The Processor-Memory bottleneck: Problems and Solutions.

by Nihar R. Mahapatra *and* Balakrishna Venkatrao

Abstract

The rate of improvement in microprocessor speed exceeds the rate of improvement in DRAM (Dynamic Random Access Memory) speed. So although the disparity between processor and memory speed is already an issue, downstream someplace it will be a much bigger one. Hence computer designers are faced with an increasing **Processor - Memory Performance Gap** [1] , which now is the primary obstacle to improved computer system performance. This article examines this problem as well as its various solutions.

Introduction

Motivation and Background

A computer's memory system is the repository for all the information used by and produced by the computer's central processing unit (CPU). The phenomenal increase in microprocessor performance places significant demands on the memory system. A perfect memory system is one that can supply immediately any datum that the CPU requests. This ideal memory is not practically implementable, however, as the three factors of memory (capacity, speed, and cost) are in direct opposition. An economical solution, then, is a memory hierarchy organized into several levels, each smaller, faster, and more expensive per byte than the next. The goal is to provide a memory system with cost almost as low as the cheapest level of memory and speed almost as fast as the fastest level. The concept of memory hierarchy takes advantage of the principle of locality. Briefly, this principle states that accessed memory words will be accessed again quickly (temporal locality) and that memory words adjacent to an accessed word will be accessed soon after the access in question (spatial locality). Further *Smaller is faster* - smaller pieces of hardware will generally be faster than larger pieces. This simple principle is particularly applicable to memories built from similar technologies for two reasons. First, larger memories have more signal delay and require more levels to decode addresses to fetch the required datum. Second, in most technologies we can obtain smaller memories that are faster than larger memories. This is primarily because the designer can use more power per memory cell in a smaller design. The fastest memories are generally available in smaller number of bits per chips at any point in time, and they cost substantially more per byte. Based on the above reasons, the memory hierarchies of modern general purpose computers generally contain the following:

- **Registers:** These are the small, fast storage buffers within the CPU. The Compiler is responsible for managing their use, deciding which values should be kept in the available register at each point in the program.

- **Cache:** This is small, fast memory located close to the CPU that holds the most recently accessed code or data. The caches are generally made up of **SRAM (static random access memory)**. When the CPU finds the requested data item in the cache, it is called a *cache hit*. When the CPU does not find the data item it needs in the cache, it results in *cache miss*, and the data item must be fetched from main memory. Caches typically exploit the principle of spatial locality of reference by prefetching a fixed amount of data contiguous to the referenced value. Caches can vary widely in their size and organization, and there may be more than one level of cache in the hierarchy. The widening gap between the processor and memory has lead to a keen interest in multi-level caches. Many of the present day caches have a two level structure, with the level closer to the processor designated as L1 cache and the next level designated as L2 cache. Though adding levels in the hierarchy is straightforward, it complicates design and performance analysis.

- **Main memory:** This is the next level down in the hierarchy. Main memory satisfies the demand of caches and serves as the I/O interface. The main memory is generally made up of **DRAM (Dynamic random access memory)** and has relatively large storage capacity as compared to the caches (SRAMs). The DRAMs also have relatively larger access times as compared to SRAMs. Further, the dynamic nature of the DRAMs accounts for their reduced performance as compared to SRAMs. However the simplified design structure and cost feasibility has made DRAMs the choice for main memory.

- **Virtual Memory:** Systems generally require a greater number of memory locations than are available in the main memory (i.e., a larger address space). The entire portion of the address space that the CPU uses is stored on large magnetic or optical disks. The most frequently used sections of the virtual memory are kept in the main memory and are moved back and forth.

Performance metrics

The performance of a computer system is related directly to its execution time. The execution time is given by

*CPU time = IC * CPI * Clock period*

IC = number of instructions executed
CPI = average clock cycles required per instruction

The performance of a computer depends on the interface between the processor and the memory. If this interface is not correct, a significant increase in CPI can result. A better measure, which captures the essence of the above statement, is the average time to access memory.

*Average memory access time = Hit time + Miss rate * Miss penalty.*

where Hit Time is the time to hit the cache, Miss rate is the fraction of access that are not in the cache and Miss penalty is the additional clock cycles to service the miss.

The two parameters, which characterize the processor-memory interface, are:

- **Memory Bandwidth:** The rate at which the memory system can service requests from the processor. Thus, in short, it refers to the amount of data that is transferred in each access.

- **Memory Latency:** The time between the initiation of a memory request and its completion.

The Processor-Memory performance gap

The rate of improvement in microprocessor speed exceeds the rate of improvement in DRAM memory speed. There are a number of reasons to account for this growing disparity. The division of the semiconductor industry into microprocessor and memory camps is the prime reason, though it has its own advantages. First and foremost, a fabrication line can be tailored to the needs of the device. Microprocessor fab lines offer fast transistors to make fast logic and many metal layers to accelerate communication and simplify power dissipation, while DRAM fabrication lines offer many polysilicon layers to achieve both small DRAM cells and low leakage current to reduce DRAM refresh rate. Seperate chips also mean separate packages, allowing microprocessors to use expensive packages that dissipate high power (5 to 50 watt) and provide hundreds of pins to make wider connection to external memory, while allowing DRAMs to use inexpensive packages which dissipate low power (1 watt) and use only a few dozen pins. Separate packages in turn mean computer designers can scale the number of memory chips independent of the number of processors.

Having the industry split into two camps - memory manufacturers and processor manufacturers, also has its inherent disadvantages. While microprocessor performance has been improving at a rate of 60 percent per year, the access time to DRAM has been improving at less than 10 percent per year [2]. Thus though each is improving exponentially, the exponent for the microprocessor is substantially larger than that for the 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. Hence computer designers are faced with an increasing *Processor - Memory Performance Gap*, which now is the primary obstacle to improved computer system performance.

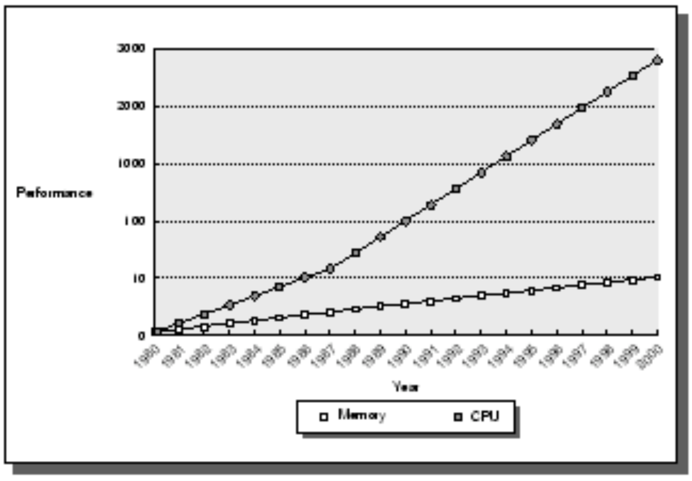

FIGURE: Starting with 1980 performance as a baseline, the performance of memory and CPUs are plotted over time.

Processor -Memory Performance Gap

Because of the growing memory access latencies (measured in processor cycles), any request that misses in the caches may eventually take hundred of cycles to satisfy. Thus system speed will now be dominated by memory performance. Quantitatively the problem involved is reducing the average memory access time. This involves primarily improving each of the three factors of Hit time, Miss penalty and Miss rate (refer to [1] for a more involved discussion on optimization of each of the three parameters). However these latency tolerance (or reduction) techniques may increase a processor's memory bandwidth needs by causing the processor to request the same stream of operands in less time or by causing the processor to request more data from memory. Thus implementation of aggressive memory latency tolerance techniques aggravate stalls due to finite memory bandwidth, which actually become more significant than stalls resulting from uncongested memory latency alone. Thus the two factors of memory latency and memory bandwidth are closely related. Improvement in memory bandwidth is critically necessary to support aggressive memory latency techniques. The range of techniques to improve effective memory bandwidth includes [3]:

1. Wider and faster connection to memory.

   As pointed out earlier, memory bandwidth is the amount of data bits transferred per second. Therefore traditional approaches to improving the memory bandwidth include speeding up the memory clock (this is to reduce the access latencies), increasing the bus width, or both. For conventional DRAMs, these approaches are reaching their practical limits. While clock rate scaling causes stringent system timing requirements, which dictate precise component and PCB modeling, the increase of bus width comes at the expense of increasing the pin count, increasing I/O power, and creating a host of mechanical and PCB layout problems. Although large increases in pin counts have recently occurred - and significant breakthroughs in packaging technology undoubtedly lie on the horizon- the issues of reliability, power and especially cost will prevent pins from sustaining growth in numbers commensurate with the growth rate of processor performance. Of late, novel memory system interfaces like Rambus (RDRAM) and Synchronous link (SLDRAM) have emerged which promise bandwidth on the order of multi-gigabytes/second. Another technique to buy more bandwidth is to use memory interleaving, but this comes at the expense of increased cost and problems associated with memory expansion. The key question is not whether providing enough memory bandwidth with each generation is possible, but whether providing enough for each generation is cost effective.

2. Larger on-chip caches.

   Larger caches improve effective bandwidth by sending fewer requests (misses) across the interconnect. Present day on-chip caches are reaching the megabyte range. Although caches this large will be able to hold the working sets for many applications, there will always be programs whose access patterns aren't amenable to caching. There will also be programs with working sets that are too large to fit in these caches. Further, large on-chip caches have the affect of increasing the system cost.

3. Dynamic access ordering [4]

   This method maximizes memory performance for streaming computations (e.g., signal/image processing, multimedia compression and decompression). This approach is based on access ordering, or changing the order of memory requests to improve the rate at which those requests are serviced by a memory system with non-uniform access times. This method combines a hardware and software approach: the compiler arranges for the processor to transmit a stream of information to a Stream Memory Controller, or SMC, at run-time, and the SMC dynamically reorders the access, attempting to issue them in a sequence that maximizes effective memory bandwidth. The processor issues its memory requests in the natural order of the computation, and the streamed data is buffered within the controller until requested by the processor (for memory loads) or written to memory by the controller (for memory stores).

4. More efficient on-chip caches.

   Although caches are quite effective at capturing temporal and spatial locality in the dynamic reference stream, they do so in a crude fashion, and are far from optimal in their use of buffering active data. An effort to improve the content of caching is the Impulse Project [5]. Impulse is a new memory system architecture that adds two important features to a traditional memory controller. First, Impulse supports application specific optimizations through configurable physical address remapping. By remapping physical addresses, applications can control how their data is accessed and cached, which enables them to improve their cache and bus utilization. Second, Impulse supports prefetching at the memory controller. By prefetching data at the controller, the memory system can hide much of the DRAM accesses. Because it requires no modification to processor cache, or bus design, impulse can be adopted to a conventional system.

5. Logic/DRAM integration.

   The processor-memory bandgap is becoming an increasing impediment to performance, which might shift the focus to integrating the processor on the same die (or in the same package) as the main memory. This eliminates the need for expensive, high bandwidth interchip interconnects. However, as mentioned earlier, the difference between manufacturing processors (logic) and memory gives rise to a lot of challenges to make this technology feasible. Much research is being conducted in this area. Particularly noteworthy is the Intelligent RAM (IRAM) project at University of California at Berkeley [2].

Computing by Compression (CC) paradigm

Another interesting method for improving the effective bandwidth of an interconnect is compression. Researchers have proposed and/or implemented schemes to use compression for data, addresses and code. All these schemes increase effective bandwidth to memory at the expense of some extra hardware on the CPU/Memory end. Current technology trends (computation growing cheaper relative to expensive communication) will make compression more attractive in the future.

Cache Operation and Information Theory

An alternative view of the operation of cache memory is that caches work because of redundancy in the sequence of address requests made to memory by the processor. It is this redundancy that allows the simple demand fetch hardware algorithm to accurately predict the future reference requirement of the processor. In a now well known study on the subject [6], Hammerstrom showed that there was generally no more than one percent information content in an address trace. Given knowledge of past reference, 99 bits out of every 100 bits of address reference made to the memory system by the processor are predictable. The remaining bit of information represents the occurrence of references to new localities undeterminable until the time of program execution.

Thus the paradigm Computing by Compression (CC) is particularly interesting as it brings to surface the important question - *Are we really communicating intelligent information between the memory and CPU?* How much redundancy can be exploited in the information that is exchanged between the processor and memory? For example we already know that the programs exhibit temporal and spatial locality. In the above circumstance, is it really wise to dedicate an accruing number of address lines (pins) to communicate these addresses when their temporal and spatial features can be better exploited? Therefore it becomes very important to analyze the present traffic (data, instructions and addresses) and exploit the redundancy involved to devise specific compression techniques to send the optimal information between the processor and memory.

Related Work

1. Creating wider Bus Caching Techniques [7]

   The effective bandwidth of a bus (communication channel) can be increased by using a variant of data compression techniques, exploting the ``*Principle of Locality*'' of the data values. The compaction is performed by caching the higher order bits into a table and sending the index into a table along with the lower order bits. A coherent table at the receiving end expands the word into its original form. Compaction/expansion units can be placed between the memory and processor, between the processor and local bus and between the devices that access the system bus.

2. Address Compression through Base register caching [8]

   With the increase in the address space, more and more address lines are required for address communication between the processor and the CPU. However, limitations are imposed by the number of pins. Address reference streams exhibit large amounts of temporal and spatial locality and are consequently very predictable. Therefore, a typical address word has very low information content. By caching the higher order portions of address references in a set of dynamically allocated base registers, it becomes possible to transmit small register indices between the processor and memory instead of higher order bits themselves, thereby increasing available processor bandwidth.

3. Instruction Compression in Embedded Systems [9, 10]

   This method is basically used for compressing programs in embedded processors, where instruction memory size dominates cost. In this method, a post compilation analyzer examines a program and replaces common sequences of instruction with a single instruction code word which acts as a pointer to a dictionary (maintained at the processor end) containing the sequences of instructions it represents. Thus, effective compression of program can be obtained. Another research effort has been to use ideas of text compression for efficient reduction of code size.

Conclusion

The problem of *Memory Wall* [11] is becoming increasingly important and is emerging as one of the greatest impediments to the microprocessor system performance. It is now time to start thinking about how to avoid hitting this wall. Though many techniques have been suggested to circumvent this problem, most of them provide one time boosts of either bandwidth or latency. While these delay the date of impact, they don't change the fundamentals. The most convenient resolution would be the discovery of a cool, dense memory technology whose speed scales with that of processors. Though nothing concrete has been taken in this direction, a good approach, at present time, would be to affect changes at the architectural level. The CC concept has a lot of potential but it warrants more research and analysis before actual implementation.

References

1

John L Hennessy and David A Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, CA, 1996.

2

David Patterson, Thomas Anderson et al., *A Case for Intelligent RAM : IRAM*, IEEE Micro, April 1997.

3

D. Burger, J.R. Goodman, and A. Kagi. Memory Bandwidth Limitations of Future Microprocessors, *Proc. 23rd Ann. Int'l Symp. Computer Architecture*, Assoc. of Computing Machinery, pp. 79-90, Aug. 1996.

4

McKee, S.A. *Dynamic Access Ordering : Bounds on Memory Bandwidth*, University of Virginia, Technical Report CS-94-14, April 1994.

5

J. Carter et al. Impulse: An Adaptable Memory System. Submitted to *5th Int'l Symp. Computer Architecture*, Assoc. of Computing Machinery, Jan 1999.

6

Hammerstrom, D. and E. Davidson. Information Content of CPU Memory Referencing Behavior, *Proc. 4th Ann. Int'l Symp. Computer Architecture*, Assoc. of Computing Machinery, pp. 184-192. March 1977.

7

L. Rudolph and D. Criton. Creating a Wider Bus using Caching Techniques. *Proc. 1st Int'l Symp. High Performance Computer Architecture*. IEEE Comp Society Press, pp. 90-99. 1995.

8

M. Farrens and A. Park. Dynamic Base regsiter caching: A Technique for reducing address bus width. *Proc 18th Int'l Symp. Computer Architecture*, ACM, pp.128-137. 1991.

9

T. Mudge and P. Bird. An Instruction stream Compression Technique. *Proc of Micro* -30, Dec 1997.

10

Srinivas Devadas et al. A text Compression based method for code size minimization in Embedded System. To appear in ACM TODAES., Vol 4, No 1, Jan 1999.

11

Wm. A. Wulf and Sally A McKee. Hitting the Memory wall: Implications of the Obvious. *Computer Architecture News*, 23(1), pp. 20-24, March 1995.

FIGURE inserted with kind permission of Dr. David A Patterson, University of California, Berkeley.

About the authors:

Dr Nihar Mahapatra is an Assistant Professor in the Computer Science and Engineering Department at the State University of New York at Buffalo. His research interests include Computer Architecture, Parallel Processing and VLSI design.

Balakrishna Venkatrao is a graduate student in the Computer Science and Engineering Department at the State University of New York at Buffalo. His research interests include Computer Architecture, VLSI and molecular electronics.

Want more articles about Computer Architecture? Go to the index, the the next one or to the previous one.

Last Modified:
Location: www.acm.org/crossroads/xrds5-3/pmgap.html