A GPU-Like Soft Processor for High-Throughput Acceleration

Jeffrey Kingyens and J. Gregory Steffan
Department of Electrical and Computer Engineering
University of Toronto
10 King's College Road, Toronto, Canada
{kingyen,steffan}@eecg.toronto.edu

1. INTRODUCTION

As FPGAs become increasingly dense and powerful, with high-speed I/Os, hard multipliers and plentiful memory blocks, they have consequently become more desirable platforms for computing. Recently there is building interest in using FPGAs as accelerators for high-performance computing, leading to commercial products such as the SGI RASC which integrates FPGAs into a blade server platform, and XtremeData and Nallatech that offer FPGA accelerator modules that can be installed alongside a conventional CPU in a standard dual-socket motherboard.

The challenge for such systems is to provide a programming model that is easily accessible for the programmers in the scientific, financial, and other data-driven arenas that will use them. Developing an accelerator design in a hardware description language such as Verilog is difficult, requiring an expert hardware designer to perform all of the implementation, testing, and debugging required for developing real hardware. Behavioral synthesis techniques—that allow a programmer to write code in a high-level language such as `C` that is then automatically translated into custom hardware circuits—have long-term promise [3, 6, 9], but currently have many limitations.

What is needed is a high-level programming model specifically tailored to making the creation of custom FPGA-based accelerators easy. In contrast with the approaches of custom hardware and behavioral synthesis, a more familiar model is to use a standard high-level language and environment to program a processor, or in this case an FPGA-based soft processor. In general, a soft-processor-based system has the advantages of (i) supporting a familiar programming model and environment and (ii) being portable across different FPGA products and families, while (iii) still allowing the flexibility to be customized to the application. Although soft processors themselves can be augmented with accelerators that are in turn created either by hand or via behavioral synthesis, our long-term goal is to develop **a soft processor architecture that is more naturally capable of fully-utilizing the FPGA**.

1.1 A GPU-Inspired System

Another recent trend is the increasing interest in using the Graphics Processing Units (GPUs) in standard PC graphics cards as general-purpose accelerators, including NVIDIA's CUDA and AMD (ATI)'s Close-to-the-Metal (CTM) [2] programming environments. While the respective strengths of GPUs and FPGAs are different—GPUs excel at floating-point computation, while FPGAs are better suited to fixed-point and non-standard-bit-width computations—they are both very well-suited to highly-parallel and pipelinable computation. These programming models are gaining traction which can potentially be leveraged if a similar programming model can be developed for FPGAs.

In addition to the programming model, there are also several main architectural features of GPUs that are very desirable for a high-throughput soft processor. In particular, while some of these features have been implemented previously in isolation and shown to be beneficial for soft processors, our research highlights that when implemented in concert they are key for the design of a high-throughput soft processor.

Multithreading Through hardware support for multiple threads, a soft processor can tolerate memory and pipeline latency and avoid the area and potential clock frequency costs of hazard detection logic—as demonstrated in previous work for pipelines of up to seven stages and support for up to eight threads [1, 4, 8]. In our high-throughput soft processor we essentially avoid stalls of any kind for very deeply pipelined functional units (64 stages) via hardware support for many concurrent threads (currently up to 256 threads).

Vector Operations A vector operation specifies an array of memory or register elements on which to perform an operation. Vector operations exploit data-level parallelism as described by software, allowing fewer instructions to command larger amounts of computation, and providing a powerful axis along which to scale the size of a single soft processor to improve performance [10, 11].

Predication To allow program flexibility it is necessary to support control flow within a thread, although any control flow will make it more challenging to keep the datapath fully utilized—hence we support predicated instructions that execute unconditionally, but have no impact on machine state for control paths that are not taken.

Multiple Processors While multithreading can allow a single datapath to be fully utilized, instantiating multiple processors can allow a design to be scaled-up to use available FPGA resources and memory bandwidth [5]. The GPU programming model specifies an abundance of threads, and is agnostic to whether those threads are executed in the multithreaded contexts of a single processor or across multiple processors. Hence the programming model and architecture are fully capable of supporting multiple processors, although we do not yet evaluate such systems.

2. INITIAL IMPLEMENTATION

Together, the above features provide the latency tolerance, parallelism, and architectural simplicity required for a high-throughput soft processor. Rather than invent a new programming model, ISA, and processor architecture to support these features, as a starting point for this research we have ported an existing GPU programming model and architecture to an FPGA accelerator system. Specifically, we have implemented a `system-C` simulation of a GPU-inspired soft processor that (i) is programmable via NVIDIA's high-level `C`-based language called `Cg` [7], (ii) supports an *application binary interface* (ABI) based the AMD CTM r5xx GPU ISA [2], and (iii) is realizable on an XtremeData XD1000 development system composed of a dual-socket motherboard with an AMD Opteron CPU and the FPGA module which communicate via a HyperTransport (HT) link. We find that our heavily-multithreaded GPU-inspired architecture can overcome several key challenges in the design of a high-throughput soft processor for acceleration—namely (i) the port limitations of on-chip memories in the design of the main register file, (ii) tolerating long latencies to memory, and (iii) tolerating the long latency of deeply-pipelined functional units.

We avoid the problem of port limitations for the main register file by exploiting the fact that all threads are executing different instances of the same shader program: all threads will execute the exact same sequence of instructions, since even control flow is equalized across threads via predication. This symmetry across threads allows us to group threads into *batches* and execute the instructions of batched threads in lock-step. This lock-step execution in turn allows us to *transpose* the access of registers to alleviate the ports problem: rather than reading all of the operands of a single instruction from a single bank of the register file in a given cycle, we instead read only one operand across instructions in a batch each cycle. This takes several cycles to read all of the operands for a single instruction, but at that point all of the instructions in the batch are ready to execute.

We also tolerate the memory and pipeline latency by exploiting the numerous and batched threads: we store the contexts of multiple batches in hardware, and dynamically switch between batches every cycle. We capitalize on the fact that all threads are independent across batches as well as within batches, switching between batches to hide both pipeline and memory latency. This allows us to potentially fully-utilize the pipelined datapath.

3. RESULTS AND ONGOING WORK

We have built a `SystemC`-based simulator of our high-throughput soft processor that also models the memory bandwidth and latency of the XtremeData system. Our implementation is compatible with the CTM GPU driver interface, allowing us to simply re-link existing CTM applications with our own library. Initial measurements of floating-point matrix-multiplication benchmarks show that with 16 and fewer batches (powers of two) of four threads per batch the datapath is under-utilized because of having to conservatively observe potential data hazards across the deep floating-point pipeline. However, we find that with 32 batches we can achieve 100% utilization of the datapath, and that support for 64 batches are unnecessary.

Our long-term research goal is to use this system to gain insight on how to best architect a soft processor and programming model for FPGA-based acceleration. We envision that several aspects of this architecture can be extended in future implementations to better capitalize on the strengths of FPGAs: we can scale the soft processor in the vector dimension as in previous work [10,11]; rather than focusing on floating-point computation, we can instead focus on non-standard bit-width computation or other custom functions; finally, we can scale the number of soft processor accelerators via multiprocessor implementations to fully-utilize available memory bandwidth.

4. REFERENCES

[1] B. Fort, D. Capalija, Z. G. Vranesic, and S. D. Brown. A multithreaded soft processor for sopc area reduction. *Field-Programmable Custom Computing Machines, 2006. FCCM '06. 14th Annual IEEE Symposium on*, pages 131–142, April 2006.

[2] J. Hensley. Amd ctm overview. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 7, New York, NY, USA, 2007. ACM.

[3] J. Koo, D. Fernandez, A. Haddad, and W. Gross. Evaluation of a high-level-language methodology for high-performance reconfigurable computers. *Application-specific Systems, Architectures and Processors*, July 2007.

[4] M. Labrecque and J. Steffan. Improving pipelined soft processors with multithreading. *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 210–215, Aug. 2007.

[5] M. Labrecque, P. Yiannacouras, and J. G. Steffan. Scaling soft processor systems. In *IEEE Symposium on Field-Programmable Custom Computing Machines*, 2008.

[6] D. Lau, O. Pritchard, and P. Molson. Automated generation of hardware accelerators with direct memory access from ansi/iso standard c functions. *Field-Programmable Custom Computing Machines, 2006. FCCM '06. 14th Annual IEEE Symposium on*, pages 45–56, April 2006.

[7] W. R. Mark, R. S. Glanville, K. Akeley, and M. J. Kilgard. Cg: a system for programming graphics hardware in a c-like language. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 896–907, New York, NY, USA, 2003. ACM.

[8] R. Moussali, N. Ghanem, and M. A. R. Saghir. Supporting multithreading in configurable soft processor cores. In *CASES 2007*, pages 155–159.

[9] J. L. Tripp, K. D. Peterson, C. Ahrens, J. D. Poznanovic, and M. Gokhale. Trident: An fpga compiler framework for floating-point algorithms. *FPL*, pages 317–322, 2005.

[10] P. Yiannacouras, J. G. Steffan, and J. Rose. Vespa: Portable, scalable, and flexible fpga-based vector processors. In *CASES'08: International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2008.

[11] J. Yu, G. Lemieux, and C. Eagleston. Vector processing as a soft-core cpu accelerator. In *Proceedings of the 16th international ACM/SIGDA symposium on Field programmable gate arrays*, 2008.