

A Scalable Processor with Embedded Software for Large-Scale Scientific Applications

Daniel Alex Finkelstein and Haldun Hadimioglu
Computer and Information Science
Polytechnic University
Brooklyn, NY 11201
Email: dfinke01@cis.poly.edu haldun@photon.poly.edu

I. PREFACE

We present a scalable, dynamically reconfigurable FPGA-based processor design that encompasses both reconfigurable circuitry and software programmability for supercomputing applications. Advanced FPGA chips contain both reconfigurable logic blocks and embedded processor cores, providing the developer with an environment for embedded system design¹. Since the reconfigurable fabric and the embedded processor cores can be programmed and used independently of each other or in any combination with each other, the designer has a flexible platform in which new avenues of research are possible.

II. RELATED WORK

The necessity of simulation to measure performance and validate computer architectures is widely accepted, however the expense with respect to simulation time and computer resources required to execute simulations is an ongoing challenge. Current CPU simulators can produce cycle-accurate results with reasonable hardware resources (a desktop PC may be sufficient) [4] [5] [12]. System simulators have gained in popularity and relevance as workloads beyond traditional kernels or programs from the SPEC CPU 2000 suite are tested [2]; they require both more sophistication in the simulator environment as well as greater investment in hardware resources to run them within tolerable execution times. The Virtutech Simics system simulator [3] requires a complete operating system image and sufficient dedicated memory to test workloads that often involve transaction and application services. Full system simulators require dedicated hosts with resources (CPU, memory, and disk) that exceed those commonly found on workstations.

Nonetheless, software-based cycle accurate simulators are considerably slow [6]. Thus, object-oriented processor simulators allow differing levels of granularity in the algorithm's model [4] [12] and extensions to full system simulators utilize the simulator's API to provide narrower and more focussed

¹Xilinx Virtex II Pro [14] and higher devices include so-called 'hard' PowerPC embedded RISC processor cores, while the same Xilinx products and Altera FPGAs support 'soft' cores, which are functionally similar RISC processor cores configured within the FPGA CLB fabric.

areas of investigation for architects within a larger framework [10]. In contrast to these software-only simulation environments, FPGAs offer a bifurcated simulation path for hardware algorithm modeling, prototyping, and performance testing.

FPGA usage can be classified into two general areas: FPGAs as a prototype substrate (with the eventual goal of fabrication on silicon) and FPGAs as a reconfigurable substrate. The Raw project [11] and Heithecker's paper [8] use FPGAs to rapidly implement hardware algorithms and obtain performance measurements prior to silicon fabrication. FPGA-based systems Molen [13] and Garp [7] use FPGAs as a prototyping platform for dynamic application acceleration, although Garp suggests that the FPGA will remain part of the final processor architecture. However, Garp binds a RISC processor externally to an FPGA as opposed to embedding it within the FPGA fabric as in the case of the Xilinx Virtex II Pro processors. This may be nothing less than a semantic difference or could, upon further inspection, result in fundamentally different instruction- and data-flow properties. The Cray XD1 uses the Virtex II Pro FPGA purely as an application accelerator, coupled between AMD processors and DRAM main memory [1].

III. CONTRIBUTION

We expand the notion of reconfigurability by introducing a processor that is both reconfigurable and programmable. The processor is reconfigurable in the sense that the configurable logic blocks of the FPGA fabric can be remapped to perform different functions at will. The processor is programmable in the sense that the embedded RISC core runs both application code as well as dedicated internal-only code that serves control and monitoring functions.

A. Platform

Our development environment consists of Xilinx ML310 development boards [14] with Virtex II Pro XC2VP30 FPGA processors, ISE Foundation programmable logic design tool (ISE) 7.1i, Embedded Development Kit and Platform Studio (EDK) 7.1i, and ModelSim 6.1.

The Virtex II Pro XC2VP30 FPGA contains more than 30,000 logic cells, over 2,400 Kb of BRAM, and dual PPC405 processors. The ML310 contains an Ethernet MAC/PHY, 256 MB DDR memory, multiple PCI slots, and standard PC I/O

ports within an ATX form factor board. An integrated System ACE CF controller is deployed to perform board bring-up and to load applications from the included 512MB CompactFlash card. The Virtex II Pro can be partially and dynamically reconfigured.

The ISE provides an integrated development environment for custom VHDL and Verilog programming while the EDK produces a processor environment through a base system builder, to which custom hardware resources are added in Verilog or VHDL. The EDK is required for programming the hard (PowerPC) and soft (MicroBlaze) cores on the FPGA as well as integrating the peripheral components on the development board.

B. Design

We use both the reconfigurable logic blocks (the ‘hardware’) and the programmable embedded processor cores (the ‘software’) in varying combinations so that applications can be rapidly tested for performance by selectively partitioning the algorithm into portions suitable for the hardware and software resources on the FPGA. In contrast, current FPGA research concentrates on application acceleration to find optimal partition points of algorithms (or programs, as they are often identified) in which computationally challenging portions are split off into dedicated hardware as in an ASIC through either static analysis of program code or dynamic runtime performance measurement. Our use of the hybrid FPGA device, instead, extends the processor’s capabilities. The programmable embedded processor core has dual roles: it controls the flow of execution and data movements within the processor as well as executing instructions on the application’s behalf.

The logic blocks are reconfigured by the Xilinx-provided IP for communication with peripherals on the development board. We insert our own VHDL code for the application and for most of the processor’s other functionality into the same IP. Higher level control and monitoring, which is typically dynamic, is relegated to the embedded software portion of the processor design. This includes Xilinx-provided hardware drivers and the code we write as an interface between the application and Xilinx drivers. Communication between the hardware and software on the hybrid FPGA is accomplished through software-visible registers that are mapped to VHDL ports in the custom hardware.

The hybrid FPGA is conducive to a flexible execution platform encompassing both traditional processor components and dynamic programmable structures². Traditional processors can be fashioned wholly in the FPGA reconfigurable fabric or by using on-chip processor cores (or both), however the hybrid FPGA contains more logic available for the architect’s use. Expansion of the processor’s role in execution is possible by integrating this logic within the boundary of the processor, including the functions of controllers. With an inclusive execution platform, supervisory (control and monitoring) software

tasks signal the processor to dynamically adjust its performance in reaction to operational parameters.

We plan to implement parameterized execution through a vector of parameters in which power requirements, memory availability, disk latencies and availability and throughput, network latencies and throughput, and priority weightings among them are passed to the processor so that control decisions can be modified to cause the processor’s performance to more closely match the desired performance. Memory access is of particular interest due to its inextricable relevance to overall performance; to that end, the bifurcated processor environment enables a new method for reducing random access penalties in DRAM and cache misses in cache memories. If a program can be treated as two streams – instruction and data – and the entry point for the instruction stream is the high level, or software-based, controller of the processor, then the processor may selectively retrieve data from peripherals based on values in the performance vector and the availability of computational resources tracked by the processor.

Retrieving data from tertiary storage, whether from local or networked fixed disks in addition to traditional network data, is often orders of magnitude slower than other resources further down the memory hierarchy closer to the CPU. A persistent challenge is to prevent the CPU from waiting for data no matter from what source it must be retrieved, but we posit that the load-store architecture is not optimal for this task. As networks approach throughput speeds closer to that of the processor local bus and chip designs include the network controller directly on the processor local bus [9], network latencies become less pressing an issue, especially in comparison to fixed disks. In large systems, however, clusters of disks can be controlled in a multiplexed manner such that overall throughput may approach that of network throughput we currently observe.

Latencies in peripheral access remain a concern with respect to application execution delays. At this juncture we suggest that the common wisdom of keeping a CPU busy with tasks may be misdirected; rather, we suggest that an *application* not be delayed during execution. The processor may be idle without concern so long as this is not due to a resource contention, unavailable data, or other factor related to execution of the application. The hybrid FPGA processor described in this abstract drives additional application performance by allocating portions of the instruction stream among the programmable and reconfigurable cores, dynamically routing both instruction and data streams to areas of the processor best suited for the particular execution task. When considering the capability of the processor to subsume control functions of peripherals and higher-level memory hierarchy components, the programmable portion of the processor may reduce (or eliminate) the need for load-store instructions by requesting data streams from off-chip storage as per the application’s instruction sequence. When data structures and operations upon them are recognized by the processor either through explicit information in the instruction stream or implicit detection by the programmable core, data

²Programmable cores are akin to microengines found in network processors where dedicated circuits are not required yet high execution speed is desired.

can be prefetched and cached with precise ordering rather than traditional locality and heuristic based approaches.

IV. CONCLUSIONS

We present a scalable reconfigurable processor with embedded software for large-scale scientific applications. The tasks assigned to the hardware and software can be varied dynamically based on run-time and user parameters.

Because the processor includes HDL code, it can more easily scale in a multiprocessor configuration through dynamic data structures that are challenging to implement in fixed, nonprogrammable circuits. Though we don't address the important issue of control in a distributed system here, a system consisting of several hybrid processors can cooperatively exist through processor-processor interaction enabled by adding elements to the performance vector.

A. Future Work

Traditional power conservation techniques usually rely on adjusting the global clock or selectively powering on or off off-processor circuits; our approach allows for multiple domain clocks to operate independently of the global clock by virtue of control from within the processor itself. An unavoidable off-chip throughput delay cannot be compensated for by the processor, so we allow the processor to adjust its related execution circuits to match the input lines by either slowing the domain clock of the related execution circuits and deallocating functional execution units (if necessary), or vice versa should the throughput of the input lines increase. In a low-power mode, all bets are off: the processor's high level controller will continue to attempt a best-performance allocation scheme but only if the configuration does not exceed the power requirement.

V. ACKNOWLEDGMENTS

We wish to acknowledge the support of Xilinx in providing us with ML310 development boards, design tools, and technical support. This work was supported in part by a research fellowship from the U.S. Department of Education GAANN.

REFERENCES

- [1] Cray XD1.
- [2] SPEC CPU 2000, <http://www.spec.org/cpu2000>.
- [3] Virtutech Simics, <http://www.virtutech.com>.
- [4] N. L. Binkert, E. G. Hallnor, and S. K. Reinhardt. Network-Oriented Full-System Simulation using M5. *Sixth Workshop on Computer Architecture Evaluation using Commercial Workloads*, February 2003.
- [5] D. C. Burger and T. M. Austin. The SimpleScalar Tool Set, Version 2.0. Technical Report CS-TR-1997-1342, University of Wisconsin-Madison, 1997.
- [6] D. Chiu. FAST: FPGA-based Acceleration of Simulator Timing Models. *Workshop on Architecture Research using FPGA Platforms*, February 2005.
- [7] J. R. Hauser and J. Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Coprocessor. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 12–21. IEEE, April 1997.
- [8] S. Heithecker and R. Ernst. Traffic shaping for an FPGA based SDRAM controller with complex QoS requirements. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 575–578, San Diego, California, 2005. ACM Press.

- [9] Intel IOP331 I/O Processor. <http://www.intel.com/design/iio/iop331.htm>.
- [10] M. M. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood. Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *Computer Architecture News*, 2005.
- [11] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffmann, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpfen, M. Frank, S. Amarasinghe, and A. Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. *IEEE Micro*, 22(2):25–35, March-April 2002.
- [12] M. Vachharajani, N. Vachharajani, D. A. Penry, J. A. Blome, and D. I. August. Microarchitectural Exploration with Liberty. In *Proceedings of the 35th International Symposium on Microarchitecture (MICRO)*, pages 271–282, November 2002.
- [13] S. Vassiliadis, S. Wong, and S. Cotofana. The MOLEN μ -coded processor. *Lecture Notes in Computer Science*, 2147:275–285, 2001.
- [14] Xilinx ML310 Embedded Development Board. <http://www.xilinx.com>.