Dynamic Computation Structures: User-level Hardware for Reconfigurable Architectures

Jonathan Babb and Matthew Frank *
jbabb@lcs.mit.edu, mfrank@lcs.mit.edu
MIT Laboratory for Computer Science

Introduction

Configurable computers based on Field Programmable Gate Arrays (FPGAs) are capable of accelerating suitable applications by several orders of magnitude when compared to traditional processor-based architectures [3]. However, the programming paradigm for such machines is often no more than a set of hand-crafted circuits, one per FPGA in the system. Programmers must explicitly account for machine-level details, such as FPGA capacity and interconnect, prohibiting the development of architecturally-independent applications.

We are addressing these issues as a part of the Reconfigurable Architecture Workstation (RAW) project, work to implement a reconfigurable computer based on a computational model that does not mandate either a fixed instruction set or a fixed memory system model; rather it builds a user configurable instruction set on top of fixed, core instruction set, and synthesizes a user-configurable system model on top of primitive, commodity, hardware elements such as DRAM and FPGAs. This work leverages our previous FPGA work, Virtual Wires [1], in conjunction with behavioral compilation technology, to view an array of FPGAs as a machine-independent computing fabric.

The RAW machine comprises very simple commodity hardware including SRAMs, DRAMs, a large amount of primitive programmable logic in the form of Field-Programmable Gate Arrays (FPGAs), and fast microprocessors. Unlike current computers in which the system logic is hardwired for all time, RAW machines permit user-level, compiler, or operating system assisted configuration of the architecture of the computer. Possible configuration models for the RAW machine include: simple changes to the memory model, creation of new instructions, and complete configuration of application-specific, user-level hardware. We introduces a compilation paradigm for the last model.

*Advisor: Anant Agarwal. Support: in part by ARPA contract N00014-94-1-09885, by NSF Experimental Systems grant MIP-9012773, and by an NSF Presidential Young Investigator Award to Prof. Agarwal. Matthew Frank is in part supported by an NSF Graduate Fellowship. Address: Room 633 (Babb) and Room 627 (Frank), 545 Technology Square, Cambridge MA 02139. Web: http://www.cag.lcs.mit.edu/ jbabb, http://www.cag.lcs.mit.edu/ mfrank

Dynamic Computation Structures

To explore user-level hardware for RAW, we introduce *dynamic computation structures*, dynamic code [2] for reconfigurable computing. Rather than dynamically generating new processor instructions based on the input data-set, we dynamically modify the underlying hardware architecture as a function of the user's input. Our system translates specific user input into dynamic computation structures, generating synthesizable behavioral verilog. Functionally, our technique extends dynamic code generation, in which specialization enables constant folding, dead code elimination, and removal of branches to expose instruction parallelism. To specialize below the instruction set abstraction, our technique automatically creates new operators, minimizes data widths, and reduces many complex data structures to simple wire permutations.

While the generated verilog code is application-specific, it is architecturally-independent and can be targeted at any FPGA devices or arrangements. To this end, the following components of our current system consist of the Synposys behavioral and RTL compilers, the Virtual Machine Works partitioning and virtualization software, and the Xilinx FPGA-specific place-and-route (Figure 1). These layers hide successive levels of detail from the DCS generator: the FPGA layer hides the internal FPGA details; the VMW layers hides the inter-FPGA topology and FPGA gate and pin capacity; the RTL layer hides the VMW/FPGA technology libraries; and the behavioral layer hides state machine and datapath details.

Experimental Results

In our current experiments, we use dynamic computation structures to specialize directed graph instances onto a Virtual Machine Works logic emulator. Our research is motivated by the fact that logic emulation software already maps a special case of directed graphs, netlists, onto an FPGA array. The problems we solve, including transitive closure and shortest path, exhibit the general properties of closed semirings, an algebraic structure for solving directed paths. These semirings are iterative subproblems of flow and matching linear programs, which in turn often form the inner loop

Machine (compiler)	nodes	edges	edge (us)	iter (us)	total (us)	rate (Hz)
SPARC 20 (gcc -02)	64	256	0.110	28.13	1800	555 Hz
SPARC 20 (specialized)	64	256	0.078	20.00	1270	787 Hz
SPARC 20 (Synopsys)	64	256	3.500	3.50	222	4500 Hz

Figure 2: Shortest Path Results

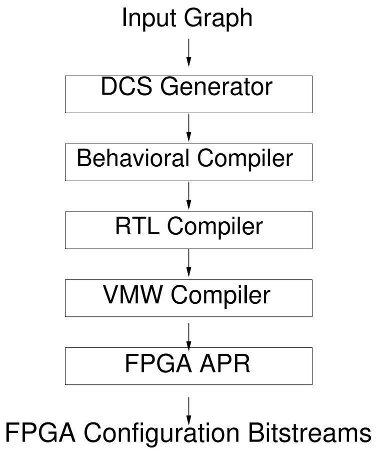

Figure 1: Compiler Flow

Structure	Combinational	Sequential	Total
Edge	224 gates	128 gates	352 gates
Node (degree 0)	64 gates	172 gates	236 gates
Node (degree 1)	178 gates	172 gates	350 gates
Node (degree 2)	310 gates	172 gates	482 gates
Node (degree 3)	429 gates	172 gates	601 gates
Node (degree 4)	544 gates	172 gates	716 gates
Node (degree 5)	645 gates	172 gates	817 gates
Node (degree 6)	732 gates	172 gates	904 gates
Node (degree 7)	891 gates	172 gates	1063 gates
Node (degree 8)	1009 gates	172 gates	1181 gates

Figure 3: Dynamic Computation Structure Sizes

of complex, combinatorial optimizations for power generation and transmission, water resource allocation, and silicon compilation.

We implemented both a DCS generator and a dynamic code generator for the the classic Bellman-Ford shortest path algorithm. For the problem size shown, we achieve a 40 percent improvement with dynamic code generation on a 125 SPECint workstation, and an order of magnitude improvement with dynamic computation structures on 1.1 million FPGA gates, when compared to the same workstation (Table 2). While code specialization benefits significantly by unrolling the inner loop and de-referencing pointers, further architectural specialization allows parallelization of the entire inner loop, producing speedup proportional to graph edges. Table 3 shows the resulting combinational and non-combinational VMW gate count details (1 VMW gate $\approx$ 2.5 Xilinx FPGA gates) for the generated structures.

Future Work

Unlike circuit design, where compilation and synthesis time are secondary to optimization area/timing results, in reconfigurable computing compilation time is a primary concern. To this end we are developing a relocatable hardware library as a key input to the behavioral compiler. This library will consist of pre-compiler macro-blocks for computation to reduce synthesis time. Example macros include adders, multipliers, registers, and parameterized datapaths. These macros will be compiled down to the FPGA-bitstream level ahead of time, and prototype software to translate and stitch bitstreams together is under implementation.

In addition to speeding compile time, we will be addressing language issues for expressing computation structures as well as operating system approaches for securely multiplexing RAW hardware among multiple user-level specializations. While emulation hardware is useful for targeting the application-specific portion of the user-level hardware, we will be implementing a RAW prototype with both FPGAs and an on-board processor to further explore this new design realm.

References

[1] J. Babb, R. Tessier, and A. Agarwal. Virtual Wires: Overcoming pin limitations in FPGA-based logic emulators. In *Proceedings IEEE Workshop on FPGA-based Custom Computing Machines*, pages 142–151, Napa, CA, April 1993. IEEE. Also as MIT/LCS TM-491, January 1993.

[2] D. R. Engler. Vcode: A very fast, retargetable, and extensible dynamic code generation system. In *PLDI '96*, 1996.

[3] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweeney, and D. Lopresti. Building and using a highly parallel programmable logic array. *Computer*, 24(1), Jan. 1991.