COMPARING FPGAS TO GRAPHICS ACCELERATORS AND THE PLAYSTATION 2 USING A UNIFIED SOURCE DESCRIPTION

Lee W. Howes, Paul Price, Oskar Mencer, Olav Beckmann, Oliver Pell

Department of Computing, Imperial College London
email: {lwh01, paulp, oskar, ob3, op}@doc.ic.ac.uk

ABSTRACT

Field programmable gate arrays (FPGAs), graphics processing units (GPUs) and Sony's PlayStation 2 vector units offer scope for hardware acceleration of applications. We compare the performance of these architectures using a unified description based on *A Stream Compiler* (ASC) for FPGAs, which has been extended to target GPUs and PS2 vector units. Programming these architectures from a single description enables us to reason about optimizations for the different architectures. Using the ASC description we implement a Montecarlo simulation, a Fast Fourier Transform (FFT) and a weighted sum algorithm. Our results show that without much optimization the GPU is suited to the Montecarlo simulation, while the weighted sum is better suited to PS2 vector units. FPGA implementations benefit particularly from architecture specific optimizations which ASC allows us to easily implement by adding simple annotations to the shared code.

1. MOTIVATION

We consider accelerating software with coprocessors and classify them into custom and general purpose coprocessors. Custom coprocessors execute a single task, such as MPEG decoding or encryption. General purpose coprocessors such as graphics processing units (GPUs) and PlayStation 2 vector units (PS2) can be used for a variety of tasks. In addition we see FPGAs which are able to implement a custom coprocessor dynamically. These are widely different technologies and it is unclear which is best suited to a given task.

Deciding which acceleration technology is most appropriate poses a challenge. Programming methodologies range from circuit design for FPGAs through high level language support for GPUs to assembly programming for the vector units. We present a system that generates implementations for FPGAs, GPUs and PlayStation vector units from a single parallel description. Unlike behavioral synthesis approaches that rely on complex analyses to infer parallelism, we serialize a stream description onto each architecture using ASC, A Stream Compiler [1] for FPGAs.

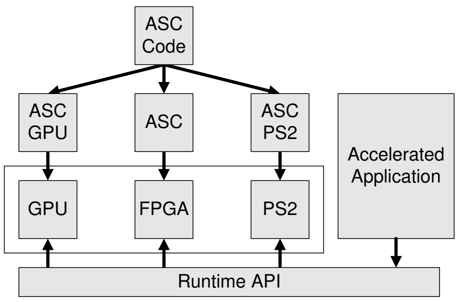


Fig. 1. A Stream Compiler (ASC) code as a unified description which compiles to GPUs, to the Sony PlayStation 2's Vector Units and to FPGAs.

We offer the following contributions:

- A unified description using ASC code to describe algorithms for graphics processing units (GPUs), the PlayStation 2 (PS2) and FPGAs, allowing for testing and performance comparisons on different platforms.
- A performance comparison of three application benchmarks on GPUs, the PS2 and FPGAs: a Montecarlo simulation, an FFT and a weighted sum algorithm.

Figure 1 shows the general concept of our system.

1.1. Background

There is a wide spectrum of architectures available for coprocessors and hardware accelerators. Small scale coprocessors range from the floating point units (FPUs) that we see as extensions to general purpose processors to Intel's Streaming SIMD Extensions (SSE) [2] and semi-independent vector coprocessors. IBM, Sony and Toshiba's new Cell [3] processor is a further extension of the principle of vector coprocessors.

Various architectures exist for tasks where large scale computations can be moved to coprocessors. Domain-specific hardware, optimized for a specific class of problems, is common. Network processing hardware often uses a domain-specific processing unit known as a network processor [4]. Network processors are highly programmable but structurally

Table 1. Features of FPGAs, Sony PlayStation 2 and GPUs. Communication refers to data transfers between the main CPU and the accelerator. Good, bad and medium are defined in terms of assistance towards obtaining high performance.

Feature	FPGA PCI card	PlayStation 2	GPU	Pentium	Pentium with SSE
Communication latency	bad	good	medium	good	good
Performance latency	bad	good	medium	good	good
Performance throughput	**good**	**bad**	**good**	**bad**	**medium**
Flexibility	good	medium	medium	good	bad
Ease of programming	bad	medium	medium	good	medium

optimized for processing network traffic flows. Philips produces the TriMedia processor [5], a VLIW processor optimized for processing media data often found in set-top boxes and similar units. The Imagine Stream Processor [6] from Stanford University is a general purpose stream-based architecture benchmarked on media processing, polygon rendering and similar applications.

Graphics processing units (GPUs) are widespread and used to accelerate the graphics processing that modern personal computer users, and particularly game players, demand. GPU technology now offers large, complicated processors with higher transistor counts than even many of the latest general purpose processors. Modern GPUs are highly programmable and can be used for accelerating computation that is no-longer limited to graphics [7]. Additional hardware for general purpose acceleration, based on GPU technology, is in production; for example, ClearSpeed's CSX architecture [8].

Reconfigurable technology offers another approach to hardware acceleration: rather than a fixed architecture programmed by instructions, we can reconfigure the architecture. Configuration can be performed at the gate level, as in FPGAs, or at higher levels, for example in Morphosys' reconfigurable SIMD system-on-chip product [9].

Cope et al. [10] investigate comparisons between GPUs and FPGAs in the area of video processing. Their comparisons do not, however, make use of a unified high level representation.

2. TARGET ARCHITECTURES

The three architectures targeted by this work are FPGAs, Graphics Processing Units and the vector units in the Sony PlayStation 2's Emotion Engine processor. Some features of the architectures, and of the Pentium Processor, are described in Table 1.

FPGAs: In the context of this work we are looking at SRAM based FPGAs as the compilation target of ASC. The FPGA itself is treated as a flexible data-processing device. Application specific integrated circuits can be used as accelerator coprocessors and are often faster than FPGA implementations, however FPGA flexibility allows them to accelerate multiple applications making them comparable with GPUs and vector accelerators.

The Graphics Processing Unit (GPU): The main component in a computer system dedicated to the acceleration of computer graphics, modern GPUs offer programmable execution for processing vertices and pixels and we can map these programmable stages onto general purpose computation. Figure 2 demonstrates how data flows through the GPU and how the GPU fits in the computer system. The Pixel, or fragment, processor presents the programmer with input and output pixel arrays. These pixel arrays can be treated as rectangular single precision floating point data buffers for general computation [7]. Programming the GPU is generally performed in a graphics oriented language. NVIDIA's Cg [13] and the GL Shader Language (GLSL) present C-like languages but require understanding of OpenGL or DirectX graphics programming. Brook for GPUs [11] treats the GPU as a stream architecture and abstracts the complexities of graphics programming, but is currently limited in scope to GPUs only. None of these languages allows development in a form that can easily be applied to other architectures.

The Sony PlayStation2: A games console that has to date achieved sales of over 90 million units, at the core of the PS2 is the collection of processors known collectively as the *Emotion Engine*, see Figure 3. The emotion engine comprises a general purpose MIPS processor with floating point unit and SIMD extensions, two vector coprocessors, a graphics processor and associated peripheral components. The Vector units, VU0 and VU1, are both highly programmable with flow and branching control. The vector units are limited to accessing their own local memory and as a result can only work on data passed from main memory. Data transfers into the memory of the vector units are performed via DMA and over a fast (2.4Gb/s) bus.

3. ASC - A STREAM COMPILER

A Stream Compiler (ASC) is a compiler that generates stream architectures for FPGAs. ASC uses an object oriented approach [15] to development and allows optimization of the design at the algorithm, architecture and arithmetic levels. As its name suggests, ASC is optimized for describing stream architectures. An ASC program represents a data-flow system which can be seen as a stream. ASC code uses a C++ class library and an example can be seen in Figure 5. ASC is not a behavioral synthesis tool like the work by Venkatara-

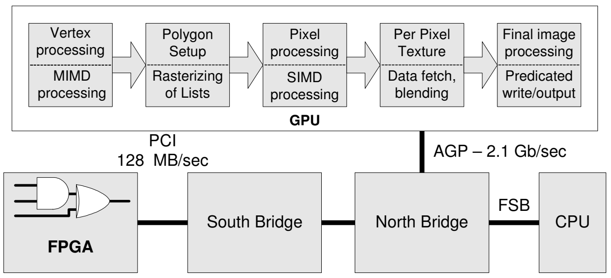


Fig. 2. Location of the GPU and FPGA in the PC system identifying the important buses.

mani et al. [16]. Noting the difficulties of behavioral synthesis, ASC allows direct implementation of a hardware design, using C++ to reduce the programmer and tool chain overhead for development. Unlike other circuit design tools such as Handel-C and Streams-C [17], ASC combines the algorithm, architecture and arithmetic levels into a single tool. Development access to multiple levels of the design hierarchy gives the ASC programmer great flexibility in the implementation method when required. Performance, bitwidth and area requirements of each part of the stream can be separately optimized [18] with little necessary programming burden.

4. COMPILING TO GPUS AND PLAYSTATIONS

Compiling ASC code to GPUs and PlayStations requires backends to target the different technologies. In its basic form ASC is tied to FPGA circuit generation and as a result the most efficient approach is to reimplement high level objects representing the ASC API itself when using the GPU or PS2 backends. ASC code represents a data-flow model of a stream program. Each assignment statement creates a connection in the dataflow graph rather than a static data assignment. Internally the frontend creates a dataflow model of the program in an object-oriented, easily traversable form.

The choice of architecture affects the manner in which the data-flow graph is processed. Architectural differences are transparent to the programmer, requiring only the selection of the appropriate target as indicated in Figure 1. ASC generates a single executable combining the overall program with the necessary runtime system to program the accelerator.

4.1. Translating ASC code for the GPU

ASC programs represent the flow of data in an abstract stream processor. Execution of an ASC program generates a dataflow graph. The ASC dataflow graph is processed internally to generate a set of abstract syntax trees (ASTs) representing fragment programs necessary to represent the algorithm. Each fragment program is output as code to pass to the Brook [11] compiler, which performs simple operations to enable use

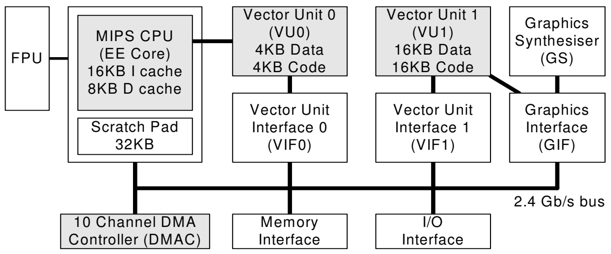

FPU
MIPS CPU (EE Core) 16KB I cache 8KB D cache
Scratch Pad 32KB
Vector Unit 0 (VU0) 4KB Data 4KB Code
Vector Unit 1 (VU1) 16KB Data 16KB Code
Graphics Synthesiser (GS)
Vector Unit Interface 0 (VIF0)
Vector Unit Interface 1 (VIF1)
Graphics Interface (GIF)
2.4 Gb/s bus
10 Channel DMA Controller (DMAC)
Memory Interface
I/O Interface

Fig. 3. The *Emotion Engine* at the heart of the PlayStation 2 architecture [14] showing the MIPS processor and dual vector units.

of the Brook runtimes. Brook's runtimes abstract away the OpenGL or DirectX calls to hardware.

Most intermediate nodes of the dataflow graph, representing temporary stream variables, appear as registers in a fragment program. In each executing instance of a fragment program, generating a single output stream element, these registers will take intermediate values of the ASC temporary variables. Each fragment program execution works from a given set of input buffers to an output buffer. A buffer of *n* elements represents a variable over *n* stream iterations.

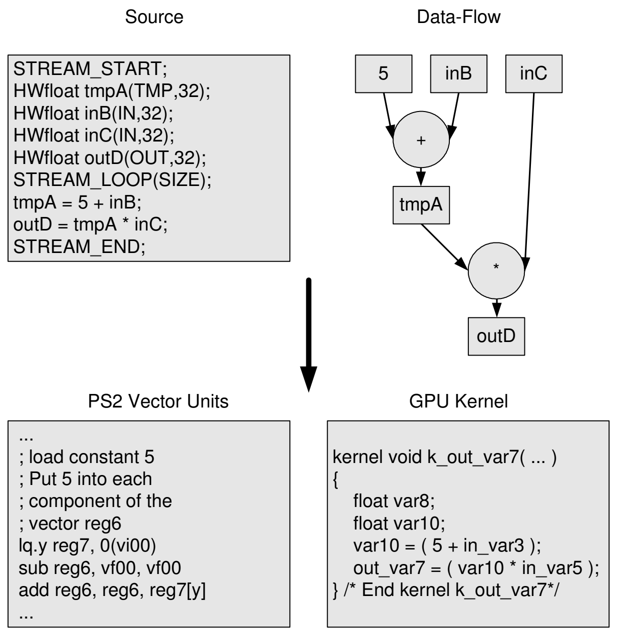
The implementation of registers, memories and data-flow cycles common in finite state machine descriptions is limited on the GPU by the lack of inter-fragment communication. The GPU must be treated as a pure stream architecture, where each output can be calculated based on input values alone. Due to the stream limitation, registers and memories are only converted from the ASC implementation if they are read only. Given the read-only restriction on registers and memories, feedback cycles are removed entirely. Delays or FIFOs, common in digital circuits and also in stream applications must be recreated either through additional code at different temporal offsets, or through the use of intermediate buffers storing the same data item at various time points.

Final compilation makes use of NVIDIA's Cg compiler to produce high quality assembly code for a wide range of GPU architectures. Each transfer from one intermediate buffer to another is performed by a separately compiled application kernel.

4.2. Translating ASC code for the Sony PlayStation 2

Unlike the GPU, which processes a large block of stream data in each kernel but where each kernel performs only a small amount of processing, the PS2 vector units implement an entire computation in a single program.

The original data-flow graph generated by the ASC frontend is used to create an Abstract Syntax Tree (AST) resulting in assembly code as in Figure 4. The AST represents the combination of operations required to perform the stream computation. Both single element and vector instructions are possible in the AST such that an instruction can be generated to process four data points simultaneously whilst still



```
STREAM_START;
HWfloat tmpA(TMP,32);
HWfloat inB(IN,32);
HWfloat inC(IN,32);
HWfloat outD(OUT,32);
STREAM_LOOP(SIZE);
tmpA = 5 + inB;
outD = tmpA * inC;
STREAM_END;
```

```
...
; load constant 5
; Put 5 into each
; component of the
; vector reg6
lq.y reg7, 0(vi00)
sub reg6, vf00, vf00
add reg6, reg6, reg7[y]
...
```

```
kernel void k_out_var7( ... )
{
   float var8;
   float var10;
   var10 = ( 5 + in_var3 );
   out_var7 = ( var10 * in_var5 );
} /* End kernel k_out_var7*/
```

Fig. 4. ASC code, the data-flow graph generated from it and resulting partial code for both the PlayStation 2 vector units and the GPU.

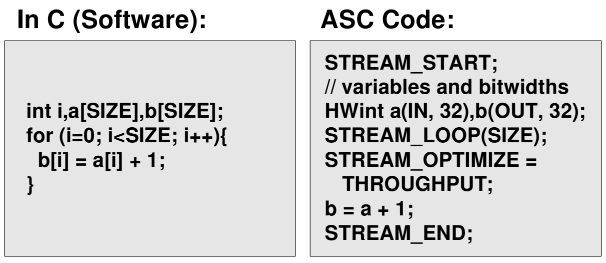
allowing the generation of control-flow instructions.

The input stream is divided into small parallel work units which are processed in the vector units as necessary.

5. APPLICATIONS

To demonstrate our approach and to investigate performance comparisons we look at three applications to accelerate: a Montecarlo simulation, a Fast Fourier Transform and a simple weighted sum calculation. Since current GPUs are limited to single precision floating point we restrict ourselves to single precision to make the comparison fair, although the FPGA supports adaptable precision. In each case performance results of the raw implementations are be compared against each other, and also against an implementation of the same algorithm running on a fast Pentium 4 processor.

Montecarlo simulation: Montecarlo methods are algorithms employing random (or pseudo-random) numbers to solve computational problems. One example of the use is in area sampling: rather than sampling every point, random points spread evenly through the region are used. As a result information is provided that can be generalized across the region. In this case we implement a slight simplification of a Montecarlo simulation originally intended for simulating the value of European call options based on given parameters. The original asset price is specified and a set of randomly generated sequences of subsequent asset prices is generated over a given time-frame. The Montecarlo simulation contains a static loop which maps well onto the GPU


In C (Software):
ASC Code:

```
int i,a[SIZE],b[SIZE];
for (i=0; i<SIZE; i++){
  b[i] = a[i] + 1;
}
```

```
STREAM_START;
// variables and bitwidths
HWint a(IN, 32),b(OUT, 32);
STREAM_LOOP(SIZE);
STREAM_OPTIMIZE =
   THROUGHPUT;
b = a + 1;
STREAM_END;
```

Fig. 5. C code with a simple loop compared with ASC code representing a hardware stream version of the same loop. Optimization mode setting is for maximum throughput.

architecture.

FFT: Fourier transforms have many uses, particularly in audio and visual applications. In this case we look at an implementation of a radix-2 butterfly. This was originally an ASC FPGA example and has been implemented for both the GPU and PlayStation 2 largely to show that it is possible to do so with little or no work.

Weighted sum: The weighted sum algorithm multiplies the last four values in the stream by constants, totals those values, and then totals the last four of those sums. The weighted sum is a simple calculation of a form seen in filtering algorithms. The algorithm makes use of the *prev* function that inserts a delay in hardware fairly heavily and uses intermediate buffers on the GPU as a result.

In the case of the Montecarlo and FFT computations we can use ASC abstractions to easily optimize the FPGA implementation utilizing BlockRAMs to keep data on the chip. The Montecarlo static loop prohibits the use of registers between logic elements, computing the inner loop between buffers repeatedly offers full scope for pipelining. The FFT requires multiple passes, in the naive implementation data reordering between passes is performed in software. In the buffered version, intermediate data is stored and reordered on chip.

6. RESULTS

We show results of the Montecarlo, FFT and weighted sum algorithms on our three target architectures. In addition we show execution times for the same algorithms written in C and running on a Pentium 4. GPU timings are performed on a Athlon 64 2000+ machine, FPGA tests on a 2 GHz Pentium 4 and Pentium results on a 3.2 GHz Pentium 4.

We use a Xilinx Virtex-II 6000 FPGA running on an ADM-XRCII PCI card and an NVIDIA 6800Ultra GPU in an AGP slot. We compile the ASC code for the FPGA and GPU using GCC 3.3, on the PS2 using GCC 2.95 (the maximum available for the architecture) and the Pentium code using Intel C++ version 9.0. The Intel compiler is used on the Pentium due to its support for vectorization and SSE-2 extensions. Timing results are obtained using real transfer-to-

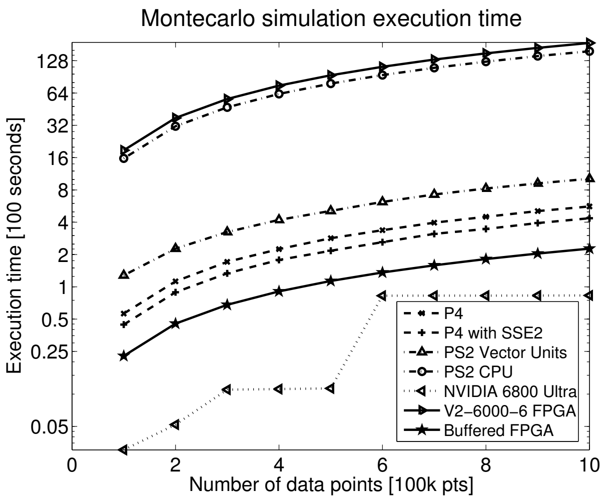


Fig. 6. Montecarlo simulation using a Pentium 4 3.2 GHz CPU using Intel's C compiler with -O3 optimization and full optimization including vectorization. We compare with the same execution running on an NVIDIA 6800 Ultra, the PS2's vector units, the PS2 CPU and a Xilinx Virtex2-6000-6 FPGA with and without on-chip buffering.

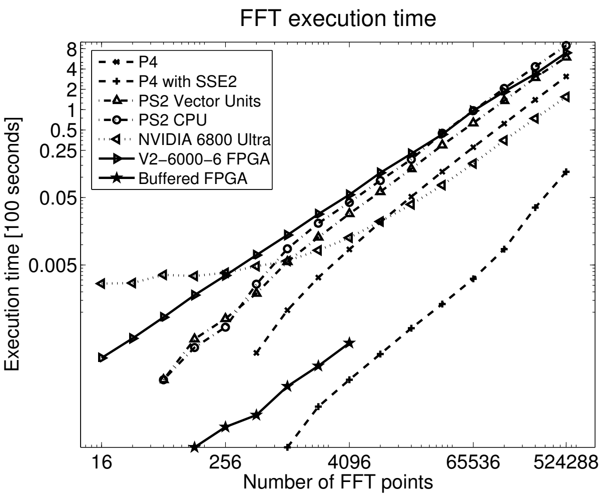


Fig. 7. Radix-2 FFT using a Pentium 4 3.2 GHz CPU using Intel's C compiler with -O3 optimization and full optimization including vectorization. We compare with the same execution running on an NVIDIA 6800 Ultra, the PS2's vector units, the PS2 CPU and a Xilinx Virtex2-6000-6 FPGA with and without on-chip buffering.

hardware times measured by calls to the *gettimeofday* function, except for the optimized FFT and Montecarlo which are based on cycle accurate estimates. FPGA computations are optimized for throughput in all cases except the naive Montecarlo.

Figure 6 and Figure 8 show the performance comparison between the architectures for the Montecarlo and weighted sum algorithms working on dataset sizes ranging from 100,000 data points to one million data points. Figure 7 uses a range of powers of two from 16 to 524288 for the FFT, however the optimized implementation cannot currently support more than 8192 points due to memory limitations. This small FFT could be used to calculate results for larger transforms.

The Montecarlo simulation shows how well the GPU can perform when the algorithm is well matched to its architecture. The GPU execution is 3 times faster than the nearest competitor. Each executing fragment program of the GPU maps onto a single Montecarlo simulation leading to as much parallelism as the GPU can offer. The FPGA Montecarlo simulation shows how the static LOOP construct makes the FPGA implementation inefficient and some idea of how this can be rectified is shown by the buffered Montecarlo (which is implemented in 24 bit precision due to area limitations which can be corrected with a larger FPGA). The circuit is unpipelined and clocks at 0.4 MHz against the pipelined version clocking at 34 MHz. The optimized FFT circuit only runs at around 50 MHz indicating that there is still room for improvement in ASC's circuit generation.

Results from the FFT show a wide performance range. The inefficient memory rearranging and high transfer time percentage on the GPU (shown in Figure 9) lead to low efficiency on block transfer architectures compared with the Pentium 4. The Pentium 4 performs the simulation 13 times faster than the GPU and 57 times faster than the unoptimized FPGA. FPGA performance is improved vastly by on-chip buffering. Current progress in this area limits us to 8192 points but this will improve with time.

The weighted sum results show high performance for the SSE-2 optimized Pentium 4. The algorithm is largely register bound and hence highly efficient in a general purpose processor. The block transfers harm the performance of the FPGA and GPU greatly. PS2 vector units are closer to the CPU and are therefore less affected by data transfer times, and hence we achieve higher performance. We see that the Pentium 4 with SSE only performs 28% faster than the non-SSE Pentium, but 58 times faster than the FPGA.

7. CONCLUSIONS

Our work enables performance comparisons between FPGAs, graphics accelerators and PlayStation 2 vector units as coprocessors using a single unified representation. We discuss how a single description can be beneficial, and demonstrate one approach to achieving this goal using a stream based compiler. The presence of a unified description offers a direct route to comparison.

We notice that the performance of a single description compiled to different architectures does not yield optimal results for each of the target technologies. Nevertheless, we obtain a fair starting point which we then refine to optimize

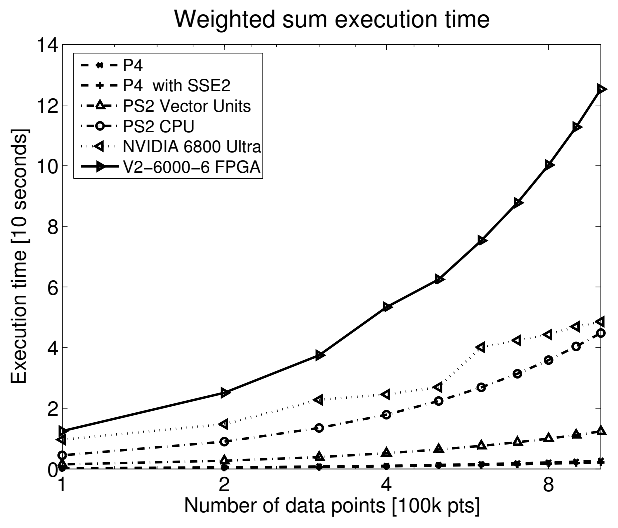


Fig. 8. Weighted sum calculation using a Pentium 4 3.2 GHz CPU using Intel's C compiler with -O3 optimization and full optimization including vectorization. We compare with the same execution running on an NVIDIA 6800 Ultra, the PS2's vector units, the PS2 CPU and a Xilinx Virtex2-6000-6 FPGA.

for each individual technology. In a sense we combine technology evaluation and optimization in the same framework with the result of increasing productivity. Optimizing a partial - and tested - implementation is often easier than starting from scratch, and there are often beneficial low level optimizations that one can perform. Some optimizations, such as the buffers demonstrated in this paper, are easily implemented in our current system. The performance of FPGA technology is particularly dependent on the amount of optimization in a design. Support for much more low level optimizations is clearly the ultimate challenge in this project.

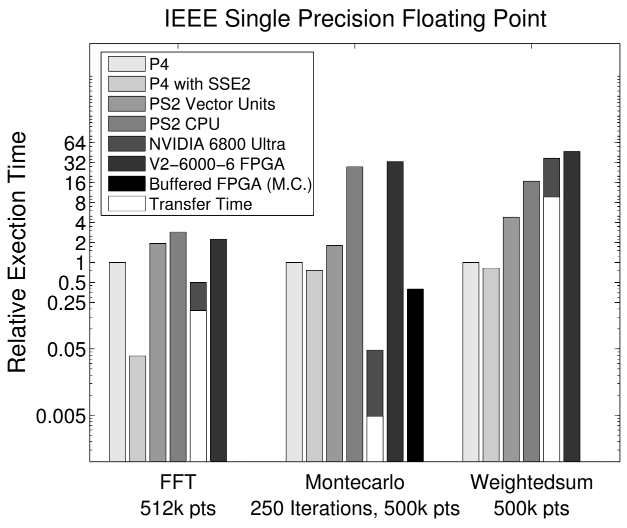


Fig. 9. Direct comparison of the three examples including, where relevant, a breakdown of execution time and transfer overhead. The results are normalized for each application relative to the respective Pentium 4 non-SSE execution time. Buffered FPGA results are currently only available for the Montecarlo at these dataset sizes.

8. REFERENCES

[1] O. Mencer. *ASC, a stream compiler for computing with FPGAs.* IEEE Transactions on CAD, 2006.

[2] S. K. Raman et al. *Implementing streaming SIMD extensions on the Pentium III processor.* IEEE Micro.

[3] H. P. Hofstee. *Power efficient processor architecture and the cell processor.* Proc. HPCA, IEEE 2005.

[4] N. Shah. *Understanding network processors.* Master's thesis, University of California, Berkeley, September 2001.

[5] J. T. J. Van Eijndhoven and E. J. D. Pol. *Trimedia CPU64 architecture.* Proc. ICCD, IEEE 1999.

[6] U. Kapasi et al. *The Imagine stream processor.* Proc. ICCD, IEEE 2002.

[7] S. Venkatasubramanian. *The graphics card as a stream computer.* Proc. MPDS, ACM 2003.

[8] M. Meißner, S. Grimm, W. Straßer, J. Packer, and D. Latimer. *Parallel volume rendering on a single-chip SIMD architecture.* Proc. PVG, IEEE 2001.

[9] H. Singh, M.-H. Lee, G. Lu, N. Bagherzadeh, F. J. Kurdahi, and E. M. C. Filho. *Morphosys: An integrated reconfigurable system for data-parallel and computation-intensive applications.* IEEE Transactions in Computing, 2000.

[10] B. Cope et al. *Have GPUs made FPGAs redundant in the field of video processing?* Proc. FPT, IEEE 2005.

[11] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. *Brook for GPUs: Stream computing on graphics hardware.* Proc. SIGGRAPH, ACM 2004.

[12] J. Kruger and R. Westermann. *Linear algebra operators for GPU implementation of numerical algorithms.* Proc. SIGGRAPH, ACM 2003.

[13] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. *Cg: A system for programming graphics hardware in a c-like language.* Proc. SIGGRAPH, ACM 2003.

[14] *EE Overview, Sony Documentation with Linux kit*, 2001.

[15] O. Mencer. *PAM-Blox II: Design and evaluation of c++ module generation for computing with FPGAs.* Proc. FCCM, IEEE 2002.

[16] G. Venkataramani, M. Budiu, T. Chelcea, and S. C. Goldstein *C to asynchronous dataflow circuits: An end-to-end toolflow.* Proc. IWLS, IEEE/ACM 2004.

[17] M. B. Gokhale, J. M. Stone, J. Arnold, and M. Kalinowski. *Stream-oriented FPGA computing in the Streams-C high level language.* Proc. FCCM, IEEE 2000.

[18] O. Mencer, D. J. Pearce, L. W. Howes, and W. Luk. *Design space exploration with A Stream Compiler.* Proc. FPT, IEEE 2003.