Abacus: A High-Performance Architecture for Vision

Michael Bolotski Rajeevan Amirtharajah Weip Chen
Timothy Kutscha Thomas Simon Thomas F. Knight, Jr.

MIT Artificial Intelligence Laboratory
Cambridge, MA 02139

Abstract

Abacus is a second generation distributed bit-parallel (DBP) architecture based on a reconfigurable mesh. A DPB organization distributes each bit of a data item to a different processor to exploit bit-level parallelism. This paper describes details of the VLSI-oriented architecture, the implementation of several arithmetic operations, and performance figures for various early vision tasks. A single IC in a non-aggressive technology is capable of 1-5 billion 16-bit operations per second (GOPS). A prototype machine currently under construction, with a completion date of October 1994, is expected to deliver approximately 250 GOPS.

1 Introduction

By now it is clear that the enormous computing requirements of image processing and early vision can only be satisfied by parallel computers. Existing architectures do not take full advantage of the potential parallelism inherent in image understanding. Coarse-grained commercial microprocessors perform complex arithmetic operations, but they transform 64 bits per cycle at most. Even large parallel arrays of these processors affect mere hundreds of pixels per cycle. Massively parallel SIMD networks of bit-serial processors operate on more data words, but change only one bit of a data word per cycle.

In an earlier paper, we introduced the idea of a *distributed bit-parallel* (DBP) architecture, in which a simple processing element is allocated to each bit of a data word [3]. In this work

Acknowledgments: This research is supported in part by the Advanced Research Projects Agency under contract DABT63-92-C-0039.

we describe the Abacus machine, a high-performance implementation of a DBP architecture. A single Abacus IC is expected to attain 1-5 giga-operations per second (GOPS) on 16-bit arithmetic operations such as addition, shift, and accumulation. This is approximately 20 to 100 times faster than microprocessors implemented in a comparable VLSI technology. As a result of this increased computational performance, the energy expended per operation is very low, on the order of 1-4 mJ/MOP. A 128 by 128 pixel array will fit in a $2'$ cube, dissipate 1 KW, and deliver 100-500 GOPS.

The performance is due to a combination of architecture and algorithms. In particular, we *maximize the number of bits transformed in each cycle.* This design imperative motivates a variety of architectural and algorithmic factors. The architecture is bit–parallel, yet very compact, since circuits such as shifters and adders are emulated in software. As a result of this emulation, operations can be performed on almost arbitrary word sizes, allowing precision to be matched to that required by the task. No power or silicon area is wasted on computing with noise. Due to the very fine–level control over bit operations, novel efficient arithmetic and communication primitives become possible, as do optimizations within a sequence of these operations.

From a resource consumption viewpoint, massive parallel computation is best provided by a single-instruction multiple-data stream (SIMD) architecture. The Abacus architecture consists of one-bit processors, connected in a reconfigurable mesh network. The individual components are mostly conventional, and the architecture could easily be misinterpreted as yet another bit-serial mesh machine. The distinctions are subtle, but result in a significant performance advantage over bit-serial systems.

The key observation is that typical bit-serial processor elements (PEs) dedicate much more silicon area to memory than to processing. This occurs because data–parallel algorithms frequently require storage of tens of 16-bit quantities at each site. Since a minimal one-bit ALU can be implemented in the same area as approximately 20 memory bits, the memory to ALU ratio is at least 16. The idea of the DBP architecture is to replace a single PE with a large memory by several PEs with smaller memories, and allocate a PE to each bit of a word. As a result, the entire data word can be transformed in parallel, increasing the performance by a factor equal to the word length, with only a small increase in area. In the example above, a group of 16 PEs with 32 bits of memory each, requires only twice the area of the original PE, while delivering as much as 16 times the performance.

We call this data organization *distributed bit–parallel* (DBP), as each data word is distributed among a group of PEs, and is operated on in parallel. Such a group operating on a single word is called a *cluster*. Cluster organization is purely a software construct, and the PEs can be grouped to manipulate data of various word sizes.

In the body of the paper, we describe related work, describe the architecture and algorithms (both arithmetic and communication). We then examine the performance on several early vision algorithms.

2 Previous Work

Massively parallel architectures, especially mesh–based one-bit SIMD machines, have been the topic of significant research. Early examples include Goodyear's MPP [11], NCR's GAPP [5] , and ICL's DAP [6]. Subsequent commercial vendors such as Thinking Machines and Maspar augmented the simple one-bit mesh with hypercube-style communication networks, and floating-point support.

There are also several research designs which have not been put into commercial production. These designs explored different aspects of SIMD architectures. For example, IBM's polymorphic torus concentrated on adding connection autonomy by the addition of locally reconfigurable network switches. The MCNC Blitzen project [2] updated the original MPP design for VLSI technology by adding on-chip RAM, local modification of addresses, and an X-grid 8-neighbor interconnect. A unique Some/None network intended for associative processing was stressed in the CAAAP architecture developed at the University of Massachusetts at Amherst. As in the polymorphic torus, network switches were locally controlled, but groups of processors could be electrically connected. This feature allowed efficient connected components and global statistics algorithms.

We introduced the DBP architecture in [1]. As discussed in that work, none of the earlier architectures are suitable for DPB operation which requires very fast reconfiguration, very low ALU and network circuitry area overhead, as well as sub-cycle communication time. For example, the polymorphic torus and the CAAPP use one-bit PEs and therefore require multiple cycles to change their network configuration. An implementation of the Silt DBP architecture was described in [3]. Like the actual implementation of the polymorphic torus, Silt only had horizontal and vertical bypass switches rather than a full crossbar. As a result, connection patterns that turned corners had to be handled by the software stepping through all four directions at every corner. The Abacus architecture described in the following sections removes this restriction as well as incorporating external memory and data interfaces as part of the architecture.

3 Architecture

The architecture reflects the interaction and constraints of VLSI, arithmetic algorithms, and vision tasks. The natural development of the architecture is therefore from the bottom up, from the processing element, to the interconnection network, the external memory and data interface, to the final system configuration.

3.1 The Processing Element

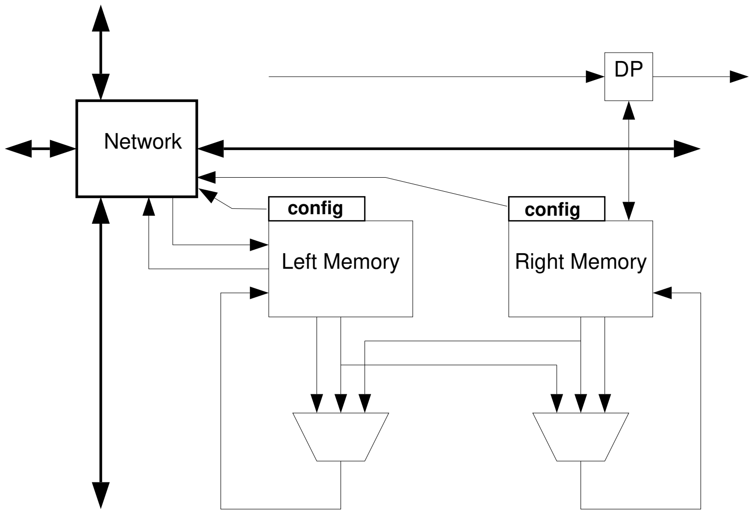

Figure 1: Processing Element Block Diagram.

The processing element is extremely simple. It contains 64 bits of dual-ported memory arranged in two banks, so that four bits can be read and two bits can be written per cycle. The PE has two 3-input ALUs, each of which takes two inputs from its memory bank and one input from the other bank. The two banks result in significant processing advantages. First, the potential computation rate is doubled. Second, three-input ALUs become possible, since there are four memory bits being read each cycle. In effect, this the first "super-scalar" bit-serial processor.

Several special control registers are incorporated in the memory banks. The *active* register common to most SIMD architectures is required to support conditional operations. When the register is cleared, the PE is inactive; when set, operations proceed as usual. Conditionals are implemented by setting the active register to the result of the conditional test, executing the `if-true` branch, toggling the active register, executing the `else` branch, and resetting the active register.

Three other registers control network operation, and two more are used for the network and data plane interfaces. Of the memory 64 bits, 58 are left to the programmer.

An optimized VLSI implementation resulted in a PE size of 80 by 425 microns in a non-aggressive (by 1994 standards) 0.8/1 micron technology available through the MOSIS service. At the worst case corner, simulations show that a 8 ns cycle time can be easily sustained.

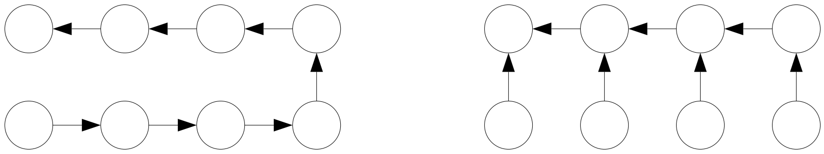

Figure 2: Network configuration possibilities. The snake mode on the left is suitable for arithmetic operations, while the tree mode on the right is more suited for cluster-wide logical operations

3.2 The Interconnection Network

At each PE, the network is composed of a wired-OR bus and four isolating switches. The PE can select its input from the far side of each switch, so that with all switches open, the network operates as a simple nearest-neighbor mesh. Two network control bits specify which of the four directions is connected to the PE input.

Each PE can also close the switch in the read direction. The *connect* bit controls whether the switch is open or closed; the other three switches remain open unless closed by a neighboring PE. A switch is closed if either of the PEs on either side of it request a connection. Connected processors now form a multiple-writer wired-OR bus: if any PE on the bus writes a one, all readers will receive a one. If no processors write a one, the bus remains at zero.

These three configuration registers are sufficient to allow tree topologies as in Figure 2. For example, PEs in a square cluster can be configured in a snake-like nearest-neighbor mode for arithmetic operations. Alternatively, to reduce propagation delay for an associative operation, a tree configuration is preferred.

The VLSI implementation of the network consists of a pre-charged bus which is pulled down by any PE writing a one. Inversion on the network input restores the appropriate polarity. Quadratic RC propagation delay is eliminated by a local accelerator circuit at each node. As soon as the voltage drops by a threshold, the circuit activates and pulls the node down completely.

At the worst case processing corner, simulations show that a message can propagate through 18 PEs in a single 8 nanosecond cycle. Thus, cluster-wide operations must insert appropriate no-operation cycles to ensure data validity. Fortunately, useful computation can be done while waiting for the propagation, as long as it does not involve the network.

Chip boundaries present a special problem at the target operating speed of 125 MHz. On-chip signals must be buffered up to cross chip boundaries, and our current I/O pads do not allow simultaneous bi-directional propagation of information. As a result, the wired-OR operation is handled by software at chip boundaries; bits are sent first in one direction, then the other, and then enabled onto the wired-OR bus. It is our view that modern high-speed architectures will not be able to abstract away physical realities without paying a high performance penalty.

3.3 External Memory Interface

The on-chip local memory will be exhausted by many algorithms. The next level of the memory hierarchy is a set of DRAM chips located adjacent to each processor chip. Data from the external memory is delivered to the PEs without interrupting computation by shifting it through a chain of shift registers, one at each PE. When the data has been shifted through the 32-element chain, the PEs can access the data plane register as if it were an ordinary memory location.

An on-chip subsystem called the *Data Plane Manager*, or DPM, incorporates a DRAM controller and controls the shifting operation. The latency of an external memory access is approximately 40 cycles (32 for the background shift, plus another 8 for DRAM initialization). The Abacus compiler schedules loads 40 cycles in advance of when the data is required. If the external memory is referenced no more than once in 40 cycles, each PE operates as if all external DRAM memory was immediately accessible, with no speed degradation.

This very high bandwidth is achieved by using specialized 35 ns DRAM which incorporates a 15 ns SRAM cache. Since memory accesses involve contiguous addresses in 1-Kbit chunks, every access will result in a cache hit, and only the initial miss requires the full 35 ns. We use two interleaved 32-bit wide memory modules to match the 8 ns cycle time of the Abacus chip.

3.4 External Data Interface

Image data I/O rate is frequently a critical system performance parameter for fine-grained parallel machines, as the amount of computation becomes limited by the data rate. The data I/O system uses an 8-bit data port on each chip to transfer information in and out. This approach requires 128 cycles to load an image all 1024 processors on a chip. However, this loading occurs in the background and is overlapped with computation of the next frame. The latency is only 1 microsecond. Even multiple-image applications such as stereo, require at most 5 microseconds from a frame time of 33 milliseconds (60 Hz). The bottleneck is in the data system that must deliver the data to the array, not in the array itself.

3.5 Prototype Machine

The prototype consists of a machine capable of processing a 128×128 pixel image at an aggregate computing power of 100-500 GOPS. The machine is composed of the parallel

Abacus array interfaced to an engineering workstation, a video camera, and a frame buffer. The array consists of 16 boards, each containing 16 ICs, with each IC containing 1024 processing elements (PEs). Each IC is backed by 2 MB of DRAM, for a system total of 512 MB. Controlling the array is an embedded microprocessor, the Intel i960, which has its own local memory, and communicates with a Sun SPARCstation via an S-Bus interface.

4 Algorithms

The discussion in this section proceeds from the arithmetic operations which are the building blocks of higher-level algorithms, to communication primitives, to the early vision tasks themselves.

4.1 Arithmetic Algorithms

We have developed several arithmetic algorithms which take advantage of the available bit–level parallelism. This section describes DBP implementation of operations such as `shift`, `compare` and `add`. These operations can be very efficient: arithmetic shift requires only two cycles, as each PE replaces its bit with its neighbor's; the common operation of accumulating a sum can be done in three cycles with carry-save addition, *independent of word length*.

Before implementing any arithmetic algorithm, we must decide on the mapping of data bits onto the PEs in a cluster. The Abacus arithmetic libraries written so far map a word onto rectangular or square arrangement of PEs. A square organization minimizes the average intra-cluster communication distance: movement of data between clusters requires $\sqrt{k}$ instructions rather than an average of $(k+1)/2$. In effect, the data moves over a parallel bus of width $\sqrt{k}$. Within the rectangle, network connections are arranged in a snake configuration so that topologically, the data item forms a line. Logically adjacent bits are also electrically adjacent. Throughout the description of the algorithms, the cluster will be shown as a line.

Each PE is also labelled with its position in the cluster. This labelling is necessary, as in most arithmetic algorithms, PEs behave differently depending on their position in the cluster. For example, during a left shift operation the PE responsible for the LSB clears its value. Also, network configurations can be quickly computed from the position.

Throughout the discussion of arithmetic algorithms, the timings do *not* reflect the cost of reconfiguring the network. The relative cost may be substantial, since many arithmetic operations require only one or two cycles. For example, if clusters are configured to read from the MSB direction (as in a shift-right instruction), and the next instruction requires them to read from the LSB direction (as in an accumulate instruction) , a full cycle must be spent reconfiguring. The A-Scheme compiler attempts to group instructions to minimize the cost of reconfiguration.

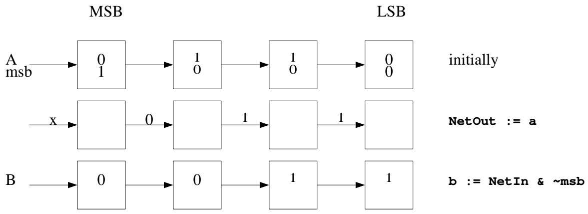

Figure 3: Logical shift right. The number 6 is shifted down to become 3. The arithmetic shift requires an additional step, as the MSB must be retained.

Logical Shift The simplest DBP arithmetic operation is the shift. As shown in Figure 3, each processor simply replaces its bit with a neighbor's. In the case of logical shifts, the MSB or LSB is cleared; for arithmetic shifts the MSB is retained. This operation is very fast, requiring only constant time, and is used as a component of other arithmetic algorithms such as multiplication and carry-save addition.

Since the clusters are in reality square or rectangular, there are interesting DBP variations of the shift operation. If the shift distance is a multiple of the cluster row size, the bits need only move vertically. Thus, a shift by 9 can be accomplished by two vertical shifts by 4, followed by a conventional snake shift by 1.

Addition Fast addition relies on quick computation of the carry bit. A DBP algorithm based on the the well-known logarithmic-time technique was described in [4]. 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 technique 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 in theory linear with the number of bits, the constant factors result in faster operation than the logarithmic-time circuit.

Sum Accumulation One of the most common arithmetic operations is accumulating a sum. The sum of a sequence of n numbers can be evaluated in $\Theta(n)$ cycles with a software version of the carry-save adder (CSA) circuit. The CSA adder computes the sum and carry bits separately for each addition. The computation is purely local, and the delay of each

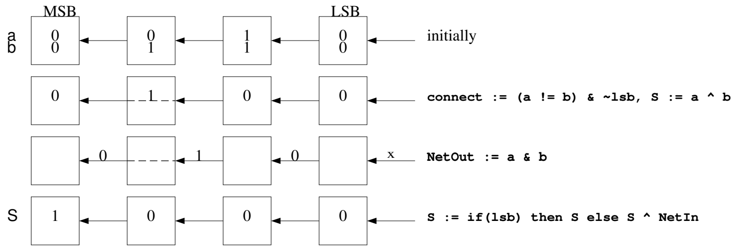

Figure 4: Manchester-carry addition. The final `if-then` function appears complicated, yet is a simple three-input Boolean function.

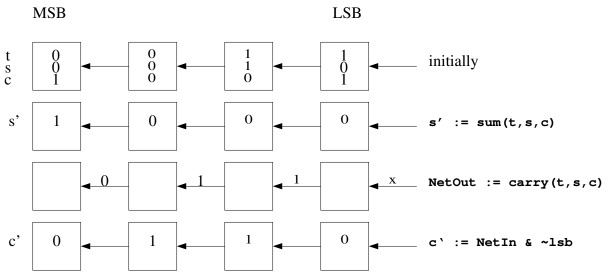

Figure 5: Accumulate. The addition of the number 3 to a sum of 11 represented as $2 + 9$. After the operation, the sum is 14, represented as 6+8. All operations are purely local or nearest neighbor.

intermediate CSA stage is therefore independent of word size. At the end of the operation the carry and sum values must be summed with a full carry-propagation adder, but this delay is amortized over many summations.

Many computations which are usually thought of as additions can be reformulated as accumulations. As a result, the speed of accumulation significantly affects the performance of many algorithms, such as region summing, counter updating, or multiplication. 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.

Multiplication A fast integer multiplication algorithm known as the modified Booth algorithm [10] is suitable for bit-parallel implementation. It uses a recoding technique to cut in half the number of required additions. A single iteration examines three adjacent bits of the multiplier and adds 0, ± 1, or ± 2 times the multiplicand to the sum. The multiplicand is then shifted by two places, and the sequence is repeated $\frac{k}{2}$ times. At the end, a full add operation converts carry-save accumulator to standard form. The multiplication of two

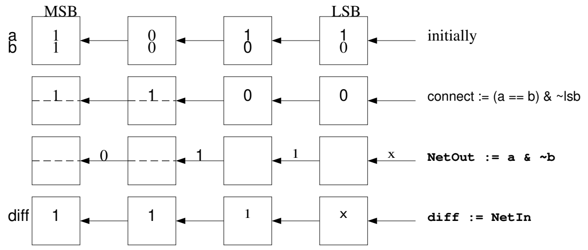

Figure 6: The `compare` operation. After the operation completes, the MSB PE has the result of the test $A > B$. Boundary cases of the LSB or MSB being the deciding bit value are omitted for clarity.

16-bit numbers to form a 32-bit result requires 655 cycles, while multiplication of two 8-bit numbers requires 352 cycles.

Compare. The previous two algorithms used purely nearest neighbor communication. The comparison algorithm takes advantage of the bypass capabilities of the network. It is based on the observation that the most significant differing bit (MSDB) between two words determines which word is greater. Thus, $A > B$ when the MSDB is 1.

The algorithm is straightforward: all PEs with identical bits bypass themselves and then send the value of their bit of A up to the MSB. If the MSB receives a 0, then A is smaller then B, otherwise A is greater. There are some fine details having to do with boundary conditions at the LSB and MSB, but they do not add significantly to the execution time.

4.2 Scan Operations

A set of computational primitives known as *parallel prefix* or *scan* operations have been found to be very useful for programming parallel machines. Scan operations are proving to be a useful primitive across different machine architectures, improving portability of parallel algorithms. These operations take a binary 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 which reduce the values in a vector to a single value using a binary associative operator. For example, *+-reduce* calculates the sum of a vector. Examples of these primitives are shown below.

Scan Algorithm Performance A message can propagate across an entire chip in 2 cycles. Crossing a chip boundary requires an additional cycle of latency. The prototype is a 16 by

$$
\begin{array}{lcl}
A & = & [4 \quad 1 \quad 7 \quad 8 \quad 3 \quad 2 \quad 1 \quad 5] \\
\textit{+-scan} & = & [4 \quad 5 \quad 12 \quad 20 \quad 23 \quad 25 \quad 26 \quad 31] \\
\textit{max-scan} & = & [4 \quad 4 \quad 7 \quad 8 \quad 8 \quad 8 \quad 8 \quad 8] \\
\textit{+-reduce} & = & 31
\end{array}
$$

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 between approximately 500 cycles, or 4 microseconds. This can be optimized further by pipelining bit transmission.

4.3 Vision Performance

We described a number of vision algorithms on the Silt architecture in [4]. The Abacus machine alters implementation details of only the arithmetic operations; the vision algorithms remained unchanged.

Edge Detection The widely used Marr-Hildreth edge detection algorithm consists of filtering the image with a Gaussian filter, computing the Laplacian of the filtered image, and locating the zero crossings of the result [8]. Convolution with a Gaussian can be approximated by repeated convolution with a triangular filter, with weights $\frac{1}{4}$, $\frac{1}{2}$, and $\frac{1}{4}$, requiring only arithmetic shifts and accumulates. Since the Gaussian filter is separable, it can be implemented by two one-dimensional convolutions. Further, since zero crossings are independent of absolute magnitude, the expensive scaling operation of the Laplacian kernel is not required.

The algorithm was implemented with 16-bit clusters to maintain precision when adding and dividing 8-bit intensity values. One iteration requires 103 cycles. The calculation of the zero crossings of convolution with a discrete 3×3 Laplacian kernel required approximately 300 cycles. Total edge detection time for a Gaussian with a σ of 1.58, requiring four iterations of the triangle filter, is approximately 500 cycles, or 4 microseconds at a 125 MHz clock rate.

Surface Reconstruction Surface reconstruction consists of determining the surface shape (height and slope) from a set of potentially sparse and noisy measurements. Harris [7] introduced the coupled depth/slope model for surface reconstruction and developed an iterative solution technique suitable for a mesh-based massively parallel computer. The update equations in each iteration are simple and local and consist of five or six additions and subtractions, followed by a division.

The summation of multiple values at a pixel is computed efficiently by the multi-operand addition algorithm. Division is expensive on a DBP architecture, as it is on a word-parallel bit-serial machine. This algorithm only requires division by a constant, which can be implemented by a special purpose sequence of shifts and adds. A single iteration of the algorithm requires approximately 250 cycles.

Optical Flow In correlation-based optical flow [9] the original image is displaced in a variety of directions, and at each displacement the difference between the two images is computed, and summed over a window. If the images are processed to produce binary image features, the difference is the exclusive or of the image pixels in the 11 by 11 neighborhood. Finally, each pixel chooses the layer with the smallest difference value in a winner take all (WTA) step. Images manipulated by the algorithm may consist either of simple brightness values or of more complex features such as edges. This algorithm has been implemented on the simulator for the binary feature case. The edge detection algorithm discussed earlier could be used to obtain the features (in this case edges) from a raw intensity image. The time requirement for a displacement of 2 pixels and a 5×5 summation region is approximately 3000 cycles.

4.4 Performance Summary

In our design, 1024 PEs can fit on a small, 7mm by 7mm IC implemented in a 0.8 micron technology. Each PE is very simple, resulting in a sustainable clock rate of 125 MHz. At this speed, each IC is capable of 4×10^9 16-bit accumulate operations per cycle. A summary of expected performance of a single IC on various computations is shown below.

Operation	8-bit		16-bit		32-bit	
	Cycles	GOPS	Cycles	GOPS	Cycles	GOPS
Add	4	4.0	4	2.0	5	0.7
Shift	2	8.0	2	4.0	2	2.0
Accumulate	3	5.2	3	2.6	3	1.3
Move	3	5.2	4	2.0	6	0.6
Compare	6	2.6	11	0.6	12	0.2
Multiply (16×16)					180	0.03

Algorithm	Cycles	Time (μsec)
Edge Detection $\sigma = 1.6$	380	3
Optical Flow, $\Delta = 2$, 5×5 region	3000	24
Surface Reconstruction (1 iteration)	370	3

5 Conclusions and On-going Work

The DBP architecture has been shown to be amenable to a very high performance implementation. We have demonstrated that many important arithmetic and communication operations can be accelerated with DBP methods. The usefulness of the accumulate primitive suggests that there may be arithmetic and logical operations which have no counterpart on traditional architectures. Further, the peephole micro-optimizations can occur between what are typically considered to be atomic operations. For example, a broadcast operation can be overlapped with an addition, so that while bits are propagating, the ALUs can start on the next operation. Similarly, when one of the two ALUs is idle due to data dependency in the current operation, the other can start on the following operation.

From the implementation viewpoint, we hope to demonstrate that the regular nature and simplicity of the Abacus SIMD array allows high-speed (125 MHz) system operation. Even at these clock rates, the external memory subsystem has been designed to deliver data with only a latency penalty.

From a theoretical viewpoint, the Abacus implementation provides a set of realistic constants that can be used to evaluate reconfigurable mesh algorithms. The aggressive (in terms of speed and integration) nature of the design explores physical limitations on architecture that will become more significant in the future. For example, it is becoming obvious that chip boundaries cannot be abstracted away in the interests of software regularity.

We are currently investigating other interesting issues such as multiple networks to reduce the need for reconfiguration, indirect addressing during external memory accesses. An exciting development is the possibility of integrating a high-speed (200+ MHz) multi-stage interconnection network port into each chip to augment the reconfigurable mesh with hypercube-like communication.

References

[1] R. A. Barman, M. Bolotski, D. Camporese, and J. J. Little. Silt: The Bit-Parallel Approach. In *International Conference on Pattern Recognition*. IEEE, 1990.

[2] D. W. Blevins, E. W. Davis, R. A. Heaton, and J. H. Reif. Blitzen: A Highly Integrated Massively Parallel Machine. In *Frontiers of Parallel Computation '88*, 1988.

[3] M. Bolotski. Distributed Bit-Parallel Architecture and Algorithms for Early Vision. Master's thesis, University of British Columbia, Aug 1990.

[4] M. Bolotski, R. Barman, J. J. Little, and D. Camporese. Silt: A Distributed Bit-Parallel Architecture for Early Vision. *International Journal of Computer Vision*, 11(1):63–74, Oct 1993.

[5] E. L. Cloud. The Geometric Arithmetic Parallel Processor. In *Frontiers of Parallel Computation '88*, 1988.

[6] T. Fountain. *Processor Arrays: Architectures and Applications*. Harcourt Brace Jovanovich, 1987.

[7] J. G. Harris. The Coupled Depth/Slope Approach To Surface Reconstruction. Master's thesis, MIT, May 1986.

[8] B. K. P. Horn. *Robot Vision*. MIT Electrical Engineering and Computer Science Series. The MIT Press, Cambridge, Massachusetts, 1986.

[9] J. J. Little and H. H. Bülthoff. Parallel Optical Flow Using Local Voting. AI Memo 929, MIT, July 1988.

[10] O. L. MacSorley. High speed Arithmetic in Binary Computers. *Proceedings of the IRE*, 49:67–91, 1961.

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