LA-UR -90-2564

LA-UR--90-2564

DE90 015058

TITLE PERFORMANCE EVALUATION OF THE IBM RISC SYSTEM/6000: COMPARISON OF AN OPTIMIZED SCALAR PROCESSOR WITH TWO VECTOR PROCESSORS

AUTHOR(S) Margaret L. Simmons and Harvey J. Wasserman

SUBMITTED TO Supercomputing '90, New York NY, November 12, 1990

DISCLAIMER

Los Alamos Los Alamos National Laboratory Los Alamos, New Mexico 87545

Performance Evaluation of the IBM RISC System/6000: Comparison of an Optimized Scalar Processor with Two Vector Processors

Margaret L. Simmons and Harvey J. Wasserman

Computer Research Group
Computing and Communications Division
Los Alamos National Laboratory
Los Alamos, New Mexico 87545

ABSTRACT

RISC System/6000 computers are workstations with a reduced instruction set processor recently developed by IBM. This report details the performance of the 6000-series computers as measured using a set of portable, standard-Fortran, computationally-intensive benchmark codes that represent the scientific workload at the Los Alamos National Laboratory. On all but three of our benchmark codes, the 40-ns RISC System was able to perform as well as a single Convex C-240 processor, a vector processor that also has a 40-ns clock cycle, and on these same codes, it performed as well as the FPS-500, a vector processor with a 30-ns clock cycle.

1. Introduction

In this paper we report the results of a study of IBM RISC System/6000 central processing unit (CPU) performance as measured by a set of Fortran benchmarks representing the scientific workload at Los Alamos National Laboratory (LANL). The RISC System/6000 is a reduced instruction set computer (RISC) whose architecture may be referred to as "superscalar" (or optimized scalar), because the hardware can carry out more than one instruction per clock period.[1] Two RISC System/6000 computers were recently loaned to the Advanced Computing Laboratory (ACL) at LANL as part of a beta-testing agreement.

Our benchmark codes have been used to test an extensive list of high-performance computers.[2-5] In particular, they have been very useful in measuring the performance of a wide variety of vector processors. In this paper we will compare the performance of the new IBM RISC System with two of these vector machines, especially a single processor of the Convex C-240 system. We concentrate on this machine simply because it has the same CPU clock cycle as one of the IBM RISC Systems. As such, the data allow direct comparison of the efficiency of a vector processor relative to that of an optimized scalar processor, the only such comparison of which we are aware. The other vector processor we include is the Floating Point Systems FPS-500 FPX.

3. Results

3.1 Performance on Primitive Vector Operations

Two of our benchmarks, VECOPS and VECSKIP, are designed to measure performance of vector operations as a function of vector length for several different memory access patterns. Although the RISC Systems do not have vector capability, the memory is organized as a hierarchy composed of registers, a (relatively) large, fast data cache, and a larger, somewhat slower main memory. On an architecture such as this these codes can measure the effect on performance of in-cache residency of data vs. out-of-cache requests. In Table 1, we report rates for "vector operations" as a function of "vector length," in order to maintain consistency with the corresponding data from a vector computer, given in Table 2. However, because the RISC Systems are not vector computers, these terms refer to an iterative operation on arrays, still denoted as "V1," "V2," etc., and a particular loop length of interest. In both VECOPS and VECSKIP, there is an inner loop that performs the indicated operation, while an outer loop simply repeats this inner loop enough times to get measurable results.

A plot of rate vs. vector length for the operation V1 = V2 + V3 on the 40 ns IBM RISC System is shown in Figure 1. The plot shows a dependence on vector length that is characteristic of a vector computer; this occurs on the RISC System because of the latency associated with loading the operands (and storing the results) through the cache. The time, T, to carry out a vector operation of length n, has been described by Hockney[8] in terms of an asymptotic rate, $\mathbf{r}_\infty$, and a vector half-length, $n_{1/2}$ (eq. 1). Others[5,9,10] have described the same functionality in terms of a linear model with a startup time, T_s, and an element time, T_e (eq. 2).

$$T = (n + n_{1/2}) / \mathbf{r}_\infty \qquad (1)$$

$$T = T_s + n\, T_e \qquad (2)$$

These two descriptions are algebraically equivalent, as noted by Lubeck.[5] In Figure 1, the data are shown by large dots (•) while the smooth curve that has been drawn represents an approximate fit to the data using the Hockney formulation with $\mathbf{r}_\infty$ = 8.2, and $\mathbf{n}_{1/2}$ = 2.45. A fit using the straight line method to these data will be discussed in more detail below.

In the operation that was carried out to produce Figure 1, all of the required data, the two operands V2 and V3, and the result, V1, fit within the 64-kbyte cache on the 40-ns RISC System. Significantly different results are observed when the amount of data required for the operation exceeds the capacity of the cache, or when the arrangement of data in main memory is such that the memory-cache mapping is non-optimal. An example of this is shown in Figure 2, which depicts a plot of rate vs. vector length for the operation V1 = V2 * V3

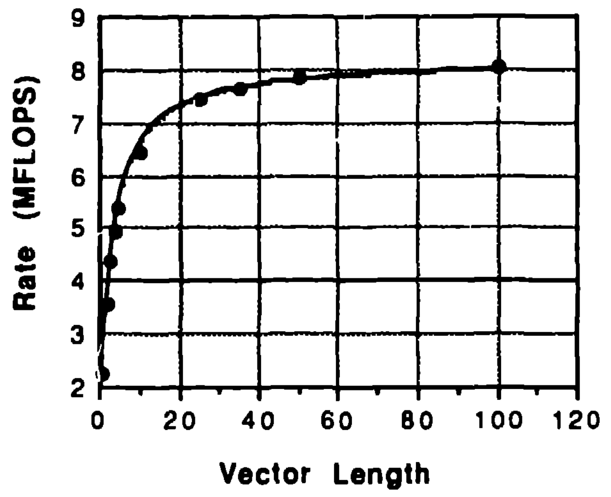

Figure 1. Plot of rate in millions of floating-point operations per second (MFLOPS), vs. vector length for the "vector" operation V1 = V2 + V3 on the 40-ns IBM RISC System/6000. Experimental data are represented by bullets(•), while the smooth curve represents a calculation using the Hockney parameters r_∞ = 8.2 MFLOPS and $n_{1/2}$ = 2.45.

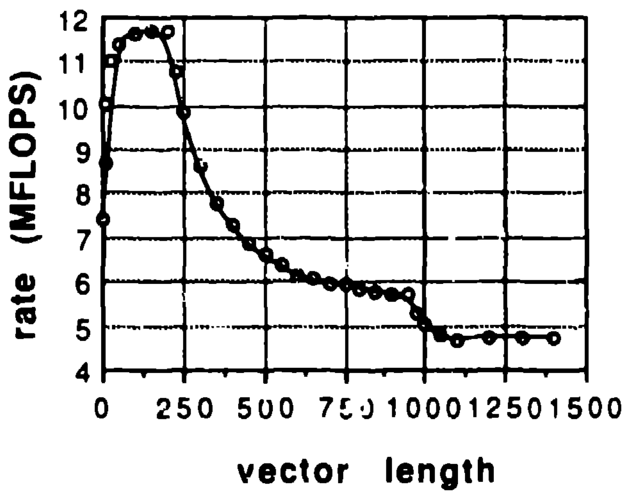

Figure 2. Plot of rate (MFLOPS) vs. vector length for V1 = V2 * V3 + V4* V5 on the 50-ns IBM RISC System/6000.

+ V4 * V5 on the 50-ns IBM RISC System/6000.

This machine, contains a 32-kbyte cache. The overall shape of the curve in Figure 2 has been qualitatively described previously using similar data collected on a single processor of the Alliant FX system.[11] The initial part of the curve shows the same functionality as in Figure 1. Cache misses become significant at about vector length 225, where a steep drop in rate is observed, and at a vector length of about 1100, the rate becomes constant. At this point the observed rate is largely a function of the rate at which the cache line (64 bytes) can be refilled from main memory. The observed rate of about 5 MFLOPS corresponds to a result every 15 clock periods, which is the number of clock periods required to reload a cache line following a miss. The sharp drop in rate shortly after vector length 800 may be due to the total capacity of the cache being exceeded, since the maximum vector length that can be accomodated for five vectors is 819 (double-precision) words.

While a detailed explanation of the cache behavior is beyond the scope of this paper, we note that performance can be a sensitive function of the relative starting addresses (in memory) of the arrays involved and the organization of the cache, as well as the loop lengths of interest and the loop strides. The 50-ns RISC System has 512 cache lines and 128 cache sets. It is possible to define "pathological" cases where each of the five arrays are mapped into the same cache line and cache set. For the five-vector system, this means that every reference to the fifth vector will require reloading a cache line. In such a case, even with unit stride, we have observed rates of less than one MFLOP per second at all vector lengths.

Table 1 lists the megaflop rates obtained for a variety of "vector" operations on the 40-ns IBM RISC System. For comparison, the corresponding data from a single processor of a Convex C-240 minisupercomputer are given in Table 2. We can make the following qualitative comparisons based on the data from the two machines. First, the IBM RISC System clearly has an advantage at short vector lengths. This advantage is quite lopsided at vector lengths as long as 25 or so (not shown in the Tables). At longer, asymptotic vector lengths, differences between the two machines mostly vanish, with two exceptions. The exceptions, which are discussed below, are for the operations V = V + S and V = S * V + S * V.

Table 3 lists the results of least-squares fits to the 40-ns IBM RISC System data using equation (2). The value of the start-up time, T_s, given in Table 3 for A(I) = B(I) + S, is much smaller than the value of 1922 nanoseconds (ns) we found for the Convex.[3] This accounts for the substantial difference in rates between the two machines at very small vector lengths.[12]

To understand the differences in rates at larger vector lengths we must look at what the model predicts for the rate at which each machine can produce a result for each operation, shown in the third column of Table 3. It is at this point we begin to see evidence of the instruction-level parallelism provided by the IBM RISC System.

Consider, for example, the operation A(I) = B(I) + C(I). This operation requires the sequence of instructions **LOAD B**, **LOAD C**, **ADD**, and **STORE A**, which consumes four clock periods for each result. However, the data indicate a time-per-element of 120 ns, which corresponds to three clock periods on the 40-ns RISC System. The RISC System accomplishes this by executing a (pre-)load for one of the operands at the same time that it performs the floating-point operation.

Table 3 includes an explanation of how the RISC System carries out several representative vector operations, showing the steps that are repeated in each loop after the pipelines have been loaded. In each case, the sequence of instructions involves a pre-

Table 1. Rate (MFLOPS) for Selected Vector Operations as a Function of Vector Length on the 40-ns IBM RISC System/6000

Operation	Vector Length				
	10	*50*	*100*	*500*	*1000*
V = V + S	8.0	9.5	9.7	9.9	9.9
V = V + S (I=1,N,2)	4.5	7.9	8.9	9.5	9.8
V = V + S (I=1,N,23)	4.5	7.9	8.9	0.8	0.8
V = V + V	6.3	8.0	8.0	8.2	8.2
V = V + S * V	12.7	16.0	16.0	16.4	16.4
V = V * V + V	10.1	12.1	12.1	12.3	12.3
V = S * V + S * V	17.3	18.4	18.4	18.5	17.8
V = V * V + V * V	12.4	14.6	14.6	14.8	14.8
V(I) = V(I) + V * V	7.2	8.2	8.1	8.2	8.2

Table 2. Rate (MFLOPS) for Selected Vector Operations as a Function of Vector Length on a Single Processor of the Convex C-240

Operation	Vector Length				
	10	*50*	*100*	*500*	*1000*
V = V + S	3.9	9.2	10.5	11.6	11.7
V = V + S (I=1,N,2)	2.4	8.7	10.8	11.1	11.1
V = V + S (I=1,N,23)	2.3	8.7	10.9	11.1	11.1
V = V + V	3.3	6.9	7.5	7.9	7.9
V = V + S * V	6.2	13.4	14.8	15.8	15.8
V = V * V + V	5.4	10.7	11.4	11.8	11.8
V = S * V + S * V	7.5	19.9	22.0	23.3	23.5
V = V * V + V * V	6.5	13.2	13.9	14.4	14.5
V(I) = V(I) + V * V	3.1	6.3	6.8	7.2	7.2

fetch being performed in parallel with a floating-point operation. By so doing, the RISC System is able to eliminate one clock period per iteration from the sequence that would otherwise be required in a single-port vector computer such as the Convex. This also allows the RISC System to achieve the same flop rate per cycle as an IBM 3090/VF processor.[13]

It is interesting to compare the efficiency of the RISC System/6000 with that of a Convex processor for the operation A(I) = S1 * B(I) + S2 * C(I). On the Convex this is a "three-chime" operation, involving the sequence **LOAD C - MULTIPLY S2** (chained), **LOAD B - MULTIPLY S1 - ADD** (chained), followed by **STORE A**. Thus the Convex can produce a result every three clock cycles, suggesting a peak rate of 25 MFLOPS; about 24 MFLOPS was observed. The RISC System, on the other hand, requires four clocks per result, suggesting a peak rate of 18.75 MFLOPS, while about 17 MFLOPS are observed. This is the only operation in the group we have studied for which the rate on a Convex processor can exceed the rate on the RISC System. Apparently, the RISC System compiler is not extracting all the available parallelism. The operation seems to involve the sequence **LOAD B_i, MULTIPLY S2, C_i - Load C_{i+1}** (simultaneously), **MULTIPLY / ADD, STORE A**. In principal, the multiply should occur simultaneously with the **LOAD B_i**, and the multiply / add should occur simultaneously with the **LOAD C_{i+1}**.

At this time we cannot explain the results for the RISC System on the operation V = V + S, which runs at the aggregate rate of one result every 2.5 cycles (10 MFLOPS asymptotic rate). An asymptotic rate of about 12.5 MFLOPS, corresponding to one result every 2 clocks, is expected, and an examination of the generated code shows that the instruction sequence given in Table 3 is nominally correct. Further explanation of the V = V + S (and V = V * S) operation is required.

3.2 Comparison of Performance on Larger Benchmarks

In Table 7 we show benchmark execution times for the 40-ns RISC System along with those from a single processor of the Convex C-240 system.[3] A complete description of the C-240 has appeared elsewhere.[14] Here, we merely note that it too is a demand-paged dynamic-memory system, and it has a 4-kbyte scalar data cache, independently pipelined functional units, and eight 128-element 64-bit vector registers. Our benchmark of the C-240 was carried out in July, 1989.

A description of the benchmark codes may be found in the appendix to Reference 2. All codes use 64-bit precision. Timing on the IBM RISC System was carried out by reading the 64-bit real-time clock using an IBM-supplied routine.

We consider first several codes in our benchmark set that do not vectorize. These are GAMTEB, for which we estimate the level of vectorization is about 20%, and ESN, SCALGAM, and PHOTON, which do not vectorize at all. In a previous report[2] we compared the performance of the 40-ns IBM RISC System/6000 with two other RISC-based

Table 3. Startup Times, T_s, and Execution Times, T_e Derived from the Least-Squares Fit, and Execution Sequences for Various Vector Operations on the 40-ns IBM RISC System/6000

Operation	T_s, *ns*	T_e *ns (clocks)**	*Execution Sequence After Startup*
A(I) = B(I) + S	253	100 (2.5)	1) Add B_i, S; Load B_{i+1} 2) Store A
A(I) = B(I) + S (I = 1, N, 2)	1233	100 (2.5)	Same as above
A(I) = B(I) + C(I)	315	120 (3)	1) Load C_i 2) Add B_i, C_i ; Load B_{i+1} 3) Store A_i
A(I) = B(I) + S * C(I)	325	120 (3)	1) Load C_i 2) Multiply / Add B_i, S_i, C_i ; Load B_{i+1} 3) Store A_i
A(I) = B(I) + C(I) * D(I)	300	160 (4)	1) Load D_i 2) Load B_i 3) Multiply / Add B_i, C_i, D_i ; Load C_{i+1} 4) Store A_i
A(I) = S1 * B(I) + S2 * C(I)	240	160 (4)	1) Load B_i 2) Multiply S2, C_i; Load C_{i+1} 3) Multiply / Add S1, B_i, result from 2) 3) Store A_i
A(I) = B(I) * C(I) + D(I) * E(I)	360	200 (5)	1) Load D_i 2) Load C_i 3) Multiply E_i, D_i ; Load B_{i+1} 4) Multiply / Add B_i, C_i, res. from 3); Load E_{i+1} 5) Store A_i

* The time-per-element is given first in nanoseconds, and then as the number of CPU clock periods to which this value corresponds.

computers, the SUN-4/390 and the Digital Equipment Corporation DECstation-3100. The results of comparing the RISC System with the Convex are almost directly opposite those for the SUN and DECstation. Relative to the SUN and the DECstation, the IBM RISC System performed best on codes that could vectorize. However, the Convex has the advantage on these codes, and the relative performance of the RISC System is best on the scalar codes in the benchmark set. The RISC System runs SCALGAM and PHOTON about 1.5 times faster than does a single processor of the Convex C-240,

Table 4. Comparison of Benchmark Execution Times[1] for the 40-ns IBM RISC System and a Single Processor of the Convex C-240

Code	Convex C-240 Time	40-ns IBM RISC System Time	Ratio: $T_{C\text{-}240}/T_{IBM}$
HYDRO	74.6	200.8	0.4
VGAM	4.4	7.0	0.6
MATRIX	140.0	147.0	1.0
WAVE	410.2	364.3	1.1
LSS300	540.2	497.3	1.1
LSS	23.5	20.2	1.2
FFT	14.7	11.5	1.3
GAMTEB	10.7	8.1	1.3
SCALGAM	242.1	163.4	1.5
PHOTON	338.2	221.3	1.5
INTMC	31.1	18.6	1.7
ESN	41.9	18.0	2.3

[1]Times are in seconds on a dedicated machine.

although a critical fix to the IBM Fortran DINT intrinsic function, a major time consumer in these two codes, was required.[2]

On the vectorized codes in the benchmark set, performance of the two machines is mixed. On MATRIX, LSS, LSS300, WAVE, and FFT, the two machines are basically equivalent. However, a single Convex C-240 processor runs HYDRO and VGAM much faster than the 40-ns RISC System. The vectorizable code that the RISC System runs fastest relative to the Convex is FFT, which involves short vector lengths that expose the relatively larger vector startup times on the Convex. In spite of comparable performance for MATRIX and LSS on the C-240 and RISC System, a single processor of the Convex C-240 runs HYDRO 1.6 times faster than does the 40-ns IBM RISC System. HYDRO involves vectors that are accessed with a stride of 100 and therefore the RISC System cache cannot support the required memory bandwidth. Although the Convex processor also has a cache, it is used only for scalar data. A more detailed discussion of the effect of bringing "vectors" of data through a cache on HYDRO may be found in the IBM-3090/VF benchmark, where similar degradations in rate were observed.[15]

The Convex runs the VGAM[17] benchmark about 1.7 times faster than the 40-ns RISC System. This code involves many gather operations at large vector lengths, which strongly suggests that the RISC System may again be suffering large numbers of cache misses.

Finally, we report preliminary results for PUEBLO3D, a new code that is to become part of our standard benchmark suite.[16] PUEBLO3D is a Lagrangian hydrodynamics code used to model point explosions in space. The code is highly vectorizable, although Cray compiler directives are currently included. Results are shown in Table 5, in which several levels of optimization for the Convex C-240 are included.

Table 5. Comparison of PUEBLO3D Execution Times[1] for the Convex C-240 and the 40-ns RISC System

40-ns RISC System Time	*Convex C-240 Time*	*Notes*
979.9 (231.4)	190.8	Single-processor Convex; no compiler directives
	98.8	Single-processor Convex with compiler directives
	50.0	Four-processor Convex

[1]Times are in seconds on a dedicated machine.

This is the only instance in this paper in which multiprocessor results are shown.

PUEBLO3D amplifies the problems associated with use of the RISC System cache on vector codes. The code is set up to run a grid of size 32 X 32 X 32 and arrays are initially dimensioned to this size. Thus, the 32K elements of this grid all map to the same cache set. This is why the performance for the 40-ns RISC System (979 seconds) is about 20% of the single-processor Convex performance. Because of the way the code is set up, it is impossible to change the array dimensions without changing the size of the problem that is run. So we ran a slightly larger case, 33 X 33 X 33, on the IBM. The execution time for this larger, "re-mapped" problem is 231 seconds, ~25% of the time for the smaller problem, and closer to the performance of the (single-processor) Convex without compiler directives. The Convex C-240 also achieves a speedup due to automatic concurrent-vector computation on PUEBLO3D, with the execution time decreasing to 50 seconds.

Table 6 presents a comparison of the 40-ns RISC System with another vector computer, the Floating Point Systems FPS-500 FPX. This machine has a 30-ns clock cycle. The FPS-500 processor consists of a RISC scalar processor rated at 33 MIPS and a vector processor containing eight 1024-element (64-bit) vector registers. We benchmarked a single-processor FPS-500 containing 128 Mbytes of memory. We used a beta-release of the FPS Version 3 Fortran compiler, which, with use of a compile-line option, inlined several intrinsic functions, notably DINT in SCALGAM and PHOTON.

The FPS-500's performance exceeds that of the 40-ns RISC System on HYDRO and VGAM, again because the non-contiguous access to memory in these codes causes too many cache misses on the RISC System, and on PUEBLO3D, due to the cache alignment problems discussed above. The relative ordering of the codes in Table 6 is roughly the same as it was in Table 4, except for the effect of the FPS-500's longer vector registers on LSS, LSS300, and FFT.

Table 6. Comparison of Benchmark Execution Times[1] for the FPS-500 and the 40-ns RISC System

Code	FPS-500 Time	40-ns IBM Time	Ratio: $T_{FPS\text{-}500} / T_{IBM}$
PUEBLO-3D	97.0	979.9 (231.4)	0.1 (0.4)
HYDRO	79.2	200.8	0.4
VGAM	4.1	7.0	0.6
LSS300	485.5	497.3	1.0
WAVE	388.3	364.3	1.1
MATRIX	170.0	147.0	1.2
SCALGAM	224.2	163.4	1.4
PHOTON	310.3	221.3	1.4
LSS	29.7	20.2	1.5
GAMTEB	14.3	8.1	1.8
INTMC	38.8	18.6	2.1
FFT	30.1	11.5	2.6
ESN	66.8	18.0	3.1

[1]Times are in seconds on a dedicated machine.

4. Conclusions

In a previous paper we compared the performance of the IBM RISC System/6000 machines with other RISC-based computers.[2] We reported that the biggest advantage the "superscalar" IBM architectures had the over other RISC-based systems was on the vectorizable floating-point codes in our benchmark set. This was particularly true for LSS and MATRIX, because these codes are dominated by the SAXPY operation, and thus could take advantage of the RISC

System's pipelined multiply-add instruction.

It is not surprising that the IBM RISC System held this advantage over the more conventional RISC systems on vectorized codes. In general, vectorization represents a highly-ordered form of computation that is amenable to any highly pipelined architecture, of which the RISC System/6000 is one. An important question, though, is whether a highly pipelined (super)scalar machine can achieve performance comparable to that of a vector processor. This study has provided some answers to this question.

We compared the IBM RISC System/6000, a machine with no vector capabilities, with two representative mid-range vector processors. On all but three of our benchmark codes, the 40-ns RISC System was able to perform as well as a vector processor having the same CPU clock cycle, and on these same codes, it performed as well as a vector computer having a faster CPU cycle. On codes that are largely scalar in nature, the RISC System performed better than both of the vector machines, possibly because of the RISC System's novel branch prediction/execution capabilities as well as its decreased pipeline startup time. The vector machines gained more of an advantage over the IBM workstations as vector lengths increased, and only on the codes that accessed memory with large strides did the vector processors perform significantly better than the IBM machine. In these cases, the cache on the RISC System prevented the hardware from maintaining a fully loaded floating-point pipeline. The cache is the only "weak link" that we have been able to discover in the RISC System/6000 architecture. Many of the codes developed at Los Alamos use constant non-unit strides through memory because the codes were developed for the Cray Research line of computers, on which the penalty for non-unit strides is less significant than on other architectures.^{5}

Finally, we remind the reader that the our benchmarks are intended to represent the scientific computing workload at LANL. Benchmarking is a highly workload-dependent endeavor, and thus one must use caution when comparing our results with those obtained using other workloads.

Acknowledgements

We wish to thank many people at IBM, John MacDonald, Cecil Severs, Deanna Collins, Minerva Carrera, Steve Vaughan, and particularly Tung Nguyen, for showing a great deal of enthusiasm during our joint-study project. We also thank Ann Hayes, Jerry DeLapp, and Andy White of the Los Alamos Advanced Computing Laboratory for making generous availability of ACL resources. We are indebted to Ronald Gray of Convex Computer Corporation for a great deal of help in running our benchmark codes on the C-240, to Chuck Niggley of FPS Computing for help in running our benchmarks on the FPS-500, and to Olaf Lubeck and James Moore of LANL for helpful discussions concerning cache behavior.

References and Notes

[1] IBM RISC System/6000 Technology, SA23-2619, International Business Machines Corporation,Austin, TX.

[2] M. L. Simmons and H. J. Wasserman, "Los Alamos Experiences with the IBM RISC System/6000 Workstations," Los Alamos National Laboratory report LA-11831-MS (1990).

[3] R. J. Koskela, M. L. Simmons, and H. J. Wasserman, "Performance Characterization of the Convex C-240 Computer System," Los Alamos National Laboratory report LA-11769-MS (February, 1990).

[4] H. J. Wasserman, "Los Alamos National Laboratory Computer Benchmarking 1988," Los Alamos National Laboratory report LA-11465-MS (December, 1988), and references therein.

[5] O. M. Lubeck, "Supercomputer Performance: The Theory, Practice, and Results," Los Alamos National Laboratory report LA-11204-MS (1988).

[6] International Business Machines Corporation, "Using the AIX Version 2.2.1 Operating System," IBM publication #SC23-2007-0, April, 1988. An updated version of this manual, designed especially for use with AIX Version 3.1 on the RISC Systems, will be available soon.

[7] International Business Machines Corporation, "IBM AIX RISC/xxx XL Fortran V 1.1 User's Guide," IBM publication #SC09-1257-00, October, 1989.

[8] R. W. Hockney and C. R. Jesshope, *Parallel Computers* (Adam Hilger, Bristol, 1981).

[9] O. Lubeck, J. Moore, and R. Mendez, "A Benchmark Comparison of Three Supercomputers: Fujitsu VP-200, Hitachi S810/20, and CRAY X-MP/2," IEEE Computer, **18** (1985) 10-29.

[10] I. Y. Bucher and M. L. Simmons, "A Close Look at Vector Performance of Register-to-Register Vector Computers and a New Model," Los Alamos National Laboratory document LA-UR-86-3886 (May, 1987).

[11] W. Abu-Sufah and A. D. Maloney, "Vector Processing on the Alliant FX/8 Multiprocessor," in *Proc. 1986 Internat. Conf. Parallel Processing*, IEEE Computer Society, 1986, pp. 559-566.

[12] The value of T_s for the Convex was obtained using a linear model in which the "stripmine" time, the time to reload the vector registers after each 128 have been processed, was included.

[13] P. Carnevali and M. Kindelan, "A Simplified Model to Predict the Performance of Fortran Vector Loops on the IBM 3090/VF," *Comput. 13* (1990) 35-46.

[14] M. Chastain, G. Gostin, J. Mankovich, and S. Wallach, "The Convex C240 Architecture," in *Proc. Supercomputing '88*, IEEE Computer Society, 1988, pp. 321-329.

[15] R. G. Brickner, H. J. Wasserman, A. H. Hayes, and J. W. Moore, "Benchmarking the IBM 3090 with Vector Facility," Los Alamos National Laboratory document LA-UR-86-3300 (September, 1986).

[16] PUEBLO3D, written by Eugene Symbalisty of LANL, is a "stripped-down" version of the CAVEAT 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).

[17] P. T. Burns, M. Christon, R. Schweitzer, O. M. Lubeck, H. J. Wasserman, M. L. Simmons, and D. V. Pryor, "Vectorization of Monte Carlo Particle Transport: An Architectural Study Using the LANL Benchmark 'GAMTEB'," *Proc. Supercomputing '89*, IEEE Computer Society, 1989, pp 10-20.