M. Wong, and G.G. Pechanek. The Sum-Absolute-Difference Motion Estimation Accelerator. In *EUROMICRO 24*, pages 559–566, 1998.