

Minimizing the Effect of the Host Bus on the Performance of a Computational RAM Logic-in-Memory Parallel-Processing System

Peter M. Nyasulu, Ralph Mason, W. Martin Snelgrove, and Duncan G. Elliott[‡]

Department of Electronics, Carleton University, Ottawa, Ontario, Canada

[‡]Department of Electrical and Computer Engineering, University of Alberta, Edmonton, Alberta, Canada

Abstract - This paper describes the system design techniques that have been employed to minimize the effect of the host bus on the performance of a Computational RAM (CRAM) logic-in-memory parallel-processing system. Specifically, we describe how the architectural features of the CRAM controller affect instruction execution, utilization of processing elements, time to initialize parallel variables from the host computer, and execution time of scalar operations. Finally, we show that because of the performance-enhancement features of the controller, the transfer characteristics of the host bus has very little effect on the performance of a CRAM system. This means that a CRAM system can be implemented on a wide variety of platforms, including those with slow external buses such as ISA-based computers and embedded systems that use slow microcontrollers.

I. INTRODUCTION

Computational RAM (CRAM) is a SIMD-memory hybrid architecture [1], with 1-bit processing elements (PEs) integrated at the sense amplifiers of a standard DRAM/SRAM. CRAM is designed to improve the speed of executing massively-parallel applications by utilizing the high bandwidth available at the sense amplifiers [2], [3], [4]. Several PEs can have access to the data at the sense amplifiers and operate on it without the need to read the data out of the RAM and transmit it over long buses to the processor. This improves performance and also reduces power consumption. Figure 1 shows the architecture of CRAM and its bit-serial PE. A PE instruction consists of multiplexer truth-table and the result control bits. These are multiplexed with data and address on the RAM data and address buses, respectively.

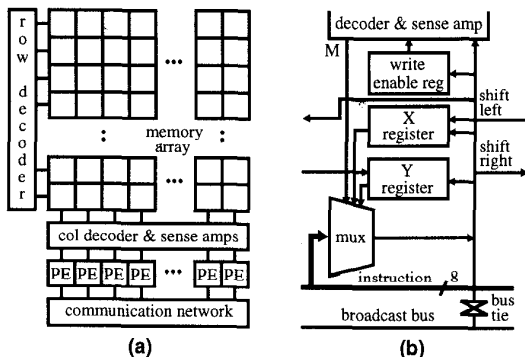


Fig. 1. (a) CRAM Architecture (b) Processing Element

Other architectures similar to CRAM include IMAP [5], PIP-RAM [6] and Terasys PIM Array [7]. However, unlike most logic-in-memory systems which are designed for specific applications, especially pixel processing and graphics ([5], [6], [8], [9], [10]), and are implemented mostly on the VME bus, CRAM is intended as a general-purpose computational system on different platforms. One platform specifically targeted in our initial prototypes is the Personal Computer (PC) environment. Figure 2 shows CRAM in a typical PC system. We have implemented a 10 MHz ISA CRAM system prototype, and are currently implementing a PCI system (shaded boxes in Figure 2). It is our goal that eventually CRAM will replace or coexist with the standard DRAM chips as the computer main memory (dashed boxes in Figure 2).

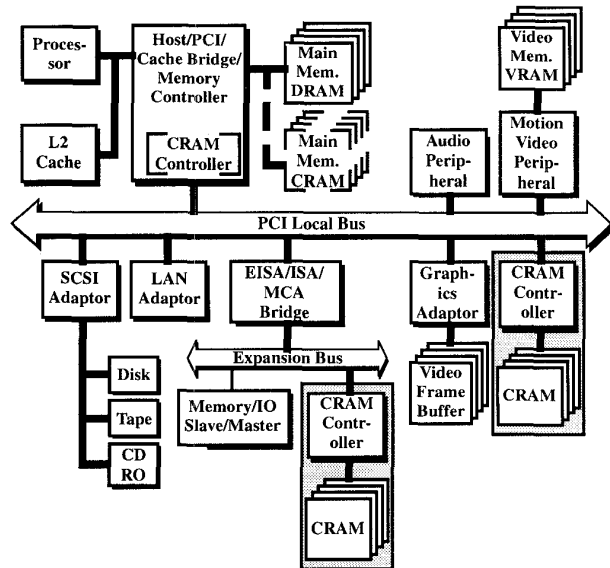


Fig. 2. CRAM in a Typical Computer System

II. CRAM CONTROLLER ARCHITECTURE

As shown in Figure 2, a CRAM controller interfaces the CRAM to the host computer. Its use is twofold. Since the CRAM PEs do not have control structures, the controller is used to broadcast uniform control and address information to all PEs. It also reduces the bandwidth of issuing instructions from the host computer. Figure 3 shows the architecture of the controller [11]. It consists of a CRAM-Host Interface Unit, an

instruction queue unit, a microprogram sequencer, an address unit, and parameter, command and status registers. The following sections describe the features of the controller that improve the performance of all operations that are initiated from the host computer, i.e. instruction execution, variable initialization/reading, and scalar operations.

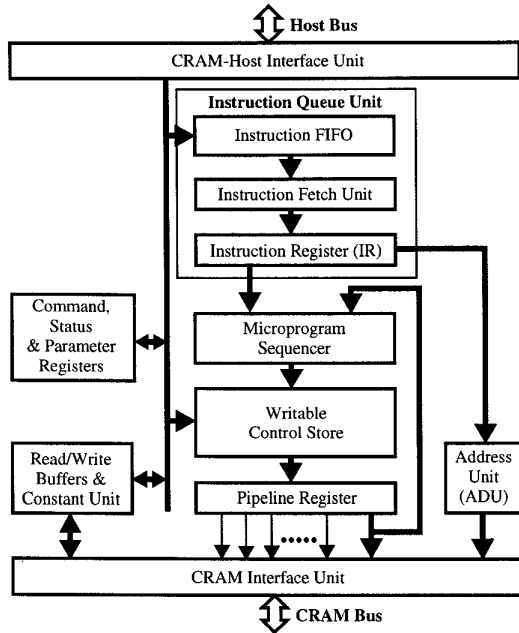


Fig. 3. CRAM Controller Architecture

A. Instruction Queue Unit

All CRAM instructions from the host processor are first loaded into the Instruction Queue Unit (IQU). Once in the IQU, instructions are executed by the CRAM controller completely independent of the host processor.

The FIFO removes the need to synchronize host bus transfers of CRAM instructions. It also allows the host to transfer instructions in advance without waiting for the previous instruction to finish executing, thus improving PE utilization. Otherwise, if instructions were loaded from the external bus directly into the instruction register, there would be breaks in the instruction flow (idle clock cycles between instructions) due to bus transaction delays. For buses such as the PCI, EISA and VME, the instruction FIFO allows the host processor to transfer instructions at increased speed using burst cycles. Burst cycles are typically twice as fast as single cycles and result in less activity and execution time on the host processor when setting up the transfers. Also note that for the CRAM controller, the time to load and setup parameters of the instruction is negligible since almost all instruction parameters are coded into the instruction word. These three features of the IQU provide more than 100% improvement in

the execution time of short sequence instructions, and allow the controller to approach an ideal PE controller (more than 90% PE utilization) with as few as 16 microinstructions per instruction. While short-sequence instructions (30 or less microinstructions) are dismissed as not typical in application-specific systems, they are common in a general-purpose system such as CRAM. Such instructions are essential for operations and extensions, updating of controller registers, and optimization of many operations. Therefore, this performance improvement for short sequences is of vital importance for CRAM. Also, as illustrated in Section III, the IQU makes the controller performance less dependent on the speed of the host bus.

B. Read/Write Buffer and Constant Unit

Like CRAM instructions, data transfers from the host processor to the CRAM chip are buffered through the read/write buffers. In this way, data can be transferred quickly from the host processor to the controller using burst cycles, and then later transferred to CRAM. This feature also allows parallelism. When the CRAM is executing instructions currently in the FIFO, the host can preload data into the write buffer, and then issue an instruction (which will be queued in the FIFO) to transfer this data from the buffer to CRAM. This can reduce the time of initializing or reading CRAM variables by as much as 60% and makes it less dependent on the speed of the host bus (Section III).

More important, the fact that the PE opcodes are multiplexed on the RAM data and address buses makes the design of the constant unit more efficient. Typically, SIMD processors implement operations with scalar constants by multiplexing the microinstructions depending on the bit coming out of the select (constant) register [8]. This approach complicates the design of the microprogram sequencer and increases the width and/or depth of the control store. In addition, the size of the constant is limited by the size of the constant register unless special programming techniques are employed. This limitation is acceptable for fixed-size processing (such as pixel-processing) but would limit the versatility of the CRAM in which the ability to vary the bit-length of operands is one of the major strengths.

Our constant broadcast approach is shown in Figure 4. It takes advantage of the existing write buffer to store the constant(s), and multiplexes the opcode to the CRAM with the opcode to set/reset a PE register depending on the constant bit being currently processed. The 3-bit counter, which selects the constant bit out of the byte coming from the buffer, is coupled to the address register of the buffer. The 4-to-1 multiplexer replaces the original 2-to-1 multiplexer, and adds minimal extra delay and area when compared to the other critical paths in the microprogram sequencer. Apart from the simplicity of this constant broadcast approach, there are other very important advantages. First, the constant size is now

only limited by the size of the write buffer, which is large. Note that the size of the write buffer was made large not because we wanted to accommodate large constants, but rather because we wanted such a large buffer for efficient data transfers. The second advantage is that now a number of constants can be preloaded using the faster burst cycles, or constants can be preloaded in parallel with CRAM executing other instructions in the FIFO. This can reduce the time of executing operate-immediate instructions by more than 40%, especially if a fast CRAM system is interfaced to a slow host bus. It also makes the execution time of these instructions less dependent on the speed of the host bus (Section III).

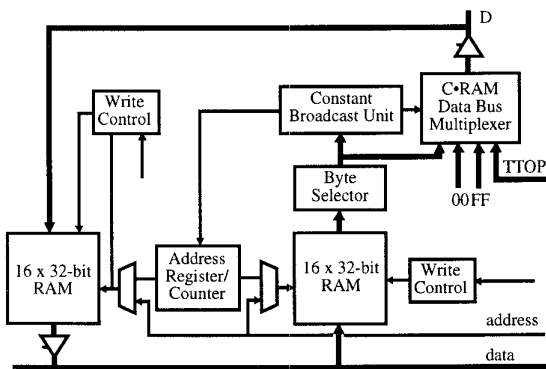


Fig. 4. Read/Write Buffers and Constant Unit

III. PERFORMANCE ANALYSIS

In this section, we investigate the effect of the host bus on the performance of a CRAM system. The parameters used to measure the effect are execution time of a sequence of instructions, PE utilization, time to initialize or read CRAM variables from the host computer, and execution time for operate-immediate instructions. Unless stated otherwise, the CRAM cycle time is 100 ns (based on the CRAM prototype):

A. Execution Time

Figure 5 shows the effect of the host bus on the execution time of a sequence of instructions with varying number of microinstructions per instruction. The effect is only restricted to instructions with a small number of microinstructions. A major finding from this analysis is that because of the performance-enhancement features of the CRAM controller, the transfer characteristics of the host bus has very little effect on the performance of a CRAM system. For example, a PCI CRAM system, with more than 16 times the host bus data transfer rate of the ISA bus, has an execution time that is only about 1.4 times that of an ISA CRAM system. And this is only for a small group of instructions. This means that CRAM systems can be implemented on a variety of platforms, including slow host systems such as ISA-based computers and embedded systems that use slow microcontrollers.

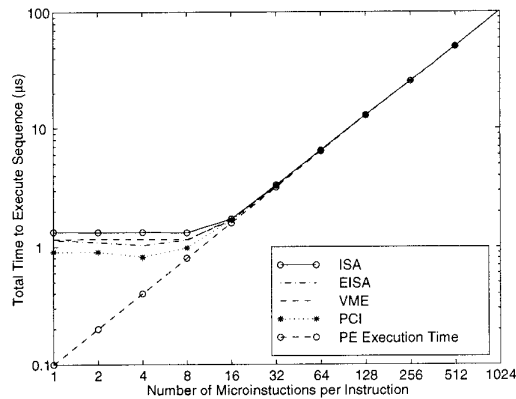


Fig. 5. Effect of Host Bus on Instruction Execution Time

B. PE Utilization

PE utilization (Figure 6) is another way of presenting the results in Section A. This is measured as the percentage of the total execution time for which the PEs are busy. Again, the effect of the host bus is only on short sequences.

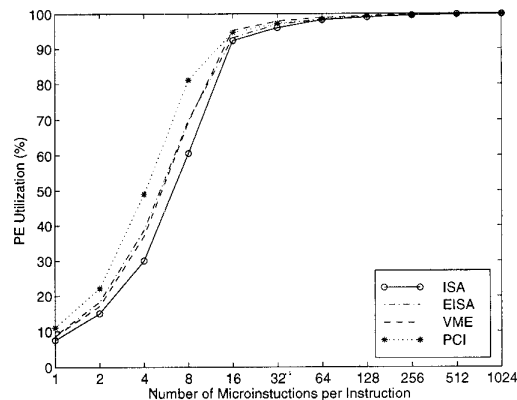


Fig. 6. Effect of Host Bus on PE Utilization

C. Variable Initialization

Figure 7 shows the effect of the host bus on the time to initialize 16 vectors of a 32-bit CRAM variable. The data buffers allow the host computer to transfer data to the controller in parallel with the transfer of data from the controller to CRAM. This not only reduces the variable initialization time, but also reduces the difference in these times between systems interfaced to different host buses. For example, for systems with a simple data register/buffer (buffer size of 4 bytes or less in Figure 7), the difference in variable initialization time between a PCI and an ISA system is as much as 40 μ s. On the other hand, a system using the parallelism possible with data buffers reduces this difference to only about 10 μ s. For slower CRAM systems, this is reduced even further to

about 4 μs (about 90% reduction). These results confirm the observation in Section A that the host bus does not have significant effect on the performance of a CRAM system.

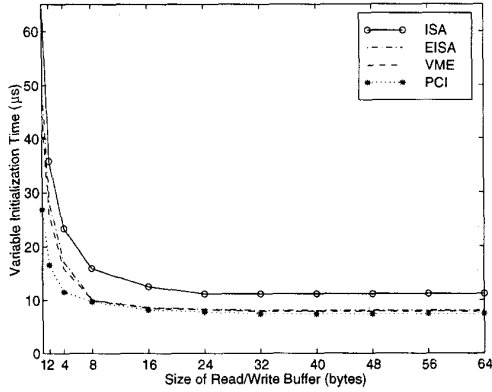


Fig. 7. Effect of Host Bus on Variable Initialization

D. Operate-Immediate Instructions

The performance of operations with immediate values is measured separately from the other instructions because both the instruction and its operand are loaded from the host computer. Figure 8 shows the effect of the host bus on the execution time of adding 64 8-bit integer constants to an 8-bit CRAM variable on a system with a cycle time of 25 ns. For systems that use a simple select (constant) register (in Figure 8, these are systems with buffer size of 4 bytes or less), the effect of the host bus on operate-immediate instructions is very significant. For example, in Figure 8, there is as much as 40 μs difference between the fast PCI bus and the slow ISA bus. On the other hand, a CRAM system that employs the new constant broadcast unit, with a buffer size of 16 bytes or more, almost completely eliminates the effect of the host bus on the execution time of operate-immediate instructions.

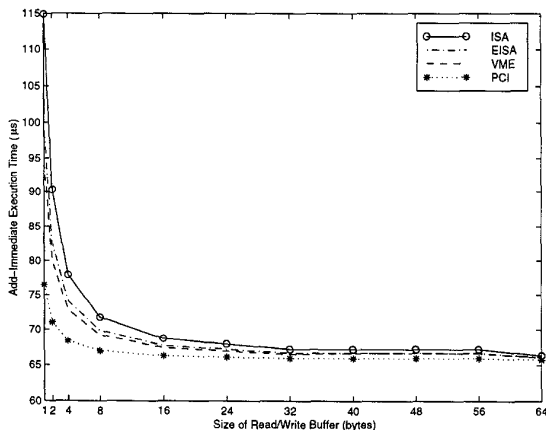


Fig. 8. Effect of Host Bus on Scalar Operations

IV. CONCLUSION

The use of a FIFO in the instruction queue improves PE utilization and reduces the execution time of short sequence instructions. It also improves the overall performance of a CRAM system by making the controller approach an ideal PE controller at a smaller number of microinstructions per sequence. This, coupled with the use of read/write buffers and a novel constant broadcast unit, greatly reduces the effect of the CRAM-host interface on the overall performance of a CRAM system. This allows the implementation of a CRAM system on a variety of platforms, including those with slow external buses.

ACKNOWLEDGMENT

The following have contributed to the CRAM project: R. Mackenzie, C. Cojocaru, T. Le, and P. Lauzon.

REFERENCES

- [1] D. G. Elliott, W. M. Snelgrove, and M. Stumm, "Computational RAM: A Memory-SIMD Hybrid and its Applications to DSP", *IEEE Custom Integrated Circuit Conference*, pp. 30.6.1-4, May, 1992.
- [2] T. M. Le, S. Panchanathan, and W. M. Snelgrove, "Computational-RAM Implementation of Vector Quantization for Image Processing", *Proceedings of the IEEE Workshop on Visual Signal Processing and Communication '94*, pp. 157-162, Sept., 1994.
- [3] T. M. Le, W. M. Snelgrove, and S. Panchanathan, "Computational RAM Implementation of MPEG-2 for Real-Time Encoding", *SPIE Proceedings of Multimedia Hardware Architectures '97*, pp. 182-193, Feb. 1997.
- [4] D. Elliott, M. Snelgrove, C. Cojocaru, and M. Stumm, "A PetaOp/s is Currently Feasible by Computing in RAM", *PetaFLOPS Frontier Workshop*, Feb. 1995.
- [5] Nobuyuki Yamashita, et al, "A 3.84 GIPS Integrated Memory Array Processor with 64 Processing Elements and a 2-Mb SRAM", *IEEE Journal of Solid-State Circuits*, Vol. 29, No. 11, pp 1336-1343, November, 1994.
- [6] Yoshiharu Aimoto, et al, "A 7.68 GIPS 3.84 GB/s 1W Parallel Image-Processing RAM Integrating a 16 Mb DRAM and 128 Processors", *IEEE International Solid-State Circuits Conference*, pp 372-373, February, 1996.
- [7] M. Gokhale, B. Holmes, and K. Iobst, "Processing in Memory: The Terasys Massively Parallel PIM Array", *IEEE Computer*, pp 22-31, April, 1995.
- [8] J. C. Gealow, F. P. Herrmann, L. T. Hsu, and C. G. Sodini, "System Design for Pixel-Parallel Image Processing", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, pp. 32-41, March, 1996.
- [9] Kazutoshi Kobayashi, et al, "A Memory-Based Parallel Processor for Vector Quantization: FMPP-VQ", *IEICE Trans. Electron*, Vol. E80-C, No. 7, pp 970-975, July, 1997.
- [10] R. Torrance, et al, "A 33 GB/s 13.4 Mb Integrated Graphics Accelerator and Frame Buffer", *IEEE International Solid-State Circuits Conference*, pp 340-341, February, 1998.
- [11] Peter M. Nyasulu and W. Martin Snelgrove, "Architecture and Implementation of A Computation RAM Controller", *International Conference on Massively Parallel Computing Systems*, April, 1998.