FPGAs, GPUs and the PS2 - A Single Programming Methodology

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

Department of Computing, Imperial College London
email: {*lwh01, oskar*}*@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. Implementing algorithms on multiple architectures can be a long and complicated process. We demonstrate an approach to compiling for FPGAs, GPUs and PS2 vector units using a unified description based on *A Stream Compiler* (ASC) for FPGAs. As an example of its use we implement a Monte-carlo simulation using ASC. The unified description allows us to evaluate optimisations for specific architectures on top of a single base description, saving time and effort.

1. Motivation

We consider accelerating software using coprocessors. Coprocessors can be classified as custom or as general purpose. General purpose coprocessors include devices such as graphics processing units (GPUs) and vector units such as those in Sony's Playstation 2 console. Custom coprocessors execute a single task, such as MPEG decoding or encryption. In addition we see FPGAs which are capable of implementing custom processors dynamically. These technologies have very different properties 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 reduce the development overhead a single programming model that covers all these architectures is beneficial. We present a system that generates implementations for FPGAs, GPUs and Playstaion vector units from a single description as shown in Figure 1. Rather than performing behavioural synthesis we serialize a program written for *A Stream Compiler* (ASC [1]) for FPGAs.

2. ASC - A Stream Compiler

A Stream Compiler (ASC) generates stream architectures for FPGAs using a C++ based object-oriented approach. ASC allows optimization of a design at the algorithm, architecture and arithmetic levels. An ASC program represents

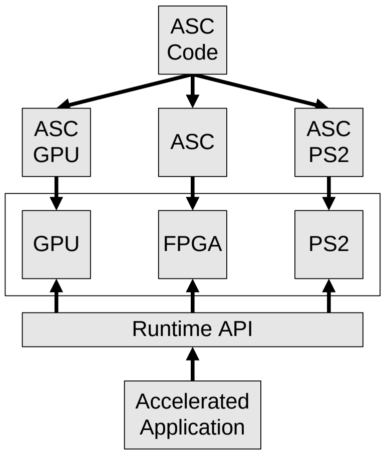


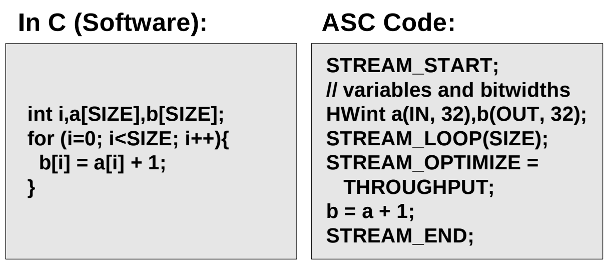
Figure 1: ASC code can be linked with the FPGA, GPU or PS2 backend leading to a result executable on the desired hardware via the runtime API.

a dataflow system which can be seen as a stream. To avoid the difficulties often associated with behavioural synthesis ASC allows direct implementation of a hardware design using C++ to reduce the programmer and toolchain overhead for development. Developer access to multiple levels in the design hierachy offers the ASC programmer great flexibility of implementation when required. Figure 2 gives an example of ASC code in use.

3. Target Architectures

FPGAs: SRAM based FPGAs are the primary compilation target of ASC. FPGAs can support very high rates of data throughput when high parallelism is utilized in circuits implemented in the reconfiguable fabric. Although slower and less power efficient than comparable ASICs, FPGA reconfigurability offers a flexibility that makes FPGAs comparable with GPUs or vector accelerators as flexible coprocessors.

The Graphics Processing Unit: Modern graphics processing units offer programmable computation of vertices and pixels. It is becoming increasingly common to use this flexible computation in a more general purpose sense then



In C (Software):

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

ASC Code:

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

Figure 2: 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.

limiting computation to graphics work [2]. Generally programming is performed in a graphics oriented language such as NVIDIA's Cg or the GL Shader Language. These languages do not generally allow development in a form that can be easily applied to other architectures.

The Sony PlayStation 2 is a games console that has to date achieved sales of over 90 million units. At the core of the PS2 is the *Emotion Engine* consisting of a general purpose MIPS processor with floating point unit and SIMD extensions, two vector coprocessors and a graphics processor. The vector units are highly programmable with flow and branching control, and have their own local memory filled from main memory by DMA transfers.

4. Compiling programs to GPUs and the PS2

Internally the ASC program is represented by a dataflow graph. Each assignment statement creates a new connection in the graph, rather than directly assigning data. The choice of architecture affects the manner in which the graph is processed. Architectural differences are transparent to the programmer in a basic implementation, requiring only the selection of an appropriate target architecture. As necessary optimizations on implementations can be performed making use of architecture specific extensions.

Translating for the GPU involves separating the dataflow graph into individual execution kernels with intermediate data buffers. Each buffer represents a single variable over a wide range of time points (a datastream). Each kernel is converted into an AST representing the language used to program a GPU fragment program usually used to program pixel data. In the current version of the system the language used is Brook for GPUs [3]. The GPU is limited to pure streaming programs, so data-flow cycles and anything that depends on them, such as finite state machines, cannot be implemented in a program targetted at GPUs.

Translating for the PS2 involves taking the entire program and implementing it as a vector unit program. The AST representation of a vector unit program supports both

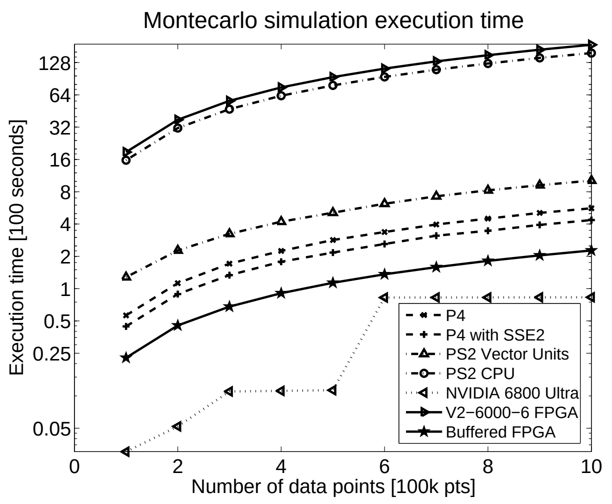


Figure 3: Montecarlo simulation using a Pentium 4 3.2GHz CPU using Intel's C compiler with -O3 optimization and full optimization including vectorization. We compare with the same execution running on an NVIDIA 6800 Ultra, the PS2's vector units, the PS2 CPU and a Xilinx Virtex2-6000-6 FPGA with and without on-chip buffering.

single element and vector instructions, such that multiple datapoints can be operated on simultaneously. Performance gains are possible as a result of the parallel vector operations.

5. Performance comparisons

We can use the single base representation to compare the performance of the architectures on an algorithm. Figure 3 shows the results obtained when comparing an implementation of a montecarlo simulation on the discussed architectures against an implementation on the Pentium 4. With a base implementation we can perform simple optimizations on each target to improve the performance. The "Buffered FPGA" line on the graph represents an optimization of the FPGA implementation using on-chip data buffers, which are easy to implement in ASC. In this case optimization is necessary as the base Montecarlo implementation includes a loop which renders hardware pipelining impossible.

References

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

[2] NVIDIA Corporation R. Fernando. *Trends in GPU evolution.* Eurographics, September 2004.

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