A Survey of Memory Bandwidth and Machine Balance in Current High Performance Computers

Author:
John D. McCalpin
Silicon Graphics Computer Systems
mccalpin@sgi.com
http://reality.sgi.com/mccalpin/
Keywords:
memory bandwidth, hierarchical memory, shared memory, vector processors, machine balance

---

Abstract

The ratio of cpu speed to memory speed in current high-performance computers is growing rapidly, with significant implications for the design and implementation of algorithms in scientific computing. I present the results of a broad survey of memory bandwidth and machine balance for a large variety current computers, including uniprocessors, vector processors, shared-memory systems, and distributed-memory systems. The results are analyzed in terms of the sustainable data transfer rates for uncached unit-stride vector operations for each machine, and for each class. A Case Study showing the applicability of this performance metric for a large computational fluid dynamics code is also presented.

Introduction

It has been estimated that the cpu speed of the fastest available microprocessors is increasing at approximately 80% per year [Bas91], while the speed of memory devices has been growing at only about 7% per year [Hen90]. The ratio of the cpu to memory performance is thus also growing exponentially, suggesting the need for fundamental re-thinking of either the design or computer systems or the algorithms that scientific users employ on them [Bur94], [Wul94]. For example, 10 years ago, floating-point operations were considered quite expensive, often costing 10 times as much as an uncached memory reference. Today the situation is dramatically reversed, with the fastest current processors able to perform 100 or more floating-point operations in the time required to service a single cache miss. Because of this fundamental change in the balance of the underlying technology, this report presents a survey of the memory bandwidth and machine balance on a variety of currently available machines.

Interestingly, despite the large amount of academic research on improving the performance of cache-based systems (e.g., the review in [Bur94]), almost all of the systems here (which represent most of the machines sold in the U.S.) are either vector machines or standard hierarchical memory machines. No pre-fetching, cache bypass, or other novel techniques are represented here, and of the machines tested, only the IBM Power2 (models 990 and 590) allows more than one outstanding cache miss (it allows two).

The major sections here include a definition of Machine Balance, a discussion of the Stream Benchmark, a Case Study illustrating the importance of bandwidth and balance in a scientific application code, and a Discussion of the implications of current trends for high performance computing.

Machine Balance

The concept of *machine balance* has been defined in a number of studies (e.g.,[Cal88]) as a ratio of the number of memory operations per cpu cycle to the number of floating-point operations per cpu cycle for a particular processor.

$$\frac{\text{peak floating ops/cycle}}{\text{peak memory ops/cycle}}$$

This definition introduces a systematic bias into the results, because it does not take into account the true cost of memory accesses in most systems, for which cache miss penalties (and other forms of latency and contention) must be included. In contrast, the "peak floating ops/cycle" is not strongly biased, because extra latencies in floating-point operations are due to floating-point exceptions, and we will assume that these are rare enough to be ignored.

Therefore, to attempt to overcome the systematic bias incurred by the use of this definition, the definition of machine balance used here defines the number of memory operations per cpu cycle in terms of the performance on long, uncached vector operands with unit stride.

$$\text{balance} = \frac{\text{peak floating ops/cycle}}{\text{sustained memory ops/cycle}}$$

With this new definition, the "balance" can be interpreted as the number of FP operations that can be performed during the time for an "average" memory access.

It would be foolish to claim too much applicability for this concept of "average", but it should give results that are representative of the performance of large, unit-stride vector codes. It is clearly not a "worst-case" definition, since it assumes that all of the data in the cache line will be used, but it is not a "best-case" definition, since it assumes that none of the data will be re-used.

As a single data point regarding its applicability, I present this Case Study of the performance of one of my vector application codes to demonstrate the way in which the obtained performance correlates rather well with the sustainable memory bandwidth.

Interestingly, information on sustainable memory bandwidth is not typically available from published vendor data (perhaps because the results are generally quite poor), and had to be measured directly for this project by use of the STREAM benchmark code.

The "peak floating-ops/cycle" is derived from the vendor literature, much of it by way of Dongarra's LINPACK benchmark report. Most current machines have a sustainable floating-point operations rate that is very close to the peak rate (provided that one is using data in registers), so this usage does not introduce a significant bias into the results. An alternative value, such as the LINPACK 1000 or LINPACK scalable result would be equally useful here, but would result in no qualitative changes to the conclusions.

The STREAM Benchmark

The memory bandwidth data was obtained by use of the *STREAM* benchmark code. *STREAM* is a synthetic benchmark, written in standard Fortran 77, which measures the performance of four long vector operations.

The *STREAM* benchmark and the current results are presented on the STREAM Benchmark Home Page.

What is perhaps most shocking about the results is the poor sustainable memory bandwidth of the hierarchical memory machines. The actual values obtained from user code are often only a small fraction of the impressive "peak" bandwidth numbers stated by a number of vendors. The discrepancy will be investigated further in the Discussion. Unfortunately, even "peak bandwidth" numbers are stated by so few vendors that it is not possible to do a comprehensive survey of the ratio of "peak" to "sustainable" bandwidth.

In order to make some sense of the diversity of the numbers, the machines have been divided up into various categories based on their memory system type. The three categories used here are

- Shared-memory
- Distributed Memory
- Uniprocessor

Note that the "Vector" category has been removed -- those results are clearly distinguishable in the "Shared Memory" and "Uniprocessor" categories by their low "balance" parameter. The results are plotted in Figure 1, below.

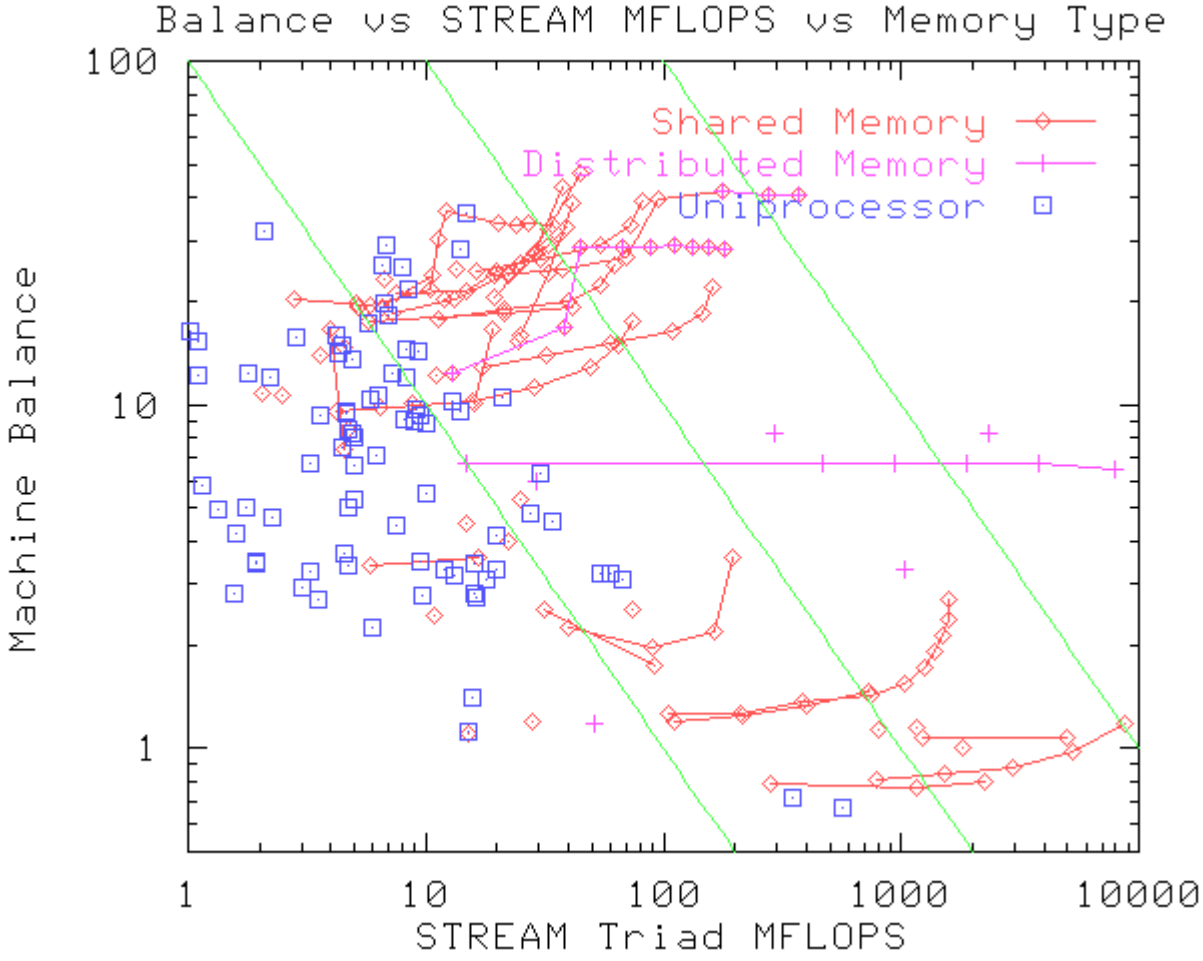

Figure 1 STREAM TRIAD MFLOPS and Machine Balance for a variety of recent and current computers. Each point represents one computer system, and connected points represent either multi-processor results from parallel systems, or different cpu speeds within the same cpu family for uniprocessor systems. Individual systems can be identified by referring to the annotated image. The diagonal green lines represent **peak** performance values of 100 MFLOPS, 1 GFLOPS, and 10 GFLOPS (from left to right).

Discussion

The results in Figure 1, above, show a remarkably clear distinction between the memory categories:

- Shared-memory: poor balance, fair scalability, moderate performance.
- Vector: good balance, moderate scalability, high performance.
- Distributed Memory: fair balance, perfect scalability, high performance.
- Uniprocessor: fair to good balance, low to moderate performance.

Uniprocessor Results

On hierarchical memory machines, the key determinant of the sustainable memory bandwidth for a single cpu is the cache miss latency. In the last few years, the memory systems of cached machines have experienced significant shifts in the ratio of the relative cost of latency vs transfer time in the total cost of memory accesses, going from an approximately even split in the typical 20 MHz machines of 1990, to being strongly dominated by latency in the typical 100 MHz machine of 1995. This trend is especially strong in shared-memory machines, for which the cost of maintaining cache coherence is a significant contributor to the latency.

The results for the uniprocessor systems clearly show two strategies for optimizing the performance of the cache systems:

- The first strategy is to optimize for unit-stride accesses. The approach is exemplified by the IBM RS/6000 series, which uses long cache lines (64, 128, or 256 bytes) and has minimal latency. The models in that line which use the Power2 cpu have a further reduction in the effective latency because the two "Fixed-Point Units" in the cpu can each handle independent cache misses at the same time, thus overlapping their latencies with the latency and transfer time of the other unit.
- The second strategy is to minimize memory traffic due to unused data. In the limit, this would correspond to single-word cache lines. Given sufficient latency tolerance, this can be optimal ([Kon94]), but since these machines do not have effective latency tolerance mechanisms, this approach is too inefficient for the important case of unit-stride accesses. The most common compromise used for this case is a 32 byte line size, as is used in the HP PA-RISC and DEC Alpha (21064) systems. While this approach is reasonably effective in low latency situations, there is minimal gain from short line sizes in high latency situations, since effective transfer rates are limited largely by latency rather than by a busy bus. In multiprocessor systems, this approach is perhaps more justifiable, since unnecessary bus traffic (due to overly long cache lines) will interfere with the other processors as well.

Shared Memory Results

It should be noted that the vector machines are all shared-memory (except for one distributed memory machine), and so manage to maintain their good balance, scalability, and performance despite the negative factors that reduce the performance of the hierarchical-memory shared-memory machines. The vector machines with the best performance characteristics do not employ hierarchical memory, thus greatly simplifying the coherence issue and associated latency penalty. In general, the vector machines are more expensive than the shared-memory, hierarchical-memory machines, but the larger configurations of the hierarchical-memory systems do overlap with the price range of the traditional supercomputers. When normalized for STREAM TRIAD performance, the traditional vector supercomputers are always more cost-effective than the shared-memory, hierarchical memory systems, as well as being marginally more cost-effective than the most cost-effective uniprocessors in the table.

Both the cached and vector shared-memory machines have absolute memory bandwidth limitations, which are visible in Figure 1 as a sharp increase in the machine balance parameter as a the number of processors reaches a critical level. (This is not visible on most of the vector machines because they are deliberately limited in the number of processors supported in order to avoid this imbalance.)

Typically, the shared memory system (whether it is implemented via a bus, switch, crossbar, or other network) is non-blocking between processors, which allows multi-cpu parallelism to act as a latency tolerance mechanism. The "wall" hit by the machines when using many processors is a combination of latency, absolute bandwidth limitations (due to the limited number of DRAM banks), and bus/network/switch controller limitations. On vector machines, the limitation is usually bandwidth rather than latency for both single and multiple cpus.

Although extra processors can be used to provide latency tolerance in parallelized applications, this approach is both expensive and contributes to greatly increased (i.e. poorer) machine balance. It seems likely that it would be more efficient to have special-purpose load/store units stall on cache misses, rather than entire cpus. This is the approach taken by the IBM Power 2 processor (with two fixed-point units to handle independent loads and stores), and by the new DEC 21164, HP PA-7200, SGI/MIPS R10000 processors, the latter two of which are designed to handle four outstanding cache miss requests simultaneously.

It remains to be seen whether such "linear" solutions will be able to keep up with what is essentially an exponential increase in machine balance, or whether more fundamental architectural changes will be required in the very near future.

Performance Characterization

The problem of optimizing performance in a memory-bandwidth-poor computational environment has been addressed often in the literature. A useful definition was proposed by [Don85], who defined performance in excess of the "STREAM TRIAD" performance as *super-vector* performance on the Cray-1 supercomputer. A similar nomenclature is perhaps appropriate for hierarchical memory machines. The performance of a code will be referred to as *super-streaming* if it exceeds the performance of the "STREAM TRIAD" benchmark test. Such codes will typically be *cpu-bound*, while codes that perform more slowly than the "STREAM TRIAD" test (or the "STREAM SUM" test if the adds and multiplies are not approximately equally balanced) will typically be *memory-bandwidth bound*. This is discussed a bit more in the context of the *QG_GYRE* code in the Case Study .

Trends in Hardware

The large size of the computer industry and the rapid turnover of each model of computer combine to make surveys difficult. Using the data acquired in this study, we will nevertheless make an attempt to examine trends in the performance characteristics of computer hardware, in the context of sustainable memory bandwidth measurements. Using a subset of the data representing various models of Cray, IBM, SGI, DEC, and HP computers, Figure 2, shows the following quantities:

- Peak MFLOPS
- SPECfp92
- Sustainable Memory Bandwidth in MWords/second
- "Efficiency" defined as:

$$\frac{\text{Sustained MWords/second}}{\text{Peak MFLOPS}} * 100$$

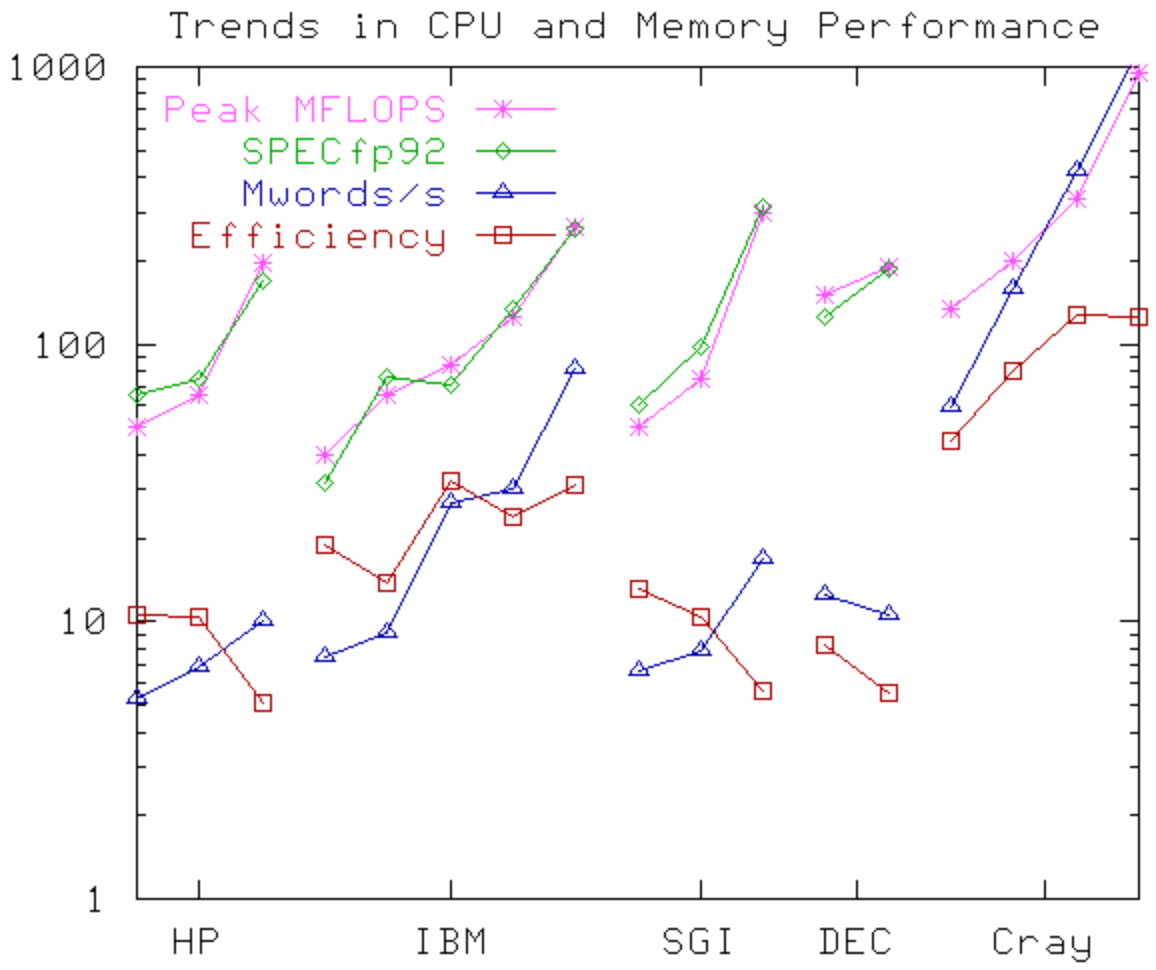

Figure 2 Trends in Peak MFLOPS, SPECfp92, Sustainable Memory Bandwidth (Mwords/s), and "Efficiency" (defined above).

The specific models used in Figure 2 are:

- HP: HP 9000/720, HP 9000/720, HP 9000/735
- IBM: RS/6000 Models 250, 320, 950, 980, 990
- Silicon Graphics: Indigo R4000 (100 MHz), Challenge (150 MHz), Power Challenge
- DEC: 3000/500, 4000/710
- Cray: EL-98, J916, Y/MP, C90

In general, the machines are ordered such that either time or cost increases left to right within each vendor family. Although the relationships between the machines are not at all simple, we observe that for the machines tested from DEC, HP, and SGI, the peak cpu performance is increasing significantly faster than the sustainable memory bandwidth, thus resulting in decreasing "Efficiency". In contrast, the machines from Cray and IBM show a relatively constant "Efficiency" despite increases in peak performance that are similar to those of the other set of vendors.

One might conclude from this that DEC, HP, and SGI have placed a relatively high priority on improving the performance of the SPEC benchmark in their recent development, while IBM and Cray have favored a more "balanced" approach. While vendor attention to realistic benchmarks is generally a "good thing", in this case it may have acted to deflect attention from the difficult problem(s) of increasing memory bandwidth and

maintaining machine balance, since the SPECfp92 benchmarks are relatively undemanding with respect to memory size and bandwidth requirements. The new SGI/MIPS R10000 and HP PA-7200 and PA-8000 appear to be the beginnings of a deliberate counter-trend, with an advertising emphasis on improving "real-world" performance by larger factors than the improvement in SPECfp92 performance.

Some Current Software Issues

A variety of software issues are related to the problem of the increasing imbalance of current microprocessor-based computers. Current compiler technology is limited in its ability to perform code transformations to efficiently make use of memory hierarchies ([Bac94],[Lam91]), and current languages have no direct method of specifying locality, data re-use, and data dependence information necessary to fill in the gaps in the compiler's abilities. Some current trends, such as the use of Fortran 90 array notation, actually exacerbate the problem of memory hierarchies by removing inter-statement data dependency information, resulting in the generation of extra temporaries and excessive memory traffic [Mcc95].

Conclusions

A review of the sustainable memory bandwidth of a large variety of current and recent computer systems reveals strong systematic variations in the machine balance according to memory type. In particular, hierarchical-memory, shared-memory systems are generally strongly imbalanced with respect to memory bandwidth, typically being able to sustain only 5-10% of the memory bandwidth needed to keep the floating-point pipelines busy. Thus only algorithms which re-use data elements many times each can be expected to run efficiently. In contrast, vector shared-memory machines have very low machine balance parameters and are typically capable of performing approximately one load or store per floating-point operation. Of course, this capability is a strong requirement for good performance on the systems, since they typically have no cache to facilitate data re-use.

A Case Study of the performance of a two-dimensional finite-difference ocean model is used to illustrate these points as a function of problem size and machine type. The performance characteristics across a broad range of machines are well correlated with the sustainable memory bandwidth and poorly correlated with any of the SPECfp92 tests. This latter point is likely due to the inflated SPECfp92 ratios on machines with large (>1 MB) caches.

The recent shift in machine balance of current high performance computers strongly suggests that steps need to be taken soon to increase the memory bandwidth more rapidly. It is likely that merely increasing bus width and decreasing latency will not be adequate, given the rapid increase in cpu performance. What is needed instead is a set of more fundamental architectural changes to enable the systems to use information about data access patterns in order to effectively apply latency tolerance mechanisms (e.g. pre-fetch, block fetch, fetch with stride, cache bypass, etc.). At the same time, these systems should not preclude the use of "dumb" caches in the memory hierarchy when the memory access patterns are not visible to the compiler. This merger of "vector/flat memory" and "scalar/hierarchical memory" architectures should be a major subject of research in high performance computing in the closing years of this millennium.

Bibliography

[Bac94] D.F. Bacon, S.L. Graham, and O.J. Sharp. Compiler Transformations for High-Performance Computing. ACM computing surveys, Volume 26, Number 4, December 1994, pp 345ff.

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

[Bur95] D.C. Burger, J. R. Goodman, and Alain Kägi. The Declining Effectiveness of Dynamic Caching for General-Purpose Microprocessors. University of Wisconsin, Department of Computer Science, Technical Report, TR-1261, 1995.

[Cal88] D. Callahan, J. Cocke, and K. Kennedy. Estimating Interlock and Improving Balance for Pipelined Architectures. *Journal of Parallel and Distributed Computing*. 5:334-358, 1988.

[Don85] J.Dongarra, L. Kaufman and S. Hammarling. Squeezing the Most out of Eigenvalue Solvers on High-Performance Computers. Argonne National Laboratory, Technical Memorandum no. 46.

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

[Kon94] L.I. Kontothanassis, R.A. Sugumar, G.J. Faanes, J.E. Smith, and M.L. Scott. Cache Performance in Vector Supercomputers. Proceedings, SuperComputing'94, IEEE Computer Society Press.

[Lam91] M.S. Lam, E.E. Rothberg, and M.E. Wolf. The Cache Performance and Optimizations of Blocked Algorithms. ASPLOS IV, 1991.

[Mcc95] J.D. McCalpin. On the Optimization of Fortran 90 Array Notation: A Case Study. Submitted to *Scientific Programming*, December 1994.

[Wul94] W.A. Wulf and S.A. McKee, Hitting the Wall: Implications of the Obvious, Computer Science Report No. CS-94-48, University of Virginia, Dept. of Computer Science, December 1994.

John D. McCalpin mccalpin@sgi.com