A Performance Evaluation of the Cray X1 for Scientific Applications

Leonid Oliker[1], Rupak Biswas[2], Julian Borrill[1], Andrew Canning[1], Jonathan Carter[1], M. Jahed Djomehri[2], Hongzhang Shan[1], and David Skinner[1]

[1] CRD/NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA 94720
{loliker, jdborril, acanning, jtcarter, hshan, deskinner}@lbl.gov
[2] NAS Division, NASA Ames Research Center, Moffett Field, CA 94035
{rbiswas, djomehri}@nas.nasa.gov

Abstract. The last decade has witnessed a rapid proliferation of superscalar cache-based microprocessors to build high-end capability and capacity computers primarily because of their generality, scalability, and cost effectiveness. However, the recent development of massively parallel vector systems is having a significant effect on the supercomputing landscape. In this paper, we compare the performance of the recently-released Cray X1 vector system with that of the cacheless NEC SX-6 vector machine, and the superscalar cache-based IBM Power3 and Power4 architectures for scientific applications. Overall results demonstrate that the X1 is quite promising, but performance improvements are expected as the hardware, systems software, and numerical libraries mature. Code reengineering to effectively utilize the complex architecture may also lead to significant efficiency enhancements.

1 Introduction

The last decade has witnessed a rapid proliferation of superscalar cache-based microprocessors to build high-end capability and capacity computers [12]. This is primarily because their generality, scalability, and cost effectiveness convinced computer vendors, buyers, and users that vector architectures hold little promise for future large-scale supercomputing systems.

Two major elements have dramatically affected this common perception. The first is the recent availability of the Japanese Earth Simulator [3] that, based on NEC SX-6 vector technology, is the world's most powerful supercomputer both in terms of peak (40.96 TFlops/s) and sustained LINPACK (87.5% of peak) performance. The second is the constant degradation in sustained performance (now typically less than 10% of peak) for scientific applications running on SMP clusters built from processors using superscalar technology. The Earth Simulator, on the other hand, has demonstrated sustained performance of almost 65% of peak for a production global atmospheric simulation [11].

In order to quantify what a vector capability entails for scientific communities that rely on modeling and simulation, it is critical to evaluate it in the context of demanding computational algorithms. In earlier work [9], we conducted an

evaluation of the SX-6 with the NAS Parallel Benchmarks and a suite of six applications. Here, we compare the performance of the Cray X1 vector testbed system at Oak Ridge National Laboratory [8] with that of the SX-6, and the superscalar cache-based IBM Power3 and Power4 architectures for scientific codes.

We first compare memory bandwidth and scatter/gather hardware behavior of a triad summation using regularly and irregularly strided data. Next, we use two synthetic benchmarks: a fast Fourier transform and an N-body solver to further compare the four target architectures. Finally, we present performance results for three scientific applications from cosmology (MADCAP), materials science (PARATEC), and fluid dynamics (OVERFLOW-D). For each of these benchmarks and applications, we examine the effort required to port them to the X1. Both intra- and inter-node parallel performance (in GFlops/s per processor) is reported for our application suite when running a small and a large test case. X1 results are generally promising; however, we expect performance improvements as the machine stabilizes to production mode. Note that none of the applications have been specifically optimized for the X1, so appropriate reengineering to utilize the architecture effectively would also be beneficial.

2 Target Architectures

We give a brief description of the salient architectural features of the X1, as well as those of the other machines that we examined. Table 1 presents a summary of their node characteristics and measured inter-node MPI latency [8, 13]. Observe that the X1 and the SX-6 are designed to sustain a higher bytes/flop ratio.

Table 1. Architectural specifications of the X1, Power3, Power4, and SX-6 nodes

Node Type	CPU/ Node	Clock (MHz)	Peak (GFlops/s)	Mem. BW (GB/s)	Peak Bytes/Flop	MPI Latency (μsec)
X1	4	800	12.8	34	2.7	7.3
Power3	16	375	1.5	1.0	0.67	16.3
Power4	32	1300	5.2	6.4	1.2	17.0
SX-6	8	500	8.0	32	4.0	5.6

2.1 Cray X1

The recently-released X1 is designed to combine traditional vector strengths with the generality and scalability features of superscalar cache-based parallel systems (see schematic in Fig. 1). The computational core, called the single-streaming processor (SSP), contains two 8-stage vector pipes running at 800 MHz. Each SSP contains 32 vector registers holding 64 double-precision words, and operates at 3.2 GFlops/s peak. All vector operations are performed under a bit mask, allowing loop blocks with conditionals to compute without the need for scatter/gather operations. Each SSP can have up to 512 addresses simultaneously in flight, thus hiding the latency overhead for a potentially significant number of memory fetches. The SSP also contains a two-way out-of-order superscalar

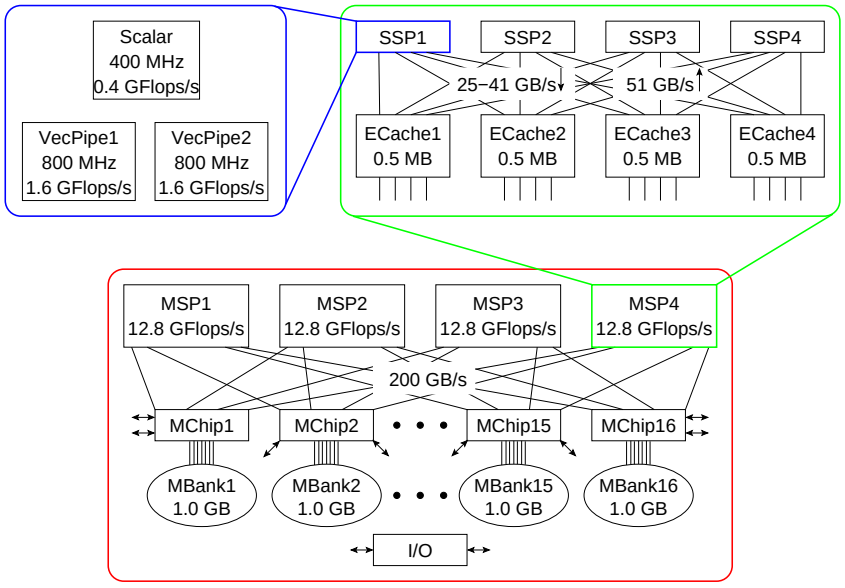

Fig. 1. Architecture of the X1 node (in red), MSP (in green), and SSP (in blue).

processor running at 400 MHz with two 16KB caches (instruction and data). The scalar unit operates at an eighth of 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 for vector architectures that allows extremely high bandwidth (25–51 GB/s) for computations with temporal data locality. An X1 node consists of four MSPs sharing a flat memory through 16 memory controllers (MChips). Each MChip is attached to a local memory bank (MBank), for an aggregate of 200 GB/s node bandwidth. Additionally, MChips can be used to directly connect up to four nodes (16 MSPs) and participate in remote address translation. To build large configurations, a modified 2D torus interconnect is implemented via specialized routing chips. The torus topology allows scalability to large processor counts with relatively few links compared with the crossbar interconnect of the IBM Power systems or the Earth Simulator; however, the torus suffers from limited bisection bandwidth. Finally, the X1 is a globally addressable architecture, with specialized hardware support that allows processors to directly read or write remote memory addresses in an efficient manner.

The X1 programming model leverages parallelism hierarchically. At the SSP level, vector instructions allow 64 SIMD operations to execute in a pipeline fashion, thereby masking memory latency and achieving higher sustained performance. MSP parallelism is obtained by distributing loop iterations across the four SSPs. The compiler must therefore generate both vectorizing and multi-streaming instructions to effectively utilize the X1. Intra-node parallelism across the MSPs is explicitly controlled using shared-memory directives such as OpenMP

or Pthreads. Finally, traditional message passing via MPI is used for coarse-grain parallelism at the inter-node level. In addition, the hardware supported globally addressable memory allows efficient implementations of one-sided communication libraries (SHMEM, MPI-2) and implicitly parallel languages (UPC, CAF).

All X1 experiments reported here were performed on the 128-MSP system running UNICOS/mp 2.3.07 and operated by Oak Ridge National Laboratory.

2.2 IBM Power3

The Power3 runs were conducted on the 380-node IBM pSeries system running AIX 5.1 and located at Lawrence Berkeley National Laboratory. 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 GFlops/s. It has a pipeline of only three cycles, 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.

2.3 IBM Power4

The Power4 experiments were performed on the 27-node IBM pSeries 690 system running AIX 5.1 and operated by Oak Ridge National Laboratory. Each 32-way SMP consists of 16 Power4 chips (organized as four 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 of 5.2 GFlops/s. The superscalar out-of-order architecture can exploit instruction level parallelism through its eight execution units. Branch prediction hardware minimizes the effects of the relatively long pipeline (six cycles) necessitated by the high frequency design. Each processor has its own private L1 cache (64KB instruction, 32KB data) with prefetch hardware; however, both cores share a 1.5MB unified L2 cache. The L3 can operate as a stand-alone 32MB cache, or be combined with other L3s on the same MCM to create a 128MB interleaved cache. Multi-node configurations employ the Colony interconnect, but future systems will use the lower latency Federation switch.

2.4 NEC SX-6

The SX-6 results were obtained on the single-node system running SUPER-UX at the Arctic Region Supercomputing Center. The 500 MHz processor contains an 8-way replicated vector pipe capable of issuing a MADD per cycle, for a peak performance of 8.0 GFlops/s. The processors contain 72 vector registers, each holding 256 64-bit words. For non-vectorizable instructions, the SX-6 has a 500 MHz scalar processor with 64KB instruction and data caches, and 128 general-purpose registers. The 4-way superscalar unit has a peak of 1.0 GFlops/s and supports branch prediction, data prefetching, and out-of-order execution.

Unlike conventional architectures, the SX-6 vector unit lacks data caches. It therefore masks memory latencies by overlapping pipelined vector operations with memory fetches. The SX-6 uses high speed SDRAM with a peak bandwidth of 32 GB/s per CPU; enough to feed one operand per cycle to each of the replicated pipe sets. The nodes can be used to build large-scale multi-processor systems; for instance, the Earth Simulator contains 640 SX-6 nodes (but with a slightly faster memory), connected through a single-stage crossbar.

3 Microbenchmarks

We are developing a microbenchmark suite (called Xtreams) that measures low-level machine characteristics such as memory subsystem behavior and scatter/gather hardware support. Fig. 2(a) presents asymptotic memory bandwidth of the triad summation: $a(i) = b(i) + s \times c(i)$ using various strides for the four architectures in our study. The SX-6 achieves the best bandwidth: up to one, two, and three orders of magnitude better than the X1, Power4, and Power3, respectively. Observe that certain strides impact X1 and SX-6 bandwidth quite pronouncedly, by an order of magnitude or more. Analysis shows that strides containing multiples of two worsen performance due to increased DRAM bank conflicts. On the Power architectures, a precipitous drop in the transfer rate occurs for small strides, due to loss of cache reuse. This drop is more severe on the Power4, because of its more complicated cache structure. Note that the purpose of this benchmark is to measure the expected memory behavior via simple Fortran code fragments; however, machine theoretical performance can be approached by using architecture-specific source transformations. For example, the X1 average bandwidth can be increased by up to a factor of 20x through F90 syntax and compiler directives for non-caching data and loop unrolling.

Fig. 2(b) presents single-processor memory bandwidth of indirect addressing through vector triad gather/scatter operations of various data sizes. For smaller

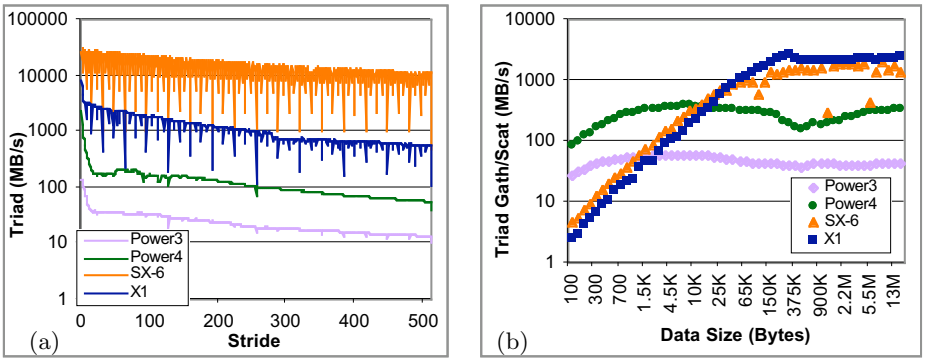

Fig. 2. Single-processor Xtreams triad performance (in MB/s) using (a) regularly strided data of various strides, and (b) irregularly strided data of various sizes.

sizes, the cache-based architectures show better data rates for indirect access to memory; but the X1 and the SX-6 effectively utilize their gather/scatter hardware to outperform the cache-based systems for larger data sizes. The X1 memory bandwidth behavior is comparable to that of the SX-6 (asymptotic performance is marginally higher); however, the X1 processor is more than 50% faster, indicating a poorer architectural balance between computational rate and memory subsystem performance (see Table 1).

4 Synthetic Benchmarks

We next use two synthetic benchmarks to compare and contrast the performance of the four target architectures. The first is FT, a fast Fourier transform (FFT) kernel, from the NAS Parallel Benchmarks (NPB) suite [7]. FFT is a simple evolutionary problem in which a time-dependent partial differential equation is transformed to a set of uncoupled ordinary differential equations in phase space. The bulk of the computational work consists of a set of discrete multi-dimensional FFTs, each accessing the same single large array multiple times, but with different strides every time.

The second synthetic benchmark is N-BODY, an irregularly structured code that simulates the interaction of a system of particles in 3D over a number of time steps, using the hierarchical Barnes-Hut algorithm. There are three main stages: building an octree to represent the distribution of the particles; browsing the octree to calculate the forces; and updating the particle positions and velocities based on the computed forces. Data access is scattered, and involves indirect addressing and pointer chasing. N-BODY requires all-to-all all-gather communication and demonstrates unpredictable send/receive patterns. The MPI version of this benchmark is derived from the SPLASH-2 suite [14].

FT and N-BODY results in GFlops/s per processor are presented in Fig. 3. Initially, FT (problem size Class B) did not perform well on the vector machines because it used a fixed block size of 16 words. To increase vector length and thus improve performance, a block size of 64 was chosen for the X1, even though a non-blocking strategy performed best on the SX-6. Figure 3(a) shows that the vector architectures have comparable raw performance, with the SX-6 sustaining a significantly higher fraction of peak. However, both the X1 and SX-6 are much faster than the scalar platforms, achieving almost a 17x and 4x speed up over the Power3 and Power4, respectively. Scalability deteriorates for the X1 (at $P = 32$) and the Power4 (at $P = 64$) due to inferior inter-node communication.

Performance for the N-BODY benchmark (see Fig. 3(b)) is completely different in that the scalar processors are far superior than the vector processors. For example, the Power3 achieves 7% of peak while the X1 runs at 0.3%. The dominant part of the code is browsing the octree and computing the inter-particle forces. Octree browsing involves significant pointer chasing, which is a very expensive operation on vector machines and cannot be vectorized well. We tried to improve vectorization by separating the octree browsing and force calculation phases; however, results were only negligibly better. These results demon-

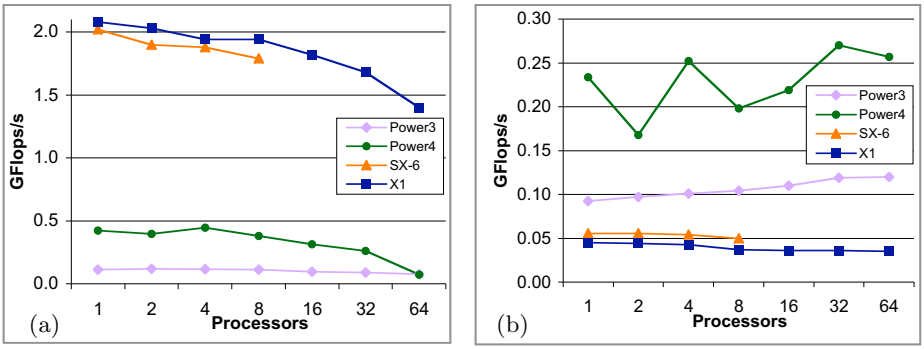

Fig. 3. Per-processor performance (in GFlops/s) for (a) FT and (b) N-BODY.

strate that although the vector architectures are equipped with scalar units, non-vectorized codes will perform extremely poorly on these platforms.

5 Scientific Applications

Three scientific applications were chosen to measure and compare the performance of the X1 with that of the SX-6, Power3, and Power4. The applications are: MADCAP, a cosmology code that extracts tiny signals from extremely noisy observations of the Cosmic Microwave Background; PARATEC, a first principles materials science code that solves the Kohn-Sham equations to obtain electronic wavefunctions; and OVERFLOW-D, a computational fluid dynamics production code that solves the Navier-Stokes equations around complex aerospace configurations. One small and one large case are tested for each application. SX-6 results are presented for only one node ($P \leq 8$) while those on Power3 ($P > 16$), Power4 ($P > 32$), and X1 ($P > 4$) are for multi-node runs. Performance is shown in GFlops/s per processor, and were obtained with `hpmcount` on the Power systems, `ftrace` on the SX-6, and `pat` on the X1.

5.1 Cosmology

From the application area of cosmology, we examined the Microwave Anisotropy Dataset Computational Analysis Package (MADCAP) [1]. MADCAP 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 ScaLAPACK libraries for its efficient parallel solution. The goal is to maximize the likelihood function of all possible cosmologies given the data d. With minimal assumptions, this can be reduced to maximizing d's (reduced to a pixelized sky map) Gaussian log-likelihood $\mathcal{L}(d|C_b) = -\frac{1}{2}(d^T D^{-1} d - \mathrm{Tr}\,[\ln D])$ over all possible power spectrum coefficients C_b, where $D = \langle dd^T \rangle$ is the data correlation matrix.

Using Newton-Raphson iteration to locate the peak of $\mathcal{L}(d|C_b)$ requires the evaluation of its first two derivatives with respect to C_b. This involves first building D as the sum of the experiment-specific noise correlations and theory-specific signal correlations, and then solving a square linear system for each of the $\mathcal{N}_b$ spectral coefficients. To minimize communication overhead, this step is computed through explicit inversion of D (symmetric positive definite) and direct matrix-matrix multiplication. These operations scale as $\mathcal{N}_b \mathcal{N}_p^3$ for a map with $\mathcal{N}_p$ pixels, typically 10^4–10^5 for real experiments.

To take advantage of large parallel computer systems while maintaining high performance, the MADCAP implementation has recently been rewritten to exploit the parallelism inherent in performing $\mathcal{N}_b$ independent matrix-matrix multiplications. The analysis is split into two steps: first, all the processors build and invert D; then, the processors are divided into gangs, each of which performs a subset of the multiplications (gang-parallel). Since the matrices involved are block-cyclically distributed over the processors, this incurs the overhead of redistributing the matrices between the two steps.

X1 Porting Porting MADCAP to vector architectures is straightforward, since the package utilizes ScaLAPACK to perform dense linear algebra calculations. However, it was necessary to rewrite the dSdC routine that computes the pixel-pixel signal correlation matrices, scales as $\mathcal{N}_b \mathcal{N}_p^2$, and does not have any BLAS library calls. The basic structure of dSdC loops over all pixels and calculates the value of Legendre polynomials up to some preset degree for the angular separation between these pixels. On superscalar architectures, this constituted a largely insignificant amount of work, but since the computation of the polynomials is recursive, prohibiting vectorization, the cost was appreciable on both the X1 and the SX-6. At each iteration of the Legendre polynomial recursion in the rewritten dSdC routine, a large batch of angular separations was computed in an inner loop. Compiler directives were required to ensure vectorization for both the vector machines.

Performance Results Table 2 first shows the performance of MADCAP for a small synthetic dataset with $\mathcal{N}_p = 8192$ and $\mathcal{N}_b = 16$ on each of the four architectures under investigation, without the use of gang-parallelism. Note that for maximum efficiency, the processor sizes are set to squares of integers. As expected, the single processor runs achieve a significant fraction of the peak performance since the analysis is dominated by BLAS3 operations (at least 60%). However, as we use more processors with a fixed data size, the density of the

Table 2. Per-processor performance of MADCAP on target machines

	Power3		Power4		SX-6		X1	
P	GFlops/s	% Peak	GFlops/s	% Peak	GFlops/s	% Peak	GFlops/s	% Peak
Small synthetic case: $\mathcal{N}_p = 8192$, $\mathcal{N}_b = 16$, without gang-parallelism								
1	0.844	56.3	1.98	38.1	4.55	56.9	5.36	41.9
4	0.656	43.7	1.23	23.7	2.66	33.2	4.26	33.3
9	0.701	46.7	1.11	21.3	—	—	3.22	25.2
16	0.520	34.7	0.731	14.1	—	—	2.48	19.4
25	0.552	36.8	0.628	12.1	—	—	1.84	14.4
Large real case: $\mathcal{N}_p = 14996$, $\mathcal{N}_b = 16$, with gang-parallelism								
4	0.831	55.4	1.90	36.5	4.24	53.0	5.66	44.2
8	0.688	45.9	1.62	31.2	3.24	40.5	5.22	40.8
16	0.615	41.0	1.26	24.2	—	—	4.09	32.0
32	0.582	38.8	0.804	15.5	—	—	3.35	26.2
64	0.495	33.0	0.524	10.1	—	—	2.15	16.8

data on each processor decreases, the communication overhead increases, and performance drops significantly. Observe that the X1 attains the highest raw performance, achieving factors of 6.5x, 3.5x, and 1.6x speedup compared with the Power3, Power4, and SX-6 respectively on 4 processors. With increasing processor counts, the fraction of time spent in BLAS3 computations decreases, causing performance degradation. The average vector length was about 62 per SSP (vector length 64) for all processor counts. It is therefore surprising that MADCAP did not attain a higher percentage of peak on the X1. We believe this is due, in part, to overhead associated with MADCAP's I/O requirements, which is currently under investigation.

Table 2 also shows the performance of the gang-parallel implementation of MADCAP for a larger real dataset from the Millimeter-wave Anisotropy Experiment Imaging Array (MAXIMA) [5], with $\mathcal{N}_p = 14996$ and $\mathcal{N}_b = 16$. MAXIMA is a balloon-borne experiment with an array of 16 bolometric photometers optimized to map the CMB anisotropy over hundreds of square degrees. The gang parallel technique is designed to preserve the density of data during matrix-matrix multiplication and produce close to linear speedup. In this regard, the results on the Power3 and Power4 are somewhat disappointing. This is because the other steps in MADCAP do not scale that well, and the efficiency of each gang suffers due to increased memory contention. The X1 shows significantly faster run times compared with the Power3/4 (about a factor of 4x); however, its sustained performance declines precipitously with increasing numbers of processors. This is due to the non-vectorizable code segments (such as I/O and data redistribution), which account for higher fractions of the overall run time.

5.2 Materials Science

The application that we selected from materials science is called PARATEC: PARAllel Total Energy Code [10]. The code performs ab-initio quantum-mechani-

cal total energy calculations using pseudopotentials and a plane wave basis set. The pseudopotentials are of the standard norm-conserving variety. 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, having a quantum mechanical treatment of the electrons, to calculate the structural and electronic properties of materials. Codes based on DFT are widely used to study properties such as strength, cohesion, growth, magnetic, optical, and transport for materials like nanostructures, complex surfaces, and doped semiconductors.

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 are built. For this reason, PARATEC generally obtains a high percentage of peak performance on different platforms. The code exploits fine-grained parallelism by dividing the plane wave components for each electron among the different processors [4].

X1 Porting PARATEC is an MPI package designed primarily for massively parallel computing platforms, but can also run on serial machines. Since much of the computation involves FFTs and BLAS3, an efficient vector implementation of the code requires these libraries to vectorize well. While this is true for the BLAS3 routines on the X1, the standard 1D FFTs run at a low percentage of peak. Some code transformations were therefore required to convert the 3D FFT routines to use simultaneous ("multiple") 1D FFT calls. Compiler directives were also inserted in certain code segments to force vectorization on the X1 in loops where indirect indexing is used.

Performance Results The results in Table 3 are for 3 CG steps of 250 and 432 Si-atom bulk systems and a standard LDA run of PARATEC with a 25 Ry cut-off using norm-conserving pseudopotentials. A typical calculation would require between 20 and 60 CG iterations to converge the charge density.

Results show that PARATEC vectorizes well on the SX-6 for the small test case. The X1 is significantly slower, even though it has a higher peak speed. One reason is that, on the X1, the code spends a much smaller percentage of the total time in highly optimized 3D FFTs and BLAS3 libraries than any of the other machines. This lowers sustained performance as other parts of the code run well below peak. Some of the loss in X1 scaling for $P > 4$ is also due to inter-node communication and shorter vector lengths. The parallel 3D FFTs require a transformation of the distributed grid which results in global communication. The average vector length drops from 49 for the uni-processor run to 44 on 16 processors. Performance for the large test case is poor for $P = 128$, primarily due to low average vector length (35) and increased communication overhead.

Table 3. Per-processor performance of PARATEC on target machines

	Power3		Power4		SX-6		X1	
P	GFlops/s	% Peak	GFlops/s	% Peak	GFlops/s	% Peak	GFlops/s	% Peak
Small case: 250 Si-atom bulk system								
1	0.915	61.0	2.29	44.0	5.09	63.6	2.97	23.2
2	0.915	61.0	2.25	43.3	4.98	62.3	2.82	22.0
4	0.920	61.3	2.21	42.5	4.70	58.8	2.65	20.7
8	0.911	60.7	2.09	40.2	4.22	52.8	2.48	19.4
16	0.840	56.0	1.57	30.2	—	—	1.99	15.5
32	0.725	48.3	1.33	25.6	—	—	1.67	13.0
Large case: 432 Si-atom bulk system								
32	0.959	63.9	1.49	28.7	—	—	3.04	23.8
64	0.848	56.5	0.75	14.4	—	—	2.82	22.0
128	0.739	49.3			—	—	1.91	14.9

PARATEC runs efficiently on the Power3, but performance on the Power4 is affected by the relatively poor ratio of memory bandwidth to peak performance. Nonetheless, the Power systems obtain a much higher percentage of peak than the X1 for all runs. Faster FFT libraries and some code rewriting to increase vectorization in conjunction with compiler directives could significantly improve the performance of PARATEC on the X1. Initial tests of systems larger than 432 Si-atoms show better scaling performance across all architectures.

5.3 Fluid Dynamics

In the area of computational fluid dynamics (CFD), we selected the NASA Navier-Stokes production application called OVERFLOW-D [6]. The code uses the overset grid methodology [2] 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, unlike OVERFLOW [2], 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 takes advantage of the overset grid system, which offers a natural coarse-grain parallelism. A bin-packing algorithm clusters individual grids into groups, each of which is then assigned to an MPI process. The grouping strategy uses a connectivity test that inspects for an overlap between a pair of grids before assigning them to the same group, regardless of

the size of the boundary data or their connectivity to other grids. The grid-loop in the parallel implementation is subdivided into two procedures: a group-loop over groups, and a grid-loop over the grids within each group. Since each MPI process is assigned to only one group, the group-loop is executed in parallel, with each group performing its own sequential grid-loop. The inter-grid boundary updates within each group are performed as in the serial case. Inter-group boundary exchanges are achieved via MPI asynchronous communication calls.

X1 Porting The MPI implementation of OVERFLOW-D is based on the sequential version that was designed to exploit early Cray vector machines. The same basic program structure is used on all four target architectures except for a few minor changes in some subroutines to meet specific compiler requirements. In porting the code to the X1, three binary input files also had to be converted from the default IEEE format to 64-bit (`integer*8` and `real*8`) data types. Furthermore, for data consistency between FORTRAN and C, the `int` data in all C functions were converted to `long int`. The main loops in compute-intensive sections of the code were multi-streamed and vectorized automatically by the compiler; however, modifications were made to allow vectorization of certain other loops via pragma directives.

Performance Results The results in Table 4 are for 100 time steps of an OVERFLOW-D simulation of vortex dynamics in the complex wake flow region around hovering rotors. A typical calculation would require several thousand time steps. Note that the current MPI implementation of OVERFLOW-D does not allow uni-processor runs. The grid system for the small test case consists of 41 blocks and about 8 million grid points, while that for the large case has 857 blocks and 69 million grid points. The Cartesian off-body wake grids surround the curvilinear near-body grids with uniform resolution, but become gradually coarser upon approaching the outer boundary of the computational domain.

Results for the small case demonstrate that the X1 is about a factor of 2x slower than the SX-6 and 1.5x (4x) faster than the Power4 (Power3), based on GFlops/s per processor. The SX-6 outperforms the other machines; e.g. its execution time for $P = 8$ (1.6 secs) is 90%, 50%, and 21% of the $P = 16$ timings for the X1, Power4, and Power3. The SX-6 also consistently achieves the highest percentage of peak performance while the X1 is the lowest. Scalability is similar for all machines except the Power4, with computational efficiency decreasing for a larger number of MPI tasks primarily due to load imbalance. Performance is particularly poor for $P = 32$ when 41 zones have to be distributed to 32 processors. On the X1, a relatively small average vector length of 24 per SSP explains why the code achieves an aggregate of only 3.6 GFlops/s on 8 processors (3.5% of peak). Although the same general trends hold for the large case, performance is slightly better (for the same number of processors) because of higher computation-to-communication ratio and better load balance. Reorganizing OVERFLOW-D would achieve improve vector performance; however, extensive effort would be required to modify this production code.

Table 4. Per-processor performance of OVERFLOW-D on target machines

	Power3		Power4		SX-6		X1	
P	GFlops/s	% Peak	GFlops/s	% Peak	GFlops/s	% Peak	GFlops/s	% Peak
Small case: 8 million grid points								
2	0.133	8.9	0.394	7.6	1.13	14.1	0.513	4.0
4	0.117	7.8	0.366	7.0	1.11	13.9	0.479	3.7
8	0.118	7.9	0.362	7.0	0.97	12.1	0.445	3.5
16	0.097	6.5	0.210	4.0	—	—	0.433	3.4
32	0.087	5.8	0.115	2.2	—	—	0.278	2.2
Large case: 69 million grid points								
16	0.115	7.7	0.282	5.4	—	—	0.456	3.6
32	0.109	7.3	0.246	4.7	—	—	0.413	3.2
64	0.105	7.0	0.203	3.9	—	—	0.390	3.0
96	0.100	6.7	0.176	3.4	—	—	0.360	2.8
128	0.094	6.3	0.146	2.8	—	—	0.319	2.5

6 Summary and Conclusions

This paper presented the performance of the Cray X1 vector testbed system, and compared it against the cacheless NEC SX-6 vector machine and the superscalar cache-based IBM Power3/4 architectures, across a range of scientific kernels and applications. Microbenchmarking showed that the X1 memory subsystem behaved rather poorly, given its large peak memory bandwidth and sophisticated hardware support for non-unit stride data access. Table 5 summarizes the performance of the X1 against the three other machines, for our synthetic benchmarks and application suite. All our codes, except N-BODY, were readily amenable to vectorization via minor code changes, compiler directives, and the use of vendor optimized numerical libraries. Results show that the X1 achieves high raw performance relative to the Power systems for the computationally intensive applications; however, the SX-6 demonstrated faster runtimes for several of our test cases. Surprisingly, the X1 did not achieve a high fraction of peak performance compared with the other platforms, even though reasonably large vector lengths were attained for the applications. The N-BODY kernel is an irregularly structured code poorly suited for vectorization, and highlights the extremely low performance one should expect when running scalar codes on this architecture.

Table 5. Summary overview of X1 performance

Application	Scientific	Lines	X1 Speedup vs.					
Name	Discipline	of Code	P	Power3	Power4	SX-6	P	Power3	Power4
FT	Kernel	2,000	8	17.3	5.1	1.1	64	18.4	19.4
N-BODY	Kernel	1,500	8	0.4	0.2	0.7	64	0.3	0.1
MADCAP	Cosmology	5,000	4	6.5	3.5	1.6	64	4.3	4.1
PARATEC	Mat. Sc.	50,000	8	2.7	1.2	0.6	64	3.3	3.8
OVERFLOW-D	CFD	100,000	8	3.8	1.2	0.5	128	3.4	2.2

Overall, the X1 performance is quite promising, and improvements are expected as the hardware, systems software, and numerical libraries mature. It is important to note that X1-specific code optimizations have not been performed at this time. This complex vector architecture contains both data caches and multi-streaming processing units, and the optimal programming methodology is yet to be established. For example, increasing cache locality through data blocking will lower the memory access overhead; however, this strategy may reduce the vector length and cause performance degradation. Examining these tradeoffs will be the focus of our future work.

Acknowledgements

The authors thank ORNL for providing access to the Cray X1. All authors from LBNL are supported by Office of Advanced Science Computing Research in the U.S. DOE Office of Science under contract DE-AC03-76SF00098. M.J. Djomehri is an employee of CSC and supported by NASA under contract DTTS59-99-D-00437/A61812D with AMTI/CSC.

References

1. J. Borrill, MADCAP: The Microwave Anisotropy Dataset Computational Analysis Package, in: *Proc. 5th European SGI/Cray MPP Workshop* (Bologna, Italy, 1999) astro-ph/9911389.
2. P.G. Buning *et al.*, Overflow user's manual version 1.8g, *Tech. Rep.*, NASA Langley Research Center, 1999.
3. Earth Simulator Center, See URL *http://www.es.jamstec.go.jp*.
4. G. Galli and A. Pasquarello, First-principles molecular dynamics, *Computer Simulation in Chemical Physics*, Kluwer, 1993, 261–313.
5. S. Hanany *et al.*, MAXIMA-1: A measurement of the Cosmic Microwave Background anisotropy on angular scales of $10'$–5°, *The Astrophysical Journal*, 545 (2000) L5–L9.
6. R. Meakin, On adaptive refinement and overset structured grids, in: *Proc. 13th AIAA Computational Fluid Dynamics Conf.* (Snowmass, CO, 1997) Paper 97-1858.
7. NAS Parallel Benchmarks, See URL *http://www.nas.nasa.gov/Software/NPB*.
8. Oakridge National Laboratory Evaluation of Early Systems, See URL *http://www.csm.ornl.gov/evaluation/*.
9. L. Oliker *et al.*, Evaluation of cache-based superscalar and cacheless vector architectures for scientific computations, in: *Proc. SC2003* (Phoenix, AZ, 2003).
10. PARAllel Total Energy Code, See URL *http://www.nersc.gov/projects/paratec*.
11. S. Shingu *et al.*, A 26.58 Tflops global atmospheric simulation with the spectral transform method on the Earth Simulator, in: *Proc. SC2002* (Baltimore, MD, 2002).
12. Top500 Supercomputer Sites, See URL *http://www.top500.org*.
13. H. Uehara *et al.*, MPI performance measurement on the Earth Simulator, *Tech. Rep.*, NEC Research and Development, 2003.
14. S.C. Woo *et al.*, The SPLASH-2 programs: Characterization and methodological considerations, in: *Proc. 22nd Intl. Symp. on Computer Architecture* (Santa Margherita Ligure, Italy, 1995) 24–36.