Command Vector Memory Systems: High Performance at Low Cost

Jesus Corbal   Roger Espasa   Mateo Valero*

Departament d'Arquitectura de Computadors,
Universitat Politècnica de Catalunya–Barcelona, Spain
e-mail: {jcorbal,roger,mateo}@ac.upc.es

Abstract

The focus of this paper is on designing both a low cost and high performance, high bandwidth vector memory system that takes advantage of modern commodity SDRAM memory chips. To successfully extract the full bandwidth from SDRAM parts, we propose a new memory system organization based on sending commands to the memory system as opposed to sending individual addresses. A command specifies, in a few bytes, a request for multiple independent memory words. A command is similar to a burst found in DRAM memories, but does not require the memory words to be consecutive. The command is sent to all sections of the memory array simultaneously, thus not requiring a crossbar in the proper sense. Our simulations show that this command based memory system can improve performance over a traditional SDRAM-based memory system by factors that range between 1.15 up to 1.54. Moreover, in many cases, the command memory system outperforms even the best SRAM memory system under consideration. Overall the command based memory system achieves similar or better results than a 10ns SRAM memory system (a) using fewer banks and (b) using memory devices that are between 15 to 60 times cheaper.

1 Introduction

The memory system architecture is a critical factor that determines overall performance of supercomputers. This is specially true in the case of parallel vector processors, since vector cpu's typically demand very high bandwidths at relatively low latency from the memory system.

Traditional vector supercomputers use multiple sections and multiple banks per section to achieve these very high bandwidths. Moreover, to achieve low latencies, high performance SRAM memory banks are typically used. Figure 1 shows the block diagram of a typical multi-ported parallel vector processor. The entire system is composed of P vector processors with T memory ports each. In order to guarantee the required bandwidth two conditions must be met. First, the memory array needs S sections (or memory busses) where $S \geq P \times T$. Second, the number of banks per section (N in the figure), has to be at least equal to the latency of

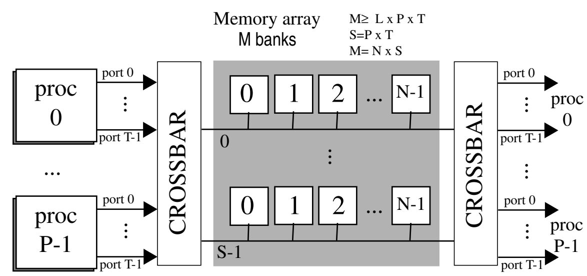

Figure 1: Typical multiport memory system of a multiprocessor vector machine.

each individual memory bank as measured in processor cycles (denoted L in the figure).

These types of memory systems have a very high cost. This is due to the combination of several factors. First, SRAM chips, used to achieve low latency, are very expensive and offer a modest capacity. Therefore, a large number of chips is needed to build a large memory system, typical of supercomputers. Second, the interleaving required to provide high bandwidth requires many independent sections and large crossbars, which consume large numbers of chips and board interconnects.

Designing less expensive high-performance memory systems will become a primary objective in the next generation of vector machines. We believe that central to that goal will be the use of alternative memory devices with better performance/cost ratios. In particular, the use of commodity DRAM parts instead of expensive SRAM ones will probably be mandatory.

Except for large problems that benefit more from fitting in memory than from using fast memories, conventional DRAM is still not a good alternative for vector memory systems. This is due to two main reasons. First, DRAM memories offer lower bandwidth than SRAM. About 4 to 8 times more banks are required in a DRAM-based system to match the peak bandwidth of a SRAM-based system. Therefore, the benefits from the lower cost per bit of DRAM memories might be offset by the higher interconnection costs and additional control logic. Lastly, DRAM memories have a much longer latency than SRAM memories. Even though vector machines are known for their ability to tolerate long memory latencies, it has been shown in [3] that typical vector codes are actually very sensitive to increases in memory latency.

The new advances in DRAM architectures may change this situation. We believe that modern SDRAM

*This work was supported by the Ministry of Education of Spain under contract CICYT TIC-0429/95 and by the CEPBA.

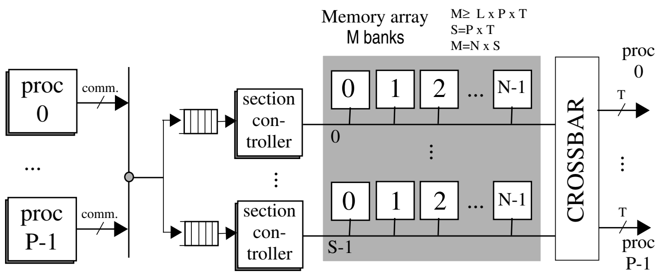

Figure 2: The command memory system design.

memories are a suitable alternative for a cost-effective high performance vector memory system. They deliver higher bandwidth per bank than SRAM memories and, therefore, a memory system based on SDRAM chips can be built using fewer overall banks. Fewer banks implies a lower cost in interconnections and less additional glue logic. Not only that, each SDRAM chip is, by itself, much cheaper than the equivalent SRAM one. The combination of both factors can yield a vector memory system that is several orders of magnitude cheaper than traditional vector memory systems.

The focus of this paper is on designing both a low cost and high performance, high bandwidth vector memory system that takes advantage of modern commodity SDRAM memory chips.

To successfully extract the full bandwidth from SDRAM parts, we propose a new memory system organization based on sending commands to the memory system as opposed to sending individual addresses. A command specifies in a few bytes a request for multiple independent memory words (similar to a burst found in DRAM memories, but does not require the memory words to be consecutive). The command is sent to all sections of the memory array simultaneously (thus not requiring a crossbar in the proper sense). At the head of each section, a section controller examines the command and analytically determines which memory words of the command it can service. Then, the section controller generates the appropriate sequence of addresses and sends them to the individual banks. The individual banks perform the final memory access and send the requested word to a data crossbar that routes it to the corresponding processor.

Figure 2 illustrates the proposed design. As it can be seen, in contrast with the traditional architecture, there are no memory ports. Instead, we find a single command port where all vector requests are sent sequentially. Having only one command bus does not imply a loss of address bandwidth, since a command describes a large set of addresses using 2 words.

The commands from different processors compete for a common bus. In sharp contrast with a traditional design, there is no address crossbar, just a multiplexed bus. Once a command seizes the common bus, it is broadcasted to *all* section controllers simultaneously. The commands are queued and serviced when possible by the section controller. Note that each section controller processes commands in parallel and, thus, is able to generate vector addresses independently from other sections.

This paper will show that there are several advantages to using a command memory system over a traditional design. First, and most important, using commands instead of individual (and independent) memory addresses matches very well the access schemes of modern SDRAM parts (which favor multiple accesses to the same row). Second, a command is effectively compressing multiple address requests into a small packet. Therefore, multiple commands can be potentially sent to the memory system in much less time than it was previously required for a single vector stream. The net effect is that commands reach the section controllers many cycles in advance, thus providing a latency tolerance mechanism. Third, having multiple commands queued at a section controller can be used to improve overall throughput by intelligently multiplexing the execution of two or more commands.

2 Background on memory devices

DRAM and SRAM are the two basic memory archetypes. DRAM memories are typically bigger and cheaper than SRAM ones, whereas SRAM memories are faster and have more bandwidth due to their inherently faster cycle time.

Accessing a SRAM device is relatively straightforward. The memory controller presents a full address to the SRAM and after a fixed amount of time the data appears at the output pins. By contrast, the DRAM access scheme is substantially more complex. In order to reduce pin count, DRAM memories decouple the data element access in two different stages: `row access`, where a row of data (256-1024 elements typically) is loaded from the DRAM core and hold in a row of sense amplifiers, and a `column access`, where the selected data is accessed from the row of sense amplifiers. This two stages are usually referred as RAS (Row Address Strobe) access, and CAS (Column Access Strobe) access. Furthermore, before accessing a new row, the active row must be regenerated into the DRAM core (due to the destructive read-out nature of dynamic accesses) by initiating a `precharge operation`. Due to all these limitations, plus the high capacitance of DRAM cells, DRAM memories suffer from the lack of enough speed and bandwidth.

This situation has been changing recently. DRAM vendors have designed new memory architectures which offer dramatic improvements in performance. Examples of such architectures are *Cached DRAM* (CDRAM) [4], *Rambus DRAM* (RDRAM) [5] and Synchronous DRAM (SDRAM) [6].

SDRAM operation

A SDRAM resembles a conventional DRAM. It is dynamic, and must be periodically refreshed. However, SDRAM uses pipelining to improve throughput. Furthermore, the data from the selected row is held on the sense amplifiers while multiple column addresses are selected from it. Therefore, if we access a data element located in the current active row (what we will call `row hit`) we no longer need to repeat the row access, reducing dramatically the access delay time. The SDRAM

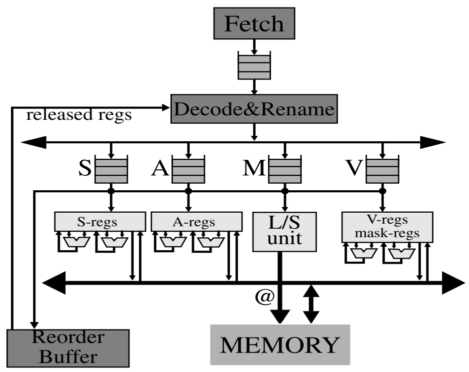

Figure 3: The Out-of-Order vector architecture studied in this paper.

synchronous interface is optimized so that we are able to make a column access every cycle.

When an access is initiated on a SDRAM memory bank, we have to execute the following sequence of operations:

1. Control whether a row hit has been produced or not
2. If we have a row hit, we can initiate the column access to get the desired data.
3. If we have a row miss, we have to initiate a precharge operation (in order to regenerate the active row) and then, a new row access is required. Once we have loaded the new row, we can initiate the column access.

3 Experimental setup

In this section we present the tools and benchmarks used to (1) evaluate our proposed command-based memory system, and (2) compare it to more traditional vector memory designs.

3.1 The Vector Processor

We use as our vector cpu an out-of-order version of a Convex C3400 [7]. This out-of-order design was introduced in [3] and uses register renaming in a similar fashion as a R10000 processor [8] to achieve out-of-order execution of all types of vector and scalar instructions (see figure 3).

Instructions flow in-order through the Fetch and Decode/Rename stages and then go to one of the four queues present in the architecture based on instruction type. At the rename stage, a mapping table translates each virtual register into a physical register. There are 4 independent mapping tables, one for each type of register: *address* (integer), *scalar* (floating point), *vector* and *mask* registers. When instructions are accepted into the decode stage, a slot in the reorder buffer is also allocated.

The A, S and V queues monitor the ready status of all instructions held in the queue and as soon as one instruction is ready, it is sent to the appropriate functional unit for execution. All instruction queues can hold up to 16 instructions. Both scalar register files (integer and floating point) have 64 physicals registers each. The mask register file has 8 physical registers. The fetch stage, the decode stage and all four queues only process a maximum of 1 instruction per cycle while committing of instructions proceeds at a rate of up to 4 instructions per cycle.

The vector unit has two functional units connected to eight logical vector registers (backed up by 16 physical vector registers). The former unit is a general purpose arithmetic unit capable of executing all vector instructions. The latter unit is a restricted functional unit that executes all vector instructions *except* multiplication, division and square root. Both functional units are fully pipelined and each vector register holds 128 eight-byte register.

3.2 Why an Out-of-Order architecture ?

As we have already seen, SDRAM memory chips have longer latencies than the typical high-performance SRAM memory chips used in vector memory systems. Typically, access time latency increases a factor of 3 to 6.

Traditionally, vector cpu's have been considered very good for tolerating long memory latencies. However, the results presented in [3] show that for very long latencies (between 50 and 100 cycles) this might no longer be true. In [3] , simulations of a traditional in-order vector machine and an out-of-order vector architecture were carried out under varying memory latency conditions. From the results of this work, we can easily conclude that the *out-of-order* vector architecture has a good latency tolerance whereas the conventional architecture suffers severe degradations in performance when increasing memory latency.

These results lead us to believe that we require the benefits of an out-of-order vector architecture to be able to explore the design tradeoffs involved in command memory systems.

3.3 Memory subsystem assumptions

Although we believe command memory systems will be a very good match for vector multiprocessors, this is the first study on the design and characteristics of the command-based paradigm. Therefore, to better understand the tradeoffs involved in command memory systems and to reduce the design space to be explored, we have chosen to study only a single processor with a single memory port.

Since we want to evaluate memory systems as close to reality as possible, we will use real specifications for both SDRAM and SRAM parts. We have selected two different SRAM chips for our simulations: a high-end BiCMOS SRAM with a cycle time of 10 ns, and a middle/high-end CMOS SRAM with a cycle time of 20 ns [9]. For our SDRAM simulations we have designed a detailed model of a real SDRAM chip, the Fujitsu MB81117822A [10]. The Fujitsu MB8 is a 2M x 8bits, 125 Mhz SDRAM, with two interleaved banks. Each bank has its own DRAM core and its own row of sense amplifiers, but they share the same common data and address bus.

3.4 Simulation Tools and Benchmarks

To gather the performance results on the architecture described we have taken a trace-driven simulation approach. Using a pixie-like tool called Dixie [11] we have extracted instruction and memory traces from Convex C3400 binaries. Dixie is able to produce a trace of basic blocks executed as well as a trace of the values contained in the *vector length* (`vl`) register and *vector stride* (`vs`) register. This allows us to accurately simulate on a cycle-by-cycle basis the behaviour of the two architectures just described. See [3] for details of the simulation procedure.

We have vectorized all programs from the *Perfect Club* and *SPECfp92* suites and we have processed them using Dixie. Of all these programs, we have selected for our experiments the ten most vectorizable ones.

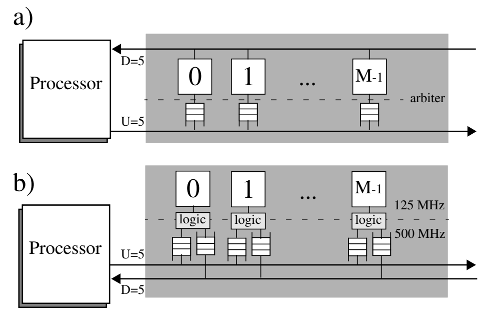

Figure 4: Model for simulations: a) SRAM model; b) SDRAM model

4 Performance of Conventional Memory Systems

We start by analyzing the performance of conventional memory systems for our benchmarks on the out-of-order vector architecture. We have considered the four following memory systems.

1. An ideal memory system with no collisions and 1 cycle of fixed latency
2. A 10ns-SRAM 1-section conventional memory system.
3. A 20ns-SRAM 1-section conventional memory system.
4. A 125Mhz-SDRAM 1-section conventional memory system, with row hit/miss control logic.

The ideal memory system modeled is an optimal memory system which delivers a maximum bandwidth of one 64-bit word per processor cycle. A vector access to memory of Vl elements is served in $X + Vl$ cycles, where X is always fixed to a single cycle.

The two SRAM memory systems are modeled following figure 4(a). The processor sends individual addresses to the memory system at a rate of one address per processor cycle. Although we only model a single section, we force addresses to spend a few cycles ($U = 5$) crossing the upward interconnection network before reaching the section. Once an address arrives at the section it will be sent to the appropriate bank. If the bank is busy, the address will be queued at a FIFO in front of the bank. Each bank FIFO can hold an unlimited number of addresses. Each cycle, an arbiter selects one single bank to initiate the access within all those banks which have an available address in the head of its FIFO. The selection is prioritized based on the "age" of each address item; that is, address items corresponding to vector requests issued earlier by the processor will be given higher priority. The bank access, after either 10ns or 20ns depending on the SRAM chosen, will produce a data item that will be returned, after $D = 5$ cycles of downward latency, to the processor. Since we have fixed latency and we assume that only one single access per cycle is allowed, we will never have collisions at the downward network.

The SDRAM memory system is modeled very similarly to the SRAM systems (see figure 4(b)). However, the dynamic nature of the memory devices and their different clock cycle time force a few differences. First, since the SDRAM chips operate at a different frequency than the processor, we need extra logic to perform the frequency conversion. We assume a logic which receives address requests at the processor frequency and is able to generate RAS and CAS accesses (plus precharge operations) at the memory frequency. Second, it is assumed that this logic includes a hit/miss detection logic able to detect whether we have the needed row already loaded. Additionally, since SDRAM latency may vary depending of whether we have a row hit or not and since several SDRAM accesses can be initiated every cycle, we need both output and input FIFO's. An arbiter selects from all the output FIFO's heads one data item to be returned to the processor. Again, the selection is prioritized based on the "age" of each data item.

4.1 Performance Results

Using the tools seen in section 3.4, we have simulated the four memory systems just described. The results are shown in Figure 5. For each program, we present performance results for different number of banks (X-axis). Performance is plotted relative to the ideal memory system (1 cycle memory latency, no collisions). That is, a value of 1.0 is the performance of each program under the ideal memory system.

The results can be clearly divided in two different groups of programs:

A first group of programs, **`swm256`**, **`hydro2d`**, **`arc2d`**, **`su2cor`**, **`tomcatv`** and **`bdna`**, show a very small difference between the SRAM and SDRAM memory systems. Moreover, all these programs show a performance quite close to the ideal memory system. This is not surprising, given what we saw in [3] . These programs happen to be highly latency tolerant and, therefore, the extra latency of the SDRAM devices does not affect their final performance. Moreover, when the number of banks is low (say, the under-matched cases) the SDRAM system shows a better performance than the 20ns-SRAM one. This can be explained because the SRAM system is bandwidth-limited, whereas the SDRAM, given a high enough row hit ratio, is not.

The second group of programs, **`dyfesm`**, **`flo52`** and **`trfd`**, are quite far from the ideal memory system performance. These happen to be the less latency tolerant

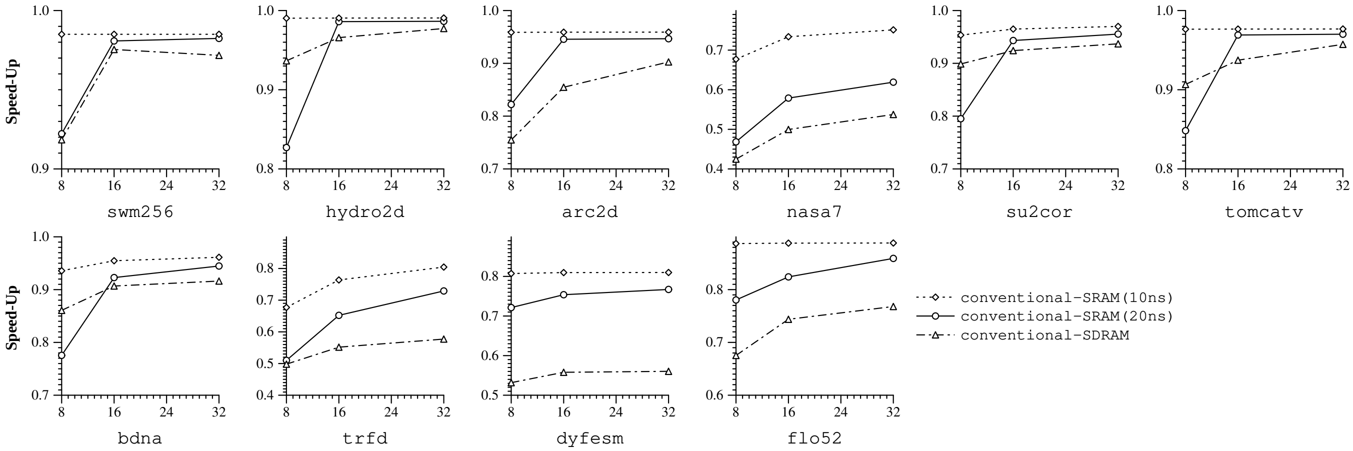

Figure 5: Performance of conventional memory systems for different numbers of memory banks (X-axis). Performance is relative to the ideal memory system (1 cycle memory latency, no collisions).

programs and the higher bandwidth of SDRAM cannot compensate its higher latency. This also occurs even when the number of banks is low.

4.2 Limitations of conventional access

The results presented in the previous section are somewhat mixed. On one hand, the SDRAM system achieves a performance close to the 20ns SRAM system, sometimes even close to the 10ns SRAM one. Therefore, for a certain class of programs, the SDRAM system represents a very good cost-performance tradeoff. On the other hand, despite its theoretical higher peak bandwidth, in many cases the SDRAM system falls far behind the SRAM system.

The origin of the problem can be found in the access scheme of the SDRAM memories we are using. In the conventional scheme, each address sent to the SDRAM banks is treated independently from previous and/or following addresses. Therefore, in the most common case, we have to follow the long sequence of operations described in section 2 for each individual memory address. This causes the SDRAM access scheme to be inefficient since we cannot hide neither the *precharge operation time* nor the *row access time*.

Ideally, one would desire to advance all the expensive operations of a SDRAM access (row hit/miss detection, precharge and row access) as much as possible so that their associated latencies could be effectively hidden.

4.3 Advanced access schemes

In fact, SDRAM vendors recommend another access scheme to take full advantage from the high bandwidth of SDRAM chips. All SDRAM vendors integrate two or more banks in each SDRAM chip to maximize the effective bandwidth. By intelligently interleaving accesses to each bank, it is possible to hide the precharge operation time and the row access time. The mechanism used is to request a burst of data items to one bank while the other bank is being precharged and/or is loading a new row. A

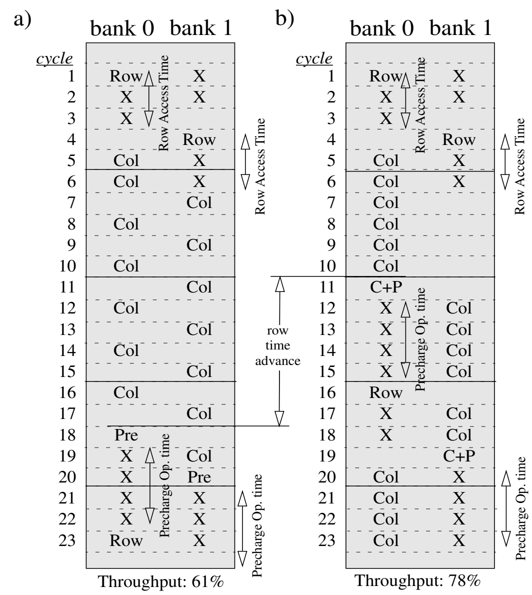

Figure 6: Comparison between a) a conventional SDRAM access scheme and b) an efficient SDRAM access scheme. Row, *Row access*; Col, *Column Access*; C+P, *Column Access* with *Precharge Operation*; Pre, *Precharge Operation*; X, Forced Delay.

conventional access schemes tends to distribute consecutive accesses among as many banks as possible, in order to reduce the number of memory collisions. By contrast, the SDRAM access scheme must take advantage of burst capabilities to extract consecutively as many data items as possible from the same row. This is due to the fact that a single SDRAM chip with two banks inside can almost provide one data item per memory cycle. Therefore, the addition of more banks per section does not increase substantially the global section data throughput.

Figure 6 shows an example in order to compare the conventional SDRAM access scheme and the advanced access scheme proposed by the vendors. With the advanced access scheme precharge and row access operations are successfully hidden by accessing burst of data to the other bank. In other words, we are anticipating the row access several cycles before it is really needed. Note that the throughput (i.e. the rate of data accessed per cycle), is remarkably higher in the advanced scheme. Note, however, that is no longer possible to generate addresses in-order (i.e. following the natural order of the different elements of a vector).

5 Command Memory Systems

In this section we present our proposed vector memory system based on the use of commands. In this paper, the term *command* refers to a vector memory access, whether a load or a store. A command is characterized by a 3-tuple ($\langle A_0, Vl, Vs\rangle$) that contains the initial memory address, the vector length and the vector stride of the memory access.

The main goal of the proposal is to exploit the advanced access scheme of SDRAM memories described in the previous section. Therefore, our proposal is based around one central idea: instead of sending individual addresses to the memory system we will send requests for multiple data items (commands). These requests are semantically richer than individual addresses because they describe multiple related memory accesses in a compact way. Their semantics will allow us to efficiently schedule memory accesses to the SDRAM banks.

The basis of our command memory system is to translate a vector request (a command) into a set of bursts, rather than chopping a vector request into its individual addresses. A burst consists of a row address followed by a certain number of column accesses. The concept of burst is limited to a single row, but can be easily extended to deal with an entire bank, leading to what we will call a *subcommand*. Given a certain vector memory access, a subcommand describes all the data items specified in the vector access that are contained in a single memory bank.

At the core of our proposal is the question of how to generate from a certain vector memory access (command) multiple subcommands to be sent (in parallel and simultaneously) to all memory banks. Figure 7 shows a simplified scheme of the proposed command memory system, assuming the number of banks per section is 2.

The command memory system is able to translate a vector request into M subcommands, where M is the number of SDRAM banks in the memory array. In order to do so, the command issued by the vector cpu is broadcasted to all sections of the memory system. At each section, we have a section controller. The section controller contains a "command processing unit" per each memory bank of the section. The original command sent by the cpu is processed simultaneously by all command processing units, and is transformed into one subcommand per bank. Once the subcommands have been generated, they are queued at a per-bank FIFO, where they await being serviced.

At each section controller, a "RAS/CAS generator" monitors the head of all the bank FIFOs of the section.

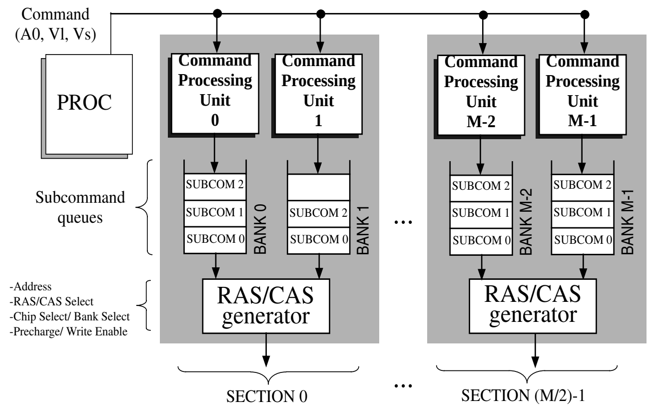

Figure 7: Command memory system block diagram (only includes the address generation paths).

The address generator faithfully reproduces the access scheme shown in figure 6(b) deciding at each cycle what FIFO should be serviced.

Overall, the command memory system has the following advantages:

a) Depending of the characteristics of the vector stride, a vector access can be unable to generate one subcommand for every bank. In conventional memory systems, this often produces memory collisions. In command memory systems, this queue gaps can be filled with subcommands from following vector accesses. That is, the RAS/CAS generator can effectively multiplex several subcommands in order to take advantage of the whole available bandwidth.

*b)*Sending a command takes very few cycles, much less than a traditional address generation process. This greatly increases the incoming request rate to the section controllers and, in turn, favors the multiplexing capabilities of our system.

c) Out-of-order access to the memory banks avoids producing inter-stream collisions, that is, we can avoid producing bank busy conflicts between the last accesses of one vector request and the first accesses of the following vector request.

5.1 Subcommand Generation

A command is a 3-tuple containing all the information of a vector request: the initial memory address (A_0), the vector length (Vl) and the vector stride (Vs). Each command processing unit is responsible of computing which subset of the data specified in the command is contained in its bank. This subset of data will be described using a subcommand and queued into the bank FIFO. Note that it might happen that the subset is empty. In this case, no subcommand is generated. A subcommand is a 4-tuple that contains:

1. RAS_0, initial row address.
2. CAS_0, initial column address.
3. Δk, burst stride.
4. DAT, number of data items contained in the bank.

We have developed an algorithm that takes as input a command and a bank index and generates the appropriate subcommand. This algorithm develops the ideas contained in [12] assuming low-order interleaving. A full description of this algorithm is beyond the scope of this paper and can be found in [13].

From the point of view of the results presented in this paper, the most important characteristic of the algorithm is its computational cost, which depends on the vector stride. The computational cost of the algorithm depends strongly on the vector stride. Vector accesses with power-of-two strides (90% of the cases, since they include stride 1) can be easily processed with a computational cost of only 2 memory cycles. In the other cases, this computational cost increases to 15 memory cycles. Despite this extra cycles, the command memory system architecture has very good latency tolerance properties

5.2 RAS/CAS Generation

Each section controller shown in figure 7 contains a block of logic able to generate the row addresses, column addresses and other control signals (write enable, auto-precharge, etc.) needed to control the operation of all the SDRAM chips in that particular section. This logic, which we call the "RAS/CAS generator", has to monitor the status of all the FIFO queues associated to all the banks contained in the section.

The RAS/CAS generator contains registers to control the state of each bank (inactive, row accessing, column accessing or precharging) and the state of each subcommand being served (number of data items remaining in the bank). Also, row miss detection logic is required in order to detect column overflows during a subcommand service. The RAS/CAS generator follows the following rules to ensure maximum data throughput in the section:

1. Only one row address or column address can be generated every cycle.
2. When the column address to be generated refers to the last data item of a subcommand, then a CAS access plus precharge operation is generated (instead of a simple CAS access). An example of the application of this rule can be seen in figure 6(b), cycle 11.
3. Whenever a bank is in inactive state and a subcommand for that bank is available, a row access must be initiated, even if another bank is in column accessing state. This means that a sequence of CAS addresses being generated will be momentarily interrupted to allow the RAS address for this inactive bank to proceed as soon as possible (see figure 6(b), cycle 16).

5.3 Managing Stores

So far we have only discussed how the processor sends requests to the section controllers and how each section controller intelligently responds to these requests by returning the required data. However, stores pose a different problem. A vector store consists of two "information streams": a stream of addresses, which can be nicely compressed into a command, and a stream of data items (one data item per address). Unfortunately, the data items must be sent one by one from the cpu to the memory banks.

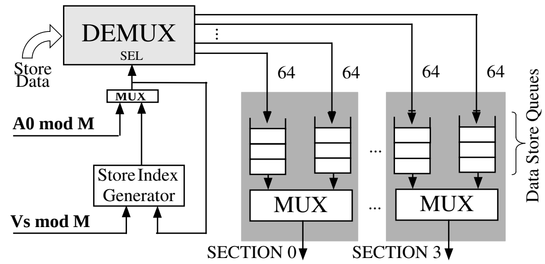

Figure 8: Proposed store managing system scheme.

In our command memory system, stores are handled as follows. First, a command describing all the addresses of the store operation is sent to the memory system. Then, the memory port is used to send, one by one, each data item that must be stored into memory. Two problems must be solved: first, store data is sent out of the cpu in order, while the addresses will be generated out-of-order at the section controllers (the RAS/CAS generators work in parallel and *independently*). Second, although the cpu sends the store data right away after the command, it is not possible to guarantee that the store (sub)commands will be serviced immediately. Therefore, some form of buffering for store data must be provided at the section controllers.

The solution to both problems is based on having store data buffers at the section controllers and have the RAS/CAS generators request data from these buffers to be sent to the SDRAM banks. Figure 8 shows this scheme. There is one FIFO queue per bank to hold the corresponding store data. Data flows out of the cpu in-order and enters each bank FIFO also in order. This guarantees that the first data element of each FIFO corresponds precisely to the first column access generated by the corresponding subcommand (i.e. ordering of data within a queue is correct).

When a RAS/CAS generator accesses a column in a store operation, it will force the head element of the bank store FIFO to be sent to the section data bus. In could happen that, by the time the RAS/CAS generator is ready to start a write (store) operation, there is no available data at the corresponding FIFO queue. In this case, the RAS/CAS generator aborts the access and attempts servicing some other command. At some later time, the generator will retry the store access.

The only remaining problem is how the cpu routes each individual data item to its proper FIFO queue (that is, to is corresponding memory bank). The cpu uses the initial address and the vector stride to find the bank FIFO where the store data must be buffered.

5.4 Full architecture

We now describe a full memory system architecture using the concept of commands. As we already mentioned in previous sections, our performance results will focus on a single processor with a single memory port. Thus the memory system should be able to provide one word (64 bits) per processor cycle. Our goal is to achieve this performance while minimizing overall cost, thus preferring as many pieces of our design working at the lowest possible frequency. In particular, we would like to clock the whole memory system at the frequency of our SDRAM's (125Mhz).

Figure 9 shows the final architecture used in our simulations. Since the chosen processor working frequency (500Mhz) exceeds the SDRAM working frequency, one section with two banks is unable to match the required bandwidth. To overcome this 4:1 frequency ratio, we need 4 independent sections working in parallel delivering 4 words per memory cycle to be able to sustain a bandwidth of 1 word per processor cycle.

Stores can become a severe performance problems. Despite stores represent around a 30% of all memory operations, our initial simulations indicated that it is important to send the store data to the bank FIFOs as fast as possible. Therefore, we decided to have a 128-bits output bus from the cpu to the memory system. This bus allows sending two 8-byte words of store data per processor cycle. In those cases where the processor knows FIFO collisions will occur (two consecutive data items map to the same FIFO queue), the output rate goes down to one data item per cycle. Note that this output bus is not larger than what can be found in a typical microprocessor. In a conventional design an address bus and two data busses (one for loads and one for stores) are needed. In our design, we have one load data bus and the 128-bits output bus. Overall, the pin requirements are very similar. Nevertheless, it can be seen in [13] that using a more conventional 64-bit wide bus only causes a small reduction of performance.

Note that the store data paths are decoupled into two parts. The first part is composed of the index generator and two multiplexers which route two store data every processor cycle. The second part, the store FIFOs, are distributed among all section controllers so that each section controller has all it's corresponding bank FIFOs integrated in it. There is an interesting frequency mismatch problem at the store buffers. Store data comes out of the cpu at 500Mhz, while the section controllers are assumed to work at 125Mhz. We use a serial-to-parallel frequency converter, similar to what can be found in the Rambus memory chips [4].

6 Performance Results

Using the same tools already seen in section 4.1, we have modeled the 4-section command memory system described above. The results are shown in Figure 10 and are compared to the results already obtained for the conventional memory system designs.

As we did in section 4.1 performance results are relative to the performance of the ideal memory system and for different number of memory banks. For the command memory system we only simulate two cases: 8 banks and 16 banks (that is, one SDRAM chip per section and two SDRAM chips per section).

Two general trends can be observed. First, in all programs but one (`flo52`) the command memory system improves performance over the conventional SDRAM memory system. Thus, for a similar cost, command memory systems show a clear advantage. Second, the command memory system requires much fewer banks to achieve similar or even better performance than the conventional memory systems. In other words, while the conventional memory system shows improvements when we increase the number of banks, the command memory system performance is almost flat for most programs. Third, in five out of ten programs the command memory system even outperforms the fastest and much more expensive SRAM system (the 10ns one).

If we go into a program by program analysis, we see again that results are strongly dependent on the latency tolerance characteristics of each benchmark. The programs can be broadly classified in three groups: (1) `hydro2d`, `arc2d`, `su2cor`, `tomcatv`, and `bdna`, (2) `swm256` and `nasa7` and (3) `trfd`, `dyfesm` and `flo52`.

In the first group, the simulations show *command memory systems* clearly outperform all conventional memory systems. Even the 10ns-SRAM systems with 32 banks is outperformed by a command system which has only 8 or 16 banks. What is more, in some cases, the command memory system outperforms even the ideal memory system (1 cycle latency, no collisions). Although intuitively this might seem impossible, the reason is that the command memory system sends stores to the memory system twice as fast as the ideal system due to the 128-bits bus. Sending stores faster reduces the memory port reservation time and frees physical registers sooner. Having more physical available registers helps increasing the processing rate and freeing the memory port sooner allows sending more commands in advance. Both effects combine to reduce the critical path of the program with respect to the ideal memory system case.

Program `nasa7` shows a very large improvement, outperforming 20ns-SRAM memory system. Surprisingly, the conventional SDRAM did not perform very well. This is explained by the multiplexing capabilities of the command memory system. Program `nasa7` has very large strides, thus having a lot of memory collisions, since data does not distribute evenly among all banks. Our RAS/CAS generators, are able to multiplex several independent commands with large strides and maximize the usage of the section bandwidth.

In the last group, say, those benchmarks with poor properties of latency tolerance, only moderate results are achieved. Only conventional SDRAM memory systems are slightly outperformed by our command memory system. What is worse, in one of the benchmarks (`flo52`), our proposed memory system is unable to improve over the conventional SDRAM scheme. Several reasons explain these results. First, the low rate of incoming command arrivals has the negative effect of exposing all the command processing latency. Combined with this, the short vector lengths of these programs imply that each bank contains very few data items. Therefore, the costs of the row access and precharge operation times can not be amortized and cause the SDRAM output throughput to fall down to only a 30%-50% of the theoretical maximum throughput.

7 Related Work

There have been several studies dealing with a possible migration towards DRAM technology in vector memory systems [1, 2]. These studies have considered differ-

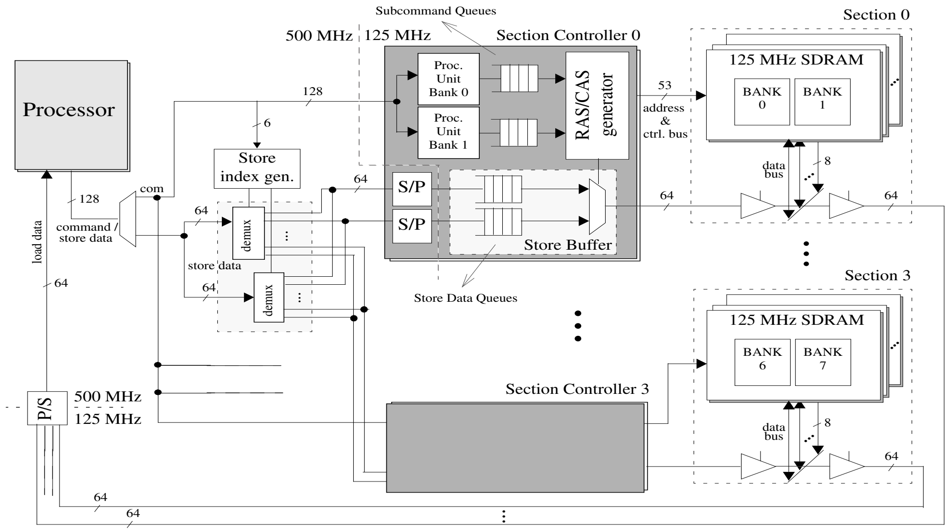

Figure 9: Proposed final architecture.

ent alternatives to overcome the two problems of dynamic memories mentioned in the introduction.

Kontothanassis et. al. in [2] focus on the bandwidth problem as the main reason for the loss of performance in DRAM vector memory systems. Therefore, they looked at the performance impact of caches in a vector processors with DRAM memory system, showing that SRAM memory systems provided the best overall performance given a small amount of banks. Even with cache sizes of 1 Mbyte, DRAM vector memory systems suffered losses of performance of more than 50% when compared to the SRAM memory system (at equal number of banks).

Hsu, Smith et. al. in [1] used CDRAMs in both conventional mono-processor and multi-processor vector systems. Their work was primary targeted at achieving a memory system able to deliver cost-effective bandwidth in vector supercomputers, rather than at achieving high performance. A CDRAM (Cached DRAM) can be seen as a SDRAM with multiple lines instead of one, working as an integrated cache on-chip. Additionally, they simulated several interleaving strategies in order to find the optimal distribution of the data across the memory array (trading-off data locality and 'hot bank' elimination).

These proposals essentially try to overcome the lower bandwidth of DRAM chips by using some form of caching (either with a proper cache in [2] or with a distributed cache as in [1]). In our proposal, there is also some form of caching, since we may consider the SDRAM active line as part of a distributed cache. Therefore, our 8-bank experiments could be seen as using a 256 Kbytes cache. However, our proposal differs from these previous ones in one fundamental aspect: we introduce a new access scheme which forces accessing first all vector data contained in a row (cache line) before starting to service a new vector access. Since there is no interleaving of accesses within heterogeneous vector streams, we avoid unnecessary row misses, improving overall efficiency. Our proposal also includes a second important feature: we are able to multiplex vector accesses with unbalanced strides so that bandwidth is always maximized.

8 Summary

This paper has looked at the problem of designing a low-cost high-performance vector memory system. To achieve low cost, solutions based on SRAM chips must be ruled out and, instead, dynamic memory must be considered. However, current DRAM variants cannot match the good bandwidth and latency parameters offered by SRAM packages. Therefore, achieving both good performance and low cost might seem unlikely.

Our simulations of conventional memory system designs based on both SRAM and newer SDRAM chips have shown that the performance of an SDRAM memory system can be up to a 30% worse than the SRAM one. An analysis of the SDRAM memory system shows that this degradation in performance is mostly due to the inadequate memory access pattern generated by a typical vector cpu. Instead of using bursts of addresses to the same memory bank a vector cpu using low-order interleaving tends to scatter addresses across many different banks.

We have presented a new memory system design based around one central idea: we send full vector requests (commands) to the memory system as opposed to sending individual addresses. A command specifies in a few bytes a request for multiple independent memory words. By using intelligent memory controllers, we can efficiently schedule all the individual requests specified in a command so that bandwidth is maximized and all the expensive SDRAM operations (row open, precharge, etc.) are successfully hidden. The advantages of such scheme are many-fold:

First, sending a command to the memory system takes very few cycles, much less than a traditional address generation process. Second, the commands are *broadcasted* to *all* memory sections using a simple bus structure. That is, there is no address crossbar in our design. At each section, the controller computes which part of the command can be serviced from the banks in that particular section. Third, multiple commands with very

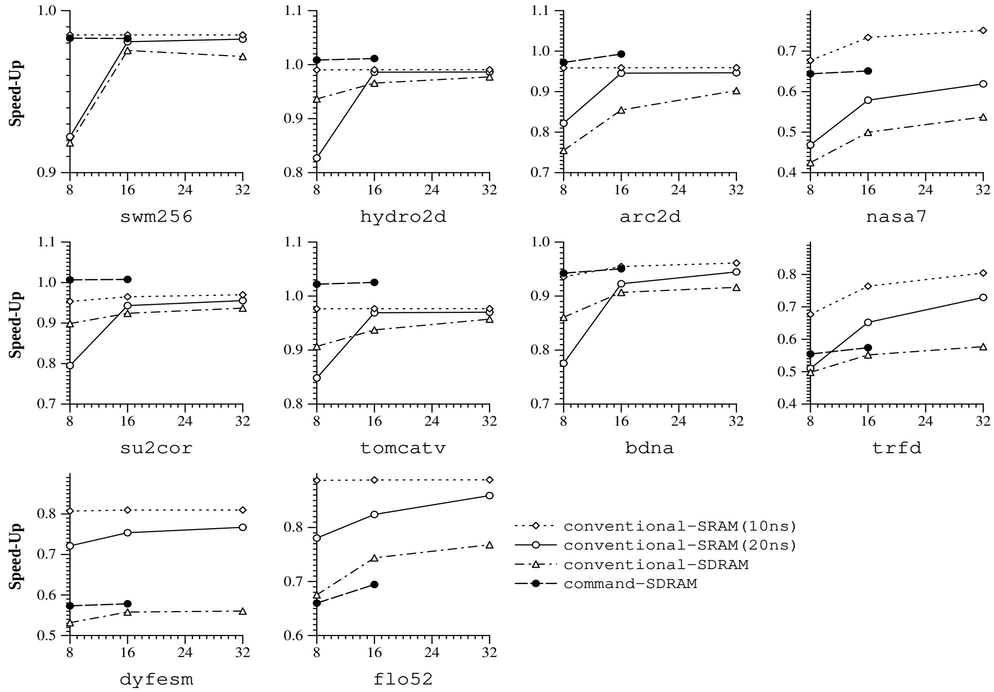

Figure 10: Command system performance results over a 500Mhz ooo-vector machine.

different strides can be simultaneously queued at the section controllers. The section controllers are intelligent enough to multiplex the several subcommands from independent vector requests, effectively pre-advancing row accesses.

Our simulations show that this command based memory system can improve performance over a traditional SDRAM-based memory system by factors that range between 1.15 up to 1.54. Moreover, in many cases, the command memory system outperforms even the best SRAM memory system under consideration, although by a small margin. Overall the command based memory system achieves similar or better results than a 10ns SRAM memory system (a) using fewer banks and (b) using memory devices that are between 15 to 60 times cheaper.

This paper has focused on the uniprocessor case. We believe that the command paradigm would yield even better results in a multiprocessing environment, since it can drastically reduce conflicts between competing processor requests.

References

[1] W. C. Hsu and J. E. Smith. Performance of cached dram organizations in vector supercomputers. *Computer Architecture News*, 21(2):327–336, 1993.

[2] L.I. Kontothanassis, R. A. Sugumar, G. J. Faanes, J. E. Smith, and M. L. Scott. Cache performance in vector supercomputers. In *Proceedings of Supercomputing'94*, Washington D.C., November 1994. IEEE Computer Society Press.

[3] Roger Espasa, Mateo Valero, and James E. Smith. Out-of-order Vector Architectures. In *MICRO-30*, pages 160–170. IEEE Press, December 1997.

[4] Charles A. Hart. CDRAM in a Unified Memory Architecture. In *COMPCON'94*, 1994.

[5] Richard Crisp. Direct rambus technology: The new main memory standard. *IEEE Micro*, 7:18–28, November/December 1997.

[6] Betty Prince. *High Performance Memories.* Wiley & Sons, Ltd., 1996.

[7] Convex Press, Richardson, Texas, U.S.A. *CONVEX Architecture Reference Manual (C Series)*, sixth edition, April 1992.

[8] Keneth C. Yager. The Mips R10000 Superscalar Microprocessor. *IEEE Micro*, pages 28–40, April 1996.

[9] *NEC RAM Selection Guide.* NEC Electronics, 1996.

[10] *Fujitsu 64 Mbit Synchronous DRAM: target specifications.* FUJITSU Microelectronics, http://www.fujitsumicro.com, 1996.

[11] R. Espasa, M. Valero, D. Padua, M. Jiménez, and E. Ayguadé. Quantitative analysis of vector code. In *Euromicro Workshop on Parallel and Distributed Processing.* IEEE Computer Society Press, January 1995.

[12] Montse Peiron, Mateo Valero, Eduard Aygaudé, and Tomás Lang. Vector multiprocessors with arbitrated memory access. In *ISCA-22*, pages 243–252, Santa Margherita Ligure, Italy, June 22–24, 1995.

[13] Roger Espasa Jesus Corbal and Mateo Valero. Command vector memory systems: High performance at low cost. Technical Report UPC-DAC-1998-8, Universitat Politècnica de Catalunya, 1998.