On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance

Richard Murphy

Sandia National Laboratories*
PO Box 5800, MS-1110
Albuquerque, NM 87185-1110
rcmurph@sandia.gov

Abstract

Since the first vector supercomputers in the mid-1970's, the largest scale applications have traditionally been floating point oriented numerical codes, which can be broadly characterized as the simulation of physics on a computer. Supercomputer architectures have evolved to meet the needs of those applications. Specifically, the computational work of the application tends to be floating point oriented, and the decomposition of the problem two or three dimensional. Today, an emerging class of critical applications may change those assumptions: they are combinatorial in nature, integer oriented, and irregular. The performance of both classes of applications is dominated by the performance of the memory system. This paper compares the memory performance sensitivity of both traditional and emerging HPC applications, and shows that the new codes are significantly more sensitive to memory latency and bandwidth than their traditional counterparts. Additionally, these codes exhibit lower base-line performance, which only exacerbates the problem. As a result, the construction of future supercomputer architectures to support these applications will most likely be different from those used to support traditional codes. Quantitatively understanding the difference between the two workloads will form the basis for future design choices.

1. Introduction and Motivation

Supercomputing is in the midst of large technological, architectural, and application changes that greatly impact the way designers and programmers think about the system. Technologically, the constraints of power and the speed of light dictate that multicore architectures will form the basis for the commodity processors that constitute the heart of massively parallel processing (MPP) supercomputers. This impacts the architecture in three ways:

1. Power constraints dictate that clock rates will not improve appreciably as they have in the past.
2. The combination of power, the constraint of the speed of light, and the architectural limits of instruction level parallelism dictate that the trend in scalar processors towards higher performing individual cores will not hold.
3. Even though die area is increasing, cost is still dictated by packaging, and the number of pins available for external communication will likely not grow as quickly as the number of cores.

These technological and architectural trends are no less significant than the those which dictated the transition from vector-based supercomputers to commodity MPP architectures in the early 1990's. In fact, given the maturity of vector architectures, such a change was likely inevitable. One critical architectural trend still holds across any implementation: the challenge posed by the memory wall. In today's supercomputers, the memory wall manifests itself as a dramatic difference between the increase in processor clock rate and the rate of a memory access. In multicore machines – even with flat clock rates – the memory wall manifests itself due to the increasing number of cores compared to available *channels* (or independent access paths) to memory.

Unlike prior large-scale technological and architectural changes, today's architects must also contend with a shift in the application base. Historically, supercomputing has been dominated by the simulation of physics on a computer, which itself can be thought of as fundamentally structured in nature. In a three dimensional universe, problem decomposition can be performed in three dimensions (by dividing the simulated area into cubes). The types of supercom-

*Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

puter architectures that have been adopted often reflect this structure. For example, 3d mesh topologies reflect the kind of "nearest neighbors" communication pattern common in physics codes. Furthermore, the "work" performed in these applications is fundamentally floating point: the computation of temperature, pressure, volume, etc. Many emerging applications, however, are different. They are combinatorial in nature, fundamentally unstructured, and often consist of integer computations.

This paper examines the memory performance of a suite of **real world** applications from both the traditional and emerging problem domains. It examines the impact of memory latency and bandwidth on the applications. The results demonstrate that both sets of applications are fundamentally dominated by memory latency, but that the emerging applications both begin with a lower baseline performance, and are more sensitive to memory than their traditional counterparts. This represents a significant challenge to supercomputer architects, and quantitatively establishes how emerging applications differ from their traditional counterparts.

The remainder of this paper is organized as follows: Section 2 discusses the related work. Section 3 examines the applications under study. Section 4 specifies the methodology and metrics used for evaluation. Section 5 presents the results Conclusions are given in Section 6.

2. Related Work

Characterizing performance is a richly studied area of computer architecture. The SPEC suite has been extensively characterized[10, 8, 12, 14, 26], as have other workloads such as OLTP[13, 4, 2]. These codes are generally chosen to represent "typical" machine workloads for a class of applications.

In the area of supercomputing, specialized benchmarks have been constructed to test specific areas of machine performance. The HPC Challenge RandomAccess benchmark[6] and the STREAM benchmark[15] have been specifically constructed to measure the performance of the memory system. Although this information is useful to architects, it is difficult to directly map back to application performance.

The applications in this work have been studied extensively[19, 18, 20]. The floating point suite is similar in structure to other real world supercomputer applications that have been previously examine[21, 28].

Numerous definitions of spatial and temporal locality have been proffered to characterize the memory performance of applications, both canonical [22, 9] and experimental[19, 29, 5, 7, 27, 25]. This study examines the performance impact of the memory system on applications.

The memory wall is also an extremely well studied problem in computer architecture[30, 16]. It has been argued that this is the dominant problem in computer architecture[11]. Indeed, the results of this work further support this conclusion by affirming that key supercomputer applications are memory latency dominated in performance, which is the classic definition of the memory wall.

3. Applications

This study examines two classes of important applications from Sandia National Laboratories: a traditional set of primarily floating point codes, and an emerging class of primarily integer codes. These codes have been discussed extensively previously, and are significantly different from traditional suites such as SPEC CPU[20, 19]. Their description and basic instruction mix follow.

3.1. Traditional Floating Point Codes

Each of the floating point applications are production MPI codes designed to run at very large scale. Broadly speaking they are scientific and engineering applications which represent physical simulations. They are:

- **ALEGRA** is a finite element shock physics code capable of modeling near- and far-field responses to explosions, impacts, and energy depositions.
- **CTH** is a multi-material, large deformation, strong shock wave, solid mechanics code developed at Sandia National Laboratories over the last 30 years. CTH models multi-phase, elastic viscoplastic, porous and explosive materials with multiple mesh refinement methods.
- **Cube3** Cube3 is a generic linear solver that drives the Trilinos framework for parallel linear and eigensolvers. It mimics a finite element analysis problem by creating hexagonal elements, then assembling and solving a linear system. The width, depth, and degrees of freedom (e.g., temperature, pressure, velocity, etc.) can be varied.
- **ITS** performs Monte Carlo simulations of linear time-independent coupled electron/photon radiation transport.
- **MPSalsa** is a high resolution 3d simulation of reacting flow. The simulation requires both fluid flow and chemical kinetics modeling.
- **Xyce** is a parallel circuit simulation system capable of modeling very large circuits at multiple layers of abstraction (device, analog, digital, and mixed-signal). It includes both SPICE-like models and radiation models.

3.2. Emerging Integer Applications

Unlike their traditional counterparts, the emerging applications studied in this work are integer-oriented, generally problems from Discrete Math, typically unstructured, and written for a variety of programming models. They are at the core of important problems in proteomics, genomics, data mining, pattern matching, and computational geometry. Although many are classic algorithms problems (DFS, for example), the implementations are typically "newer" than their traditional counterparts (that is, there are no three decade old codes among these applications).

- **DFS** implements a depth-first search on a large graph and forms the basis for several high-level algorithms including connected components, tree and cycle detection, solving the two-coloring problem, finding Articulation Vertices, and topological sorting.
- **Connected Components** breaks a graph into components. Two vertices are in a connected component if and only if there is a path between them.
- **Subgraph Isomorphism** determines whether or not a subgraph exists within a larger graph.
- **Full Graph Isomorphism** determines whether or not two graphs have the same shape or structure.
- **Shortest Path** computes the shortest path between two vertices using a breadth first search. Real world applications include path planning and networking and communication.
- **Graph Partitioning** is used extensively in VLSI circuit design and adaptive mesh refinement. The problem divides a graph in to k partitions while minimizing the *cut* between the partitions (or the total weight of the edges that cross from one partition to another).
- **BLAST** is the Basic Local Alignment Search Tool and is the most heavily used method for quickly search nucleotide and protein databases in biology.
- **zChaff** is a heuristic for solving the Boolean Satisfiability Problem. In propositional logic, a formula is *satisfiable* if there exists an assignment of truth values to each of its variables that make the formula true.

3.3. Instruction Mix for Each Suite

Figure 1 shows the instruction mix for the integer and floating point suites. Although the "work" for the floating point suite is primarily floating point, real applications perform significantly less floating point than do typical benchmark suites. For example, the SPEC CPU 2000 suite averages 32% floating point, while real Sandia codes average

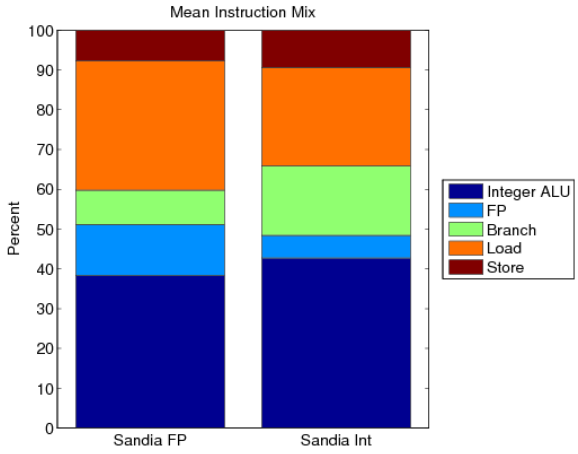


Figure 1. Sandia Integer and Floating Point Suite Instruction Mix

only about 12% floating point[19]. Other key differences between the integer and floating point codes are apparent. The integer codes perform 15% fewer memory references (although those references are much more likely to miss the cache, as Section 5 will demonstrate). Furthermore, the integer codes perform twice as many branches as their floating point counterparts. This is unsurprising since scientific codes tend to have larger basic blocks due to complex formula calculations[23].

The combination of an increased cache miss rate and an increased number of branches makes the integer codes challenging for modern superscalar processors.

4. Methodology and Metrics

This section discusses the application traces used in this work, the metrics used for evaluation, and the simulation environment. The simulation methodology is similar to that employed in prior studies of this application base, although the metrics are entirely new.

4.1. Application Traces

Each of the applications used in this work was analyzed using traces of 4 billion sequential instructions produced by the Amber[1] instruction trace generator for the PowerPC. These traces have been used in several prior studies[19, 18, 20, 23, 17, 24] and are well understood. The traces typically represent multiple executions of the main loop of the program and were originally generated with the input from applications experts and platform profiling tools. In the case of the MPI programs, traces were of single node execution.

Table 1. Processor Configuration

Parameter	Val
Issue Width	8
Commit Width	4
RUU Size	64
L1 Instruction/Data Cache	64k
	2-way Set Associative
	64-byte block
	Least Recently Used
L1 Cache Latency	3 cycles
L2 Unified Cache	1 MB
	16-way Set Associative
	64-byte block
	Least Recently Used
L2 Cache Latency	20 cycles
Integer ALUs	3
Integer Multiplier/Dividers	1
FP ALUs	2
FP Multiplier/Divider	1
Clock Rate	2.5 GHz

4.2. Metrics

This work defines bandwidth and latency as follows:

- **Latency:** the time between when the processor requests a memory value and when the first byte of that request arrives.
- **Bandwidth:** the transfer speed of the second and all subsequent bytes of a memory request.

The simulations in this work vary the memory bandwidth and latency according to this definition for each run.

4.3. Simulation Environment

The traces were used as inputs to Sandia's Structural Simulation Toolkit (SST). The SST is a parallel machine simulator (for both shared and distributed memory architectures) that uses an enhanced version of SimpleScalar's `sim-outorder` processor simulator[3] as the baseline processor. SST has significantly enhanced cache and memory models, and has been used to simulate several supercomputer architectures.

The memory simulated in this work is a DDR-like interface which performs transfers in 16-byte blocks.

Table 1 summarizes the conventional superscalar processor configuration used in this study. The memory latency and bandwidth were varied and the committed Instructions Per Cycle (IPC) measured.

The memory latencies examined were: 15ns, 30ns, 60ns, 120ns, and 240ns.

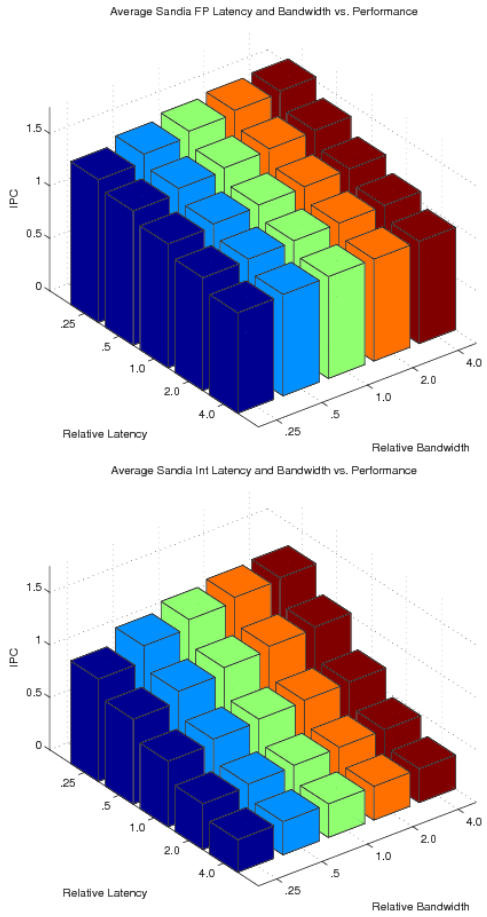


Figure 2. Average Latency and Bandwidth Effects for the Sandia Floating Point and Integer Suites

The bandwidths in this experiment were: 2.5 GB/sec, 5 GB/sec, 10 GB/sec, 20 GB/sec, and 40 GB/sec.

The baseline memory latency and bandwidth numbers in this study look somewhat more aggressive than a modern Opteron (60ns latency and 10 GB/sec of bandwidth). However, several points around that baseline were examined, and can be used for comparison given the reader's assumptions about what is appropriate. Those points were generated by halving and doubling each configuration parameter (latency and bandwidth) twice.

5. Results

Figure 2 shows the average effect of varying latency and bandwidth (60ns of memory latency and 10 GB/sec of memory bandwidth). The center point for each graph (relative latency and bandwidth of 1.0) is this baseline, and each bar represents the IPC achieved at that latency/bandwidth point.

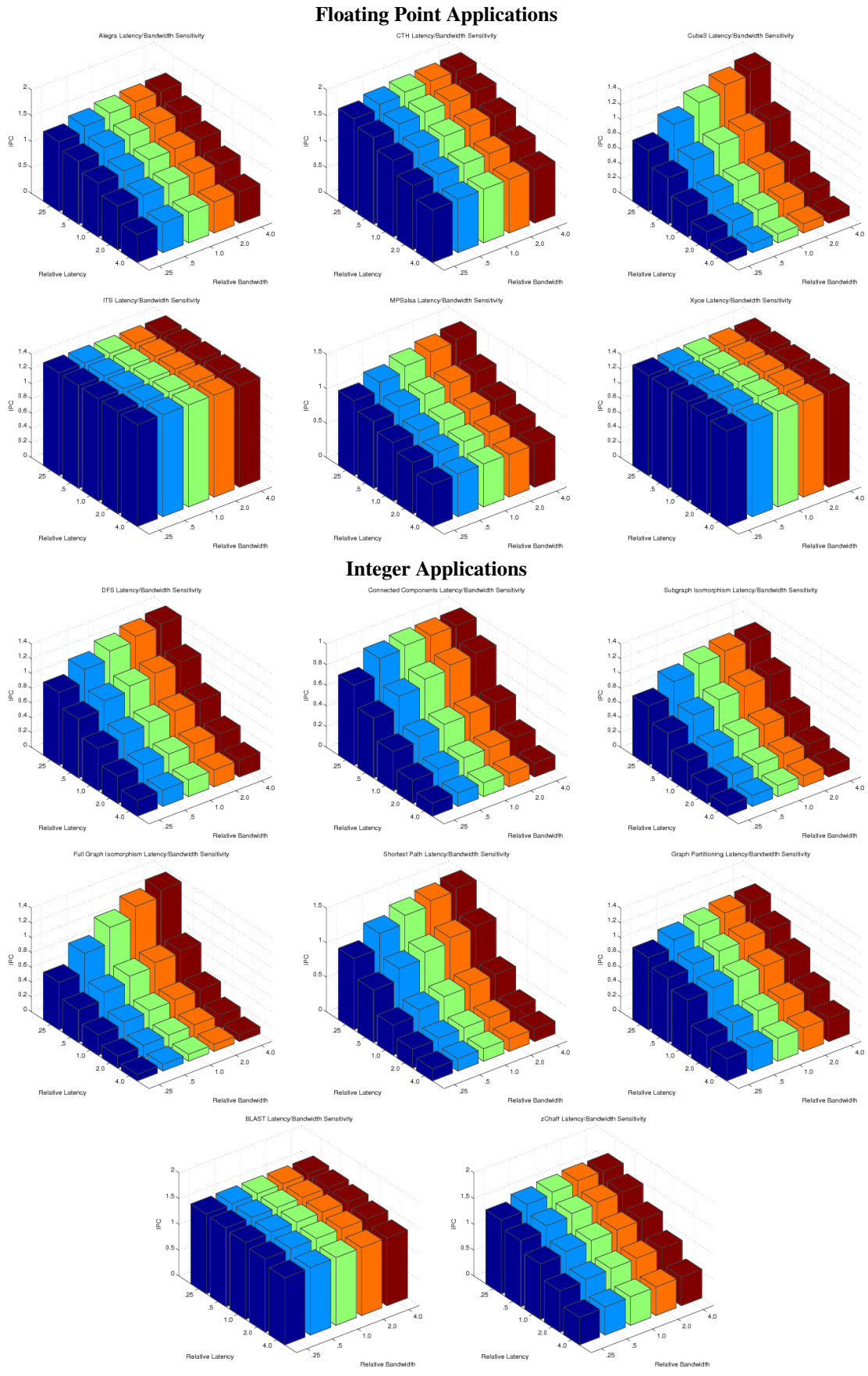


Figure 3. Complete Results for the Integer and Floating Point Suites

Figure 3 depicts the latency/bandwidth results for each application in both the integer and floating point suites.

On average, in the floating point case, halving the available bandwidth results in an average drop in performance of 1.24%. In the case of the integer suite, the average drop in performance is 3.59%. By contrast, doubling the memory latency leads to a respective drop in performance of 11% and 32% respectively. Thus, all of the codes are more latency than bandwidth dominated. This is a critical point for two reasons:

1. Memory bandwidth is the typical unit of memory performance measure discussed by supercomputer architects. This may be because the construction of MPP-based supercomputers is dominated by the construction of a high performance network interface, and MPP system architects therefore tend to think in more networking oriented terms. It may also be because bandwidth is an easier system characteristic to affect than latency. Regardless, the more critical unit of measurement is quite conclusively latency.

2. As discussed in Section 1, one of the key concerns for supercomputer architects is the transition to multicore machines. If today's instruction streams in a unicore processor were memory bandwidth bound, and available bandwidth did not grow with the number of cores, performance would suffer. However, these results indicate that there is potentially headroom in bandwidth.

Additionally, the structure of these applications generally makes it very difficult for the processor to compute memory addresses quickly enough to keep the memory bus busy. This tends to make bandwidth less important than latency.

The integer codes are clearly more sensitive to memory performance than their traditional floating point counterparts (doubling the latency or halving the bandwidth has $2.9\times$ the impact on the integer suite as compared to the floating point suite).

It is also critical to note that the baseline performance of each suite is significantly different. The floating point suite has a baseline IPC of 1.22, while the integer suite has a baseline performance of 0.70. Thus, the integer suite baselines with 43% less performance than the floating point suite, and is significantly more sensitive to latency and bandwidth variations after that.

Figure 4 shows the most affected applications for the floating point and integer suites. Cube3 from the floating point suite and Shortest Path from the integer suite look remarkably similar. Cube3 has a baseline performance only 10% lower than Shortest Path, and both applications experience a 55% drop in performance if the memory latency is doubled. Cube3 experiences a 6% drop in performance if the bandwidth is halved while Shortest Path experiences a 7% drop in performance. This is not surprising since

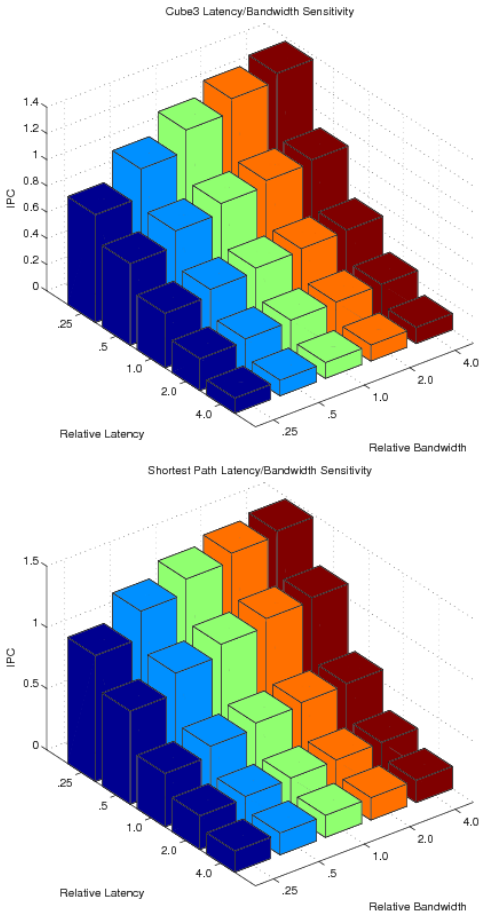


Figure 4. The Most Sensitive Applications From Each Suite

sparse graphs can be represented as sparse matrices, similar to those that are fundamental to linear algebra problems. The sparse matrix representation is not used in this particular instance of the graph problem, however the fundamental nature of the data structures used are similarly sparse.

The least affected applications from each suite (Xyce from the floating point suite and BLAST from the integer suite) show nearly flat latency/bandwidth curves. Doubling the memory latency shows less than a 1.25% performance degradation for Xyce and less than a 5% performance degradation for BLAST. Similarly, halving the bandwidth yields less than half a percent performance degradation. These applications are known to generally be more compute intensive, and this result confirms that prior knowledge. These are the only applications with relatively flat curves.

Cube3 and Alegra are the most sensitive to latency or bandwidth changes of the floating point suite, while connected components and shortest path dominate the integer suite.

To better quantify each suite's sensitivity to bandwidth and latency effects, the *sensitivity* must be defined. Intuitively, the sensitivity can be thought of as the slope of either the latency or bandwidth lines. As a first-order approximation, these curves can be thought of as linear (they are very nearly so). For a given latency or bandwidth, the slope of the resulting 2-dimensional slice of Figure 2 (either latency vs. IPC or bandwidth vs. IPC) can be calculated. For example, given the baseline latency of 60ns, the bandwidth sensitivity (or slope of the bandwidth line) can be calculated as follows:

$$\frac{P_{60ns,2.5GB/sec} - P_{60ns,40GB/sec}}{15ns - 240ns} \quad (1)$$

Where $P_{latency,bandwidth}$ represents the performance at a given latency and bandwidth. Similarly, given the baseline bandwidth of $10GB/sec$, the latency sensitivity can be computed by:

$$\frac{P_{15ns,10GB/sec} - P_{240ns,10GB/sec}}{2.5GB/sec - 40GB/sec} \quad (2)$$

The sensitivity to bandwidth is a positive number because an increase in bandwidth leads to an increase in performance, while the sensitivity to latency is a negative number because an increase in latency leads to a decrease in performance.

Figure 5(a) shows the average sensitivity to latency and bandwidth for both suites, while (b) includes each individual application for the floating point suite, and (c) includes the same information for the integer suite. As discussed earlier, the integer suite is clearly more sensitive to both bandwidth and latency than the floating point suite. The integer suite is $42 - 60\%$ more sensitive to latency than the floating point suite. While it begins 66% more sensitive to bandwidth than the floating point suite, at very high memory latencies the integer suite is nearly 30% less sensitive to bandwidth than the floating point suite.

6. Conclusions and Future Work

This paper has examined the impact of memory latency and bandwidth on a set of traditional floating-point oriented and emerging integer oriented supercomputer applications. The results demonstrate that both application suites are more dominated by latency than bandwidth. It has further shown that the emerging integer applications are $2.9\times$ more sensitive to a halving of the memory bandwidth or doubling of the memory latency than their traditional counterparts, which is critical to understanding the nature of these emerging codes.

This result has two critical impacts to the field of supercomputer architecture: first, it demonstrates that there is some degree of "bandwidth headroom" in the construction of multicore supercomputers; and second, it quantitatively shows that emerging applications are much more memory sensitive than traditional scientific computing codes. This provides further evidence in support of the long-held belief that the lessons of scientific computing have little applicability to these applications, particularly as they relate to data decomposition.

Because the transition to multicore MPPs will impact the memory system in future machines, understanding the on-node memory performance of real applications is critical. The technology determines the number of available independent channels into memory (which affects aggregate latency by constraining the number of simultaneous memory accesses the memory system can sustain), and the speed and width of those channels (which affects bandwidth). Both are likely to be more constrained than the number of cores that can placed on a die. Choosing the right balance for supercomputer applications will depend on characterizing the requirements of the workloads of those machines. This study does so, and shows that this is an even more significant problem for emerging codes than for classical supercomputing applications.

Finally, this study identifies four critical applications (two from each suite) that demonstrate the most sensitivity to latency and bandwidth. Cube3 and Alegra from the floating point suite and Connected Components and Shortest Path from the integer suite. As a whole, problems in graph theory are shown to be particularly challenging to the memory system.

Future work will extend this study to examine the impact of moving to multicore architectures with simpler cores.

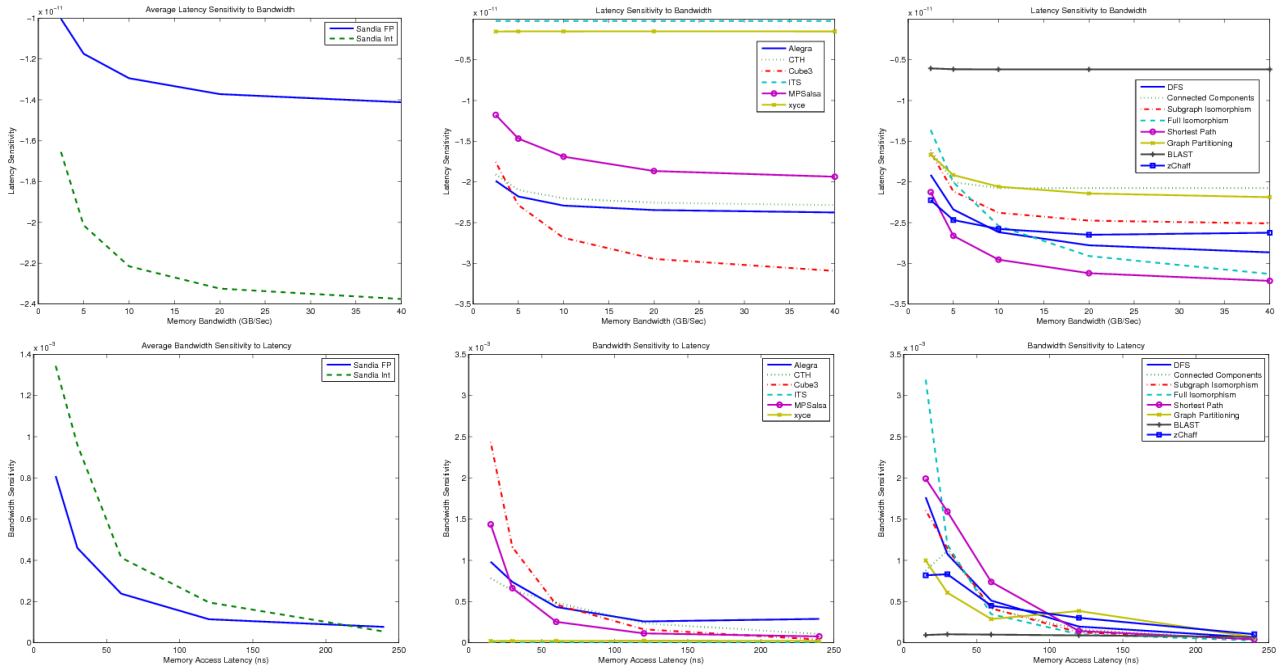
References

[1] Apple Architecture Performance Groups. *Computer Hardware Understanding Development Tools 2.0 Reference Guide for MacOS X*. Apple Computer Inc, July 2002.

[2] Luiz Andre Barroso, Kourosh Gharachorloo, and Edouard Bugnion. Memory system characterization of commercial workloads. In *Proceedings of the 25th annual international symposium on Computer architecture*, pages 3–14. IEEE Computer Society, 1998.

[3] Doug Burger and Todd Austin. *The SimpleScalar Tool Set, Version 2.0*. SimpleScalar LLC.

[4] Z. Cvetanovic and D. Bhandarkar. Characterization of alpha AXP performance using TP and SPEC workloads. In *Proceedings of the 21ST annual international symposium on Computer architecture*, pages 60–70. IEEE Computer Society Press, 1994.



(a) Suite Mean **(b) Floating Point** **(c) Integer**

Figure 5. Latency and Bandwidth Sensitivity. It should be noted that bandwidth sensitivity is positive because increasing bandwidth increases performance while latency sensitivity is negative because increasing latency decreases performance. Part (a) of the figure depicts the average for each of the benchmark suites, while (b) shows the floating point applications and (c) the integer.

[5] Peter J. Denning. The working set model for program behavior. In *Proceedings of the first ACM symposium on Operating System Principles*, pages 15.1–15.12. ACM Press, 1967.

[6] J. Dongarra and P. Luszczek. Introduction to the HPC-Challenge Benchmark Suite. Technical Report ICL-UT-05-01, 2005.

[7] Domenico Ferrari. A generative model of working set dynamics. In *Proceedings of the 1981 ACM SIGMETRICS conference on Measurement and modeling of computer systems*, pages 52–57. ACM Press, 1981.

[8] Swathi Tanjore Gurumani and Aleksandar Milenkovic. Execution characteristics of SPEC CPU2000 benchmarks: Intel C++ vs. Microsoft VC++. In *Proceedings of the 42nd annual Southeast regional conference*, pages 261–266. ACM Press, 2004.

[9] John L. Hennessy and David A. Patterson. *Computer Architecture a Quantitative Approach*. Morgan Kaufmann Publishers, 2002.

[10] Kimberly Keeton, David A. Patterson, Yong Qiang He, Roger C. Raphael, and Walter E. Baker. Performance Characterization of a Quad Pentium Pro SMP using OLTP Workloads. In *Proceedings of the International Sympoisum on Computer Architecture*, pages 15–26, 1998.

[11] Peter M. Kogge, Jay B. Brockman, and Vincent Freeh. Processing-In-Memory Based Systems: Performance Evaluation Considerations. In *Workshop on Performance Analysis and its Impact on Design held in conjunction with ISCA, Barcelona, Spain*, June 27-28, 1998.

[12] Dennis C. Lee, Patrick Crowley, Jean-Loup Baer, Thomas E. Anderson, and Brian N. Bershad. Execution characteristics of desktop applications on windows NT. In *International Symposium on Computer Architecture*, pages 27–38, 1998.

[13] Jack L. Lo, Luiz Andre Barroso, Susan J. Eggers, Kourosh Gharachorloo, Henry M. Levy, and Sujay S. Parekh. An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors. In

International Symposium on Computer Architecture, pages 39–50, 1998.

[14] Ann Marie Grizzaffi Maynard, Colette M. Donnelly, and Bret R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *Proceedings of the Sixth International Conference on Architectural support for Programming Languages and Operating Systems*, pages 145–156. ACM Press, 1994.

[15] McCalpin, John D. *Stream: Sustainable memory bandwidth in high performance computers*, 1997.

[16] Sally A. McKee. Reflections on the memory wall. In *CF '04: Proceedings of the 1st conference on Computing frontiers*, page 162, New York, NY, USA, 2004. ACM Press.

[17] Richard C. Murphy. *Traveling Threads: A New Multithreaded Execution Model*. Ph.D. Dissertation, University of Notre Dame, May 2006.

[18] Richard C. Murphy, Jonathan Berry, William McLendon, Bruce Hendrickson, Douglas Gregor, and Andrew Lumsdaine. DFS: A Simple to Write Yet Difficult to Execute Benchmark. In *IEEE International Symposium on Workload Characterization 2006 (IISWC06)*, October 25-27, 2006.

[19] Richard C. Murphy and Peter M. Kogge. On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and its Implications. *To Appear In IEEE Transactions on Computers*.

[20] Richard C. Murphy, Arun Rodrigues, Peter Kogge, and Keith Underwood. The implications of working set analysis on supercomputing memory hierarchy design. In *The 2005 International Conference on Supercomputing*, June 20-22, 2005.

[21] Leonid Oliker, Andrew Canning, Jonathan Carter, John Shalf, and Stephane Ethier. Scientific Computations on Modern Parallel Vector Systems. In *Proceedings of Supercomputing*, page 10, 2004.

[22] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface, 2ed.* Morgan Kaufmann Publishers, 1997.

[23] Arun Rodrigues, Richard Murphy, Peter Kogge, and Keith Underwood. Characterizing a New Class of Threads in Scientific Applications for High End Supercomputers. In *Proceedings of the 18th Annual ACM International Conference on Supercomputing*, pages 164–174, June 26-July 1 2004.

[24] Arun F. Rodrigues. *Programming Future Architectures: Dusty Decks, Memory Walls, and the Speed of Light*. Ph.D. Dissertation, University of Notre Dame, 2006.

[25] Juan Rodriguez-Rosell. Empirical working set behavior. *Communications of the ACM*, 16(9):556–560, 1973.

[26] Rafael Saavedra and Alan Smith. Analysis of benchmark characteristics and benchmark performance prediction. *ACM Transactions on Computer Systems*, 14(4):344–84, 1996.

[27] D. R. Slutz and I. L. Traiger. A note on the calculation of average working set size. *Communications of the ACM*, 17(10):563–565, 1974.

[28] Jeffrey S. Vetter and Andy Yoo. An Empirical Performance Evaluation of Scalable Scientific Applications. In *Proceedings of Supercomputing*, pages 1–18, 2002.

[29] Jonathan Weinberg, Michael McCracken, Alan Snavely, and Erich Strohmaier. Quantifying Locality In The Memory Access Patterns of HPC Applications. In *Supercomputing 2005*, page 50, November, 2005.

[30] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall: implications of the obvious. *SIGARCH Comput. Archit. News*, 23(1):20–24, 1995.