September 2005
NAS Technical Report NAS-05-012

Integrated Performance Monitoring of a Cosmology Application on Leading HEC Platforms

J. Borrill, J. Carter, L. Oliker, D. Skinner
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA 94720
{jdborrill,jtcarter,loliker,dskinner}@lbl.gov

R. Biswas
NAS Division
NASA Ames Research Center
Moffett Field, CA 94035
rbiswas@mail.arc.nasa.gov

Abstract

The Cosmic Microwave Background (CMB) is an exquisitely sensitive probe of the fundamental parameters of cosmology. Extracting this information is computationally intensive, requiring massively parallel computing and sophisticated numerical algorithms. In this work we present MADbench, a lightweight version of the MADCAP CMB power spectrum estimation code that retains the operational complexity and integrated system requirements. In addition, to quantify communication behavior across a variety of architectural platforms, we introduce the Integrated Performance Monitoring (IPM) package: a portable, lightweight, and scalable tool for effectively extracting MPI message-passing overheads. A performance characterization study is conducted on some of the world's most powerful supercomputers, including the superscalar Seaborg (IBM Power3+) and CC-NUMA Columbia (SGI Altix), as well as the vector-based Earth Simulator (NEC SX-6 enhanced) and Phoenix (Cray X1) systems. In-depth analysis shows that in order to bridge the gap between theoretical and sustained system performance, it is critical to gain a clear understanding of how the distinct parts of large-scale parallel applications interact with the individual subcomponents of HEC platforms.

Keywords: Cosmic Microwave Background, MADCAP, Altix Columbia, Earth Simulator, X1 Phoenix, Power3 Seaborg, parallel performance characterization

1 Introduction

The Cosmic Microwave Background (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 cosmology, and measuring their detailed statistical properties has been a high priority in the field since its serendipitous discovery in 1965. Since these anisotropies are $O(10^{-5})$ in temperature and $O(10^{-7})$ or less in polarization, and are imprinted on a background that has been cooled by the expansion of the Universe to only 2.7 K today, harnessing the extraordinary scientific potential of the CMB requires precise measurements of the microwave sky at high resolution. The progressive reduction of the resulting datasets, first to a pixelized sky map, then to an angular power spectrum, and finally to cosmological parameters, is a computationally intensive endeavor. The problem is exacerbated by an explosion in dataset sizes as cosmologists try to make more and more accurate measurements of the CMB. High-end computing (HEC) has become an essential part of CMB data analysis, and the effective use of such resources requires a detailed understanding of their performance under the demands of real CMB data analysis algorithms and implementations.

CMB data analyses have typically been performed on superscalar-based commodity microprocessors due to their generality, scalability, and cost effectiveness. However, two recent innovative parallel-vector architectures — the Earth Simulator (ES) and the Cray X1 — promise to narrow the growing gap between sustained and peak performance for many classes of scientific applications. In addition, the new Columbia system at NASA, constructed in just four months and continuously operational during the build, brings an unprecedented level of computational power at a fraction of the cost of typical supercomputers. In order to characterize what these platforms offer scientists that rely on HEC, it is imperative to critically evaluate and compare them in the context of demanding scientific applications [2, 3, 4, 5].

In this work, we present MADbench, a lightweight version of the Microwave Anisotropy Dataset Computational Analysis Package (MADCAP) CMB power spectrum estimation code [1] that retains the operational complexity and integrated system requirements. We compare the performance of MADbench on the ES (NEC SX-6 enhanced) and Phoenix (Cray X1) against those obtained on Columbia (SGI Altix) and Seaborg (IBM Power3). To quantify communication behavior

Table 1. Architectural specifications of the Power3, Altix, ES, and X1

Platform	CPU/ Node	Clock (MHz)	Peak (GF/s)	Mem BW (GB/s)	Peak (bytes/flop)	MPI Lat (usec)	Netwk BW (GB/s/CPU)	Bisection BW (bytes/flop)	Network Topology
Power3	16	375	1.5	0.7	0.47	16.3	0.13	0.087	Fat-tree
Altix	2	1500	6.0	6.4	1.1	2.8	0.40	0.067	Fat-tree
ES	8	500	8.0	32.0	4.0	5.6	1.5	0.19	Crossbar
X1	4	800	12.8	34.1	2.7	7.3	6.3	0.088	2D torus

across this spectrum of architectures, we developed and utilized the Integrated Performance Monitoring (IPM) package: a portable, lightweight, and scalable tool for effectively extracting MPI message-passing overheads. In-depth analysis shows that in order to bridge the gap between theoretical and sustained system performance, it is critical to gain a clear understanding of how the distinct parts of large-scale parallel applications interact with the individual subcomponents of HEC platforms.

2 Target HEC Platforms

We begin by briefly describing the salient features of the four parallel HEC architectures that are examined here (Table 1 presents a summary). Note that the vector machines have higher peak performance and better system balance than the superscalar platforms. The ES and X1 have high memory bandwidth relative to peak CPU speed (bytes/flop), allowing them to more effectively feed the arithmetic units. Additionally, the custom vector interconnects show superior characteristics in terms of measured inter-node MPI latency [6, 9], point-to-point messaging (network bandwidth), and all-to-all communication (bisection bandwidth) — both in raw performance and as a ratio of peak processing speed. Overall, the ES appears to be the most balanced system, while the Altix shows the best architectural characteristics among the superscalar platforms.

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 very 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. In this model, disk I/O uses the switch fabric, sharing bandwidth with message-passing traffic. 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). The distributed filesystem was configured with 16 GPFS servers, each with 32GB of main memory that can be used to cache files and metadata. The total size of the filesystem was 30TB, with a block size of 256KB.

2.2 Columbia (Altix 3000)

Introduced in early 2003, the SGI Altix 3000 systems are an adaptation of the Origin 3000, which use SGI's NUMAflex global shared-memory architecture. Such systems allow access to all data directly and efficiently, without having to move them through I/O or networking bottlenecks. The NUMAflex design enables the processor, memory, I/O, interconnect, graphics, and storage to be packaged into modular components, called "bricks." The primary difference between the Altix and the Origin systems is the C-Brick, used for the processor and memory. This computational building block for the Altix consists of four Intel Itanium2 processors (in two nodes), local memory, and a two-controller ASIC called the Scalable Hub (SHUB). Each SHUB interfaces to two CPUs in one node, along with memory, I/O devices, and other SHUBs. The Altix cache-coherency protocol is implemented in the SHUB that integrates both the snooping operations of the Itanium2 and the directory-based scheme used across the NUMAflex interconnection fabric. 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.

The 64-bit Itanium2 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, a relatively large register set helps mitigate this issue. The superscalar processor implements the Explicitly Parallel Instruction set Computing (EPIC) technology where instructions are organized into 128-bit VLIW bundles.

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. All Altix experiments reported here were performed on the 10,240-processor Columbia system running 64-bit Linux version 2.4.21, the world's second-most powerful supercomputer [8] located at NASA Ames Research Center. The Columbia experiments used a 6.4TB parallel XFS filesystem with a 35-fiber optical channel connection to the CPUs.

2.3 Earth Simulator

The vector processor of the Earth Simulator (ES) 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 ES 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 ES 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 ES vector unit is cache-less; memory latencies are masked by overlapping pipelined vector operations with memory fetches. Each SMP contains eight processors that share the node's memory. The ES is the world's third-most powerful supercomputer [8], 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 ES runs Super-UX, a 64-bit Unix operating system based on System V-R3 with BSD4.2 communication features. Each group of 16 nodes has a pool of RAID disks (720GB per node) attached via fiber channel switch. The filesystem used for our tests is SFS, with a block size of 4MB. Each node has a separate filesystem, in contrast to the other architectures studied here. As remote access is not available, the reported experiments were performed during the authors' visit to the ES Center and by ES Center collaborators in late 2004.

2.4 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. All reported X1 results were obtained on Phoenix, the 256-MSP system running UNICOS/mp 2.4 and operated by Oak Ridge National Laboratory (ORNL). This machine has four nodes available for I/O, each of which is connected to a RAID array using fiber channel arbitrated loop protocol. Data transfer from a batch MSP must travel over the interconnect to one of the I/O nodes. The filesystem used in this study is a 4TB XFS filesystem, with a block size of 64KB.

3 Integrated Performance Monitoring

Integrated Performance Monitoring (IPM) is a portable performance profiling infrastructure that binds together communication, computation, and memory information from the tasks in a parallel application into a single application-level profile. IPM provides a lightweight portable mechanism for workload-wide parallel profiling that does not require user intervention and scales to thousands of processors. As the application executes on the parallel platform, IPM records a per-process profile of computation and communication using a small fixed memory footprint and very low CPU overhead. When the application terminates, a report of the aggregate profile is generated. In this work, IPM was used on all the target architectures as a probe of the amount of communication. There are other ways one can obtain this information within a procedural context. For example, MPIP [10] is a name-shifted profiling library that records the call stack for each MPI call. Cur-

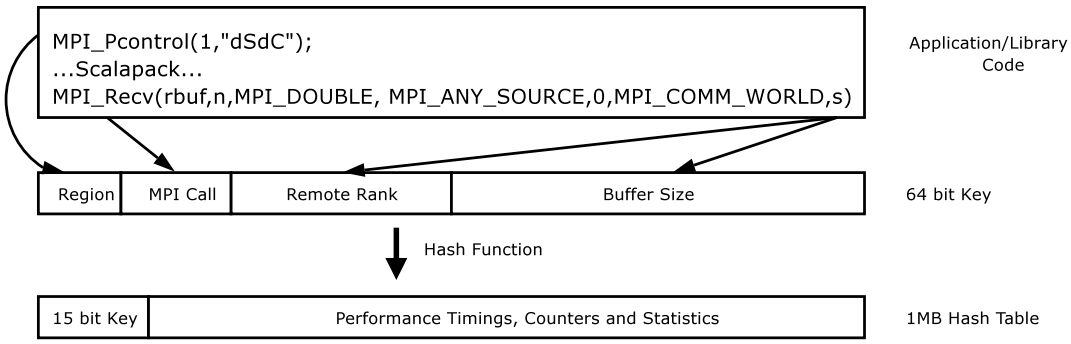


Figure 1. Program control flow from the application through MPI to the IPM layer

rently, determining the call stack on a variety of architectures presents a technical challenge we chose to avoid.

The basic mechanism by which IPM operates is the name-shifted profiling interface specified in the MPI standard. Name-shifted profiling wrappers have been widely used [7] to determine the nature of communication within parallel codes. The entry point to each MPI call is replaced with another that wraps a call to the PMPI entry point. `MPI_Pcontrol` is called directly from the application to mark the code region to be profiled. The profile is stored in a hash table that is keyed off the region, MPI call, message size, and the rank to which the message was sent or received (in the case of point-to-point communications). Figure 1 shows a schematic of this process. We are thus able to compute basic statistics like average, minimum, and maximum for the times that each rank spends in an MPI call for every buffer size. For point-to-point calls, we also track the other rank involved in the communication and thereby determine the topological character of the communication.

The principal benefit of using IPM is that it provides sufficient contextual clarity to separately analyze the communication in each of the distinct computational steps within MADbench. Since each functional component has a specific algorithmic or data movement role in the overall calculation, having region-specific timings allows one to compare measurements with estimates derived analytically or from microbenchmarks. Analyses of parallel performance that treats the application as a whole does not provide this level of detail. Because MADbench uses ScaLAPACK extensively, IPM also provides insights into the communication primitives used by an otherwise opaque library call.

To ensure that IPM accurately measures communication time, we investigated the different implementations of ScaLAPACK across the target machines. Since IPM records the time spent in MPI calls, we must know that all communication is done via MPI or that non-MPI communication is negligible. For the architectures studied here, we were able to verify this either by checking directly with vendor documentation or by testing with platform-specific profiling tools (PAT, ftmpirun, etc.). Phoenix and ES spend negligible time in non-MPI communication within ScaLAPACK, while the Seaborg and Columbia implementations are based entirely on MPI.

In the current work, we use IPM strictly for determining information about communication occurring within the MADbench code. No analysis of data obtained from hardware performance counters was conducted. The present analysis can be extended by investigating the distribution of message sizes, communication topology, and hardware performance counters occurring within each of the code regions and across architectures. We currently lack the right tools for fully analyzing the performance profiles generated by IPM. In future, we plan to expand the scope and level of detail by such analysis.

4 MADbench Overview

The MADCAP CMB angular power spectrum estimator uses Newton-Raphson iteration to locate the peak of the spectral likelihood function. This involves calculating the first two derivatives of the likelihood function with respect to the power spectrum coefficients C_l (since the CMB is azimuthally symmetric, the full spherical harmonic basis can be reduced to include the l-modes only). For a noisy pixelized CMB sky map $d_p = s_p + n_p$ (data is signal plus noise) and pixel-pixel correlation matrix $D_{pp'} = S_{pp'}(C_l) + N_{pp'}$, the log-likelihood that the observed data comes from some underlying set of C_l is

$$\mathcal{L}(d_p|C_l) = -\frac{1}{2}\left(d^T D^{-1} d + \mathrm{Tr}[\ln D]\right)$$

whose first two derivatives with respect to the C_l are

$$\frac{\partial \mathcal{L}}{\partial C_l} = \frac{1}{2}\left(d^T D^{-1}\frac{\partial S}{\partial C_l} D^{-1} d - \mathrm{Tr}\left[D^{-1}\frac{\partial S}{\partial C_l}\right]\right)$$

$$\frac{\partial^2 \mathcal{L}}{\partial C_l \partial C_{l'}} = -d^T D^{-1}\frac{\partial S}{\partial C_l} D^{-1}\frac{\partial S}{\partial C_{l'}} D^{-1} d + \frac{1}{2}\mathrm{Tr}\left[D^{-1}\frac{\partial S}{\partial C_l} D^{-1}\frac{\partial S}{\partial C_{l'}}\right]$$

yielding a Newton-Raphson correction to an initial guess of the spectral coefficients

$$\delta C_l = \left[\frac{\partial^2 \mathcal{L}}{\partial C_l \partial C_{l'}} \right]^{-1} \frac{\partial \mathcal{L}}{\partial C_{l'}}$$

Typically, the data has insufficient sky coverage and/or resolution to obtain each multipole coefficient individually, and so we bin them instead, replacing C_l with C_b.

Given the size of the pixel-pixel correlation matrices, it is important to minimize the number of matrices simultaneously held in memory; the MADCAP implementation is constrained to have no more than three matrices in core at any one time. It's operational steps, and their scalings for a map with NPIX pixels and NMPL multipoles in NBIN bins, are:

1. Calculate the signal correlation derivative matrices $\partial S/\partial C_b$; complexity is O(NMPL $\times$ NPIX^2).
2. Form and then invert the data correlation matrix $D = \sum_b C_b \, \partial S/\partial C_b + N$; complexity is O(NPIX^3).
3. For each bin, calculate $W_b = D^{-1} \partial S/\partial C_b$; complexity is O(NBIN $\times$ NPIX^3).
4. For each bin, calculate $\partial \mathcal{L}/\partial C_b$; complexity is O(NBIN $\times$ NPIX^2).
5. For each bin-bin pair, calculate $\partial^2 \mathcal{L}/\partial C_b \partial C_{b'}$; complexity is O(NBIN^2 $\times$ NPIX^2).
6. Invert the bin correlation matrix $\partial^2 \mathcal{L}/\partial C_b \partial C_{b'}$ and calculate the spectral correction δC_b; complexity is O(NBIN^3).

Note that steps 4–6 are computationally subdominant as NBIN << NMPL << NPIX.

All pixel-pixel correlation matrices are ScaLAPACK block-cyclic distributed, so each processor carries a unique subset of the rows and columns of each matrix. All I/O is performed in terms of this distribution, with each processor writing its subset of the full matrix to its own file. This has the advantage of simplicity when the matrices need to be read back in, and avoids any problems with the ES's distributed filesystem. However, since the reading or writing of a matrix now involves all processors simultaneously trying to perform their own I/O, there is the possibility for significant contention. This is minimized by restricting the actual number of processors performing concurrent I/O to a user-specified fraction of the total, implemented using a simple token-passing scheme. The specified fraction can be different for reading (RMOD) and writing (WMOD), and numerical experiments are performed to optimize the values of these parameters — as well as the ScaLAPACK block size (BSIZE) — on each architecture.

The full MADCAP spectral estimator includes a large number of special-case features, from preliminary data checking to marginalization over foreground templates, that dramatically increase the size and complexity of the code without altering its basic operational structure. For simplicity, we have therefore developed a stripped-down version, called MADbench, expressly designed for benchmarking that preserves all the computational challenges of the problem while removing extraneous bells and whistles. MADbench consists of three functions: dSdC, invD, and W, that respectively perform the dominant steps 1–3 above. (W also performs a slightly simplified version of the subdominant steps 4–6 to confirm code correctness.) In order to understand the overall performance, it is useful to lay out the overall sequence of calculation, communication, and I/O for each function:

dSdC For each bin b, calculate the local subset of the $\partial S/\partial C_b$ matrix using Legendre recursion and write it to disk. This function involves neither communication nor reading. The innermost loop of the recursion had to be rewritten to take advantage of the vector architectures.

invD For each bin b, read in the local subset of the $\partial S/\partial C_b$ matrix and weighted-accumulate it to build the local subset of the data correlation matrix D; invert this by Cholesky decomposition and triangular solution using the ScaLAPACK pdpotrf and pdpotri routines. This function involves no writing.

W For each bin b, read in the local subset of the $\partial S/\partial C_b$ matrix and perform the dense matrix-matrix multiplication $W_b = D^{-1} \partial S/\partial C_b$ using the ScaLAPACK pdgemm routine. This function again requires no writing. Note that the multiplications are entirely independent of one another; we therefore compare two implementations: the first proceeding as above, and the second introducing a level of gang-parallelism, with NGANG of the $\partial S/\partial C_b$ matrices each being remapped to a different subset of the processors using the ScaLAPACK pdgemr2d function and all of the processor-gangs then simultaneously calling pdgemm.

5 Parallel Performance Results

MADbench requires six parameters to be specified. Two set the size of the run (NPIX and NBIN), and one the degree of gang-parallelism (NGANG). The other three parameters (RMOD, WMOD, and BSIZE) need to be tuned for each architecture and were set as follows: Seaborg (4,1,128), Columbia (1,1,32), ES (1,1,64), and Phoenix (1,1,128). We ran MADbench on P = 16, 64, and 256 processors on all architectures, and on P = 1024 where possible (Seaborg and ES). In each case, the number of bins was fixed at NBIN = 16, and the number of pixels chosen to keep the total data volume per processor constant, NPIX = 5000$\times$P/16. Each configuration was run with no gang-parallelism (NGANG = 1), and then only W was rerun with NGANG = 16. Each experiment was repeated several times, with the best runtimes reported.

5.1 Overall Performance

We first give a high level overview of MADbench performance on the architectures described in Section 2 for a variety of problem sizes. In the next subsection (Section 5.2), we increase the level of detail by conducting similar performance analyses within the context of each important functional component (dSdC, invD, and W) that constitutes MADbench. In a broad sense, MADbench spends almost all of its time calculating, communicating, or reading/writing data to disk. We identify the time associated with each of these activities as CALC, MPI, and I/O in the remainder of this paper. We add another metric, called LBST, which measures load balancing including synchronization time. When proceeding from one functional component to the next, we impose a barrier in order to have a well-defined boundary between the phases. LBST records the time when all processes do not reach these barriers at exactly the same time.

It is useful to identify the runtimes associated with each of these four broad categories to most directly understand which subsystems of a computing platform are stressed during the course of a calculation. As the problem size and concurrency increase, we expect changes in the relative fraction of time spent in I/O, MPI, CALC, and LBST to provide the clearest indication of the nature of the scaling bottlenecks present in MADbench.

Ultimately, however, the absolute timings are the most important factor when making architectural performance comparisons. Table 2 shows the execution times for each functional component of MADbench for various processor counts on each of the four platforms. For all but the very smallest problem size (P = 16), the ES shows the best absolute performance in terms of time-to-solution (Phoenix outperforms the ES for P= 16). In terms of percentage of theoretical peak performance (% TPP), Seaborg demonstrates the best results except for P = 1024 when the ES is superior.

We now focus on the scaling of I/O, MPI, CALC, and LBST timings for each architecture. Figure 2 shows the relative amount of time spent in each of these activities.

On Seaborg, the relative amount of time spent in I/O decreases as the problem size and concurrency increase. However, the I/O time actually increases more than threefold as P grows from 16 to 1024. This shows that while there is contention for the I/O resource, it is not sufficient to impact code scalability significantly. The relative amounts of time spent communicating (MPI) and computing (CALC) remain constant for P = 64, 256, and 1024. Since Seaborg is composed of 16-way SMPs, the MPI time for P = 16 is misleading because no data movement occurs over the switch. CALC is the predominant activity; this is understandable given the relatively slow (375 MHz) CPUs in this SP cluster.

Table 2. Overall MADbench performance

Platform	P	Time (s) dSdC	invD	W	TOTAL	% TPP TOTAL
Seaborg (Power3)	16	42.9	36.5	311.2	390.6	45.0
	64	44.5	60.1	608.8	713.4	49.0
	256	56.1	107.7	1209.9	1373.7	50.7
	1024	121.3	214.8	2466.7	2802.8	49.6
ES (SX-6 enhanced)	16	23.8	43.8	63.3	130.9	25.2
	64	28.9	48.9	98.9	176.7	37.1
	256	29.2	58.6	173.9	261.7	49.9
	1024	31.8	94.5	321.4	447.7	58.3
Phoenix (X1)	16	3.1	9.2	45.5	57.8	35.6
	64	51.8	106.0	86.1	243.9	16.8
	256	1029.3	379.0	421.4	1829.7	4.5
Columbia (Altix)	16	58.2	7.2	163.2	228.6	19.2
	64	117.2	14.1	306.6	437.9	19.9
	256	483.4	23.8	409.4	916.6	19.0

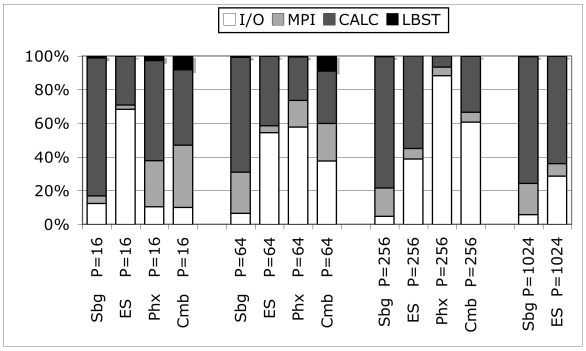


Figure 2. Relative timings for MADbench

On the ES, the I/O timings do not increase appreciably with concurrency. The filesystems on the ES are semi-local RAID arrays attached to each 128 CPUs. Since the I/O resources scale with concurrency, the performance trend makes sense. As with Seaborg, the relative amount of time spent in I/O decreases as the problem is scaled up. This however comes at the cost of not having a parallel filesystem. In calculations where external data must be read in, the staging time of datasets to the different filesystems could become a significant bottleneck. The overall trends in MPI and CALC are smooth functions of concurrency. The amount of MPI time increases dramatically (10x) between 16 and 1024 tasks, but represents only a small fraction of total time. CALC increases by a factor of 6x over the same range but doubles its contribution to the overall runtime.

On Phoenix, the trends seen in CALC, MPI, and I/O are smooth functions of concurrency. For this X1 architecture, the scaling of I/O becomes the predominant bottleneck at about P = 64. It is unclear to the authors what is the expected level of I/O parallelism for this machine. From our measurements, it appears that a scaling threshold in I/O has been crossed or that the manner

in which MADbench performs I/O interferes with the filesystem. The primary contribution to the I/O time was from writing and synchronizing at the barrier following the writes. This is consistent with the presence of a limit to the number of simultaneous parallel writes and thus a serialization in I/O for large problem sizes.

Unlike the other platforms, the trends are not as smooth on Columbia. For some runs, we observe the MPI time to decrease with increasing problem size. This is clearly unexpected and the variation in these timings is under investigation. For instance, although all calculations were performed on the same 512-way SMP, it is unclear whether the placement of tasks was done in a consistent fashion by the OS. However, results demonstrate that I/O is the predominant component of runtime at and above 256-way runs. As with Phoenix, most of the I/O time is spent in writing data within dSdC. We note that a significant (3.6x) increase in I/O occurs between P = 16 and 64. Since there are 35 independent links to the filesystem in use, it is possible that the number of channels to disk had become saturated.

5.2 Performance of Individual Functions

The detailed performance of MADbench is better understood by separately examining each of the functional components of the code.

dSdC Table 3 presents the timing breakdowns for dSdC. Results show that for P = 64, the ES achieves the highest raw performance, approximately 1.5x, 1.8x, and 4x faster than Seaborg, Phoenix, and Columbia, respectively. Due to the weak scaling nature of the problem, CALC remains roughly constant as concurrency increases, with Phoenix showing the fastest values; however, ES attains the best CALC TPP (19.0%) followed by Phoenix (16.5%), Seaborg (6.7%) and Columbia (2.7%). The original dSdC implementation relied on fine-grained recursive computations that prevented effective vectorization. A customized version was therefore developed for the ES and X1 so that at each recursion a large batch of angular separations is computed in the inner loop, allowing high vector performance. This, coupled with their superior memory bandwidth characteristics lead to the significantly higher performance of the vector architectures over the superscalar machines.

In terms of dSdC's I/O behavior (dominated by data writing), all systems show significant performance degradation at the highest concurrencies — except for ES whose local filesystem is insensitive to the degree of parallelism. This is particularly true for Phoenix and Columbia where performance drops precipitously for 256 processors, resulting in I/O bandwidth of only 0.2 Mb/s/P and 0.4 Mb/s/P, respectively.

Table 3. Detailed timings for dSdC

Platform	P	Time (s)				Mb/s/P	% TPP	
		LBST	I/O	CALC	TOT	BW	TOT	CALC
Sbg	16	3.3	9.7	29.9	42.9	20.6	4.7	6.7
	64	4.1	11.1	29.3	44.5	18.0	4.5	6.8
	256	4.1	22.2	29.8	56.1	9.0	3.6	6.7
	1024	4.6	86.9	29.8	121.3	2.3	1.6	6.7
ES	16	0.1	21.7	2.0	23.8	9.2	1.6	19.2
	64	0.1	26.8	2.0	28.9	7.5	1.3	19.0
	256	0.1	27.1	2.0	29.2	7.4	1.3	18.9
	1024	0.7	29.1	2.0	31.8	6.9	1.2	18.5
Phx	16	0.1	1.6	1.4	3.1	124.2	7.6	16.6
	64	0.1	50.3	1.4	51.8	4.0	0.5	16.5
	256	0.1	1027.8	1.4	1029.3	0.2	0.0	16.3
Cmb	16	17.6	21.3	19.3	58.2	9.4	0.9	2.6
	64	18.4	80.4	18.4	117.2	2.5	0.4	2.7
	256	1.0	464.1	18.3	483.4	0.4	0.1	2.7

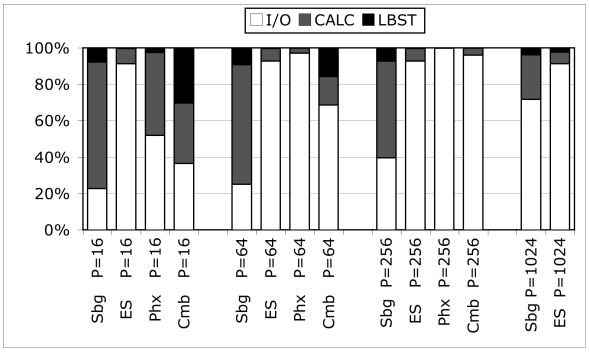


Figure 3. Relative timings for dSdC

Finally, the LBST metric (computational load imbalance) shows a non-trivial cost for the superscalar systems, accounting for approximately 15% of the total overhead on Seaborg and Columbia for the 64-processor case; the vector systems are mostly unaffected.

Figure 3 shows the relative performance breakdown of dSdC for each platform. These results clearly demonstrate that at high concurrencies, the relative I/O cost increasingly dominates the overall runtime. Note, however, that on the ES, the ratio between computation and I/O remains roughly constant due to the local filesystem.

invD The breakdown of invD runtime components are shown in Table 4. Since this function performs dense linear algebra operations, CALC is expected to grow linearly with increasing numbers of processors and pixels, while I/O requirements remain constant. Columbia shows the best overall performance in terms of total runtime and TPP, followed by ES, Seaborg, and Phoenix. For example, for P = 64, Columbia achieves a total TPP of 24.6% compared to only 1.5% on Phoenix. Since the numerical kernel of invD is computationally inten-

Table 4. Detailed timings for invD

Platform	P	Time (s) MPI	I/O	CALC	TOT	Mb/s/P BW	% TPP TOT	CALC	MPI+ CALC
Sbg	16	4.3	21.0	11.0	36.5	9.5	19.0	63.0	45.2
	64	19.7	22.6	17.7	60.1	8.9	23.1	78.3	37.1
	256	50.5	23.7	33.3	107.7	8.5	25.8	83.4	33.1
	1024	109.4	40.4	64.7	214.8	5.0	25.9	85.8	31.9
ES	16	0.8	41.0	2.0	43.8	4.9	3.0	65.1	47.3
	64	1.8	43.3	3.8	48.9	4.6	5.3	68.0	46.4
	256	3.7	47.3	7.6	58.6	4.2	8.9	68.4	46.2
	1024	7.6	71.7	15.1	94.5	2.8	11.0	69.2	45.9
Phx	16	5.0	2.4	1.8	9.2	84.7	8.9	45.7	12.0
	64	13.8	89.1	3.1	106.0	2.2	1.5	51.7	9.6
	256	41.4	331.7	5.9	379.0	0.6	0.9	55.6	6.9
Cmb	16	3.4	1.1	2.6	7.2	180.2	24.0	67.6	28.9
	64	8.5	1.2	4.3	14.1	166.7	24.6	81.1	27.2
	256	13.3	1.0	9.4	23.8	210.5	29.1	73.8	30.6

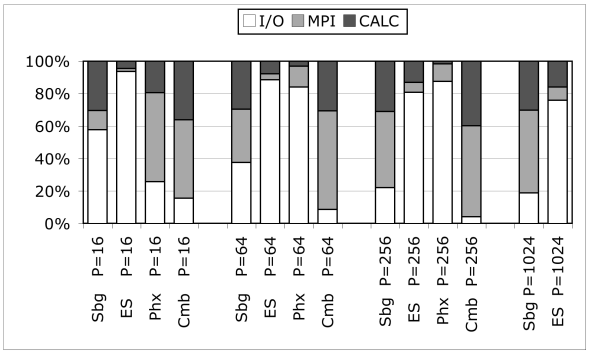


Figure 4. Relative timings for invD

sive, all architectures sustain a high CALC TPP: for P = 64, Columbia achieves 81.1% of peak, compared with 78.3%, 68.0%, and 51.7% on Seaborg, ES, and Phoenix, respectively. Thus, for this functional component of MADbench, the superscalar architectures outperform the vector systems in computational efficiency.

In terms of MPI costs, the high-performance single-stage switch of the ES shows the lowest runtime requirements: 4.7x, 7.6x, and 11x lower than the communication overhead of Columbia, Phoenix, and Seaborg, respectively. The read-dominated I/O overhead varies greatly among the architectures. Columbia shows the most impressive performance, approximately 20x, 36x, and 75x higher than Seaborg, ES, and Phoenix. Note that for P = 256, Phoenix's I/O bandwidth diminishes to only 0.5 Mb/s/P. The source of these large discrepancies is primarily due to memory caching that affects read/write rates differently on each platform.

Figure 4 presents the relative cost of invD components for each of the studied architectures. Observe that on the vector systems, I/O is responsible for a significant fraction of the total runtime, while the MPI overhead is relatively negligible. The opposite is true with the superscalar systems, which show relatively low I/O overheads compared with the MPI requirements. Finally, note that the computational requirements consume a roughly constant fraction of overhead for each architecture regardless of processor count.

W Table 5 presents the breakdown of timing components for W. Overall, the vector architectures achieve higher performance than the superscalar systems. For example, for P = 64, Phoenix is approximately 1.1x, 3.6x, and 7x faster than ES, Columbia, and Seaborg, respectively. However, ES achieves the highest overall TPP at 63.2% while Columbia shows the lowest at 27.2%. Like invD, W performs dense linear algebra calculations and therefore achieves high CALC TPP across all architectures. ES shows the most impressive results for the numerical computation, sustaining over 92% of peak, compared with 69% or less on the other platforms.

Both MPI and I/O shows similar performance characteristics to that of invD, since it is also comprised of dense algebra calculations. The ES once again achieves the lowest communication overhead, while the relatively old switch technology of Seaborg results in the highest MPI time. Columbia demonstrates impressive I/O (read dominated) characteristics, sustaining over 215 Mb/s/P for P = 64: 1.6x, 20x, and 28x faster than Phoenix, ES, and Seaborg, respectively. However, for P = 256, both Phoenix and Columbia show anomalously slow I/O behavior, achieving only 0.8 Mb/s/P and 2.2 Mb/s/P.

The relative breakdown of costs for W is presented in Figure 5. These results show that, due to the large computational requirements, the MPI and I/O overheads represent a small fraction of the total runtime — thereby allowing high sustained performance across all architec-

Table 5. Detailed timings for W

Platform	P	Time (s) MPI	I/O	CALC	TOT	Mb/s/P BW	% TPP TOT	CALC	MPI+ CALC
Sbg	16	13.1	17.5	279.5	311.2	11.4	53.6	59.6	57.0
	64	79.3	18.2	509.2	608.8	11.0	54.8	65.5	56.6
	256	180.2	18.9	1008.3	1209.9	10.6	55.1	66.1	56.1
	1024	413.4	30.5	2019.5	2466.7	6.6	54.1	66.0	54.8
ES	16	2.7	26.6	33.9	63.3	7.5	49.4	92.1	85.4
	64	5.5	26.0	67.4	98.9	7.7	63.2	92.8	85.7
	256	12.3	27.2	134.4	173.9	7.4	71.9	93.0	85.2
	1024	25.3	27.5	268.5	321.4	7.3	77.8	93.1	85.1
Phx	16	10.8	2.0	31.2	45.5	100.5	42.9	62.6	46.5
	64	24.7	1.5	58.4	86.1	130.7	45.4	66.8	47.0
	256	49.2	257.0	113.7	421.4	0.8	18.5	68.7	48.0
Cmb	16	81.3	0.4	80.4	163.2	500.0	25.5	51.8	25.8
	64	155.2	0.9	149.1	306.6	215.1	27.2	55.9	27.4
	256	40.3	91.6	276.1	409.4	2.2	40.7	60.4	52.7

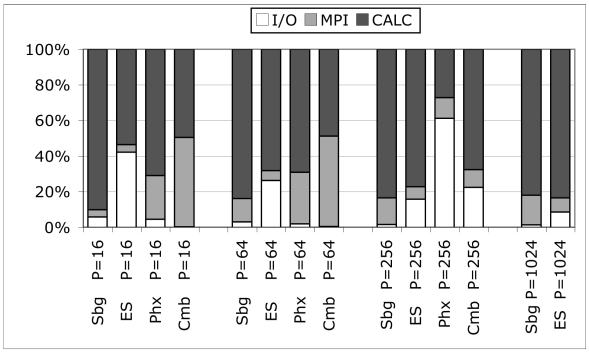


Figure 5. Relative timings for W

tures for this phase of the MADbench calculation.

5.3 Performance with Multi-Gang

In this section, we analyze the trade-offs inherent in the multi-gang strategy. As mentioned in Section 4, the function W can operate in two modes: either all of the processors work on each matrix-matrix multiplication in turn, or the processors divide into `NGANG` gangs and each gang independently performs `NBIN`/`NGANG` multiplications. Increasing the data density in this fashion should increase the efficiency of this step, but since the matrices are initially block-cyclically distributed over the whole processor grid, they must be redistributed over the gang processor grid before multiplication. The relative efficiency between single- and multi-gang approaches is therefore a trade-off between the benefit of the faster multiplication and the cost of the remapping.

Table 6 and Figure 6 show the absolute and relative timings for W using 16 gangs (with the remap times for the 1-gang runs shown in parentheses of Table 6). Compared to Figure 5 (single gang W performance), observe that the MPI time drops considerably on all architectures, as expected. In addition, it is clear that the CALC time changes by only a small amount. This is because the optimized `dgemm` algorithm performs well over a large range of matrix sizes.

The relative I/O cost shows no overall pattern, but depends on the architecture. Within W, the amount of I/O is identical in the single- and multi-gang cases; however, the interleaving of I/O and calculation is changed. For the 1-gang runs, one matrix is read and then multiplied; whereas for the 16-gang runs, all the matrices are read in and remapped, and only then are all 16 multiplies performed simultaneously. On Seaborg, there is a modest increase in I/O time when going to 16 gangs at all concurrencies. On the ES, I/O varies only slightly. Phoenix and Columbia show only a small change for P = 16 and 64, but large changes at P = 256. In the case of Phoenix,

Table 6. Detailed timings for W using 16 gangs (and remap cost for 1-gang runs)

Platform	P	Time (s)					% TPP		
		MPI	I/O	CALC	REMAP	RTOT	RTOT	CALC	MPI+CALC
Sbg	16	0.0	23.7	207.3	50.4 (9.6)	281.5	59.2	80.4	80.4
	64	13.8	20.8	509.3	74.4 (10.5)	618.3	53.9	65.4	63.7
	256	30.9	36.0	1016.9	193.6 (11.6)	1277.3	52.2	65.6	63.6
	1024	83.8	43.7	2005.1	246.1 (26.1)	2378.7	56.1	66.5	63.8
ES	16	0.0	23.7	31.4	19.6 (12.8)	74.6	41.9	99.7	99.7
	64	0.5	24.6	66.8	25.5 (15.6)	117.4	53.2	93.6	92.8
	256	1.9	24.7	133.3	35.1 (31.9)	195.0	64.1	93.8	92.4
	1024	7.1	24.8	266.4	61.6 (51.8)	359.8	69.5	93.9	91.4
Phx	16	0.0	0.9	23.3	7.9 (1.9)	32.0	61.0	83.9	83.9
	64	4.3	2.0	54.9	7.2 (3.2)	68.5	57.1	71.1	65.9
	256	17.1	303.5	110.9	73.3 (38.2)	504.7	15.5	70.4	61.0
Cmb	16	0.0	1.0	50.4	15.8 (1.0)	67.2	62.0	82.7	82.7
	64	12.0	1.6	161.4	24.6 (1.0)	199.6	41.7	51.6	48.1
	256	12.7	7.6	313.0	— (—)	—	—	53.2	51.2

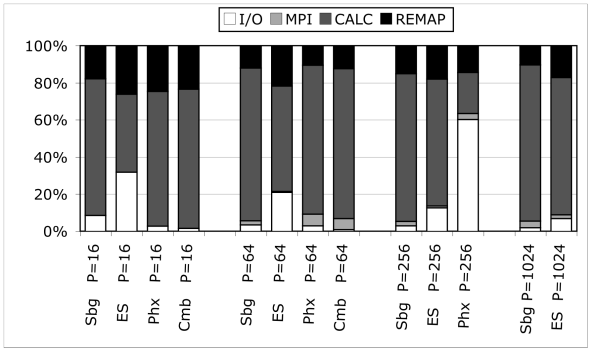


Figure 6. Relative timings for W (16 gangs)

the I/O seems generally very sensitive at P = 256; in the case of Columbia, the cause of the difference is unknown at this time.

The remapping is performed using the ScaLAPACK routine `pdgemr2d`. (The ScaLAPACK remapping function on Columbia failed for P = 256; SGI engineers are currently addressing this problem.) For simplicity, and to maintain a consistent mapping of the data in memory, the routine is called even in the single-gang case. However the current implementation of `pdgemr2d` takes no advantage of potential simplifications, but always assumes the worst case data-remapping scenario. It is thus extremely slow, as evidenced by the parenthetical 1-gang timings in Table 6, where the remappings are equivalent to each processor performing 17 completely local 12.5Mb memcopies — over 51 seconds on 1024 processors of the ES! Given this, we are in the process of developing a custom remapping function which will considerably reduce the overhead of this phase for both single- and multi-gang simulations.

6 Summary and Conclusions

In this paper, we presented MADbench, a synthetic benchmark that preserves the full computational complexity of the underlying scientific application. We tested the performance of its computation, communication, and I/O modules both individually and collectively on four of the world's most powerful supercomputers.

Figure 7 illustrates the percentage of theoretical peak performance obtained on each architecture, using 16 gangs in W. Each entry actually consists of four overlayed bars showing the percentage of peak achieved by considering (i) the total runtime, (ii) all but the remapping time, (iii) all but the remapping and I/O times, and (iv) all but the remapping, I/O, and MPI times. (Note that the 256-processor Columbia run is missing the white bar since the remapping function failed.)

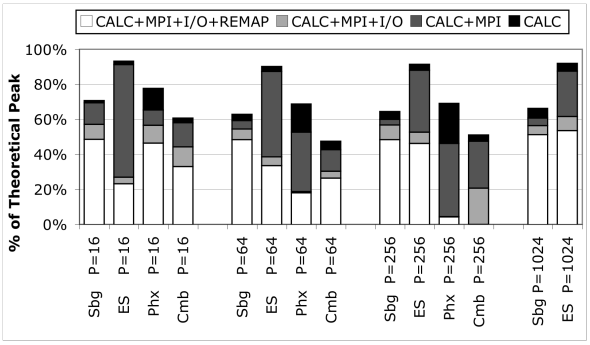


Figure 7. Percentage of peak performance for MADbench with 16-gang parallelism

A more perceptive way to interpret Figure 7 is that the height of the light grey bars reflects the relative cost of remapping, dark grey the relative cost of I/O, and black the relative cost of MPI. From this perspective, I/O is a minor issue for Seaborg, but a significant factor for the ES at all concurrencies, for Phoenix 64-way or more, and for Columbia 256-way. The I/O cost for increasing parallelism is broadly flat for Seaborg, significantly decreasing for the ES (reflecting its slow but perfectly scalable filesystem), but dramatically increasing for Phoenix and Columbia. It is also apparent that MPI constitutes a small overhead for all but Phoenix where it is significant and increases with concurrency.

We also introduced a new performance profiling tool, called IPM, and showed that it can be used successfully on a wide variety of computational platforms to extract useful performance data. In particular, we were able to identify the ScaLAPACK function `pdgemr2d` as a bottleneck in our algorithm.

A more general conclusion of this work is that greater clarity in the application context and overall specificity of performance timings greatly benefit understanding of how the distinct parts of large-scale parallel applications interact with the major subsystems of HEC platforms. It is therefore insufficient to report only the total runtime for a full-blown scientific application and expect to understand its parallel performance. As witnessed in Figures 2-7, the achieved performance will not approximate that seen in simple computational kernels which model only the compute phase and often vastly overestimate sustained performance. Such in-depth analysis is critical in first understanding and then bridging the gap between theoretical and sustained parallel performance.

Acknowledgments

The authors thank the ESC for providing access to the ES, and ORNL for access to the X1. The authors are grateful to Yoshinori Tsuda, David Parks, and Michael Wehner for collecting much of the latest ES data. All authors from LBNL were supported by the Office of Advanced Scientific Computing Research in the DOE Office of Science under contract DE-AC03-76SF00098.

References

[1] J. Borrill. MADCAP: The Microwave Anisotropy Dataset Computational Analysis Package. In *5th European SGI/Cray MPP Wkshp.*, 1999.

[2] T. Dunigan Jr., M. Fahey, J. White III, and P. Worley. Early evaluation of the Cray X1. In *SC2003*, 2003.

[3] K. Nakajima. Three-level hybrid vs. flat MPI on the Earth Simulator: Parallel iterative solvers for finite-element method. In *6th IMACS Intl. Symp. on Iterative Methods in Scientific Computing*, 2003.

[4] 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 *6th Intl. Mtg. on High Performance Computing for Computational Science*, pages 219–232, 2004.

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

[6] ORNL Cray X1 Evaluation. http://www.csm.ornl/∼dunigan/cray.

[7] R. Rabenseifner. Recent advances in Parallel Virtual Machine and Message Passing Interface. In *6th European PVM/MPI Users' Group Mtg.*, volume LNCS 1697, pages 35–42, 1999.

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

[9] H. Uehara, M. Tamura, and M. Yokokawa. MPI performance measurement on the Earth Simulator. Technical Report 15, NEC R&D, 2003.

[10] J. Vetter and A. Yoo. An empirical performance evaluation of scalable scientific applications. In *SC2002*, 2002.