Evaluation of Cache-based Superscalar and Cacheless Vector Architectures for Scientific Computations

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

Stephane 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 has become a well-known problem in high performance computing. The recent development of parallel vector systems offers the potential to bridge this gap for a significant number of computational science codes and deliver a substantial increase in computing capabilities. This paper examines the intranode performance of the NEC SX6 vector processor and the cache-based IBM Power3/4 superscalar architectures across a number of key scientific computing areas. First, we present the performance of a microbenchmark suite that examines a full spectrum of low-level machine characteristics. Next, we study the behavior of the NAS Parallel Benchmarks using some simple optimizations. Finally, we evaluate the performance of several numerical codes from key scientific computing domains. Overall results demonstrate that the SX6 achieves high performance on a large fraction of our application suite and in many cases significantly outperforms the RISC-based architectures. However, certain classes of applications are not easily amenable to vectorization and would likely require extensive reengineering of both algorithm and implementation to utilize the SX6 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. 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 bridge this performance gap for a significant number of scientific codes, and to increase computational power substantially. This was highlighted dramatically when the Japanese Earth Simulator System's [2] results were published [14, 15, 18]. The Earth Simulator, based on NEC SX6[1] *vector technology*, achieves five times the LINPACK performance with half the number of processors of the IBM SP-based ASCI White, the world's fourth-most powerful supercomputer, built using *superscalar technology* [7]. 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.

In this paper, we compare the performance of the NEC SX6 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 three of the well-known NAS Parallel Benchmarks (NPB) [4, 11], using problem size Class B. Finally, we present performance results for several numerical codes from scientific computing domains, including astrophysics, plasma fusion, fluid dynamics, 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.

*Employee of Computer Sciences Corporation.
[1]Also referred to as the Cray SX6 due to Cray's agreement to market NEC's SX line.

For non-vectorizable instructions, the SX6 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 SX6 vector unit is significantly more powerful than the 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 SX6 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 SX6 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 multiprocessor systems; for instance, the Earth Simulator contains 640 SX6 nodes, connected through a single-stage crossbar.

The vector results in this paper were obtained on the single-node (8-way) SX6 system (named Rime) running SUPER-UX 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 [6]); and point-to-point communication, network/memory contention, and barrier synchronizations (using 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 unit-stride memory bandwidth behavior of the triad summation: $a(i) = b(i) + s \times c(i)$, using the STREAM benchmark [6]. It effectively captures the peak bandwidth of the architectures, and shows that the SX6 achieves about 14 and 48 times the performance of the Power3 and Power4, respectively, on a single processor. Notice also that the SX6 shows negligible bandwidth degradation for up to eight tasks, while the Power3/4 drop by almost 50% for fully packed nodes.

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 SX6 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 SX6 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 Power3/4, a precipitous drop in data transfer rate occurs for small strides, due to loss of cache reuse. This drop is more complex on the Power4, because of its more complicated cache structure.

Finally, Figure 2 presents the memory bandwidth of indirect addressing through vector triad gather and scatter operations of various data sizes. For smaller sizes, the cache-based architectures show better data rates for indirect access

P	Power3	Power4	SX6
1	661	2292	31900
2	661	2264	31830
4	644	2151	31875
8	568	1946	31467
16	381	1552	—
32	—	1040	—

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

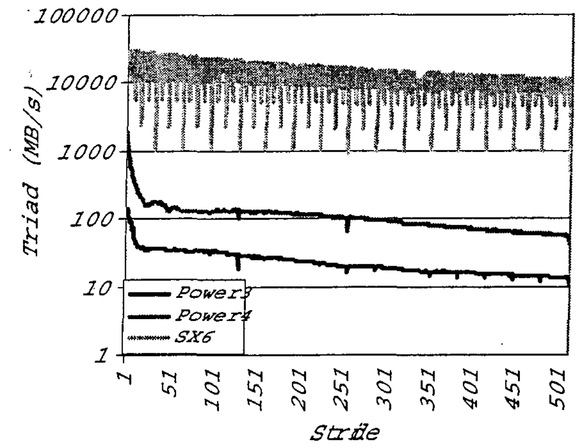

Figure 1: Single-processor STREAM triad performance (in MB/s) using regularly strided data.

P	Power3	Power4	SX6
2	17.1	6.7	5.0
4	31.7	12.1	7.1
8	54.4	19.8	22.0
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, and although their performance was meant to be good across a whole range of systems, they were written with cache-based systems in mind. Here we investigate the work involved in producing good NPB vector code. Of the eight published NPB, we select the three most appropriate for our current study: CG, a sparse-matrix conjugate-gradient algorithm marked by irregular stride resulting from indirect addressing; 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. In Table 5 we present MPI performance results for these codes on the SX6 and Power3 for medium problem sizes, commonly referred to as Class B. Performance results are reported in Aggregate Mflops/s (AM). 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).

	Power3									SX6								
	CG			FT			BT			CG			FT			BT		
P	AM	L1	TLB	AM	L1	TLB	AM	L1	TLB	AM	AVL	VOR	AM	AVL	VOR	AM	AVL	VOR
1	53.76	68.0	.058	133.4	91.1	.204	144.3	96.9	.039	469.5	198.6	96.9	2021.	256.0	98.4	3693.	100.9	99.2
2	110.9	71.9	.040	239.2	91.2	.088	—	—	—	517.0	147.0	96.0	2691.	255.7	98.4	—	—	—
4	215.9	73.4	.027	467.9	91.6	.087	509.8	97.2	.031	1013.	147.9	96.5	5295.	255.2	98.4	9581.	51.2	98.7
8	438.3	80.7	.032	897.6	91.6	.084	—	—	—	1045.	117.1	95.0	9933.	254.0	98.4	—	—	—
9	—	—	—	—	—	—	1097.	97.4	.032									
16	765.3	85.6	.030	1519.	91.6	.070	1646.	97.4	.025									

Table 5: NAS Parallel Benchmarks Class B performance results. AVL is Average Vector Length, VOR is Vector Operation Ratio, and AM is Aggregate MFlops/s

Although the CG code vectorizes fully and exhibits fairly long vector lengths, uni-processor SX6 performance is not very good due to many bank conflicts resulting from the indirect addressing. Multi-processor SX6 speedup degrades as expected with the reduction in vector length. Power3 scalability is very good, mostly because uni-processor performance is so poor due to the serious lack of data locality. FT did not perform well on the SX6 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), SX6 uni-processor performance improved markedly due to increased vector length. Speedup from one to two processors was 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 was not called in the uni-processor run), but subsequent scalability was excellent. Power3 scalability was fairly good overall, despite the large communication volume, due to improved data locality of the multi-processor implementation. The BT baseline MPI code performed poorly on the SX6, 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 leads to long vector lengths throughout the single-processor code, and good performance. Increasing the number of processors on the SX6 causes reduction of vector length (artifact of the three-dimensional domain decomposition) and a concomitant deterioration of the speedup. Power3 scalability is fair up to 9 processors, but degrades severely on 16 processors. The reason is the fairly large number of synchronizations per time step that are costly on a fully saturated 16-processor Power3 node. Experiments with a two-node computation involving 25 processors show a remarkable recovery of the speedup.

are hardly affected by the parallelization, and artificially changing the volume of communications has negligible effect on performance.

	Power3			Power4			SX6		
P	Mflops/s	L1	TLB	Mflops/s	L1	TLB	Mflops/s	AVL	VOR
1	273.8	99.4	.030	672.1	92.2	.01	3912.	126.7	99.6
2	236.2	99.4	.030	582.4	92.6	.01	3500.	126.7	99.5
4	248.9	99.4	.020	618.8	93.2	.01	2555.	126.7	99.5
8	251.4	99.4	.030	599.6	92.4	.01	2088.	126.7	99.3
16	226.5	99.5	.020	537.6	93.0	.01			
32	—	—	—	379.4	97.0	.00			

Table 6: Performance of the Cactus BenchADM kernel on a 127^3 grid.

7 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 [16] performs a 2D simulation of high-temperature plasma using a hexagonal lattice and the BGK collision operator.

7.1 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 first two steps are floating-point intensive, the third consists of data movement only. The problem is ideally suited for vector architectures. The first two steps are completely vectorizable, since the computation for each grid point is purely local. 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.

7.2 Porting Details

After initial profiling on the SX6 using basic vectorization compiler options (`-C vopt`), a poor result of 280 Mflops/s was achieved for a small 64^2 grid using a serial version of the code. `ftrace` 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.

7.3 Performance Results

Parallel TLBE performance using a production grid of 2048^2 is presented in Table 7. The SX6 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 is observed when running 8 MPI tasks, which is most likely due to network/memory contention in the SMP.

For both the Power3 and Power4 architectures, the collision routine rewritten for the SX6 performed somewhat 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

AVL and limited VOR explain why the code achieves a maximum of only 1.9 Gflops/s on 8 processors. Reorganizing OVERFLOW-D would achieve higher vector performance; however, extensive effort would be required to modify this production code.

	Power3			Power4	SX6		
P	sec	L1	TLB	sec	sec	AVL	VOR
2	46.7	93.3	.245	15.8	5.5	87	80%
4	26.6	95.4	.233	8.5	2.8	84	76%
8	13.2	96.6	.197	4.3	1.6	79	69%
16	8.0	98.2	.143	3.7			
32	—	—	—	3.4			

Table 8: Performance of OVERFLOW-D on a 8 million-grid point problem

9 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) [13] 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.

9.1 Methodology

GTC solves the gyroaveraged Vlasov-Poisson (gyrokinetic) system of equations [12]) using the particle-in-cell 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. 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).

9.2 Porting Details

GTC's scatter phase presented some challenges when porting the code to the SX6 architecture. It is difficult to implement efficiently due to its non-contiguous writes to memory. The particle array is accessed sequentially, 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 flop/byte ratio.

Another source of performance degradation was a short inner loop located inside two large particle loops that the SX6 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 in order to allow vectorization.

the scalar unit. The BUILD_TEMP (also used on the Power3/4) approach 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 SX6 achieves only 165 Mflops/s, or 2% of peak, slightly outperforming the Power3 and trailing the Power4 by about a factor of two in runtime. Effectively utilizing the SX6 would likely require extensive reengineering of both the algorithm and the code.

Power3			Power4			SX6: NO_EXCL			SX6: BUILD_TEMP		
sec	L1	TLB	sec	L1	TLB	sec	AVL	VOR	sec	AVL	VOR
15.7	99.8	0.01	7.8	98.8	.001	19.7	78	0.03%	16.1	134	34.8%

Table 10: Serial performance of Mindy on a 92224-atom system with two different SX6 optimization approaches

11 Summary and Conclusions

This paper presented the performance of the NEC SX6 vector processor and compared it against the cache-based IBM Power3/4 superscalar architecture, across a range of scientific computations. Experiments with a set of microbenchmarks demonstrated that for low-level program characteristics, the specialized SX6 vector hardware significantly outperforms the commodity-based superscalar designs of the Power3 and Power4. Next we examined the NPBs, 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. Results enabled us to identify classes of applications both at odds with and well suited for vector architectures, with performance ranging from 5% to 46% of peak on a single SX6 processor.

Several applications from key scientific computing domains were also evaluated. Table 11 summarizes the relative performance results. Since most modern scientific codes are designed for (super)scalar systems, we examined the effort required to port these applications to the vector architecture. Results show that the SX6 achieves high performance for a large fraction of our application suite and in many cases significantly outperforms the scalar architectures. The computationally intensive Cactus BenchADM code showed the best uni-processor vector performance (46%), achieving a factor of 14.3 (5.8) improvement over the Power3 (Power4), while only requiring recompilation on the SX6.

Name	Discipline	Lines of Code	Power3 % Pk	Power4 %Pk	SX6 % Pk	*P*	SX6 Speedup vs. Power3	SX6 Speedup vs. Power4
Cactus-ADM	Astrophysics	1200	16.7	11.5	26.1	8	14	5.8
TLBE	Plasma Fusion	1500	7.3	9.0	38.1	8	27.8	6.5
OVERFLOW-D	Fluid Dynamics	100000	13.0	10.1	24.3	8	5.0	3.7
GTC	Magnetic Fusion	5000	11.1	6.3	4.9	8	2.3	1.2
Mindy	Molecular Dynamics	11900	6.3	4.7	2.1	1	1.0	0.5

Table 11: Summary overview of application suite performance

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 codes 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 decent performance on both the cache-based and vector machines, but algorithmic support for these different architectures required substantial differences in programming styles. Some experiments were also carried out with hybrid programming (MPI plus OpenMP), but this had negligible benefit; these results were not reported.

Finally, we presented two applications with poor vector performance: GTC and Mindy. They feature indirect addressing, many conditional branches, and loop carried data-dependencies, making high vector performance challenging. This was especially true for Mindy, whose use of C++ objects made it difficult for the compiler to identify