Proc. 17th Conf. on Advanced Research in VLSI, September 15–17, 1997. ©IEEE CS

Kestrel: Design of an 8-bit SIMD parallel processor

David M. Dahle Jeffrey D. Hirschberg Kevin Karplus Hansjörg Keller
Eric Rice Don Speck Douglas H. Williams Richard Hughey

Department of Computer Engineering
Jack Baskin School of Engineering
University of California, Santa Cruz, CA 95064
rph@cse.ucsc.edu

Abstract

Kestrel is a high-performance programmable parallel co-processor. Its design is the result of examination and reexamination of algorithmic, architectural, packaging, and silicon design issues, and the interrelations between them. The final system features a linear array of 8-bit processing elements, each with local memory, an arithmetic logic unit (ALU), a multiplier, and other functional units. Sixty-four Kestrel processing elements fit in a 1.4 million transistor, 60 mm^2, 0.5 μm CMOS chip with just 84 pins. The planned single-board, 8-chip system will, for some applications, provide supercomputer performance at a fraction of the cost.

This paper surveys four of our applications (sequence analysis, neural networks, image compression, and floating-point arithmetic), and discusses the philosophy behind many of the design decisions. We present the processing element and system architectures, emphasizing the ALU and comparator's compact instruction encoding and design, the architecture's facility with nested conditionals, and the multiplier's flexibility in performing multiprecision operations. Finally, we discuss the implementation and performance of the Kestrel test chips.

1: Introduction

Kestrel is a simple yet powerful VLSI parallel processor. The focus of our initial design effort was the sequence analysis applications from computational biology. As more and more DNA, RNA, and protein molecules are sequenced into their constituent nucleotides or amino acids (essentially, alphabets with four or twenty characters), the need for high-speed sequence analysis is becoming more and more urgent. The Genbank database, for example, has over one million entries with over 700 million nucleotides [1].

Because computational biology (and indeed any field) is constantly discovering new analysis methods, programmability is a key issue in designing a system that can serve the needs of this vast community. Thus, Kestrel has been designed for use with a host of different methods, from the standard Smith and Waterman algorithm [37] to hidden Markov models [23], with many variations in between and yet to come.

Our goal, however, has never been to build just a sequence analysis machine. Once the requirement for programmability in a sequence analysis machine is realized, it becomes obvious that the entire machine should have a general purpose slant, usable for a great variety of applications.

This work was supported in part by NSF grant MIP-9423985 and its REU supplement. Keller was supported in part by a sabbatical leave grant from the State of Berne, Switzerland. Dahle and Rice were supported in part by ARCS Foundation fellowships. The work was initially funded by the UCSC Division of Natural Sciences. This work made use of equipment purchased under NSF grants CDA-9022505 (IMS chip tester) and CDA-9115268 (MasPar parallel computer). http://www.cse.ucsc.edu/research/kestrel

Thus, our design process has focussed not on designing and implementing a high-speed dynamic programming engine (a problem for which there are many existing solutions), but on building a multi-purpose machine that can accelerate many different algorithms. For this reason, we have made several parts of the Kestrel system more general than required for simple sequence analysis, and have also included an 8×8 multiplier.

The resulting machine is a horizontally-microcoded, fine-grain parallel processor. It is a linear array of 8-bit processing elements, each with its own local memory, arithmetic logic unit (ALU), and multiplier, among other features. The first Kestrel system is targeted to have 512 processing elements on a single PCI board operating at 33 MHz.

Since freezing the major components of the architecture, we have had a chance to turn to applications beyond our development targets. Our recent studies of several very different problems have led to a few tunings of the architecture that have greatly enhanced performance in these new domains.

The remaining sections of this paper discuss Kestrel's system architecture, several applications, Kestrel's processing element (PE) architecture, the design of three of the more interesting parts of the Kestrel PE, the results of our test chip fabrications, and a brief review of comparable architectures.

The Kestrel design process has always been a tightly integrated examination and co-design of algorithms, system and PE architecture, and very large scale integration (VLSI) implementation. Indeed, it is impossible to separate these issues in the construction of a high-performance parallel processor. As the following sections attempt not just to describe Kestrel, but to explain the philosophy that led to Kestrel, there is a necessary interweaving of these topics, not all of which can be eliminated from this sequential medium.

2: Kestrel System Architecture

The general goals for the Kestrel system included programmability for flexibility, small size and low cost to enable use by individual computational biologists and other researchers; and a simple interface to standard personal computers, workstations, and disks.

The Kestrel design, and that of any single instruction stream, multiple data stream (SIMD) processor, is a continual compromise between the processor array's four forms of interaction with the outside world: data input and output (I/O), inter-PE communication, instruction broadcast, and cooperation with the host computer.

2.1: Data Input and Output

Kestrel is primarily a linear array of processing elements. There are two reasons for this. First, linear arrays are the natural choice for the target sequence analysis applications (Section 3.1). Second, linear arrays have the simplest I/O requirements of any topology: two connections for each array, processor, or chip. As a result of this simple topology, we are able to fit 64 Kestrel processing elements in an 84-pin package, providing considerable savings in board area for our small PCI co-processor system.

Data input and output has also affected the design of the processing elements. In the sequence analysis domain, for example, one can easily conceive of a special-purpose design to implement the appropriate computation. Such a design, including several adders and comparators, could process characters at a high rate. This would seem to be good, except on the realization that such a cycle time may outpace the disk storing the database.

There are at least three solutions to this problem of high-speed processing elements. First, a high performance storage system could become an integral part of the system. This would make a fast and balanced parallel processor, but would also significantly raise the cost. Such storage systems are more in the domain of general-purpose supercomputers than cost-effective parallel co-processors.

Second, for many applications, data buffering and recirculating among a small number of high-speed processors can be an effective solution. This idea has been adapted by a number of special-purpose machines [4, 24, 40].

Third, programmable processing elements that require several instructions to compute the basic function can be used. This solution, used by Kestrel, essentially allows the addition of programmability into the processing elements with little penalty over the single-purpose system. If, for example, data can be retrieved from disk at a sustained rate of 1–5 Mbyte per second and 33 million instructions can be processed per second, then a balanced application will execute at least 6–33 instructions per byte of data.

The inner loop of one common form of sequence analysis requires 21 instructions on Kestrel. Our target single board, 512-PE system running at 33 MHz will perform this application at a speed similar to the vastly more expensive 16 384-processing element Maspar MP-2 [2, 15].

2.2: Instruction broadcast

SIMD instruction broadcast can be slow, especially with many processing elements. The typical SIMD instruction memory is centralized for all processing elements to eliminate the overhead of local control. Sending an instruction to the array requires a broadcast of the instruction to each board, each chip, and each processing element within each chip, every clock cycle. The broadcast time can easily exceed processing time even for complex instructions.

One solution to the instruction broadcast bottleneck is to partially or completely eliminate instruction broadcast. This can be done in several ways, each with its own cost. A multiple instruction stream, multiple data stream (MIMD) machine, obviously, has no instruction broadcast (the program may be broadcast initially) with the penalty of local instruction memories and sequencers, not appropriate for the small and simple processing elements that we are proposing. A reconfigurable machine can be thought of as a SIMD machine with a single exceedingly long instruction that is executed many times [3, 41, 11]. Unfortunately, this great flexibility can cause programming to be more difficult than on a true SIMD machine, and can cause a much lower ratio of processing power to area.

Alternatively, instructions can be cached at the PE (for a MIMD machine), chip, or board level. With, for example, per-chip instruction caching, the programming model can either be SIMD or multiple-SIMD (MSIMD). For SIMD control, synchronization between the chips must still be broadcast. For MSIMD control, communication between chips will require queues or some other form of synchronization, and may degrade overall performance. Thus, we chose board-level instruction issue and broadcast.

Because instruction broadcast can take a fair amount of time, it is important to match the broadcast time with the instruction processing time. Thus, an architecture should do many things during each instruction broadcast, reducing the rate at which instructions are required. A processing element must be complex enough to be able to do useful work during the cycle time of instruction broadcast, much the same way it was necessary to balance cell program length and data I/O requirements. Our initial estimates of maximum broadcast speed were 40–50 MHz, thus we were interested in designing a processor that could process instructions at approximately this rate, and in designing an instruction set that packed as much concurrent operation as feasible and usable.

The Kestrel processing element, described in more detail below, has a number of independent functional units. Control of these units requires a 52-bit instruction broadcast to all the processing element chips each clock cycle. This instruction includes three register addresses, an 8-bit immediate, and control fields for the functional units. In a single instruction, Kestrel can perform an ALU operation, comparison, result selection, storing of the selector bit, address computation, memory access, and communication. All of these capabilities are used concurrently in our sequence comparison program's inner loop.

2.3: Inter-PE communication

In systolic algorithms, data moves between PEs as partial results are calculated. Sometimes values are required by multiple PEs to compute a result. Both of these are true for the dynamic programming solutions to sequence analysis. The ability to create a fine-grain pipeline is key

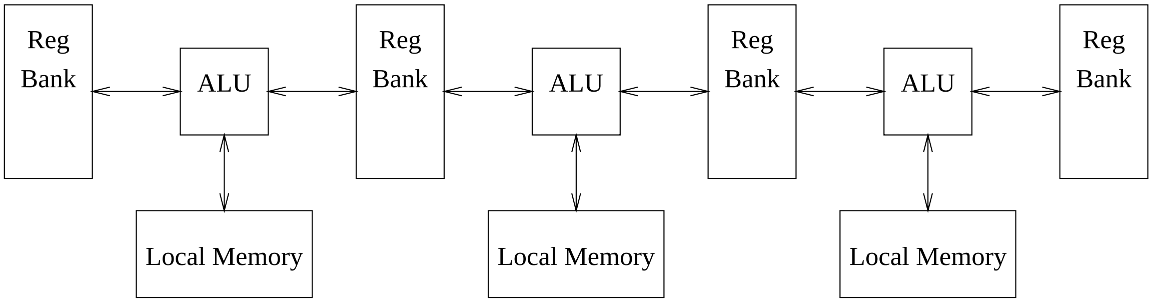


Figure 1. Linear systolic shared register machine.

to making systolic arrays an effective solution. Thus, a good inter-PE communication scheme is essential for any array processor.

Kestrel employs Systolic Shared Registers (SSRs) for inter-PE communication (Figure 1). Rather than being local to each PE, as register files usually are, SSRs reside between PEs. This allows neighboring PEs to share a register file.

With shared registers, communication and computation do not require distinct instructions but occur concurrently. When a result is stored after an instruction is executed, communication automatically takes place. The programmer can naturally think about data streaming through the array as values are computed. To pass a value from one PE to the next using SSRs, all that is necessary is to store the value in a register file. For example, if each PE is calculating a value to be used by the PE to its right, then each PE stores its result into the register file on its right. During a subsequent instruction, each PE can read from its left register file to get the previously-stored value. Because all addresses for these register files are issued globally, adjacent PEs never write to the same register bank. The first machine to include SSRs was B-SYS [16], though two earlier machines included shared memory chips, one with a port for each processing element [19], the other with a combination of asynchronous local operation and synchronized shared memory operation to eliminate contention problems [6]. SSRs make for an elegant programming model and provide a low-overhead solution to the inter-PE communication problem—the cost is one bit per register address.

2.4: Instruction sequencing

The final part of co-processor design is the instruction sequencer and interface to the host workstation. For the first Kestrel machine, we are building a simple board interface (Figure 2). Instructions are stored in a local memory, while data transfers to and from the host uses queues to simplify addressing. The instruction sequencer is programmed using a 12-bit field appended to each array instruction, making the total instruction width a convenient 64 bits. Three of these bits have specific and constant meanings: whether or not data is read from the input queue, whether or not data is written to the output queue, and whether or not the array instruction's immediate field should be replaced with data from the controller. The remaining nine bits enable branches, jumps, calls, returns, and host interrupts to be performed concurrently with array operations. Kestrel effectively has 0-cycle jump and branch instructions. The controller has a small amount of local data and arithmetic circuitry to enable loop counting and the recirculation of data in the processor array.

3: Kestrel Applications

After the primary components of the architecture were solidified, we turned our thoughts to applications beyond the initial domain of sequence analysis. This application analysis led us to several changes in the processor design, such as the addition of the multiplier.

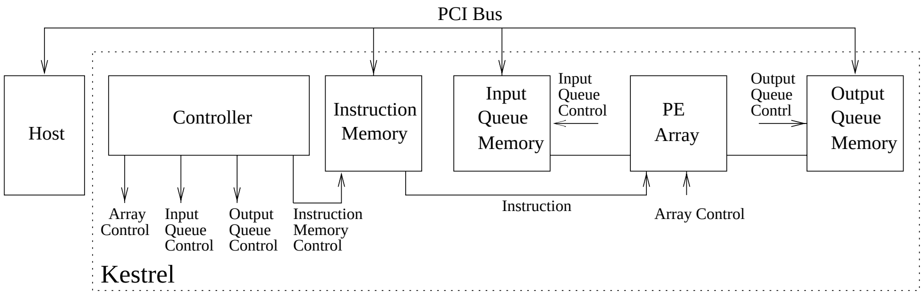


Figure 2. The Kestrel co-processor.

The following applications refined and exploit the special features of Kestrel, such as conditional execution, zero overhead for loop control, and multiprecision arithmetic operations. Because of limited local memory, the applications trade memory usage for a number of instructions wherever possible. The shared register banks allow easy linear data pipelining in both directions.

3.1: Sequence Analysis

Many sequence analysis techniques rely on aligning sequences in the database to a model or to other sequences, often with a variant of the following edit-distance computation. Given two sequences of characters, a and b, dynamic programming determines a total cost to transform one sequence into the other through three basic operations: deletion of a character, insertion of a character, and mutation of a character. The following dynamic programming recurrences are used to compute "edit distance":

$$c_{i,j} \quad = \quad \min \begin{cases} c_{i-1,j-1} & + \quad \mathrm{dist}(a_i, b_j) \quad \text{match} \\ c_{i-1,j} & + \quad \mathrm{dist}(a_i, \phi) \quad \text{insert} \\ c_{i,j-1} & + \quad \mathrm{dist}(\phi, b_j) \quad \text{delete,} \end{cases}$$

where $\mathrm{dist}(a_i, b_j)$ is the cost of matching a_i to b_j, $\mathrm{dist}(a_i, \phi)$ is the gap cost of not matching a_i to any character in b, and $\mathrm{dist}(\phi, b_j)$ is the gap cost of not matching b_j to any character in a. Sequence comparison with affine gap penalties and various other features, greatly preferred by biologists, involves three interconnected recurrences of a similar form [29, 35, 37].

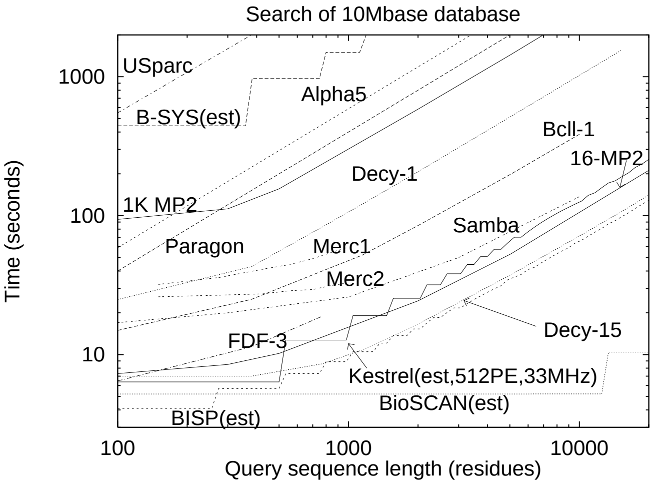
The dynamic programming calculation easily maps to a linear array of processing elements [15]. A common mapping is to assign one PE to each character of the query string, and then to shift the database through the linear chain of PEs. Typical query strings are hundreds to thousands of characters, matching Kestrel's 512-PEs, though longer queries require storing several adjacent characters in each processing element's local memory (i.e., a virtual processor ratio greater than one).

Architecturally, sequence comparison can be aided by multiprecision arithmetic (the dist values can be at a lower precision than the c values), modulo arithmetic, minimization, and addition. The twin problem of sequence alignment, finding the minimizing correspondence between two sequences, requires the saving of the selector bits of the minimizations and recirculation of sequence data [13].

Kestrel's single-board sequence analysis performance is expected to be comparable to both a 16 384-processor MasPar computer and a 15-board FPGA-based system (Figure 3). The one system expected to be significantly faster than Kestrel is the BioSCAN machine, a single-purpose machine that calculates scores for ungapped alignment segments ($c_{i,j} = c_{i-1,j-1} + \mathrm{dist}\, a_i, b_j$), and then performs statistical post-processing to gauge similarity [36]. Further details on parallel hardware for sequence analysis can be found in two comparative studies [15, 39].

3.2: Neural Networks

Neural networks (NNs) [28] have become an tool for pattern recognition and classification tasks. The standard NN configuration consists of 3 layers (input, hidden, and output), each containing a



Machines: BISP (single purpose, not completed) [5], BioSCAN (single purpose) [36], 15-board Decypher-II (FPGA, $173k) [40], 5-board FDF (single purpose, $50k), 16,536-PE MasPar, [34], SAMBA (single purpose) [24], 1- and 2-board Mercury (single purpose) [4], Biocellerator (FPGA, about $50k) [7], 1-board Decypher-II ($18k), 32-node Paragon [31], 1024-PE MasPar, 5 DEC Alpha 3000 workstations [31], B-SYS [16], and Sun Ultrasparc 140.

Figure 3. Database search time as a function of protein query length.

number of nodes that receive all components of the output vector from the previous layer. Each node computes a weighted sum of its inputs, adds a bias, and normalizes the output by the so-called 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 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.

NNs are well-suited for Kestrel, for both training and classification. The PEs implement just the hidden and output layers, as data pipelining allows direct distribution of the 8-bit input vector components to the complete hidden layer. After initialization and loading of the 16-bit weight and bias data, a neural net used as a classifier can accept a continuous stream of input vectors as a systolic pipeline and produce output vectors in overlapping shift operations. We found that the fastest implementation shifts input data serially through all nodes of a layer, which then do local multiply and accumulate (MAC) operations using the stored weight factors. At the end of such a cycle the nodes calculate the activation function and write the result back into the pipeline of registers. The sigmoidal activation function accepts the 24-bit wide input of the accumulator and yields an 8-bit output in 20 clock cycles.

If n_1 denotes the number of input vector components, n_2 and n_3 the number of nodes in the hidden and output layers, performance is limited by $\max(n_1, n_2, n_3)$. Assuming $n_1 = 40$ and full PE usage on a single chip at $n_2 = n_3 = 32$, 76 040 vectors per second can be entered at a data rate of 3.04×10^6 byte/s. This yields a figure of merit of 147×10^6 interconnections/s for one chip. Figure 4 shows decision and input data rates as functions of the vector size and the maximum number of nodes in a layer, respectively. This performance is equal to the performance range of commercial, single-purpose NN chips [25].

Training of a neural network by backpropagation uses pipelines of registers in both directions. The input vectors and the intermediate outputs are stored in each node for later use, so that the forward part is split into a transfer and store phase followed by the local MAC operations. Only a few instructions are required to adjust the weights of the output layer. Changing the weights of one hidden layer node however is computationally much more intensive because each output node yields its own contribution. It seems best to accumulate partial results and shift them backwards from the output to the hidden layer requiring $n_2 + n_3$ MACs. After multiplication with the derivative of the activation function, the weights can be adjusted for all nodes in $\max(n_1, n_2, n_3)$ steps.

Training is obviously slower than classification and the dependence on n_1, n_2 and n_3 is also more complex. We estimate that the maximal example cited above allows a training input of 18 000 vectors per second and this number increases to about 50 000 v/s for a more typical case of $n_1 = 8$,

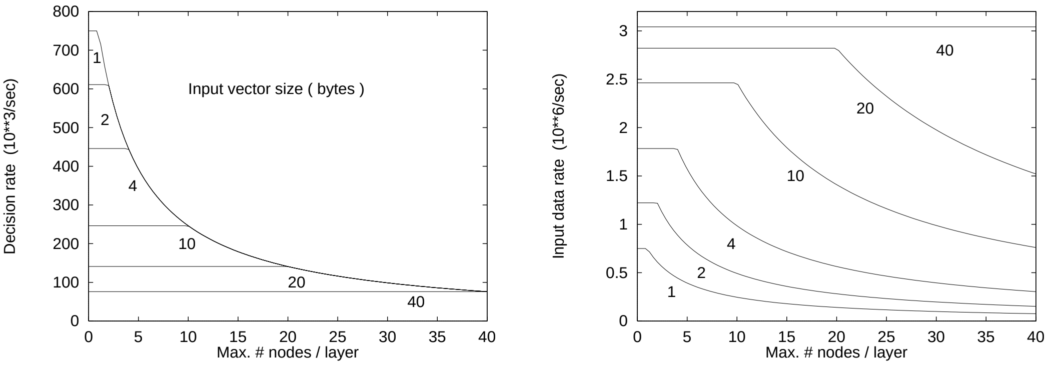


Figure 4. Decision (left) and input (right) rates of a neural network classifier.

$n_2 = 12$, and $n_3 = 6$.

3.3: Discrete Cosine Transform

The discrete cosine transform (DCT) and its inverse (IDCT) [18], as part of video compression and decompression standards, process 8×8 arrays either as 8-bit pixels $I(x, y)$ or as 12-bit to 14-bit coefficients $a(u, v)$. Real-time video compression and decompression leads to a continuous stream of DCT and IDCT operations which can account for 25% to 50% of CPU usage unless dedicated hardware is available. The DCT (and IDCT) resemble the discrete 2-D Fourier transforms but remain in the domain of real numbers using cosine terms as coefficients:

$$a(u, v) = \frac{1}{4} C(u) C(v) \sum_{x=0}^{7} \sum_{y=0}^{7} I(x, y) \cos(\pi(2x + 1)\frac{u}{16}) \cos(\pi(2y + 1)\frac{v}{16})$$

$$C(x) = 1/\sqrt{2} \text{ when } x = 0, \;\; C(x) = 1 \text{ otherwise.}$$

Both the DCT and IDCT are separable into sequential 1-D transforms. According to the equation, each element would need 64 multiply and accumulate operations, but more efficient algorithms similar to FFT exist. Our implementation was derived from the Telenor H.263 software distribution [38] and averages 9.5 additions, 6 multiplications and 2 scaling operations per pixel.

It would be tempting to implement the DCT in pipelined fashion, shifting in a pixel stream and doing the MAC operations on the coefficients while shifting them toward the output. A careful analysis shows considerable overhead for this solution due to more costly internal shifts and staggered PE reinitializations. Separate phases of shifting data in and out followed by a full DCT per PE are more efficient and allow coding of the coefficients as immediate operands. Taking advantage of Kestrel's multiprecision multiply and accumulate instructions, a full DCT executes in 3768 clock cycles and the associated I/O requires 8234 cycles, leading to a rate of 176 kDCT/s. A single 64-PE chip running at 33 MHz is thus capable of handling an input rate of 11.26×10^6 pixel/sec at a maximum latency of $364\,\mu$s. There is a potential for considerable speedup with future changes of the I/O architecture for higher bandwidth and background PE to PE communication.

3.4: Numerical Algorithms

Although not designed for floating-point computation, we have extensively studied the application of the Kestrel PE to floating-point arithmetic, both single and double precision, to ensure future applications would not be overly penalized by Kestrel's 8-bit architecture.

Floating-point addition and multiplication are implemented in the obvious way. In the case of addition, the multiplier is used to perform shifts. For division, we have optimized the use of

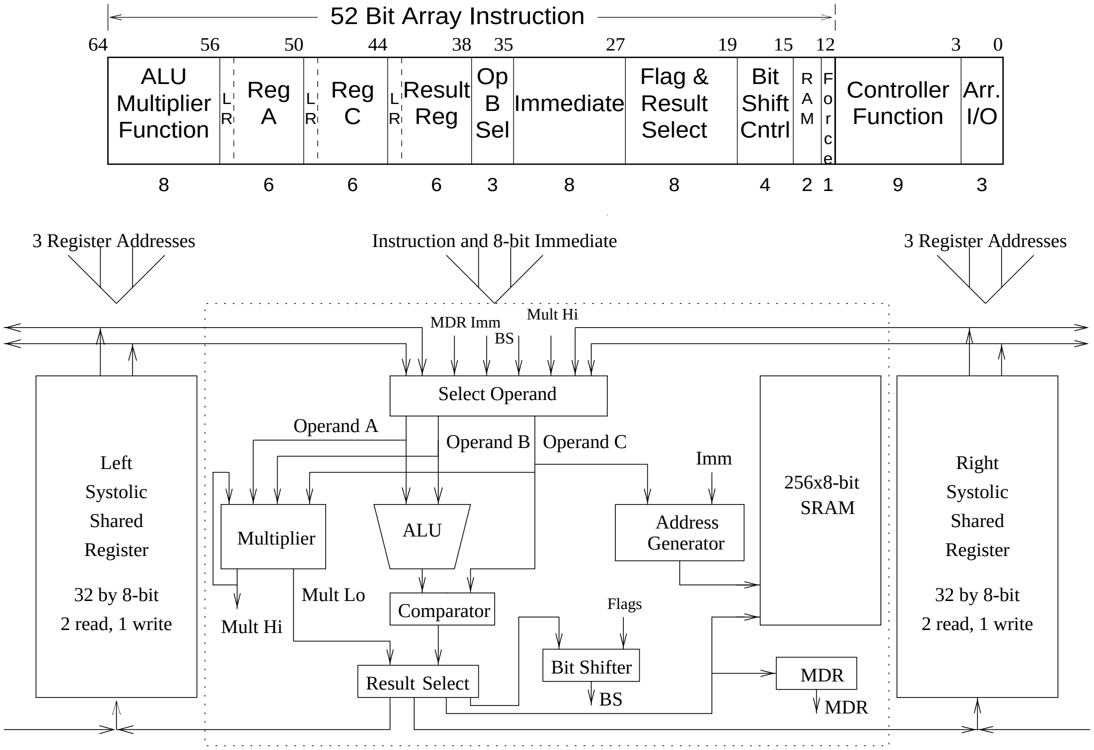


Figure 5. The Kestrel horizontal microcode format and processing element.

Newton-Raphson iterations, in which an initial reciprocal approximation to a/b, r_0, is refined using equations such as $r_{i+1} = r_i[2 - br_i]$, for use with Kestrel's 8×8 multiplier. Among other techniques, we save steps by using low precision arithmetic in the early stages of iteration [32].

The 512 PE, 33 MHz system will be able to perform single-precision addition, multiplication, and division at 160, 460, and 260 million floating-point operations per second (MFLOPS), and the double-precision variants at 65, 150, and 100 MFLOPS.

4: Kestrel Processing Element Architecture

The Kestrel PE architecture is the result of a number of requirements and goals. The PEs had to be fast and small to result in an inexpensive system. The PEs had to be flexible to allow for the development of new algorithms.

The Kestrel PE and its horizontal microcode format are shown in Figure 5, the components of which are described below. Each PE can operate on up to three operands per instruction: Operand A, Operand B, and Operand C. Operand A and Operand C are two independently selected registers. Operand B can come from the multiplier, the bit shifter, memory data register (MDR), the same register as Operand C, or a globally-issued immediate value. To aid in multiprecision operations when operands differ in length, Operand B can also be the sign extension of Operand C, the MDR, or MultHi (the high byte of a previously-computed multiplier result).

The relative sizes of several of the Kestrel PE functional units, as well as their usefulness to various applications, are displayed in Table 1. The ALU, comparator, and local SRAM access are highly tuned for sequence analysis, while the multiplier aids numerical calculations and multi-bit shifting.

SSR Kestrel's SSRs are 32×8-bit, with two read ports and one write port. This port configuration provides good balance between speed and area. One goal of Kestrel was to make all instructions execute in a single cycle. Since instructions commonly use two registers for operands, two read ports

Application	Kestrel	Slowdown without			
	Instr	compare	SRAM	bit shift	mult
Edit distance	3	1.7			
16-bit Affine	21	1.7	2.0		
16-bit Alignment	30	1.5	10	1.3	
16-bit multiplication	5				20
16-bit division	57	1.1		1.1	2.2
SP multiplication	36				8.3
SP division	64	1.1		1.1	6.0
8×8 DCT	3768		∞		6.4
Neural Network 8/12/6	164		∞	1.6	3.1
Portion of Area		4%	48%	6%	18%

Table 1. Kestrel cell program times and slowdown with loss of each feature.

are essential. Having a third read port would have been useful but costly in terms of area and access time. With proper instruction scheduling, the MDR or the immediate value can often be used as a third operand source without any time penalty compared to a register file with three read ports.

SRAM Each PE contains a 256×8-bit local static random access memory (SRAM). The SRAM supplements the storage and bandwidth of the SSRs and provides locally-addressable storage. The SRAM contains a single bidirectional port to conserve area, so an instruction may read a value into the MDR or write a value, but not both. The MDR alleviates the potential bottleneck of a single port because values can be read several instructions prior to use.

Each PE has an address generator and decoder for its SRAM. The SRAM has two addressing modes: absolute and indexed. For absolute addressing, the address is just the 8-bit immediate specified in the instruction. For indexed addressing the immediate is added to Operand C (typically meaning that a concurrent ALU or multiplier operation must use a different source for Operand B).

The size of the SRAM is based on three factors. First, 256 bytes will allow the 512-PE Kestrel to process protein hidden Markov models of length 3000, and DNA models of about 10000 positions without host partitioning (simpler analysis methods can handle much longer sequences). Second, 256 bytes is a natural choice for a PE with 8-bit local addressing. Third, the SRAM size helps keep the PE small, enabling a high density of processing elements.

ALU The ALU operates on Operand A and Operand B and a carry in to produce its result. The carry in can be specified as part of the instruction, or it can come from a latch that holds a previously computed carry out. The ALU is discussed in more detail in Section 5.1.

Bit Shifter The bit shifter is an 8-bit loadable shift register that is capable of shifting left or right by one bit. The bit shifter serves two purposes: it can be used for data manipulation and for conditional processing. The bit shifter is discussed in more detail in Section 5.2.

Multiplier The multiplier multiplies Operand A and Operand B to produce a 16-bit result split into two bytes, MultLo and MultHi. The lower byte is treated as the result of the current instruction, while the higher byte is stored in the MultHi register for future use if needed. The multiplier is discussed in more detail in Section 5.3.

Comparator The comparator compares the output of the ALU with Operand C, and the minimum or maximum can be selected as the result of instruction execution. This is done by subtracting Operand C from the output of the ALU. The subtraction produces three flags, the borrow-out from the subtraction, the most significant bit (msb) of the subtraction, and the true sign of the subtraction (the sign that would be produced had the operands been sign-extended to nine bits, similar to a feature in the Intel i860). The three flags allow for three types of comparison: unsigned comparison (borrow-out), modulo 256 comparison (msb), and signed comparison (true sign).

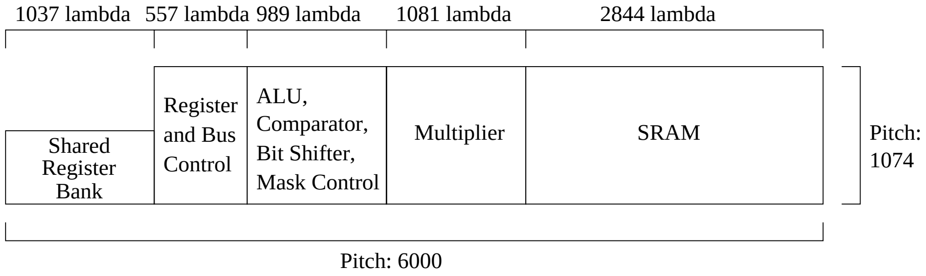


Figure 6. Floor plan of the Kestrel PE.

The comparator is an example of Kestrel's facility with multiprecision operations. With the aid of an equality test, multiprecision comparison is done top-down by byte, saving several cycles over the standard two-step process of a multiprecision subtraction followed by a multiprecision selection.

Result Selector The result selector chooses the one-byte result of executing an instruction. For multiply instructions, the result is always the low byte of the multiplication. Otherwise, the result is either the output of the ALU or Operand C. The result can be forced to one or the other of these values, or it can be chosen by a flag from the ALU, comparator, bit shifter, or a neighboring PE's bit shifter.

5: Kestrel Design

Kestrel's full custom VLSI layout was done using the Magic tool suite [26]. With hindsight, a combination of custom layout of the PEs and standard cell for the global logic may have saved some time without incurring high area or power costs. It is only the regularity of the design that enabled the hand layout of 1.4 million transistors.

The designs use the scalable submicron rules, and we have implemented test chips using $2\,\mu$m and $0.8\,\mu$m CMOS processes using the MOS Implementation Service (MOSIS). The final 64-PE chip will have a $0.5\,\mu$m feature size. Layout verification was performed with a functional simulator programmed on a MasPar parallel processor, a Verilog model, and irsim (part of the Magic suite).

Kestrel uses a two phase non-overlapping clock. Most arithmetic operations occur during phase one. The SRAM access and register write of the result takes place during phase two. This allows ample time for inter-chip writes to the shadow register bank (at chip boundaries, there are two adjacent and coherent register banks to minimize communication). As register writes can be disabled by the writing PE, mask setup at the register bank must be complete by the beginning of phase two. This means that inter-chip communication occurs during both phases.

Figure 6 shows a floor plan of the Kestrel PE with dimensions in scalable λ units, where λ is approximately one half the feature size of the CMOS process [27]. The height of the register bank is slightly less than half the height of the PE, so the register banks from the adjacent column can be fit into the extra space by turning the adjacent column sideways, reducing the horizontal pitch of the PE by about $500\,\lambda$. The main constraint on the layout was the vertical pitch of the PE, set by the SRAM, and the need to efficiently route local buses and global control signals. Over half of the PE's area and two thirds of the PE's transistors are used by the SRAM and register bank.

In the next sections, we take a close look at three of the most interesting Kestrel components: the integrated ALU and comparator, the bit shifter, and the multiplier.

5.1: ALU and Comparator

The bit-slice ALU is composed of a carry chain and logic for result bit calculation. The carry chain uses generate (G) and kill (K) signals, each a 4-bit lookup table indexed by the bits of the operand. The result bit for each slice is simply the exclusive OR of the carry input and the propagate

F Code	$\mathsf{F_4F_3 = 00}$		$\mathsf{F_4F_3 = 01}$		$\mathsf{F_4F_3 = 10}$		$\mathsf{F_4F_3 = 11}$	
$\mathsf{F_2F_1F_0}$	$\mathsf{C_{in} = 0}$	$\mathsf{C_{in} = 1}$	$\mathsf{C_{in} = 0}$	$\mathsf{C_{in} = 1}$	$\mathsf{C_{in} = 0}$	$\mathsf{C_{in} = 1}$	$\mathsf{C_{in} = 0}$	$\mathsf{C_{in} = 1}$
	Increment Functions		Increment Functions		Addition Functions		Decrement Functions	
000	-1	zero	$\overline{\mathsf{A \wedge B}}$	$-(\mathsf{A \wedge B})$	-1	zero	-1	zero
001	$\mathsf{A \vee B}$	$(\mathsf{A \vee B}) + 1$	$\mathsf{A \oplus B}$	$(\mathsf{A \oplus B}) + 1$	$\mathsf{A + B}$	$\mathsf{A + B} + 1$	$(\mathsf{A \wedge B}) - 1$	$\mathsf{A \wedge B}$
010	$\mathsf{A \vee \overline{B}}$	$(\mathsf{A \vee \overline{B}}) + 1$	$\mathsf{\overline{B}}$	$-\mathsf{B}$	$\mathsf{A - B} - 1$	$\mathsf{A - B}$	$(\mathsf{A \wedge \overline{B}}) - 1$	$\mathsf{A \wedge \overline{B}}$
011	A	$\mathsf{A} + 1$	$\mathsf{A \wedge \overline{B}}$	$(\mathsf{A \wedge \overline{B}}) + 1$	$\mathsf{A + A}$	$\mathsf{A + A} + 1$	$\mathsf{A} - 1$	A
100	$\mathsf{\overline{A} \vee B}$	$(\mathsf{\overline{A} \vee B}) + 1$	$\mathsf{\overline{A}}$	$-\mathsf{A}$	$\mathsf{B - A} - 1$	$\mathsf{B - A}$	$(\mathsf{\overline{A} \wedge B}) - 1$	$\mathsf{\overline{A} \wedge B}$
101	B	$\mathsf{B} + 1$	$\mathsf{\overline{A} \wedge B}$	$(\mathsf{\overline{A} \wedge B}) + 1$	$\mathsf{B + B}$	$\mathsf{B + B} + 1$	$\mathsf{B} - 1$	B
110	$\overline{\mathsf{A \oplus B}}$	$-(\mathsf{A \oplus B})$	$\overline{\mathsf{A \vee B}}$	$-(\mathsf{A \vee B})$			$(\mathsf{A \oplus B}) - 1$	$\mathsf{A \oplus B}$
111	$\mathsf{A \wedge B}$	$(\mathsf{A \wedge B}) + 1$	zero	1			$(\mathsf{A \vee B}) - 1$	$\mathsf{A \vee B}$

Table 2. ALU function encoding.

$(\mathsf{P} = \overline{\mathsf{G}} \wedge \overline{\mathsf{K}})$ signal. The ALU is a static CMOS design because operands are not stable until several nanoseconds into phase one.

Our desire to keep pin count and instruction width low meant that we could not afford unencoded generate and kill lookup tables, as were used in B-SYS and the OM-2 [16, 27]. However, we required symmetric operations. In the Kestrel ALU, Operand A is always a register while Operand B can come from several sources. It would be a tremendous handicap for programmers and compiler writers if $A - B$ was supported but not $B - A$. The resulting 5-bit encoding is shown in Figure 2.

Conversion of the function codes to the carry chain control signals is straightforward. The truth table for the carry kill table, $\mathsf{K} = \mathsf{K_{11}K_{10}K_{01}K_{00}}$, is the same as the lower bits of the function table, $\mathsf{F_3F_2F_1F_0}$, except K is zero for decrement functions. The truth table for the generate function, $\mathsf{G} = \mathsf{G_{11}G_{10}G_{01}G_{00}}$, is the bit-reversal of the function code, except G is zero for increment functions, and $\mathsf{G_{00}}$ is always 0. The alternative function codes for the 8 logic functions that appear in both the decrement group and in the increment group can be used to set the $\mathsf{C_{out}}$ of the carry chain (this is not needed with Kestrel's flexible flag selection logic, but may be useful in other designs).

This compact ALU function encoding compares quite favorably to that of the classic 74181 chip. For the same number of function bits, the Arithmetic Operations mode of the 74181 supplied the first four functions of each type, leaving out all unary functions of B, as well as $\mathsf{B - A}$ and $-\mathsf{A}$. The 74181 could not negate at all.

The ALU, comparator, and equality test circuit have been tightly integrated (Figure 7). Because all three carry chains operate in parallel, the total delay is only slightly higher than that of the ALU alone. Since it is only 8-bits wide, carry lookahead is not needed.

The ALU and comparator pack 980 transistors in $356\,428\,\lambda^2$, including multiprecision and flag selection hardware. The control logic for the multiprecision features and flag selection occupy a proportionally large amount of space compared to the ALU and comparator bit slice due to the number of global control signals.

5.2: Bit shifter

The bit shifter provides bit manipulation of data and conditions. The data functions for packing and unpacking bits were included to support the sequence alignment function in sequence analysis methods. The conditional processing functions greatly enhance the SIMD processing element's ability to evaluate nested conditions based on local values.

Fast evaluation of local conditions is critical in SIMD processing. Each clause of a conditional must be broadcast to the array in turn so that those PEs for which the condition is true can execute the proper instructions. For short conditionals, such as processing boundary cases in floating-point arithmetic, the process of evaluating and nesting the conditions could, on some architectures, require more processing time than the actions themselves. For this reason, we integrated the bit shifter with the PE's local mask register, a single bit that determines whether or not the PE is active or

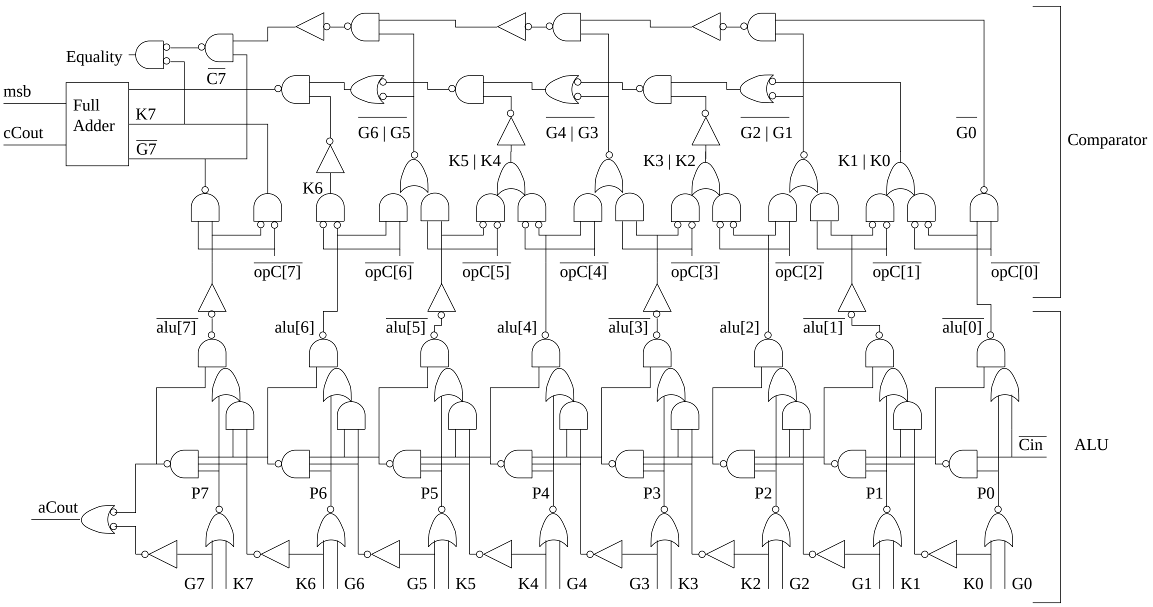


Figure 7. Logic diagram of the ALU and comparator.

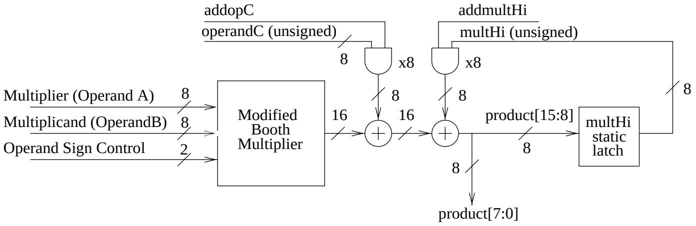
Function	Use
Clear	Erase all conditions, activate all PEs.
Push	Begin an `if` clause, adding a condition to the stack.
AND TOS	Replace the top of stack (TOS, the bit shifter's msb) with the AND of the TOS and another bit. Evaluate the second part of an `if (X | Y)` clause. The AND is used to evaluate OR clauses because bits are asserted low in the bit shifter.
OR TOS	Evaluate the second part of an `if (X & Y)` clause.
Complement TOS	Invert the current condition for an `else` clause.
Pop	Complete a condition, restoring a previous state.
Pop and Complement TOS	Remove one layer of nesting and start an `else` clause, as in `if (X) { if (Y) {}} else`.
Replace TOS	Complete one condition and start another at the same nesting level, as in `if (X) { } if (Y)`.
Store	Save current state. Free bit shifter for other tasks. Can be used with Load, Clear, and Set to process more than 8 nested conditions.
Load	Restore a previous state.

Table 3. Bit shifter functions associated with processing nested conditionals.

inactive (as with most SIMD designs, the mask register can be overridden globally).

The NOR of the 8 bit shifter bits can be used to set a PE's mask register. If any of the bits are 1 (making the NOR 0), the PE will be masked, and will not execute broadcast instructions. Pushing additional bits onto the bit shifter can further refine the set of active PEs, while popping bits will restore a previous set. Bit shifter microcode fields for masking are always unconditional (i.e., ignore the mask bit of the PE) to ensure the same number of conditions are present in all PEs.

The great advantage of Kestrel's horizontal microcode is that the bit shifter functions for conditional operations (Table 3) can in a large part be done concurrently with data processing, so that there is, for example, a 0-instruction cost for the 'else' of an if-else construct, and an ability to shift



Operand Bytes	Cycles for $A \times B$ with $\times$	$\times+$	$\times{+}{+}$
1	1	1	1
2	13	7	5
4	59	29	19
8	247	121	71

Figure 8. Multiplier (left) and savings due to multiply-accumulate-accumulate (right).

8-bit data left or write concurrently with an ALU instruction.

5.3: Multiplier

The multiplier (Figure 8) is implemented using the modified Booth's algorithm. It can treat either operand as signed or unsigned by considering each to be nine bits long, and manipulating the ninth bit based on how that operand should be treated. The lower eight bits of the product are the result, and the upper eight bits can be stored internally by the multiplier in the MultHi latch. The contents of the MultHi latch can then be read out on the Operand B bus or added to a subsequent multiply. Both Operand C and MultHi can be added internally to the product with no danger of overflow [22]. The ability to perform a multiply-accumulate-accumulate greatly speeds multiprecision multiplication (Figure 8).

The multiplier packs 2900 transistors in 1 162 075 λ^2, including bus drivers, operand latches, and control logic. The major design concern was finding an acceptable balance between speed and power consumption. To reduce dynamic power consumption, operand latches were introduced to prevent operands from changing when a multiply is not being performed. Also, as the upper eight bits do not have to be computed until the end of phase two, the upper carry chain could be slowed.

6: Chip Fabrications

Multiplier Test Chip The multiplier test chip (Figure 9) was fabricated in the Orbit 2.0 μm process using a TinyChip standard pad frame. We received four chips and they all worked as expected, except for one problem not found in simulation. The clock and data lines were reversed on the nMOS part of the dynamic latches that controlled the addition of MultHi into the product of a subsequent multiply. The resulting charge sharing problem caused the addition of erroneous values into subsequent multiplications. The rest of the design was carefully scrutinized to ensure this mistake was not made again.

SRAM and Register Bank Test Chip The memory test chip (Figure 9) measured 2.49 mm by 2.49 mm and had 18 996 transistors. We fabricated 25 chips in the HP 0.8 μm process. Of the 25 chips, one had a defect in two adjacent bits of the SRAM, and another had a defect that caused unpredictable behavior in the register bank. The SRAM could perform a read or write in approximately 7.5 ns, and the register bank could perform a read or write in 4 ns or less. Our ability to fully test the register bank speed was limited by our tester due to the arrangement of on chip control lines. SPICE simulations of the SRAM in the HP 0.5 μm process indicated that it would be fast enough to operate with a system clock speed up to 66 MHz, with a read/write time of about 5.0 ns. SPICE simulations of the register bank in 0.5 μm indicated a read time of about 3 ns.

Kestrel Test Chip The Kestrel PE test chip (Figure 10) has 51 966 transistors in 4.29 mm by 4.05 mm, and was fabricated in the HP 0.8 μm process. The test chip has two processing elements, three shared register banks, and instruction latches and decoders. This chip could be used interchangeably with the final chip, as it uses the same packaging, pin assignment, and functionality.

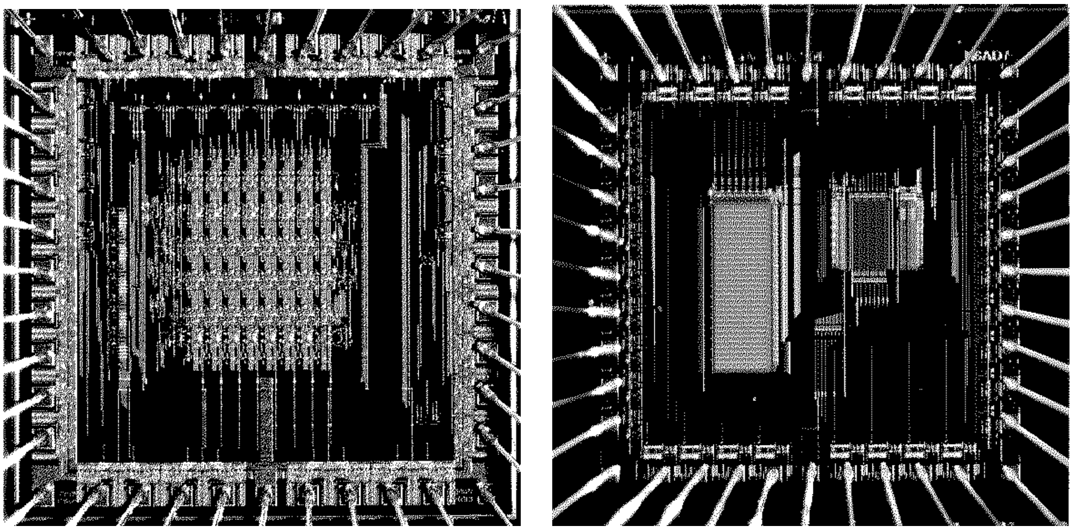


Figure 9. Multiplier (left) and the SRAM and register bank (right) test chips.

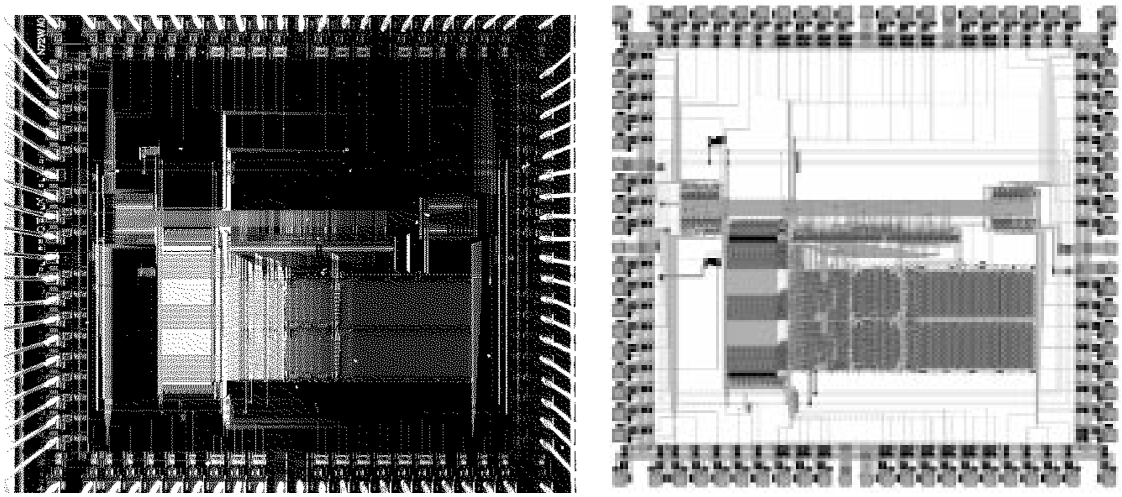


Figure 10. Photograph (left) and symbolic layout (right) of Kestrel test chip.

The Kestrel PE test chip, received in April, 1997, is fully functional apart from a minor control logic error that did not affect any of our test programs. Twenty-one chips work out of twenty-five chips fabricated. The chips can run with a 35 ns cycle time, and there is little variation in the timing performance of the chips, apart from two "fast" chips that can run approximately 10% faster. Because the final chip is being fabricated in a smaller, much faster process, the 35 ns cycle time bodes well for creating a 33–50 MHz system.

Sixty-four PE Chip The design of the 64-PE chip (Figure 11) was submitted for fabrication in June, 1997. It has 1.4 million transistors in eight columns of eight PEs and is about 7.2 mm by 8.3 mm in size, 80 percent larger than our original estimate. We estimated a need for 7 nF of on-chip distributed bypass capacitors, and the heat dissipation estimates indicate the chips may require heat sinks and fans. We expect to receive the chips in August, at which point we will test them and assemble the complete system.

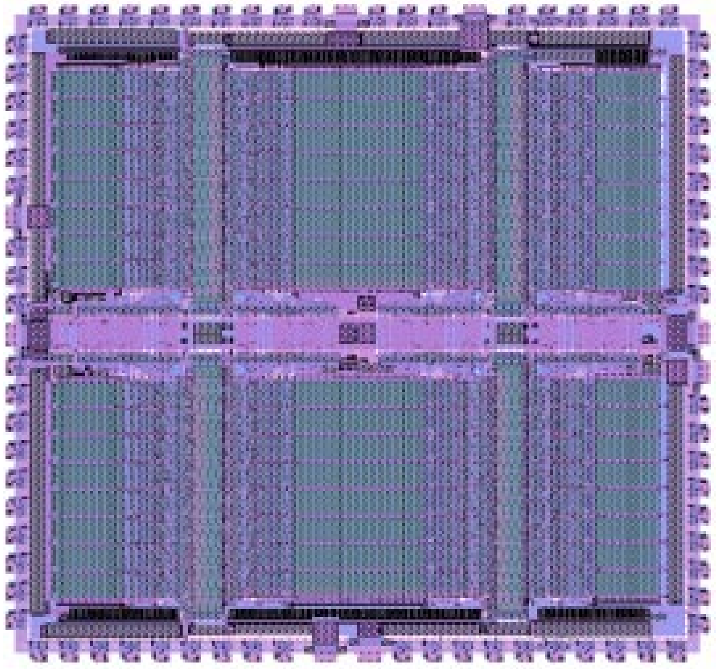


Figure 11. Symbolic layout of the Kestrel 64-PE chip.

Operation	Data Bytes		Kestrel	MasPar-2	MGAP-2	MMX	CNAPS	RaPiD
	Operands	Result						
Addition	1	2	2112	100	1000	1600	960	4800
Addition	2	2	1056	100	505	800	960	4800
Addition	4	4	528	133	253	400	(480)	2400
Multiplication	1	2	2112	40	150	600	960	1600
Multiplication	2	4	352	22		400	480?	1600
MAC	1	2	1056	29	130	400	960	1600
MAC	2	4	302	18		200	480?	1600
Peak I/O MB/S			66	20?	800?	1600?	60?	>4000?

Table 4. Performance in MOPS of selected SIMD chips.

7: Architectural Comparison

There is a great diversity among SIMD processing chips. In this section, we attempt a comparison of several typical architectures, both recent and planned. For simplicity, comparisons are based on performance per chip; it could be argued that performance per gate is more appropriate because this considers pure logic rather than access to the latest technology.

As with Kestrel, each of the machines described below has its target applications areas and advantages and disadvantages. This comparison does not highlight Kestrel's independent functional units, conditional evaluation, or programming model.

Table 4 shows estimated performance of several chips on integer addition, multiplication, and MAC operations of various sizes in millions of operations per second (MOPS). The peak I/O bandwidth data relates to transfer between local buffer memory or data cache.

CNAPS CNAPS is a linear SIMD array of 16-bit processing elements. Each PE has an 8 × 16-bit multiplier, a 16-bit ALU and a 32-bit adder, a 4 Kbyte local memory, and a 32-element 16-bit single-port register bank. CNAPS is clearly targeting MAC operations, and is the SIMD machine most similar to Kestrel. The large local memory includes support for sparse data representations. The latest chips contain 32 PEs and run at 30 MHz. An earlier 64-PE chip with 16 spares had 14 million transistors; the 32-PE chip would have around 6 million transistors, not including spares. I/O between the PEs and the controller uses shared 8-bit buses. Inter-PE communication uses a 2-bit bus, and instructions are 31 bits. Parallelism of execution for the different components allows 1 billion 16-bit MAC operations per second [14].

MasPar The MasPar MP-2 is a mesh-based SIMD supercomputer-class machine with a global router. Processing elements feature 40 32-bit registers, a 32-bit ALU with functional units to aid floating-point calculation, and a 64 Kbyte off-chip, locally-addressed memory. Each chip of just under one million transistors has 32 processing elements and two router interfaces. The ALU and memory interface can operate concurrently to partially alleviate the bandwidth problem of off-chip communication. The chips have a 12.5 MHz clock, with simple operations requiring three clock cycles [2].

Pentium MMX The Intel Pentium's MMX unit provides small-scale SIMD processing of 8-, 16- or 32-bit subfields of its 64 bit data. The Pentium architecture is cached and features two execution pipelines which under optimal conditions enable execution of 2 MMX instructions per cycle. The MMX unit has just 8 64-bit registers, and MMX instructions have only two operands. To aid image processing applications, MMX includes instructions for saturate arithmetic, clamping results to the maximum or minimum possible value to avoid overflow and underflow. The table displays peak rates for the 200 MHz Pentium MMX; real performance for applications that do not fit in the 8 MMX registers or have data hazards between instructions is likely to be considerably less [17].

MGAP The MGAP-2 is a very fine grained architecture. Each chip has 1536 1-bit PEs connected in an octagonal mesh. Each PE has a 16-bit dual-port local storage and two 3-input, 1-output function multiplexers for calculation. Based on published layout descriptions, the chips may have on the order of one half to two thirds of a million transistors. The configuration is stored in a local register but memory access is globally controlled. The system runs at 50 MHz. Performance is 6.8 kDCT/s (8×8 blocks of 8-bit pixels) for the MGAP-2 chip [20, 21].

RaPiD The RaPiD system is the most hardware-oriented among these examples: it is a "coarse grained FPGA," with 16 cells per chip arranged in a linear array. Each cell consists of three 16-bit ALUs, one multiplier, six datapath registers and three 32 by 16-bit memory blocks and some glue logic. The interconnection between these functional units, as well as between the cells, uses ten 16-bit segmented buses. A separate control path allows dynamic scheduling of the pipeline operations and some data path flow control. Transistor estimates are not yet available. The reference's estimate of fitting 16 cells on a chip running at 100 MHz will allow close to 1.6 million DCTs/sec (8×8 blocks of 8-bit pixels) [8].

8: Conclusions

Kestrel steers a course between special-purpose co-processors, reconfigurable-logic systems, and general-purpose supercomputers. Our design grew from an initial desire to build a small, fast, and inexpensive sequence analysis machine into a general purpose parallel accelerator. The design does not exist in a vacuum, however. Its evolution has been shaped in particular by several architectures including B-SYS [16], MasPar [30], Blitzen [2], and the unbuilt MISC machine [33], as well as MGAP and PIM [3, 12]. Contemporaneous sets of design choices can be found with the CNAPS [14], RaPiD [8], Rapid-2 (not related) [9], Samba [24], and SIMPil [10] architectures, among others.

The resulting design provides high performance at low cost on its prime target application, biological sequence analysis, as well as on other applications amenable to SIMD parallelization. The design has been the result of a constant interplay between algorithm development, system design requirements, processing element architectures, and VLSI and board design constraints.

References

[1] D. Benson, M. Boguski, D. Lipman, and J. Ostell, "Genbank," *Nucleic Acids Research*, vol. 25, no. 1, pp. 1–6, 1997.

[2] D. W. Blevins *et al.*, "BLITZEN: A highly integrated massively parallel machine," *J. Parallel and Distributed Computing*, vol. 8, pp. 150–160, Feb. 1990.

[3] M. Borah, C. Nagendra, R. Owens, and M. J. Irwin, "The MGAP: A high-performance, user-programmable, multifunctional architecture for DSP," in *Proc. Hawaii Int. Conf. System Sciences*, pp. 96–104, IEEE, 1994.

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

[5] 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.

[6] N. H. Christ and A. E. Terrano, "A micro-based supercomputer," *Byte*, pp. 145–160, Apr. 1986.

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

[8] C. Ebeling, D. C. Conquist, and P. Franklin, *RaPiD — Reconfigurable Pipelined Datapath*, pp. 126–135. New York: Springer-Verlag, 1996.

[9] P. Faudemay and L. Winckel, "An abstract model for a low cost SIMD architecture," in *Proc. Int. Conf. ASAP*, (Los Alamitos, CA), pp. 145–154, IEEE CS, July 1996.

[10] A. Gentile *et al.*, "Real-time implementation of full-search vector quantization on a low memory SIMD architecture," in *IEEE Data Compression Conference*, p. 438, Mar. 1996.

[11] M. Gokhale *et al.*, "Building and using a highly parallel programmable logic array," *Computer*, vol. 24, pp. 81–89, Jan. 1991.

[12] M. Gokhale *et al.*, "Processing in memory: The Terasys massively parallel PIM array," *Computer*, vol. 28, pp. 23–31, Apr. 1995.

[13] J. A. Grice, R. Hughey, and D. Speck, "Reduced space sequence alignment," *CABIOS*, vol. 13, no. 1, pp. 45–53, 1997.

[14] D. W. Hammerstrom and D. P. Lulich, "Image processing using one-dimensional processor arrays," *Proc. IEEE*, vol. 84, no. 7, pp. 1005–1018, 1996.

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

[16] 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.

[17] Intel Corporation, http://developer.intel.com/drg/mmx/manuals/prm/prm.htm, *MMX Technology Developer's Programmer's Reference Manual*, 1997.

[18] ISO/IEC, "IS-10918: Compression and coding of continuous-tine still images."

[19] N. Jagadish, J. M. Kumar, and L. M. Patnaik, "An efficient scheme for interprocessor communication using dual-ported RAMs," *IEEE Micro*, pp. 10–19, Oct. 1989.

[20] T. P. Kelliher, E. S. Gayles, R. M. Owens, and M. J. Irwin, "The MGAP-2: An advanced, massively parallel VLSI signal processor," in *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, vol. 5, pp. 3219–22, IEEE, May 1995.

[21] H.-N. Kim, M. Borah, R. M. Owens, and M. J. Irwin, "2-D discrete cosine transforms on a fine grain array processor," in *Proc. VLSI Signal Processing VII*, pp. 356–367, IEEE, Oct. 1994.

[22] D. E. Knuth, *The Art of Computer Programming*, vol. 2. Reading, MA: Addison-Wesley, 2nd ed., 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. Mol. Biol.*, vol. 235, pp. 1501–1531, Feb. 1994.

[24] D. Lavenier, "SAMBA: Systolic accelerators for molecular biological applications," Tech. Rep. 988, IRISA, 35042 Rennes Cedex, France, Mar. 1996.

[25] 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.

[26] R. N. Mayo *et al.*, "1990 DECWRL/Livermore Magic Release," Research Report 90/7, Digital Western Research Laboratory, Palo Alto, CA, Sept. 1990.

[27] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems.* Reading, MA: Addison-Wesley, 1980.

[28] K. Mehrotra *et al.*, *Elements of Artificial Neural Networks.* Cambridge, MA: MIT, 1997.

[29] 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.

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

[31] W. R. Pearson, "Personal communication," 1995.

[32] E. Rice and R. Hughey, "Multiprecision division on an 8-bit processor," in *Proc. 13th IEEE Symp. Computer Arithmetic*, IEEE CS, July 1997.

[33] J. D. Roberts, *MISC: A parallel architecture for AI.* PhD thesis, University California, Santa Cruz, CA, 1995.

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

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

[36] R. Singh *et al.*, "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.

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

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

[39] T. A. Thanaraj and T. Flores, "Assessment of Smith-Waterman sequence search tools implemented in Biocellerator, FDF, and MasPar," tech. rep., European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge, Feb. 1997. Available from http://industry.ebi.ac.uk/~thanaraj/seqassess/report.html.

[40] Time Logic Inc., "Decypher II product literature." Incline Village, NV, http://www.timelogic.com, 1996.

[41] J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, *et al.*, "Programmable active memories: reconfigurable systems come of age," *IEEE Trans. VLSI Systems*, vol. 4, no. 1, pp. 56–69, 1996.