The UCSC Kestrel Application-*Unspecific* Processor

Richard Hughey Andrea Di Blas
Department of Computer Engineering
University of California, Santa Cruz, CA 95064
{rph, andrea}@soe.ucsc.edu

Abstract

The UCSC Kestrel parallel processor is part of an evolution from application-specific to specialized to application-unspecific processing. Kestrel combines an ALU, multiplier, and local memory, with Systolic Shared Registers for seamless merging of communication and computation, and an innovative condition stack for rapid conditionals. The result has been a readily programmable and efficient co-processor for many applications. Experience with Kestrel indicates that programmable systolic processing, and its natural combination with the Single Instruction-Multiple Data (SIMD) parallel architecture, will be an effective design choice for years to come.

1. Introduction

Biological sequence analysis has long been a standard problem for application-specific processing. The underlying algorithms are simple and regular, and the amount of data for analysis is prodigious. University and commercial projects have tackled this problem since the early days of VLSI, and many of these approaches are chronicled within the proceedings of ASAP and its predecessor conferences.

The first such machine was Lopresti's 1986 Princeton Nucleic Acid Comparator (P-NAC) [19, 20], a single-purpose systolic array that provided great speedup on its algorithm, and set the stage for the many application-specific VLSI and FPGA machines that followed.

The second wave included the use of single-chip (MICSMACS) or single-board (Warp) processors to create short systolic arrays designed for signal processing but suitable for sequence analysis and other applications [6, 1]. This period also introduced three more specialized machines with moderate to extreme numbers of processing elements (PEs) on each chip. BISP (16 PEs/chip) and BioSCAN (892 PEs/chip) were application-specific, each suited to one specific algorithm [2, 25]. The successor to P-NAC and predecessor to Kestrel, the Brown Systolic Array (B-SYS, 47 PEs/chip), incorporated full programmability as an *application specialized* processor [14, 15, 10].

Flexibility and performance continued to increase with new machines from research projects and industry. The introduction of FPGA-based computing [29, 7] led to their use as sequence analysis engines [7, 3, 28]. More general custom VLSI solutions also appeared [27], and the MasPar mini-supercomputer [21] became a common tool for bioinformatics research. The UCSC Kestrel processor moved from the domain of application-specialized to application-unspecific, while preserving many features of the B-SYS SIMD programming model and high computational density. As a result, it was possible to apply Kestrel to a far broader selection of algorithms than biological sequence analysis [11, 12, 9, 23, 5, 4].

In this paper, we discuss the architectural and programming features that lead to Kestrel's efficiency, moving from biological sequence analysis, to B-SYS, and then to Kestrel. Many of the features that made Kestrel successful reflect the values of good design: simplicity, regularity, and attention to detail. We conclude with an evaluation of the place of application-*unspecific* SIMD processing within the context of FPGA-based computation, multi-cores, graphics cards, and application-specific processing.

2. Biological Sequence Analysis

A core problem of bioinformatics is determining the relationship between molecules such as DNA, RNA, and proteins. Ideally, the biologist wishes to discover the physical relationship between the molecules (do they fold up the same way? are important residues conserved?), but often must rely on just the information relationship.

One of the best ways to discover the information relationship between two sequences A and B is by using one of a family of related dynamic programming algorithms for pairwise sequence comparison. For two sequences, the correspondence is determined with a three-state model. The cost $c_{i,j}$ of aligning $a_0 \ldots a_i$ to $b_0 \ldots b_j$ is calculated by taking the minimum of three alternatives: (1) the two characters correspond to each other, in which case a match cost is added to $c_{i-1,j-1}$; (2) the b_j character is not present in A,

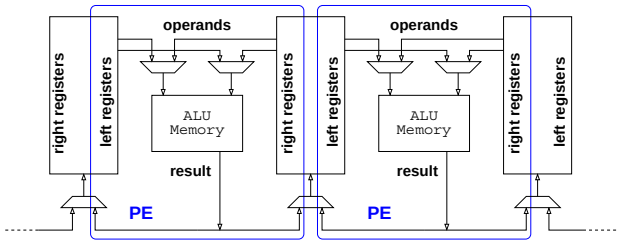

Figure 1. Systolic Shared Registers

in which case a deletion cost is added to $c_{i,j-1}$; or (3) the a_i character is not present in B, in which case an insertion cost is added $c_{i-1,j}$:

$$c_{i,j} = \min \begin{cases} c_{i-1,j-1} + cost(\text{match/mutate}) \\ c_{i,j-1} + cost(\text{delete}) \\ c_{i-1,j} + cost(\text{insert}). \end{cases}$$

In practice, biologists use a more complicated form called the Smith-Waterman algorithm [26], which involves three interleaved recurrence equations as well as gap initiation and continuation costs. For the most exacting searches, bioinformatics practitioners use profile hidden Markov models (HMMs) [18], which enable comparison and alignment of a single sequence to a family of sequences. HMMs also use three interleaved recurrences, but with higher precision and more complicated arithmetic.

These $O(n^2)$ algorithms have many mappings to linear arrays. For programmable machines, one of the best mappings is to preload one sequence (or the HMM) into the array, and then move the second sequence through the array from left to right. The dynamic programming matrix is thus calculated along diagonals, with $c_{i,j}$ calculated at step $i+j$ in PE_i using values just calculated in PE_i and PE_{i-1}.

Because of the vast size of genomic and protein databases, many universities and companies have worked to accelerate the sequence comparison algorithms [11]. P-NAC, the first application-specific solution, placed 30 PEs on a chip, and implemented the above equation over a four-character (DNA or RNA) alphabet, with an insert or delete cost of 1 [19, 20]. This simplicity enabled fast dynamic programming, critical for analyzing large sequence databases, but the lack of support for Smith-Waterman and other algorithms reduced the utility for biologists.

3. The Brown Systolic Array

The goal of the Brown Systolic Array (B-SYS) was to develop a successor co-processor to P-NAC. Maintaining simplicity similar to that of P-NAC was critical to ensure that many PEs could be placed on a single die, enabling large arrays to be created with only a handful of chips. It was also important to introduce programmability into the system. The family of sequence comparison algorithms is broad, and all or most of the many variations can only be accelerated on a programmable machine.

The natural choice for B-SYS was a linear systolic array, the most efficient architecture for sequence analysis. The linear array can be placed in low pin-count packages and is tremendously scalable. Of course, the cost of that scalability is ever increasing latencies from one end of the array to the other. As the Kestrel project was to demonstrate, the linear array is far more flexible as a computation engine than might be generally thought.

B-SYS introduced the concept of Systolic Shared Registers (SSRs, Figure 1), in which register banks are shared between adjacent PEs. The SSRs enable seamless integration of computation and communication, as a single instruction can specify that values should be read from the left register bank, computed upon, and written to the right register bank. On a SIMD machine, there will be no bank conflicts because every PE is reading the same operand registers and writing the same destination register. The cost of SSRs is one instruction bit per operand and one extra register bank per chip to ensure all operands are on chip.

B-SYS (Figure 2) consisted of 10 custom chips, each with 47 processing elements implemented with 85,000 transistors [15]. Each processing element had a flexible 8-bit ALU, 1 mask flag, 7 general-purpose flags, and sixteen 8-bit registers. A 38-bit instruction is broadcast each clock cycle, though an earlier design proposed systolic instructions [14]. Due to the lack of an on-board instruction memory and controller, the test system only ran at 250 kHz on a 25 MHz host, although the chips could run 10 times faster.

The SSRs enabled implementation of P-NAC's inner loop in 6 instructions (6 clock cycles) by, for example, shifting a character from left to right at the same time as comparing it to the stored character. As sequence analysis performance corresponds directly to the length of the inner loop, this was a particularly notable achievement. Even at 250 kHz, the single-board, 470-PE B-SYS system performed sequence comparison at about the same speed as the much larger 16 K PE Thinking Machines CM-2 [15]. At the time, somewhat faster solutions included the first FPGA-based sequence comparison solution on the Splash machine with 32 Xilinx chips [7], and two single-purpose VLSI systems on much larger chips [25, 2].

B-SYS included a unified software environment with both assembly language and a higher-level language, the New Systolic Language (NSL). NSL used a stream programming model for systolic algorithms, joining the I/O characteristics of a stream, such as a sequence, with the parallel stream variables. The language overloaded C++ operators for parallel computation, performed rudimentary code optimization, and managed file I/O between the host and array or simulator [10].

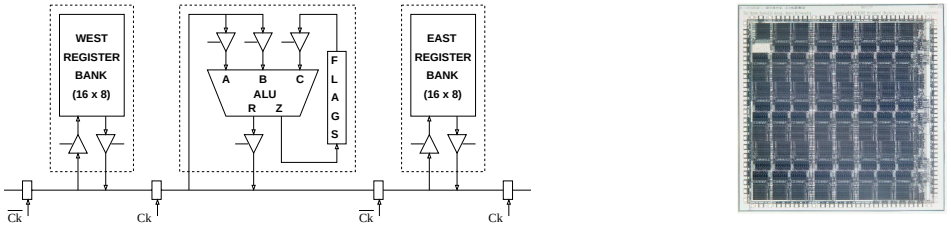

Figure 2. The B-SYS ALU with two flanking Systolic Shared Registers, and the B-SYS chip.

The B-SYS experience provided several lessons:

- Design the full system, not just the chips. The instruction and data bottlenecks between the host machine cost a factor of 10 in performance.
- Know your algorithms. B-SYS was designed for DNA sequence comparison with its four-character alphabet. Protein sequence comparison involves a 20-character alphabet, and requires lookup tables in each PE. Although B-SYS' programmability enabled the clustering of PEs to implement protein analysis, the clustering had considerable overhead, retarding performance.
- Build *application-specialized* processors rather than application-specific. In addition to one dozen sequence comparison variations (with inner loops of 6 to 54 instructions), B-SYS was also quickly programmed for classic algorithms like sorting and Horner's method, and other problems as well.
- Use simple architectures to enable dense implementations. The great advantage of the SIMD architecture is the lack of local instruction sequencing or decoding. With common register addressing and ALU control, instructions can be latched and decoded once per physical row of PEs within the chip.
- Include more memory per processing element. A recurring theme of first-generation system design. The SSRs were just 40% of B-SYS PE area, with the ALU, flags, and (minimal) local control using the rest. This is an inefficient ratio when grouping is used to expand memory per logical PE.

4. The UCSC Kestrel Parallel Processor

Kestrel grew from three converging experiences: B-SYS, the growing bioinformatics efforts at UCSC, and experience with a MasPar parallel computer [9, 4, 11].

In the 1990s, UCSC pioneered the use of HMMs for sequence analysis [18]. The technique is now standard throughout bioinformatics, and is a core component of multiple alignment and protein structure prediction algorithms [17]. While scoring against a database of HMMs is similar to performing Smith-Waterman (though slower due to the more complicated algorithm), creating an HMM is a time-consuming iterative process, adding another factor of 50 to

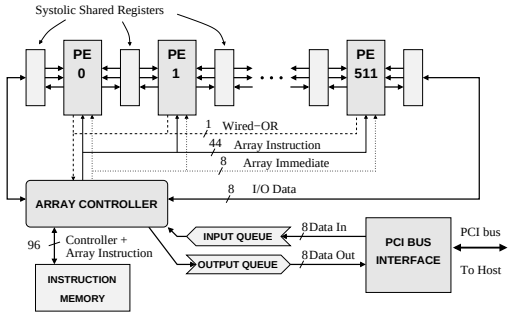

Figure 3. Kestrel board architecture

100. For these reasons, we implemented the HMM algorithms on a MasPar parallel computer [13].

The MasPar MP-2 was a 32-bit SIMD machine with local memory addressing, a mesh connection, and a global router [21]. Instructions were microcoded, with basic operations requiring 4–40, or more, cycles. The system included 32 PEs per chip, and 1K PEs per (large) board, in configurations up to 16K PEs. While the MasPar worked well as a somewhat specialized supercomputer, it was much larger and more complicated than necessary, and in some ways inefficient, for the family of sequence analysis algorithms.

Kestrel grew out of a desire to support bioinformatics algorithms impossible on B-SYS (with its lack of local memory) and more efficiently than the MasPar.

Kestrel maintains the linear SSR architecture and the 8-bit word. We determined that 8 bits continued to be an appropriate mix of flexibility for performing 8, 16, 24, and 32-bit sequence analysis problems, and bitwise parallelism.

Kestrel adds several distinctive features to the B-SYS linear architecture with shared systolic registers (Figure 4):

- Multiplier. Multiplication was required for implementation of the HMM algorithms. The multiplier was key to making Kestrel a general-purpose systolic array, and its multiply-accumulate-accumulate operation more than doubled the speed of 32-bit multiplies.
- Local memory. In addition to the now 32 SSRs, each PE has 256 bytes of locally-addressed memory. The local addressing is critical for protein and HMM sequence analysis, and of course has many other uses.
- Condition Stack. One of the most innovative aspects of

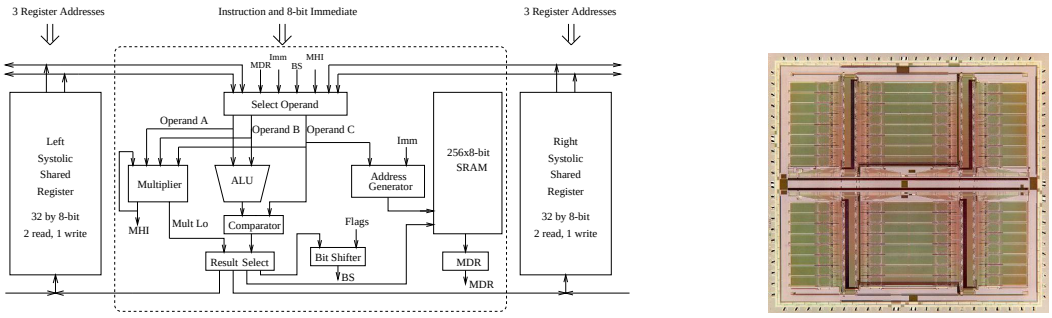

Figure 4. Kestrel processing element architecture and the Kestrel chip.

Kestrel is each PE's eight-bit condition stack. General purpose SIMD processing frequently requires masking PEs while instructions for a subgroup are broadcast. The condition stack enables processing of "if…else" clauses and similar constructs concurrently with other instructions. This enables Kestrel to have the most rapid PE activity switching of, to our knowledge, any SIMD array ever designed.

- ALU/Comparator. Geared toward the sequence analysis applications, Kestrel includes an integrated ALU and comparator, able to perform single-cycle addition and minimization. This feature motivated Kestrel's three-operand design. While important for Smith-Waterman performance, this feature is not a requirement for general-purpose systolic computing.
- Multiprecision operation. Special care was taken in the ALU, comparator, and multiplier to ensure efficient multi-byte operation.

The system architecture includes on-board instruction memory, an FPGA-based instruction sequencer, data input and output queues, and clock circuitry. Every 96-bit Kestrel instruction includes 54 bits of instruction for the Kestrel PEs, and 42 bits for the instruction sequencer.

In a parallel processor, it is critical to keep the PE array busy performing useful calculations. Kestrel achieved this goal at several levels. The orthogonal instruction set enables every broadcast instruction to access the memory with indirect addressing, modify the condition stack, perform an ALU/Comparator or Multiplier operation, and, via the SSRs, move data through the array. The single-clock execution of an instruction is well-balanced to the broadcast time of the instruction and decoded register addresses. At the same time, the Kestrel controller may be passing data back and forth to the array, evaluating loop conditions based on a counter or wired-or, or branching.

These many levels of parallelism enable the core Smith-Waterman function on 24-bit numbers to be performed in only 17 instructions. Compared to a 500 MHz UltraSparc-II, Kestrel achieved speedups of 100 for Smith-Waterman, 44 for Viterbi (add/min) HMM scoring, and 8 for Forward (mult/add) HMM scoring [4].

Kestrel also does very well against general-purpose, special-purpose, and FPGA processors. It is difficult to compare machines across technologies and budgets. Newer technologies boast higher clock speeds, and larger budgets lead to larger chips and systems. One way of comparing sequence analysis performance is in terms of performance per transistor [4]. Although this does not correct for clock speed increases between CMOS generations, it does correct for scaling. Kestrel's 1997 0.5μm chips at 20 MHz, half their achievable speed, are 360 times more efficient on protein sequence analysis than the 6-year-earlier MasPar MP-2 (1992, 1.0μm at 12.5 MHz). Kestrel's 20 MHz efficiency is also better than 3 machines from the next two CMOS generations. These include a very large general-purpose VLSI processor (Fuzion 150, 2000, 0.25 μm, 200 MHz [24]), a commercial FPGA sequence analysis machine (DeCypher, 2001, 0.18 μm? [28]), and a commercial sequence analysis ASIC (GeneMatcher2, 2001, 0.13μm? 192 MHz, [22]) [4].

The Kestrel project provided several lessons, some of which echo those learned with the earlier system:

- Spend as much time designing the full system as the chips. The Kestrel 1 controller, now on board with the instruction memory, was quickly prototyped as a single-cycle sequencer. Unfortunately, we did not originally design a pipelined board and controller, which would have enabled the Kestrel chips to function at 40 MHz, twice as fast as the Kestrel 1 system.
- Create new algorithms. One of the most amazing aspects of the Kestrel project has been the variety of algorithms, many not obviously parallelizable on a linear array. While high Kestrel performance was expected on sequence analysis, Kestrel's successful application to graph problems, asynchronous algorithms, floating-point arithmetic, machine learning, and conformation analysis was a surprise.
- Compilers are important but hard to do well. For several years, we worked to create a full-fledged compiler for Kestrel. The limited local memory, reducing the ability to spill and retrieve registers, made the implementation difficult. Although able to generate code, we never used it to program the array.

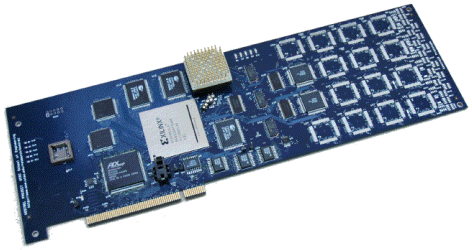

Figure 5. The Kestrel 2 board.

- Include more memory. A recurring theme of second-generation system design as well. In Kestrel's case, the primary memory issues are the need for on-board memory and a potential increase in the local memory. Unlike B-SYS, however, conditional execution and processor grouping is sufficiently efficient so that this is not a major problem.

5. Application-*Unspecific* Processing

The most important feature of Kestrel is its flexibility. We have programmed Kestrel for computational chemistry, protein conformation analysis, neural networks, floating-point arithmetic, high-speed division, and a variety of image processing algorithms [4, 23]. We also developed a paradigm of phased programming for implementing asynchronous algorithms on SIMD arrays [5], and applied these techniques to graph and neural network problems.

The computational chemistry application was particularly interesting for its use of PE grouping. Blocks of three PEs perform subproblems (analyzing a specific conformation of a small molecule). For every eight blocks, we allocated an additional eight PEs for storing local results, so that values could be accumulated within the array close to their calculation rather than being shifted out of the array after each completed calculation. Similarly, because of the lack of data memory on the Kestrel board, another 196 of the 512 PEs were used as staging memory.

Most recently, we created a Kestrel application for the object recognition and tracking of red blood cells in microscope images [16]. The application also enhances the video resolution to be able to accurately measure the cells' parameters needed for the experiment — area, perimeter, and sphericity. Finally, the goal was to monitor each cell's parameters over time to study the cells' reactivity. This was done by tracking all cells across frames in a video stream. Our implementation has a C front-end interfacing with Kestrel for the most data-intensive processing, such as bilinear interpolation and bidimensional convolution. The speedup over the original Matlab-based implementation brought processing time down from several hours to a few tens of seconds for a typical video, as was expected. However, it was still far from real-time processing. We have shown that with our next processor generation — Kestrel 2 (Figure 5), currently under development — the speedup would enable almost-real-time processing.

6. Conclusions

Are programmable systolic arrays a technology that has come and gone? At the time of B-SYS, software-programmable logic was just starting to become viable for computing. B-SYS provided a much more readily programmable environment, and a much higher density of computation. At that time a revolution in computing machines was happening: the availability of high-scale integration that enabled systolic arrays was making possible their extension to more flexible architectures. The merging of SIMD parallel architecture with the systolic structures resulted in machines like Kestrel.

Not exactly as power- and area-efficient as systolic arrays or ASICs, they were — and still are — flexible enough to cover *a range* of applications. While the optimal mapping of general problems on SIMD is still unsolved, we have seen that it is not difficult to map highly parallel problems in a way that makes these architectures competitive.

With a single control unit, almost all the power and area are used for datapath and memory. In this sense, they are therefore "optimal". For problems in which the computation flow is data-independent or with small data-dependent conditional branches, these architectures are orders of magnitude faster and more efficient than multi-core or multi-CPU systems. Even when compared with an emerging parallel coprocessor, the video card used for general-purpose processing (GPGPU) [8], SIMD machines still come up winners in many cases. The graphics pipeline produces a tremendous amount of horsepower, but it is much less flexible than a fully-programmable SIMD parallel processor. Moreover, as of now, the programmer has very little control over the actual degree of parallelism in a video card.

Hardware-programmable logic devices (FPGAs) are interesting in that one supposedly achieves ASIC-like performance with CPU-like programmability. However, FPGAs require knowledge of hardware design, and of FPGA-specific design techniques to achieve performance. For the same application, an FPGA's computational power and density is about one order of magnitude lower than that of an equivalent ASIC, while its power consumption is an order of magnitude higher. On several emerging applications that are highly demanding in terms of computation and highly parallel (e.g. image processing, video processing, video compression/decompression, etc.), SIMD machines can achieve ASIC-like performance and efficiency, and still be employable in a broad range of applications.

The experience with Kestrel, and especially the comparison of its performance with technologies several genera-

tions later, show the tremendous efficiency of a compact and tightly-designed architecture.

References

[1] M. Annaratone et al. The Warp computer: Architecture, implementation and performance. *IEEE Trans. Comput.*, 36(12):1523–1537, Dec. 1987.

[2] E. Chow, T. Hunkapiller, J. Peterson, and M. S. Waterman. Biological information signal processor. In M. Valero et al., editors, *Proc. Int. Conf. ASAP*, pages 144–160, Los Alamitos, CA, Sept. 1991. IEEE CS.

[3] Compugen Ltd. Biocellerator information package. Obtained from `compugen@datasrv.co.il`, 1994.

[4] A. Di Blas, D. Dahle, M. Diekhans, L. Grate, J. Hirschberg, K. Karplus, H. Keller, M. Kendrick, F. Mesa-Martinez, D. Pease, E. Rice, A. Schultz, D. Speck, and R. Hughey. The UCSC Kestrel parallel processor. *IEEE Trans Parallel and Distributed Systems*, 16(1):80–92, Jan. 2005.

[5] A. Di Blas and R. Hughey. Explicit SIMD programming for asynchronous applications. In E. E. Swartzlander et al., editors, *Proc. Int. Conf. ASAP*, pages 258–267, Los Alamitos, CA, July 2000. IEEE CS.

[6] P. Frison, D. Lavenier, H. Leverge, and P. Quinton. MICSMACS: A VLSI programmable systolic architecture. In J. McCanny, J. McWhirter, and J. Earl Swartzlander, editors, *Systolic Array Processors*, pages 146–154. Prentice-Hall, Englewood Cliffs, NJ, 1989.

[7] M. Gokhale, W. Holmes, A. Kopser, S. Lucas, R. Minnich, D. Sweely, and D. Lopresti. Building and using a highly parallel programmable logic array. *Computer*, 24(1):81–89, Jan. 1991.

[8] GPGPU. General-purpose computation using graphics hardware. http://www.gpgpu.org.

[9] J. D. Hirschberg, R. Hughey, K. Karplus, and D. Speck. Kestrel: A programmable array for sequence analysis. In J. Fortes et al., editors, *Proc. Int. Conf. ASAP*, pages 25–34, Los Alamitos, CA, July 1996. IEEE CS.

[10] R. Hughey. Programming systolic arrays. In E. Lee and T. Meng, editors, *Proc. Int. Conf. ASAP*, pages 604–618, Los Alamitos, CA, Aug. 1992. IEEE CS.

[11] R. Hughey. Parallel sequence comparison and alignment. In P. Capello et al., editors, *Proc. Int. Conf. ASAP*, pages 137–140, Los Alamitos, CA, July 1995. IEEE CS.

[12] R. Hughey. Parallel sequence comparison and alignment. *CABIOS*, 12(6):473–479, 1996.

[13] R. Hughey and A. Krogh. Hidden Markov models for sequence analysis: Extension and analysis of the basic method. *CABIOS*, 12(2):95–107, 1996.

[14] R. Hughey and D. P. Lopresti. Architecure of a programmable systolic array. In K. Bromley, S. Y. Kung, and E. Swartzlander, editors, *Proc. First Int. Conf. Systolic Arrays*, pages 41–50. IEEE CS, May 1988.

[15] R. Hughey and D. P. Lopresti. B-SYS: A 470-processor programmable systolic array. In C. Wu, editor, *Proc. Int. Conf. Parallel Processing*, volume 1, pages 580–583, Boca Raton, FL, Aug. 1991. CRC Press.

[16] C. Ionescu-Zanetti, L. Wang, D. D. Carlo, P. Hung, A. Di Blas, R. Hughey, and L. P. Lee. Alkaline hemolysis fragility is dependent on cell shape: Results from a morphology tracker. *Cytometry Part A*, 65A(2):116–123, April 2005.

[17] K. Karplus, R. Karchin, J. Draper, J. Casper, Y. Mandel-Gutfreund, M. Diekhans, and R. Hughey. Combining local-structure, fold-recognition, and new-fold methods for protein structure prediction. *Proteins: Structure, Function, and Genetics*, 53(S6):491–496, 2003.

[18] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler. Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.*, 235:1501–1531, Feb. 1994.

[19] R. J. Lipton and D. Lopresti. A systolic array for rapid string comparison. In H. Fuchs, editor, *1986 Chapel Hill Conference on VLSI*, pages 363–376. Computer Science Press, Rockville, MD, 1985.

[20] D. P. Lopresti. P-NAC: A systolic array for comparing nucleic acid sequences. *Computer*, 20(7):98–99, July 1987.

[21] J. R. Nickolls. The design of the Maspar MP-1: A cost effective massively parallel computer. In *Proc. COMPCON Spring 1990*, pages 25–28, Los Alamitos, CA, Feb. 1990. IEEE Computer Society Press.

[22] Paracel, Inc. Genematcher2 product literature. http://www.paracel.com, 2001.

[23] E. Rice and R. Hughey. Multiprecision division on an 8-bit processor. In T. Lang, J.-M. Muller, and N. Takagi, editors, *Proc. 13th IEEE Symp. Computer Arithmetic*, pages 74–81. IEEE CS, July 1997.

[24] B. Schmidt, H. Schroder, and M. Schimmler. Massively parallel solutions for molecular sequence analysis. In *International Parallel and Distributed Processing Symposium*, pages 186–192. IEEE, Apr. 2002.

[25] R. K. Singh, D. L. Hoffman, S. G. Tell, and C. T. White. BioSCAN: a network sharable computational resource for searching biosequence databases. *CABIOS*, 12(3):191–196, 1996.

[26] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[27] T. A. Thanaraj and T. Flores. Assessment of Smith-Waterman sequence search tools implemented in Biocellerator, FDF, and MasPar. Technical report, European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge, Feb. 1997. http://www.ebi.ac.uk/industry/Documents/publications/report.pdf.

[28] Time Logic Inc. Decypher II product literature. http://www.timelogic.com, 2002.

[29] J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H. H. Touati, and P. Boucard. Programmable active memories: reconfigurable systems come of age. *IEEE Trans. VLSI Systems*, 4(1):56–69, 1996.