The UCSC Kestrel Parallel Processor

Andrea Di Blas, *Member*, *IEEE*, David M. Dahle, Mark Diekhans, Leslie Grate, Jeffrey Hirschberg, Kevin Karplus, *Senior Member*, *IEEE*, Hansjörg Keller, Mark Kendrick, Francisco J. Mesa-Martinez, David Pease, Eric Rice, Angela Schultz, *Member*, *IEEE*, Don Speck, and Richard Hughey, *Senior Member*, *IEEE*

Abstract—The architectural landscape of high-performance computing stretches from superscalar uniprocessor to explicitly parallel systems to dedicated hardware implementations of algorithms. Single-purpose hardware can achieve the highest performance and uniprocessors can be the most programmable. Between these extremes, programmable and reconfigurable architectures provide a wide range of choice in flexibility, programmability, computational density, and performance. The UCSC Kestrel parallel processor strives to attain single-purpose performance while maintaining user programmability. Kestrel is a single-instruction stream, multiple-data stream (SIMD) parallel processor with a 512-element linear array of 8-bit processing elements. The system design focuses on efficient high-throughput DNA and protein sequence analysis, but its programmability enables high performance on computational chemistry, image processing, machine learning, and other applications. The Kestrel system has had unexpected longevity in its utility due to a careful design and analysis process. Experience with the system leads to the conclusion that programmable SIMD architectures can excel in both programmability and performance. This paper presents the architecture, implementation, applications, and observations of the Kestrel project at the University of California at Santa Cruz.

Index Terms—Parallel processing, SIMD, systolic array, biological sequence analysis, DNA, computational chemistry, image processing, VLSI system design, computer architecture, high performance computing.

1 Introduction

The UCSC Kestrel parallel processor is a single-board coprocessor with a 512-element linear array of 8-bit, single-instruction stream, multiple-data stream (SIMD) processing elements. The system was designed and built at the University of California at Santa Cruz, where work on the Human Genome Project and other bioinformatics applications motivated development of a sequence analysis engine that could efficiently analyze databases containing billions of characters from DNA, RNA, or proteins.

The Kestrel project had three original goals [1]. The first goal was to develop a platform for efficient biological sequence analysis, in particular, the $O(n^2)$ Smith-Waterman (SW) and hidden Markov model (HMM) algorithms. These algorithms are particularly sensitive, but because computation time is proportional to the product of the sequence lengths, they cannot be used on single workstations for large-scale discovery.

The second goal was to create a programmable architecture that would support a variety of sequence analysis strategies and other more general fine-grained applications. Some of these target applications for our architecture include image processing, computational chemistry, and neural networks. In order to ensure generality on a wide range of applications, Kestrel includes features such as a multiplier and a conditional processing unit.

The third goal was to build a balanced system by matching computation with data input and output speeds. For the target sequence analysis applications, this meant a balance between sustained disk transfer rates and the ability for the array to process individual characters in that database. While sequence analysis has moderate I/O requirements, for which Kestrel is balanced, the system is unbalanced for image processing and some other I/O intensive applications.

An additional consideration was to make a system that would be inexpensive to manufacture and require only a single board. A high production 512 or 1,024-element Kestrel system (without the overhead of a prototype chip run) would have a low fabrication cost per system, effectively leveraging the high design cost of full-custom VLSI.

For any class of applications, there is a wide range of choices for accelerating the problem that depends on the structure of the problem and the need for high performance. Fundamental characteristics of such choices include flexibility, programmability, and computational density. Flexibility is an attribute of the hardware: Is it possible for the specific hardware to be used for different applications? Programmability is a system-level attribute that transcends

- *A. Di Blas, L. Grate, K. Karplus, M. Kendrick, F.J. Mesa-Martinez, E. Rice, A. Schultz, and R. Hughey are with the Department of Computer Engineering, University of California at Santa Cruz, 1156 High Street, Santa Cruz, CA 95064. E-mail: {andrea, karplus, tanru, javi, elrice, ang, rph}@soe.ucsc.edu, lesliegrate@attbi.com.*
- *D. Dahle is with Intel Corporation. E-mail: ddahle@fc.hp.com.*
- *M. Diekhans is with the Center for Biomolecular Science and Engineering, Baskin Engineering, University of California at Santa Cruz, 1156 High Street, Santa Cruz, CA 95064. E-mail: markd@cbse.ucsc.edu.*
- *J. Hirschberg is with Intel Corporation, DP2-400, 2800 Center Drive, DuPont, WA 98327-5050. E-mail: jeffrey.d.hirschberg@intel.com.*
- *H.J. Keller is with the Berne University of Applied Sciences. Switzerland. E-mail: keller@hta-bi.bfh.ch.*
- *D. Pease is with the IBM Almaden Research Center, 650 Harry, Road, San Jose, CA 95044. E-mail: pease@almaden.ibm.com.*
- *D. Speck is with Synaptics, 2381 Bering Drive, San Jose, CA 95131. E-mail: don@synaptics.com.*

Manuscript received 26 Apr. 2002; revised 8 June 2003; accepted 9 June 2004; published online 23 Nov. 2004.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 116424.

the specific hardware: Is it simple or difficult to create new, high-performance applications? Computational density is a hardware and software attribute, referring to the amount of computation per area, volume, or power, and can correspond inversely to the price per performance. These attributes define a continuum across five overlapping categories.

Category 1 systems are the least flexible, and include single-purpose, nonprogrammable systems, such as ASICs designed only for sequence analysis (surveyed previously [2]) or specific image processing routines. These systems can have the highest performance per area or power and, thus, can maintain their computational advantage over uniprocessors for extended periods of time. Unfortunately, such systems may not be able to maintain their advantage as algorithms and methods improve. For example, a chip designed specifically for SW looses its usefulness as HMMs are found to be a far better algorithm [3]. Similarly, a single-purpose chip designed for a specific type of neural network calculation will not be useful for other methods or activation functions. As flexibility increases, this first category merges with the next.

Category 2 includes flexible systems that are not generally user-programmable. Many Field Programmable Gate Array (FPGA) based systems for specific applications fall into this category [4], [5], as well as some bit-serial machines depending on ease of programming. These machines can have performance gains near that of Category 1, and also may be able to easily leverage off of new generations of FPGAs or semiconductor processes. Since they are programmable, though usually only by experts, it is possible to adapt these systems to new algorithms and methods. For example, FPGA and specialized FPGA systems that were originally dedicated to SW can now perform some HMM variations and other algorithms, as the product of painstaking FPGA design work [4]. There have been several projects to extend the programmability of Category 2 machines from the expert designer to the expert user, though all require detailed hardware understanding [6], [7], [8], [9].

Category 3 systems are flexible and user-programmable, though with extensive training. Programming these systems requires varying levels of knowledge of the underlying processor architecture and interconnection network, and how that might affect efficiency, but does not require detailed hardware knowledge about, for example, FPGAs or bit-serial programming. This category includes machines that can have computation densities within a factor of 10 of the first two categories, as in SIMD arrays such as Kestrel, MPP, CM-2, MasPar, Fuzion 150, or other SIMD machines [10], [11], [12], [5]. Algorithms with fine-grain parallel mappings to these systems can have excellent performance, and the machine performance may be able to scale with the underlying technology.

Category 4 systems are flexible and user-programmable, but with less extensive training. These machines, tightly coupled MIMD multiprocessors and large clusters, can maintain the high performance of Categories 1-3, but have a computation density on an individual application thousands or millions of times lower than the single-purpose machines [13], [14]. These are the most programmable of the high-performance group, and can often be used effectively with minimal knowledge of the underlying technologies. A sequence analysis program in comparative genomics can be easily subdivided into millions of individual tasks, which can then be distributed to multiple processors with only a little programming difficulty and minimal knowledge of the architecture. Similarly, in image processing, individual frames or parts of frames, or multiple video streams can be partitioned among a group of processors.

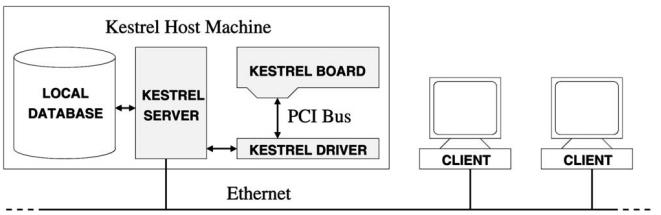


Fig. 1. Client/server model for the Kestrel system.

Category 5 consists of flexible high-performance uniprocessors that require no specialized knowledge for programming. Currently, this group includes super-scalar and multithreaded architectures that maintain a serial programming interface [15]. The computational efficiency of Categories 1, 2, and 3 are lost, but the easy programmability makes these ideal for application and algorithm development. Such applications can often be quickly migrated to Category 4 machines, and with more difficulty Category 3, and are occasionally important enough to implement in the hardware of Categories 1 and 2.

The Kestrel goal of high performance and high flexibility expands the boundaries of Category 3 in both directions. The highly tuned VLSI architecture enables Category 1 or 2 performance on sequence analysis and other applications, and the attention to programmability has enabled a wide range of application developments. Additional software development, such as compilers and comprehensive software libraries beyond the scope of our university project, could even place the system among the most programmable of Category 3.

The architectural decisions surrounding Kestrel (primarily made in 1993-1995) led to a particularly high density of computation that enables the single-board system with 9 million transistors of custom VLSI, an FPGA, and various memory chips, to outperform a current workstation 10 years later. This possibility of careful design enabling longevity in a given technology is an important feature of Categories 1-3 that tends to vanish with Categories 4 and 5 with their much lower density of computation. Algorithmic development and advances are best handled by Categories 5 (most easily), 4, and 3 (with some difficulty), and can only be handled in Category 2 with great difficulty, the remapping or redesign of the firmware, and cannot be handled by single-purpose Category 1 machines. Thus, in this landscape of flexibility, programmability, and density of computation, Category 3 is one of the most attractive points for appropriately specialized computation.

2 System Architecture

Kestrel is a single-board coprocessor designed for Linux and Windows NT PCs (Fig. 1). The host runs a network server and a board driver. The server provides a program and data interface with clients for remote connection, while the driver manages all the details of program execution.

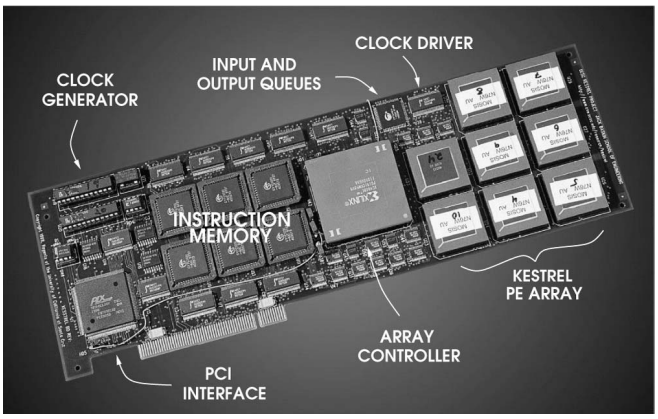


Fig. 2. The Kestrel prototype board showing the physical location of all major components.

When the program completes, the server returns an output file and status information to the client.

2.1 Kestrel Board

The Kestrel board includes a 512-PE array on eight 64-PE full-custom chips, a controller, an instruction memory, input and output queues, and a PCI interface chip. The FPGA-based on-board controller provides an interface between the host machine and the PE array, performing instruction issue, program control, and I/O control (Figs. 2 and 3).

Kestrel's processing elements are organized in a linear array, with inter-PE communication between nearest neighbors. This topology is the minimal requirement for sequence comparison algorithms and is easily scalable at the chip and at the system level.

The data flow originates from an input file sent from the client to the server. A system driver sends the data to the board through the PCI bus. A PCI interface chip fills a 4 KB input queue FIFO that is read by the controller. The controller sends data to the PEs either through one of the bidirectional, 8-bit data buses at the ends of the array or to all PEs via an 8-bit immediate bus. Data is output from the end PEs to the controller to a 4 KB output queue. The queue is read by the PCI interface chip and sent, upon driver's request, to the host and then back to the client.

The control flow originates from a program sent from the client to the server. The Kestrel driver loads the program through the PCI bus into the board's instruction memory. The driver then instructs the controller to begin execution. The controller fetches one instruction per clock cycle and broadcasts the instruction to the entire array. Instructions are 96 bits wide, 44 of which are used by the controller, 44 are used by the array, and 8 are used by the immediate bus. The tight integration of controller and array instruction is a key feature of this architecture.

In addition to managing all I/O operations, the controller acts as instruction sequencer and includes three control mechanisms: unconditional jumps, conditional jumps based on the wired-or, and loops based on the value of a 16-bit counter (the controller has a counter stack that allows up to 15 nested loop counts). The controller also generates interrupts when an I/O queue needs servicing, when the program terminates, or for single-stepping.

2.2 Processing Elements

The heart of the Kestrel system is the processing element and the Systolic Shared Registers (SSRs) (Fig. 4). Each PE contains an integrated ALU/comparator, a multiplier, a bit shifter, flag logic, a shared 32-byte register file, and 256 bytes of static random access memory. To maximize the performance and provide flexibility, these components operate independently when possible. Horizontally microcoded instructions specify an arithmetic/logic instruction, up to three source operands, one destination operand, local memory access, flag selection or latching, and PE masking/conditional instructions. This encoding enables parallelism at the instruction level, which significantly improves performance in most applications. The PE is a single-cycle machine, with all operations—including multiply-accumulate-accumulate—completing in one clock cycle.

The data path is 8 bits wide. This size was originally chosen as an appropriate balance between PE size, especially the multiplier, and cycle time, with special emphasis on the sequence analysis applications. The data path supports both signed and unsigned operations, with full support for multiprecision across multiple clock cycles.

2.2.1 Systolic Shared Registers

The SSRs are located between adjacent PEs and provide storage and communication [16]. Each SSR contains 32 8-bit registers with two read ports and one write port. Source and destination registers are specified in any combination

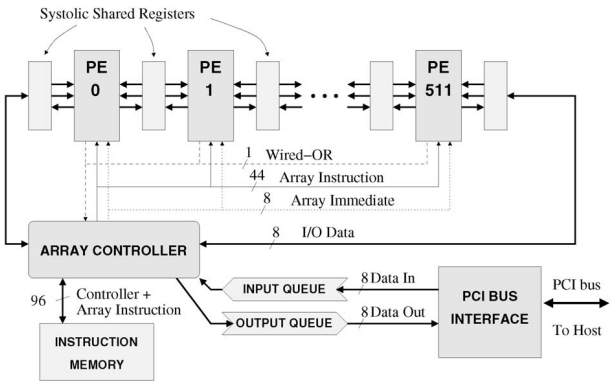


Fig. 3. Block diagram of the Kestrel board.

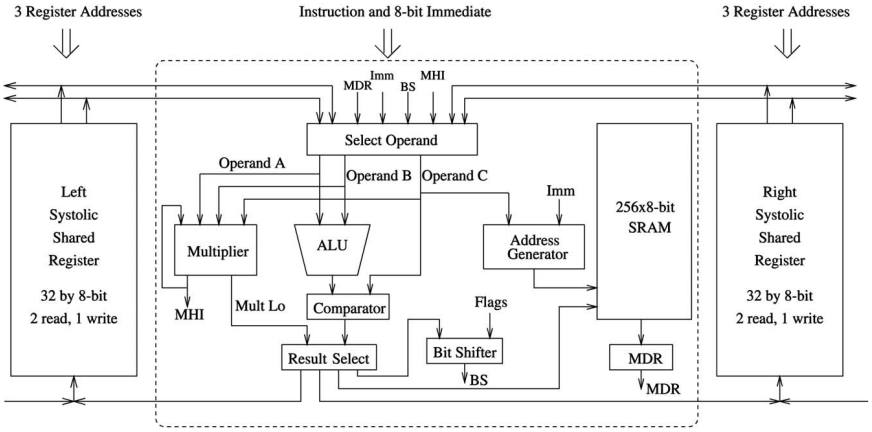


Fig. 4. A Kestrel processing element with systolic shared registers.

from a PE's right and left SSRs, allowing no-overhead communication when partial results need to be shifted through the PE array. Because instructions are broadcast, there is no chance for conflict in SSR use between PEs.

To speed register reads at chip boundaries, each 64-PE chip has 65 SSRs. In multichip systems, coherency is maintained between the adjacent SSRs. For example, when a write occurs to the left, a value is written to the leftmost SSR of a chip and sent off-chip to the rightmost SSR of the adjacent chip. Register reads are always made from on-chip SSRs.

2.2.2 ALU/Comparator

The ALU combines two 8-bit operands and a carry-in bit to produce an 8-bit result and a carry-out bit that can be latched for multiprecision operations. The ALU is capable of all common logic functions as well as addition and subtraction, and is tightly coupled with a comparator [17]. Many sequence-analysis algorithms repeatedly find the minimum of a value and a sum, The ALU/comparator accelerates this by allowing the comparison of an ALU result with another operand and the selection of the minimum or maximum value in one cycle. This operation motivated Kestrel's 3-operand instruction format, a feature that is also exploited in the multiplier.

2.2.3 Multiplier

The multiplier takes two independently signed or unsigned 8-bit operands and conditionally adds one or two other 8-bit operands to produce a 16-bit result. The high-byte of the result is stored in a special register, and the low byte of the result is stored as specified by the instruction. The multiply-accumulate-accumulate operation [18] is enabled by Kestrel's 3-operand instructions, which can specify the multiplier, multiplicand and one of the addends. The other optional addend is the high byte of the result of the previous multiplication. This arrangement accelerates multiprecision multiplication, allowing a partial product to accumulate both the high-order byte of the previous multiplication (in a row of partial products) as well as the corresponding byte from a previous row of partial products. For example, multiplying two unsigned 4-byte numbers requires 20 instructions instead of the 47 that would be required without the double accumulation feature. The multiplication and the two additions all complete in one clock cycle.

2.2.4 Condition Stack

The condition stack is an 8-bit register that can shift left or right by one bit position, providing a byte-wide stack for efficiently maintaining nested conditionals. Because Kestrel has broadcast instructions, PEs in which a condition is not satisfied must be "turned off" while execution continues in the remaining PEs. Each PE has a mask bit, the NOR of the bits in its condition stack. When set, the PE is on, when it is clear, the PE will not execute any instruction except for the ones that manipulate the condition stack. All operations involving the condition stack execute in parallel with the ALU/comparator or multiplier, local SRAM access, and flag setting/selection, effectively making conditional execution a no-overhead operation. For conditionals nested more than eight 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 condition stack can also be used as a simple bit shifter and can provide input to the wired-or bus or to a bit-serial log-reduction network used to speed the counting of active PEs.

2.2.5 Local PE Memory

Each PE contains 256 bytes of 6-transistor static random access memory (SRAM) and an address generator. Global addressing uses the immediate field of the instruction, and for local addressing, the immediate is added to a specified register. Reads load the value into the memory data register (MDR), which can then be used as an operand in subsequent instructions. Using a special-purpose register allows local memory reads to be performed during unrelated arithmetic operations. This helps alleviate the potential bottleneck of the single-port memory and follows the philosophy of maximizing parallel operation of the PE subunits.

2.2.6 Physical Design

Kestrel's full-custom CMOS chips were fabricated in late 1997 and the complete system was running by late 1998 [1], [17]. Thanks to a careful design of the PE and to the simple connection scheme of a linear array, 64 Kestrel processing elements fit in a single chip (Fig. 5). The chip has 1.4 million

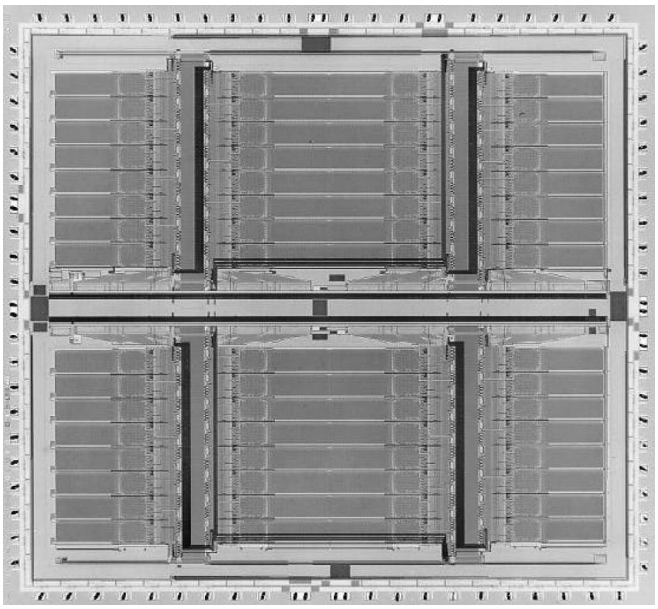


Fig. 5. A photograph of the Kestrel 64-PE chip.

transistors on a 7.2 mm $\times$ 8.3 mm die in HP 0.5μm CMOS technology. The PE's floorplan is 8 percent systolic shared registers, 4 percent ALU, 4 percent comparator, 18 percent multiplier/accumulator, 6 percent condition stack, 48 percent local SRAM, and 12 percent connections and additional control. The power dissipation is within the 1W range that can be dissipated by the 84-pin ceramic PGA package without heat sinks or fans. Although the chips have been tested to run at up to 45 MHz, the Kestrel board runs at 20 MHz because of a slow controller (single-cycle, nonpipelined) and signal integrity problems on the board.

2.3 Kestrel Programs

Kestrel is programmed in macroassembly language. Each instruction specifies one or more of the largely orthogonal activities listed in Table 1. While most instructions simply specify an ALU/comparator or multiplier operation and their required operands, additional actions can be specified by a series of fields separated by commas. For example,

```
add R1, L1, L2, read (#20)
```

reads the contents of memory location 20, storing it in the MDR, while at the same time adding the contents of left SSR registers L1 and L2 and placing the result in right SSR register R1. As a better example of the degree of parallelism at PE subunit level, consider the code

```
BEGINLOOP 512
ENDLOOP multsab addmhi R10, L5, MDR, addmc R10, read(#35), arrtoq, qtoarr
```

This code initializes the controller's loop counter to 512, in one clock cycle (`BEGINLOOP 512`). It then executes 512 times, in 512 clock cycles total, a multiplication of two signed operands (`multsab`, L5, and MDR), adding to the product the multiplier's high byte from the previous multiplication (`addmhi`) and the current content of the destination register R10 (`addmc R10`). Also, memory location 35 is read and loaded into the MDR, one byte is read from the input queue into register L10 of the PE at the "left" end of the array (`arrtoq`), and one byte is sent to the output queue from register R10 of the PE at the "right" end of the array (`qtoarr`). In parallel, the controller decrements the loop counter and performs a termination check and a jump (`ENDLOOP`). While this example seems extreme, similar situations are frequent in actual Kestrel programs and are one reason for the high performance of the machine.

3 Applications

We have implemented a variety of applications on the Kestrel board, ranging from the sequence analysis algorithms for which Kestrel was designed to applications requiring more creative problem mappings. Kestrel continues to perform well on many of these applications, even when current technologies enable gigahertz clocks and hundreds of millions of transistors on a chip. On others, our experiences with the Kestrel hardware and software shows room for improvement in PE and system design that would enable Kestrel to similarly excel.

3.1 Biosequence Analysis

Computer aided sequence analysis is a critical part of current biological research. A 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

TABLE 1
Operations That Can Be Specified in a Kestrel Instruction

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	

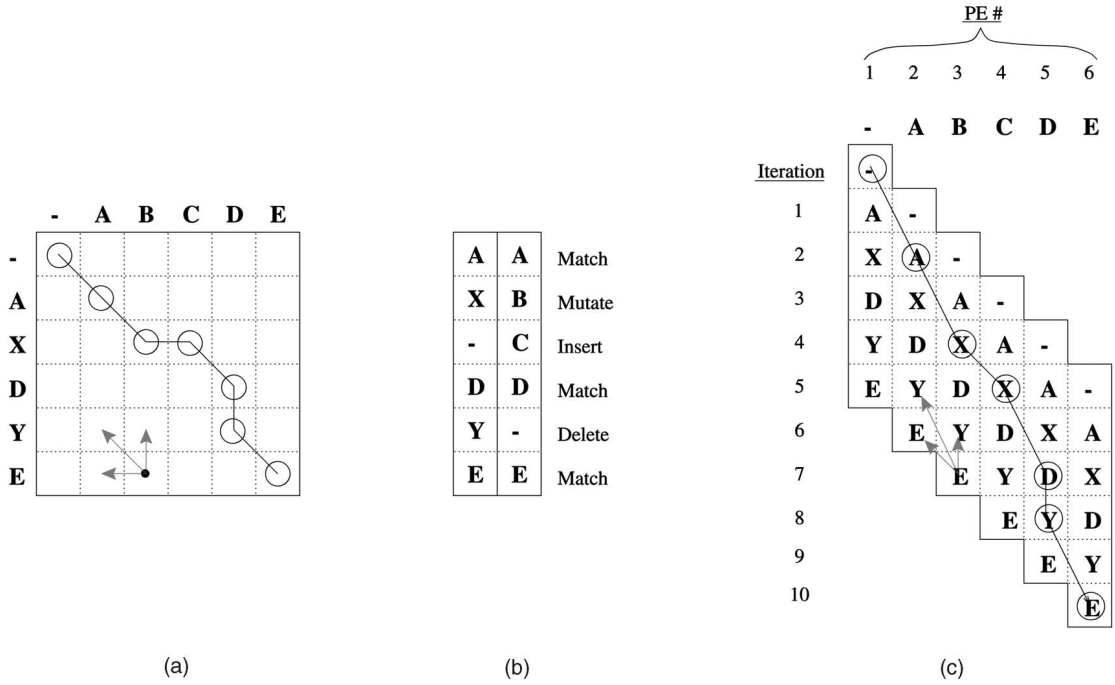


Fig. 6. Dynamic programming matrix for calculating the best alignment between two strings. In (a), each cell contains the score for the best alignment between prefixes of the strings, and can be calculated from results in three neighboring cells. The path between the top left and the bottom right indicates the best alignment (b), which is determined by backtracking after the matrix is completed. To map to a linear parallel processor (c), the query sequence (ABCDE) is loaded into the PE array and the second 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, from the upper left to the bottom right.

most familiar sequence analysis tool is BLAST [19], a 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 good as they could be, sometimes missing related sequences that can be found by more sensitive alignment algorithms [20], [21].

Sequence alignment algorithms such as Smith-Waterman [22] and hidden Markov models [23] add the possibility of insertions and deletions in their comparisons, more closely reflecting how biologists would like to make evaluations. Unfortunately, these algorithms are computationally intensive—$O(n^2)$ instead of $O(n)$—thus, 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 [21], as well as others that may be developed in the future.

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 (Fig. 6). 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 three 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 (HMM), which compare strings to probabilistic models, use more complicated but related recurrence equations.

Sequence analysis is easily parallelized with pipelining or by distributing pair-wise comparisons among a set of independent processors [2], [24], [4], [25]. On a linear processor array, the query string or model can be loaded into the PE array, and the second string (the database) can be shifted through the array. Each PE calculates one column of the score matrix (Fig. 6c), completing a diagonal $i + j = t$ in parallel during each time step t. To score a sequence, 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 or HMM probabilities. When an alignment is required, the correspondences are found by backtracking through the matrix, requiring additional stored information, either by redoing the calculation on a serial machine or using one of several low-memory strategies on a parallel processor [26], [27].

We have implemented SW and HMM algorithms on Kestrel [28]. For the HMM algorithms, we have implemented both global and local versions. Global algorithms, such as the simple recurrence above, consider the problem of matching two complete sequences. Local algorithms, such as SW, find the score of the best matching subsequences of the two sequences, and have slightly more complicated recurrence equations. The slower local algorithms are almost always preferred for biological sequence analysis; a more detailed discussion of these algorithms is presented elsewhere [28].

To perform a database search with an HMM, the two most common algorithms are the local versions of the

TABLE 2
SW and Local HMM Search Times in Seconds for a 10-Million-Character, Randomly Selected Subset of a Standard Protein Database on Kestrel and a 500 MHz UltraSPARC-II

	Smith-Waterman			Viterbi HMM			Forward HMM		
Query Size	Serial Time	Kestrel Time	Kestrel Speedup	Serial Time	Kestrel Time	Kestrel Speedup	Serial Time	Kestrel Time	Kestrel Speedup
100	220	13	17	326	40	8	686	434	1.5
250	630	13	49	851	40	21	1716	434	4
500	1287	13	99	1769	40	44	3510	434	8

Viterbi and the forward algorithms. The Viterbi algorithm finds the score of the single most likely path through the HMM [29]. The calculation is done on log-probabilities, and involves summations and maximizations as with both SW and the simplified recurrence above. The forward algorithm calculates the probability that the sequence could be generated by the model over any possible path. This involves multiplying and adding probabilities represented as log probabilities to avoid underflow. Our serial code [30] uses table lookup in a 7,600-element table to sum probabilities represented as log-probabilities. This is not practical with just 256 bytes of memory, so we experimented with two solutions: a custom floating-point format with 24-bit exponents and 8-bit mantissas, and piecewise-linear approximation on 32-bit log probabilities, both of which are enabled by the Kestrel PE's multiplier. The second solution turned out to be more efficient.

Speedups over a serial machine are shown in Table 2; the forward algorithm has poor relative performance because of the large number of temporary values that must be maintained during execution of this algorithm, requiring significant data movement between memory and registers.

On this problem, Kestrel is slightly slower than the Category 3 MasPar (which has 32 times as many PEs each with a 32-bit ALU) [2], faster than the similarly sized SAMBA system, a dedicated (Category 1) SW engine [31], about six times faster than a 1,024-PE 1.0μm bit-serial (Category 3) Systola system and about six times slower than the 0.25μm, 76-million-transistor Fuzion 150 single-chip SIMD array [5]. There are several commercial sequence analysis systems. Paracel's GeneMatcher2 is a VLSI-based Category 2 system with 3,072 PEs in 16 chips, and appears to be three times faster than Kestrel [24]. The DeCypher Series G, based on four Altera FPGAs, appears to be nine times faster than Kestrel [4]. The entry-level systems for each of these products are costly ($75,000-$100,000) compared to a Kestrel board that could be manufactured for about $1,000 (0.5 percent of development cost) in medium-volume production.

3.2 Computational Chemistry

One of the first Kestrel applications outside of its targeted domain was a problem from computational drug design. One step of this process is to search immense synthetic combinatorial libraries for candidate molecules exhibiting certain geometrical attributes. The process is made tractable by generating bit-vectors or "fingerprints" for each molecule which (once generated) can be quickly examined for targeted features.

Calculating the fingerprints is the computational bottleneck. The fingerprints are generated by classifying atoms according to their electro-chemical properties and then creating a 14kbit bit-vector based on whether specific configurations of three labeled atoms can occur in the conformational space of the molecule. Because most molecules contain several rotatable bonds, this can lead to significant calculation—for libraries containing hundreds of thousands of molecules with thousands of conformations each, trillions of atom triplet calculations can be required. While performing identical calculations on a large data set is a classic SIMD problem, fingerprint generation was a major departure from the sort of data flow anticipated during Kestrel's design. On a problem with 80 million conformations, Kestrel sped the problem by a factor of 35 in comparison to 25 hours on one processor of an SGI Origin 2000 [32].

This application illustrated the limitations and the flexibility of the Kestrel memory and I/O system. The repetitive nature of the calculation meant that identical pieces of data needed to be broadcast to the array many times. The Kestrel prototype has no onboard data memory, so the first implementation downloaded the information to the board multiple times. It turned out to be much faster to reserve 40 percent of the array simply to store this data (rather than actively compute) because of the rapid reuse. The board architecture clearly should have included local data memory.

We divided the remaining PEs into super blocks of 32, and within each group of 32, eight PEs were used to store temporary results. Because Kestrel is a linear array with a bandwidth of only 1 byte per clock, it was faster to save results locally and shift them out of the array once the entire fingerprint calculation was complete. This bottleneck could be reduced with a higher-bandwidth path for shifting data in and out of the array. For nonsystolic algorithms, these shifts cannot be done in parallel with computation.

The remaining PEs in each group were clustered into blocks of three to store individual atom coordinates (in Angstroms) and all the pairwise distances (binned into six distance ranges). With more local memory, for example 1,024 bytes, two out of each group of three PEs would not be idle. However, since the 256-byte local memory is roughly half of the PE area, the mapping is effectively using 2/3 of the total area allocated to each group. Increasing the PE memory would reduce the number of PEs and, thus, reduce performance on problems like SW that are not memory bound, all other things being equal.

TABLE 3
Performance of Some Image Filters on Kestrel

Algorithm	Serial Time	Kestrel Time	Kestrel Speedup	Kestrel Fast I/O Time	Kestrel Fast I/O Speedup
2D Gaussian convolution (5×5)	280	222	1.2	16	18
2D Gaussian convolution (7×7)	340	225	1.5	17	20
2D Gaussian convolution (9×9)	410	227	1.8	18	23
2D Gaussian convolution (11×11)	490	231	2.1	19	26
2D Non-separable conv. (15×15)	4620	565	8.1	87	53
Edge detector	270	562	0.5	18	15
2D Discr. Wavelet Transf. (spline 5/3)	340	226	1.5	18	19
2D Discr. Wavelet Transf. (CDF 9/7)	530	227	2.3	23	23

All times are in milliseconds for a 512×512-pixel frame, 8 bpp. The serial machine is a 500 MHz UltraSPARC-II. "Kestrel Fast I/O" is the performance with a host-board bandwidth that could keep up with the actual processing power of the board.

3.3 Image Processing

We also implemented a number of image processing applications on Kestrel (Table 3). Category 3 SIMD computers are known to perform well on these highly-parallel, computationally-intensive problems with little or no data dependencies, especially on linear arrays [33], [34].

A bidimensional convolution with a Gaussian kernel is commonly used to filter image noise. Even though typical image processing applications do not use kernel sizes larger than 7×7, we report results for higher sizes to confirm that parallelism and speedup increase with kernel size. A bidimensional Gaussian convolution is *separable* because it can be decomposed into two linear convolutions, which significantly reduces the computational load. A *nonseparable* convolution requires computing the full bidimensional dot product for every pixel—the larger kernel size and the high degree of parallelism led to considerable speedup.

An edge detector performs a discrete derivative along the x and y direction and returns the magnitude of the resulting vector. This application requires the extraction of a square root for which we used a third-order Chebyshev polynomial approximation since Kestrel does not have a floating-point unit. It is interesting to analyze this function to gain a quantitative idea of how some of Kestrel's architectural features impact performance. The routine requires 109 machine instructions (109 clock cycles). Of these, 17 execute conditional branching and ALU operation at the same time, nine execute local memory access and ALU operations at the same time, 29 execute a multiplication and an addition in the same clock cycle, and nine execute one multiplication and two additions in the same clock cycle. These features, in this example, improve the performance by almost 40 percent with respect to a function that would have otherwise required 176 instructions.

The last two applications are the discrete wavelet transforms used in JPEG2000 [35]. The algorithm is basically a separable convolution, with the difference that even and odd pixels are convolved using different kernels that are also different in size. This makes the algorithm slightly more complex, but with a performance similar to that of a standard separable convolution.

For all applications, we report two sets of performance numbers. In the column labeled "Kestrel," we report the actual computation times that include the I/O communication time across the PCI bus. In the column labeled "Kestrel Fast I/O," we report the computation times that we measure on the same applications assuming a much faster transfer rate over the PCI bus. Kestrel's I/O subsystem, designed for sequence analysis applications, can sustain 2.5 MB/s. While this is more than sufficient for sequence analysis, it limits image processing applications, which ideally would see 25 MB/s, difficult to achieve on the 32-bit, 33 MHz PCI bus.

3.4 Floating-Point Library

Kestrel does not have hardware support for floating-point arithmetic, as such would have required significant increase in PE size. For cases where floating-point arithmetic is required (it is best to use fixed-point arithmetic whenever possible), we created floating-point libraries for five formats. 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 these formats both with and without special representations of zero, infinity, and NaN. In the first case, Kestrel performs addition at 86 MLOPS, multiplication at 162 MFLOPS, and division at 93 MFLOPS. The second option is offered because of the high overhead required for dealing with these special cases, making the user responsible for such cases as needed. This improves performance to 96 MFLOPS for addition, 330 MFLOPS for multiplication, and 132 MFLOPS for division.

Add/subtract is the most expensive because of the cost of aligning operands before the operation and normalizing the result afterward. The cost of normalizing a subtraction result can be high because subtraction can zero any number of bits. When both operands are known to be of the same sign in all PEs, the cost of addition can be reduced by 25-30 percent. The cost of add/subtract can be reduced by more than 15 percent by using a floating-point format based on base 256—where the leading word of the mantissa is maintained as an integer

TABLE 4
Hopfield Neural Networks for MaxClique on Some DIMACS Benchmark Graphs

Graph	Vertices	Edges	Serial Time	MasPar Time	Kestrel Time	Speedup to Serial	Speedup to MasPar
MANN_a27	378	70551	68	16	30	2.2	0.5
c-fat500-10	500	46627	91	15	42	2.2	0.3
johnson32-2-4	496	107880	12	14	8	1.4	1.7
san400_0.9_1	400	71820	26	13	8	3.2	1.7

In all cases a total of 8K restarts is performed per graph. The serial machine is a 500 MHz UltraSPARC-II, the MasPar MP2 is a 12.5 MHz, 32-bit architecture with 4,096 PEs.

between 1 and 255. (A similar strategy based on base 16 was used in the IBM S/360 processor [36].) On Kestrel, this would remove the need for bit-wise normalization. On the other hand, the cost of multiplication and division would increase, and each floating-point value would require an extra word for the same precision. Division is implemented using a modified Newton-Raphson algorithm, which uses multiplication to produce quadratically converging estimates of a reciprocal [37].

3.5 Neural Networks

Neural networks are a tool for pattern recognition, classification, function approximation, and combinatorial optimization. Hopfield networks are one form of neural network that we implemented on Kestrel.

Hopfield networks are fully connected networks useful for combinatorial optimization problems. On Kestrel, we used Hopfield nets to solve the Maximum Clique problem, an NP-hard graph problem [38]. For each graph, each vertex is mapped to a PE which stores edge information to the other vertices. Each node also contains a Boolean variable indicating its inclusion in a developing clique. After initializing this variable, the program randomly selects a PE for which a change in the variable's value would decrease an energy function. The energy function is 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 is 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.

We tested this algorithm on several DIMACS benchmark graphs [39] on two Category 3 machines, Kestrel and a MasPar MP-2, as well as on a Category 5 serial machine, a Sun workstation (Table 4) [40], [41]. The MasPar implementation used a slightly different mapping to try to maximize PE utilization based on our "SIMD Phase Programming Model" [42].

4 Discussion

The UCSC Kestrel parallel processor was designed to accelerate biosequence analysis, matching computation and I/O, and at the same time to be as flexible as possible and to have a measure of simplicity that could ensure its low unit cost. Kestrel succeeded in these goals, and remains useful a decade after its original design, a testament to broadly applicable, user-programmable SIMD (Category 3) computing. The design and implementation of Kestrel continuously considered computational density, flexibility, programmability, and performance. Its success is due to many people and design choices and, in this section, we attempt to isolate some of the important decisions.

Computational Density. The close coupling of architectural design and VLSI design enabled creation of a dense full-custom chip. This high computational density, even in a technology that was somewhat out of date on fabrication, meant that the 1.4 million transistor chip would still be useful even as fabrication generations passed it by. In the sequence comparison domain, computation density can be represented in SW cells updates per second per transistor, CUPS/T (Table 5). This measure is not fully independent of technology scaling, as it does not take into account increases in clock speed between fabrication generations. A Kestrel system implemented in $0.25\,\mu$m would be expected to run, on a well-designed board, at 80 MHz (twice as fast as the current maximum speed of the $0.5\,\mu$m chips), giving 140 MCUPS/T. The most interesting point about this table is that Kestrel, and Fuzion in its more advanced technology, have higher densities of computation than machines designed solely for sequence analysis.

Flexibility. There are three primary approaches to flexibility: hardware configuration, distributed software control, and global software control.

Hardware configuration, as in FPGAs, can provide very high performance on specific algorithms. The control bits are located within the controlled units, so there is no time required, after initial configuration, for control distribution. FPGAs are thus excellent for single-purpose or few-purpose machines, but restrict user programmability because they require hardware design. Distributed software control, as in MIMD machines, solves the programmability problem but greatly reduces computational density by requiring significant local control and memory. As designers continually strive to maintain the uniprocessor model on super-scalar machines, control becomes increasingly complex, making such processors a poor basis for high-performance single-chip multiprocessor chips.

Global software control, as on SIMD machines, enables computational density higher than either FPGAs or MIMD

TABLE 5
Smith-Waterman Performance and Technology Efficiency for Different Architectures

Platform	Year	Technology		MHz	MT/C	PE/C	C/S	MCUPS	CUPS/T
MP2	1992	1.0 μm	ASIC	12.5	1	32	512	500	0.1
Kestrel	1997	0.5 μm	ASIC	20	1.4	64	8	400	36
Fuzion 150	2000	0.25 μm	ASIC	200	76	1536	1	2500	33
DeCypher	2001	0.18 μm?	FPGA	?	69?	?	4	7500	27?
GeneMatcher2	2001	0.13 μm?	ASIC	192	> 40	192	16	16000	< 25

(M)T stands for (Millions of) Transistors, C for processing element (PE) Chips, S for System, (M)CUPS for (Millions of) Cell Updates per Second.

machines, but may reduce programmability in comparison to MIMD machines and may reduce performance in comparison to distributed hardware configuration. Instruction broadcast has two primary disadvantages: not all PEs may need the broadcast instruction, and instruction broadcast can take time.

Kestrel addresses local control with its condition stack and associated processing. This reduction in processing time for conditionals minimizes the processing time wasted for simply managing conditions (as is also similarly wasted on serial machines), reducing the overhead to the bare minimum: the set of data-stream operations that are performed as part of the "if" and as part of the "else." Architecturally, the condition stack and its tight integration with the processing elements is one of Kestrel's most important innovations. Kestrel's global execution control based on local conditions (wired-OR) and local indirect addressing of PE memory add additional processor autonomy that is critical to many applications [43].

Programmability. The SIMD programming paradigm requires more training than serial programming or MIMD programming. However, the efficiency gains in computational density (and, hence, performance per price) of Category 3 SIMD machines can outweigh the specialized understanding required for these machines.

The simplicity of the architecture, especially the integration of computation and communication with the Systolic Shared Registers, provides a certain level of intuitiveness and elegance to Kestrel programming. Many undergraduate and graduate students have quickly learned kestrel assembly language to complete class projects including HMM, FFT, floating-point, neural networks, trigonometric functions, discrete wavelet transforms, discrete cosine transforms, and irregular applications.

As with many predecessor machines, we contemplated the ideal programming system for Kestrel. Starting with MasPar's MPL language, we considered a variety of additions to enable full exploitation of the Kestrel architecture and to simplify many of the higher-level tasks. We did not complete the project, in part due to personnel changes and in part due to the difficulty of optimizing code for 32 registers and 256 bytes of memory. As a university project, our time was better spent creating new applications, accelerating the inner cores by hand to explore the frontiers of specialized SIMD computation.

As the category of single-chip-SIMD machines expands, as well as the code base, these machines may approach the higher programmability of those in Category 4. Although the underlying architecture constrains low-level programming, it does not constrain the development of comprehensive libraries for various application domains, or the development of programming paradigms to assist user development of new library functions. This will fit well as efforts in sequence analysis, image and graphics processing, and machine learning become subroutines called as part of a processing pipeline to, for example, predict the structure of a protein, design a molecule, or understand a video stream.

Performance. Kestrel performs exceptionally well a decade after its original design. One design strategy that proved particularly important was the balancing of computation with broadcast time. Simply put, if it takes a significant amount of time to broadcast an instruction, make sure that the computational units determine clock speed. Each single broadcast must do one or more useful operations; an SIMD machine that requires separate broadcasts for each bit, or microcodes each operation over several cycles, will quickly collapse under its own weight.

Kestrel has a "somewhat long" instruction word. Each instruction has complete information about operands from SSRs and special registers, ALU and multiplier operations, condition stack operations, local memory access, and result storage. The instruction memory to PE instruction broadcast is not a determining factor of Kestrel clock speed.

Instruction broadcast though our FPGA array controller does currently limit Kestrel's speed. The solution to this problem, as to the problem of designing 1 GHz SIMD arrays, is pipelining. Unlike serial code, the instruction sequence for most SIMD applications has little dependence on the data being processed. Serial machines skip over conditional code that is not needed. With 512 (or thousands or millions of) processing elements, all alternatives of a conditional section are broadcast. Thus, as with vector processors, a high-performance SIMD machine with a 4-8 clock (or higher) pipelined instruction broadcast will rarely see the 4-8 cycle stall.

Our experience with the system has led to several thoughts on how to improve the design for better performance:

- Onboard memory would significantly aid the computational chemistry and other applications.
- Increasing PE memory, for example, with a small 4KB DRAM, could aid many applications, such as object recognition and tracking.
- Adding a faster I/O bus throughout the array would speed nonsystolic applications.

- Increasing word size to 16-bits would improve computational performance and computational density on most applications.

4.1 Conclusion

We envisioned Kestrel as a single-board parallel processor, able to perform its target applications as fast as a single-purpose accelerator, and also able to be intuitively programmed. The design enabled exploration of the landscape between the flexible and easily programmable systems of Category 4 and the flexible but hard-to-program FPGA-based systems of Category 2. In the end, we found Kestrel to be not so much a programmable sequence analysis engine, but a small-scale massively parallel processor.

A decade after the design of Kestrel, it is possible to place a large SIMD array and controller, host processor, and instruction and data memories on a single chip. Many of the issues we have identified in the design and use of our single-board Kestrel system would exist in the construction of a single-chip Kestrel system. A careful design with pipelined instruction distribution, higher-bandwidth data connections and, above all, an emphasis on simplicity and programmability, would provide extraordinary computational power within a single chip. We hope that our experiences will inform designers to come about the effective use of hardware and software resources.

Acknowledgments

The authors thank Ken Kennedy for valuable discussions and encouraging comments on several versions of this paper. This work was supported in part by US National Science Foundation (NSF) grants MIP-9423985 and EIA-9905322 and their REU supplements, NSF REU Site grant NSF EIA-0244016, a grant from the Affymax Research Institute, and seed funding from the University of California at Santa Cruz. The authors 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, Brian Feaster, David Fulton, Jason Guilford, 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 Manju Anand, Elizabeth Avila, Cyrus Bazeghi, Kaushik Narayanun, Goeff Ryder, Gang Wang, and Doug Williams.

References

[1] J.D. Hirschberg, D. Dahle, K. Karplus, D. Speck, and R. Hughey, "Kestrel: A Programmable Array for Sequence Analysis," *J. VLSI Signal Processing,* vol. 19, pp. 115-126, 1998.

[2] R. Hughey, "Parallel Sequence Comparison and Alignment," *Proc. CABIOS Conf.,* vol. 12, no. 6, pp. 473-479, 1996.

[3] 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," *J. Molecular Biology,* vol. 284, no. 4, pp. 1201-1210, 1998.

[4] Time Logic Inc., Decypher II Product Literature, http://www.timelogic.com, 2002.

[5] B. Schmidt, H. Schroder, and M. Schimmler, "Massively Parallel Solutions for Molecular Sequence Analysis," *Proc. Int'l Parallel and Distributed Processing Symp.,* pp. 186-192, Apr. 2002.

[6] 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.

[7] M. Gokhale et al., "Processing in Memory: The Terasys Massively Parallel PIM Array," *Computer,* vol. 28, pp. 23-31, Apr. 1995.

[8] J. Frigo, M. Gokhale, and D. Lavenier, "Evaluation of the Streams-C C-To-FPGA Compiler: An Applications Perspective," *Proc. ACM/SIGDA Ninth Int'l Symp. Field Programmable Gate Arrays,* pp. 134-140, 2001.

[9] C. Ebeling, D.C. Conquist, and P. Franklin, "Rapid—Reconfigurable Pipelined Datapath," *Proc. Sixth Int'l Workshop Field-Programmable Logic and Applications,* pp. 126-135, 1996.

[10] *The Massively Parallel Processor,* J.L. Potter, ed., Cambridge, Mass.: MIT Press, 1985.

[11] L.W. Tucker and G.G. Robertson, "Architecture and Applications of the Connection Machine," *Computer,* vol. 21, pp. 26-38, Aug. 1988.

[12] J.R. Nickolls, "The Design of the Maspar MP-1: A Cost Effective Massively Parallel Computer," *Proc. COMPCON Conf. Spring 1990,* pp. 25-28, Feb. 1990.

[13] K. Hwang and Z. Xu, *Scalable Parallel Computing.* New York: McGraw-Hill Book Co., 1998.

[14] D.E. Culler and J.P. Singh, *Parallel Computer Architecture.* Los Altos, Calif.: Morgan Kaufmann Publishers, 1999.

[15] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach,* third ed., Los Altos, Calif.: Morgan Kaufmann Publishers, 2002.

[16] R. Hughey and D.P. Lopresti, "B-SYS: A 470-Processor Programmable Systolic Array," *Proc. Int'l Conf. Parallel Processing,* C. Wu, ed., vol. 1, pp. 580-583, Boca Raton, Fla.: CRC Press, Aug. 1991.

[17] 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," *Proc. 17th Conf. Advanced Research in VLSI,* pp. 145-162, Sept. 1997.

[18] D.E. Knuth, *The Art of Computer Programming,* vol. 2, Reading, Mass.: Addison-Wesley, second ed., 1981.

[19] S.F. Altshul, W. Gish, W. Miller, E. Myers, and D. Lipman, "Basic Local Alignment Search Tool," *J. Molecular Biology,* vol. 215, pp. 403-410, 1990.

[20] W. Pearson, "Comparison of Methods for Searching Protein Sequence Databases," *Protein Science,* vol. 4, pp. 1145-1160, 1995.

[21] 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," *J. Molecular Biology,* vol. 284, no. 4, pp. 1201-1210, 1998. http://www.mrc-lmb.cam.ac.uk/genomes/jong/assess_paper/assess_paperNov.html.

[22] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," *J. Molecular Biology,* vol. 147, pp. 195-197, 1981.

[23] A. Krogh, M. Brown, I.S. Mian, K. Sjölander, and D. Haussler, "Hidden Markov Models in Computational Biology: Applications to Protein Modeling," *J. Molecualar Biology,* vol. 235, pp. 1501-1531, Feb. 1994.

[24] Paracel, Inc., Genematcher2 Product Literature, http://www.paracel.com, 2001.

[25] Y. Yamaguchi and T. Maruyama, "High Speed Homology Search with Fpgas," *Proc. Pacific Symp. Biocomputing 2002,* pp. 271-282, 2002.

[26] D.S. Hirschberg, "A Linear Space Algorithm for Computing Maximal Common Subsequences," *Comm. ACM,* vol. 18, pp. 341-343, June 1975.

[27] J.A. Grice, R. Hughey, and D. Speck, "Reduced Space Sequence Alignment," *Proc. CABIOS Conf.,* vol. 13, pp. 45-53, Feb. 1997.

[28] L. Grate, M. Diekhans, D. Dahle, and R. Hughey, "Sequence Analysis with the Kestrel SIMD Parallel Processor," *Proc. Pacific Symp. Biocomputing 2001,* pp. 263-274, 2000.

[29] L.R. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proc. IEEE,* vol. 77, pp. 257-286, Feb. 1989.

[30] R. Hughey and A. Krogh, "Hidden Markov Models for Sequence Analysis: Extension and Analysis of the Basic Method," *Proc. CABIOS Conf.,* vol. 12, no. 2, pp. 95-107, 1996, http://www.cse.ucsc.edu/research/compbio/sam.html.

[31] D. Lavenier, "Speeding up Genome Computations with a Systolic Accelerator," *SIAM News,* vol. 31, no. 8, pp. 6-7, 1998.

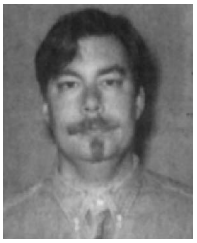
[32] E. Rice and R. Hughey, "Molecular Fingerprinting on the SIMD Parallel Processor Kestrel," *Proc. Pacific Symp. Biocomputing 2001,* pp. 323-334, 2000.
[33] P.P. Jonker, "Why Linear Processor Arrays are Better Image Processors," *Proc. Int'l Conf. Pattern Recognition (ICPR),* vol. 3, pp. 334-338, 1994.
[34] S. Kyo, S. Okazaki, Y. Fujita, and N. Yamashita, "A Parallelizing Method for Implementing Image Processing Tasks on SIMD Linear Processor Arrays," *Computer Architectures for Machine Perception,* pp. 180-184, Oct. 1997.
[35] D. Taubman and M. Marcellin, *JPEG2000—Image Compression Fundamentals, Standards and Practice.* Dordrecht, The Netherlands: Kluwer Academic Publishers, 1999.
[36] J.P. Hayes, *Computer Architecture and Organization.* New York: McGraw-Hill Book Co., 1976.
[37] E. Rice and R. Hughey, "Multiprecision Division on an 8-Bit Processor," *Proc. 13th IEEE Symp. Computer Arithmetic,* pp. 74-81, July 1997.
[38] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York: Freeman, 1979.
[39] Center for Discrete Mathematics and Theoretical Computer Science, http://dimacs.rutgers.edu, 2004.
[40] A. Di Blas, A. Jagota, and R. Hughey, "Parallel Implementations of Optimizing Neural Networks," *Proc. Artificial Neural Networks in Eng. Conf. (ANNIE 2000),* pp. 153-158, Nov. 2000.
[41] A. Di Blas, A. Jagota, and R. Hughey, "Optimizing Neural Networks on SIMD Parallel Computers," *Parallel Computing,* pending publication.
[42] A. Di Blas and R. Hughey, "Explicit SIMD Programming for Asynchronous Applications," *Proc. Int'l Conf. ASAP,* pp. 258-267, July 2000.
[43] D.M. Hawver and G.B. Adams III, "Processor Autonomy and Its Effect on Parallel Program Execution," *Proc. Symp. Frontiers of Massively Parallel Computing,* pp. 144-153, Oct. 1996.
[44] *Proc. Pacific Symp. Biocomputing 2001,* London: World Scientific, 2001.

Andrea Di Blas received the MS and PhD degrees in electrical engineering from Politecnico di Torino, Italy, in 1994 and 2000. He has been a researcher at the University of California at Santa Cruz since 1999, where he is also a lecturer with the Department of Computer Engineering. His research interests include parallel processing methodologies and applications, computer architecture, reconfigurable computing, image processing, combinatorial optimization, and neural networks. He is a member of the IEEE and the IEEE Computer Society.

David M. Dahle received the BS degree in computer engineering and physics, and the MS degree in computer engineering from the University of California at Santa Cruz. He began work on on the full-custom Kestrel chip design as an undergraduate, and the system was the focus of his Master's thesis. He is presently working on IA-64 processor design and verification at Intel.

Mark Diekhans received the bachelors degree in computer science and biology from Indiana University 1985. He is currently a staff engineer at the Center for Bimolecular Science and Engineering at the University of California at Santa Cruz, as well as a graduate student in computational biology. His work currently involves gene and cDNA clone validation and computational gene finding. Previously, he was a software engineer developing operating system and server software.

Leslie Grate received the PhD degree in computer engineering in 2000 from the University of California at Santa Cruz. He is presently a comptuational biologist at Lawrence Berkeley National Laboratory. His work on the Kestrel project included the systems fully operational and making the sequence analysis algorithms as fast as possible.



Jeffrey Hirschberg received the BS degree in computer engineering from the University of Washington. He received the MS degree in computer engineering from the University of California at Santa Cruz. His graduate research focused on the development of Kestrel, an SIMD processor for DNA and protein analysis. He currently works for Intel Corporation.

Kevin Karplus received the BS degree in mathematics from Michigan State University, and the MS degree in mathematics and PhD degree in computer science from Stanford University. He is now a professor in the Biomolecular Engineering Department at the University of California at Santa Cruz (recently moved from the Computer Engineering Department). He is undergraduate and graduate director for bioinformatics at UCSC. His main research interest is the prediction of protein structure from amino acid sequences. He is a senior member of the IEEE and the IEEE Computer Society, and a member of the International Society for Computational Biology (ISCB).

Hansjörg Keller received the PhD degree in applied physics from the University of Berne, Switzerland. He joined the Kestrel group for a year-long sabbatical from the faculty of the Berne University of Applied Sciences, Switzerland. While at the University of California at Santa Cruz, he developed the array controller and several applications.

Mark Kendrick received the BS degree in computer engineering and physics from the University of California at Santa Cruz. He is currently in a staff research position working on the next generation Kestrel system, Kestrel2. His work is mainly focused on the low-level system design and printed circuit board design. He will begin graduate studies in physics in Fall 2004.

Francisco J. Mesa-Martinez is a PhD candidate in the Computer Engineering Department at the University of California at Santa Cruz. His research interests include multithreaded processor architectures, parallel processing, and reconfigurable computing. Additional research involvement includes autonomous distributed robotic systems.

David Pease is a lecturer and is studying for the PhD degree in the Computer Engineering and Computer Science Departments at the University of California at Santa Cruz. He is also a senior technical staff member of computer science at the IBM Almaden Research Center in San Jose, California. Besides his work on Kestrel, he has recently led IBM's Storage Tank distributed file system research project, and is currently investigating autonomic management of storage subsystems.

Eric Rice received the BS degree in computer engineering from the University of California at Santa Cruz, where he is working on the PhD degree. He joined the project as an undergraduate working on algorithms for division in software on Kestrel. His graduate work concerns hardware algorithms to accelerate division.

Angela Schultz is a graduate student in the Department of Computer Engineering at the University of California at Santa Cruz. Her academic interests include reconfiguable computing, parallel architectures, and applications of FPGA-based systems. She is a member of the IEEE, ACM, and SWE.

Don Speck received the BS degree in engineering and applied science from the California Institute of Technology and the MS degree in computer engineering at the University of California at Santa Cruz. Prior to Kestrel, he spent six years at the California Institute of Technology working on the Mosaic multicomputer project, for which he designed the DRAM. He is presently an engineer with Synaptics in San Jose, CA, where he designs mixed-signal VLSI chips for touchpads.

Richard Hughey is a professor and chair of the Computer Engineering Department at the University of California at Santa Cruz (UCSC) and is also affiliated with the Department of Biomolecular Engineering. He received the BS degree in mathematics and the BA degree in engineering from Swarthmore College, Pennsylvania, and the ScM and PhD degrees in computer science from Brown University. His research includes bioinformatics and parallel processing and, especially, the combination of the two as in the Kestrel project. His additional interests include academic program planning and development. He is a senior member of the IEEE and the IEEE Computer Society, and a member of ASEE, ISCB, and the Society of Women Engineers, in which he serves as faculty advisor to the UCSC student chapter.