Abacus: A 1024 Processor 8 ns SIMD Array

M. Bolotski T. Simon
C. Vieri R. Amirtharajah T. F. Knight, Jr.

MIT Artificial Intelligence Laboratory
Cambridge, MA 02139

1 Introduction

We describe Abacus, a SIMD architecture designed for early vision applications. A single Abacus chip implemented in a 1μ technology is expected to deliver 2 billion 16-bit arithmetic operations per second. The key idea of the architecture is to use a configurable number of bit-slice processing elements to operate on a single word in parallel. This approach allows higher performance than a bit-serial (BS) design for fixed-time applications, while retaining the flexibility of variable precision. The simplicity of each bit slice allows a high-speed implementation, which leads us to consider system-level issues of high-speed SIMD machines. These issues must be faced not only by our implementation, but by future high-speed SIMD designs.

The paper consists of three major parts. The first describes the architecture of the Abacus bit-slice PE, a set of reconfigurable bit-parallel (RBP) arithmetic and communication algorithms enabled by the architecture, and our VLSI implementation. The second part uses the implementation parameters in a silicon efficiency model to demonstrate that an RBP organization can be more area-efficient than a bit-serial one. The third part discusses some of the design challenges faced by a high-speed SIMD system and our solutions to these problems.

2 Background

Replication of simple elements was an early approach to low-cost high-performance computing. In massively parallel systems containing thousands of processors, the control flow has always been SIMD, saving the silicon area of control circuitry and instruction memory. The remaining architectural issues have been the interconnection network linking the PEs, the complexity of each PE, and the amount of on-chip memory.

Our early computer vision problem domain reduces the design space considerably. These computations are characterized by dense computations on a 2-dimensional mesh, with low-precision integers of 16 or fewer bits. A mesh interconnection with enhancements for data reduction operations (such as counting or maximum) is sufficient. Floating point support, while important for scientific computations, is not relevant here. The main architectural issue now becomes the allocation of on-chip silicon to ALU and memory resources to maximize overall performance.

Early massively parallel designs relied on bit-serial computation, in which every ALU was one bit wide and therefore performed arithmetic on one bit of a word at a time [2]. Later designs enhanced the bit-serial ALU with special-purpose hardware such as a shift-register in order to speed multiplication [4]. From that point designs took two directions. One approach, as exemplified by the Maspar MP-1 and MP-2 increased the ALU size to 4 bits and then 32 bits [6]. The other approach was to group one-bit PEs into a reconfigurable bit-parallel *cluster* and process multiple bits of a word simultaneously [5].

The obvious tradeoff is that simple one-bit PEs are capable of arbitrary precision and can run at a high clock speed, but require many cycles per operation. Hardware bit-parallel PEs require few cycles, but run slower and waste valuable silicon when computing with data narrower than their ALU width. We have chosen to explore the reconfigurable bit-slice approach, as it retains the bit-serial flexibility and bit-parallel performance. In the Abacus architecture, each bit-slice is one bit wide.

3 Abacus PE Architecture

The Abacus processing element (PE) consists of 64 1-bit registers organized into two banks. There are two 3-input ALUs, each of which takes two inputs from its memory bank and one input from the other bank. The four available data bits allow complex boolean functions, and two result bits can be written in each cycle, one bit to each bank.

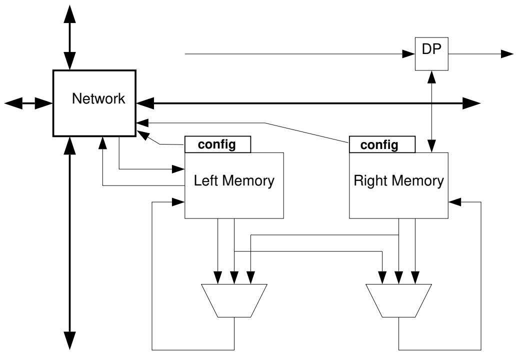

Figure 1. Processing Element Block Diagram.

A PE also has a 1-bit network interface and background I/O interface. Seven of the registers are used for control, leaving 57 general-purpose registers.

Network The reconfiguration network serves for both bit-slice interconnection and inter-word communication. The network is a wired-OR reconfigurable mesh. As in a conventional mesh topology, each PE listens to the output node of one its four mesh neighbors. Each PE is provided with a configuration register that specifies which neighbor to listen to. Unlike a mesh, each PE can connect its output node to the selected neighbor's output node. Shorted nodes behave as a wired-OR bus: if any PEs in a connected chain send a 1, all PEs listening to the chain receive a 1.

Activity Register Conditional execution of an `if (cond) then ... else ...` sequence on SIMD arrays operates by disabling those PEs for which `cond` is true, executing the true branch, disabling the other set of PEs, and executing the false branch. One of the

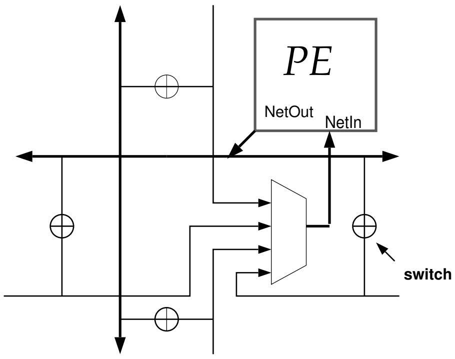

Figure 2. Network

Abacus PE registers serves as the `activity` bit which, when cleared, disables computation by inhibiting the write of the result to the destination register.

The ability of the ALU to serve as a multiplexer greatly reduces the use of the activity register. For example, the sequence `if` A `then` $B \Leftarrow C$ incurs two cycles of overhead if implemented with the activity register. Alternately, the operation can be expressed as $B \Leftarrow$ `mux(`A`,`$C|B$`)`, eliminating all overhead.

Background I/O Each PE contains a data plane (DP) register, which is used for background off-chip data transfers. The DP registers of PEs in a column of the array form a 32-bit shift register. At the edge of the PE array, the 32 shift registers are connected to an off-chip memory port. These registers can be shifted without interfering with PE computation. Although the DP register is not used in arithmetic operations, it is essential in hiding the latency of external memory accesses.

4 RBP Algorithms

RBP arithmetic algorithms resemble circuits more than bit-serial programs, since they are unrolled in both space and time. During each time step, each PE emulates part of the logic comprising an arithmetic circuit, reconfiguring connections if necessary. In the examples of this section, clusters are shown organized as a line in order to resemble a circuit. In reality, clusters are often arranged as squares or rectangles (a square organization minimizes the average inter-cluster communication distance), and topologically form a ring. In arithmetic circuits, boundary logic cells such as the LSB and MSB are configured differently from the middle cells. Similarly, PEs corresponding to cluster edges are marked as MSB or LSB. In addition, each PE is labelled with its bit position within the cluster. The LSB/MSB bit can be computed from this position, as can network configuration bits.

Logical Shift. The simplest RBP arithmetic operation is the shift. As shown in Figure 3, each PE simply replaces its bit with its neighbor's. For logical shifts, the MSB or LSB is cleared; for arithmetic shifts the MSB is retained.

Sum Accumulation. Accumulating a sum is a very common operation. The sum of a sequence of n numbers can be evaluated in $\Theta(n)$ cycles with a carry-save (CS) algorithm.

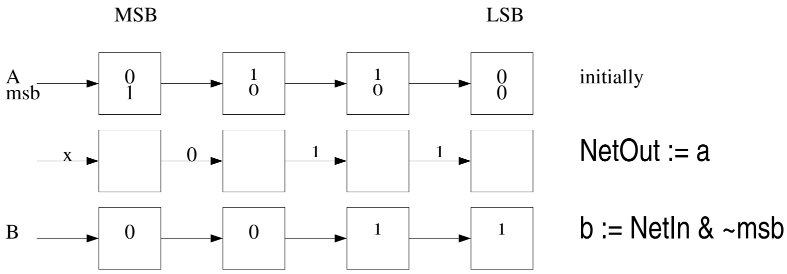

Figure 3. Logical shift right. The number 6 is shifted down to become 3.

The CS adder computes the sum and carry bits separately for each addition. The computation is purely local, and the delay of each CS stage is independent of word size. At the end of a summing sequence the carry and sum values must be summed with a full carry-propagation adder, but this delay is amortized over many summations. Many computations usually expressed as additions can be reformulated as accumulations, improving the performance of algorithms such as region summing. It is interesting to note that accumulation is an inherently simpler and faster operation than addition, and yet conventional processors do not make the distinction.

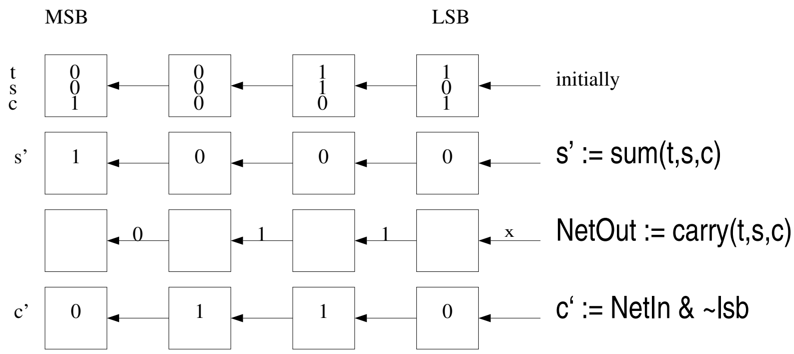

Figure 4. Accumulate. The addition of the number 3 to a sum of 11 represented as 2+9. After the operation, the sum is 14, represented as 6+8. All operations are purely local or nearest neighbor. Steps 1 and 2 can happen during the same cycle if a copy of t is present in both memory banks.

Addition. Fast addition relies on quick computation of the carry bit. A RBP algorithm based on the the well-known logarithmic-time technique was described in [1]. The algorithm "looks-ahead" at the carry by computing the eventual carry into each one-bit adder. It can be shown that the carry c_k into bit k can be expressed as $c_{k+1} = g_k + p_k c_k$, where g_k is the *generate* bit and p_k is the *propagate* bit. Since the computation of successive p and g values can be expressed in terms of a binary associative operator, it can be performed in time logarithmic with the number of bits. This technique forms the basis of many hardware implementation of adder circuits.

The Abacus architecture uses a different approach, one based on the hardware implementation of Manchester carry computation. In this technique, the carry bit propagates electrically down the line of bits unless prevented by a switch controlled by the p signal. Although computation time is linear with the number of bits, the constant factors result in faster operation than software emulation of the logarithmic-time circuit.

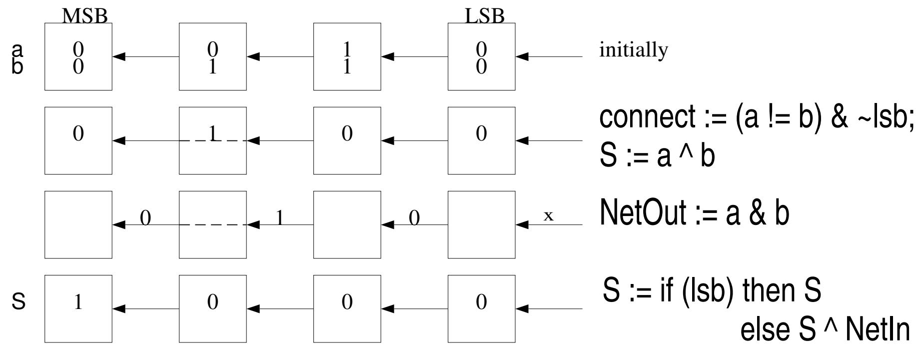

Figure 5. Manchester-carry addition. A NOP cycle occurs after step 2 to allow the carry bit to propagate. Steps 1 and 2 are actually merged into one cycle, and the `S` bit can be computed during the NOP cycle.

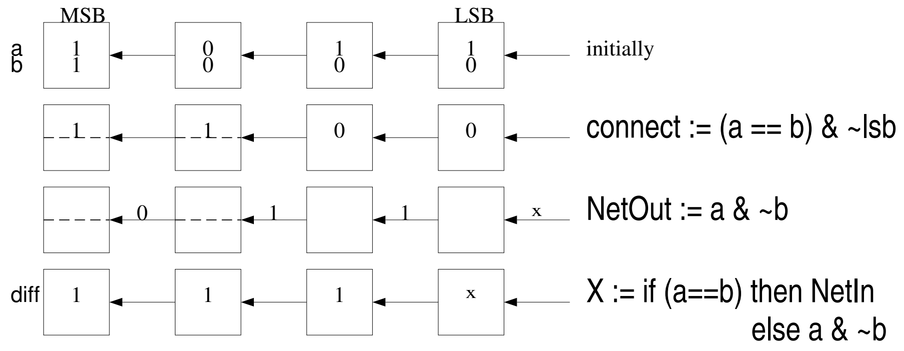

Figure 6. The `compare` operation. After the broadcast, the MSB PE receives A>B for the rest of the word. It then merges the result with the MSB bit result.

Comparison. The comparison algorithm is based on the observation that the most significant differing bit (MSDB) between two words determines which word is greater. Thus if the MSDB occurs in position i, $A > B$ when $A_i > B_i$, or equivalently $A_i = 1$ and $B_i = 0$. In the first step of the algorithm, all PEs with identical bits bypass themselves and do not participate in the rest of the computation. Then, all PEs send $A \cdot \overline{B}$ towards the MSB PE. Now, the MSB PE knows the value of $A > B$ for the rest of the word, and merges the result with its own data bits to produce the final answer. This entire process is equivalent to performing a Manchester-carry subtraction and testing for the borrow bit at the MSB.

Match. The match operation compares two bit patterns for equality. First, all PEs form a wired-OR bus. Then, all PEs write `a != b` to the bus. If any bits differ, the bus becomes a 1; if all bits are identical, the bus remains at 0.

Multiplication. RBP integer multiplication is composed of iterating through a sequence of shifts, broadcasts and carry-save adds. These operations can be partially overlapped so that an iteration requires 7 cycles. Thus, a 16 by 16 multiply requires 112 cycles, plus about 10 cycles of overhead. These numbers are for unsigned multiplication with a double-length result. If only the low word of the result is required, an iteration requires only 6 cycles. Two's complement multiplication is implemented with a software version of the Baugh-Wooley multiplier, and requires one additional iteration. While hardware multipliers frequently use the modified Booth algorithm to halve the number of iterations, this algorithm is not very efficient on the Abacus architecture, since it requires the broadcast of three bits.

Mesh Move. In a mesh move, each cluster rearranges itself into several shift registers in the direction of movement. The move operation is relatively expensive, as k bits must be transferred through a $\sqrt{k}$-wide bus, requiring $\sqrt{k}$ time steps. Further, unlike arithmetic operations, mesh moves cross chip boundaries and therefore suffer an additional 1-cycle latency cost.

Scan Operations. A set of computational primitives known as *parallel prefix* or *scan* operations have been found to be useful programming constructs across a variety of machine architectures. These operations take a binary associative operator $\oplus$, and an ordered set $[a_0, a_1, \ldots, a_{n-1}]$ and return the ordered set $[a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots a_{n-1})]$. Common scan operations include *or-*, *and-*, *max-*, *min-*, and *+-scan*. There are also corresponding *reduce* primitives that reduce the values in a vector to a single value. For example, *+-reduce* calculates the sum of a vector.

The bypass capability of the Abacus network can be used to implement fast scan operations. A message can propagate across a chip in two cycles. Crossing a chip boundary requires an additional cycle of latency. The prototype is a 16 by 16 chip array, so that 48 cycles are required for a bit to cross the array. Assuming a 4 by 4 bit organization, propagation of a 16-bit cluster requires 200 cycles. Since each stage of a logarithmic scan doubles propagation time, an entire scan operation requires approximately 500 cycles, or 4 microseconds. This can be optimized further by pipelining bit transmission.

Performance Summary. The performance of a single Abacus chip on the operations described in this section is given is in Table 1. The left half of the table gives cycle counts, while the right gives the MOPS rating assuming 1024 PEs per chip and a 125 MHz clock rate.

These ratings are somewhat pessimistic for two reasons. First, they include an additional one-cycle penalty for reconfiguration. For example, if a right shift instruction is followed by an add instruction, PEs must be reconfigured from reading from the MSB direction to reading from the LSB direction. This penalty does not occur during every instruction, and a clever compiler can group instructions with identical configurations to reduce reconfiguration costs.

Second, as discussed earlier, microcode for several arithmetic operations can be overlapped. By the table entries above, the sequence $a = b + 2d + 2c$ performed on 16-bit values appears to require 14 cycles (two shifts, an accumulate, and an add). Yet a handcoded

	Cycle Count			MOPS		
Operation	8-bit	16-bit	32-bit	8-bit	16-bit	32-bit
Add, Compare	4	4	5	4000	2000	700
Shift	3	3	3	5300	2700	1300
Accumulate	4	4	4	4000	2000	1000
Mesh Move	5	6	8	3200	1300	500
Multiply	66	126	235	242	63	17

Table 1. Single-chip arithmetic performance

program performs the same operations in only 9 cycles by eliminating three reconfiguration steps, overlapping two broadcasts, and duplicating a data bit in both memory banks.

5 VLSI Implementation

Processing Element Almost half of the PE's area is consumed by registers, which are implemented as standard 6T SRAM cells, and arranged into two banks of eight rows and four columns. The `bit` and `/bit` lines are used single-ended to provide two read ports, allowing four bits to be read and two written in each cycle. Bitline precharging to $V_{dd} - V_t$ is done through the column selects, precharging only bitlines that will be read. Each ALU is implemented as a precharged 8-to-1 multiplexer, that uses three data bits to select one of eight instruction bits. During each 8 ns cycle, a PE performs a bitline precharge, SRAM read, ALU evaluate, and SRAM write. The background DP register can be shifted simultaneously without interfering with the ALU. The simplicity of each PE resulted in a compact, 83μ by 452μ implementation in a 2.6μ minimum metal pitch technology.

Network The on-chip interprocessor communications network consists of a precharged output node and four isolating switches. The node is a simple precharged inverter whose input is driven by the `NetOut` register. The four switches are NMOS gates driven by a 2-4 precharged decoder. The precharge pulse is controlled by software to allow multi-cycle propagation times.

Propagation delay across a series of pass transistors is usually quadratic in the number of devices. Our implementation greatly reduces this delay by a local accelerator circuit at each node which regeneratively pulls a node to ground as soon as the node voltage drops by a transistor threshold. The circuit is a dynamic NORA style circuit, and is well suited to the precharged operation of the network. We found it to be several times faster than an implementation based on a complementary inverter. At the nominal process corner, simulations show that a bit propagates through 18 PEs in one 8 ns cycle.

This circuit has a noise margin of V_T, so careful layout was done to reduce coupling capacitance to the pre-discharged node. Further, the chip is provided with on-chip bypassing and over 200 power and ground pins to reduce dI/dt noise.

Chip Organization. The chip is organized into three major parts: the PE array, split into two halves; the control signal drivers in the middle; and control logic at the left. The control logic includes address decoders, instruction decode, external memory control, and a scan path controller. The PE core is organized as 16 strips of 64 PEs each. Each strip also contains two sets of 76 buffers that drive control signals outward from the center.

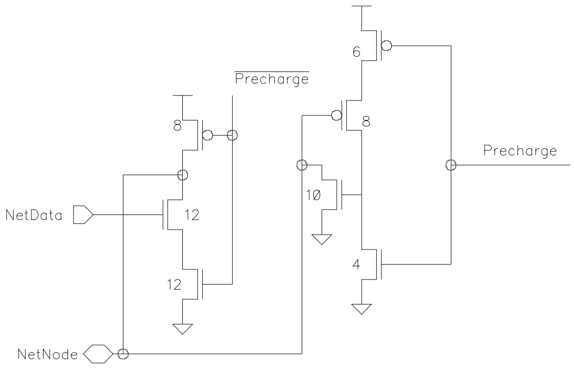

Figure 7. Network accelerator in the NORA circuit style.

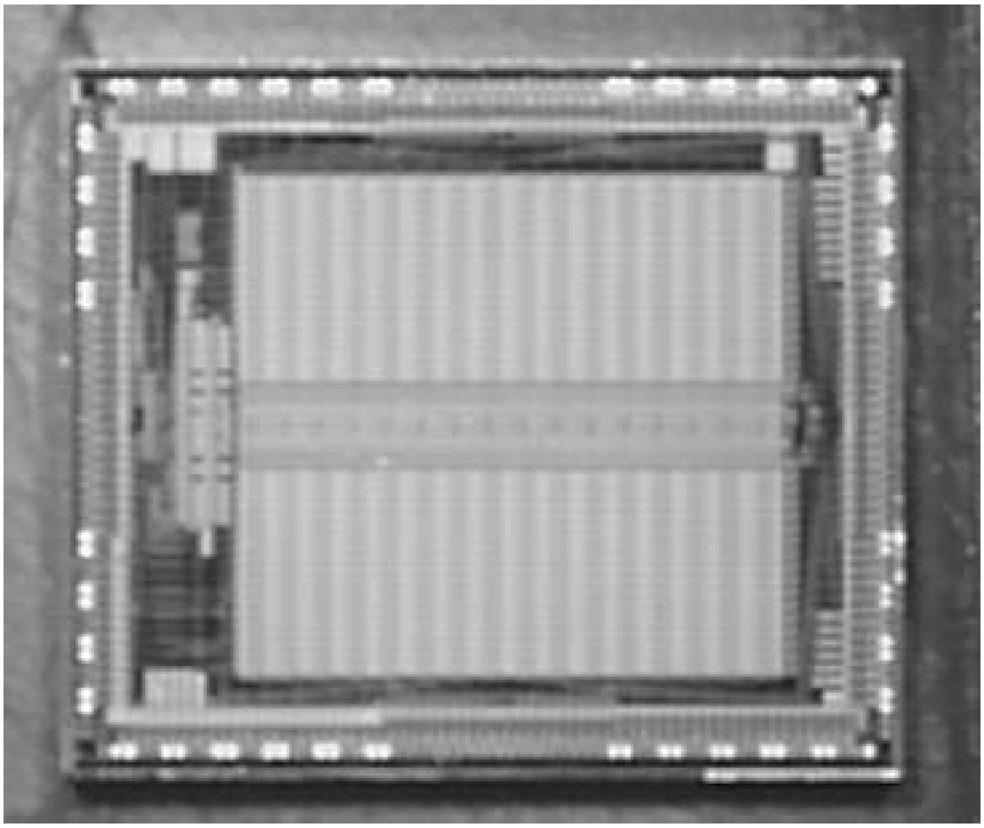

Figure 8. Chip photograph.

Skew is reduced by distributing signals vertically in a lightly-loaded metal-3 bus, buffering horizontally with identical drivers, and equalizing the load on each horizontal control wire.

We have received manufactured chips in January 1995, and are about to submit custom packages for fabrication. We expect to test packaged chips in early March. A photograph of the 10.8 by 8.8 mm chip is shown in Figure 8. The PE array accounts for approximately 40% of the chip area, the pad ring for 30%, signal distribution buffers for 10%, and control logic for 4%.

6 RBP Performance Model

In this section we provide an argument that the RBP organization is not only capable of higher performance than BS, but can also be more area efficient. We assume that a processing unit, be it a bit-serial PE or a cluster of RBP PEs, requires a certain amount of on-chip storage to operate without frequent off-chip memory accesses. If this requirement is not met, performance will degrade catastrophically due to the limited off-chip bandwidth.

In the following discussion, let B be the number of bits in a data word; W, the number of words in the register file; V, the area of the non-memory overhead (ALU and network) component of the PE; and S, the number of cycles required by the bit parallel operation. Area measurements are normalized to the size of one memory bit. A bit-serial organization requires B cycles to step through each bit of a data word and occupies $V + BW$ area. An RBP cluster requires S cycles and occupies $B(V + W)$ area. The silicon efficiency can be expressed as performance per unit area, or simply as $1/AT$. The ratio of efficiency, bit-parallel to bit-serial, is:

$$R = \frac{V + BW}{S(V + W)}$$

For the vision algorithms we are interested in, computations on 8-bit pixels are done with 16 bits to preserve fractions, and as many as 64 words are frequently accessed. In our implementation of the PE, V was approximately 50. Our algorithms have S ranging from 3 to 6, averaging around 4. A graph of R versus W, for relevant values of B, S and V is shown in Figure 9.

7 System Design and Off-Chip Interfaces

The Abacus PE chip contains several mechanisms required for high-speed system-level operation: skew-tolerant instruction distribution pads, high-bandwidth local DRAM interface, low voltage swing impedance-matched mesh pads, and low pin-count I/O interface.

7.1 System Synchronization

Careful synchronization is essential to a high-speed SIMD architecture. On the Abacus PE board the system clock is distributed in differential ECL to each PE chip. Propagation delays are matched by snaking traces so that trace lengths are equalized. To compensate for skew introduced by process variations in the on-chip clock distribution, each chip phase locks its internal clock to the received clock. Phase locking is performed by a delay locked loop (DLL) based on an all-digital delay line. The DLL uses the simple and accurate arbiter circuit shown in Figure 10. The arbiter has a perfectly symmetric 30 ps uncertainty window at nominal process.

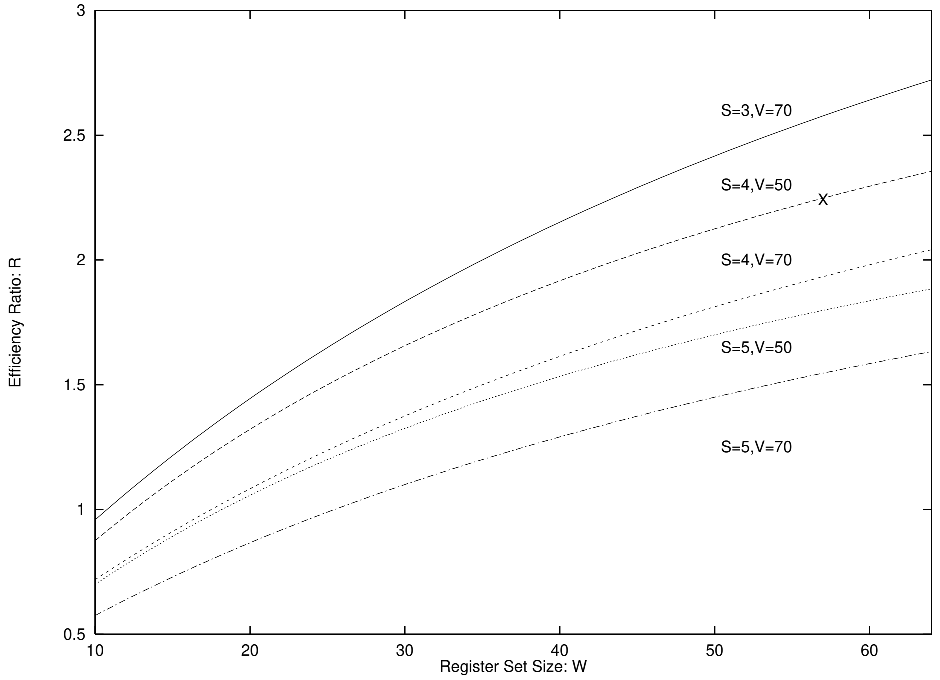

Figure 9. Efficiency ratio R vs register file size W, for various values of non-memory overhead V and algorithmic inefficiency S. The 'X' indicates the design point of our implementation.

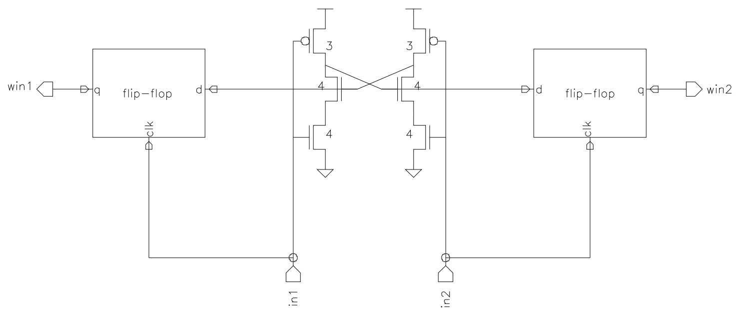

Figure 10. Arbiter used in the delay-locked loop

The remaining question is: which of the internal clock phases should be locked to the system clock? There are several internal clocks: instruction register, PE register read, network precharge, etc. We choose to synchronize to the clock that drives the mesh output pads. As a result, off-chip communication is synchronized system-wide.

7.2 Instruction Distribution

A spatially distributed SIMD system must ensure not only that all clocks are synchronized, but also that all chips are executing the same instruction at the same time. Clock distribution can be controlled by snaking traces to equalize propagation delay, or by using specialized clock distribution parts with programmable delays. Neither technique is feasible for the 30-bit instruction bus running at 250 MHz. Since each PE board is approximately 60 cm on a side, even with signals driven from the center of each board, distribution skew can be as large as 2 ns.

Another source of instruction skew is the variability of discrete parts in the distribution path. For example, the clock to output delay on a high-performance ECL shift register has an uncertainty of 0.4 ns. A signal traversing two parts can be skewed by a much as 0.8 ns. The combined effect of these skews is that some chips receive an instruction a cycle later than others, even though all chips are supposed to be executing the same instruction at once. Worse yet, bits of an instruction can be mixed with bits of the next instruction due to parts variability.

This skew problem is solved by retiming logic in the instruction pads that synchronizes instruction delivery across all chips. Retiming occurs as part of the system startup process and proceeds in three stages. The first finely adjusts the delay of each bit until the sampling clock occurs in the middle of the data valid period. The second stage lines up all bits within a chip to the same pipeline stage. Finally, pipeline delays are equalized across the system by software. The various stages of the initialization process are controlled through a scan path by the array controller.

7.3 Inter-Chip Mesh Communication

The next system level challenge is to provide synchronous high-bandwidth, low-latency mesh communication between PE chips. Due to pin limitations, the edges of the 32 by 32 PE mesh on each chip are time multiplexed onto 16-bit ports. The required clock rate is therefore 250 MHz.

Each of the four mesh ports sends out an escort clock along with the data. Since the off-chip drivers and propagation time are identical for data and clock, the receiving chip can safely use the escort clock (delayed by the setup time of the receiving registers) to sample the data. Metastability problems do not arise as we ensure that data is sampled well before the receiving data needs to use it.

The communication pads are driven with custom 1V, on-chip series terminated drivers and matching receivers in order to minimize power dissipation and transmission line reflections [3]. Impedance calibration for the series termination is performed at system initialization. Each of the four ports on a chip is calibrated independently, since on-board and board-to-board impedances may differ.

In addition to the pipelined, high bandwidth mode of operation, the mesh pads can operate in a flow-through mode that bypasses the on-chip pulldown network with a single wire, allowing fast multi-chip broadcast algorithms. Propagation through a chip is reduced from three clock cycles to half a cycle. However, the time multiplexing is disabled, reducing the effective bandwidth.

7.4 Image I/O

While early SIMD designs were concerned with limited image I/O bandwidth, it is not an issue today's high-speed designs. Since I/O is distributed among 256 chips, a single pin on each component results in a very wide bus. The Abacus PE chip has a one-bit single-ended ECL input and a one-bit differential ECL output for external data I/O. The aggregate bandwidth of a 32-byte, 125 MHz bus is 4 GB/sec, which is almost three orders of magnitude than the 7.7 MB/sec required by 512 by 512 images at 30 Hz. The I/O bottleneck is clearly going to occur upstream of the PE array.

The I/O bandwidth can be scaled down to reduce the interface logic and wiring cost. For example, if only the 16 chips comprising the top row of the 256-chip array were connected, the 4 GB/sec bandwidth would be reduced to 250 MB/sec, which is more than an order of magnitude higher than frame rate.

A single PE instruction initiates a burst transfer of 32 bits from the edge of the PE mesh to the output pin (or vice versa). This background transfer frees the array to perform computation, or in the case of the lower cost interface, to use the mesh connections to shift data towards the connected chips.

7.5 External Memory Interface

Since many algorithms will exhaust the 64 Kbit on-chip memory, each PE chip is backed by two 1M x 32 specialized DRAM modules. The data interface is 64 bits wide and operates at 16 ns, which matches the 32-bit, 8 ns PE mesh. The 16 ns cycle time is made possible by the Ramtron extended DRAM part, which contains an SRAM cache integrated in the DRAM.

Control signals for the memory chips are synchronized to the PE chips by sending them as part of the instruction stream. Fine control over DRAM timing is made possible by the small instruction cycle time, which is comparable to that of dedicated DRAM controller. A useful side-effect of this approach is the elimination of memory glue logic. A counter in each PE chip generates the external memory addresses. This counter can be loaded from the top row of the PE array, allowing indirect addressing.

8 Summary and Conclusions

This paper described the Abacus machine at a number of levels. We presented the microarchitecture of the PE comprising the reconfigurable bit-parallel array, a set of arithmetic and communication primitives, details of the VLSI implementation, and system-level design issues of a high-speed SIMD array.

The most concrete goal of the Abacus project was to design and build a machine that could be used by members of the MIT Artificial Intelligence Laboratory for real-time early vision processing. Along the way, we explored several architectural ideas.

First, we tested the limits of the premise that simple one-bit PEs allow a faster overall clock rate. Although this is a common argument in research papers, existing MPP SIMD systems use a clock rate substantially slower than that of commercial bit-parallel microprocessors. The 125 MHz clock of the Abacus chip is the highest of any massively parallel SIMD systems that we're aware of. Further increases in clock speed are limited by instruction bandwidth rather than PE complexity, and therefore transfer the difficulty to off-chip interfaces and printed circuit board design. Retaining the clock speed while increasing the work done per cycle holds more promise as the approach for incorporating smaller and faster VLSI technology.

Second, we determined how effectively a collection of simple one-bit PEs could emulate bit-parallel hardware. As shown in Section 4, a variety of arithmetic circuits can be emulated at a cost of two to five cycles. However, while reconfiguration reduces the silicon cost, it increases the time requirements in two ways. Bit-level reconfiguration precludes hardware pipelining by introducing dependencies on network switch settings. Conventional techniques for dealing with data dependencies (such as forwarding) cannot be used. Additionally, reconfiguration overhead is a significant fraction of arithmetic operation times. We plan to address these issues in our next multi-bit bit-slice design.

Third, we wanted to complete an entire system to see if there are any unexpected bottlenecks and performance hits in high-speed SIMD designs. Consideration of board-level issues led to the development of instruction retiming logic, high-speed mesh signaling, low pin-count data I/O, and software control over DRAM timing. We now have a system framework that allows us to replace details of the PE core and remain plug compatible with the rest of the machine. Since the clock rate is already aggressive, we'll be able to hold it steady for a number of years while researching how to get more work done per cycle.

Specialized architectures have fallen out of favor as commercial pressure continually drives up the performance of general purpose microprocessors. The current trend in parallel architecture is to construct parallel systems from slightly modified commodity processors. The Abacus project demonstrates that architectures optimized for specific computations can be an order of magnitude faster than such general purpose systems.

Acknowledgements

This research is supported in part by the Advanced Research Projects Agency under contract DABT63-92-C-0039. We would like to thank Cadence Design Systems for providing VLSI design tools.

References

[1] M. Bolotski, R. Barman, J. J. Little, and D. Camporese. Silt: A distributed bit-parallel architecture for early vision. *International Journal of Computer Vision*, IJCV-11:63–74, 1993.

[2] Eugene L. Cloud. The geometric arithmetic parallel processor. In *Frontiers of Parallel Computation '88*, 1988.

[3] André DeHon, Thomas F. Knight Jr., and Thomas Simon. Automatic impedance control. In *ISSCC Digest of Technical Papers*, pages 164–165. IEEE, February 1993.

[4] Jerry L. Potter, editor. *The Massively Parallel Processor*. MIT Press Series in Scientific Computation. The MIT Press, Cambridge, Massachusetts, 1985.

[5] Andrew Rushton. *Reconfigurable Processor-Array: a bit-sliced parallel computer*. Research Monograph in Parallel and Distributed Computing. Pitman, 1989.

[6] Russ Tuck and Won Kim. MasPar MP-2 PE Chip: A Totally Cool Hot Chip. In *IEEE 1993 Hot Chips Symposium*, March 1993.