Performance Evaluation of the SX-6 Vector Architecture for Scientific Computations

Leonid Oliker, Andrew Canning, Jonathan Carter, John Shalf, David Skinner
CRD/NERSC, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Stéphane Ethier
Princeton Plasma Physics Laboratory, Princeton University, Princeton, NJ 08453

Rupak Biswas, Jahed Djomehri*, and Rob Van der Wijngaart*
NAS Division, NASA Ames Research Center, Moffett Field, CA 94035

Abstract

The growing gap between sustained and peak performance for scientific applications is a well-known problem in high performance computing. The recent development of parallel vector systems offers the potential to reduce this gap for many computational science codes and deliver a substantial increase in computing capabilities. This paper examines the intranode performance of the NEC SX-6 vector processor, and compares it against the cache-based IBM Power3 and Power4 superscalar architectures, across a number of key scientific computing areas. First, we present the performance of a microbenchmark suite that examines many low-level machine characteristics. Next, we study the behavior of the NAS Parallel Benchmarks. Finally, we evaluate the performance of several scientific computing codes. Overall results demonstrate that the SX-6 achieves high performance on a large fraction of our application suite and often significantly outperforms the cache-based architectures. However, certain classes of applications are not easily amenable to vectorization and would require extensive algorithm and implementation reengineering to utilize the SX-6 effectively.

1 Introduction

The rapidly increasing peak performance and generality of superscalar cache-based microprocessors long led researchers to believe that vector architectures hold little promise for future large-scale computing systems [15]. Due to their cost effectiveness, an ever-growing fraction of today's supercomputers employ commodity superscalar processors, arranged as systems of interconnected SMP nodes. However, the growing gap between sustained and peak performance for scientific applications on such platforms has become well known in high performance computing.

The recent development of parallel vector systems offers the potential to reduce this performance gap for a significant number of scientific codes, and to increase computational power substantially [14]. This was highlighted dramatically when the Japanese Earth Simulator [2] results were published [20, 21, 24]. The Earth Simulator, based on NEC SX-6[1] *vector technology*, achieves five times the LINPACK performance with almost half the number of processors of the IBM SP-based ASCI White, one of the world's most powerful supercomputers [8], built using *superscalar technology*. In order to quantify what this new capability entails for scientific communities that rely on modeling and simulation, it is critical to evaluate these two microarchitectural approaches in the context of demanding computational algorithms.

---

*Employee of Computer Sciences Corporation.

[1] Also referred to as the Cray SX-6 due to Cray's agreement to market NEC's SX line.

In this paper, we compare the performance of the NEC SX-6 vector processor against the cache-based IBM Power3 and Power4 architectures for several key scientific computing areas. We begin by evaluating memory bandwidth and MPI communication speeds, using a set of microbenchmarks. Next, we evaluate five of the well-known NAS Parallel Benchmarks (NPB) [4, 11], using problem size Class B. Finally, we present performance results for a number of numerical codes from scientific computing domains, including plasma fusion, astrophysics, fluid dynamics, materials science, magnetic fusion, and molecular dynamics. Since most modern scientific codes are already tuned for cache-based systems, we examine the effort required to port these applications to the vector architecture. We focus on serial and intranode parallel performance of our application suite, while isolating processor and memory behavior. Future work will explore the behavior of multi-node vector configurations.

2 Architectural Specifications

We briefly describe the salient features of the three parallel architectures examined. Table 1 presents a summary of their intranode performance characteristics. Notice that the NEC SX-6 has significantly higher peak performance, with a memory subsystem that features a three to six times larger bytes/flop ratio than the IBM Power systems.

Node Type	CPU/ Node	Clock (MHz)	Peak (Gflops/s)	Memory BW (GB/s)	Peak Bytes/Flop	Memory Latency (μsec)
Power3	16	375	1.5	1.0	0.67	8.6
Power4	32	1300	5.2	6.4	1.2	3.0
SX-6	8	500	8.0	32	4.0	2.1

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

2.1 Power3

The IBM Power3 was first introduced in 1998 as part of the 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 GFlops/s. The Power3 has a short pipeline of only three cycles, resulting in relatively low 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 and a 128KB 128-way set associative L1 data cache, as well as 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 IBM Colony switch using an omega-type topology.

The Power3 experiments reported in this paper were conducted on a single Nighthawk II node of the 208-node IBM pSeries system (named Seaborg) running AIX 5.1, Parallel Environment 3.2, C 6.0, Fortran 8.1, and located at Lawrence Berkeley National Laboratory.

2.2 Power4

The pSeries 690 is the latest generation of IBM's RS/6000 series. 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 performance of 5.2 Gflops/s. Two load-store units, each capable of independent address generation, feed the two double precision MADDers. The superscalar out-of-order architecture can exploit instruction level parallelism through its eight execution units. Up to

eight instructions can be issued each cycle into a pipeline structure capable of simultaneously supporting more than 200 instructions. Advanced branch prediction hardware minimizes the effects of the relatively long pipeline (six cycles) 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. Certain data access patterns may therefore cause L2 cache conflicts between the two processing units. The directory for the L3 cache is located on-chip, but the memory itself resides off-chip. 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 interleaved cache of up to 128MB. Multi-node Power4 configurations are currently available employing IBM's Colony interconnect, but future large-scale systems will use the lower latency Federation switch.

The Power4 experiments reported here were performed on a single node of the 27-node IBM pSeries 690 system (named Cheetah) running AIX 5.1, Parallel Environment 3.2, C 6.0, Fortran 7.1, and operated by Oak Ridge National Laboratory.

2.3 SX-6

The NEC SX-6 vector processor uses a dramatically different architectural approach than conventional cache-based systems. Vectorization exploits regularities in the computational structure to expedite uniform operations on independent data sets. Vector arithmetic instructions involve identical operations on the elements of vector operands located in the vector register. Many scientific codes allow vectorization, since they are characterized by predictable fine-grain data-parallelism that can be exploited with properly structured program semantics and sophisticated compilers. The 500 MHz SX-6 processor contains an 8-way replicated vector pipe capable of issuing a MADD each cycle, for a peak performance of 8 Gflops/s per CPU. The processors contain 72 vector registers, each holding 256 64-bit words.

For non-vectorizable instructions, the SX-6 contains a 500 MHz scalar processor with a 64KB instruction cache, a 64KB data cache, and 128 general-purpose registers. The 4-way superscalar unit has a peak of 1 Gflops/s and supports branch prediction, data prefetching, and out-of-order execution. Since the vector unit of the SX-6 is significantly more powerful than its scalar processor, it is critical to achieve high vector operation ratios, either via compiler discovery or explicitly through code (re-)organization.

Unlike conventional architectures, the SX-6 vector unit lacks data caches. Instead of relying on data locality to reduce memory overhead, memory latencies are masked by overlapping pipelined vector operations with memory fetches. The SX-6 uses high speed SDRAM with peak bandwidth of 32GB/s per CPU: enough to feed one operand per cycle to each of the replicated pipe sets. Each SMP contains eight processors that share the node's memory. The nodes can be used as building blocks of large-scale multi-processor systems; for instance, the Earth Simulator contains 640 SX-6 nodes, connected through a single-stage crossbar.

The vector results in this paper were obtained on the single-node (8-way) SX-6 system (named Rime) running SUPER-UX 12.1, C++ rev053, and F90 rev264 at the Arctic Region Supercomputing Center (ARSC) of the University of Alaska.

3 Microbenchmarks

This section presents the performance of a microbenchmark suite that measures some low-level machine characteristics such as memory subsystem behavior and scatter/gather hardware support using STREAM [7]; and point-to-point communication, network/memory contention, and barrier synchronizations via PMB [5].

3.1 Memory Access Performance

First we examine the low-level memory characteristics of the three architectures in our study. Table 2 presents asymptotic unit-stride memory bandwidth behavior of the triad summation: $a(i) = b(i) + s \times c(i)$, using the STREAM benchmark [7]. It effectively captures the peak bandwidth of the architectures, and shows that the SX-6 achieves about 48 and 14 times the performance of the Power3 and Power4, respectively, on a single processor. Notice also that the SX-6 shows negligible bandwidth degradation for up to eight tasks, while the Power3/4 drop by almost 50% for fully packed nodes.

P	Power3	Power4	SX-6
1	661	2292	31900
2	661	2264	31830
4	644	2151	31875
8	568	1946	31467
16	381	1552	—
32	—	1040	—

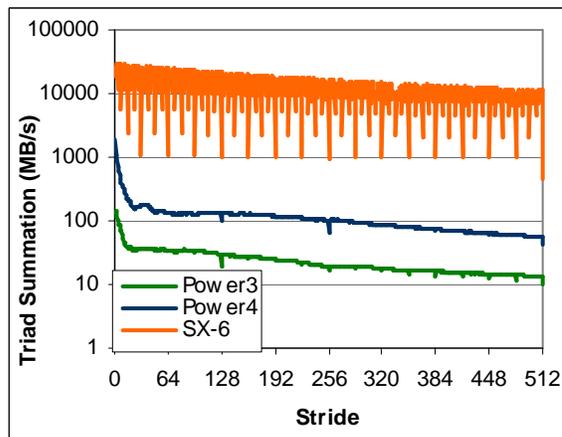Table 2: Per-processor STREAM triad performance (in MB/s) for unit stride.

Figure 1: Single-processor STREAM triad performance (in MB/s) using regularly strided data.

Our next experiment concerns the speed of strided data access on a single processor. Figure 1 presents our results for the same triad summation, but using various memory strides. Once again, the SX-6 achieves good bandwidth, up to two (three) orders of magnitude better than the Power4 (Power3), while showing markedly less average variation across the range of strides studied. Observe that certain strides impact SX-6 memory bandwidth quite pronouncedly, by an order of magnitude or more. Analysis shows that strides containing factors of two worsen performance due to increased DRAM bank conflicts. On the Power architectures, a precipitous drop in data transfer rate occurs for small strides (strides less than 20), due to loss of cache reuse. This drop is more complex on the Power4, because of the extra level of L3 cache.

Finally, Figure 2 presents the memory bandwidth of indirect addressing through vector triad gather and scatter operations of various data sizes on a single processor. For smaller sizes, the cache-based architectures show better data rates for indirect access to memory. However, for larger sizes, the SX-6 is able to utilize its hardware gather and scatter support effectively, outperforming the cache-based systems.

3.2 MPI Performance

Message passing is the most widely used programming paradigm for high-performance parallel systems. The MPI library has become the de facto standard for message passing. It allows both intranode and internode communications, thus obviating the need for hybrid programming schemes for distributed-memory systems. Although MPI increases code complexity compared with shared-memory programming paradigms such as OpenMP, its benefits lie in improved control over data locality and fewer global synchronizations.

Table 3 presents bandwidth figures obtained using the Pallas MPI Benchmark (PMB) suite [5], for exchanging intranode messages of various sizes. The first row shows the best-case scenario when only two processors within a node communicate. Notice that the SX-6 has significantly better performance, achieving more than 19 (7) times the bandwidth of the Power3 (Power4), for the largest messages. The effects of
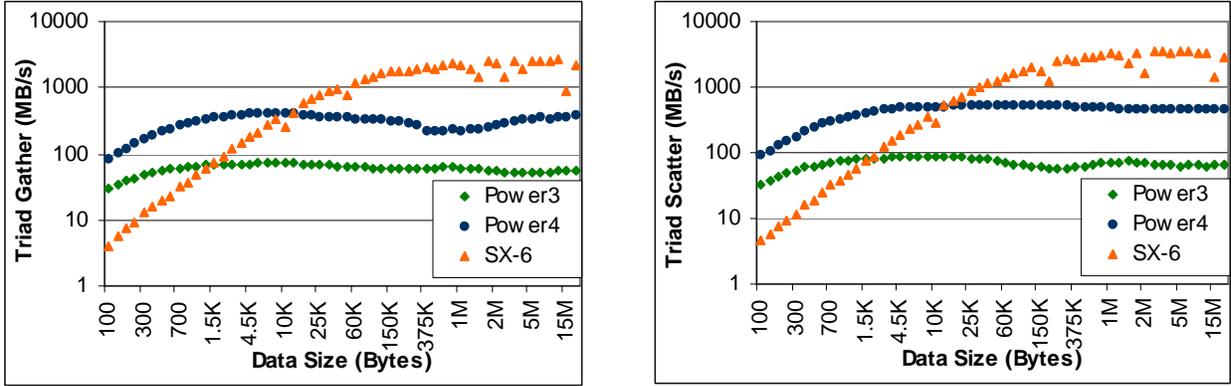
Figure 2: Single-processor STREAM triad performance (in MB/s) using irregularly strided data of various sizes: gather (left) and scatter (right).

network/memory contention are visible when all processors within each SMP are involved in exchanging messages. Once again, the SX-6 dramatically outperforms the Power architectures. For example, a message containing 524288 (2^{19}) bytes, suffers 46% (68%) bandwidth degradation when fully saturating the Power3 (Power4), but only 7% on the SX-6.

P	8192 Bytes			131072 Bytes			524288 Bytes			2097152 Bytes		
	Power3	Power4	SX-6	Power3	Power4	SX-6	Power3	Power4	SX-6	Power3	Power4	SX-6
2	143	515	1578	408	1760	6211	508	1863	8266	496	1317	9580
4	135	475	1653	381	1684	6232	442	1772	8190	501	1239	9521
8	132	473	1588	343	1626	5981	403	1638	7685	381	1123	8753
16	123	469	—	255	1474	—	276	1300	—	246	892	—
32	—	441	—	—	868	—	—	592	—	—	565	—

Table 3: MPI send/receive performance (in MB/s) for various message sizes and processor counts.

Table 4 shows the overhead of MPI barrier synchronization (in μsec). As expected, the barrier overhead on all three architectures increases with the number of processors. For the fully loaded SMP test case, the SX-6 has 7.9 (4.2) times lower barrier cost than Power3 (Power4); while for eight processors, the SX-6 performance is 5.4 and 2.0 times better than the Power3/4.

P	Power3	Power4	SX-6
2	17.1	6.7	5.0
4	31.7	12.1	7.1
8	54.4	19.8	10.0
16	79.1	28.9	—
32	—	42.4	—

Table 4: MPI synchronization overhead (in μsec).

4 Scientific Kernels: NPB

The NAS Parallel Benchmarks (NPB) [4, 11] provide a good middle ground for evaluating the performance of compact, well-understood applications. The NPB were created at a time when vector machines were considered no longer cost effective. Although they were meant not to be biased against any particular architecture, the NPB were written with cache-based systems in mind. Here we investigate the work involved in producing good vector versions of the published NPB that are appropriate for our current study: CG, a sparse-matrix conjugate-gradient algorithm marked by irregular stride resulting from indirect addressing; MG, a quasi-multi-grid code marked by regular non-unit strides resulting from communications between grids of different sizes; FT, an FFT kernel; BT, a synthetic flow solver that features simple recurrences in a different array index in three different parts of the solution process; and LU, a synthetic flow solver featuring recurrences in three array indices simultaneously during the major part of the solution process. We do not report results for the SP benchmark, which is very similar to BT. Table 5 presents MPI performance results for these codes on the Power3, Power4, and SX-6 for medium problem sizes, commonly referred to as Class B (this was the largest problem size that fits in the memory of a single node for all three target machines). Performance results are reported in Mflops/s per processor. To characterize vectorization behavior we also show *average vector length* (AVL) and *vector operation ratio* (VOR). Cache effects are accounted for by TLB misses in % per cycle (TLB) and L1 hits in % per cycle (L1). All performance numbers except Mflops/s—which is reported by the benchmarks themselves— were obtained using the `hpmcount` tool on the Power3/4 and `ftrace` on the SX-6.

Although the CG code vectorizes well and exhibits fairly long vector lengths, uni-processor SX-6 performance is not very good due to the cost of gather/scatter resulting from the indirect addressing. Multi-processor SX-6 speedup degrades as expected with the reduction in vector length. Power3/4 scalability is good, mostly because uni-processor performance is so poor due to the serious lack of data locality.

MG also vectorizes well, and SX-6 performance on up to four processors is good. But a decreased VOR and AVL, combined with the cost of frequent global synchronizations to exchange data between small grids, causes a sharp drop on eight processors. The mild degradation of performance on the Power systems is almost entirely due to the increasing cost of communication, as cache usage is fairly constant, and TLB misses even go down a bit due to smaller per-processor data sets.

FT did not perform well on the SX-6 in its original form, because the computations used a fixed block length of 16 words. But once the code was modified to use a block length equal to the size of the grid (only three lines changed), SX-6 uni-processor performance improved markedly due to increased vector length. Speedup from one to two processors is not good due to the time spent in a routine that does a local data transposition to improve data locality for cache based machines (this routine is not called in the uni-processor run), but subsequent scalability is excellent. Power3/4 scalability is fairly good overall, despite the large communication volume, due to improved data locality of the multi-processor implementation. Note that the Power4's absolute FT performance is significantly better than its performance on CG, although the latter exhibits fewer L1 and TLB misses. The sum of L1 and L2 (not reported here) cache hits on the Power4 is approximately the same for CG and FT for small numbers of processors. We conjecture that FT, with its better overall locality, can satisfy more memory requests from L3 (not measured) than CG.

The BT baseline MPI code performed poorly on the SX-6, because subroutines in inner loops inhibited vectorization. Also, some inner loops of small fixed length were vectorized, leading to very short vector lengths. Subroutine inlining and manual expansion of small loops lead to long vector lengths throughout the single-processor code, and good performance. Increasing the number of processors on the SX-6 causes reduction of vector length (artifact of the 3D domain decomposition) and a concomitant deterioration of the speedup. Power3 (Power4) scalability is fair up to 9 (16) processors, but degrades severely on 16 (25) processors. The reason is the fairly large number of synchronizations per time step that are costly on (almost) fully saturated nodes. Experiments with a Power3 two-node computation involving 25 processors show a

6

	CG								
	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	54	68.0	0.058	111	65.6	0.013	470	198.6	96.9
2	55	71.9	0.039	111	69.9	0.014	258	147.0	96.0
4	54	73.0	0.027	114	71.8	0.015	253	147.9	96.5
8	55	79.7	0.031	151	77.0	0.020	131	117.1	95.0
16	48	82.5	0.029	177	78.6	0.025	—	—	—
32	—	—	—	149	85.2	0.020	—	—	—

	MG								
	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	207	97.4	0.067	407	87.3	0.029	2207	160.4	97.2
2	213	97.5	0.067	542	87.5	0.037	2053	160.1	97.1
4	193	97.3	0.061	470	85.6	0.033	1660	161.7	97.1
8	185	97.4	0.049	425	90.0	0.028	620	104.7	95.2
16	148	97.3	0.045	337	87.8	0.023	—	—	—
32	—	—	—	292	86.1	0.016	—	—	—

	FT								
	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	133	91.1	0.204	421	52.6	0.086	2021	256.0	98.4
2	120	91.2	0.088	397	57.5	0.022	1346	255.7	98.4
4	117	91.6	0.087	446	56.3	0.024	1324	255.2	98.4
8	112	91.6	0.084	379	57.1	0.022	1242	254.0	98.4
16	95	91.3	0.070	314	58.4	0.020	—	—	—
32	—	—	—	259	60.7	0.016	—	—	—

	BT								
	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	144	96.8	0.039	368	86.9	0.023	3693	100.9	99.2
4	127	97.1	0.116	208	85.6	0.018	2395	51.2	98.7
9	122	97.0	0.032	269	87.5	0.017	—	—	—
16	103	97.3	0.025	282	87.5	0.011	—	—	—
25	—	—	—-	208	98.4	0.013	—	—	—

	LU								
	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	186	96.6	0.304	422	75.2	0.087	740	100.2	77.7
2	247	97.1	0.293	595	76.4	0.020	656	51.8	77.2
4	257	97.1	0.421	636	77.5	0.092	684	53.0	77.3
8	263	97.0	0.235	636	78.9	0.009	142	29.4	74.5
16	267	96.9	0.173	558	79.6	0.007	—	—	—
32	—	—	—	566	78.4	0.006	—	—	—

Table 5: Per-processor performance of the NAS Parallel Benchmarks Class B.

remarkable recovery of the speedup.

LU fared poorly as expected on the SX-6, because data dependencies in the main part of the solver prevented full vectorization, as evidenced by the low VOR. Performance of the parts that do vectorize degrades significantly as the number of processors increases, because of the pencil domain decomposition in the first and second array dimensions. These factors do not play a role on the Power3/4, whose performance actually improves as the number of processors grows. This is because the communication overhead is rather small, so that the improved cache usage dominates scalability. Note that LU sustains the highest performance of all NPB on the Power architectures, but it also has the highest rate of TLB misses on the Power3. This suggests that the cost of a TLB miss is relatively small.

In sum, all NPB except LU suffer significant performance degradation on both architectures when a node is (nearly) fully saturated. AVL and especially VOR are strongly correlated with performance on the SX-6, but the occurrence of irregular stride (requiring support of gather/scatter units) or many small messages also have significant influence. Except for CG with its irregular memory references, there is a strong correlation between the smaller L1 cache on the Power4 (0.25 times that of the Power3) and the number of L1 misses. Nevertheless, L1 hits and TLB misses alone are weak predictors of performance on the Power3/4, whereas communication volume and frequency play a significantly larger role than on the SX-6. In a subsequent study, more detailed performance indicators will be examined that can more fully explain the observed behavior.

5 Scientific Applications

Six applications from diverse areas in scientific computing were chosen to measure and compare the performance of the SX-6 with that of the Power3 and Power4. The applications are: TLBE, a fusion energy application that performs simulations of high-temperature plasma; Cactus, an astrophysics code that solves Einstein's equations; OVERFLOW-D, a CFD production code that solves the Navier-Stokes equations around complex aerospace configurations; PARATEC, a materials science code that solves Kohn-Sham equations to obtain electron wavefunctions; GTC, a particle-in-cell approach to solve the gyrokinetic Vlasov-Poisson equations; and Mindy, a simplified molecular dynamics code that uses the Particle Mesh Ewald algorithm. The compiler options for running these six applications are listed in Table 6. Performance results are reported in Mflops/s per processor, except where the original algorithm has been modified for the SX-6 (these are reported as wall-clock time). As was the case for the NPB, AVL and VOR values are shown for the SX-6, and TLB and L1 values for the Power systems. All performance numbers were obtained with `hpmcount` on the Power3/4 and `ftrace` on the SX-6.

Application	Power3 and Power4	SX-6
TLBE	`-O3 -qtune=auto -qarch=auto`	`-C vopt`
Cactus-ADM	`-O3 -qstrict -qtune=auto -qarch=auto`	`-C vopt`
OVERFLOW-D	`-O3 -q64 -qfixed -qnosave`	`-C vsafe -f0 -size_t64`
PARATEC	`-O3 -qarch=auto`	`-C vopt`
GTC	`-O3 -qstrict -qtune=auto -qarch=auto -qcache=auto -Q`	`-f4 -C vopt -Wf,"-pvctl loopcnt=10000000 vwork=stack"`
Mindy	`-O3 -qstrict -qtune=auto -qarch=auto`	`-f4 -C vopt`

Table 6: Compiler options for running our application suite on the three target architectures.

6 Plasma Fusion: TLBE

Lattice Boltzmann methods provide a mesoscopic description of the transport properties of physical systems using a linearized Boltzmann equation. They offer an efficient way to model turbulence and collisions in a fluid. The TLBE application [22] performs a 2D simulation of high-temperature plasma using a hexagonal lattice and the BGK collision operator. Figure 3 shows an example of vorticity contours in the 2D decay of shear turbulence simulated by TLBE.
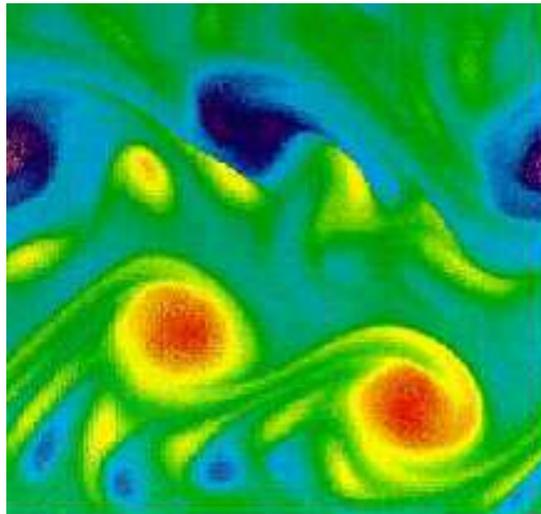

Figure 3: TLBE simulated vorticity contours in the 2D decay of shear turbulence.

6.1 Methodology

The TLBE simulation has three computationally demanding components: computation of the mean macroscopic variables (integration); relaxation of the macroscopic variables after colliding (collision); and propagation of the macroscopic variables to neighboring grid points (stream). The problem is ideally suited for vector architectures: the first two steps sweep through a regular 2D grid in column-major order performing a set of floating-point operations dependent only on local information; the third step consists of a set of strided copy operations. In addition, distributing the grid via a 2D decomposition easily parallelizes the method. The first two steps require no communication, while the third has a regular, static communication pattern in which the boundary values of the macroscopic variables are exchanged.

6.2 Porting Details

After initial profiling on the SX-6 using basic vectorization compiler options (`-C vopt`), only 280 Mflops/s (3.5% of peak) was achieved for a small 64^2 grid using a serial version of the code. Considering the major routine is computationally intensive and has no recurrence, this was a poor result. The `ftrace` tool showed that VOR was high (95%) and that the collision step dominated the execution time (96% of total); however, AVL was only about 6. We found that the inner loop over the number of directions in the hexagonal lattice had been vectorized, but not a loop over one of the grid dimensions. Invoking the most aggressive compiler flag (`-C hopt`) did not help. Therefore, we rewrote the collision routine by creating temporary vectors, and inverted the order of two loops to ensure vectorization over one dimension of the grid. As a result, serial performance improved by a factor of 7, and the parallel TLBE version was created by inserting the new collision routine into the MPI version of the code.

6.3 Performance Results

Parallel TLBE performance using a production grid of 2048^2 is presented in Table 7. The SX-6 results show that TLBE achieves almost perfect vectorization in terms of AVL and VOR. The 2- and 4-processor runs show similar performance as the serial version; however, an appreciable degradation due to memory-bank conflicts is observed when running eight MPI tasks. These conflicts do not appear for a single-processor run when the arrays are sized in the same manner as a single task of the 8-way run; hence they are due to interactions among all eight tasks executing on a single node.

	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	70	90.5	0.500	250	58.2	0.069	4060	256.0	99.5
2	110	91.7	0.770	300	69.2	0.014	4060	256.0	99.5
4	110	91.7	0.750	310	71.7	0.013	3920	256.0	99.5
8	110	92.4	0.770	470	87.1	0.021	3050	255.0	99.2
16	110	92.6	0.730	460	88.7	0.019	—	—	—
32	—	—	—	440	89.3	0.076	—	—	—

Table 7: Per-processor performance of TLBE on a 2048^2 grid.

For both the Power3 and Power4 architectures, the collision routine rewritten for the SX-6 performed slightly better than the original. On the cache-based machines, the parallel TLBE showed higher Mflops/s (per CPU) compared with the serial version. This is due to the use of smaller grids per processor in the parallel case, resulting in improved cache reuse. The more complex behavior on the Power4 is due to the competitive effects of the three-level cache structure and saturation of the SMP memory bandwidth. In summary, using all eight CPUs on the SX-6 gives an aggregate performance of 24.4 Gflops/s (38% of peak), and a speedup factor of 27.7 (6.5) over the Power3 (Power4), with minimal porting overhead.

7 Astrophysics: Cactus

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 Albert Einstein Institute in Potsdam, Germany, developed the Cactus code [1, 10] 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 (see Figure 4) and the gravitational waves that radiate from that event.

7.1 Methodology

In order to numerically solve Einstein's equations, it is generally necessary to decompose the GR 4D formulation (three spatial and one temporal dimension) into a set of equations describing the evolution of a 3D slice of space-time in time — the so-called 3+1 split. One of the earliest such splits lead to the ADM formalism (used in Cactus), which consists of four constraint equations and 12 evolution equations. The evolution equations can be solved using a number of different numerical methods, including staggered leapfrog, Mc-Cormack, Lax-Wendroff, and iterative Crank-Nicholson schemes. The initial data for the evolution must be chosen to obey the constraint equations, which can subsequently be used to determine how far the numerical solution has drifted from a correct solution to Einstein's equations. Analytically, the evolution preserves the constraints; however, there is usually some numerical drift.
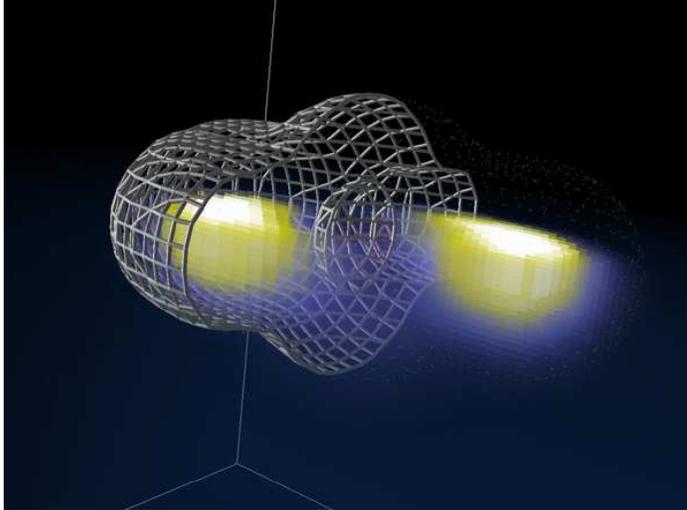
Figure 4: Visualization from a recent Cactus simulation of an in-spiraling merger of two black holes.

The freedom to choose coordinates in the space-time slice leads to a corresponding set of "gauge" functions in the ADM formalism: the lapse function, which describes the amount of coordinate time which two points on adjacent space-time slices differ by; and the shift vector, which describes the relationship between spatial coordinates in adjacent slices. These quantities can be chosen arbitrarily, and help prevent problems in the numerical evolution of the physical variables, e.g. by preserving some property of the space-time.

For performance evaluation, we focused on a core Cactus ADM solver, the Fortran77-based ADM kernel (BenchADM [9]), written when vector machines were more common; consequently, we expect it to vectorize well. BenchADM is computationally intensive, involving 600 flops per grid point. The loop body of the most numerically intensive part of the solver is large (several hundred lines of code). The enormous size of this loop results in moderate register pressure, as evidenced by the number of register spills in the Power3/4 assembly code. Normally, loop splitting is performed to alleviate this register pressure, but the expressions evaluated in the inner loop have such complex dependencies that any splitting arrangement requires storage of intermediate results. Numerous past attempts to split this loop have yielded no benefits since the increased number of load/stores tends to outweigh the advantages of reducing register spilling.

7.2 Porting Details

BenchADM vectorized almost entirely on the SX-6 in the first attempt. However, the vectorization appears to involve only the innermost of a triply nested loop (x, y, and z-directions for a 3D evolution). The resulting effective vector length for the code is directly related to the x-extent of the computational grid. This dependence led to some parallelization difficulties because the typical block-oriented domain decompositions reduce the vector length, thereby affecting uni-processor performance. In order to decouple parallel efficiency and uni-processor performance, the domain was decomposed using Z-slices.

7.3 Performance Results

Table 8 presents performance results for BenchADM on a 127^3 grid. The mild deterioration of performance on the Power systems as the number of processors grows up to 16 is due to the communication cost, with a steep drop in performance as a Power4 node gets fully saturated (32 processors). Increasing the grid size by just one point in each dimension to 128^3 results in severe performance degradation due to cache-line aliasing. This is a well-understood problem with set-associative caches that can be handled automatically

	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	274	99.4	0.030	672	92.2	0.010	3912	126.7	99.6
2	236	99.4	0.030	582	92.6	0.010	3500	126.7	99.5
4	249	99.4	0.020	619	93.2	0.010	2555	126.7	99.5
8	251	99.4	0.030	600	92.4	0.010	2088	126.7	99.3
16	226	99.5	0.020	538	93.0	0.010	—	—	—
32	—	—	—	379	97.0	0.001	—	—	—

Table 8: Per-processor performance of the Cactus BenchADM kernel on a 127^3 grid.

by the compiler using array padding; however, the compilers we were using could not infer a reasonable padding to prevent the aliasing.

While for smaller grid sizes the SX-6 performance is mediocre, the 127^3 grid uni-processor computation returns an impressive 3.9 GFlops/s with a sizable AVL and a VOR of almost 100%. The SX-6 is immune to the effects of power-of-two aliasing because of the absence of cache memory. The vector memory subsystem is not affected by bank conflicts because most accesses are still unit stride. To date, SX-6's 49% of peak performance is the best achieved for this benchmark on any current computer architecture. SX-6 multi-processor performance deteriorates fairly rapidly due to the rising cost of interprocessor synchronization; however, the AVL and VOR are hardly affected. Performance counters indicated that boundary exchange accounts for 12% of the execution time for a 2-processor run, but increases to 47% when running on eight processors. We determined empirically that the high cost of boundary exchange on the SX-6 is most likely due to latencies caused by the calling overhead and/or copying operations rather than any memory subsystem deficiencies. When we artificially doubled the size of the messages, the fraction of time spent in the boundary exchange increased only from 12% to 14% (for a 2-processor run); if the communication were predominantly bandwidth limited, the overhead would have nearly doubled.

8 Fluid Dynamics: OVERFLOW-D

OVERFLOW-D [23] is an overset grid methodology [12] for high-fidelity viscous Navier-Stokes CFD simulations around realistic aerospace configurations. The application can handle complex designs with multiple geometric components, where individual body-fitted grids are easily constructed about each component. OVERFLOW-D is designed to simplify the modeling of problems when components are in relative motion (dynamic grid systems). At each time step, the flow equations are solved independently on each grid ("block") in a sequential manner. Boundary values in grid overlap regions are updated before each time step, using a Chimera interpolation procedure. The code uses finite differences in space, and a variety of implicit/explicit time stepping.

8.1 Methodology

The MPI version of OVERFLOW-D (in F90) is based on the multi-block feature of the sequential code, which offers a natural coarse-grain parallelism. The sequential code consists of an outer "time-loop" and an inner "grid-loop". The inter-grid boundary updates in the serial version are performed successively. To facilitate parallel execution, grids are clustered into groups; one MPI process is then assigned to each group. The clustering 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 contains two levels, a loop over groups ("group-loop") and a loop over the grids within each group. The group-loop is executed in parallel, with each group performing its own sequential grid-loop and inter-grid updates. The inter-grid boundary updates across the groups are achieved via MPI. Further details can be found in [13].

8.2   Porting Details

The MPI implementation of OVERFLOW-D is based on the sequential version, the organization of which was designed to exploit early Cray vector machines. The same basic program structure is used on all three target architectures except that the code was compiled with the `-C vsafe` option on the SX-6. A few minor changes were made in some subroutines in an effort to meet specific compiler requirements.

8.3   Performance Results

Our experiments involve a Navier-Stokes simulation of vortex dynamics in the complex wake flow region around hovering rotors. The grid system consisted of 41 blocks and approximately 8 million grid points. Figure 5 presents a sectional view of the test grid and the vorticity magnitude contours of the final solution.
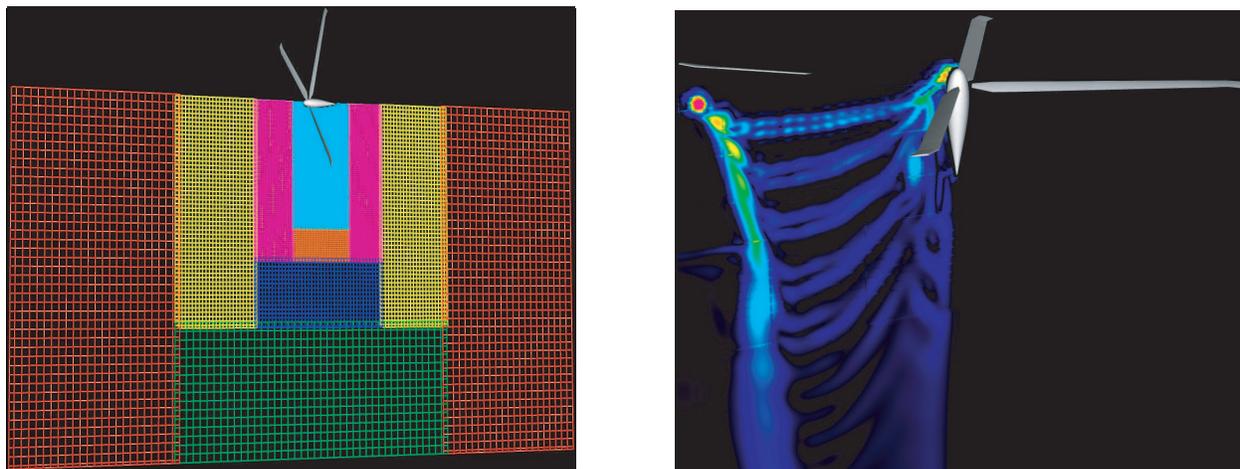

Figure 5: Sectional views of the OVERFLOW-D test grid system and the computed vorticity magnitude contours.

Table 9 shows execution times per time step (averaged over 10 steps) on the Power3, Power4, and SX-6. The current MPI implementation of OVERFLOW-D does not allow uni-processor runs. Results demonstrate that the SX-6 outperforms the cache-based machines; in fact, the run time for eight processors on the SX-6 is less than three-fourths the 32-processor Power4 number. Scalability is similar for both the Power4 and SX-6 architectures, with computational efficiency decreasing for a larger number of MPI tasks primarily due to load imbalance. It is interesting to note that Power3 scalability exceeds that of the Power4. On the SX-6, the relatively small AVL and limited VOR explain why the code achieves a maximum of only 7.8 Gflops/s on eight processors. Reorganizing OVERFLOW-D would achieve higher vector performance; however, extensive effort would be required to modify this production code of more than 100K lines.

P	Power3			Power4			SX-6		
	sec	L1	TLB	sec	L1	TLB	sec	AVL	VOR
2	46.7	93.3	0.245	17.1	84.4	0.014	5.5	87.2	80.2
4	26.6	95.4	0.233	9.4	87.5	0.010	2.8	84.3	76.0
8	13.2	96.6	0.187	5.6	90.4	0.008	1.6	79.0	69.1
16	8.0	98.2	0.143	3.2	92.2	0.005	—	—	—
32	—	—	—	2.2	93.4	0.003	—	—	—

Table 9: Per-processor performance of OVERFLOW-D on a 8 million-grid point problem.

9   Materials Science: PARATEC

PARATEC (PARAllel Total Energy Code) [6] performs first-principles quantum mechanical total energy calculations using pseudopotentials and a plane wave basis set. The approach is based on Density Functional Theory (DFT) that has become the standard technique in materials science to calculate accurately the structural and electronic properties of new materials with a full quantum mechanical treatment of the electrons. Codes performing DFT calculations are among the largest consumers of computer cycles in centers around the world, with the plane-wave pseudopotential approach being the most commonly used. Both experimental and theory groups use these types of codes to study properties such as strength, cohesion, growth, catalysis, magnetic, optical, and transport for materials like nanostructures, complex surfaces, doped semiconductors, and others. Figure 6 shows the induced current and charge density in crystalized glycine, calculated using PARATEC. These simulations were used to better understand nuclear magnetic resonance experiments [25].
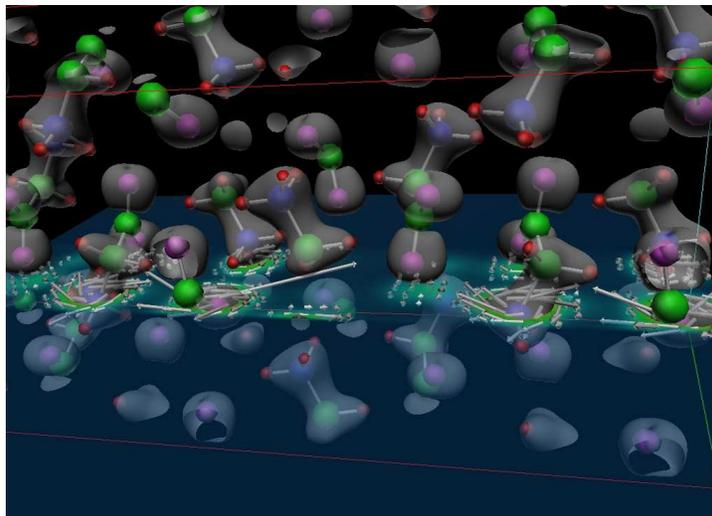

Figure 6: Visualization of induced current (white arrows) and charge density (colored plane and grey surface) in crystalized glycine, calculated using PARATEC [25].

9.1   Methodology

PARATEC uses an all-band conjugate gradient (CG) approach to solve the Kohn-Sham equations of DFT to obtain the wavefunctions of the electrons. A part of the calculations 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 (over 80% for a large system) in vendor supplied BLAS3 and 1D FFTs 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. For a review of this approach with applications, see [16, 19].

9.2 Porting Details

PARATEC, an MPI code designed primarily for massively parallel systems, also runs on serial machines. Since much of the computation involves vendor supplied 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 SX-6, the standard FFTs (e.g., ZFFT) run at a low percentage of peak. It is thus necessary to use the simultaneous 1D FFTs (e.g., ZFFTS) to obtain good vectorization. A small amount of code rewriting was required to convert the 3D FFT routines to simultaneous ("multiple") 1D FFT calls.

9.3 Performance Results

The results in Table 10 show scaling tests of a 250 Si-atom bulk system for a standard LDA run of PARATEC with a 25 Ry cut-off using norm-conserving pseudopotentials. The simulations are for three CG steps of the iterative eigensolver, and include the set-up and I/O steps necessary to execute the code. A typical calculation using the code would require 20 to 60 CG steps to converge the charge density.

	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	915	98.3	0.166	2290	95.6	0.106	5090	113.0	98.0
2	915	98.3	0.168	2250	95.5	0.104	4980	112.0	98.0
4	920	98.3	0.173	2210	96.6	0.079	4700	112.0	98.0
8	911	98.3	0.180	2085	95.9	0.024	4220	112.0	98.0
16	840	98.4	0.182	1572	96.1	0.090	—	—	—
32	—	—	—	1327	96.7	0.064	—	—	—

Table 10: Per-processor performance of PARATEC on a 250 Si-atom bulk system.

Results show that PARATEC vectorizes well and achieves 64% of peak on one processor of the SX-6. The AVL is approximately half the vector register length, but with a high fraction of VOR. This is because most of the time is spent in 3D FFTs and BLAS3. The loss in scalability to eight processors (53% of peak) are due primarily to memory contention and initial code set-up (including I/O) that do not scale well. Performance increases with larger problem sizes and more CG steps: for example, running 432 Si-atom systems for 20 CG steps achieved 73% of peak on one processor.

PARATEC runs efficiently on the Power3; the FFT and BLAS3 routines are highly optimized for this architecture. The code ran at 61% of peak on a single processor and at 56% on 16 processors. Larger physical systems, such as the one with 432 Si-atoms, ran at 1.02 Gflops/s (68% of peak) on 16 processors. On the Power4, PARATEC sustains a much lower fraction of peak (44% on one processor) due to its relatively poor ratio of memory bandwidth to peak performance. Nonetheless, the Power4 32-processor SMP node achieves high total performance, exceeding that of the 8-processor SX-6 node. The L1 hit rate is primarily determined by the serial FFT and BLAS3 libraries; hence it does not vary much with processor count. We conclude that, due to the high computational intensity and use of optimized numerical libraries, these types of codes execute efficiently on both scalar and vector machines, without the need for significant code restructuring.

10 Magnetic Fusion: GTC

The goal of magnetic fusion is the construction and operation of a burning plasma power plant producing clean energy. The performance of such a device is determined by the rate at which the energy is transported out of the hot core to the colder edge of the plasma. The Gyrokinetic Toroidal Code (GTC) [18] was developed to study the dominant mechanism for this transport of thermal energy, namely plasma microturbulence. Plasma turbulence is best simulated by particle codes, in which all the nonlinearities are naturally included. Figure 7 presents a visualization of electrostatic potential fluctuations in a global nonlinear gyrokinetic simulation of microturbulence in magnetically confined plasmas.
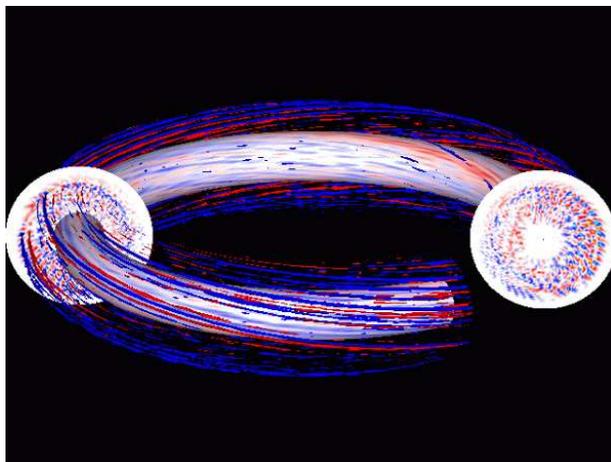
Figure 7: Electrostatic potential fluctuations of microturbulence in magnetically confined plasmas using GTC.

10.1 Methodology

GTC solves the gyroaveraged Vlasov-Poisson (gyrokinetic) system of equations [17] using the particle-in-cell (PIC) approach. Instead of interacting with each other, the simulated particles interact with a self-consistent electrostatic or electromagnetic field described on a grid. Numerically, the PIC method scales as N, instead of N^2 as in the case of direct binary interactions. Also, the equations of motion for the particles are simple ODEs (rather than nonlinear PDEs), and can be solved easily (e.g. using Runge-Kutta). The main tasks at each time step are: deposit the charge of each particle at the nearest grid points (scatter); solve the Poisson equation to get the potential at each grid point; calculate the force acting on each particle from the potential at the nearest grid points (gather); move the particles by solving the equations of motion; find the particles that have moved outside their local domain and migrate them accordingly.

The parallel version of GTC performs well on massive superscalar systems, since the Poisson equation is solved as a local operation. The key performance bottleneck is the scatter operation, a loop over the array containing the position of each particle. Based on a particle's position, we find the nearest grid points surrounding it and assign each of them a fraction of its charge proportional to the separation distance. These charge fractions are then accumulated in another array. The scatter algorithm in GTC is complicated by the fact that these are fast gyrating particles, where motion is described by charged rings being tracked by their guiding center (the center of the circular motion).

10.2 Porting Details

GTC's scatter phase presented some challenges when porting the code to the SX-6 architecture. It is difficult to implement efficiently due to its non-contiguous writes to memory. The particle array is accessed sequen-

tially, but its entries correspond to random locations in the simulation space. As a result, the grid array accumulating the charges is accessed in random fashion, resulting in poor cache performance. This problem is exacerbated on vector architectures, since many particles deposit charges at the same grid point, causing a classic memory dependence problem and preventing vectorization. We avoid these memory conflicts by using temporary arrays of vector length (256 words) to accumulate the charges. Once the loop is completed, the information in the temporary array is merged with the real charge data; however, this increases memory traffic and reduces the flops/byte ratio.

Another source of performance degradation was a short inner loop located inside two large particle loops that the SX-6 compiler could not vectorize. This problem was solved by inserting a vectorization directive, fusing the inner and outer loops. Finally, I/O within the main loop had to be removed to allow vectorization.

10.3   Performance Results

Table 11 shows GTC performance results for a simulation comprising of 4 million particles and 1,187,392 grid points over 200 time steps. The geometry is a torus described by the configuration of the magnetic field. On a single processor, the Power3 achieves 10% of peak, while the Power4 performance represents only 5% of its peak. The SX-6 single-processor experiment runs at 701 Mflops/s, or only 9% of its theoretical peak. This poor SX-6 performance is unexpected, considering the relatively high AVL and VOR values. We believe this is because the scalar units need to compute the indices for the scatter/gather of the underlying unstructured grid. However, in terms of raw performance, the SX-6 still outperforms the Power3/4 by factors of 4.6 and 2.5, respectively.

	Power3			Power4			SX-6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	153	95.1	0.130	277	89.4	0.015	701	186.8	98.0
2	155	95.1	0.102	294	89.8	0.009	653	184.8	98.0
4	163	96.0	0.084	310	91.2	0.007	548	181.5	97.9
8	167	96.6	0.052	326	92.2	0.007	391	175.4	97.7
16	155	97.3	0.025	240	92.8	0.006	—	—	—
32	—	—	—	275	92.7	0.006	—	—	—

Table 11: Per-processor performance of GTC on a 4-million particle simulation.

Parallel results demonstrate that scaling on the SX-6 is not nearly as good as on the Power3/4. In fact, both the Power3 and Power4 initially (through $P = 8$) show superlinear speedup, a common characteristic of cache-based machines. This is explained by the higher L1 hit rates and lower TLB misses with increasing processor count. Superlinear scaling for a fixed problem size cannot be maintained past a certain number of processors since all the data ultimately fits in cache while the communication-to-computation ratio continues to increase. Limited scaling on the SX-6 is probably due to the 1D decomposition which reduces the length of the biggest vector loops as the number of processors increases; however, this is not the final word. More work is being done on GTC to improve its scalability and efficiency on vector architectures.

11   Molecular Dynamics: Mindy

Mindy is a simplified serial molecular dynamics (MD) C++ code, derived from the parallel MD program called NAMD [3]. The energetics, time integration, and file formats are identical to those used by NAMD.

11.1 Methodology

Mindy's core is the calculation of forces between N atoms via the particle mesh Ewald algorithm. Its $O(N^2)$ complexity is reduced to $O(N \log N)$ by dividing the problem into boxes, and then computing electrostatic interaction in aggregate by considering neighboring boxes. Neighbor lists and a variety of cutoffs are used to decrease the required number of force computations.

11.2 Porting Details

Modern MD codes such as Mindy present special challenges for vectorization, since many optimization and scaling methodologies are at odds with the flow of data suitable for vector architectures. The reduction of floating point work from N^2 to $N \log N$ is accomplished at the cost of increased branch complexity and nonuniform data access. These techniques have a deleterious effect on vectorization; two strategies were therefore adopted to optimize Mindy on the SX-6. The first severely decreased the number of conditions and exclusions in the inner loops, resulting in more computation overall, but less inner-loop branching. We refer to this strategy as NO_EXCL.

The second approach was to divide the electrostatic computation into two steps. First, the neighbor lists and distances are checked for exclusions, and a temporary list of inter-atom forces to be computed is generated. The force computations are then performed on this list in a vectorizable loop. Extra memory is required for the temporaries and, as a result, the flops/byte ratio is reduced. This scheme is labeled BUILD_TEMP.

Mindy uses C++ objects extensively, hindering the compiler to identify data-parallel code segments. Aggregate datatypes call member functions in the force computation, which impede vectorization. Inlining member functions of C++ classes is harder for a compiler than inlining C functions given the possibility of inheritance, runtime templates, and virtual functions. Compiler directives were used to specify that certain code sections contain no dependencies, allowing partial vectorization of those regions.

11.3 Performance Results

The case studied here is the apolipoprotein A-I molecule (see Figure 8, a 92224-atom system important in cardiac blood chemistry that has been adopted as a benchmark for large-scale MD simulations on biological systems.
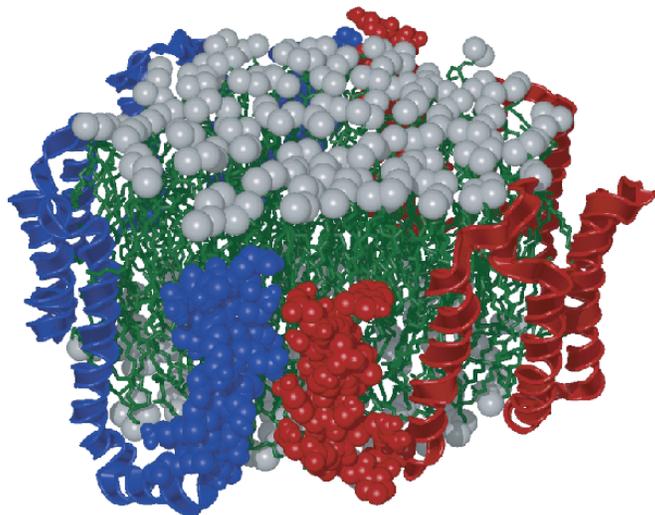

Figure 8: The apolipoprotein A-I molecule, a 92224-atom system simulated by Mindy.

Table 12 presents performance results of the serial Mindy algorithm. Neither of the two SX-6 optimization strategies achieves high performance. The NO_EXCL approach results in a very small VOR, meaning that almost all the computations are performed on the scalar unit. The BUILD_TEMP strategy (also used on the Power3/4) increases VOR, but incurs the overhead of increased memory traffic for storing temporary arrays. In general, this class of applications is at odds with vectorization due to the irregularly structured nature of the codes. The SX-6 achieves only 165 Mflops/s, or 2% of peak, slightly outperforming the Power3 and trailing the Power4 by about a factor of two in run time. Effectively utilizing the SX-6 would likely require extensive reengineering of both the algorithm and the object-oriented code.

Power3			Power4			SX-6: NO_EXCL			SX-6: BUILD_TEMP		
sec	L1	TLB	sec	L1	TLB	sec	AVL	VOR	sec	AVL	VOR
15.7	99.8	0.010	7.8	98.8	0.001	19.7	78.0	0.03	16.1	134.0	34.8

Table 12: Serial performance of Mindy on a 92224-atom system with two different SX-6 optimization approaches.

12   Summary and Conclusions

This paper presented the performance of the NEC SX-6 vector processor and compared it against the cache-based IBM Power3 and Power4 superscalar architectures, across a wide range of scientific computations. Experiments with a set of microbenchmarks demonstrated that for low-level program characteristics, the specialized SX-6 vector hardware significantly outperforms the commodity-based superscalar designs of the Power3/4.

Next we examined the NAS Parallel Benchmarks, a well-understood set of kernels representing key areas in scientific computations. These compact codes allowed us to perform the three main variations of vectorization tuning: compiler flags, compiler directives, and actual code modifications. The resulting optimized codes enabled us to identify classes of applications both at odds with and well suited for vector architectures, with performance ranging from 5.9% to 46% of peak on a single SX-6 processor, and from 1.6% to 16% on a fully saturated node of eight processors. Similar percentages of peak performance were achieved on eight processors of the Power3 and Power4, although the top performing codes on vector and cache systems were not the same. Absence of data dependencies in the main loops and long vector lengths in FT produced the best results on the SX-6, whereas good locality and small communication overhead made LU the best performing code on the Power systems.

Several applications from key scientific computing domains were also evaluated; however, extensive vector optimizations have not been performed at this time. Since most modern scientific codes are designed for (super)scalar systems, we simply examined the effort required to port these applications to the vector architecture. Table 13 summarizes the overall performance, sorted by SX-6 speedup against the Power4. Results show that the SX-6 achieves high sustained performance (relative to theoretical peak) for a large fraction of our application suite and, in many cases, significantly outperforms the scalar architectures.

The Cactus-ADM kernel vectorized almost entirely on the SX-6 in the first attempt. The rest of our applications required the insertion of compiler directives and/or minor code modifications to improve the two critical components of effective vectorization: long vector length and high vector operation ratio. Vector optimization strategies included loop fusion (and loop reordering) to improve vector length; introduction of temporary variables to break loop dependencies (both real and compiler imagined); reduction of conditional branches; and alternative algorithmic approaches. For applications such as TLBE, minor code changes were sufficient to achieve good vector performance and a high percentage of theoretical peak, especially for the multi-processor computations. For OVERFLOW-D, we obtained fair performance on both the cache-based and vector machines using the same basic code structure. PARATEC represented a class of applications

Application	Scientific	Lines	Power3	Power4	SX-6		SX-6 Speedup vs.	
Name	Discipline	of Code	% Pk	% Pk	% Pk	P	Power3	Power4
TLBE	Plasma Fusion	1,500	7.3	9.0	38.1	8	27.8	6.5
Cactus-ADM	Astrophysics	1,200	16.8	11.5	26.1	8	8.3	3.5
OVERFLOW-D	Fluid Dynamics	100,000	7.8	5.3	12.2	8	8.2	3.5
PARATEC	Materials Science	50,000	60.7	40.1	52.8	8	4.6	2.0
GTC	Magnetic Fusion	5,000	11.1	6.3	4.9	8	2.3	1.2
Mindy	Molecular Dynamics	11,900	6.3	4.7	2.1	1	1.0	0.5

Table 13: Summary overview of application suite performance.

relying heavily on highly optimized BLAS3 libraries. For these types of codes, all three architectures performed very well due to the regularly structured, computationally intensive nature of the algorithm. On a single SX-6 processor, PARATEC achieved 64% of peak, while TLBE and Cactus-ADM were at 50%; however, TLBE showed a factor of 58.0 (16.2) performance improvement over the Power3 (Power4).

Finally, we presented two applications with poor vector performance: GTC and Mindy. They feature indirect addressing, many conditional branches, and loop carried data-dependencies. These particle-in-cell and particle mesh Ewald algorithms are representative of sparse computational methods at odds with vector parallelism, since multiple particle forces may potentially contribute to the energy calculation at a particular grid point. Effectively vectorizing these applications requires the removal of potential data dependencies through techniques such as copied temporary arrays, particle sorting, or graph coloring. However, these code transformations may increase the vector efficiency at the cost of significant computational (or memory) overheads, thus limiting overall efficacy. For Mindy, an additional problem was the use of C++ objects, which made it difficult for the compiler to identify data-parallel loops. Effectively utilizing the SX-6 would likely require extensive reengineering of both the algorithm and the implementation for these applications.

Acknowledgements

References

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

[2] Earth Simulator Center. http://www.es.jamstec.go.jp.

[3] Mindy: A 'minimal' molecular dynamics program.
    http://www.ks.uiuc.edu/Development/MDTools/mindy.

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

[5] Pallas MPI Benchmarks. http://www.pallas.com/e/products/pmb.

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

[7] STREAM: Sustainable memory bandwidth in high performance computers. http://www.cs.virginia.edu/stream.

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

[9] A. Abrahams, D. Bernstein, D. Hobill, E. Seidel, and L. Smarr. Numerically generated black hole spacetimes: interaction with gravitational waves. *Phys. Rev. D*, 45:3544–3558, 1992.

[10] G. Allen, T. Goodale, G. Lanfermann, T. Radke, E. Seidel, W. Benger, H.-C. Hege, A. Merzky, J. Massó, and J. Shalf. Solving Einstein's equations on supercomputers. *IEEE Computer*, 32(12):52–58, 1999.

[11] D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski, R. Schreiber, H. Simon, V. Venkatakrishnan, and S. Weeratunga. The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, 1994.

[12] 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.

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

[14] T.H. Dunigan, Jr., M.R. Fahey, J.B. White III, and P.H. Worley. Early evaluation of the Cray X1. In *Proc. SC2003*, CD-ROM, 2003.

[15] R. Espasa, M. Valero, and J.E. Smith. Vector architectures: past, present and future. In *Proc. Intl. Conf. on Supercomputing*, pages 425–432, 1998.

[16] G. Galli and A. Pasquarello. *First-Principles Molecular Dynamics*, pages 261–313. Computer Simulation in Chemical Physics. Kluwer, 1993.

[17] W.W. Lee. Gyrokinetic particle simulation model. *J. Comp. Phys.*, 72:243–262, 1987.

[18] Z. Lin, S. Ethier, T.S. Hahm, and W.M. Tang. Size scaling of turbulent transport in magnetically confined plasmas. *Phys. Rev. Lett.*, 88:195004, 2002.

[19] M.C. Payne, M.P. Teter, D.C. Allan, T.A. Arias, and J.D. Joannopoulos. Iterative minimization techniques for ab initio total-energy calculations: Molecular dynamics and conjugate gradients. *Rev. Mod. Phys.*, 64:1045–1098, 1993.

[20] H. Sakagami, H. Murai, Y. Seo, and M. Yokokawa. 14.9 TFLOPS three-dimensional fluid simulation for fusion science with HPF on the Earth Simulator. In *Proc. SC2002*, CD-ROM, 2002.

[21] S. Shingu, H. Takahara, H. Fuchigami, M. Yamada, Y. Tsuda, W. Ohfuchi, Y. Sasaki, K. Kobayashi, T. Hagiwara, S. Habata, M. Yokokawa, H. Itoh, and K. Otsuka. A 26.58 Tflops global atmospheric simulation with the spectral transform method on the Earth Simulator. In *Proc. SC2002*, CD-ROM, 2002.

[22] G. Vahala, J. Carter, D. Wah, L. Vahala, and P. Pavlo. *Parallelization and MPI performance of Thermal Lattice Boltzmann codes for fluid turbulence*. Parallel Computational Fluid Dynamics '99. Elsevier, 2000.

[23] A.M. Wissink and R. Meakin. Computational fluid dynamics with adaptive overset grids on parallel and distributed computer platforms. In *Proc. Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, pages 1628–1634, 1998.

[24] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, and Y. Kaneda. 16.4-Tflops Direct Numerical Simulation of turbulence by Fourier spectral method on the Earth Simulator. In *Proc. SC2002*, CD-ROM, 2002.

[25] Y. Yoon, B.G. Pfrommer, S.G. Louie, and A. Canning. NMR chemical shifts in amino acids: effects of environments, electric field and amine group rotation. *Phys. Rev. B*, submitted.