

MEMORY - IMPROVING MEMORY PERFORMANCE*

Charles Severance

Kevin Dowd

This work is produced by OpenStax-CNX and licensed under the Creative Commons Attribution License 3.0[†]

Given the importance, in the area of high performance computing, of the performance of a computer's memory subsystem, many techniques have been used to improve the performance of the memory systems of computers. The two attributes of memory system performance are generally *bandwidth* and *latency*. Some memory system design changes improve one at the expense of the other, and other improvements positively impact both bandwidth and latency. Bandwidth generally focuses on the best possible steady-state transfer rate of a memory system. Usually this is measured while running a long unit-stride loop reading or reading and writing memory.¹ Latency is a measure of the worst-case performance of a memory system as it moves a small amount of data such as a 32- or 64-bit word between the processor and memory. Both are important because they are an important part of most high performance applications.

Because memory systems are divided into components, there are different bandwidth and latency figures between different components as shown in Figure 1 (Simple Memory System). The bandwidth rate between a cache and the CPU will be higher than the bandwidth between main memory and the cache, for instance. There may be several caches and paths to memory as well. Usually, the peak memory bandwidth quoted by vendors is the speed between the data cache and the processor.

In the rest of this section, we look at techniques to improve latency, bandwidth, or both.

1 Large Caches

As we mentioned at the start of this chapter, the disparity between CPU speeds and memory is growing. If you look closely, you can see vendors innovating in several ways. Some workstations are being offered with 4- MB data caches! This is larger than the main memory systems of machines just a few years ago. With a large enough cache, a small (or even moderately large) data set can fit completely inside and get incredibly good performance. Watch out for this when you are testing new hardware. When your program grows too large for the cache, the performance may drop off considerably, perhaps by a factor of 10 or more, depending on the memory access patterns. Interestingly, an increase in cache size on the part of vendors can render a benchmark obsolete.

*Version 1.3: Aug 25, 2010 11:06 am -0500

[†]<http://creativecommons.org/licenses/by/3.0/>

¹See the STREAM section in Chapter 15 for measures of memory bandwidth.

Simple Memory System

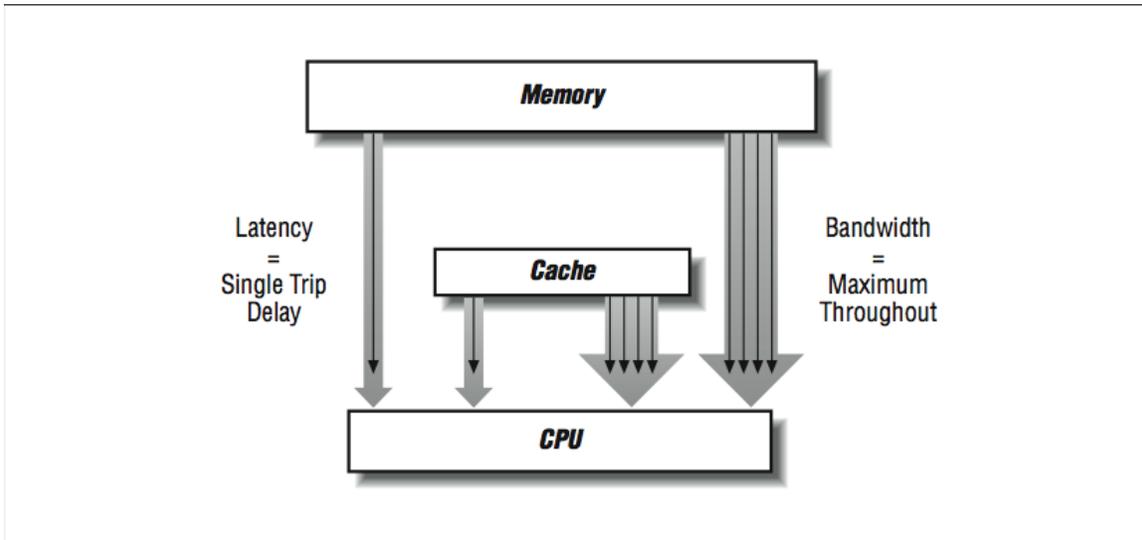


Figure 1

Up to 1992, the Linpack 100×100 benchmark was probably the single most-respected benchmark to determine the average performance across a wide range of applications. In 1992, IBM introduced the IBM RS-6000 which had a cache large enough to contain the entire 100×100 matrix for the duration of the benchmark. For the first time, a workstation had performance on this benchmark on the same order of supercomputers. In a sense, with the entire data structure in a SRAM cache, the RS-6000 was operating like a Cray vector supercomputer. The problem was that the Cray could maintain and improve the performance for a 120×120 matrix, whereas the RS-6000 suffered a significant performance loss at this increased matrix size. Soon, all the other workstation vendors introduced similarly large caches, and the 100×100 Linpack benchmark ceased to be useful as an indicator of average application performance.

2 Wider Memory Systems

Consider what happens when a cache line is refilled from memory: consecutive memory locations from main memory are read to fill consecutive locations within the cache line. The number of bytes transferred depends on how big the line is — anywhere from 16 bytes to 256 bytes or more. We want the refill to proceed quickly because an instruction is stalled in the pipeline, or perhaps the processor is waiting for more instructions. In Figure 2 (Narrow memory system), if we have two DRAM chips that provide us with 4 bits of data every 100 ns (remember cycle time), a cache fill of a 16-byte line takes 1600 ns.

Narrow memory system

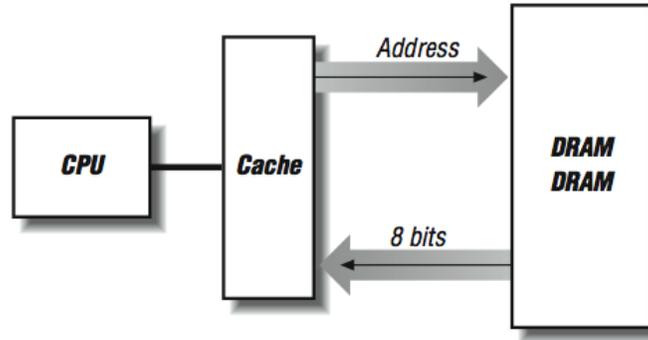


Figure 2

One way to make the cache-line fill operation faster is to “widen” the memory system as shown in Figure 3 (Wide memory system). Instead of having two rows of DRAMs, we create multiple rows of DRAMs. Now on every 100-ns cycle, we get 32 contiguous bits, and our cache-line fills are four times faster.

Wide memory system

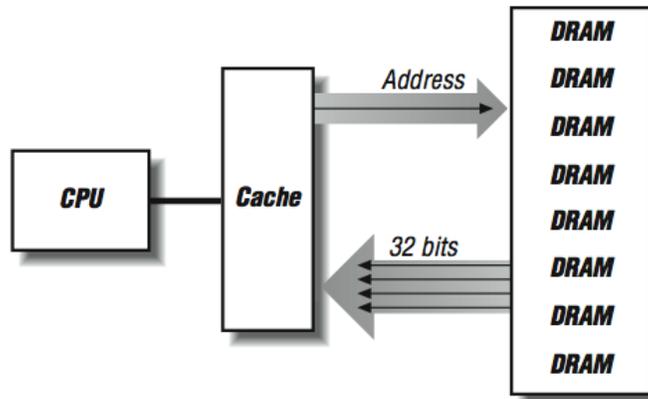


Figure 3

We can improve the performance of a memory system by increasing the width of the memory system up to the length of the cache line, at which time we can fill the entire line in a single memory cycle. On the SGI Power Challenge series of systems, the memory width is 256 bits. The downside of a wider memory system is that DRAMs must be added in multiples. In many modern workstations and personal computers, memory is expanded in the form of single inline memory modules (SIMMs). SIMMs currently are either 30-, 72-, or 168-pin modules, each of which is made up of several DRAM chips ready to be installed into a memory sub-system.

3 Bypassing Cache

It's interesting that we have spent nearly an entire chapter on how great a cache is for high performance computers, and now we are going to bypass the cache to improve performance. As mentioned earlier, some types of processing result in non-unit strides (or bouncing around) through memory. These types of memory reference patterns bring out the worst-case behavior in cache-based architectures. It is these reference patterns that see improved performance by bypassing the cache. Inability to support these types of computations remains an area where traditional supercomputers can significantly outperform high-speed RISC processors. For this reason, RISC processors that are serious about number crunching may have special instructions that bypass data cache memory; the data are transferred directly between the processor and the main memory system.² In Figure 4 (Bypassing cache) we have four banks of SIMMs that can do cache fills at 128 bits per 100 ns memory cycle. Remember that the data is available after 50 ns but we can't get more data until the DRAMs refresh 50–60 ns later. However, if we are doing 32-bit non-unit- stride loads and have the capability to bypass cache, each load will be satisfied from one of the four SIMMs in 50 ns. While that SIMM refreshed, another load can occur from any of the other three SIMMs in 50 ns. In a random mix of non-unit loads there is a 75% chance that the next load will fall on a “fresh” DRAM. If the load falls on a bank while it is refreshing, it simply has to wait until the refresh completes.

A further advantage of bypassing cache is that the data doesn't need to be moved through the SRAM cache. This operation can add from 10–50 ns to the load time for a single word. This also avoids invalidating the contents of an entire cache line in the cache.

Adding cache bypass, increasing memory-system widths, and adding banks increases the cost of a memory system. Computer-system vendors make an economic choice as to how many of these techniques they need to apply to get sufficient performance for their particular processor and system. Hence, as processor speed increases, vendors must add more of these memory system features to their commodity systems to maintain a balance between processor and memory-system speed.

²By the way, most machines have uncached memory spaces for process synchronization and I/O device registers. However, memory references to these locations bypass the cache because of the address chosen, not necessarily because of the instruction chosen.

Bypassing cache

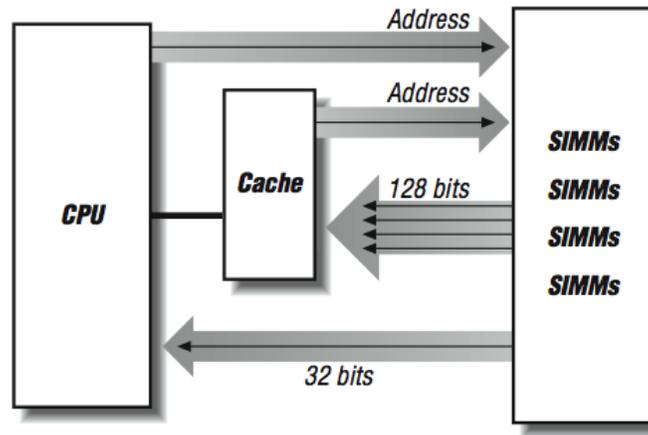


Figure 4

4 Interleaved and Pipelined Memory Systems

Vector supercomputers, such as the CRAY Y/MP and the Convex C3, are machines that depend on multi-banked memory systems for performance. The C3, in particular, has a memory system with up to 256-way interleaving. Each interleave (or bank) is 64 bits wide. This is an expensive memory system to build, but it has some very nice performance characteristics. Having a large number of banks helps to reduce the chances of repeated access to the same memory bank. If you do hit the same bank twice in a row, however, the penalty is a delay of nearly 300 ns — a long time for a machine with a clock speed of 16 ns. So when things go well, they go very well.

However, having a large number of banks alone is not sufficient to feed a 16-ns processor using 50 ns DRAM. In addition to interleaving, the memory subsystem also needs to be pipelined. That is, the CPU must begin the second, third, and fourth load before the CPU has received the results of the first load as shown in Figure 5 (Multibanked memory system). Then each time it receives the results from bank “n,” it must start the load from bank “n+4” to keep the pipeline fed. This way, after a brief startup delay, loads complete every 16 ns and so the memory system appears to operate at the clock rate of the CPU. This pipelined memory approach is facilitated by the 128-element vector registers in the C3 processor.

Using gather/scatter hardware, non-unit-stride operations can also be pipelined. The only difference for non-unit-stride operations is that the banks are not accessed in sequential order. With a random pattern of memory references, it's possible to reaccess a memory bank before it has completely refreshed from a previous access. This is called a *bank stall*.

Multibanked memory system

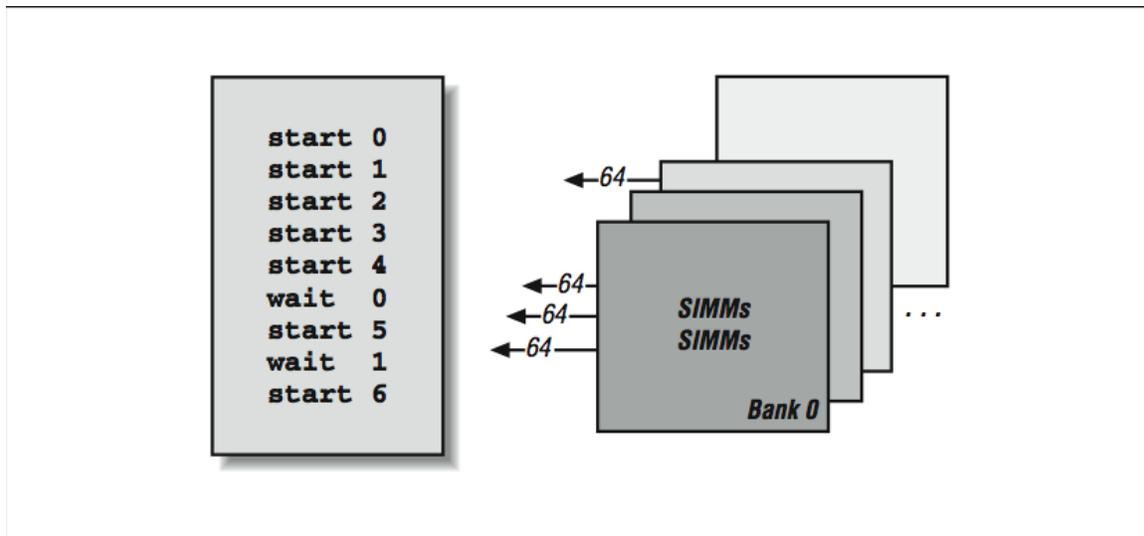


Figure 5

Different access patterns are subject to bank stalls of varying severity. For instance, accesses to every fourth word in an eight-bank memory system would also be subject to bank stalls, though the recovery would occur sooner. References to every second word might not experience bank stalls at all; each bank may have recovered by the time its next reference comes around; it depends on the relative speeds of the processor and memory system. Irregular access patterns are sure to encounter some bank stalls.

In addition to the bank stall hazard, single-word references made directly to a multibanked memory system carry a greater latency than those of (successfully) cached memory accesses. This is because references are going out to memory that is slower than cache, and there may be additional address translation steps as well. However, banked memory references are pipelined. As long as references are started well enough in advance, several pipelined, multibanked references can be in flight at one time, giving you good throughput.

The CDC-205 system performed vector operations in a memory-to-memory fashion using a set of explicit memory pipelines. This system had superior performance for very long unit-stride vector computations. A single instruction could perform 65,000 computations using three memory pipes.

5 Software Managed Caches

Here's an interesting thought: if a vector processor can plan far enough in advance to start a memory pipe, why can't a RISC processor start a cache-fill before it really needs the data in those same situations? In a way, this is priming the cache to hide the latency of the cache-fill. If this could be done far enough in advance, it would appear that all memory references would operate at the speed of the cache.

This concept is called prefetching and it is supported using a special prefetch instruction available on many RISC processors. A prefetch instruction operates just like a standard load instruction, except that the processor doesn't wait for the cache to fill before the instruction completes. The idea is to prefetch far enough ahead of the computation to have the data ready in cache by the time the actual computation occurs. The following is an example of how this might be used:

```

DO I=1,1000000,8
  PREFETCH(ARR(I+8))
  DO J=0,7
    SUM=SUM+ARR(I+J)
  END DO
END DO

```

This is not the actual FORTRAN. Prefetching is usually done in the assembly code generated by the compiler when it detects that you are stepping through the array using a fixed stride. The compiler typically estimate how far ahead you should be prefetching. In the above example, if the cache-fills were particularly slow, the value 8 in I+8 could be changed to 16 or 32 while the other values changed accordingly.

In a processor that could only issue one instruction per cycle, there might be no payback to a prefetch instruction; it would take up valuable time in the instruction stream in exchange for an uncertain benefit. On a superscalar processor, however, a cache hint could be mixed in with the rest of the instruction stream and issued alongside other, real instructions. If it saved your program from suffering extra cache misses, it would be worth having.

6 Post-RISC Effects on Memory References

Memory operations typically access the memory during the execute phase of the pipeline on a RISC processor. On the post-RISC processor, things are no different than on a RISC processor except that many loads can be half finished at any given moment. On some current processors, up to 28 memory operations may be active with 10 waiting for off-chip memory to arrive. This is an excellent way to compensate for slow memory latency compared to the CPU speed. Consider the following loop:

	LOADI	R6,10000	Set the Iterations
	LOADI	R5,0	Set the index variable
LOOP:	LOAD	R1,R2(R5)	Load a value from memory
	INCR	R1	Add one to R1
	STORE	R1,R3(R5)	Store the incremented value back to memory
	INCR	R5	Add one to R5
	COMPARE	R5,R6	Check for loop termination
	BLT	LOOP	Branch if R5 < R6 back to LOOP

In this example, assume that it take 50 cycles to access memory. When the fetch/ decode puts the first load into the instruction reorder buffer (IRB), the load starts on the next cycle and then is suspended in the execute phase. However, the rest of the instructions are in the IRB. The INCR R1 must wait for the load and the STORE must also wait. However, by using a rename register, the INCR R5, COMPARE, and BLT can all be computed, and the fetch/decode goes up to the top of the loop and sends another load into the IRB for the next memory location that will have to wait. This looping continues until about 10 iterations of the loop are in the IRB. Then the first load actually shows up from memory and the INCR R1 and STORE from the first iteration begins executing. Of course the store takes a while, but about that time the second load finishes, so there is more work to do and so on...

Like many aspects of computing, the post-RISC architecture, with its out-of-order and speculative execution, optimizes memory references. The post-RISC processor dynamically unrolls loops at execution time to compensate for memory subsystem delay. Assuming a pipelined multibanked memory system that can have

multiple memory operations started before any complete (the HP PA-8000 can have 10 off-chip memory operations in flight at one time), the processor continues to dispatch memory operations until those operations begin to complete.

Unlike a vector processor or a prefetch instruction, the post-RISC processor does not need to anticipate the precise pattern of memory references so it can carefully control the memory subsystem. As a result, the post-RISC processor can achieve peak performance in a far-wider range of code sequences than either vector processors or in-order RISC processors with prefetch capability.

This implicit tolerance to memory latency makes the post-RISC processors ideal for use in the scalable shared-memory processors of the future, where the memory hierarchy will become even more complex than current processors with three levels of cache and a main memory.

Unfortunately, the one code segment that doesn't benefit significantly from the post-RISC architecture is the linked-list traversal. This is because the next address is never known until the previous load is completed so all loads are fundamentally serialized.

7 Dynamic RAM Technology Trends

Much of the techniques in this section have focused on how to deal with the imperfections of the dynamic RAM chip (although when your clock rate hits 300–600 MHz or 3–2 ns, even SRAM starts to look pretty slow). It's clear that the demand for more and more RAM will continue to increase, and gigabits and more DRAM will fit on a single chip. Because of this, significant work is underway to make new super-DRAMs faster and more tuned to the extremely fast processors of the present and the future. Some of the technologies are relatively straightforward, and others require a major redesign of the way that processors and memories are manufactured.

Some DRAM improvements include:

- Fast page mode DRAM
- Extended data out RAM (EDO RAM)
- Synchronous DRAM (SDRAM)
- RAMBUS
- Cached DRAM (CDRAM)

Fast *page mode* DRAM saves time by allowing a mode in which the entire address doesn't have to be re-clocked into the chip for each memory operation. Instead, there is an assumption that the memory will be accessed sequentially (as in a cache-line fill), and only the low-order bits of the address are clocked in for successive reads or writes.

EDO RAM is a modification to output buffering on page mode RAM that allows it to operate roughly twice as quickly for operations other than refresh.

Synchronous DRAM is synchronized using an external clock that allows the cache and the DRAM to coordinate their operations. Also, SDRAM can pipeline the retrieval of multiple memory bits to improve overall throughput.

RAMBUS is a proprietary technology capable of 500 MB/sec data transfer. RAMBUS uses significant logic within the chip and operates at higher power levels than typical DRAM.

Cached DRAM combines a SRAM cache on the same chip as the DRAM. This tightly couples the SRAM and DRAM and provides performance similar to SRAM devices with all the limitations of any cache architecture. One advantage of the CDRAM approach is that the amount of cache is increased as the amount of DRAM is increased. Also when dealing with memory systems with a large number of interleaves, each interleave has its own SRAM to reduce latency, assuming the data requested was in the SRAM.

An even more advanced approach is to integrate the processor, SRAM, and DRAM onto a single chip clocked at say 5 GHz, containing 128 MB of data. Understandably, there is a wide range of technical problems to solve before this type of component is widely available for \$200 — but it's not out of the question. The manufacturing processes for DRAM and processors are already beginning to converge in some

ways (RAMBUS). The biggest performance problem when we have this type of system will be, “What to do if you need 160 MB?”