

Compilation for Scalable, Paged Virtual Hardware

Ph.D Thesis Proposal

Executive Summary

Eylon Caspi

February 22, 2001

Reconfigurable computing devices such as Field Programmable Gate Arrays (FPGAs) have demonstrated 10x–100x gain over conventional microprocessors in performance and functional density (operations per area-time) for a variety of applications [6]. The strength of reconfigurable computing comes from its combining of spatial execution with programmability—the former allows computational data paths to be highly parallel, while the latter allows data paths to be highly specialized to the application at hand. The commercial marketplace has relegated reconfigurable devices to be used primarily as ASIC replacements, executing only a single static configuration. This usage ignores many of the key performance benefits of reconfigurable technology, for instance dynamic reconfiguration and run-time specialization.

The Berkeley SCORE project [4] contends that the present underuse of reconfigurable technology is due in great part to a lack of any unifying compute model to support its key technology benefits, to ease programming effort, and to enable software to survive and automatically scale to ever-improving, next-generation hardware. To this end, SCORE introduces a parallel computation model based on streaming data-flow graphs of communicating state-machines. A key element of the SCORE execution model is hardware virtualization, wherein a computation is partitioned into fixed-size hardware pages (analogous to virtual memory pages) that are automatically loaded and executed in available physical pages. Paging enables resource hiding in the programming model, liberating the programmer’s algorithmic decisions from device size constraints. Paging also enables binary compatibility between page-compatible devices, as well as performance scaling, in that application performance will automatically improve on a larger device with more physical pages, without recompilation. The efficiency of paged execution in SCORE relies heavily on the streaming data-flow aspect of the compute model, which exposes a program’s communication patterns and enables the construction of good partitions and good schedules for run-time page loading.

A salient feature of the SCORE compute model is that its primitives are consistent between the programming model (*i.e.* concrete language) and the execution model (*i.e.* technology-specific page configurations). In both cases, a computation is composed as a data-flow graph of compute operators (in the sense of a Kahn process network), where each operator contains a data path along with a finite state machine that implements data-flow input-output semantics and possibly data-dependent sequencing. The difference is in the binding of resource constraints. Operators in the execution model are hardware pages with fixed area, fixed number of IOs, and technology-specific timing. Operators in the programming model have no such constraints on size, IO, timing, or other complexity. Thus, compilation and page synthesis for SCORE can be formulated as a mapping from one SCORE graph to another (*i.e.* a transformation on streaming state machines) such that the resulting operators satisfy hardware constraints.

The purpose of this Ph.D project is to explore and develop the mapping of SCORE programs from the programming model to the paged execution model. My focus will be on partitioning to satisfy the page area, IO, and timing constraints of a parameterized hardware model (note, this focus is different from and does not replace traditional compiler optimizations, which are meant to improve code parallelism and reduce work, independent of a hardware model). I will develop clustering techniques to partition and regroup state machines with data-paths under area and IO constraints. Although partitioning problems are richly explored in the literature, existing partitioning heuristics are not sufficient for SCORE in and of themselves, for a variety of reasons discussed below. I will implement the proposed partitioning techniques within the existing SCORE compiler framework. I will evaluate the "efficiency" of the proposed techniques in terms of their effect on circuit area (control overhead, page fragmentation) and performance (delay, total run-time) for a variety of applications and hardware parameter settings. The goal of this project as a supporting component of SCORE is to demonstrate that automatic page generation can be efficient in the sense that it does not lead to substantial degradation in circuit area and performance.

The problem of partitioning SCORE computations is unique for a number of reasons. The partitioning must be robust to potentially very long inter-page communication latency (*e.g.* when pages are physically far away or not simultaneously loaded) and thus must avoid inter-page feedback loops. The partitioning must also retain a notion of streams across cuts and should prefer to cut low-activity streams so as to reduce the run-time storage required for cut streams. Hence the traditional approach of synthesizing a netlist and partitioning/covering it to minimize a single metric is not optimal (examples include wire minimization by min-cut [10] [13] [14] and Fiduccia-Mattheyses [7]; critical-path delay minimization by cone covering [12] [15] [9] [5]; wire length minimization by spectral partitioning [8]). Similarly, traditional techniques for FSM partitioning are not sufficient because they do not consider the costs of cutting associated data-paths and their respective data streams (exam-

ples include FSM decomposition to minimize logic [1]; minimize wires [11]; minimize power [2]). Efficient partitioning of a SCORE computation must be done at a level of abstraction higher than the netlist and broader than the FSM, optimizing for a combination of factors.

The developmental and experimental aspects of my work will be done within the existing SCORE software infrastructure (of which I was a key developer). In particular, I will extend the SCORE compiler [3], which presently includes a language front-end, rudimentary optimizations, and a back-end to emit C++ behavioral simulation code for pages. Performance evaluation will be done using the existing device simulator, which runs together with an existing run-time scheduler that loads and executes pages. The hardware model is parameterized and lends itself well to architectural experimentation such as varying page size. This sizeable software infrastructure is under continuing support and development by the members of the Berkeley BRASS research group (including myself).

Initial development of a complete synthesis and partitioning flow for SCORE is already under way and will be discussed in the qualifying exam.

References

- [1] P. Ashar, S. Devadas, and A. R. Newton. Optimum and heuristic algorithms for an approach to finite state machine decomposition. *IEEE Trans. on Computer-Aided Design*, 10(3):296–310, March 1991.
- [2] L. Benini, G. De Micheli, and F. Vermeulen. Finite-state machine partitioning for low power. In *Proc. IEEE Int'l Symposium on Circuits and Systems (ISCAS '98)*, volume 2, pages 5–8, Monterey, California, May31–June3, 1998.
- [3] Eylon Caspi. Programming SCORE. BRASS group internal technical report, April 2000.
- [4] Eylon Caspi, Michael Chu, Randy Huang, Nicholas Weaver, Joseph Yeh, John Wawrzynek, and André DeHon. Stream computations organized for reconfigurable execution (score): Extended abstract. In *Conference on Field Programmable Logic and Applications (FPL '2000)*, LNCS, pages 605–614. Springer-Verlag, August 28–30 2000.
- [5] Jason Cong and Yuzheng Ding. Flowmap: An optimal technology mapping algorithm for delay optimization in lookup-table based fpga designs. *IEEE Transactions on Computer-Aided Design*, 13(1):1–12, January 1994.
- [6] André DeHon. *Reconfigurable Architectures for General-Purpose Computing*. PhD thesis, MIT, 545 Technology Sq., Cambridge, MA 02139, September 1996.

- [7] C. M. Fiduccia and R. M. Mattheyses. A linear time heuristic for improving network partitions. In *Proceedings of the 19th Design Automation Conference*, pages 175–181, 1982.
- [8] L. Hagen and A. Kahng. New spectral methods for ratio-cut partitioning and clustering. *IEEE Trans. on Computer-Aided Design*, 11(9):1074–85, September 1992.
- [9] Kurt Keutzer. Dagon: Technology binding and local optimization by dag matching. In *Proceedings of the 24th ACM/IEEE Design Automation Conference (DAC)*, pages 341–347, 1987.
- [10] B. Krishnamurthy. An improved min-cut algorithm for partitioning vlsi networks. *IEEE Trans. on Computers*, C-33(5):438–446, May 1984.
- [11] Ming-Ter Kuo, Lung-Tien Liu, and Chung-Kuan Cheng. Finite state machine decomposition for i/o minimization. In *Proc. IEEE Int'l Symposium on Circuits and Systems (ISCAS '95)*, volume 2, pages 1061–4, Seattle, Washington, April28–May3, 1995.
- [12] Rajmohan Rajaraman and D. F. Wong. Optimal clustering for delay minimization. In *Proc. 30th Int'l Conf. on Design Automation Conference (DAC '93)*, pages 309–314, Dallas, Texas, June 14–18, 1993.
- [13] L. A. Sanchis. Multiple-way network partitioning. *IEEE Trans. on Computers*, 38(1):62–81, January 1989.
- [14] H. Yang and D. F. Wong. Efficient network flow based min-cut balanced partitioning. In *Proc. IEEE Int'l Conf. on Computer Aided Design (ICCAD '94)*, pages 50–55, November 1994.
- [15] H. Yang and D. F. Wong. Circuit clustering for delay minimization under area and pin constraints. *IEEE Trans. on Computer-Aided Design*, 16(9):976–986, September 1997.