

Computational RAM: Implementing Processors in Memory

DUNCAN G. ELLIOTT

University of Alberta

MICHAEL STUMM

University of Toronto

W. MARTIN SNELGROVE

Carleton University

CHRISTIAN COJOCARU

Philsar Electronics

ROBERT MCKENZIE

MOSAID Technologies

COMPUTATIONAL RAM is a processor-in-memory architecture that makes highly effective use of internal memory bandwidth by pitch-matching simple processing elements to memory columns. Computational RAM (also referred to as C•RAM) can function either as a conventional memory chip or as a SIMD (single-instruction stream, multiple-data stream) computer. When used as a memory, computational RAM is competitive with conventional DRAM in terms of access time, packaging, and cost. As a SIMD computer, computational RAM can run suitable parallel applications thousands of times faster than a CPU. Computational RAM addresses many issues that prevented previous SIMD architectures from becoming commercially successful. While the SIMD programming model is somewhat restrictive, computational RAM has applications in a number of fields, including signal and image processing, computer graphics, databases, and CAD.

Motivation

One motivation behind today's emerging smart memories¹⁻⁷ is to exploit the chips' wide internal data paths. Another is to exploit the energy efficiencies that result from better utilization of memory bandwidth and localization of computations on a millimeter scale. Signal pathways within memory chips provide memory bandwidth many orders of mag-

nitude higher than that available to an external processor. For example, a 256-Mbyte memory system has about 3,000 times as much accessible memory bandwidth within the chips than that available to an external processor (optimistically assuming a 100% cache hit rate). For applications with poor cache behavior, the difference can increase to 15,000 times as much. But to effectively utilize this internal memory bandwidth, logic and memory must be more tightly integrated than merely being located on the same chip.

We can add processors to memory with minimal overhead if we use methods compatible with the efforts that memory designers have made to minimize the cost of memory (DRAM in particular). A key goal is to remain compatible with commodity DRAM in cost, silicon area, speed, packaging, and IC process, while accessing a significant fraction of the internal memory bandwidth. To preserve the memory's economics, we must work within the existing number of routing layers, preserve the number of memory cells per row address decoder and sense amplifier, and use redundancy to correct manufacturing defects.

To harvest as much memory bandwidth as possible, we pitch-match the computational RAM processing elements to a small number (for example, 1, 2, 4, or 8) of memory columns, as shown in Figure 1. The use of a common memory row address shared by a

Adding logic to memory is not a simple question of bolting together two existing designs. Computational RAM integrates processing power with memory by using an architecture that preserves and exploits the features of memory.

row of processing elements dictates that the processing elements have a SIMD architecture. To design such a narrow processing element, we followed a minimalist architectural philosophy. We chose circuits with a narrow VLSI implementation and reused the same circuits at different times to perform different functions. With this design, area overhead can range from 3% to 20%, while power overhead can be 10% to 25%, compared to the memory alone. Such chips could add a massively parallel processing capability to machines and systems that currently use DRAM. Figure 2 shows how computational RAM chips could serve as both computer main memory and graphics memory.

In the computational RAM architecture's programming model, a host CPU can read and write to any memory location during an external memory cycle. During an operate cycle, all processing elements execute the same common instruction and optionally access the same memory offset within their private memory partitions. In other words, computational RAM is a SIMD processor with distributed, nonshared, uniformly addressed memory. A bus facilitates interprocessor communication and is useful for combinational operations. In addition, a linear interconnect that extends to two dimensions at the ends of rows is useful for 1D and 2D nearest-neighbor operations.

Memory bandwidth and power consumption

Consider the internal structure of a DRAM, as shown in Figure 3. DRAMs consist of large numbers of subblocks, each with a row-column structure. Typical DRAMs have a simple sense amplifier at the bottom of each 256-bit column and use shared circuitry to select a single row of data.

Computational RAM gains its key advantages—access to internal memory bandwidth and low energy usage—from the fact that the processing elements are integrated into the memory arrays. The row-column logical architecture is usually roughly square because similar numbers of address bits are allocated to row and column. (Historically, the address pins are multiplexed between row and column, and the square architecture minimizes their number.)

Memory bandwidth. DRAM is organized with a very wide internal data path at the sense amplifiers. A $4M \times 1$ -bit

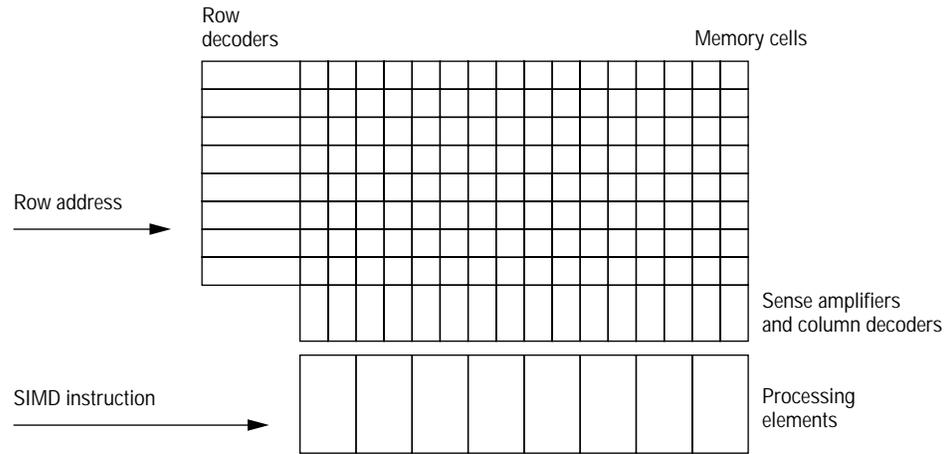


Figure 1. Processing elements incorporated in memory.

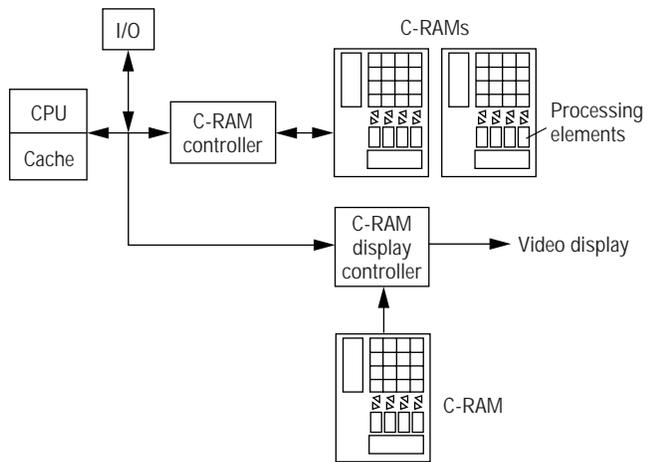


Figure 2. Replacing DRAM with computational RAM and redefining support logic.

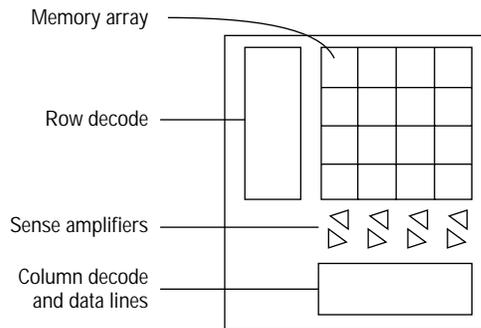


Figure 3. Basic DRAM structure.

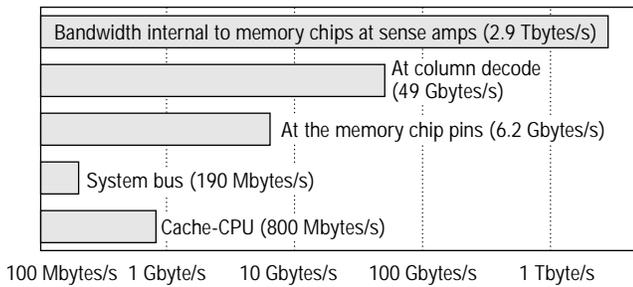


Figure 4. Bandwidths at various points in a workstation.

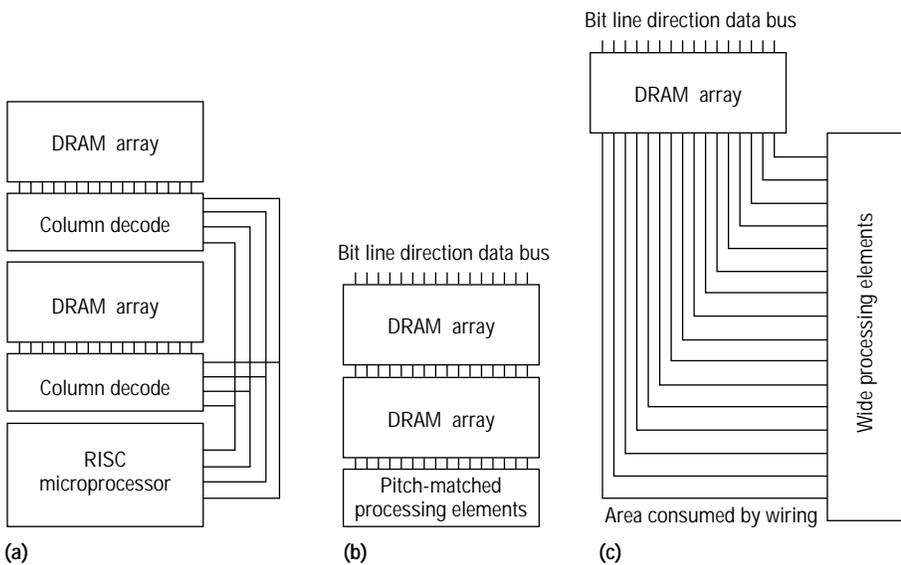


Figure 5. Architectural alternatives for processing in memory: a RISC processor connected after column decoders (a); pitch-matched processing elements connected to memory—the computational RAM approach (b); non-pitch-matched processing elements connected by wiring (c).

chip fetches (at least) 2 Kbits when the row address is given and then selects one bit for output according to the column address. Similarly, a $1\text{M} \times 16\text{-bit}$, 1K-cycle-refresh DRAM selects 16 Kbits with the 10-bit row address, and then one of 1,024 sixteen-bit words for output when the column address is available. In each case, the width of the internal data path is 1K to 2K times the width of the external data path. In systems with large amounts of memory, multiplexing banks of RAM onto a narrow bus limits bandwidth even further.

As an example, consider a 100-MHz workstation with a 64-bit bus, equipped with a total of 256 Mbytes formed from 16-Mbit, 50-ns page-mode DRAM. The data path at the sense amplifiers is 2 Mbits wide, resulting in a memory bandwidth more than four orders of magnitude higher than that available at the system bus (see Figure 4). Even an ideal cache improves the bandwidth by only a factor of four, leaving a gap of three

and a half orders of magnitude between the bandwidth available in the memory and at the CPU. Redoing the example with a smaller or larger computer gives similar ratios, because memory size tends to scale with processing power.

Adding one processing element per DRAM sense amplifier is not very practical, because sense amplifiers are placed on a very narrow pitch. But it is practical enough for one processing element to share four sense amplifiers over many generations of DRAM processes.

A more mainstream architectural alternative to pitch-matching processing elements to groups of sense amplifiers is to put a single RISC or vector processor in a DRAM chip.^{8,9}

This approach allows a wide variety of conventional programs to be compiled and run without modification or attention to data placement and communication patterns. Such a processor has access to a wider bus (128 to 256 bits) for cache or vector register fills than it would have if implemented on a separate chip. Still, by connecting to the memory after the column decoders, the 16-Kbit-wide data path at the sense amplifiers multiplexes down by a factor of 64 or more. In contrast, computational RAM, with processing elements pitch-matched to four sense amplifiers, can make effective use of 25% of the internal memory bandwidth.

Figure 5 illustrates these approaches to connecting processors and memory. Figure 5a shows a wide bus coming from the DRAM's sense amplifiers, multiplexed by the column decode circuitry onto a narrower on-chip data bus. Figure 5b shows a wide bus running through the DRAM arrays, connecting the sense amplifiers to the processing elements. (In several of our chips, this wide bus has negligible length because the processing elements simply abut the sense amplifiers, as shown in Figure 1.) The use of the wide bus does not require an additional layer of wiring, since it allows us to omit column decode signals, typically routed in the same direction.¹⁰ A small decoder (not shown) selects one of four (for example) sense amplifiers to connect to a bus signal. If processing elements are designed with a pitch of a different number of sense amplifiers, the bandwidth utilization changes. Simply ignoring the sense amplifiers' pitch

and relying on wiring to compensate is potentially wasteful of silicon area, as Figure 5c illustrates.

Power consumption. Power consumption is rapidly becoming a key measure of merit of computer architecture because it has been increasing with processing speed and because of the interest in portable computing. High-power chips require expensive packaging at both chip and system levels (to conduct and convect heat), and they are unreliable because of the rapid aging that accompanies high-temperature operation.

An internal DRAM bus is much more energy efficient (as well as faster) than an external bus because shorter wires must be driven. The power required to drive a wire is $CV_{DD}V_{swing}f$. Here, C is the wire's capacitance (in the range of 0.2 pF/mm, depending on geometries), V_{DD} is the power supply voltage, V_{swing} is the voltage swing used to represent data, and f is the rate at which data are clocked. C favors short buses directly, and V_{swing} favors them indirectly (and relatively slightly) in that we need a smaller noise margin for reliable logic operation when currents remain small and coupling is minimized. In a typical 16-Mbyte DRAM, C is 0.3 pF for a single bit line but would be about 100 times larger for a single bit of an off-chip bus running to a CPU. In the same memory, V_{swing} would normally be about 3.3 V, both on and off chip. The pins of high-speed memories often use reduced voltage swings with properly terminated transmission lines, reducing the ringing effects that would otherwise call for a good noise margin, but such systems dissipate power at the terminations.

$CV_{DD}V_{swing}$ measures switching energy, coming to about 330 pJ for a 30-pF bus wire at $V_{DD} = V_{swing} = 3.3$ V. For the bit line, V_{swing} is $1/2 V_{DD}$, so the switching energy is 1.6 pJ. We can express these energies more mnemonically as 330 μ W/MHz and 1.6 μ W/MHz, respectively: 16K bit lines cycling at 10 MHz require 270 mW; 16 bus lines clocked at 100 MHz (cycling at 50 MHz) require 260 mW. We can save a sizable portion of the power by not driving signals off chip.

Computational RAM architectures

The design space available for integrating processing with memory is very large, and even the low-cost SIMD corner that we have been exploring is large. Figures 6 and 7 show two candidate computational RAM processing elements. We have implemented both designs in silicon using static RAM (SRAM) memories. We have demonstrated their compatibility with DRAM by creating physical designs in 4-Mbit and 16-Mbit DRAM processes.

The simpler of the two processing elements, in Figure 6, supports bit-serial computation and has left-right and wired-AND based communication. The ALU, consisting of an 8-to-1 multiplexer, has a compact VLSI implementation. Thus,

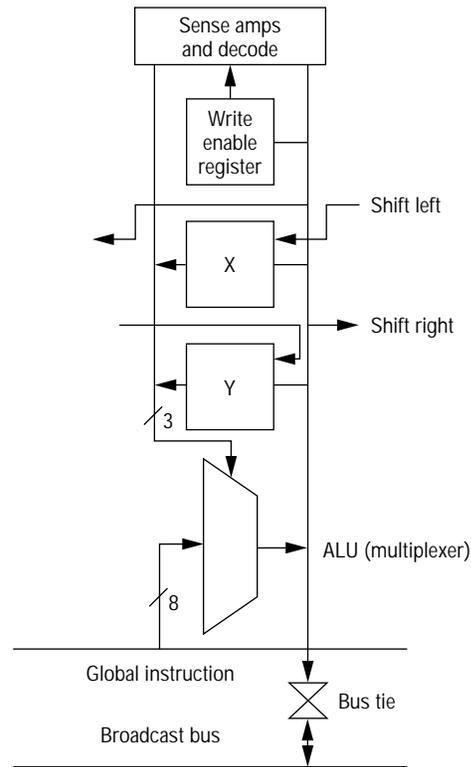


Figure 6. A simple processing element that can be implemented with fewer than 100 transistors, using a dynamic logic multiplexer.

we can implement an entire processing element (including the off-chip read/write path) with as few as 88 transistors using dynamic logic. This number is small compared to the number of transistors used to implement the processing element's local memory in the columns above it. The control signals (derived from a 13-bit SIMD instruction) are routed straight through a row of processing elements.

In this architecture, the ALU can perform an arbitrary Boolean function of three inputs: X and Y registers and memory. The ALU opcode, connected to the data inputs of the ALU multiplexer, is the ALU's truth table. The result can be written back to either the memory or the X, Y, or write-enable register. The write-enable register is useful for implementing conditional operations.

This processing element was designed in the pitch of one column of SRAM and four columns of DRAM. Since the processing element requires a pitch of only seven wires, the design fits in the pitch of eight bit lines (four folded bit-line pairs or columns) across many generations of DRAM. Each processing element's connection to at least four sense amplifiers means the processing elements can have fast page-mode access to at least 4 bits; the sense amplifiers then form

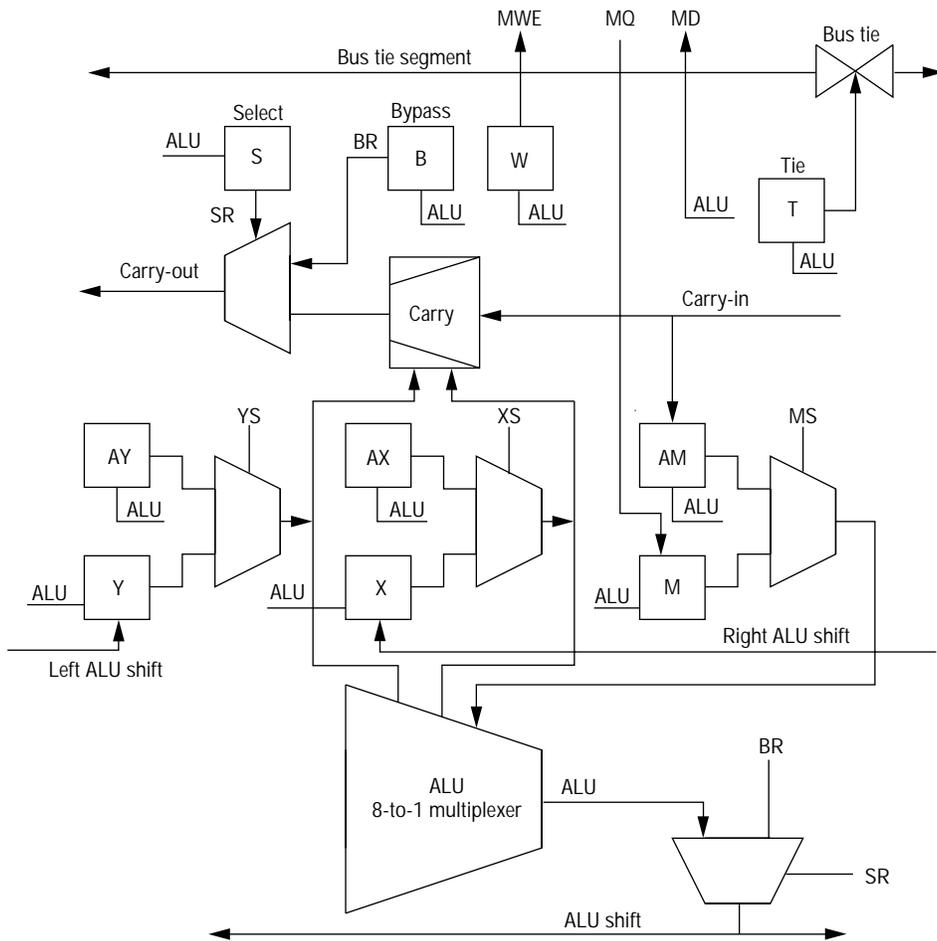


Figure 7. A more complex processing element using 147 transistors supports bit-parallel grouping. The additional registers (S, B, T, AX, AY, AM) make faster multiplication and new modes of communication possible.

a primitive cache. The processing elements and support circuitry add 18% to the area of an existing DRAM design. A single processing element occupies an area of approximately 360 bits of memory (including sense amplifier and decoder overhead).

To make the most effective use of silicon area, structures in this processing element often serve multiple purposes. We use the X and Y registers to store results of local computations (such as sum and carry) as well as to act as the destination for left and right shift operations between adjacent processing elements. The simultaneous bidirectional bus transceiver can drive the broadcast bus, receive from the bus, or do both at once (0 dominates). During communication operations, we use the ALU to route signals.

We originally developed the more complex, 147-transistor implementation (Figure 7),⁵ to complement an SRAM on an ASIC process, which has much larger memory cells and

in which memory costs dominate. Its principal enhancements are more registers to reduce the number of RAM cycles required and direct support for grouping adjacent processors to work on multibit data. In particular, the ripple-carry AND registers allow a reduction in latency for bit-parallel multiplication. This is particularly valuable when an application doesn't have sufficient parallelism to use all processing elements.

One reason why these processing elements require so few transistors is that a SIMD controller performs all instruction fetching and decoding operations centrally. The native computational RAM instruction consists of a memory address (typically sent first), an ALU operation, a destination (register or memory), and other control signals including those controlling communication.

We have experimented with two well-known approaches^{4,11} to issuing SIMD instructions. The first is to have the host microprocessor

issue native computational RAM instructions directly via a memory-mapped IO register. The second approach uses a microcoded SIMD controller that receives macroinstructions (such as "ADD 32-bit") from the host processor. The controller translates the macroinstructions into multiple native computational RAM instructions. The direct approach requires less hardware, whereas the microcoded controller can overlap SIMD execution with host execution.

The computational RAM processing elements, especially in the first architecture, require that data be stored orthogonally to the way the host would expect to access it. When the processing elements perform bit-serial arithmetic, the bits of each vector element are stored along a column during separate memory accesses. The host, however, would typically access a processor word of memory with the bits belonging to the same memory row. A corner-turning cache resolves both views of memory by mapping processor words to vec-

for elements with the same number of memory accesses a conventional cache memory system would make.

Effects of DRAM pinout

DRAMs traditionally conserve pins by multiplexing row and column addresses and by having relatively narrow I/O buses. This has helped to keep packaging and board-level system costs low. The large die sizes of modern DRAMs—1 to 2 square centimeters—make larger numbers of pins practical. But when processing elements are added, inter-processor communication, I/O, and control are still constrained by the number of pins available. On a platform, for example, we can

implement computational RAM in the standard, 44-pin thin small-outline package (TSOP) by multiplexing opcodes with addresses (and data), since they aren't required simultaneously. The standard, 16-Mbyte, 1K-cycle refresh DRAM uses 37 of the 44 package pins (16 data, 10 address, 6 power, and 5 control). Computational RAM requires one additional control pin, the opcode strobe (\overline{OPS}), and four communications pins, which fit in the same package as a JEDEC DRAM. (Conserving pins is also a power control issue, as discussed earlier.)

The interprocessor communications constraint arises when a large system is to be composed of many computational RAM chips. For example, a shuffle network extendable to a multiple-chip system may require that one line per processor cross the chip boundary, resulting in a requirement of 4K pins for interprocessor communications alone. This in turn would require nearly a thousand power supply pins to handle the drive currents. This is clearly unrealistic and would result in enormous power dissipation if the pins ran at full speed. Any network efficient enough to move all 4 Kbits in or out of a chip in a cycle will have this problem, so we are forced to limit interprocessor communications.

Even a 2D interconnect would be expensive if 4,096 processing elements were arranged in a square, since the periphery would have 256 wires needing connection to neighboring processing elements on other chips. For this reason, we favor 1D structures such as buses and shift registers, perhaps interconnected externally to facilitate 2D shifting,

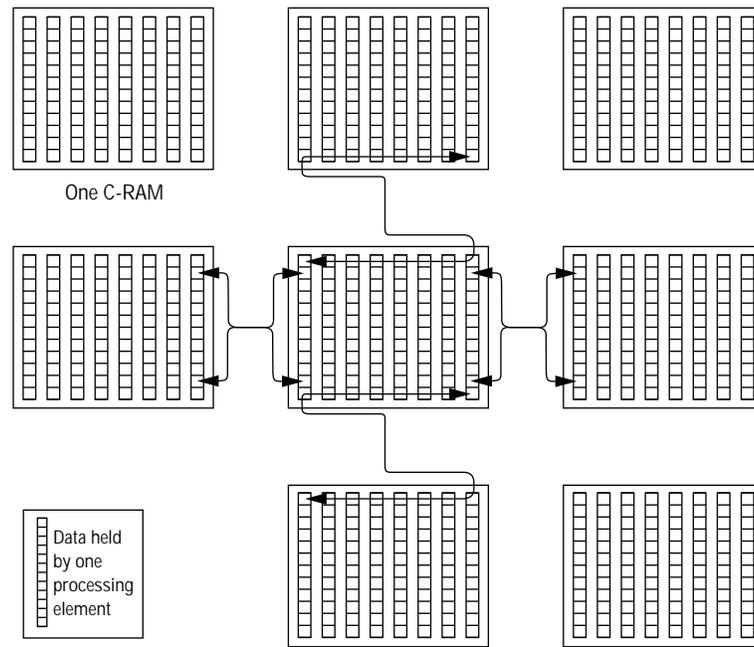


Figure 8. Chip-to-chip communication as a square array of shift registers.

as shown in Figure 8. For a 2D image-processing problem in which each pixel depends on its immediate neighbors, we assign the pixels to processing elements in vertical stripes, one pixel wide. For east-west communication, we shift the pixel values left or right by one. For north-south communication, we shift out the top (bottom) elements of each stripe in an entire row of processing elements into the chip above (below) it. Concurrently, we shift in a row from the chip below (above). These links can be single-bit for minimum cost (four pins total) or widened for better performance.

Effects of DRAM technology

DRAM technology is quite different from the technologies usually used for processors,¹⁰ presenting certain problems for computational RAM. In particular, DRAMs may use one to three layers of metal, because that is enough for a memory array. In contrast, high-performance processors use four or five layers. The difficulty is not technical, but economic: if the processor needs five layers of metal, the extra metal layers are wasted over the DRAM array. In a competing architecture that segregates processing and memory, the dominant silicon area devoted to memory will cost less.

The characteristics and operating conditions of DRAM transistors make them slower than transistors in an equivalent ASIC or digital logic process. To maximize the DRAM refresh interval, we must control transistor leakage currents by increasing the transistor threshold voltage and applying a negative bias to the substrate (back bias). To allow appli-

Table 1. A brief history of SIMD systems (PE: processing element).

Machine	Year	On-chip main mem.	Mem. redundancy	Mem. bits/PE	Local mem.	PEs/chip	PE redundancy	Max. PEs	Data path width	Auton. mem. addressing	Auton. net-work
Staran	1972			256		8		1K	1		
ICL-DAP	1976			4,096	✓	16		1K	1		
Vastor	1978				✓	1			1		
MPP	1981				✓	8		16K	1		
GAPP	1984	✓		128	✓	72			1		
CM-1	1985			4,096		16		64K	1	*	✓
Pixel Planes 4	1987	✓		72	✓	64		256K	1		
AIS	1988			32K	✓	8		1K	1		
Blitzen	1989	✓		1,024	✓	128		16K	1	✓	
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	✓	1,024			1		
CM-2	1990			1M		16		64K	1 & 64	*	✓
MasPar MP-2	1993				✓	32		16K	32	✓	✓
SRC-PIM	1994	✓	**	2,048	✓	64		256K	1		
NEC-IMAP	1994	✓	**	32K	✓	64			8	✓	✓
Execube	1994	✓	✓	512K	✓	8			16	✓	✓
C-RAM	1995	✓		480	✓	512		4,096	1+		
NEC-PIPRAM	1996	✓	**	128K	✓	128	✓		8		
Sony Linear Array	1996	✓		256	✓	4,320			1		
Gealow & Sodini	1997	✓		128	✓	4,096			1		
Accelerix	1998	✓	✓	3,264	✓	4,096	✓				

* emulated in software
 ** replacement of entire memory blocks

cation of a boosted voltage to the word lines, the gate oxide of the cell access transistors must be thicker. The increased threshold voltage, negative back bias, and thicker oxide diminish the transistors' drive, reducing their speed compared to ASIC transistors.

Several IC manufacturers that offer merged logic-DRAM processes have addressed these three problems through the use of 1) a separate implant mask for the memory cell array, 2) a separately biased well for the memory cell array, and 3) two thicknesses of gate oxide. Faster logic in DRAM is available at the expense of these extra process steps. Since it is the DRAM cycle time that largely determines computational RAM's performance, computational RAM would see only a small benefit from a faster merged logic-DRAM process. Designed as it is in commodity DRAM processes, computational RAM can be manufactured at lower cost than in a merged logic-DRAM process.

Cycle times in 16- to 256-Mbit DRAMS are typically in the

range of 150 to 68 ns, limited by the word-line and bit-line lengths. They can be faster, but at a cost in memory cell efficiency as the proportion of chip area devoted to overhead such as drivers and amplifiers rises. In any technology, a processing element has shorter lines and hence can cycle faster (and dissipate less power) than the DRAM array. This makes it practical to interpose two processor cycles in each memory cycle.

DRAM also relies heavily on redundancy to improve yield, which would otherwise be quite low due to high densities and large dies. Processors, on the other hand, are usually designed without redundancy. Building processing elements without redundancy is acceptable since they occupy a small fraction of die area and therefore have limited effects on yield. But using redundancy also enhances performance by permitting the use of smaller-feature design rules.¹⁰ Row redundancy is transparent to the processing elements, but column redundancy is more difficult to deal with. Computational RAM processing el-

ements can certainly be replaced column by column together with groups of bit lines, so most of the logic in computational RAM can benefit from column redundancy.

However, maintaining interprocessor communication is a problem. If we swap out a bad column, we must also rewire the connections to its neighbors. Thus, it is important to use a simple interprocessor communication scheme that allows wiring around a bad processor without too much overhead. One solution is a simple switching network that can rewire the connections between processing elements and local memory past a bad column.³

Computational RAM as a SIMD system

The computational RAM processing elements share a common instruction and thus operate in a SIMD mode. SIMD computing has a mathematical elegance that has tended to draw interest, but the interest is often followed by disappointment. As a result, SIMD has a long and rather checkered history, as outlined in Table 1.^{2,7,11,12} Since the middle of this decade, essentially all new massively parallel SIMD designs have used embedded memory. Of these, Execube, PIPRAM, and Accelerix use high-density, one-transistor-cell DRAM.

The following are some of the problems of SIMD computers:

- They tend to be large, expensive, low-volume or even prototype machines rather than commodity computers.
- Wide buses between processing elements and memory consume many pins and hence much board area and often limit the number of processing elements that can be integrated in one chip.
- They tend to have a bottleneck at the point where data transfers to and from the controlling host.
- Some processing elements sit idle during execution of conditional code because the shared instruction stream forces the controller to execute all paths. (That is, it executes both the “then” and the “else” of an “if” statement, the maximum number of iterations of a loop, and so on.)
- SIMD computers get high performance only on applications that offer the appropriate parallelism.
- There aren’t many programmers who have experience with the model.
- At present, SIMD application code typically is not portable.

Many of these problems are tightly linked. For example, the machines’ size and cost are driven largely by their generally low level of integration. This in turn is driven by the need to minimize nonrecurring costs rather than unit costs in designing for a specialized high-cost market. Although the last four problems listed are fairly inherent to SIMD architectures, computational RAM’s integration of processors

into memory is key to solving the other problems.

Pin count, size, and price. Until the late 1980s, many SIMD designers used large numbers of IC pins because they decided to cut development costs by using commodity memory. As a result, they lost access to the wide, low-power, internal data bus that is computational RAM’s *raison d’être*. In this technology, we can obtain low unit costs only by integrating the processors tightly with memory; there is no cheap route to the consumer market. Also, in computational RAM designs, the memory itself dominated the chip area, since a consumer facing a choice between 16 Mbits of DRAM and 8 Mbits of computational RAM for the same price will probably choose the DRAM.

Host-SIMD bottleneck. Computational RAM does not attempt to handle the serial fraction of a computation well, leaving that to the host CPU. The path of intermediate results between host and SIMD machines must not become a bottleneck. It must allow sequential portions of applications to run on the host without expensive transfers.

Since computational RAM is the host’s memory, there is in principle no need for data to move. This argument needs a qualifier, however: Because the data organizations best suited to the host and to computational RAM are different, we may need to transpose data.

Benchmarks

We developed a C++ compiler/simulator to generate and simulate computational RAM instructions, counting the cycles needed for applications. Table 2 (next page) shows representative timings for 32 Mbytes of computational RAM (128K processing elements) simulated with a conservative 150-ns cycle versus a 70-MHz microSparc (measured). For the applications considered, computational RAM runs several orders of magnitude faster than the conventional workstation. The speedup is so large because the computational RAM processor can directly access the DRAM’s internal bandwidth. Since CPU speeds and typical memory sizes grow over time, we anticipate that the computational RAM approach will continue to offer speedup through many generations of systems. Elliott¹² gives details of the applications.

However, we do not claim this type of performance for all or even most applications. For example, the Dhrystone or SPEC rating of computational RAM would be very poor. The computational RAM philosophy is that largely sequential applications belong on the host, and the massively parallel component belongs in the memory.

Incidentally, testing is another application for which computational RAM obtains a parallel speedup. The processing elements can be tested and then, themselves, perform the memory tests in less total time than it would take to test a

Table 2. Benchmark speedups.

Program	C-RAM runtime	Sun Sparc runtime	C-RAM speedup ratio	Parallelism
Vector quantization	25.7 ms	33.8 s	1,312	Image space
Masked blt	18.2 μ s	443 ms	24,310	Image space
3 \times 3 convolution 16M	17.6 ms	113 s	6,404	Image space
FIR 128K tap, 40-bit	99 μ s	312 ms	3,144	Coefficients
FIR 4M tap, 16-bit	1.04 ms	5.14 s	4,929	Coefficients
LMS matching	0.20 ms	251 ms	1,253	Records
Data mining	70.6 ms	192 s	2,724	Rule space
Fault simulation	89 μ s	3.9 s	43,631	Fault space
Satisfiability	23 μ s	959 ms	41,391	Solution space
Memory clear	1.6 μ s	8.8 ms	5,493	Memory

similar-capacity memory.

Table 2 does not estimate power, but computational RAM's energy improvements are related to its speed improvements because processing element power is smaller than sense amplifier power. As a result, a computer equipped with computational RAM consumes little more power than one without but finishes the task much sooner, thereby consuming less energy per application. In addition, the reduced need to pump data out and back over a bus should save power. Computational RAM, however, uses more sense amplifiers at once and typically has less effective memory caching, reducing the power advantage. Also, the memory access patterns of the parallel algorithms are not necessarily the same as those of the sequential algorithm.

AS WE HAVE SHOWN, adding logic to memory is not a simple question of bolting together two existing designs. Memory and logic technologies have different characteristics, and a memory looks very different inside the chip than it does at the pins. Computational RAM successfully integrates processing power with memory by using an architecture that preserves and exploits the features of memory. Additional information about computational RAM is available at <http://www.ee.ualberta.ca/~elliott/cram/>. 

Acknowledgments

This article is based on a paper presented at the SPIE Multimedia Hardware Architectures Conference in San Jose, Calif., February 1997. Figures and tables from that paper are reprinted here with the permission of the Society of Photo-Optical Instrumentation Engineers.

MOSAID Technologies has hosted and supported our work on this project for several years, and its staff has been generous in ex-

plaining the real-world constraints of DRAM to us. We thank Peter Nyasulu, Dickson Cheung, Sethuraman Panchanathan, Tet Yeap, Wayne Loucks, Thinh Le, Albert Kwong, Bruce Cockburn, Roger Mah, Dick Foss, Peter Gillingham, Graham Allan, Iain Scott, Randy Torrance, David Frank, David Somppi, Randy Gibson, Howard Kalter, John Barth, Richard White, and Tom Little for technical exchange and feedback. In addition to MOSAID, our work has received support from IBM, Nortel, Accelerix, NSERC, CMC, and Micronet.

References

1. H.S. Stone, "A Logic-in-Memory Computer," *IEEE Trans. Computers*, Vol. C-19, No. 1, Jan. 1970, pp. 73-78.
2. D.G. Elliott, W.M. Snelgrove, and M. Stumm, "Computational RAM: A Memory-SIMD Hybrid and Its Application to DSP," *Proc. Custom Integrated Circuits Conf.*, IEEE, Piscataway, N.J., 1992, pp. 30.6.1-30.6.4.
3. N. Yamashita et al., "A 3.84GIPS Integrated Memory Array Processor with 64 Processing Elements and 2Mb SRAM," *IEEE J. Solid-State Circuits*, Vol. 29, No. 11, Nov. 1994, pp. 1336-1343.
4. M. Gokhale, B. Holmes, and K. Iobst, "Processing in Memory: The Terasys Massively Parallel PIM Array," *Computer*, Vol. 28, Apr. 1995, pp. 23-31.
5. C. Cojocaru, *Computational RAM: Implementation and Bit-Parallel Architecture*, master's thesis, Carleton Univ., Dept. of Electronics, Ottawa, Ont., Canada, 1995.
6. J.C. Gealow and C.G. Sodini, "A Pixel-Parallel Image Processor Using Logic Pitch-Matched to Dynamic-Memory," *Proc. Symp. VLSI Circuits*, IEEE, Piscataway, N.J., 1997, pp. 57-58.
7. R. Torrance et al., "A 33GB/s 13.4Mb Integrated Graphics Accelerator and Frame Buffer," *Proc. Int'l Solid-State Circuits Conf.*, IEEE, Piscataway, N.J., 1998, pp. 340-341.
8. T. Shimizu et al., "A Multimedia 32b RISC Microprocessor with 16Mb DRAM," *Proc. Int'l Solid-State Circuits Conf.*, IEEE, Piscataway, N.J., 1996, pp. 216-217.
9. D. Patterson et al., "A Case for Intelligent RAM," *IEEE Micro*, Vol. 17, No. 2, Mar. 1997, pp. 34-44.
10. R.C. Foss, "Implementing Application Specific Memory," *Proc. Int'l Solid-State Circuits Conf.*, IEEE, Piscataway, N.J., 1996, pp. 260-261.
11. W.D. Hillis, *The Connection Machine*, MIT Press, Cambridge, Mass., 1985.
12. D. Elliott, *Computational RAM: A Memory-SIMD Hybrid*, doc-

toral thesis, Univ. of Toronto, Dept. of Electrical and Computer Engineering, 1998.



Duncan G. Elliott is an assistant professor in the Department of Electrical and Computer Engineering at the University of Alberta, Edmonton, Canada. His research interests are logic-enhanced memories, computer architecture, and parallel processing. His application-specific memory inventions have been adopted by industry. Previously, he has worked at Nortel in data communications, at MOSAID Technologies as a DRAM designer, and at IBM Microelectronics as a contractor in application-specific memory design. Elliott received his BSc in engineering science and his master's and doctorate degrees in electrical and computer engineering from the University of Toronto. He is a member of the IEEE, the Computer Society, the Solid-State Circuits Society, and the ACM.

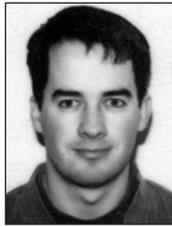
Michael Stumm is a professor in the Department of Electrical and Computer Engineering and the Department of Computer Science at the University of Toronto. His research interest is operating systems for distributed and parallel computer systems. Stumm received a diploma in mathematics and a PhD in computer science from the University of Zurich. He is a member of the IEEE Computer Society and the ACM.



W. Martin Snelgrove is the director of R&D at Philsar Electronics of Ottawa and the chief technology officer of Wireless System Technologies. His recent research work has been in adaptive analog and digital filtering, data converter architecture and circuits, and highly parallel architectures for signal processing.

He contributed to this article while he was a professor at Carleton University in Ottawa, where he held the OCRI/NSERC Industrial Research Chair in High-Speed Integrated Circuits. He has published about 100 papers, one of which won the 1986 CAS Society Guillemin-Cauer Award. Snelgrove received a BSc in chemical engineering, and an MSc and a PhD in electrical engineering from the University of Toronto. He is a member of the IEEE.

Christian Cojocar works at Philsar Electronics, Ottawa, on mixed-signal ICs for integrated radio transceivers. He received the DiplEng degree in electrical engineering from the Polytechnic Institute of Bucharest, and the MEng degree in electronics from Carleton University, Ottawa. He is a member of the IEEE.



Robert McKenzie works at MOSAID Technologies, Kanata, Ontario, Canada, on the design of application-specific memories for graphics engines and networking components. He received the BSc in computer engineering from the University of Toronto and the MEng from Carleton University.

Send comments and questions about this article to Duncan Elliott, Dept. of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada T6G 2G7; duncan.elliott@ualberta.ca.

1999 Special Issues

D&T focuses on practical articles of near-term interest to the professional engineering community. To further this goal, the Editorial Board has set the following special issues:

April-June

Reengineering Digital Systems

Guest Editor: Vijay K. Madiseti, Georgia Institute of Technology; vke@ee.gatech.edu

July-September

Test and Product Life Cycle

Guest Editors: Tony Ambler, Univ. of Texas at Austin; ambler@ece.utexas.edu
Ben Bennetts, Bennetts Associates; benb@burridge.demon.co.uk

October-December

Microelectromechanical System Design and Test

Guest Editors: Shawn Blanton, Carnegie Mellon Univ.; blanton@ec.e.cmu.edu
Bernard Courtois, TIMA-CMP; bernard.courtois@imag.fr

IEEE Design & Test is a quarterly publication of the IEEE Computer Society.
Editor-in-Chief: Yervant Zorian, LogicVision Inc., 101 Metro Drive, Third Floor, San Jose, CA 95110; zorian@lvision.com.