INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

Computational RAM: A Memory-SIMD Hybrid

by

Duncan George Elliott

A thesis submitted in conformity with the requirements
for the Degree of Doctor of Philosophy,
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Computational RAM: A Memory-SIMD Hybrid

Doctor of Philosophy dissertation, 1998

Duncan George Elliott

Department of Electrical and Computer Engineering

University of Toronto

In this thesis, a novel computer architecture called Computational RAM (C•RAM) is proposed and implemented. C•RAM is semiconductor random access memory with processors incorporated into the design, while retaining a memory interface. C•RAM can be used to build an inexpensive massively-parallel computer. Applications that contain the appropriate parallelism will typically run thousands of times faster on C•RAM than on the CPU. This work includes the design and implementation of the architecture as a working chip with 64 processor elements (PEs), a PE design for a 2048-PE 4 Mbit DRAM, and applications.

C•RAM is the first processor-in-memory architecture that is scalable across many generations of DRAM. This scalability is obtained by pitch-matching narrow 1-bit PEs to the memory and restricting communications to using 1-dimensional interconnects. The PEs are pitch-matched to memory columns so that they can be connected to the sense amplifiers. The 1-bit wide datapath is suitable for a narrow, arrayable VLSI implementation, is compatible with memory redundancy, and has the highest performance/cost ratio among hardware arithmetic algorithms. For scalability, the memory arrays and memory-style packaging limit the internal interprocessor communications to 1-dimensional networks. Of these networks, both a broadcast bus network and a left-right nearest-neighbour network are implemented.

C•RAM requires little overhead over the existing memory to exploit much of the internal memory bandwidth. When C•RAM PEs are added to DRAM, more than 25% of the internal memory bandwidth is exploited at a cost of less than 25% in terms of silicon area and power. The memory bandwidth internal to memory chips at the sense amplifiers can be 3000 times the memory bandwidth at the CPU. By placing SIMD PEs adjacent to those sense amplifiers, this internal memory bandwidth can be better utilized.

The performance of C•RAM has been demonstrated in a wide range of application areas, and speed-ups of several orders of magnitude compared to a typical workstation have been shown in the fields of signal and image processing, database, and CAD.

Acknowledgments

I am grateful for the support from and technical exchange with MOSAID Technologies and IBM Microelectronics; IC fabrication provided by Northern Telecom through the Canadian Microelectronics Corporation; and support from the Natural Sciences and Engineering Research Council of Canada, Micronet, and the University of Toronto.

Professor David Wheeler of the Computer Laboratory, University of Cambridge introduced me to the idea of putting the computing in the memory chips. My supervisors, Professors Michael Stumm and Martin Snelgrove kept me on track without a single change of topic. Michael Stumm put in more than a reasonable effort in encouraging me and organizing my work. Martin Snelgrove was instrumental in making connections to potential industrial partners and taught me valuable business lessons. I have learned lessons on DRAMs that no textbook could tell from Peter Gillingham, Graham Allan, Randy Torrance, Dick Foss, Iain Scott, David Frank, Howard Kalter, and John Barth. My lab-mates and members of the semiconductor design staff at MOSAID improved the quality and the enjoyment of my learning. Oswin Hall provided the first user feedback on C•RAM and proposed the PE shift register circuit. For strengthening this work by offering their questions and suggestions I also thank: Christian Cojocaru, Dickson Tak Shun Cheung, David Galloway, Robert McKenzie, Wayne Loucks, Tet Yeap, Martin Lefebvre, Jean-Francois Rivest, Sethurman Panchanathan, Wojciech Ziarko, Thinh Le, Glenn Gulak, David Wortman, Graham Jullien, Safwat Zaky, and Zvonko Vranesic.

This dissertation would not have seen the light of day without encouragement from my whole family. I would especially like to thank my wife, Janet Elliott, who was a constant source of enthusiasm, had streams of writing suggestions, and in the end sat me down to write this document.

Contents

1 Introduction 1

1.1 Dissertation Organization 5

2 Prior Art 7

2.1 Massively Parallel SIMD Processors 7

2.2 Memories 20

2.2.1 DRAM Circuits 22

2.2.2 DRAM Chip Interfaces 24

2.2.3 DRAM IC Process 25

2.2.4 Redundancy 27

2.2.5 Smart Memories 28

2.3 Summary 29

3 Computing in the Memory: the Abstraction 30

3.1 Memory Bandwidth 31

3.2 Maintaining Compatibility with Memory 34

3.2.1 Using DRAM IC Processes 34

3.2.2 Physical Dimensions and Performance 34

3.2.3 Coping with Long Wordlines 35

3.2.4 Redundancy Considerations 36

3.2.5 Packaging Considerations 38

3.3 Additional Design Issues 39

3.3.1 Asymptotic Advantages of Bit-Serial PEs 39

3.3.2 PE Aspect Ratio 42

3.4 Basic C•RAM Architecture 45

3.5 Advantages of C•RAM Architecture 46
3.5.1 System Cost 46
3.5.2 Performance 47
3.5.3 Scalability 47
3.5.4 Power Consumption 49
3.6 Design Space Alternatives 49
3.7 Summary 50

4 Architectural Details **52**
4.1 Processing Element 52
4.1.1 ALU Design 53
4.1.2 Conditional Operations 58
4.2 Interconnection Network 59
4.2.1 Suitability of Interconnection Networks 60
4.3 System IO 67
4.4 Summary 67

5 Implementation **69**
5.1 Prototype 69
5.1.1 Memory 70
5.1.2 Processor Element 74
5.1.3 Testing and Performance 81
5.1.4 Summary 81
5.2 High Density Design 82
5.2.1 Reference DRAM Architecture 83
5.2.2 C•RAM Memory 83
5.2.3 PE-Memory Interface 84
5.2.4 Processor Element 85
5.2.5 Power 88
5.2.6 External Interface 90
5.2.7 Multiple Memory Arrays per PE 92
5.3 Summary 93

6 Applications **95**

6.1 Programming Model 95

6.2 Experimental Method 96

6.3 Signal Processing and Graphics Applications 98

6.3.1 Convolution 99

6.3.2 Vector Quantization Image Compression 100

6.4 Database Applications 102

6.4.1 Least Squares Match 102

6.4.2 Data Mining 103

6.5 Computer Aided Design Applications 106

6.5.1 Fault Simulation with Multiple Faults 106

6.6 Exhaustive Solution-Space Search 108

6.6.1 Satisfiability Problem 108

6.7 Memory Operations 110

6.7.1 Clearing Memory to Zero 110

6.8 Summary 110

7 Conclusions **112**

7.1 Future Work 113

A Carrying On **114**

A.1 Technology Transfer to Industry 114

A.2 Carrying On in Academia 114

B Implementation of Common C•RAM Operations **116**

B.1 Memory Copy 117

B.2 Addition 118

B.3 If-else 119

B.4 Multiplication 119

B.5 Find Minimum Element 120

B.6 Two-dimensional communication 121

B.7 Summary 123

C Optimal Data Placement for Two-Dimensional Grid Problems **124**

C.1 Left-Right Communication 124

C.2 Scalable Left-Right Interconnect 125

D Glossary **127**

List of Figures

1.1 Memory cell array incorporating PEs 2

2.1 X-Net interprocessor communication network. 14
2.2 Generic DRAM cell array 21
2.3 DRAM memory cell 21
2.4 DRAM memory column 23

3.1 Conventional workstation architecture 31
3.2 Memory bandwidth throughout system 32
3.3 Redundant memory columns and PEs 36
3.4 Yield as a function of PE width 37
3.5 PE dimensions 43
3.6 The effect of PE pitch on chip area per PE 44
3.7 C•RAM computer architecture 46

4.1 Generic single data register C•RAM PE 54
4.2 Arithmetic performance vs. PE area 55
4.3 Minimalist NAND processor element 55
4.4 256 Function processor element 57
4.5 Shift register interconnection 62
4.6 The mapping of pixels to PEs 62
4.7 The mapping of pixels to PEs for scalable interconnection 63
4.8 Scalable left-right (4-pin) interconnection 64
4.9 Scalable left-right interconnection with increased bandwidth 65
4.10 256-Function processor element including interconnection 66

5.1 64 processor prototype C•RAM die photo 70
5.2 Detail of 12 C•RAM PEs 71
5.3 Prototype processor element architecture 72
5.4 Detail of write-enable and bus-transceiver control 73
5.5 (A) Classic dynamic logic, (B) Evaluate signal combined 77
5.6 Dynamic logic 8-to-1 multiplexor 78
5.7 PE register 79
5.8 Simultaneous bi-directional bus transceiver 79
5.9 PE column decode multiplexor 85
5.10 Dynamic logic 8-to-1 multiplexor 86
5.11 PE register with second input 87
5.12 Timing of a memory access followed by an operate cycle 90
5.13 Memory with bitline direction datalines 92

List of Tables

2.1 Characteristics of massively parallel processors 8

3.1 ALU algorithm and complexity 40

5.1 C•RAM SIMD instruction bits 91

5.2 Other additional pins for C•RAM 92

6.1 Performance summary 98

Chapter 1

Introduction

The key contribution of this dissertation is the proposal, design, and implementation of Computational RAM (C•RAM), an architecture that integrates memory and Single Instruction stream Multiple Data stream (SIMD) parallel processing onto a single chip to allow processors access to greater memory bandwidth. A C•RAM chip is a conventional memory chip, with the addition of processing elements. The processing elements are typically narrow-pitch 1-bit processors that have the same width as a small number (1-4) of memory columns. Figure 1.1 depicts a typical memory cell array augmented by processing elements, as would be found in a C•RAM chip. With the exception of the Processing Elements (PEs) at the bottom, the array corresponds to a conventional basic memory layout.

With this architecture, each C•RAM chip will contain many PEs. For example, in section 5.1 an 8 Kb SRAM chip with 64 PEs, and in section 5.2 a 4 Mb DRAM chip design with 2 K PEs will be described. A typical 64 Mb DRAM chip could contain as many as 32 K PEs. Each PE is kept simple, typically implemented with less than 100 transistors, as opposed to the 10 million transistors used to implement current high-end microprocessors. Nevertheless, we will show in chapter 6 that C•RAM can outperform workstations by more than a factor of 1000 for a large body of applications. All PEs execute the same instruction in the same cycle, making this a SIMD (Single Instruction stream, Multiple Data stream) architecture. An off-chip controller, in place of the DRAM controller, coordinates memory accesses and issues SIMD instructions to all PEs. The PEs each have "local" access to the physically adjacent portion of the memory array, namely the columns "above" them. Due to the fact that the memory wordlines run across a memory array shared by multiple PEs, all PEs must, within a single memory cycle, make memory accesses to the same address within their local

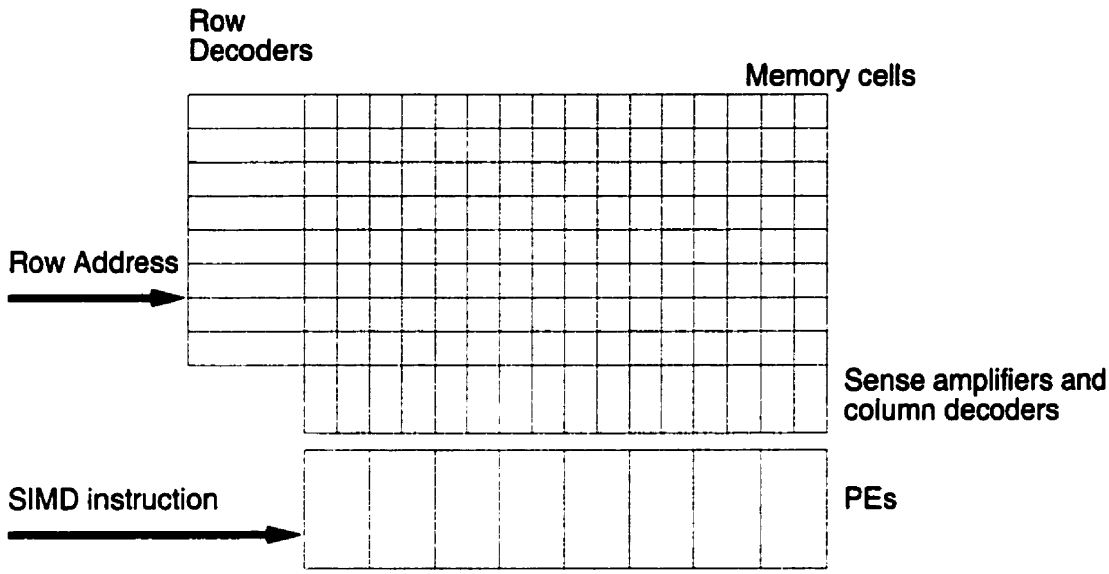

Figure 1.1: Memory cell array incorporating PEs

partition of memory; that is, C•RAM uses uniform memory addressing. A low-cost communication network allows for inter-PE communication.

The C•RAM architecture is primarily motivated by two considerations: bandwidth and the performance/cost ratio. The first consideration is that an enormous amount of memory bandwidth exists within a memory chip and only a small portion of that is accessible from off-chip through the pins. The ratio of internal to external bandwidth is 470:1 for the 16 Mb DRAM described in section 3.1. Furthermore, we compare the memory bandwidth internal to the chips in a 32 MB memory sub-system with the bandwidth at the microprocessor (assuming optimistically a 100% cache hit rate) and show that the ratio is 3600:1. For applications with poor memory temporal locality and thus poor cache performance, the bandwidth in the memory can be 15,000 times greater than that available to the CPU. Memory bandwidth is an important limiting factor in the performance of current computer architectures [1]. Integrating processors into the memory chips by adding simple PEs to the memory columns allows much of the internal bandwidth to be exploited by the PEs directly.

The second consideration that motivates C•RAM is that, by integrating processors and memory, it is possible to provide a huge amount of processing power at minimal cost. We will show that PEs can be added to DRAM with only an 18% increase in area. The marginal cost increase for C•RAM relative to the cost of the entire system is small. For this small incremental price, C•RAM could be used to replace the system or graphics DRAM in a workstation and be used either as traditional

memory, or, for suitable applications, as a powerful SIMD parallel processor. For some scalable applications, the C•RAM performance scales linearly or nearly linearly with the number of C•RAM chips in a system.

We are particularly interested in DRAM (as opposed to SRAM) as a platform for C•RAM. DRAM is the most commonly used memory found in today's computers and video frame buffers, because of its high density and low cost. A commercial version of C•RAM based on low-cost DRAM could replace commodity memory in a variety of applications, thus adding array processing capability to the hardware. In order to do so economically, we examine massively parallel architectures where the PE adds a minimal incremental cost to the memory, while maintaining maximum compatibility with the commodity memory in function, density, and packaging.

When C•RAM is used as main memory or frame buffer memory for a conventional processor, the SIMD array and the host communicate by using shared memory without the latency of sending messages through an interprocessor communications network. C•RAM can be read and written at speeds similar to conventional memories. Further, the host can steal memory cycles from the C•RAM PEs (e.g. for filling a cache line) without altering the state of the PEs.

While SIMD computer architectures have been studied for three decades, there are several reasons why massively parallel SIMD machines have not become commonplace. The first reason relates to economics. Most SIMD machines have been designed for low quantity production and very high-end applications, so they are quite expensive.

The second reason why massively parallel SIMD machines have not become common place is that the memory and processors have not been typically implemented in the same chip. Hence, the maximum number of processors per chip is limited by the number of pins in an IC package. This separation of memory and processor is usually a result of design for small volume production. Using commodity memory chips for a particular SIMD design reduces design time, eliminates the need for memory designers on the project, and permits the use of an ASIC (application specific integrated circuit) manufacturing processes for the processor portion. ASIC processes have lower one-time costs per chip, but are not optimized for the manufacture of memory. When the processors have a large number of pins, the IC packaging cannot be as dense as memory packaging, resulting in computers with large physical volumes.

Thirdly, the host-to-SIMD interface often poses a bottleneck when results have to be shipped between the sequential host processor and SIMD array (often housed in different cabinets). Low bandwidths between host and SIMD array are common in these previous systems, and high latencies

(milliseconds) to set up the transfer — even for transfers of a single word — are not uncommon.

Fourthly, SIMD machines are conceptually more difficult to program than conventional von Neumann machines because the application (and data) must be partitioned among the processing elements by the application programmer (or a clever compiler).

We attempt to address all four of these issues in this dissertation. In contrast to most of the previous efforts, with C•RAM we aim to *(i)* leverage high volume DRAM manufacturing to keep marginal costs low, *(ii)* integrate PEs and memory for the bandwidth and packaging advantages, and *(iii)* provide a standard memory interface so communication between host and SIMD array can have low latency. While we do not provide a clever parallelizing compiler, we have found that the parallel applications are not much longer and are sometimes slightly shorter than the corresponding sequential versions. Also, the low latency shared-memory interface between host and C•RAM permits a gradual conversion of non-parallel application code to code that explicitly makes use of C•RAM's parallelism, starting with the most computationally-intensive portions.

In our opinion, because of the inherent restriction of SIMD architectures that each PE must perform the same operation in the same cycle, SIMD machines must have a significantly better performance/price ratio than MIMD (Multiple Instruction stream, Multiple Data stream) machines in order to compete. As we will show, a common pitfall in SIMD design is to continually add features that adversely affect performance/price and even net performance — reminiscent of the art of processor design prior to the adoption of RISC (Reduced Instruction Set Computer) philosophies. Once a SIMD architecture is encumbered by too many features, MIMD becomes more attractive. In contrast, we accept the fact that many applications will not be suited to run well on C•RAM even though C•RAM can run applications from a wide range of fields well. Hence, we do not pursue features that would assist only limited numbers of applications or would encumber the PEs.

In spending resources only on features that will enhance overall performance, our philosophy mimics that of early RISC designers. Two deviations from the RISC philosophy, however, are evident. First, we assume an abundance of parallelism in our applications, so that we can trade PE complexity for additional PEs. Second, our application set is not as well defined at the start, so our architectural tuning does not have a performance measure which is as objective as a SPEC benchmark set. The fastest computers (including parallel computers) probably won't be used exclusively for the same applications mix as the previous generation of machines. As Briggs noted [2], faster computers will make more computationally intensive applications viable.

"If you give people a machine with plenty-o-parallelism, someone's going to hack their

code to take advantage of it (and all the other hardware resources)."

Throughout this dissertation, we make the assumption that high-density DRAM-based C•RAM would be manufactured in high volume so that the cost per C•RAM chip might approach the cost of a similar sized DRAM chip. Getting from a design to high volume DRAM manufacture, or even a prototype is a difficult task in the business sense. Manufacturing one batch of DRAM wafers can displace nearly one million dollars of salable product[1]. There are further costs for mask creation (50-300 thousand dollars) and for introducing a different design into the production stream. Thus the non-recurring (i.e. one-time or "up-front") costs for building processors into DRAM are high and there is no broker service (such as CMC or MOSIS) which helps to share the up-front costs by merging designs into "multi-project" chips. Although the initial costs are high, the performance/cost ratio of C•RAM can also be very high if manufactured in a DRAM process.

In the interest of minimizing up-front costs for a new massively-parallel processor, it is tempting to buy standard memory chips rather than integrating PEs into memory. However, with processors and memory on separate chips, the chip pin count will limit the number of processors that can be integrated on a chip. With limited processing power per chip, the further temptation exists to use standard microprocessors instead of designing custom chips. We resist these temptations by looking to the long-term picture and discounting the up-front costs.

The scope of C•RAM's applicability should not be limited to high-end computer needs. We hope that C•RAM will be used in applications where memory is currently employed and the addition of massively parallel processing would be of some use. This should include graphic and video frame buffers and desk-top computers. We believe that C•RAM would also be highly suitable for use in embedded systems for applications ranging from radar signal processing to high definition television.

1.1 Dissertation Organization

An overview of past work in the fields of SIMD architectures and dynamic memory is given in chapter 2. In chapters 3 and 4 the design trade-offs and architecture of C•RAM are developed. To

[1] NEC [3] produces 240 16 Mb dies on an 8-inch wafer. DRAM wafers are processed in large batches of, for example, 100 wafers. Functional 16 Mb DRAM chips sell for $55US to $60US (July 1995). The opportunity cost of making a batch of prototype chips in a production DRAM foundry instead of readily marketable DRAMs is therefore approximately $(\frac{\$55}{chip} \times yield \times \frac{240\ dies}{wafer} \times \frac{100\ wafers}{batch})$ = yield times 1.3 million US dollars. Of course memory prices fluctuate and yields improve as a process matures.

show that our architecture is viable, we designed and tested a prototype chip and designed a DRAM based PE, both of which are described in chapter 5. Examples of applications that are able to exploit C•RAM are given in chapter 6.

Chapter 2

Prior Art

In this chapter, we briefly describe some SIMD multiprocessor architectures and memory technology, since the design of C•RAM involves both. A large number of SIMD architectures have been proposed over the years. Here, we focus on massively parallel SIMD architectures that either have been manufactured in quantity or are particularly interesting.

For DRAM, on the other hand, a detailed description of typical circuits and architectures is provided. We focus on those aspects of DRAM that might affect the integration of PEs.

2.1 Massively Parallel SIMD Processors

Flynn defined the SISD, SIMD, and MIMD classes of computers in 1966 [4]. SISD (Single Instruction stream, Single Data stream) is the classic uniprocessor. Multiple copies of these uniprocessors, each with its own instruction fetch and decode units, and datapath, can be tightly coupled together to build a MIMD (Multiple Instruction stream, Multiple Data stream) computer. If, however, multiple processor datapaths are driven by a single instruction decode circuit, then the computer is classified as SIMD. SIMD computers have the obvious advantage of economizing on instruction decode circuitry and feature intrinsically synchronized operation. Chapter 3 describes other advantages of SIMD architectures as applied to computing in memory.

The SIMD systems we describe in this section are listed in table 2.1. The table provides:

- year of introduction
- whether the PEs and memory are on the same chip
- if the PEs and memory are integrated

Machine	year	on-chip main mem.	mem. redun-dancy	mem. b / PE	local mem.	PEs / chip	PE redun-dancy	max PEs	data-path	autonomous memory addressing	autonomous network
STARAN	1972			256		8		1K	1		
ICL-DAP	1976			4096	✓	16		1K	1		
VASTOR	1978				✓	1			1		
MPP	1981				✓	8		16K	1		
GAPP	1984	✓		128	✓	72			1		
CM-1	1985			4096		16		64K	1	note 1	✓
Pixel Planes 4	1987	✓		72	✓	64		256K	1		
AIS	1988			32K	✓	8		1K	1		
BLITZEN	1989	✓		1024	✓	128		16K	1	✓	
Pixel Planes 5	1989										
MasPar MP-1	1989				✓	32		16K	4	✓	✓
C•RAM	1989	✓		128	✓	64			1		
DAP 610C	1990			256K	✓	64		4K	1 & 8		
VIP	1990	✓		256	✓	256		256	1		
TI-SVP	1990	✓		320	✓	1024			1		
CM-2	1990			1M		16		64K	1 & 64	note 1	✓
MasPar MP-2	1993				✓	32		16K	32	✓	✓
SRC-PIM	1994	✓	note 2	2048	✓	64		256K	1		
NEC-IMAP	1994	✓	note 2	32K	✓	64			8	✓	✓
EXECUBE	1994	✓	✓	512K	✓	8			16	✓	✓
C•RAM	1995	✓		480	✓	512		4096	1+		
NEC-PIPRAM	1996	✓	note 2	128K	✓	128	✓		8		
SONY-linear array	1996	✓		256	✓	4320			1		
C•RAM-DRAM		✓	✓	2048	✓	2048	✓	1M	1		

Table 2.1: Characteristics of massively parallel processors

note 1: autonomous memory addressing implemented in software
note 2: memory redundancy implemented with spare blocks

- whether that integrated memory is protected against manufacturing defects by redundancy
- the amount of memory per PE
- the number of PEs per chip
- the number of PEs in a fully configured system
- the width of the PE datapath
- whether or not memory addresses are autonomously controlled by each PE
- whether or not interprocessor communications network destinations are autonomously controlled by each PE

We use the term *autonomous memory addressing* to describe SIMD machines that allow the PEs to calculate their own memory address for accessing their private memories. The opposite is *uniform memory addressing* where all PEs access the same address within their local memories. Similarly, interprocessor communication network routing is classified as *autonomous network routing* if PEs

can independently specify the target PEs and *uniform network routing* if all messages are routed in the same pattern.

It is interesting to note that none of these systems has been particularly economically successful to date. It is difficult to identify the reasons for this failure, but we believe that the reasons given in the introduction played an important role: i.e., *(i)* most SIMD machines were not designed for quantity; *(ii)* many lack or have poor SIMD processor - memory integration; *(iii)* these machines typically have a poor host - PE data interface; and *(iv)* without automatic parallelizing compilers, SIMD architectures present a more difficult programming model[1]. We suspect (and cite examples) that these limitations are due to the fact that the goal of minimizing volume manufacturing costs was traded-off for minimizing one-time design costs.

An example of minimizing design effort is the use of standard memory chips. If, instead, memory and PEs had been integrated, then pin limitations would have been avoided, thereby permitting more PEs per chip; yet this was seldom pursued at the time when we started our work, due in part to the higher up-front costs. Of those designs that do combine PEs and memory, most do not fully exploit the internal memory bandwidth, nor do most use the most dense memory available.

Stone was perhaps first to propose the integration of processors and memory in the memory of a computer in 1970 [6]. At a time when ferromagnetic core memory dominated main memory systems, he proposed integrating processors into the semiconductor cache memory. In the two decades that followed, the understanding of how to integrate memory and processors, as well as the will to do so, has been weak. C•RAM was the first chip with processors integrated into memory such that the memory was externally accessible and could be used as computer main memory [7, 8].

Where processors and memory have been integrated, redundancy (to permit the repair of manufacturing defects) has not been widely used, limiting their yield or transistor count, thus increasing the price of parts or limiting levels of integration, respectively. This is another reason why massively parallel processors currently occupy a high price niche.

In the following, we examine for each of these systems: the class of applications targeted, the features of the PEs, the style of the memory, and the interprocessor communications network. As well, we elaborate on other unique or interesting features. The machines are presented in order of date of introduction of the machine, product family, or chip.

---

[1] Although automatic techniques for detecting, categorizing, and exploiting parallelism in programs are well known [5], I believe that in the majority of SIMD application programs, the programmer has either explicitly declared the parallelism or has written the program so as to make the parallelism more apparent to the compiler.

STARAN

STARAN was the earliest production SIMD machine [9-15]. It was built by Goodyear Aerospace under contract from NASA Goddard Space Flight Centre. It was used for applications such as satellite image processing and radar signal processing for AWACS (Airborne Warning And Control System). The processing elements are bit serial, meaning that the datapaths (ALU, registers, etc.) are 1 bit wide and that operations on larger data items (e.g. integers) are synthesized from a sequence of 1-bit operations. Each PE can perform two arbitrary boolean functions of 2 inputs (the two inputs being a data register and the output of the interprocessor communications network) simultaneously in one cycle. Each PE contains a mask register and 2 data registers. The mask register is used to selectively halt some PEs when executing conditional code, a common feature in SIMD machines. STARAN shares a common pool of memory among the PEs, which is an uncommon feature among SIMD machines. Most SIMD machines have memory permanently dedicated to each PE (i.e., they implement *private distributed memory*). The *flip* permutation network always supplies operands to the ALUs. When the PEs perform operations on local data, the "identity permutation" is used.

Being an expensive machine, only five STARAN systems were produced (2 with 512 PEs, 3 with 1024 PEs). A variant of STARAN was used into this decade in AWACS (airborne military radar, a high budget application). The design of the PEs, memory, and interconnect make high levels of integration difficult. Only 8 PEs are integrated on a die. Since memory and PEs are in different chips, the chips would become pin-limited before large numbers of PEs could be integrated.

DAP

The ICL DAP (Distributed Array Processor) was first designed to be a multi-purpose active memory module for the ICL 2900 mainframe [16]. The architecture was intended to be general purpose (as far as SIMD can be) and has been used for signal processing and text indexing, as well as for classified military applications. The first implementation had 1024 PEs built out of medium scale integration TTL components. A CMOS version with 16 PEs per chip has since been developed. Each one-bit processor contains a full adder and 3 registers. Standard memory chips are used for the 4 Kb of memory per PE. Each PE communicates with its 4 neighbours on a 2-dimensional grid. A 4-to-1 multiplexor in each PE selects the network output of one of the adjacent PEs as that PE's network input.

Active Memory Technologies Ltd. further developed the DAP 500 as a SIMD "back-end

processor" to complement a VAX minicomputer or SUN workstation host [17, 18]. The DAP 500 contains a grid of 32 × 32 PEs with 64 PEs integrated into one chip. Thirty-two bus lines in each of the row and column directions are provided explicitly for IO.

To improve arithmetic performance, the DAP 510C (32 × 32 PEs) and DAP 610C (64 × 64 PEs) have an 8-bit math co-processor associated with each PE, but located on a separate chip. Each math co-processor has an 8-bit adder and 8-bit registers, but no multiplier. The separation of the PE and co-processor into different chips allowed the AMT DAP designers to re-use the existing PE chip, but in doing so, they restricted the 8-bit co-processor's memory path to the 1-bit path provided for the original PE.

VASTOR

VASTOR (Vector Associative Store - TORonto) was designed at the University of Toronto for applications involving searches and other associative operations [19-22]. The TTL implementation was built around the Motorola MC14500B industrial control unit, which had a one-bit-wide datapath, an accumulator, a seven-instruction ALU, and write enable logic. VASTOR has since been implemented both with TTL MSI components and custom silicon [23, 24]. Custom ICs containing a single PE were built, but were never functional due to manufacturing defects. High level language programming was done in APL [25].

Interprocessor communication is provided by a linear nearest-neighbour communication network (a 1-dimensional shift register) and two combine networks. The shift register is also used for IO. A combine network takes multiple inputs and produces a single result, such as, for example, a summation tree built of adders. The output of the flag register in each PE is connected to the two combine networks. The OR combine network can be used to signal the controller that one or more matches have been found. A novel "≥ 2 responding" combine network reports if more than one PE has its flag set, and can be used to determine if the result of a search has been pared down to a single item. The global-OR and "≥ 2 responding" combine networks are implemented with linear chains of logic that have $O(n)$ depth and delay.

MPP

The Massively Parallel Processor (MPP) [26, 27] built by Goodyear Aerospace for NASA Goddard Space Flight Center contains several 128 × 128 processor planes. The MPP was designed

to process Landsat satellite images so it makes heavy use of its 2-dimensional grid connectivity. Each PE contains a one-bit full adder. These bit-serial processors are packaged 8 to a chip. Memory is off-chip from the PEs. The MPP was used for more than a decade before it was de-commissioned.

GAPP

NCR Corporation, using an architecture licenced from Martin Marietta Electronic Systems, produced a chip with a 6×12 array of bit-serial processors and local memory, which they call the Geometric Arithmetic Parallel Processor (GAPP) [28, 29]. GAPP was designed for image processing and has been used in military applications. The PEs operate on instructions with 5 fields that control the full adder, 3 data registers and a communications register. Each PE can communicate with its four nearest neighbours on a 2-dimensional grid and with a private 128-bit on-chip memory. Interprocessor communication and ALU operations can proceed concurrently.

Due to their complexity, the 72 PEs take up slightly more than half the chip. The memory arrays in GAPP do not support redundancy, unlike most commercial RAMs. While integrating the PEs and memory saves IC pins and communication overhead, GAPP does not include a direct mechanism for a host processor to access the memory in the GAPP chips. Instead, data to be sent to the host must be shifted through PEs to the edge of the GAPP chip. If the chips are connected in an array as the manufacturer intended, the data must be shifted to the edge of the array of processors.

Connection Machine

The Connection Machine (CM) was designed by Thinking Machines Corp. for use in connectionist (synthetic neural network) research [30]. Most CM cycles, however, have been used for scientific computation. A fully configured CM-1 has 64 K 1-bit processors with 4 Kb of memory per PE. The CM-1 native instructions specify 3 memory addresses, a register, and two ALU operations. A dual ALU performs two arbitrary 3-input boolean functions of a register and two memory locations, placing the two results in a register and a third memory location. Three-operand ALU instructions predominate in RISC microprocessor architectures, where a triple-ported register file provides the necessary operand throughput, but we question the use of a three-operand instruction for the CM-1 where each operand requires a separate cycle for a memory access. Commodity DRAM is used as main memory. Sixteen PEs plus an autonomous hypercube network router are placed on a chip. The peak integer performance of a 64 K PE CM-1 is one billion 32-bit additions per second.

The CM-1 *C** compiler generates code that runs on the host processor and issues PARIS (PARallel Instruction Set) *macro* instructions (e.g. ADD_16bit) to the CM-1 sequencer. This microcoded sequencer in turn translates the macro instructions and issues native SIMD instructions to the PEs. Branches and other scalar operations coded in the high level language do not get sent to the sequencer but are executed on the host.

The next generation machine, the CM-2, adds a 64-bit floating-point chip to every group of 16 PEs [27]. Also, within groups of 32 PEs, the PEs can access each other's local memories. The large 1-Mb-per-PE memory system is built with commodity DRAM chips. Its larger-than-average size was chosen to permit multi-tasking. The hypercube communications network can be used for autonomous message routing for send and get operations, as a uniformly routed network for patterned communications, or as a combine network.

While Thinking Machines Corporation has enjoyed the most publicity and highest sales of any massively parallel SIMD machine manufacturer to date, the limitations of their designs caused them to switch from manufacturing SIMD systems to MIMD systems in 1991. Without memory integrated into the PE chips, the number of PEs on a chip is limited to 16 PEs in the CM-2. According to Thinking Machines, the designers of the CM-5 chose to build a MIMD machine using commercially available microprocessor chips (SPARC) with one 32-bit wide processor per chip instead of custom chips with 16 1-bit wide processors per chip for reasons of cost, performance, and availability.

Pixel-Planes

The Pixel-Planes machine [31-35], as the name suggests, was designed for computer graphics applications. In particular, it is capable of rendering all pixels in an object in parallel. The Pixel-Planes-4 machine integrates 64 bit-serial PEs, each with a small 72-bit memory, on a chip together with a pipelined tree-structured bit-serial linear expression evaluator. Expressions of the form $F(x,y) = Ax + By + C$, (where x and y are the PE co-ordinates, and A, B, C are scalar variables) are efficiently evaluated in parallel, exploiting the similarities in the binary addresses of neighbouring PEs. Such linear expressions are used, for instance, to shade a pixel or determine if it lies inside a polygon.

The next generation Pixel-Planes-5 integrates 128 PEs, each with 208 bits of memory, and a quadratic expression evaluator into a single chip. The Pixel-Planes-5 was developed as a MIMD-SIMD hybrid to address the problem that graphic objects are often small, making it inefficient to involve the entire SIMD array in rendering an object which only occupies a few pixels. The MIMD

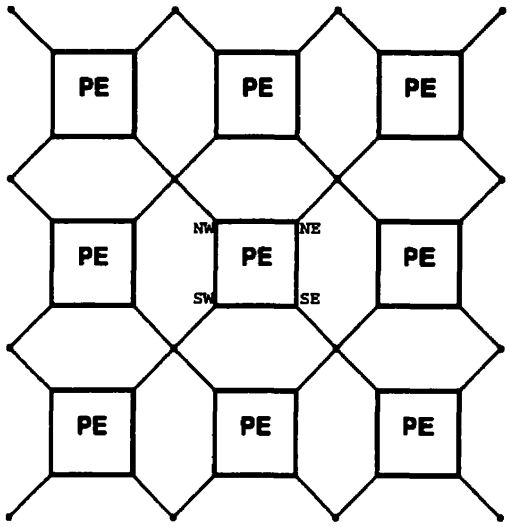

Figure 2.1: X-Net interprocessor communication network.
A bi-directional communications port on each corner of each PE is wired to 3 neighbours.

subsystem, built with DSP chips, pre-processes the graphic objects (including co-ordinate transforms and clipping) and sends them to the appropriate SIMD chip. The SIMD subsystem divides up image space among the chips so that different chips can be rendering different objects.

By building a massively parallel machine for a specific application, the designers of Pixel-Planes-5 were able to create a reasonably well balanced system and the fastest rendering machine of its time. The authors comment that for many operations, data transfer between the host and Pixel-Planes dominates execution time. Each existing version of Pixel-Planes has been a one-off design, built at significant expense.

C•RAM

In November 1989, the first C•RAM chip designed at the University of Toronto was sent for IC fabrication [7]. These designs had narrow PEs, the pitch of an SRAM column, integrated into the memory array. A memory interface allowed direct off-chip memory access. The architecture and chip are described in more detail in the following chapters. This work has been referenced by some of the subsequent designers [36, 37, 38, 39].

BLITZEN

BLITZEN [40, 41, 42] is a successor to the MPP. Designed at the Microelectronics Center of

North Carolina, it was partially funded by NASA Goddard Space Flight Center. A BLITZEN PE contains a 1-bit full adder; a 30-bit shift-register/accumulator; and registers for masks, conditional operations, carry, and communications. A system contains 16 K PEs, 128 per chip. Each PE has 1 Kb of on-chip memory that can be addressed uniformly by the SIMD controller, autonomously using the contents of the shift register, or a combination of the two.

The X-Net interprocessor communication network, shown in figure 2.1, is a 2-dimensional grid plus diagonals so that every PE is connected to 8 neighbours (North, NE, E, SE, S, SW, W, NW). The bi-directional port on, for example, the North East corner of one PE is wired to the ports on the NW, SW, and SE corners of its E, NE, and N neighbours respectively. A 2-dimensional X-Net node can communicate with its 8 neighbours (including 4 diagonal neighbours) while using the same number of "pins" as a grid (which lacks communication with diagonally adjacent nodes). All PEs uniformly transmit to one corner and listen on another. If only one network input can be read at a time, then 8-neighbour connectivity has no advantage over 4-neighbour connectivity when the values from all 8 neighbors are required.

An OR-tree connecting all PEs (not shown) is also provided for combine operations. Moreover, IO is accomplished with a 4-bit bus for every 16 PEs.

While autonomous addressing is a useful feature for some applications, it has a cost in that the row-decode circuitry occupies nearly half the area of BLITZEN. This high area cost is due to the fact that the row-decode circuitry is replicated for the 4 columns of memory associated with every PE. In contrast, DRAM designs tend to use one row decoder for 2048 or more columns. Fewer row decoders driving longer wordlines could have been used in BLITZEN, had only uniform addressing been implemented. Because of the autonomous addressing and the low density ASIC memory, only 128 Kb of RAM and 128 PEs fit on the large 11.0×11.7 mm die in a 1μm process. The dies are packaged in 176 pin pin-grid-array chips, a physically large package. While the design contains more memory (128 Kb) than previous SIMD chips, its density is still 8 to 32 times lower than that used in similar generation IC processes designed for SRAM and DRAM respectively. Redundancy was not used to protect chips against defective memory cells or PEs. Consequently, according to project participants, yields were quite low.

MasPar

MasPar Corp. designed the MP-1 as a moderately priced SIMD computer for general scientific computing with a high performance/price ratio [43, 44, 45, 46]. The MP-1 uses a DEC workstation

as a host. Chips containing 32 4-bit processors are assembled into machines with 1 K to 16 K processors. The ALU contains an adder, multiplier, and support for logical operations. MasPar integrates a significant amount of static register memory into the PE die but uses off chip commodity DRAM for PE main memory. PEs can autonomously address their local memory, but with a slower access rate than when uniformly addressed. The memory bandwidth of the 16 K processor machine is 12 Gbytes/s, about one tenth of the register bandwidth of 117 GB/s, suggesting that memory bandwidth may pose a bottleneck, especially considering the fact that this design and others listed in this chapter do not have cache memories. Communication is supported by a 2-dimensional 8-neighbour X-Net as described for BLITZEN (figure 2.1), and a lower bandwidth circuit-switched global router which implements autonomous routing.

The next generation MasPar MP-2 chip has 32 PEs as did the MP-1, but the processor datapath has been increased from 4 to 32 bits [47]. Each PE contains approximately 14000 transistors, not including registers, or 450 transistors per bit of processor width. Despite the 8-fold increase in processor width, multiplies are only 4 times as fast as on the MP-1. A fully configured system would have 16 K PEs on 512 custom processor chips plus 3072 commodity DRAM chips. The total system memory bandwidth is 23 GB/s.

VIP

The Video Image Processor (VIP) was designed at Linköping University to process a video raster line at the horizontal refresh rate [48, 49, 50]. VIP chips are thus more likely to be combined in a pipeline (each performing different image processing operations) than in an array. A chip contains 256 PEs, each with 256 bits of memory and a 1-bit adder. A relatively high density is achieved with a compact ALU and a 3-transistor-memory-cell DRAM array. The 3-transistor cell is denser than the SRAM used in prior SIMD-plus-memory designs. A 1-transistor memory cell in an ASIC or DRAM processes would be denser still, but would have required more design effort. Video IO is handled by separate input and output shift registers that run through the PEs.

SVP

The Serial Video Processor (SVP) was designed at Texas Instruments for real-time signal processing on a video stream [51, 52]. The SVP chip contains 1024 bit-serial PEs, each containing a full adder and connected to 320 bits of memory. The PEs are organized in a row and are connected to

two memory banks so that two independent 1-bit operands can be accessed every cycle. Video-rate IO is handled by dual ported memory with a 40-bit wide path for input and a 24-bit wide path for output. Interprocessor communication is performed through a 1-bit wide left-right communications network that connects adjacent PEs, as well as next-to-adjacent PEs.

The prototype receives SIMD instructions from off-chip, but the designers intended to put instruction memory and decode circuitry on a production version of the chip so it could be used as a stand-alone video filter.

While the SVP used DRAM memory cells, which are smaller than SRAM memory cells, the large 1.5 million transistor count suggests that a large, 3-transistor dynamic memory cell was used. The consequence is a large 14.4 mm $\times$ 11.0 mm die. Since the chip has 107 signal pins plus an unspecified number of power pins, it is packaged in a large 179 pin PGA. This package is 15.7 cm^2 or 10 times the actual die area. If minimizing printed circuit board area and system cost are desirable then SIMD chips would have to be designed with fewer pins[2].

IMAP

The IMAP chip (Integrated-Memory Array Processor) was built in a 0.55 μm BiCMOS process at NEC for the purpose of low-level image processing [53, 54]. Each chip contains 64 8-bit PEs and 2+Mb of SRAM. Each PE contains an 8-bit adder, but multiplication is performed by table lookup in the PE's memory. The 2 Mb of SRAM (not including redundant banks) is arranged as a 4 K $\times$ 8 bit array per PE and can be addressed uniformly or the low order bits (column address) can be addressed autonomously. The IMAP uses a 4-transistor SRAM cell which is more expensive than DRAM, but also faster. Since the IMAP SRAM is twice as fast as the PE, the full memory bandwidth cannot exploited by the PEs. Interprocessor communication can be performed using the 8-bit bi-directional shift paths between left-right adjacent PEs or using a 1-bit wired-OR bus. IO transfers are done through a 4-bit memory bus or one of two 8-bit unidirectional shift registers. The memory access bus allows low-latency[3] memory accesses by a host processor.

The IMAP designers recognized the need for memory redundancy to improve chip yield. Their particular implementation of redundancy is, however, more expensive than redundant memory rows or columns (as described later in section 2.2.4). Eight pairs of PEs can be connected to 8 blocks

[2] At this time, minimizing system cost precludes using more expensive, high pin count chip-on-board techniques.

[3] We consider memory accesses via the external memory access bus to have low latency because the bus allows external memory operations with access times similar to conventional memories and the data takes a direct path, rather than traversing multiple hops in a communications network.

of memory plus 2 spare blocks. For a 25% overhead in memory area plus bus switches, only 2 manufacturing defects (in different blocks) can be repaired per 16 PEs. Each PE has approximately 7000 transistors (875 transistors per bit of datapath) and occupies an area of 419 μm $\times$ 2628 μm.

A follow-on design by NEC, the PIPRAM, integrates 128 8-bit PEs of a similar design into 16 Mb of DRAM memory.

EXECUBE

EXECUBE [55, 56] was designed at IBM Federal Systems (now Loral Federal Systems-Owego) for embedded high performance processing including radar tracking and pattern matching. A chip contains 8 16-bit PEs and 4.5 Mb of DRAM memory. The 8 PEs can operate in a MIMD mode where the PEs fetch their own instructions, or a SIMD mode where the PEs receive a broadcast instruction from off-chip. Each PE is connected to two 32 K$\times$9 bit DRAM arrays, with the ninth bit being parity. Since each DRAM array, with its own support logic, is complete and independent, autonomous memory addressing as well as row and column memory redundancy are easily supported. Each PE has four point-to-point links for interprocessor communication. The 8 PEs are cube-connected on-chip, but only 8 links are available for off-chip communication (less than the 24 links required for a full 3-dimensional mesh).

EXECUBE is based on high-density low-cost 1-transistor commodity DRAM built in a 4 Mb DRAM process. The DRAM arrays occupy 63% of the 14.7 mm squared die. The processors are implemented in "sea of gates" semi-custom logic. The EXECUBE's large die, large number of signal pins, and moderate power consumption require a large 208 pin IC package.

The SIMD/MIMD modes and autonomous memory addresses gives the chip excellent flexibility. In the MIMD mode, the programmer is presented with multiple conventional RISC microprocessors with message-passing communication between them. By building *onto* rather than *into* a DRAM array, EXECUBE's designers avoid working with low voltage signals, thus reducing design time, and increasing the certainty of the chip being functional. However, the implementation uses pre-defined memory macro-cells (the DRAM arrays) which has a cost in terms of power and delivered bandwidth. For example, after the column decoding, the data from each macro is accessed through an 8-bit data bus[4], even though 128 sense amplifiers are simultaneously active internally. Also, the chip has a 2.7 W power consumption, in part because all memory arrays on the chip (2 arrays per

[4]The 1 bit of parity per 8 bits of data is not included in the memory and datapath sizes described.

processor) are active during a memory access. The internal memory bandwidth, with its 128-bit wide datapath per array, goes largely underutilized, providing only page-mode "caching" of a memory row.

Terasys

The Terasys system is a SIMD processor in memory built by the Center for Computing Sciences (formerly the Supercomputing Research Center) and funded by the NSA (US National Security Administration) [57, 58, 36]. The prototype 32 K PE Terasys system attaches to a Sparc-2 and can be accessed as memory. Each of 512 chips contain 64 bit-serial PEs with 2 Kb of SRAM per PE.

In anticipation of building large systems and running long duration applications, error detection and correction is provided for correcting single-bit errors and detecting two-bit errors. The PEs and memory system allow masked writes. In this implementation of masked writes, the PEs first read the memory to obtain the old values, and write back the old or new values depending on the PEs' flag register. Memory redundancy, but not PE redundancy, is provided to improve yield. The 128 Kb of SRAM memory per chip was several generations behind the highest SRAM densities available at the time.

Interprocessor communication in the Terasys is supported by a global OR network, a *partitioned OR network*, and a *parallel prefix network*. The partitioned OR network can be partitioned into groups with sizes which are powers of two greater than or equal to 64 (the size of a chip). The parallel prefix network can be set to different *levels* allowing PEs separated by different distances to communicate. At level 0, the parallel prefix network provides communication to the left and right neighbouring PEs. At level l, processor $i2^l$ can receive data from the processor 2^l processors to the right of it through the parallel prefix network.

The designers of Terasys make effective use of the internal memory bandwidth. Their high performance goals, including the use of the Terasys design as memory for a vector supercomputer, makes the cost of SRAM acceptable.

Cray Computer built and tested a prototype 256 K PE "octant" of Processors In Memory (PIM) chips for the Cray-3, to be called the Cray-3/SSS (Super Scalable System).

Linear Array DSP

The Sony Linear Array DSP [59] has 4320 PEs, each with 256 bits of ASIC DRAM memory. Each ALU contains a 1-bit full adder and a Booth decoder for multiplies. As this chip was designed to speed up video format conversion, it has a 64-bit input channel and a 32-bit output channel. The die is packaged in a bulky 256 pin PGA. As with other designs built with ASIC DRAM, the memory cells are smaller than SRAM cells but larger than standard DRAM cells.

Summary

Many of these massively parallel SIMD machines, especially the earlier ones, do not attempt to integrate memory and processors (see table 2.1). Fortunately, however, the designers of more recent chips and systems have seen the importance of this integration. Of those which do integrate memory and processors, all but two (EXECUBE and PIPRAM) use less dense varieties of memory than the 1-transistor DRAM cell. Only Terasys, IMAP, and their successors are usable as memory. Insufficient attention is paid to improving yield through the use of redundancy. Where redundancy is used, it is most often not implemented with the efficiency of that seen in commodity memories. Most of these designs have large numbers of pins and are mounted in IC packages which occupy many times the area and volume of the original die.

In contrast to the systems described above, C•RAM integrates processors and memory, uses dense memory designs with redundancy to improve yield, and has a low pin count so that it can be packaged in a TSOP package like memory. These features are described in more detail in chapters 3, 4, and 5.

2.2 Memories

C•RAM integrates processors into the memory. This section gives a brief introduction to DRAM circuits, with emphasis on those circuit elements that will interact with the PEs that are added to memory.

Memory is a driving force in integrated circuit technology — it leads in number of devices per die and device dimensions. Due to its prominent position, the available literature is enormous. See [60-127] for some of the more relevant articles. The book *Semiconductor Memories* by Betty Prince [116] is a fairly comprehensive reference. In this section, a tutorial on DRAM circuits and

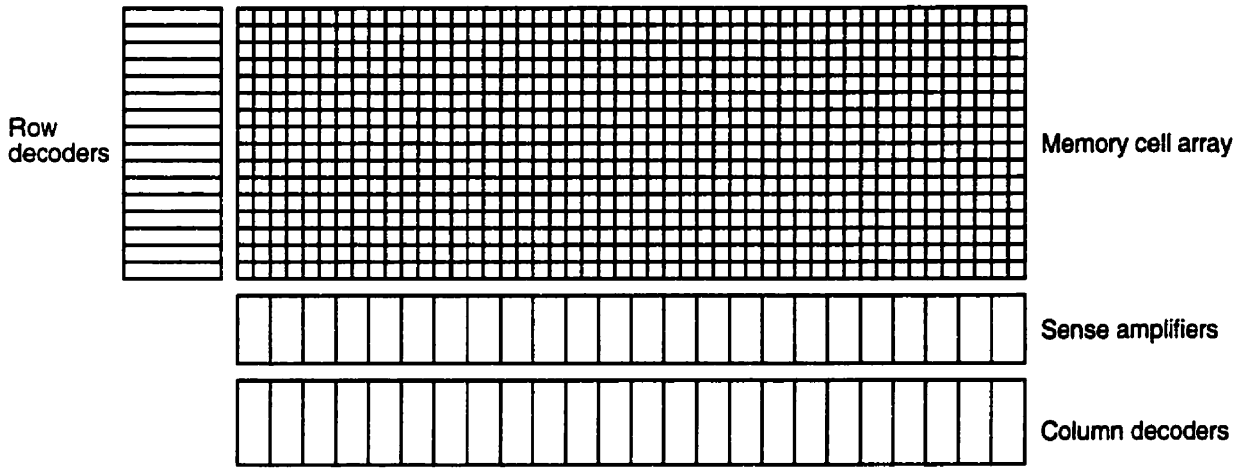

Figure 2.2: Generic DRAM cell array

operation is provided, followed by descriptions of DRAM chip interfaces, characteristics of DRAM IC processes, memory redundancy, and "smart memories".

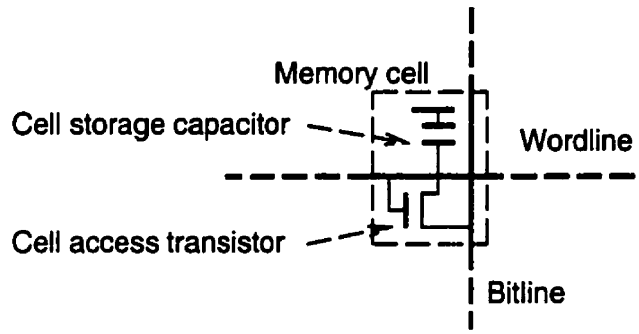

Figure 2.3: DRAM memory cell

Briefly, commercial dynamic memory is organized in 2-dimensional arrays of 1-bit memory cells (shown in figure 2.2). A memory chip contains multiple copies of these memory arrays. Row decoders are located at one side of the array, and sense amplifiers (sense amps) and column decoders are placed along the bottom. In 16 Mb and later generation DRAMs the column decoders and sense amps are often shared by multiple arrays in order to save area.

Figure 2.3 shows the components of a 1-transistor DRAM cell. Each cell is connected to a wordline and a bitline. Wordlines, which run in the row direction (also referred to as the X direction in the literature), select a row of cells. Bitlines run in the column direction (Y direction) and conduct the contents of selected cells to the sense amplifiers at the bottom of the array. Each bit is stored as a charge on a capacitor. A MOS *access transistor*, under the control of a wordline, selectively connects the capacitor to the bitline.

In order to reduce the energy used in a memory cycle, not all of the sense amps or associated memory arrays are active during any given cycle. The most significant bits of the row address determine which memory arrays and sense amps will be used in a cycle. DRAM specifications will quote the required number of refresh cycles in order to restore every memory cell in the chip. Since a sense amp can refresh exactly one memory cell, the number of active sense amps is $\frac{memory\ capacity}{refresh\ cycles}$.

While parameters within memory architectures keep changing, some typical ratios are provided. Wordlines can cross 4096 or more bitlines; column decoders can select 2 to 16 sense amps; the number of refresh cycles can range from 1 K to 16 K, and all are growing over time. Through multiple DRAM generations, however, a sense amp is typically connected to a pair of (active) bitlines, each with 128 memory cells.

2.2.1 DRAM Circuits

In this section, DRAM circuits as used in the high density C•RAM are first described. DRAM architectures vary widely, but many of the underlying circuits bear a strong resemblance to each other. Since the DRAM "platform" on which the high density C•RAM PE design is based is proprietary, we use the DRAM design by Gillingham et al. (described in [94]) as a reference (which is another MOSAID Technologies design incorporating some of the same design philosophies) and will be referred to as the *reference DRAM architecture*. Figure 2.4 is a schematic of one column of the memory array, with only four of the memory cells shown. A folded-bitline architecture is used, meaning a differential sense amplifier is connected to a folded bitline pair comprised of a *true* and a *complemented* bitline, which run parallel to each other and are matched in size and capacitance. Power (typically 3.3V or 5V, and decreasing with technological advancements) and ground are given the names V_{DD} and V_{SS} respectively.

The precharge, sensing, and memory access operations are now explained in detail, with explanations of what voltages are used. Before a row access, the memory is in the precharged state with bitlines precharged to $\frac{V_{DD}}{2}$, isolation transistors turned on with the *isolate* signal at V_{DD}, and sense amps and wordlines turned off.

When a row of memory is accessed the wordline[5] is driven to the boosted potential V_{PP}. The cell capacitor is now connected to and shares its charge with the bitline. Since the bitline capacitance (C_b) may be 10 times greater than the cell capacitance (C_c), the voltage of the signal is "diluted"

[5]The boosted wordline is needed later so that the cell capacitor can be restored to its full voltage. ($V_{PP} > V_{DD} + V_{NMOS\ threshold}$)

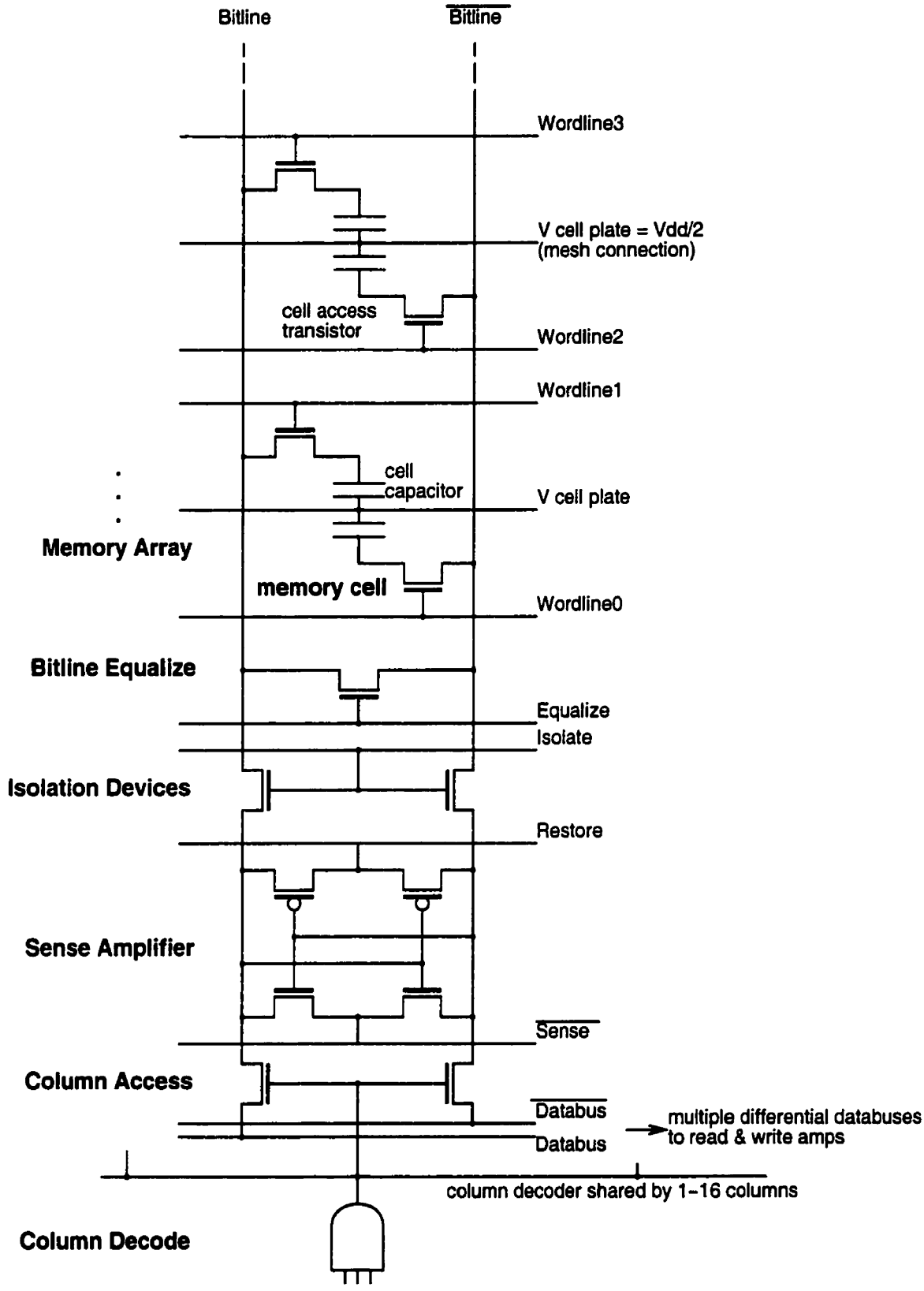

Figure 2.4: DRAM memory column

by $\frac{C_c}{C_c+C_b}$.

The sense amplifiers, along the bottom of the array, must then amplify the diluted signal, restoring the bitlines and memory cells to the full power supply potentials. The signals $\overline{sense}$ and *restore* are then asserted to power the sense amplifier. The sense amp draws the most supply current when the sense amp nodes are near their precharged voltages of $\frac{V_{DD}}{2}$. The isolation devices make sensing faster and reduce power consumption by initially limiting the sense amp current, leaving the sense amp to charge the bitlines. When there is a significant voltage differential on the sense amp nodes, the *isolate* signal is raised from V_{DD} to V_{PP} so that the bitlines can be restored fully. The column(s) selected for read or write is connected to the differential (two-wire) data bus where the sense amp signal is either detected by a read amplifier or overdriven by a write amplifier (both at the edge of the memory array, not shown on figure). The path from (to) the read and write amps is multiplexed further before reaching the chip IO pins buffers.

After the memory operation, the row decoder returns the active wordline to V_{SS} and the bitlines are precharged to $\frac{V_{DD}}{2}$. Since the restore operation leaves the bitlines at V_{DD} and V_{SS}, the bitline-equalize transistor alone can do most of this task. To allow for bitline capacitance mismatch, the databus is connected to a $\frac{V_{DD}}{2}$ supply, and all column decoders are enabled to pass the precharge voltage level.

2.2.2 DRAM Chip Interfaces

DRAM chips have far fewer data pins than the width of the internal datapath, so only a small proportion of the internal bandwidth (at the sense amps) can be made available to external consumers. Here, we will briefly describe DRAM memory interfaces designed to exploit more of the internal DRAM bandwidth by transferring more data during a memory cycle and at higher clock rates. The memory chips that incorporate these interfaces all use existing DRAM array structures and IC processes in order to remain price competitive.

The classic $\overline{RAS}$-$\overline{CAS}$ memory cycle permits a single memory word (e.g. 1, 4, or 16 bits) to be accessed for read, write, or both. This is quite restrictive when one considers that, in the process of accessing the one memory word, a much wider internal word (or *page*) of memory has been sensed and could be made available with a smaller latency than the original memory access. *Nibble-mode* DRAMs permit 4 consecutive bits (modulo 4) to be accessed in rapid succession during a memory cycle. *Fast page mode* adds the ability to randomly access "any" number of words of memory from within the current page during a memory cycle. *Static column mode* and *EDO* (Extended Data Out)

DRAMs offer random access within a page, as fast-page-mode DRAMs do, but have slightly higher data rates. Since most timing is relative to the edges of the control signals, these DRAMs are also known as asynchronous DRAMs.

DRAM data rates picked up considerably with the introduction of a clock and internal pipelining [101, 102]. *Rambus* DRAM (RDRAM) has a 500 MHz data rate, a request-response "packet" interface, and it "caches" multiple lines of data in its sense amps. Faster RDRAMs with an 800 MHz data rate are in development. *Synchronous* DRAMs (SDRAMs) have a slower 100 MHz clock, but a wider datapath (up to 18 pins). DRAM chips using the follow-on SLDRAM interface are in development. Even these fast DRAM interfaces, however, provide only a small fraction of the internal memory bandwidth.

Over time, the DRAM interface has evolved in order to increase memory bandwidth by increasing the external datapath width (number of data pins) from 1 bit to 18 or more bits and by clocking more data through those pins during a memory cycle, increasing from one access per memory cycle ($\geq$90 ns) to one access every 2 ns. Current DRAM interfaces are still bandwidth limited by the number of pins that can be placed on a chip and by the physical rate at which data can be clocked over a memory–host bus. The 1.25 ns RDRAM bus cycle is close to the physical limit of the round trip path delay for the length of the memory bus, and is therefore difficult to improve upon significantly. At this point, memory can only make significant performance increases by, for instance, increasing the number of pins, pushing up the data rate further still, or by placing the consumer of the memory bandwidth on chip, as we do with C•RAM.

2.2.3 DRAM IC Process

DRAM manufacturing is very sensitive to price and thus DRAM processes are optimized to make small efficient DRAM cells. After all, memory cells make up most of the chip. Each memory cell requires a capacitor and a transistor which will exhibit low leakage when turned off and will pass the full sense-amp voltage when turned on. In contrast, IC processes that are designed for digital logic (including microprocessors) require fast transistors and plenty of interconnection for signals, clocks, and power. Some features of the DRAM process, such as the small number of metal layers (1 or 2 layers in the 4 Mb DRAM generation as opposed to 3 - 5 metal layers for logic) and the transistor characteristics, make digital logic design in a DRAM process more difficult, require more area, and make the logic run slower.

DRAM IC processes offer special cell capacitor structures which minimize area by placing the

capacitor under the cell access transistor (trench capacitors) or above the access transistor (stacked capacitors). These DRAM-process capacitor structures can reduce the cell area by a factor of 7 when compared to planar capacitors in an ASIC process [124]. The height of the stacked capacitors can cause "depth of field" problems in the photo-lithography when placing multiple layers of finely spaced metal over both the cell array and other parts of the die.

The characteristics and operating conditions of transistors in DRAMs make them slower than the transistors in an equivalent ASIC or digital logic process. In order to maximize the DRAM refresh interval, transistor leakage currents must be controlled. This is done by increasing the transistor threshold voltage[6] and applying a negative bias to the substrate (back bias)[7]. In order to apply a boosted voltage (V_{PP}) to the wordlines, the gate oxide of the cell access transistors must be thicker[8]. Each of the increased threshold voltage, negative back bias, and thicker oxide diminish the drive of the transistors, and hence, reduce their speed when compared to transistors in ASIC processes.

In summary, DRAM processes have been optimized for the cost-sensitive and density-sensitive DRAM market. Building memory in a logic process results in a much less dense design. Modifying a DRAM process for a single IC design would be prohibitively expensive. Hence, making the best use of an existing DRAM process and DRAM array design is key to building low-cost high-performance computing. This is what C•RAM does.

Logic Design in a DRAM Process

DRAM processes typically have fewer wiring layers than logic processes, and consequently, the routing of signals in DRAM can be as demanding of the designer as is the design of DRAM circuits. In the 4 Mb and earlier generations, DRAMs have one pin each for V_{DD} and V_{SS}, (power and ground) and often have a single layer of metal. Since the other IC routing layers are resistive, this constrains V_{DD} and V_{SS} to be "river-routed" to all significant loads across the chip without these power lines crossing or using a via.

Another electrical concern is signal propagation along the wordlines, which in a 4 Mb DRAM for example, must pass through 4 K bitlines. Because the wordlines of polysilicon or polysilicide (the cell access transistor gate material) are too resistive, the signal is "backed-up" with metal

[6] In some DRAM processes, the threshold voltages of the cell access transistors and digital logic transistors are controlled selectively by introducing additional mask and implantation steps.

[7] For PMOS DRAM arrays [106] and for multiple-well processes, different back-biases for logic and memory cells become possible.

[8] One way to avoid this problem is to design a process with two layers of gate polysilicon with different gate oxide thicknesses in order to provide transistors for logic as well as for DRAM.

running above it. Each metal wordline is "strapped" to its polysilicon wordline at regular intervals, producing gaps in the memory array called *wordline strappings*. Control signals for the sense amps and column decoder are similarly strapped, and signals for any additional logic (such as control signals for C•RAM PEs) in the memory should be strapped as well.

DRAM design has several other prominent features that make it different to other digital logic design. For example, the small signals on the bitlines require more attention to precision and noise than the "rail-to-rail" signals typically found in digital logic circuits. DRAM timing is complicated and critical, so many of the internal control signals in DRAM are "self timed" to achieve the greatest speed while maintaining reliability. The power supplies within DRAM are noisy because multiple copies of circuits (e.g. sense amps) switch on simultaneously, generating spikes in current, yet DRAM packages have few power pins (2 to 6) compared to digital logic chips.

2.2.4 Redundancy

The use of redundant rows and columns in DRAM designs is necessary in order to increase manufacturing yield. DRAM designs can incorporate 2 orders of magnitude more transistors on a chip before yield begins to fall off, when compared to other silicon designs, such as microprocessors. For example, in 1995, the largest memory chips contained over one billion transistors, while the largest processor chips contained fewer than 10 million transistors. This difference is partly due to the regular structure of DRAM, but also due to the use of redundant structures. Having redundant columns (and/or rows) allows defective columns (rows) to be replaced. The disabling of defective columns (rows) and the activating of the address decoders for the redundant columns (rows) is typically done by cutting polysilicon fuses with a laser. The substitution of redundant columns (rows) is performed in small adjacent groups (e.g. of 4 rows or columns).

For high-yield, high-density memory designs, it is clearly important to have either column or row redundancy. In practice, it is useful to have both in order to be able to repair additional classes of faults, such as bitline to wordline shorts. The majority of published DRAM designs today have both.

In conventional redundancy schemes, a redundant group of rows or columns assumes the address range of the defective group it replaced. Since the redundant groups can't always be physically adjacent to the defective groups they replace — indeed, it is common design practice to place all redundant groups at the edge of the memory array — adjacent addresses will not map to physically adjacent groups. This discrepancy will become significant when PEs with interprocessor

communication are added to the memory array.

As a result of the use of redundancy to repair defects, DRAM processes typically have two sets of design rules: a larger-feature-size more-conservative set of rules that are used for the "periphery" and a smaller-feature-size set of rules that are used for the "core" or memory array. ASIC processes typically have one set of design rules. Transistors designed using core, as opposed to periphery, design rules are smaller, faster, and lower power, but have lower yields and hence need to be protected by redundancy. If PEs accompanying DRAM could be protected against defects through the use of redundant PEs then this logic could be designed using the core design rules.

2.2.5 Smart Memories

The term *Smart Memory* has been in existence for years and has been applied to a variety of designs. Until recently, none of these memories has offered an external memory interface while adding flexible computing. The majority of self-described smart memories in the literature are simply memories with an augmented memory capability, such as: content addressable memories, dual-ported memories, semaphore memories, etc. [128]. Some proposed memories can perform higher-level memory related operations such as pointer chasing [129]. Since little parallelism is available in implementing these operations and a conventional external processor would likely require the same number of memory cycles, there are limited advantages of adding this type of functionality to a memory chip. Steve Morton [130], founder of Oxford Computers, has produced several memories with dedicated logic for specific operations such as convolution and synthetic neural networks. None of these designs listed in this paragraph offer both a memory interface and a general processing capability.

It is interesting to note that processors can and have been added to other parts of the computer memory hierarchy. There is ongoing work on adding microprocessors to the disks to build smart disk arrays for database applications [131].

The idea of placing one or more processors on the memory chips has been proposed previously [6, 24]. Concurrent to our work, Patterson [132] mentions in 1990 "The most important processor of the 90's is going to be the IRAM (Intelligent RAM)" (meaning RAM with some kind of processing on-chip). His group has been working on the design of a vector processor in memory [133, 37]. Others have created single processor in memory designs [134].

Microprocessors Combined with Memories

Some argue the case for putting as many of the current generation high-performance microprocessors in a memory chip as possible. Since the highest performance processors and highest density memories are built as large as possible near the limits of the lithography and manufacturing yield, combining advanced microprocessors with high density memories requires compromises in area for both, in order to be economically viable. Consider, for example, the quad-issue PowerPC chip [135] with an area of 311 mm^2, and a 256 Mb DRAM[9] with an area of 304 mm^2. This high performance microprocessor is larger than the DRAM, even when the processor's external level-2 cache is discounted. Furthermore, the IC processes for microprocessors and DRAM are quite different as noted in section 2.2.3. While processors have been integrated into DRAM [56] and other microprocessor-in-memory designs will likely follow and be successful, compromises will probably have to be made in the memory capacity and/or processor complexity.

2.3 Summary

Several of the massively parallel processors mentioned in Section 2.1 integrate processors and memory on the same die (GAPP, VIP, Pixel-Planes, BLITZEN, Terasys, IMAP, Linear Processor Array). However, only a couple of these chips (EXECUBE and PIPRAM) use high density one-transistor DRAM. Only two (Terasys and IMAP) are usable as memories. To retrieve a value from a PE's memory in the GAPP, for instance, the value must be shifted out through a 12×6 grid on chip and very likely through a cascade of chips.

C•RAM was the first published implementation of memory with added flexible processing capabilities, supporting both full memory function plus programmable computation [7, 8].

[9]Since microprocessor and DRAM die sizes increase each generation, we selected a microprocessor generation (quad-issue) which will likely be in volume production before the DRAM (256 MB).

Chapter 3

Computing in the Memory: the Abstraction

The design of C•RAM was driven primarily by the desire to eliminate as much of the von Neumann bottle neck as possible in order to obtain an extremely high performance/price ratio. Section 3.1 identifies the memory bandwidth that is available in different parts of a computer system, and shows that there is an immense, but underutilized memory bandwidth at the sense amplifiers within semiconductor memories, such as DRAMs. The aggregate bandwidth at the sense amps is several orders of magnitude higher than what is available anywhere else in a typical computer system. The only way to fully exploit this bandwidth is to place the consumers of the bandwidth, namely the processor(s), into the memory chip adjacent to the sense amps. This observation was the primary motivation for designing C•RAM with processors integrated into the memory.

The second goal in the design of C•RAM was to keep the cost of C•RAM comparable to DRAM. In order to achieve this goal, C•RAM must be compatible with commodity DRAM with respect to density, packaging, redundancy, and IC process. Remaining compatible with DRAM allows C•RAM to leverage the manufacturing technology of DRAM and allows easier integration with existing computer system architectures. In particular, C•RAM should be viewed as a replacement for DRAM.

The requirement for DRAM compatibility and the desire for placing processing power adjacent to the sense amps naturally results in an architecture with many small PEs with narrow pitch that operate in a SIMD fashion. This is established in section 3.2. Section 3.3 establishes that the best processor configuration for C•RAM is an array of 1-bit-wide PEs with narrow aspect ratios. The

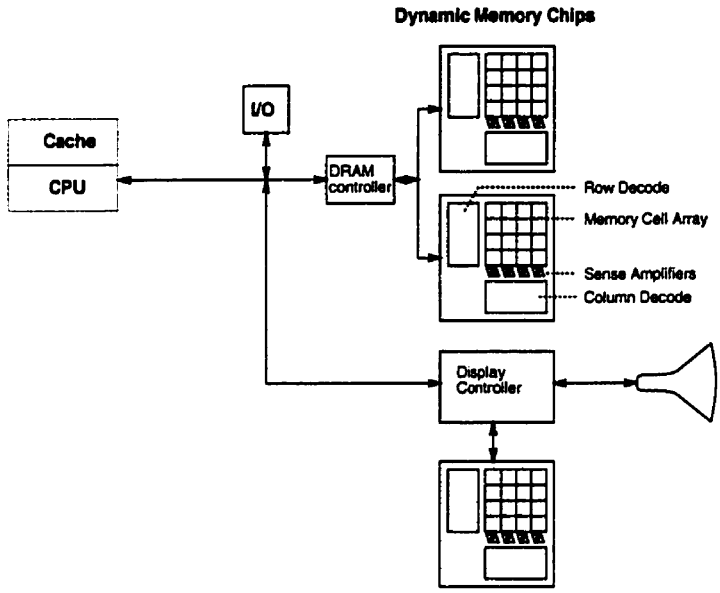

Figure 3.1: Conventional workstation architecture

resulting C•RAM architecture is described in section 3.4 and the advantages of this architecture are discussed in section 3.5. Chapter 4 will cover more detailed architectural tradeoffs and design.

3.1 Memory Bandwidth

In a typical workstation, the memory bandwidth can vary from a few hundred megabytes/s at the system bus, to several terabytes/s inside the memory chips themselves.

Consider, for example, a 100 MHz 64-bit workstation with 256 Mbytes of 16 Mb DRAM, as shown in figure 3.1. In this example, we use 1 M×16b DRAM chips[1] with a row access time of 50 ns, a cycle time of 90 ns, and a fast page mode cycle time of 35 ns. This chip requires 1024 cycles to refresh every memory cell, which implies that there is one active sense amplifier per 1024 memory cells. Thus, the width of the internal datapath in one chip is 16 Kb. In this 256 Mbyte system with 128 memory chips, the internal datapath is 2 Mb wide, with an aggregate memory bandwidth of 2.9 terabytes/s for reads (as shown in the top bar of figure 3.2) and an aggregate memory bandwidth of 2.9 terabytes/s for writes (5.8 TB/s total memory bandwidth).

As the 2 million wire datapath is multiplexed in multiple stages onto a 64-bit bus, memory bandwidth is lost. The first such constriction of bandwidth occurs inside the memory chips when

[1]The NEC μPD4218160LE-50 1 M×16b DRAM [113] is used in this example.

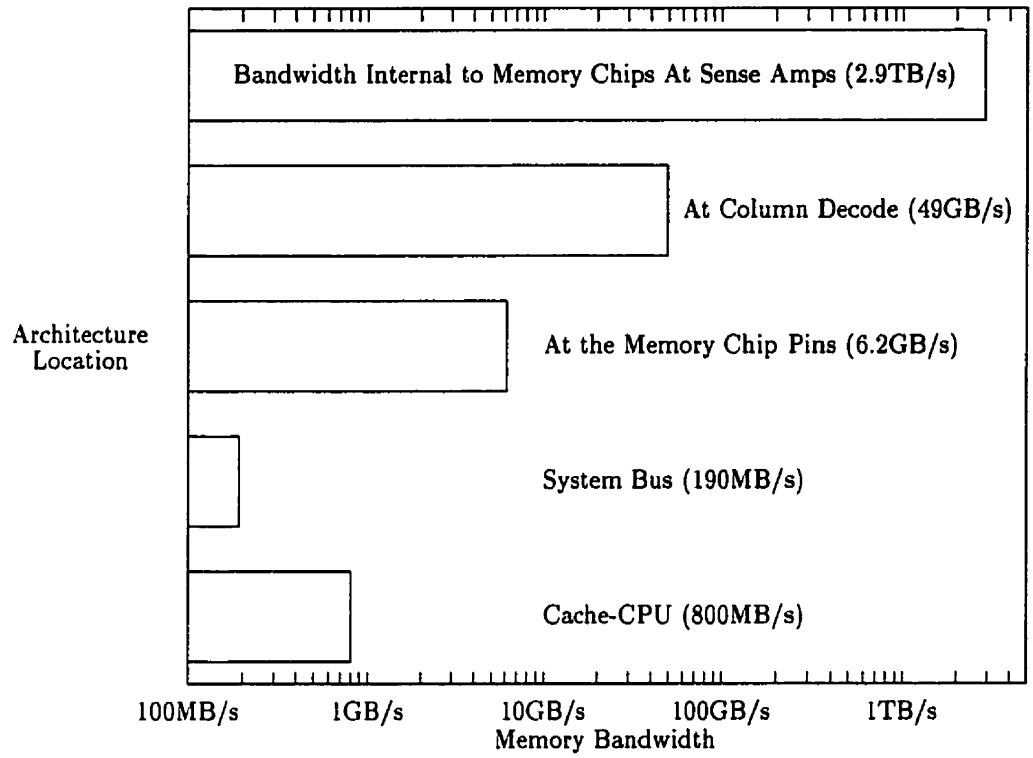

Figure 3.2: Memory bandwidth throughout system

the memory column decoders multiplex the datapath down to 128 bits per chip[2], resulting in an aggregate system bandwidth of 23 gigabytes/s using the standard DRAM cycle. If 4 fast-page-mode accesses are made in 166ns[3], the aggregate bandwidth would be 49 gigabytes/s. Despite the lower bandwidth at the column decoders as opposed to the sense amps, many other designs attach processors to DRAM after the column decoders[4].

If, however, memory is only accessed from off-chip, the 16 data pins per chip (2 K pins total) limit the aggregate bandwidth to 2.8 gigabytes/s using the standard DRAM cycle, or 6.2 gigabytes/s in fast page mode transferring 4-word bursts between row accesses. When the memory is read or written over a 64-bit system bus, the bandwidth is further constricted to 190 megabytes/s when accessed in the fast-page-mode.

[2]The 128-bit datapath after column decode assumes a memory organization of 32 arrays of 2 K×256 bits with 8 arrays active and a 16-bit databus per array.

[3]The 4-word burst transfers are reasonable for a cache line fill or write-back.

[4]Most existing designs that incorporate processing into DRAM treat the DRAM (including it's column decode and narrow data bus) as a macro-cell [56, 136, 137, 138, 53, 37], thereby using the bandwidth after the column decoders which is only a fraction of the bandwidth available at the sense amps.

If a cache hierarchy is used then the effective peak (cache access) memory-data bandwidth is 800 megabytes/s which is three and a half orders of magnitude less than the untapped bandwidth available at the sense amps. For applications or routines with poor data temporal locality, such as the inner product of long vectors, the system bus dictates performance, with four orders of magnitude less bandwidth than at the sense amps.

The bandwidths at different points in the memory system architecture will, of course, vary from system to system. The numbers presented above were based on a common DRAM chip that requires only 1 K refresh cycles (in order to refresh all memory cells on the chip), implying a greater number of active sense amplifiers and thus greater internal memory bandwidth (and higher power) than DRAMs that require more refresh cycles. For the other parameters of this hypothetical system, numbers were chosen conservatively, using parameters which decrease the ratio of memory bandwidth at the sense amplifiers to bandwidth elsewhere. For example, fast page mode more than doubles the bandwidth at the decode, pins, and bus compared with operation without fast page mode. Moreover, the bandwidth achieved with the 4-word burst transfer we assumed is within 19% of the peak bandwidth that would be achieved had all memory accesses been made continuously to the same memory page. The DRAM we chose had a wide data bus (16 pins), with a correspondingly high bandwidth at the memory pins. Circuit board signal delays and skews were optimistically assumed to be zero. The CPU data bandwidth is based on one data access every cycle with an optimistic 100% data cache hit rate. Higher-clock-rate DRAM chip interfaces have improved the bandwidth characteristics, but only slightly. For example, Rambus DRAMs with a single controller have a peak transfer rate of 500 MB/s at the "system bus" or 1/6000 of the internal memory bandwidth.

Other parameters in our calculations were chosen to be typical but could be chosen differently. For example, adding additional DRAM chips would increase the bandwidth at the sense amps, decode and pins; a higher density memory would offer fewer total data pins in the system with proportionately lower bandwidth at the pins; a higher density memory may also have fewer active sense amplifiers with lower bandwidth at the sense amps and decode; using slower DRAM chips would result in lower bandwidth at the sense amps, decode, pins, and bus; and interleaving would result in improved bandwidth at the system bus. As we have discussed, different assumptions will alter the bandwidth ratios, but not enough to change the conclusion, namely that *there is significantly greater memory bandwidth available inside the memory chips.*

Of course it is well understood that a designer, for a price, can *buy* any desired amount of memory

bandwidth by using more or faster memory components[5], without having to integrate processors and memory. The numbers of chips needed to obtain the high memory bandwidths off-chip, however, would require a great deal of space and would be expensive. Moreover, with more memory chips, the aggregate on-chip memory bandwidth would also be correspondingly higher.

From the example provided above, it is clear that, for memory-bandwidth limited computations, the processors should be placed next to the sense amps. The increased memory bandwidth could result in a potential speed-up, relative to a workstation, of 3000 to 15,000 depending on the workstation's cache performance on the particular application.

3.2 Maintaining Compatibility with Memory

A key goal of C•RAM is to remain compatible with commodity DRAM with respect to cost, silicon area, speed, packaging and IC process, while still being able to access a significant fraction of the internal memory bandwidth. To preserve the economics of the memory, we must work within the existing limited number of routing layers, preserve the number of memory cells per row address decoder and sense amp, and use redundancy to correct manufacturing defects.

3.2.1 Using DRAM IC Processes

In order to take advantage of volume DRAM fabrication, the IC process should not be changed. In particular, DRAM designers put great effort into designing the most economical DRAM cell. As described in section 2.2.3, we have to design with transistors and IC layers which have been optimized for DRAM. There are fewer metal layers available in a DRAM process relative to logic processes, and the transistors are slower.

3.2.2 Physical Dimensions and Performance

C•RAM should be a viable substitute for DRAM. If C•RAM is to be used as enhanced main memory, the price should not be significantly higher than DRAM and C•RAM should not be noticeably slower than DRAM. Hence, the increase to the area of the memory chips as a result of adding PEs should be kept small. While this guideline is vague, we show in chapter 5 that significant processing power can be fit into DRAM with an area increase of less than 20%.

[5] In conventional designs, more memory bandwidth can be obtained by using more memory components arranged in parallel or interleaved, as required.

Most of the chip area of C•RAM should be taken by the memory arrays, as is already the case for DRAM. Since the DRAM array is already highly optimized for area, cost, and yield, the layout of the basic DRAM array should not be changed for C•RAM. The processing capability must therefore be placed along the perimeter of the DRAM array. In fact, the goal of exploiting the memory bandwidth at the sense amps indicates that the processing capability be placed at the "bottom" of the array where the sense amps are located with connections made to the array at regular intervals.

Likewise, if C•RAM is noticeably slower than DRAM, it won't be attractive to use as an enhanced replacement for DRAM. An increase in chip area will likely result in an insignificant increase in signal delays over the longer wires resulting in marginally slower memory accesses. The same circuits as found in DRAM, or circuits with equivalent speed, should be used for the performance-critical paths.

3.2.3 Coping with Long Wordlines

Architectures which place processors at the memory sense amps are constrained by the long wordlines found in DRAM designs, which effectively dictate a SIMD architecture with uniform addressing. The wordlines are designed to be long in order to minimize the area taken by row decoders and other drivers. In standard DRAM designs, all columns (or bitline pairs) in an array are addressed by a single row address. If the row decode and long wordlines found in commodity DRAM memories are not to be modified, then the C•RAM processors are potentially presented with data from typically 2048 to 4096 columns of memory, all from the same row. This 2048-bit, or more, memory word is too wide to use effectively as a processor word (in any conventional sense). Hence, it makes much more sense to use multiple, smaller processing elements. The DRAM's wide memory word can be divided among multiple processors with smaller datapaths, each being able to access data only in the columns above it. Thus, each processor effectively sees a private memory as in a distributed memory multiprocessor. The long wordlines require that each processor access the same memory row in any given cycle.

The PEs thus have uniform memory addressing. To allow smaller blocks of memory to be separately addressed would require more row decoders with an additional overhead in area. Since row decoders can be greater than 300 times the width of a DRAM memory cell, adding extra row decoders is an expensive option.

It would be unworkable for a MIMD multiprocessor to benefit from having processors autonomously executing their own instruction streams but performing loads, stores and instruction

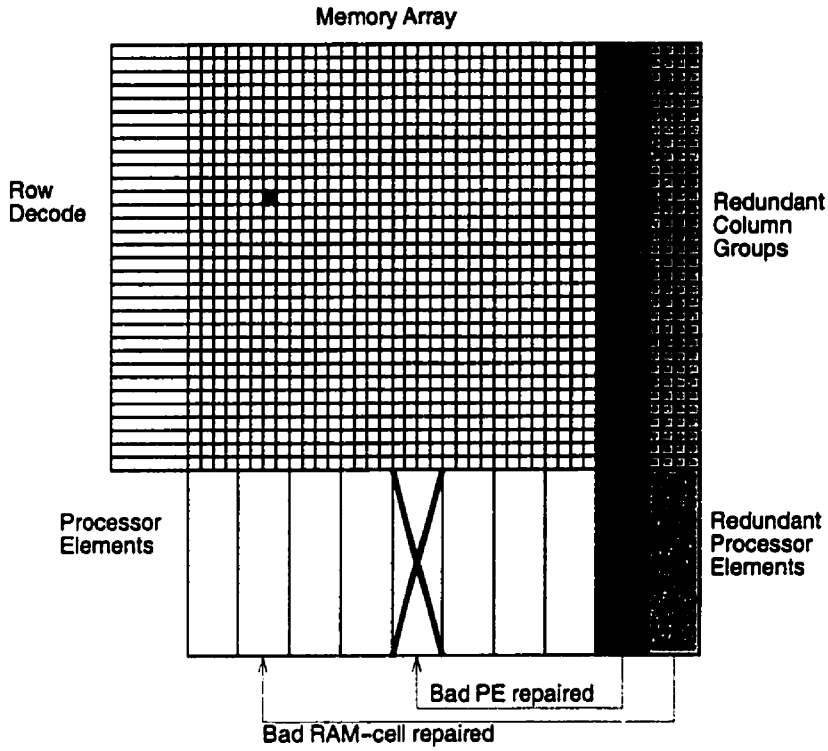

Figure 3.3: Redundant memory columns and PEs
Defective memory cells and PEs are effectively repaired by electrically replacing them with redundant units which contain a group of memory columns and a PE.

fetches to the same addresses in local memory at the same time. The shared memory address stream thus suggests the use of a shared instruction stream as well.

In summary, in keeping with existing DRAM architectures, we are constrained to use multiple processing elements located close to the DRAM sense amps that operate in a SIMD fashion using uniform memory addressing.

3.2.4 Redundancy Considerations

Providing redundancy in DRAM increases manufacturing yields and reduces manufacturing cost per working chip. In C•RAM, row redundancy is easy to implement since it has no impact on PE design. The presence of column redundancy, however, does complicate the design of C•RAM, since each PE gets its *identity* both from the column address of its memory and from the PEs adjacent to it on a communications network. When a defective group of memory columns is replaced, the redundant group of memory columns must be connected to a PE. Rather than connecting a redundant group of columns to a PE through a switch and wires which may run the width of the memory array,

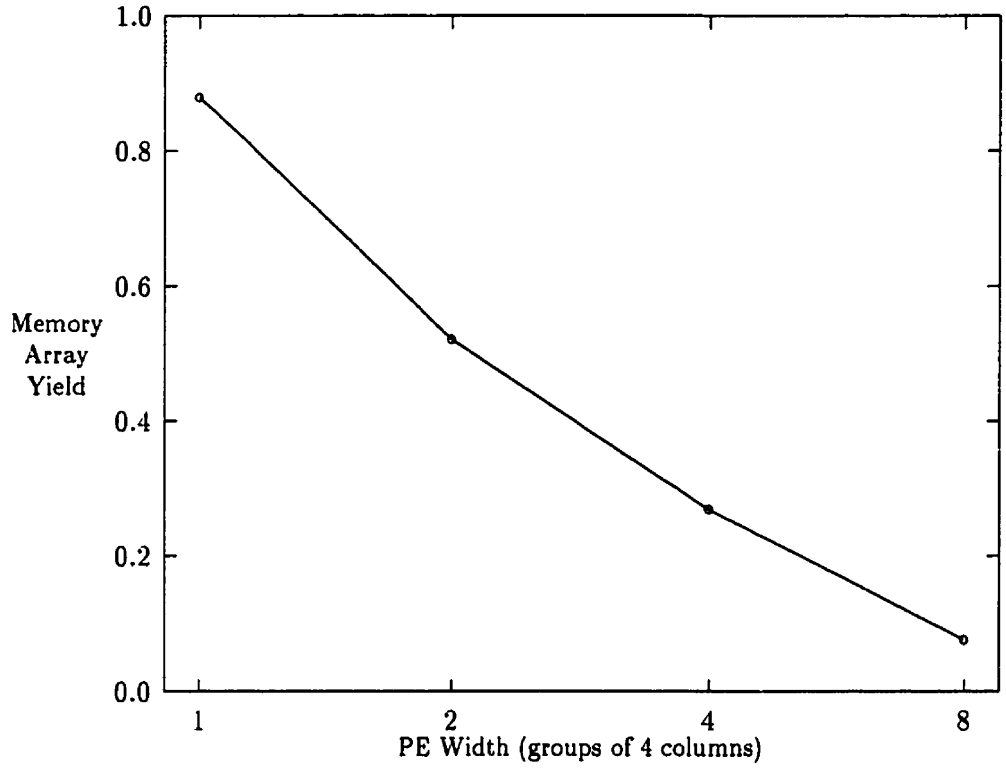

Figure 3.4: Yield as a function of PE width

the signal delay is reduced if the redundant group of memory columns is directly connected to a redundant PE, as shown in figure 3.3. In the figure, a faulty memory cell and faulty PE are depicted with crosses. To employ the redundant column groups, the column groups containing the faulty circuit elements are disabled and the redundant column groups (shaded) are set to the address of the defective column groups. The addition of the redundant PEs further improves yield by also allowing the replacement of defective PEs. Adding redundant PEs thus allows PEs to be designed using lower yielding and finer "core" IC design rules, resulting in a PE with a smaller area and faster transistors.

A narrow PE pitch can assist in making memory column redundancy more effective. In fact, the narrower the PE, the less expensive it is to produce the chip. If, when a PE or memory cell is defective, all memory columns associated with that PE are to be replaced as a unit, then a wider PE will result in fewer substitutable groups of columns from a fixed-size pool of redundant columns. This effect is depicted in the graph in figure 3.4 that shows the yield of an array of memory as a function of the width of a PE, given a fixed-size pool of redundant columns. The yield decreases with increasing PE width as larger numbers of redundant columns from the fixed size pool must be

attached to each redundant PE. For the case where the PE is 8 times the width of a group of columns, there are not enough redundant columns to repair a single defect[6].

The highest yield in graph 3.4 corresponds to the pitch of the PE being no wider than the pitch of the minimum size group of columns which can be substituted.

3.2.5 Packaging Considerations

DRAM can be packaged very densely. For example, SIMMs (Single Inline Memory Modules) and DIMMs (Dual Inline Memory Modules) are small cards that plug vertically into the main circuit board and hold, for example 8 or 9 memory chips[7]. When the DRAM IC packages are mounted on SIMMs which are then mounted on a main printed circuit board, the memory chips are on their edges, taking up even less area. This dense SIMM packaging is possible because memory has small IC packages, uniform bus connections, and a small number of pins. In contrast, some massively parallel processors are packaged in bulky pin grid array IC packages. For example, the 208 pin PGA (Pin Grid Array) used by the IMAP requires 19.8 square centimetres, or 13 times the area of a typical 4-Mb DRAM package[8].

Relative to DRAM, C•RAM requires a few extra pins per chip for communication and possibly a slightly larger package to accommodate a larger die. Strict pin compatibility between DRAM and C•RAM is not an important issue since we assume that C•RAM will be designed into a C•RAM-specific board or system which incorporates a C•RAM controller. A typical 4-Mb DRAM (our reference 1 M × 4 b memory architecture in sections 2.2.1 and 5.2.1) uses 20 pins on a TSOP (Thin Small Outline Package) which could accommodate 26 pins, so the unused 6 pin-positions could be used for C•RAM specific signals. Alternatively, a package with more pins could be used. The widths of standard TSOP and SOJ (Small-Outline J-lead) packages are specified in 0.05 inch increments so a wider die can be accommodated in a slightly wider package. Eight C•RAM chips with widths 0.05 inches wider than typical DRAMs will easily fit on a standard sized 72-pin SIMM, but the SIMM pin-outs must be changed. The other constraint for using SIMMs is the pin count

[6]The assumptions used in this graph are: the smallest group of columns which can be substituted is 4 columns, the memory block contains 1024 columns plus 16 spares, a defect rate per column of 1% is arbitrarily chosen, defects are uncorrelated and modeled with a binomial distribution. The defect rate of 1% (yield of 99%) for individual columns is for demonstration purposes. Actual column yields of DRAMs can be higher or lower than this. For these calculations, the column yield is taken after row redundancy (if any) has been applied. The 4 column group corresponds to the pitch of some of the more narrow column decoders. The size of a group of redundant columns cannot be less than the column decoder controlling it.

[7]Common SIMMs range in size from holding 2 to 24 chips.

[8]The 26/20 pin 300 mil SOJ (Small-Outline J-lead) 4 Mb DRAM package has an area of 1.47 square centimetres.

of the SIMMs themselves. The 160 IC pins of 8 DRAM chips can be connected to the SIMM's 72 pins because most signals are bused to each chip. Similarly, most of the wires connecting the many SIMMs form buses on the circuit board that the SIMMs plug in to.

In order to package C•RAM dies in TSOPs, C•RAM must have few extra pins compared to DRAM. In order to package C•RAM chips together on SIMMs, which are also pin-constrained, most signals must be common buses, which is the case for C•RAM. A SIMD architecture helps reduce the number of signals since all chips use the same instruction and address. Multiplexing the SIMD instruction with the memory address can economize on pins. Chip and SIMM pin count should also be considered when choosing a communications network. Dense memory-style packaging for C•RAM is desirable in order to minimize cost, but it also has a large impact on system size[9].

3.3 Additional Design Issues

Section 3.2 established that it is beneficial to use many PEs with narrow pitch, located at the bottom of the memory arrays, next to the sense amps. For the design of C•RAM, we chose to make our PEs 1-bit wide (bit serial). Section 3.3.1 argues that this is also a good choice for efficient use of silicon from an asymptotic point of view. Given 1-bit wide PEs, section 3.3.2 predicts the optimal PE aspect ratio.

3.3.1 Asymptotic Advantages of Bit-Serial PEs

Assuming an abundance of parallelism in applications, simple ALU structures can have greater performance per silicon area compared to the complex ALU structures found in conventional processors because simple ALU structures can be more efficient in the use of transistors and circuit area. In a bit-serial ALU, for instance, every bit of every output of every stage of logic is active during every ALU cycle — there is only one stage with one output bit. The designers of modern high performance microprocessors make a tradeoff and choose complex ALUs with low latency over simpler ALUs with reduced area and transistor count.

To compare several ALU algorithms without looking at specific silicon implementations, we look at their asymptotic complexity as functions of operand size in bits (n) with $O()$ notation in

[9]The advantages of this dense packaging are significant; 1 million 1-bit-datapath C•RAM PEs could fit in 25×23cm of circuit board space in present technology if packaged as SIMMs. For comparison of packaging, the Thinking Machines CM-1 and CM-2 (introduced in 1985 and 1990 respectively) mounted 32 1-bit-datapath PEs on a module (printed circuit board). The maximum configuration used 2048 circuit boards to hold 64 K PEs; 1 M PEs would have required 32 K circuit boards.

Hardware Algorithm	datapath width	transistors	area	time	area time product
Addition					
bit serial	1	$O(1)$	$O(1)$	$O(n)$	$O(n)$
ripple carry	n	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
carry save	n	$O(n)$	$O(n)$	$O(1)$	$O(n)$
carry save, binary out	n	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
carry look-ahead	n	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(n^2 \log n)$
Multiplication					
bit serial	1	$O(1)$	$O(1)$	$O(n^2)$	$O(n^2)$
ripple carry	n	$O(n)$	$O(n)$	$O(n^2)$	$O(n^3)$
carry save	n	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
Wallace tree	n	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(n^2 \log n)$
Bit permutation					
bit serial	1	$O(1)$	$O(1)$	$O(n)$	$O(n)$
bit-wise	n	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
memory look-up	n	$O(n2^n)$	$O(n2^n)$	$O(1)$	$O(n2^n)$
Batcher banyan	n	$O(n \log n)$	$O(n^2)$	$O(\log n)$	$O(n^2 \log n)$

Table 3.1: ALU algorithm and complexity

The area and delay are compared for different hardware algorithms applied to addition, multiplication, and bit permutation. Bit-serial arithmetic is unbeaten in area·time complexity.

table 3.1. We assume that a sufficiently large memory for these operations is available and that this memory's latency is independent of n. Our asymptotic analysis of algorithms ignores constant coefficients and lower order terms, which are not necessarily insignificant for common 8-bit to 64-bit data-types. While table 3.1 examines the latency of the algorithms considered, we should note that pipelining can be used to increase the throughput of multi-bit multipliers (and occasionally adders) by (at most) the number of pipeline stages. Our measure of performance is the reciprocal of latency for an n bit operation, and we use silicon area as an estimate of the cost. Hence, the area·time product is an estimate of the cost/performance ratio, which we wish to minimize.

For addition, the bit-serial ALU with a one-bit datapath requires a constant number of transistors and area, regardless of the size of the operands, but has a long latency that is linear in the length of the operands. Nevertheless, it fairs very well as one of the two algorithms with the lowest order area·time product of $O(n)$. Ripple-carry addition uses an n-bit datapath, but is not faster asymptotically than the bit-serial addition with $O(n)$ latency.

Galloway proposed using carry-save addition for C•RAM [139]. Similarly, SILT is designed

to use carry-save arithmetic, storing one bit of a value per PE. Carry-save addition postpones propagation of the carries to reduce latency. Without the carry propagation, the latency for addition is $O(1)$ and the area·time product is $O(n)$. The result is provided in the redundant carry-save notation, which is suitable as an operand for another addition. When a *binary* representation result is required, any carry which does not increase the order of the hardware cost (e.g. ripple carry) makes carry-save addition just as slow as the ripple-carry adder.

A carry look-ahead ALU is designed for low latency but at a greater hardware cost than the previous ALUs. It can produce a binary result in $O(\log n)$ time. However, the carry-propagation path requires $O(n^2)$ transistors to achieve the fastest carry propagation of these adders. Thus the low latency of the carry-look-ahead adder comes at the cost of an even greater area, resulting in the worst area·time product [140]. Other low-latency adders, such as logarithmic carry select [141] have similarly large asymptotic bounds on area·time product. In our opinion, if the parallelism is available, silicon area is better spent on having more copies of simpler ALU structures such as ALUs which perform bit-serial or carry-save arithmetic vs. fewer large low-latency ALUs.

Considering multiplication with n-bit operands, the bit-serial and carry-save ALUs again have the best area·time product. The ripple-carry ALU is less competitive. For all three of these ALUs, multiplication is performed as iterative addition. A Wallace-tree [142] combinational multiplier includes a tree of carry-save adders. In contrast to the previous three ALUs, this hardware is poorly utilized as each gate in the Wallace-tree of combinational logic is used only once per new operand. The Wallace tree has the lowest latency, but a high cost in area.

The implementation of the ALU will determine how much advantage can be had by *skipping over zeroes* of the multiplicands [143]. This skipping-over-zeroes optimization does not change the order of the complexity. A hard-wired combinational multiplier must support the worst case operands. If the multiplication is performed iteratively, those partial products of zeroes can be skipped[10]. For the n-bit carry-save ALU, skipping over a zero still requires a shift operation while the bit-serial ALU does not require computation. The bit-serial ALU is most efficient at "skipping over zeros" in one of the multiplicands.

For some applications such as graphics and logic simulation, individual bits must be manipulated. Manipulating bits on a processor with a word wider than one bit (e.g. 32 bits or 64 bits) can be inefficient. As an example of bit manipulation, consider arbitrary bit permutation. Both a bit-serial processor and an n-bit-word processor can perform an arbitrary reordering of the bits in an n-bit

[10] In order to skip over zeros on a SIMD machine, one multiplicand should be a scalar variable.

word *one bit at a time* but the bit-serial processor can do it with significantly less hardware. The n-bit processor can perform a fixed permutation faster using a memory lookup table, but only at great hardware expense. While a dedicated permutation switch (such as a Batcher banyan network) is less expensive, it still has an area·time product of $O(n^2 \log n)$. Hence we conclude that the bit-serial processor is the most effective for performing an arbitrary bit permutation of an n-bit word. [11]

We have shown that for addition, multiplication, and bit permutation, the bit-serial ALU has the lowest order area·time product which corresponds to the best performance/cost ratio.

Bit-serial arithmetic has other advantages and disadvantages. With bit-serial arithmetic, any sized operand can be handled efficiently on the same hardware, including different-sized operands to the same operator. Results from addition and multiplication can be sized to maintain full accuracy. In contrast to the low latency ALU designs, the same bit-serial hardware is used for addition, multiplication, and any other operation. One downside to bit-serial arithmetic, however, is that an $O(1)$ PE cannot store an n-bit partial sum during a multiplication operation, so more memory operations are required when compared to an n-bit wide multiplier.

Overall, because of the advantages stated here and in section 2.2.3 and 3.2.4, we only consider bit-serial ALU designs for our C•RAM design.

3.3.2 PE Aspect Ratio

We have already shown in section 3.2.4 that narrow PEs have an advantage with regards to yield. The PE aspect ratio (height to width) also affects how many PEs can be placed on a die of a given size.

We developed a simple analytic model which can be used to represent the approximate relationship between PE pitch and area without time-consuming manual layout of the large set of alternatives. This model bases the PE area on a fixed area ($A_{PE\_circuit}$) for the PE circuitry, a fixed width for horizontal (H_{bus}) and vertical (W_{bus}) wiring channels, and a fixed memory height (H_{memory}) as depicted in figure 3.5.

The relationships between the height, width, and area for the PE and memory shown in figure 3.5 are described in these equations:

$$H_{PE} = H_{bus} + H_{PE\_circuit}$$

[11] Of course each PE in a SIMD bit-serial machine with uniform memory addressing must perform the *same* bit permutation.

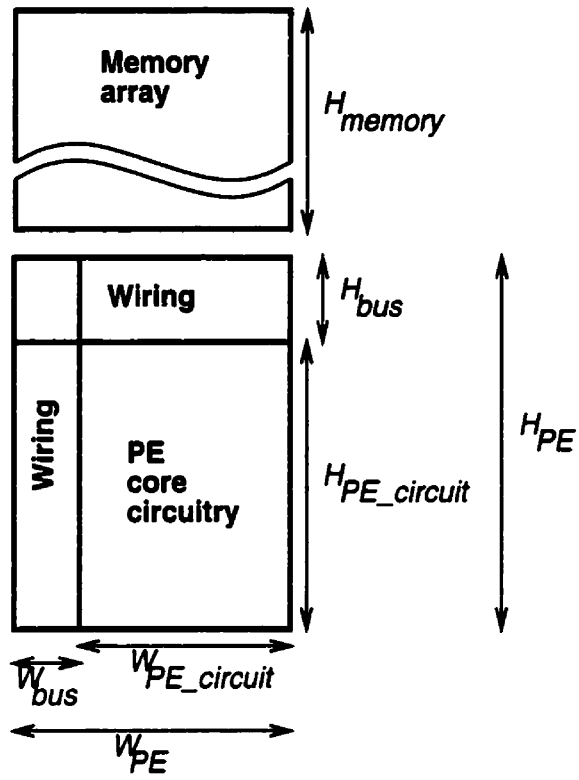

Figure 3.5: PE dimensions

$$
\begin{aligned}
W_{PE} &= W_{bus} + W_{PE\_circuit} \\
A_{PE} &= H_{PE} W_{PE} \\
A_{PE} &= (H_{bus} + H_{PE\_circuit})(W_{bus} + W_{PE\_circuit}) \\
A_{PE\_circuit} &= H_{PE\_circuit} W_{PE\_circuit} \\
A_{memory} &= H_{memory} W_{PE} \\
A_{PE} &= W_{PE}\left(\frac{A_{PE\_circuit}}{W_{PE} - W_{bus}} + H_{bus}\right) \\
A_{combined} &= A_{PE} + A_{memory}
\end{aligned}
$$

We made several simplifying assumptions. In actual IC layout, signal busses can be routed over-top of transistor circuits to varying degrees, but we discount this in the model. As well, we assume that area is a continuous function of pitch, instead of an integer multiple of sense-amp pitches (preferably a power of 2). We also assume there is no minimum memory requirement per PE.

In order to be independent of specific IC design rules, widths and heights are specified in wire

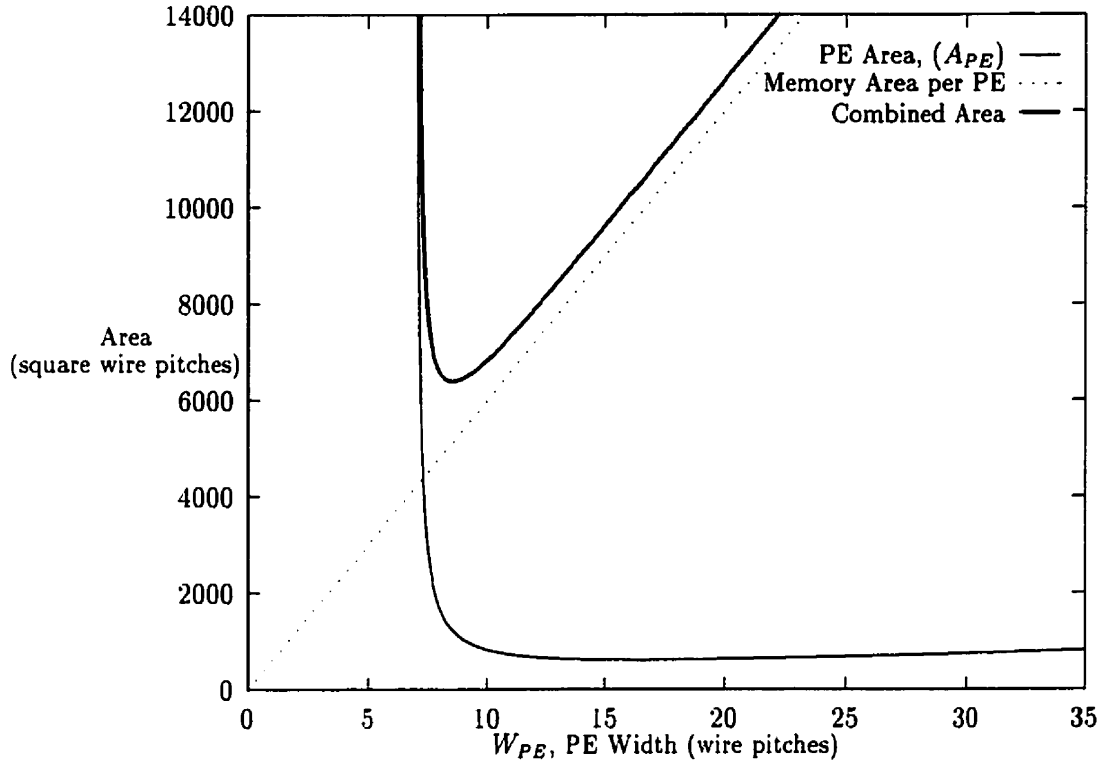

Figure 3.6: The effect of PE pitch on chip area per PE

pitches[12], and area is specified in square wire pitches. Remember that a column of memory (folded bitline pair) requires 2 wire pitches.

Rather than pick arbitrary parameters for this model, we take parameters from the circuit and circuit topology information for the PE described in section 5.2, but not the layout information. The DRAM-based C•RAM PE, described in section 5.2 requires a minimum of 7 wires running vertically at the widest point. A bus of 16 control signals runs through the PE in the horizontal direction. The remaining PE circuitry, including 76 transistors, is estimated to occupy 200 square wire pitches. We assume that there are 2 memory arrays attached to each PE, one above and one below. The combined height of the two memory arrays (H_{memory}) is approximated as 600 wire pitches, since each of the memory arrays above and below the PEs requires 256 wordlines plus an assumed 44 wire pitches for the sense amp and decode. Hence, we arrive at these parameters:

$A_{PE\_circuit} = 200$, area of PE excluding buses

[12]The wire pitches in the horizontal and vertical directions may indeed be different sizes owing to using different process layers.

$W_{bus} = 7$, signals running vertically contributing to PE width

$H_{bus} = 16$, signals running horizontally contributing to PE height

$H_{memory} = 600$, height of memory array associated with PE

The area-pitch relationship is shown in the graph in figure 3.6. We found that a wide range of reasonable parameters will produce similarly shaped curves. The main feature of the PE plus memory area ($A_{PE} + A_{memory}$) curve is a steep minimum where the PE is tall and narrow. The total PE area (circuitry plus buses) becomes infinite as the PE core circuitry is extended infinitely high, which corresponds to the total PE width going to W_{bus}. The total PE area also becomes infinite as the PE core circuitry is extended infinitely wide, which corresponds to the total PE width becoming infinite. For the given parameters, the PE area has a minimum when the PE pitch is 16.4 wire pitches and the aspect ratio (height:width) is 3.3:1. When the PE and its associated memory are treated as a combined unit, the minimum area produced by this model corresponds to a PE width of 8.5 wire pitches and an aspect ratio of 17:1. This narrower PE aspect ratio has a 40% greater area than the 3.3:1 PE aspect ratio, but clearly requires less total area when memory is considered. The goal in designing the C•RAM PE is to maximize the number of PEs (of a given complexity) that fit in a chip with fixed dimensions. This is equivalent to minimizing the area for a single PE and its associated memory. The optimum width is close to the chosen width of 8 bitline pitches (4 sense-amp pitches) used for the DRAM-based C•RAM PE described in section 5.2.

In summary, we have presented a model to help visualize the height-width trade-offs and this model suggests implementing the C•RAM PE circuit with a tall narrow layout.

3.4 Basic C•RAM Architecture

In the previous sections, it has been argued that our desire to exploit the memory bandwidth that is available at the sense amps requires us to move the processors into the memory chips, and our desire to keep the new memory-processor chips as compatible with memory chips as possible require us to use many narrow 1-bit wide SIMD PEs with uniform memory addressing. This leads to the computer architecture, depicted in figure 3.7. As shown, these C•RAM chips could replace DRAM chips in the graphics frame buffer and in the computer main memory. In the conventional workstation of figure 3.1, the DRAM controller was responsible for refreshing memory and relaying memory requests from the CPU. The DRAM controller would have to be replaced by a C•RAM controller that has the additional task of accepting requests for parallel tasks to be executed from the

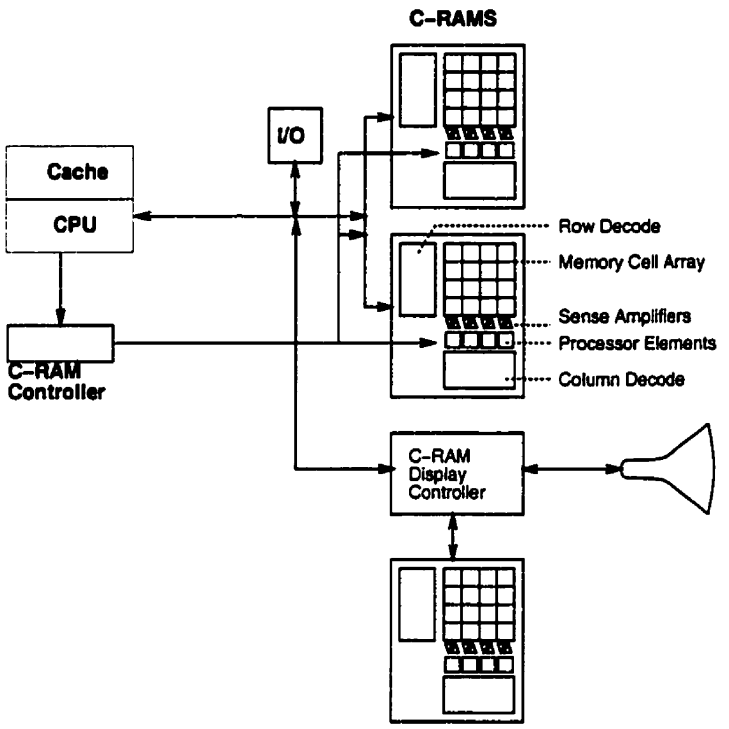

Figure 3.7: C•RAM computer architecture

PEs are added at the sense amps and the DRAM controller is replaced with a C•RAM controller.

host processor (CPU) and issuing SIMD instructions to the C•RAMs.

3.5 Advantages of C•RAM Architecture

The basic C•RAM architecture described in section 3.4 has numerous advantages other SIMD architectures. Some of them are described in this section. We defer detailed performance evaluation to chapter 6, where we we will show performance improvements in suitable parallel applications by factors of more than 1000 for a C•RAM-based workstation when compared to a workstation without C•RAM, with total component costs (assuming volume production) less than twice as high.

3.5.1 System Cost

We consider the economic cost of an entire computer system, assuming it is produced in quantity. While silicon area is a prominent contributor to cost, other aspects of the total system cost, such as packaging, pin counts, power, and the complexity of the IC process are also important. In a

massively parallel processor (MPP), combining processors and memory saves on pins that would otherwise be required to move data between the memory chips and the PE chips. When PEs are placed in the memory, the number of PEs which can be placed on a chip is not limited by the number of pins. The short datapath between PE and memory further reduces the cost of components along that datapath.

Most of the transistors in a desk-top or larger computer are in the memory (e.g. >90% in a typical computer with 64 MB of DRAM). There is a cost savings in being able to use the same memory cells for two purposes, namely as main memory and as local memory for the SIMD PEs. The goal of low cost brings with it constraints on chip area, pin-count, usability as main memory, and compatibility with existing DRAM processes.

3.5.2 Performance

Peak performance numbers for computer architectures can be deceptive, especially for parallel processors where communication overhead can be significant. To be realistic, we measure performance as the execution time of a given application. However, it is clear that the performance of many of the applications is closely tied to memory bandwidth and this is where C•RAM excels.

In chapter 6, we show through simulation that a workstation with C•RAM can execute some applications (such as image compression, database operations, and signal processing) in one thousandth of the time taken by a conventional workstation. The performance gain is achieved by exploiting the full extent of parallelism that exists in a typical C•RAM system.

Amdahl's law [144] applied to parallel processing dictates that the limit of parallel speed-up is the reciprocal of the fraction of computation which is sequential. In an application which has alternating parallel and sequential phases of computation, the "work" that the sequential (host) processor must do can be thought of as including collecting the results it needs from a parallel phase of computation from the SIMD array, performing the sequential phase, and returning its result to the SIMD array. For greatest performance, these transfers between the parallel phases of computation must be done as quickly as possible. When the host's memory *is* the SIMD array (C•RAM in this case), this passing of results between SIMD array and host is intrinsic.

3.5.3 Scalability

Computing integrated with memory scales with the quantity of memory and with foreseeable advances in the memory technology. When PEs are integrated into the memory, the number of PEs

grows in proportion to the amount of memory in a system. For applications that can effectively utilize these PEs, such as image processing applications or non-indexed database searches, performance will also grow with the memory size. Since the performance requirements of problems are continually outgrowing existing computing hardware, one can expect that large enough problems will exist to utilize growing numbers of PEs.

As long as memory retains its 2-dimensional array structure (wordlines × bitlines), the PEs added to the memory will also scale in density as memories become denser and IC feature sizes get smaller. If the pitch of a PE is constrained by the number of wires running vertically through it at its widest point (as is the case for C•RAM), and a bitline is at least the pitch of a wire, the pitch of the PE should also scale with shrinking IC feature sizes.

The ratio of memory bandwidth at the sense amps to elsewhere in the architecture scales with memory size and new technology over time. According to Patterson and Hennessy, memory requirements grow by a factor of 1.5 to 2 per year [144]. Obviously, the number of sense amps grows linearly with the number of memory chips. The number of sense amps (or bit-line pairs[13]) has also been growing linearly with memory density. Indeed, DRAMs have typically been designed with 128 bits per bitline (256 bits per sense amp) in the 1 Mb through to the 256 Mb generations. Different DRAM designs in the same DRAM generation can have different numbers of active sense amps. The maximum number of active sense amps grows nearly linearly with DRAM density (e.g. 1 K cycle refresh DRAMs are available in the 1 Mb through 16 Mb generations), while the minimum number of active sense amps grows more slowly with the square root of memory density. While improvements in logic speed out-pace improvements in memory cycle times (improvement by factors of 1.25 and 1.04 per year respectively), the growth in memory system size, and therefore internal memory bandwidth, is still greater, suggesting that C•RAM performance may grow faster than microprocessor performance. The number of data pins per memory chip (or CPU chip), however, is growing more slowly than the sizes of main memory, widening the bandwidth gap between internal memory bandwidth and the off-chip consumers of that bandwidth.

[13] Some memory architectures share a sense amp between two folded bitline pairs, using the isolation transistors to do so. For the purposes of C•RAM, this reduction by half of the number of sense amps doesn't reduce the number of *active* sense amps which provide the internal bandwidth.

3.5.4 Power Consumption

Computing in the memory can result in energy savings[14] (depending on the application). We have shown that conventional computers utilize a tiny fraction of the available memory *bandwidth* by not using the data from each memory column. Similarly, the *power* used to charge the bitlines could be better utilized if more of the data in a memory row was actually used by the processor(s). In a DRAM, the major consumer of power is the sensing current required to charge the bitlines and restore the voltage in the memory cells. The effective utilization of DRAM memories when measured as *data pins/active sense amps* typically ranges from 0.4% to 0.1% (based on a single access per memory cycle). By adding PEs at the sense amps and running appropriate applications, the energy used to charge those bitlines can be better utilized, resulting in fewer total memory cycles and fewer total sensing operations during the execution of a given application.

The energy savings from using C•RAM will depend on the memory access requirements of the parallel application compared to that of the sequential version. Parallel vector addition is an example of a task which provides high memory utilization when executed on C•RAM, since a vector addition can utilize one sense-amp's output per PE and not introduce any memory operations not found in the sequential version of the code. The full power savings will not, however, be obtained by parallel applications which disable some PEs or are inefficient in their use of parallelism.

Driving signals across printed circuit board traces is another significant consumer of power. When the computing is performed in the memory, power is saved by driving fewer signals "off-chip". While the SIMD instructions would still be driven across printed circuits, this power usage is amortized over thousands of processors per chip performing those computations.

3.6 Design Space Alternatives

We must meet the requirements of a sufficient number of applications with few or just one design to achieve volume while not including so many features as to drive the price out of reach. Our C•RAM design effort is directed towards our perception of the most relevant applications. Many design space alternatives exist, and are favourable for other applications. The two most significant alternatives are to use a wider datapath in the PE and to use an SRAM memory.

While we have chosen 1-bit PEs for the best asymptotic performance/cost (see section 3.3.1),

[14] While power consumption during continuous operation is a commonly used measure, the energy consumed to complete a task is the better measure here.

it is clear that a more complex PE may be desirable under some circumstances. For instance, if a set of applications has limited parallelism, having many PEs won't help, but PEs with a wider datapath may speed up the task. If cost is less important, extra support for multiplication, division, and floating point operations could be added.

In our work we have focused on the economics of adding PEs to DRAMs, but PEs can readily be added to SRAM as well (see section 5.1). Compared to DRAM, SRAM is typically larger, more expensive, faster, and lower in static power dissipation. SRAM would be preferable if speed takes precedence over cost or size. This would be particularly true if PEs were to be added to a system which already employed SRAM, such as the main memory of a vector supercomputer. SRAM is also suited for smaller volume production. ASIC fabrication processes are capable of building moderate density SRAMs. The one-time costs are lower and it is easier to obtain access to IC fabrication in an ASIC process than in a DRAM process. For special purpose ICs, which combine C•RAM and custom logic or mixed analog-digital circuits, the memory may no longer dominate the chip area and an ASIC process may be better suited for the design. An SRAM or ASIC process may be preferred for fast logic since the substrate is maintained at V_{SS}, unlike DRAM designs which apply a negative voltage to the substrate to reduce storage cell leakage. For aerospace applications, SRAM has an advantage over DRAM in that SRAM can be more readily radiation hardened.

We have focused on narrow PEs in order to maximize the number of PEs. However, if the cost per bit of C•RAM memory must be reduced to make it more competitive with DRAM, then the total area taken by PEs can be reduced by reducing the number of PEs. Wider PEs would not have to be as high as the narrow PEs described in this thesis. Since there would be a reduction in the area taken by wiring in a PE with a less narrow aspect ratio, the area per PE would typically be less than that of a narrow PE.

3.7 Summary

We have illustrated that computing in the memory has advantages in memory bandwidth, IC packaging, size, energy per calculation, cost, performance, and scalability. To make full use of these advantages, we must design C•RAM with some constraints which are not typically found elsewhere in computer architecture. Since we intend to design chips where the memory area is dominant, we must design with the existing memory array design and memory IC process. This implies designing PEs with narrow pitches. The long wordlines essentially constrain the architecture to be SIMD

if most of the offered bandwidth is to be utilized. We argued that bit-serial arithmetic has higher performance/cost by examining the asymptotic case. By associating these bit-serial pitch-matched PEs to DRAM sense amps, we have created the first processor-in-memory architecture that is scalable across many generations of DRAM. With IC transistor counts growing faster than pin counts, data movement between chips for the purpose of computation will become increasingly expensive, and processing-in-memory architectures will become increasingly important over time.

Chapter 4

Architectural Details

In this chapter, given the constraints of building the PEs into memory, we consider alternatives in PE architecture and network architecture, discuss various design tradeoffs, and settle on a specific architecture.

Some of the key C•RAM architectural decisions have been discussed in chapter 3:

- the PEs have a 1-bit wide datapath
- all PEs will use the same SIMD instruction provided from off-chip
- all PEs will use the same local memory address (uniform memory addressing)

The C•RAM architecture, the targeted applications, and the circuits that implement C•RAM all evolved together influencing one another. Performance can only truly be measured on real applications. However, for making initial design choices and tuning the architecture we used vector integer addition as a benchmark, since addition is representative of the performance of other operations including multiplication. We still require that the architectures be able to perform a variety of arithmetic and logical operations, not just addition. The applications in chapter 6 show good performance on a variety of operations.

4.1 Processing Element

A processing element consists of an ALU, registers, and a memory interface. A generic architecture is shown in figure 4.1. The memory and register(s) act as sources and/or destinations for the ALU. The ALU must have at least two inputs. The memory interface consists of the read

and write path from/to memory and a means of selecting one of the memory columns if more than one column is connected. The term SIMD implies a shared instruction with an opcode controlling the ALU and operand(s) specifying registers and, in the uniform memory address case, memory address. In order to implement SIMD conditional operations, it is useful for the memory interface to be able to control writing to the memory on a per PE basis; this is provided by the write-enable register (see section 4.1.2).

Since one of our design goals is to keep the PE simple and low cost (relative to a column of memory), we do not include a large register file. With few registers, memory accesses must be frequent if the ALU is to be well utilized. In this case, the register file can be reduced to the minimum size, i.e. just a sufficient number of registers such that all ALU inputs are connected to a data source (memory or register). In this case, the outputs of registers can be hard-wired to the ALU inputs, thus saving register-decoding circuitry.

4.1.1 ALU Design

We consider several ALU designs which all have narrow, small-area implementations. Their names are derived from the functions which the ALU performs: *NAND ALU, NAND/XOR ALU, NAND/OR ALU, 16 Function ALU, 256 Function ALU,* and *64 K Function ALU.* The NAND and NAND/XOR ALUs were proposed by Snelgrove [21]. All ALUs take operands from the contents of memory and from one or two single-bit registers. The SIMD instruction includes the memory address, the ALU operation, and the destination of the ALU result. The PEs corresponding to the first 4 ALUs listed each have only a single data register. Figure 4.1 shows the generic architecture for these 4 PEs. The 256-function-ALU and 64 K-function-ALU PEs each have an additional data register. With the exception of the 64 K-Function-ALU PE, the ALU results can be directed to any register or to memory. The PE can perform a read followed by a write to the same memory address within the same memory cycle (read-modify-write), a feature of memory which other MPP designs fail to exploit.

Below, we use the number of memory cycles and time required to perform a 2-operand addition (`a=a+b`) to compare performance for the various ALU alternatives. For this computation with arbitrarily sized operands, a minimum of two memory accesses are required per bit being added, with one of these accesses being a read-modify-write. For the PEs discussed, one or two ALU cycles may be performed in a memory cycle, with the memory cycle containing two ALU cycles being 11% slower than the memory cycle containing one ALU cycle.

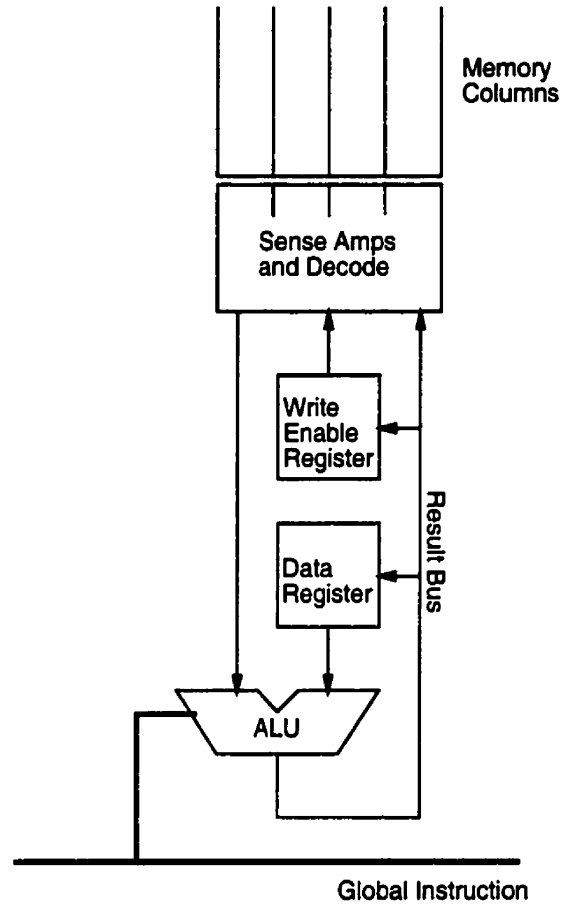

Figure 4.1: Generic single data register C•RAM PE

The PEs which incorporate the aforementioned ALUs are compared in figure 4.2 by an estimate of their area[1] and their performance. The benchmark used for performance is 32-bit integer addition. One million 32-bit additions performed in a bit-serial fashion by a single PE each second represents 1 MIPS on the graph. The timing assumed is 135 ns for a read-modify-write ALU and memory cycle time; and 150 ns for a memory cycle with two ALU operations. All memory accesses (to a different address than the current memory address) are assumed to take the same time[2].

[1] Estimates of PE areas are based on transistor counts and how the transistors pack together. To make a fair comparison, all PE area estimates include write-enable and communications logic. Only the 16-Function-ALU and 256-Function-ALU PEs have actually been built (see section 5.1). PE layouts are constrained to fitting in the pitch of a static memory column. The PE areas would most likely be different if no pitch constraint were imposed.

[2] Some memory accesses can be made faster. Section 5.2.2 describes how row caching can take advantage of spatial locality.

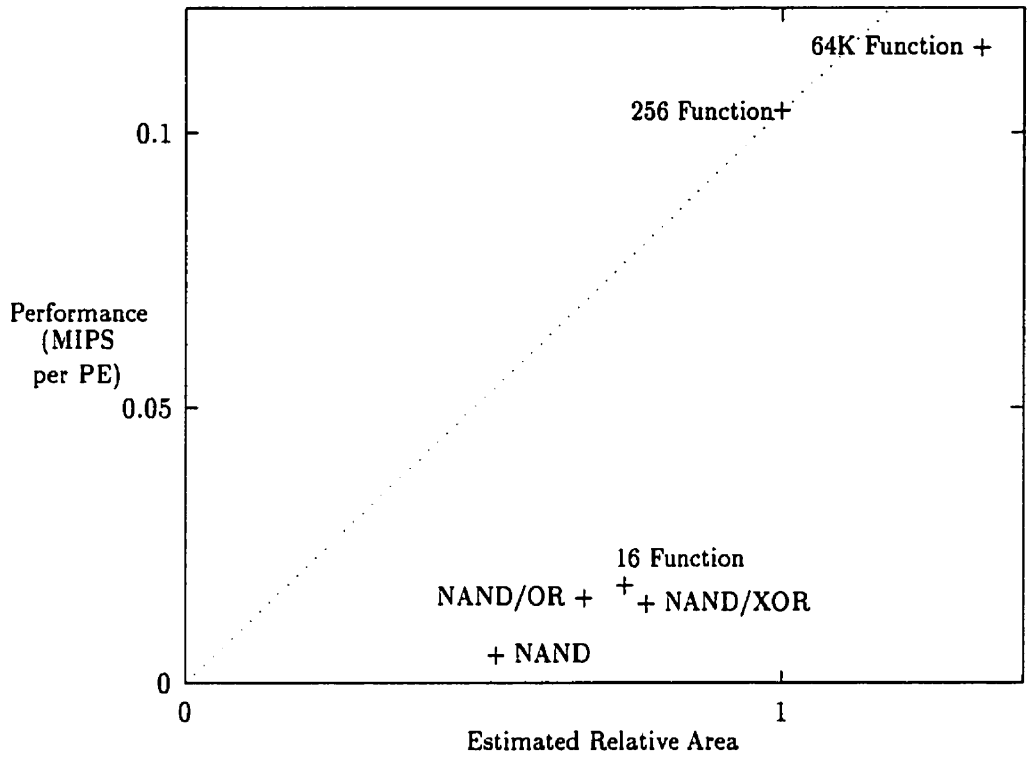

Figure 4.2: Arithmetic performance vs. PE area

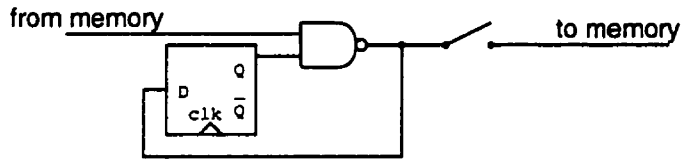

Figure 4.3: Minimalist NAND processor element

NAND ALU

If an ALU is to support only one function, then that function must be logically complete. NAND and NOR are examples of logically complete functions. An example of a PE based on a NAND gate is shown in figure 4.3. The inputs of the NAND gate are from memory and a register. The output of the NAND gate, under control of the SIMD instruction, can be selectively written to either the memory or the register or both. The NAND-ALU PE is small and complete, but slow. It can synthesize a one-bit full add in 44 memory cycles.

NAND/XOR ALU

For the one-bit full add, the NAND ALU requires most of the cycles to synthesize the exclusive-or (XOR) operation. Hence, adding an XOR to the ALU can improve performance. The PE opcode (from the SIMD controller) selects the result of either the NAND or XOR operation to be written to the destination (register or memory). A 1-bit full add with the NAND/XOR ALU takes 16 cycles.

NAND/OR ALU

The *NAND/OR ALU* adds extra functionality, relative to the NAND/XOR ALU, yet requires fewer transistors. It can perform a full add in 15 cycles. The NAND/OR ALU also implements the XOR operation: when both NAND and OR operations are selected simultaneously, the wired-AND result of the NAND and OR outputs is an XOR of the operands.

16-Function ALU

The *16-Function ALU*, under control of the opcode, can perform any boolean function of 2 inputs. The number of functions such an ALU can perform is:

$$(number\ of\ output\ levels^{number\ of\ input\ levels^{number\ of\ inputs}}) = 2^{2^2} = 16$$

It is well known that a multiplexor can be used to implement arbitrary boolean functions [145]. A 4-to-1 multiplexor is required to implement this 16-function ALU. The 4-bit ALU-operation field of the PE instruction is applied to the multiplexor data inputs. This 4-bit ALU operation field is, in effect, the "truth table" for the function being implemented. The input from memory and a single data register select one of the 4 bits in this "truth table" to be the result. This sounds unnecessarily general, but the multiplexor has a compact VLSI implementation and the extra functions are useful for operations other than addition (such as subtract, multiply, and boolean operations). Requiring 13 cycles to perform a full add, this ALU outperforms the previously mentioned architectures.

256-Function ALU

The *256-Function ALU*, shown in figure 4.4, has two data registers. The ALU is an arbitrary function of two registers plus a memory location and is implemented as an 8-to-1 multiplexor with 2^{2^3} functions. With this design, a full add requires two memory cycles, the absolute minimum for 2 operand addition (`a=a+b`) with source and destination in memory. For addition, the ALU is used

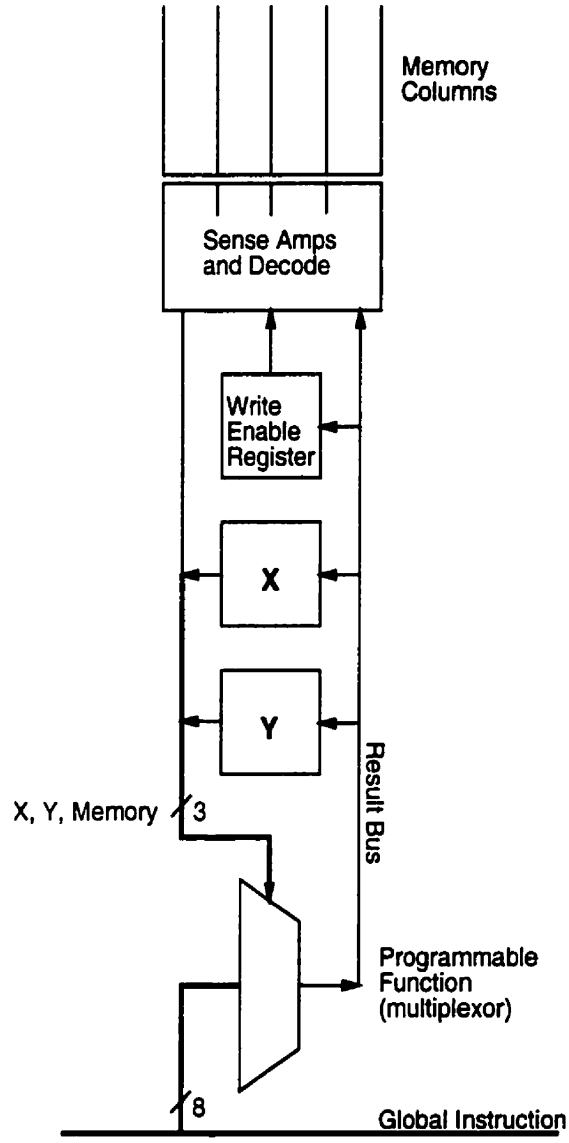

Figure 4.4: 256 Function processor element

twice per memory cycle (to first calculate the half-adder sum and then the carry), requiring the memory cycle to be longer than for the aforementioned ALU designs.

64 K-Function ALU

The *64 K-Function ALU* has two 8-to-1 multiplexors and operates similarly to the *256-Function ALU* except that the pair of ALU operations performed sequentially in the 256-Function ALU are performed concurrently in the 64 K-Function ALU. Like the design in figure 4.4, one multiplexor result can be selectively written to the *X* register, the write-enable register, or to memory. The design of this ALU differs from the previous one in that a second multiplexor exists which writes only to the *Y* register (typically used for carry). Like the 256-Function ALU, a full add requires 2 memory cycles, but because the two ALU operations are performed concurrently (rather than back-to-back), the memory cycles are about 10% faster.

Conclusion

We use PE area as an estimate of cost. The dotted "iso-performance/cost" line running through the 256-Function PE point indicates that the other labeled ALU designs (all below the line) have lower performance per area.

Figure 4.2 shows that, of the 4 single-data-register PEs, the 16-Function-ALU PE has the best performance/cost. Overall the 256-Function-ALU PE has the best performance/cost of those considered and has been built (as described in section 5.1). While the 64 K-Function-ALU PE is the fastest overall, it has a lower performance/price and its speed does not justify the area. Other plausible PE designs exists with complexities that lie between and beyond those examined. While there is likely to be valuable future work in further tuning the PE architecture, the performance and cost of the *256-Function ALU* make it our choice for hardware development.

4.1.2 Conditional Operations

In a SIMD array, hardware support for conditional operations is useful. Since many SIMD applications require that PE operations be conditionally performed, subject to the value in each element of a parallel variable, SIMD processors with hardware support for conditional operations can run these applications faster than SIMD processors which implement the conditional operations in software. Conditional writes to memory are an example of such a conditional operation. A

hardware *write enable* circuit can accomplish the task. Write-enable registers were included in figures 4.1 and 4.4. The write-enable register allows operations to be performed conditionally in a data-parallel fashion, taking the place of a conditional branch in sequential processor architectures.

To initiate a conditional operation, each PE evaluates an expression producing a boolean result (e.g. `a>=b`), which is written to the write-enable register. The write operations of those PEs with a '0' in their write-enable registers are effectively suppressed. Writes to registers are not affected by the write-enable register; this saves transistors and also permits new conditions to be evaluated and written to the write-enable register. Finally, the PE write-enable register does not affect external memory write operations (e.g. from the host processor).

4.2 Interconnection Network

C•RAM requires inexpensive, scalable interprocessor communication that is also compatible with DRAM memory. Without communication among PEs, C•RAM could not solve many useful problems efficiently. In the design of C•RAM, however, we take a more cost conscious approach to interconnection networks than most SIMD parallel processor designers do; we would like to keep the increase in size to a minimum. Our target is to use less IC area for communication than for the non-communication portions of the PEs. As mentioned in the introduction to this chapter, we realize that by using low cost networks, some applications will not run well on our architecture.

Network traffic switching can be accomplished in *time*, *space*, or a combination of the two. An example of *time* switching is multiplexing all outputs onto a broadcast network (time division multiple access). At the other extreme, a network which can send data from multiple inputs to multiple outputs without "sharing" any link (e.g. a crossbar), switches only in *space*.

Some important communications operations an interconnection network may provide are: autonomous communication, point-to-point communication, broadcast, and combine operations.

Autonomous communication

A network has autonomous routing if the destination can be specified by each sender. For reasons of cost, we do not consider hardware support for autonomous routing. Instead, all communication follows centrally controlled patterns (i.e. "patterned communication").

Point-to-point communication

For point-to-point communications operations, as the name implies, each message has a single sender and single receiver. The messages could travel to their destination in a single hop or multiple hops. Since we will not support autonomous routing, if multiple messages are to be sent at once, the messages must be *uniformly* routed in a *pattern* specified centrally by the controller. An example of point-to-point patterned communication is the sharing of adjacent data values between PEs on a 2-dimensional grid for a convolution with a 3×3 kernel.

Broadcast communication

Broadcast communication occurs between one sender and multiple receivers. An example of its use is to distribute a new upper bound in a branch-and-bound minimization problem.

Combine-operations

Combine-operations, as the name implies, combine the outputs of multiple PEs. The simplest combine-operations are boolean and can be easily implemented with a *wired-OR* or *wired-AND* bus. The OR of all the PE's outputs can be used to determine if an associative search has returned a match, if a solution to a non-deterministic problem has been determined, or if an exception (e.g. arithmetic overflow) has occurred in any of the PEs.

4.2.1 Suitability of Interconnection Networks

In the following, we describe a number of interconnection networks, categorized by their complexity, and discuss their suitability for C•RAM. To be suitable, the networks must have a low cost, and be compatible with both the memory arrays and memory-style IC packaging. We will discuss later in greater detail, those networks which are most suitable for C•RAM. For a more detailed analysis of all of these networks, see [146, 147].

In order to achieve our goal of adding processing to memory with low overhead, both the network and the PE must occupy minimal silicon area. This imposes a practical limitation of networks that require $O(n)$ area for n processors. Networks that require $O(n^2)$ switching nodes (e.g. a crossbar) or networks with fewer switching nodes but require $O(n^2)$ area (e.g. a butterfly network) will dominate the cost of a SIMD array of 128 K PEs or more.

Many parallel applications require logic 2-dimensional grid communication for which physical grid interconnections would be ideally suited. While a basic 2-D grid interconnection requires $O(n)$ area, it is awkward to get signals over or around the memory array. Low-cost packaging considerations alone preclude the 2-D grid, because 2048 PEs (organized as 32×64) would require 192 additional IC pins for the network.

Buses and variations on 1-D grid (left-right nearest neighbour) networks require $O(n)$ area and are compatible with the layout of a row of PEs adjacent to memory arrays. These networks have been adopted for C•RAM and are discussed further.

Bus Interconnect

A broadcast medium (such as a bus) is inexpensive to implement, but is restrictive in its communications bandwidth, since it switches in time rather than space, so the aggregate communications bandwidth of a simple bus does not grow with the number of processors. Nevertheless, a broadcast bus is useful in a massively parallel computer if multiple PEs can make concurrent use of it for broadcast operations (single transmitter, multiple receivers) and combine operations (multiple transmitters, single or multiple receivers). While broadcast and combine operations could be implemented on any fully connected network topology, a bus is arguably the least expensive. The bus can run along a row of PEs without interfering with the memory array. In section B.5, we show that a network which performs combine operations and provides the results to the PEs is useful for synthesizing higher level combine operations such as finding the maximum value in an array.

For this reason, we have decided to use it in C•RAM, even though its low bandwidth makes it unsuitable as the *only* interconnection network. In our implementation, the broadcast bus is arranged as a tree of *bus segments* with repeaters to increase drive and reduce delays. By selectively disabling these repeaters, local interconnect between groups of PEs sharing bus segments is provided and the network bandwidth can scale with the number of PEs. The size of a logically shared bus segment may be the entire system (global bus), groups of chips, a single chip, a row of PEs, a fraction of a row between wordline-strappings, or (as implemented by Cojocaru [148]) bus segments can be selected down to a pair of PEs.

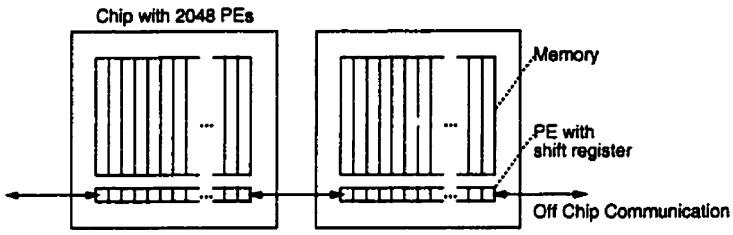

Figure 4.5: Shift register interconnection

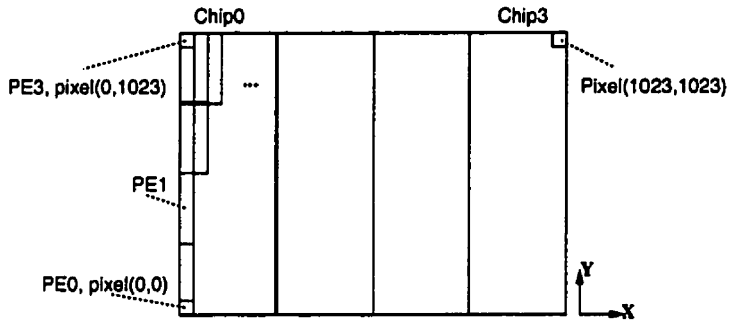

Figure 4.6: The mapping of pixels to PEs

This figure shows a 1024×1024 pixel image partitioned among 4 C•RAM chips with a total of 8192 PEs. Each large rectangle represents the portion of the image stored in one C•RAM chip. Each small rectangle represents the portion of the image which is stored in the local memory of a given PE. Any processing of those pixels is performed by that PE. The small squares represent the pixels themselves.

Left-Right Interconnect

Since the C•RAM PEs are arranged in rows, one-dimensional (left-right nearest neighbour) communication (shown in figure 4.5) is inexpensive to implement and it is compatible with the memory array and memory IC packaging.

The shift register between adjacent PEs is implemented by putting a second input on both data registers and tying these to the result buses of adjacent PEs [149]. Using one PE register for each direction allows the registers' speeds to be matched.

Special provisions must be made for column redundancy, however, since the PEs can be identified by both their memory address and by the PEs they are adjacent to in communications (see section 5.2).

This network is obviously suited for applications that require one-dimensional grid communication. Small two-dimensional grid applications can also be trivially mapped to PEs connected in a

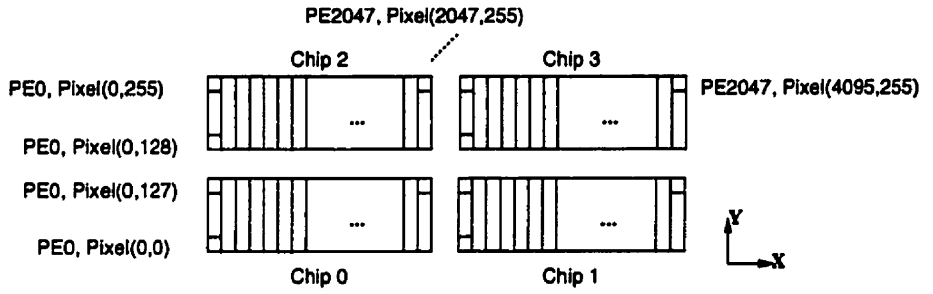

Figure 4.7: The mapping of pixels to PEs for scalable interconnection

This figure shows a 4096×256 pixel image (or a portion of a much larger image). Once again, each large rectangle represents the portion of the image that is stored in one C•RAM chip. Each small rectangle represents the portion of the image that is stored in the local memory of a given PE. Any processing of those pixels is performed by that PE. The small squares represent the pixels themselves.

one-dimensional grid by assigning a column or columns of grid elements to a PE. Moderately sized two-dimensional grid applications can be efficiently mapped onto a 1-D grid by assigning less than a full column to each PE. A 2-dimensional application can be partitioned by assigning a rectangular block of elements (pixels in the case of an image) to each PE as shown in figure 4.6. We show in appendix C the that optimal width of this rectangle is one pixel. Vertical communication between adjacent blocks is one hop when using the 1-D interconnect[3]. Horizontal communication requires multiple hops equal to the number of blocks that the array is high (four hops in the case of figure 4.6). With this structure, in our simulations, a convolution of a 1024×1024 pixel image has less than 10% communication overhead, but this approach does not scale well. The communication bandwidth does not scale linearly with the number of chips, since for large applications the communications latency grows in proportion to the height of the problem.

Communication between chips can be accomplished by extending the left-right linear communication between chips through bi-directional pins, as shown in figure 4.5. The data elements on the problem boundaries must be treated specially. Each PE receives data from the PE above and below.

The left-right interconnection has a low cost, is compatible with memory, is effective for 1-D problems and moderately sized 2-D problems, but does not scale for large 2-D problems.

[3]Snelgrove proposed labeling the single hop direction vertical in order to reduce the buffering requirements for horizontally scanned video [150]. With the data in this organization, all chips can participate in providing data which is part of any given raster scan line.

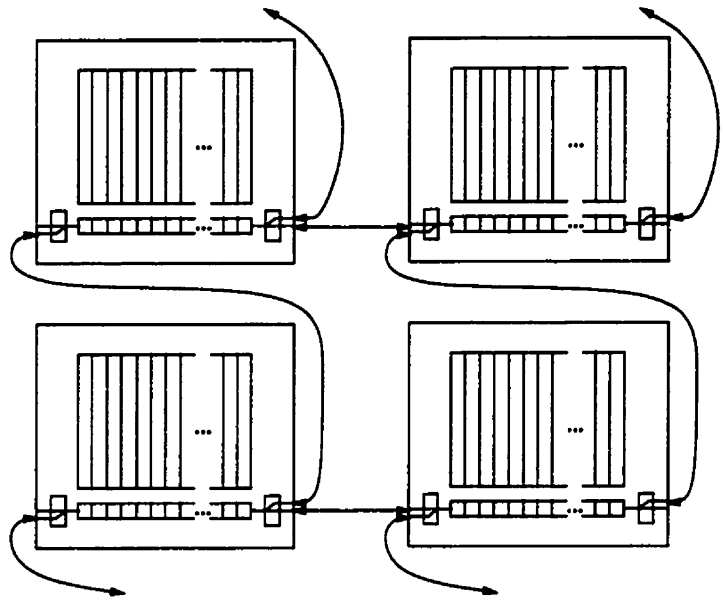

Figure 4.8: Scalable left-right (4-pin) interconnection

Scalable Left-Right Interconnect

One-dimensional communication is quite compatible with the structure of memory and C•RAM PEs, but the capacity doesn't scale linearly for 2-D problems on large numbers of chips, and implementing full two-dimensional communication uses too many pins for dense memory packaging. It is possible to combine both structures by using one dimensional communication within the row of PEs (e.g. 512 PEs), and a 2-D grid (or higher dimension to suit the application) between arrays or between chips (i.e. a superset of the left-right interconnect).

An application with pixels mapped to PEs as shown in figure 4.7 can be supported with the configuration shown in figure 4.8. A vertical stripe of pixels is mapped to each PE. Adjacent PEs hold horizontally adjacent stripes of pixels. The arrangement of data among C•RAM chips mimics their spatial organization, which is shown in figure 4.8. The left and right ends of the shift register running through a row of PEs are switched for either horizontal or vertical communication. For vertical communications, a row of values are loaded into the shift register; shifted by the number of PEs in a row; and then stored in the PEs above or below, depending on the direction of the communication. This communication is described in more detail in appendix B.

The switches to switch between vertical and horizontal communication can be located on-chip or off-chip. Placing the switches off-chip saves pins on the C•RAMs, but placing the switches on-chip avoids the need for a routing chip and reduces the total system chip and pin count (which is usually

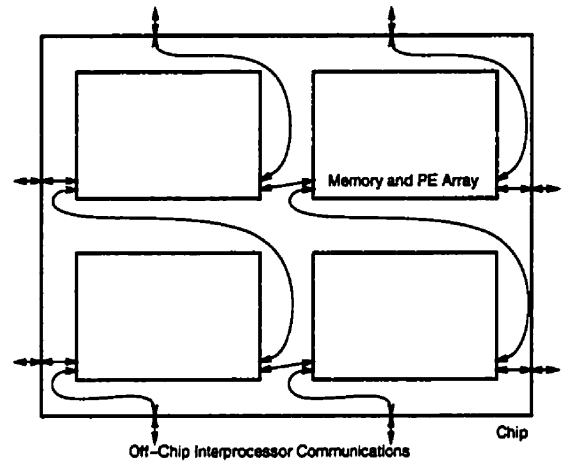

Figure 4.9: Scalable left-right interconnection with increased bandwidth

Rows of PEs within a chip are interconnected in a 2-D grid. In this figure, 8 pins are used for communications instead of 4 pins.

preferred).

The multiple rows of PEs on a chip (e.g. 4 rows of 512 PEs for the DRAM based C•RAM in section 5.2) can be connected as one long shift register (2048 PEs requiring 4 communications pins), or as a mesh of shorter shift registers (2×2 groups of 512 PEs requiring 8 communications pins) for greater bandwidth. Figure 4.9 shows how rows of PEs are internally connected to bring more communications pins off-chip.

Like the left-right interconnect, boundary conditions can be applied in software or a band of read-only elements can be set up. Unlike the left-right interconnect, it is possible to send the boundary values in through the communication network without additional switches (e.g. always send zeros).

This 1-D/2-D interconnection is scalable; compatible with memory; economical in chip area and chip pins; and provides sufficient bandwidth for communication-intensive applications (as will be demonstrated in section 6.3.1). It is easy to use, since all PEs can be treated identically in software. Figure 4.10 shows the PE with this scalable left-right interconnect, together with the broadcast bus, as used in the high density DRAM C•RAM design (described in section 5.2.1).

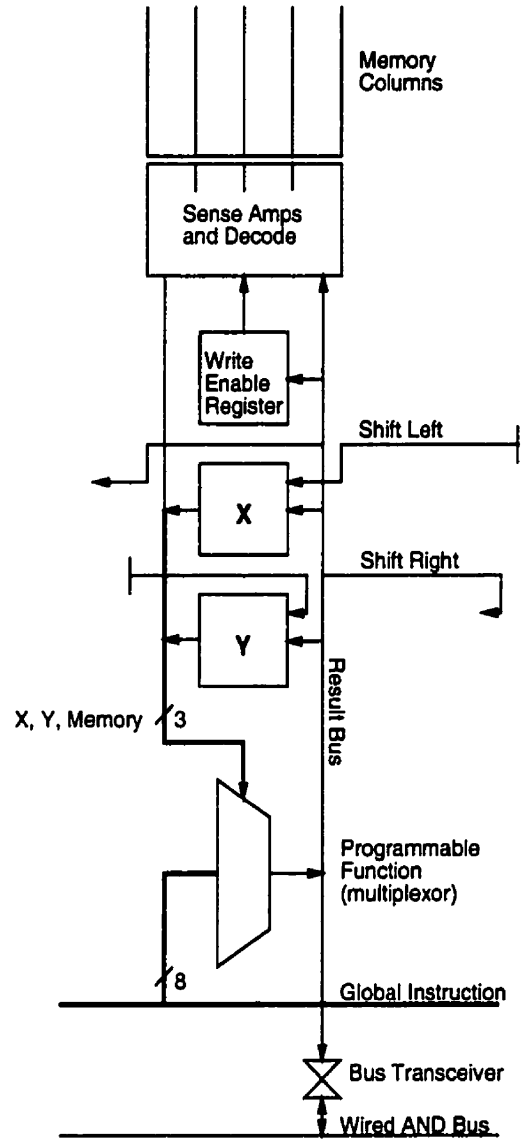

Figure 4.10: 256-Function processor element including interconnection

Interconnection Network Summary

The two interprocessor communications networks chosen for C•RAM are a broadcast bus to implement combine operations; and a scalable-grid communications network with 1-D communication within the rows of PEs for compatibility with memory and 2-D communication among chips for scalability. The latter communications network can double as an IO port. The PE with network support is shown in figure 4.10.

4.3 System IO

There are several options for performing IO transfers with C•RAM, depending on IO bandwidth requirements. For transfers at conventional memory speeds, IO devices could be connected to the memory bus for conventional DMA (direct memory access). However, if a single memory bus is used, the memory bandwidth does not scale with the number of C•RAM chips and PEs. For higher speed transfers, IO should be connected to the interprocessor communication network. A dedicated video-RAM port could be added at an additional cost for still higher bandwidth and an uninterrupted flow of data.

For example, the scalable left-right interprocessor communication network could double as an IO port, where IO connections are made at the perimeter of the 2-D grid of C•RAM chips. For still higher IO bandwidth, IO connections could be made to each chip. Each high density C•RAM interprocessor communication pin, however, has a sustained bandwidth of 65 Mb/s (based on a 4 Mb C•RAM chip with a 512-bit-long shift register, 120 ns memory cycle plus 15 ns operate cycle, $1b(\frac{120\ ns}{512} + 15\ ns)$). A memory system containing 32 MB of C•RAM would then have an aggregate IO bandwidth of 8.4 Gb/s (64 chips, each with 2 active IO pins in each direction). Input and output can be performed to each chip concurrently. In this configuration the transfer of 1 M pixels of 24 bits (for both input and output) would take only 3 ms. With a 60Hz video refresh rate, this would constitute an 18% IO overhead. IO using the communication network and computation cannot be overlapped in our design.

If IO bandwidth greater than 8.4 Gb/s is required, a video-RAM sequential access memory and video memory port could be added to C•RAM. The VRAM approach has the added advantage of providing IO concurrent to computation. The VRAM approach can also provide continuous data transfer (e.g. for video data) without the need for elastic buffers. VRAMs commonly have two or more banks of sequential access memory (SAM) so one can be providing or receiving data while the contents of the other are being transferred to or from memory. We don't recommend the expenditure of silicon area for a VRAM port unless the application genuinely requires the IO performance.

4.4 Summary

In this chapter, we have defined the C•RAM architecture in more detail and discussed our design tradeoffs. The chosen PE architecture has the best performance/cost of the narrow PEs examined. The one-bit-wide PE consists of two data registers, one write-enable register, an ALU that

performs a programmable arbitrary function on three inputs, and interprocessor communications. Two communications networks have been chosen. Both networks require wiring in only one dimension within a row of PEs and are thus scalable, and compatible with the memory array and memory packaging. The broadcast bus is useful for broadcasting a single value or performing combine operations among all PEs. The one/two-dimensional network is efficient for passing values to nearest neighbours in one or two dimensions, as well as high speed IO. The performance and cost of the PE architecture and communication networks can scale linearly with memory density and number of C•RAM chips in a system.

Chapter 5

Implementation

We have designed and built an SRAM-based prototype of C•RAM in a 1.2 μm CMOS process, referred to as *prototype C•RAM*, as well as designed a PE in a 4 Mb stacked-capacitor DRAM process (referred to as *high-density C•RAM* or *4 Mb C•RAM* in section 5.2). We will show that C•RAM utilizes a good portion of the internal memory bandwidth while making small demands on power consumption and silicon area.

5.1 Prototype

Prototype C•RAM chips have been fabricated in Nortel's 1.2μm CMOS process in order to prove the feasibility of the PE and the memory interface. The particular IC process we used was chosen because it was available to universities through the Canadian Microelectronics Corporation. The prototype memory capacity and PE count were limited by the maximum IC area permitted per project and the IC process's limited suitability for memory.

Figure 5.1 is a micrograph of a 64 PE by 128 bit C•RAM incorporating the 256-function PE. 8 Kb of 6-transistor static memory cells occupy the central dark rectangle. The PEs fit in the pitch of a sense amp which is the same as the pitch of a column of memory. These PEs, located along the bottom of the array, occupy less than 9% of the chip area.

Figure 5.2 shows 12 processing elements in detail. For convenience, the architectural overview schematic is given again in figure 5.3. Each PE communicates with other PEs through the wired-AND bus, but no left-right interconnection is provided since, at the time, we favoured straight-forward column redundancy over the extra communication.

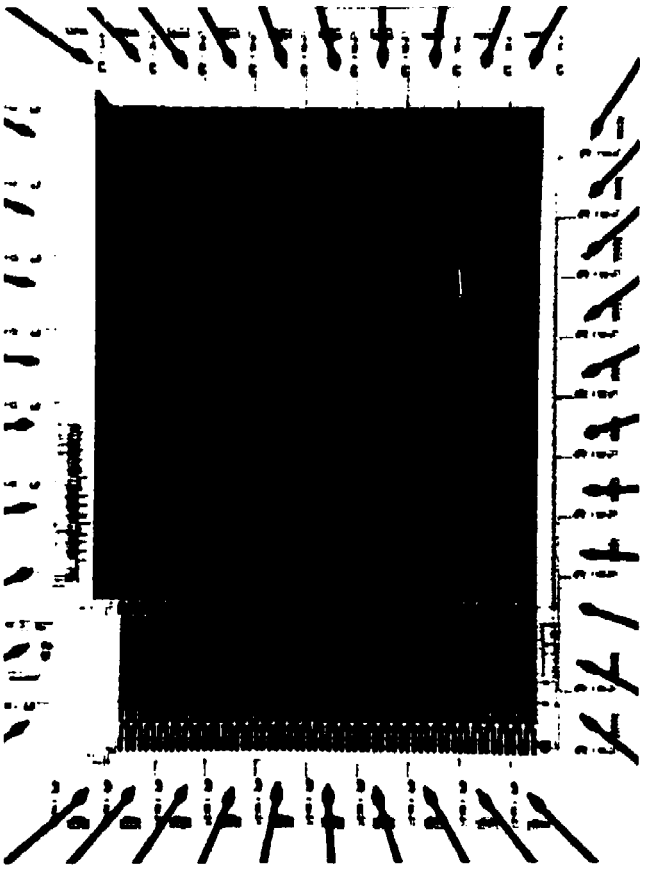

Figure 5.1: 64 processor prototype C•RAM die photo

5.1.1 Memory

The memory portion of the prototype C•RAM employs a 6 transistor CMOS cell which does not have a high density. The IC process we used was designed for mixed analog-digital circuits, not memory. Since our end goal is to build on existing high density memories rather than create the highest density memory possible in the available ASIC process, we made conservative design decisions in the memory. There was little point in designing and using a higher density variety of memory, which would make no advances in memory technology, but instead increase the risk of the chip not functioning. Dynamic memory cells (1-, 3-, or 4-transistor cells) were not used since we did not have enough information on transistor leakage current (the Canadian Microelectronics Corporation had not provided SPICE models for process corners, or even models which had been verified against silicon at the time) and we would rather not provide memory refresh during initial testing. A 4-transistor static memory cell requires a high resistance polysilicon layer which wasn't available. A small area 6-transistor NMOS memory cell was evaluated, but the quiescent power

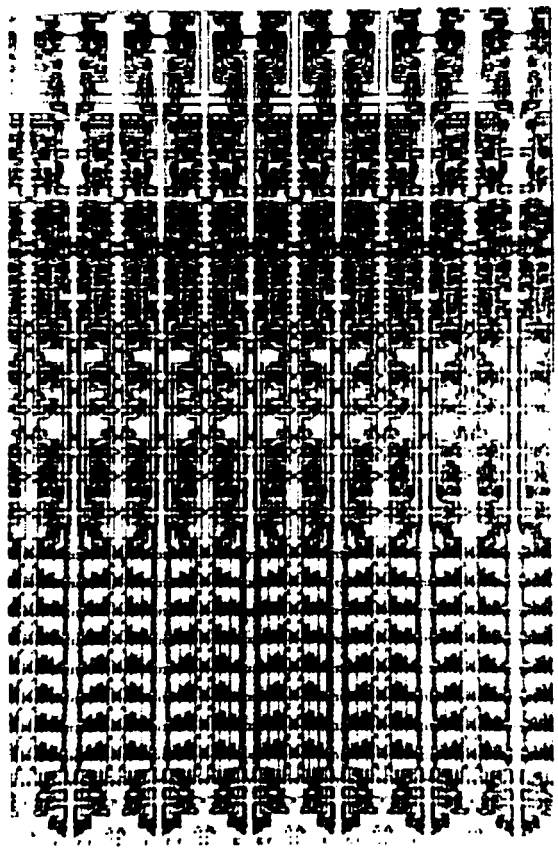

Figure 5.2: Detail of 12 C•RAM PEs

consumption was too high. Thus, we chose a 6-transistor CMOS static memory cell.

The row decoders (left side of the memory array in figure 5.1) and column decoders (bottom of the array) are CMOS AND gates provided with quaternary pre-decoded address inputs[1]. Two address lines plus their complements require 4 wires and two inputs on each decoder. The one-of-four pre-decoding uses the same number of wires feeding the decoders as a binary-plus-complements signal scheme but reduces the number of inputs on each decoder by half. At least one of the groups of four pre-decoded address signals must be gated to control the start and end of the row address operation.

We experimented with simultaneous writes to multiple memory rows and columns. Writing to multiple memory rows allowed the host or PEs to perform, in a single memory cycle, an assignment with multiple destinations (e.g. a=b=c) or a block fill or clear. The number of wordlines that can be selected is limited by the drive of the write buffers, but any number of columns can be

[1]This technique has since appeared in other designs [94].

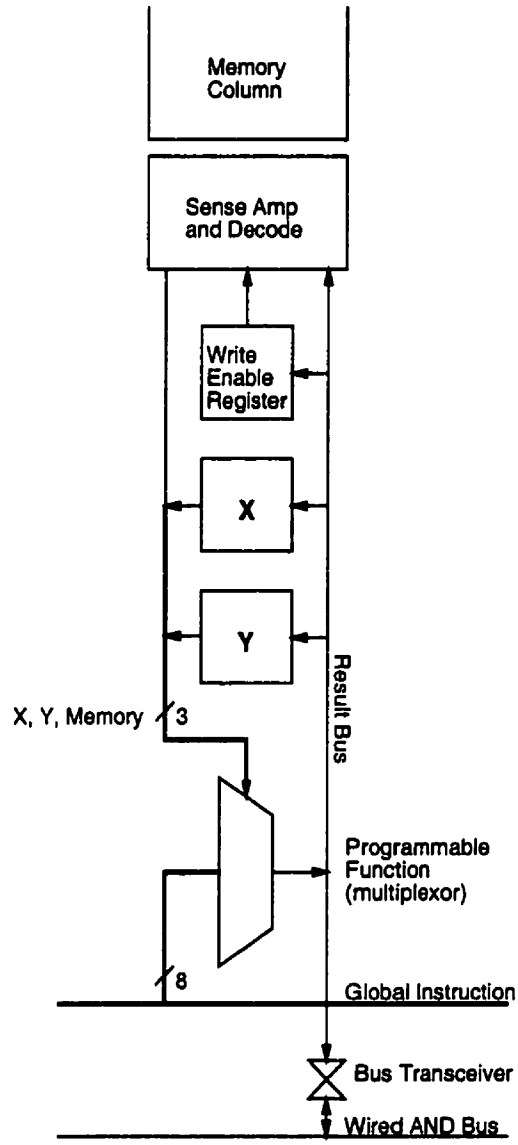

Figure 5.3: Prototype processor element architecture

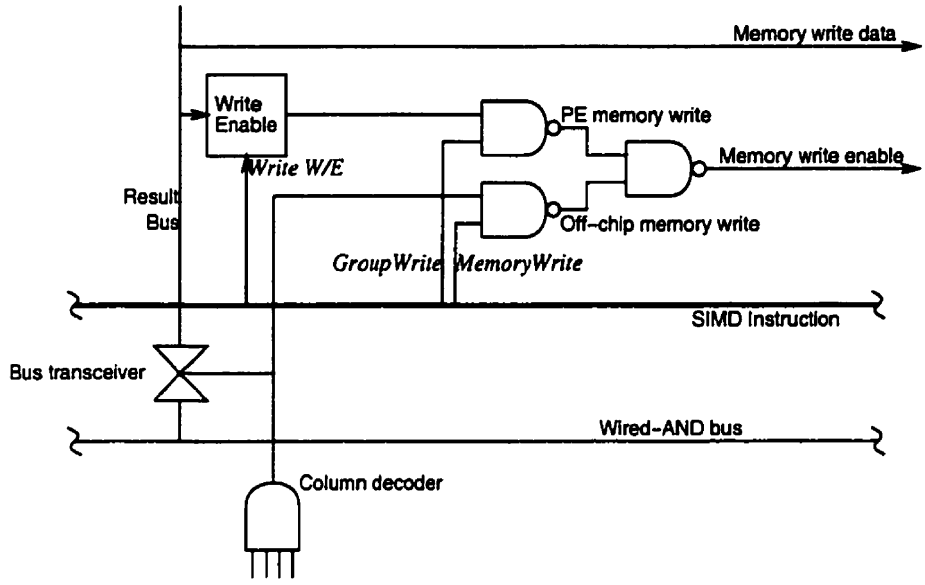

Figure 5.4: Detail of write-enable and bus-transceiver control

selected. Because it doesn't appear that this multiple row selection technique will be commonly used (especially from a high level language), it was discontinued.

C•RAM memory accesses, whether they be made by the PEs or from off-chip, use some common structures. A simple four-transistor differential voltage sensing amplifier is used for reading the memory (after precharging the bitlines to V_{DD}). During a write, the ALU result is differentially driven onto the bitline pair. During a cycle of computation, the data read from memory is always connected to an input of the ALU. The ALU result is written back to memory during a SIMD operation when the $GroupWrite$ signal is asserted, and then only on those PEs with the WriteEnable register set. The PE write-control logic is shown in figure 5.4. For off-chip memory operations, the read and write datapaths reuse the broadcast bus, saving some circuitry compared with having separate memory data buses.

In the DRAM-based design which follows this one, the off-chip read/write datapath does not go through the broadcast bus. The broadcast bus is "single-ended" requiring a significant voltage swing for a transition to be recognized. In order to achieve fast access times, the majority of commodity memory designs, use differential signaling so that a slight difference in voltage can be amplified and detected, reducing interconnection delay when compared to single-ended signaling [116]. Indeed, by combining differential signaling and direct bitline sensing, Nagai et al. have demonstrated a 17 ns (t_{RAC}) DRAM [115].

The timing for all C•RAM events originates off-chip. Self-timed circuits (such as those used

in DRAMs) could be provided to generate some of the events, but a timing failure (race condition) would render the chip inoperable. With all timing generated off-chip, the duration of each phase of operation can be adjusted during testing to determine the maximum performance of the design. By not multiplexing row-address, column-address, and instruction signals, more pins were required, but testing was simplified.

5.1.2 Processor Element

Circuit and layout aspects of the 256-Function prototype C•RAM PE are explained in this subsection. Some of the major design requirements for the PE, as described in chapter 3, are that it should have a narrow pitch, have small area, have low power consumption, be easily arrayed, and be reasonably fast. A narrow PE pitch is required so that as many PEs as possible may be fit together along the edge of a memory array in order to obtain the highest aggregate performance. A PE was designed which could fit in the pitch of a single SRAM cell[2]. Of course, the PE would be wider than a DRAM cell since DRAM cells are smaller than SRAM cells.

The PEs must also be easily arrayed. This is achieved by alternating the design and its mirror image (thus sharing bus lines and contacts), and routing the opcode through the PE, connecting to the gate of only one transistor in each PE. Speed has some significance as the PE cycle time adds to the memory cycle time for a read-modify-write operation. We have found, however, that because the PE has smaller capacitances than the memory bitlines, it is not difficult to build a PE which is faster than the memory cycle.

Figure 5.3 shows the datapaths found in the PE. Recall that the ALU is implemented as an 8-to-1 multiplexor. The ALU-operation portion of the shared PE opcode is fed as a "truth table" into the data inputs of the multiplexor. The values from the memory and two registers are fed to the three select inputs to select one bit from the truth table. This result is placed on the precharged $ResultBus$. A simultaneous bi-directional bus transceiver connects the $ResultBus$ to the $BroadcastBus$. An inverted version of the $ResultBus$ feeds the three registers and the sense amp.

For speed, we chose circuits that place minimal load on the PE control signals (the SIMD instruction) which are routed in the wordline direction. We anticipate future designs (including DRAM) will have wide arrays with long control signal wires with non-negligible resistance and no opportunity to buffer these signals. Hence, in the circuit designs chosen, the PE control signals drive

[2] We designed a compact SRAM cell before starting the PE layout.

only a single transistor gate which would typically have minimum dimensions.

Logic Family for ALU Implementation

There are several options for implementing the multiplexor used as the PE's ALU. A dynamic-logic multiplexor has several advantages in size, power consumption, and system timing, but requires careful design. A static CMOS multiplexor and a passive CMOS transmission gate multiplexor are also examined below.

The dynamic-logic multiplexor (figure 5.6) uses one PMOS transistor for precharge, 32 NMOS transistors for evaluation, and a CMOS inverter for drive; giving a total of 35 transistors. The 32 evaluation transistors are compactly arrayed as 8 stacks (connected in parallel) of 4 series transistors each. Since most of the evaluation transistors' drains and sources are abutted, only 16 source or drain contacts are required[3]. The 32 evaluation transistors could be replaced by a tree of 2, 4, 8 and 8 transistors (for a total of 22 instead of 32 transistors), but this has the potential to be slower, array less well, and not end up saving area[4].

A static CMOS version of the multiplexor (with actively driven output) would require an additional 32 PMOS evaluation transistors (complementing the 32 NMOS devices for a total of 64 transistors), and significantly more contacts. The PMOS transistors would be configured as 8 groups (connected in series) of 4 parallel transistors each. The PMOS portion does not array as well and requires 36 source or drain contacts ($(8+1)4$) (or long diffusion wires which prevent routing signals in gate-poly). In most IC processes, a contact is longer than a minimum length transistor.

A multiplexor can be built from CMOS transmission gates that passively route one of the 8 ALU-operation bits to the result. This circuit would use a 3-level binary tree with 8+4+2 transmission gates for a total of 28 transistors. The globally distributed ALU-operation signals (buffered only once at each memory array) must drive the transmission gates and result bus in each PE. This large capacitive load would produce long delays; or require large drivers and wide metal tracks to distribute the signal.

Dynamic logic is more difficult to design with than static CMOS. Because dynamic logic circuits store logic values as charge, dynamic logic can be susceptible to errors due to power supply transients, signal coupling and transistor leakage. The extra effort in design and test can be justified if the

[3]Diffusion "wires" could be used as an alternative to some of the source or drain contacts; but the design would be larger due to the diffusion wires and additional contacts between metal and gate-polysilicon; and slower due to the diffusion wire resistance and capacitance.

[4]This tree option is still worth evaluating in each new technology.

circuit will be replicated, as is the case for the PE. After all, memories have been designed with more than 1 billion dynamic circuit nodes.

The dynamic logic multiplexor consumes less power than the other circuits considered. With just over half the transistors of a static CMOS multiplexor, the combined transistor parasitic capacitance being charged and discharged is lower. With non-overlapping precharge and evaluate clocks, there is no crowbar current passing through the evaluation transistors from V_{DD} to V_{SS}. When the multiplexor is not being cycled, the standby supply current is near zero with only leakage current flowing. Likewise, the static CMOS multiplexor consumes negligible power when all inputs remain unchanged in a defined state, but when inputs change, causing a change in output, some crowbar power is drawn. If some inputs are left at an intermediate level, continuous crowbar power can be drawn. This can occur when the bitlines are being pre-charged.

The dynamic logic multiplexor has other advantages in addition to small size and low power consumption: it is fast, acts as a required storage node, and is easy to clock. The multiplexor is fast because it does not have series-connected PMOS transistors (of which the static CMOS multiplexor has 8 in series) which are slower than the NMOS transistors. The capacitive loading on each of the ALU-operation bits (which run the width of a memory block) is limited to the gate of one minimum-dimension transistor per PE. This load is important, since in a large design, the speed at which the opcode signal can be driven across the PEs is much slower than the speed at which the PEs can be clocked[5].

The PE must prevent data from "recirculating" when the ALU destination is also one of the inputs on which the ALU result depends[6]. If a register or memory cannot be both the source and destination [151] then the performance would be reduced to near that of the 16-Function ALU unless more registers are added. Without crippling the instruction set and performance, data recirculation can be prevented with edge-triggered or master-slave flip-flops. Implementing each register as edge-triggered would be expensive, but it is reasonable to put a latch on the ALU output, forming the master stage of a master-slave flip-flop (the data registers form the slave stage). Conveniently, the dynamic multiplexor output is intrinsically latched. The dynamic multiplexor holds its result between the evaluate and the next precharge so that the registers can latch in new data without signals

[5]Simulation shows that each phase (precharge, evaluate, write-register) requires less than 1ns, yet driving a pulse of less than 4ns through a line of PEs is impractical (in terms of driver size and required wire width).

[6]The two data registers and memory are always inputs to the ALU, but for some opcodes, not all inputs affect the result. The critical situation which can produce erroneous results comes when the opcode inverts the destination (data register or memory) for some value of the other operands. This scenario creates an oscillator.

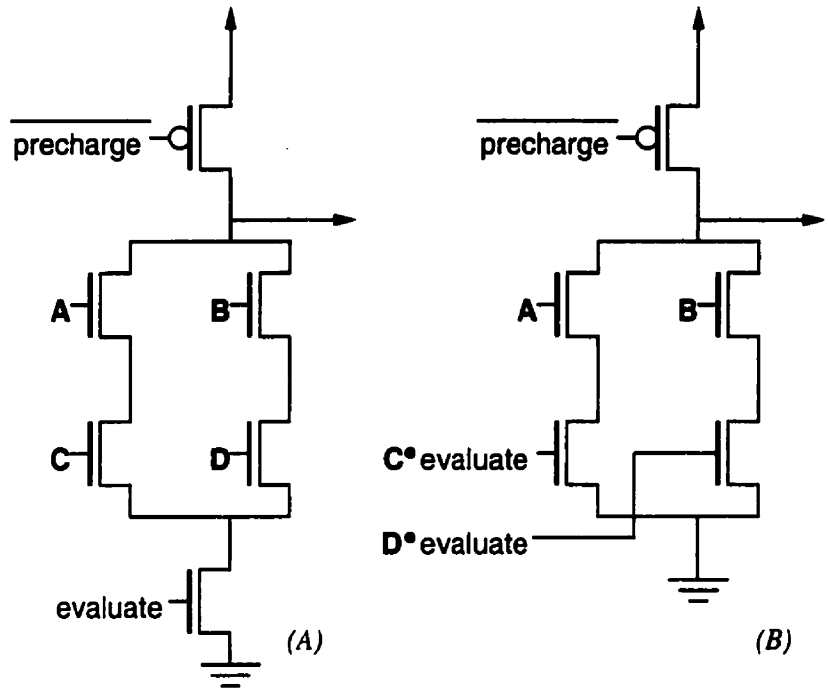

Figure 5.5: (A) Classic dynamic logic, (B) Evaluate signal combined

recirculating. A non-clocked multiplexor would require an additional latch to hold its output or edge triggered registers, either of which would require more transistors.

Using dynamic logic, interaction with the broadcast bus is also simplified by connecting the bus transceiver described later to the dynamic node of the multiplexor output.

One signal and some circuitry have been saved by combining the evaluate clock with the centrally generated ALU-operation bits. Figure 5.5 illustrates (A) a classic dynamic logic circuit which implements the function $\overline{AC + BD}$ and (B) the same function with the evaluate transistor removed and the evaluate signal combined with C and D. The dynamic logic multiplexor in the PE ALU uses a centrally generated 8-bit ALU-operation which is ANDed with an evaluate clock off-chip.

Simulation of the extracted layout showed that if the transistors switched by the clocked opcode bits were placed on the top of the stack of NMOS transistors, instead of at the bottom (connected to V_{SS}), then precharge power consumption was reduced while signal levels were acceptable. Charge-sharing between the precharged node and the stacks of NMOS transistors produces a voltage sag of less than the $V_{threshhold}$ of a PMOS transistor. Testing showed a greater charge sharing problem than observed in simulation using the unverified SPICE models and extraction parasitics (see section 5.1.3). In the high density design, the opcode drives the transistors connected to V_{SS}

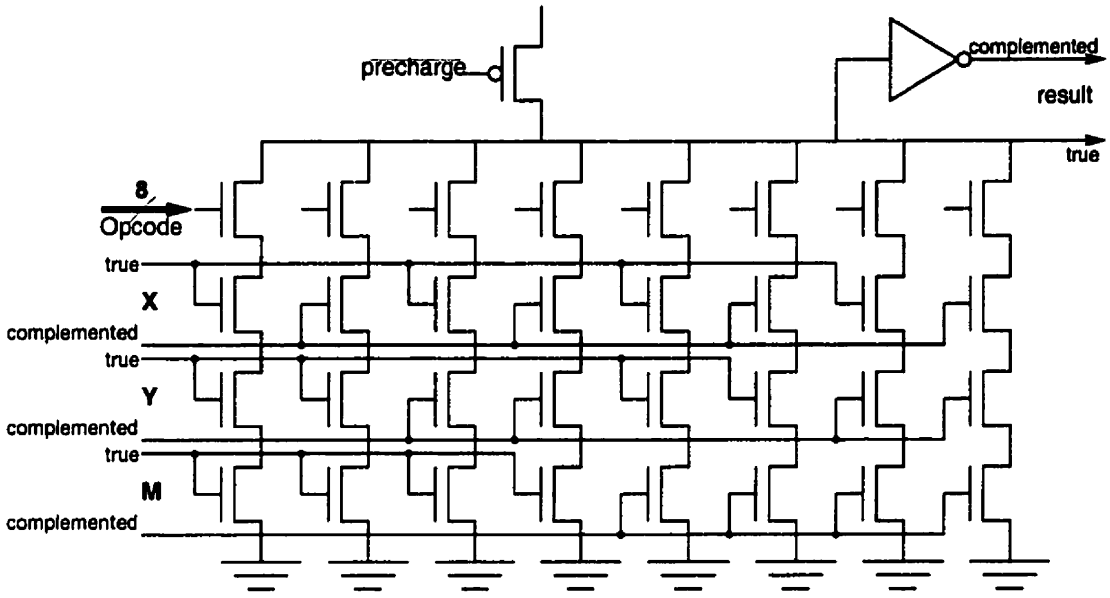

Figure 5.6: Dynamic logic 8-to-1 multiplexor

The 8-bit opcode is gated with an evaluate clock.

(the transistors on the bottom of each stack) to avoid this charge-sharing problem. As long as the precharge is held past the time when X, Y, and M become stable, all relevant circuit nodes in the multiplexor will be precharged. Figure 5.10 shows the multiplexor with the transistors gated by the opcode moved to the "bottom".

Registers

The PE data registers (X and Y, shown in figure 4.10) must retain data for long periods of time. Dynamic latches would lose charge if not re-written, should the host processor (with its higher priority) use the C•RAM for memory access for an extended period. Five-transistor static latches (as shown in figure 5.7) are used in the PE because of their compact size and indefinite data retention (while power is applied). These latches only require a single enable input to the input pass-transistor (and not the complement as well). A strong inverter provides the (inverted) output and a weak inverter provides feedback for data retention. Since the ALU requires true and inverted versions of the data register outputs, another inverter provides the complementary output.

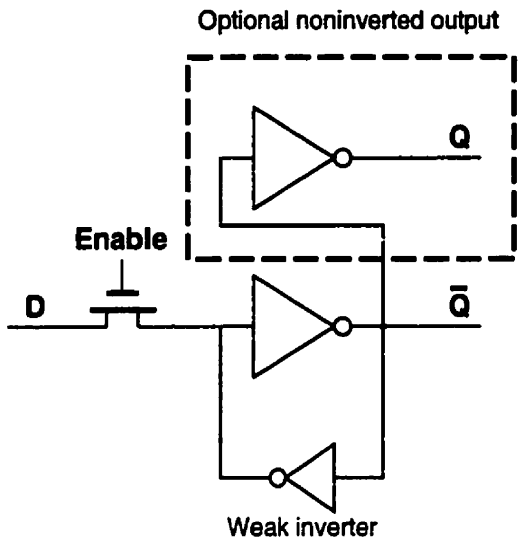

Figure 5.7: PE register

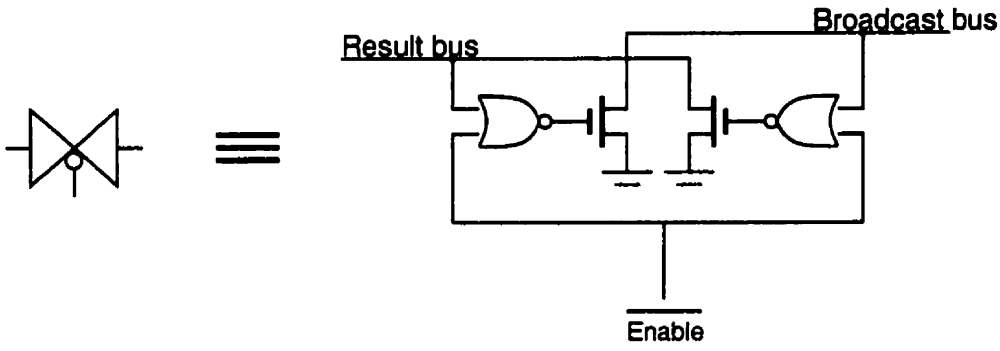

Figure 5.8: Simultaneous bi-directional bus transceiver

Bus Transceiver

The need for a simultaneous bi-directional bus transceiver is described in sections 4.2.1, B.5 and in some of the applications in chapter 6. The simultaneous bi-directional bus transceiver we designed employs a novel circuit design. The first prototype had a passive bus tie implemented as an NMOS pass-transistor which was inherently bi-directional. This meant that the ALU in any selected PE was responsible for driving the broadcast bus and all the other result buses in selected PEs without the benefit of amplification. Since both the PE's result bus and the broadcast bus are precharged high, only a logic 0 need be driven. The circuit in figure 5.8 is used to bridge the buses.

The enable signal comes from the column decoder (figure 5.4). One advantage of the column decoders receiving the pre-decoded column address from off-chip is that bus communication to a

single PE, a group of PEs, or all PEs can be enabled.

In future designs, the column decode will only be used by the off-chip read/write datapath. Instead of using the column decode signal, the bus ties in all PEs will be enabled by a single signal. PEs can be programmed to not participate by putting a '1' on their result buses which will not affect the bus.

Redundancy

Although these PE and communication architectures are compatible with row and column redundancy, redundancy was not implemented on the chip. While the architecture and circuits of the prototype chip were designed with redundancy (for yield enhancement) in mind, we did not have access to an appropriate laser or design rules for the required laser fuses in the CMOS process we were using. Memory row redundancy would be implemented in the same manner as standard memories. Column redundancy in C•RAM, however, affects PE design.

In the prototype C•RAM, a PE plus its column of memory are identified only by the column decoder which controls the bus transceiver. This bus transceiver is used for all IO and interprocessor communications. Using this architecture and standard redundancy techniques to disable the column decoder, a PE/memory column pair containing a defective PE or memory cell could be replaced with with a spare PE/memory column pair.

This simple solution to column and PE redundancy gets more complicated when a separate differential memory databus and 1-dimensional communication are added to the memory and PE designs in section 5.2.

Layout Methodology

The IC layout for this C•RAM design had a vertical stripe of N-well running the length of the PE so that NMOS transistors could be near PMOS transistors. To further decrease the pitch of PEs in future designs, the N-wells will be placed in horizontal stripes so that the PE pitch is not constrained by the N-well design rules.

A second optimization can be made in power distribution. These prototypes used a single pair of heavy wires carrying V_{SS} and V_{DD} horizontally across the row of PEs; and smaller wires to distribute power within the PEs (each PE shared a V_{SS} and V_{DD} wire with its neighbours). Power could alternatively be distributed horizontally to each point along the length of the PE where

power is required. Also, strappings (connections) between same-polarity power-lines could be made periodically where wordlines are strapped. This power routing could potentially save area and permit horizontal routing (of wires) on the process layer formerly used for distributing power within the PEs. In the prototype, instruction signals are distributed only in polysilicon which is resistive. Freeing up a layer of metal from distributing power vertically within the PE would allow instruction signals to be backed up in low resistance metal.

5.1.3 Testing and Performance

Functional 64-PE C•RAM chips with the 256-opcode ALU were fabricated in 1.2 μm double-metal CMOS by Nortel, through CMC. The chips were tested using an IMS-XL60 IC tester, as well as an HP8180A data generator and oscilloscope. Four parallel applications were run on these chips.

The PE and PE + memory speeds were tested separately with V_{DD} = 5 V. The minimum cycle times were 59.8ns for an ALU cycle and 114ns for a "read-modify-write" memory and ALU cycle. This cycle time compares favourably to the Thinking Machine's CM-2's 143ns (7 MHz) cycle for an ALU operation and one of read or write. The maximum cycle time is constrained by the data retention time on the dynamic node of 221ns, beyond which soft errors occur.

A two-operand addition requires two memory accesses per bit and two ALU operations (sum and carry) per memory access (for a read-operate-operate-write cycle time of 114ns+59.8ns). Hence, a 32 bit addition takes 11.1μs.

The charge-sharing (mentioned in section 5.1.2) was more significant than in the extracted simulation — when the opcode was applied, a logic "1" result sagged as much as 2 volts. The noise margins were improved by overlapping the tail of the *PrechargeResult* with the onset of the *Opcode*, allowing the voltage level for a logic '1' result to rise fully. The disadvantages of the overlapped precharge and opcode signals are wasted (crow-bar) power and time.

Data-parallel programs written in C++ (see section 6) including fault simulation, satisfiability problem, masked pixel copy, and memory test were compiled for this C•RAM and run using the IMS tester as a controller and sequencer to issue SIMD instructions.

5.1.4 Summary

The 1.2μ CMOS prototype C•RAM was useful for exploring architectural and VLSI characteristics of PE logic pitch-matched to memory. While we didn't consider this low-density prototype 64-PE chip to have commercial potential, the 64-PE PIM [58] was built into a high-end

256 K PE immersion-cooled system. Some features of our prototype are not expected to be placed in future designs. The row and column group-select of memory rows and columns is easy to implement, but requires extra pins. After gaining experience from writing applications for C•RAM, we don't expect the group-select feature to be used often, and its function can be simulated in software with only tens of cycles of overhead.

For interprocessor communication, these designs contain the broadcast bus, but not the left-right communication (shift register). The left-right communication was omitted, favouring easier future use of column redundancy. Performance simulation of applications (chapter 6) has shown the need for this interconnection. A charge sharing problem was detected during chip testing which was not evident from layout extraction and simulation; and required the C•RAM to be run at reduced speed. The solution requires that the order of several transistors be changed, resulting in slightly more power being drawn each time the PE is cycled. The PE, containing a vertical band of N-well, is sufficiently narrow to fit in the width of a 6-transistor ASIC SRAM cell. To make the PE narrower still, subsequent IC layouts use horizontal bands of N-well.

Through innovative circuit design and layout we have integrated 64 processing elements, each with 128 bits of memory, into a small die. The majority of SIMD machines produced by this date (see table 2.1) integrate fewer PEs and no memory. The aggregate performance of 32 Mbytes of this prototype C•RAM is 188 GIPS for 32 bit addition. The PEs are the pitch of a memory cell and occupy less than 9% of the chip area. The knowledge gained through the fabrication of prototypes has been well worth the effort spent.

5.2 High Density Design

We demonstrate the compatibility of C•RAM PEs with commodity DRAM memory by laying out PEs in a 4 Mb DRAM technology. Through MOSAID Technologies, I was fortunate to have access to a proprietary DRAM design and its IC process design rules.

DRAM memory cells are smaller than SRAM cells, and consequently 4 sense amplifiers fit in the pitch of one PE (less than 20μm). The design we developed has a cycle time of 150 ns for read, two ALU operations and write-back. The cycle times of DRAM operations are not significantly increased. With fewer PEs for a given amount of memory, the DRAM version has lower performance than the SRAM prototype, but its density (and expected manufacturing cost) makes it a more economically viable chip.

The following subsections describe the memory, PE-memory interface, PE, power concerns, the external interface, and future enhancements.

5.2.1 Reference DRAM Architecture

The *high density DRAM-based C•RAM* we describe in this dissertation is based on a 4 Mb DRAM designed at MOSAID. It uses similar circuits to those described by Gillingham et al. [94]. Each sense amp is connected to 256 bits of memory, 128 cells per bitline (plus redundancy). The row-address cycle time is 110 ns.

The reference DRAM design supports the standard JEDEC DRAM features such as "$\overline{CAS}$-before-$\overline{RAS}$ refresh" and "write-per-bit" [94]. This DRAM is organized externally as 1 M × 4 bit. It has 16 basic memory arrays, each with 1024 sense amps and bitline pairs. Each bitline pair is connected to 256 memory cells. The chip requires 1 K refresh cycles per refresh interval, which implies that one quarter of the 16 K sense amps are powered in a given $\overline{RAS}$ cycle. The datapaths, from the bitlines through to the data buses which connect to the peripheral logic adjacent to the bonding pads, are fully differential for maximum speed. In a read, 2 bits are drawn from each of 2 adjacent basic memory arrays and directed to the 4 IO pins. This design uses isolation transistors with gates that swing between V_{DD} and V_{PP}.

Given that 2048 PEs are to be added to this memory, each PE has 2048 bits of local memory.

5.2.2 C•RAM Memory

Some minor organizational changes were made to the reference memory architecture to accommodate the C•RAM PEs. Additional pins were added for the additional C•RAM functionality, although the basic external 1 M × 4 bit interface to the DRAM is left unchanged. Since every PE must be connected to an active sense amp, and this single-metal process has no extra layers of interconnect available to connect the sense amps of multiple arrays, the PEs can at best be connected to two basic memory arrays (one array above and one below as shown in figure 5.9). This organization requires that twice as many sense amps be powered vs. the original DRAM design. However, compared to the alternative of cycling all memory arrays (with one row of PEs per memory array), the proposed scheme only uses half the sensing power. Since changes to this datapath are insignificant (die size, wire lengths, and capacitive loading are increased slightly) off-chip memory access times will be barely impacted (<2%).

With the left-right communication network, memory column redundancy requires special attention. The left-right communication network must bypass PEs which are either defective or are connected to defective memory columns. The correct correspondence between memory address and PE connections to neighbours can be maintain using the column decode scheme reported in [90] which preserves a linear ordering.

The PEs work with variables one bit at a time, whereas the host will require all bits of a variable together as a word. In order to minimize the degree to which data must be transposed when transferred between host and C•RAM, the memory architecture is altered such that an off-chip read fetches the entire 4-bit memory word from 4 adjacent columns of the *local memory* of one PE.

The high-density C•RAM design provides the same memory interface and virtually the same performance as the reference DRAM by maintaining the same circuits in the off-chip memory read/write datapath and most of the control circuitry.

5.2.3 PE-Memory Interface

As noted above, the PE is located between two memory arrays. Figure 5.9 shows the circuitry used to multiplex the data from 8 sense amps to the PE. The 8 PE column-select signals are produced by decoding the most significant bit and the 2 least significant bits of the PE local memory address. The memory data path between the read amplifier and write amplifiers, both within the PE, and the sense amps uses differential signaling.

The write amp in the PE, used for writes from the PE to local memory, must overdrive the selected sense amp and is implemented as a tri-state differential driver. The data for the drivers comes from the *result* bus, while the enable signal is the product of *group_write* and the write-enable register.

The read amp takes the differential signal from the sense amps and produces true and complemented signals representing memory data. With the V_{DD}-V_{PP} switched isolation transistors used in the reference memory design, simple CMOS inverters could be used to amplify the sense amp nodes. These inverters, however, would draw crowbar current when the bitlines are precharged. Instead, the design uses CMOS NAND gates, gated by a qualifying signal (*PE_read*). When the read amplifier is disabled, the memory data signal and its complement are set to '1', aiding in precharging the dynamic multiplexor.

The need for a read amp can be eliminated if latched sensing [107] is employed. With latched sensing, the sense amp nodes quickly approach V_{DD} and V_{SS}, and can be used as the memory data

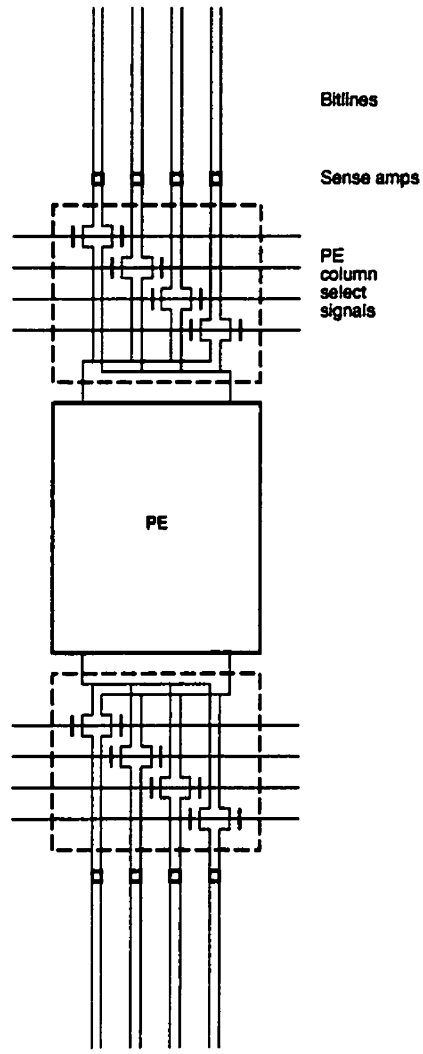

Figure 5.9: PE column decode multiplexor

and complement signals. Used in C•RAM, latched sensing also saves power by avoiding restoring the bitline voltages only to have some of those sense amps overwritten with new data by the PEs later. With latched sensing, this needless rewriting can be avoided by postponing the bitline-restore operation until the PEs have written their result to the sense amps.

5.2.4 Processor Element

The PE for the high-density design has only a few changes relative to the prototype. These changes lie in the multiplexor, registers, and broadcast bus. As with the prototype, the high-density C•RAM PEs are placed in a row in the word-line direction. With the narrow PE pitch and the long wordlines, the control signals for a row of 512 PEs must be driven from one driver without additional

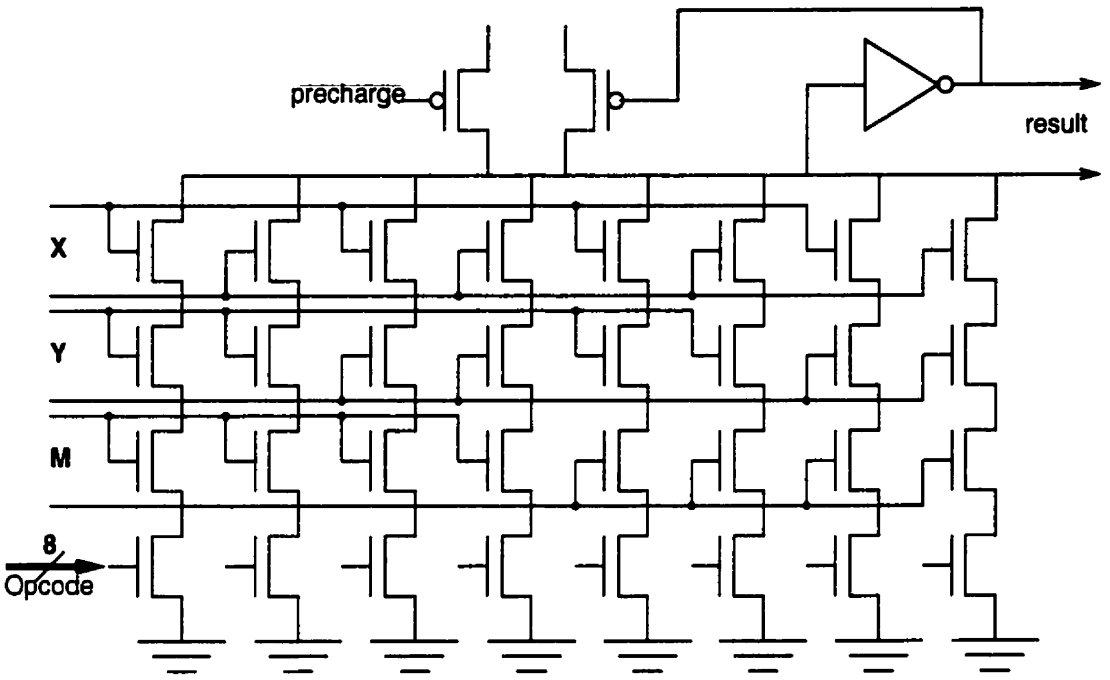

Figure 5.10: Dynamic logic 8-to-1 multiplexor

buffering. The rise and fall times for these control signals dominate the PE cycle time. As was the case for the prototype, all PE control signals (the SIMD instruction) drive a single transistor per PE — usually a transistor of minimum dimensions and load. Like the wordlines, the PE control signals are routed in polysilicon and are backed up in metal with strappings at the wordline strapping locations.

The dynamic logic multiplexor, shown in figure 5.10, has been reorganized to solve the charge-sharing problem. The 8 transistors gated by the opcode have been moved to the "bottom" (connected to V_{SS}) to allow all circuit nodes coupled to the *result* node to be fully precharged. Since the precharge phase ends after the multiplexor inputs become stable, only circuit nodes that will not become coupled to the *result* node during the evaluate phase (and don't need precharging) may not get precharged. A weak "latch-back" PMOS transistor has been added in parallel with the precharge transistor. Without this latch-back, charge injected on the $\overline{result}$ signal from the write-register signals would capacitively couple through the multiplexor's inverter and raise the voltage of this dynamic *result* node above V_{DD}. This permits charge to leak out of the dynamic node through the forward-biased source — N-well junction of the precharge transistor. Without the latch-back transistor, after the register-write signal returns low, the voltage of the dynamic node would dip. No similar problem was detected when the dynamic node was low, since the register-write signals are

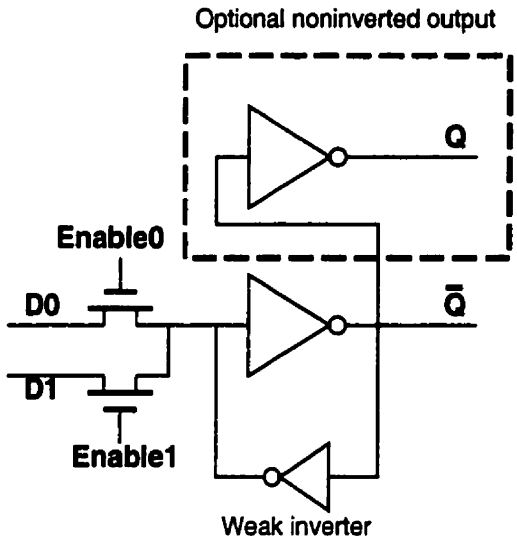

Figure 5.11: PE register with second input

positive pulses.

The high-density C•RAM design implements "left-right" communications between adjacent PEs, as shown in figure 4.10. The simple circuit implementation of this communication, suggested by Hall [149], adds one transistor to each of the X and Y registers. Each latch (shown in figure 5.11) has a second input transistor used to latch in data from the result bus of the neighbouring PE. Using different registers for receiving data from each direction balances the loads on the two registers thereby matching their speeds. For chips designed for lower voltage power supplies than 5 V, a different latch would be used that does not rely on an N-MOSFET to pass a logic level "1".

In high-density C•RAM, the bidirectional bus transceiver is controlled differently than in the prototype. On the prototype chip, the datapath for off-chip memory accesses runs through the PE, requiring the column decoder to enable the bidirectional bus transceiver in each PE. For this high-density design, the datapath for off-chip memory accesses does not involve the PE. As previously noted, the configuration of the column decoder to select single PEs or groups of PEs for interprocessor communication can be emulated in software with negligible overhead. With the transceiver no longer having to be addressable, all transceivers are now controlled by a single *BroadcastBusEnable* signal in the SIMD instruction word.

The PE design fits in the pitch of 8 columns of memory (4 sense amps) and adds 17% to the height of the memory core. With additional buses, drivers, and peripheral circuits considered, C•RAM is 18% larger than the reference DRAM chip on which it is based.

5.2.5 Power

Power consumption and power supply noise are important concerns if a chip design is to be produced and perform reliably. Power consumption is important to system design. Excessive power consumption can constrain packaging as well as limit which applications C•RAM could be used in. Power supply noise can affect the function of the individual part. The circuitry added to the memory (primarily the PEs) must not generate power supply noise detrimental to the existing memory design and must be tolerant of existing power supply noise.

In DRAM design, there is a trade-off between power consumption and refresh overhead. In low power DRAMs, fewer memory arrays are enabled, thus fewer sense amps per chip are powered, but these require more refresh cycles (e.g. 2048 instead of 512) during the memory cells' maximum data retention time (e.g. 16-64 ms)[7]. In C•RAM, the PEs must be connected to an active memory array during each operate cycle. As mentioned in section 5.2.2, to conserve power in the 4 Mb design, the PEs reside between 2 arrays, with only one of the two arrays powered during a given operate cycle. In a 4 Mb DRAM with 256 bits per sense amp, the PEs' requirement for memory dictates a 512 cycle refresh. To operate with full PE utilization, every PE must be connected to an active memory array. This can be accomplished by having many active memory arrays and higher power consumption, or by having each PE connected to more than one memory array (permitting fewer PEs per chip) with one of those memory arrays being active. The latter option is discussed in more detail in section 5.2.7. Apart from cycling more than the conventional number of memory arrays, the additional circuitry consumes little additional power.

The sensing power typically dominates the C•RAM chip's power consumption followed by output buffers[8] and wordline drivers. A comparison between PE and sensing power can be drawn which is relatively independent of technology. A PE (with far fewer devices than a bitline) has less internal capacitance than a bitline. A PE has 7 wires in the column direction, thus requiring a minimum pitch of 8 bitlines. Therefore, there are 8 active bitlines (4 active sense amps in a folded bitline architecture) per PE. The PE signal swing is V_{DD} to V_{SS}, with a 50% probability of a transition on the main result bus. The bitlines, however, are precharged to $\frac{V_{DD}}{2}$ consuming half the charge to restore the level, but switching every cycle (one of the pair going to V_{DD}, one to V_{SS}). Recall that the energy dissipated[9] in charging plus discharging a capacitor is $E = (C\Delta V)V$.

[7] These lower power DRAMs also have less page "depth".

[8] Output buffers may dominate the power consumption, depending on number and load but during a C•RAM operate cycle the output buffers are not used.

[9] assuming the use of standard dissipative circuits as opposed to adiabatic circuits [152]

Ignoring *crowbar current* (the current conducted from power to ground directly without charging the capacitance at a circuit node), the ratio between sensing current and PE current is:

$$\begin{aligned} E_{PE\ cycle} &= \frac{1}{2}C_{PE}V^2 \\ E_{bitline\ cycle} &= C_{bitline}\frac{V}{2}V \\ C_{PE} &< C_{bitline} \\ E_{PE\ cycle} &< \frac{1}{4}(4\ E_{bitline\ cycle}) \end{aligned}$$

Hence, the average energy consumed by a PE per operate cycle ($E_{PE\ cycle}$) is less than a quarter of the energy consumed by the adjoining 4 sense amps ($4\ E_{bitline\ cycle}$). Since the sense amps draw more crowbar current than the PEs, the difference between their energy consumption is actually greater. At the beginning of sensing, before there is a significant voltage difference on the sense amp nodes, most of the sensing current is conducted from power to ground with little current going to charge the bitlines. The dynamic logic used by the PE, however, has (ideally) zero crowbar current. With mostly minimum-sized transistors in the PE, the loads are small and switch quickly, minimizing crowbar current.

DRAMs have some of the noisiest on-chip power buses. To avoid problems, high current (and high $\frac{di}{dt}$) operations are scheduled where they will cause little interference with other circuits' power demands and voltage references, and the circuits themselves are designed to operate over the voltage extremes seen within these chips. Similarly, the additional C•RAM circuitry must not be susceptible to power supply noise and must not interfere with the existing DRAM operations.

C•RAM's circuits are tolerant to significant power supply noise and the PEs are inactive while the power rails are the noisiest. While the DRAM must sense small differences in bitline voltage and drives low voltage signals across the chip, all PE control signals have a full V_{DD} to V_{SS} swing giving them large noise margins. When C•RAM's PEs are active, the row decoders are stable and the data pad output drivers are unused. A single output pad driver, however, may be used for the broadcast communications bus or nearest-neighbour communications, but only after the ALUs have arrived at a result. The sense amps will be drawing current to restore bitline data during PE operations, but this occurs after the extreme positive spike in $\frac{di}{dt}$ and the highest currents found at the start of sensing.

Since C•RAM avoids overlapping operations with the DRAM which C•RAM is designed to accommodate, the PEs' power demands interfere little with the existing memory. Since the PEs don't perform an operation until they have a value from memory, and the PEs' read amplifiers require

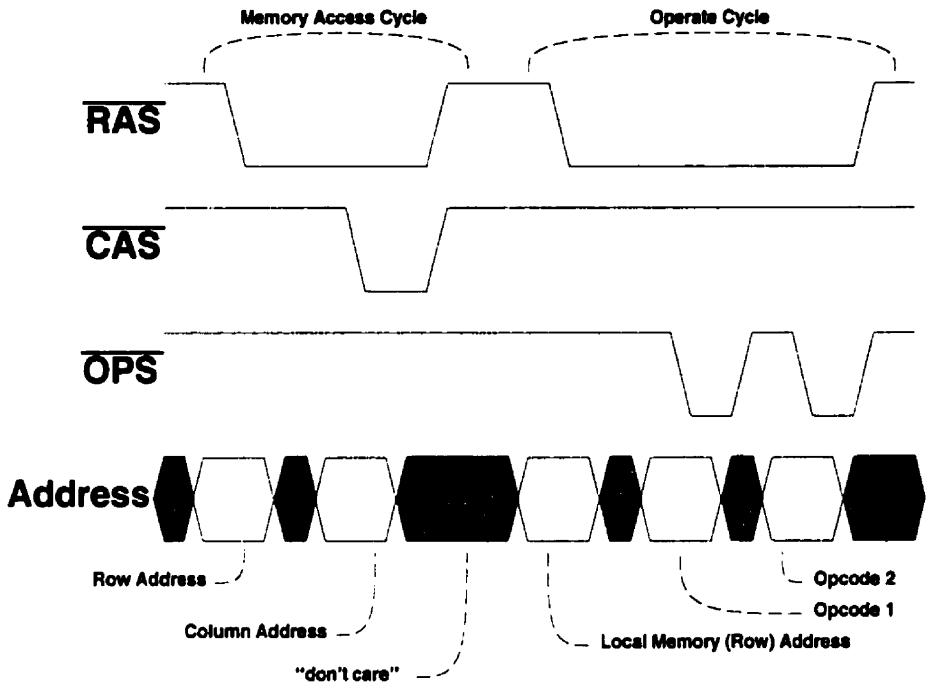

Figure 5.12: Timing of a memory access followed by an operate cycle

a significant voltage differential (>1V), the sense amplifiers already have too great a noise margin for a bit of data to become corrupted.

By design, the C•RAM logic and memory are compatible. The power requirements of the PEs are a fraction of that of the memory.

5.2.6 External Interface

The C•RAM chip interface can be based on most standard DRAM interfaces. This high-density C•RAM uses the $\overline{RAS}$-$\overline{CAS}$ interface provided from the reference 4-Mb DRAM design. Pins are conserved by multiplexing instructions through the address lines. C•RAM behaves as a standard DRAM during a $\overline{RAS}$-$\overline{CAS}$ cycle, but takes in a row address and an instruction during a $\overline{RAS}$-$\overline{OPS}$ (operate strobe) cycle. Multiple $\overline{OPS}$ cycles can occur while $\overline{RAS}$ is held low. As viewed from the pins, the chip is organized as 1 M×4 bits.

Figure 5.12 shows the timing of a $\overline{RAS}$-$\overline{CAS}$ memory cycle, followed by two C•RAM operate cycles. When $\overline{RAS}$ falls, the row address is latched in and memory rows are accessed and sensed — regardless of whether a $\overline{CAS}$ or $\overline{OPS}$ cycle is to follow. The row address acts as a row address for a $\overline{CAS}$ cycle or as a local memory address for an $\overline{OPS}$ cycle. On the falling edge of $\overline{OPS}$ the C•RAM SIMD instruction (multiplexed through the address lines) is latched, decoded and sent to

8	opcode (ALU function)
1	broadcast bus enable
3	destination, decoded 3-to-8 to select one of write enable register X register Y register shift left input for X register shift right input for Y register shift up: shift left input for X register with shift up/down pins selected shift down: shift right input for Y register with shift up/down pins selected no destination
2	The 2 least significant local memory addresses can be received on the edge of $\overline{RAS}$ or $\overline{OPS}$. *optional*
1	during an $\overline{OPS}$ cycle, the chip WE pin functions as PE memory write
12-14	bits of instruction total
2-4	IC pins in addition to the existing 10 address pins

Table 5.1: C•RAM SIMD instruction bits

all PEs. Pins are conserved by reusing existing standard DRAM pins and by using a binary encoding for the ALU-result destination. The C•RAM instruction is provided to the chip as described in table 5.1. As an alternative to adding pins, some of the SIMD instruction bits could be multiplexed through datalines.

Of the 11 bits of address for the 2048 bit PE local memory, 9 bits are required at the onset of the row-address cycle for selecting the array above or below the PEs (1 bit) and 1 of 256 wordlines (8 bits). The remaining 2 bits of address may optionally be issued with the instruction. If the sequence of C•RAM memory accesses exhibits temporal locality, then the data for the next memory access may already be latched in a sense amp. Indeed, the 4 sense amps behave as a one-line cache of 4 words of 1 bit each [10]. The increased performance comes at a cost of 2 additional pins per chip. Without this caching, the PEs are connected to only 1 in 4 sense amps per memory cycle, permitting a maximum utilization of 25% of the internal bandwidth. If the additional two pins are used to select one of the 4 sense amps during each operate cycle, then, in an extended 180 ns memory cycle containing 4 operate cycles, the PEs can access all active columns of memory to achieve 61% bandwidth utilization.

In addition to delivering the instruction, other functions and their pins are required as listed in table 5.2.

[10] The size of this cache can be further increased by not precharging the sense amps on the array connected to the other side of the PE.

4	bidirectional communication pins (left, right, up, down)
1	serial IO pin *optional*
1	global broadcast bus, pin doubles as D0
2	additional V_{DD} and V_{SS} pins if low internal noise is required *optional*
1	$\overline{OPS}$ signal
5-8	additional IC pins

Table 5.2: Other additional pins for C•RAM

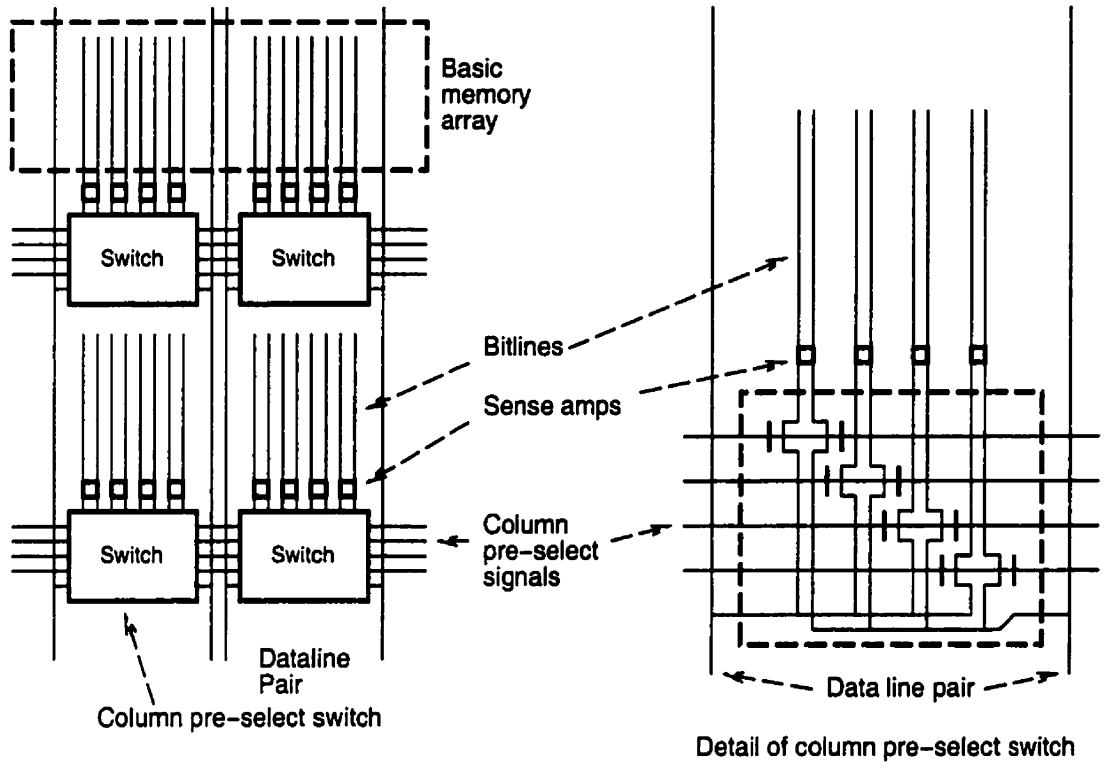

Figure 5.13: Memory with bitline direction datalines

Our reference DRAM architecture uses a $\overline{RAS}$-$\overline{CAS}$ interface and the C•RAM interface builds on top of that. Interfaces for C•RAM built on EDO, synchronous DRAM, and Rambus DRAM are similarly straight forward.

5.2.7 Multiple Memory Arrays per PE

So far, we have discussed abutting PEs to 2 memory arrays so that, in a given cycle, each PE is connected to an active memory array while the other array is inactive, conserving power. Connecting a row of PEs to more than 2 memory arrays cannot be done by simply abutting them, but requires another level of interconnection running among the multiple memory arrays.

Fortunately, the 16 Mb generation of DRAM as well as many designs beyond this density have

an additional layer of metal interconnect which is currently used for column decode. The column decode signals are routed, in the bitline direction, on top of the memory arrays so that one set of column decoders can service more than one basic memory array. This layer of metal is also typically used to carry power wires interspersed with the column decode signals.

Instead of bringing the column decode signals to the memory arrays, this metal layer can be used to bring the data to the edge of the group of memory arrays. We call this memory architecture, shown in figure 5.13, *bitline direction dataline*. Because higher routing layers typically have wider minimum routing pitches, and the bitlines, in a lower routing layer, would be designed to have nearly the minimum pitch, we would expect a differential dataline pair to be wider than a bitline pair, suggesting a (power-of-2) ratio of 1 dataline pair per 2 bitline pairs. Furthermore, if power lines are to be interspersed with datalines, a ratio of 1 dataline pair per at least 4 bitline pairs is reasonable. This is compatible with the ratio of 4 sense amps per PE mentioned earlier in section 5.2.4.

Using this bitline-direction dataline memory architecture, the PE can be connected to more than 2 memory arrays. Because each active PE needs an active memory array during a read or write, connecting PEs to more memory arrays provides the option of decreasing the number of active memory arrays, and therefore the chip power consumption. This also provides flexibility in the design for increasing the amount of local memory per PE. This architecture is compatible with staggered sense amps found in some designs in the 16 Mb generation and up. Although the IC process and memory cell remain the same, changes must be made to the memory array and datapath.

5.3 Summary

We have demonstrated circuits for a PE that are well suited to be arrayed together to form a row of narrow PEs. The PE can be implemented with a pitch of 7 wires at its widest, which suggests that the PE can be pitch-matched to 8 bitlines over a range of IC process feature sizes. The area overhead of the PEs ranges from 9% to 18%. C•RAM chips containing these circuits have been built, tested, and have run applications. The PEs and related overhead add less than 25% to the die area of an existing DRAM design. The addition of few pins and little additional area make the design suitable for DRAM-style packaging, including SIMMs or DIMMs. The energy required by a C•RAM PE cycle is less than a quarter of that required by a memory cycle for sensing. The narrow PEs are a requirement for not only having a small impact on chip area, but for utilizing a good fraction of the memory bandwidth. By placing a PE next to 4 active sense amps, 25% to 61% of the memory

bandwidth can be utilized effectively.

Chapter 6

Applications

It is not difficult to build hardware capable of performing billions of operations per second on contrived problems such as vector addition (used in the previous chapters to make architectural decisions). The utility of a particular computer architecture is better demonstrated on real-world applications and kernels.

Applications most suited for massively parallel SIMD machines such as C•RAM have fine grain parallelism and regular communication patterns. In this chapter, we examine a number of such applications and compare the simulated performance of the applications executed on 32 MB of C•RAM (128 K PEs) to the measured performance of those applications executed on conventional workstations [153].

The applications shown in this chapter use fixed-point arithmetic. The C•RAM architecture can also perform floating point operations, but with approximately a factor of 10 decrease in throughput compared to operations on short integers [154].

6.1 Programming Model

The applications programmer should have at least an abstract understanding of the C•RAM architecture. Any memory location in C•RAM can be read and written by a host CPU during an external memory cycle. During an operate cycle, all PEs execute the same common instruction and optionally access the same memory offset within their private partitions of memory. In other words, C•RAM is a SIMD processor with distributed non-shared uniformly-addressed memory.

Three forms of interprocessor communication are available. First, the broadcast bus can perform combine operations among all PEs in the system. The primitive boolean-AND combine operation

performed by the broadcast bus can be used to perform higher-level combine operations such as finding the maximum element of a parallel variable. Second, communication can be uniformly routed via a 2-dimensional grid. The contents of a parallel variable can be shifted past a *scalar* number of PEs in the X or Y directions. Third, the host can explicitly copy between arbitrary memory locations associated with different PEs. Since this host-initiated reading from and writing to memory is not parallel, it is slow and therefore of little importance for PE-PE communications.

C•RAM Compiler

The C•RAM programming environment includes a compiler and a simulator. The compiler generates code for the host processor which in turn issues SIMD native instructions to C•RAM. The simulator decodes the SIMD instructions and simulates them on each PE at the register-transfer level.

The C•RAM high level language, based on *C++*, abstracts groups of related memory locations as *parallel variables*, which are implicit arrays with each element of the array in the private memory partition of a different PE. The number of elements in a parallel variable is equal to the number of PEs in the system.

The source code for all applications, the compiler users' manual, and the compiler implementation details are available in [147].

6.2 Experimental Method

For each application in this chapter, we describe the serial and parallel algorithms and their execution times. The execution times of the applications and kernels are measured on workstations for sequential execution and on a simulated DRAM-based C•RAM system. To obtain the C•RAM execution times, each sequential application program was rewritten as a data-parallel program.

The C•RAM simulator simulates at the register transfer level for a specified number of PEs. The simulator was given the timing parameters for the 4 Mb DRAM-based C•RAM[1]. The simulation assumes that 32 Mbytes of C•RAM, containing 128 K PEs, is available. 32 Mbytes was chosen as a modest amount of memory for a workstation today — a larger memory would give proportionately

[1] While the DRAM-based C•RAM has lower performance per MB of memory than the SRAM-based C•RAM, its higher density and lower cost per Mbyte make the DRAM-based C•RAM more attractive for commercial development. For this reason, the simulated performance is given for DRAM-based C•RAM.

higher performance for most applications. DRAM refresh overhead is accounted for, although the refresh overhead for 512 refresh cycles of 150 ns every 32 ms increases execution time by only 0.19%. In fact, many applications require no refresh since they access, and therefore refresh, all memory locations which contain data more often than once every 32 ms.

The C•RAM controller is assumed to be able to issue instructions as fast as the PEs can execute them. Issuing two instructions in a 120 ns memory cycle or issuing identical shift instructions once every 15 ns is not a significant technical challenge; several of the SIMD instruction sequencers described in [147] are capable of this performance.

The SUN Microsystems workstation used for sequential measurements was a SPARCstation 5 with a 70 MHz microSPARC II processor. Faster memories and workstations have been introduced since the microSPARC II and the 4 Mb DRAM generation, but these are comparable platforms (in fact the microSPARC II was introduced after the 4 Mb DRAM). The programs were compiled using the AT&T C++ version 2.1.0 translator and the SunOS Release 4.1.3_U1 C compiler with optimization level 4.

To conservatively compare C•RAM simulated execution with real sequential[2] execution on equal terms, we examined several sources of experimental error. The C•RAM execution has no overhead from interrupts, demand-paged memory, operating system, or other processes competing for the CPU. We executed the sequential programs on lightly loaded or unloaded machines to minimize interrupts and paging. Virtual memory usage was kept well below the amount of physical memory installed in the test machines to eliminate or minimize paging. Each application was run twice: once to page in all necessary memory, and a second time for timing with minimum paging effects. To further reduce measuring any overhead or start-up effects in sequential applications, we record only the 'user CPU usage' using the `getrusage()` operating system call before and after the routine.

To keep the measurement error small, under 1%, we ensure that the sequential applications run times are greater than 1s, by executing the application subroutine multiple times in a loop if necessary. Times that are counted in the sequential execution time but not the C•RAM execution time are: the overhead of one loop iteration (120 ns); a function call (210 ns); and the overhead of calling `getrusage()` (3100 ns). In our analysis of the experimental error in measuring the sequential execution time, we found that the 10 ms resolution of the 'user' CPU time dominates all other sources of error. Measurements were also repeatable to within this resolution.

For all of the applications and kernels considered, the problems sizes were chosen to fit the the

[2] Some workstations, although referred to here as "sequential machines", contain super-scalar microprocessors.

program	C•RAM IO (ms)	C•RAM (ms)	host (ms)	speedup
3x3 Convolution 16M	32.70	17.6067	112760.0	6404
Vector Quantization	2.56	25.7460	33780.0	1312
LS Matching	included	0.2003	250.9	1253
Data Mining	included	70.6600	192450.0	2724
Fault Simulation	included	0.0894	2380.0	26626
Satisfiability	included	0.0232	959.0	41391
Memory Clear	not applicable	0.0016	8.8	5493

Table 6.1: Performance summary

128 K C•RAM PEs in order to maximize the speedup. This is a reasonable approach since many users choose a problem size appropriate for a machine, or for high-end applications, configure the machine for a problem, thus achieving the best performance/cost. If a more conservative approach were taken, the runtimes for the optimal problem size + 1 would approximately double, halving the performance improvement of C•RAM vs. running the optimal sized problem.

Table 6.1 shows the applications and kernels for which we have simulated C•RAM and measured workstation run times. Speedup is the primary performance metric used. For most applications, IO time is included in the C•RAM run time. The least-squares record matching application relies on a memory-resident database for both the sequential and parallel versions. IO time is not applicable to the memory clear, since most users would not copy their data *onto* a SIMD processor to have it cleared. For the remaining applications, the total time taken for input and output is given[3]. IO does not pose a bottleneck for C•RAM running these applications.

6.3 Signal Processing and Graphics Applications

Signal processing and graphics are fields rich in applications for SIMD computers. Here, communication is usually required within a small neighbourhood of any given pixel (in 2 or 3 dimensions) and the parallelism often extends down to the pixel level.

[3] As these are video applications, if C•RAM is used in a video frame buffer then the IO time can be partially or completely discounted.

6.3.1 Convolution

Introduction

Convolution on a 2-dimensional image has many uses, including filtering noise. The discrete 2-D convolution of array P with filter W ($\acute{P} = W \otimes P$) can be expressed as:

$$\acute{P}_{x,y} = \sum_i \sum_j W_{i,j} P_{x-i,y-j} \tag{6.1}$$

for arbitrarily sized arrays. The size of this non-zero region of W is referred to as the kernel size.

Significance

For image processing, 2-D convolution can be used to attenuate noise, or smooth artifacts of other processing such as image compression.

Sequential Algorithm

We perform a convolution using a 3×3 kernel on a 16M-pixel monochrome image with 8-bit resolution. Each new filtered pixel is the weighted sum of the original pixels in a 3×3 square centred around the new pixel's coordinates (i.e. self and neighbours in 8 directions). We define the values outside of the boundaries of the image to be zero. The results of the convolution are written back "in place" to the same array, but with an offset, overwriting data which is no longer required, thus saving memory when compared to a simpler algorithm which uses two separate arrays.

Parallel Algorithm

The parallel algorithm is the same as the sequential algorithm. The image space is mapped to the PEs in 128 × 1 stripes. The algorithm uses scalable left-right interprocessor communication (described in section 4.2.1) and the 2-D communication algorithm (described in B.6) to obtain the values of neighboring pixels. The boundaries are fixed by sending zeroes into the edges of the communication network, possibly by grounding these pins.

Results

The workstation requires 113s to perform the convolution on the 16M pixel image while C•RAM can perform the convolution in 17.6ms, or 6400 times as fast as the workstation. For this size of image, C•RAM can perform the convolution in slightly more than half the refresh interval of a 30 Hz video signal.

6.3.2 Vector Quantization Image Compression

Introduction

Vector quantization is a method of data compression which is applicable to many forms of data. This compression can be done in a lossless or lossy (inexact) way. Vector quantization is a suitable choice for data compression when data items in a neighbourhood exhibit a correlation. It is particularly well suited to still and moving image compressions because adjacent pixels in an image tend to be correlated [155, 156]. In the algorithm, pixels are grouped into blocks called *vectors*; which are mapped to an entry in a *code-book* of vectors. The index of the code-book entry is the compressed output. Decompression is accomplished by using the code-book index to retrieve the appropriate vector from an identical codebook. The compression is termed *lossy* if, after compression and decompression, the data (image) is only an approximation of the original. If lossy compression is acceptable, which is the norm for image compression and is what we implement, then the code-book can be much smaller than the set of all unique vectors in the image, resulting in a higher compression ratio. In this case, the index is determined by choosing the *closest match* between an image vector and the vectors in the code-book.

Significance

Vector quantization has applications in data compression for still and moving images and can be used for multimedia, teleconferencing, and digital broadcasting of images. Vector quantization image decompression has the advantage that it is implemented as a straight forward table look-up which can be done on microprocessors without special hardware [157]. Image compression, however, is computationally challenging for real-time applications since a linear search for the closest match through the entire code-book is necessary for each vector. Faster approximate algorithms are not guaranteed to find the closest match, so we do not use these.

Sequential Algorithm

The vector quantization image compression algorithm translates vectors of pixels into indices into the code-book. A sum of absolute differences or sum of squares of differences (L_1 or L_2 norm) can be used. When comparing the differences of the image vector and code-book vectors for the closest match, Panchanathan[158] argues that the L_1 and L_2 norms provide similar quality compression. We chose the L_1 norm because it requires less computation. The sequential algorithm is as follows:

```
CompressImage( image, compressed_image )
    for all vectors in image
        for all entries in code_book
            calculate L1 norm of (vector - code_book_entry)
            if this codebook entry is the best fit seen so far
                record L1 norm and code_book index
        output code_book index to compressed_image
```

Parallel Algorithm

To parallelize this algorithm, we divide the computation spatially over the image. Each PE has one vector and finds the best-fit codebook entry.

```
CompressImage( image, compressed_image )
    for all vectors in image in parallel
        for all entries in code_book
            calculate L1 norm of (vector - code_book_entry)
            if this codebook entry is the best fit seen so far
                record L1 norm and code_book index
        output code_book index to compressed_image
```

In our C•RAM implementation of the algorithm, each PE's memory holds the values of the pixels corresponding to its vector(s). The code-book is held centrally by the controller and is "broadcast" to all PEs, one bit at a time, in the process of finding the L_1 norm.

Results

We used a vector size of 2×2 pixels, 8 bits per pixel, with 2 vectors per PE and a static code-book with 256 entries on an image of 1024×1024 8-bit monochrome pixels. The simulated C•RAM execution time was 26 ms, easily less than the period of 30 frames/s video. This is 1312 times faster than the execution of the sequential algorithm on the workstation which took 34 s.

6.4 Database Applications

Many database applications have a high degree of parallelism. For some, each record can be processed in parallel. Stone reminds us with a database example that speedup should be measured relative to the *best* sequential algorithm [159]. To make this point, Stone demonstrated that a standard indexing technique run on a workstation ran faster than a published massively parallel text search run on a 64K-processor Connection Machine.

6.4.1 Least Squares Match

Introduction

Inexact searches using a key containing multiple fields (criteria) do not lend themselves to the use of indices. While an index can be built for each field of all records, the nearest match for the combination of criteria may not be the best match for any single criteria. A popular search criteria is the "distance" between the multiple fields in the key and record. Hence, we search for the records with the Least Squares (LS) error when compared to the key.

For the sequential and parallel versions of the LS match, we perform a brute-force search of all records in a memory-resident database, and update the records which contain the best match(es) (including ties).

Sequential Algorithm

The sequential algorithm marches through all records looking for the (one or more) records which best match the search key.

```
for all data records
    compute squared error between search key and record
    if error < least_error_observed
        update least_error_observed
for all best matching data records
    update record
```

Parallel Algorithm

The parallel algorithm maps one record to each C•RAM PE. After the PEs compute the squared error between the broadcast key and their record, the broadcast bus is used to find the minimum of the errors and identify the PEs containing the closest matches.

```
for all data records in parallel
    compute squared error between search key and record
    find LS error
    if PE contains a record with LS error
        update record
```

Results

For 128K records of 4 8-bit fields, the match and replace operation takes 200 μs on C•RAM, which is 1250 times faster than the workstation which takes 251 ms.

6.4.2 Data Mining

Introduction

Data mining or *knowledge discovery* is the examination of data for correlations or trends without specifying a particular hypothesis. "Applications of data mining have been reported in astronomy, supermarket sales analysis, health care, stock market, direct marketing, insurance fraud detection, manufacturing, software analysis, and other areas[160, 161]." Our examination of data mining was inspired by a seminar "Mining Patterns from Data: Rough Sets Approach" given by Ziarko [162, 163].

Significance

An enormous amount of data exists ready to be mined, especially with the corporate trend of collecting and storing raw data instead of just totals and statistics. Consequently, data mining is becoming a significant consumer of corporate CPU cycles. The computing demands for sequential algorithms can grow exponentially with the number of condition attributes.

Data mining can be used to detect correlations between attributes (data items) as they appear in records in a database. The data mining problem considered here is limited to finding the *rule* which selects the subset of records with the highest average value of a single *decision attribute*, given a set of boolean-valued *condition attributes*.

One possible use of data mining is illustrated in the following example. If data mining was used to determine a criteria for choosing who should get an expensive disease-preventing drug, then the raw data to be mined might be a set of patient records that include a rating of the severity of the disease (the decision attribute) and a list of possible risk factors (the condition attributes) for each

patient. Without understanding the disease, it would be difficult to intuitively formulate a useful hypothesis about which combination of medical circumstances predispose a patient to be at higher risk of getting the disease. Data mining, however, can test all medical criteria in all combinations and produce a rule which defines a subset of individuals in the general population who are at a greater risk and should receive the drug[4].

Sequential Algorithm

With c conditional attributes, we assign each data record to one or more of the 2^c subsets defined by all possible combinations of the condition attributes. Each of these subsets is defined by a boolean expression or *rule*, corresponding to the boolean product (AND) of all combinations of the condition attributes[5]. A record can belong to more than one subset. In each subset containing a given record, the decision attribute is added to a sum and a count is incremented so that the average of the decision attributes can later be calculated. This average is used to measure the usefulness of each rule. The rule corresponding to the greatest average, (subject to having a sufficient population in the set to be significant), is selected as the best rule.

The algorithm in its simplest form is:

```
for all data records
    for all rules
        if the data record satisfies the rule
            accumulate decision attribute
            increment count
for all rules
    calculate average of decision attributes
    select the rule with the highest average of decision attributes
    discounting rules where count is too low
```

This algorithm can be optimized, as it is not necessary to visit all subsets (rules) to determine if a data record belongs to that set or not. Picture a binary tree with each level representing one of the

[4] As a trivial example of a medical application, consider mining patient records to discover possible risk factors for heart disease. The records show if the patient has high blood pressure, high blood cholesterol, and blue eyes (the condition attributes), and the severity of their heart condition, if any (the decision attribute). Presumably, data mining would discover the rule "high blood pressure AND high blood cholesterol" as being the best predictor of heart disease (depending on the raw data, of course).

For another example, consider a manufacturer who wishes to mail free samples of a product which wasn't previously available in the area to the most promising potential customers. The criteria for selecting the "most promising potential customers" can be mined from consumer data collected in an area where the product is already available. Given a record for each consumer of the volume of purchases of the new product (decision attribute), and purchases of other products, etc. (the condition attributes), we would mine these records to determine a rule which we can use to determine which potential customers should be sent free samples.

[5] Because the rules are the products of combinations of condition attributes, the algorithm doesn't look for anti-correlations.

condition attributes, and the rules being leaves. This tree of 2^c rules can be pruned in a deterministic manner. For each data record, we use recursion to traverse only those branches of the tree for which the current data record satisfies some of the rules. This improved algorithm replaces the two pseudo-code statements: "`for all rules / if the data record satisfies the rule`" with recursion.

Parallel Algorithm

We parallelize the algorithm over the *rules* loop. The SIMD-parallel algorithm reverts back to the unoptimized looping algorithm, resulting in lower efficiency since most of the PEs are not utilized during the insertion of a typical data record.

```
for all data records
    for all rules in parallel
        if the data record satisfies the rule
            accumulate decision attribute
            increment count
for all rules in parallel
    calculate average of decision attributes
    select the rule with the highest average of decision attributes
    discounting rules where count is too low
```

The rules are implicitly mapped across the PEs, where each PE determines which rule numbers it is responsible for by comparing to its own PE number. The data records are broadcast across all PEs. After the records are tabulated, the broadcast bus is used to find the set with the greatest average of the decision attributes, subject to population.

Results

Using 10000 random data records and 2^{17} rules to be tested, the C•RAM version (using 2^{17} PEs) ran in 70.7 ms while the workstation with the tuned algorithm took 192 s, for a speedup of 2724. The C•RAM run time includes the distribution of all data and the retrieval of the result, a total of 34 kB of IO, between host processor and C•RAM. If the raw data and results are to be kept on secondary storage (disk), then the C•RAM version requires a sustained IO bandwidth of 480 kB/s, which is not difficult to provide.

The sequential algorithm was tuned to prune those subtrees to which a record does not apply. With tree-pruning, the sequential algorithm runs over 14 times faster than the sequential brute-force

variant, but has a runtime which is highly data dependent[6].

The C•RAM code we used contains 3 assembler instructions, one of which is in a loop, which makes it run 2.8 times faster than if only standard C expressions are used. For the sequential code, the expression `((rule | attributes) == attributeMask)` compiles to two RISC instructions for the SPARC architecture, so no benefit could be had by using hand-coded assembly language.

6.5 Computer Aided Design Applications

There are many computationally intensive Computer Aided Design (CAD) algorithms which could benefit from massively parallel speedup, particularly in Electronic Design Automation (EDA). Here, we consider one representative algorithm, namely fault simulation.

6.5.1 Fault Simulation with Multiple Faults

Introduction

A fault simulator is an EDA tool which is used to verify the effectiveness of a set of test vectors at finding manufacturing faults in a circuit. Test vectors are sets of stimuli and expected responses for ICs undergoing test on a chip tester during manufacture. Test vectors are designed to detect all, or at least most of the possible faults which are possible under a fault model. The fraction of all possible modeled faults is referred to as the *test coverage*. A fault simulator simulates the test vectors being applied to the circuit for each fault modeled. If the simulation *with* a modeled fault produces a different result to the simulation with no faults, then the fault will be detected if it occurs in a manufactured IC.

The types of faults which can be modeled depend on the type of simulation. A *logic* fault simulator typically models circuit nodes permanently stuck at high or low. A *switch-level* fault simulator can model open or shorted wires as well.

Significance

Most commercial digital IC designs will undergo fault simulation. Because modeling all single faults is already CPU intensive, faults are typically not modeled in combination. Fault simulation

[6]The algorithm with tree-pruning requires $O(2^{number\ of\ true\ attributes})$ time to insert a record. In the pseudo-randomly generated data, true and false valued condition attributes have reasonably equal probability.

with single faults, will not necessarily detect a combination of faults, potentially allowing a defective IC to pass all tests.

With a massively parallel processor, it becomes possible to simulate multiple faults, or even all possible faults for small circuits or sub-circuits. In particular, "scan" design-for-test methodologies make small sub-circuits both controllable and observable. While this level of care may not be justified for testing home-entertainment electronics, it could be justified for human-life critical applications such as anti-lock automobile brakes or expensive un-repairable systems like unmanned spacecraft. To demonstrate fault simulation with multiple faults as a C•RAM application, we run a simple synchronous-logic fault simulation modeling all stuck-at-zero faults in a small circuit. This simple demonstration does not model stuck-at-one faults.

Sequential Algorithm

The fault simulation algorithm we use simulates by brute force, all 2^n n-node circuits which contain stuck-at-zero faults in all possible combinations.

Two optimizations are common. One runs multiple logic simulations in parallel for each bit of the CPU word (in effect mimicking 32 or 64 bit SIMD computation). The other optimization exits the simulation of a particular set of faults early if the test vector has caused the fault to be observed. Applying both optimizations has less benefit, since the simulation could exit early only if all parallel simulations are ready to exit. We implement only the early-exit optimization in the sequential code. The outer loop is not exited early if the test vector has been found to be inadequate since we need to know which faults were not detected. The sequential algorithm is as follows:

```
for all permutations of faults, no-faults first
    for each step in simulation
        apply test vector stimulus
        simulate circuit with faults applied
        compare outputs to no-fault case
            exit loop if outputs differ fault has been observed
    record whether fault was detected
```

For the simulation itself, a compact "executable specification" logic simulator was created. For the sequential case, all methods are compiled "inline" and the compiler generates efficient, straight-line, branch-free, compiled code for the simulation, from application of inputs right through to new output.

Parallel Algorithm

For the C•RAM implementation, the outer loop in the fault simulator is parallelized. Each PE simulates a circuit with a different permutation of faults, as determined by the PE's number. The optimizations are not implemented.

```
for all permutations of faults in parallel
    for each step in simulation
        apply test vector stimulus
        simulate circuit with faults applied
        compare outputs to no-fault case
    record whether fault was detected
```

Results

We fault-simulated all permutations of faults for a 17 gate circuit containing combinational logic and a flip-flop. The circuit size and the choice to simulate only stuck-at-zero faults allows one complete set of faults to be trivially mapped to each PE. The workstation took 2.38 s for thoroughly fault simulating this small circuit, while the simulated C•RAM execution time was 89.4μs, yielding a speedup of 26600.

6.6 Exhaustive Solution-Space Search

One use of massively parallel hardware is finding exact or optimal answers for NP-Complete and NP-Hard problems by exhaustive search of the solution space [164]. C•RAM can accomplish this by allocating different parts of the solution space to different PEs. NP, NP-Complete, and NP-Hard are defined in the glossary on page 127. We shall examine a problem that has a 2^n solution space. Problems with $n!$ solution spaces ($n > 2$) do not map exactly machines where the number of PEs is a power of two.

6.6.1 Satisfiability Problem

Introduction

The satisfiability problem is to determine if there exists a set of values of variables for which a given Boolean expression evaluates to true. This set of values is said to satisfy the expression. For an expression of n variables, there are 2^n sets of values for the variables. The non-deterministic

approach to testing if the expression is satisfiable, is to randomly pick values for the variables and evaluate the expression. If the expression is satisfied, then the problem is solved.

Significance

While this problem is rather academic, it is representative of problems which can be solved with a search of a solution space which grows exponentially with the number of variables. The satisfiability problem was Cook's canonical NP-complete problem [165], to which other NP-complete problems were reduced.

Sequential Algorithm

Our sequential algorithm performs a simple brute-force search of all sets of values to see if the expression can be satisfied.

```
for all sets of values for the variables
        evaluate the given expression
        if the expression evaluates to true, output true and exit
output false
```

Tree-pruning and heuristic algorithms exist which may find solutions faster on average, but these still run in exponential time in the worst case.

Parallel Algorithm

We parallelize the problem over the search space.

```
for all sets of values for the variables in parallel
        evaluate the given expression
        OR these results together
output the ORed result
```

Results

For a boolean expression of 17 variables which we know is not satisfiable, the workstation takes 959 ms while C•RAM requires 23.2 μs for a speed-up of 41000. If the expression was satisfiable, the sequential algorithm could arrive at an answer sooner. We do not use tree-pruning or heuristics for the sequential or parallel program.

The speedup suggests that C•RAM could be useful for other NP-hard problems.

6.7 Memory Operations

6.7.1 Clearing Memory to Zero

Significance

While clearing (or filling) memory is a simple operation, it has many uses, including array initialization, page clearing in a virtual memory system, and clearing a graphics screen or window.

Sequential Algorithm

In this example, an array of 128K 32-bit values is zeroed.

```
for all elements in an array
    write zero to the array element
```

The writes are made to sequential memory addresses in ascending order, offering the workstation memory system some spatial locality to exploit.

Parallel Algorithm

The parallel algorithm writes to the same memory locations, but exploits the wider internal datapath.

```
for all elements in an array in parallel
    write zero to the array element
```

Results

Zeroing the array takes 8.8 ms on the workstation vs. 1.6 μs on C•RAM, giving a speedup of 5493.

6.8 Summary

Problems in this chapter have been parallelized by distributing various elements over the PEs in various ways. The distributed elements included: groups of pixels, searchable data records, rules , faults, solution, and of course the PEs' local memory. Not surprisingly, bit-oriented problems (fault simulation and satisfiability) obtained the greatest speed-up. Problems with workloads consisting

largely of multiplication (e.g. LS matching) had lower speedups. All of these problems make some re-use of data.

Table 6.1 presents the runtimes for the applications described in this chapter. The speedups (ratios of runtimes) range from 1000 to 40,000. The simulated 128 K-PE C•RAM system has only 32 MB of memory which could fit on 8 SIMMs. Despite using a minimalist philosophy in selecting the ALU and communications network, we have demonstrated that C•RAM can achieve significant speedups over a conventional workstation in a wide range of application areas.

Chapter 7

Conclusions

We have developed, and shown the advantages of, a novel computer architecture, C•RAM, which incorporates pitch-matched SIMD PEs into memory at the sense amps and maintains a memory interface. We have shown that the C•RAM architecture is scalable with advances in DRAM density, makes effective use of the internal memory bandwidth, and is useful for a wide range of applications.

By pitch-matching narrow 1-bit PEs to the memory and restricting communications to using 1-dimensional interconnects, C•RAM is able to scale across generations of DRAM. The PEs are pitch-matched to memory columns so that they can be connected to the sense amplifiers. Since the C•RAM PEs can fit in less than the pitch of 8 bitlines, we anticipate that the C•RAM architecture will scale with advances in DRAM technology and shrinking topologies. For scalability, the memory arrays and memory-style packaging limit the internal interprocessor communications to 1-dimensional networks. Of these networks, both a broadcast bus network and a left-right nearest-neighbour network are implemented.

We show that placing narrow pitch-matched SIMD PEs in memory is an effective and inexpensive way to utilize the extremely wide internal datapath available at the DRAM sense amps. After all, the memory bandwidth internal to the memory is 3000 times greater than the bandwidth at the CPU in a typical workstation. C•RAM can utilize 25% to 61% of this internal bandwidth while increasing silicon area and energy usage per operation by less than 25%.

We have demonstrated the feasibility of C•RAM through the design and implementation of major components of a C•RAM system. We have implemented a prototype C•RAM IC, designed a PE for a DRAM process, and simulated the system at the register-transfer and circuit levels. Despite

the minimalist philosophy used in the design of the PE and interconnection networks, a range of applications run thousands of times faster on C•RAM than a workstation.

On the downside, C•RAM is not as flexible as conventional microprocessors and the initial costs for implementation are high. The SIMD programming model is restrictive compared to the MIMD programming model, and C•RAM can only achieve useful speedups if there is enough parallelism in the application to keep PE utilization high. Designing C•RAM in a DRAM process requires a diverse set of design skills and the one-time costs associated with production are high.

We believe that we have addressed some of the failings of other SIMD designs with respect to cost, packaging, and host interface. By looking at the requirements of high-volume manufacture in a DRAM process, we hope to be able to achieve a low marginal cost for the performance compared to other SIMD designs. By integrating the "consumer" of the memory bandwidth, the processors, with the memory, we eliminate the pin-count limitation on the number of PEs which can be incorporated into a chip and reduce the size and cost of packaging. Finally, C•RAM has a memory interface and can be accessed by a host processor or other bus master at speeds similar to standard memory.

In conclusion, C•RAM can achieve a very high performance/price on applications from a variety of fields when compared to workstations.

7.1 Future Work

Our work has demonstrated the feasibility of C•RAM at the circuit, architecture, chip, and application levels. Future work could focus on any of these levels. An important step is to move C•RAM from chips to systems. C•RAM chips should be integrated into a microcomputer, possibly on a circuit board for an expansion bus in a desk-top computer. A SIMD controller for this circuit board must be designed in detail. Further work on an optimizing compiler will benefit the project. For safe multi-tasking use of C•RAM, operating system support must be provided. There is still research to be done on processing element architecture, both for broadly focused SIMD machines like the one presented in this dissertation and for C•RAM designs specific to a single application. Design challenges lie ahead for integrating processors into denser DRAMs and memories with different architectures. The bulk of future work, though, will likely be in designing parallel applications to run on C•RAM.

Appendix A

Carrying On

A.1 Technology Transfer to Industry

Early on in our work, we (Elliott and Snelgrove) patented key aspects of the C•RAM architecture and circuits. This turned out to be useful for winning the interest of industry and, in the end, profitable.

MOSAID Technologies has been a long-time supporter of C•RAM work since my summer employment there. Their support has included the time of their engineers, financial support for university research, trips to customers in Asia to find a "fab" partner, and sponsoring and licencing of the C•RAM patent.

The first commercial project to use the C•RAM patent is the Accelerix Inc. AX256 [39]. This single-chip graphics accelerator and frame buffer is based on technology developed by MOSAID Technologies Inc. and Symbionics Group Ltd. as well as C•RAM.

A.2 Carrying On in Academia

Many people have joined the C•RAM effort since the first chip (described in chapter 5) was built. The faculty members of University of Toronto, Waterloo, Carleton, and Ottawa who have been involved are: Michael Stumm, Martin Snelgrove, Wayne Loucks, Tet Yeap, Martin Lefebvre, Jean-Francois Rivest, and Sethurman Panchanathan.

Their students worked on a number of interesting projects. As these brief descriptions of their work don't do them justice, I encourage the reader to get their papers and theses. Christian Cojocaru added to the C•RAM architecture a ripple carry chain between adjacent PEs for bit-parallel operation, added switches to the broadcast bus controlled by the PEs, implemented and

tested chips, and completed an M.Eng. degree [148]. Dickson Tak Shun Cheung implemented a colliding pucks simulator for C•RAM [166] and added features to the compiler during his B.Sc. Robert McKenzie designed a 1 K PE 16 Mb C•RAM in IBM's 16 Mb trench DRAM process, wrote C•RAM applications and completed an M.Eng. Thinh Le has been working on C•RAM implementations of image compression algorithms, has completed a masters, and is working on a Ph.D. Peter Nyasulu is working towards a Ph.D. on controllers for C•RAM.

Andrew Bishop at University of British Columbia taught a VLSI project course where Anders Svensson, Maneesha Agarwal, Paul Fornari, Minaz Fazal, and Trac-Ky Hoang designed C•RAMs.

Appendix B

Implementation of Common C•RAM Operations

Several examples are provided below to illustrate how the C•RAM PEs can be programmed. The arithmetic examples use *bit-serial* data representations, which implies that all of the bits of an element of a parallel variable are stored in the memory associated with one PE and since the PE's datapath is one bit wide, one bit is accessed at a time. An assembler notation has been created here to describe C•RAM's use. The assembler nomenclature is:

`M:` currently selected memory location

`X, Y:` X and Y data registers

`L, R:` aliases for the X and Y data registers of the adjacent PEs used for shift left or right operations

`vertical` sets shift path multiplexors at the ends of rows of PEs to direct communications to the vertically adjacent row

`W:` write enable register

`=:` left-hand side of '=' is the destination; the right-hand side is the expression performed by the ALU

`&, |, !, ^:` AND, OR, NOT, and XOR operators

`0, 1:` constants

`select`: select the specified local memory address for PE read or write operations. (This corresponds to a memory row address operation[1].)

`for endfor`: loops executed by the SIMD instruction sequencer

All expressions on a line are evaluated in one PE cycle. The architecture supports multiple destinations for a result. For example, the ALU result can write a result both to a register and memory such as in `M = X = X&Y`). Each line of code is indented to one of 3 levels according to whether it is a controller operation (loop), memory access, or PE operation.

The degree of overlap between C•RAM memory and ALU operations depends on the relative cycle times. For SRAM architectures, the memory and ALU operations could be similar, justifying clocking them together. For typical DRAM architectures (which are optimized for density, not speed) several ALU cycles can fit in a DRAM row access cycle. DRAM access by PEs to different memory columns within the same row address (selecting 1 of 4 columns) takes less time than a row access and can typically be performed as part of the ALU operation cycle. The assembler notation, however, simply shows memory accesses ("`select`") as occurring before the ALU operation which requires memory. The row of memory is implicitly restored before the next memory access. Any overlap between the memory and ALU operations not shown in the assembler notation can still be accommodated.

B.1 Memory Copy

The following code copies the contents of one parallel variable (`B`) into another (`A`) of the same size (i.e. `A = B`). This trivial example helps to illustrate the assembler semantics.

```
for j = least significant bit .. most significant bit
        select B[j]
                X = M  /* copy one bit of B to a register */
        select A[j]
                M = X  /* and write to the corresponding bit of A */
endfor
```

The C•RAM controller orchestrates the following events:

[1] When the page mode extension to the architecture is used (as described in section 5.2.6), a portion of the memory address is given in each instruction.

controller	calculate address of `B[0]`
memory	start memory access cycle for address `B[0]`
PE	`X = M`
memory	end memory cycle and precharge for next memory cycle
controller	calculate address of `A[0]`
memory	start memory access cycle for address `A[0]`
PE	`M = X`
memory	end memory cycle and precharge for next memory cycle
controller	calculate address of `B[1]`
	...

B.2 Addition

Addition requires 4 ALU operations in either 2 or 3 memory cycles for 2 or 3 operands respectively. For each operand, a sum and carry are computed. For efficiency, rather than initially clearing the sum and carry (in registers X and Y respectively), the first bit of the bit-serial addition is treated as a special case. The following code is for a two-operand unsigned addition (`A = A + B`), with both operands having the same number of bits.

```
            select B[least_significant_bit]
                    X = M   /* first bit is a special case with no carry in */
            select A[least_significant_bit]
                    Y = M&X /* calculate carry */
                    M = X = X^M /* calculate sum and write back */
    for j = least significant bit + 1 .. most significant bit
            select B[j]
                    X = M^Y /* sum B[j] and carry in */
                    Y = M&Y /* carry out */
            select A[j]
                    Y = Y|(M&X)  /* carry out */
                    M = X = X^M  /* sum A[j] and partial sum, and write back */
    endfor
```

The above example code only handles the case where the operands are of the same size. The compiler can generate additional code to correctly handle two-operand and three-operand addition with sources and destination each of different sizes. Three operand addition naturally requires three memory accesses per bit. When the operands for addition have different lengths, the remaining bits of the other operands are processed correctly and efficiently without redundant memory accesses.

If the operands are two's-complement signed integers, the addends are implicitly sign-bit extended and the sum is explicitly sign-bit padded.

B.3 If-else

The write-enable register is used to selectively inhibit writes by a PE to memory. The data-parallel equivalent of conditional execution is implemented by enabling writes in only those PEs which satisfy the condition in the "`if`". This code fragment is a simplified implementation of `if(flag1) first_code_segment ; else second_code_segment;`.

```
/* select memory location of parallel boolean variable */
select flag1
       W = M       /* enabled if flag1 true */
first_code_segment  /* conditionally executed code */
select flag1
        W = !M      /* enabled if flag1 false */
second_code_segment
        W = 1       /* restore all PEs enabled */
```

The full implementation of high-level-language conditional statements, including support for nesting conditional statements, is discussed in [147].

B.4 Multiplication

For the simple case of multiplying two unsigned parallel variables, the shift and conditional-add algorithm is used. The bits of the first multiplicand are examined from least to most significant, and, conditional on the current bit being a 1, the second multiplicand is added with the correct bit-offset to the partial product.

When the first multiplicand is signed, the most significant bit, bit b, has weight -2^b and a subtraction is performed instead. When the second multiplicand is signed, the partial product must also be signed. Propagating carries over the entire partial product is wasteful. Instead, the partial product can be treated as a shorter integer, initially the length of the second multiplicand plus 1.

Multiplication by a scalar can be further optimized by recoding the scalar operand and propagating the carry when a zero is encountered, rather than performing an unnecessary addition.

B.5 Find Minimum Element

The boolean AND over all elements of a parallel boolean variable is performed in a single C•RAM operation. The following code performs this boolean AND operation on all elements of the parallel boolean variable FLAG and places the result in all elements of the same variable:

```
select FLAG
        /* ALU result is memory ANDed with all other ALU results */
        M = M, bus-tie enabled
```

High level combine-operations, such as *minimum*, can be synthesized by the PEs from boolean combine-operations, provided that the result of the boolean combine-operations can be fed back to the PEs. This feedback could be gathered by the controller and sent out to all PEs; or the PEs could see the result of the combine-operations directly. If the PEs can read the result of a boolean combine-operation directly (rather than the controller receiving and acting on the result) then extra communication delay and possibly a branch delay depending on the implementation of the controller, can be avoided. It is therefore important to allow the PEs to simultaneously transmit and receive on the broadcast bus. A circuit that permits simultaneous transmit and receive is presented in figure 5.8.

The algorithm to find the minimum of all elements of a parallel variable (i.e. across all PEs) consists of a loop over each bit of the parallel variable from the most significant to the least significant bit. On comparing each bit with all others presented on the bus, elements (PEs) are disqualified from the set of those "qualifying" if the bit is not the minimum of those present. Essentially the same algorithm is implemented in combinational logic in the IEEE *Future Bus* prioritized bus arbitration scheme. The pseudo code for this algorithm follows:

```
qualify all PEs for transmitting
loop for all bits of the parallel variable, from most to least significant bit
    transmit the current bit
    on each PE:
        if the result of the AND combine-operation and the
            current bit differ disable the PE for transmitting
end loop
```

The PE (or PEs) which remains qualified for transmitting after the least significant bit has been processed holds the element with the least value. In a tie, multiple PEs remain qualified.

Here is the same "find minimum algorithm" expressed in assembler code. Because the PEs can *read* as well as write to the broadcast bus, a feature of this architecture, the controller does not have to determine and act on the output of the broadcast bus, thus saving a parallel-data-dependent branch

in this loop. The X register records whether a PE is qualified. The PEs discover if their element of the parallel unsigned integer "A" is the minimum value and write the result to the parallel boolean "result".

```
                X = 1 /* qualify all PEs for transmitting */
for j = most_significant_bit .. least_significant_bit
        select A[j]
                /* transmit value from memory, subject to X */
                /* Y = AND_over_all_PEs (M|(!X)) */
                Y = M|(!X), bus-tie enabled
                /* clear X if this PE had a 1 while other PEs write a 0 */
                X = X&((!M)|Y)
end for
        select result
                M = X
```

For signed integers, the number representation is converted to "excess 2^{n-1} notation" by inverting the most significant bit of the two's-complement value before transmitting it on the broadcast bus.

B.6 Two-dimensional communication

Two-dimensional communication in C•RAM is composed of, naturally, horizontal and vertical communication. Since the implementations of horizontal and vertical communication are different (as depicted in figure 4.8), example code is given for each. Following the example code, an efficient implementation of 2-dimensional communication for grid-oriented problems is described.

The following code copies the contents of A (the source) with all elements moved shift_distance PEs right into the destination B: "B.cshiftx(+shift_distance,A)"

```
for j = most_significant_bit .. least_significant_bit
        select A[j]
                /* result bus = memory */
                /* X register in adjacent PE reads result bus */
                /* switches at bank ends set to horizontal mode */
                R = M, horizontal
  for k = 2 .. shift_distance
                R = X, horizontal
  endfor
        select B[j]
                M = X
endfor
```

The compiler handles the additional case where the shift_distance is negative, implying a shift left. For the left shift, the source is the Y register and the destination is "L" or the Y register in the adjacent PE.

The following code copies the contents of A (the source) with all elements moved shift_distance PEs up into the destination B. Because of the physical organization of this communications network, the data to be sent vertically must be sent past an entire row of PEs, making the vertical communication slower.

function B.cshifty(+shift_distance,A)

```
for j = most_significant_bit .. least_significant_bit
        select A[j]
                /* result bus = memory */
                /* X register in adjacent PE reads result bus */
                /* switches at bank ends set to vertical mode */
                R = M, vertical
  for k = 2 .. shift_distance * PE_ROW_LENGTH
                R = X, vertical
  endfor
        select B[j]
                M = X
endfor
```

The compiler handles the additional case where the shift_distance is negative, implying a shift down.

We've shown how data can be shared among a PE's 4 neighbours using the horizontal and vertical communication. Vertical communication is, however, slower than horizontal communication. When more than one grid element is mapped to each PE, the number of vertical communications operations can be minimized, as described in section B.6, by mapping a vertical stripe of elements into each PE's memory. Several applications, such as a 2-dimensional convolution with a 3×3 kernel, in order to calculate the new value for an element (pixel) require the original values from that element and its 8 neighbours (including diagonals). The following *descriptive* pseudo code shows the communication for an "in-place" implementation of such a convolution. Rather than maintain two entire arrays for old and new values, the in-place implementation achieves the same effect by writing results back to the same array with an offset.

```
/* index calculations have been omitted for brevity */
cint stripe[PIXELSperPE+KERNEL] /* stripe of pixels held in each PE plus
room to write back with an offset */
cint buffer[2][3] /* temporarily holds pixels NE,E,SE,SW,W,NW */
/* obtain pixel values from above and below stripe from other PEs */
/* this is the slowest communication so use it the least */
stripe[].cshifty(+1, stripe[])
stripe[].cshifty(-1, stripe[])
/* fill first 2 rows in buffer from adjacent PEs*/
for first 2 rows
        buffer[].cshiftx(+1,stripe[])
```

```
        buffer[].cshiftx(-1,stripe[])
for each pixel in stripe
        /* get a new row in buffer */
        buffer[].cshiftx(+1,stripe[])
        buffer[].cshiftx(-1,stripe[])
        stripe[less offset] = convolution(stripe, buffer)
```

B.7 Summary

We have shown that the C•RAM architecture is capable of efficiently supporting arithmetic operations, conditional operations, and interprocessor communication including combine operations (such as find minimum), as well as one-dimensional and two-dimensional grid communication.

Appendix C

Optimal Data Placement for Two-Dimensional Grid Problems

In this appendix, the optimal data placement for a 2-D problem requiring communication with 8 neighbours is derived. Specifically, the dimensions of the rectangle of "pixels" to be mapped onto each PE is determined.

C.1 Left-Right Communication

When mapping c data elements from a 2-D problem onto a C•RAM with 1-D (left-right) communication, we should first choose the size of the rectangle $(n_x \times n_y)$ which will provide the minimum communication overhead. The communication delay for single-bit communications with eight neighbours on a 2-D grid is:

$$\begin{aligned} T_8 &= 2 \text{ time of}(\text{X communications} + \text{corners} + \text{Y communications}) \\ &= 2(n_y \frac{N_y}{n_y} T_x + 2T_x + n_x T_x) \\ &= 2T_x(N_y + 2 + n_x) \end{aligned}$$

where:

$N_x \equiv$ the number of columns in the data array

$N_y \equiv$ the number of rows in the data array

$n_x \equiv$ the number of columns of data stored in the local memory of one PE

$n_y \equiv$ the number of rows of data stored in the local memory of one PE

$c \equiv$ the number of data elements per PE

$P \equiv$ the number of PEs in the system

$N_x, N_y, n_x, n_y, c, P \in N$

$T_x \equiv$ the time taken by an x-direction shift operation

$T_x \in R$

$n_x n_y = c$

$cP = N_x N_y$

By inspection, for constant N_x, N_y, c, the communication time is minimum for $n_x = 1$

C.2 Scalable Left-Right Interconnect

The communication delay for single-bit communications with eight neighbours on a logical 2-D grid using scalable left-right communication is:

$$\begin{aligned} T_8 &= 2(\text{X communications} + \text{corners} + \text{Y communications}) \\ &= 2(n_y T_x + 2T_x + n_x(L + L_0)T_y) \end{aligned}$$

where:

$L \equiv$ the number of PEs arranged in a row as one long shift register

$L_0 \equiv$ the number of extra pipeline stages in a vertical shift

$N_x, N_y, n_x, n_y, c, P, L, L_0 \in N$

$T_y \equiv$ the time taken by an y-direction shift operation

$T_x, T_y \in R$

and other variables are as defined in the previous section.

To calculate the dimensions of the block which will give the minimum communication delay, assume that $T_x = T_y$, substitute $n_y = \frac{c}{n_x}$ and then find the minimum of T_8 with respect to n_x. The application of differential calculus to find the minimum of this discrete valued problem is justified since the continuous function $T_8(n_x)$ has only one extremum which is a minimum. The legal values of n_x adjacent to the minimum of the continuous function must then be examined to find the

minimum of the discrete function [167].

$$\begin{aligned} T_8 &= 2T_x(\frac{c}{n_x} + 2 + n_x(L + L_0)) \\ \frac{dT_8}{dn_x} &= -\frac{2cT_x}{{n_x}^2} + 2T_x(L + L_0) = 0 \\ n_x &= \sqrt{\frac{c}{L + L_0}} \end{aligned}$$

Given $L \geq c$ and $n_x \in N$, we conclude that communication delay is minimal for the data organization where $n_x = 1$ for constant N_x, N_y, c. Again the optimal block width is one.

It is interesting to note that in the degenerate case where $L + L_0 = 1$, the scalable left right interconnect becomes a grid and the proportions of the optimally shaped block is are square, as one would expect.

Appendix D

Glossary

ADC analog to digital converter

addend a quantity to be added

AIS a manufacturer of SIMD processors for image processing

ALU Arithmetic Logic Unit, a portion of a processor

AMT Active Memory Technologies, makers of the DAP

ASIC Application Specific Integrated Circuit; also refers to general purpose IC technology designed for lower quantity production of custom chips (as opposed to commodity large volume production)

autonomous memory addressing the ability of SIMD PEs to independently address local memory (opposite: uniform addressing)

autonomous network routing a communications network property whereby each node can specify the destination of a message (opposite: uniform network routing)

AWACS Airborne Warning And Control System (airborne military RADAR)

bandwidth when applied to a digital communications channel, the channel capacity in bits per second

BATMOS BiCMOS Analog-Telecom MOS, Norther-Telecom's 0.8μm BiCMOS IC process

BiCMOS Bipolar and CMOS, generic term for IC processes combining vertically integrated bipolar transistors and CMOS (see CMOS).

bit-serial an architecture or algorithm that performs arithmetic one bit at a time

bitline the signal path used for transferring data in and out of a memory array Bitlines run in the memory column direction.

bitline pair see "folded bitline pair"

BLITZEN a follow-on to the MPP, funded by NASA

byte a sequence of adjacent binary digits operated on as a unit by a computer[168] An 8 bit byte (octet) is assumed for all calculations in this dissertation.

B 8 bit Byte

b bit, binary digit

CAD Computer Aided (also Assisted or Augmented) Design

CAS Column Address Strobe; a DRAM timing signal

CCITT Comité Consultatif International Télégraphique et Téléphonique (now the International Telecommunication Union (ITU))

CISC Complex Instruction Set Computer

CMC Canadian Microelectronics Corporation; a Federal Government sponsored organization that co-ordinates IC fabrication and design tools for its members.

CMOS Complementary MOS, a generic term for IC processes combining n-channel and p-channel MOS transistors (see MOS)

CM Connection Machine, product of Thinking Machines Inc.

column decoder the circuitry in a memory that selects one column or group of columns to be connected to the read and write amplifiers for access

combine network (parallel processing jargon) a network, commonly a tree, used to perform combine operations

combine operations (parallel processing jargon) operations that combine a vector of like objects into a scalar (e.g. AND, minimum, sum)

commodity memory common variety memory with (typically) JEDEC standard pinout and functionality Other typical characteristics include high volume, multiple suppliers, and competitive pricing.

CPU Central Processing Unit (where most of the computing *used* to be performed)

C•RAM Computational RAM, the name of our SIMD PEs-in-memory architecture

crowbar current the (wasted) current conducted from V_{DD} through transistors to V_{SS} when a CMOS logic gate is in transition

DAP Distributed Array Processor, the name of SIMD computers manufactured by ICL and AMT

datapath width we refer to the width of the ALU, registers, etc. (the datapath)

DCT Discrete Cosine Transform; a particular transform between time and frequency domains

DC Direct Current

DEC Digital Equipment Corporation

DFT Discrete Fourier Transform

die (*pl.* dies) an integrated circuit as cut from a wafer (not including packaging)

differential signaling a signaling convention where two wires are used to convey one signal The signal is the difference between the two levels

DIP Dual Inline (IC) Package

DMA Direct Memory Access; an IO technique where the IO device becomes the bus master and communicates directly with (main) memory

DNA deoxyribonucleic acid

DRAM Dynamic Random Access (read-write) Memory

DRC Design Rule Check(er); a CAD tool and the act of using it

DSP Digital Signal Processing

FCC Federal Communications Commission, US regulatory body

FCT Fast Cosine Transform; an $O(nlogn)$ implementation of a DCT (has the advantage over the FFT that all values are real — not complex)

FFT Fast Fourier Transform; an $O(nlogn)$ implementation of a DFT

FIR filter Finite Impulse Response filter

folded bitline pair a pair of wires (bitlines) connected to the differential terminals of a sense amplifier in DRAM

FPGA Field Programmable Gate Array

full adder a full (binary) adder takes three interchangeable inputs (two addends and a carry in) and produces the sum and a carry out

GAPP Geometric Array Parallel Processor, product of NCR

GIPS Giga Instructions Per Second

half adder a half (binary) adder takes two addends as inputs and produces the sum and carry

HDTV High Definition Television

HP Hewlett-Packard Ltd.

HSPICE a variant on SPICE produced by Meta Software

ICL a U. K. computer manufacturer

IC Integrated Circuit

IEEE Institute of Electrical and Electronics Engineers

IMAP Integrated Memory Array Processor, an NEC SIMD chip

IMS Integrated Measurement Systems Ltd.

IO Input-Output

IRAM Intelligent RAM

ISSCC International Solid State Circuits Conference

JEDEC formerly the Joint Electronic Devices Engineering Council, now the Solid State Products Engineering Council; a standards organization

JFET Junction Field Effect Transistor

JPEG Joint Photographic Experts Group; a standards committee concerned with DCT-based compression of still images.

JSSC IEEE Journal of Solid State Circuits

Kb 1024 bits

KB 1024 bytes

K a common prefix in computer literature meaning 1024 (which clashes with the S.I. unit for temperature, Kelvin)

k S.I. prefix meaning 1000

local memory address the memory address in a distributed-memory SIMD PE's local memory

LSB Least Significant Bit (or Byte)

M a prefix meaning 1048576 (K^2) for memory size or 10^6 (the SI definition) for frequencies and data rates (name clash found in the literature)

macro-cell A cell design-library contains layouts for circuit components, "cells", which can be placed in an IC design. A macro or macro-cell is a cell of significant size or complexity, such as the DRAM memory array.

massively parallel a subjective term meaning "lots" of processors, where the definition of "lots" depends on the technology available at the time

Mb 1048576 bits

MB 1048576 bytes (octets)

memory array as used in this document, an array of memory cells bounded by the extent of continuous bitlines and wordlines

method a C++ member function where the object is an implicit formal parameter

MFLOPS Millions of Floating Point Operations Per Second

Micronet a Network of Centres of Excellence in Microelectronics, funded by the Government of Canada

MIMD Multiple Instruction stream, Multiple Data stream (parallel processor)

minuend a quantity from which another is to be subtracted

MIPS Millions of Instructions Per Second

MMU Memory Management Unit (used to implement memory protection or virtual memory)

MOSAID Technologies a company in Ottawa, Canada whose products include memory designs and memory testers

MOSFET Metal Oxide Silicon Field Effect Transistor; an anachronistic name for insulated-gate field effect transistors that are most commonly built with *polysilicon* gates, oxide dielectric, and silicon source, channel, and drain.

MOS contraction of MOSFET

MPEG I, MPEG II Moving Picture Experts Group; standards for DCT-based compression of motion pictures such as video

MPP Massively Parallel Processor; both generic, and the name of a processor commissioned by NASA

MSI Medium Scale Integration

multiplicand a quantity that is to be multiplied by another

MUX abbreviation for multiplexor

N, NE, E, SE, S, SW, W, NW compass directions

NAND negated output AND boolean logic gate

NASA National Aeronautics and Space Administration (USA)

NCR an electronics manufacturer, formerly National Cash Register

NEC Nippon Electric Corporation, an electronics manufacturer

NMOS n-channel MOSFET (transistor or technology)

NOR negated output OR boolean logic gate

NP-Complete a set or property of computational decision problems which is a subset of NP (i.e. can be solved by a nondeterministic Turing Machine in polynomial time), with the additional property that it is also NP-hard. Thus a solution for one NP-complete problem would solve all problems in NP. Many (but not all) naturally arising problems in class NP are in fact NP-complete. There is always a polynomial-time algorithm for transforming an instance of any NP-complete problem into an instance of any other NP-complete problem. So if you could solve one you could solve any other by transforming it to the solved one. The first problem ever shown to be NP-complete was the satisfiability problem. Another example is Hamilton's problem. [169]

NP-Hard a set or property of computational search problems. A problem is NP-hard if solving it in polynomial time would make it possible to solve all problems in class NP in polynomial time. Some NP-hard problems are also in NP (these are called "NP-complete"), some are not. If you could reduce an NP problem to an NP-hard problem and then solve it in polynomial time, you could solve all NP problems. [169]

NSERC Natural Sciences and Engineering Research Council (of Canada)

$O()$ the upper bound on the growth of a function

$\Omega()$ the lower bound on the growth of a function

$\overline{OPS}$ OPerate Strobe, a C•RAM signal that causes an operate (compute) cycle to occur

OS Operating System

parallel variable a variable that has an element associated with each processor (applicable to SIMD or MIMD with local memories) (e.g. Thinking Machines' *parallel variable* or MasPar's *plural variable*)

parallelize a verb formed from the word parallel by computer scientists. In the context of this thesis, it is the act of converting a sequential algorithm to a parallel algorithm suitable for a parallel processor.

PARIS The Connection Machine 1&2 programming model that implements 1,8,16,32, and 64 bit operations (in microcode on the sequencer).

PDE Partial Differential Equation

PE Processing Element

PGA Pin Grid Array (an IC package)

pitch the distance between objects as measured at their centres (e.g. the pitch of uniformly spaced wires on a surface is equal to the wire width plus the spacing).

polysilicon polycrystalline silicon, a conductive layer in IC designs

PSRAM Pseudo Static Random Access Memory; a memory with DRAM internals and an SRAM interface.

RADAR RAdio Detecting And Ranging

Rambus a proprietary standard for memory interface, signaling, and protocol

RAM Random Access Memory (usually meaning read-write RAM, as used here)

RAS Row Address Strobe; a DRAM timing signal

RDRAM Rambus DRAM

refresh the process of restoring the charge on DRAM storage capacitors

RISC Reduced Instruction Set Computer

row decode the circuitry in a memory that selects one wordline, and hence, all the memory cells on that row

RTL Register Transfer Level (simulation); synchronous simulation that concerns itself with the data to be stored in registers rather that the implementation or the timing of the combinational logic which produces that data

SAM Sequential Access Memory (used for high speed sequential access of Video RAMs)

SA Sense Amplifier

SDRAM Synchronous-mode DRAM; a proposed JEDEC memory interface and operation standard

sense amp sense amplifier; used to amplify and restore signal levels to memory cells during DRAM refresh

SIMD Single Instruction stream Multiple Data stream (parallel processor)

SIMM Single Inline Memory Module (a removable card holding typically 8 or 9 memory chips, but as many as 24 chips

SOJ Small Outline J-lead IC package

SPARC the SUN Microsystems RISC architecture

SPICE Simulation Program with Integrated Circuit Emphasis, a circuit simulator written at U.C. Berkeley

SPMD Single Program Multiple Data stream; a pseudo-SIMD programming model for MIMD hardware (*also* a euphemism for MIMD, used by Thinking Machines Inc., for customers already sold on SIMD)

SRAM Static RAM

STARAN a MPP commissioned by NASA

subtrahend a quantity to be subtracted from another

SUNOS a SUN operating system

SUN a contraction of SUN Microsystems Inc.

SVP Serial Video Processor

TI Texas Instruments Inc, an electronics manufacturer

TTL Transistor-Transistor Logic; also a standard for logic levels

uniform memory addressing a SIMD property whereby all PEs must simultaneously access the same local memory location specified by the controller (opposite: autonomous addressing)

uniform network routing a communications network property whereby the entire network delivers messages in one of many predetermined patterns (for instance: shift left, shift north-east, FFT butterfly pattern) (opposite: autonomous network routing).

VASTOR Vector Associative Store - TORonto

VAX a classical DEC CISC architecture

VIP Video Image Processor

VLIW Very Long Instruction Word [processor]. The industry had defined "Very" as 2 or more operations.

VLSI Very Large Scale Integration (chips)

VM Virtual Memory

VRAM Video RAM

V_{DD} the positive power supply on NMOS (and anachronistically, CMOS) circuits, usually +5V, +3.3V or +2V relative to V_{SS}

V_{PP} a boosted internal power supply in some DRAMs, typically about 2V above V_{DD}

V_{SS} the negative power supply on NMOS (and anachronistically CMOS) circuits

wordline strappings the regions where connections are made between wordline signals routed in different layers, one medium resistivity gate polysilicon and one low resistivity metal

wordline a signal path in a memory array that selects a row of memory for access

XOR boolean exclusive or

yield in the context of semiconductor manufacturing, the ratio of functional parts to total parts manufactured

Bibliography

[1] Wm. A. Wulf and Sally A. McKee. Hitting the Memory Wall: Implications of the Obvious. *Computer Architecture News*, 23(1):20–24, March 1995.

[2] Preston Briggs. electronic communication. 1995.

[3] Brian Fuller and David Lammers. NEC Plans Process Shift. *Electronic Engineering Times*, 1(856):1–8, July 1995.

[4] M. J. Flynn. Very High-Speed Computing Systems. *Proceedings of the IEEE*, 54(12):1901–1909, December 1966.

[5] S. Y. Kung. *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[6] Harold S. Stone. A Logic-in-Memory Computer. *IEEE Transactions on Computers*, C-19(1):73–78, January 1970.

[7] D. G. Elliott and W. M. Snelgrove. C•RAM: Memory with a Fast SIMD Processor. In *Proceedings of the Canadian Conference on VLSI*, pages 3.3.1–3.3.6, Ottawa, October 1990.

[8] Duncan G. Elliott, W. Martin Snelgrove, and Michael Stumm. Computational RAM: A Memory-SIMD Hybrid and its Application to DSP. In *Custom Integrated Circuits Conference*, pages 30.6.1–30.6.4, Boston, MA, May 1992.

[9] Daniel P. Siewiorek, C. Gordon Bell, and Allen Newell. *Computer Structures: Principles and Examples*. McGraw-Hill, New York, 1982.

[10] Kenneth E. Batcher. The Flip Network in STARAN. In *Proceedings of the International Conference on Parallel Processing*, pages 65–71, Detroit, Michigan, August 1976.

[11] R. G. Lange. High Level Lnaguage for Associative and Parallel Computation with STARAN. In *Proceedings of the International Conference on Parallel Processing*, pages 170–176, Detroit, Michigan, August 1976.

[12] Robert Katz. Analysis of the AWACS Passive Tracking Algorithms on the RADCAP STARAN. In *Proceedings of the International Conference on Parallel Processing*, pages 177–186, Detroit, Michigan, August 1976.

[13] Edward C. Stanke. Automatic Track Initialization Using the RADCAP STARAN. In *Proceedings of the International Conference on Parallel Processing*, pages 187–190, Detroit, Michigan, August 1976.

[14] Kenneth E. Batcher. The Multidimensional Access Memory in STARAN. *IEEE Transactions on Computers*, 26(2):174–177, February 1977.

[15] Jack A. Rudolph and Kenneth E. Batcher. A Productive Implementation of an Associative Array Processor: STARAN. In Daniel P. Siewiorek, C. Gordon Bell, and Allen Newell, editors, *Computer Structures: Principles and Examples*, pages 317–331. McGraw-Hill, New York, 1982.

[16] Terry Fountain. *Processor Arrays: Architecture and Applications*. Academic Press, 1987.

[17] D. Parkinson, D. J. Hunt, and K. S. MacQueen. The AMT DAP 500. In *COMPCON Spring 88*, pages 196–199, San Francisco, CA, March 1988. Active Memory Technology.

[18] Peter M. Flanders, Richard L. Hellier, Huw D. Jenkins, Cliff J. Pavelin, and Sven van den Berghe. Efficient High-Level Programming on the AMT DAP. *Proceedings of the IEEE*, 79(4):524–539, April 1991.

[19] W. M. Loucks, W. M. Snelgrove, and S. Zaky. VASTOR 1978. A report for the special topics course ELE 1718, June 1978.

[20] Wayne M. Loucks, Martin Snelgrove, and Safwat G. Zaky. A Vector Processor Based on One-Bit Microprocessors. *IEEE Micro*, 2(1):53–62, February 1982.

[21] W. M. Snelgrove, W. M. Loucks, and S. Zaky. Intelligent RAM. 1981.

[22] Wai Hung Lo. VASTOR Controller and its Programming Environment. Master's thesis, University of Toronto, July 1984.

[23] T. H. Yeap, W. H. Lo, M. Snelgrove, W. M. Loucks, and S. G. Zaky. A VLSI Implementation of a 1-Bit Processing Element for the VASTOR Array Processor. In *CCVLSI*, pages 18–21, 1983.

[24] Tet Hin Yeap. Design of VASTOR Processing Element Suitable for VLSI Implementation. Master's thesis, University of Toronto, Dept. of Electrical Engineering, 1984.

[25] Kenneth E. Iverson. *A Programming Language*. J. Wiley, New York, 1962.

[26] Kenneth E. Batcher. Design of a Massivley Parallel Processor. *IEEE Transactions on Computers*, 29(9):836–840, September 1980.

[27] Robert J. Baron and Lee Higbie. *Computer Architecture Case Studies*. Addison Wesley, Reading, Massachusetts, 1992.

[28] Eugene L. Cloud. The Geometric Arithmetic Parallel Processor. In *The 2nd Symposium on the Frontiers of Massively Parallel Computation*, pages 373–381, Fairfax Virginia, October 1988. Martin Marietta Electronic Systems.

[29] NCR Corporation. Geometric Arithmetic Parallel Processor. Technical Report NCR45CG72, NCR Corporation, Dayton, Ohio, 1984.

[30] W. Daniel Hillis. *The Connection Machine*. The MIT Press, 1985.

[31] Henry Fuchs, Jack Goldfeather, Jeff P. Hultquist, Susan Spach, John D. Austin, Jr. Frederick P. Brooks, John G. Eyles, and John Poulton. Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancemts in Pixel-Planes. In *SIGGRAPH'85 Conference Proceeding*, pages 111–120. San Francisco, July 1985.

[32] John Poulton, Henry Fuchs, John Austin, John Eyles, and Trey Greer. Building a 512×512 Pixel-Planes System. *Advanced Research in VLSI - Proceedings of the 1987 Stanford Conference*, pages 57–71, 1987.

[33] Henry Fuchs, John Poulton, John Eyles, Trey Greer, Jack Goldfeather, David Ellsworth, Steve Molnar, Greg Turk, Brice Trebbs, and Laura Israel. Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories. In *SIGGRAPH'89 Conference Proceeding*, pages 79–88, Boston, July 1989.

[34] Jack Goldfeather, Steven Molnar, Greg Turk, and Henry Fuchs. Solid Modeling: Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning. *IEEE Computer Graphics and Applications*, 9(3):20–28, May 1989.

[35] Steve Molnar, John Eyles, and John Poulton. PixelFlow: High-Speed Rendering Using Image Composition. In *SIGGRAPH'92 Conference Proceeding*, pages 231–240, Chicago, July 1992.

[36] Maya Gokhale, Bill Holmes, and Ken Iobst. Processing in Memory: the Terasys Massively Parallel PIM Array. *Computer*, 28(3):23–31, April 1995.

[37] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM. *IEEE Micro*, 17(2):34–44, March 1997.

[38] Jeffrey C. Gealow and Charles G. Sodini. A Pixel-Parallel Image Processor Using Logic Pitch-Matched to Dynamic Memory. In *Symposium on VLSI Circuits*, pages 57–58, Hawaii, June 1997. IEEE.

[39] R. Torrance, I. Mes, B. Hold, D. Jones, J. Crepeau, P DeMone, D. MacDonald, C. O'Connell, P. Gillingham, R. White, S. Duggins, and D. Fielder. A 33GB/s 13.4Mb Integrated Graphics Accelerator and Frame Buffer. In *International Solid-State Circuits Conference*, pages 340–341, San Francisco, February 1998.

[40] D. W. Blevins, E. W. Davis, R. A. Heaton, and J. H. Reif BLITZEN: a Highly Integrated Massively Parallel Machine. In *Proceedings of the 2nd Symposium on the Frontiers of Massively Parallel Computation*, pages 399–406, Fairfax VA, October 1988.

[41] R. A. Heaton and D. W. Blevins. BLITZEN: a VLSI Array Processing Chip. In *Custom Integrated Circuits Conference*, pages 12.1.1–12.1.5, San Diego, CA, May 1989.

[42] J. M. Jennings, E. W. Davis, and R. A. Heaton. Comparative Performance Evaluation of a New SIMD Machine. In *Proceedings of the 3nd Symposium on the Frontiers of Massively Parallel Computation*, pages 255–258, College Park MD, October 1990.

[43] Tom Blank. The MasPar MP-1 Architecture. In *COMPCON Spring 90*, pages 20–24, San Francisco, February 1990. MasPar.

[44] John R. Nickolis. The Design of the MasPar MP-1: A Cost Effective Massively Parallel Computer. In *COMPCON Spring 90*, pages 25–28, San Francisco, February 1990. MasPar.

[45] Peter Christy. Software to Support Massively Parallel Computing on the MasPar MP-1. In *COMPCON Spring 90*, pages 29–33, San Francisco, February 1990. MasPar.

[46] MasPar. MP-1 Family Data-Parallel Computers. Technical Report PL006.0190, MasPar Computer Corporation, Sunnyvale, CA, 1990.

[47] Won Kim and Russ Tuck. MasPar MP-2 PE Chip: A Totally Cool Hot Chip. In *Hot Chips V*, pages 3.3.1–3.3.13, Stanford, August 1993.

[48] K. Chen and C. Svenson. A 512-Processor Array Chip for Video/Image Processing. *Proc. of From Pixels to Features II*, pages 349–361, August 1990.

[49] Per-Erik Danielsson, Pär Emanuelsson, Keping Chen, and Per Ingelhag. Single-Chip High-Speed Computation of Optical Flow. In *IAPR International Workshop on Machine Vision Applications*, pages 331–335, November 1990.

[50] Per-Erik Danielsson. Smart Algorithms for Bit-Serial Arrays. In *Sixth International Conference on Image Analysis and Processing*, pages 1–21. Linköping University.

[51] Jim Childers, Peter Reinecke, and Hiroshi Miyaguchi. SVP: A Serial Video Processor. *IEEE 1990 Custom Integrated Circuits Conference*, pages 17.3.1–17.3.4, May 1990.

[52] Hiroshi Miyaguchi, Hujime Krasawa, and Xhinichi Watanabe. Digital TV with Serial Video Processor. *IEEE Transactions on Consumer Electronics*, 36(3):318–326, August 1990.

[53] Nobuyuki Yamashita, Tohru Kimura, Yoshihiro Fujita, Yoshiharu Aimoto, Takashi Manaba, Shin'ichiro Okazaki, Kazuyuki Nakamura, and Masakazu Yamashina. A 3.84GIPS Integrated Memory Array Processor LSI with 64 Processing Elements and 2Mb SRAM. In *International Solid-State Circuits Conference*, pages 260–261, San Francisco, February 1994. NEC.

[54] Nobuyuki Yamashita, Tohru Kimura, Yoshihiro Fujita, Yoshiharu Aimoto, Takashi Manaba, Shin'ichiro Okazaki, Kazuyuki Nakamura, and Masakazu Yamashina. A 3.84GIPS Integrated Memory Array Processor with 64 Processing Elements and 2Mb SRAM. *IEEE Journal of Solid-State Circuits*, 29(11):1336–1343, November 1994.

[55] Peter M. Kogge. EXECUBE - A New Architecture for Scalable MPPs. In *1994 International Conference on Parallel Processing*, pages I77–I84, August 1994.

[56] Peter M. Kogge, Toshio Sunaga, Hisatada Miyataka, Koji Kitamura, and Eric Retter. Combined DRAM and Logic for Massively Parallel Systems. In *Conference on Advanced Research in VLSI*, pages 4–16, Chapel Hill, NC, March 1995.

[57] Maya Gokhale, Bill Holmes, Ken Iobst, Alan Murray, and Tom Turnbull. A Massively Parallel Processor-in-Memory Array and its Programming Environment. Technical Report SRC-TR-92-076, Supercomputer Research Center - Institute for Defense Analyses, 17100 Science Drive, Bowie, Maryland, November 1992.

[58] Loring Wirbel. NSA taps Cray Computer, National. *Electronic Engineering Times*, 1(816):39–40, September 26 1994.

[59] Masuyoshi Kurokawa, Akihiko Hashiguchi, Ken'ishiro Nakamura, Hiroshi Okuda, et al. 5.4GOPS Linear Array Architecture DSP for Video-Format Conversion. In *International Solid-State Circuits Conference*, pages 254–255, San Francisco, February 1996. IEEE.

[60] Texas Instruments. *MOS Memory Data Book*, 1989.

[61] M. Inoue, T. Yamada, H. Kotani, H. Yamauchi, et al. A 16-Mbit DRAM with a Relaxed Sense-Amplifier-Pitch Open-Bit-Line Architecture. *IEEE Journal of Solid-State Circuits*, 23(5):1104–1112, October 1988.

[62] M. Aoki, Y. Nakagome, M. Horiguchi, H. Tanaka, et al. A 60-ns 16-Mbit CMOS DRAM with a Transposed Data-Line Structure. *IEEE Journal of Solid-State Circuits*, 23(5):1113–1127, October 1988.

[63] M. Aoki, Y. Nakagome, M. Horiguchi, S. Ikenaga, and K. Shimohigashi. A 16-Level/Cell Dynamic Memory. *IEEE Journal of Solid-State Circuits*, 22(2):297–299, April 1987.

[64] M. Horiguchi, M. Aoki, Y. Nakagome, S. Ikenaga, and K. Shimohigashi. An Experimental Large-Capacity Semiconductor File Memory Using 16-levels/Cell Storage. *IEEE Journal of Solid-State Circuits*, 23(1):27–33, February 1988.

[65] T. Yamada, H. Kotani, J. Matsushima, M. Inoue, et al. A 4-Mbit DRAM with Concurrent ECC. *IEEE Journal of Solid-State Circuits*, 23(1):20–26, February 1988.

[66] Jean-Daniel Nicoud. Video RAMs. *IEEE Mirco*, pages 8–27, February 1988.

[67] Nicky C. C. Lu. Advanced Structures for Dynamic RAMs. *IEEE Circuits and Devices Magazine*, 5(1):27–36, January 1989.

[68] Richard F. Lyon and Richard R. Schediwy. CMOS Static Memory with a New Four-Transistor Memory Cell. *Advanced Research in VLSI - Proceedings of the 1987 Stanford Conference*, pages 111–132, 1987.

[69] S. Fujii, M. Ogihara, M. Shimizu, et al. A 45-ns 16-Mbit DRAM with Triple-Well Structure. *IEEE Journal of Solid-State Circuits*, 24(5):1170–1175, October 1989.

[70] S. Chou, T. Takano, A. Kita, F. Ichikawa, M Uesugi, et al. A 60-ns 16-Mbit DRAM with a Minimized Sensing Delay Caused by Bit-Line Stray Capacitance. *IEEE Journal of Solid-State Circuits*, 24(5):1176–1183, October 1989.

[71] K. Arimoto, K. fujishima, Y. Matsuda, et al. A 60-ns 3.3-V-Only 16-Mbit DRAM with Multipurpose Register. *IEEE Journal of Solid-State Circuits*, 24(5):1184–1190, October 1989.

[72] C. Kim, Y. Choi, D.-S. Min, H. S. Hwang, et al. An Experimental 16-Mbit DRAM with Reduced Peak-Current Noise. *IEEE Journal of Solid-State Circuits*, 24(5):1191–1197, October 1989.

[73] G. B. Bronner, K. Kitamura, R. E. Scheuerlein, et al. A 22-ns 1-Mbit CMOS High-Speed DRAM with Address Multiplexing. *IEEE Journal of Solid-State Circuits*, 24(5):1198–1205, October 1989.

[74] M. Matsui, H. Momose, Y. Urakawa, T. Maeda, et al. An 8-ns 1-Mbit ECL BiCMOS SRAM with Double-Latch ECL-to-CMOS-Level Converters. *IEEE Journal of Solid-State Circuits*, 24(5):1226–1233, October 1989.

[75] K. Mashiko, M. Nagatomo, K. Armoto, et al. A 4-Mbit DRAM with Folded-Bit-Line Adaptive Sidewall-Isolated Capacitor (FASIC) Cell. *IEEE Journal of Solid-State Circuits*, 22(5):643, October 1987.

[76] K. Kimura, K. Shimohigashi, J. Etoh, et al. A 65-ns 4-Mbit CMOS DRAM with a Twisted Driveline Sense Amplifier. *IEEE Journal of Solid-State Circuits*, 22(5):651, October 1987.

[77] G. Kitsukawa, Ryoichi Hori, Y Kawajiri, et al. An Experimental 1-Mbit BiCMOS DRAM. *IEEE Journal of Solid-State Circuits*, 22(5):657, October 1987.

[78] T. Ohsawa, T. Furuyama, Y. Watanabe, et al. A 60-ns 4-Mbit CMOS DRAM with Built-In Self-Test Function. *IEEE Journal of Solid-State Circuits*, 22(5):663, October 1987.

[79] H. Miyamoto, T. Yamagata, S. Mori, et al. A Fast 256K×4 CMOS DRAM with a Distributed Sense and Unique Restore Circuit. *IEEE Journal of Solid-State Circuits*, 22(5):861, October 1987.

[80] M. Horiguchi, M. Aoki, H. Tanaka, J. Etoh, Y. Nakagome, et al. Dual-Operating-Voltage Scheme for a Single 5-V 16-Mbit DRAM. *IEEE Journal of Solid-State Circuits*, 23(5):1128–1132, October 1988.

[81] R. Pinkham, D. Russell, and A. Balistreri. A 128K×8 70-MHz Multiport Video RAM with Auto Resister Reload and 8×4 Block Write Feature. *IEEE Journal of Solid-State Circuits*, 23(5):1133–1139, October 1988.

[82] N. Lu, H. Chao, and W. Hwang. A 20-ns 128-kbit×4 High-Speed DRAM with 330-Mbit/s Data Rate. *IEEE Journal of Solid-State Circuits*, 23(5):1140–1149, October 1988.

[83] K. Sasaki, K. Ishibashi, T. Yamanaka, et al. A 9-ns 1-Mbit CMOS SRAM. *IEEE Journal of Solid-State Circuits*, 24(5):1219–1225, October 1989.

[84] Y. Nakagome, M. Aoki, and S. Ikenaga. The Impact of Data-Line Interference Noise on DRAM Scaling. *IEEE Journal of Solid-State Circuits*, 23(5):1120–1127, October 1988.

[85] Laureen H. Parker and Al F. Tasch. Ferroelectric Materials for 64 Mb and 256Mb DRAMs. *IEEE Circuits and Devices Magazine*, 6(1):17–26, January 1990.

[86] Tetsuya Iizuka. Session 6 Overview: High-Density DRAM. In *International Solid-State Circuits Conference*, page 103, San Francisco, February 1991. IEEE.

[87] Shigeru Kikuda, Hiroshi Miyamoto, Shigeru Mori, Mitsutaka Niiro, and Michihiro Yamada. Optimized Redundancy Selection Based on Failure-Related Yield Model for 64Mb DRAM and Beyond. In *International Solid-State Circuits Conference*, pages 104–105, San Francisco, February 1991. IEEE.

[88] Katsutaka Kimura, Takeshi Sakata, Kiyoo Itoh, Tohru Kaga, Takashi Nishida, and Yoshifumi Kawamoto. A Block-Oriented RAM with Half-Sized DRAM Cell and Quasi-Folded Data-Line Architecture. In *International Solid-State Circuits Conference*, pages 106–107, San Francisco, February 1991. IEEE.

[89] Toshio Yamada, Yoshiro Nakata, Junko Hasegawa, Noriaki Amano, Akinori Shibayama, Masaru Sasago, et al. A 64Mb DRAM with meshed Power Line and Distributed Sense-Amplifier Driver. In *International Solid-State Circuits Conference*, pages 108–109, San Francisco, February 1991. IEEE.

[90] Shigeru Mori, Hiroshi Miyamoto, Yoshikazu Morooka, Shigeru Kikuda, Makoto Suwa, Mitsuya Kinoshita, et al. A 45ns 64Mb DRAM with a merged Match-line Test Architecture. In *International Solid-State Circuits Conference*, pages 110–111, San Francisco, February 1991. IEEE.

[91] Masao Toguchi, Hiroyoshi Tomita, Toshiya Uchida, Yasuhiro Ooishi, Kimiaki Sato, et al. A 40ns 64Mb DRAM with Current-Sensing Data-BUS Amplifier. In *International Solid-State Circuits Conference*, pages 112–113, San Francisco, February 1991. IEEE.

[92] Yukihito Oosaki, Kenji Tsuchida, Yohji Watanabe, Daisaburo Takashima, et al. A 33ns 64Mb DRAM. In *International Solid-State Circuits Conference*, pages 114–115, San Francisco, February 1991. IEEE.

[93] Rambus. Rambus: Architectural Overview. Technical report, Mountain View, California, 1992.

[94] Peter Gillingham, Richard C. Foss, Valerie Lines, Gregg Shimokura, and Tomasz Wojcicki. High-Speed, High-Reliability Circuit Design for Megabit DRAM. *IEEE Journal of Solid-State Circuits*, 26(8):1171–1175, August 1991.

[95] Richard C. Foss. The Design of MOS Dynamic RAMs. In *International Solid-State Circuits Conference*, pages 140–141, February 1979.

[96] Lanny L. Lewyn and James D. Meindl. Physical Limits of VLSI DRAMs. In *International Solid-State Circuits Conference*, pages 160–161, feb 1984.

[97] Richard C. Foss and Robert Harland. Periferal Circuits for One-Transistor Cell MOS RAM's. *IEEE Journal of Solid-State Circuits*, 10(5):255–261, October 1975.

[98] Nicky C. C. Lu. Advanced Cell Structures for Dynamic RAMs. *IEEE Circuits and Devices Magazine*, 5(1):27–36, January 1989.

[99] Amr Moshen, Roger I. Kung, Carl J. Simonsen, Joseph Schutz, Paul D. Madland, Esmat Z Hamdy, and Mark T. Bohr. The Design and Performance of CMOS 256K Bit DRAM Devices. *IEEE Journal of Solid-State Circuits*, 19(9):610–618, October 1984.

[100] Nicky Chau-Chun Lu and Hu H. Chao. Half-Vdd Bit-Line Sensing Scheme in CMOS DRAM's. *IEEE Journal of Solid-State Circuits*, 19(4):415–454, August 1984.

[101] Richard Comerford and George F. Watson. Memory Catches Up. *IEEE Spectrum*, 29(10):34–35, October 1992.

[102] Richard C. Foss and Betty Prince. Fast Interfaces for DRAMs. *IEEE Spectrum*, 29(10):54–57, October 1992.

[103] Ron Wilson. Jedec Hustling to Spec new SDRAM. *Electronic Engineering Times*, (685):1,8, March 23 1992.

[104] D. A. T. A. Business Publishing. *D. A. T. A. Digest: Memory*. Englewood, CO, 38 edition, 1991.

[105] Micron Technology Inc. 2 Meg VRAM. Technical report, January 1992.

[106] Howard L. Kalter, Charles H. Stapper, jr. John E. Barth, John DiLorenzo, Charles E. Drake, John A. Fifield, jr. Gordon A. Kelley, Scott C. Lewis, Willem B. Van Der Hoeven, and James A. Yankosky. A 50-ns 16-Mb DRAM with a 10-ns Data Rate and On-Chip ECC. *IEEE Journal of Solid-State Circuits*, 25(5):1118–1128, October 1990.

[107] Toshio Takeshima, Masahide Takada, Hiroki Koike, et al. A 55-ns 16Mb DRAM with Built-in Self-Test Function Using Microprogram ROM. *IEEE Journal of Solid-State Circuits*, 24(4):903–911, August 1990.

[108] Hiroyuki Yamauchi, Toshikazu Suzuki, Akihiro Sawada, Tohru Iwata, et al. A 20ns Battery-Operated 16Mb CMOS DRAM. In *International Solid-State Circuits Conference*, pages 44–45, San Francisco, February 1993. Matsushita.

[109] Takehiro Hasagawa, Daisaburo Takashima, Ryu Ogiwara, Masako Ohta, et al. An Experimental DRAM with a NAND-Structured Cell. In *International Solid-State Circuits Conference*, pages 46–47, San Francisco, February 1993. Toshiba.

[110] Goro Kitsukawa, Masashi Horiguchi, Yoshiki Kawajiri, Takayuki Kawahara, et al. 256Mb DRAM technologies for File Appliations. In *International Solid-State Circuits Conference*, pages 48–49, San Francisco, February 1993. Hitachi.

[111] Tadahiko Sugibayashi, Toshio Takeshima dn Isao Naritake, Tatsuya, et al. A 30ns 256Mb DRAM with Multi-Divided Array Structure. In *International Solid-State Circuits Conference*, pages 50–51, San Francisco, February 1993. NEC.

[112] Nadia Lifshitz, Serge Luryi, Mark R. Pinto, and Conor S. Rafferty. Active-Gate Thin-Film Transistor. *IEEE Electron Device Letters*, 14(8):394–395, August 1993.

[113] NEC Corporation. *Memory Products Databook*, 1993.

[114] Hitoshi Miwa, Shoji Wada, Yuji Yokoyama, Masayuki Nakamura, and all. A 17ns 4Mb BiCMOS DRAM. In *International Solid-State Circuits Conference*, pages 56–57, San Francisco, February 1991. Hitachi.

[115] Takeshi Nagai, Kenji Numata, Masaki Ogihara, Mitsuru Shimizu, and all. A 17ns 4Mb CMOS DRAM Using Direct Bit-Line Sensing Technique. In *International Solid-State Circuits Conference*, pages 58–59, San Francisco, February 1991. Toshiba.

[116] Betty Prince. *Semiconductor Memories: A Handbook of Design, Manufacture and Applications*. John Wiley & Sons, Chichester, 2 edition, 1991.

[117] T. Sugibayashi, I. Naritake, S. Utsugi, K Shibahara, R. Oikawa, and all. A 1Gb DRAM for File Applications. In *International Solid-State Circuits Conference*, San Francicso, February 1995.

[118] M. Horigucho, T. Sakata, T. Sekiguchi, S. Ueda, H. Tanaka, and all. An experimental 220MHz 1Gb DRAM. In *International Solid-State Circuits Conference*, San Francicso, February 1995.

[119] Mikio Asakura, Tsukasa Ooishi, Masaki Tsukude, Shigeki, and all. An Experimental 256Mb DRAM with Boosted Sense-Ground Scheme. *IEEE Journal of Solid-State Circuits*, 29(11):1303–1309, November 1994.

[120] Hisakazu Kotani, Hironori Akamatsu, Yasushi Naito, Toyokazu Fujii, and all. A 256Mb DRAM with 100MHz Serial I/O Ports for Storage of Moving Pictures. *IEEE Journal of Solid-State Circuits*, 29(11):1310–1316, November 1994.

[121] Hisakazu Kotani, Hironori Akamatsu, Junko Matsushima, Shozo Okada, Tsuyoshi Shiragasawa, and all. A 50-MHz 8-Mbit Video RAM with a Column Direction Drive Sense Amplifier. *IEEE Journal of Solid-State Circuits*, 25(1):30–35, February 1990.

[122] Betty Prince. Memory in the Fast Lane. *IEEE Spectrum*, 31(2):38–41, February 1994.

[123] NEC. *Memory Products Data Book: Volume 1 DRAMs, DRAM Modules, Video RAMs*. NEC Electronics, USA, 1993.

[124] Richard C. Foss. Implementing Application Specific Memory. In *International Solid-State Circuits Conference*, pages 260–261, San Francisco, February 1996. IEEE.

[125] Peter Gillingham, Betina Hold, Ian Mess, Cormac O'Connell, Paul Schofield, Karl Skjaveland, Randy Torrance, Tomasz Wojcicki, and Hanery Chow. A 768k embedded DRAM for 1.244Gb/s ATM Switch in a 0.8μm Logic Process. In *International Solid-State Circuits Conference*, pages 262–263, San Francisco, February 1996. IEEE.

[126] Glen Giaclone, Robert Busch, Frank Creed, Alex Davidovich, Sri Divakaruni, Charles Drake, and all. A 1MB, 100MHz Integrated L2 Cache Memory with 128b interface and ECC Protection. In *International Solid-State Circuits Conference*, pages 370–371, San Francisco, February 1996. IEEE.

[127] T. Murotani, I. Naritake, T. Matano, T. Ohtsuki, N. Kasai, H. Koga, K. Koyama, K. Nakajima, H. Yamaguchi, H. Watanabe, and T. Okuda. A 4-level Storage 4Gb DRAM. In *International Solid-State Circuits Conference*, pages 74–75, San Francisco, February 1997. IEEE.

[128] Tom Goodman. Application-Specific RAM Architectures Attack New Applications. In *WESTCON 86*, pages 4.3.1–4.3.5, Anaheim, CA, November 1986.

[129] Tony R. Martinez. Smart Memory Architecture and Methods. *Future Generation Computer Systems*, 6(2):145–162, November 1990.

[130] Bob Cushman. Matrix Crunching With Massive Parallelism. *VLSI Systems Design*, 9(12):18–32, December 1988.

[131] J. Hellerstein K. Keeton, D. Patterson. A Case for Intelligent Disks (IDISKs). *ACM SIGMOD Record*, 27(3), September 1998.

[132] Microprocessor Report. Personal Views on the Future of Microprocessors. *Microprocessor Report*, pages 11–14, November 7 1990.

[133] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. Inteligent RAMs: Chips That Remember and Compute. In *International Solid-State Circuits Conference*, pages 224–225, San Francisco, February 1997. IEEE.

[134] Toru Shimizu, Jiro Korematu, Mitsugu Satou, Hiroyuki Kondo, et al. A Multimedia 32b RISC Microprocessor with 16Mb DRAM. In *International Solid-State Circuits Conference*, pages 216–217, San Francisco, February 1996. IEEE.

[135] D. Bearden, R. Bailey, B. Beavers, C. Gutierrez, and all. 133MHz 64b Four-issue CMOS Microprocessor. In *International Solid-State Circuits Conference*, San Francisco, February 1995.

[136] Sun Microsystems. 3D RAM: Revolutionary Frame Buffer Memory. Technical report, Mountain View, CA, July 1994.

[137] Michael F. Beering, Stephen A. Schlapp, and Michael G. Lavelle. FBRAM: A New Form of Memory Optimized for Graphics. In , 1992.

[138] Kazunari Inoue, Hisashi Nakamura, Hiroyuki Kawai, Takahiro Tani, and all. A 10Mb 3D Frame Buffer Memory with Z-Compare and Alpha-Blend Units. In *International Solid-State Circuits Conference*, pages 302–303, San Francisco, February 1995.

[139] David R. Galloway. personal communications. 1990.

[140] Kai Hwang. *Computer Arithmetic.* John Wiley & Son, New York, 1979.

[141] Daniel Booberpuhl, Richard Witek, Randy Allmon, Robert Anglin, and Sharon Britton. A 200MHz 64b Dual-Issue CMOS Microprocessor. In *International Solid-State Circuits Conference*, pages 106–107, San Francisco, February 1992.

[142] C. S. Wallace. A suggestion for a Fast Multiplier. In Jr. Earl E. Swartzlander, editor, *Computer Arithmetic*, pages 114–117. IEEE Computer Society Press, 1990.

[143] Andrew D. Booth. A Signed Binary Multiplication Technique. In Jr. Earl E. Swartzlander, editor, *Computer Arithmetic*, pages 100–104. IEEE Computer Society Press, 1990.

[144] John L. Hennessy and David A. Patterson. *Computer Architecture: a Quantitative Approach.* Morgan Kaufman, San Mateo CA, fourth printing edition, 1990.

[145] Zvi Kohavi. *Switching and Finite Automata Theory.* McGraw Hill, New York, second edition, 1978.

[146] Jeffery D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, Rockville, Maryland, 1984.

[147] Duncan Elliott. Computational RAM: the Design and Implementation of a Memory-SIMD Hybrid. Technical Report TR-98-06-01, University of Toronto, Department of Electrical and Computer Engineering, Toronto, Canada, June 1998.

[148] Christian Cojocaru. Computational RAM: Implementation and Bit-Parallel Architecture. Master's thesis, Carleton University, January 1995.

[149] Oswin Hall. personal communication. 1990.

[150] W. Martin Snelgrove. personal communications. October 1990.

[151] Hamacher, Vranesic, and Zaky. *Computer Organization.* McGraw-Hill, 1978.

[152] Alex G. Dickinson and John S. Denker. Adiabatic Dynamic Logic. In *Custom Integrated Circuits Conference*, pages 12.6.1–12.6.4, May 1994.

[153] Duncan Elliott, Martin Snelgrove, Christian Cojocaru, and Michael Stumm. Computing RAMs for Media Processing. In *Multimedia Hardware Architectures 1997*, pages 7.1–7.12, San Jose, February 1997. SPIE.

[154] Duncan Elliott, Martin Snelgrove, Christian Cojocaru, and Michael Stumm. A PetaOp/s is Currently Feasible by Computing in RAM. In *PetaFLOPS Frontier Workshop*, Washington DC, February 1995.

[155] Allan Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer, Dordrecht, 1992.

[156] Sethuraman Panchanathan and Morris Goldberg. A Content-Addressable Memory Architecture for Image Coding Using Vector Quantization. *IEEE Transactions on Signal Processing*, 39(9):2066–2078, September 1991.

[157] Bernard C. Cole. PC Makers opt for Vector Quantization. *Electronic Engineering Times*, (773):49–52, November 1993.

[158] Sethuraman Panchanathan. Personal correspondence. February, 1993.

[159] Harold S. Stone. Parallel Querying of Large Databases: A Case Study. *Computer*, 20(10):11–21, October 1987.

[160] Gregory Piatetsky-Shapiro. An Overview of Knowledge Discovery in Databases: Recent Progress and Challenges. In Wojciech P. Ziarko, editor, *Rough Sets, Fuzzy Sets and Knowledge Discovery*, pages 1–10. Springer-Verlag, London, 1994.

[161] Usama Fayyad and Ramasamy Uthurusamy. Data Mining and Knowledge Discovery in Databases. *Communications of the ACM*, 39(11):24–26, November 1996.

[162] Wojciech P. Ziarko. Rough Sets and Knowledge Discovery: An Overview. In Wojciech P. Ziarko, editor, *Rough Sets, Fuzzy Sets and Knowledge Discovery*, pages 11–15. Springer-Verlag, London, 1994.

[163] Zdzislaw Pawlak, Jerzy Grzymala-Busse, Roman Slowinski, and Wojciech Ziarko. Rough Sets. *Communications of the ACM*, 38(11):89–94, November 1995.

[164] Hiroto Yasuura, Taizou Tsujimoto, and Keikichi Tamaru. A Functional Memory Type Parallel Processor Architecture for Solving Combinational Problems. *Electronics and Communications in Japan - Part III: Fundamental Electronic Science*, 73(1):23–41, January 1990.

[165] Stephen Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[166] Dickson T. S. Cheung. Colliding Pucks Simulation with C•RAM : A Memory-SIMD Hybrid. December 1994.

[167] Janet A. W. Elliott. personal communications. August 1995.

[168] *Webster's New Collegiate Dictionary*. G. & C. Merriam Co., 1980.

[169] Denis Howe. The On-line Dictionary of Computing. http://wombat.doc.ic.ac.uk/foldoc, 1995.