Reflections on the Memory Wall*

Sally A. McKee
Computer Systems Laboratory
Cornell University
Ithaca, New York
sam@csl.cornell.edu

ABSTRACT

This paper looks at the evolution of the "Memory Wall" problem over the past decade. It begins by reviewing the short Computer Architecture News note that coined the phrase, including the motivation behind the note, the context in which it was written, and the controversy it sparked. What has changed over the years? Are we hitting the Memory Wall? And if so, for what types of applications?

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Performance Attributes; B.3.3 [**Memory Structures**]: Performance Analysis and Design Aids

General Terms

Performance

Keywords

memory performance, system balance

1. INTRODUCTION

The year is 1991. Researchers at NASA Ames Research Center and at the University of Virginia's Institute for Parallel Computation (also supported by NASA) are trying to understand the performance of scientific computations on Intel's iPSC/860 hypercube, the state of the art in message-passing multicomputer. Lee [6] at NASA and Moyer [14] at UVa find that performance of even highly optimized, hand-coded scientific kernels can be up to orders of magnitude below peak. The reason: an imbalance between memory bandwidth and processor speed in the iPSC/860 node architecture.

Two years pass. Lee has developed the NAS860 library of routines modeled after Cray instructions [7] and Moyer has developed compiler algorithms to statically reorder memory accesses [15]; both use nonintuitive optimizations to make better use of the node memory system.

Processor speeds keep rising dramatically — at approximately 75% per year. Issue widths rise. DRAM speeds increase at a steady, paltry 7% per year. Given the technology trends, the results of Moyer's research, and the challenge of designing a memory system for a novel architecture called WM [21] (whose ISA included instructions to stream data to and from memory), I begin working with my advisor, Bill Wulf, to investigate the design of high-performance memory controllers to help software take better advantage of lower-level memory system resources.

The mainstream computer architecture community is still largely focused on increasing processor performance. Wide-issue architectures, latency-tolerance techniques such as prefetching and nonblocking caches, and the promise of the announced Rambus [4] memory interface technology convince most computer architects that solving our memory problems largely amounts to building larger caches and more latency tolerance into our chips. When I talk to colleagues about my work, they tend to dismiss it as unimportant.

In response to this, Wulf suggests that we write a short note that will compel people to think about the memory problem in a different light. The note would be submitted to Computer Architecture News, given its wide readership. I am dubious, but I produce the data, Wulf writes most of the text, and we send the article out. Wulf assures me "Just wait. This will be referenced a lot." I, the graduate student, am still dubious.

Wulf was right.

2. THE NOTE

Given the brevity of the original note [20], the time that has passed since its publication, and the fact that (though apparently widely read by Computer Architecture News subscribers) many may not have seen it, I quote it here (with minor modifications and annotations for readability) for the reader's convenience.

*This work is based on an earlier work: *Hitting the Memory Wall: Implications of the Obvious*, in ACM SIGARCH Computer Architecture News, 23, ISSN:0163-5964 , March 1995 ACM, 1995. http://doi.acm.org/10.1145/216585.216588

CF'04, April 14–16, 2004, Ischia, Italy.

Hitting the Memory Wall: Implications of the Obvious

Wm.A. Wulf and Sally A. McKee

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.*

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 [5]. For the sake of argument let's take the lower number, 20%. That means that, on average, during execution every fifth instruction references memory. We will hit the wall when t_{avg} exceeds five 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 [5] 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 [5] and use Baskett's estimate that microprocessor performance is increasing at the rate of 80% per year [2], 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.

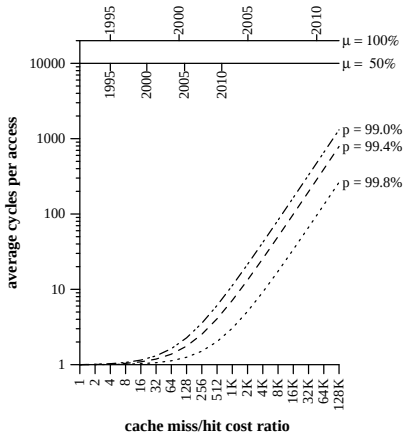

Figure 1: Trends for a Cache Miss/Hit Ratio of 4 (in 1994) and High Hit Rates

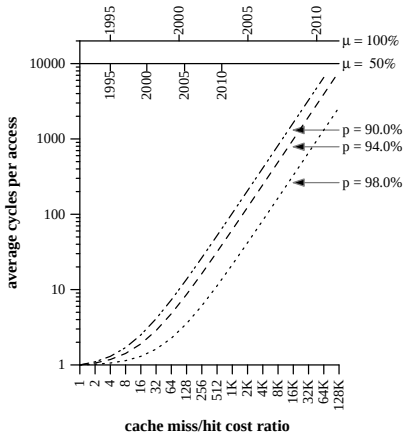

Figure 2: Trends for a Cache Miss/Hit Ratio of 4 (in 1994) and Lower Hit Rates

Figures 1-Figure 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 and Figure 2 assume that cache misses are currently four times slower than hits; Figure 1 considers compulsory cache miss rates of less than 1% while Figure 2 shows the same trends for caches with more realistic miss rates of 2-10%. Figure 3 is a counterpart of Figure 1, but assumes that the current cost of a cache miss is 16 times that of a hit.

Figure 4 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

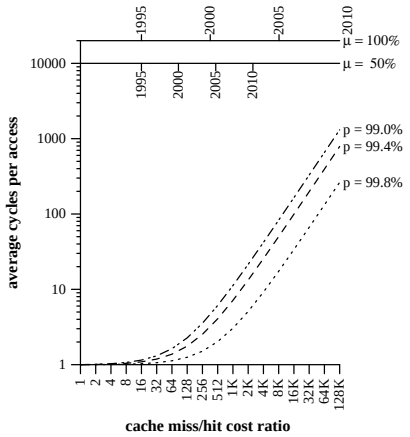

Figure 3: Trends for a Cache Miss/Hit Ratio of 16 (in 1994) and High Hit Rates

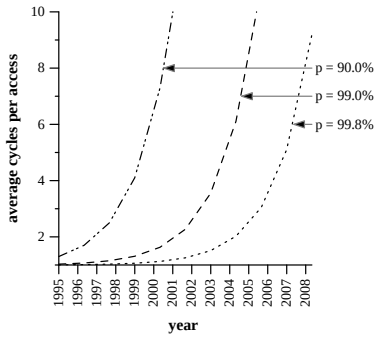

Figure 4: Average Access Cost for 80% Increase in Processor Performance

four cycles as our starting point, performance hits the five-cycles-per-access wall in 11-12 years. At a hit rate of 99and at 90%, within five years.

Note that changing the starting point — the "current" miss/hit cost ratio — and the cache miss rates doesn'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.

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 [11, 12], 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?*

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.

3. THE WRITING ON THE WALL

We were certainly not the first people to recognize an impending problem. A 1989 Digital Equipment Corporation Western Research Lab Report and subsequent Usenix Summer Conference publication by John Ousterhout examine the performance of a set of OS microbenchmarks, and discuss the hardware and software ramifications of the results. Ousterhout concludes that "The first hardware related issue is memory bandwidth: ... it is not keeping up with CPU speed. ... If memory bandwidth does not improve dramatically in future machines, some classes of applications may be limited by memory performance." [16] This article reached the OS community, but apparently few members of the Computer Architecture community.

When we published our note, it did indeed incite people to think about the problem. Many people were eager to argue with me about why we were wrong (even though we had said "we're probably wrong" in the text of the note). One person even argued with me on the public transportation system in Stockholm, Sweden. Obviously, we struck a nerve.

Many responses appear in the comp.arch newsgroup; they are well thought out, and consider the argument in much greater depth than we had (intentionally) addressed it in our note. For instance, our increases in "processor performance" did not distinguish between architectural improvements and fabrication process improvements. A recent search of the ACM Digital Library yields a set of 59 references to papers

containing the phrase "memory wall". Several of these are also notes that appeared in Computer Architecture News. For instance, Wilkes argues that we will meet the "wall" of CMOS process limitations before we hit a memory wall [19].

In 1995, McCalpin defines "machine balance", and states that "steps need to be taken soon to increase memory bandwidth more rapidly. ... merely increasing bus width and decreasing latency will not be adequate." [8] Another 1996 article by Burger, Goodman, and Kägi shows that current caches are thousands of times larger than necessary for optimal cache, and are less than 20% efficient — most data in L2 caches is dead at any given time. [3]

In a 1996 position piece entitled "It's The Memory, Stupid!" Richard Sites states that "across the industry, today's chips are largely able to execute code faster than we can feed them with instructions and data. ... The real design action is in the memory subsystems — caches, buses, bandwidth and latency." [18]. In an article published the same year at the Second USENIX Symposium on Operating System Design and Implementation, Perl and Sites conclude that "processor bandwidth can be a first-order bottleneck to achieving good performance. This is particularly apparent when studying commercial benchmarks. Operating system code and data structures contribute disproportionately to the memory access load." [17]

In 2000, 2001, and 2002, workshops held at the International Symposium on Computer Architecture address "Overcoming the Memory Wall" and "Memory Performance Issues".

4. SOME TRENDS

One of the best documented indicators of memory performance has been McCalpin's STREAM benchmark suite [9]. The STREAM benchmark measures bandwidth sustainable by ordinary user programs, and not the theoretical "peak bandwidth" that vendors advertise. The benchmark became the de facto standard, and vendors began to pay attention to it. Most have worked hard to provide a reasonable amount of memory bandwidth, but at an ever-increasing latency. See the official STREAM webpage (www.cs.virginia.edu/stream) for the latest numbers for uniprocessor and shared memory machines; the following figures come from that page. Figures 5-Figure 7 show CPU speeds versus bandwidth trends, and MFLOPS over time. The first of these, Figure 5, illustrates general machine imbalance trends. MFLOPS have been increasing at a greater rate than bandwidth: most machines have been increasing MFLOPS by about 50% per year (and the highest performing machine at only 18% per year), whereas bandwidth has generally be increasing at about 35% per year (and the highest performing machine at only 15%). Alan Charlesworth of Sun Microsystems produced a graph of historical bandwidth trends, shown in Figure 8.

Unfortunately, we have not tracked latency numbers as closely as bandwidth numbers. Statistics [10] indicate that for the IBM RS/6000 line based on the POWER family, the extremes of lmbench [13] latency were 11 processor cycles (22 FLOPS) in 1990 (20 MHz RS/6000-320 based on the original POWER implementation) and 425 processor cycles (1700 FLOPS) in 2003 (1700 MHz pSeries 655 based on the POWER4+ processor). That is 85× in frequency, or about 40× in raw latency, and about 80× in "equivalent FLOPS", in a 13-year interval. The quad-CPU p655+ costs about twice as much per processor as the RS/6000-320 did in 1990.

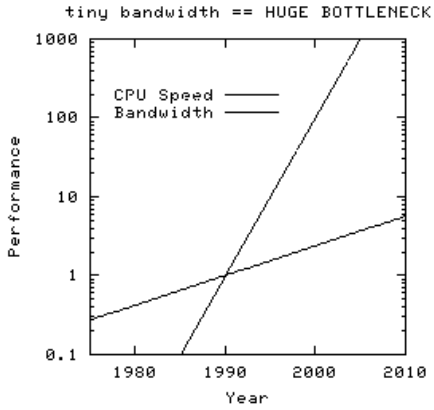

Figure 5: McCalpin's CPU Speed vs. Bandwidth

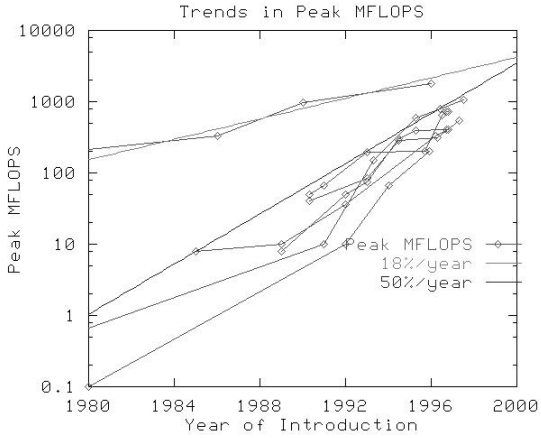

Figure 6: McCalpin's Peak MFLOPS over Time

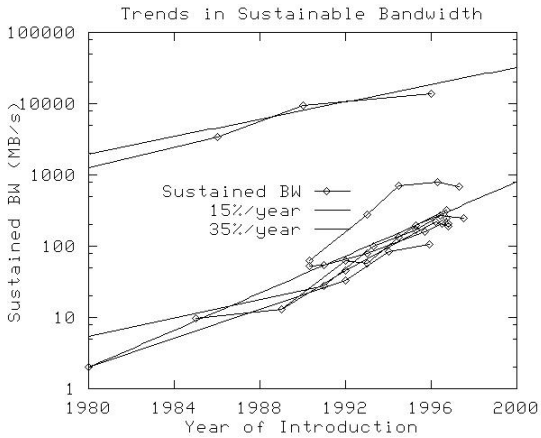

Figure 7: McCalpin's STREAM Bandwidth Trends over Time

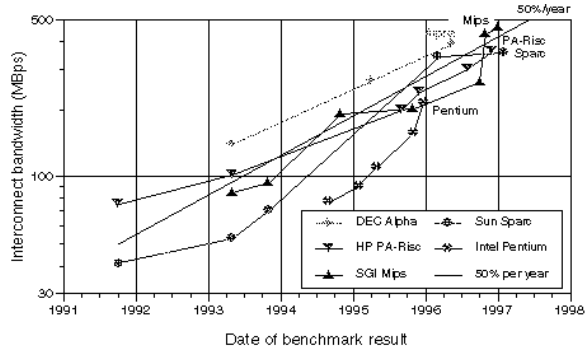

Figure 8: Charlesworth's Sustained Bandwidth Trends from the '90s

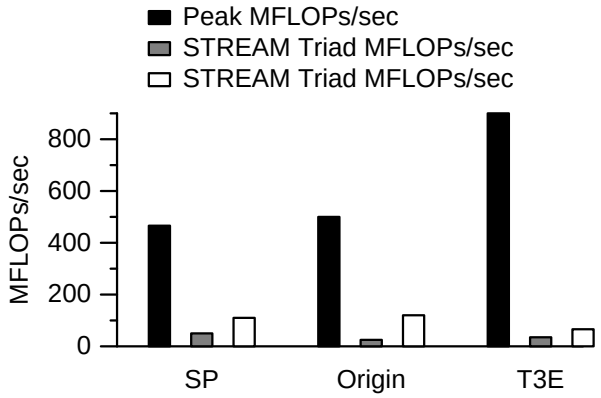

Figure 9: Observed Performance vs. Peak Performance (Sequential Performance of PTSCc-FUN3D for a Coarse Mesh of 22,677 Vertices [with Four Unknowns per Vertex])

5. WHERE IS THE WALL?

Many types of applications have not yet observed a memory wall problem. For instance, multimedia applications (especially for platforms with ISA extensions) have tended to improve in performance with improvements in processor speeds. On the other hand, commercial applications such as transaction processing workloads see 65% node idle times, and high-performance scientific computations see 95% node idle times: much of this is due to memory bottlenecks.

Anderson et al.'s data from Supercomputing 1999 [1], shown in Figure 9, drive home the problem: as peak MFLOPS rise from one platform to another, STREAM performance drops, and so does observed MFLOPS.

Are we hitting the memory wall? Yes, for many types of applications, but certainly not for all types. Can we avoid or postpone hitting the wall for the other types? Can we alleviate the problem for those applications that have already hit the wall? Probably. How? That is an open question.

6. ACKNOWLEDGMENTS

The author thanks Bill Wulf for having the wisdom to begin this discussion that has spanned most of a decade. Thanks go to Mateo Valero and Stamatis Vassiliadis for organizing the special session on the memory wall. John McCalpin and Bronis de Supinski participated in interesting discussions, both philosophical and quantitative. Ultimately, it has been the comp.arch readers, the SIGARCH Computer Architecture News readers, the members of the community — students and researchers — who have kept this discussion alive, and who keep asking the questions that we thought we knew the answers to, but whose answers keep changing.

7. REFERENCES

[1] W. Anderson, W. Gropp, D. Kaushik, D. Keyes, and B. Smith. Achieving high sustained performance in an unstructured mesh cfd application. In *Proceedings of Supercomputing '99*, Nov. 1999.

[2] F. Baskett. Keynote address. In *International Symposium on Shared Memory Multiprocessing*, Apr. 1991.

[3] D. Burger, J. Goodman, and A. Kägi. The declining effectiveness of dynamic caching for general-purpose microprocessors. Technical Report CS-TR-95-1261, University of Wisconsin – Madison, Jan. 1995.

[4] R. Crisp. Direct rambus technology: The new main memory standard. *IEEE Micro*, 17(6):18–28, November/December 1997.

[5] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., first edition, 1990.

[6] K. Lee. Achieving high performance on the i860 microprocessor. Technical Report NAS Technical Report RNR-91-029, NASA Ames Research Center, Oct. 1991.

[7] K. Lee. The NAS860 library user's manual. Technical Report NAS Technical Report RND-93-003, NASA Ames Research Center, Mar. 1993.

[8] J. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, Dec. 1995.

[9] J. McCalpin. Stream: Sustainable memory bandwidth in high performance computers. http://www.cs.virginia.edu/stream/, 1999.

[10] J. McCalpin. Personal Communication, Jan. 2004.

[11] S. McKee, R. Klenke, A. Schwab, W. Wulf, S. Moyer, C. Hitchcock, and J. Aylor. Experimental implementation of dynamic access ordering. In *Proceedings of the IEEE 27th Hawaii International Conference on Systems Sciences*, pages 431–440, Jan. 1994.

[12] S. McKee, S. Moyer, W. Wulf, and C. Hitchcock. Increasing memory bandwidth for vector computations. In *Proceedings of International Conferences on Programming Languages and System Architectures, Lecture Notes in Computer Science 782*, pages 87–104, Mar. 1994.

[13] L. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. In *Proc. 1996 USENIX Technical Conference*, pages 279–295, Jan. 1996.

[14] S. Moyer. Performance of the ipsc/860 node architecture. Technical Report IPC-TR-91-007, Institute for Parallel Computation, University of Virginia, 1991.

[15] S. Moyer. *Access Ordering Algorithms and Effective Memory Bandwidth*. PhD thesis, University of Virginia, May 1993.

[16] J. Ousterhout. Why Aren't Operating Systems Geting Faster As Fast as Hardware? In *USENIX Summer Conference*, June 1990.

[17] S. E. Perl and R. Sites. Studies of Windows NT performance using dynamic execution traces. In *Proceedings of the Second Symposium on Operating System Design and Implementation*, pages 169–184, Oct. 1996.

[18] R. Sites. It's the Memory, Stupid! *Microprocessor Report*, 10(10):2–3, Aug. 1006.

[19] M. Wilkes. The memory wall and the CMOS end-point. *ACM SIGArch Computer Architecture News*, 1995.

[20] W. Wulf and S. McKee. Hitting the wall: Implications of the obvious. *ACM SIGArch Computer Architecture News*, 23(1):20–24, Mar. 1995.

[21] W. Wulf. Evaluation of the WM architecture. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 382–390, May 1992.