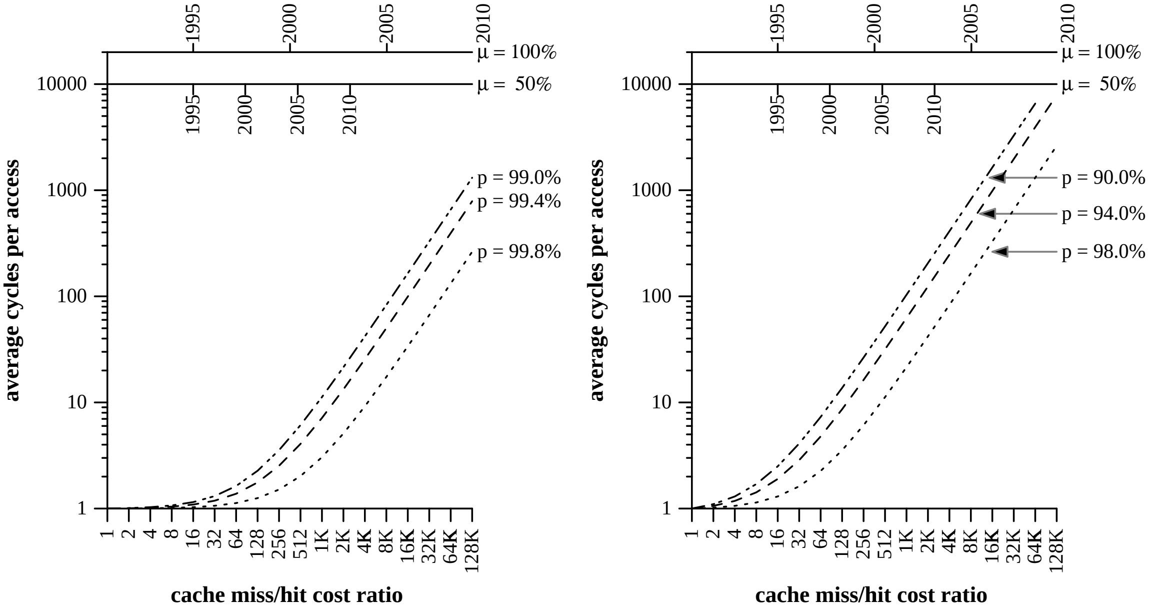
References

[Bas91] F. Baskett, Keynote address. International Symposium on Shared Memory Multiprocessing, April 1991.

[Hen90] J.L. Hennessy and D.A. Patterson, *Computer Architecture: a Quantitative Approach*, Morgan-Kaufman, San Mateo, CA, 1990.

[McK94] S.A. McKee, et. al., “Experimental Implementation of Dynamic Access Ordering”, Proc. 27th Hawaii International Conference on System Sciences, Maui, HI, January 1994.

[McK94a] S.A. McKee, et. al., “Increasing Memory Bandwidth for Vector Computations”, Proc. Conference on Programming Languages and System Architecture, Zurich, March 1994.



(a)

(b)

Figure 2 Trends for a Current Cache Miss/Hit Cost Ratio of 16

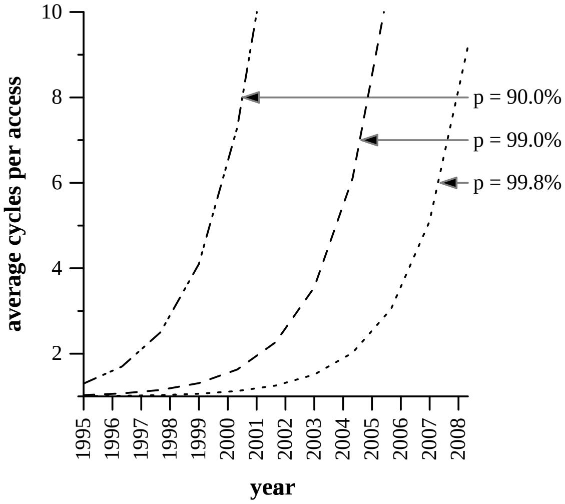


Figure 3 Average Access Cost for 80% Annual Increase in Processor Performance

As noted above, the right solution to the problem of the memory wall is probably something that we haven’t thought of — but we would like to see the discussion engaged. It would appear that we do not have a great deal of time.

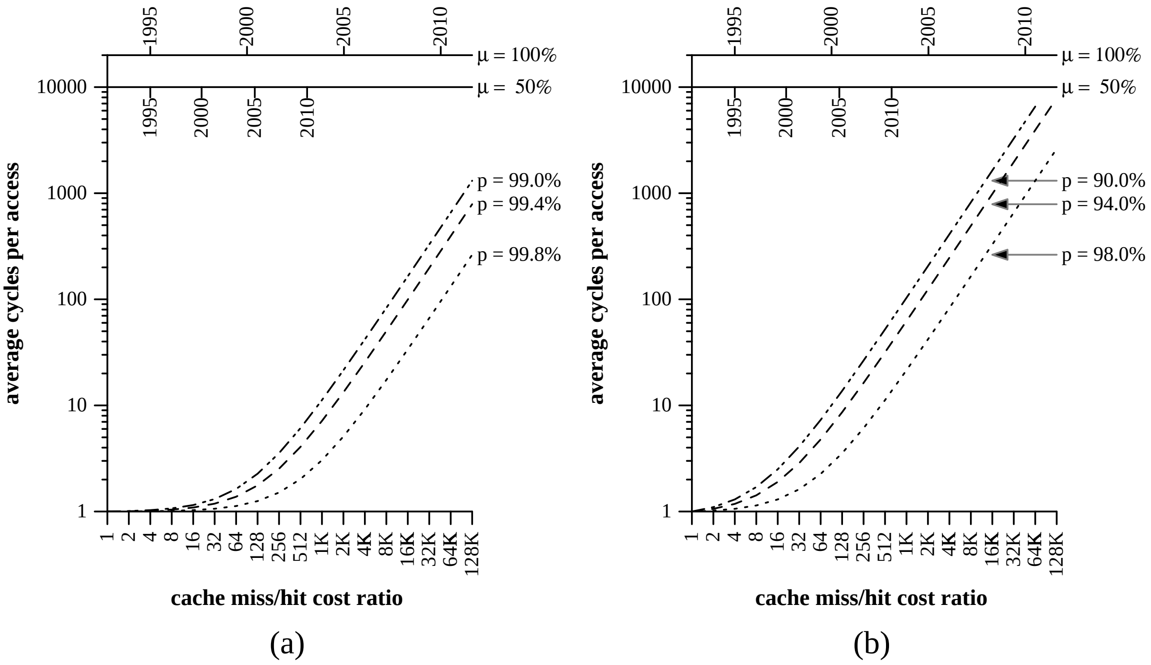


Figure 1 Trends for a Current Cache Miss/Hit Cost Ratio of 4

Our prediction of the memory wall is probably wrong too — but it suggests that we have to start thinking “out of the box”. All the techniques that the authors are aware of, including ones we have proposed [McK94, McK94a], provide one-time boosts to either bandwidth or latency. While these delay the date of impact, they don’t change the fundamentals.

The most “convenient” resolution to the problem would be the discovery of a cool, dense memory technology whose speed scales with that of processors. We are not aware of any such technology and could not affect its development in any case; the only contribution we can make is to look for architectural solutions. These are probably all bogus, but the discussion must start somewhere:

- Can we drive the number of compulsory misses to zero? If we can’t fix t_m, then the only way to make caches work is to drive p to 100% — which means eliminating the compulsory misses. If all data were initialized dynamically, for example, possibly the compiler could generate special “first write” instructions. It is harder for us to imagine how to drive the compulsory misses for code to zero.
- Is it time to forgo the model that access time is uniform to all parts of the address space? It is false for DSM and other scalable multiprocessor schemes, so why not for single processors as well? If we do this, can the compiler explicitly manage a smaller amount of higher speed memory?
- Are there any new ideas for how to trade computation for storage? Alternatively, can we trade space for speed? DRAM keeps giving us plenty of the former.
- Ancient machines like the IBM 650 and Burroughs 205 used magnetic drum as primary memory and had clever schemes for reducing rotational latency to essentially zero — can we borrow a page from either of those books?

Figures 1-3 explore various possibilities, showing projected trends for a set of perfect or near-perfect caches. All our graphs assume that DRAM performance continues to increase at an annual rate of 7%. The horizontal axis is various cpu/DRAM performance ratios, and the lines at the top indicate the dates these ratios occur if microprocessor performance (μ) increases at rates of 50% and 100% respectively. Figure 1 assumes that cache misses are currently 4 times slower than hits; Figure 1(a) considers compulsory cache miss rates of less than 1% while Figure 1(b) shows the same trends for caches with more realistic miss rates of 2-10%. Figures 2 is a counterpart of Figure 1, but assumes that the current cost of a cache miss is 16 times that of a hit.

Figure 3 provides a closer look at the expected impact on average memory access time for one particular value of μ, Baskett's estimated 80%. Even if we assume a cache hit rate of 99.8% and use the more conservative cache miss cost of 4 cycles as our starting point, performance hits the 5-cycles-per-access wall in 11-12 years. At a hit rate of 99% we hit the same wall within the decade, and at 90%, within 5 years.

Note that changing the starting point — the "current" miss/hit cost ratio — and the cache miss rates *don't change the trends*: if the microprocessor/memory performance gap continues to grow at a similar rate, in 10-15 years each memory access will cost, on average, tens or even hundreds of processor cycles. Under each scenario, system speed is dominated by memory performance.

Over the past thirty years there have been several predictions of the eminent cessation of the rate of improvement in computer performance. Every such prediction was wrong. They were wrong because they hinged on unstated assumptions that were overturned by subsequent events. So, for example, the failure to foresee the move from discrete components to integrated circuits led to a prediction that the speed of light would limit computer speeds to several orders of magnitude slower than they are now.

First let's assume that the cache speed matches that of the processor, and specifically that it scales with the processor speed. This is certainly true for on-chip cache, and allows us to easily normalize all our results in terms of instruction cycle times (essentially saying $t_c = 1$ cpu cycle). Second, assume that the cache is perfect. That is, the cache never has a conflict or capacity miss; the only misses are the compulsory ones. Thus $(1 - p)$ is just the probability of accessing a location that has never been referenced before (one can quibble and adjust this for line size, but this won't affect the conclusion, so we won't make the argument more complicated than necessary).

Now, although $(1 - p)$ is small, it isn't zero. Therefore as t_c and t_m diverge, t_{avg} will grow and system performance will degrade. In fact, it will hit a wall.

In most programs, 20-40% of the instructions reference memory [Hen90]. For the sake of argument let's take the lower number, 20%. That means that, on average, during execution every 5th instruction references memory. We will hit the wall when t_{avg} exceeds 5 instruction times. At that point system performance is totally determined by memory speed; making the processor faster won't affect the wall-clock time to complete an application.

Alas, there is no easy way out of this. We have already assumed a perfect cache, so a bigger/smarter one won't help.We're already using the full bandwidth of the memory, so prefetching or other related schemes won't help either. We can consider other things that might be done, but first let's speculate on when we might hit the wall.

Assume the compulsory miss rate is 1% or less [Hen90] and that the next level of the memory hierarchy is currently four times slower than cache. If we assume that DRAM speeds increase by 7% per year [Hen90] and use Baskett's estimate that microprocessor performance is increasing at the rate of 80% per year [Bas91], the average number of cycles per memory access will be 1.52 in 2000, 8.25 in 2005, and 98.8 in 2010. Under these assumptions, the wall is less than a decade away.

Hitting the Memory Wall: Implications of the Obvious

Wm. A. Wulf
Sally A. McKee
Department of Computer Science
University of Virginia
{wulf | mckee}@virginia.edu

December 1994

This brief note points out something obvious — something the authors "knew" without really understanding. With apologies to those who did understand, we offer it to those others who, like us, missed the point.

We all know that the rate of improvement in microprocessor speed exceeds the rate of improvement in DRAM memory speed — each is improving exponentially, but the exponent for microprocessors is substantially larger than that for DRAMs. The difference between diverging exponentials also grows exponentially; so, although the disparity between processor and memory speed is already an issue, downstream someplace it will be a much bigger one. How big and how soon? The answers to these questions are what the authors had failed to appreciate.

To get a handle on the answers, consider an old friend — the equation for the average time to access memory, where t_c and t_m are the cache and DRAM access times and p is the probability of a cache hit:

$$t_{avg} = p \times t_c + (1 - p) \times t_m$$

We want to look at how the average access time changes with technology, so we'll make some conservative assumptions; as you'll see, the specific values won't change the basic conclusion of this note, namely that we are going to hit a wall in the improvement of system performance unless something *basic* changes.

Appeared in Computer Architecture News, 23(1):20-24, March 1995.

Hitting the Memory Wall: Implications of the Obvious

Wm. A. Wulf and Sally A. McKee
{wulf | mckee}@virginia.edu

Computer Science Report No. CS-94-48
December, 1994

Appeared in Computer Architecture News, 23(1):20-24, March 1995