Chapter 1

Introduction

Computer vision, the understanding of images by a computer, is both an exciting component of artificial intelligence and a daunting computational task considered as one of the Grand Challenge problems of parallel processing. The scope of the challenge becomes apparent when the computational requirements of early vision are considered. A moderately sized image contains approximately 65,000 pixels, each of which may be used a hundred times in a computation. A single vision algorithm may require ten iterations per image, and as many as fifty algorithms may be required to extract essential information such as depth, motion, shapes, and shadows. All of this processing must be repeated thirty times per second to sustain real time response. The aggregate sustained computing power is therefore on the order of 100 billion operations per second (GOPS).

Hundreds of parallel system designs have been proposed to attain this level of performance. These systems can be broadly categorized as Single-Instruction Multiple Data (SIMD) machines that consist of identical processing elements (PEs) executing the same instructions in lockstep, and Multiple-Instruction Multiple-Data (MIMD) machines, whose PEs execute independent programs and use explicit communication to synchronize operations. Historically, SIMD machines have been built with large numbers of simple PEs, while MIMD machines relied on a small number of complex PEs.

Currently, the most prevalent parallel architectures are MIMD machines consisting of collections of workstation-class RISC processors. Indeed, only one major SIMD vendor, MasPar, still exists, while there are many MIMD vendors, including Cray, Convex, IBM, and Intel. This overwhelming commercial success is due to the constant improvement in microprocessor performance, driven by marketplace pressure and fueled by enormous investment.

Parallel MIMD machines are excellent general–purpose computing and development platforms, since they come equipped with the software inherited from their workstation heritage. As with the uniprocessors on which they are based, these machines excel at computations with irregular and unpredictable instruction streams. Unfortunately, the price of this flexibility is reduced performance. On problems with a data–parallel, highly regular structure, such as image processing, these processors continue to be unnecessarily flexible at every instruction step.

The MIMD model incurs enormous overhead for every functional unit that actually manipulates data. This overhead consists of silicon area required to store the program (a 10K word program of 32 bits each corresponds to 1.8 million transistors!), instruction decoding, addressing, and control circuitry. The overhead can easily be a factor of ten larger than the functional unit actually processing the data.

Despite this apparently inherent advantage, there are few SIMD systems in existence. The reason for the lack of widespread adoption of SIMD machines becomes obvious when the clock rates of RISC and SIMD systems are compared. Figure 1-1 compares the improvement in clock rate within several families of RISC processors to (mostly academic) SIMD designs.

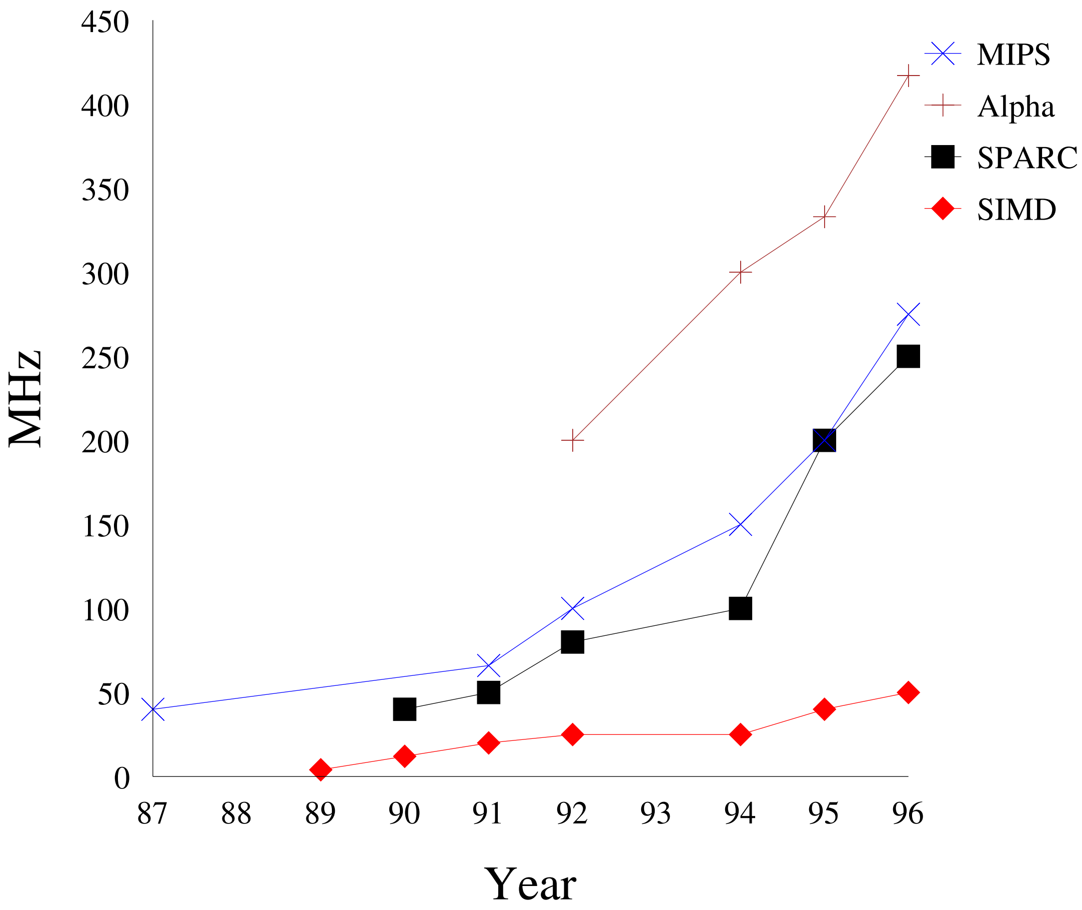

Figure 1-1: CPU and SIMD System Comparison

Thesis

The thesis of this dissertation is that *for the problem domain of early vision, parallel computers based on a SIMD reconfigurable bit-slice architecture can outperform those based on conventional processors by over an order of magnitude using the same silicon resources.*

Hopefully, the thesis will also have several positive cultural effects. It will provide a convincing demonstration of the effectiveness of SIMD processing. The advent of VLSI technology gave SIMD systems a substantial advantage over their discrete-component based uniprocessor competition. With time, this advantage disappeared as SIMD clock rates remained unnecessarily low. This work aims to equalize the clock rates of SIMD and microprocessor systems and therefore regain the performance lead. Experience obtained through the detailed design shows that the clock rate barrier is largely illusory.

Important applications of SIMD technology are near but may be hampered by the unconvincing performance and architectural deficiencies of existing SIMD processor arrays. Vision-related tasks, such as MPEG coding, OCR, paper document processing, digital picture manipulation, medical imagery, face recognition, vehicle collision avoidance, and automatic vehicle guidance all provide important practical applications which can be solved on SIMD machines. An existing system design will allow the construction of a cheap tera-op level supercomputer for vision research. Such a platform will allow important image, signal, and text processing applications to be prototyped. A successful design will encourage research on specialized architectures, which has been decreasing recently due to the success of RISC-based MIMD machines. Finally, the project will demonstrate that high performance design can be done in a university environment.

The research described in this document develops a reconfigurable architecture optimized for high speed VLSI, completes an aggressive implementation of the architecture, and evaluates the performance on standard computer vision algorithms. Parameters from the implementation are then used to investigate an analytical performance model. The existence of a concrete instantiation of the architecture allows empirical evaluation of the costs and benefits of reconfiguration.

This thesis demonstrates the performance advantages of a high-speed VLSI reconfigurable bit-parallel (RBP) SIMD architecture on early vision tasks. The overall research approach is to:

1. Design, implement, and test a VLSI chip containing Abacus processing elements. A working chip will both provide architectural parameters for an analytical model and unearth any inaccurate assumptions about high speed SIMD operations.

2. Complete a system-wide design to ensure that there are no unexpected system-level bottlenecks.

3. Use the IC implementation experience and the analytical model to suggest a design for a second generation architecture.

The expected contributions of this work can be grouped under three categories: implementational, analytical, and cultural. A successful implementation will demonstrate a SIMD chip with the highest clock rate to date. It will force the development of system interfaces generally ignored in the academic field. This development is absolutely essential as input/output interfaces may prove to be the bottleneck in high speed designs.

The analytical model allows the selection of an optimal point in implementation space as technology parameters vary. For example, novel high-bandwidth devices such as Rambus memory components may allow reduction in the amount of on-chip memory. The model will allow quantification of these tradeoffs.

Abacus

The Abacus architecture can be classified as a reconfigurable bit-parallel (RBP) machine. The key idea of a RBP design is to provide programmable interconnect to link multiple PEs into a single Processing Site (PS) that can operate on several bits of a data word simultaneously. Reconfiguration allows word width to be matched to the need of each algorithm. Even more significantly, on-chip memory capacity at each processing site can also be adjusted by changing the number of PEs in a PS. Since off-chip memory references can be a factor of 20 slower than on-chip accesses, decreasing the miss rate at the cost of ALU capacity may lead to substantial performance increases for many algorithms.

Reconfigurable processors provide a smooth transition between the two extremes of a slow, small bit-serial PE and a complex, fast bit-parallel PE. The tradeoff between the two 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. RBP designs retain bit-serial flexibility and bit-parallel performance.[1]

Organization

Chapter 2 presents an overview of the substantial body of previous work relating to the topics of this thesis, from SIMD machines to modern architectural alternatives: the SRC PIM chip, Berkeley's PADDI-2 DSP, and FPGA based computing platforms. After demonstrating the the SIMD approach is competetive if the clock speed can be raised above previous levels, the discussion turns to issues of high-speed SIMD system designs. The chapter concludes with a review of related architectural performance evaluations.

Chapter 3 describes in detail the motivations for an RBP organization, RBP arithmetic algorithms, and specifics of the Abacus design. Chapter 4 describes the implementation of the Abacus-1 chip at a fairly detailed level, as well as chip test results. Chapter 5 describes the system-level design of the Abacus machine, including the approach to issues of high-speed SIMD operation.

[1] Carlin says "Crap"

Chapter 6 is the performance evaluation. It introduces the benchmark algorithms used to evaluate the architectural tradeoffs and evaluates architectural tradeoffs such as local memory size, ALU width, off-chip memory bandwidth and network bandwidth, based on instruction traces from the parallel algorithms.

Chapter 7 describes the lessons learned from the design of the Abacus-1 architecture, and the set of changes required to produce a smaller yet more powerful design. Looking farther ahead, it introduces modifications that allow a redesigned Abacus element to function in multiple-SIMD mode, as a systolic processor, or to emulate random logic circuits effectively.

Chapter 8 concludes with the implications of what has been learned from the design and suggests avenues for future research.

Chapter 2

Previous and Related Work

Many systems have been developed to perform vision and image processing. The performance of the Abacus-1 design must be considered in the context of other major approaches. When the clock rate is boosted with aggressive design, SIMD machines are more than competetive with other approaches. A survey of previous work on the challenges of high-speed SIMD design indicates that an effective high speed design has not yet been worked out. Once such a design is developed, the processor/memory bottleneck becomes apparent. Dynamic reconfiguration may be used for tuning on-chip memory size to the needs of a particular algorithm at run-time. After analyzing previous reconfigurable machines, it is clear that none of them have been optimized for this sort of reconfiguration.

2.1 Architectural Design

There has been a considerable amount of research on architectures for image processing and image understanding. A variety of processor designs and interconnection networks have been proposed (but few have been implemented). This review concentrates on SIMD designs and on alternate approaches with comparable performance.

Early examples of machines designed for image processing include Goodyear's MPP[1], NCR's GAPP [2], and ICL's DAP [3]. The most widely produced SIMD machine was Thinking Machines' CM-2. A contemporary commercial product was produced by MasPar [4], and recently upgraded to the MP-2 [5].

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 [6] concentrated on adding connection autonomy by the addition of locally reconfigurable network switches. The MCNC Blitzen project [7] 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 [8] developed at the University of Massachusetts at Amherst. As in the polymorphic torus, network switches were locally

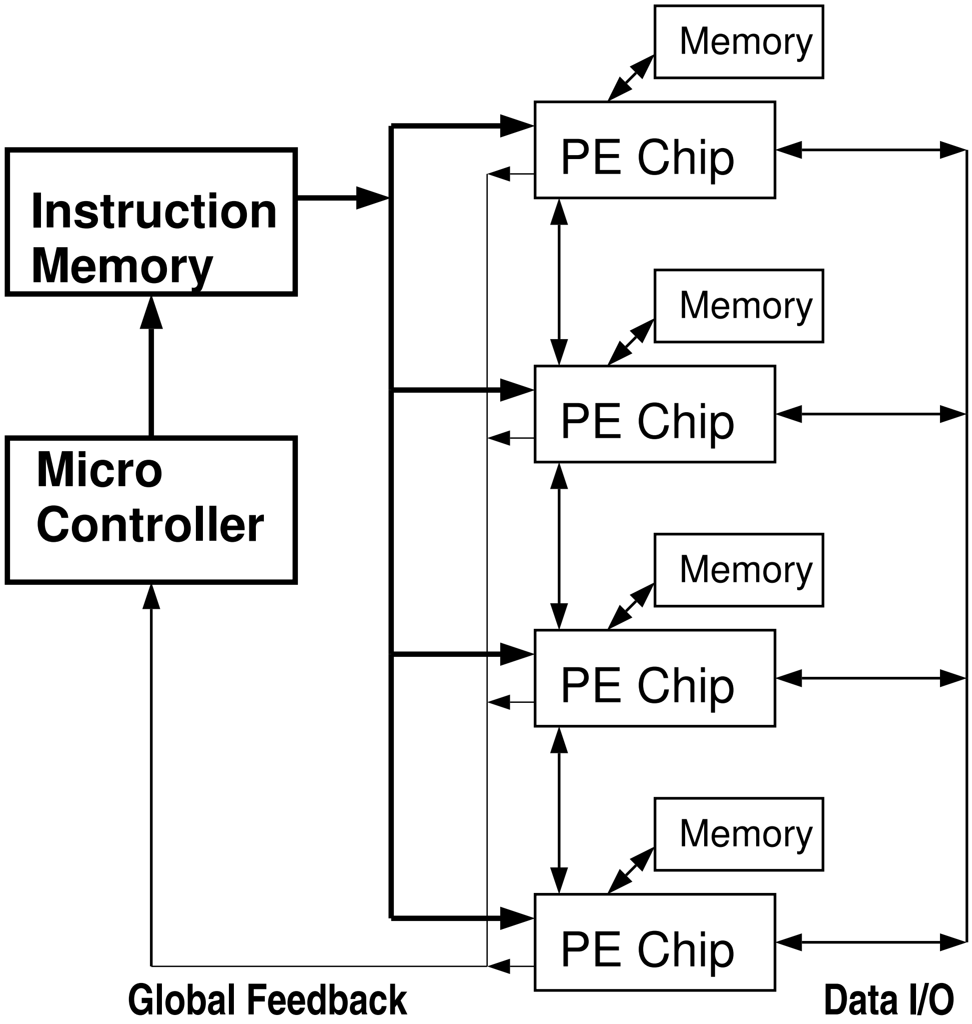

Figure 2-1: SIMD System Block Diagram. The five interfaces shown are: local memory, instruction distribution, global feedback, inter-chip communiction, and data input/output. Although the PEs are shown arranged in an array, most architecture organize them in a 2-dimensional mesh.

controlled, but groups of processors could be electrically connected. This feature allowed efficient connected components and global statistics algorithms.

A number of recent research efforts promise to achieve substantially higher performance than these early designs. A 256-element SIMD array driven by an on-chip RISC-like processor has been designed at the University of Sheffield [9]. The performance is reasonable, given the number of processors, but is limited by the instruction issue rate of the RISC controller, the low I/O bandwidth, and the lack of off-chip memory support.The Supercomputer Research Center has developed a processor-in-memory chip which augments a 128 Kbit SRAM array with bit-serial processors at each row [10]. The performance of this design is rather low due to the small number of PEs and modest clock rate.

An alternate processor-in-memory MIMD/SIMD architecture has been developed by IBM [11]. The Execube chip incorporates 8 16-bit PEs, each with 64 KB of DRAM memory, and can be operated in SIMD mode from a global instruction bus. The performance is again rather low due to the long cycle time of DRAM and because instructions are stored in the same memory as data.

An integrated micro-MIMD chip, integrating 48 16-bit processors and an interconnection network on each chip has been designed at UC Berkeley [12]. The IC delivers an impressive 2.4 GOPS on a variety of DSP algorithms and is designed to communicate efficiently with other chips. However, since each PE's instruction memory is only 8 words deep, with no provision for expansion, the architecture is not suitable for more complex algorithms.

A class of DSP chips, most notably the Texas Instruments C40 and the Analog Devices SHARC processors, have been designed to support efficient execution over data arrays and inter-processor communication. These appear more promising than the RISC based systems, but still lack performance.

Another technology that delivers comparable performance to Abacus and PADDI is the use of reconfigurable logic as exemplified by Field Programmable Gate Arrays. An array of programmable logic can be used to configure applications specific hardware and thereby obtain excellent performance. For example, research at DEC Paris have implemented algorithms ranging from Laplace filters to binary convolutions [13]. If the per-chip performance is computed as aggregate performance divided by the number of FPGA chips in the system, each Xilinx chip delivers approximately 500 million 16-bit operations per second.

Machine	Performance 16-bit MOPS	Clock MHz	I/O BW MB/sec	Mem BW Internal	Mem BW External	Tech μ
TMC CM-2	8	8	8	32	16	1.5
SRC TeraSys	20	10	5	80	5	1
IBM EXECUBE	50	25	300	400	0	.8
TI C40	50	50	200	0	200	.8
MasPar MP-2	133	12.5	45	1600	45	1
HP8000	200	200	800	0	960	.5
Sheffield DIP	320	40	160	2560	0	1
Xilinx XC3090	500	20	NA	NA	NA	1
Berkeley PADDI-2	2400	50	800	14400	0	1
MIT Abacus-1	**2000**	**125**	**1000**	**32000**	**512**	**1**

Table 2.1: Architectural performance comparison

It is unfair to directly compare the 16-bit performance of Abacus compared to the wider-width Execube and MasPar PEs, as they incorporate floating-point support. However, even granting a factor of five in area, their performance still lags Abacus. Both of these designs make the mistake of using single-ported register files. The implication of requiring three cycles to perform a single instruction is that three times the area could have been allocated to triple-porting the memory. And the area in this case refers to the *entire* silicon area, including communication pads and support circuitry.

The only architecture comparable in performance is the Berkeley PADDI-2 chip, and it is not suitable as a general computing element as it can only store eight instructions per PE. Of course, when the computation task can be expressed as piping data through a systolic array, the PADDI is an ideal high-performance, low chip-count solution.

No other research SIMD machine in the literature even approaches the Abacus design in terms of performance per silicon area on low-precision integer operations. The advantage occurs for a variety of reasons, including an aggressive clock rate of 125 MHz, made possible by careful custom design, high density manual circuit layout, and the inherent advantage of RBP operations.

2.2 High-Speed SIMD Issues

Recently a number of researchers have started to address system-level high-speed SIMD design issues [14, 15, 16].[1] The consensus problems are clock distribution, instruction generation and broadcasting, non-local data access.

Clock Distribution Since the clock skew budget is typically less than 10% of the clock period, a 200 MHz SIMD array must maintain skew below 500 picoseconds. To overcome this problem, Weems [16] proposes a loosely synchronized SIMD processor. This degree of complexity is somewhat surprising since commercial chip sets with tunable delay times are easily available, and the salphasic [17] clock distribution scheme has been used to provide a 160 MHz clocks to 60 system boards with less than 200 psec of skew across a 10-foot-long system.

Instruction Broadcasting Rockoff [14] believes that distribution of finely synchronized instructions to hundreds of PE chips is a fundamental limitation on the clock rate due to transmission skew. His doctoral thesis is based on the idea of using instruction caches internal to each PE chip to allow operation faster than the transmission limit. The approach appears to be a complex solution to a non-problem, since sufficiently frequent registers along the transmission path can resynchronize the instruction stream.

The cost of this retiming is increased latency in instruction distribution. Allen analyzes a tree-structured transmission and retiming model to determine an optimal number of stages. The model is mostly theoretical as it uses RC delays and clock rate as limitations instead of the actual transmission-line effects encountered at high speeds. Further, there is no good reason for different stages to have identical fanout factors, since there are differences both between inter-board and on-board wire delays, and between the receiving circuitry.

The Abacus system design presents a method for synchronized instruction distribution along widely varying transmission paths that adds programmable delay elements to the

[1] Mention Pixel-Planes 5!

instruction pads on each chip and retimes the signals on a pin-by-pin basis. This scheme allows propagation variations on the order of several cycle times with a skew equal to the clock skew.

Instruction Generation Another suggested bottleneck in SIMD performance is instruction generation. The traditional SIMD model is that instructions for operations such as 16-bit addition are sent from the host to the sequencer, where they are converted to a sequence of microinstructions to be executed by the PE array. Weems claims that between 20 and 50 RISC-like instructions are required to transform a single array instruction. The system design chapter presents three instruction generation techniques that in combination reduce the ratio of sequencer/array cycles to 1. In contrast, Weems proposes to solve the problem by substantially increasing the complexity of each PE chip to allow on-chip expansion of broadcast instructions.

Data Cache As in mainstream conventional microprocessors, high-speed implementations widen the gap between processor speed and main memory speed. The common approach is to insert a medium-speed memory between the small fast register file and the large slow main memory. SIMD machines are inherently cache-unfriendly, since *any* cache miss by any PE will cause the entire machine to stall, and with a large enough machine, such a miss is virtually guaranteed. Allen observes that as long as PEs are allowed only direct memory accesses, all PEs will either miss or hit at the same time, and therefore proposes *direct-only-data* caches. This is an interesting approach, but many vision and image processing algorithms are oblivious in the sense that their execution does not change as a function of data, and therefore the memory access patterns are known at compile time. As a result, the compiler can optimally manage the slower memory without using additional hardware at run-time.

High Latency Issues Allen points out that one of the costs of instruction synchronization via pipelined distribution is the increased latency of the array response to a sequencer command. This is an important issue, especially in the Abacus-1 design, which has a 10-stage pipeline. However, since the global OR computation already requires about 5 or 6 cycles, the extra two cycle cost of instruction pipelining is negligible, especially when the controller runs at least two (and possibly four) times slower than the parallel array. For any reasonable implementation, the pipeline cost is either zero or one one extra stall cycle per compare.

In the pathological case of a very tight loop, there is a software technique for reducing the effect of a pipeline stall. Most iterative code does not damage the solution if it executes an extra iteration or two. Our proposed solution is to unroll the loop, checking for completion only once every 3 or 4 iterations. Since iterative code usually executes for at least dozens of iterations, the extra two or three iterations at the end are reduced to a 10% cost.

2.3 Reconfigurable Architectures

The idea of dynamically changing the data path width to better match the algorithm requirements has been around since the earliest days of SIMD computing. The Illiac-IV, initiated in 1965 and completed in 1975 [18], had 64 64-bit PEs that could be partitioned as 128 32-bit PEs or as 512 8-bit PEs. Three years before its completion, work started in Britain on the Distributed Array Processor (DAP) [3], a bit-serial machine whose PEs could be configured together with a ripple-carry path to perform 64-bit arithmetic. This capability was not central to the architecture and operated with one eighth the throughput of the bit-serial mode. A prototype was produced in 1976 and a commercial model in 1980.

The first VLSI-era member of the class, the reconfigurable processor array (RPA) [19] was developed between 1985 and 1989 at the University of Southampton. Although a chip was designed and fabricated, a full scale computer was not completed. The RPA uses a mixed approach to reconfigurability. Although each PE processes two bits at a time and a data word is shared among several PEs, considerable hardware resources are devoted to supporting bit-slice operations. As a result, each PE is quite complex for a fine-grained machine.

A substantially finer-grain architecture, Silt [20], was designed at the University of British Columbia between 1988 and 1990. The design took a minimalist path, exploring the approach of minimal area and high clock speed. The Silt chip led to the Abacus design presented in this thesis. Work on Abacus began in 1991, and a full-scale chip was fabricated in 1995.

In 1992, a reconfigurable processor, MGAP, was developed at the Pennsylvania State University. Its key distinction was the orientation towards a fully-redundant radix-4 arithmetic system. A second generation design, the MGAP-2, was completed in 1996.

Mainstream processors have recently rediscovered and fully embraced reconfigurable processing. Driven by multimedia applications with similar characteristics to image processing, Sun's UltraSparc, Hewlett-Packard's PA-RISC, and Intel's MMX chips all support the partitioning of the 64-bit data path into 8 independent byte-wide processors.

2.4 Architectural Studies

SIMD machines are an attractive class of architectures to study since the effects of data path enhancement can be easily isolated from issues of control circuitry or cache organization improvements. A number of studies have been done, especially for vision applications.

2.4.1 Normalized Analysis

A methodology for comparing parallel computer performance was described by Holman and Snyder [21, 22]. They introduced the distinction between three types of analysis:

1. *cost-free* analysis, where absolute performance is the relevant metric, and the amount of consumed resources is not relevant. For example, cache studies are typically cost-free analyses since the comparison metric is absolute performance, neglecting the resource cost of the caches.

2. *budget-constrained* analysis, where different systems are compared under the constraint that they use identical amounts of hardware.

3. *normalized* analysis, where an improvement in a PE is compared to using the same hardware as the original PE, but using more processors.

The approach is based on the generally true observation that "There are two ways to improve any parallel architecture using additional hardware - by speeding up the processor elements or by adding more processor elements."

There are clearly situations in which neither approach is very effective and indeed can be counterproductive. For example, in memory-bandwidth or communication-limited designs, speeding up the PE is not going to have a substantial effect on performance. Similarly, if the computation is already fully parallellized, with one PE per data item, adding PEs cannot improve the performance.

The curious part of the work is that while analyzing optimal data path size, due to a fraction inversion error, the authors drew the exact opposite conclusion from the data. Specifically, analysis assumed that any area applied to making a PE more powerful could instead be applied to create a larger number of simple PEs, and that the resulting speedup is linear. Similarly, the area cost is linear in the size of the register file and ALU, and quadratic in the size of the shifter.

The machine composed of many simple elements is termed the XPE machine, and the machine composed of fewer complex elements is the IPE machine. The basic unmodified architecture is the BA (base architecture) machine. They derive the following performance metrics for execution time, assuming SU is the speedup provided by the enhancement, f is the fraction of instructions affected by the enhancement, and c is the chip area of the respective machines.[2]

$$T_{IPE} = T_{BA}\left(1 - f + \frac{f}{SU}\right) \tag{2.1}$$

$$T_{XPE} = T_{BA}\frac{c_{BA}}{c_{IPE}} \tag{2.2}$$

Combining the two yields

$$\text{effective}su = \frac{T_{XPE}}{T_{IPE}} = \frac{c_{BA}}{c_{IPE}}\left(1 - f + \frac{f}{SU}\right)^{-1} \tag{2.3}$$

[2]This is still kinda crappy –Carlin

To choose an example from the paper, consider a 4-bit PE and a 32-bit PE, with area ratio approximately 1:8. The Batcher implementation has $f = 0.75$, leading to $su = 0.36$ (the paper computed 0.41). This is as expected, since the area penalty was a factor of 8, while the performance improvement was only a factor of 3. Nevertheless, the authors conclude that “the best performance is obtained with a 32-bit data path”.

We will see later that the determining factor in this design choice turns out to be the amount of silicon area dedicated to the memory, and the memory access patterns of the algorithm. The Holman/Snyder model does not account for the diminished storage capacity of each simplified PE. If the algorithm needs to access 16 32-bit words per pixel for one iteration, and only 2 32-bit words are available, the machine will perform a large number of memory spills. With the growing gap between processor and memory speeds, this effect becomes critical.

2.4.2 Simulation-Based Performance Evaluation

A simulation approach to SIMD performance evaluation was taken by Herbordt [23]. His *trace compilation technique* compiled traces generated on an abstract virtual machine for a specific target architecture. The technique can be two orders of magnitude faster than a detailed simulation. This flexibility allows quick evaluations of different architectural parameters, such as register file size or communication latency.

Although the centerpiece of his dissertation is the trace compilation methodology, the tool was used to perform a number of architectural studies, and resulted in some interesting conclusions. For example, some studies showed that increasing ALU width substantially improved performance only up to a width of 8 bits. After that, performance improved only slightly.

The simulator is an excellent tool for evaluating system performance. It allows the architect to experiment with different virtualization ratios, datapath width, register file size, and off-chip memory latency. The potential danger is the unknown effect of the trace compiler quality on performance. The other inherent problem with the methodology is that algorithms were not optimized on a per-case basis: the same address trace was used for a wide variety of system parameters. Algorithms designed for the virtual machine are not necessarily optimal for a large memory or narrow datapath variant. Similarly, the virtual machine compiler generated code for a uniform memory access model, which is almost guaranteed to be suboptimal for a contemporary machine with a high disparity between on-chip and off-chip access times.

Nevertheless, many of the results in Herbordt's work can be used directly as a starting point for selecting good concatenation factors for reconfigurable architectures.

2.4.3 Dynamic Concatenation: Audet et al.

The performance advantages of reconfigurable architectures due to memory limitations were first evaluated by Audet et al [24], who referred to reconfiguration as the Dynamic Concatenation Approach. The goal of the research was to determine the optimal number of PEs to be grouped into a PU, given particular hardware and algorithmic parameters. This number is called the concatenation factor, w.

Although the critical issue of off-chip memory traffic minimization was addressed, the published work contained some shortcomings. For example, a particularly strange pipelining model was chosen, and execution times were scaled by factors based on pipelining effects. The method of deriving these parameters is based on arbitrary assumptions about compilers and hardware implementation. Quantifying pipelining effects applies more to a VLIW-style system where multiple functional units can be triggered by a single instruction. The model appears to apply to multi-cycle, relatively complex functional units, which are not usually present in simple, small SIMD PEs.

The model assumes a static type of register assignment: the most frequently used variables are placed into registers, and the rest into memory locations. Algorithms that use one variable frequently in the first half of operation, and use another during the second half, would be forced to assign different registers even if their use did not overlap in time. As the methodology used UNIX-based C profiling tools to determine the often referenced variables, it is not clear whether the compilation was optimized for the right number of registers, or whether loop unrolling optimizations were made.

Virtualization requirements were not considered and program segments appear trivially short. Further, the only program example shown in detail is of a systolic type. Short computation chains between virtual PE synchronizations could lead to a large number of register save and restore operations. Virtualization could play a large role due to the time involved in restoring context.

2.4.4 Phase-Specific Reconfiguration

Unlike earlier comparisons that maintained the same concatenation factor throughout an algorithm, this work broke vision algorithms into phases, and evaluated the performance of a particular CF on each phase [25]. They found that changing CF between phases could improve performance by approximately 25%.[3]

[3] Cut or expand –Carlin

Chapter 3

Reconfigurable Bit-Parallel Architecture

This section describes the basic idea of RBP, provides an argument of its advantage over a bit-serial organization, describes the Abacus PE architecture, and some RBP arithmetic implementations.

3.1 The Basic Idea of RBP

The basic idea of RBP is the concatenation of several processing elements into a processing site. The advantages of the technique can be viewed from several angles. First, if an algorithm requires many data bits per pixel, then the silicon area of the ALU is small compared with total data area, and therefore adding a few more ALUs will improve performance linearly, with a tiny increase in total area. This interpretation assumes that all data is kept on chip and physically near the ALU.

Alternately, RBP may be viewed as a method of adding the appropriate number of registers to a processing site to minimize the frequency of expensive off-chip accesses. The improvement in performance is due mostly to decreased stall time rather than additional ALU bits. [1]

3.2 RBP Performance Model

This section quantifies the regime in which the RBP organization is superior to a bit-serial one. We assume that a processing site, be it a bit-serial PE or a group of RBP PEs, requires a certain amount of on-chip storage to operate without frequent off-chip memory accesses.

[1] What really counts is that it's a variable-capacity register file –Carlin

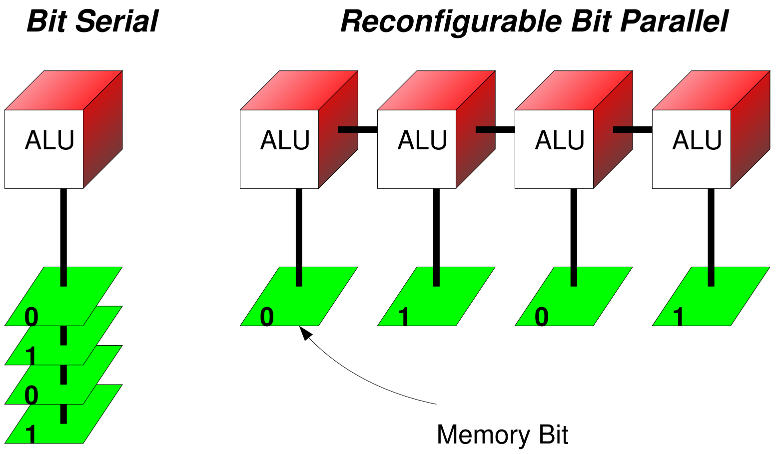

Figure 3-1: A bit-serial organization compared to a reconfigurable bit-parallel one. Each of the RBP processors is also capable of acting as a bit-serial unit, if appropriate.

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 PS 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 **??**.

The important conclusion is that not only does the RBP approach result in faster (lower-latency) computation, but that it is also more efficient in terms of computational power per silicon area. Of course, this can be seen as arguing for hardware bit-parallel.

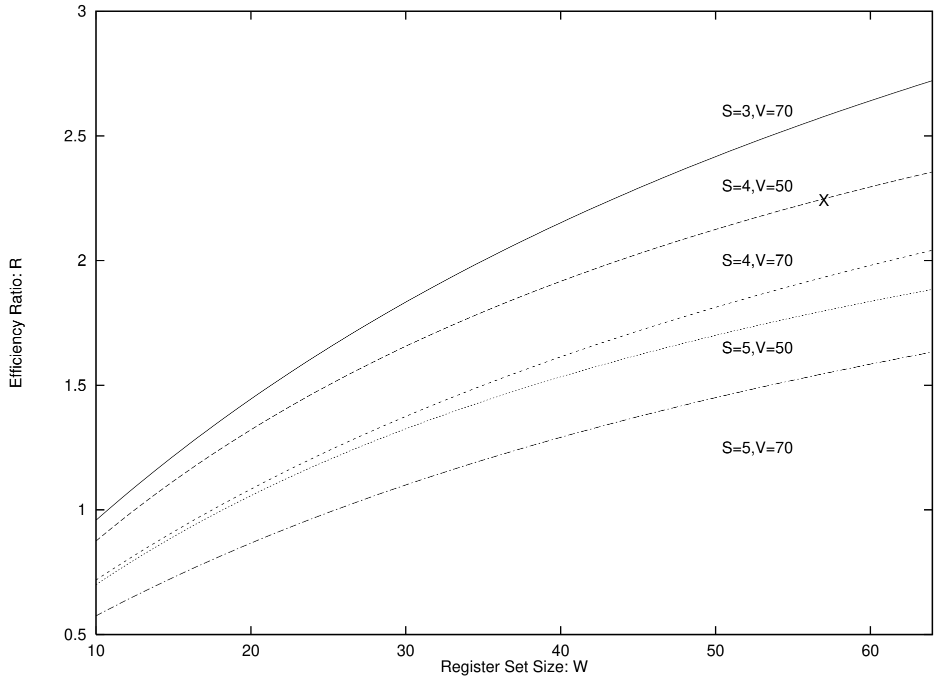

Figure 3-2: Efficiency ratio R vs register file size W, for various values of non-memory overhead V and algorithmic inefficiency S. The 'X' indicates the design point of our implementation.

3.3 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.

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.

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 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 $B \Leftarrow$

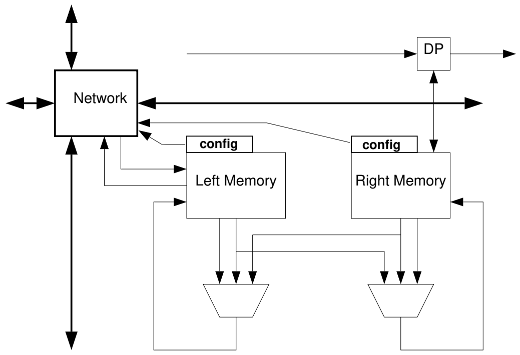

Figure 3-3: Processing Element Block Diagram.

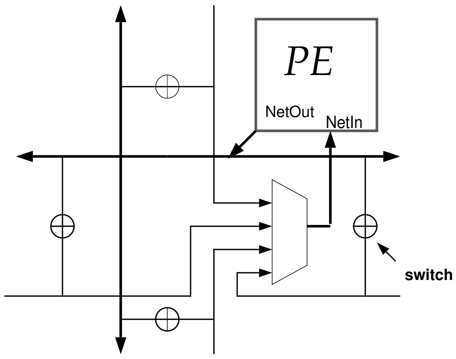

Figure 3-4: Network

`mux(`A`,`$C|B$`)`, eliminating all overhead.

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.

Background I/O Each PE contains a data plane (DP) register, 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.

3.4 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, PSs 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 labeled with its bit position within the cluster. The LSB/MSB bit can be computed from this position, as can network configuration bits.

Data Representation RBP architectures can use a wide variety of number representations. The usual representation is the conventional binary one. In the case of the accumulate operation, the redundant carry-save representation is preferred, as carry computation can be postponed until a non-redundant operation is required. The MGAP designers are proponents of the redundant radix-4 representation where each processor stores a digit in the range of $-3..3$ [26]. This is not a very memory-efficient representation, as it requires five bits to store a two bit number.

Logical Shift. The simplest RBP arithmetic operation is the shift. As shown in Figure **??**, 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.

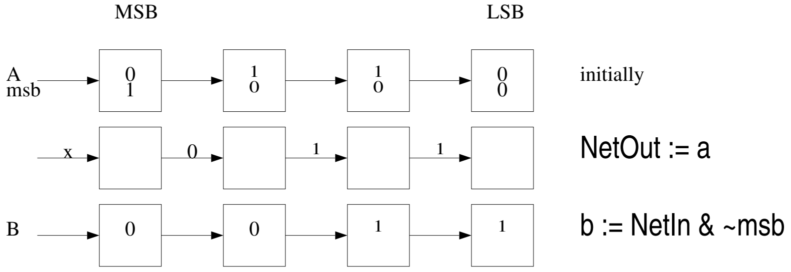

Figure 3-5: Logical shift right. The number 6 is shifted down to become 3.

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. 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.

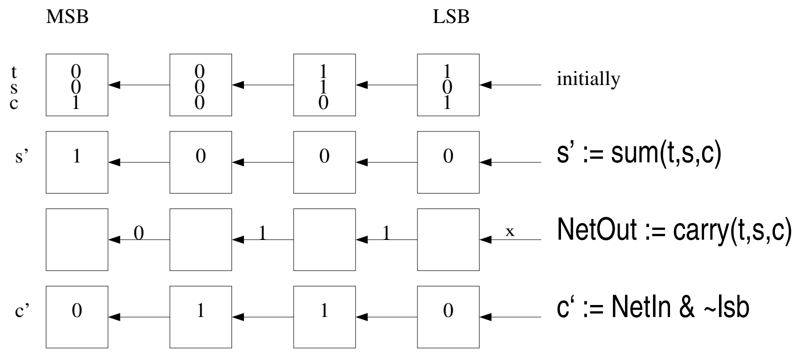

Figure 3-6: Accumulate. The addition of the number 3 to a sum of 11 represented as 2+9. After the operation, the sum is 14, represented as 6+8. All operations are purely local or nearest neighbor. Steps 1 and 2 can happen during the same cycle if a copy of t is present in both memory banks.

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 [27]. 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.

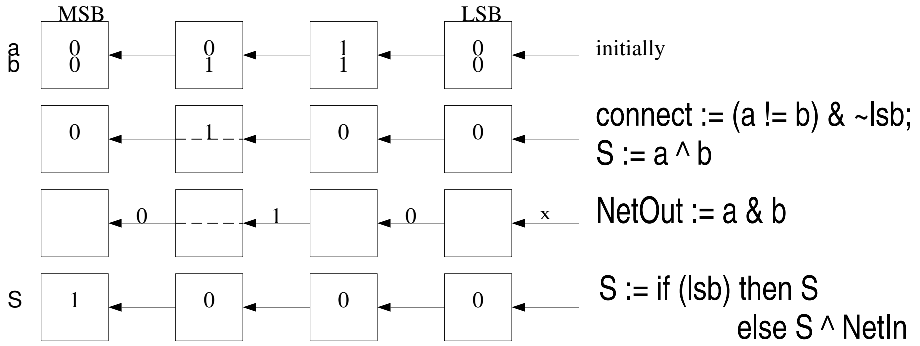

Figure 3-7: Manchester-carry addition. A NOP cycle occurs after step 2 to allow the carry bit to propagate. Steps 1 and 2 are actually merged into one cycle, and the `S` bit can be computed during the NOP cycle.

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.

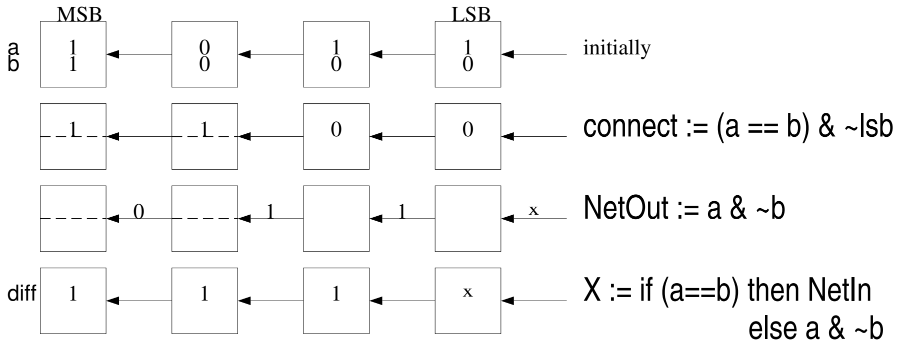

Figure 3-8: The `compare` operation. After the broadcast, the MSB PE receives A¿B for the rest of the word. It then merges the result with the MSB bit result.

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.

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, as shown in Figure ??.

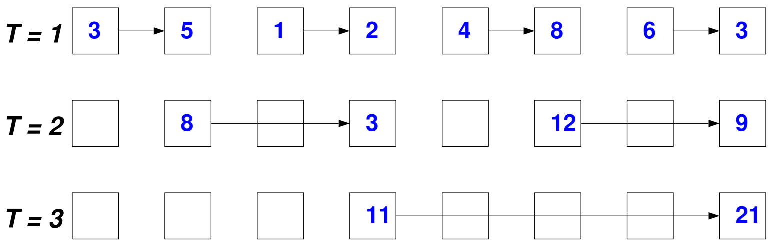

Figure 3-9: A +-reduce operation.

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 PS 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.

3.5 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.

	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 3.1:

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 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.

A high fraction of peak performance has been observed on simulations of vision algorithms.

Chapter 4

Abacus Chip

This chapter describes the design, implementation, and testing of the Abacus-1 chip in considerable detail. After reading this section, the reader should understand the issues involved in the design of a high-speed SIMD computer. The discussion focuses on the implementation, assuming that the broad architectural issues were previously decided. Of course in reality the design did not proceed unidirectionally from architecture to implementation; low level issues significantly affected the architecture.

The design goals of the Abacus-1 chip were not simply to obtain the highest possible performance, but more importantly to investigate issues of high-speed SIMD computers. Therefore, performance improvements at the cost of higher complexity and lower clock rate are not desirable.[1] Further, the high speed must transfer over to the system level, so the chip design must handle inter-chip synchronization effectively. Proceeding up the hierarchy, the lessons learned from the design must apply to practical systems. A critical requirement of real systems is the capability of expanding the memory allocated to each processing element. Finally, the system must support high speed (at least real-time) I/O of image data.

A large collection of on-chip subsystems is required to satisfy these goals. The subsystems can be divided into three broad categories: the processing element core that performs the computation; the off-chip interface that delivers network, instruction, and data streams to the core; and the control logic that binds the two together. This chapter addresses each of these categories in turn.

4.1 Processing Element Core

4.1.1 Memory

Design Processor memory size is a critical design parameter as it is the main determinant of the overall processor size and therefore the number of PEs per chip. Too little on-chip

[1] Hard to justify –Carlin

memory, and machine performance becomes limited by off-chip memory bandwidth; too much memory and the amount of processing power on a chip drops. Thus, the memory size should be carefully chosen based on extensive simulations.

In the case of the Abacus-1 chip, the choice was mostly guided by intuition. There are only a few feasible sizes, ranging between 32 and 1024 bits. The low end of the range is inefficient as much of the memory is consumed by configuration and temporary bits. The 32-bit configuration, in vertical mode, cannot perform any g algorithm that requires more than two 16-bit data items. The 1024-bit configuration, in horizontal mode, allocates 1 Kword per pixel site, which is far more memory than needed. Two other disadvantages of larger memories are a longer access time, and more address bits, which translates directly into higher instruction bandwidth, already one of the more challenging board-level design tasks. The Abacus-1 design has 64 bits of memory per PE.

Implementation The 64-bit register file is arranged as two banks of eight rows and four columns. Each cell is a standard six-transistor SRAM cell. The bit and $\overline{\text{bit}}$ lines are used single-ended to provide two read ports, allowing four bits to be read and two written in each cycle.

The bitlines are precharged to $V_{dd} - V_T$ to speed pulldown time and reduce power dissipation. Precharging is done through the column select transistors, further reducing power by charging only those bitlines that are about to be read.

The SRAM cell is susceptible to false writes, especially because the bitlines are typically not charged to opposite values. Thus, the high-capacitance bitlines, charged to opposing values during the read phase, can act as a line driver, overwriting cells not selected by the column selects. This problem is avoided by shorting the bit and $\overline{\text{bit}}$ lines during the precharge interval, even if those lines are charged. In other words, all bitline pairs are shorted, but only the ones to be read are charged.

One of the key space-saving ideas was the integration of the five control registers and the network interface into the SRAM array. The initial design included fully static master-slave flip-flops, multiplexers to connect them to read busses, and separate control wires. This auxiliary circuitry substantially increased the non-ALU PE area. In the current implementation, edge cells in the SRAM array, together with buffers that isolate the internal data storage node, are used as control registers. In the case of the network and data plane registers, the multiplexing occurs inside the SRAM cell.

4.1.2 ALU

Design The main design issue of the ALU is its range of computations. The initial ALU design was a 4:1 multiplexer that could compute all 16 two-input one-output functions. Computation of a full binary one-bit addition therefore required five cycles, emulating all five 2-input gates.

The limited complexity is due to available data sources and sinks. A single-ported memory bank can produce only two bits for reading, and can absorb only a one bit result. If operations such as addition are required, the extra input bit must come from another source such as an auxiliary register or the communication network.

Previous bit-serial computers incorporated dedicated adders and carry registers. Abacus does not follow this approach because of the additional area and because the circuitry is only useful in vertical mode. A dedicated full adder circuit would be useful in horizontal mode, assuming its outputs could be steered appropriately, but it requires three input bits while the memory only produces two per cycle. The Abacus design's solution was based on the realization that the memory required much more area than the ALU. Splitting the memory into two banks doubles the memory bandwidth and adding another ALU doubles the peak computing power. If the ALU requires one tenth of the memory area, the twin-bank, twin-ALU (TBTA) design occupies 1.1 times the area, but has 2.0 times the performance, for an overall advantage of 1.8 over the single ALU.

	SBSA		TBTA	
Circuit Element	1 ALU, 1 bank		2 ALU, 2 bank	
	Quant	**Area**	**Quant**	**Area2**
Memory Array	1	205	2	102
Column Mux	8	20	8	20
Read/Write Logic	1	18	2	36
Aux Logic	1	64	1	64
ALU	1	38	2	76
Network	1	52	1	52
Total		397		452

Table 4.1: Silicon area requirements of the single-banked, single-ALU and the twin-banked, twin-ALU designs. As expected, silicon area increases by only 15%.

A significant penalty occurs because the chip is pad-limited. The inner pad-ring is arranged in a 56 by 44 rectangle. Without the 20 extra instruction bits (corresponding to 10 double-cycled pads) the pad ring could be reduced to a 51 by 44 rectangle. Assuming a 170 micron pad pitch, the area decrease is 6.4 mm^2. Since the entire PE array occupies 38.4 mm^2, the extra instruction bits cost an additional 17% overhead. As long as the design remains pad limited, this effect will grow worse as technologies scale down, mostly because the pad size stays constant while the PE density improves. Thus, any design decision that increases the number of chip pads should be considered carefully. This is an important example of how architectural analysis must be done at all levels of design in order to be accurate.

The algorithmic advantage is more difficult to quantify. Each 3-input ALU can perform an arbitrary 3-input operation each cycle, which could take a 2-input ALU at least 3 (and as many as 5) cycles. To see this, consider the an arbitrary function of three boolean variables Z = f(A,B,C). Using the Shannon expansion, Z can be expressed as

$$Z = (\overline{A} \cdot f_{A=0}(B,C)) + A \cdot *f_{A=1}(B,C)) \tag{4.1}$$

where the two subfunctions are obtained from the truth table entries corresponding to the cases where A is 0 and 1 respectively. Most useful functions are actually simpler. For example, consider the the multiplexing operation `Z = (!A & B) | (A & C)`.[2] For this function, $f_{A=0}(B, C)$ is simply B, and steps 1 and 3 of Figure 4-1 can be eliminated.

```
T1  =  f0(B,C)
T1  =  T1 & !A
T2  =  f1(B,C)
T2  =  T2 & A
Z   =  T1 + T2
```

Figure 4-1: Evaluation of an arbitrary 3-input function by a two-input ALU. Two auxiliary memory locations are required.

Converting to two 3-input ALUs improves the peak bit-manipulation performance by a factor of ten, but several effects limit this improvement. First, the addressing is limited to sharing certain bits between the two banks. Second, these five-cycle operations are relatively rare. Third, data dependencies require waiting for bits to arrive from the network.

Table 4.2 shows the number of cycles required per arithmetic operation, including reconfiguration overhead. The ratio is about 1.75. Of course, the proper instruction mix for real applications should be evaluated for true improvements, but this analysis suffices to bound performance improvement in the range of 1.25 to 2.1.

Operation	SBSA (cycles)	TBTA (cycles)
Add	9	4
Shift	4	3
Grid Move	7	6
Accumulate	10	4
Average	7.5	4.3

Table 4.2: Number of cycles required for common operations in horizontal mode. The count includes reconfiguration cost.

There is another subtle effect of a more complicated ALU. A 2-input ALU can require up to two temporary storage locations to evaluate most 3-input functions. When operating near the memory capacity, two memory spills can have a substantial effect on performance.

We can now evaluate the overall advantage of the twin ALU design. The area cost is dominated by the pad ring increase, and is a factor of 1.35. The performance improvement

[2] This function is used frequently when some bits must be conditionally updated. The idiom is `Z := mux(flag, Z, newZ)`.

is 1.75. Thus, the overall advantage of the design is only about 30%. This advantage will increase slightly as compiler technology improves and as memory accesses become more expensive.

Implementation. Each ALU is implemented as a precharged 8-to-1 multiplexer that uses three data bits to select one of eight instruction bits. Initial design used predecoded cells but this was slower and larger due to the non-regularity. This design does have four transistors in the pulldown path, but the timing and capacitances are designed to minimize the body effect and reduce charge-sharing problems.

Immediate Constant. The ALU also incorporates an additional circuit that allows the architecture to support effective constant distribution. Without this feature, there is no clean method of distributing a run-time value generated in the scalar unit to the array. Indeed the only way of transferring such a value would be to send it via the image IO interface.

Instead, the output of the left ALU passes through a NAND gate controlled by a global wire common to each column of PEs. In normal operation, these 32 global "literal" wires are held at 1. When a constant is to be loaded, the instruction decoder generates a `set` ALU op code, and loads the literal bus with the logical inverse of the constant. Thus, in every column whose bus is at 1, the result will be forced to a 0, as desired.

4.1.3 Network

Design The two questions in network design are where the PE obtain its network data and how does it route data around itself. Two alternatives of routing data are shown in Figure 4-2. The first, the two-channel crossbar, allows messages to pass in a wide variety of directions. For example, the PE can connect the north port to the east port while simultaneously connecting the west port to the south. The second is far less flexible, and can only make one connection.

The original network design used four independent switches in a mesh topology. Each switch had a dedicated control register. The independent switch control allowed many cluster configurations.

The price of this flexibility is additional registers and increased reconfiguration time. If a register is allocated to each switch, the four registers represent 5% of the total addressable registers. More importantly, reconfiguration time doubles. Even with dual ALUs (which were added primarily to speed reconfiguration), two cycles are required to change topology. Since reconfiguration may be required after each arithmetic operation, and since most operations require only two or three cycles, the additional 1 cycle penalty is a performance hit of approximately 20%.

Reconfiguration time can be reduced at a slight loss in flexibility. Since information flow in arithmetic operations is usually along the MSB/LSB path, configuration in a line is usually

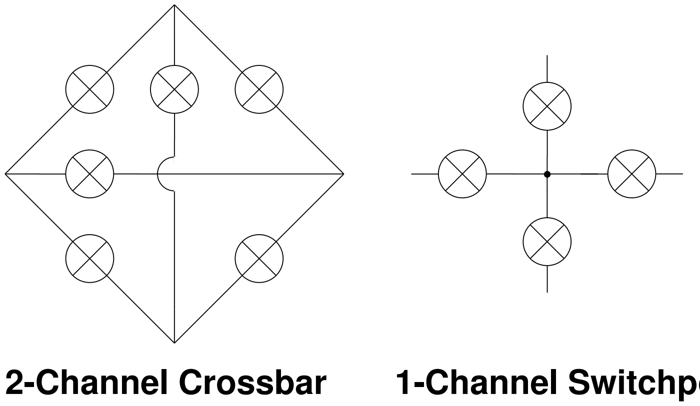

Figure 4-2: Various network types. The Abacus type is of the single channel, but even more restrictive in that each PE can turn on only one switch.

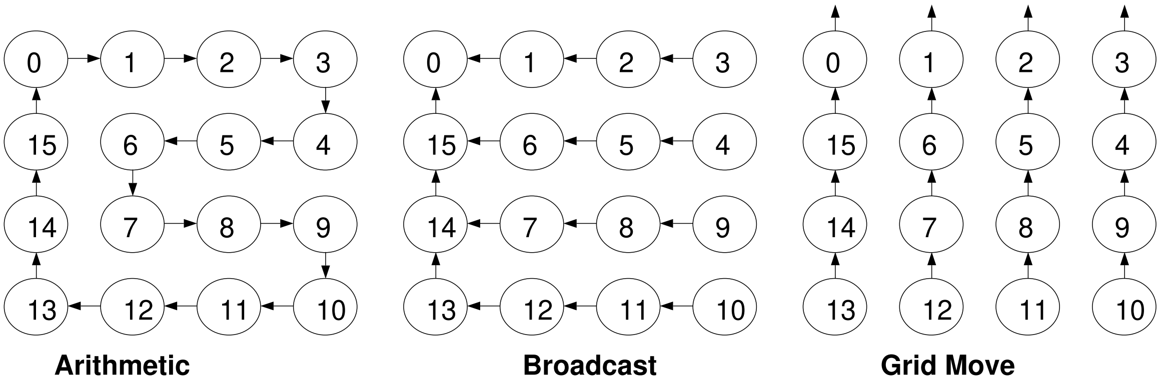

Figure 4-3: Various network configurations. Arithmetic, bit broadcast, and mesh move modes are shown. For example, broadcast of a bit to all 16 cluster elements can be done with only 8 ($2\sqrt{N}$) switch delays if the cluster is configured as a tree with fanout of 4.

sufficient. The design implemented in Abacus-1 incorporates a decoder to select only one of the four switches, and a *connect* configuration bit that controls whether any of the switches are activated at all. The addressing cost of the two approaches is not substantially different: three registers instead of four. But now, changing reconfiguration directions requires only one cycle, and conditional bypass, another frequent operation, also requires only one cycle instead of two.

Implementation. The on-chip communication network consists of a precharged output node and four isolating switches. The node is a simple precharged inverter whose input is

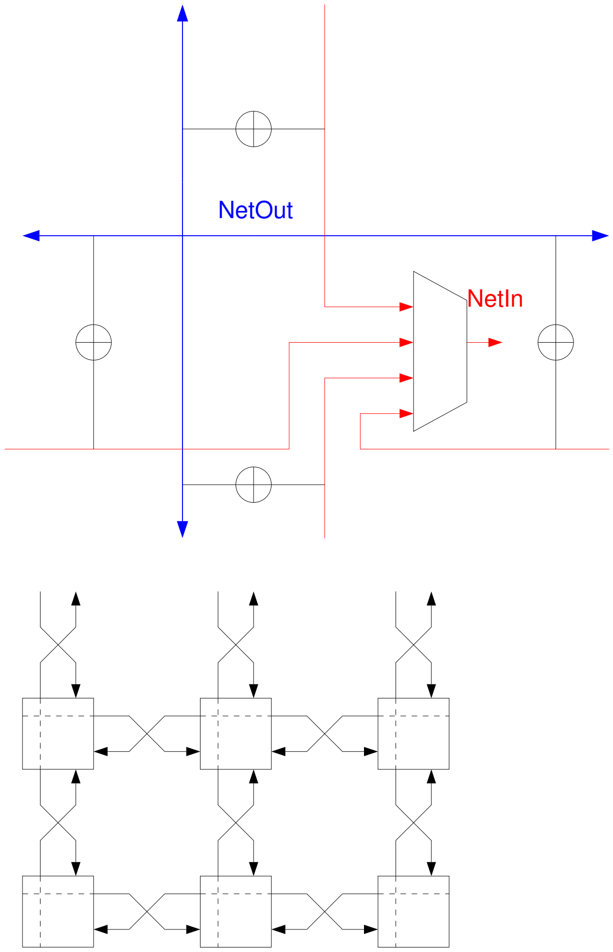

Figure 4-4: Network Block Diagram. Magnified view and mesh view.

driven by the network output register. The four switches are NMOS transistors driven by a 2:4 precharged decoder.

Communication with signaling times longer than a cycle period is performed by controlling network precharge via software. As long as precharge is turned off, processors connected to a bus node continue to discharge it.

Propagation delay across a series of pass transistors is usually quadratic in the number of devices. This delay is greatly reduced 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 switches and long network wires in one 8 ns cycle.

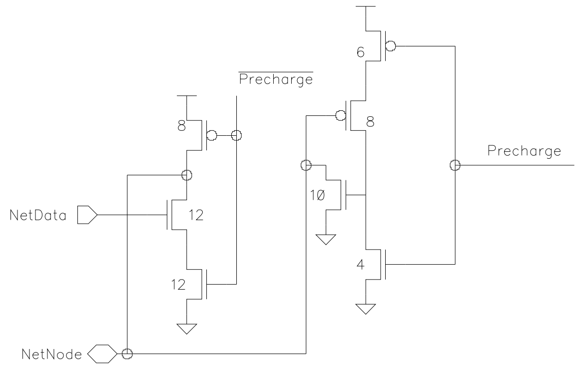

Figure 4-5: Network accelerator in the NORA circuit style.

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 bypass capacitance and over 200 power and ground pins to reduce dI/dt noise.

4.1.4 Data/IO Planes

The architecture supports a background loading mechanism for both the external memory data and image data. Data is shifted through the PE array without interfering with computation. One memory location in each PE is dedicated as the data plane (DP) registers. All DP registers in a column are connected in a 32-element shift register. This shift register is clocked by a global signal. The PE addresses the DP register as a normal SRAM cell, and is not aware of the alternate write path.

4.1.5 Global OR Operation

The global OR (GOR) mode is used to signal completion of some computation by the entire PE array to the sequencer. Off-chip, the GOR is computed by dedicated logic. On-chip, the GOR can be computed with the multiple-writer capability of the network. In this mode, all PEs connect their network nodes into one chip-wide node. The shortest propagation path is obtained by connecting all PEs in a row to the left-most PE of that row, and then connecting all left-edge PEs together vertically, as in Figure 4-3. The on-chip evaluation time is five cycles in the worst case.

4.1.6 Layout Summary

The simplicity of each PE resulted in a compact, 83 micron by 453 micron implementation in a 2.6 micron contacted metal pitch technology.

Subsystem	Area (Kμ^2)	Percentage
Memory	22.9	61.0
ALU	6.3	16.8
Network	6.2	16.6
Misc PE	1.2	3.1
Data Plane	1.0	2.6

Table 4.3: PE Area resources

This data differs from the earlier table in that configuration registers are now allocated to the resource they serve instead of to the memory array. For example, there are four control registers (2 select, 1 connect, 1 data) dedicated to the network. Similarly, two are dedicated to the I/O and data planes.

Figure 4-6: PE Layout with the second layer of metal removed. **ADD OVERLAYS**

4.2 On-Chip Signal Distribution

A previously unanticipated design task was the challenge of distributing control signals to the PEs. As discussed in Section 4.2.3, there are four phases in the clock cycle, and each phase must complete before the other can begin. For example, all word lines in the register file must be unasserted while bitlines are being precharged, and any overlap may cause memory cells to be overwritten. This nonoverlap is guaranteed by separating the clock phases sufficiently to account for the worst case rise/fall times. Signal transitions that occur in the same phase but must follow each other are separated by inserting an inverter or two as delay elements.

There are two problems with this approach. Given 8 ns cycle time of the PE, each of the four phases has only 2 ns to rise, stay level, and fall. This means that even if the level time is 1 ns long, each rise/fall time must be 500 ps or less. This is an aggressive RC product, given the long and narrow signal wires and the high gate loading of the wire. The usual technique of simply widening the signal wires cannot be used due to the wire density.

The second problem is that even if the four phases can be safely separated, non-overlap within the cycle by using inverters as delay elements may not work because the variation in wire loading may dwarf inverter delays. Both of these problems can be addressed by local buffering of the global signals.

4.2.1 Load Equalization

Local buffering both reduces the load on each wire and decreases the variation between signal arrival times due to unequal loads. This section describes the calculations used to obtain the load values, wire widths, and row buffer sizes.

In addition to reducing the capacitive load by local buffering, the RC product was improved by reducing wire resistance by widening the signal traces. Although the wire capacitance increases, the lowered resistance decreases the effect of the load capacitance, as shown in Equation **??**.

$$\tau = RC = (R_{\mathrm{wire}} + R_{\mathrm{driver}})\,(C_{\mathrm{wire}} + C_{\mathrm{driver}}) \tag{4.2}$$

The equation can be made more rigorous to account for the distributed wire loads and the lumped driver load, but those calculations do not capture the most difficult characterization: the variation in wire capacitance due to fringing fields and to the material underlying the wires. Further, the wire pitch, and therefore width and spacing, undergoes different constraints in different parts of the circuit. Instead of obtaining an analytical solution for the wire width, spreadsheet models were developed for common wiring situations and driver configurations, and used to determine case-specific trace widths.

4.2.2 Layout Issues: The Trench

The Abacus-1 design had to deliver 76 signals across a 1 cm chip with very little skew even though the array is tolerant of gradual signal skew across the part since communication is purely local. Problems arise in two cases: intra-PE signal skew and interfaces to logically synchronous zones at at the pad ring. The task was decomposed into two parts: delivering the signals to the horizontal row drivers and then their outputs. Vertical signal delivery was not an issue due to the low resistivity of the metal 3 layer. The design easily tolerated high uniform latency in instruction distribution so the row buffers could be composed of several scale-up stages. As a result, the loads presented to the vertical wires were that of minimum sized inverters, and therefore dwarfed by wiring capacitance. The vertical wires were metal-3, 3.2 microns wide, and spaced by 3.3 microns. The nominal capacitance of a 7 mm vertical wire is 2.5 pF, and the nominal resistance is 75 ohms. The distributed RC delay of this wire is only 184 picoseconds.

The problem arises after the signals have been delivered on the vertical wires. The row buffers are rather large, and there are 152 of them packed into one PE height, 76 for each side. This occupies a significant area, and spans 1.1 mm! Fully one-sixth of the core area is dedicated to these buffers. To connect a row buffer at the extreme edge of the trench to the vertical wire requires a horizontal wire of 500 microns. This horizontal load almost doubles the signal capacitance, and therefore skew. Also, a little bit of horizontal skew is introduced, as the row buffer output wires end up differing in length, depending on the buffer position. [3] [4]

4.2.3 Timing

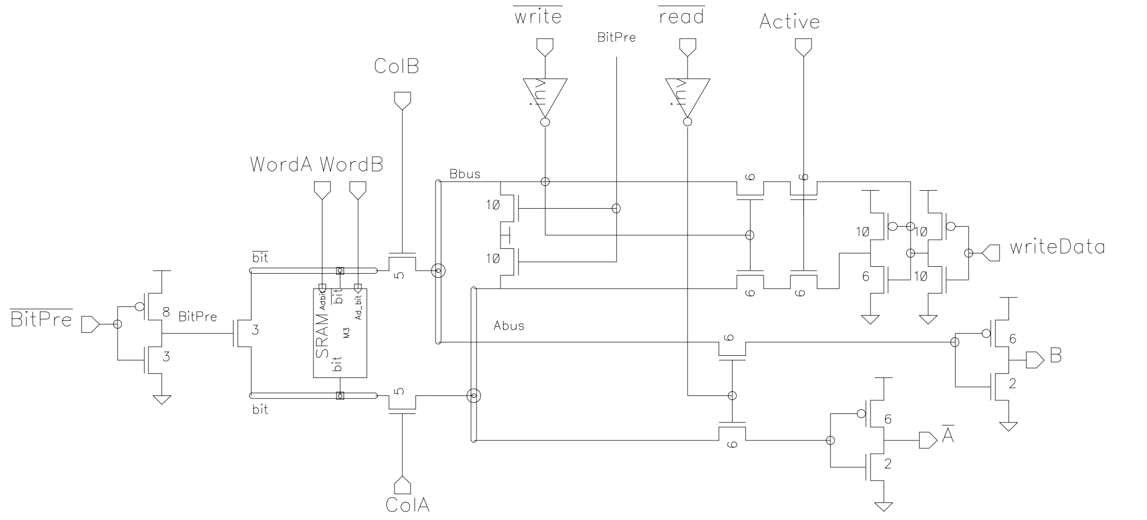

Figure 4-7: Memory read and write path (one of the banks). No reference in text –Carlin.

The processor timing is driven by the basic design choice of double-cycling the register file to perform a read and a write operation in one cycle. The cycle starts with the instruction clock, which clocks out the decoded instruction bits. At the same time, the word lines are gated off to avoid overwriting memory cell contents in the upcoming precharge interval. After the word lines are safely off, , the bitline precharge signal is asserted. This signal enables precharge transistors onto the A and B busses, charging them, and the bitlines that are about to be read. The column selects must therefore be stable at their read phase values during this phase. A circuit diagram of the memory system is shown in Figure 4-7.

After the bitline is unasserted, the read phase executes. The word lines are enabled, allowing the selected SRAM cells to pull down their respective bitlines, and through the column selects, to pull down the A and B lines. When the read cycle is over, the contents of the A and B busses are latched on a simple NMOS dynamic latch. The ALU evaluate cycle begins next (the ALU evaluate is the read signal delayed by one inverter). All inputs to

[3] Notice that the routing channel alone is 500 microns. This means that row buffers had better drive only one row of signals. Otherwise, distribution of signals to other rows will require another such routing channel on each half of the array, boosting the trench width to 1500 microns!

[4] Need A Diagram!

the ALU were guaranteed high for long enough during its precharge phase (since the A and B busses were precharged). The ALU outputs are stable until the next read cycle. During the ALU evaluate the bitlines are equilibrated and precharged. As in the read precharge phase, the word lines must be gated off. During this off interval, they may safely transition to the write addreses.

When the ALU cycle completes, the write cycle begins. The write is a double-sided write and occurs through the write enables, the column selects, and the SRAM pass transistor, so it is fairly slow. In fact, once the coupled inverters are flipped to their future state, the write completes slightly into the next cycle. This delay is safe because the written state is not examined until after the precharge phase of the next cycle.

The write signal also controls the network precharge/evaluate timing. As soon as the write is over, the network evaluates. The dynamic decoder evaluates and selects one of the four switches. Simultaneously, the X node is pulled down (or not, based on the NetData register). Thus, for neighbor to neighbor communication, the just written bit has the combined time of the bit precharge period and the read period to propagate to the neighbor and pull down the bitline.

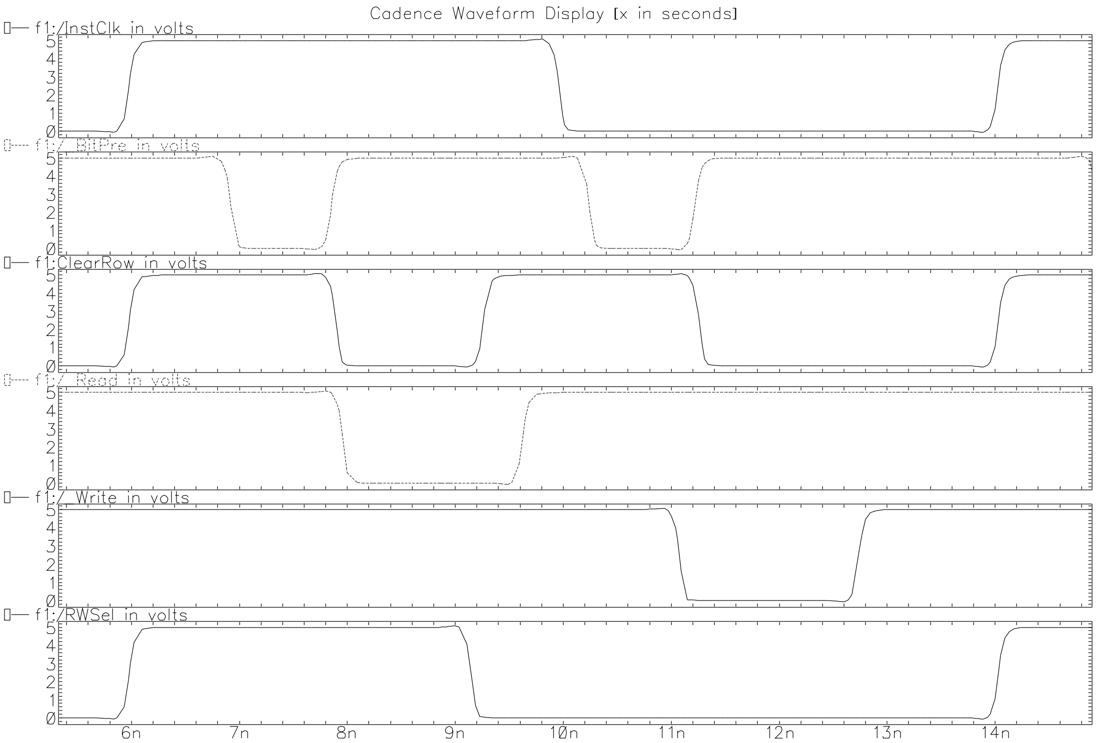

Figure 4-8: PE Timing.

4.3 External Interfaces

4.3.1 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 a mostly–digital delay line with a fine-tuning analog section. The DLL uses the simple and accurate arbiter circuit shown in Figure 4-9. The arbiter has a symmetric 30 ps uncertainty window at nominal process.

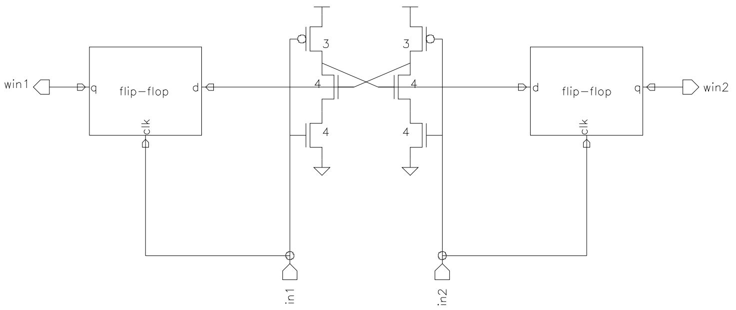

Figure 4-9: Arbiter used in the delay-locked loop

The next design issue is the choice of which of the several signals should be synchronized across chips. There are several internal clocks: instruction register, PE register read, network precharge, etc. Abacus synchronizes to the clock that drives the mesh output pads. As a result, off- chip communication is synchronized system-wide.

4.3.2 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. Several effects complicate this requirement.

First, signal propagation time across a large circuit board is significant at high frequencies. Propagation time is approximately 2 nanoseconds per foot. Thus for a 60 cm board, chips near the distribution point receive their signals four nanoseconds before chips on the far edge. Even wires connecting to the same chip can deliver signals at slightly different times due to routing differences. For example, wires routed on outside PCB layers experience a different effective ϵ_r and therefore different propagation velocities. Finally, PCB thickness variations can cause impedance variations that affect signal velocity. Two common techniques to overcome this problem are snaking traces to equalize propagation delay, or by using specialized clock distribution parts with programmable delays such as the Clock-Works family [28]. Neither technique is feasible for a board with four 30-bit instruction busses running at 250 MHz.

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 component variability.

Thus, the instruction distribution scheme must solve three problems. First, it must sample safely within the 4 nanosecond data window. Second, it must align all bits of an instruction to arrive at the same time. Third, it must align instructions across all chips in the system. The Abacus-1 instruction pads incorporate retiming logic, shown in Figure 4-10 that solves each of these problems in turn. Retiming occurs as part of the system startup process and proceeds in three stages.

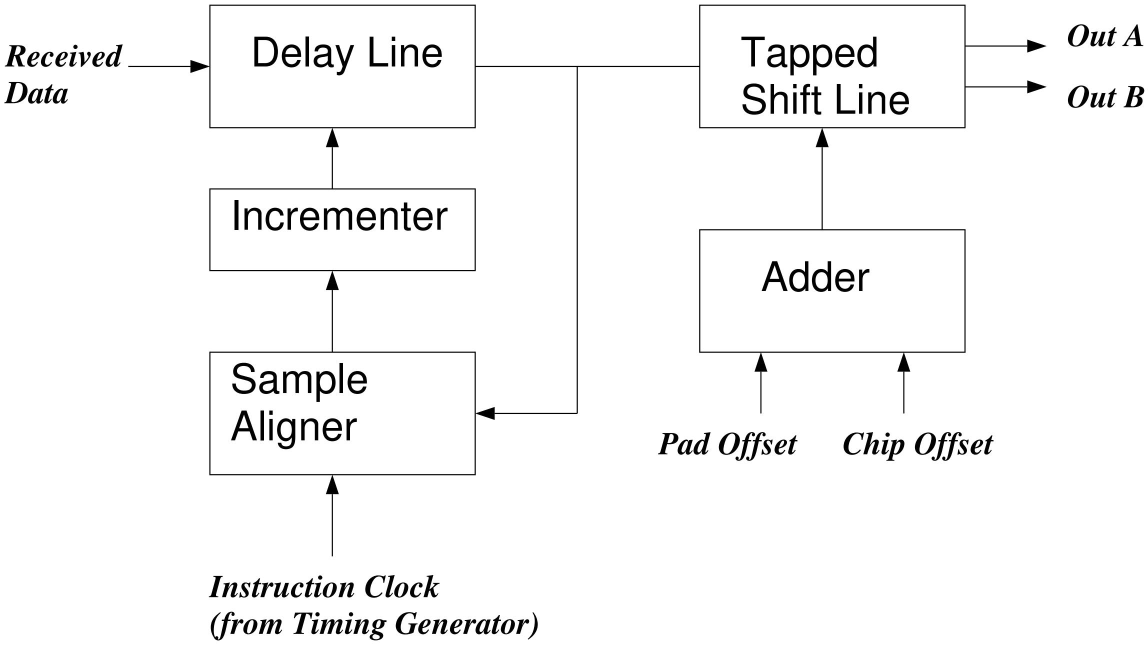

Figure 4-10: Instruction Pad Block Diagram.

The sampling alignment stage first finely adjusts the delay of each bit until the sampling clock transitions safely after the data is stable. The variable delay is performed by an all-digital delay line consisting of inverter chains and multiplexers. A binary value generated by a free-running counter selects the delay. The counter runs until the sampling clock transitions after a delayed version D' of that data. That event disables the counter, freezing the delay. Ideally D' is one quarter of the cycle time, so that the sampling occurs in the middle of the data valid period, but the quarter-delay line is limited to approximately 30 inverter delays. Notice that the instruction generator must ensure that the data transitions on every cycle during this phase.

Once this phase is complete, the bit alignment stage lines up all instruction bits. This technique proceeds by passing the received data bits through a shift register. The shift register outputs are ORred together to generate a `got_one` bit. As soon as all instruction pads assert this bit, the shift register is disabled. The position of the leading one selects the delay to be applied to the data of that particular pad by selecting the tap location from another tapped delay line.

Finally, the instruction sequencer performs the system-wide alignment phase by broadcasting instructions and using the global OR feedback to determine the delay to each chip. Once this delay is known, the sequencer uses the boundary-scan interface to set a register in each chip that determines another delay to be applied to all pads in the chip. To eliminate the need for yet another delay line, the control register value is added (in unary) to the delay computed in the previous phase, and the sum is actually used to select the tap location.

The development history of this design is somewhat interesting. When it became apparent that instruction distribution time was comparable than cycle time, a proposed design used the PE chips as pipeline registers, receiving and sending instructions point to point. To synchronize instruction execution, successive chips in a column would have to execute out of different stages in an internal instruction buffer. A natural next step is unpipeline the process, and have the chips load the internal delay buffer direction from the bus. Once the idea of cycle-level delay is accepted, it leads to using nanosecond-level delays at the pin level.

4.3.3 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 array on each chip are time multiplexed onto 16-bit ports. The required clock rate is therefore 250 MHz, which presents a number of design problems.

Chips are separated by significantly varying transmission times. For example, chips on board edges communicate over wires as long as 2 feet, while neighboring on-board chips are separated by less than two inches. Also, different signal paths have different propagation speeds. For example, the ribbon cables used between board have a characteristic impedance of 90 ohms, compared to 50 ohms for PCB traces, and are therefore almost twice as slow.

Process skews between chips introduce more differences. Although the chips are synchronized at the start of a cycle, subsequent edges diverge in time. Therefore, the faster chips can sample the second transmitted bit too early, getting a copy of the first bit.

Unlike the instruction pads, the mesh port design assumes that all wires in a port are bundled and experience similar delays. 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 [29]. 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. This was added to investigate the practicality of algorithms designed under the architectural model of meshes with reconfigurable busses.

Crossing a chip boundary costs an extra cycle for move operations. That occurs because on-chip nearest neighbor is on the edge of making it to the adjacent processor, never mind across a board. (Note that with aggressive cycle times, worst-case travel in a real system is at least a cycle time).

The edge interface does not present a uniform mesh model to the programmer. [5]

4.3.4 External Memory Interface

Any SIMD architecture that does not support relatively efficient access to off-chip memory is simply unable to process to data sets that do not fit in the small on-chip memory. This type of hard limit does not fit the common expectation that a computer should be able to process larger than optimal data sets, albeit at a performance degradation. Any such architecture is limited to algorithms with little memory requirements such as systolic processing.

Each Abacus 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 array. The 16 ns cycle time is made possible by the Ramtron extended DRAM part, which contains an SRAM cache integrated in the DRAM. When an address not in the cache is referenced, a relatively slow (35 ns) DRAM read cycle is initiated, and in each memory chip, 512 bits are loaded into the SRAM cache. The long DRAM read latency is encountered only once, after which all reads operate at SRAM speeds. This mode of operation is ideal for the block transfer requirements of Abacus, in which 1024 PEs request a bit stored in the same address.

[5] And...? –Carlin

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 a dedicated DRAM controller. A useful side-effect of this approach is the elimination of memory glue logic, and the ability to use different memory components by changing only the control software. 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.

4.3.5 Image I/O Interface

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, if anywhere, is clearly going to occur upstream of the PE array. This very high I/O bandwidth makes the architecture well-suited to applications requiring real-time processing of large amounts of data such as video and synthetic aperture radar.

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 array to the output pin (or from input pin to the PE array). 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.

4.4 Chip Floorplan and Control Logic

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, as shown in Figure 4-11. 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. 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.

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.

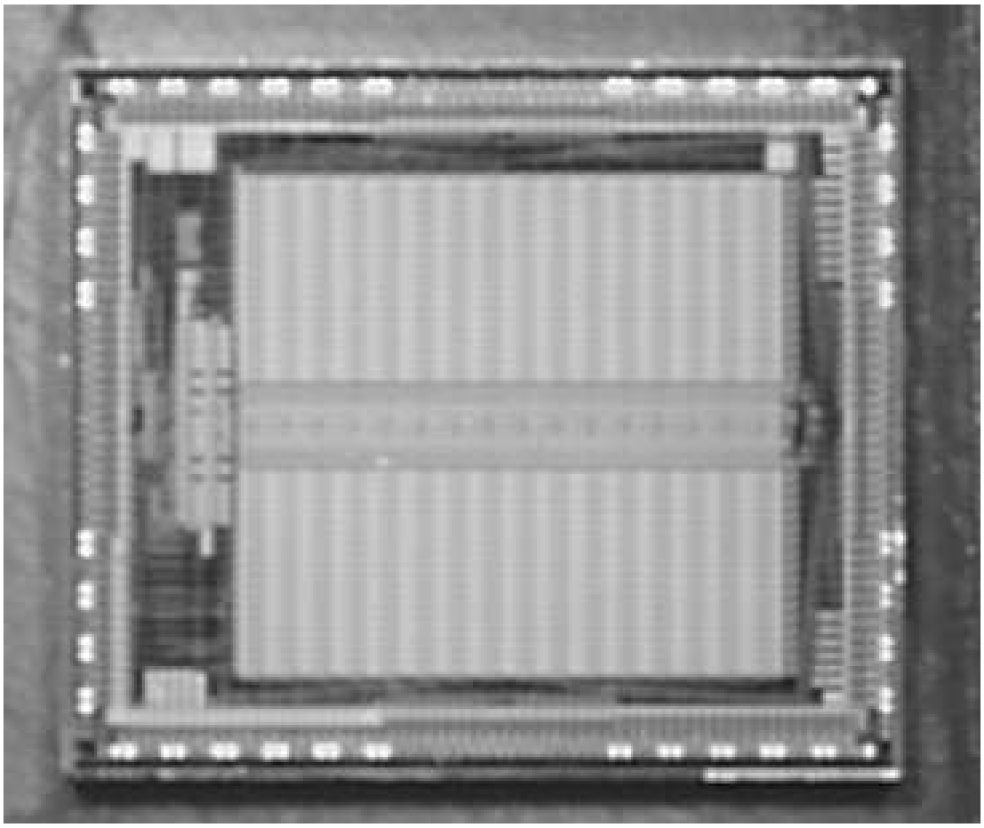

Figure 4-11: Chip photograph.

System	Area mm^2	Fraction %
Pad Ring	24.23	25.4
Control Logic	9.43	9.9
PEs	38.30	40.1
Row buffers	7.95	8.3
Busses	13.09	13.7
Edge Interfaces	2.57	2.7
Total	**95.57**	

Table 4.4: Chip Area by Subsystem

The PE array formed approximately 40% of the chip area and 56% of the chip core area.

4.4.1 Timing Generator

The timing generator is a straightforward design, based on tapping a delay line composed of inverters. Clocks are generated by ANDing a tapped signal and a delayed version. This method provides an accurate method of generating pulses quantized in terms of inverter delays, and therefore tracking process variation. Another useful property of this timing method is that all clocks are generated from the leading edge of the delay line input, so the chip can operate at a variable clock rate.

The timing generator includes a debugging feature that doubles the delay of each inverter stage by opening a pass gate to a capacitor. This technique stretches out the clock waveforms uniformly, affecting both pulse widths and pulse start times. The feature was intended to allow testing of the chip in case the original pulse widths were too narrow. The original design was based on an analog-controlled delay line [30], in which the voltage controlled the gate voltage on the switch. This flexibility was unnecessary and routing an analog line to each chip in a system was judged to be impractical.

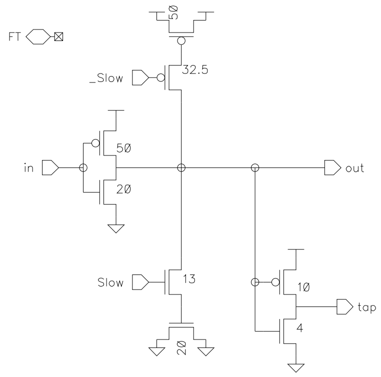

Figure 4-12: Time Unit.

4.4.2 Pad Ring Summary

4.4.3 Instruction Decode

Instruction decode is very simple, primarily because the instruction word is effectively horizontal microcode. The decoder logic is equivalent to approximately forty two-input gates. A sizeable fraction of those gates detect when a special instruction is specified. Once that occurs, further decoding of special instructions requires at most two more gate delays. Ordinary instructions are not decoded at all; addresses are sent directly to the address decoders

Pins	Number	Direction	Voltage	Freq
Grid clock in	4	I	1V	250
Grid clock out	4	O	1V	250
Grid data	64	IO	1V	250
ECL Vref	1	I	ECL	DC
Clock	2	I	ECL	125
Data In	1	I	ECL	125
Data Out	2	O	ECL	125
Global OR	2	O	ECL	125
Instruction	30	I	ECL	250
DRAM address	11	O	CMOS	62.5
DRAM control	10	O	CMOS	62.5
DRAM data	64	IO	CMOS	62.5
TAP in	4	I	CMOS	125
TAP out	1	O	CMOS	125
Debug	4	O	CMOS	125
Total signal	204			
Power (5V)	74			
Power (1V)	32			
Ground	114			
Total	424			

Table 4.5: Pin Requirements

and ALU op codes directly to the pipeline registers. Including the address decoders, the logic depth of the decode stage is approximately eight gates.

4.4.4 TAP Control

The Abacus chip incorporates a standard IEEE Test Access Port (TAP) interface. This interface is used not only for testing the chip, but is a necessary part of system initialization and configuration. It is used to initiate clock synchronization and instruction pad retiming, to program impedance settings on the mesh communication ports, and to inform each chip about its location on the circuit board.

The test interface provides access to the decoded instruction pipeline registers. These registers can be scanned out to evaluate instruction pad functionality or scanned in to override the instruction decoder and allow testing of the PE array in case of a design fault.

The current design has a fault in that the initial value of the pipeline registers upon powering up is unknown. As a result, all decoded address lines could be activated and cycling at the full clock rate. Thus, until a correct bit pattern is scanned in, power consumption could be higher than in normal operation.

4.5 Design Style

The high clock speed of the Abacus-1 chip is due to careful circuit design and physical layout. This section identifies the factors that contributed to fast execution.

Layout Style. The processing element was implemented entirely in full custom using the Cadence tool suite. A number of small standard cells used in the control logic were synthesized from schematics. The control section, including instruction decoding and TAP control, was assembled from these cells and interconnected with a greedy router locally developed by Larry Dennison. The router introduced some layout violations, and those errors were manually corrected.

Layout synthesis was also used in creating the scan registers surrounding the instruction decode, but this turned out to be a mistake. A rough estimate indicates that a manual layout would have required approximately the same amount of design effort, and would have achieved almost 30% area reduction.

A substantial contributor to layout density was the use of vendor-specific layout rules instead of the common scalable design rules provided to universities by MOSIS. The effort invested in customizing the Cadence tools to the Hewlett-Packard technology was quickly repaid, as the area penalty due to scalable rules is estimated at 40%, especially for full custom layout.

Circuit Design. A number of design principles independent of layout optimizations contributed to the high clock speed.

1. Dynamic logic was used extensively in the ALUs, network configuration decoder, and the network pathway itself.
2. As a corollary, the design was optimized to use mostly NFETs for pass gates and precharge transistors. Not only are N devices smaller, leading to denser layout, but they also load the control signal more lightly.
3. Global signals were carefully distributed by the use of matched wiring, driver buffers, and dummy logic elements. As a result, timing was more predictable, enabling smaller timing margins.
4. Some circuit elements were used for multiple purposes. For example, precharging bitlines through the column select transistors required only one precharge transistor per bus, instead of one per bitline.
5. Noise margins were sacrificed where necessary. For example, the bitlines were precharged to a V_T drop below V_{dd}. Also, inverters were ratioed based on preferred transistion directions. Thus, if a particular node was precharged high and evaluated low, the corresponding inverter was ratioed to switch at a higher voltage.
6. Dynamic latches were used to capture data and allow the previous stage to be reset as soon as the data was latched. This increased the effecive recovery time.

7. Signal buffering was optimized both at the schematic and layout levels. A tool was written to probe a wire in a schematic and calculate its total capacitive load. As the design changed, the program constantly recalculated optimal buffer sizes. A layout macro was optimized to generate a fingers-style layout [31], greatly reducing parasitic diffusion capacitance.

Functional Optimizations. Some performance optimizations relied on the use of software to compensate for layout quirks. For example, reading from the network port produced the inverse of what was written. Correct execution depended on the assembler to detect the network port as one of the sources and modify the ALU operation, relying on the flexibility of the ALU to generate a compensating function.

Another class of optimizations arose as a result of removing abstraction boundaries. For example, reading from the $\overline{\text{bit}}$ bus results in inverted values, but this value need not be corrected until it reaches the ALU inputs, and there the correction need not add an inverter delay but simply reverse the outputs of a local buffer.

4.6 Testing the Chip

This section describes the methodology and results of testing the chip. Its primary purpose is to document the testing environment to validate the test measurements.

The test setup consisted of four hardware and software modules: the test board itself, a clock driver, a scan controller, and a Sun workstation. The test chip was mounted on the test board described earlier. The clock was provided by a clock board which was essentially an ECL flip flop in a divide-by-two configuration connected to an oscillator socket. The socket was driven by an ECL-level oscillator or a waveform generator for high and low-speed tests, respectively. The flip-flop ensured that the clock duty cycle would be close to 50%. The scan controller was a DSP board based on the TI C30 processor. The controller interfaced with the test board through a four-wire CMOS-level IEEE TAP interface. A Sun workstation communicated with the controller through a conventional RS-232 interface.

4.6.1 Test Board

The test board consisted of 12 layers, as described in Table 4.6. As well as testing the chip, designing the board served to demonstrate that routing of the fairly wide busses, both instruction and mesh, was possible. The routed single-chip wiring can easily be replicated as a macro route multi-chip boards. Six signal layers proved sufficient to route all signals. The test board also demonstrated that PCB design software was able to route differential ECL signals well.[6]

[6] Whee! –Carlin

The board wiring methodology was controlled impedance *offset stripline*, with horizontal and vertical pairs of routing between two power or ground planes. These planes isolated the layers, and the orthogonal orientation between the planes ensured that signals within the same ground plane pair did not couple over long distances.

The board accepted four separate power supplies. Even though the ECL circuits were operated at CMOS levels in offset mode, between 0 and 5 volts, instead of -5.2 V and 0 V, the board also included a 3 V termination voltage plane, and a 1 V mesh signalling plane.

Layer	Name	Orientation	Purpose
1	TOP	Random	SMT/via connectors.
2	V_{TT}	Plane	ECL termination voltage. Tied to 2V.
3	X1	Horizontal	Routing
4	Y1	Vertical	Routing
5	V_{CC}	Plane	Positive power supply. Tied to 5V.
6	X2	Horizontal	Routing
7	Y2	Vertical	Routing
8	V_{EE}	Plane	Negative power supply. Tied to 0V.
9	X3	Horizontal	Routing
10	Y3	Vertical	Routing
11	V_{CE}	Plane	1 V power supply. Tied to 1V.
12	BOT	Plane	Negative power supply. Tied to 0V.

Table 4.6: Board Stackup

The top and bottom layers of the PC board were not used for wiring. The bottom was unused because the dense wiring area around the PE chip was covered by the interposer clamp block.

4.6.2 Test Process

The chip was tested through the Test Access Port (TAP) interface as follows:

1. The user writes a C-based test program on the Sun workstation using calls to the Abacus assembler library.
2. The test program runs on the Sparc and generates a driver C program targeted for the DSP board.
3. The user downloads the driver object code to the DSP board.
4. The driver runs on the DSP board and uses the C30 digital I/O ports to control the TAP interface on the Abacus chip.

The clock board was driven by one of two crystals, either 200 MHz or 160 MHz, or a waveform generator with a maximum frequency of approximately 25 MHz. Since the clock board divided the clock by two, the resulting chip test speed was either 100, 80, or 12.5 MHz.

4.6.3 Test Results

All data shown in this section was obtained with a Tektronix TDS 460 digital sampling oscilloscope. The probes were custom low-capacitance 21:1 shop probes made with a 1 KΩ isolating resistor and RT58 coax cable.

Heartbeat monitor. The first test was to ensure that the differential ECL clock signal was received properly and reached the on-chip timing generator. One of the four debug pins output a copy of the `Read` clock taken from the eastern edge of the PE array. This 3.3 ns pulse is shown in Figure 4-13. As expected, the pulse occurs a fixed amount of time after the rising clock edge and is not affected by the duration of the clock period.

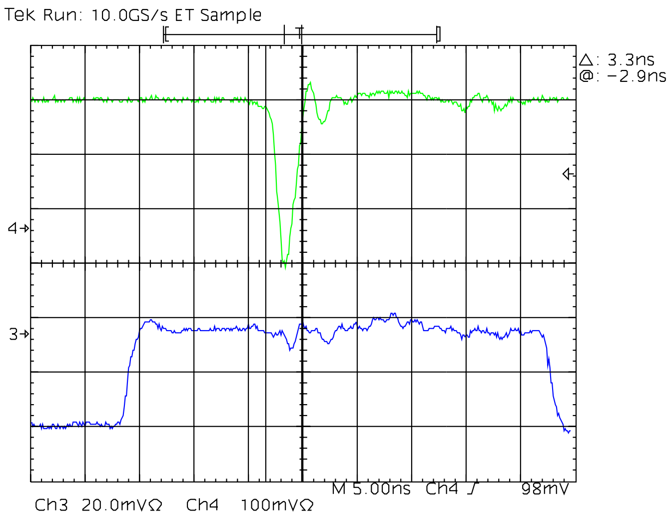

Figure 4-13: The heartbeat monitor: a copy of the read clock.

TAP Interface. The next testing step was to communicate with the TAP interface. Figure 4-14 shows the output of the serial *TDO* (TAP Data Out) pin after the reset signal was asserted. This test demonstrated that the TAP was functional and that further testing could proceed.

IO Burst. A bit pattern was loaded into the top PE row with the literal load command. It was then transferred to the IO plane, and then to the output register. Finally, the IO burst was enabled to produce the waveform shown in Figure 4-15. The specific bit pattern is

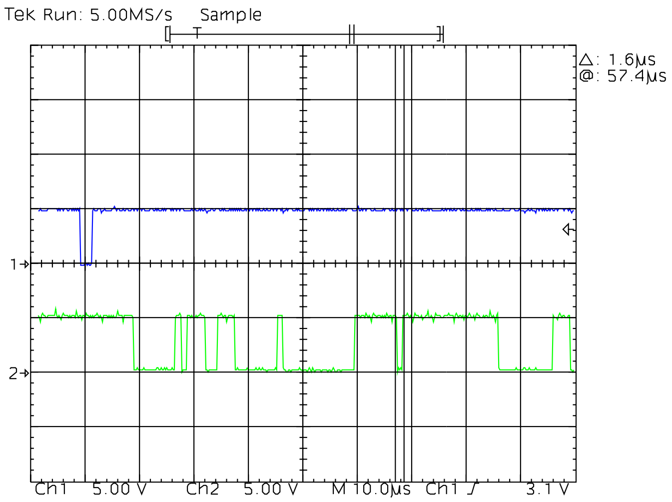

Figure 4-14: TAP Interface

0xFF30A7. The pattern is reversed, since the LSB appears first on the output of the scope. The ground/supply voltage bounce is about 200 mV. Although only a single- ended trace is shown in the figure, the output is driven differentially, eliminating the common-mode bounce.

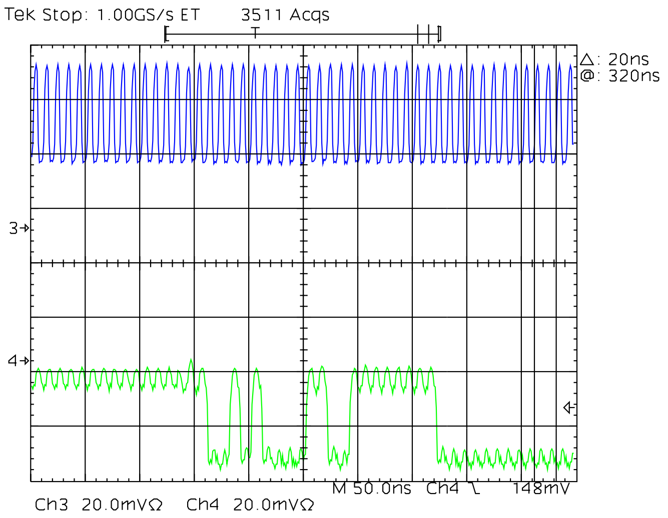

Figure 4-15: Output Burst Trace, 100 MHz.

Data Plane Shift. The next test involved testing the operation of the data plane. The DP register was set high, and then the data plane shifting was enabled. The DRAM I/O pins floated low, so that a zero propagated through the data plane as shown in Figure 4-16. As expected, after thirty-two cycles, the zero appeared at the top row. For this test, the global OR output pin was used, testing both subsystems.

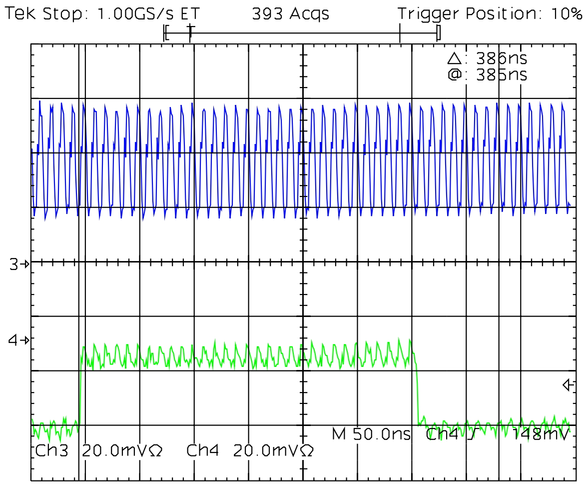

Figure 4-16: Data plane toggle at 100 MHz. The data plane is loaded with 1s. A zero is then introduced at the input. 32 cycles later the 0 is has been shifted through.

External Memory Pads. The DRAM pads were toggled at 100 MHz.

Register File Toggle. The next test showed the first failure of the chip. The repeated instruction was very simple: toggle a memory bit and copy it onto the burst output pin. This operation started failing at 80 MHz. Instead of a pattern of alternating zeroes and ones, the pin showed two high values followed by a low.

mesh Shift Operation. The on-chip mesh communication was tested next. As shown in Figure 4-19, the pattern was shifted out. Although only 20 MHz operation is shown, the mesh shifting worked at 80 MHz, and failed (by shifting incorrect data) at 100 MHz.

4.6.4 Testing Summary

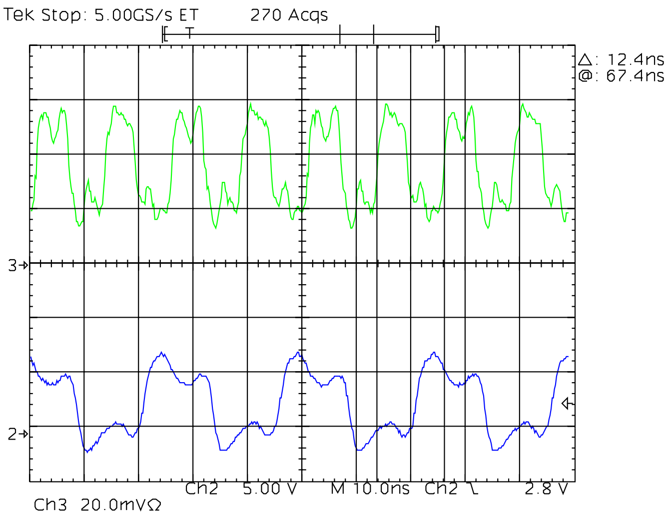

Figure 4-17: DRAM pads were designed to toggle at only 62.5 MHz.

Subsystem	Status
Burst Output	Operational at 100 MHz
Internal Memory toggle	Operational at 80 MHz
On-chip mesh communication	Operational at 80 MHz
Data Plane shift	Operational at 100 MHz
TAP interface	Fully operational

Table 4.7: Testing Summary

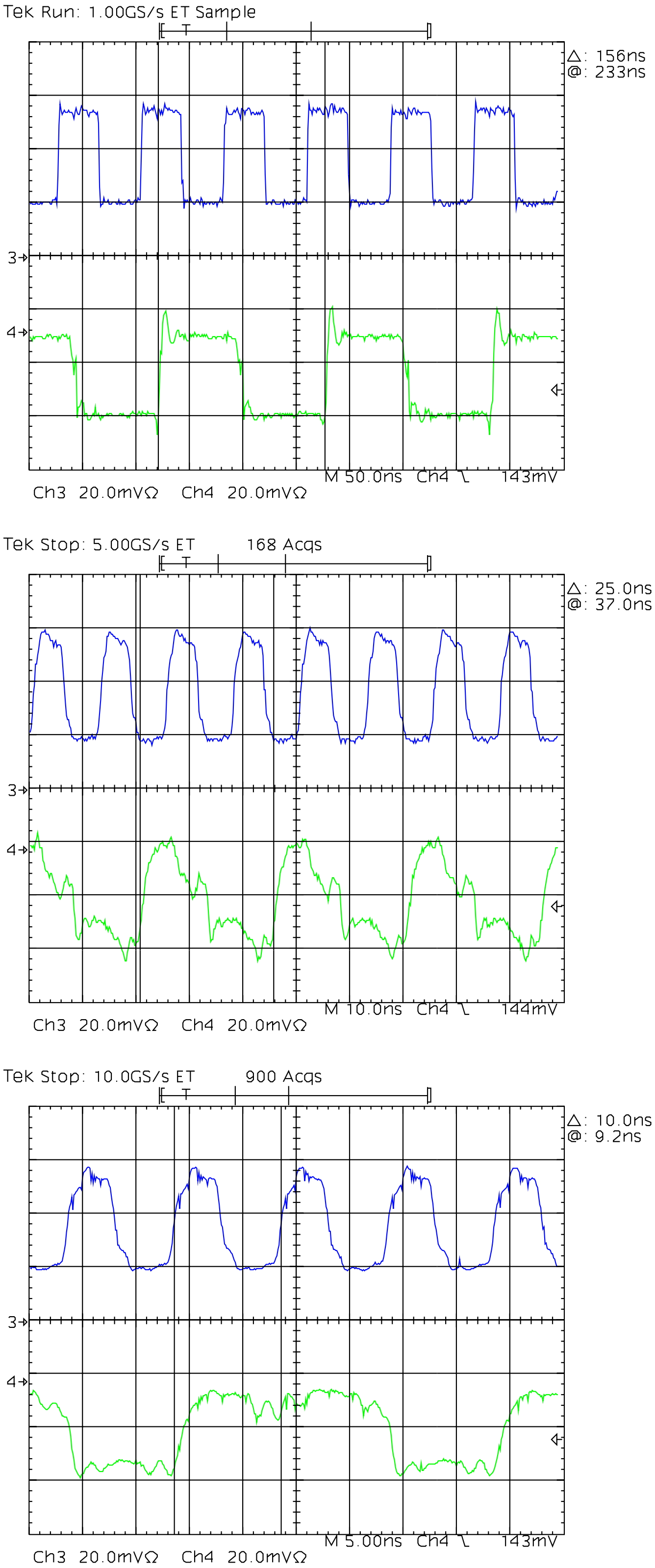

Figure 4-18: IO Toggle

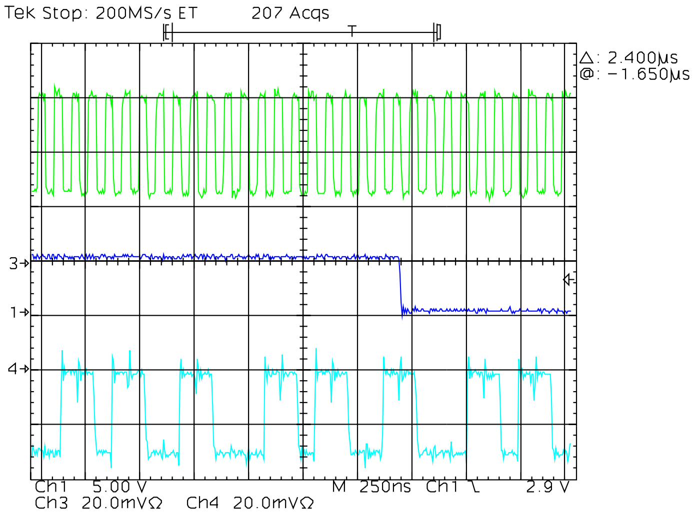

Figure 4-19: mesh shifting at clock rate. Only the 20 MHz operation is shown.

Chapter 5

Abacus-1 System-Level Design

This section contains a detailed design of the entire system, including sequencer, PE array, and I/O interfaces. The topics of this section have largely been ignored by architecture researchers, with most of the attention going to the more theoretically interesting processing element and interconnection network design. However, system interfaces are precisely those elements most responsible for bottlenecks and therefore most likely to cripple the performance of the carefully designed PEs and network. This preliminary design is therefore essential to the viability of the Abacus chip in a realistic environment. Although the system description in this chapter uses present tense, the system has not been built. However, various subsystems, such as the I/O boards, were designed to the point of specifying the chips and completing timing diagrams for typical modes of operation.

The Abacus-1 system consists of the sequencer and instruction memory, the PE array, the input and output boards, and the video I/O boards. They are connected by a shared 32-bit bus and several direct interfaces.

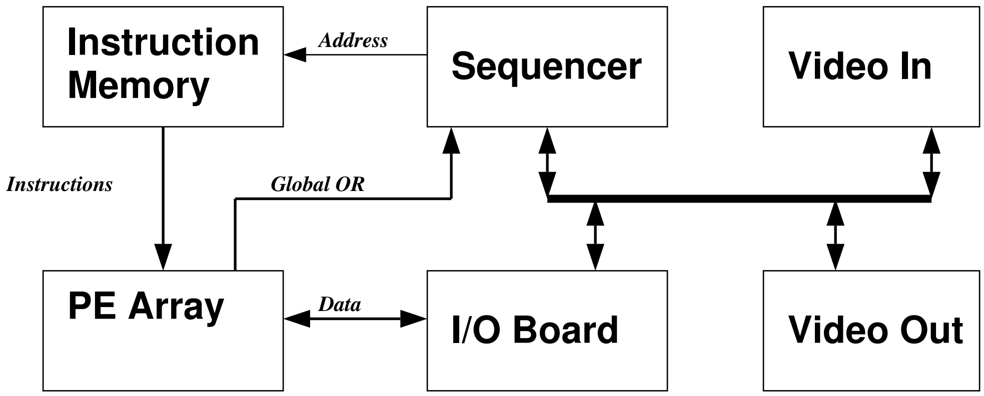

Figure 5-1: Abacus-1 System Block Diagram

5.1 Sequencer

In conventional SIMD systems, low-level microinstructions that drive the processor array are generated at runtime from high-level macroinstructions. For example, an 8-bit integer addition operation expands to eight bit-serial add-with-carry instructions. Since the PE array can operate much faster than the host computer, this instruction expansion places high demand on the host's computation and bandwidth. Even if the host's cycle time is comparable to that of the array, it must typically execute multiple instructions to interpret the macroinstruction.

The typical solution is to place a circuit called a *sequencer* or *microcontroller* betwen the host and the array. The sequencer is responsible for interpreting macroinstructions sent from the host and sending the microinstructions to the array. The sequencer thus acts as a bandwidth amplifier between the host and PE array.

In this model, the host computer executes scalar code on its own. When it is time to perform a parallel operation, the host sends a request to the sequencer to execute a block of instructions.

Although the simplest implementation of the sequencer is little more than an address counter and a microcode dispatcher, its complexity invariably grows. The first step down the slippery slope of complexity is the observation that loops are common and the extra latency of repeated block issues by the host is substantial. Thus, a loop counter is added. Unconditional loops become too restrictive, so sequencer gains the ability to sample the global OR flag from the array. Conditional loops often require a single scalar value obtained from the array (such as the active pixel count) to be compared with some other quantity, and the overhead of transferring this value to the host for comparison is too high. Now registers are added to the sequencer, as well as simple program structure. The sequencer begins to resemble a conventional microcontroller.

This process is of course familiar as the wheel of reincarnation, as more and more functionality migrates between the main computing system and the formerly dumb controller. This migration occurs as a result of technological changes as more silicon area becomes available, as new algorithms are developed, or as subsystem speeds change. It has occurred in systems ranging from disk controllers to graphics renderers.

The Abacus-1 sequencer design is a new point in the evolutionary cycle. It is based on recognition of the following technological factors:

- Memory density has ben growing far faster than code size, especially the highly repetitive code of image processing. There is no need to interpret instructions at runtime when the microinstructions themselves can be stored.
- Contemporary microprocessors already internally run faster than the PE array, and their external bus speeds are within a factor of two of array speeds.

The Abacus-1 sequencer integrates the conventional roles of the host and sequencer. It executes scalar code intermixed with parallel code at a much finer level than before. Specifically, it executes parallel instructions by performing a read operation to a set of addresses,

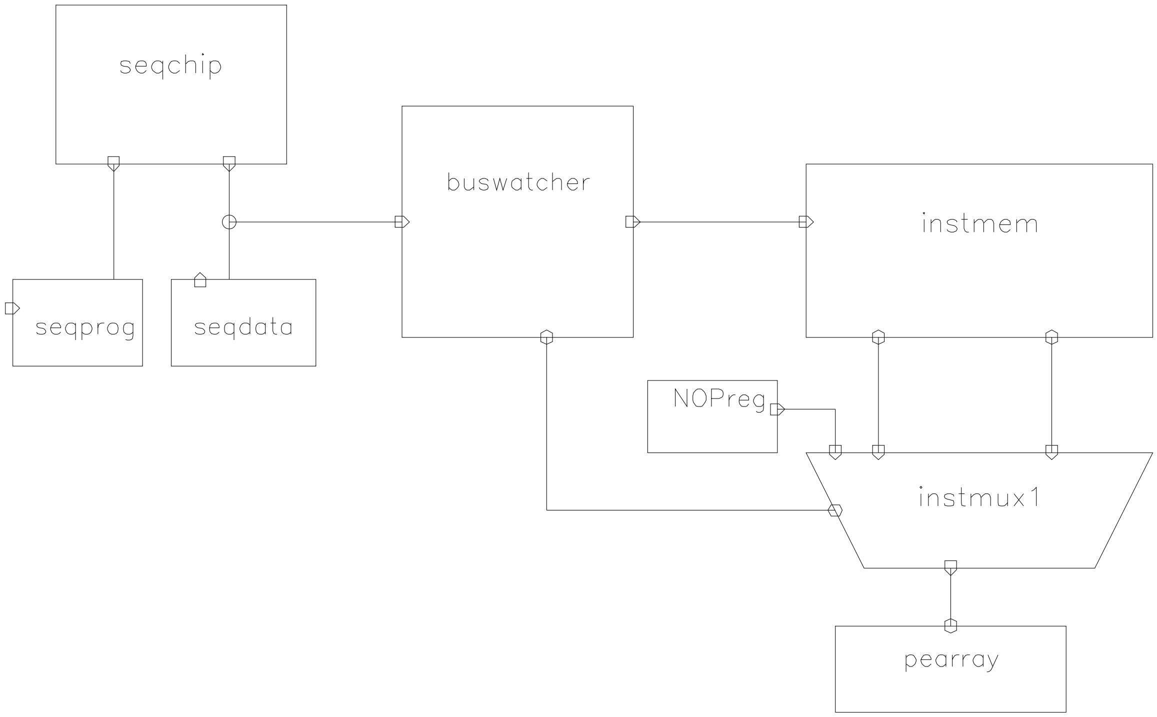

Figure 5-2: Sequencer and instruction delivery path.

as shown in Figure 5-3. A specialized circuit watches the address bus, detects the read, and converts it to a parallel instruction address (PIADDR). The PIADDR is applied to the parallel instruction memory (PIMEM), which is then sent to the PE array. If no read request is issued during a particular cycle, the buswatcher sends a NOP (no operation) instruction to the array. This ability allows variable timing on the part of the sequencer (due to cache effects, for example).

Bandwidth matching between the sequencer's slow bus speed and the array's fast instruction speed is accomplished with the straightforward approach of grouping microinstructions together into blocks. Each read request causes several consecutive instructions to be fetched at once and then delivered to the array in sequence. The disadvantage of blocking is that when control flow changes, NOP instructions have to be inserted to fill the block.

A number of interesting optimizations are possible with this approach. Since the instructions are referenced indirectly, with the read address functioning as a pointer, identical microinstructions need only be stored once despite being referenced from multiple program locations.

The Abacus sequencer consists of a conventional medium-speed DSP chip, the Texas Instruments C40. It has uniform cycle times as it does not use caches. The DSP chip is limited

```
ADD  R3, R1, R2          ;  some scalar operation
MULT R4, R4, R5          ;  another scalar operation
LD   R0, 0xF1000000      ;  NETSOURCE := EAST
LD   R0, 0xF1000001      ;  NET_OUT   := P1
LD   R0, 0xF1000002      ;  P2        := P2 + NET_IN
LD   R0, 0xF1000003      ;  NETSOURCE := WEST
LD   R0, 0xF1000001      ;  NET_OUT   := P1
LD   R0, 0xF1000002      ;  P2        := P2 + NET_IN
ADD  R5, R3, R2          ;  back to scalar code
```

Figure 5-3: Example of interleaved scalar and parallel code. The parallel code implements a 2-point horizontal boxcar filter. Notice that the issued addresses are identical for different microinstructions. This reduces instruction memory requirements. In actual code, instruction reuse will not be as frequent, since instructions are grouped in pairs.

to clock rates of 40 MHz, leading to 3:1 blocking: for every instruction address, three instructions are fetched from the memory. Controller versions of modern fast microprocessors such as the PowerPC or the Alpha support 66 MHz bus speeds, and will allow 2:1 blocking.

5.2 PE Board

The processor board contains 16 PE chips, their associated memories, and instruction distribution circuitry. A floorplan is shown in Figure 5-4.

Instruction Distribution The processor chips are organized in a 4 by 4 array. Each column of PE chips shares a bus of 30 instruction wires. These wires are driven by single-ended ECL drivers at 250 MHz. The bus is parallel-terminated to a 3V power plane (-2 in unoffset ECL) by 50 ohm resistors. The resistors, housed in a SIP with integrated bypass capacitors, serve both as pulldowns and as line terminations. Each set of column buffers is in turn driven by differential line receivers connected to the instruction distribution ribbon cable. The termination for the initial bus is identical to the column bus.

Data Distribution Image data is delivered to the board at 125 MHz, over a 32-wire differentially-driven cable. It is retimed by a set of registers. The retiming allows for different time of flight from the IO boards to different PE boards. Similarly, output data is driven differentially from the PE chips to retiming registers. This retiming compensates for the varying time of flight on the PCB board itself, and eliminates the need to snake traces from nearby PE chips to equalize delays. The registers drive the retimed data differentially off-board. Global OR differential signals from the chips are registered before being presented to an OR gate.

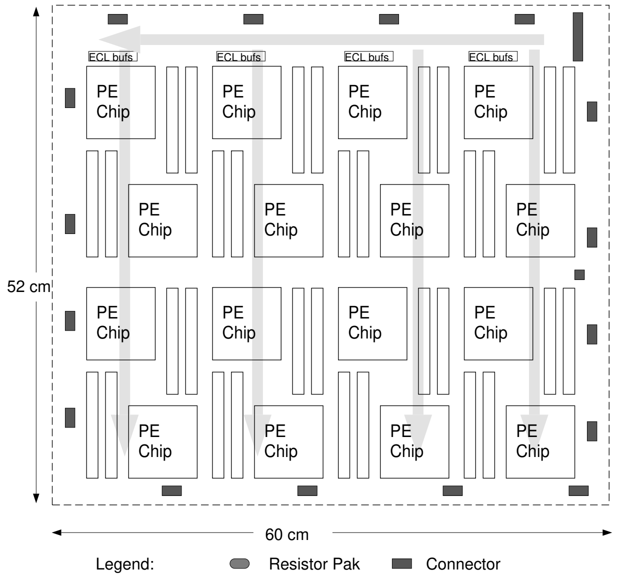

Figure 5-4: PE Board

Part	Desc	Qty	Comment
E122	9-bit buffer	16	4 chips/instruction bus
E451	6-bit D register, diff-in	6	1 register/ chip (GOR and DATAI)
E452	5-bit D register, diff	4	1 register/ chip (DATAO)
E101	Quad 4-input OR/NOR	5	1 gate/chip, 1 to sum (GOR)
E116	Quint diff receivers	6	30 instruction bits
E111	Clock repeater	3	1 / 8 clocked chips
Total		37	

Table 5.1: Board Level SSI ECL Parts Count

Although the Abacus-1 chip was designed for glueless integration with memory, the SSI chips lower the integration level. In production, a two custom VLSI chips, each with 164 signal pins, can replace the 37 SSI chips listed in Table 5.1.

Inter-Chip Network Wiring. Mesh inter-board wiring uses conventional ribbon cable in a GSG (ground-signal-ground configuration). Although this is a surprisingly low-tech approach for 250 MHz signalling, calculcations show that it is very effective over short distances [32]. Each ribbon cable is wrapped in a ground shield to reduce interference between cables in the rack.

Power And Thermal Considerations. Each PE chip dissipates 18 W at 125 MHz. In addition, EDRAM parts are estimated to have a static column read current of 105 mA per chip at 16 ns cycle times, or about 500 mW. Sixteen of these chips, forming two SIMMS, consume 8 W. Termination resistor power is negligible, amounting to 24 mW per resistor, or 2.9 W for the system.

Neglecting termination power, a fully populated PE board requires 416 W and 8 A. Large heat sinks are used on the chip to reduce thermal resistance, and forced air cooling is clearly required. As a result, the boards cannot be spaced closely together. This limitation increases the worst-case signal travel time on mesh wires. Although the cooling and power delivery systems have not been designed, arrays of small fans have been successfully used to cool large dense TTL boards dissipating 3000 W.

5.3 I/O Boards

A key design decision in the array I/O subsystem is the access of the sequencer to the array data. Direct access to the PE chip internals requires additional data pathways. This was judged to be too pin-intensive. Instead, the access model is that an entire plane is dumped from the array to a memory bank close to the PE. This approach supports dense data transfers such as that of image data directly to the frame buffer. In a sparse access, the sequencer accesses a small subset of the computed results. A block transfer of one data plane requires approximately 1000 cycles, or 8 microseconds. We assume that this is not a substantial overhead compared to the overall algorithm time.

Block transfer is used in both the input and output boards. Since the boards are fairly symmetrical in design, only the output (array to sequencer) board is described.

The first challenge is to receive the large number of ECL level signals from the PE boards. There are 16 signals per board and 16 boards in the system, for a total of 256 signals at a data rate of 125 Mbits/second. The received signals must then be converted from a bit-serial to bit-parallel form. This is more complicated than a simple serial-to-parallel conversion since a single data word is scattered across several serial transmissions (four in the case of a 16-bit four by four arrangement). After the pixels are reformatted, they must be stored in memory as a linearly accessible frame buffer. This organization allows the sequencer to operate on the image efficiently, with loops and auto-incrementing of pointers.

All IO from a group of 16 chips comprising a board is processed by a collection of circuitry called a reformat unit. A reformat unit is composed of a 32-word, 16-bit ECL FIFO, two ECL/CMOS level converters, a SHMUX unit, and two SRAMs. The SHMUX is composed of two FPGAs and is effectively a chain of shift registers and multiplexers. The multiplexer outputs are connected to two SRAM chips. The reformatting proceeds as follows:

1. The FIFO is enabled for input and reads in the 32-bit burst.
2. The FIFO is enabled for output. The ECL signals are converted to CMOS levels and entered into the SHMUX.

3. The controller sequences the multiplexers inside the SHMUX units to select the appropriate bit groups for output. At the same time, appropriate addresses are presented to the SRAM so that the bitwise reformatted data are image-wise reformatted.

Notice that the data transfer is pipelined, as the FIFO can be loaded at the same time as the SHMUX is trasferring data to the SRAMs.

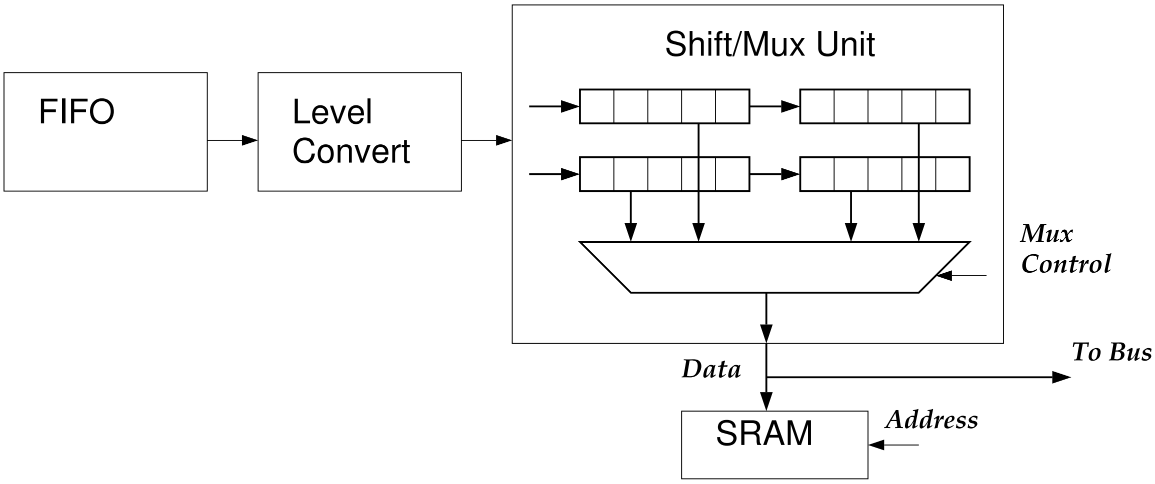

Figure 5-5: Reformatting Circuitry Block Diagram.

The 16 reformat units are controlled by the format controller. The controller consists of a small state machine, a counter, and an address lookup table. The table contains addresses to be presented to the SHMUX units for appropriate reformatting.

The sequencer accesses the board memory by presenting the desired address to the format controller, which maps the address to the appropriate SRAM chip based on the preprogrammed image size table. This address translation adds two cycles of latency to sequencer memory accesses, but allows fast random access.

The input board symmetrically reverses the reformatting process by loading the SHMUX chips from the SRAMs, loading the FIFO from the SHMUX, and then executing burst transfer operations. If need be, the functionality of the two boards can be combined into one with a few multiplexers and extra wiring demands.

The total chip count for the board is 192 plus the 8 or so chips implementing the format controller for a total of 200. Assuming 1 inch chip pitch, and a double-sided surface mount design, each board is about ten inches on a side, which is quite reasonable. We estimate that a single custom VLSI chip can replace two reformat units, for a substantial improvement in integration and cost. The major limitation is the number of pins required: 16 input pins and 32 output pins.

Chip	Quantity
Differential receivers	4
FIFO	1
Level converters	3
FPGA	2
SRAM	2
Total	12

Table 5.2: Reformatting subsystem part count

Chapter 6

Vision Algorithm Implementation

This section describes the implementation of several vision and communication algorithms on the Abacus architecture.

6.1 Basic Vision Algorithms

Edge Detection The widely used Marr-Hildreth edge detection algorithm consists of smoothing the image with a Gaussian filter, computing the Laplacian of the filtered image, and locating the zero crossings of the result [33]. 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.

Surface Reconstruction Surface reconstruction is the task of computing the surface shape (height and slope) from a set of potentially sparse and noisy measurements. Harris [34] 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 RBP 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.

Optical Flow In correlation-based optical flow [35] the original image is displaced in a variety of directions. 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 at a pixel is the exclusive OR of the shifted and unshifted images in a window around the pixel. neighborhood. Finally, each pixel chooses the displacement with the smallest difference value in a winner take all 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.

(a) (b)

Figure 6-1: Optical flow calculation with maximum displacement of 2 pixels and a 5 ×5 summation region. (a) original image. (b) displaced image. The top shape moved up by one pixel and partially off the image; the right shape moved down by one pixel and the lower shape moved three units down and one to the left.

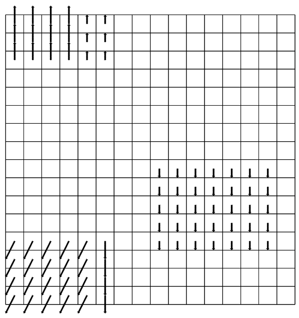

Figure 6-2: Optical flow field resulting from the raw data in Figure 6-1. Some confusion is caused by the movement of the top shape off the image, and by the movement of the lower shape by a distance exceeding the maximum displacement layer.

6.2 The DARPA IU Benchmark

This section describes the subset of the DARPA Image Understanding Benchmark that falls in the domain of early vision.

Connected Components: Broadcast. A common problem in image analysis involves labeling the connected regions of constant pixel intensity, where unique regions are constituted from a definition of either 4-connectedness (horizontally or vertically adjacent), or 8-connectedness (horizontally, vertically, or diagonally adjacent). This operation is discussed for binary images in the literature ([36], [37], [38], [39], [40]) but has been extended for grayscale values in the following three algorithms. The first approach described here is the most straightforward but requires the most time, the second alternative is faster but requires considerable amounts of memory, and the third is a modification of the second that compresses memory requirements drastically at the cost of increased computation.

In the simplest algorithm, each pixel of the intensity image is labeled uniquely from its PE row and column. This intensity and its corresponding label (of $O(\log N)$ bits) are broadcast to each of its 8-connected neighbors. At each PE, the neighbors of matching intensity are determined, and the minimum of the their corresponding labels and the current label becomes the new label. This process continues until no more labels in the mesh are updated (checked by a global compare). The number of broadcasting operations required is proportional to the largest "intrinsic diameter" of all connected components in the image, defined as the maximal shortest connected path between any two pixels in the region [37]. For spirals and other high-curvature images, this intrinsic diameter can be as high as $O(N^2)$ in the worst case, resulting in $O(N^2 \log N)$ bit operations.

Connected Components: Shrinking. Levialdi's region shrinking operation [36] provides a iterative method to directionally compress each region down to a single pixel and then remove it entirely without fragmenting or fusing separate regions. If the results of each operation are saved away, the operation can be reversed to generate a unique label when a region consists of only one pixel, and that label can be transmitted to all possible neighbors in the direction of expansion so that they can make a decision should they become connected in the next stage. The third algorithm modifies the basic one by storing only a subset of the shrunk images and reconstructing on the fly by repeating the shrinking operation.

K-curvature Tracking and Corner Detection The connected components map is processed to produce K-curvature values for those pixels on the component borders, which are then smoothed with a Gaussian filter to eliminate multiple peaks near corners. Pixels with smoothed curvature values exceeding a threshold value (peaks) are intersected with the zero-crossings of the first derivative of smoothed curvature to extract candidate corners in the image. Border pixels are defined to be any pixel adjacent (N,E,W,S) to a pixel belonging to another component. K-curvature is defined at each border pixel as the interior angle between the two lines passing through the current pixel and those K border pixels away in either direction along the region's border. See Figure **??**.

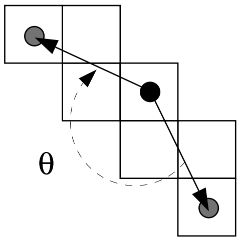

Figure 6-3: K-curvature definition. In this figure, θ is the K-curvature for K=3. Note that all pixels shown are edge pixels.

Median Filter The median filter is a common image processing operation. Unlike linear filters, each pixel replaced not by a linear combination of its neighbors but rather by the median value. The operation is effective for removing high frequency "speckle" noise without degrading the rest of the image.

Gradient Magnitude By computing the magnitude of a discrete intensity gradient in the image and thresholding the result, strong direction-independent edges in the map can be located. Every pixel intensity in a 3x3 neighborhood about the current PE is multiplied by a weight according to the Sobel X and Y masks and summed to form the Sobel X and Y magnitudes. Since the weights are either 0, 1, or 2, the multiplications are converted to shifts.

The gradient magnitude is the square root of the sum of the squares of the X and Y magnitudes. Results greater than a threshold value are flagged to create a boolean edges map. The operation is constant in space and time with respect to N.

Hough Transform The Hough transform partitions a binary image into discrete bands one pixel thick and oriented at a certain angle, and sums the values within the band to yield a set of projections that can be scanned to locate strong edges in the original image at that angle [41]. Usually transforms are computed for many angles, so that the aggregate data can provide insights into the image properties.

The implemented algorithm partitions the image into bands of constant line offset ρ for a given angle θ according to the equation: $\{(x, y) : x \cos\theta + y \sin\theta = \rho\}$ where (x, y) are PE coordinates and θ is assumed to be in the range $\pi/2 \leq \theta < 3\pi/4$ (the other angles can be accommodated by pre-rotation of the image). A "band total" variable visits all PEs in its assigned band by shifting east, and north if necessary (a result of the angle range is that at most two pixels in the same column belong to the same band). The PEs in the first column are visited first, and then the variable travels eastward across the columns. Since there are many angles to be projected, they are pipelined one column at a time, yielding P projections on an NxN image in $O(N+P)$ time.

6.3 Sorting and Routing Algorithms

Sorting algorithms can be converted into routing algorithms by associating data with the sort key and ensuring that the combined packet is moved together. This allows packets to be routed to a destination. Two types of sorting algorithms were examined: those with a fixed execution time, and those with a data-dependent execution time.

ShearSort and RevSort ShearSort is a $\sqrt{N}(\log N + 1)$ step sorting algorithm [37]. It consists of sorting rows in alternating opposite directions starting at the left edge of the array, then sorting the columns from top to bottom. ShearSort relies on the TranspositionSort [42] procedure to get things sorted. TranspositionSort is the basic linear array sort. It works similar to the uniprocessor BubbleSort: comparing then exchanging (where necessary) alternating pairs of items.

An improvement of ShearSort is called RevSort [43]. It is identical to ShearSort, except CyclicSort is used to sort rows instead of Transposition Sorts. This algorithm finishes in $\sqrt{N}(\log\log N + 3)$ steps and the complexity of the each step is equivalent to each step of ShearSort. CyclicSort is a descendant of the TranspositionSort, and performs as fast as its ancestor. The smallest item is sorted to a chosen processor instead of the leftmost processor.

Mesh Greedy Routing Algorithm A very different packet-based routing algorithm was developed by Herbordt [44]. Each PE emulates two communication channels, one vertical and one horizontal. Packets are moved one step through the X channel until the correct X coordinate for the packet is reached. At that point the packet is moved to the Y channel and proceeds vertically. The X and Y routing steps are interleaved, so that one of each occurs during each iteration. Notice that packets can be blocked from switching to the Y channel if that section of the channel is full. When this occurs, packets behind the blocked PE in the X direction are also blocked. The blocking information propagates backward along the X channel one step at a time. Since each PE contains space for two packets, collisions will not overwrite data, and the blocking information can afford to propagate only one step per cycle.

The key difference from the user's point of view is that this algorithm is not guaranteed to finish in $O(\sqrt{N})$ iterations. In fact, the worst case performance is $O(N)$, where N is the number of PEs. According to Herbordt, the worst case is very unlikely to arise in either a completely random, or typical routing conditions. In fact, the algorithm generally does not need to be iterated more than 2.5 times the optimal case of $2\sqrt{N} - 2$. The paper demonstrates that this property holds for a number of common permutations, such as transpose, bit shuffle, and rotations.

6.4 Performance Summary

In the vision algorithms table, computation times are shown for a 128 by 128 array processing 128 by 128 pixel images, without virtualization. Processing larger images would not scale up linearly, since communication time would be reduced, but data I/O time would be increased. Memory-intensive algorithms would therefore degrade much faster.

Algorithm	Cycles 1000s	Time (μsec)	Memory 16-bit regs
Edge Detection $\sigma = 2.0$	0.45	3.6	3
Optical Flow, $\Delta = 5$, 5×5 region	10	80	8
Surface Reconstruction (1 iteration)	0.38	3	6
Connected components, $D = 256$	42.5	340	9
Hough transform, $P = 90$, multiplies	113	904	22
Hough transform, $P = 90$, differencing	25.6	204	22
Hough transform, $P = 90$, precompute	10.4	83	67
K-curvature, $K = 4$	8.2	66	46
Gradient Magnitude	0.5	4	9
Shear Permute, 1 data item	167	1340	15
Rev Permute, 1 data item	78	624	15
MGRA (average case), 1 data item	45	360	18

Table 6.1: Vision Algorithm Performance Summary

Chapter 7

Analytical Modeling

7.1 Optimal Bit Width Analysis

Selection of the bit slice width in a SIMD computer is yet a solved problem. Early PEs were bit-serial due to VLSI area constraints. Some designs have upgraded to a conventional 32-bit datapath, while others moved to only 8 bits, even in a contemporary technology. This section develops a simple model that examines various slice widths in terms of silicon area efficiency.

Development proceeds in four steps. First, we estimate the area, delay, and cycles-per-instruction (CPI) of a given bit width and simply multiply to obtain the figure of merit. This metric, called *quality* in this work, is normally expressed in bit operations per nanosecond per unit area, but for easier intuition has been scaled to billions of 16-bit operations per second per standard chip.

Second, we use Amdahl's law to account for operations which are not sped up by a wider datapath. Third, recognizing that off-chip memory latency is a critical bottleneck, we determine the effect of cache misses. Finally, we evaluate the performance gain of the background loading mechanism on the Abacus architecture. We begin by describing the assumptions and limitations of the model.

Parameter	Value	Description
A_{ALU}	8	ALU area, in units of SRAM cells
f_A	0.9	Fraction of operations affected by wider datapath
T_{mem}	3 ns	Register file access time
T_{alu}	0.7 ns	ALU datapath bit delay
T_{ovr}	1.0 ns	Timing overhead

Table 7.1: Analytical Model Parameters

7.1.1 Ideal Model

PE Area A processing element's area is modeled as consisting of three parts: the memory cells, the ALU, and the other overhead circuitry, including the network and the data plane registers. The basic unit of area is the SRAM cell.

$$A_{PE} = N_{mem} + kA_{ALU} + A_{ovr} \tag{7.1}$$

where k is the width of the datapath and N_{mem} is the number of memory bits. Thus, area growth is linear in the datapath. There will be slight area growth in overhead components, such as in the size of the buffers controlling the datapath, but that can be folded into the ALU area. An important immediate assumption is that AT^2 circuits such as multipliers and barrel shifters are not present. These elements have important applications and should be analyzed in later work, but their inclusion in a massively parallel system must be evaluated carefully, since their area grows quadratically with decreased cycle time, and parallelism provides a more efficient alternative.

Cycle Time The model assumes an unpipelined design, with the cycle time consisting of four parts: register file read, ALU execute, register file write, and miscellaneous time margin. Register file access times are assumed identical for read and write, and include precharge and equilibrate time.

$$T_{PE} = 2T_{mem} + kT_{alu} + T_{ovr} \tag{7.2}$$

ALU times are also assumed linear. This holds true for both the ripple-carry adder and the Manchester-carry adder (although in that case the fixed part of the cycle will grow due to the PG and Sum cells, and T_{alu} will decrease). Many more sophisticated adder structures are possible but they require more area, and gain in speed only at larger word sizes.

Performance In the ideal model, speedup is linear in the datapath width, since the clock cycle has been extended. The combined expression for the performance per unit area is

$$Q_{ideal} = \frac{k}{A_{PE}T_{PE}} = \frac{k}{(N_{mem} + kA_{ALU} + A_{ovr})(2T_{mem} + kT_{alu} + T_{ovr})} \tag{7.3}$$

This function is shown in Figure 7-1. Several conclusions can be reached from this plot. First, smaller memory sizes lead to higher performance. This is not surprising, since smaller PEs with identical performance lead to better quality measures. Second, as memory increases, the optimal data width also increases. This occurs because allocating a few more datapath bits from a large PE improves performance without a substantial change in area. The final conclusion is that the function has a very flat optimum, so that it does not cost much to err on the side of larger k (wider datapath).

The next observation to be gained comes from normalizing the performance to the bit-serial

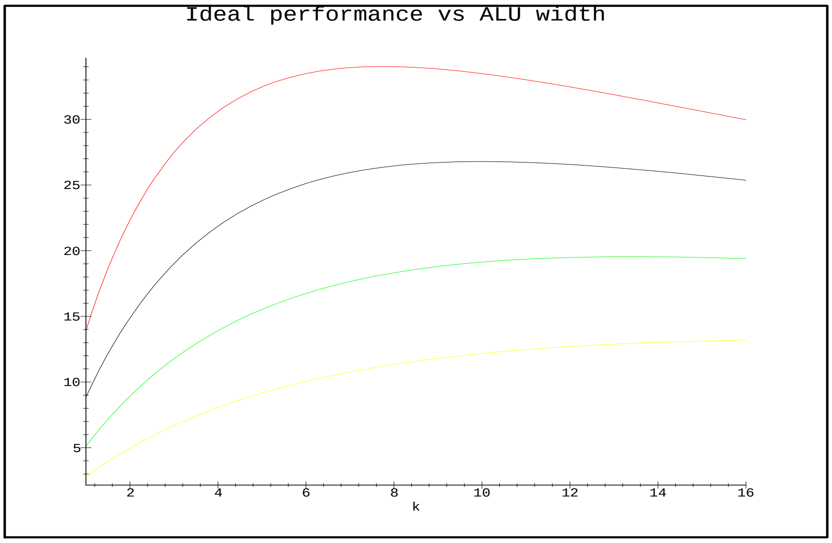

Figure 7-1: Ideal Performance (GOPS/chip) as a function of datapath width. The four graphs correspond to memory sizes of 32, 64, 128, and 256, from top to bottom.

case ($k = 1$). Figure 7-2 shows that wide datapaths can be up to 4.5 times more efficient than the bit serial case. Unfortunately, the greatest relative advantage occurs at a lower absolute performance level.

7.1.2 Amdahl's Law

The next refinement to the model is the application of Amdahl's law, which essentially states that if parallelization (or any speedup) affects only a subset of the total computation, the remaining fraction will greatly reduce the effectiveness of the speedup. Mathematically:

$$S_A = \frac{1}{\frac{f_A}{S} + (1 - f_A)} \tag{7.4}$$

where f_A is the fraction of the computation affected by the speedup S, and S_A is the actual obtained speedup. This is the basic equation of computer architecture.

In the context of the current design, f_A represents the fraction of arithmetic oriented operations in a program. Non-arithmetic operations that are not sped up by a wider datapath include network communication and logical flag-manipulating operations. In a multi-chip system, every communication must cross a chip boundary, and the number of pins does not grow with data path width. Simple filter-style algorithms such as convolutions require one mesh move operation for each multiply and add. If the multiplication is by a known constant, an average eight-bit multiplication requires four shifts and five adds. Thus, ap-

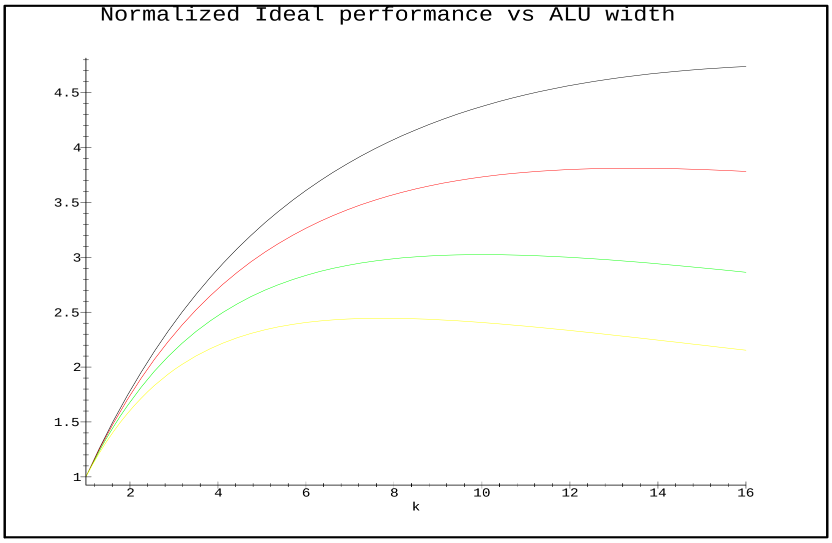

Figure 7-2: Normalized Ideal Performance. This shows the relative advantage of the wide datapath over the bit-serial case.

proximately 90% of the operations benefit from a wider datapath.

The value of f_A used in the model was chosen to be 0.9. It was supported by the results of two related publications. Holman [21] explicitly listed f_A for a variety of algorithms, as in Table **??**. The obtained values are slightly high, as most of the algorithms are dominated by floating point manipulations.

Program	f
Bitonic Sort	0.75
Matrix Product	0.94
LU Decomposition	0.76
Cholesky Decomposition	0.25
Jacobi Method	0.91
SOR Method	0.91
SIMPLE	0.71

Table 7.2: Fraction of Operation Costs Affected by Data Path Width. From [21]

Herbordt's [23] evaluation of parallel architectures included a set of simulation results relating datapath width to execution time. Amdahl's function was fitted to the data points and generated good agreement with data (although half of the algorithms had unexpectedly high performance for the case $k = 1$). The fitted data is tabulated in Table 7.3. The algorithms clearly fall into two classes: mostly bit oriented, with only $f_A = 0.2$ and mostly word oriented, with $f_A = 0.9$. The IU Benchmark entry is a composite of several algorithms.

Program	f
Region-based Line Finder	0.15
Curve-Fitting Filter	0.92
Correspondence Problem	0.21
Fast Line Finder	0.07
IU Benchmark	0.41
Depth From Motion	0.80

Table 7.3: Fraction of Operation Costs Affected by Data Path. From [44]

The same configurations as in the ideal case but with Amdahl's-law correction are shown in Figure 7-3. The greatest difference occurs in the high-performance low-memory case. Here, the additional dead weight of the wide datapath is a relatively large penalty compared to the small overall PE size. The maxima are much sharper than in the ideal case, but level off quickly as memory size is increased.

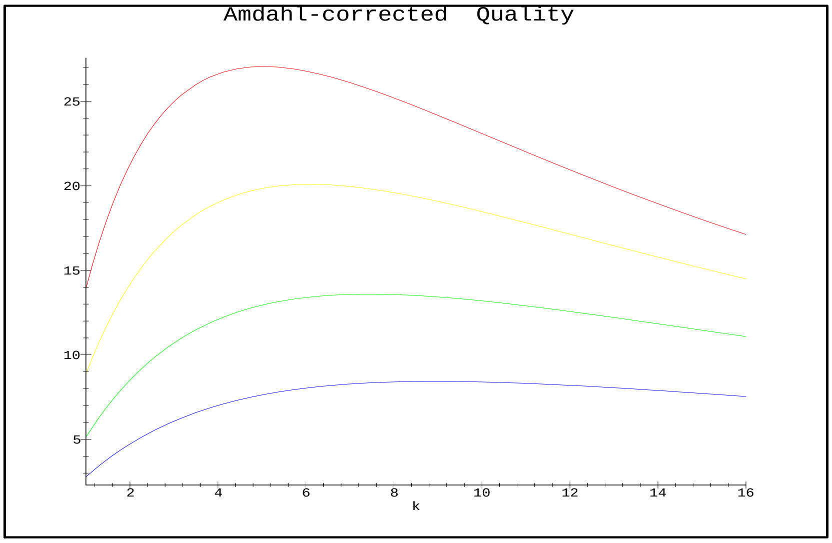

Figure 7-3: Amdahl-corrected quality values.

The normalized equations behave similarly to the ideal case, except that maxima now become apparent. Also, the relative overall advantage of wider paths decreases from a former peak of 4.5 to 3.

The effect of varying the instruction fraction is shown in Figure 7-5. The axis are actually

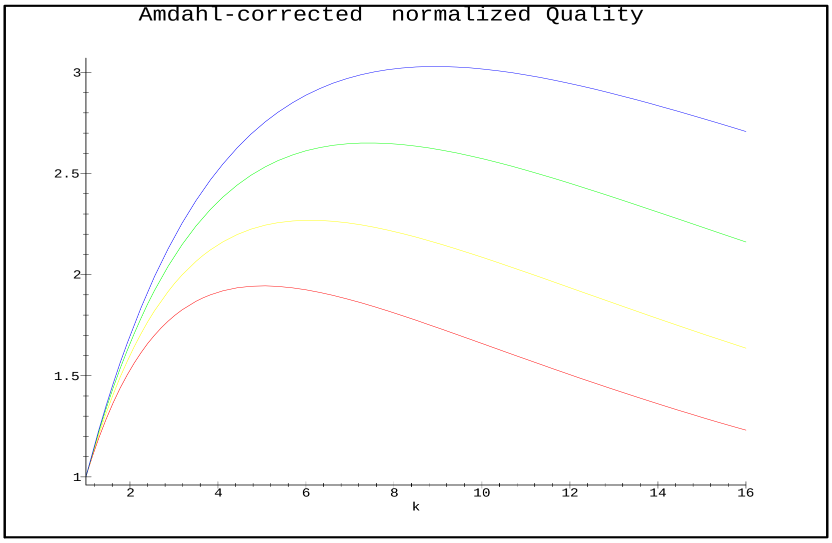

Figure 7-4: Amdahl-corrected quality values

labelled with $(1 - f_A)$, the fraction not affected by the speedup. The expected interaction of the two dimension occurs. The best k for this configuration appears to be 4-6 bits over a wide range of f_A, 0.8 to 0.98. Notice that in the low f_A case, the multibit configurations perform worse than the bit-serial ones.

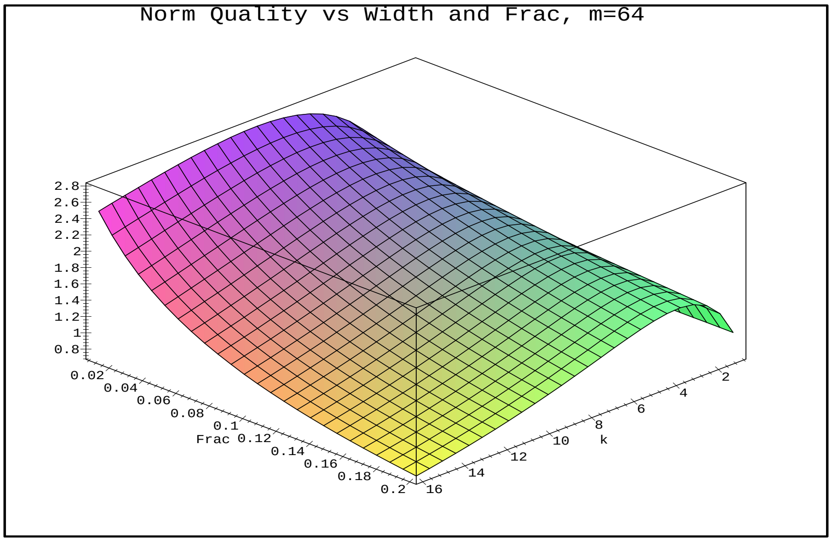

Figure 7-5: Normalized quality vs k and f_A

7.1.3 Off-Chip Data

The preceding analysis applies to algorithms that execute from the on-chip registers. As soon as a data item must be brought in from off-chip, a significant delay occurs. This section analyzes the impact of this delay.

An implicit assumption being made here is that commercially available memory chips are least one technology jump from that available to a university or even a low-volume commercial design. If the SIMD chip was implemented in a comparably dense technology, other design points would become feasible, and the model will have to expanded. A recent example of research in this direction is the Execube chip, which integrates several 32Kx8 DRAM memories with 16-bit PEs.

Once off-chip memories are part of the design, the model must determine two parameters: the available bandwidth per pin, and the number of pins allowed on the PE chip. The former is limited by the capabilities of commercial memory devices, and the latter by packaging constraints. A comparison of contemporary high-bandwidth memory alternaties are shown in Table 7.4.

Name	Width (max)	Cycle (MHz)	Initial Latency (ns)	Data Rate bits/pin/ns	Addr Line Overhead
Rambus	9	600	50	0.6	4
Synchronous SRAM	32	100	5	0.1	18
Synchronous DRAM	32	100	50	0.1	22
Extended DRAM	8	66	30	0.066	22

Table 7.4: Contemporary Memories

The parameter that controls how long PEs must stall waiting for the data is latency: the time from the data request until the data arrives. Different memory organizations have varying latencies, from the 4 ns of high-speed SRAM [45] to the 35 ns of a fast DRAM. SIMD architectures encounter another cause of latency: bandwidth limitation. Since all PEs must wait until data is received, the latency is increased by this wait period. Because of this phenomenon, memory organizations with high latency but with a high burst bandwidth may well exhibit lower latency, especially since SIMD-oriented transfers occur in units of hundreds (or thousands) of bits. The initial latency is amortized over all elements in the burst. The Rambus part falls into this category. Unfortunately, even though products are beginning to incorporate this interface, but the technology is still difficult to design with.

A comparison of these latencies is shown in Table 7.5.

The more conservative design described here assumes a conventional SRAM operating at a 16 ns cycle time through a 64-bit port. The aggregate bandwidth is therefore 4 bits/ns. An important realization incorporated in the model is that for a fixed chip size, bandwidth (and therefore latency, as discussed earlier), is also fixed, and may therefore be expressed as bits per nanosecond per unit area. This somewhat odd normalization is useful because PE

Configuration	Pins Total	Pins Signal	BW bits/ns	Latency initial	data	total
Rambus, 1 port	13	9	5.4	50	190	240
Rambus, 2 ports	26	18	10.8	50	95	145
SSRAM, 32 bits	50	32	3.2	0	320	320
SSRAM, 64 bits	82	64	6.4	0	160	160

Table 7.5: Latencies of Memories

area fluctuates as a function of datapath width, and the available bandwidth changes with it. Based on the Abacus design, the PE area consisted of approximately 100,000 SRAM cell equivalents. Thus the bandwidth per unit area, B, is 0.00004.

The time to provide a data word to all PEs in the array from an off-chip chip memory is:

$$T_l = \frac{w}{\text{PE bandwidth}} = \frac{w}{A_{pe}B} \tag{7.5}$$

where w is the data word width. It will eventually cancel out, so its exact value is not important.

The next parameter to be determined is T_s, the amount of time that passes between load operations. Given these two quantities, the performance degradation D_{io} is given by:

$$D_{io} = \frac{T_s}{T_s + T_l} \tag{7.6}$$

We use a very simple model of load events, namely a fixed frequency. Conventional caches often use Poisson (or other) distributions but control flow is much more predictable in SIMD systems, and loops are likely to access variables in a repetitive stride. A load event occurs with a period of f_{miss}^{-1} instructions. Another important realization of the model is that the duration of a cache hit sequence is a function of the average instruction duration.

$$T_s = \left(\frac{1}{f_{miss}}\right) T_{pe} \frac{w}{kS_a} \tag{7.7}$$

An interesting effect is shown in Figure 7-6. The ratio of T_l to T_s exhibits a maximum at about $k = 6$. This occurs because T_s decreases very quickly at first, far faster than T_l drops due to higher bandwidth per PE. The shape of the ratio curve should appear familiar, as can be seen from the following derivation.

$$D_{io} = \left(1 + \frac{T_l}{T_s}\right) \tag{7.8}$$

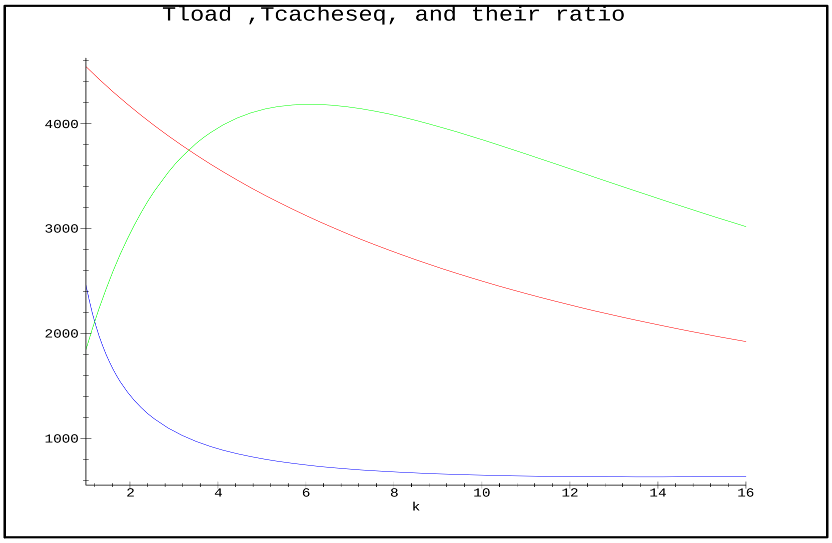

Figure 7-6: Load Time Compared to Load Period. The upper monotonic curve is T_l, while the lower one is T_s. Their ratio, scaled up, is also shown

$$= \left(1 + \frac{\frac{w}{A_{pe}B}}{\frac{T_{pe}w}{f_{miss}}kS_a}\right) \tag{7.9}$$

$$= \left(1 + \frac{fkS_a}{A_{pe}BT_{pe}}\right) \tag{7.10}$$

$$= \left(1 + \frac{fQ_a}{B}\right) \tag{7.11}$$

The performance degradation due to off-chip accesses therefore has a simple form:

$$D_{io} = \frac{1}{1 + \rho Q_a} \tag{7.12}$$

where $\rho = f/B$. This equation says that as the quality factor increases, the impact of off-chip accesses grows. The method of achieving high performance is irrelevant. For a fixed algorithm (f_{miss}) and a fixed memory bandwidth (B), higher performance will throttle itself back at the memory level.

This effect can be seen in Figure 7-7. At high Q_a values, Q_io begins to approach $\frac{1}{\rho}$.

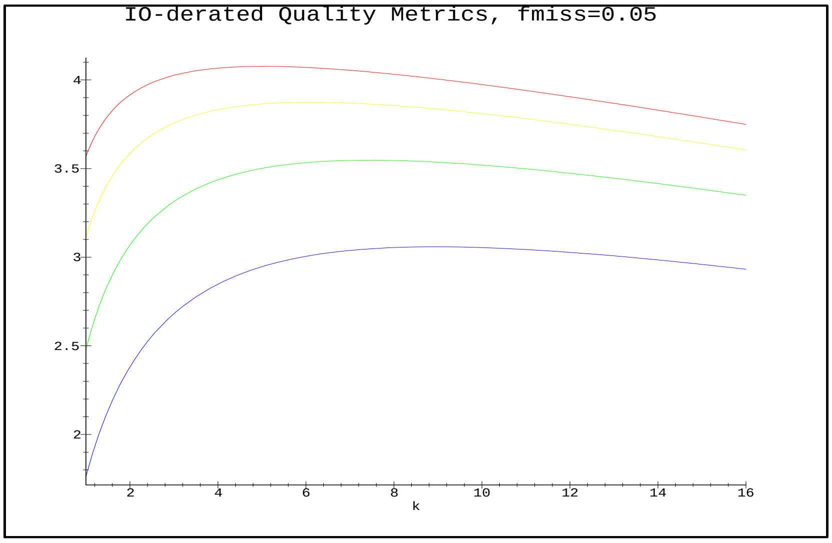

Figure 7-7: IO-derated Quality Metrics

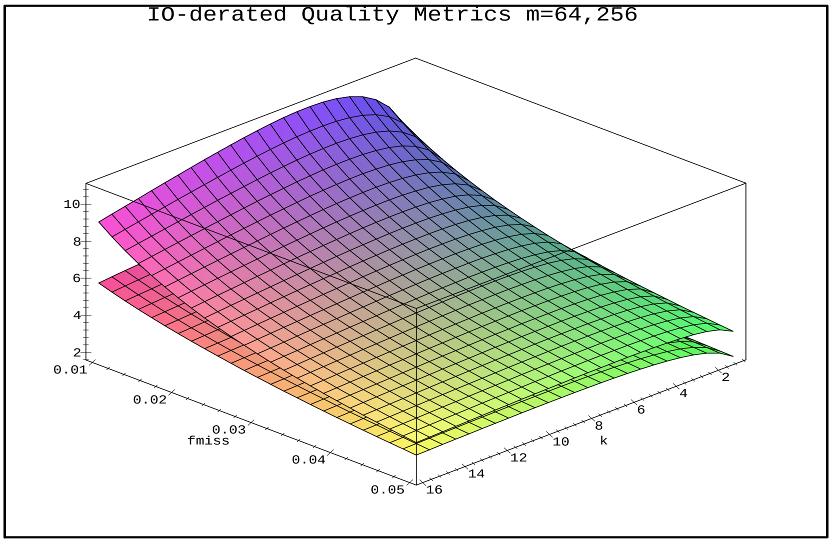

Figure 7-8: IO-derated Quality Metrics (m=64,256))

7.1.4 Background Loading

Computer systems with predictable memory access patterns can be optimized with the technique of prefetching or background loading. The Abacus-1 design allows a memory access to be initiated before the data is actually required, and relies on the compiler to perform this scheduling rather than analyzing access patterns at runtime.

The net effect of background loading is to reduce the loading time penalty. The effective loading time is:

$$T_{eff} = max(0, T_l - T_s) \tag{7.13}$$

and the derating factor becomes

$$D_{back} = \frac{T_s}{T_s + T_{eff}} \tag{7.14}$$

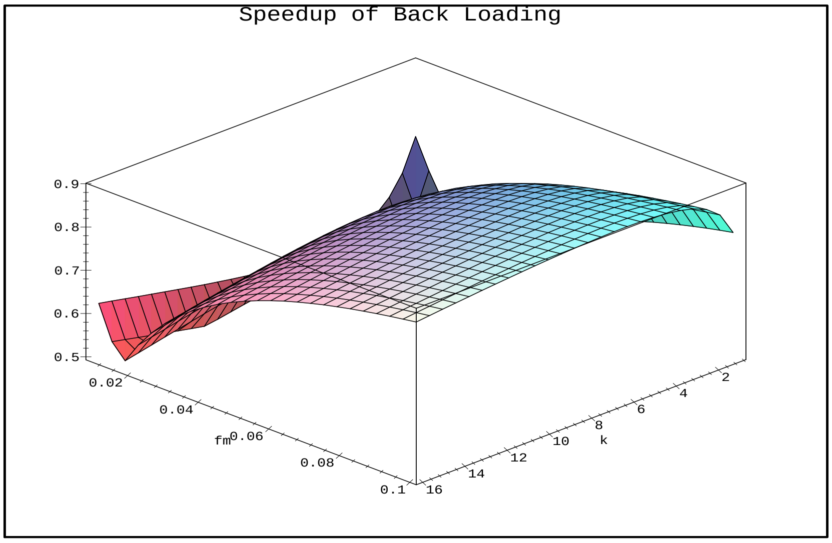

Figure 7-9: Performance improvement due to background loading.

The derating reduction improvement due to prefetching is shown in Figure 7-9. There are several interesting aspects of this graph. First, at high miss rates ($f_m = 0.1$), prefetching only helps by about 10%. The improvement is very rapid as hit rates increase, reaching the 50% "sweet spot" at $f_m = 0.02$. Second, the relative improvement begins to worsen again at very low miss rates, when prefetching completely eliminates the load penalty and therefore cannot improve performance further, while the non-prefetched design is still reaping the benefits of less frequent accesses.

Interesting effects can also be seen in the absolute performance curves in Figure 7-10.

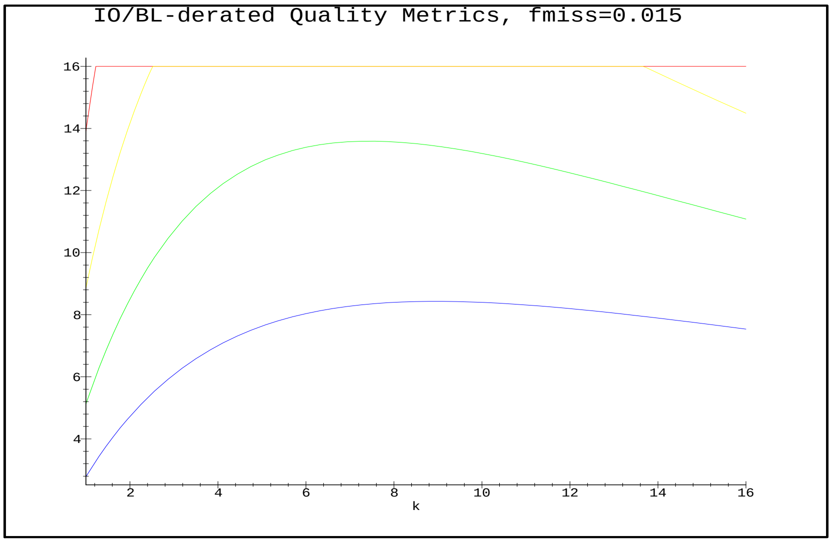

Figure 7-10: Absolute performance, background loading.

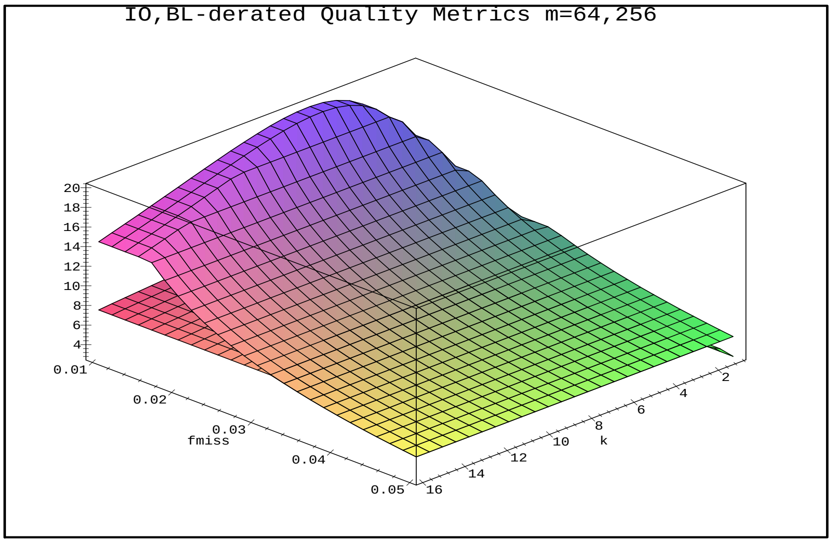

Figure 7-11: Prefetching-derated quality metrics

Prefetching is so effective that performance saturates at the memory bottleneck. A two-dimensional version of this plot is shown in Figure 7-11.

Repeating the mathematical analysis of the previous section leads to very interesting results. First, we need only look in the case where $T_{eff} > 0$, since otherwise the load penalty is zero, and $D_{back} = 1$.

$$D_{back} = \frac{T_s}{T_s + (T_l - T_s)} = \frac{T_s}{T_l} = \frac{1}{\rho S_a} \tag{7.15}$$

This equation says that prefetching makes performance independent bit-slice width! When this computation is made explicit, the intuitive explanation becomes apparent: a long as a sequence of computations takes less time that the load delay, there is no benefit to improving the datapath efficiency. There are several observations:

- Prefetching makes the IO system behave in two qualitatively different modes. If the load penalty is smaller than the sequence delay, the external memory seems to disappear, since the PEs operate as if all data was in local registers. If the load penalty is greater than the sequence delay, the external memory becomes the only factor controlling performance.
- For a memory-limited design, increases in computational efficiency are invisible. Even if a clever data representation halves the computing time at no area penalty, the overall performance will be completely unaffected.

The advantage of reconfigurable systems now becomes clear. Whenever an algorithm enters a bandwidth-saturated phase, a reconfigurable system can configure itself as a wider datapath with more local memory to reduce the miss frequency and therefore improve performance. For purposes of illustration assume miss frequency is linear in memory size. Using the data of Figure 7-11, a {k=4,m=64} system operating at $f_{miss} = 0.04$ can become {k=16,m=256} system at $f_{miss} = 0.02$ and gain 30% improvement.

Less Predictable Access Patterns

We would like to determine the benefit of background loading compared to the worst-case penalty of 40 cycles. The key parameters are the latency of a reference, and a distribution probability on consecutive non-memory references. Our more pessimistic first order model assumes a uniform probability p of an off-chip memory reference. Thus, the probability of a sequence of k on-chip references before an off-chip reference is:

$$P(N = k) = (1 - p)^{k-1} p \tag{7.16}$$

This distribution is shown in Figure 7-12. Notice that the probability of long sequences is small if p is high. Therefore, we expect background loading to help only in the case of low p or low latency. Since the loading time T given a string of k available cycles is $L - k$, the expected value of T is:

$$E[T] = \sum_{k=1}^{L} q^{k-1} p(L - k) = \frac{(1 - p)^L + pL - 1}{p} \tag{7.17}$$

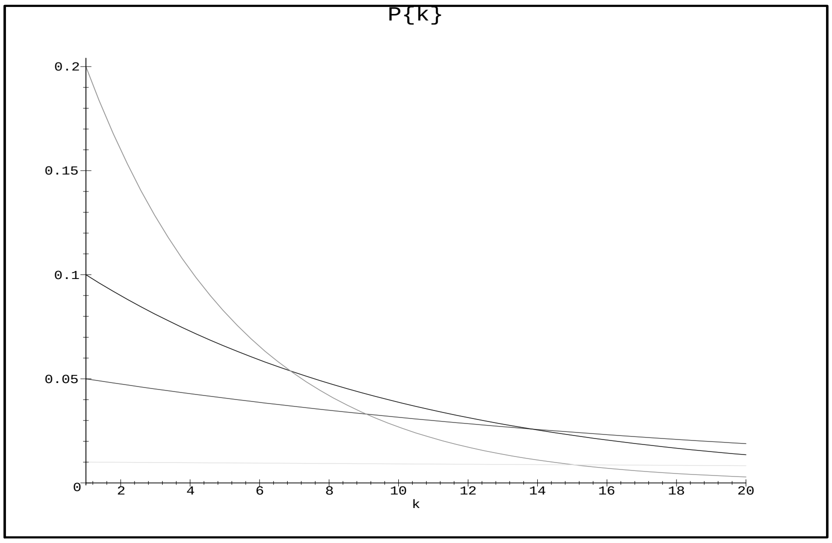

Figure 7-12: Probability of an on-chip hit sequence of length k for different values of p, the miss probability. Note that for high p almost all the area under the curve is near small values of k, and therefore long load times.

7.1.5 Conclusions

In the sense of raw performance, the design is optimal when the off-chip memory pathway is constantly busy with a minimal number of PE stalls. At that design point, the internal PE area has been pared down to provide as much on-chip processing power as possible. However, this mode of operation is probably not optimal under the power consumption metric, since large off-chip buffers are constantly active. This multivalued optimization problem is an interesting avenue of future research.

7.2 What's Missing

What this section really needs is to determine how performance on an algorithm varies as a function of the size on-chip register file, and the off-chip memory latency. The limiting cases are obvious. When the entire computation fits into the register file, the off-chip latency becomes irrelevant. When latency is very low, there is no need for a register file at all. If we plot performance as a function of register file size s and off-chip memory latency l, we get a slope falling off to the northwest, where both latency is high and on-chip capacity is low.

So, for an given acceptable level of performance, we can travel the curve, trading off capacity for latency. To convert this expected graph into a design optimization tool, we need to assign resource costs and physical limitations, based on current technology. This cost assignment

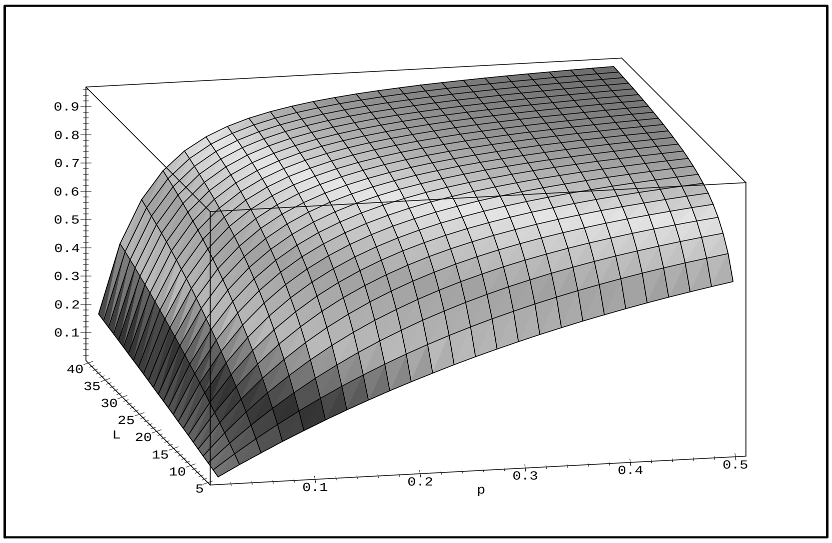

Figure 7-13: Effective latency vs latency due to background loading as a function of L and p. The advantage of background loading occurs in a relatively narrow zone: where the frequency of loads is very low to begin with, and where latency is low. For low values of L, the ratio may not be really important, since it approaches internal access time.

is a challenging task, since it incorporates a good number of technology assumptions.

Chapter 8

Abacus 2: The Next Generation

After designing the Abacus-1 chip, a number of design flaws and potential improvements became apparent. In a sense, this discussion belongs in a conclusions chapter, but it is a prerequisite for the design of the next generation.

The goal of the design changed, from building a supercomputer-class machine to a co-processor device for a workstation. Most significantly, having shown that high speed design is possible, power dissipation emerged as the main bottleneck.

This chapter incorporates the lessons of Abacus-1 into the design of the next generation machine, Abacus-2. Although presented in the context of a low-power, low-cost coprocessor, the design can be converted back to a supercomputer-style system by returning to large, fast chips housed in specialized packages and cooled by carefully designed fan systems.

The implications of a co-processor style design are:

- Low power.
- Low cost packaging.
- High virtualization factors.
- More off-chip memory per PE chip.
- No specialized IO boards.

The design retains constant performance, while reducing the chip area, power dissipation, and packaging cost. The Abacus-2 chip fits 64 8-bit PEs on a single 8 $\times$ 8 mm die, and operates at 66 MHz in a 3.3 V (or lower) environment.

A single Abacus-2 chip delivers 4.2 billion 16-bit additions per second and approximately 180 million 8-bit multiplies per second, in a non-aggressive 1 micron VLSI technology. Implementation in a current sub-micron technology would lead to higher performance numbers.

The new design is amenable to integration with a simple sequencer and I/O subsystem, allowing scalability in system size from a 16-chip 64 GigaOps system that can be utilized as a co-processor attached to a conventional RISC-chip based node of a MIMD system, to a standalone 256-chip TeraOp SIMD machine. The Abacus-2 chip is designed to operate as smart memory, accessible by the controller with low latency, and without intervening corner-turning circuitry.

8.1 Lessons of Abacus-1

This section presents some lessons learned from the process of implementing the Abacus-1 chip.

Reconfiguration overhead is excessive. The design of the on-chip interconnection network requires a reconfiguration cycle whenever direction of data flow changes. This occurs when, for example, an ASR (arithmetic shift right) instruction, in which bits flow from left to right is followed by an ADD instruction, in which the carry bit flows from right to left. Although the cost is only one cycle, many operations require only three or four cycles, meaning reconfiguration costs approximately 20% of performance.

This problem was not obvious at first because initial program coding was done manually, and an experienced programmer could reorganize the code to group together instructions of a given orientation, reducing reconfiguration overhead by approximately factor of two. Only when the compiler became operational and performance dropped unexpectedly, did the flaw become obvious. Due to time constraints, there was no opportunity to investigate compilation strategies for automatic reorganization. Instead, a peephole optimization step was added that noted when two successive instructions required identical configuration, and used a non-reconfiguring version of the microcode for the second instruction. Neither software technique helps in the very common case of rapidly alternating mesh moves and arithmetic operations.

Arbitrary data formats are expensive. The one-bit bit slice elements could be grouped in almost arbitrary configurations, as long as each set of PEs responsible for a data word formed a topological circle. This choice required the overhead of network and configuration circuitry at every bit of a datapath. Additionally, on-chip data could not be accessed by the sequencer without a complex corner-turning reformatting process that requires dedicated hardware.

In retrospect, this flexibility is mostly useless. The constraints on cluster organization are that they must be tile-able, which essentially means rectangular, and both dimensions must be a factor of 32 to fit evenly onto a chip. These constraints allow a limited number of configurations, so the bit slices may as well have been grouped as multi-bit units.

Rise/fall times are a substantial fraction of cycle time . When rise/fall times approach 500 picoseconds, three or four non-overlapping control signal transitions consume as much as 3 ns. When combined with safety margins for process variation uncertainties, the actual active time shrinks further. For example, an address wire must transition four times during a cycle: one each for the read and write phases, and once for the precharge phases between the read and write. The 1.6 ns of rise/fall time is fully 20% of the cycle time. This penalty can be eliminated by reducing the number of timing signals, using only the initial edge of a signal to initiate activity, or lengthening cycle time through the use of pipelining.

Simple ALUs do insufficient work per cycle. The very simple ALU operates in approximately 400 picoseconds, a tiny fraction of the overall cycle time. The common battlecry of "keeping the silicon busy" does not completely apply when computation actually occurs during 5% of the cycle time. This is a central weakness of conventional bit-serial SIMD systems: they spend far more time fetching and storing data than actually operating on it. The implication of this analysis is that the main constraint on ALU complexity is area rather than computation time. This issue is discussed in greater depth in Chapter 7.

Pipelining is important. The Abacus-1 design could not be effectively pipelined since much of its bit steering depended on network reconfiguration, which was manipulated by writing control registers. Forwarding could not be used because the arriving bit was a function of (as of yet unwritten) control registers in many other PEs. As a result, the very common network operations would have introduced pipeline bubbles.

Without pipelining, a single cycle had to encompass the read, write, and execute phases. Cycle times thus stretched considerably longer than necessary.

Long instruction words are expensive. Although the double-banked register files and twin ALUs allow two result bits to be written per cycle, the cost of this flexibility is higher than first expected. As discussed in Chapter 4, the PE area impact is not significant. However, the required board-level instruction bandwidth is almost doubled. An intermediate possibility is to use double-banked register files but generate addresses from the instruction code. For example, the carry and sum bits could be written to identical addresses in the two banks.

ECL components lead to low integration. ECL signalling was chosen for instruction distribution and I/O paths before the system design was complete. The high parts count of SSI ECL components did not become apparent until after the chip was designed. So although the interface to the external memory chips is glueless, each PE board has two SSI chips per PE chip. In addition sheer number of chips, ECL designs are complicated because they use different voltage levels, discrete terminators and bypass capacitors, and a more complex circuit board structure.

As pointed out in Section 5.3, the discrete circuitry could be eliminated with two custom ASICs. But the question arises: if a special chip is required, why not use CMOS signaling to begin with? For example, at lower frequencies, low-swing CMOS signaling consumes significantly less power than that dissipated in ECL terminators alone.

Burst I/O is overconstraining. The Abacus chip allocates three pins for data I/O. A single I/O instruction initiates a 32-bit burst at 125 MHz. This design choice was intended to both increase the overlap of computation and I/O by requiring only a single instruction per 32 bits, and to reduce the number of chip pins.

The design places heavy demands on system-level IO reformatting circuitry. It must capture 32 256-bit words arriving at 8 ns intervals. High performance parts are required to handle this quantity of data, yet they sit idle almost all of the time, as I/O transfers are very rare.

Network model must be maintained transparently between chips. Abacus-1 implemented a reconfigurable mesh network. PEs could be linked together by almost arbitrary topologies, and operate as a multiple-writer bus. This is a powerful machine model, and is the subject of considerable algorithmic research. Unfortunately, the multiple-writer model breaks across chip boundaries, and therefore cannot be used for system-wide algorithms. Of course, reconfigurable mesh algorithms can be emulated on Abacus, but their theoretical performance levels will not be achieved.

8.2 Abacus-2

In order to see how each of these lessons was applied to the Abacus-2 architecture, this section gives a quick overview of the design. An Abacus-2 processing element consists of an 8-bit ALU, 512 bits of single-banked memory, a carry bit interconnection network and a mesh interconnection network. The carry network provides two unidirectional links running in opposite directions. The mesh interconnection network allows PEs to select data from each of their four neighbors.

8.2.1 Pipelining Analysis

Processor performance is frequently improved through pipelining the various phases of the clock cycle. Pipelining involves adding registers between circuit stages to allow parts of different instructions to execute at the same time. As long as the pipeline is kept full, the throughput rate of a pipelined processor increases by a factor equal to the number of stages. Most modern microprocessors have between 5 and 15 pipeline stages.

Pipeline performance degrades when data or control hazards occur. For example, if a register is written in one instruction and referenced in the next, the computed data item has not yet been written to the register file and therefore cannot be read. The pipeline must be stalled until the data arrives. Alternatively, a forwarding path can be added between the ALU input and the write stage pipeline register, so the data can be directly shunted without a pipeline bubble. This forwarding circuitry and pipeline registers are the main area costs of a pipelined implementation. In conventional processors, the area cost is negligible compared to the four-fold performance gain. This advantage is not quite as clear for SIMD systems, whose entire datapath is no bigger than two or three registers.

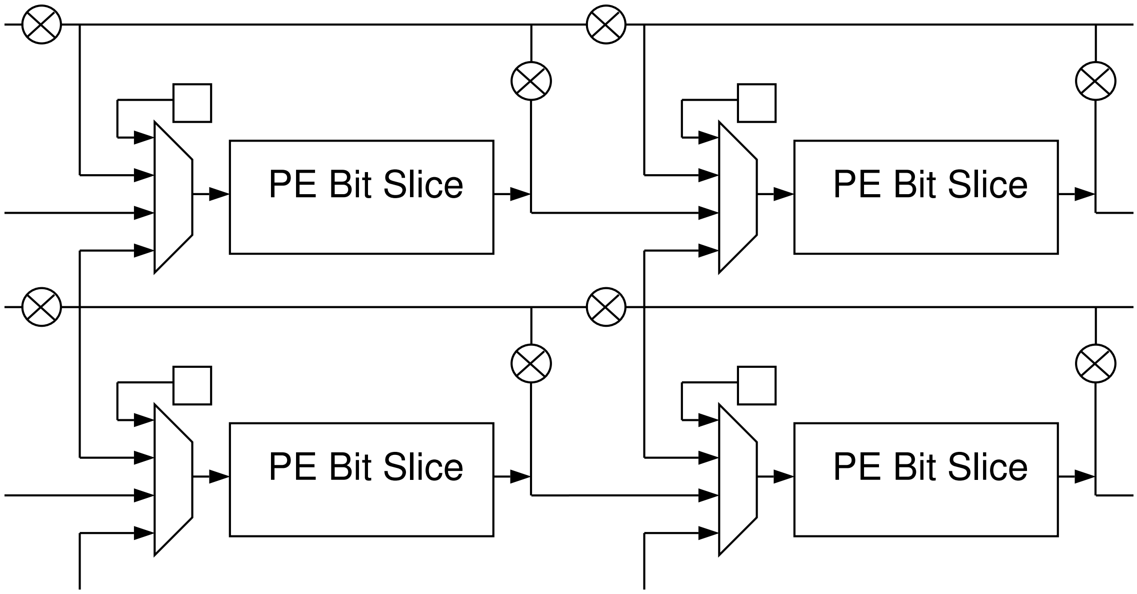

Figure 8-1: Simplified view of the Abacus-2 carry interconnection. The wiring and switch requirements are approximately four times that shown in the image: double to complete the ring topology and double again for the MSB to LSB direction.

SIMD systems encounter three other hazards that decrease the effectiveness and increase the cost of pipelines. First, communications operations occur frequently in image processing code. Every such operation introduces a stall in the pipeline, since network data, unlike internal data, cannot be forwarded.

Second, the value of the *active* register that controls conditional execution must be forwarded to the register file writeback controller, as well as the global bit that disables the writeback control so that the *active* register may be written. An alternative to expending area on forwarding circuitry is to stall the pipeline in software.

Third, unlike conventional pipelining, when the forwarding logic detects the read of a register that is still being written, it cannot simply forward the write stage inputs to the ALU. The problem is that the data on the write stage inputs may not be actually written to the register file (based on the *active* bit), and it would therefore an error to forward that data. The ALU actually needs the value stored in the register file. Thus, writing a value and reading it out immediately causes a stall, unless the compiler is certain that the active bit is off.

The benefit of pipelining in a system already operating at the top clock rate allowed by the instruction delivery system is not expressed in performance gains. Instead, power can be reduced by lowering the voltage and therefore slowing operation back to the unpipelined rate. Since power dissipation is the next barrier to high performance, this tradeoff appears worthwhile.

A standard three-stage pipeline consists of read, execute, and write phases. Two pipeline registers and a three-to-one multiplexer are required for each phase per data port. Assuming dynamic circuits, the area cost is estimated as approximately 30 inverters per datapath bit. This is approximately 50% of the existing non-memory circuitry.

A more significant area cost is the growth of the memory cell due to the addition of an extra write port. A typical area ratio between a two and a three-port cell is approximately 1.5. The combined ALU and memory-caused area increase is thus also 1.5.

The increased throughput thus comes at an area cost of approximately 60%, and an additional latency of two gate delays in each cycle. Assuming the clock rate can be increased (or voltage lowered), the overall performance improvement is a factor of approximately 1.8.

8.2.2 Improvements To Abacus-1

Reduced Reconfiguration. Reconfiguration overhead has been almost completely eliminated in this design by two major changes. First, reconfiguration within arithmetic operations has been reduced by providing two wires for bidirectional signaling between bit slices so that computations flowing in opposing directions (such as an add followed by a shift right) do not require reconfigurations.

Second, reconfiguration has been eliminated for inter-pixel operations by separating the roles of the carry chain and the mesh communication. The design recognizes that for mesh operations, all pixels usually communicate in SIMD mode, and do not use the "current neighbor" facility of network configuration. Therefore, the direction of mesh inputs is selected globally for all PEs.

This redesign allows reconfiguration to be eliminated completely except for changes in the word unit (WU) size. As a result, the configuration register write time can be very slow. The Abacus-2 design does not take advantage of this optimization opportunity.

Fixed Data Formats. The word format ordering has been fixed so that bits are oriented horizontally, with the LSB on the right side. This addresses the issues of constant distribution and corner turning. The implementation allows simple, efficient constant generation by the sequencer with only a few byte shift instructions. Similarly, the four typical word formats (8, 16, and 32) can be easily manipulated even in a bit-serial stream. The disadvantage of this organization is that bits are no longer configured in a nearest-neighbor head-to-tail snake pattern, and therefore propagation times between slices are longer.

Rise/Fall Time Issues. These problems have decreased in importance with the lengthening of the read and write times due to pipelining. The main effect is due to the reduced number of clock signals relevant to each pipeline stage. Thus, the cycle time of the read circuitry is not affected by how many control signals are required by the ALU.

CMOS Replacement for ECL. The instruction delivery interface is redesigned to use CMOS signalling. A custom CMOS chip provides instruction distribution to two Abacus chip columns.

Wider I/O Port. Burst I/O has been eliminated in favor of an 8-bit dedicated output port. Each column of PE chips shares an I/O bus. Each chip is given a column position identifier, and IO requests specify that position. For example, an IO instruction specifies that byte 3 of chip 1 be placed on the output bus.

An idea to be explored is the use of the wide instruction word to transmit data back to the instruction distribution chip (IDC). In this approach, an I/O instruction causes the IDC to stop driving the instruction bus, and the processor chip in a particular column to drive the contents of a PE row onto the instruction bus, while executing a NOP internally. Although peak computing power is slightly reduced, the use of a wide, 32-bit bus increases I/O bandwidth to 1 transfer per 2 cycles per column, or 4 bits per chip.

Alternate Active Bit Abacus-1 supported the use of an arbitrary bit in memory as a steering bit in a multiplexing operation. A useful application of this capability is to conditionally update a value in memory by selecting between the old value and the new one. This technique reduces operations on the active register and therefore memory traffic. Since Abacus-2 does not have three read ports, the multiplexing trick cannot be used. Instead, a temporary active register is conditionally ANDed with the primary active register based on a global signal.

External Memory Interface The external memory interface is ideally suited to a high throughput, block-transfer based memory technology such as Rambus, but that technology is not yet easily available. The design therefore resorts to off-chip SRAM instead of DRAM. This approach reduces the memory capacity but eliminates the need to send carefully-timed DRAM control signals through the instruction stream and to design lockout circuitry to handle refresh timing.

Several suitable memory chips are available, including the 32K x 32 Micron 10 ns synchronous SRAM. With this IC, memory capacity can be expanded at the finer granularity of 1 Kword per processing site. More address pins are required than for the DRAM, since the row and column addresses are no longer multiplexed. External chip select pins provide for future memory expansion.

Network Model One of the main drawbacks of the reconfigurable mesh model supported by the Abacus-1 design is its inefficient support of images larger than the physical array. Although signals propagate at electrical speeds, they must be re-registered and digitally updated at each iteration. The problem is that signals may propagate in arbitrary directions, so that virtualization reduces to cycling through all array tiles at every iteration.

Abacus-2 uses the theoretically weaker but practically more efficient model of a reconfigurable bus. In this model, each PE can select whether to output its own value or that of its downstream neighbor onto the bus, as shown in Figure **??**.

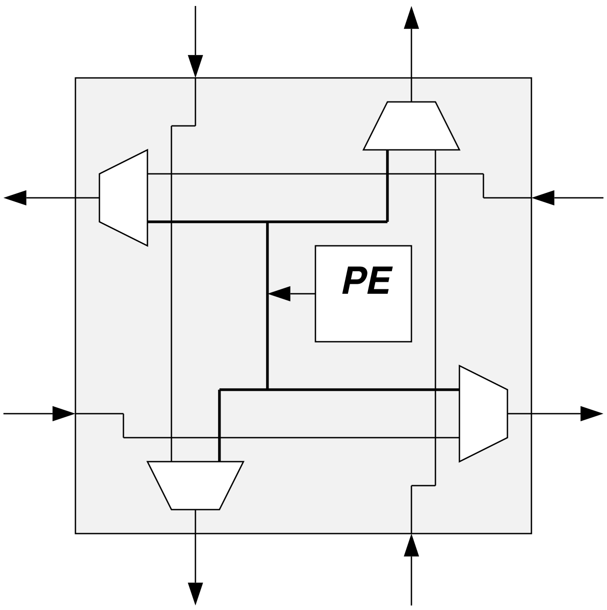

Figure 8-2: Abacus-2 Mesh Network

Speed estimation is straightforward. If bypass is implemented with a buffered multiplexer, propagation past one PE requires approximately 0.6 ns. An 8-PE row presents a delay of only 4.8 ns. I/O pad delay is estimated at 3 ns, for a total chip delay of approximately 8 ns. The time-multiplexed mesh pins are clocked so that propagation cannot be flowthrough, and therefore incurs quantization delays. On the other hand, the clocking allows signals to be pipelined.

Virtualization is easily handled by using a tile strategy instead of a neighborhood strategy, as described earlier. Propagation occurs at electrical speeds across a tile. The data is then registered under software control, and the physical array simulates the next tile. Since reconfigurable busses operate in one dimension at a time, and in one direction at a time, this approach is very efficient.

Multiplies The Abacus-2 design can perform multiplications at the rate of three cycles per multiply step. Thus, an 8-bit multiply requires 24 cycles and produces a 16-bit result. The chip-wide computing rate is 64 multiplies every 24 cycles, or 2.75 multiplies per cycle. At a 66 MHz clock rate, this translates into 183 MOPS.

If multiplies become more important in applications, multiplication can be substantially improved with dedicated hardware. All of these approaches are more effective with a wider slice. Approaches include:

- A dedicated shift register and Booth recoder to halve the number of iterations, and reduce the iteration time from 3 cycles to two. Estimated computation time is 4 iterations of 2 cycles each, for 8 cycles plus 2 overhead cycles. Estimated performance is therefore 10 cycles, or 6.4 multiplies per cycle, leading to 422 MOPS at 66 MHz.
- A self-timed iterative multiplier with a Booth recoder. Estimated performance is dependent on cycle time, but a single iteration of a 16-bit add should take about five nanoseconds. This requires four iterations, or two 15 ns cycles plus a carry computing cycle at the end. At 21 multiplies per cycle, the aggregate performance is 1.4 GOPS.

The dedicated circuitry will cost chip area, but the additional circuitry is at most four times larger than the simple 8-bit ALU, which is itself only approximately 20% of the PE area. Thus, the core area will approximately double, increasing the chip area by 50%.

Pass Gate Accelerators Like the Abacus-1 design, Abacus-2 incorporates accelerators to reduce the quadratic delay growth with the number of pass switches. Unlike the noise-intolerant NORA-based accelerator described in Chapter 4, the new design uses a slightly slower but more robust static circuit [46].

Global OR Unlike the Abacus-1 design, PEs can no longer be joined into one large electrically connected net. Specialized global OR circuitry is therefore provided. The precharged carry circuitry is optionally connected to a dedicated column wire. The eight columns are reduced to a single bit by a wide conditional pseudo-NMOS NOR gate.

Memory Organization The Abacus-2 memory is organized as 32 rows by 16 columns. This organization requires only four column decoder wires, for a total of 20. Since the memory organization dictates PE organization, the PE aspect ratio will be much more square than the Abacus-1. Simulations show that the bitline discharge time is approximately 60% longer for the 16-cell bitline than for the 8-cell bitline, but this interval is a small fraction of the overall read cycle.

8.2.3 Physical Characteristics

Pin Budget. The chip package is an important aspect of overall design performance. To maintain low cost, the Abacus-2 is housed in a 240-pin plastic quad flat pack (PQFP). The chip contains 64 8-bit PEs in an 8x8 arrangement. Each side has 16 pins for mesh communication, sending data on both clock edges, or 32 bits per cycle.

Abacus-2 PE Area Estimation Sixty-four of these processors occupy approximately 30 mm^2, or about 75% of the area of the Abacus-1 array.

Type	Quantity
Instruction	20
Mesh	76
External memory address	15
External memory data	32
External memory control	4
IO	8
TAP	5
Misc	5
Signal total	165
Power/ground	75

Table 8.1: Abacus 2 Pin Budget

Circuit	Area (μ^2)	Qty	Total (Kμ^2)
SRAM Cell	400	512	205
PG, sum, mux	6400	8	50
Pipeline	4000	8	32
Mesh Network	4300	8	35
Carry net,mux	3000	8	24
Aux Logic	10000	8	80
Total			425

Table 8.2: Abacus-2 PE Area

Cycle Time Estimation The limiting factor on the Abacus-2 ALU cycle is the propagation path of the Manchester carry chain across 12 internal switches, 2 inter-slice switches, and 2 multiplexers (for a 16-bit add), followed by the sum calculation. This should take approximately 8 ns.

Power Estimation Current power consumption at 100 MHz is 3 A at 5 V, or 15 W. Halving the amount of circuitry on chip reduces the current draw to 1.5 amps. Pipelining while halving the cycle time leads to a voltage reduction of a factor of almost four, down to 1.5 V. The new estimated power cost is 3.5 W. With lower voltage swings on the external memory drivers, the power can be reduced to 2.5 W.

8.3 Beyond SIMD: A Family of Abaci

Once a high performance SIMD core is developed, a number of interesting architectural ideas present themselves.

Dimensions	8 mm $\times$ 7 mm
PEs	64
Memory	32 Kb (512 bits/PE)
Pins	240
Technology	1 um CMOS
Voltage	2.5 V
Clock Rate	66 MHz
8-bit GOPS	4.2
16-bit GOPS	2.1

Table 8.3: Abacus-2 Chip Spec

Abacus-M: Multi-SIMD Operation. One way of viewing the Abacus-2 chip is as the integration of a large number of traditional discrete bit-slices (such as the AMD2903) and a flexible interconnection network on a single chip. If a bitslice family sequencer was also integrated, the chip could operate independently as a uni-processor, ignoring the broadcast SIMD instruction and treating the on-chip memory as registers and the off-chip memory as instruction and data store. This is essentially a superset of the recently common idea of integrating a RISC-style controller and with a SIMD array on chip.

The key difference is that several Abacus-M chips can work together as a larger SIMD system. This mode of operation entails dynamically selecting one of the chips as a master and then having all slaves listen on the instruction bus while the master drives the bus. The collection of Abacus-M chips could now operate in a mixed MIMD/SIMD mode, as required by the algorithm. This mode would not be as efficient as pure SIMD, but algorithms more suitable to MIMD operation could be implemented efficiently.

Abacus-S: Systolic Operation. A small augmentation of the Abacus-2 PE would allow operation in a systolic mode, where each PE executes a different instruction. The only required hardware is a locally controlled latch on every control line. Every PE could be now programmed to perform a different operation. For example, directing some PEs to multiply their network input by an internal constant and place the result on the network output, while others summed their network inputs, results in a pipelined FIR filter. A more powerful interconnection network is also necessary for effective operation in this mode.

Abacus-F: FPGA Emulation. The addition of local address decoding to an Abacus-S PE allows emulation of an SRAM-based FPGA cell. A richer interconnection network would again be required, but much of the network traffic could be time-multiplexed by a software-based routing scheme.

Abacus-U: Universal Computing Element. The combination of all three features described above leads to an almost universal computing element, able to operate as a SIMD node, as a uniprocessor with a set of very wide vector registers, a collection of systolic computing elements, or as programmable logic. Furthermore, different parts of the chip could be operating in different modes. For instance, some PEs could be serving as logic gates configured as specialized find-first-one hardware co-processors for other SIMD-mode elements.

Chapter 9

Discussion

Now that the design has been discussed and the analysis is complete, it is time to step back and examine what was learned in the process, how the newly found knowledge may be applied, and future research based on this work. [1]

9.1 Chip Design

The chip design process has served a number of purposes. Theoretical analysis or simulations would have been insufficient to conclusively demonstrate the soundness of the design. The act of fabricating an artifact and measuring consumed area and execution speed produced a number of important lessons.

Abacus has demonstrated that high clock rates are achievable with simple circuits, even in a slow process technology. Although this demonstration should not be surprising, since near-gigahertz level bit-serial adders have been fabricated in comparable technologies, the SIMD architecture community has not internalized this lesson. The Abacus design demonstrates the performance gains achievable in SIMD systems through the use of high-speed VLSI techniques and raises the bar for future architectural development.

Abacus has provided a framework for future chips design. Having accounted for all the major system issues dealing with high-speed SIMD design, Abacus can serve as a checklist for designers. The analysis of memory size and ALU complexity provides designers with a design tool more quantitative than experience and intuition for architecture development. And this design has exposed a number of "fallacies and pitfalls," such as "ALUs should always be small and fast" and "PEs should have as much memory as possible." [2]

The experience gained through the Abacus design also suggests that the developer design a processing element and immediately develop the instruction and data distribution network *before* performing circuit optimizations, since the global structure will affect the PE

[1] So sit back, light up a stogie, and read on.

[2] These may not be the best examples, but I'm sure you can think of better ones.

in potentially unanticipated ways. Identifying possible optimizations using software compensation of layout quirks and removal of abstraction boundaries for layout improvement are also possible only when the global PE structure is well defined.

Some of the lessons learned are apparently the same as those discovered by the mainstream architecture community. [3] The instruction distribution and sequencer limitations are similar to the microcoded CISC processors' speed limitations. The current Abacus design is comparable to a first or second generation RISC design. As clock speeds rise due to architectural innovation and improvements in circuit design, power dissipation becomes a significant issue. The transition to this power-sensitive regime occurs more quickly for a SIMD machine since unlike a uniprocessor, many power-hungry ICs share a board and must be cooled. The next Abacus design must therefore pay more attention to power dissipation.

Fortunately, much of the power in a SIMD system such as Abacus is dissipated in very regular, highly capacitive structures such as instruction wires. These structures are suitable for novel power saving techniques such as partial energy recovery.

9.2 System Design

9.3 Algorithmic Analysis

The application of the set of DARPA IU and other benchmarks to Abacus has elucidated the difference between virtual emulation of conditional algorithms. Algorithms that use nested parallel *where* clauses require relatively expensive storage of the *active* bit between context switches. This costs both time and memory capacity.

The analysis has underscored the need for understanding memory requirements as well as time requirements of algorithms. Given the enormous difference in access times between on-chip and off-chip memory, algorithms with theoretically poorer performance but a smaller working set can outperform supposedly efficient but memory-intensive algorithms. A true algorithmic figure of merit must include assumptions about the underlying machine model.

9.4 Architectural Analysis

A number of research projects concern implementing DRAM-based SIMD devices [47]. Surprisingly, these designs still use bit-serial processors. A section FOO has already pointed out the disparity between memory access time and the amount of time spent computing by a one-bit adder. The disparity is worse by an order of magnitude when DRAM access times reach 50 ns. Having paid an extraordinary amount of time for the access, the ALU should extract as much computation as possible from the data before writing it back. In general, a good rule of thumb is that the slower the memory, the more complex the ALU.

[3] Which is interesting considering how far outside the mainstream we are.

In another example, the MGAP-2 design [48] relies on a redundant representation to avoid carries. Yet at the targeted clock rate of 50 MHz in a 1 micron VLSI technology, the carry can easily ripple through 16 bits.

The inverse effect can be seen if fast DRAM cells are developed. An early design for the Abacus memory was based on dense, fast, DRAM cells. If this approach is adopted, what is the effect on the optimal datapath width? The ALU area is increased relative to the memory, and therefore assigns a higher penalty to wider datapaths.

Analysis of the Abacus architecture has led to the insight that area-efficient memory systems require at least four edge times to execute a memory access. ALU delays must be at least as long as that time interval. Since this time is already greater than that required for a four-bit ripple carry adder, and edge times will increase with improved VLSI technology (and therefore smaller feature sizes), we may proclaim bit-serial designs dead, at least for memory-based PEs.

The passing of bit-serial processors appears even more inevitable when pipelined processing is considered. Since pipeline stage delays are (ideally) equal, the execute part of the cycle does almost no work for the same reason as in the unpipelined case. Even worse, the overhead of pipelining registers and forwarding logic comes with no performance gain and therefore decreases performance for narrow-width datapaths.

Another obvious observation from the first-order model is that low-memory PEs deliver higher performance (for applications that fit in memory). In the limit, processors with no memory offer the very highest performance, which explains why FPGAs perform so well: if the algorithm can be converted into a systolic form with little storage per processing node, then all of the silicon is busy computing. If the algorithm run on a low-memory chip requires many off-chip memory references, system performance becomes equal to memory access speed.

9.5 Observations

SIMD processors are beginning to enter into the everyday computing realm under the guise of multimedia processors. For example, Chromatic's Mpact chip contains 8 9-bit processors operating in SIMD mode.

9.6 Parting Remarks

First, the Abacus project 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, other MPP SIMD systems use a clock rate substantially slower than that of commercial bit-parallel microprocessors. The 125 MHz Abacus chip clock is the highest of any of the massively parallel SIMD systems in the literature. 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. Maintaining clock speed while increasing the work done per cycle holds more promise as the approach for incorporating smaller and faster VLSI technology.

Second, analysis 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. The next multi-bit bit-slice design effort intends to address these issues.

Third, completion of an entire system brought to light 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. Abacus now has a system framework that allows modification of the PE core while retaining plug compatiblity with the rest of the machine. Since the clock rate is already aggressive, it may be held steady for a number of years while researching how to complete more work per cycle.

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.

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, through this dissertation, has shown that for the problem domain of early vision, parallel computers based on a SIMD reconfigurable bit-slice architecture can outperform those based on conventional processors by over an order of magnitude using the same silicon resources.

Bibliography

[1] J. L. Potter, ed., *The Massively Parallel Processor.* MIT Press Series in Scientific Computation, Cambridge, Massachusetts: The MIT Press, 1985.

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

[3] D. Parkinson and J. Litt, eds., *Massively Parallel Computing with the DAP.* MIT Press Research Monographs in Parallel and Distributed Computing, Cambridge, Massachusetts: The MIT Press, 1990.

[4] T. Blank, "The maspar mp-1 architecture," in *International Conference on Computer Architecture*, 1990.

[5] R. Tuck and W. Kim, "MasPar MP-2 PE Chip: A Totally Cool Hot Chip," in *IEEE 1993 Hot Chips Symposium*, March 1993.

[6] H. Li and M. Maresca, "Polymorphic-torus architecture for computer vision," *IEEE Trans. Pattern Analysis and Machine Intellegence*, vol. 11, no. 3, pp. 233–243, 1989.

[7] 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.

[8] D. B. Shu, G. Nash, and C. Weems, *Image Understanding Architecture and Applications*, ch. 9, pp. 297–355. Springer-Verlag, 1989.

[9] N. Thacker, P. Courtney, S. Walker, S. Evans, and R. Yates, "Specification and Design of a General Purpose Image Processing Chip," in *International Conference on Pattern Recognition*, pp. 268–273, IEEE, 1994.

[10] M. Gokhale, B. Holmes, and K. Iobste, "Processing in Memory: The Terasys Massively Parallel PIM Array," *IEEE Computer*, pp. 23–31, April 1995.

[11] P. Kogge, T. Sunaga, H. Miyataka, K. Kitamura, and E. Retter, "Combined DRAM and Logic Chip for Massively Parallel Systems," in *Proceedings of the 16th Conference on Advanced Research in VLSI*, pp. 4–16, IEEE Computer Society Press, March 1995.

[12] A. Yeung and J. Rabaey, "A 2.4 GOPS Data-Driven Reconfigurable Multiprocessor IC for DSP," in *ISSCC Digest of Technical Papers*, pp. 108–110, IEEE, February 1995.

[13] Patrice Bertin and Didier Roncin and Jean Vuillemin, "Programmable active memories:

A performance assessment," in *Research on Integrated Systems Symposium*, 1993.

[14] T. E. Rockoff, "An analysis of instruction-cached simd computer architecture," Tech. Rep. CMU-CS-93-218, Computer Science Division, Carnegie Mellon University, 1993.

[15] J. D. Allen and D. E. Schimmel, "Issues in the design of high performance simd architectures," *IEEE Trans. on Parallel and Distributed Systems*, vol. To Appear, 1995.

[16] C. Weems, "The Next Generation Image Understanding Architecture," in *DARPA Image Understanding Workshop*, 1994.

[17] V. L. Chi, "Salphasic Distribution of Clock Signals for Synchronous Systems," *IEEE Trans. on Computers*, vol. 39, May 1994.

[18] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, "The ILLIAC IV Computer," *IEEE Trans. on Computers*, pp. 746–757, May 1968.

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

[20] R. A. Barman, M. Bolotski, D. Camporese, and J. J. Little, "Silt: A bit-parallel approach," in *International Conference on Pattern Recognition*, IEEE, 1990.

[21] T. J. Holman and L. Snyder, "Architectural Tradeoffs in Parallel Computer Design," in *Proceedings of the 10th Conference on Advanced Research in VLSI*, pp. 317–334, IEEE Computer Society Press, March 1989.

[22] S. Ho and L. Snyder, "Balance in Architectural Design," in *Proceedings of the 17th Annual International Symposium on Computer Architecture*, IEEE Computer Society Press, 1990.

[23] M. C. Herbordt, *The Evaluation of Massively Parallel Array Architectures.* PhD thesis, University of Massachusetts at Amherst, 1994.

[24] D. Audet, Y. Savaria, and J.-L. Houle, "Performance improvements to vlsi parallel systems, using dynamic concatenation of processing resources," *Parallel Computing*, vol. 18, pp. 149–167, 1992.

[25] W. B. Ligon and U. Ramachandran, "Evaluating Multigauge Architectures for Computer Vision," *Journal of Parallel and Distributed Computing*, vol. 21, pp. 323–333, 1994.

[26] M. J. Irwin and R. M. Owens, "A two-dimensional, distributed logic architecture," *IEEE Trans. on Computers*, vol. 40, October 1991.

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

[28] "Synergy semiconductor product data book," 1995.

[29] A. DeHon, J. Thomas F. Knight, and T. Simon, "Automatic impedance control," in *International Solid State Circuits Conference*, IEEE, 1993.

[30] M. G. Johnson, "A variable delay line pll for cpu-coprocessor synchronization," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 1218–1223, October 1988.

[31] N. Weste and K. Eshragian, *Principles of CMOS VLSI Design*. Addison-Wesley VLSI Systems Series, Reading, Massachusetts: Addison-Wesley, 1993.

[32] H. W. Johnson and M. Graham, *High-Speed Digital Design: A Handbook of Black Magic*. Prentice-Hall, 1993.

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

[34] J. G. Harris, "The coupled depth/slope approach to surface reconstruction," Master's thesis, MIT, May 1986.

[35] James J. Little and Heinrich H. Bülthoff, "Parallel optical flow using local voting," AI Memo 929, MIT, July 1988.

[36] S. Levialdi, "On Shrinking Binary Picture Patterns," *Communication of the ACM*, vol. 15, Jan 1972.

[37] F. Thomson Leighton, *Introduction to Parallel Algorithms And Architectures: Arrays, Trees, Hypercubes*. Morgan-Kaufmann, 1992.

[38] Sotirious G. Ziavras, "Connected Component Labelling on the BLITZEN Massively Parallel Processor," *Image and Vision Computing*, vol. 11, Dec 1993.

[39] Alok Choudhary and Janak H. Patel, *Parallel Architectures and Parallel Algorithms for Integrated Vision Systems*, pp. 69–78. Kluwer Academic Publishers, 1990.

[40] R. E. Cypher and Jorge Sanz and Larry Snyder, "Algorithms for Image Component Labelling on SIMD Mesh-Connected Computers," *IEEE Trans. on Computers*, vol. 39, Feb 1990.

[41] R. E. Cypher and Jorge Sanz, "The Hough Transform Has O(N) Complexity on SIMD MxN Mesh Array Architectures," *IEEE Trans. on Computers*, vol. 39, Feb 1090.

[42] M. J. Quinn, *Designing Efficient Algorithms for Parallel Computers*. McGraw-Hill, 1987.

[43] C. P. Schnorr and A. Shamir, "An Optimal Sorting Algorithm for Mesh Connected Computers," in *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, ACM, 1986.

[44] Martin C. Herbordt and James C. Corbett and Charles C. Weems and John Spalding, "Practical Algorithms for Online Routing on Fixed and Reconfigurable Meshes," *Journal of Parallel and Distributed Computing*, vol. 20, pp. 341–356, 1994.

[45] T. I. Chappell, B. A. Chappell, S. Schuster, J. W. Allen, S. P. Klepner, R. V. Joshi, and R. L. Franch, "A 2-ns cycle, 3.8-ns access 512-kb cmos ecl sram with a fully pipelined architecture," *IEEE Journal of Solid-State Circuits*, vol. 26, no. 11, pp. 1577–1584, 1991.

[46] I. Dobbelaere, M. Horowitz, and A. E. Gamal, "Regenerative feedback repeaters for programmable interconnections," in *International Solid State Circuits Conference*, IEEE, 1995.

[47] J. C. Gealow, F. P. Herrmann, L. T. Hsu, and C. G. Sodini, "System design for pixel-parallel image processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 4, pp. 32–41, Mar. 1996.

[48] E. S. Gayles, R. M. Owens, and M. J. Irwin, "The MGAP-2: A Micro-Grained Massively Parallel Array Processor," in *Proceedings of ASIC-95*, IEEE Computer Society Press, Sep 1995.