Benchmark Tests on the Digital Equipment Corporation Alpha AXP 21164-Based AlphaServer 8400, Including a Comparison of Optimized Vector and Superscalar Processing *

Harvey J. Wasserman

Scientific Computing Group
Los Alamos National Laboratory
Los Alamos, NM 87545

Abstract: This paper reports single-processor performance of the DEC 8400 system, a multi-cpu mainframe based on the DEC 21164 microprocessor. Performance is compared with single processors of the CRAY J90, the IBM RISC System/6000 Model 39H, and the SGI Power Onyx (75 MHz MIPS R8000). Benchmark codes representing Los Alamos applications with a range of computational characteristics were used. An important part of the comparison uses a particle transport application with two implementations, one that is highly vectorized and one that is unvectorizable with memory access patterns more suitable for superscalar processors. The results suggest that the best architecture/implementation match is the vectorizable code running on the vector processor.

Introduction

The second generation of the Digital Equipment Corp. (DEC) Alpha AXP microprocessor is referred to as the 21164. An important difference between it and its predecessor, the 21064, is that the 21164 has twice the multiply/add throughput per clock period (CP), a maximum of two floating point operations (FLOPS) per CP vs. one for 21064. Thus, the peak performance of the 21164 that we tested can be calculated as twice its CPU clock rate in MHz (300 MHz, 600 MFLOPS). A version of the 21064 running at 150 MHz is used in the Cray Research Inc. CRAY T3D and a version of the 21164 will be used in the CRAY T3E.

* This work was performed under the auspices of the U.S. Department of Energy by Los Alamos National Laboratory under contract W-7405-ENG-36.

ICS'96, Philadelphia, PA,USA
0-89791-803-7/96/05

The AlphaServer 8400 is a shared-memory multiprocessor with up to 12 central processing units (CPUs) and up to 14 GB of memory. In this report we compare single-processor performance of the 8400 with that of the 66-MHz IBM RISC System/6000 POWER-2, the 75-MHz SGI MIPS R8000, and the Cray Research CRAY J90. (The Alpha 21164 is also available in DEC workstations such as the AlphaStation 800 5/300, and so comparison with a single-processor workstation, such as the RISC System/6000, is appropriate.) The performance comparison is based on Fortran benchmark codes that represent a portion of the Los Alamos National Laboratory supercomputer workload. The codes have already been used to evaluate performance of a variety of computing systems [1-3]. The advantage of using these codes, in addition to their specific workload representation and the extensive database of existing benchmark data using them, is that the codes also span a wide range of computational characteristics, such as vectorizability, problem size, and memory access pattern. The primary disadvantage of using them is that detailed, quantitative analysis of performance behavior of all codes on all machines is difficult.

Appearing in this report is a new implementation of a benchmark that had been used previously. Whereas the older version was written for a vector processor, the newer version is more optimized for microprocessor architectures. This new version presents the opportunity to measure performance on a single application using implementations that expose the respective strengths of vector and superscalar architecture.

All results in this report are from single processors. A subsequent article will explore shared-memory multiprocessing performance of the 8400 system.

The Alpha 21164

The 21164-based system that was used for these tests is a four-processor, DecServer 8400 shared-memory system containing 1 GByte total memory. This system is installed in the open computing network at Lawrence Livermore National Laboratory.

Many details of the Alpha microprocessor have been published elsewhere [4]; a few salient features are as follows. The Alpha consists of five independent functional units: An integer arithmetic unit with two separate 64-bit pipelines, a floating-point unit, also with two 64-bit pipelines, a load/store unit, and a bus interface/cache control unit that provides extra-chip communication. Four-way superscalar technology is employed, which allows simultaneous issue of certain combinations of load/store, integer arithmetic, floating-point arithmetic, and branch instructions.

The floating-point unit consists of two pipelines, one for addition and division (although division operations are not pipelined) and one for multiplication. Add and multiply latencies have been reduced from six CP on the earlier Alpha, to four CP each on the 21164. The floating-point register file holds 32 64-bit words and has a total of nine ports: two read ports and one write port for each arithmetic pipeline, two ports for loads from cache, and one port for stores.

The Alpha 21164 has the same 8-Kbyte, direct-mapped, write-through, on-chip data cache that its predecessor had, with the same four 64-bit word line size. A load instruction that hits in the data cache will suffer an initial latency of two cycles (reduced from 3 cycles in the 21064). What is altogether new in the 21164 is that the next level of the memory hierarchy consists of an on-chip, 96-KByte, 3-way set associative second-level cache. For this cache there is a 7-cycle (23.3-ns) latency for access to the first 64 bytes of data and a maximum transfer rate of 1 64-bit word each four cycles (4.8 GB/s) after that. The 21164 also contains another structure, called a miss address file, which can reduce latency for secondary cache access by merging multiple load misses that access the same 32-byte block of memory into a single read request.

Alpha AXP systems can be configured with a third-level, off-chip cache up to 64 MBytes in size that is direct-mapped and requires an additional penalty of about 28 CP per 64-bit word. The machine we used was equipped with a 4-MB third-level cache.

Results

All data were collected using DEC Fortran V3.8-711 and KAP/Digital_UA_F version 2.1. Compiler options used for optimization were -O5 and -tune ev5. In all of the Tables in this report, the codes are listed in order of increasing vectorization effects; that is, in order of increasing performance on a single CRAY C90 processor as measured by the Cray Hardware Performance Monitor. The actual performance monitor data are listed in the Appendix.

Table 1 presents a comparison of benchmark execution times from the 21064 (unpublished results from an AXP/3500 system) and the 21164. (Not all of our current benchmark codes were run on the 21064.) The expected speedup between the 21064 and 21164 is 4, based on two-fold increase in clock speed and 2-fold increase in FLOPS per CP. Some of the codes exhibit speedups that are below the expected value, and although the reason is not clear, the most likely explanation is that the compiler is not yet producing optimal code for the 21164. Note that the codes that exhibit the lowest speedups are highly-vectorizable codes with either long vector lengths or large strides (PUEBLO and HYDRO). For such codes it is possible that the second on-chip cache used in the 21164 causes an additional inefficiency relative to the 21064.

Table 2 lists benchmark execution times in seconds from single processors of the DEC 8400, IBM RS/6000 39H, SGI POWER Onyx, and CRAY J90. The result that stands out most prominently is the AXP's performance on MCNP, an important particle-transport code that is entirely non-vectorizable. The working set size for MCNP is also relatively small, since the code tracks each Monte Carlo "particle" individually and relatively few data are required for each particle. MCNP is thus a good example of the high performance obtainable from the 21164 on largely cache-resident codes. The AXP's advantage on MCNP is greatest when compared with the vector processor J90, but it is still substantial when compared with the other microprocessor-based systems. Although not shown in Table 2, the Alpha's time on our MCNP test is more

than four times faster than a single 6.0-ns CRAY Y-MP processor. (The Y-MP remains an important production platform within the LANL Central Computing Facility.)

Among the remaining codes there is a great deal of variation in the ordering and relative performance of the four machines, but several results seem clear. The first concerns relative performance vs. peak speed. There are only two cases in which the AXP's performance relative to the other systems approaches that of the peak speed ratios. This occurs with two moderately vectorizable codes (operations count-based vectorization levels in 60-80% range). Thus, for the vast majority of the codes studied here, the AXP's high theoretical peak performance is a poor indicator of its performance on real applications. The AXP does provide the fastest execution time in more than half of the total comparisons, but its advantage is mostly 30-50% or less, not the factors of 2.25, 2.0, and 3.0 predicted by the peak speeds (AXP: 600 MFLOPS, RS/6000: 267 MFLOPS, R8000: 300 MFLOPS; J90: 200 MFLOPS.

The second important result concerns the J90. On codes with no or only moderate vectorization levels, the AXP, with much faster clock period, lower operation latencies, and data caching, is much faster than the J90 processor. However, the J90 is faster than the AXP system on all of the codes with high vectorization levels. Although the J90 is only about 15% faster on HYDRO, it is more than 3.5 times faster on the larger SWEEP-D problem and it is on average (harmonic mean) 2.2 times faster on these six vectorizable codes . This is in spite of the AXP's 3-fold advantage in peak speed relative to the J90. Indeed, the J90 has the lowest peak speed of all the machines considered here; yet is the fastest on the highly vectorizable codes.

Among the microprocessor-based systems, the SGI and IBM systems are most similar architecturally, in that both can process a maximum of four FLOPS per CP whereas the AXP can process a maximum of two per CP. The results do not allow a precise determination of which method (slower clock with greater concurrent instruction issue vs. faster clock with fewer concurrent instructions) is better. In fact, with a few exceptions (as noted below), the three microprocessor systems provide nearly equivalent performance on the vectorizable codes.

The processors mentioned in this report all use different architectural strategies in order to minimize memory latency. The IBM RISC System/6000 uses a relatively large primary cache (128 KB) and a secondary cache, whereas the SGI MIPS R8000 system has no primary data cache (for floating-point data) and uses only a large, off-chip "streaming" cache. The Alpha AXP has a very small (8-KB), on-chip cache, a larger secondary cache (96-KB), also on-chip, and also a large, off-chip cache as well. The CRAY J90 uses vector registers, the primary advantage of which is compiler control, rather than dynamic placement of data. Since the vectorizable codes in the benchmark suite depend heavily on memory bandwidth, they can give some indication as to the relative success of each of these strategies.

HYDRO provides an interesting example of performance improvement on vectorizable code with use of external caches. HYDRO is a 2-D Lagrangian Hydrodynamics code that is representative of a large class of codes in use at Los Alamos, and it is the only vectorizable code in this study on which microprocessor performance approaches that of the single-processor CRAY J90. In older reports, microprocessor performance on HYDRO relative to vector machines has generally been lower than that of the other vectorizable codes in our test suite. For example, in 1990, HYDRO performance on a 33-ns IBM RS/6000-540 was only about 7% that of a single CRAY Y-MP processor, whereas five other vectorizable codes averaged at least twice that [5]. In HYDRO the majority of vector accesses occur with a stride equal to the size of the grid (~100) and the non-unit stride access can cause particularly poor primary data cache reuse. HYDRO also uses a vectorizable binary search routine that involves gather operations which also can cause large numbers of data cache misses. However, all three

Table 1. Comparison of DEC EV4 and EV5 Performance

Code	150-MHz AXP 21064 Time	300-MHz AXP 21164 Time*	Ratio
MCNP5000	64.8	16.0	4.1
TWODANT93	144.4	44.0	3.3
TWODANT915	80.9	18.2	4.4
WAVE	190.0	50.5	3.8
HYDRO	64.5	22.8	2.8
PUEBLO32	111.2	41.7	2.7

* in seconds

Table 2. Comparison of Benchmark Execution Times (seconds) on Single Processors

Code	DEC 21164 300MHz Time	IBM RISC System/6000-39H Time [2]	SGI MIPS R8000 Time	CRAY J90 Time
MCNP	16.0	44.1	64.1	128.8
TWODANT93	44.0	51.0	86.2	97.8
WAVE	50.5	60.6	55.4	67.8
TWODANT915	18.2	27.6	27.0	24.5
SWEEP-L50	23.2	14.7	19.0	20.2
SWEEP-L75	79.4	48.0	64.3	64.1
HYDRO	22.8	26.9	21.1	19.7
NEUT	1117.6	1543.2	1523.2	557.0
SWEEP-D50	32.7	29.0	29.9	10.4
SWEEP-D75	129.8	114.9	217.1	35.2
POP	120.2	83.7	140.0	42.0
PUEBLO32	41.7	47.6	51.6	17.8

[1] 2-MB L2 Cache.

microprocessors in this report are equipped with off-chip caches into which HYDRO (~0.75 MB) fits completely. For this 2-D code the secondary cache allows all three microprocessors to perform within 40% or so of the J90/1. In comparison, the RS/6000 Model 590, which has no secondary cache, is more than two times slower than the J90/1 on HYDRO [2].

At the very least, these results show the level of inaccuracy that can result from using solely cache-resident benchmark codes when determining microprocessor performance. Factors of two to six separate the 2-D HYDRO benchmark from the results of the 3-D vectorizable codes.

NEUT is another code on which microprocessor performance has always been considerably out of line with expectation (e.g., RS/6000 performance 30X worse than CRAY C90, rather than factors of four or so [2]). In contrast with HYDRO, the most time-consuming routine in NEUT involves all straight-forward, stride-1 computation. However, in this routine, four loops using 28 arrays of length 32k are used (~7 MB), and so certainly on-chip data cache reuse, and even off-chip cache reuse is probably poor. Interestingly, the DEC AXP system is about 30% faster than both the IBM and SGI systems, suggesting that the AXP's 3-cache memory system may provide improvement on codes with very long, contiguous vectors.

Another scan of the execution data suggests a possible weakness of the SGI cache strategy relative to the AXP on larger problems. The SGI R8000-based system is faster than the AXP system on two codes, HYDRO and the smaller SWEEP-D benchmark. The codes on which the AXP system is faster are all larger problems, e.g., 1-D, order 32K for NEUT, 75-cubed for SWEEP-D75, 256 X 128 X 20 for POP, and 1-D, order 32K for PUEBLO. In other words, to the extent that execution times depend solely on memory throughput and the data cache strategy (not true; this neglects, at the least, per-cycle floating-point throughput and compiler effects), the benchmark data suggest that for larger problems the AXP system is superior to that of the SGI system. As noted in a separate comparison of the IBM RISC System/6000 Model 590 and the SGI POWER Onyx [6], the MIPS R8000 suffers from relatively poor memory bandwidth between the streaming cache and main memory, so that when a code's working set size exceeds that of the cache, MIPS R8000 performance degrades.

Optimization of Microprocessor Performance. For several years we have been making performance comparisons between microprocessor-based systems and vector processors. The caveat we had to employ each time was that the benchmark codes were written with vector processors in mind and thus may have

contained certain characteristics that enhanced vector performance and at the same time degraded microprocessor performance.

With SWEEP (a 3-D neutral particle transport code that uses the Sn method, [7]), we now have two very different implementations of the same code, one optimized for vector processors and another optimized for cache-based processors. The comparison of these two versions of SWEEP shows the extent to which reorganization of a vector code can provide significant benefit on microprocessors.

In both versions of SWEEP the main part of the computation consists of a "balance" loop in which particle flux out of a cell in three Cartesian directions is updated based on the fluxes into that cell and on other quantities such as local sources, cross section data, and geometric factors. The cell-to-cell flux dependence, i.e., a given cell cannot be computed until all of its upstream neighbors have been computed, implies a recursive or wavefront structure.

The difference between the two implementations of SWEEP is best shown using data from the Cray Hardware Performance Monitor running on a CRAY C90 system (see Table 3). In one version, (labeled "SWEEP-D") the mesh is swept using diagonal planes, which enables the balance loop to be vectorized. In this version gather/scatter operations must be used to obtain local source and cross sectional values. The CRI Hardware Performance Monitor shows that SWEEP-D has a average computational intensity, defined as the number of FLOPS divided by the number of loads and stores, of 0.5, i.e., the code is highly memory bandwidth-dependent. However, on the Cray, it is greater than 96% vectorized (based on operation counts), and it achieves about 30% of peak CRAY C90 processor performance. It should be noted that SWEEP has a relatively low floating-point intensity in general; i.e., without additional computation such as flux fixup or flux leakage there are less than 40 FLOPS per grid point per discrete direction per iteration regardless of the implementation.

Table 3. Characteristics of Two Versions of SWEEP as Determined by the CRAY C90 Hardware Performance Monitor (Single-Processor).

	SWEEP-L50	SWEEP-D50
Average MFLOPS	126.5	293.1
Average Hardware Vector Length	50.5	122.1
Percent Vector Operations	66.8	97.0
Average Computational Intensity [1]	1.11	0.54

[1] Total FLOPS divided by total memory references.

The version of SWEEP labeled "SWEEP-L" does not use a diagonal plane sweep; rather, the three Cartesian directions are swept explicitly in a 3-D loop nest. This "line-sweep" version eliminates the need for any gather/scatter operations; in fact, all memory accesses are now unit-stride. Furthermore, there is a substantial reduction of memory traffic through "scalarization" of several arrays, so that the computational intensity is increased to 1.11. However, with the balance loop proceeding along columns and rows rather than along the diagonal, recursion now prohibits complete vectorization. (Using the Cray *fpp* preprocessor the balance loop is split and some of the computation is vectorized.) On the C90, the operation-count vectorization level is about 66% and per-processor performance is reduced to 125 MFLOPS (12% of peak).

The execution time data in Table 2 show that on the vector-optimized, diagonal-sweep version (SWEEP-D) the CRAY J90 processor is three times faster than all of the microprocessor systems for the smaller grid and 3 - 6 times faster for the larger grid.

Using the microprocessor-optimized line-sweep version (SWEEP-L), microprocessor (AXP, IBM, and SGI) performance improves by factors of 1.5 - 2 for the small grid and by factors of 1.6 - 3.4 for the larger problem. However, because of poor vectorization, using the line-sweep version, CRAY J90 performance *decreases* by a factor of two, and on this version microprocessor performance is the same as (or in the case of the IBM RS/6000 better than) that of the J90 processor.

Nevertheless, the fastest implementation/architecture match is that of the diagonal sweep version running on the vector processor. Comparing the best implementation on each type of machine shows that the J90 is twice as fast as the SGI and DEC processors and 1.4 times as fast as the IBM processor.

Note again, the difference in peak speeds of the processors: DEC: 600 MFLOPS; SGI: 300 MFLOPS; IBM: 270 MFLOPS; J90: 200 MFLOPS.

In other words, even with extensive restructuring of a vector code and concomitant two-fold improvement in microprocessor performance, vector processor performance is still superior to workstations with peak speeds that are 1.5 to 3 times higher.

Furthermore, there are implications for code developers: The results for this code show how optimization for microprocessors adversely affects vector processor performance (e.g., SWEEP-D50 CRAY J90 time = 10.4 seconds, SWEEP-L50 CRAY J90 time = 20.2 seconds).

Two different sizes of both SWEEP implementations were run in order to further assess the effect of cache performance on microprocessor execution time. The J90 vector processor shows no dependence on problem size on a per-gridpoint basis. Among the microprocessor-based machines, both the Alpha and RS/6000 systems also show little problem size effect. However, using the non-optimal diagonal-sweep version, SGI performance is worse on the bigger problem (which is 3.3-times larger but runs more than seven times slower than the smaller one). Again, previous tests have shown how SGI MIPS R8000 performance degrades significantly due to low memory-to-cache bandwidth once the working set size exceeds the capacity of the secondary cache [6]. However, note that using the more optimal line-sweep version of the code eliminates this problem size dependence on the SGI entirely.

Note, also, that both the 50-cubed and 75-cubed problems run in this benchmark are much smaller than the problem sizes that are desired to be run; thus, the cache effect observed in the comparison of the vectorized version of the code would be even more exaggerated.

Conclusions

One conclusion from the SWEEP comparison is that benchmark codes written for vector processors may not be the best way to measure microprocessor performance. However, if an existing vector workload is to be ported to a microprocessor-based system then the vector codes must be used to obtain an estimate of initial performance on the microprocessor system without tuning. The results presented here show the kind of extensive re-organization of vector codes that must be done in order to optimize for microprocessors, and give an estimate of the kind of performance improvement that can be expected. The key optimizations were elimination of scatter/gather operations and drastic reduction in memory traffic. However, even with this extensive rewriting, and even using microprocessors with fairly large secondary cache structures, overall performance of the vector processor was still superior.

The DEC AXP 21164 processor provides, as expected, a significant improvement in performance relative to its predecessor, the 21064. The performance of the 21164 relative to other contemporary microprocessor-based workstations and compute servers varies widely depending on the characteristics of the benchmark code. In particular, codes that do not vectorize run extremely well on the single processor of the DEC 8400 that we tested. However, on codes that do vectorize, despite its very high CPU clock speed and associated theoretical peak computing rate, performance of the AXP 21164 is not significantly better than that of IBM and SGI microprocessor-based systems.

Comparing Alpha AXP 21164 and Cray C90 benchmark execution times along with operation counts from the C90 Hardware Performance Monitor suggests that on none of our codes does the Alpha exceed 10% of its theoretical peak computation rate. The percentage of peak theoretical performance achieved is an important metric. Performance on real applications requires *balance* between various processor architecture components, particularly between the floating-point units and the memory subsystem. The results suggest that on some codes, substantial tuning is necessary in order to accommodate the relative imbalance between the Alpha's floating-point computation rate and realizable memory bandwidth.

Note, too, that *per-processor* Alpha performance as implemented in a processor such as the 8400 mainframe, with its 4-MB third-level cache, might be expected to be significantly greater than it would be in an MPP such as the CRAY T3E, in which no external cache is present.

The results suggest that in spite of the progress made in microprocessor architecture, transistor densities, cache size and organization, and CPU clock speeds, vector processors such as the CRAY J90 can still out-perform microprocessors on some vectorizable codes. The J90 uses a relatively conservative CMOS technology resulting in a low clock speed relative to today's fastest ECL/Bipolar vector processors (e.g., a factor of 5 slower). Microprocessors will undoubtedly continue to improve clock speeds, but improvement in CMOS vector processor clock speeds is likely to continue as well.

Finally, the wide variation in relative performance observed on the suite of codes used here strongly suggests that popular benchmarks which yield single-number results are inadequate measures of performance for multi-issue microprocessor and vector architectures.

Acknowledgments

The author wishes to thank the following people for either help in running the benchmarks or in interpreting the results: Kaivalya Dixit and Liau J. (Danny) Shieh (IBM, Austin); John Shakshober (DEC); Faith Shimamoto and Alice Chen (LLNL); Steve Simmonds (SGI); Richard Sandness (CRI); and Ken Koch (LANL). This research was performed in part using resources located at the Advanced Computing Laboratory of Los Alamos National Laboratory.

Appendix: Description of Benchmark Codes

MCNP: A general-purpose Monte Carlo particle transport code widely used and Los Alamos and elsewhere [8]. The code treats an arbitrary three-dimensional configuration of materials in geometric cells bounded by first-, second-, and fourth-degree surfaces. The benchmark problem transports 5,000 source particles.

TWODANT: A two-dimensional discrete ordinates particle transport code used for neutral particle transport [9]. It includes a multigrid solver and is vectorizable to some extent. Two different problems are run that exercise different portions of the code. Both problems are three-group tests with fission. TWODANT915 runs a "k-calc" computation and TWODANT93 runs a fixed-source multiplication test for a fixed value of k. The executable size is approximately 10.8 MBytes.

WAVE: A two-dimensional, relativistic, electromagnetic particle-in-cell simulation code used to study various plasma phenomena [10]. WAVE solves Maxwell's equations and particle equations of motion on a Cartesian mesh with a variety of field and particle boundary conditions. The benchmark problem involves 500,000 particles on 50,000 grid points for 20 timesteps; about 4 MW of memory are required. One routine containing loops of length 256 and considerable indirect addressing dominates the code's runtime.

HYDRO: A two-dimensional Lagrangian hydrodynamics code based on an algorithm by W. D. Schulz [11]. HYDRO is representative of a large class of codes in use at the Laboratory. The code is 100% vectorizable. A typical problem is run on a 100 X 100 mesh for 100 time steps. An important characteristic of the code is that most arrays are accessed with a stride equal to the length of the grid.

NEUT: A highly vectorizable Monte Carlo neutron transport code. that runs a k-calc computation starting with 32K neutrons. NEUT represents a Fortran77 version of Eldon Linnebur's (LANL) Connection Machine Fortran code [12].

SWEEP: SWEEP3D is a three dimensional solver for the time independent, neutral particle transport equation on an orthogonal mesh [7]. The first-order form of the transport equation is solved by sweeping through the spatial mesh along discrete directions (ordinates). The solution algorithm in SWEEP3D is vectorized/parallelized by sweeping though the mesh along diagonal planes, which requires large numbers of data gathers/scatters and extensive array indexing. Two problem sizes are run as benchmark codes, using a 50 X 50 X 50 or 75 X 75 X 75 grid (~85 MBytes). The benchmark is a Fortran77 implementation of a data-parallel version of the code.

POP: A global ocean model developed on the Thinking Machines Inc. CM-2 and translated into Fortran77 [13]. POP is based on the Bryan-Cox-Semtner model but uses reformulated barotropic equations to solve for surface-pressure field rather than a volume-transport streamfunction. It uses a preconditioned conjugate-gradient solver.

PUEBLO: A 3-dimensional Lagrangian hydrodynamics code used to model point explosions in space [14]. The code is highly vectorizable, although Cray compiler directives are currently included. The most common loop length is on the order of n^3, where n = 32 for PUEBLO32 or 64 for PUEBLO64.

Characteristics of the Benchmark Codes as Determined by the CRAY C90 Hardware Performance Monitor (Single-Processor Results)				
CODE	Average MFLOPS	Average Hardware Vector Length	Percent Vector Operations	Average Computational Intensity *
MCNP	11.6	12.4	0.2	0.58
TWODANT93	54.3	15.3	58.8	0.58
WAVE	77.5	66.3	63.0	0.88
TWODANT915	96.9	70.4	79.8	0.70
SWEEP-L50	126.5	50.5	66.8	1.11
HYDRO	177.7	92.9	94.4	1.00
NEUT32	278.0	111.2	96.6	0.87
SWEEP-D50	293.1	122.1	97.0	0.54
POP	362.1	122.9	96.8	0.64
PUEBLO32	458.4	119.9	98.2	1.31
* Defined as total floating-point operations divided by total memory references				

References

[1] M. L. Simmons and H. J. Wasserman, "Benchmark Tests on the Cray Research, Inc. CRAY J90," Los Alamos National Laboratory Unclassified Release LA-UR-95-3827, http://www.c3.lanl.gov/~hjw/Web_Papers/j90/j90.ps.Z, 1995.

[2] H. J. Wasserman, "Benchmark Tests on the New IBM RISC System/6000 590 Workstation," *Scientific Programming*, Vol. 4, No. 1, pp23-34, Spring, 1995.

[3] M. L. Simmons, H. J. Wasserman, O. M. Lubeck, C. Eoyang, R. Mendez, H. Harada, and M. Ishiguro, "A Performance Comparison of Four Supercomputers," *Comm. of the ACM*, **35**, 116-124, 1992.

[4] P. Bannon and J. Keller, "Internal Architecture of Alpha 21164 Microprocessor," Digest of Papers COMPCON '95, March 5, 1995, IEEE Computer Society Press (Los Alamitos, CA), pp 79-87.

[5] M. L. Simmons and H. J. Wasserman, "Los Alamos Experiences with the IBM RISC SYSTEM/6000 Workstations," Los Alamos National Laboratory Manuscript LA-11831-MS, 1990.

[6] H. J. Wasserman, "Benchmark Tests on a Silicon Graphics R8000-Based Workstation," http://www.c3.lanl.gov/~hjw/Web_Papers/sgi/sgi.ps.Z, 1995.

[7] Koch, K. R., Baker, R. S. and Alcouffe, R. E., "Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor," Trans. of the Amer. Nuc. Soc., **65**, 198, 1992.

[8] J. Briesmeister, ed. "MCNP: A General Monte Carlo Code for Neutron and Photon Transport," Los Alamos National Laboratory report LA-7396-M Rev 2, September, 1986.

[9] R. D. O'Dell, F. W. Brinkley, Jr., D. R. Marr, and R. E. Alcouffe, "Revised User's Manual for ONEDANT: A Code Package for One-Dimensional Diffusion-Accelerated, Neutral Particle Transport," Los Alamos National Laboratory Manual LA-9184-M, December, 1989.

[10] R. L. Morse and C. W. Neilson, "Numerical Simulation of the Weibel Instability in One and Two Dimensions, " Phys. Fluids, Vol 14, p 4, 1971.

[11] W. D. Schulz, "Two-Dimensional Lagrangian Hydrodynamic Difference Equations," Meth. Computational Phys. Vol 3, p1, 1964.

[12] O. M. Lubeck, M. L. Simmons, and H. J. Wasserman, "The Performance Realities of Massively Parallel Processors: A Case Study," Proc. Supercomputing '92, IEEE Computer Society Press, 403-411, 1992.

[13] (a) R. Smith. R. Malone, and J. Dukowicz, "Parallel Ocean General Circulation Modeling, " Physica D 60, pp 38-61, 1990. (b) J. Dukowicz and R. Smith, "Implicit Free-Surface Method for the Bryan-Cox-Semtner Ocean Model," Los Alamos National Laboratory Unclassified Release LA-UR-93-2031.

[14] PUEBLO3D, written by Eugene Symbalisty of LANL, is a "stripped-down" version of the LANL CAVEAT production code. See F. L. Addessio, et al., "CAVEAT: A Computer Code for Fluid Dynamics Problems with Large Distortion and Internal Slip," Los Alamos National Laboratory Report LA-10613-MS (1986).