DOE's Institute for Advanced Architecture and Algorithms: an application-driven approach

Richard C. Murphy
Sandia National Laboratories, PO Box 5800, MS-1319, Albuquerque, NM 87185
E-mail: rcmurph@sandia.gov

Abstract. This paper describes an application driven methodology for understanding the impact of future architecture decisions on the end of the MPP era. Fundamental transistor device limitations combined with application performance characteristics have created the switch to multicore/multithreaded architectures. Designing large-scale supercomputers to match application demands is particularly challenging since performance characteristics are highly counter-intuitive. In fact, data movement more than FLOPS dominates. This work discusses some basic performance analysis for a set of DOE applications, the limits of CMOS technology, and the impact of both on future architectures.

1. Introduction

As the MPP era draws to a close, the energy inefficiency of modern supercomputer architectures is rapidly becoming the limiting factor for architectural scalability. The MPP era's total reliance on commodity processors that are optimized for less challenging application spaces has created energy inefficiencies throughout the system. Further, fundamental transistor device physics has forced scaling in the form of increased parallelism at the chip level, rather than the exponential increase in single thread performance that characterized the MPP era. Indeed, the transition to multicore architectures is driven entirely by the aging of the CMOS fabrication process. Without a well-defined successor capable of Moore's Law scaling, the transistor limits the potential of future supercomputers deployed in energy constrained environments. As a result, changes to computer architecture are required to enable newer, more highly-scalable applications. However, choosing the right architectural path is extremely difficult. Generally, computer architects do everything possible to hide the details of the hardware implementation from the programmer: cache sizes, branch prediction policies, out-of-order execution mechanisms, and thread scheduling at the hardware level are all examples of critical architectural mechanisms and structures that are typically observed only indirectly by the programmer.

More worrisome still, each of these structures that architects use to boost single thread performance and hide the depths and complexities of modern communication hierarchies (memory or interconnect) consumes energy that could otherwise be available for the computation if the programmer could exploit simpler, lighter-weight mechanisms. It has long been argued that these mechanisms should be exposed to the programmer[9, 5, 1, 4], but the transition to multithreaded/multicore architectures leaves little choice in an energy constrained environment.

This paper presents a synthesis of application analysis results derived from simulation, analysis tools, and real hardware. The remainder of the paper is structured as follows: Section

2 discusses some of the applications studied in this work; Section 3 examines the limits of CMOS transistors; Section 4 provides results to show that performance bottlenecks tend to be latency and concurrency dominated; and Section 5 describes the impact of these trends on future architectures.

2. Applications Discussed In This Work

The application studies referenced in this work are divided into two basic application classes: physics codes, which tend to be floating point oriented; and informatics codes, which tend to be more integer oriented. The full application analysis and simulation methodology has been previously described[7, 6, 8]. The results are summarized in this paper.

2.1. Physics Applications

Broadly speaking, the physics application suite analyzed by Sandia are large-scale MPI applications, and represent real world 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.

This suite of applications is somewhat more narrow than those of interest to the Office of Science, but future work in the DOE Institute for Advanced Architecture will apply the same fundamental analysis techniques to specific Office of Science applications.

2.2. Informatics Applications

Recently, new large-scale applications in the area of informatics have emerged as important and substantially different from physics applications that have been the core of supercomputing over the past three decades. These applications tend to be more integer-oriented, and represent problems in discrete math. They are typically less structured (in terms of memory access patterns), and are often techniques used to analyze the output of a physical simulation, or to examine real world data sets in an attempt to find patterns or form hypotheses. Additionally, because Informatics applications tend to be less structured in nature, they are written using a variety of programming models. The application set discussed are:

- **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.

These applications have the potential to significantly change the methodology used in computer-driven science, and fundamentally represent a new and disruptive use of high performance computing resources.

3. CMOS Transistor Limitations

The switch to multicore is fundamentally driven by a change in the CMOS transistor that forms the implementation technology for commodity processors. Power dissipation (manifesting itself as heat which must be removed from the device) can be broken into two components: Static Power Dissipation (P_{stat}), which is dissipated continuously while the circuit is powered and Dynamic Power Dissipation (P_{dyn}) which is dissipated during switching. Although increasingly important, static power dissipation cannot be addressed by the computer architect (other than by powering off parts of the machine not in use), and is primarily a function of the fabrication process. It is defined as:

$$P_{stat} + I_{leakage} V_{dd}$$

Where $I_{leakage}$ is the leakage current of the device, and V_{dd} is the voltage. It should be noted that while the computer architect cannot do much to control leakage current, the system architect and environment in which the machine is deployed can. At the junction, leakage is generally caused by thermally generated carriers, and *increases* exponentially as temperature rises. In fact, at 85°C, leakage current increases by a factor of 60 over room temperature values. These factor and increased failure rates must be taken into account in the “hotter running” compute environments that have become a topic of recent significant interest.

The majority of power dissipation is dynamic, and is defined as as:

$$P_{dyn} = C_L V_{dd}^2 f$$

Where C_L is the capacitance of the entire circuit, V_{dd} the voltage, and f the switching frequency (proportional to the clock frequency since not all devices will switch every cycle). This equation is critically important in describing the switch to multicore.

Moore’s Law as originally defined demands an exponential increase in the number of transistors, which increases C_L. Specifically, for a given area, the total capacitance is the sum of the capacitance of each device. Although capacitance per device decreases as the feature size decreases (decreasing the individual transistor's contribution to C_L), the number of devices is increasing exponentially. Similarly, until the multicore era the frequency increased substantially in each generation. To keep the total dynamic power dissipation essentially constant (so that chips could be cooled with relatively inexpensive heat syncs at roughly room temperature), the equation was balanced by *decreasing* V_{dd}, which as it approaches 1 volt is nearing a fundamental limit (in fact, the rate at which V_{dd} has decreased has already begun to flatten).

Although there are additional manufacturing benefits to moving to multicore architecture, including simplifying the verification of designs, this fundamental technology limit is the real driver. The devices are capable of higher clock rates, at higher densities, but engineering a system to cope with the power dissipation is nearly impossible. The era is characterized as an increasing bounty of transistors (Moore's original observation), but the inability to increase single thread performance (through faster clock rates). The inevitable result is more parallelism.

Of course the end of the CMOS scaling curve in the next decade or so will cause additional fundamental physical limitations, unless a solution can be found.

4. Performance Bottlenecks Tend to Be Counterintuitive

The following section discusses basic application properties, and performance bottlenecks in the network and memory system.

4.1. Real Floating Point Applications Do Less Floating Point Than Expected

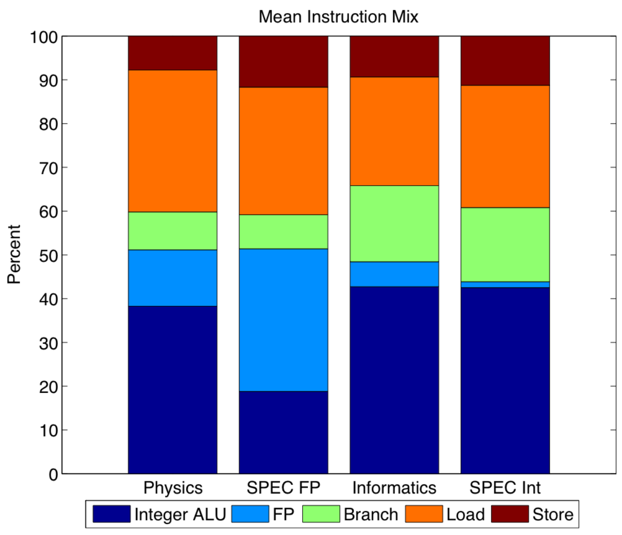


Figure 1. Instruction Mix

The instruction mix for the Physics and Informatics suites is depicted in Figure 1, and compared to the industry standard SPEC CPU 2000 suite. While SPEC FP averages approximately one floating point instruction in three, real Sandia Physics codes average only 12% (and is typically much less than 10%) because they exhibit more complex memory addressing patterns, which, in turn require more integer instructions to calculate the addresses. In fact, nearly 85% of integer instructions are calculating memory addresses, with the remaining 15% split between real integer data and branch (boolean) support.

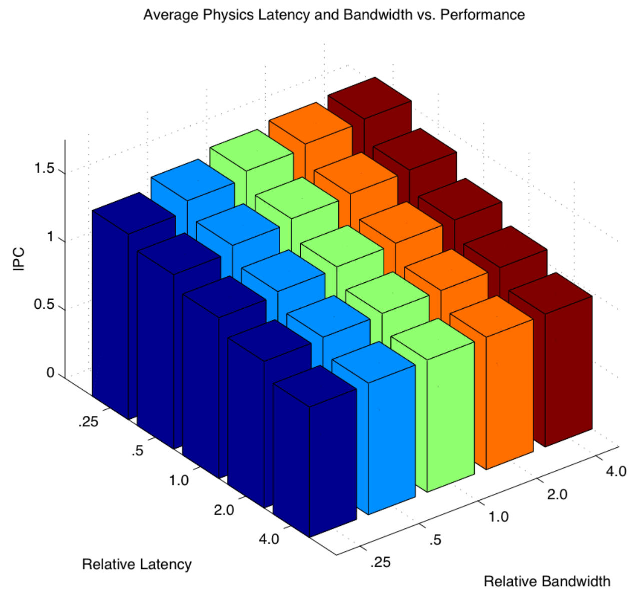
The Informatics applications are very different from the Physics codes as well, suggesting the possibility of two different supercomputer architecture classes in the future. They perform 15% fewer memory references, and those references are much more likely to miss the cache, which stresses the memory subsystem. Additionally, the Informatics applications perform nearly twice as many branches as their Physics counterparts, leading to relatively smaller basic blocks. (This result is unsurprising due to the complex floating point calculations typically performed by the Physics codes[9].) This combination makes the Informatics codes more difficult to scale on modern superscalar processors than their physics counterparts.

4.2. Latency and Concurrency Matter More than Bandwidth

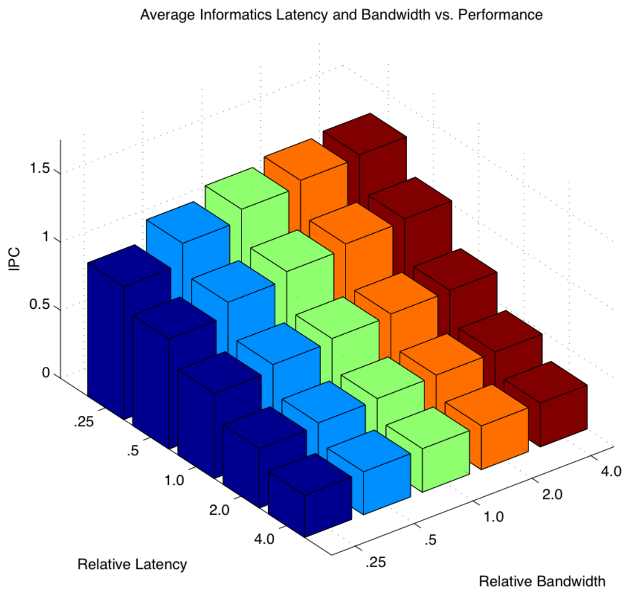
Data movement systems are typically rated by bandwidth, but most evidence points to applications being more sensitive to latency or concurrency. This can be simplistically described by Little's Law (from Economics) which says that:

$$throughput = \frac{concurrency}{latency}$$

Particularly in the case of memory systems, increasing the number of memory requests outstanding (currently limited by hardware and instruction sequences) or decreasing the latency required to receive a response has significantly more impact than changing the bandwidth.



(a)



(b)

Figure 2. Latency and Bandwidth Sensitivity for (a) Physics Applications and (b) Informatics Applications

Figure 2 shows the change in Instructions Per Cycle (IPC, the computer architect's typical measure of performance) as relative latency and bandwidth are altered (the center bar being the latency and bandwidth of a typical processor's memory system. The full results and methodology have been described previously[6]. The results demonstrate that both application suites are significantly more sensitive to latency than bandwidth. A decrease in bandwidth by half on either suites affects performance by less than 5%, while doubling the memory latency halves performance[1]

[1] It should be noted that this bandwidth measure is more strict than that presented for a typical memory part, which will also include a decrease in latency in subsequent generations.

These result combined are confirmed by observations from performance counters and other real system measurements going back to ASCI Red that demonstrates that real applications have difficult saturating the memory bus on a modern architecture because address generation (the integer instructions discussed in Section 4.1) is the bottleneck more than typical memory performance.

This is further demonstrated by typical out-of-order processors that are designed to mask the multi-cycle latency of a Level 1 cache **hit**, more than the latency of a memory access (due to limitations of both the incoming instruction stream and the implementation technology).

The performance of the two application suites is also significantly different. The Physics application achieve reasonable performance on the simulated out-of-order execution unit, achieving an IPC of 1.22 for the base case. The processor is capable of retiring 4 instructions per cycle, so the achieved IPC is approximately 30% of maximum. However, any IPC greater than one is quite good for a typical application suite. In contrast, the baseline case for the Informatics suite is only 0.70, indicating that the applications are significantly more memory intensive, causing the processor additional idle time. Key graph algorithms, including both isomorphism problems and connected components show IPCs significantly less than 0.5. Aside from being less than an eighth of achievable performance, the results indicate that significant power is being wasted on high clock rates for those applications.

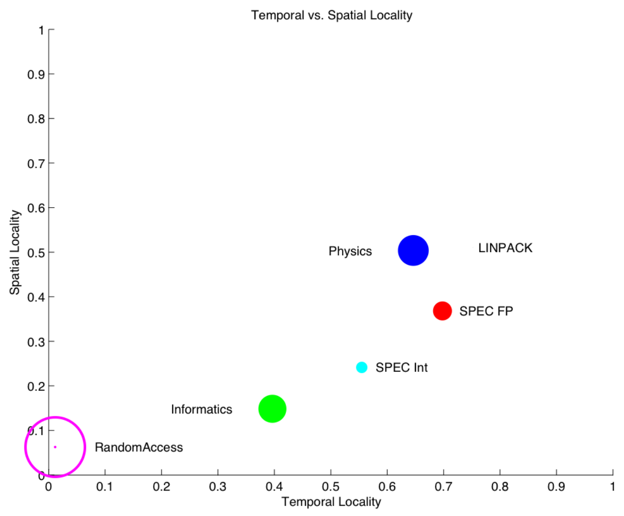


Figure 3. Application Memory Performance Properties

Figure 3 describes the fundamental memory performance properties of each benchmark suite, and has been fully described previously[7]. The spatial locality describes the probability that data near previously referenced data will be referenced. The temporal locality describes the probability that data already referenced will be reused. And the relative size of the dots represents the amount of unique data that each application suite consumes over a fixed interval of instructions. These fundamental application properties dictate the performance observed in Figure 2, with applications consuming more data, and doing so in a more random fashion (i.e., closer to the origin on the graph) showing similarly low performance in comparison to applications with smaller data sets consumed in a more structured pattern. These fundamental application properties determine the class of architecture required to provide high performance at low power more than any other issue.

4.3. Network Performance

These issues are much more difficult to measure for the network, but preliminary results and prior experience show the importance of network injection rate vs. raw bandwidth.

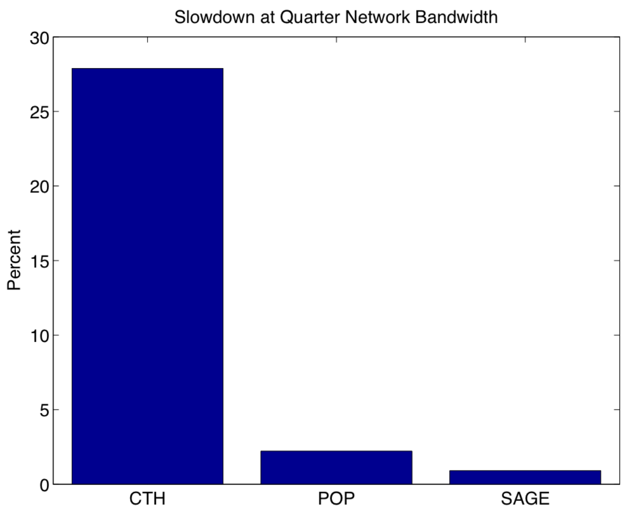


Figure 4. Measurement of Bandwidth Restricted Performance on Red Storm. This is a recast of data collected by Kurt Ferreira at Sandia, a subset of which is discussed in [3]

The trade-off between latency, bandwidth, and message rates are difficult to measure for modern networks, and the trade-offs appear to be highly application dependent. This state of affairs makes drawing firm conclusions on minimal network design difficult, which may lead to either poor application performance for a subset of applications or over-provisioned networks for some subset of applications.

Figure 4 shows the effects of running three applications on Sandia's Red Storm machine, with bandwidth degraded in the mesh by 75%, while not degrading small message latency or injection rate. The mesh is intentionally capable of approximately twice the bandwidth of the link between processor and NIC to offer more stable performance in the face of message contention, whether from non-3D communication patterns or poor application and process placement. The applications shown are CTH, SAGE, and POP. CTH and Sage were run at 2,048 nodes, and POP at 2,500. CTH and Sage were run with weak scaling data sets, and POP as a strong scaling problem.

POP is limited by a combination of message rate and latency, and it is therefore not surprising that reducing network bandwidth does not impact application performance. SAGE, like POP, appears to be immune to bandwidth changes, although it is unclear whether this is problem specific. CTH, on the other hand, shows a 27% performance degradation when mesh bandwidth is reduced to half of injection bandwidth, and is traditionally thought to be insensitive to latency and message rate. The difference between behavior of POP and CTH suggests a significant challenge for hardware design – a need to balance hardware cost, energy usage, and performance across a wide variety of applications.

In an energy optimized environment, one could envision a system that dynamically scales network bandwidth performance to meet both application and overall system demands (contention, I/O, etc.) when full topological bandwidth is not required for the current set of running applications.

5. Impact on Future Architectures: Multicore, Multithreading

Section 3 described the fundamental technology changes pushing multicore processors. The discussion of IPC in Section 4.2 shows that the processor is tremendously underutilized, especially for applications with very large, sparse data sets. Because the problem is fundamentally one of latency, there are only two basic solutions: avoiding the latency through faster devices, memory hierarchies, and caches; or tolerating the latency via mechanisms such as multiple outstanding loads, out-of-order execution, vector pipelines, or threads.

Avoiding latency via faster devices is fundamentally an economically and technologically driven process, and often provides a "one time" benefit rather than a sustained and continuous benefit over multiple generations. For example, cheap 3D integration of memory onto the processor would reduce wire lengths between the processor and memory from centimeters to microns, providing a one-time latency reduction.

Architectural techniques for avoiding latency such as caching also show diminishing returns. Caches place frequently used items closer to the processor, but become less effective proportional to cost as they increase in size. They are also large consumers of energy.

In terms of memory toleration, many of the classic techniques have reached their limits. Given the relatively aggressive out-of-order execution in modern processors and the observed IPCs from Figure 2, it is clear that there is not much room for improvement. In fact, most processors are moving towards simpler latency toleration mechanisms because the power dissipation for out-of-order execution is so high compared with the relatively small benefits. As discussed previously, most applications fail to fully utilize memory system bandwidth, meaning that the problem of generating more outstanding loads is not due to a hardware limitation, but fundamental application properties. For the Informatics suite in particular, the pointer chasing requires the results from one load to be available before another load can be generated, limiting architectural options for solving the problem.

The remaining option for power-efficient latency toleration is threads, and it can be seen in the successes of early throughput-oriented architectures such as the Sun Niagara. Threads are:

- Relatively inexpensive to implement in hardware, requiring additional memory structures to support larger register files, but not requiring complex logic as is required for out-of-order execution;
- Efficient in that switching to another thread while the currently executing thread waits on a long latency event still allows "useful" work to be performed during the idle time (in contrast with techniques such as speculative execution which simply create heat when the architecture predicts an erroneous path of execution);
- Low power, primarily because useful work is always being done and they replace more complex hardware structures;
- Capable of fully utilizing the memory system by exposing more address generation streams to the programmer (if the programmer can exploit them), which can be seen in the trend toward fewer memory controllers per core in modern processors; and,
- Potentially plentiful, in that for the desktop and server space for which the processors are optimized there typically exists sufficient parallelism (even at the task level) to allow for multithreaded processors not to starve, which is potentially a significant application problem for future supercomputers.

Consequently, the trend towards multicore/multithreaded architectures is inevitable, especially in highly energy constrained environments. The DARPA Exascale Report[2] describes dominance of data movement in future exascale architecture energy budgets.

Acknowledgments
I would like to thank Kurt Ferreira for sharing his data used in the networking discussion, as well as Arun Rodrigues, Brian Barrett, Scott Hemmert, and Jim Ang for many useful discussions over the course of preparing this paper.

References
[1] Jay B. Brockman, Peter M. Kogge, Vincent Freeh, Shannon K. Kuntz, and Thomas Sterling. Microservers: A new memory semantics for massively parallel computing. In *ICS*, 1999.

[2] Peter M. Kogge (editor). *ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems.* University of Notre Dame, CSE Deptartment Technical Report TR-2008-13, September 28, 2008.

[3] Kurt Ferreira, Ron Brightwell, and Patrick Bridges. Characterizing Application Sensitivity to OS Interference Using Kernel-Level Noise Injection. In *Supercomputing 2008 (SC08), Austin TX, November 2008.*

[4] Mary Hall, Peter Kogge, Jeff Koller, Pedro Diniz, Jacqueline Chame, Jeff Draper, Jeff LaCoss, John Granacki, Apoorv Srivastava, William Athas, Jay Brockman, Vincent Freeh, Joonseok Park, and Jaewook Shin. Mapping Irregular Applications to DIVA, A PIM-based Data-Intensive Architecture. In *Supercomputing, Portland, OR*, November 1999.

[5] Shannon K. Kuntz, Richard C. Murphy, Michael T. Niemier, Jesus Izaguirre, and Peter M. Kogge. Petaflop Computing for Protein Folding. In *Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, VA*, March 12-14, 2001.

[6] Richard C. Murphy. On the Effects of Memory Latency and Bandwidth on Supercomputer Application Performance. In *IEEE International Symposium on Workload Characterization 2007 (IISWC2007)*, September 27-29, 2007.

[7] Richard C. Murphy and Peter M. Kogge. On the Memory Access Patterns of Supercomputer Applications: Benchmark Selection and its Implications. *IEEE Transactions on Computers*, 56(7):937–945, July 2007.

[8] 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.

[9] 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.