Science Driven Supercomputing Architectures: Analyzing Architectural Bottlenecks with Applications and Benchmark Probes

The Berkeley Institute for Performance Studies

S. Kamil, B. Kramer, L. Oliker, J. Shalf,
H. Shan, E. Strohmaier, K. Yelick
kayelick@lbl.gov

There is a growing gap between the peak speed of parallel computing systems and the actual delivered performance for scientific applications. In general this gap is caused by inadequate architectural support for the requirements of modern scientific applications, as commercial applications and the much larger market they represent, have driven the evolution of computer architectures. This gap has raised the importance of developing better benchmarking methodologies to characterize and to understand the performance requirements of scientific applications, to communicate them efficiently to influence the design of future computer architectures. This improved understanding of the performance behavior of scientific applications will allow improved performance predictions, development of adequate benchmarks for identification of hardware and application features that work well or poorly together, and a more systematic performance evaluation in procurement situations.

The Berkeley Institute for Performance Studies has developed a three-level approach to evaluating the design of high end machines and the software that runs on them: 1) A suite of representative applications; 2) A set of application kernels; and 3) Benchmarks to measure key system parameters. The three levels yield different type of information, all of which are useful in evaluating systems, and enable NSF and DOE centers to select computer architectures more suited for scientific applications. The analysis will further allow the centers to engage vendors in discussion of strategies to alleviate the present architectural bottlenecks using quantitative information. These may include small hardware changes or larger ones that may be out interest to non-scientific workloads. Providing quantitative models to the vendors allows them to assess the benefits of technology alternatives using their own internal cost-models in the broader marketplace, ideally facilitating the development of future computer architectures more suited for scientific computations. The three levels also come with vastly different investments: the benchmarking efforts require significant rewriting to effectively use a given architecture, which is much more difficult on full applications than on smaller benchmarks.

1 Performance Requirements

1.1 Categorizing Scientific Algorithms for Scientific Computing

National supercomputing centers have to serve a very large and diverse users base. While the field of scientific algorithms used in these different application areas is large and diverse, the algorithmic methods fall into a modest set of categories, coined the "Seven Dwarfs" by Philip

Colella. We show a slightly modified version of his list together with their key performance requirements:

1. Dense linear algebra: unit and strided memory accesses, high computational intensity
2. Sparse linear algebra (direct methods): indexed (scatter/gather) memory operations, low computational intensity (for elliptic problems) and communication latency sensitivity due to dependencies and load imbalance.
3. Unstructured meshes and iterative sparse methods: indexed (scatter/gather) memory operations, and low computational intensity
4. Spectral methods: strided memory accesses and high bisection bandwidth needs for global transpose operations in a parallel setting.
5. Particle methods: pointer-based data structures (trees), random memory accesses. This include both particle-mesh and particle-particle methods.
6. Block-structured meshes, regular and adaptive: unit and strided memory accesses via "stencil" operations, load imbalance (adaptive) and small messages.
7. Monte Carlo methods: random number generation independent parallelism.

While this list does not cover all algorithmic techniques, these methods and combinations of them dominate many scientific applications. Some centers may find the sorting or search or other non-numerical algorithms are important in their workload, and we have not studied Monte Carlo methods in detail, because they seem to provide little insight into system requirements.

2 Performance Metrics

Performance metrics are an important aspect of any procurement. Instead of elaborating in detail on specific metrics, we like to refer to the position paper about the NERSC SSP metric. The SSP metric is based on the individual performance values of a set of application benchmarks and can be used with any appropriate set of benchmarks, such as the one proposed in the following section. The purpose of science-driven architecture is inform system designers by drilling down into particular application needs through smaller, but related, benchmarks.

3 Potential Benchmark Codes

Application benchmarks for procurements are used to establish a common metric to normalize the performance of supercomputing systems in order to make them comparable during a procurement cycle. These benchmark suites typically attempt to reflect and model the anticipated workload of a given system and are founded on a characterization of the target workload.

While application benchmarks greatly improve the quality of system procurements, they are of limited benefit when considering technology alternatives or changes in the system balance that can significantly improve the cost-effectiveness of a system. Any single metric based on application performance alone only describes **what** the performance of the system is, but not **why** it is so, or **how** to improve it. For instance, is application communication performance limited by interconnect bandwidth or is it latency bound? Do communication libraries implementations show performance anomalies? Could alternative programming models provide better performance? These detailed questions are best answered by smaller kernel benchmarks or application-independent architectural benchmarks. An example of the use of categorization to identify applications and as well as kernel benchmarks is shown in the table below.

Category	Representative Application	Kernel/Probe Benchmark
Dense Matrices	MADCAP	Linpack
Sparse Matrices	*Nimrod, Omega3P*	SuperLU
Unstructured Meshes	Overflow-D	Bebop SPMV
Spectral Methods	FVCAM, Paratec	NAS FT, FFTW
Particle Methods	PMEMD, GTC	
Block-Structured Meshes	LBMHD, Cactus	NAS MG, Stencil Probe

Table 1: Application codes and parametric benchmarks that can be used to model their performance.

3.1 Application Benchmarks

The first component of our performance evaluation involves full application studies. The goal is to select applications that reflect important scientific problems, port and tune them on a variety of machines, and analyze the performance of emerging ultrascale computing architectures such as the Earth Simulator, BG/L and the Cray X1. A subset of the BIPS application suite is shown in Table 1, although we have not yet experiment with Omega3P and Nimrod. The applications include: MADCAP, a simulation of cosmic microwave background which uses dense linear algebra and has significant I/O requirements; FVCAM, the Community Atmosphere Model with the finite-volume solver option for the dynamics; GTC, a magnetic fusion application that uses the particle-in-cell approach to solve non-linear gyrophase-averaged Vlasov-Poisson equations; LBMHD3D, a plasma physics application that uses the Lattice-Boltzmann method to studymagneto-hydrodynamics; and PARATEC, a first principles materials science code that solves the Kohn-Sham equations of density functional theory to obtain electronic wavefunctions; Cactus, an astrophysics code that solves Einstein's equations; OVERFLOW-D, a CFD production code that solves the Navier-Stokes equations around complex aerospace configurations; PMEMD, a Life Sciences Molecular Dynamics code that uses a Particle Mesh method; SuperNova, an astrophysics simulation using Adaptive Mesh Refinement; Omega3P, an accelerator modeling code uses sparse iterative and direct solvers; and Nimrod, a magnetohydrodynamic fusion modeling code, uses sparse direct solvers. None of the characterizations are complete: each fo these codes may use other methods in the list in addition to the one lists. The point is that a relatively modest set of applications can cover the space of many numerical methods. These codes represent candidate ultra-scale applications that have the potential to fully utilize leadership-class computing systems and are detailed in a number of papers [13,14,15,16].

The performance analysis of these codes has produced surprising results. For instance, both the SX-6 and the Cray X1 employ vector architecture, the performance characteristics of the two systems are dramatically different. This provides a caution to gross generalizations about code performance based on high-level architectural model without regard to the full system implementation. Likewise, the performance differences between systems could be dramatic. For instance, the computation of boundary conditions for Cactus on many microprocessor based systems is so inconsequential that it does not typically appear on performance profiles. However, on the vector machines, the unvectorized boundary calculation was consistently one of the most expensive routines in the application. The extracted kernel of the Cactus GR solver would have missed this behavior – leading to some caution regarding the replacement of full application benchmarks with extracted kernels. The nonlinear performance behavior exhibited by many of the evaluated systems motivated the development of parametric probes that are capable of finding

these “performance cliffs” in modern parallel architectures. Such cliffs would be difficult to find with microbenchmarks that produces a single-valued result.

3.2 Application and Parameterized Probes

The last column of Table 1 shows a set of application kernels, which are easy to understand and can be used to study single architectural features in isolation from other system components. Some of these are standard benchmarks from NAS parallel benchmarks suite, since some applications are well-resented for either single processor or parallel performance.

A new class of *parametric benchmark probes* is being developed at BIPS, which have input parameters designed to reflect critical aspects for the performance of algorithms. Through variation of these characteristic parameters performance surfaces reflecting system behavior for a variety of conditions can easily be generated and visualized. Due to their compact size these probes are also easily usable with simulators for future systems. Overall, well designed parametric benchmark probes provide an ideal complement to application benchmarks alone. We give two examples here.

The *Stencil Probe* was developed as a proxy to applications such as Cactus and LBMHD that perform nearest-neighbor calculations on a multidimensional grid. The main computational kernel for these applications was extracted and built into an easily-modifiable probe that functions as a proxy to the full applications, yielding similar performance and providing a platform for developing optimizations without needing to port full applications. One optimization we explored was *cache blocking*, which subdivides the grid into cache-sized segments to improve memory hierarchy utilization. In the course of implementing cache blocking in the Stencil Probe, we found that although we were able to lower the number of cache misses, overall time-to-solution *increased*. We hypothesized that this optimization, while reducing cache misses, resulted in a non-prefetch-friendly memory access pattern. This was verified using an application-independent benchmark to measure prefetch behavior, which is describe below.

The *Bebop SPMV benchmark* was designed to provide a realistic, synthetic sparse matrix benchmark [18]. The benchmark performs a Sparse Matrix-Vector Multiplication, a key component of most iterative methods. The NAS suite include a Conjugate Gradient benchmark for this purpose, but uses a randomly generated sparse matrix which has led performance results and algorithms designs that to not reflect the best practice for real applications. The Bebob benchmark uses a set of randomly generated matrices, but they contain dense sub-blocks which are controllable by a benchmark parameter. We have found that on single processor machines this benchmark matches that of real matrices; work on the parallel benchmark is ongoing.

3.3 Generic Benchmarkings

Generic benchmarking probes explore features of the network, memory, and processor architecture to show features and bottlenecks of the each machine. They can be used to compare absolute performance across machines, as well as discontinuities in the performance profile that may reveal challenges in tuning full applications for the machine. In the BIPS project we have developed two such benchmarks, which complement many other in the literature.

3.3.1 APEX-Map

APEX-Map is a *memory access probe* that was developed as part of the Application Performance Characterization (APEX) project [6]. The execution profile of APEX-Map can be tuned using a

set of input parameters to match the characteristics of a chosen scientific application --- allowing it be used as a proxy for the performance behavior of the underlying codes.

Our initial work on APEX-Map focused mainly on the cache and main memory levels of a single processor memory system. An underlying assumption is that the data access pattern of most codes can be described as several concurrent streams of addresses, which in turn can be characterized by a single set of performance parameters, including regularity, data-set size, spatial locality, and temporal locality. Regularity refers to the data access patterns, where the two extreme cases being random access and regular-strided access. Data-set size relates to the total volume of memory accessed --- an increasingly important factor as the complexity of memory hierarchies continues to grow in modern system architectures. The spatial locality parameter controls the number of contiguous memory locations accessed in succession. Finally, temporal locality refers to the average re-use of data items, and is defined independently of hardware concepts such as cache size. Temporal locality is controlled in the probe using a novel approach based on generation of a pseudo-random address stream, where increasing the probability of a repeated address in the stream can increase the temporal locality.

Early experiments examined the validity and accuracy of the Apex-MAP approach on six processors that are common in high end computing, including superscalar and vector processors using five scientific kernels radix sorting, FFT, matrix multiplication, nbody simulation, and conjugate gradient. The results showed that application performance could be captured to within 25% across the suite of application codes for a variety of memory sizes, using a few simple parameters with up to two simulated memory streams [7]. An example for the performance surfaces produces with APEX-Map on a single processor is show in figure 1. It compares an IBM Power4 processor and a Cray X1 processor and the achieved bandwidth numbers clearly highlight the different design principles for these two processors.

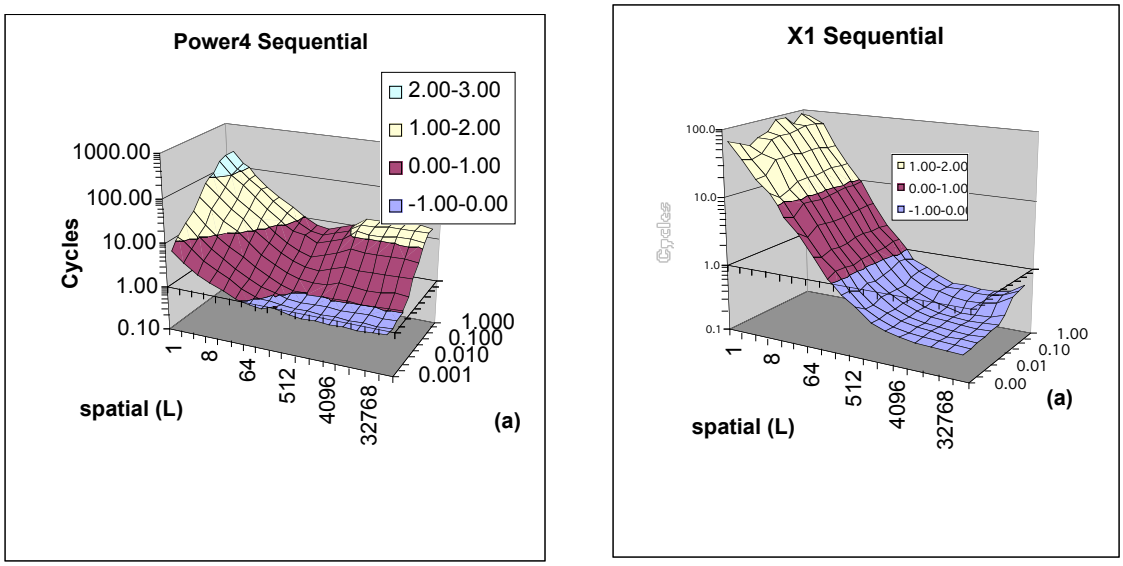

Figure 1: Sequential Performance Surface generated with Apex-Map on an IBM SP Power4 processor and a Cray X1 system. Horizontal axes are L=vectorlength; a= characteristic exponent for temporal locality; vertical axis is bandwidth in MB/s.

We have also performed work showing that the ideas in APEX-Map can be directly extended to include inter-process communication, thereby capturing the behavior of parallel systems. For parallel execution the same basic definitions of temporal and spatial locality can be used. The resulting code permits to explore the whole performance space of scientific applications and can be used to generate characteristic performance surfaces on large-scale multiprocessors. Figure 2 shows an example of the parallel APEX-Map results on a IBM Power3 system, where bandwidth varies dramatically, but smoothly, as temporal (the "a" axis) or spatial (the "L" axis) are varied.

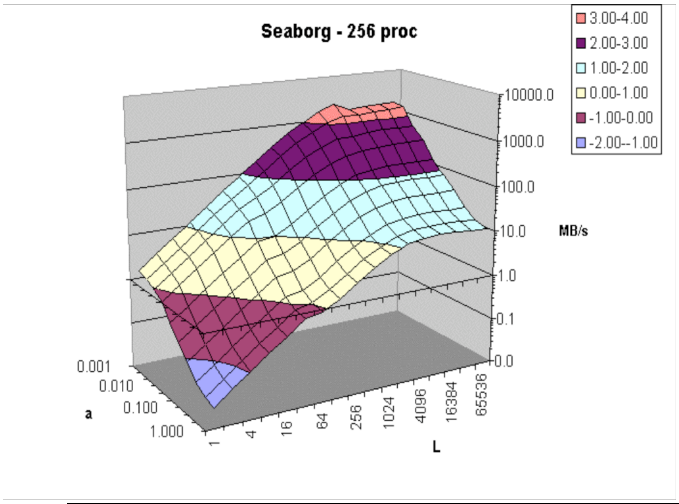

Figure 2: Parallel Performance Surface generated with Apex-Map on a 256 Processors IBM SP Power3. Horizontal axes are L=vectorlength and temporal locality; vertical axis is global bandwidth in MB/s.

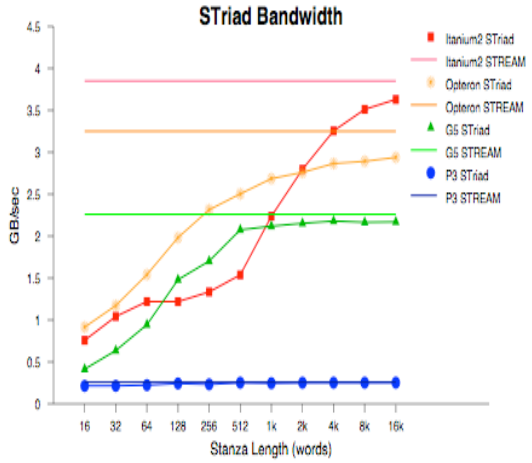

Figure 3: STriad benchmark showing memory bandwidth as a function of stanza length (length of a unit stride run between jumps in memory).

3.3.2 Stanza Triad

Our investigations into the stencil probe and mesh-based applications led to the development of Stanza Triad, a smaller microbenchmark geared towards measuring prefetch effects. The Stanza Triad measures how many consecutive elements must be accessed in a stream in order to reach close to peak performance. The chart below shows that small numbers of consecutive accesses results in poor performance; by chopping our grid into cache blocks, we were in fact reducing overall performance by using a prefetch-unfriendly access pattern.

With the insight gained by our small architectural microbenchmark, we then implemented cache blocking in the Stencil Probe but insured that the unit-stride dimension was not blocked. As a result, the memory access pattern was much more prefetch-friendly due to the long stride-1 accesses. This resulted in performance improvements over the non-blocked stencil code. Finally, we then implemented cache blocking into our applications, and, as the chart below shows, improved performance by an average of 21% in our test cases.

4 References:

[1] NAS Parallel Benchmarks: http://www.nas.nasa.gov/Software/NPB/

[2] SPEC: http://www.spec.org/

[3] HPCC: http://icl.cs.utk.edu/hpcc/

[4] STREAM: http://www.cs.virginia.edu/stream/

[5] MAPS: http://www.sdsc.edu/PMaC/MAPs/maps.html

[6] APEX-Map: http://ftg.lbl.gov/twiki/bin/view/FTG/ApeX

[7] E. Strohmaier, H. Shan, *Architecture Independent Performance Characterization and Benchmarking for Scientific Applications*. International Symposium on Modeling, Analysis and Simulation of Computer and Telecom. Systems, Volendam, The Netherlands, Oct. 2004

[8] Gorden Griem, Leonid Oliker, John Shalf, Katherine Yelick, "*Identifying Performance Bottlenecks on Modern Microarchitectures using an Adaptable Probe*", 3rd International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS), 2004.

[9] Linpack: http://www.top500.org/lists/linpack.php

[10] Shoaib Kamil, John Shalf, Parry Husbands, Leonid Oliker, Kaushik Datta, Paul Hargrove, Katherine Yelick, "*Optimizing Stencil Computations on Modern Architectures,*" MSP 2005.

[11] James Demmel, Jack Dongarra, Victor Eijkhout, Erika Fuentes, Antoine Petitet, Richard Vuduc, R. Clint Whaley, Katherine Yelick. "Self-Adapting Linear Algebra Algorithms and Software, *Proceedings of the IEEE, Special Issue on Program Generation, Optimization, and Adaptation*, 2005, to appear.

[12] The Berkeley Institute for Performance Studies. http://crd.lbl.gov/html/bips.html

[13] L. Oliker, J. Borrill, J. Carter, D. Skinner, R. Biswas, "Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms",International Conference on Parallel Processing: ICPP 2005. *(Nominated: Best paper award)*

[14] L. Oliker, R. Biswas, "Parallelization of a Dynamic Unstructured Application using Three Leading Paradigms",Supercomputing '99, 1999. *(Winner: Best Paper Award)*

[15] L. Oliker, A. Canning, J. Carter, J. Shalf, and S. Ethier, "Scientific Computations on Modern Parallel Vector Systems", Supercomputing 2004. *(Nominated: Best paper award)*

[16] S.A. Kamil, J. Shalf, L. Oliker, D. Skinner," Understanding Ultra-Scale Application Communication Requirements," IEEE International Symposium on Workload Characterization (IISWC) Austin Texas, October 6-8, 2005.

[17] H.D. Simon, W.T. Kramer, W. Saphir, J. Shalf, D.H. Bailey, L. Oliker, M. Banda, C. W. McCurdy, J. Hules, A. Canning, M. Day, P. Colella, D. Serafini, M.F. Wehner, P. Nugent, "Science-Driven System Architecture: A New Process for Leadership Class Computing," *Journal of the Earth Simulator,* vol 2, pp 2-10, March 2005.

[18] The Berkeley Benchmarking and Optimization Group. http://bebop.cs.berkeley.edu