Trends toward Spatial Computing Architectures

André DeHon
University of California at Berkeley, Berkeley, CA, USA

Figure 1 shows the peak computation offered per unit silicon for RISC processors and FPGAs over the past couple of decades. Advances in processor architecture have allowed us to mostly turn additional silicon area into additional performance. The time unit (seconds) is absolute, not normalized to process, so this does represent some actual sacrifice of the increasing capabilities provided by the fabrication process to the design process, more complicated architectures, and increasing memory imbalance.

Also shown in Figure 1 is the peak computational density for FPGAs. This shows at least a 10× gap in raw density between the processor architectures and the FPGAs.

What does this show and what does it mean for the design of post-fabrication, programmable computing devices? Figure 3 shows a simplified model of the architecture space in which these devices are designed.[5] The key feature highlighted here is the way processing units are controlled by *instructions*. We can vary the number of processing units which use the same instruction in SIMD fashion by varying the word width (w). We can also vary the number of different behaviors which are readily available for the programmable processing units by controlling the instruction depth (c). In a similar manner, we can vary the amount of data retimed within the computational structure by varying the data depth (d) [not shown]. Figure 2 shows the dramatic effect which these parameters can have on raw computational density.

In the mid 1980's when we had 50Mλ^2 dies, the designer had to choose between (1) instruction stores rich enough to support "large" subcomputations on the computational die and (2) larger numbers of active compute operators with finer-grain control. Primary examples of this tradeoff where the MIPS-X which offered a 32b ALU with 512 on-chip instructions [7], and Xilinx's XC2064 [2] which offered 64 4-LUTs (600-1000 programmable gates) on a chip. At this point in time, few problems had such small kernels that the entire computation could be built spatially on the FPGA device. Many important computations could fit in the 512 instruction cache for the processor. Spatial computing at this point suffered from (1) latency and bandwidth penalties for die crossings and (2) large costs in silicon area to realize any reasonably sized computation. That is, *in order to pack large computations onto the limited die space available, it was necessary to reuse the programmable processing elements heavily and store the instructions and data for the computation compactly in large, dense memories.*

In the late '90's, the growth in silicon die capacity has changed the picture. We can put 7,000 4-LUTs (100K gates) on a few Gλ^2 die. The computations and data which could only fit on a single-chip implementation by sequentially reusing a single CPU a decade earlier can be fully implemented in spatial dataflow on a single FPGA today. The advantage to these spatial implementations is the greater computational density shown in Figure 1. In the same area, the processors now hold 50-100K instruction and data words on chip and execute on a small number of 64b processing units.

Figure 4 compares the efficiency of the processor and FPGA resource allocation under the simple model above. Here, cycle length l can be taken as the critical path cycle length for a recurrence computation or simply the necessary multiplexing factor to achieve a target throughput rate. Design w is the application word width; Increasingly, we are seeing that the wide processor word widths are not well matched to application processing requirements. The comparison underscores the fact that the processors and the FPGA are at different points in the architecture space, efficient for different sets of requirements. When die space was precious, the heavy multiplexing of the processor was necessary in order to fit the problem onto the available silicon area at all. With larger dies, architectures which keep the processor distribution of memory and compute are inefficient for high throughput computational tasks.

Now that we have grown to 10Gλ^2 dies and see 1Tλ^2 dies on the horizon, both extremes (single instruction, bit-level control and deep instruction, wide-word control) in this design space make sense for different sets of application needs. Beyond that, however, is a large intermediate design space and room for hybrid designs.

By storing a few instruction locally in a traditional FPGA array, one can pack instructions more deeply onto the die without substantially decreasing computational density (*e.g.* MIT DPGA [4]). Taking this idea to an extreme, one can achieving processor-like instruction density (University of Toronto's VEGA [8]), but at a large cost in computational density. Using 16b words and shallow (8 deep) data and instructions, UCB's PADDI [3] and PADDI-2 [12] handle high-throughput, 16b DSP-oriented computations more efficiently than traditional DSP architectures.

This is a direct case of trading memory die area for greater active computing area, and the resulting architectures are very efficient for high data rate filtering applications. At the extreme of very wide word widths, MIT's Abacus [1] controls 1024 SIMD processing units with each instruction, achieving 3× the computational density of even modern FPGAs for such wide SIMD problems. MIT's MATRIX architecture [9] uses 8b RF-ALU building blocks, allowing the application to drive some of its own tradeoff between instruction, data, and computing resources; it sacrifices some of the peak density exhibited by FPGAs and SIMD arrays for more robust yield of computational power over a broader application spectrum.

Full applications tend to have a mix of processing requirements. As we see here the architectural design space is broad. As we move to 100G–1Tλ^2 system-on-a-chip dies, hybrid components which mix multiple design points onto a single IC become promising. Reviewing Figure 4, notice that the processor and the FPGA are both less than 1% efficient at their cross points, suggesting a hybrid device dedicating half the area of the chip to each would be more efficient on many applications than a homogeneous die with either architecture. Exploiting our common 90/10 rule of thumb, the small fraction of the code which accounts for most of the computing time can be handled on the spatial portion of the die, while the large remaining portion of the computation can be implemented compactly using the more traditional processor organization. This is the basic motivation behind Harvard's PRISC [10], UCB's GARP [6], and National Semiconductor's NAPA [11], designs which couple a datapath computing array for regular computational tasks along with a RISC processing core for control and less frequently executed tasks.

A decade ago when IC space was a premium, versatility and programmability virtually required a processor-like architecture with heavily multiplexed and reused datapaths. Now that IC capacities have grown 1000×, the space of alternative programmable architectures is much broader and more interesting. For many applications, more spatially oriented computing architectures will offer greater performance per unit area while retaining the benefits of post-fabrication programmability. This trend goes hand-in-hand with the greater space available and should continue to broaden as die sizes grow.

References

[1] Michael Bolotski, Thomas Simon, Carlin Vieri, Rajeevan Amirtharajah, and Thomas F. Knight Jr. Abacus: A 1024 processor 8ns simd array. In *Advanced Research in VLSI 1995*, 1995.

[2] William S. Carter, Khue Duong, Ross H. Freeman, Hung-Cheng Hsieh, Jason Y. Ja, John E. Mahoney, Luan T. Ngo, and Shelly L. Sze. A user programmable reconfigurable logic array. In *IEEE 1986 Custom Integrated Circuits Conference*, pages 233–235. IEEE, May 1986.

[3] Dev C. Chen and Jan M. Rabaey. A reconfigurable multiprocessor ic for rapid prototyping of algorithmic-specific high-speed dsp data paths. *IEEE Journal of Solid-State Circuits*, 27(12):1895–1904, December 1992.

[4] André DeHon. Dpga utilization and application. In *Proceedings of the 1996 International Symposium on Field Programmable Gate Arrays*. ACM/SIGDA, February 1996. Extended version available as Transit Note #129 `<http://www.ai.mit.edu/projects/transit/transit-notes/tn129.ps.Z>`.

[5] André DeHon. Reconfigurable architectures for general-purpose computing. AI Technical Report 1586, MIT Artificial Intelligence Laboratory, 545 Technology Sq., Cambridge, MA 02139, October 1996.

[6] John R. Hauser and John Wawrzynek. Garp: A mips processor with a reconfigurable coprocessor. In *Proceedings of the IEEE Symposium on Field-Programmable Gate Arrays for Custom Computing Machines*, pages 12–21. IEEE, IEEE, April 1997.

[7] Mark Horowitz, John Hennessy, Paul Chow, Glenn Gulak, John Acken, Anant Agarwal, Chorng-Yeung Chu, Scott McFarling, Steven Przybylski, Steven Richardson, Arturo Salz, Richard Simoni, Don Stark, Peter Steenkiste, Steven Tjiang, and Malcom Wing. A 32b microprocessor with on-chip 2k byte instruction cache. In *1987 IEEE International Solid-State Circuits Conference, Digest of Technical Papers*, pages 30–31. IEEE, February 1987.

[8] David Jones and David Lewis. A time-multiplexed fpga architecture for logic emulation. In *Proceedings of the IEEE 1995 Custom Integrated Circuits Conference*, pages 495–498. IEEE, May 1995.

[9] Ethan Mirsky and André DeHon. Matrix: A reconfigurable computing architecture with configurable instruction distribution and deployable resources. In *Proceedings of the IEEE Workshop on FPGAs for Custom Computing Machines*, April 1996.

[10] Rahul Razdan and Michael D. Smith. A high-performance microarchitecture with hardware-programmable functional units. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 172–180. IEEE Computer Society, November 1994.

[11] Charlé Rupp, Mark Landguth, Tim Garverick, Edson Gomersall, Harry Holt, Jeffrey Arnold, and Maya Gokhale. The napa adaptive processing architecture. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, pages 28–37, April 1998.

[12] Alfred K. Yeung and Jan M. Rabaey. A 2.4 gops data-drivern reconfigurable multiprocessor ic for dsp. In *Proceedings of the 1995 IEEE International Solid-State Circuits Conference*, pages 108–109. IEEE, February 1995.

Figure 1: Peak Computational Density

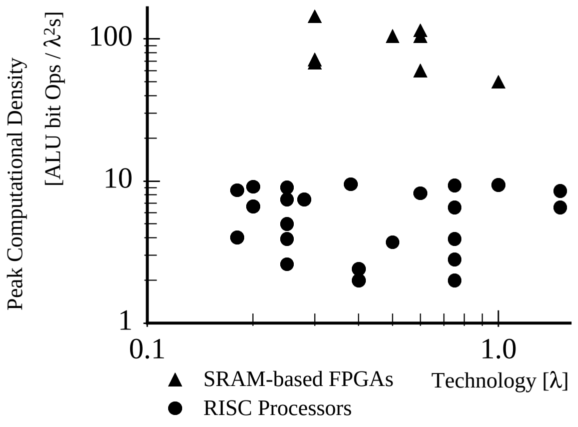


Figure 2: Relative Peak Density from Model

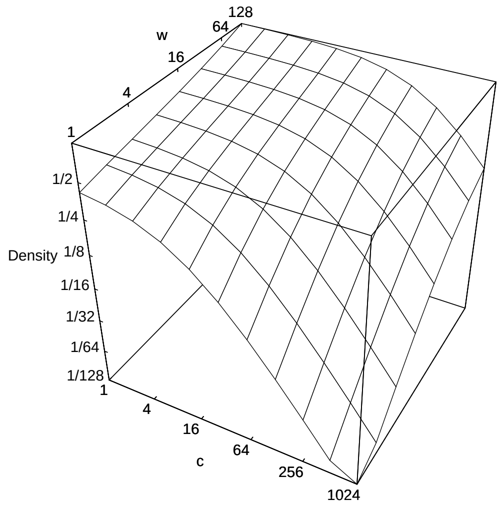


Figure 3: Model for Programmable Compute Element

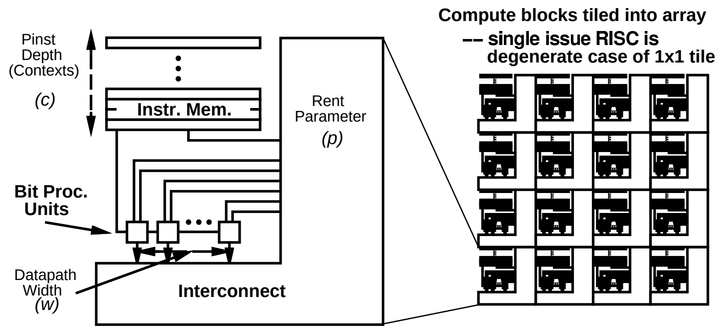
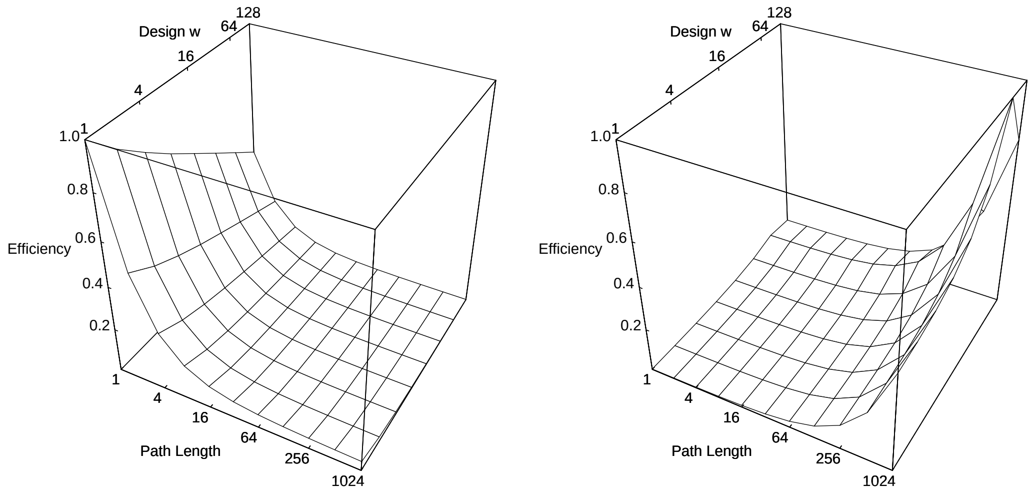


Figure 4: Processor and FPGA Area Caricatures

Figure 5: Yielded Efficiency for Processor and FPGA Caricature



$c = d = 1$, $w = 1$ $\quad$ $c = d = 1024$, $w = 64$

(at 16K Bit Processing Elements)

Figure 6: Application Comparison – FIR Throughput Density

Architecture	Reference	Feature Size (λ)	Area and Time	16b TAPs / λ^2s	8b TAPs / λ^2s
16b DSP	Kaneko ISSCC87	0.65μm	350Mλ^2, 50 ns/TAP	0.057	0.057
32b RISC/DSP	Nadehara VLSI SP95	0.25μm	1.2Gλ^2, 40 ns/TAP	0.021	0.021
64b RISC	Gronowski ISSCC96	0.18μm	6.8Gλ^2, 2.3 ns/TAP	0.064	0.064
Pentium MMX	Choudhury ISSCC97 MMX AP559	0.18μm	6.6Gλ^2, 31$\times$3.3 ns/13 TAP	0.019	0.019
FPGA	XC4K Newgard Xilinx Seminar 96	0.60μm	16b – 46 CLBs, 244 ns/8-TAPs 8b – 240 CLBs, 14.3 ns/8-TAPs CLB $\approx$1.25Mλ^2	0.57	1.9
	Altera 8K AN96	0.30μm	8b – 30 LEs$\times$0.92Mλ^2/LE, 10 ns/TAP		3.6
Reconfigurable ALU	PADDI2 Yeung ISSCC95	0.50μm	16b – 12 EXUs, 20 ns/TAP 8b – 5 EXUs, 20 ns/TAP EXU $\approx$ 10.7Mλ^2	0.39	0.93
	MATRIX Mirsky FCCM96	0.25μm	16b – 8 BFUs, 20 ns / TAP 8b – 2 BFUs, 20 ns/TAP BFU $\approx$ 29Mλ^2	0.22	0.86
Full Custom	Ruetz JSSC89	0.75μm	400Mλ^2, 45 ns/64 TAPs		3.6
	Golla JSSC90	0.60μm	140Mλ^2, 33 ns/16 TAPs		3.5
	Cai CICC90	0.75μm	235Mλ^2, 25 ns/4 TAPs	0.68	0.68
(fixed coefficient)	Laskowski CICC92	0.60μm	114Mλ^2, 6.7 ns/43 TAPs	56	56