The Hierarchical Multi-Bank DRAM: A High-Performance Architecture for Memory Integrated with Processors

Tadaaki Yamauchi[1], Lance Hammond[2], and Kunle Olukotun[2]

[1]ULSI Laboratory, Mitsubishi Electric Corporation
[2]Computer System Laboratory, Stanford University

Abstract

A microprocessor integrated with DRAM on the same die has the potential to improve system performance by reducing the memory latency and improving the memory bandwidth. However, a high performance microprocessor will typically send more accesses than the DRAM can handle due to the long cycle time of the embedded DRAM, especially in applications with significant memory requirements.

A multi-bank DRAM can hide the long cycle time by allowing the DRAM to process multiple accesses in parallel, but it will incur a significant area penalty and will therefore restrict the density of the embedded DRAM main memory. In this paper, we propose a hierarchical multi-bank DRAM architecture to achieve high system performance with a minimal area penalty. In this architecture, the independent memory banks are each divided into many semi-independent subbanks that share I/O and decoder resources.

A hierarchical multi-bank DRAM with 4 main banks each composed of 32 subbanks occupies approximately the same area as a conventional 4 bank DRAM while performing like a 32 bank one — up to 65% better than a conventional 4 bank DRAM when integrated with a single-chip multiprocessor.

1: Introduction

High speed DRAMs, such as synchronous DRAMs, have been developed to achieve high data transfer rates for serial accesses. This speedup offers a large benefit for bandwidth-intensive applications. However, improvements in memory latency have not kept up with the almost exponential increase in processor speed, so system performance is often limited by the memory access time [1].

Recently, microprocessors integrated with DRAM main memory have been developed [2]. In this architecture, the DRAM and the processor are connected using a wide internal data bus on a single die. The high speed data transfer through the internal data bus can improve the memory latency, and therefore the performance, because the load capacitance of the internal data bus is negligibly small compared to that of an external data bus.

Unlike fast SRAM memories, the *cycle* time for access to a random row in DRAMs is longer than the *access* time. The sense, restore, and precharge operations for the selected memory cells have to be done sequentially during each access cycle or the stored data in the memory cells might be destroyed. In conventional DRAMs, no other random accesses may be initiated during this time. The timing of these operations is unfortunately determined by the characteristics of the memory array, and thus will not improve when DRAM is embedded on a processor chip.

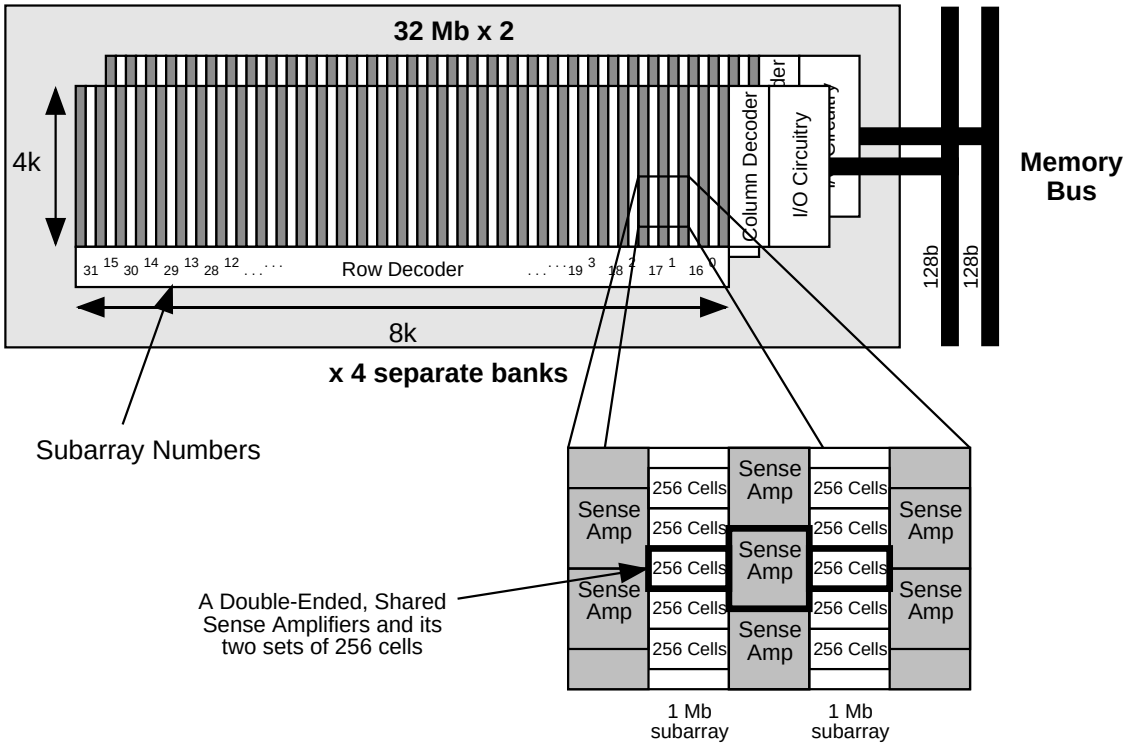

Figure 1: Block diagram of the basic embedded 256Mb DRAM built from four independent banks.

High performance microprocessors such as wide issue superscalar processors or single chip multiprocessors can produce many memory accesses quickly. For example, in a single chip multiprocessor composed of superscalar processors, the frequency of the memory references is quite large because each processor can independently issue one or more DRAM accesses. When a high performance microprocessor is integrated with DRAM main memory on a die, the numerous access requests to the embedded DRAM could accumulate in memory queues and experience significant queueing delay because of the long row cycle time. To avoid these problems, a multi-bank embedded DRAM could be implemented. The simulated results for applications with large data sets show that a 32 bank embedded DRAM is up to 1.67 times faster than a 4 bank one. However, the increase in the number of banks incurs a significant area penalty. For example, a 32 bank embedded DRAM is about 1.8 times larger than a 4 bank one. As a result, conventional multi-bank DRAM architectures significantly reduce the amount of DRAM which can be embedded on a single chip. To achieve high performance, while minimizing the area penalty, we propose the hierarchical multi-bank embedded DRAM architecture in this paper. The simulated results show that the proposed architecture performs like a conventional 32 bank architecture, with only a minimal area penalty.

The reminder of this paper is organized as follows. Section 2 gives a brief overview of our baseline DRAM architecture. Section 3 discusses the two solutions to the high access rate problem: multiple banks and the hierarchical multi-bank DRAM. The architectural model used to evaluate the system performance is covered in Section 4, and the simulation environment is discussed in Section 5. In Section 6, realistic and limiting-case systems are evaluated. Finally, we conclude in Section 7.

2: DRAM architecture

2.1: The embedded DRAM architecture

In this paper, a 256Mb DRAM is used as the embedded memory. 256Mb DRAMs are promising successors to 64Mb DRAMs, which are already mass-produced. Fig. 1 shows the block diagram of a 256Mb DRAM memory array. Four independent DRAM banks are on each die in typical 256Mb DRAM chips [4,5]. We use this as our baseline architecture. Every bank has two 32Mb memory arrays, each of which is further subdivided into 32 1Mb memory subarrays. Within each of these subarrays, there are 4K bit line pairs with 256 cells connected to each pair. Each subarray has a 2K block of sense amplifiers at each end, so sense amplifiers must be shared between neighboring arrays, as the inset of Fig. 1 indicates. The sense amplifier blocks also include equalization/precharge circuitry and a transfer gate to connect the sense amplifier to the I/O lines. During each access, 128 bits are read from each of the two 32Mb memory arrays that form one bank and are then sent to the CPU over a 256 bit wide memory bus that is shared among the four DRAM banks. The access and cycle times of the embedded DRAM are assumed to be 30ns and 60ns, respectively. These values are quite reasonable for a mass-produced 256Mb DRAM using a 0.25mm technology [4,5,9, with consideration for the fact that these numbers are for handpicked lab samples under optimum conditions]. These times are also reasonable compared with modern discrete DRAMs, which typically have 50-70ns access times, including the delay of the pin I/O drivers.

A possible problem with this integrated DRAM is that 32MB of main memory may not be enough in high-end computer systems. Since a fixed amount of memory is integrated on the die, it is difficult to adjust the amount of memory in different systems. In this case, off-chip DRAM may be added to the system to form another memory hierarchy level below the on-chip main memory. Data movement between the two can be controlled by a software modification of the existing virtual memory system. All of the applications which we are using to evaluate system performance in this paper fit within a single 32MB main memory, so virtual memory operation is not an issue in this paper, but it is covered briefly in [11].

2.2: A typical access

It is enlightening to examine the course of an access through a modern DRAM, in order to provide a basis for our proposed changes. For the purposes of this discussion, each access is divided into nine stages. In reality, some of these stages would be overlapped in time, but this breakdown divides each access into simple steps. This discussion refers heavily to Fig. 2, which depicts how the hardware within a DRAM handles the necessary operations. Fig. 3 summarizes how a physical address sent to the DRAM by the CPU will be broken up and decoded.

Before discussing how an access works in a DRAM chip, we need to understand the hardware integrated into a modern, high-density DRAM. Fig. 2(a) depicts a simplified version of a block within a DRAM array. The shaded vertical block depicts a single subarray within the DRAM array, along with the sense amplifiers (SA), equalization/precharge circuits (EQ), and NMOS transfer gates (transistor schematics) that line each side of the array and are shared with the subarrays to each side. The shaded horizontal block depicts a single column slice within the DRAM array. In all of the DRAM configurations from Section 2.1, each subarray is divided into 32 column slices that each produce 4 bits of the full 128 bits produced by a single array on each access. The layout of the column decoders necessitates this particular grouping of bits. The light grey intersection of the two blocks delineates a fundamental DRAM block within the array — a 128x256 block of DRAM cells lined with 64 sense amplifiers on each side that can read or write

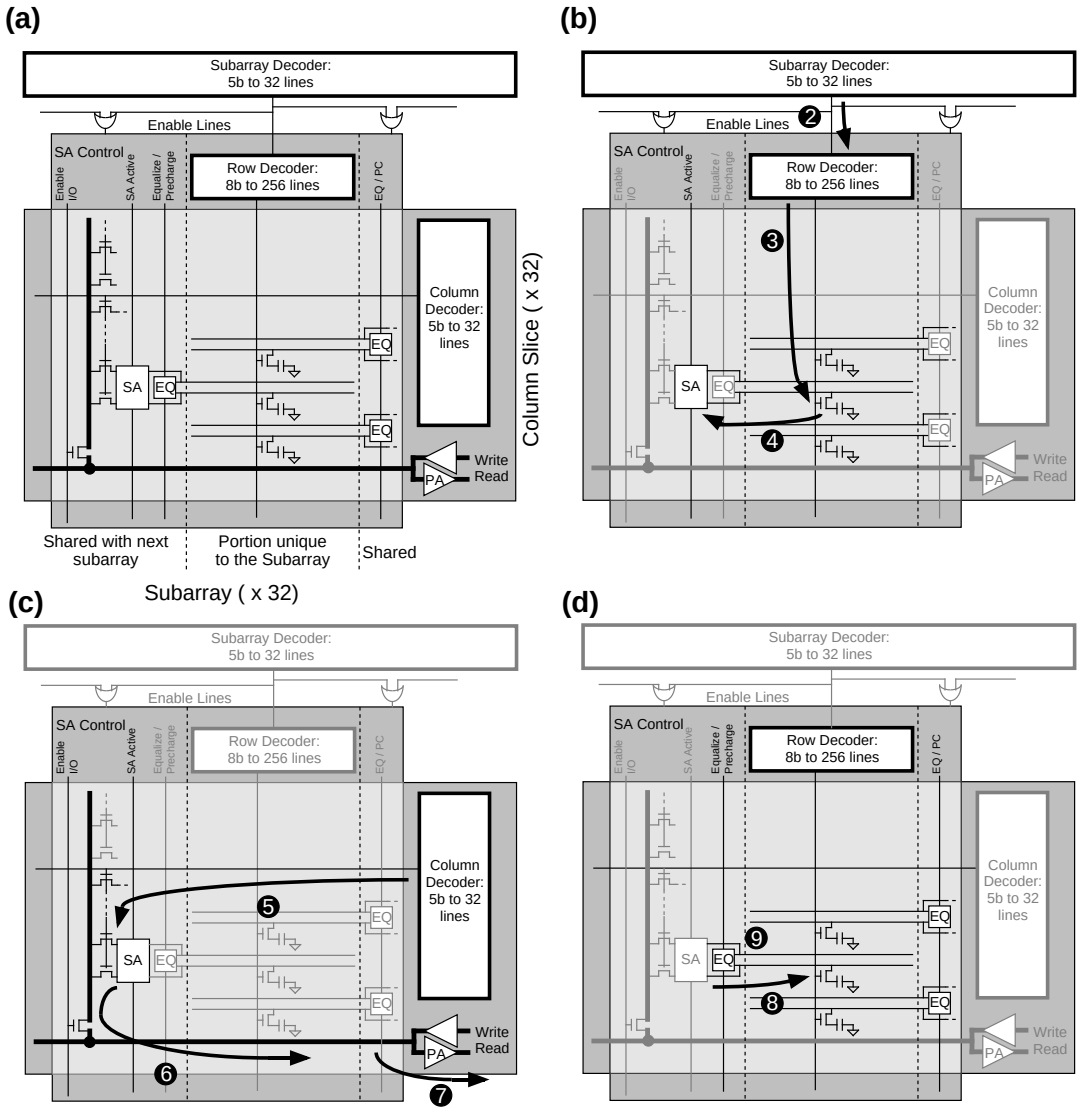

Figure 2: (a) A representative diagram of the hardware used during a DRAM access. (b) How the hardware is used during the row access. (c) How the hardware is used during the column access. (d) How the hardware is used during the restore and precharge phase.

4 bits at once. Four local I/O lines within each DRAM block — one for each bit that can be read from or written to the block at once — attach the sense amplifiers to the global I/O lines running the length of the column slice. At the edge of the subarray, a 8-to-256 row decoder and sense amplifier control drivers are shared by all 32 column slices in that subarray. Above that, a global 5-to-32 decoder selects a subarray to handle each access. Meanwhile, the edge of the column slice contains a 5-to-32 column decoder and I/O drivers that are shared by all of the DRAM blocks in that column slice.

The following steps describe a read access to this particular DRAM in detail:

1. *Bank Decoding and Access Queueing:* First, address bits 6-5 are decoded to select which of the independent DRAM banks should recover the cache line. The access is sent to the access queue for the appropriate bank and then forwarded to the memory array as soon as possible — when all previous accesses for that bank have completed.

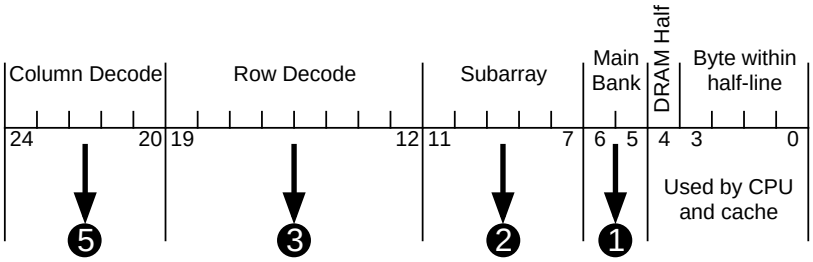

Figure 3: How a physical address from the CPU is broken down.

2. *Subarray Decoding:* (Fig. 2(b)) Once the access is started, the global subarray decoder uses bits 11-7 of the address to select a subarray. Its output enables a subarray's row decoder and the sense amplifier controllers to either side of the subarray. The output of this decoder must be kept stable for all of the following steps.
3. *Row Decode:* (Fig. 2(b)) The enabled row decoder uses bits 19-12 of the address to enable the wordline for one of 256 rows within the subarray. Driving long, high-resistivity, polysilicon wordlines used to be a speed-limiting factor in DRAMs, but modern DRAMs have their wordlines frequently strapped to a first-level metal "support" wordline running directly above the conventional polysilicon one. As a result, this step is relatively fast. The output of this decoder is kept stable until the end of step 8.
4. *Read Data to Sense Amplifiers:* (Fig. 2(b)) The sense amplifiers to either side of the selected subarray are enabled, and all 4Kb in the selected row are read out — 2Kb to either side of the subarray. There are actually 8K bitlines in each subarray, with 128 bits of storage attached to each. During any particular access, half of the lines in the subarray, one from each bitline pair, are connected to memory cells. The other member of each bitline pair is not connected to any cells, and simply acts as a reference line for its sense amplifier. The DRAM cells have been omitted from the reference lines in the figure to avoid cluttering the graphic. Afterwards, the sense amplifiers act as a set of latches to hold the read-out data until step 8.
5. *Column Decode:* (Fig. 2(c)) The column decoders, repeated in each column slice, decode the last part of the address — bits 24-20. Each then drives the appropriate column select line in its column slice. The select line in each column slice enables the I/O gates for 4 adjacent sense amplifiers in the DRAM block at the selected subarray, connecting them with the 4 local I/O lines in the DRAM block. The output from this decoder is only important for steps 6 and 7, below, that involve I/O.
6. *Send Data over I/O Lines:* (Fig. 2(c)) The subarray's I/O enable signal is activated, connecting the local I/O lines within the subarray to the global I/O lines passing through each column slice. The data selected in step 5 are then read out from the sense amplifiers to the I/O drivers at the edge of the array through the interconnected local and global I/O lines using a low-swing signal.
7. *I/O Drivers:* (Fig. 2(c)) High gain preamplifiers at the I/O drivers amplify the low-swing signals sent over the I/O lines to full-swing CMOS levels, and then drive the data out onto the memory bus to the CPU.
8. *Restore Data to Cells:* (Fig. 2(d)) As data is read by the sense amplifiers, the bit lines are driven to full swing levels and recharge the capacitors in the original memory cells as a result. During this step, the row decoder turns off its output, locking the restored charge back into the DRAM cell.
9. *Equalize and Precharge:* (Fig. 2(d)) The sense amplifier controller enables the bitline equalization circuitry, which shorts the two bitlines together and precharges them to an

intermediate voltage level in preparation for the next access. Modern DRAMs use highly resistive polysilicon bit lines because they allow more dense layouts than metal bitlines would allow, and as a result the precharge time is typically greater than 30ns. After the bitlines are equalized and precharged, the equalize circuit is left active to hold the bitlines in position until another access is initiated.

Write accesses are similar, except that data is driven from the I/O logic to the sense amplifiers during step 7, where it overrides the data previously latched into the sense amplifiers and resets the bitline levels before step 8.

3: A problem — too many accesses at once

A critical problem inherent in embedded DRAM lies in its inability to handle the large number of memory accesses that a high-performance processor can generate. Successive accesses may proceed immediately if they do not need to access the same bank, since each memory bank is completely independent. However, successive accesses that need the same bank must queue up and serialize. To make matters worse, DRAM requires that the sense, restore, and precharge operations for selected memory cells be done sequentially during a single access, before the next access may start, or the data stored in the memory cells might be destroyed. The latency of these operations is greatly dependent on characteristics of the memory array such as the parasitic resistance and capacitance of the bit lines and the sensitivity of the sense amplifiers. As a result, the cycle time of reading and restoring a row in embedded DRAM is typically larger than the access time, and thus limits the ability of individual banks to accept successive accesses. This is not a problem if the processor embedded with the DRAM is fairly low-performance, as in [2]. It also has not historically been a problem for discrete DRAMs, because the off-chip interface limits the number of accesses that can be pending at once. However, any kind of high-performance processor can send requests to embedded DRAM at a *very* high rate. This can result in many accesses being queued at each bank, waiting one or more DRAM cycle times — which will typically be tens or hundreds of CPU clock cycles — until all previous accesses complete. As a result, the system performance can be significantly degraded by the long cycle time. The rest of this section describes two ways to address this problem: adding more DRAM banks, and making a hierarchical multi-bank DRAM.

3.1: A conventional solution — more DRAM banks

Since row cycle times are only a problem when multiple accesses are sent to the same DRAM bank, the most straightforward solution is to simply increase the number of independent DRAM banks in order to lower the probability of a conflict. The cycle time can be hidden as long as succeeding references are issued to other banks while a row cycle is completing in any one bank. However, this solution causes the area of the embedded DRAM to be enlarged. Fig. 4(a) shows the block diagram of a 32 bank embedded DRAM. Instead of 4 pairs of 32Mb blocks, the memory is made up of 32 pairs of 4Mb blocks, each composed of 4 1Mb subarrays identical to those used in the 4-bank architecture. As shown in Fig. 4(a), each of these smaller memory blocks has an independent row decoder, column decoder, and I/O circuitry.

The relative area of the embedded DRAM, not considering control circuitry, is estimated in Fig. 4(b). The area penalty is caused by the increase in the number of the sense amplifiers, column decoders, and the amount of I/O circuitry. For every new bank, a duplicate set of column decoders and I/O circuitry are required, resulting in a significant area penalty directly proportional to the number of banks. The area penalty of I/O circuitry is especially large, since the embedded DRAM has a 256-bit wide internal memory bus to achieve high memory bandwidth,

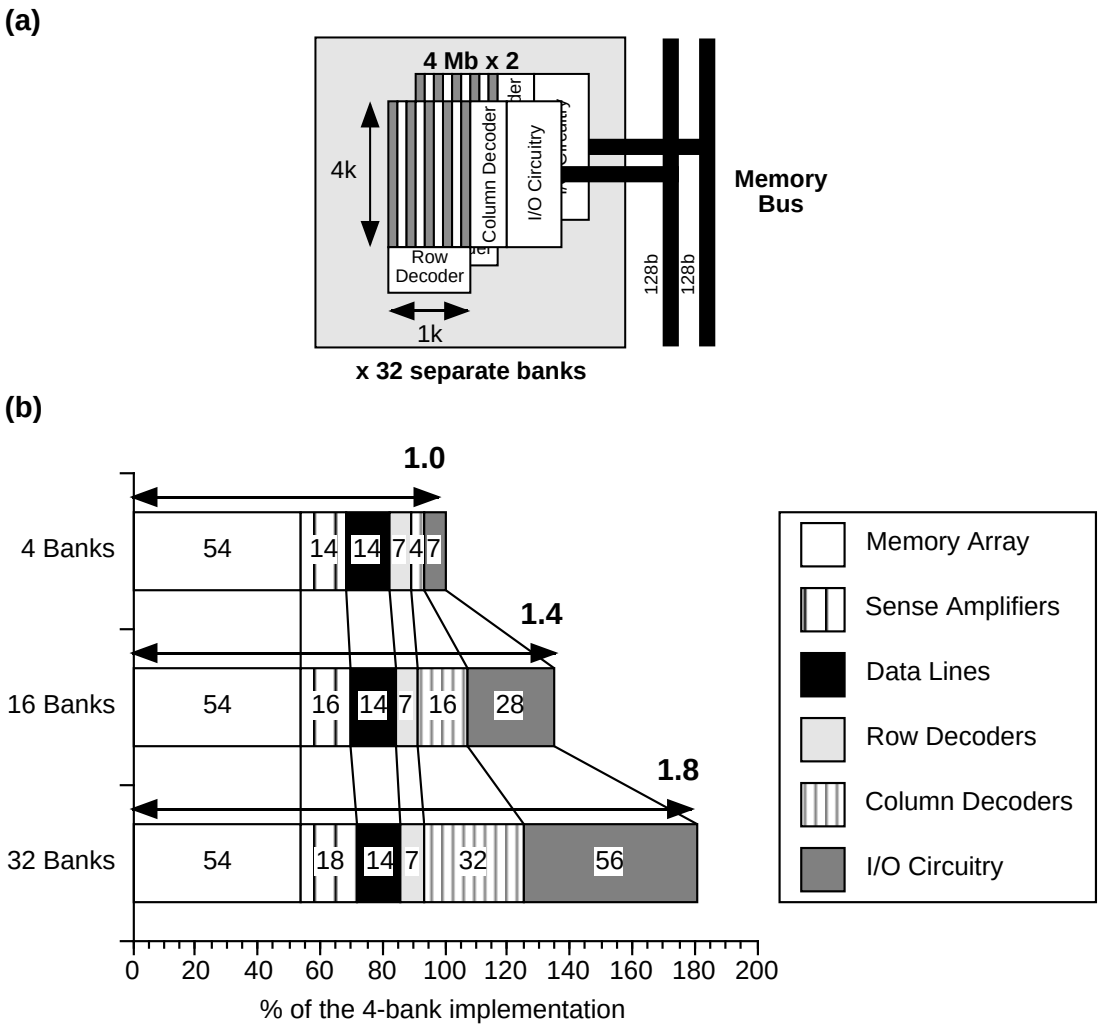

Figure 4: (a) Block diagram of the embedded 256M bit DRAM with 32 independent banks. (b) Comparison of the area requirements for different multi-banked DRAM configurations.

and each output bit requires a full set of I/O drivers. The increase in sense amplifier area is more subtle. As we mentioned in Section 2.1, each of the 1 Mb subarrays uses two sets of 2K sense amplifiers, one on each side. While most of these are shared between neighboring subarrays, the sets of sense amplifiers at the ends of each bank cannot be shared. Every new bank therefore requires an additional set of unshared sense amplifiers, resulting in a small area increase.

Overall, a 16 bank embedded DRAM is about 1.4 times larger and a 32 bank one is about 1.8 times larger than a 4 bank one. This area penalty will significantly reduce the amount of DRAM which may be economically integrated with a processor. Current DRAM designers probably think that this degree of banking is ridiculous, but no previous DRAM application has allowed such a high number of references to be pending on one DRAM chip at once due to the limitations of the off-chip interfaces in conventional DRAMs. The pin latency and bandwidth has historically been a much more serious problem, so the number of banks has never been a major performance-limiting factor. A high-performance embedded processor can change the situation entirely.

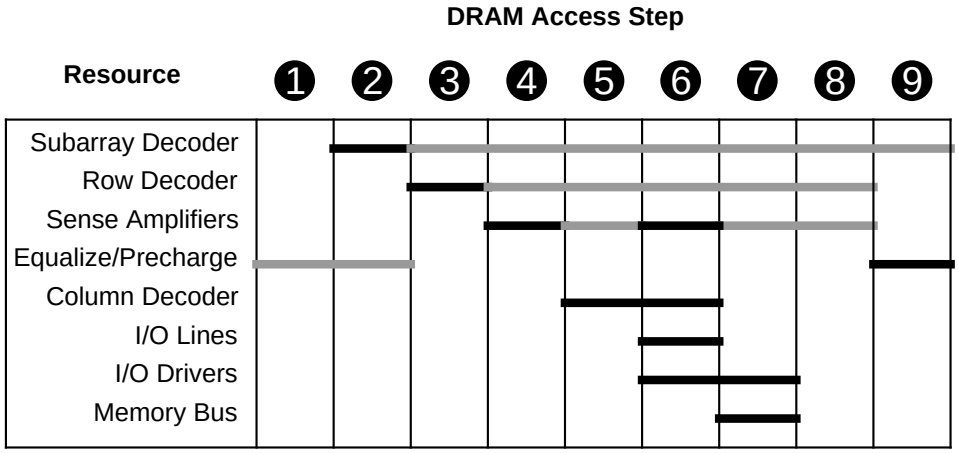

Figure 5: Resource utilization within the DRAM array. Black lines indicate active logic, grey lines indicate logic that is holding an output value steady for other units, and no line indicates that the resource is free.

3.2: Our solution — the hierarchical multi-bank architecture

As an alternative, we propose a "hierarchical multi-bank" DRAM architecture to achieve the high performance of an extensively banked DRAM with only a minimal area penalty. In this proposed architecture, we simply convert the memory subarrays within the DRAM banks into semi-independent "subbanks." This is a useful approach, since a large part of each DRAM access actually occurs only locally within individual DRAM subarrays. Fig. 5 shows the usage of the DRAM array's hardware during the various steps of a DRAM access. A few interesting observations can be made using this simple plot. First, the subarray selection decoder, shared by all arrays, is used only at the beginning of an access, and then just holds its value constant for the duration of the access. Second, the column slice resources — the column decoder and I/O hardware — are only used in steps 5–7 from Section 2.2. The time required for these steps is actually fairly short — about 8ns, based on data presented in [4]. Finally, all of the other resources are physically localized within each subarray and its associated sense amplifiers to each side, and can thus work fairly independently from the other subarrays. With the arrays configured as semi-independent subbanks, much of the other 52ns required for each access may be overlapped with other accesses to that bank. As a result, very little hardware is actually required to enable a significant amount of parallelism. Fig. 6 summarizes the necessary changes.

First, Fig. 6(a) shows the additions to each subarray — just a few pipeline registers and controls. A single set-reset flip-flop after the output from the subarray select decoder holds the subarray activation output steady for the duration of the DRAM access, allowing the decoder to initiate other accesses. The self-timed logic within the sense amplifier controller clears this flip-flop when the access is complete. Buffers are also necessary to hold the address for the access associated with this subbank. This would physically consist of two separate buffers — one near the row decoder and one near the column decoder. The row address buffer holds the input to the row decoder steady throughout an access. The sense amplifier control logic connects the column address buffer to the column decoder and activates the local I/O to global I/O transmission gate *only* during steps 5–7, allowing other accesses to use the column decoder and global I/O lines during the other steps. The actual DRAM arrays themselves are unmodified.

The second modification must be made to the DRAM access queues. They must be modified to detect accesses that will not conflict and to then initiate nonconflicting accesses in parallel as

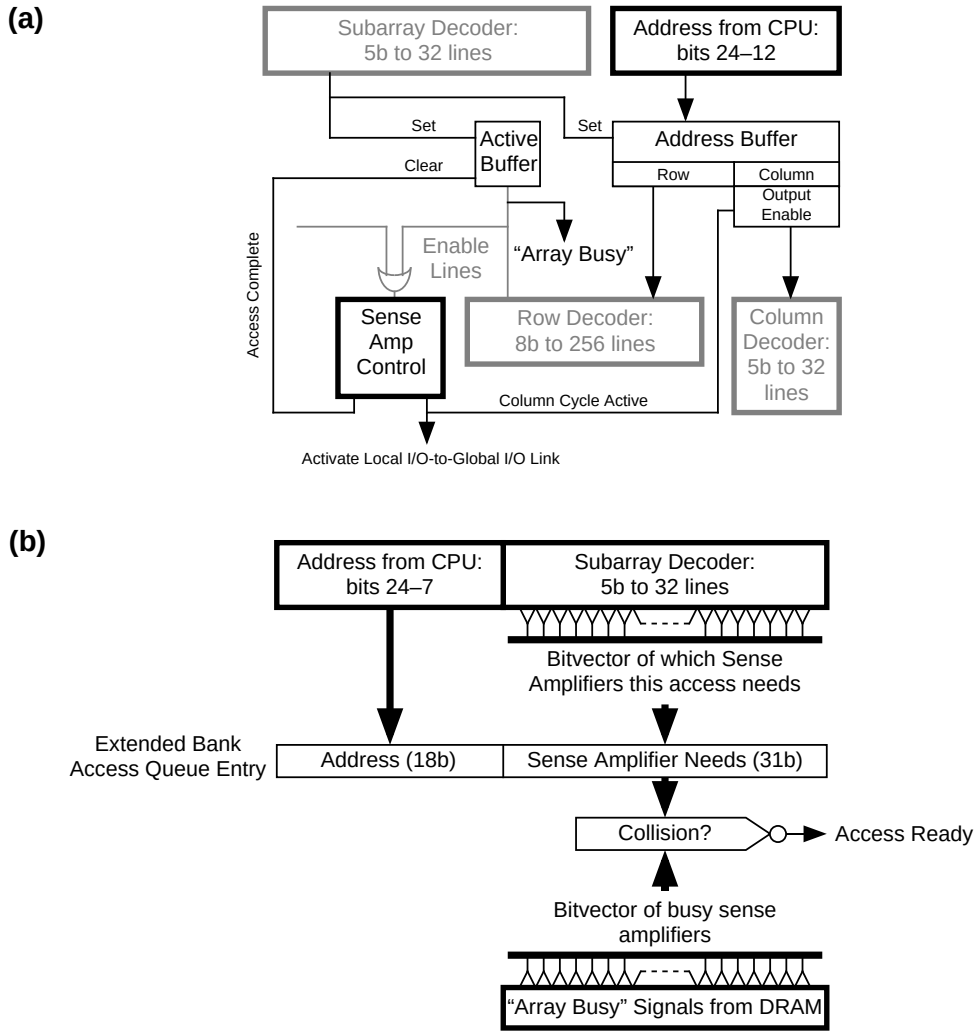

Figure 6: (a) Modifications to each DRAM subarray to turn it into a semi-independent subbank. (b) Modifications to the memory access queues.

quickly as possible, a task similar to that performed by an out-of-order processor's instruction queue. Fig. 6(b) shows the additions to the bank access queues used in step 1 of each DRAM access. As each access is added to the queue, its subarray address is passed through a subarray selection decoder, which generates a bitvector of sense amplifiers needed by the access. Since each access needs the sense amplifiers on both sides of a subarray, two adjacent bits are set in this bitvector to represent both. The only exception to this is that the sense amplifiers at either end of the memory array may be safely omitted from the bitvector, resulting in a vector of 31 bits for a 32 subarray DRAM bank. After the entry is queued, it is compared against the "busy" signals returned from the subarray-active flip-flops to detect conflicts. These signals are also doubled to represent which sense amplifiers are busy. As a result, a conflict is detected whenever there is an access already using the subarray needed by this access or an access being processed in either of its neighbors. The 0, 16, 1, 17, ... 31 numbering of the subbanks (indicated in Fig. 1) helps prevent these "neighbor" conflicts on the common case of sequential accesses. When all conflicts are cleared, the "ready" signal is raised and the DRAM bank will start the access as soon as possible, while maintaining enough time between accesses to avoid collisions at the column decoders and I/O logic. It should be noted that accesses not at the head of the memory access

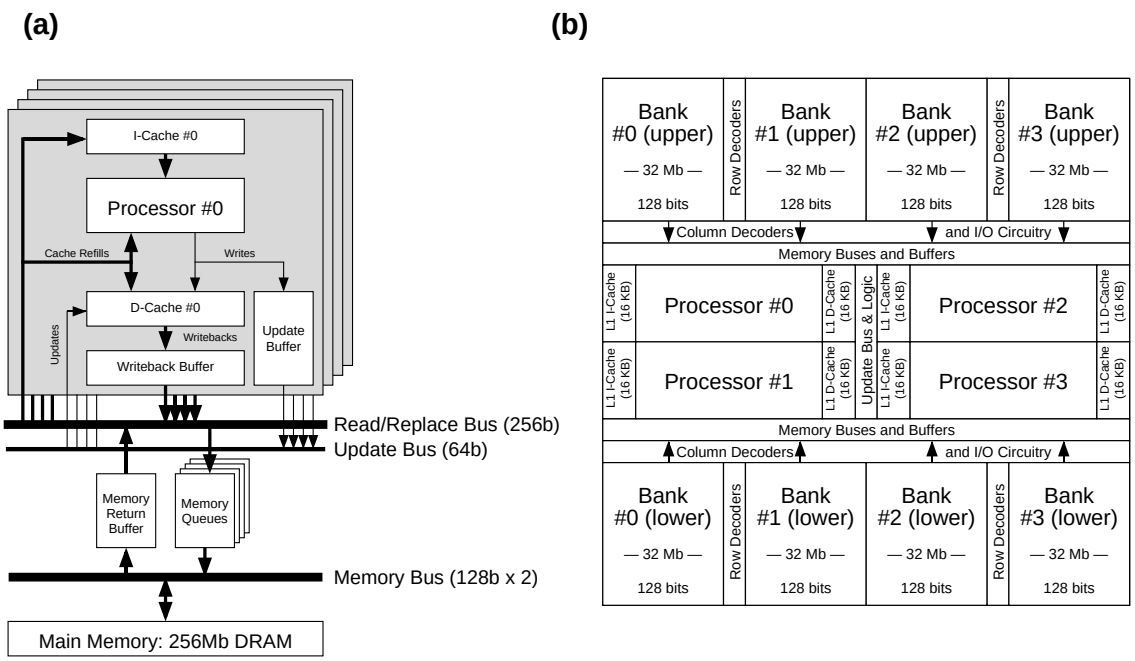

Figure 7: (a) Block diagram of the single chip multiprocessor integrated with 256Mb DRAM on the same die. (b) The floor plan for the single chip multiprocessor integrated with a 256Mb DRAM.

queue may have their "ready" signals raised and lowered multiple times if older conflicting accesses are "ready" also and started first as a result. Because of this behavior, the "ready" signal must be continually evaluated by dedicated collision detection hardware in each queue entry. Eventually, all accesses will move to the head of the DRAM access queue and be processed by the DRAM bank. This logic adds considerable complexity to the memory access queues, but these are fairly small structures relative to the size of the DRAM arrays themselves.

Fig. 6 depicts the changes necessary to fully transform all 32 subarrays within a main bank into 32 subbanks. It is also possible to group adjacent subarrays together into larger subbanks, in order to save a few registers. Groups of subarrays acting as a single subbank can share address registers from Fig. 6(a), but not subarray-enabling flip-flops. The bitvectors in Fig. 6(b) will still work and will be N-1 bits long with N subbanks. Whenever any subarray within a subbank is enabled, the bits representing the sense amplifiers at both ends of the subbank are enabled. Other than that, the hardware is identical to the full 32 subbank scheme.

Overall, the additions necessary to make the subarrays into semi-independent subbanks are logically somewhat complex, but since no major additions to the DRAM array are necessary only a very small area penalty is incurred.

4: An integrated MP architecture

To evaluate the performance of the DRAM cycle time solutions, a high speed microprocessor must be integrated with DRAM main memory. For the purposes of this paper, a single chip multiprocessor is considered. A single chip multiprocessor has been recently studied as a promising candidate for future high performance microprocessor design [3]. When present DRAM processes are applied directly to the fabrication of high performance processors, the processor's logic gates typically suffer speed and area penalties as a result. However, some DRAM manufac-

turers are developing merged process technologies which can achieve high density for the embedded DRAM without degrading the logic performance. In this paper we nominally assume that the single chip multiprocessor integrated with our high density DRAM can run at a clock frequency of 500MHz. Fig. 7(a) shows the block diagram of a single chip multiprocessor integrated with DRAM. To reduce the communication overhead, four 2-way superscalar processors are interconnected with a 256 bit wide read/replace data bus whose bit width is identical to the cache line size. As a result, each replace or line fill operation between the cache and DRAM occupies the bus for only one CPU cycle. Since the four processors share the read/replace bus, arbitration for this resource requires an extra cycle. Each of the four processors has a 16KB SRAM instruction cache and a 16KB SRAM data cache, both accessible in a single 2 ns clock cycle. Since each cache can only be accessed by a single processor or its single load/store unit, no additional overhead is incurred handling arbitration among independent memory access units within a processor. A writeback cache with a write miss allocate policy is implemented to reduce the traffic on the read/replace data bus. Coherency among the individual data caches is maintained using an update coherence protocol. The data written by each processor is broadcasted to all other data caches through the 64 bit update bus. Each data cache has two I/O ports to minimize the interference caused by the update operation. Like the read/replace bus, the memory bus and queues may only accept one access per cycle, and therefore incur an extra arbitration cycle before a processor may use them. Unlike the read/replace bus, the memory bus cycle is twice as long as a CPU clock cycle. As a result, the peak memory bandwidth without bank conflicts is 8 GB/s.

The floor plan for a single chip multiprocessor integrated with a 256Mb DRAM is shown in Fig. 7(b). In order to reduce the critical path of each memory access, the multiprocessor is located between the divided upper and lower parts of the embedded DRAM. With a 0.25mm merged process technology we estimate that the chip size will be approximately 24 millimeters on a side. The size of the 4 x 2-way multiprocessor section of the chip is extrapolated from the current MIPS R10000 processor [10].

5: Methodology

5.1: Simulation environment

We execute applications in the SimOS simulation environment [6]. With SimOS, the processors, memory hierarchy, and cache coherence issues are modeled with full consideration given to contention between processors due to shared resources such as the central data bus. SimOS emulates a multiprocessor running the full MIPS-2 instruction set interacting with a realistic set of I/O components, allowing the full Silicon Graphics IRIX 5.3 operating system to be executed under our benchmarks. SimOS supports three kinds of CPU simulators, which allow trade-offs to be made between simulation speed and accuracy. In this paper, the slowest, most detailed CPU simulator is used. This model supports multiple instruction issue in each processor, along with full emulation of dynamic scheduling, speculative execution, and non-blocking memory references. The cache and memory system components shown in Fig. 7(a) are completely event-driven and interface to the SimOS processor models.

5.2: Applications

To evaluate the system performance, five realistic applications are used. Table 1 shows the five applications: one SPEC95 integer benchmark (compress), one SPEC92 integer benchmark (eqntott), and three SPEC95 floating point benchmarks (swim, tomcatv, applu). The applications are parallelized in different ways to run on a multiprocessor. The compress benchmark cannot be

	Floating Point Applications
swim	shallow water model with 1K x 1K grid
tomcatv	mesh-generation with Thompson solver
applu	solver for parabolic/elliptic partial differential equations
	Integer Applications
compress	compresses and uncompresses files in memory
eqntott	translates logic equations into truth tables

Table 1: Applications.

effectively parallelized, so only one of the four processors is used. Eqntott is parallelized manually by modifying a single bit vector comparison routine that is responsible for 90% of the execution time of the application [7]. It is characterized by a small working set. The SPEC95 floating point benchmarks are automatically parallelized by the SUIF compiler system [8] across loop iterations at a reasonably coarse level.

6: Performance results

6.1: The overhead caused by DRAM cycle time

To estimate the influence of the long DRAM cycle times, we first simulate the case with no bank contention during row cycles at all by setting the DRAM cycle time equal to the access time. The relative speedup of this idealistic scenario over the realistic case is shown in Fig. 8.

Fig. 8 indicates that the reduced cycle time significantly enhances the performance of the swim, tomcatv, and applu benchmarks. These three applications have large reference data set sizes, greater than 16MB. The average miss rate in the four data caches is 7.6% in swim, 4.6% in tomcatv, and 2.7% in applu. Swim's performance improves the most, since its data cache miss rate is the highest among the three applications with large data sets. Integer applications such as compress and eqntott display little enhancement. The average miss rate among the four data

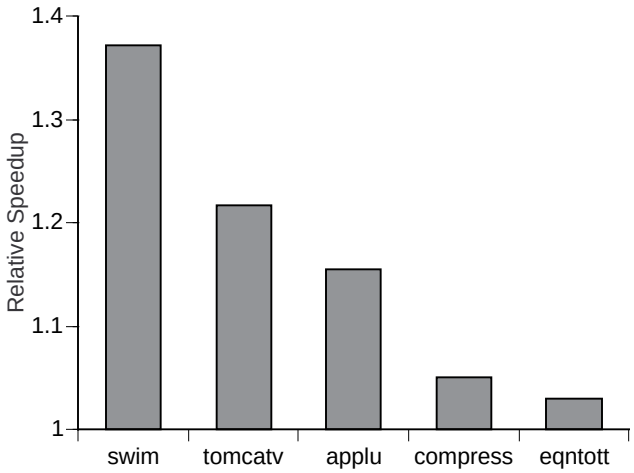

Figure 8: Relative speedup in the case of the t_c (DRAM cycle time) = t_a (access time) = 30ns, compared to the case of t_c = 60ns and t_a =30ns.

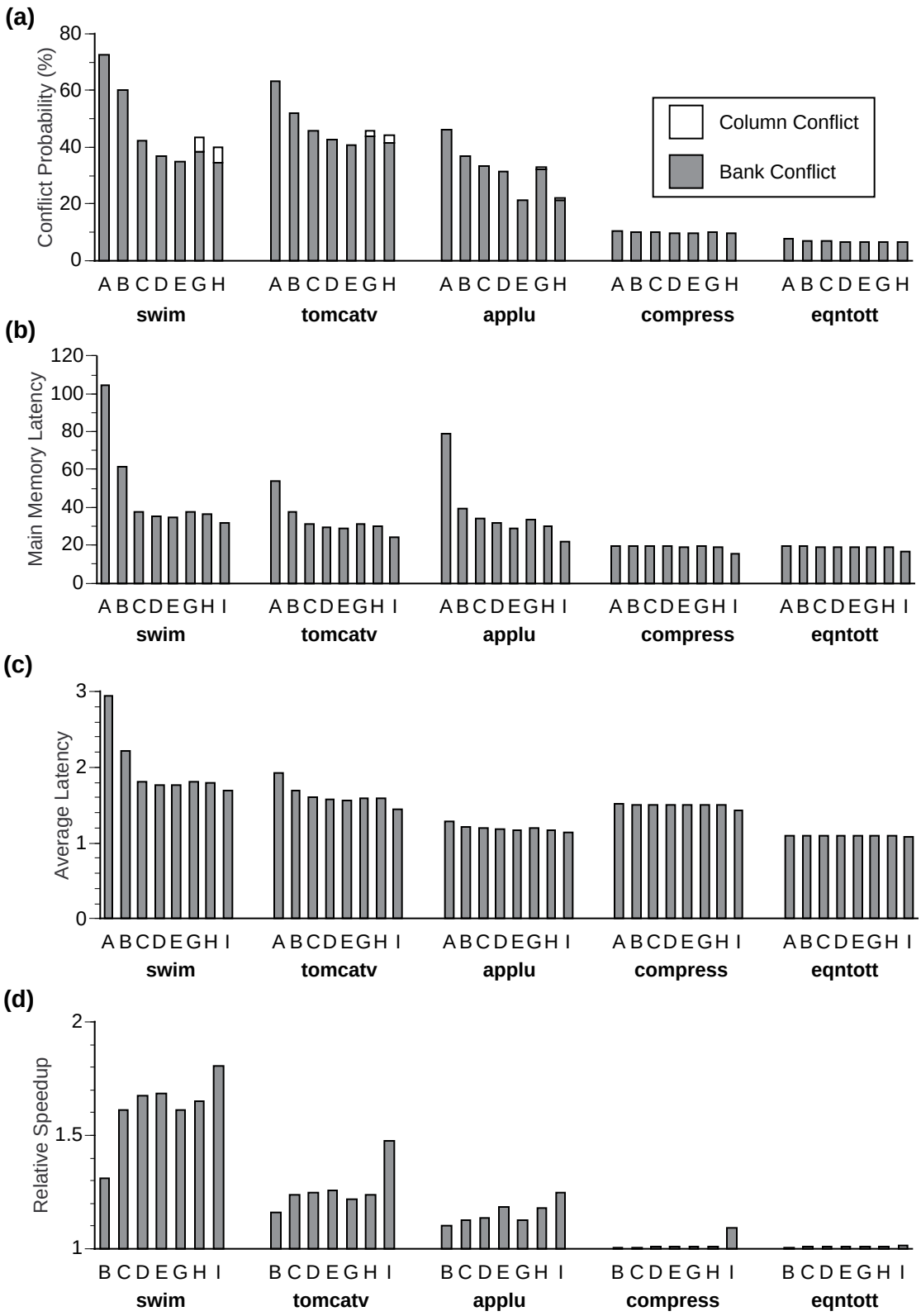

Figure 9: See Table 2 for the key to letter codes. (a) The conflict probability. (b) The average latency of the embedded DRAM. (c) Overall average memory latency. (d) The relative speedup compared to the 4 bank architecture.

Code Letter	Configuration
A	4 banks
B	8 banks
C	16 banks
D	32 banks
E	64 banks
G	4 main & 16 sub-banks (two subarrays grouped per sub-bank)
H	4 main & 32 sub-banks (all subarrays are sub-banks)
I	Ideal banking — no conflicts

Table 2: Key to simulated layouts in Figure 9.

caches is 1.5% in eqntott, better than any of the FP applications, due to the relatively small size of the data set. The miss rate of the single active data cache in the unparallelized compress is high, 10.8%, but the average number of the memory references per cycle is small because only one processor is actively issuing memory requests. As a result, there are usually too few memory accesses in flight at once to cause significant amounts of bank contention.

These results show that the long cycle time of the embedded DRAM can significantly restrict the performance when a high performance processor such as a single-chip multiprocessor is integrated on the same die and executes applications which have large reference data sets.

6.2: Architectual performance comparison

Next we show simulated results obtained using the conventional multi-bank and the proposed hierarchical multi-bank architectures. To review, the CPU clock frequency is 500MHz, the embedded DRAM access time is 30ns = 15 CPU cycles, the row cycle time is 60ns = 30 CPU cycles, and the column cycle time is 8ns = 4 CPU cycles. Here we assume that the total access path from the memory cells to the processors is identical in any multi-bank architecture and, therefore, that the access time has no dependence on the number of banks. The probability of bank conflicts is depicted in Fig. 9(a). The increase in the number of banks greatly reduces the probability of bank conflicts in the three floating point applications with large reference data sets. Even in these memory-intensive applications, the worst-case probability of column conflicts in the hierarchical multi-bank architecture is less than 6%. The 256-bit wide memory bus reduces the probability of column conflicts by minimizing the time required to move a cache line into or out of the array. Thus, the column conflicts cause only a small performance degradation in the hierarchical architecture. The number of bank conflicts due to row cycle times is greatly reduced by subbanking. On the integer benchmarks, the probability of bank conflicts does not vary significantly with the number of banks.

The average latency of the embedded DRAM, as shown in Fig. 9(b), is measured by timing accesses from their entry into the memory buffer to their return from the DRAM. Here the unit of the latency discussed in this paper is a CPU clock. The results of the three floating point applications indicate that frequent bank conflicts cause large latencies in spite of the improved access time of the embedded DRAM. The ideal architecture, with no bank conflicts at all, improves the latency down to 15–32 CPU clock cycles, reaching the random access time of the embedded DRAM itself with some applications. There is some overhead even in the ideal case because of contention for the common 256b memory bus. The proposed hierarchical multi-bank architecture

	Effective IPC of 4 banks	Effective IPC of 4 main & 32 sub-banks
swim	1.96	3.23
tomcatv	2.30	2.85
applu	2.88	3.40
compress	0.81	0.82
eqntott	1.96	1.98

Table 3: Comparison of the effective IPC of a 4x2-way multiprocessor integrated with DRAM main memory.

with 4 main banks and 32 subbanks has approximately the same DRAM latency as the conventional 32 bank architecture, while the hierarchical architecture with 4 main banks and 16 subbanks achieves the same latency as the conventional 16 bank one.

The average memory latency including both the caches and main memory is depicted in Fig. 9(c). In addition, the performance is shown as the speedup of each multi-bank architecture relative to the conventional 4 bank one in Fig. 9(d). The increase in the number of banks reduces the memory latency and enhances the performance of the three floating point applications. The swim application, which has the lowest hit ratio in the data caches among these three applications, obtains the largest performance enhancement by reducing the memory latency most significantly.

The proposed hierarchical multi-bank architecture with 4 main banks and 32 subbanks provides relative performances of 1.65, 1.24, and 1.18 in swim, tomcatv, and applu, respectively, equivalent to those achieved by the conventional 32 bank architecture. On integer applications with small data sets, such as compress and eqntott, the performance enhancement is quite small — only 1.01 times faster with the hierarchical multi-bank architecture. Moreover, as we discussed in Section 6.1, compress does not see speedup because it does not overlap memory accesses significantly.

Finally, we evaluate the system performance using the effective IPC, counting only useful instructions completed per cycle, in Table 3. Here the 4 x 2-way multiprocessor is viewed as an integrated processor with an ideal effective IPC of eight. These results show that the hierarchical multi-bank embedded DRAM can achieve significant speedups in applications with large data sets.

6.3: Clock speed variations

Previously, we have assumed an aggressive merged process technology which can achieve high density in the embedded DRAM while maintaining excellent logic performance. In this section, we evaluate the effect of the proposed hierarchical multi-bank DRAM when it is integrated with slower processors, which may be required if the merged process degrades the logic performance. In this discussion, the CPU clock frequency is varied while the memory latencies are kept constant. Fig. 10 shows the relative performance between the conventional 4 bank architecture and the hierarchical multi-bank architecture with 4 main banks and 32 subbanks over a range of CPU clock frequencies while executing the three floating point applications. When the CPU clock frequency is reduced to 333MHz, 200MHz, or 100MHz, the relative speedup of swim is 1.39, 1.16, and 1.04, respectively. These results show that the effects of the hierarchical multi-bank DRAM architectures are most significant in future processors with very high clock rates, when the processors are able to have many memory requests in flight at once.

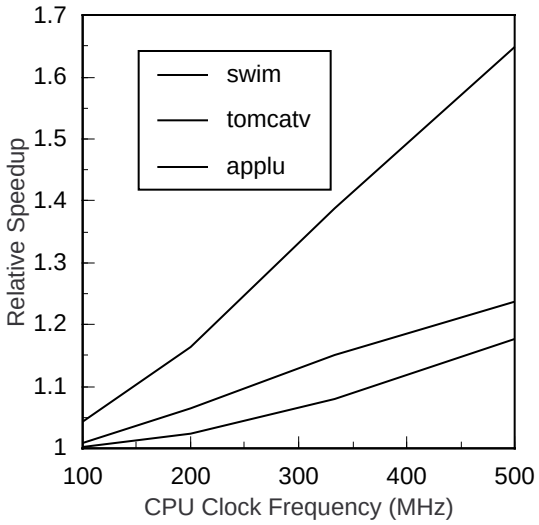

Figure 10: The relative speedup as a function of processor clock rate in three floating point applications. The hierarchical multi-bank architecture with 4 main banks and 32 sub-banks is compared with the conventional 4 bank architecture.

These results may also be interpreted as variations in the speed of the DRAM array, instead of variations in the CPU clock speed. Read this way, the results show that the hierarchical multi-bank DRAM has the greatest impact in systems with relatively slow DRAM, and less of an effect when DRAM is very fast.

7: Conclusion

The microprocessor integrated with DRAM on the same die has the potential to enhance system performance by reducing the memory latency and improving memory bandwidth. The access time of the embedded DRAM is reduced because the load capacitance of the internal I/Os is small compared to external I/Os. However, the DRAM cycle time is determined by the characteristics of the memory array, and thus will not improve when the DRAM is embedded. When a high performance microprocessor, such as a single-chip multiprocessor, is integrated with DRAM, memory access requests may be sent to the DRAM array faster than they may be processed due to the long cycle time of the embedded DRAM.

In this paper, we have discussed the implementation of a conventional multi-bank architecture and the proposed hierarchical one to solve this problem. The conventional multi-bank DRAM architecture can enhance the system performance, but it causes a significant area penalty and critically reduces the embedded main memory size. For example, a 32 bank DRAM is 1.8 times larger than a conventional 4 bank one. To achieve high performance while maintaining high DRAM density, we propose the hierarchical multi-bank architecture. In this architecture, individual subarrays within independent banks are controlled as semi-independent subbanks, that share the main bank's I/O circuitry and decoders.

The hierarchical multi-bank DRAM with 4 main banks each composed of 32 subbanks has approximately the same die size as a conventional 4 bank DRAM while achieving performance comparable to a 32 bank one. With a single-chip multiprocessor integrated with the proposed embedded DRAM, our results show that on applications which have large reference data sets the hierarchical multi-bank DRAM with 4 main bank of 32 subbanks can perform up to 65% better than a conventional 4 bank architecture.

References

[1] A. Saulsbury, F. Pong, and A. Nowatzyk, "Missing the Memory Wall: The Case for Processor/Memory Integration," *23th International Symp. on Computer Architecture*, pp. 90-101, Philadelphia, PA 1996.

[2] T. Shimizu, J. Korematu, M. Satou, H. Kondo, S. Iwata, K. Sawai, N. Okumura, K. Ishimi, Y. Nakamoto, M. Kumanoya, K. Dosaka, A. Yamazaki, Y. Ajioka, H. Tsubota, Y. Nunomura, T. Urabe, J. Hinata, and K. Saitoh, "A multimedia 32b RISC microprocessor with 16Mb DRAM," *Digest of Technical Papers, 1996 IEEE International Solid State Circuits Conference*, pp. 216-217, San Francisco, CA 1996.

[3] K. Olukotun, B. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The Case for a Single-Chip Multiprocessor," *Proceedings of the 7th International Symp. Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pp. 2-11, Cambridge, MA 1996.

[4] T. Saeki, Y. Nakaoka, M. Fujita, A. Tanaka, K. Nagata, K. Sakakibara, T. Matano, Y. Hoshino, K. Miyano, S. Isa, E. Kakehashi, J. M. Drynan, M. Komuro, T. Fukase, H. Iwasaki, J. Sekine, M. Igeta, N. Nakanishi, T. Itani, K. Yoshida, H. Yoshino, S. Hashimoto, T. Yoshii, M. Ichinose, T. Imura, M. Uziie, K. Koyama, Y. Fukuzo, and T. Okuda, "A 2.5ns Clock Access 250MHz 256Mb SDRAM with a Synchronous Mirror Delay," *Digest of Technical Papers, 1996 IEEE International Solid State Circuits Conference*, pp. 374-375, 1996.

[5] J. Han, J. Lee, S. Yoon, S. Jeong, C. Park, I. Cho, S. Lee, and D. Seo, "Skew Minimization Techniques for 256M-bit Synchronous DRAM and beyond," *Digest of Technical Papers, 1996 Symp. on VLSI Circuits*, pp. 192-193, 1996.

[6] M. Rosenblum, S. Herrod, E. Witchel, and A. Gupta., "The SimOS approach," *IEEE Parallel and Distributed Technology*, vol.4, no. 3, 1995.

[7] B. Nayfeh, L. Hammond, and K. Olukotun, "Evaluation of Design Alternatives for a Multiprocessor Microprocessor," *Proceedings of the 23th International Symposium on Computer Architecture*, pp. 66-67, Philadelphia, PA 1996.

[8] R. Wilson, R. French, C. Wilson, S. Amarasinghe, J. Anderson, S. Tjiang, S.-W. Liao, C.-W. Tseng, M. Hall, M. Lam, and J. Hennessy, "The SUIF Compiler System: A Parallelizing and Optimizing Research Compiler," Stanford University Technical Report No. CSL-TR-94-620, May 1994.

[9] M. Asakura, T. Ohishi, M. Tsukude, S. Tomishima, H. Hidaka, K. Arimoto, K. Fujishima, T. Eimori, Y. Ohno, T. Nishimura, M. Yasunaga, T. Kondoh, S. Satoh, T. Yoshihara, and K. Demizu, "A 34ns 256Mb DRAM with Boosted Sense-Ground Scheme," *Digest of Technical Papers, 1994 IEEE International Solid State Circuits Conference*, pp. 140-141, San Francisco, CA 1994.

[10] K. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, vol. 16, No. 2, pp. 28-40, April 1996.

[11] T. Yamauchi, L. Hammond, and K. Olukotun, "A Single-Chip Multiprocessor Integrated with DRAM," presented at the "Mixing Logic and DRAM" workshop before ISCA–24, June 1, 1997. Available in PostScript format at http://iram.cs.berkeley.edu/isca97-workshop/