Appearing in Third International Conference on Unconventional Models of Computation (UMC'02)

Very Large Scale Spatial Computing

André DeHon

California Institute of Technology
Department of Computer Science, 256-80, Pasadena, CA 91125, USA
andre@acm.org

Abstract. The early decades of computing were marked by limited resources. However, as we enter the twenty-first century, silicon is offering enormous computing resources on a single die and molecular-scale devices appear plausible offering a path to even greater capacities. Exploiting the capacities of these modern and future devices demands different computational models and radical shifts in the way we organize, capture, and optimize computations. A key shift is toward spatially organized computation. A natural consequence is that the dominant effects which govern our computing space change from the total number of operations and temporal locality to interconnect complexity and spatial locality. Old computational models which hide, ignore, or obfuscate communication and emphasize temporal sequences inhibit the exploitation of these modern capacities, motivating the need for new models which make communication and spatial organization more apparent.

1 Introduction

Severely limited physical capacity has been a stark reality of computer design from the 1940's. In the pre-VLSI era, we were limited by the physical bulk and cost of relays, vacuum tubes, and discrete transistors. In the early VLSI era, we were limited by the capacity of a single chip. Consequently, practical computing devices have been organized around clever ways to use small amounts of hardware to solve large problems.

The advent of compact memory technologies (core memory, IC memories, DRAMs) coupled with the observation that we could describe a computation and its state compactly, allowed us to reduce the size of a computation by time-multiplexing the active hardware across many operations in the computation. The most familiar embodiment of this is the sequential *processor*, where we use a few tens of bits to store each instruction, a large memory to store the state of the computation, and a single, or small number, of active computing devices to evaluate the large computation sequentially.

Our conventional abstractions for describing computations (the Instruction Set Architecture (ISA) at the machine code level, sequential programming languages like C, FORTRAN, and Java at the programmer level) grew out of this poverty. They allowed programmers and compilers to describe computations in a manner cognizant of the capacity limitations of these devices. They focused the developer on the costly items for these machines: the operations and the use of memory. Programs were optimized by reducing the number of operations and minimizing the amount of live state.

As we are all aware, the steady shrink of feature sizes in integrated circuit processing has produced more capacity per die and per dollar at an exponential rate. Our computing

devices have been steadily getting richer. While we may see the end of exponential IC feature size scaling in the next couple of decades [12], advances in basic science are suggesting ways to cheaply engineer computing devices at the molecular scale and in three dimensions.

With exponentially increasing capacity, we inevitably reach a point where resource capacity is no longer such a scarce capacity. That point may vary from task to task, and there are some problems that may always feel the crunch of limited resources—particularly problems we cannot attack today even with all of our well-honed tricks to reduce system size. Nonetheless, for a larger and larger set of problems, we are crossing the poverty threshold to the world of abundance.

By 2016, for example, silicon technology is promising 900 million gates in $2.3 \times 10^{12}\lambda^2$ of silicon [1]. However, we were already building quite competent single-chip processors by the early 1990's with less than $2 \times 10^9\lambda^2$ of silicon (*e.g.* [6] [23]). Various estimates suggest we may be able to achieve 10^{10} gates/cm^2 using molecular scale electronics [7] in only two dimensions; that is another $30\times$ the density of 2016 silicon.

What happens when we have an abundance of resources?

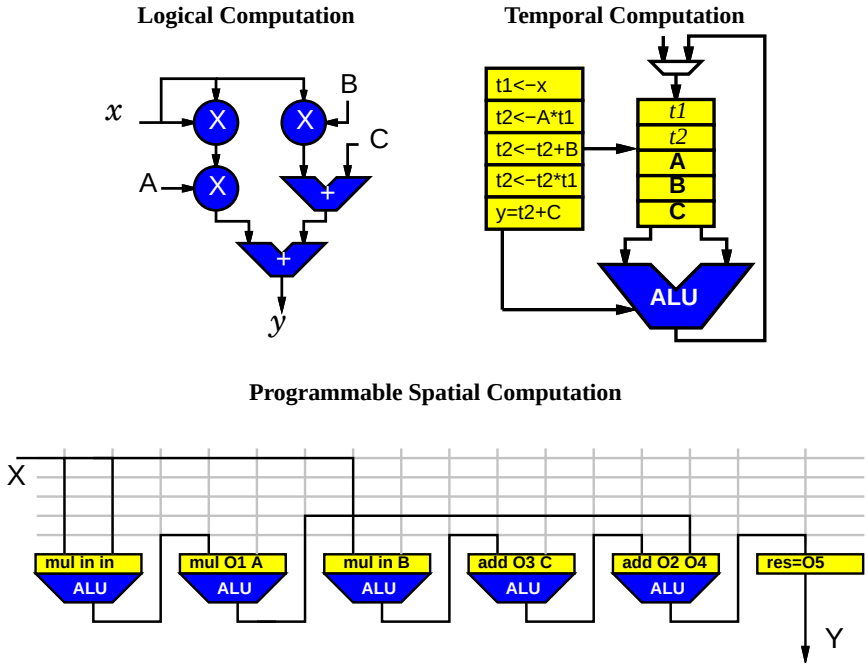
Minimal existence has the advantage that it certainly continues to work in times of plenty. But, it is not necessarily the best or more efficient way to implement a computation when more resources exist. It is this problem which computing has, for good and practical reasons, largely ignored for the past 50 years. The current and future prospects for high capacity devices and systems suggest it is now practical, beneficial, and necessary to rethink the way we formulate computations. The abstractions, metrics, and optimizations which suited capacity poor systems can be quite inappropriate when resources are less limited. Consequently, we now need to find new models, optimizations, and algorithms suitable to exploit our growing wealth of computational capacity.

2 Trend toward Spatial Computing

When capacity is abundant, we are not driven to the temporal extreme where we time-multiplex a single active computation among a large number of operators. At the opposite extreme, we give each operation its own active operator. Operations are interconnected in *space* rather than *time* (See Figure 1). Notably, this means we exploit the full parallelism available in the task, completing the task in time proportional to the longest path in the computation rather than in time proportional to the number of operations. In fact, for many designs, we may be able to pipeline the computation and produce new results every unit of operator delay.

The spatial design is larger than the minimum size temporal design, trading increased area to hold more active operators for decreased time. As area becomes less of a limiting factor in designs, we can accelerate our computation by moving to more spatial designs. Further, even programmable spatial design are more effective at exploiting high capacity than both conventional processors and multiprocessors.

Spatial designs were originally the sole domain of custom silicon application (*e.g.* custom VLSI or ASICs for dedicated signal processing tasks, video compression). They were only used for limited tasks which required computational rates infeasible with



In spatial implementations, each operator exists at a different point in space, allowing the computation to exploit parallelism to achieve high throughput and low computational latencies. In temporal implementations, a small number of compute resources are reused in time, allowing the computation to be implemented compactly.

Fig. 1. Spatial versus Temporal Computation for the expression $y = \mathrm{A}x^2 + \mathrm{B}x + \mathrm{C}$

time-multiplexed, programmable designs. As capacity has grown, programmable spatial designs have become increasingly practical using Field-Programmable Gate-Arrays (FPGAs). With the capacity available in today's FPGAs, many computing tasks can be implemented entirely spatially in a single device (*e.g.* digital filters, video and cryptographic encoding and decoding). It is worthwhile to note that the size of the FPGA device is often equivalent to the size of the sequential processor, while providing orders of magnitude greater performance [3].

Conventional, sequential processors struggle to extract more parallelism from tasks described using the capacity poor sequential ISA abstraction. The abstraction, itself, however limits and obfuscates available parallelism. Supporting the abstraction, requires substantially more hardware capacity (*e.g.* renaming, issue logic, reorder buffers) for model overhead than the capacity which is applied to the computational task.

Unlike conventional parallel processors, which are also based around heavily time-multiplexed processing nodes using the ISA abstraction, spatial computations exploit *regularity* in the computing task to build more compact, active processing nodes. A

spatial operator can be built or programmed to perform the *same* operation repeatedly on a sequence of data. In the custom hardware world, this means we build a hardwired datapath that does just one thing, but is used efficiently because we need to do that one thing over and over again. In the FPGA or programmable spatial world, this means we factor out the components of a computation which are needed repeatedly and configure a spatial unit to compute them efficiently. Signal processing, encryption, compression, and image manipulation are common examples of applications which require repeated application of the same computation to a large set of data. More generally the oft-quoted 90/10 rule suggests that large portions of typical computations (90% of the dynamic operation count) require the repeated application of identical or similar computational functions (10% of the static task description). In contrast, parallel processing nodes still require a large investment in memory to describe and hold state for a large number of different operations. As a result, a spatial processing operator is less expensive than a small sequential processing node. This, in part, is how an FPGA-like device can often require as little area as a sequential processing device while offering orders of magnitude greater performance.

3 Interconnect Dominance

As we exploit more spatial parallelism by employing more active, communicating processing elements, we must interconnect a greater number of components. For these system sizes and relative delays, interconnect issues are of paramount importance as interconnect can quickly becomes the dominant resource in the design consuming area, delay, and energy.

If we simply placed operators randomly onto the available die space, with high likelihood about half of the operators on the left half of the chip will want an input from the right half of the chip, and vice-versa. That means, we have around N/2 wires crossing the middle of the chip from left to right. We can make a similar argument from top to bottom. Consequently, given a fixed number of wiring layers, it will require a chip of size $O(N^2)$ simply to handle N communicating operators. In such a random placement, the average distance between connected operators is $1/3$ of the length of the chip in each dimension.

In the past we could ignore the effect of distance on delay because our components were too small for interconnect delay across a chip to be a dominant delay component. With growing chip capacity and shrinking feature sizes, the minimum delay across a chip is now large compared to desirable computational cycles [1]. This is especially troubling in light of the observation above that average communication distances for random placement could be 2/3 of the length of the die side. Since we know that delay is a function of distance with fundamental limits on the speed of propagation (*e.g.* speed of light), the fact that distance between operators implies delays should not be surprising. It was only the particular size and gate delay constants in early computing technologies that allowed us to ignore this effect for so long. By the 45nm node in the ITRS Roadmap [1], it looks like we will only be able to reach a radius of 200 custom 2-input gates [24] or 10 programmable bit operators without making interconnect a large fraction of gate-to-gate delay. This suggests we can, at most, travel a distance of a several thousand

custom gates or a several hundred programmable bit operators in a single cycle on an optimally buffered wire. Since our systems will be large compared to these radii, it becomes important both to layout computations to minimize communication distances (See Section 6) and to pipeline distant interconnect as in our HSRA [26].

4 New Abstractions

To serve our increasingly spatial, increasingly communication dominated, computations, we need compute models which emphasize the parallelism and communication which occurs in these devices.

Conventional models hide communication by layering it on top of memory operations to a single, monolithic memory space; this makes it hard to rediscover the links which exist between operators and impossible to identify with certainty all the links which may exist. Using memory for communication was necessary to communicate data compactly in capacity poor, time-multiplexed systems; it also facilitated a number of optimization, such as memory location reuse, which allowed one to reduce the amount of capacity needed to evaluate a computation. However, these descriptions do not facilitate efficient spatial computations.

For spatial computation, graph-based computing models offer a number of important advantages over sequential models. Most notably, they support parallelism and make communication links between operators explicit. A number of graph-based computing models exist, dating at least back to Kahn [13]. We summarize a number of such models in [2] and introduce SCORE, our own version which attempts to pull the best ideas from many sources and provide a suitable model for today's spatial computing.

In SCORE, the computation is a graph of operator nodes connected by persistent dataflow links, or *streams*. The graph can be of arbitrary size and may evolve during the computation. Once a graph or subgraph is created it operates on the data it is given through its input streams, producing new results to its output streams. Data on the streams is tagged with presence, providing deterministic, timing-independent behavior for the graph. Operators may be composed hierarchically.

In addition to exposing parallelism and communications, the persistent operators or subgraphs help capture the regularity which exists in the computation. Heavily used subgraphs with long persistence merit spatial implementations, whereas transient or infrequently used subgraphs may still benefit from temporal implementations. The division point between the spatial and temporal domain can depend on the available capacity and desired performance.

5 Architectures

Armed with these new abstractions, we can envision a class of spatial architectures suitable for this Very Large Scale Spatial Computing domain. At the highest level, we might imagine an arbitrarily large array of computation and memory nodes supported by suitable interconnect.

To manage this space cleanly, we organize the computing operators into a series of *compute pages* and the memory into a set of *memory blocks* (See Figure 2). A compute

page, for example, might hold a few hundreds or thousands of programmable bit operators or a few tens or hundreds of configurable datapath elements. The page organization has several advantages, including:

- It allows us to manage computations in modest size aggregates similar to the way we manage memory in pages in virtual memory systems. In both cases, this reduces the overhead in both time and space required to manage the mapping between virtual and physical resources.
- If we need to virtualize the physical space, pages become the unit of virtualization.
- The page size can be chosen relative to the technology so that intra-page communication can occur within an aggressive compute cycle while inter-page communications is pipelined to accommodate the greater distances.

As long as the compute pages obey the streaming dataflow communications discipline identified in the compute model, their microarchitecture becomes irrelevant to the rest of the computation. This makes it possible to build heterogeneous devices which include a wide range of spatial and temporal processing nodes (See Figure 2). The mix allows us to support the low-frequency and uncommon portions of the computation compactly in temporal form, while supporting the dominant, regular computation efficiently with a spatial implementations. The composition and mix of node types can evolve with the capacity offered by the technology while always supporting a single, unifying compute model.

6 Optimizing Spatial Computations

With abundant capacity such that it is not always necessary to time-multiplex operators to fit within available capacity, the key optimization will be to reduce the maximum *distance* along critical paths and loops. Area reduction does play a role here to the extent it makes designs more compact and reduces the distances over which critical signals must travel. Commonly communicating blocks should be arranged to optimize spatial locality. The presence of communication links in the design representation is vital to enabling these kinds of optimizations.

Area Our first concern is to place computations to reduce requisite wiring and switching. This is the traditional domain of placement and can reduce wiring requirements from the $O(N^2)$ area identified above to $O(N^{2p})$ [5], where p is the exponent in Rent's Rule [16] and typically has a value around 2/3. With sufficient wiring layer growth, it may be possible to even contain the two-dimensional active area to $O(N)$ [4].

We have some control in the design of our algorithms over the interconnect richness, p. It will be important to explore the extent to which we can design or optimize computing structures to reduce p. In the early days of VLSI, the systolic design style [14] focused on planar structures with a p of 1/2 such that interconnect scaled conveniently with computation. With interconnect area and delays becoming the truly dominating effects, these ideas may now attain even greater practical importance than when they were first introduced. Logic replication, which has long been used to minimize wires crossing chip boundaries [20] [11], may play an important role here as well.

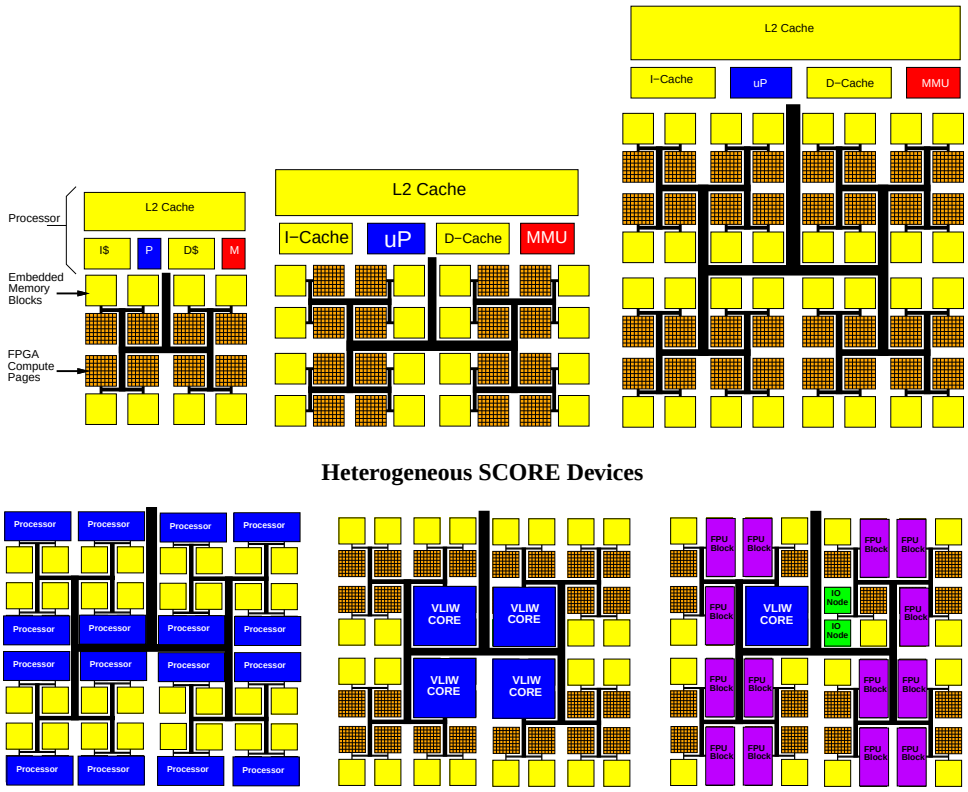


Fig. 2. Compute Model Facilitates Heterogeneous Compute Pages and Component Scaling

Space and Time A key optimization will be understanding how much to fold and unfold computations to match computational throughput rates, fit within the available area, and minimize total path delay distances. Time multiplexed designs require more area between active computing operations but allow entire computations to be compact. Sometimes it will still be desirable to make a computation compact in this manner in order to reduce the distance required to communicate to it or across it. This should be quite evident for off-critical path computations, operators in loops with large cycle bounds, and operations with low relative operating frequency. An interesting question for the future will be: when it is faster to time-multiplex even a critical path computation in order to reduce the size of the circuit and hence the distance which the signals must travel?

Spatial Locality With interconnect playing such a dominant role, we want to optimize the *spatial locality* of heavily communicating blocks. Some of this will arise out of the area and path delay minimizing optimization described above. However, we can do better in a number of important cases.

The computation may be composed of many cyclic subgraphs connected together, perhaps even nested in larger cycles of computation. The distance around each cycle will often limit the rate of computation. Consequently, it is important to place each cycle as compactly as possible. Techniques such as cycle partitioning and replication [21] may be important here. In many cases, we can cluster cycles tightly at the expense of increasing the distances between the clustered cycle and the rest of the graph; to the extent these links in and out of the cycle are not themselves on critical loops, they can be pipelined to accommodate the additional delay without adversely impacting the computation. Capturing this kind of timing freedom is an important function of the computational model.

Communication among operators will often be dynamic and data dependent. When not all communications are equally likely, we have the opportunity to preferentially cluster the blocks involved in common communications more closely than less commonly communicating blocks. In the sequential processor world, people have long exploited instruction placement to increase spatial locality and hence virtual memory [9] and cache performance [25]. Here, we need to cluster operations into spatial clusters to reduce the distance delay along the most commonly used communication paths.

7 Spatial Algorithms

The algorithms suitable for spatially organized computations may be different from the ones we have found suitable for temporal computations. The work in systolic architectures provides a number of important algorithms, such as: matrix multiplication [15], dynamic programming [8], sorting [17], and image processing. In many cases these algorithms are specifically designed to minimize and regularize interconnect. Contemporary work in FPGA computation has offered a number of additional spatial algorithms, including: sequence matching (dynamic programming) [10], satisfiability [28] and set covering [19] search, regular expression matching [22], and image processing [27] [18]. These algorithms demonstrate the power that comes from properly reformulating algorithms for spatial computation. The development and understanding of new spatial algorithms will be an important component of understanding how to exploit the rich capacities available to us. In the capacity poor past, the kinds of techniques used in spatial algorithms were almost unfathomable compared to the resources available; today and tomorrow they may be essential.

8 Conclusions

We are rapidly entering a future where the capacity of our basic computing media is more than sufficient to contain a significant fraction of our computing problems. This opens up a much larger range of implementation options for us. However, the computing models and abstractions developed during the era of limited capacity make it difficult for us to exploit the capacity now available. Spatial computing architectures and designs offer a promising alternative to temporal organization that can better exploit the rich capacity becoming available. In these spatial designs, communication is a first order concern. Consequently, it is important to develop models that expose communications

for optimization and to develop algorithms which are conscious of the costs and effects of communications in order to fully exploit the performance potential of modern and future capacity rich devices.

References

1. International Technology Roadmap for Semiconductors. <http://public.itrs.net/Files/2001ITRS/>, 2001.
2. Eylon Caspi, Michael Chu, Randy Huang, Nicholas Weaver, Joseph Yeh, John Wawrzynek, and André DeHon. Stream Computations Organized for Reconfigurable Execution (SCORE): Introduction and Tutorial. <http://www.cs.berkeley.edu/projects/brass/documents/score_tutorial.html>, short version appears in FPL'2000 (LNCS 1896), 2000.
3. André DeHon. The Density Advantage of Configurable Computing. *IEEE Computer*, 33(4):41–49, April 2000.
4. André DeHon. Compact, Multilayer Layout for Butterfly Fat-Tree. In *Proceedings of the Twelfth ACM Symposium on Parallel Algorithms and Architectures (SPAA'2000)*, pages 206–215. ACM, July 2000.
5. André DeHon. Rent's Rule Based Switching Requirements. In *Proceedings of the System-Level Interconnect Prediction Workshop (SLIP'2001)*, pages 197–204. ACM, March 2001.
6. Daniel Dobberpuhl, Richard Witek, Randy Allmon, Robert Anglin, Sharon Britton, Linda Chao, Robert Conrad, Daniel Dever, Bruce Gieseke, Gregory Hoeppner, John Kowaleski, Kathryn Kuchler, Maureen Ladd, Michael Leary, Liam Madden, Edward McLellan, Derrick Meyer, James Montanaro, Donald Priore, Vidya Rajagopalan, Sridhar Samudrala, and Sribalan Santhanam. A 200MHz 64b Dual-Issue CMOS Microprocessor. In *1992 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pages 106–107. IEEE, February 1992.
7. Seth Copen Goldstein and Mihai Budiu. NanoFabrics: Spatial Computing Using Molecular Electronics. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 178–189, June 2001.
8. L. J. Guibas, H. T. Kung, and C. D Thompson. Direct VLSI Implemenation of Combinatorial Algorithms. In *Caltech Conference on VLSI*, pages 509–525, January 1979.
9. D. J. Hatfield and J. Gerald. Program Restructuring for Virtual Memory. *IBM Systems Journal*, 10(3):168–192, 1971.
10. Dzung T. Hoang. Searching Genetic Databases on Splash 2. In Duncan A. Buell and Kenneth L. Pocek, editors, *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 185–191, Los Alamitos, California, April 1993. IEEE Computer Society, IEEE Computer Society Press.
11. L. James Hwang and Abbas El Gamal. Min-Cut Replication in Partitioned Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):96–106, January 1995.
12. Guest Editor James Meindl. Special Issue on Limits of Semiconductor Technology. *Proceedings of the IEEE*, 89(3):223–393, March 2001.
13. Gilles Kahn. The Semantics of a Simple Language for Parallel Programming. In *Proceedings of the IFIP CONGRESS 74*, pages 471–475. North-Holland Publishing Company, 1974.
14. H. T. Kung. Why Systolic Architectures? *IEEE Computer*, 15(1):37–46, January 1982.
15. H. T. Kung and Charles E. Leiserson. Systolic Arrays (for VLSI). In *Proceedings of 1978 Sparse Matrix Conference*, pages 256–282. Society for Industrial and Applied Mathematics, 1979.

16. B. S. Landman and R. L. Russo. On Pin Versus Block Relationship for Partitions of Logic Circuits. *IEEE Transactions on Computers*, 20:1469–1479, 1971.
17. Charles E. Leiserson. Systolic Priority Queues. In *Proceedings of the Conference on Very Large Scale Integration: Architecture, Design, and Fabrication*, pages 199–214. California Institute of Technology, 1979.
18. Michael R. Piacentino, Gooitzen S. van der Wal, and Michael W. Hansen. Reconfigurable Elements for a Video Pipeline Processor. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'99)*, pages 82–91. IEEE, 1999.
19. Christian Plessl and Marco Platzner. Custom Computing Machines for the Set Covering Problem. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'2002)*. IEEE, 2002.
20. Roy L. Russo. On the Tradeoff Between Logic Performance and Circuit-to-Pin Ratio for LSI. *IEEE Transactions on Computers*, 21(2):147–153, February 1972.
21. Minshine Shih and Chung-Kuan Cheng. Data Flow Partitioning for Clock Period and Latency Minimization. In *Proceedings of the 31st Design Automation Conference (DAC'31)*, June 1994.
22. Reetinder Sidhu and Viktor K. Prasanna. Fast Regular Expression Matching using FPGAs. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'2001)*. IEEE, 2001.
23. Kazumasa Suzuki, Masakazu Yamashina, Takashi Nakayama, Masanori Izumikawa, Masahiro Nomura, Hiroyuki Igura, Hideki Heiuchi, Junichi Goto, Toshiaki Inoue, Youichi Koseki, Hitoshi Abiko, Kazuhiro Okabe, Atsuki Ono, Youichi Yano, and Hachiro Yamada. A 500MHz 32b 0.4μm CMOS RISC Processor LSI. In *1994 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pages 214–215. IEEE, February 1994.
24. Dennis Sylvester and Kurt Keutzer. Rethinking Deep-Submicron Circuit Design. *IEEE Computer*, 32(11):25–33, November 1999.
25. Hiroyuki Tomiyama and Hiroto Yasuura. Code Placement Techniques for Cache Miss Rate Reduction. *ACM Transactions on Design Automation of Electronic Systems*, 2(4):410–429, October 1997.
26. William Tsu, Kip Macy, Atul Joshi, Randy Huang, Norman Walker, Tony Tung, Omid Rowhani, Varghese George, John Wawrzynek, and André DeHon. HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array. In *Proceedings of the International Symposium on Field Programmable Gate Arrays*, pages 125–134, February 1999.
27. John Villasenor, Brian Schoner, Kang-Ngee Chia, and Charles Zapata. Configurable Computer Solutions for Automatic Target Recognition. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, pages 70–79. IEEE, April 1996.
28. Peixin Zhong, Margaret Martonosi, Pranav Ashar, and Sharad Malik. Accelerating Boolean Satisfiability with Configurable Hardware. In *Proceedings of the 1998 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'98)*, pages 186–195, April 1998.

Web links for this document: <`http://www.cs.caltech.edu/research/ic/abstracts/vlssc_umc2002.html`>