Chapter 1

Performance Evaluation and Modeling of Ultra-Scale Systems

Leonid Oliker, Rupak Biswas,
Rob Van der Wijngaart, David Bailey, Allan Snavely

1.1 Introduction

The growing gap between sustained and peak performance for full-scale complex scientific applications on conventional supercomputers is a major concern in high performance computing (HPC). The problem is expected to be exacerbated by the end of this decade, as mission-critical applications will have computational requirements that are at least two orders of magnitude larger than current levels. In order to continuously increase raw computational power and at the same time substantially reap its benefits, major strides are necessary in hardware architecture, software infrastructure, and application development. The first step toward this goal is the accurate assessment of existing and emerging HPC systems across a comprehensive set of scientific algorithms. In addition, high-fidelity performance modeling is required to understand and predict the complex interactions among hardware, software, and applications, and thereby influence future design trade-offs. This survey article discusses recent performance evaluations of state-of-the-art ultra-scale systems for a diverse set of scientific applications, including scalable compact synthetic benchmarks and architectural probes. In addition, performance models and program characterizations from key scientific areas are described.

1.2 Modern High-Performance Ultra-Scale Systems

We begin by briefly describing the salient features of the leading HPC parallel architectures that are examined in this chapter.

1.2.1 Seaborg (Power3)

The Power3 was first introduced in 1998 as part of IBM's RS/6000 series. Each 375 MHz processor contains two floating-point units (FPUs) that can issue a multiply-add (MADD) per cycle for a peak performance of 1.5 Gflop/s. The Power3 has a pipeline of only three cycles, thus using the registers more efficiently and diminishing the penalty for mispredicted branches. The out-of-order architecture uses prefetching to reduce pipeline stalls due to cache misses. The CPU has a 32KB instruction cache, a 128KB 128-way set associative L1 data cache, and an 8MB four-way set associative L2 cache with its own private bus. Each SMP node consists of 16 processors connected to main memory via a crossbar. Multi-node configurations are networked via the Colony switch using an omega-type topology. The Power3 experiments reported here were conducted on Seaborg, the 380-node IBM pSeries system running AIX 5.1 and operated by Lawrence Berkeley National Laboratory (LBNL). Additional Power3 experiments were performed on Blue Horizon, a 144-node system located at San Diego Supercomputer Center (SDSC).

1.2.2 Cheetah (Power4)

The pSeries 690 Power4 is the latest commercially-available generation of IBM's RS/6000 series. Each 32-way SMP consists of 16 Power4 chips (organized as 4 MCMs), where a chip contains two 1.3 GHz processor cores. Each core has two FPUs capable of a fused MADD per cycle, for a peak performance of 5.2 Gflop/s. The superscalar out-of-order architecture can exploit instruction level parallelism through its eight execution units; however a relatively long pipeline (six cycles) is necessitated by the high frequency design. Each processor contains its own private L1 cache (64KB instruction and 32KB data) with prefetch hardware; however, both cores share a 1.5MB unified L2 cache. The L3 is designed as a stand-alone 32MB cache, or to be combined with other L3s on the same MCM to create a larger 128MB interleaved cache. The Power4 experiments were performed on Cheetah, the 27-node IBM pSeries 690 system running AIX 5.1 and operated by Oak Ridge National Laboratory (ORNL). The system employs the Federation (HPS) interconnect, with two adaptors per node.

1.2.3 Ram (Itanium2)

The SGI Altix is designed as a cache-coherent, shared-memory multiprocessor system. The computational building block consists of four Intel Itanium2 processors, local memory, and a two-controller ASIC called the SHUB. The 64-bit architecture operates at 1.5 GHz and is capable of issuing two MADDs per cycle for a peak performance of 6.0 Gflop/s. The memory hierarchy consists of 128 FP registers and three on-chip data caches (32KB L1, 256KB L2, and 6MB L3). The Itanium2 cannot store FP data in L1, making register loads and spills a potential source of bottlenecks; however, the relatively large register set helps mitigate this issue. The superscalar processor performs a combination of in-order and out-of-order instruction execution referred to as Explicitly Parallel Instruction Computing (EPIC).

The Altix platform uses the NUMAlink3 interconnect, a high-performance custom network in a fat-tree topology that enables the bisection bandwidth to scale linearly with the number of processors. It implements a cache-coherent, nonuniform memory access (NUMA) protocol directly in hardware. A load/store cache miss causes the data to be communicated via the SHUB at a cache-line granularity and automatically replicated in the local cache. Additionally, one-sided programming languages can be efficiently implemented by leveraging the NUMA layer. The Altix experiments reported here were performed on Ram, the 256-processor system at ORNL, running 64-bit Linux version 2.4.21 and operating as a single system image. The next-generation Altix 3700 is the building block of the Columbia system, currently the world's fourth-most powerful supercomputer [43] located at NASA Ames Research Center (ARC).

1.2.4 Earth Simulator (SX-6+)

The vector processor of the Japanese Earth Simulator (JES) uses a dramatically different architectural approach than conventional cache-based systems. Vectorization exploits regularities in the computational structure of scientific applications to expedite uniform operations on independent data sets. The 500 MHz JES processor (an enhanced version of the NEC SX-6) contains an 8-way replicated vector pipe capable of issuing a MADD each cycle, for a peak performance of 8.0 Gflop/s. The processors contain 72 vector registers, each holding 256 64-bit words. For non-vectorizable instructions, the JES has a 500 MHz scalar processor with a 64KB instruction cache, a 64KB data cache, and 128 general-purpose registers. The four-way superscalar unit has a peak of 1.0 Gflop/s (an eighth of the vector performance) and supports branch prediction, data prefetching, and out-of-order execution.

Like traditional vector architectures, the JES vector unit is cache-less; memory latencies are masked by overlapping pipelined vector operations with memory fetches. The main memory chip uses a specially developed high speed DRAM called FPLRAM (Full Pipelined RAM) operating at 24ns bank cycle time. Each SMP contains eight processors that share the node's memory. The JES is one of the world's most powerful supercomputers [43], and consists of 640 SMP nodes connected through a custom single-stage crossbar. This high-bandwidth interconnect topology provides impressive communication characteristics, as all nodes are a single hop from one another. The 5120-processor JES runs Super-UX, a 64-bit Unix operating system based on System V-R3 with BSD4.2 communication features. As remote JES access is not available, the reported experiments were performed during the authors' visit to the Earth Simulator Center located in Yokohama, Japan, in December 2003.

1.2.5 Phoenix (X1)

The Cray X1 combines traditional vector strengths with the generality and scalability features of modern superscalar cache-based parallel systems. The computational core, called the single-streaming processor (SSP), contains two 32-stage vector pipes running at 800 MHz. Each SSP contains 32 vector registers holding

64 double-precision words, and operates at 3.2 Gflop/s peak for 64-bit data. The SSP also contains a two-way out-of-order superscalar processor running at 400 MHz with two 16KB instruction and data caches. Like the SX-6, the scalar unit operates at 1/8-th the vector performance, making a high vector operation ratio critical for effectively utilizing the underlying hardware.

The multi-streaming processor (MSP) combines four SSPs into one logical computational unit. The four SSPs share a 2-way set associative 2MB data Ecache, a unique feature that allows extremely high bandwidth (25–51 GB/s) for computations with temporal data locality. MSP parallelism is achieved by distributing loop iterations across each of the four SSPs. An X1 node consists of four MSPs sharing a flat memory, and large system configurations are networked through a modified 2D torus interconnect. The torus topology allows scalability to large processor counts with relatively few links compared with fat-tree or crossbar interconnects; however, this topological configuration suffers from limited bisection bandwidth. The X1 has hardware supported globally addressable memory which allows for efficient implementations of one-sided communication libraries (SHMEM, MPI-2) and implicit parallel programming languages (UPC, CAF). All reported X1 results were obtained on Phoenix, the 256-MSP system running UNICOS/mp 2.4 and operated by ORNL.

1.3 Architecture Evaluation using Full Applications

This section presents synopses of six full-scale scientific applications that have been recently used in evaluating these ultra-scale HPC systems.

1.3.1 Materials Science

PARATEC (PARAllel Total Energy Code [35]) is a materials science code that performs ab-initio quantum-mechanical total energy calculations using pseudopotentials and a plane wave basis set. Forces can be easily calculated and used to relax the atoms into their equilibrium positions. PARATEC uses an all-band conjugate gradient (CG) approach to solve the Kohn-Sham equations of Density Functional Theory (DFT) and obtain the ground-state electron wavefunctions. DFT is the most commonly used technique in materials science to calculate the structural and electronic properties of materials, with a quantum mechanical treatment of the electrons. Codes based on DFT are widely used to study various material properties for nanostructures, complex surfaces, and doped semiconductors, and are one of the largest consumers of supercomputing cycles.

In solving the Kohn-Sham equations using a plane wave basis, part of the calculation is carried out in real space and the remainder in Fourier space using specialized parallel 3D FFTs to transform the wavefunctions. The code spends most of its time in vendor-supplied BLAS3 (∼30%) and 1D FFTs (∼30%) on which the 3D FFTs libraries are built. Because these routines allow high cache reuse and efficient vector utilization, PARATEC generally obtains a high percentage of peak performance across a spectrum of computing platforms. The code exploits fine-grained parallelism by dividing the plane wave (Fourier) components for each

electron among the different processors [15]. PARATEC is written in F90 and MPI, and is designed primarily for massively parallel computing platforms, but can also run on serial machines. The main limitation to scaling PARATEC to large processor counts is the distributed grid transformation during the parallel 3D FFTs that requires global interprocessor communication when mapping the electron wavefunctions from Fourier space (where it is represented by a sphere) to a 3D grid in real space. Thus, architectures with a poor balance between their bisection bandwidth and computational rate will suffer performance degradation at higher concurrencies due to global communication requirements.

Experimental data were gathered for a 432 Silicon-atom bulk system and a standard LDA run of PARATEC with a 25 Ry cut-off using norm-conserving pseudopotentials [31, 32]. Results showed that PARATEC achieves impressive performance on the JES, sustaining 4.7 Gflop/s per processor (58% of peak) on 128 processors. This compares with per processor performance of 0.7 Gflop/s (49%), 1.5 Gflop/s (29%), 3.2 Gflop/s (54%), and 1.9 Gflop/s (15%) on Seaborg, Cheetah, Ram, and Phoenix, respectively (Ram results are for 64 processors). The JES outperformed all other platforms primarily due to a superior architectural balance of bisection bandwidth relative to computation rate — a critical component for achieving high scalability during the global grid transformation of the wavefunctions.

1.3.2 Astrophysics

One of the most challenging problems in astrophysics is the numerical solution of Einstein's equations following from the Theory of General Relativity (GR): a set of coupled nonlinear hyperbolic and elliptic equations containing thousands of terms when fully expanded. The Cactus Computational ToolKit [1, 7] is designed to evolve these equations stably in 3D on supercomputers to simulate astrophysical phenomena with high gravitational fluxes, such as the collision of two black holes and the gravitational waves radiating from that event. While Cactus is a modular framework supporting a wide variety of multi-physics applications [14], our study focussed exclusively on the GR solver, which implements the ADM-BSSN method [1] for stable evolutions of black holes.

The Cactus GR components solve Einstein's equations as an initial value problem that evolves partial differential equations (PDEs) on a regular grid using finite differences. The core of the GR solver uses the ADM formalism, which decomposes the solution into 3D spatial hypersurfaces that represent different slices of space along the time dimension. In this representation, the equations are written as four constraint equations and 12 evolution equations. Additional stability is provided by the BSSN modifications to the standard ADM method [1]. The evolution equations can be solved using a number of different numerical approaches, including staggered leapfrog, McCormack, Lax-Wendroff, and iterative Crank-Nicholson schemes. A *lapse* function describes the time slicing between hypersurfaces for each step in the evolution while a *shift* vector is used to move the coordinate system at each step to avoid being drawn into a singularity. The four constraint equations are used to select different lapse functions and the related shift vectors.

For parallel computation, the global 3D grid is block domain decomposed so

that each processor has its own section. The standard MPI driver for Cactus solves the PDEs on a local region and then updates the values at the ghost zones by exchanging data on the faces of its topological neighbors. On superscalar systems, the computations are blocked in order to improve cache locality. Blocking is accomplished through the use of temporary *slice buffers*, which improve cache reuse while modestly increasing the computational overhead. These blocking optimizations were disabled on vector architectures since they reduced the vector length and inhibited performance.

The production version of the Cactus ADM-BSSN application was run using a grid of size 250×64×64 [32]. The problem was scaled with the number of processors to keep the computational load uniform. Cactus problems are typically scaled in this manner because their science requires the highest-possible resolutions. Results showed that the JES reached an impressive 2.7 Tflop/s (34% of peak) for the largest problem size using 1024 processors. This represents the highest per processor performance (by far) achieved by the production version of Cactus ADM-BSSN on any system to date. Comparing results at 64 processors, Seaborg and Ram are 33x and 6.4x slower, achieving only 6% and 7% of peak, respectively. The relatively low scalar performance on the microprocessor-based systems is partially due to register spilling, which is caused by the large number of variables in the main loop of the BSSN calculation. Phoenix is about 3.8 times slower than the JES, sustaining 0.7 Gflop/s per processor (6% of peak). The architectural imbalance between vector and scalar performance was particularly acute on the X1, which suffered a much greater impact from unvectorized code in Cactus than the JES.

1.3.3 Cosmology

The Microwave Anisotropy Dataset Computational Analysis Package (MADCAP) [4] implements the current optimal general algorithm for extracting the most useful cosmological information from total-power observations of the Cosmic Microwave Background (CMB). The CMB is a snapshot of the Universe when it first became electrically neutral some 400,000 years after the Big Bang. The tiny anisotropies in the temperature and polarization of the CMB radiation are sensitive probes of early Universe cosmology, and measuring their detailed statistical properties has been a high priority in the field for over 30 years. MADCAP was designed to calculate the maximum likelihood two-point angular correlation function (or power spectrum) of the CMB given a noisy, pixelized sky map and its associated pixel-pixel noise correlation matrix.

MADCAP recasts the extraction of a CMB power spectrum from a sky map into a problem in dense linear algebra, and exploits the ScaLAPACK library for its efficient parallel solution. Operations include explicit matrix inversion, as well as matrix-vector and matrix-matrix multiplication. Since most of the MADCAP routines utilize ScaLAPACK, code migration is relatively simple. Performance for both scalar and vector systems depends heavily on an efficient implementation of the vendor-supplied linear algebra libraries. However, explicit vectorization was required for the hand-coded calculation of the value of Legendre polynomials (up to some preset degree) for the angular separation between all pixel pairs.

To maximize its ability to handle large datasets, MADCAP generally works with many more matrices than can fit in main memory. The out-of-core disk-based storage for the other matrices in the calculation is the only practical choice, but comes at the cost of heavy I/O. All matrices are block-cyclic distributed across the processors; when a matrix is stored to disk due to memory limitations, MADCAP generates one file per processor over which the matrix is distributed. This results in matrix I/O operations that are independent, however the simultaneity of multiprocessor disk accesses can create contention within the I/O infrastructure, thus degrading overall performance.

Experimental data reported here were collected by MAXIMA [19] (Millimeter Anisotropy eXperiment Imaging Array): a balloon-borne millimeter-wave telescope designed to measure the angular power spectrum of fluctuations in the CMB over a wide range of angular scales. Simulation results using 64 processors on a dataset of 14996 pixels and 1200 multipoles showed that Phoenix achieved the best runtime at 2.0 Gflop/s per processor — 1.1x, 2.8x, and 4.4x faster than the JES, Cheetah, and Seaborg [8, 31]. However, Seaborg sustained the highest percentage of peak (36%) followed by the JES (23%), Phoenix (16%), and Cheetah (15%). Note that all evaluated architectural platforms sustained a relatively low fraction of peak, considering MADCAP's extensive use of computationally intensive dense linear algebra functions. In-depth analysis using a lightweight version of MADCAP highlights the complex interplay between the architectural paradigms, interconnect technology, and vendor-supplied numerical libraries, while isolating I/O filesystems as the key bottleneck across all platforms [5].

1.3.4 Fluid Dynamics

In the area of computational fluid dynamics (CFD), we selected the NASA Navier-Stokes production application called OVERFLOW-D [45]. The code uses the overset grid methodology to perform high-fidelity viscous simulations around realistic aerospace configurations. It is popular within the aerodynamics community due to its ability to handle complex designs with multiple geometric components. OVERFLOW-D, an extension of OVERFLOW [6], is explicitly designed to simplify the modeling of problems when components are in relative motion. The main computational logic at the top level of the sequential code consists of a time-loop and a nested grid-loop. Within the grid-loop, solutions to the flow equations are obtained on the individual grids with imposed boundary conditions. Overlapping boundary points or inter-grid data are updated from the previous time step using a Chimera interpolation procedure. Upon completion of the grid-loop, the solution is automatically advanced to the next time step by the time-loop. The code uses finite differences in space, with a variety of implicit/explicit time stepping.

The MPI version of OVERFLOW-D (in F90) takes advantage of the overset grid system, which offers a natural coarse-grain parallelism. A grouping algorithm clusters individual grids into sets, each of which is then assigned to an MPI process. The grid-loop in the parallel implementation is subdivided into two procedures: a group-loop over sets, and a grid-loop over the grids within each set. Since each MPI process is assigned to only one set, the group-loop is executed in parallel, with

each set performing its own sequential grid-loop. The inter-grid boundary updates within each set are performed as in the serial case. Inter-set boundary exchanges are achieved via MPI asynchronous communication calls. The same basic program structure is used on all target architectures except for a few minor changes in some subroutines to meet specific compiler requirements. Details about performance enhancement strategies for OVERFLOW-D can be found in [10].

Our experiments involve a simulation of vortex dynamics in the complex wake flow region around hovering rotors. The grid system consisted of 41 blocks and almost 8 million grid points [31, 33]. Results on an SX-6 platform (but not the JES) easily outperformed the superscalar machines as well as Phoenix; in fact, the runtime for eight SX-6 processors was less than 60% of the 32-processor Cheetah runtime. However, due to small vector lengths and limited vector operation ratio, the code achieved only about 1.1 Gflop/s per processor on the SX-6. Nevertheless, the SX-6 consistently achieved a high percentage of peak (14%), followed by Seaborg (8%) and Cheetah (7%), while Phoenix showed the lowest sustained performance (3.5%). Scalability on Seaborg exceeded all others, with computational efficiency decreasing for a larger number of MPI tasks primarily due to load imbalance. A hybrid MPI+OpenMP implementation [9] showed similar performance as a pure MPI approach on all systems except for Seaborg where the multilevel results were significantly better. Note that adding more OpenMP threads beyond an optimal number, depending on the number of MPI processes, does not improve performance. The primary advantage of the hybrid paradigm for overset grid problems is that it extends their applicability to large SMP clusters. A detailed performance characterization of the Columbia supercluster using OVERFLOW-D and other NASA applications is reported in [3].

1.3.5 Magnetic Fusion

The Gyrokinetic Toroidal Code (GTC) is a 3D particle-in-cell (PIC) application developed at the Princeton Plasma Physics Laboratory to study turbulent transport in magnetic confinement fusion [23, 24]. Turbulence is believed to be the main mechanism by which energy and particles are transported away from the hot plasma core in fusion experiments with magnetic toroidal devices. GTC solves the non-linear gyrophase-averaged Vlasov-Poisson equations [22] for a system of charged particles in a self-consistent, self-generated electrostatic field. The geometry is that of a torus with an externally imposed equilibrium magnetic field, characteristic of toroidal fusion devices. By using the PIC method, the non-linear PDE describing the motion of the particles in the system becomes a simple set of ordinary differential equations (ODEs) that can be easily solved in the Lagrangian coordinates. The self-consistent electrostatic field driving this motion could conceptually be calculated directly from the distance between each pair of particles using an $O(N^2)$ calculation, but the PIC approach reduces it to $O(N)$ by using a grid where each particle deposits its charge to a limited number of neighboring points according to its range of influence. The electrostatic potential is then solved everywhere on the grid using the Poisson equation, and forces are gathered back to each particle. The most computationally intensive parts of GTC are the charge deposition and gather-push

steps that involve large loops over the particles, which can reach several million per domain partition.

Although the PIC approach drastically reduces the computational requirements, the grid-based charge deposition phase is a source of performance degradation for both superscalar and vector architectures. Randomly localized particles deposit their charge on the grid, thereby causing poor cache reuse on superscalar machines. The effect of this deposition step is more pronounced on vector systems since two or more particle may contribute to the charge at the same grid point, creating a potential memory-dependency conflict. Several methods have been developed to address this issue; GTC uses the work-vector algorithm [29], where a temporary copy of the grid array is given an extra dimension corresponding to the vector length. Each vector operation acts on a given data set in the register, then writes to a different memory address, avoiding memory dependencies entirely. After the main loop, the results accumulated in the work-vector array are gathered to the final grid array. The only drawback is the increased memory footprint, which can be two to eight times higher than the nonvectorized code version.

Experiments were performed using a high resolution domain of 100 particles per cell (2 million grid points, 200 million particles), allowing for significant improvements in the overall statistics of the simulation [32, 33]. The JES achieved 1.6 Gflop/s per processor or 20% of peak on 64 processors — the highest GTC performance on any tested architecture. Phoenix showed similar performance in absolute terms (1.4 Gflop/s per processor), but a lower fraction of peak at only 11%. Comparing performance with the superscalar architectures, the JES is about 12x, 5.4x, and 5x faster than Seaborg (9% of peak), Cheetah (5%), and Ram (5%), respectively. The vector systems therefore have the potential for significantly higher resolution calculations that would otherwise be prohibitively expensive in terms of time-to-solution on conventional microprocessors.

1.3.6 Geophysics

The goal of the GeoFEM framework [16] is to model and simulate solid Earth phenomena, such as the long term prediction of the plate near the Japanese islands and core/mantle dynamics over the entire globe. GeoFEM is being specifically developed to achieve high performance on the JES. The finite element method (FEM) used here is an approximation based on contiguous interactions, which can be reduced to solving large-systems of linear equations defined over the unstructured meshes that model the underlying physical objects. Since direct methods are generally disadvantageous for solving these types of systems due to the large volume of computation involved, an iterative method such as CG is employed to address the overall degrees of freedom (DOF). To obtain a stable solution using this numerical approach requires intelligent domain partitioning as well as preconditioning for the iterative solver. The localized ILU(0) used in GeoFEM is a pseudo-preconditioner, requiring only local operations within each given domain partition, thus making it well suited for parallel systems since no interprocessor communication is required.

In order to achieve both high parallel efficiency and vector performance for the GeoFEM iterative solvers, an intelligent reordering scheme must be utilized

to mitigate the irregular nature of the underlying unstructured mesh. A Reverse Cuthill-Mckee (RCM) ordering is utilized to improve data locality and reduce global communication. RCM is a level set reordering method used to reduce the profile of the underlying matrix, which improves the convergence of the ILU preconditioner but may result in a load imbalanced computation. To create a more balanced workload, the Cyclic Multicolor (CM) technique, used to graph color adjacent nodes, is then combined with RCM to both balance the workload and provide fast convergence for the Krylov iterative solvers. For complicated geometries, traditional multicoloring is usually used instead of CM-RCM [27].

However, the well-known compressed row storage (CRS) representation of the underlying matrix results in short vector lengths, thus inhibiting vector performance. To address this deficiency, the descending-order jagged diagonal storage (DJDS) scheme is used to permute the matrix rows so that reasonably long innermost loop lengths are achieved. To effectively balance the computation load of each processor within an SMP using this approach, the DJDS array is reordered again in a cyclic fashion (PDJDS). On superscalar systems, the parallel descending-order compressed row storage (PDCRS) ordering is used. This approach is almost identical to PDJDS except that the matrices are stored in CRS format after reordering the rows by decreasing numbers of non-zeros. This strategy results in shorter vector lengths, and is generally better suited for commodity superscalar systems with limited memory bandwidth designs.

Extensive experiments were conducted for a 3D linear elastic problem with more than 2.2×10^9 DOF, and solved by a 3×3 block incomplete Cholesky (IC) preconditioned CG with additive Schwarz domain decomposition [28]. Results showed that the PDJDS/CM-RCM reordering achieved 2.7 GFlop/s per processor (33.7% of peak) on the JES using 1408 processors, while only 0.5% of peak was attained using the naive CRS approach without reordering. On Seaborg, PDCRS/CM-RCM ordering attained the highest performance; however, results were significantly poorer than the JES: only 0.11 GFlop/s per processor (7.2% of peak) was sustained on 1024 processors.

1.4 Algorithmic and Architectural Benchmarks

The purpose of any HPC system is to run full-scale applications, and performance on such applications is the final arbiter of the utility of the system. However, the complexity of using scientific applications to identify the causes of performance bottlenecks on modern architectures has raised the importance of developing better benchmarking methods to improve program characterization and performance prediction, while identifying hardware and application features that work well or poorly together. In this section, we describe both algorithmic and architectural benchmarks that strive to satisfy these objectives.

1.4.1 Scalable Compact Application Benchmarks

Scalable Compact Application (SCA) benchmarks are derivatives that capture major characteristics of the full applications, but avoid many of their idiosyncrasies by

leaving out unnecessary details. They work with synthetic data sets that can be generated with prescribed statistical properties, so that the application signature does not change qualitatively with problem size. Moreover, any fixed-size application will ultimately lose its value as systems become more powerful. Input data for SCA benchmarks can be generated on the fly and do not have to be maintained and distributed. Considering the capability of current and future HPC systems, and taking into account that useful applications typically consume (and produce) data commensurate with the amount of computation that they do, being able to generate synthetic data greatly saves on the volume that has to be maintained, stored, and distributed. Effective SCAs have a built-in verification test that has been validated on multiple machines and configurations, so that the user knows instantly whether a run was successful or not. They are portable and designed to be built and run easily on any modern platform with standard tools (e.g., MPI or OpenMP), which reduces the cost and effort on the part of vendors to produce performance numbers. SCAs also represent a single path through an application, eliminating any special-purpose code not executed. This makes them easier to understand and characterize, unlike full applications whose performance signature often depends vitally on input parameters, rendering their actual name almost meaningless when interpreting measurement results. Finally, SCAs are non-proprietary and open source, so that thorough analysis by research groups can be conducted. This makes them valuable for users outside the organization that provided the initial full-scale application from which the SCA was derived.

Microbenchmarks are codes whose performance is dominated by a single architectural feature, such as network bandwidth, memory latency, I/O speed, clock rate, etc. They are very useful to determine a principal machine characteristic, but application performance is typically influenced by multiple non-linearly interacting factors, making it difficult to distinguish between the different effects. For example, it is not useful to transfer data at a high rate if the data does not embody the requested information, which takes computational effort to generate or acquire. SCAs typically feature just a few performance influence factors, making them more realistic than microbenchmarks, but much easier to implement and analyze than full-fledged applications.

NAS Parallel Benchmarks

The NAS Parallel Benchmarks (NPB) [2, 30] were designed to provide a level playing field for HPC machines of various architectural specifications. This is reflected in the fact that they were first released as paper-and-pencil specifications. As standards for programming parallel computers matured, portable source code implementations were provided, first using MPI, and later in Java, OpenMP, and HPF. The NPB application benchmarks Lower-Upper (LU) symmetric Gauss Siedel, Scalar Pentadiagonal (SP), and Block Tridiagonal (BT), and to some extent fast Fourier Transform (FT) and Multi-Grid (MG), are SCAs that have been used extensively by vendors, compiler writers, procurement teams, and tool developers. They were especially useful at a time when unrealistic claims were being made about the computational power of emerging parallel computers and new microprocessor designs

with ever-increasing clock speeds.

As parallel computing entered the mainstream, and more and more legacy applications were converted to run on parallel machines, new performance bottlenecks showed up that were not properly tested by the NPB. Most importantly, the NPB did not test the I/O subsystem; the effects of irregular and continually changing memory accesses; and the capability to solve hierarchical problems with multiple levels of exploitable parallelism. Recent additions to the NPB have addressed all these issues, and also led to larger problem classes that keep track of the growing memory size and computing power of high-end machines.

Parallel high-performance I/O is now being tested by a variation of BT, which, like SP and LU, generates a time-dependent series of solutions to a CFD problem. Real applications usually checkpoint the solution every few time steps to recover from possible system errors or to save data for offline visualization; thus, BT was modified to write its solution to files at relatively high frequency. The reference implementation provided by NASA ARC uses either native Fortran I/O, or parallel I/O based on MPI, to write the highly fragmented data from multiple processors to the single checkpoint file.

A completely new benchmark, called Unstructured Adaptive (UA) mesh refinement [13], was developed to measure the effects of temporally and spatially irregular memory accesses. This code solves a stylized convective/diffusive heat transfer problem with a heat source whose location changes in time. The original NPB SCAs feature simple memory access patterns using small, fixed numbers of strides. In contrast, UA uses an unstructured Cartesian mesh whose topology changes periodically to track the moving source, inserting and removing refined mesh cells where needed, and therefore causing irregular and unpredictable memory access patterns. It utilizes a nonconforming spectral finite element method of moderately high order (five), resulting in a large volume of computational work per element relative to the amount of data shared between elements. Consequently, an efficient parallel implementation of UA requires good balancing of the computational load, but does not depend too heavily on the minimization of interprocessor communication. Contrasting the performance of OpenMP versions of MG and UA, the first featuring a small number of fixed—mostly unit—memory access strides, and the latter exhibiting many irregularly strided accesses, results show that UA experiences significantly larger increases in parallel efficiency as the number of threads grows [12].

The NPB Multi-Zone (NPB-MZ) version [44] was created to represent overset grid applications (e.g., OVERFLOW) with multilevel parallelism. It consists of three families of problems, each corresponding to one of the original NPB problems LU, SP, and BT. NPB-MZ solves an SCA on a collection of loosely coupled discretization meshes, called *zones*. The zones are arranged as a 2D, partially overlapping, tiling of 3D space. The solution within each zone is updated separately from all the others, thus offering easily exploited coarse-grain parallelism. However, at the end of each time step, neighboring zones exchange boundary values in the overlap regions. Fine-grain parallelism can be exploited at the loop nest level within individual zones.

The three MZ families are distinguished primarily by the way they tile space.

LU-MZ uses a 4×4 grid of equal-sized zones for all problem sizes, thus limiting the amount of coarse-grain parallelism to 16. SP-MZ also use a 2D grid of equal-sized zones, but the number of zones goes up with problem size, thus providing increasing amounts of coarse-grain parallelism. BT-MZ has the same number and layout of zones as SP-MZ, but they differ in size, with a fixed ratio of largest over smallest zone size of 20 (for all problem sizes). Coarse-level parallelism is now not so easily exploited while simultaneously balancing the load. Current parallel implementations of NPB-MZ use OpenMP for fine-grain parallelism and message passing (MPI or MLP [42]) for the coarse grain [21]. Experiments on an IBM Power3, SGI Origin 3800, and HP/Compaq SC45 show that the best performance on a fixed number of processors for NPB-MZ is always obtained by minimizing the number of threads per process. That is, if the load can be balanced with a single thread per process, it invariably yields the best performance. For SP-MZ this is usually not a problem, but is not always feasible for BT-MZ (with the same number of zones), due to the disparity in the sizes of the zones. A three-level hybrid implementation of NPB-MZ is currently being developed and will be tested on the Columbia system at NASA ARC to characterize and understand OVERFLOW performance on such platforms.

1.4.2 Architectural Probes

Synthetic benchmarks like the NPB [2, 30] and Standard Performance Evaluation Corporation (SPEC) [39] emphasize the proper representation of real applications, but are usually too large to run on simulated architectures and too complex to identify specific architectural bottlenecks. At the other extreme, microbenchmarks such as Stream [25], Livermore loops [26], Hint [18], and Linpack [11] are easy to optimize but measure the performance of only a specific architectural feature. They present a narrow view of a broad, multidimensional parameter space of machine characteristics.

We therefore differentiate an architectural probe from a microbenchmark on the basis that the latter typically returns a single-valued result in order to rank processor performance consecutively — a few reference points in a multidimensional space. A probe, by contrast, is used to explore a continuous, multidimensional parameter space. The probe's parameterization enables the researcher to uncover peaks and valleys in a continuum of performance characteristics, and explore the ambiguities of computer architectural comparisons that cannot be captured by a single-valued ranking.

Sqmat

Sqmat [17] is an architectural probe that complements the capabilities of traditional kernel benchmarks as tools that enhance the understanding of performance behavior of scientific codes. It maintains the simplicity of a microbenchmark while offering four distinct parameters to capture different types of application workloads: working-set size, computational intensity, indirection, and irregularity.

The Sqmat algorithm squares a set of matrices a given number of times. First,

the values of one matrix are loaded from memory into registers; if the matrix size exceeds the register capacity, register spilling will occur — thus controlling the working set size. Accessing the matrices from memory can be done either directly or indirectly. In the direct case, matrices are stored contiguously in memory using row-major ordering. For the indirect case, a parameter controls the degree of irregularity: a fixed number of entries are stored contiguously, followed by a jump to a random position in memory. This parameter therefore captures both indirection and the degree of data access irregularity. Finally, each matrix is squared a certain number of times thereby controlling the computational intensity — the ratio between memory transfers and arithmetic operations.

A number of important analyses can be conducted by varying these simple parameters. For example, a component of the architectural balance can be examined by measuring how much computation is required to hide the highest degree of irregularity (accessing each matrix entry at a random position) such that the achieved performance is only 50% lower than unit-stride data accesses. Results on the Itanium2 and Power4 microprocessors reveal significantly different architectural designs. The Itanium2 can hide the irregularity with a computational intensity of 9.3, showing that it is somewhat tolerant of random accesses. The Power4, however, requires a computational intensity of 74.7 for the same experiments, demonstrating that it is poorly suited for sparse matrix computations. These types of analyses help isolate architecture performance bottlenecks, helping both applications programmers and system designers.

Apex-MAP

The Application Performance Characterization project (Apex) [40] is another effort to develop tunable synthetic benchmarks. The preliminary focus of this effort is to define performance characterizations of application data access patterns and to create a corresponding memory access probe benchmark, called Apex-MAP. Apex-MAP differs from Sqmat in that its execution profile can be tuned by a set of parameters to match the signature of a chosen scientific application, allowing it to be used as a proxy for the performance behavior of the underlying codes.

Apex-MAP assumes that the data access pattern for 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 and strided accesses. The data-set size is 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 through the use of a cumulative temporal distribution function.

Preliminary testing examined the validity and accuracy of the Apex-MAP approach on leading microarchitectures including the Power3, Power4, and X1, using five scientific kernels (radix sorting, FFT, matrix multiplication, n-body simulation,

and conjugate gradient). Results showed that application performance can be captured to within 25% across the suite of codes for a variety of memory sizes, using a few simple parameters with up to two simulated memory streams. This work has recently been extended to include interprocessor communication to capture the behavior of parallel systems [41].

1.5 Performance Modeling

The goal of performance modeling is to gain understanding of a computer system's performance on various applications, by means of measurement and analysis, and then to encapsulate these characteristics in a compact formula. The resulting model can be used to obtain better insight into the performance phenomena involved and to project performance to other system/application combinations. This section focuses on the modeling of large-scale scientific computations within the Performance Evaluation Research Center (PERC) [36], a research collaboration funded through the DOE's Scientific Discovery through Advanced Computation (SciDAC) program [37]. A number of important performance modeling activities are also being conducted by other groups, particularly the efforts at Los Alamos National Laboratory (LANL) [20].

Performance models can be used to improve architecture design, inform procurement activities, and guide application tuning. Unfortunately, the process of producing performance models has historically been rather expensive, requiring large amounts of computer time and highly expert human effort. This has severely limited the number of HPC applications that can be modeled and studied. It has been observed that, due to the difficulty of developing performance models for new applications as well as the increasing complexity of new systems, our supercomputers have become better at predicting and explaining natural phenomena (such as weather) than at predicting and explaining the performance of themselves.

1.5.1 Applications of Performance Modeling

The most common application for a performance model is to enable a scientist to estimate the runtime of a job when the input sets or architectural parameters are varied. Additionally, one can estimate the largest system size that can be used to run a given problem before the parallel efficiency drops to unacceptable levels. Performance models are also employed by computer vendors in the design of future systems. Typically, engineers construct a performance model for one or two key applications, and then compare future technology options based on the computed projections. Once better performance modeling techniques are developed, it may be possible to target many more applications and technology options in the design process. As an example of such "what-if" investigations, application parameters can be used to predict how performance rates would change with a larger or more highly associative cache. In a similar way, the performance impact of various network designs can also be explored.

System and application tuning can also greatly benefit from the use of performance models. For example, if a memory model is combined with application

parameters, one can predict how cache hit-rates would change if a different cache-blocking factor were used in the application. Once the optimal cache blocking has been identified, the code can be permanently modified. Simple performance models can even be directly incorporated into an application code, permitting on-the-fly selection of different program options. Finally, the simplification of system procurement procedures may be the most compelling application of performance modeling. Once a reasonably easy-to-use performance modeling facility is available, it may be possible to greatly reduce, if not eliminate, the benchmark tests that are specified in a procurement, replacing them with a measurement of certain performance model parameters for the target systems and applications. These parameters can then be used by the computer center staff to project performance rates for numerous system options, thus saving considerable resources during the system acquisition process.

1.5.2 PERC Methodology

The PERC framework [36] is based upon *application signatures*, *machine profiles*, and *convolutions*. An application signature is a detailed but compact representation of the fundamental operations performed by an application, independent of the target system. A machine profile is a representation of the capability of a system to carry out fundamental operations, independent of the particular application. A convolution is a means to rapidly combine application signatures with machine profiles in order to predict performance. Overall, the PERC methodology consists of (i) accurately summarizing the requirements of applications (in ways that are not too expensive in terms of time/space requirements); (ii) automatically obtaining the application signature; (iii) generalizing the signatures to represent how the application would stress arbitrary (including future) machines; and (iv) extrapolating the signatures to larger problem sizes than what can be actually run at the present time.

PERC has developed a general approach to analyzing applications in order to obtain their signature, which has resulted in considerable space reduction and a measure of machine independence. First, a given application is statically analyzed, then instrumented and traced on an existing set of architectural platforms. Next, the operations performed by the application are summarized on-the-fly during execution. These operations are then indexed to the source code structures that generated them. Finally, a merge is performed on the summaries of each machine [38]. From this data, one can obtain information on memory access patterns (viz., the stride and range of memory accesses generated by individual memory operations) and communications patterns (viz., size and type of communication performed).

The second component of this performance modeling approach is to represent the resource capabilities of current and proposed machines, with emphasis on memory and communications capabilities, in an application-independent form suitable for parameterized modeling. In particular, machine profiles are gathered, which are high-level representations of the rates at which a system can carry out basic operations such as memory loads/stores and message passing. This includes the capabilities of memory units at each level of the memory hierarchy and the ability of machines to overlap memory accesses with other kinds of operations (e.g., floating-

point or communication). The machine profiles are then extended to account for reduction in capability due to sharing (e.g., to express how much the memory subsystem's or communication fabric's capability is diminished by sharing these with competing processors). Finally, the behavior of larger system can be extrapolated from validated machine profiles of similar but smaller platforms.

To enable time tractable statistical simulation, the PERC framework utilizes the convolution method, allowing quick mappings of application signatures to machine profiles. This approach closely approximates cycle-accurate predictions in a much shorter time frame by accounting for fewer details. The convolution allows relatively rapid development of performance models (full application models currently take one or two months) and results in performance predictions which can be quickly evaluated once the models are constructed (few minutes per prediction).

1.5.3 Performance Sensitivity Studies

Reporting the accuracy of performance models in terms of model-predicted time versus observed time is mostly just a validation step for obtaining confidence in the model. A more interesting and useful exercise is to explain and quantify performance differences, while allowing architectural parameter studies. Given a model that can predict application performance based on properties of the code and machine, precise modeling experiments can be performed. For example, an in-depth study was performed [38] using the Parallel Ocean Program (POP) [34], a well-known climate application, on the Blue Horizon system. POP has been ported to a wide variety of computers for eddy-resolving simulations of the world oceans and for climate simulations as the ocean component of coupled climate models.

First, the effect of reducing the bandwidth from 350MB/s to 269MB/s was examined (equivalent to switching the network from Colony to a single-rail of a Quadrics switch), but resulted in no observable performance difference. The next experiment reduced the latency from 20ms (Colony) to 5ms (Quadrics), and demonstrated a performance improvement of 1.3x — evidence that the barotropic calculations in POP (for the tested problem size) are latency sensitive. Finally, the system was modeled using the default Colony switch, but with an improved processor design based on the Alpha ES45 (1GHz versus 375MHz) and a more powerful memory subsystem capable of loading stride-1 data from L2 cache at twice the rate of the Power3. Results showed a performance improvement of 1.4x, due mainly to the faster memory subsystem. The principal observation from the above exercise is that the PERC model can quantify the performance impact of each machine hardware component. Similar experiments were conducted for varying sizes of POP problems, with results showing that POP simulations at larger processor counts become more network latency sensitive while remaining mostly bandwidth insensitive.

These types of experiments demonstrate that performance models enable "what-if" analyses of the implications of improving the target machine in various dimensions. Such studies are obviously useful to system designers, helping them optimize system architectures for the highest sustained performance on a target set of applications. These methods are also potentially useful in helping computing centers select the best system in an acquisition. Additionally, this approach can be used by appli-

cation scientists to improve the performance of their codes by better understanding which tuning measures yield the most improvement in sustained performance.

With further improvements in this methodology, we can envision a future where these techniques are embedded in application codes, or even in systems software, thus enabling self-tuning applications for user codes. For example, we can conceive of an application that performs the first of many iterations using numerous cache blocking parameters, a separate combination on each processor, and then uses a simple performance model to select the most favorable value. This combination would then be used for all remaining iterations, thus enabling automatic and portable system optimization.

1.6 Summary

To substantially increase the raw computational power of ultra-scale high performance computing (HPC) systems and then reap its benefits, significant progress is necessary in hardware architecture, software infrastructure, and application development. This chapter provided a broad performance assessment of leading HPC systems using a diverse set of scientific applications, including scalable compact synthetic benchmarks and architectural probes. Performance models and program characterizations of large-scale scientific computations were also discussed.

Acknowledgements

The authors sincerely thank their collaborators Julian Borrill, Andrew Canning, Johnathan Carter, M. Jahed Djomehri, Stephane Ethier, John Shalf, and David Skinner for their many contributions to the work presented here in this report. They also gratefully acknowledge Kengo Nakajima for reviewing the material on geophysics. L. Oliker and D. Bailey are supported by OASCR in the U.S. DOE Office of Science under contract DE-AC03-76SF00098.

Bibliography

[1] M. Alcubierre, G. Allen, B. Bruegmann, E. Seidel, and W.-M. Suen. Towards an understanding of the stability properties of the 3+1 evolution equations in General Relativity. *Physics Review D*, 62:124011, Dec. 2000.

[2] D.H. Bailey et al. The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, 1994.

[3] R. Biswas, M.J. Djomehri, R. Hood, H. Jin, C. Kiris, and S. Saini. An application-based performance characterization of the Columbia supercluster. In *Proc. SC2005*, Seattle, WA, Nov. 2005.

[4] J. Borrill. MADCAP: The Microwave Anisotropy Dataset Computational Analysis Package. In *Proc. 5th European SGI/Cray MPP Workshop*, Bologna, Italy, Sep. 1999. Article astro-ph/9911389.

[5] J. Borrill, J. Carter, L. Oliker, D. Skinner, and R. Biswas. Integrated performance monitoring of a cosmology application on leading HEC platforms. In *Proc. 34th International Conference on Parallel Processing*, pages 119–128, Oslo, Norway, Jun. 2005.

[6] P.G. Buning, D.C. Jespersen, T.H. Pulliam, W.M. Chan, J.P. Slotnick, S.E. Krist, and K.J. Renze. OVERFLOW User's Manual version 1.8g. Technical report, NASA Langley Research Center, 1999.

[7] Cactus Code Server. `http://www.cactuscode.org`.

[8] J. Carter, J. Borrill, and L. Oliker. Performance characteristics of a cosmology package on leading HPC architectures. In *Proc. 11th International Conference on High Performance Computing*, Bangalore, India, Dec. 2004.

[9] M.J. Djomehri and R. Biswas. Performance analysis of a hybrid overset multi-block application on multiple architectures. In *Proc. 10th International Conference on High Performance Computing*, pages 383–392, Hyderabad, India, Dec. 2003.

[10] M.J. Djomehri and R. Biswas. Performance enhancement strategies for multi-block overset grid CFD applications. *Parallel Computing*, 29(11-12):1791–1810, Nov.-Dec. 2003.

[11] J. Dongarra. Performance of various computers using standard linear equations software. Technical Report CS-89-85, University of Tennessee, 1989.

[12] H. Feng, R.F. Van der Wijngaart, and R. Biswas. Unstructured adaptive meshes: Bad for your memory? *Applied Numerical Mathematics*, 52(2-3):153–173, Feb. 2005.

[13] H. Feng, R.F. Van der Wijngaart, R. Biswas, and C. Mavriplis. Unstructured Adaptive (UA) NAS Parallel Benchmark, version 1.0. Technical Report NAS-04-006, NASA Ames Research Center, Jul. 2004.

[14] J.A. Font, M. Miller, W.-M. Suen, and M. Tobias. Three dimensional numerical General Relativistic hydrodynamics: Formulations, methods, and code tests. *Physics Review D*, 61:044011, Feb. 2000.

[15] G. Galli and A. Pasquarello. First-principles molecular dynamics. In *Computer Simulation in Chemical Physics*, pages 261–313. Kluwer, 1993.

[16] GeoFEM Project. `http://geofem.tokyo.rist.or.jp`.

[17] G. Griem, L. Oliker, J. Shalf, and K. Yelick. Identifying performance bottlenecks on modern microarchitectures using an adaptable probe. In *Proc. 3rd International Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems*, Santa Fe, NM, Apr. 2004.

[18] J.L. Gustafson and Q.O. Snell. HINT: A new way to measure computer performance. In *Proc. 28th Hawaii International Conference on System Sciences*, pages 392–401, Wailela, HI, Jan. 1995.

[19] S. Hanany et al. MAXIMA-1: A measurement of the Cosmic Microwave Background anisotropy on angular scales of $10'$–5°. *The Astrophysical Journal Letters*, 545(1):L5–L9, Dec. 2000.

[20] A. Hoisie, O. Lubeck, and H. Wasserman. Performance and scalability analysis of teraflop-scale parallel architectures using multidimensional wavefront applications. *International Journal of High Performance Computing Applications*, 14(4):330–346, Dec. 2000.

[21] H. Jin and R.F. Van der Wijngaart. Performance characteristics of the multi-zone NAS Parallel Benchmarks. In *Proc. 18th International Parallel and Distributed Processing Symposium*, Santa Fe, NM, Apr. 2004.

[22] W.W. Lee. Gyrokinetic particle simulation model. *Journal of Computational Physics*, 72(1):243–269, Sep. 1987.

[23] Z. Lin, S. Ethier, T.S. Hahm, and W.M. Tang. Size scaling of turbulent transport in magnetically confined plasmas. *Physics Review Letters*, 88(19):195004, Apr. 2002.

[24] Z. Lin, T.S. Hahm, W.W. Lee, W.M. Tang, and R.B. White. Turbulent transport reduction by zonal flows: Massively parallel simulations. *Science*, 281(5384):1835–1837, Sep. 1998.

[25] J. McCalpin. Memory bandwidth and machine balance in high performance computers. *IEEE TCCA Newsletter*, Dec. 1995.

[26] F.H. McMahon. The Livermore Fortran Kernels test of the numerical performance range. In J.L. Martin, editor, *Performance Evaluation of Supercomputers*, pages 143–186. North Holland, 1988.

[27] K. Nakajima. Parallel iterative solvers of GeoFEM with selective blocking preconditioning for nonlinear contact problems on the Earth Simulator. In *Proc. SC2003*, Phoenix, AZ, Nov. 2003.

[28] K. Nakajima. Three-level hybrid vs. flat MPI on the Earth Simulator: Parallel iterative solvers for finite-element method. In *Proc. 6th IMACS International Symposium on Iterative Methods in Scientific Computing*, Denver, CO, Mar. 2003.

[29] A. Nishiguchi, S. Orii, and T. Yabe. Vector calculation of particle code. *Journal of Computational Physics*, 61(3):519–522, Dec. 1985.

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

[31] L. Oliker, R. Biswas, J. Borrill, A. Canning, J. Carter, M.J. Djomehri, H. Shan, and D. Skinner. A performance evaluation of the Cray X1 for scientific applications. In *Proc. 6th International Meeting on High Performance Computing for Computational Science*, pages 51–65, Valencia, Spain, Jun. 2004.

[32] L. Oliker, A. Canning, J. Carter, J. Shalf, and S. Ethier. Scientific computations on modern parallel vector systems. In *Proc. SC2004*, Pittsburgh, PA, Nov. 2004.

[33] L. Oliker, A. Canning, J. Carter, J. Shalf, D. Skinner, S. Ethier, R. Biswas, M.J. Djomehri, and R.F. Van der Wijngaart. Performance evaluation of the SX-6 vector architecture for scientific computations. *Concurrency and Computation: Practice and Experience*, 17(1):69–93, Jan. 2005.

[34] The Parallel Ocean Program. `http://climate.lanl.gov/Models/POP`.

[35] PARAllel Total Energy Code. `http://www.nersc.gov/projects/paratec`.

[36] The Performance Evaluation Research Center. `http://www.perc.nersc.gov`.

[37] Scientific Discovery through Advanced Computing. `http://www.science.doe.gov/scidac`.

[38] A. Snavely, X. Gao, C. Lee, N. Wolter, J. Labarta, J. Gimenez, and P. Jones. Performance modeling of HPC applications. In *Proc. Parallel Computing Conference*, Dresden, Germany, Sep. 2003.

[39] Standard Performance Evaluation Corporation. `http://www.spec.org`.

[40] E. Strohmaier and H. Shan. Architecture independent performance characterization and benchmarking for scientific applications. In *Proc. International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Volendam, Netherlands, Oct. 2004.

[41] E. Strohmaier and H. Shan. Apex-MAP: A global data access benchmark to analyze HPC systems and parallel programming paradigms. In *Proc. SC2005*, Seattle, WA, Nov. 2005.

[42] J.R. Taft. Achieving 60 GFLOP/s on the production CFD code OVERFLOW-MLP. *Parallel Computing*, 27(4):521–536, Mar. 2001.

[43] Top500 Supercomputer Sites. `http://www.top500.org`.

[44] R.F. Van der Wijngaart and H. Jin. NAS Parallel Benchmarks, Multi-Zone Versions. Technical Report NAS-03-010, NASA Ames Research Center, Jul. 2003.

[45] A.M. Wissink and R. Meakin. Computational fluid dynamics with adaptive overset grids on parallel and distributed computer platforms. In *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 1628–1634, Las Vegas, NV, Jul. 1998.

Index