The Kestrel Parallel Processor

Eric Rice David Dahle Andrea Di Blas Leslie Grate Jeffrey Hirschberg
Kevin Karplus Hansjörg Keller David Pease Don Speck Richard Hughey

Computer Engineering, Baskin School of Engineering
University of California, Santa Cruz, CA 95064

Abstract

Kestrel is a 512-element linear parallel processor with 8-bit, single-instruction stream, multiple-data stream (SIMD) processing elements. The single-board system was designed and built at the University of California at Santa Cruz with particular emphasis on efficient high-throughput DNA and protein sequence analysis. It was also designed to be general purpose, with each processing element containing local memory, a multiplier, and support for conditional and multiprecision operation.

This paper briefly describes the Kestrel architecture and describes its performance on several applications including sequence analysis, computational chemistry, and neural networks. Kestrel's performance on a targeted sequence analysis algorithm is compared with the leading commercial platforms, with Kestrel achieving the highest performance per transistor. These results have particular significance to the increasingly practical SIMD system-on-a-chip, where area is a critical constraint. Such a single-chip design would not only be inexpensive but would also be able to take full advantage of faster clock rates and could implement an efficient mesh architecture.

Index Terms—Parallel processing, SIMD, systolic array, sequence analysis, computational chemistry, VLSI system design.

1 Introduction

This paper describes and evaluates the Kestrel parallel processor, a single-board co-processor based on a 512-element linear array of 8-bit, single-instruction stream, multiple-data stream (SIMD) processing elements [1, 2, 3]. The system was developed and built at the University of California at Santa Cruz, where work on the Human Genome Project and other bioinfomatics applications motivated development of a sequence analysis engine that could efficiently analyze databases containing billions of characters from DNA, RNA, or proteins.

There were three goals of the Kestrel project. The first and primary goal was to develop a platform for efficient biological sequence analysis. In particular, Kestrel was designed to efficiently implement a class of sequence alignment algorithms (including Smith-Waterman [4, 5, 6] and Hidden Markhov Models [7, 8, 9]) that produce particularly sensitive results but whose O(n^2) complexity makes use of a serial machine impractical.

The second goal was to provide a programmable architecture that would be efficient for other parallel applications. While some flexibility was required to accommodate multiple algorithms associated with sequence analysis, effort was made to expand Kestrel's capabilities further while not seriously compromising the performance of the targeted applications. Most important in this regard is inclusion of a multiplier in each processing element. The resulting architecture has been shown to be effective for a number of diverse applications, including computational chemistry and neural networks.

The third goal was to make Kestrel accessible by making it a low-cost, single-board system. A high production 512-1024 element Kestrel system (without the overhead of a prototype chip run) could be sold for a few thousand dollars. This is a fraction of the cost of alternative systems with comparable performance.

This paper is organized as follows. Section 2 describes the Kestrel architecture at the board and PE levels, and takes a brief look at the programming environment. Section 3 provides performance details of the prototype Kestrel board (as well as some proposed improvements and future plans) and describes several applications that have been implemented to date. Section 4 concludes with a discussion of some of the relative strengths and weakness of different strategies available for fine-grained parallel processing.

2 Kestrel Architecture

Many of Kestrel's design decisions were aimed at maximizing the number of PEs on each chip while still meeting the needs of the targeted sequence analysis algorithms. This motivated the linear structure of the PE array and its 8-bit datapath (Figure 1). The 8-bit datapath was chosen in part because of the flexibility it provides and because for high throughput problems there is often little or no penalty even when higher precision is needed. For example, if 16-bit precision is required, calculations without multiplication on an 8-bit processor will take up to twice as many cycles but twice as many PEs will fit per chip (assuming SRAM size is doubled in the 16-bit architecture). Because a shorter cycle time is possible with the 8-bit datapath, it is often more efficient in such cases. As a result of the simple topology, 64 Kestrel processing elements fit in a 7.2mm by 8.3mm, 84-pin, 1.4 million transistor, 0.5 μm CMOS chip, forming the building block for the PCI co-processor.

An important high-level timing constraint during Kestrel's design was balancing the computational requirements of targeted sequence analysis algorithms (especially Smith-Waterman) with the sustained transfer rates of current disk systems. Because it was possible to perform the necessary calculations in multiple programmable cycles and still meet I/O requirements, it was possible to design Kestrel for flexibility with little penalty over single-purpose disk-based systems. This flexibility was increased by adding features not needed by the sequence analysis algorithms and by employing horizontal microcode to facilitate parallel operation of the PE subunits.

2.1 Overview of Kestrel Operation

The Kestrel board is designed to fit inside any PCI-based commodity PC or Alpha workstation with WindowsNT, OSF, or Linux operating systems. A controller on the board provides an interface between the host machine and the PE array, performing instruction issue, program control, and I/O control.

The 512 PEs are linearly connected, each communicating with its left and right neighbors through shared registers located between adjacent PEs. Data can be fed into or collected from either end of the array, or broadcast to all processing elements through an immediate field in the instruction. Each PE can perform most standard arithmetic/logic functions, using any combination of left or right registers as operands. Each PE contains 256 bytes of local SRAM, an 8×8 multiplier, an integrated ALU/comparator, and support for efficient multiprecision arithmetic. (Division must be implemented in software.) All instructions complete in a single cycle. Although the same instruction is broadcast to all PEs in the array at the same time, conditional operation is possible by masking instruction execution in selected PEs.

2.2 Kestrel Board

The main components of the Kestrel PCI board are the 512-PE array, a controller, instruction memory, input and output queues, and a PCI interface chip (Figure 1). The PE array is made up of 8 Kestrel chips

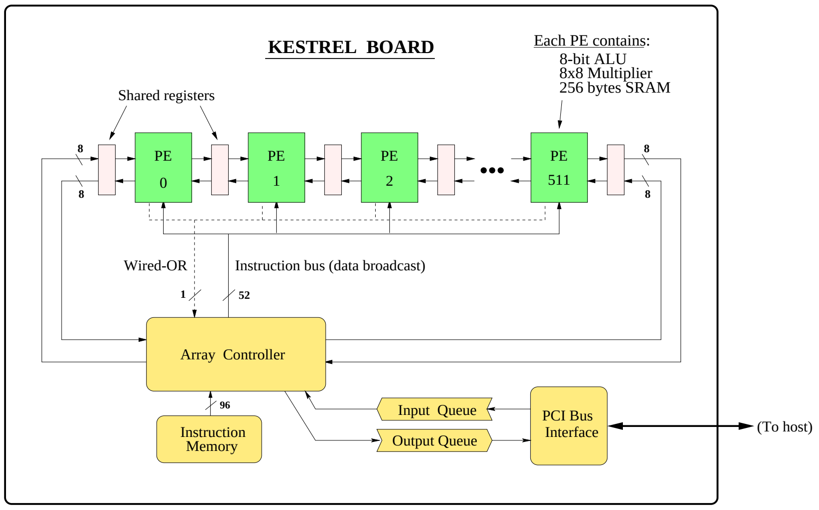

Figure 1: The Kestrel system high-level structure.

containing 64 PEs each. Kestrel's horizontal microcode instructions contain 96 bits each, 52 bits of which are used by the PEs and 44 of which are used by the controller.

The controller, which is implemented using an FPGA, coordinates board activity. It issues an instruction in each cycle and implements no-overhead loops using a stack that stores up to 15 nested loop counts. It also implements unconditional jumps as well as jumps based on conditions in the PE array, based on the array's single bit wired-OR. The controller manages data flow between the array and queues and can recirculate data from one end of the array to the other, or from one end to all PEs by inserting it into an instruction's immediate field. When a critical condition is reached by the system, the controller generates an interrupt to inform the host about the board's condition. (Most such interrupts are generated when an I/O queue needs servicing.)

2.3 Processing Elements

The heart of the Kestrel system is a 64 processing element custom VLSI chip [1, 2, 10]. Each PE contains an integrated ALU/comparator, a multiplier, a bit shifter, a shared 32-byte register, and 256 bytes of static random access memory (Figure 2). To provide maximum flexibility, these components are designed for independent operation when possible, facilitated by the horizontally microcoded instructions. The datapath supports both signed and unsigned operation, and all instructions complete in a single cycle.

The following sections describe the major PE components.

2.3.1 Systolic Shared Registers

The Systolic Shared Registers (SSRs) are located between every adjacent pair of PEs and provide the means for communication through the array [11]. Each SSR contains 32 8-bit registers with two read ports and one write port. (These are labeled L0-L31 and R0-R31 from the point of view of each PE.) Source and destination registers can be specified in any combination from a PE's right and left SSRs, allowing no-overhead communication when partial results need to be shifted through the PE array (for example by reading from the left SSRs and writing to the right SSR). Such communication is common in many systolic algorithms, including the dynamic programming algorithms used in sequence analysis.

To reduce pin count and speed up register reads at chip boundaries, each 64-PE chip contains 65 SSRs.

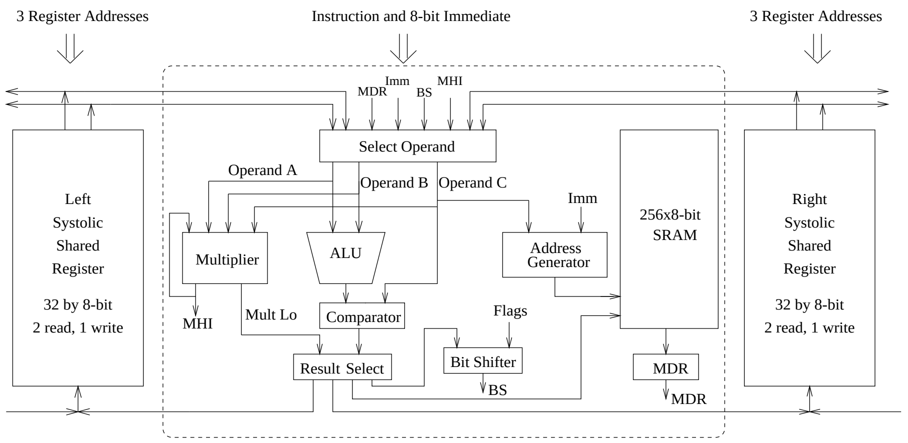

Figure 2: A Kestrel processing element with systolic shared registers.

When two or more chips are combined in a larger array, adjacent SSRs between neighboring chips are coherently linked. Thus, for example, when a write occurs to the left the value is written both to the leftmost SSR of a chip and sent off-chip so that it can be stored in the rightmost SSR of the adjacent chip. SSRs represent 8% of PE area.

2.3.2 ALU/Comparator

The ALU inputs two 8-bit operands and a carry-in bit, and outputs an 8-bit result and a carry-out bit that can be latched for multiprecision operation. The ALU is capable of all common logic functions as well as addition and subtraction. It is implemented using a unique programmable array of bit-slices controlled by a 5-bit instruction field [10].

The comparator compares the ALU result with another operand and in the same cycle can select either the minimum or maximum value. This comparison can be based on signed, unsigned, or modulo 256 comparison. Alternately, the comparator can be used to select between values based on any of several flags, and can perform efficient top-down n-byte comparisons in n cycles.

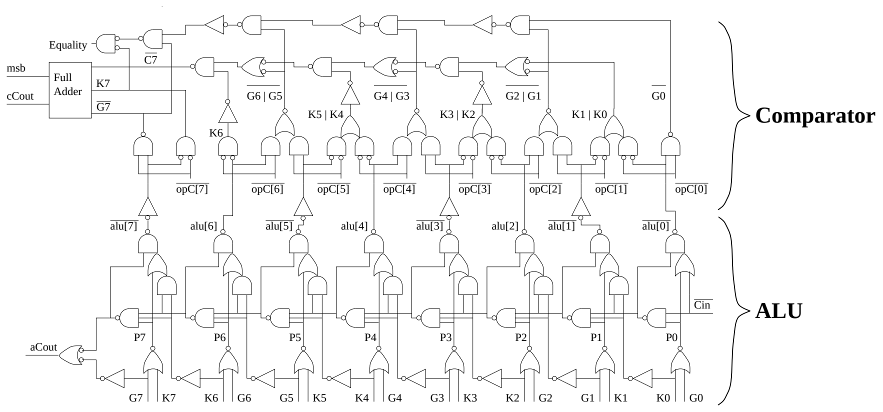

Figure 3: Logic diagram of the ALU/comparator.

The ALU and comparator hardware are tightly integrated for minimum latency. As the ALU produces each bit of output, that bit is immediately used by the borrow chain of the comparator, allowing the two units to propagate together through most of the cycle (Figure 3). As a result, the comparator adds just 15% to the latency of the adder alone and their combined operation does not limit cycle time. The ALU and comparator each represent 4% of PE area.

2.3.3 Multiplier

The Kestrel multiplier multiplies two 8-bit operands and conditionally adds one or two other operands to produce a 16-bit result. The high-byte result is stored in a special register (multHi), and the low-byte result is stored as specified by the instruction.

The multiply-accumulate-accumulate operation [12] is possible in part because of Kestrel's 3-operand instructions, which can specify the two multiplicands and one of the addends. The other addend is the multHi register. This arrangement greatly accelerates multiprecision multiplication, allowing a partial product to accumulate both the high-order byte of the previous multiplication (in a row of partial products) and the corresponding byte from a previous row of partial products. This allows, for example, multiplication of 8-word numbers in 72 cycles instead of 135 cycles with multiply-accumulate or 191 cycles with independent multiply and add.

Although the multiplier is not needed by most sequence analysis applications, incorporating it in the Kestrel architecture was seen as a requirement for creating a system that could efficiently accommodate a wide variety of applications. Among its less obvious uses, the multiplier greatly aids in the implementation of other functions, such as division [13] and shifting. The multiplier represents 18% of PE area.

2.3.4 Bit Shifter

The bit shifter is an 8-bit loadable register that can shift left or right by one bit position. Various flags from the ALU and the comparator can be shifted into either end of the shift register. Alternatively, it can be loaded with an ALU/multiplier result. The bit shifter contents are available as an operand, and its most- and least-significant bits are available as flags for controlling other PE operations.

The bit shifter serves two purposes within the PE. First, it provides a bit-wide stack for efficiently maintaining nested conditionals. Because Kestrel is SIMD, PEs in which a condition is not satisfied must be turned off while execution continues in the remaining PEs. This is done using a mask in each PE. When a new condition is evaluated, a bit representing the result can be pushed onto the bit shifter and the PE mask reset, all in a single cycle. 'Else' clauses and 'end condition' statements usually require no overhead as the required bit shifter manipulations ('invert top of stack' and/or 'pop' followed by resetting of PE masks) can be performed concurrently with the final instruction within each section of conditional code. For conditionals nested more than 8 deep, the contents of the bit shifter can be stored in a register and at the same time compressed into a single bit, clearing the remaining bits of the bit shifter for further conditions.

The second function of the bit shifter is data manipulation. Flags can be shifted into either end, and the least-significant bit (lsb) and msb can be used to select the result of instruction execution. Both of these operations are useful in the storage and retrieval of minimization costs in sequence alignment. (PE masks are of course not reset in this case.) The bit shifter comprises 6% of PE area.

2.3.5 Static Random Access Memory

Each PE contains 256 bytes of 6-transistor static random access memory (SRAM) and an address generator that supports both local and global addressing. For local addressing, the immediate field of the instruction is added to a value in a specified register. The SRAM contains a single bidirectional port to conserve area—only a single value can be read from or written to the SRAM per cycle.

When a value is read from the SRAM, it is stored in a special-purpose memory data register (MDR) that can be used as an operand in subsequent instructions. Using a special-purpose register reduces instruction requirements, allowing SRAM reads to be performed during unrelated ALU or multiplier computation. This helps alleviate the potential bottleneck of a single port and follows the philosophy of maximizing parallel operation of the PE subunits. Even with single-port access, the SRAM represents 48% of PE area. (The remaining 16% of PE area is used for control and busing.)

2.4 Kestrel Programs

Using Kestrel requires preparing a program and a corresponding input file that provides data during program execution. Similarly, an output file is generated in the host during program execution.

Although a compiler is currently being developed, programming Kestrel must currently be done in assembly language. Kestrel assembly language supports macros and provides a debugging environment for examining conditions in the PE array either at pre-specified breakpoints or by stepping through the program. Each Kestrel instruction specifies one or more of the following largely orthogonal activities:

Controller actions	**PE actions**
Date movement:	Execute ALU/comparator or multiplier function
queue $\leftrightarrow$ end of array	Select operands and destination (up to 3 SSR and 1 other)
queue $\leftrightarrow$ controller register	SRAM read (to MDR) or write
controller register $\leftrightarrow$ end of array	Bit shifter manipulation (w/ or w/o resetting PE masks)
controller register $\rightarrow$ immediate	Select flag for wired-OR
controller register $\rightarrow$ loop counter	Latch any of several flags for future use
Jumps:	Force all PEs to execute regardless of mask value
unconditional	
conditional, based on loop counter	
conditional, based on wired-OR	
Load new loop count	

While extremely complex instructions are possible, instructions often just specify an ALU/comparator or multiplier operation and its required operands. Additional actions are specified by a series of fields separated by commas at the end of an instruction, and can be presented in any order. As an example of the multiple tasks that can be performed in a single instruction,

add R1, L1, MDR, mp, read(10+[L2]), bspop, qtoarr, arrtoq, jumpwor dest1

adds the contents of SSR register L1 and the MDR (which contains the previous SRAM read) plus the carry from the previous add (the ‘mp’ indicates that this is a multiprecision operation) and puts the result in SSR register R1. It also reads the contents of the SRAM cell whose address is determined by adding 10 to the contents of SSR register L2, with the data automatically being stored in the MDR for future use (as are all SRAM reads). The ‘bspop’ is the last PE directive, indicating that this instruction is the last in a section of conditional code. The bit shifter is therefore required to shift its contents (‘popping’ this condition’s bit) and to reset PE masks. The two directives ‘qtoarr’ and ‘arrtoq’ direct the controller to input the next byte of input from the input queue to the leftmost SSR (since results are stored to the right in R1), and to output the value of the rightmost SSR to the output queue. Finally, ‘jumpwor dest1’ directs the controller to jump to the section of code labeled by ”dest1” if the array’s wired OR is a ‘0’. All of these actions will complete in a single cycle.

3 Kestrel Performance

The prototype Kestrel board became operational in late 1998, based on 64-PE chips using 0.5 μm technology. There are presently four 20 MHz Kestrel boards running at UCSC, one of which is used in a Smith &

Waterman WWW server [14]. Two other boards are currently on loan, one to UCSF for large-scale Smith & Waterman searches for genome analysis, and the other to a pharmaceutical company for accelerating protein HMM searching.

Several applications have been implemented on the first-generation Kestrel board. These range from the sequence analysis algorithms for which Kestrel was designed to applications requiring rather creative problem mappings. As the following sections show, Kestrel has performed well for all of the tested applications.

In spite of these successes, Kestrel's present 20 MHz clock rate is less than half the speed at which the 64-PE chips can run, due mainly to an initial board design that requires fetching, decoding, broadcasting, and executing one instruction per cycle. To address this bottleneck, a second generation board is being designed based on the same custom PE chips but using a pipelined controller to take advantage of the chip's 40+ MHz potential. By relocating many peripheral tasks in the FPGA, we also expect to have room to double the number of PEs on the new board. These changes will lead to a potential 4-fold performance increase over the current system. The new board will also increase performance by adding 512K bytes of on-board RAM to reduce PCI traffic when data needs to be accessed repeatedly by the array. The second generation Kestrel board is targeted for completion within the next year.

Looking ahead further, we are in the early stages of designing a chip that incorporates the entire PE array and array controller. There are several motivations for a single-chip SIMD processor. The first is to take full advantage of the potential speedup of a smaller feature size, something difficult to do in a multi-chip design because of the difficulty of maintaining single-cycle interchip communication for high clock rates. This is important because for efficient SIMD computation, the same communication rate must be maintained between all PEs—limiting this rate because of multiple-cycle interchip communication would lead to significantly reduced performance for many algorithms. A second motivation for a single-chip parallel processor is to expand Kestrel to a mesh architecture. In conjunction with increased system I/O and array-loading bandwidth, a mesh architecture would accelerate a number of applications such as neural networks and image processing. A mesh structure is less practical in a multi-chip system due to the requirement for either a high pin count on each chip or for multiplexing signals over time, again leading to multiple-cycle interchip communication. Finally, a system based on a single-chip would cost much less than the current design, which is based on 8 Kestrel chips and an FPGA. Overall, it seems possible that a mesh-on-a-chip system based on a 16-bit datapath could contain 1024 PEs and run at 200 MHz, providing a speedup of 40 or more over the present board (assuming sufficient I/O bandwidth).

We now describe some of the applications that have been implemented in Kestrel. The first two are described in greater detail; the first (sequence analysis) because it is a targeted application for Kestrel, and the second (computational chemistry) because it involves a problem that at first may not seem appropriate for a systolic array, demonstrating Kestrel's versatility.

3.1 Sequence Analysis

Because of the enormous databases involved in endeavors such as the Human Genome Project, computer aided sequence analysis is a critical part of current biological research. A very common task is to search a database for a match or near-match with a query sequence. The quality of this search is determined by the way in which the similarity of two sequences is evaluated. The most familiar sequence analysis tool is BLAST [15], a web-based serial-machine sequence matching engine based on simple one-to-one character matching. While BLAST is fast, its simple algorithm leads to alignments and scores that are not as high quality as they could be, sometimes missing related sequences that can be found by more sensitive alignment algorithms [16, 17].

Sequence alignment algorithms such as Smith-Waterman [4, 5, 6] and Hidden Markov models [7, 8, 9] add the possibility of insertions and deletions in their comparisons, more closely reflecting how biologists would make evaluations. Unfortunately, these algorithms are computationally intensive—$O(n^2)$ instead of $O(n)$—limiting their use. A primary motivation for developing Kestrel was to produce a platform that would efficiently implement this class of more sensitive alignment algorithms (as well as others that may be developed in the future).

3.1.1 Implementation

Sequence alignment algorithms are generally solved using a dynamic programming approach. A two-dimensional score matrix is formed in which each cell contains the best-alignment score for specific prefixes of the two strings being aligned (Figure 4). The final alignment score is contained in the cell representing the complete strings (the lower right cell in the figure). The value $c_{i,j}$ for each cell is calculated from 3 adjacent cells,

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

where the $cost()$ functions are specified by the particular algorithm. Hidden Markov models, which compare strings to a probabilistic models, use a slightly more complicated but related set of equations.

The recurrence above maps efficiently to a linear array [18]. By loading the query string or model into the PE array (usually one character per PE) and shifting the second string through the array, each PE calculates one column of the score matrix (Figure 4 (c)). When only the score is needed, each PE needs to store just two previous cell values at any given time and the only significant memory requirements are possible lookup tables for $cost()$ functions. When an actual alignment is required, the correspondences are found by backtracking through the matrix, requiring additional stored information. Several strategies are available for cases where there is insufficient local memory to save all of the choices made when calculating the matrix values [19, 20, 21].

3.1.2 Sequence Analysis: Results

We have implemented the scoring parts of both the Smith-Waterman and Hidden Markov Model alignment algorithms in Kestrel and compared them with a 433 MHz DEC (Compaq) Alpha series 21164.

On Smith-Waterman, Kestrel is 20 times faster when the array is fully utilized—i.e. when the query sequence or multiple query sequences total 512 elements—requiring 13 seconds to search a 10Mbase database. Compared to other parallel processors, Kestrel is slightly slower than the much larger million dollar MasPar MP-2 [22] parallel processor (with 16K 32-bit processors) [18] and faster than the similarly sized SAMBA system, a dedicated Smith-Waterman engine [23]. A comparison with the leading commercial sequence analysis engines is presented in Section 4.

We have also implemented 5 variants of the HMM algorithm. These are Viterbi Global 32-bit, Viterbi Local 32-bit, Viterbi Local 24-bit, EM Global 32-bit and EM Local 32-bit [24]. Table 1 lists the throughput of the Viterbi algorithms, with results given in characters/second of the database being searched. When the array is fully utilized, Kestrel achieves speedups of 12.5-25 on these algorithms, with performance decreasing linearly when the array is only partially filled. (The array can be filled with multiple queries when possible to avoid this performance loss.) Kestrel performance is particularly good on the local searches, which are more sensitive and therefore preferred [17, 25].

Table 2 lists the throughput for the EM algorithms. The maximum speedup here is only about 6, though we did not optimize this algorithm thoroughly. The lower speedup is due to the large number of temporary values that must be maintained during execution of this algorithm, requiring significant data movement between memory and registers.

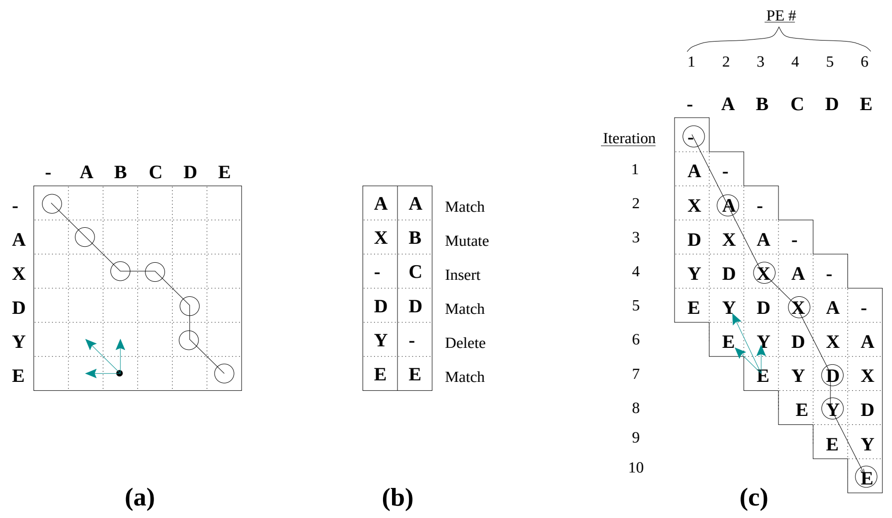

Figure 4: Dynamic programming matrix for calculating best alignment between two strings. In (a) each cell contains the score for the best alignment between prefixes of the strings being compared, and can be calculated from results in 3 neighboring cells. The path between the top left cell and the bottom right cell indicates the best alignment (b), which is determined by backtracking after the matrix is completed. (c) To map the matrix to a linear parallel processor, the query sequence (ABCDE) is loaded into the PE array and the 2nd sequence (AXDYE) is shifted through the array in reverse order. Each cell's value is calculated from results stored in either the same PE or its neighbor to the left. This calculates the original matrix diagonal by diagonal, beginning at the upper left cell and continuing to the bottom right cell.

Table 1: Viterbi Scoring (characters/second).

Length	Serial VG	Serial VL	VG32	VL24	VL32
50	113K	103K	275K	246K	180K
64	93K	78K	275K	246K	180K
128	46K	40K	275K	246K	180K
256	33K	20K	275K	246K	180K
511	22K	10K	275K	246K	180K

Table 2: EM Scoring (characters/second).

Length	Serial EMG	Serial EML	EMG	EML
100	22K	17K	29K	23K
250	9K	7K	29K	23K
500	5K	3.7K	29K	23K

3.2 Computational Chemistry

Another application implemented in Kestrel was done in a collaboration with the Affymax Research Institute. The problem related to their high-throughput approach to drug design, one step of which is to search immense synthetic combinatorial libraries for candidate molecules likely to exhibit desired biological activity. This process is made difficult by the large variety of molecular compositions and large conformational space of each molecule. To help simplify the process, a bit-vector or 'fingerprint' is created for each molecule based on whether particular features are present. Once generated, these fingerprints can be searched very efficiently for targeted features.

Calculating the fingerprints, however, is a computational bottleneck. The fingerprints are generated by classifying atoms (or atom groups) according to their electro-chemical properties and then creating a bit-vector for each molecule based on whether specific configurations of 3 labeled atoms ('3-point pharmacophores') can occur in the conformational space of the molecule [26]. For libraries containing hundreds of thousands of molecules with thousands of conformations each, this sometimes requires trillions of atom triplet calculations.

While the need for similar calculations on a large set of data made Kestrel's SIMD architecture attractive, fingerprint generation represented a significant departure from the sort of data flow anticipated during

Kestrel's development. In the end, an efficient mapping was found requiring reorganizing the bits of the fingerprint and several other algorithm transformations. The overall strategy was to load the array with multiple conformations of the same molecule so that multiple 3-point pharmacophores could be calculated in parallel. For each molecule, the main steps were:

- **Load conformations:** Atom coordinates provided through the input stream are loaded into conformation-processing blocks in the array so that each block contains a different conformation.
- **Calculate and store distance bins:** In each block, distance bins for all atom pairs are calculated and stored. (Parallel calculations and storage in all blocks.)
- **Calculate 3-point pharmacophores:** For each atom triplet of the molecule, the 3 appropriate distance bins are accessed and combined with pharmacophoric labels to specify a particular 3-point pharmacophore. (Parallel calculations in all blocks.)
- **Insert bits into FP:** Set the appropriate fingerprint bit for each 3-point pharmacophore. (Parallel insertions as described below.)

Efficient implementation of these steps required careful partitioning of the Kestrel array (Figure 5). First, due to limited local memory, either 2 or 3 PEs (depending on molecular size) needed to be grouped together to form each conformation processing block.

Second, to keep the cost of insertion into the fingerprint from being prohibitive, several copies of fingerprints were distributed along the array to allow parallel insertion of bits. These copies were later OR'ed together to form a final fingerprint. The number of fingerprint copies had to be chosen carefully because of the tradeoff with the loss of computation in the conformation-processing blocks they displaced—each fingerprint copy requiring 8 PEs. Another requirement for efficient insertion of bits was to reorganize the fingerprint structure so that all 3-point pharmacophores with the same atom labels (but different distance bins) would map to a single bit-column in a PE's SRAM. This allowed accessing the correct bit using indirect addressing based on the particular distance bins of each 3-point pharmacophore.

Finally, a large block of PEs were assigned to act as auxiliary memory. This was to reduce PCI traffic and input file size for data needing to be broadcast repeatedly to the array for a particular molecule. (An average of 70+ rounds of conformations were needed for each molecule.) Since the second generation board will contain board-level data memory, such auxiliary memory will not be necessary in the future, freeing these PEs for more productive use.

The fingerprinting program was evaluated on several subsets of a large hyperstructure library containing molecules averaging 35 atoms and nearly 10,000 conformations each. The results were compared to Affymax's serial program running on one processor of a Silicon Graphics Origin 2000, where a speedup of over 35 was achieved by Kestrel for all tested subsets.

3.3 Neural Networks

Neural networks (NNs) are a tool for pattern recognition, classification, and function approximation. Two forms of neural networks have been evaluated for implementation in Kestrel.

3.3.1 Feedforward networks

Feedforward networks commonly consist of 3 layers (input, hidden, and output), each containing multiple nodes that receive inputs from the previous layer. Each node computes a weighted sum of its inputs, adds a bias, and usually normalizes the output by an activation function. Supervised training of a NN is done by adjusting the weights for a minimal difference between the NN output from an input vector and the

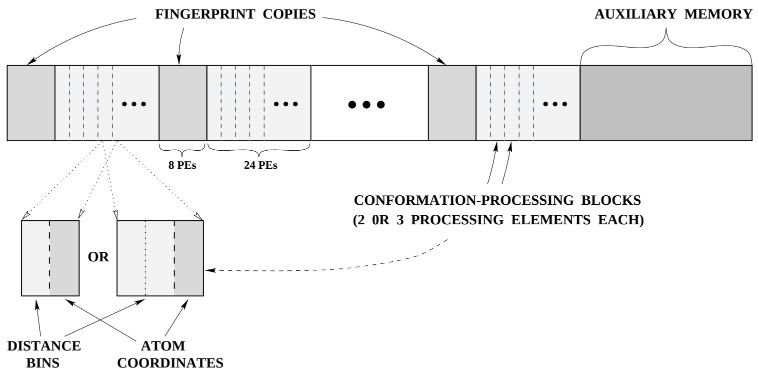

Figure 5: Partitioning of Kestrel array for fingerprint generation. Sizes of conformation-processing blocks and auxiliary memory are determined by molecular size.

desired result. Backpropagation is well-known among a large variety of training methods and derives weight corrections from gradients of this error measure traveling back through the network.

Kestrel is well-suited for feedforward NNs, for both training and classification. The PEs implement the hidden and output layers, and data pipelining allows distribution of the 8-bit input vector components to the hidden layer. Kestrel's performance is within the performance range of commercial, single-purpose neural network chips [10, 27].

3.3.2 Hopfield Nets

Another form of NNs implemented in Kestrel are Hopfield nets, fully-connected single-layer networks useful for combinatorial optimization problems. In the present case, Hopfield nets were used to solve the Maximum Clique problem, an NP-hard problem on graphs [28]. For each graph, each vertex was mapped to a PE which stored edge information to the other vertices. Each node also contained a boolean variable indicating its inclusion in a developing clique. After initializing this variable by some means, the program randomly selected a PE for which a change in the variable's value would decrease an energy function. The energy function was chosen so that when the process is repeated until a stable state occurs (i.e., where the energy cannot be reduced by changing any single variable), a clique has been found that is not the subset of a larger clique—a 'maximal' clique. Because this maximal clique is not necessarily the overall maximum clique, the process is repeated multiple times for each graph with adaptive restarts in an effort to find as large a clique as possible.

This algorithm was tested on several DIMACS benchmark graphs [29]. Performance was compared for 3 platforms: Kestrel, the MasPar parallel processor, and a serial implementation (Table 3). While the MasPar implementation used a slightly different mapping to try to maximize PE utilization (based on the "SIMD Phase Programming Model" [30]) and used 8 times as many PEs, Kestrel was able to match its performance on most graphs, and was 12-24 times faster than the serial program.

3.4 Other Applications

Several other applications have been implemented in Kestrel. The first two presented below were used to evaluate the "SIMD Phase Programming Model" (SPPM)—a method for mapping asynchronous problems

Graph	# Vertices	# Edges	Kestrel	MasPar	Serial
			(512 PEs)	(4096 PEs)	(1 PE)
p_hat500-3	500	93800	7.7	14.3	182.4
p_hat300-2	300	21928	3.4	4.4	58.2
san400_0.9_1	400	71820	8.1	13.4	230.2
sanr400_0.7	400	55869	6.8	8.3	80.8
c-fat500-10	500	46627	41.7	15.3	838.2
MANN_a27	378	70551	29.5	15.9	582.5
brock400_1	400	59723	7.4	8.8	94.4

Table 3: Average number of seconds required by Kestrel, MasPar MP-2, and a 143 MHz SUN Ultra-SPARC 1 for solving the maximum clique problem for several DIMACS benchmark graphs (8K restarts per graphs).

to a SIMD architecture using problem partitioning and array-driven scheduling of subtasks. The other studies test Kestrel's performance for a variety of problem structures.

3.4.1 Mandelbrot Set

The Mandelbrot Set is a set of points on the complex plane. Determining whether a given point is in the set requires iterating on a computation until a terminating condition is met. Mandelbrot Sets can be used to color a grid of pixels by assigning a color to each pixel based on how many iterations are needed to determine its inclusion in the Mandelbrot Set. It is not possible to predict how many iterations will be needed for a given point, however.

As a result of this irregular workload and the absence of data dependencies between results the Mandelbrot Set has been used as an asynchronous parallel application benchmark [31] and was an ideal application for parallel implementation using the SPPM strategy. By issuing one group of pixels to each PE and replacing them with new ones as each group finished, the array was kept well utilized, more than making up for the overhead of the process.

The problem was implemented on three platforms, with results shown in Table 4 (a). Kestrel's performance was significantly better than the MasPar and 12-20 times faster than the serial program.

3.4.2 N-Queens

The N-Queens problem is another widely used parallel processing benchmark application [31, 32, 33]. The problem is to place N queens on an $N \times N$ chess board in such a way that no queen attacks any other queen. Here it was required that all solutions be found. The problem was partitioned into subproblems by considering the different configurations of k queens in the first k columns. By using a large enough set of these disjoint subproblems, the array was kept well-utilized by sending in new subproblems as other finished. Results are shown in Table 4 (b).

3.4.3 Floating-point Arithmetic

Kestrel does not have hardware support for floating point arithmetic. Because of the overhead needed for software implementation, it is best to use fixed point arithmetic when possible. For cases where floating point arithmetic is required, we have created floating point libraries for five formats likely to be useful. These formats have exponents ranging from 8 to 16 bits and mantissas from 7 to 23 bits and have been optimized for Kestrel, for example with the sign bit being placed after the exponent to reduce packing and unpacking overhead. The only rounding method implemented so far is truncation. We have implemented

Maximum	Serial	Kestrel		MasPar	
# iterations	time [s]	time [s]	speedup	time [s]	speedup
1000	22.6	1.8	12.0	5	4.5
5000	104.9	6.2	16.9	13	8.0
50000	1052.2	52.9	19.9	88	11.9

(a) Mandelbrot Set

	Initial	Serial	Kestrel		MasPar	
N	branches	time [s]	time [s]	speedup	time [s]	speedup
13	6404	35.5	4.1	8.6	16	2.2
14	9632	226.0	23.4	9.6	80	2.8
15	13980	1596.3	156.3	10.2	495	3.2

(b) N-Queens.

Table 4: Execution times for the Mandelbrot Set and N-Queens programs on Kestrel, the MasPar MP-2 4096-PE parallel computer, and on a 143MHz SUN Ultra-SPARC1 serial processor. Speedups are with respect to the serial program.

these formats both with and without special representations of zero, infinity, and NaN. The second option is offered because of the high overhead required for dealing with these special cases, and makes the user responsible to provide code for such cases as needed. Division is implemented using a modified Newton-Raphson algorithm, which uses multiplication to produce quadratically converging estimates of a reciprocal [13].

Table 5: Number of Kestrel instructions required for basic arithmetic operations for five floating point formats. Numbers in parenthesis are for the option that includes representation of zero, infinity and NaN; the first number omits this option, requiring the programmer to provide software checks for these cases as needed.

Operation	Format (# Exponent/Mantissa bits)				
	8/23	16/15	8/15	16/7	8/7
Add/subtract	106(119)	101(115)	78(89)	78(90)	57(66)
Multiply	31(63)	26(58)	23(49)	20(46)	17(37)
Divide	77(109)	65(97)	62(88)	64(90)	61(81)

Table 5 shows the instruction counts for the five formats. Add/subtract is usually most expensive because of the cost of aligning operands before the operation and normalizing the result afterwards. The cost of the latter can be significant when subtraction is required (for example for addition of opposite signed numbers) since the subtraction can zero any number of bits. Because of this, when both operands are known to be of the same sign in all PEs, the cost of add/subtract can be reduced by 25–30%.

A format that could improve floating point performance would be one based on base 256, where the leading word of the mantissa is maintained as an integer between 1 and 255[1]. While this requires an extra

[1]Using a base other than 2 is not a new idea. For example the IBM S/360 processor used hexadecimal (base 16) normalization

word for the same precision and increases the cost of division (currently based on a more traditionally normalized divisor) and multiplication, it would reduce the latency of floating point addition by over 15%, since only word shifts (not bit shifts) would be required to align operands and normalize results.

3.4.4 Discrete Cosine Transforms

The discrete cosine transform (DCT) and its inverse (IDCT), as part of video compression and decompression standards, process 8×8 arrays of 8-bit pixels. Both the DCT and IDCT are separable into sequential 1-D transforms. Using standard implementation of the basic equation, each element would need 64 multiply operations, but more efficient algorithms similar to FFT exist. Our implementation was derived from the Telenor H.263 software distribution [35] and averaged 9.5 additions, 6 multiplications, and 2 scaling operations per pixel [1]. Taking advantage of multiply-accumulate instructions (but not multiply-accumulate-accumulate, which was implemented after this analysis), a full DCT required 3768 cycles and the associated I/O required 8234 cycles, leading to a rate of 107 kDCT/s on a single 64-PE chip.

Clearly, I/O is a bottleneck for this application. Extending the above analysis to the present 512-PE board accentuates this fact, with I/O requiring 95% of the execution time (65872 cycles for I/O; still 3768 for computation), and leading to just a 38% speedup over using a singe 64-PE chip. This adds motivation for increased I/O capability, such as is planned in the proposed mesh-on-a-chip discussed in Section 3.

4 Discussion: Comparing Platforms

For sequence analysis and other applications requiring high-performance parallel processing, one of three strategies is generally used: workstation clusters, FPGA-based systems, or specialized VLSI-based systems. An important underlying issue for these platforms is a tradeoff between generality and efficiency, where sacrifices in terms of speed and/or area must be made to achieve generality.

The most general purpose environment is a workstation cluster, each processor containing hardware multiplication and division as well as extensive local memory. To achieve the performance available with fine-grained approaches, however, an extremely large cluster is needed, making the cost of this approach prohibitive in most cases.

FPGA-based systems [36, 37, 38, 39, 40] achieve generality via extremely fine-grained programmability. Based on reconfigurable gates, an FPGA can implement each algorithm via an optimized parallel processor, for example basing cycle time on an algorithm's critical loop and minimizing PE area by including only as much computation hardware and local memory as required. On the other hand, this optimization comes on top of the configurable gate infrastructure, which adds extensive overhead because of the hardware needed for configurability and inefficiencies when mapping a problem to gate capabilities[2]. These factors affect both speed and area—for example FPGAs have been estimated to take anywhere from 10 [41] to 22 [42] times more area than an ASIC to implement the same logic. The FPGA-based approach is nevertheless one way to provide a platform that can accommodate multiple algorithms. An additional advantage is that the large overall FPGA market leads to improved products with a relatively small price/unit, facilitating upgrade of an FPGA-based system.

Specialized VLSI-based systems range from ASICs that implement a single algorithm [43, 44, 45, 46, 47] to more general-purpose parallel processors such as Kestrel and Paracel's Genematcher2 [48][3]. A single

in its floating point units. [34]

[2]Multiplication is particularly problematic for implementation in most FPGAs, though Xilinx's Virtex-II contains embedded multipliers to address this.

[3]The MasPar parallel processor is even more general but is no longer in production (and came at a minimum cost in the hundreds of thousands of dollars).

purpose ASIC is the most efficient way to implement any single algorithm since a truly optimized parallel processor can be designed for it (without the overhead required in an FPGA implementation). To accommodate multiple algorithms a VLSI-based system must use shorter programmable cycles, reducing efficiency by adding registers and multiplexors to the datapath. A general-purpose VLSI-based system also requires selecting a set of features for the architecture prior to manufacture, leading to a worst-case situation (often cited by FPGA-based proponents) that it may not be able to efficiently accommodate new or improved algorithms, requiring a complete hardware redesign.

Our experiences suggest that this need not be a major concern. By emphasizing generality, Kestrel has proven to be efficient not only for the targeted sequence analysis applications, but for a diverse set of applications not anticipated during its design. A brief analysis of some of the key design choices explains its versatility for high throughput performance:

- **8-bit Datapath:** Using a small word size provides flexibility with little penalty to throughput even when higher precision calculations are needed because of the reduced PE size and a corresponding increase in the number of PEs per chip. While a 16-bit word size would improve performance for many problems where throughput is not an issue, Kestrel's 8-bit datapath is well-suited to the sequence analysis algorithms [2] and the other applications that have been implemented, and is more efficient than a wider datapath for problems where 8-bit precision is sufficient.

- **256 Btyes of SRAM:** Because SRAM represents about half of PE area in Kestrel, only a factor of 2 penalty in terms of area is incurred when no SRAM is needed by an algorithm. Similarly, when more local memory is needed, PEs can grouped into blocks in which much of the computation hardware may not be useful. This again can lead to a factor of 2 penalty in terms of wasted area, though in extreme cases can lead to considerable communication and access costs as well. (This is one of the most likely reasons Kestrel would *not* be suitable for some applications.)

- **Multiplier:** At a cost of 18% of PE area, the single-cycle multiplier is important to versatility both because of the frequency of this operation and because it can be used to accelerate software division and other functions that might be required.

While Kestrel is thus well-equipped to handle many parallel applications, there are four situations where this may not be the case: **(1)** applications requiring extensive local memory (as mentioned above), **(2)** data intensive applications such as video DSP for which Kestrel's maximum I/O rate of 1 byte per cycle is insufficient (the mesh-on-a-chip being developed will have increased I/O and array-loading bandwidth), **(3)** applications requiring a large amount of high-precision floating point arithmetic (Section 3.4.3), and **(4)** applications not suitable for a linear array (the proposed mesh-on-a-chip will accommodate more applications).

In spite of these limitations, Kestrel has proven to be general purpose, each PE containing most of the qualitative features of a serial processor. Furthermore, this generality has not led to excessive loss of efficiency. To compare the fundamental efficiencies of Kestrel and the leading commercial sequence analysis engines (costing ∼$100K for entry level systems) we evaluated the performance each achieves per transistor on the Smith Waterman algorithm. A transistor-level analysis was used because of differences in chip and board sizes, preventing a fair comparison based on higher-level units. It also provides a good way to evaluate approaches for a single-chip SIMD processor, which is limited by area and therefore number of transistors.

The results of our comparison are shown in Table 4. Surprisingly, a 40 MHz Kestrel system (simulations indicate that the new board will achieve at least a 40 MHz and more likely a 45-50 MHz clock rate) will achieve the best performance even though it will still be based on 0.5 μm technology chips—a slower technology than used in the 2001 Paracel or 2001 TimeLogic systems—and even though Kestrel's multiplier represents ∼18% wasted transistors for this application. These results show that a general purpose

Table 6: Number of Smith-Waterman cells calculated per second by each transistor.

Platform	(Cells/sec) per transistor
TimeLogic's DeCypher Series G	38
Kestrel 20 Mhz board	45
Paracel's Genematcher2	64
Kestrel 40 Mhz board	91

VLSI-based system can achieve better performance/area than an FPGA-based system. They also show that Kestrel has a particularly efficient design for this application, out-performing the newer VLSI-based Paracel system.

Overall, the present Kestrel board is an extremely successful prototype, proving to be efficient for the targeted sequence analysis algorithms as well as a number of diverse applications not anticipated during its design. We hope to build on this success in a single-chip SIMD processor under development, which will incorporate a 16-bit datapath, mesh communication and increased I/O. We estimate that a single-chip design can achieve an order of magnitude improvement over the figures above using current technology, and will increasingly offer the potential for significant and versatile parallel computing power in an inexpensive system.

Acknowledgments

This work was supported in part by NSF grant MIP-9423985 and its REU supplement, NSF grant EIA-9905322, and a grant from the Affymax Research Institute. We are also particularly appreciative of the large number of undergraduates who have worked with and contributed to the Kestrel project, including: Pieris Berreitter, Alexandra Carey, Kevin Delaney, David Fulton, Jason Guilford, Mark Kendrick, Jennifer Leech, Gabriel Littman, Daniel Littrell, Justin Meyer, Michael Morrison, Eric Perlman, Osama Salem, Aaron Tomb, and Ebin Lee Warner, as well as to graduate students Cyrus Bazeghi and Doug Williams.

References

[1] D. Dahle, L. Grate, E. Rice, and R. Hughey, "The UCSC Kestrel general purpose parallel processor," *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. III, pp. 1243–1249, 1999.

[2] J. D. Hirschberg, D. Dahle, K. Karplus, D. Speck, and R. Hughey, "Kestrel: A programmable array for sequence analysis," *Journal of VLSI Signal Processing*, vol. 19, pp. 115–126, 1998.

[3] D. M. Dahle, "Kestrel: Design and Implementation of a Massively Parallel Coprocessor." Master's Thesis, Computer Engineering, UCSC. 'http://www.soe.ucsc.edu/research/kestrel/papers', Sept. 1999.

[4] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *JMB*, vol. 147, pp. 195–197, 1981.

[5] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequences of two proteins," *J. Mol. Biol.*, vol. 48, pp. 443–453, 1970.

[6] P. H. Sellers, “On the theory and computation of evolutionary distances,” *SIAM J. Appl. Math.*, vol. 26, pp. 787–793, 1974.

[7] M. Gribskov, R. Lüthy, and D. Eisenberg, “Profile analysis,” *Methods in Enzymology*, vol. 183, pp. 146–159, 1990.

[8] 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.*, vol. 235, pp. 1501–1531, Feb. 1994.

[9] R. Hughey and A. Krogh, “Hidden Markov models for sequence analysis: Extension and analysis of the basic method,” *CABIOS*, vol. 12, no. 2, pp. 95–107, 1996. Information on obtaining SAM is available at `http://www.cse.ucsc.edu/research/compbio/sam.html`.

[10] D. M. Dahle, J. D. Hirschberg, K. Karplus, H. Keller, E. Rice, D. Speck, D. H. Williams, and R. Hughey, “Kestrel: Design of an 8-bit SIMD parallel processor,” in *Proc. 17th Conf. on Advanced Research in VLSI*, pp. 145–162, IEEE Computer Society, Sept 1997.

[11] R. Hughey and D. P. Lopresti, “B-SYS: A 470-processor programmable systolic array,” in *Proc. Int. Conf. Parallel Processing* (C. Wu, ed.), vol. 1, (Boca Raton, FL), pp. 580–583, CRC Press, Aug. 1991.

[12] D. E. Knuth, *The Art of Computer Programming*, vol. 2. Reading, MA: Addison-Wesley, 2nd ed., 1981.

[13] E. Rice and R. Hughey, “Multiprecision division on an 8-bit processor,” in *Proc. 13th IEEE Symp. Computer Arithmetic* (T. Lang, J.-M. Muller, and N. Takagi, eds.), pp. 74–81, IEEE CS, July 1997.

[14] “The Kestrel WWW server.” http://www.cse.ucsc.edu/research/kestrel, 2002.

[15] S. F. Altshul, W. Gish, W. Miller, M. E. W., and L. D. J., “Basic local alignment search tool,” *JMB*, vol. 215, pp. 403–410, 1990.

[16] W. Pearson, “Comparison of methods for searching protein sequence databases,” *Protein Science*, vol. 4, pp. 1145–1160, 1995.

[17] J. Park, K. Karplus, C. Barrett, R. Hughey, D. Haussler, T. Hubbard, and C. Chothia, “Sequence comparisons using multiple sequences detect three times as many remote homologues as pairwise methods,” *JMB*, vol. 284, no. 4, pp. 1201–1210, 1998. Paper available at `http://www.mrc-lmb.cam.ac.uk/genomes/jong/assess_paper/assess_paperNov.html`.

[18] R. Hughey, “Parallel sequence comparison and alignment,” *CABIOS*, vol. 12, no. 6, pp. 473–479, 1996.

[19] D. S. Hirschberg, “A linear space algorithm for computing maximal common subsequences,” *Communications ACM*, vol. 18, pp. 341–343, June 1975.

[20] E. W. Myers and W. Miller, “Optimal alignments in linear space,” *CABIOS*, vol. 4, no. 1, pp. 11–17, 1988.

[21] J. A. Grice, R. Hughey, and D. Speck, “Reduced space sequence alignment,” *CABIOS*, vol. 13, pp. 45–53, Feb. 1997.

[22] J. R. Nickolls, “The design of the Maspar MP-1: A cost effective massively parallel computer,” in *Proc. COMPCON Spring 1990*, pp. 25–28, IEEE Computer Society, Feb. 1990.

[23] D. Lavenier, "Speeding up genome computations with a systolic accelerator," *SIAM News*, vol. 31, no. 8, pp. 6–7, 1998.

[24] L. Grate, M. Diekhans, D. Dahle, and R. Hughey, "Sequence analysis with the Kestrel SIMD parallel processor," in *Pacific Symposium on Biocomputing 2001* (R. B. Altman *et al.*, eds.), (London), pp. 263–274, World Scientific, 2000.

[25] K. Karplus, C. Barrett, and R. Hughey, "Hidden Markov models for detecting remote protein homologies," *Bioinformatics*, vol. 14, no. 10, 1998.

[26] M. J. McGregor and S. M. Muskal, "Pharmacophore fingerprinting: Application to QSAR and focused library design," *J. Chem. Inf. Comput. Sci.*, vol. 39, no. 3, pp. 569–74, 1999.

[27] C. Lindsey and T. Lindblad, "Review of hardware neural networks: a user's perspective," in *Proceedings of the Second Workshop on Neural Networks*, Elba International Physics Center, 1994. Updated version (with Bruce Denby) at http://www1.cern.ch/NeuralNets/nnwInHepHard.html.

[28] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[29] Home page of *Center for Discrete Mathematics and Theoretical Computer Science* . `http://dimacs.rutgers.edu`.

[30] A. Di Blas and R. Hughey, "Explicit SIMD programming for asynchronous applications," in *Proc. Int. Conf. ASAP* (E. E. Swartzlander *et al.*, eds.), (Los Alamitos, CA), pp. 258–267, IEEE CS, July 2000.

[31] L. Prechelt and S. U. Hänßgen, "Efficient parallel execution of irregular recursive programs," *IEEE Trans. on Parallel and Distributed Systems*, vol. 13, pp. 167–178, Feb. 2002.

[32] W. Shu and M. Wu, "Asynchronous problems on SIMD parallel computers," *IEEE Trans. on Parallel and Distributed Systems*, vol. 6, pp. 704–713, July 1995.

[33] B. Tong and H. Leung, "Performance of a data-parallel concurrent constraint programming system," *J. of Logic Programming*, May 1998.

[34] J. P. Hayes, *Computer Architecture and Organization*. New York: McGraw-Hill Book Co., 1978.

[35] Telenor Research and Development, "H.263 software version 2.0," 1996. Available from http://www.fou.telenor.no/brukere/DVC/h263_software/.

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

[37] 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*, vol. 24, pp. 81–89, Jan. 1991.

[38] D. T. Hoang, "Searching genetic databases on Splash 2," in *Proc. IEEE Workshop on FPGAs for Custom Computing Machines* (D. A. Buell and K. L. Pocek, eds.), (Los Alamitos, CA), pp. 185–191, IEEE CS, Apr. 1993.

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

[40] Y. Yamaguchi and T. Maruyama, "High speed homology search with FPGAs," *Proceedings of Pacific Symposium on Biocomputing 2002*, pp. 271–282, 2002.

[41] V. Betz, J. Rose, and A. Marquardt, *Architecture and CAD for deep-submicron FPGAs*. Dordrecht, The Netherlands: Kluwer Academic Publishers, 1999.

[42] A. Kaviani, D. Vranesic, and S. Brown, "Computational field programmable architecture," *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference*, pp. 261–264, 1998.

[43] R. K. Singh, S. G. Tell, C. T. White, D. Hoffman, V. L. Chi, and B. W. Erickson, "A scalable systolic multiprocessor system for biosequence similarity analysis," in *Symp. Integrated Systems* (L. Snyder, ed.), pp. 169–181, Cambridge, MA: MIT Press, Apr. 1993.

[44] E. Chow, T. Hunkapiller, J. Peterson, and M. S. Waterman, "Biological information signal processor," in *Proc. Int. Conf. ASAP* (M. Valero *et al.*, eds.), (Los Alamitos, CA), pp. 144–160, IEEE CS, Sept. 1991.

[45] L. Roberts, "New chip may speed genome analysis," *Science*, vol. 244, pp. 655–6, 12 May 1989.

[46] D. Brutlag, J.-P. Deautricourt, and J. Griffin. Personal Communication, Oct. 1995.

[47] P. Guerdoux-Jamet and D. Lavenier, "SAMBA: hardware accelerator for biological sequence comparison," *CABIOS*, vol. 13, pp. 609–615, Dec. 1997.

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