Benchmark Tests on the New IBM RISC System/6000 590 Workstation

HARVEY J. WASSERMAN

Computer Research Group, Los Alamos National Laboratory, Los Alamos, NM 87545

ABSTRACT

The results of benchmark tests on the superscalar IBM RISC System/6000 Model 590 are presented. A set of well-characterized Fortran benchmarks spanning a range of computational characteristics was used for the study. The data from the 590 system are compared with those from a single-processor CRAY C90 system as well as with other microprocessor-based systems, such as the Digital Equipment Corporation AXP 3000/500X and the Hewlett-Packard HP/735.

1 INTRODUCTION

The IBM RISC System/6000, first introduced in 1990, was an important step in the development of high-performance microprocessor systems. Using a *superscalar* architecture, the RS/6000 achieved performance on floating point intensive applications that exceeded most other microprocessor-based workstations and compared favorably with vector processors as well [1, 2].

Since 1990 there have been several new RS/6000 systems, all with the same design as the original ones and with incrementally faster central processing unit (CPU) clock speeds. Now IBM has developed a new RISC System, with a more advanced architecture. The performance of the RISC System/6000 model 590, as it is known, is the subject of this article. The performance of the 590 will be compared with the older model RISC System 6000/560, with two other contemporary high-performance RISC workstations, and with a high-end vector processor as well.

Received January 1994
Revised August 1994

e-mail: hjw@lanl.gov

Scientific Programming, Vol. 4, pp. 23–34 (1995)
CCC 1058-9244/95/010023-12

2 RISC SYSTEM/6000 MODEL 590 ARCHITECTURE

For brevity, model 590 is referred to as the RIOS-2 and the older system as the RIOS-1. *Superscalar* means that the RIOS-1 processor is capable of issuing four different instructions: a branch, a conditional, an integer operation, and a floating point operation each clock period (CP) [3]. Additionally, RIOS-1 has several floating point instructions that carry out multiplication and addition simultaneously. The most recent version of RIOS-1, called the RISC System/6000 model 580, runs at 62.5 MHz, which implies a theoretical peak performance of 125 MFLOPS. The most recent version of RIOS-1 installed at Los Alamos National Laboratory at the time of this writing is the model 560, which runs at 50 MHz and has a theoretical peak performance of 100 MFLOPS.

The RIOS-2 architecture has basically the same structure as the RIOS-1 but several key enhancements are included. The RIOS-2 processor chip set includes a second, independent, floating point pipeline and a second, independent, fixed point unit. The multiple issue rate has been increased so that RIOS-2 is capable of issuing six instructions per CP: a conditional, a branch, two integer, and two floating point instructions. With the multiply/add instructions this means a total of

four floating point operations (FLOPS) per CP. The clock speed of the RIOS-2 tested is 66.7 MHz. At four FLOPS per CP, this implies a theoretical peak rate of about 267 MFLOPS.

The memory subsystem of the RIOS-2 has also been improved to keep up with the increased demand caused by the second arithmetic pipeline. The data bus that feeds both floating point pipelines from the data cache is now four 64-bit words wide and the compiler can issue "load-quadword" instructions that will cause two 64-bit operands to be loaded from memory. Two such load-quad (or store-quad or a combination of both) instructions can be active simultaneously. All the RIOS-1 systems could transfer two 64-bit words from memory to the data cache each CP.

The RIOS-2 data cache can be either 128 Kbytes or 256 Kbytes, up from 32 or 64 Kbytes for the RIOS-1. Data are loaded into the cache from memory at the rate of four 64-bit words per clock cycle for the 128-Kbyte cache and eight 64-bit words per clock cycle for the 256-Kbyte cache. The data cache is still four-way set associative but the RIOS-2 uses a 256-byte line size.

The instruction length on the RIOS-2, like the RIOS-1, is 32 bits long. The functions of the branch processor remain the same, except that the instruction cache, from which the branch processor gets its instructions, is now 32 Kbytes, instead of 8 Kbytes as it was on the RIOS-1.

Each of the two floating point units in the RIOS-2 performs IEEE standard arithmetic and each has thirty-two 64-bit general purpose registers. There are thirty-two 32-bit general purpose registers in each of the fixed point units, which also support address translation and data protection.

The RIOS-2 can be equipped with up to 2 Gigabytes of main memory implemented on 16-Mbit chips. The machine tested had 512 Mbytes of memory. Other important differences between the RIOS-1 and RIOS-2 are floating point conversion to integer in hardware, floating point square root in hardware, and a new address translation mechanism.

3 COMPARISON WITH OTHER ARCHITECTURES

In later sections of this article the performance of the RIOS-2 will be compared with that of several other systems, and although detailed explanation of these other architectures is not possible here, a few important points are noted. The Digital Equipment Corp. (DEC) AXP 3000/500X workstation is based on the "Alpha" microprocessor running at 200 MHz [4]. The Alpha microprocessor is a full 64-bit implementation and is dual-issue superscalar, but it cannot overlap floating point additions and multiplies, so its theoretical peak performance is 200 MFLOPS. Alpha microprocessors contain an 8 KByte on-chip data cache with a 32-byte line size and the AXP 3000/500X we used also has a 512 KByte secondary cache. (There is no secondary cache in the IBM RISC System/6000.) A recent article [5] has examined in great detail the difference between the RIOS-1 and Alpha architectures; much of the discussion still applies to RIOS-2. The Hewlett Packard HP735 workstation, at least the second generation of the HP "Precision Architecture," is also superscalar and can overlap adds and multiplies [6]. It is currently available at a CPU speed of 99 MHz, which also corresponds to about 200 MFLOPS peak. The HP735 has a single, off-chip direct-mapped data cache with a capacity of 256 KBytes.

The CRAY C90 is the most recent version of the Cray Research Inc. line of parallel vector supercomputers [7]. It operates with a CP of 4.17 ns. In this report only single processor results are included, although the machine on which the benchmarks were run (Serial Number 4001, located at Eagan, MN) is a 16-CPU machine. Within each C90 CPU there are dual floating point pipelines (total capacity of four FLOPS/CP), although they operate differently than the two pipelines in the RIOS-2. The C90 issues a vector instruction for vector registers of maximum length 128 and the two pipelines operate on every other element of the vector(s). In the RIOS-2, two separate floating point instructions must be issued to use the dual pipe capability. All operations on the C90 are carried out in 64-bit precision.

The DEC, HP, and CRAY systems included in this article for comparison are all older than the RIOS-2, and it is expected that follow-on products to each will be made available soon.

3.1 Benchmark Codes

As with previous reports, this performance evaluation is based on a set of application benchmark codes that, in one way or another, represent a portion of the computational workload at the Los Alamos National Laboratory (LANL). Traditionally, this set of codes has comprised both full applica-

Table 1. Characteristics of the Benchmark Codes as Determined by the CRAY C90 Hardware Performance Monitor

CODE	Average MFLOPS	Average Hardware Vector Length	Percent Vector Operations
MCNP	11.6	12.4	0.2
TWODANT93	54.3	15.3	58.8
WAVE	77.5	66.3	63.0
TWODANT915	96.9	70.4	79.8
HYDRO	177.7	92.9	94.4
NEUT32	278.0	111.2	96.6
POP	362.1	122.9	96.8
PUEBLO32	458.4	119.9	98.2

tion codes as well as smaller and simpler codes containing selected basic routines or highly simplified versions of applications [8, 9]. However, this article concentrates largely on full applications as they might be run in production on traditional supercomputers. Much has been made recently of the potential for high-performance workstations to replace the conventional supercomputers in production environments [10]. Thus, it is important to compare performance of such systems on representative application codes as they might appear in those environments.

Such an approach has already been taken by other workers. For example, Hendricks and Briesmeister [11] have reported on workstation performance of LANL's MCNP code, a major production code widely used for simulation of particle transport. They reported that workstations compared very favorably with CRAY supercomputers. However, the computational characteristics of MCNP are such that it represents but one element of a spectrum of code characteristics and therefore gives a somewhat incomplete picture of relative supercomputer/workstation performance. In particular, MCNP is entirely nonvectorizable as written. In this article we consider MCNP along with a selection of codes that span a range of vectorizability as well as memory-access patterns. See Table 1 for some relevant statistics and the Appendix for brief descriptions of the benchmark codes.

4 RESULTS

4.1 Performance on Elementary Vector Operations

Table 2 lists performance of selected "vector" operations as a function of "vector" length on the RIOS-2. In Table 2, "R" and "V" represent array quantities and S1 and S2 are scalars. Using these relatively simple tests characteristics of the processor may be revealed that more complicated application benchmarks obscure. In theory these

Table 2. Rates (MFLOPS) for Selected Operations as a Function of Loop Length on the RIOS-2

	Loop Length			
Operation	10	25	100	300
R = S * V1	38.9	57.2	80.9	85.2
R = V1 + V2	33.1	46.1	67.5	73.8
R = V1 + S * V2	66.4	92.1	138.1	147.6
R = V1 * V2 + V3	53.1	72.0	95.4	99.0
R = S1 * V1 + S2 * V2	92.1	124.4	153.1	157.6
R = V1 * V2 + V3 * V4	74.7	90.3	109.9	113.0
R = V1 * V2 + V3 * V4 + V5	98.3	108.7	118.4	121.2
R = V1 * V2 + V3 * V4 + V5 * V6	89.6	104.9	119.0	122.3
R = V(IND) + S	21.4	26.7	27.6	27.8
R(IND) = V * V	17.4	25.9	29.2	29.9

tests seem straightforward and easy to carry out; in practice, they are among the most difficult. Typically we must defeat optimizations the compiler has introduced that remove, entirely, the code we wish to time. With the RIOS-2 use of the variety of compile-line options available caused significant variations in the results. For example, it was found that unrolling loops by a factor of four gave optimal performance for loops involving at most one FMA instruction (floating point multiply-add) but poorer performance for those involving more than one FMA. Conversely, unrolling by eight adversely affected single FMA operations but increased performance, often substantially, for multiple FMA operations. Thus, results for the first four operations of Table 2 were obtained using one set of compiler options and those in the lower portion were obtained using another.

It is important to note that these tests exercise the floating point processor/memory system while keeping data resident in cache, and thus they present an upper-bound estimate of processor performance. To obtain a measurable time for each loop, it is repeated until one million total operations are performed. In so doing, only the first load instruction unconditionally fetches from memory; the rest may load from cache. This is done to understand the maximum rate at which the CPU can produce results using compiled code.

As an example, consider the SAXPY operation (R = V1 + S * V2), which demonstrates the importance of loop unrolling on a multipipeline RISC architecture. Without unrolling, the IBM Fortran compiling system does not generate load-quadword instructions. The sequence of pseudoinstructions for floating point operations generated is shown in Table 3 where "double" refers to a 64-bit word. The instructions that occur on CP 0 are part of the loop start-up. This code sequence does not use the RIOS-2's second floating point pipeline. It produces one multiply/add result each two CPs, which corresponds to a rate of 66 MFLOPS. If SAXPY is timed on RIOS-2 without unrolling it about 65 MFLOPS are observed.

Now using a preprocessor directive the loop will be unrolled to a depth of two. The code sequence in this case is shown in Table 4. Now both floating point pipelines are active. In CP 1 two FMA operations occur simultaneously with load operations for four array elements to be used in the following iteration. Two multiply/add results (four FLOPS) are produced in two CPs corresponding to a rate of 133 MFLOPS. About 127 MFLOPS are observed. However, this code sequence does not fully utilize the memory bandwidth of the RIOS-2, because CP 2 only stores two words. Unrolling by eight yields a code sequence in which 8 FLOPS are produced in three CPs, for a predicted rate of 177 MFLOPS. However, only about 150 MFLOPS are observed. The disparity probably results from a combination of register spilling and insufficient store back buffer throughput.

For comparison purposes, Table 5 lists rates for selected operations from a single processor of the CRAY C90. There is not much difference between the C90 and RIOS-2 performance on loops with very short vector lengths; however, the observed rates for these lengths (especially on the C90) are subject to greater measurement errors because the loop times are on the same order as the timer overhead. Note also the additional column entry in Table 5, because C90 asymptotic vector performance is not fully reached by vector length 300, whereas this length is sufficient for asymptotic performance on the RIOS-2.

The C90's memory bandwidth is four loads and two stores per CP and the RIOS-2's bandwidth is four loads or four stores or two loads and

Table 3. Sequence of Pseudo Instructions

CP	Instruction	Array Elements Processed
0	Load Float Double	V1(1)
0	Load Float Double	V2(1)
LABEL		
1	Load Float Double	V1(2)
1	Multiply / Add	R(1)
2	Store Float Double	R(1)
2	Load Float Double	V2(2)
2	Branch to LABEL	

Table 4. Code Sequence

CP	Instruction	Array Elements Processed
0	Load Float Quad	V1(1), V1(2)
0	Load Float Quad	V2(1), V2(2)
LABEL		
1	Load Float Quad	V1(3), V1(4)
1	Multiply / Add	R(1)
1	Load Float Quad	V2(3), V2(4)
1	Multiply / Add	R(2)
2	Store Float Quad	R(1), R(2)
2	Branch to LABEL	

Table 5. Rates (MFLOPS) for Selected Operations as a Function of Vector Length on the CRAY C90

Operation	Loop Length				
	10	25	100	300	1000
R = S * V1	33.1	97.6	293.6	350.0	435.7
R = V1 + V2	31.2	88.8	270.1	335.8	380.8
R = V1 + S * V2	58.9	158.0	545.5	619.5	717.5
R = V1 * V2 + V3	52.0	109.6	333.3	342.0	412.7
R = S1 * V1 + S2 * V2	65.2	161.0	514.4	581.4	645.0
R = V1 * V2 + V3 * V4	62.8	162.1	498.5	537.9	594.9
R = V1 * V2 + V3 * V4 + V5	91.1	170.6	423.3	428.2	464.5
R = V1 * V2 + V3 * V4 + V5 * V6	84.2	188.8	484.5	522.4	585.8
R = V1 * V2 + V3 * V4 (I = 1,N,23)	67.4	154.0	376.5	445.9	518.1
R = V(IND) + S	22.5	49.9	117.6	160.0	181.6
R(IND) = V * V	26.0	54.7	90.8	124.4	133.1

two stores per CP. The ratio of C90 performance to RIOS-2 performance (at asymptotic vector lengths) is generally (but not always) greatest on operations with smaller compute intensities, i.e., smaller ratios of FLOPS to loads/stores.

The most significant performance difference between a vector processor such as the C90 and a cache-based architecture such as the RIOS-2 appears in operations in which memory is accessed noncontiguously. In Tables 2 and 5 this is shown for two operations using indirect addressing. On both machines rates for the gather/scatter operations are considerably lower than the corresponding contiguous operations, although the RIOS-2 degradation is worse than that of the CRAY's. More significant is the difference between the two machines on operations involving constant, nonunit stride through memory. For the RIOS-2, this is shown in Figure 1, a plot of rate versus vector length for the operation R = V1 * V2 + V3 * V4 for several strides. Processing rates for this operation on the RIOS-2 can degrade from about 70 MFLOPS to about 1–2 MFLOPS because of excessive data cache misses with strided access. The C90 shows no such degradation with odd strides (Table 5) because vector registers allow data to be loaded with any constant separation in memory without penalty.

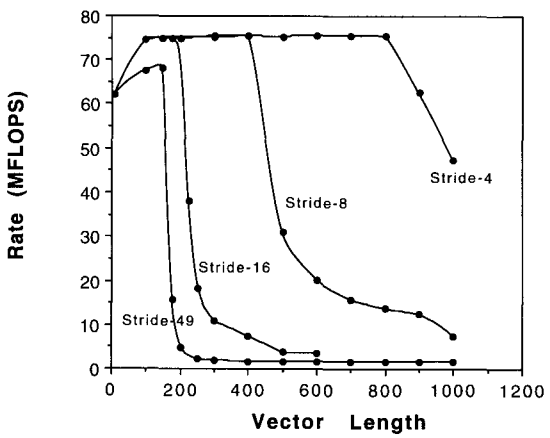

FIGURE 1 Plot of rate (MFLOPS) vs. vector length for the operation R(I) = V1(I) * V2(I) + V3(I) * V4(I), I = 1,N,ISTRIDE, for various values of ISTRIDE on the IBM RISC System/6000 model 590 (RIOS-2).

4.2 Results Using Application Benchmarks

Table 6 lists execution times for the application benchmark codes. Data were obtained in dedicated mode for the workstations but were obtained measuring CPU time (function SECOND) during regular timesharing conditions for the C90. The timer used on the RISC System/6000s is one that measures a real-time clock. For the HP and Alpha systems, the routine SECNDS was used, which measures user time. Compilers used were as follows: RIOS-1: xlf version 1.01; RIOS-2: xlf version 3.01; DEC: OSF/1 V1.3ARev 112 DEC Fortran X3.3(U2); HP: FTNOPP/HP9000 version 3.8; CRAY: cf77 version 6.0.0.4. Some of the DEC AXP results were obtained in February

Table 6. Benchmark Execution Times (seconds)

Code	C90	RIOS-560	RIOS-2	HP735	DEC AXP*	DEC AXP†
MCNP	48.8	127.6	46.1	56.3	51.0	—
TWODANT93	32.7	77.8	61.0	100.2	108.5	104.4
WAVE	34.5	172.0	63.9	122.5	153.9	148.6
TWODANT915	6.2	119.3	37.9	68.2	70.0	45.2
HYDRO	5.2	94.3	43.7	51.4	55.3	46.1
NEUT32	118.9	—	3786.8	5011.8	3222.0	2602.1
POP	8.9	263.6	78.8	228.7	186.1	148.6
PUEBLO32	4.8	114.7	52.5	93.2	104.8	83.4

* DEC 3000/500 (200-MHz, 512-Kbyte Bcache; DEC Fortran X3.3(U2).
† DEC 3000/800 (200-MHz, 2-Mbyte Bcache; DEC Fortran T3.4.

1993. All other data were obtained in August, September, and October 1993.

4.3 Comparison of RIOS-2 and RIOS-1

The 66.5-MHz RIOS-2 performance is compared with that of the 50-MHz RIOS-1, also known as RISC System/6000 model 560. Although the 560 is not the fastest existing RIOS-1 model, it is included because it is the type of processor currently used in a 16-processor cluster at LANL.

Figure 2 shows relative performance of the two machines; the ordinate indicates how much faster the RIOS-2 is on each code. Note that there is a factor of 1.3 that comes from the difference in the CPU speeds. It is interesting that all but one of the codes exceed this factor rather substantially. The exception is TWODANT93, which performs exactly as the clock ratio, and this relatively poor performance of the RIOS-2 seems to be compiler related. The RIOS-2 "xlf" compiler fails to generate any load-quadword instructions for the compute-intensive loops in three subroutines in TWODANT93, and the results from the simple vector operations above showed the importance of the load-quadword instruction in generating efficient RIOS-2 code. Generally, these loops involve fairly complicated array-index expressions.

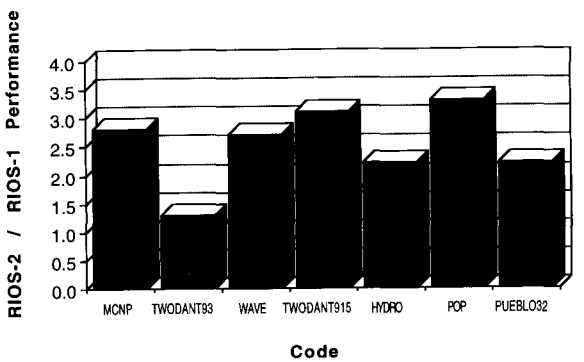

FIGURE 2 Relative performance of IBM RISC System/6000 model 590 and IBM RISC System/6000 model 560. The vertical axis shows how much faster the 590 is on each code.

On the other hand, the nonvectorizable code MCNP derives some benefit from the RIOS-2 architecture, even though it essentially has no loops to unroll. Other architectural differences between RIOS-1 and RIOS-2, such as faster instruction pipeline flush on mispredicted branches and higher memory bandwidth (two 64-bit words loaded or stored per pipeline as opposed to one word per pipeline on RIOS-1), contribute to MCNP performance.

4.4 Comparison with Other Workstations

Figures 3 and 4 present comparisons of RIOS-2 performance with that of the HP735 and DEC AXP 3000/500X, respectively. Again, the vertical axis in each figure indicates how much faster the RIOS-2 is on each code. Note that although both of these other workstations use processors whose CPU clock rate is 1.5 to 3 times faster than that of the RIOS-2, the RIOS-2 runs the application codes faster in the majority of cases. Additionally, although the peak speed of the RIOS-2 is about 1.3 times that of the AXP and the HP735, the RIOS-2 is more than 1.3 times faster on most of the codes. There is only one instance in which the RIOS-2's performance is exceeded: NEUT runs faster on the AXP system. Although it is difficult to reconcile the exact performance ratios for each code, the following points are noted.

All three machines provide nearly equivalent performance on the nonvectorizable MCNP code. The comparison of floating point operation times in Table 7 suggests comparable times on all three

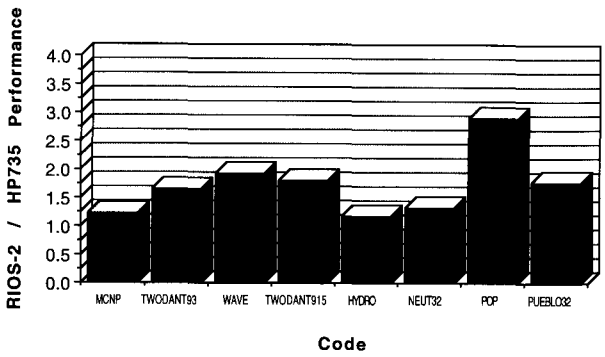

FIGURE 3 Relative performance of IBM RISC System/6000 model 590 and HP735. The vertical axis shows how much faster the 590 is on each code.

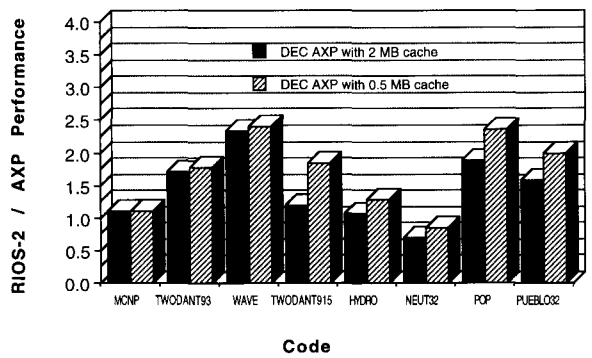

FIGURE 4 Relative performance of IBM RISC System/6000 model 590 and DEC AXP 3000/500X. The vertical axis shows how much faster the 590 is on each code.

machines for basic operations, although the RIOS-2 is significantly slower on divide operations and on loads involving cache misses. The times listed in Table 7 are operation latencies, i.e., start-up periods, after which, assuming favorable memory access conditions, results would appear at the rate of one per CP.

The RIOS-2's dual floating point pipeline architecture is optimized for repetitive array computation and so the RIOS-2 advantage over the other workstations is best demonstrated on codes that vectorize well. For example, the RIOS-2 is about 2.5 to 3 times faster than the other workstations on the highly vectorizable parallel ocean program (POP) ocean model benchmark.

A significant amount of time in the POP code [12, 13] is consumed in two routines that compute five-point and nine-point finite-difference "stencils." The five-point stencil may be expressed as follows:

$$
\begin{aligned}
R_{i,j} &= CC_{i,j} * X_{i,j} \\
&+ CN_{i,j} * X_{i,j+1} \\
&+ CS_{i,j} * X_{i,j-1} \\
&+ CE_{i,j} * X_{i+1,j} \\
&+ CW_{i,j} * X_{i-1,j}
\end{aligned}
$$

Using the same grid size as in the benchmark code the five- and nine-point stencils were timed separately on the RIOS-2 and HP735. The results are given in Table 8, with single-processor CRAY C90 data included for comparison. The stencil rates suggest why the RIOS-2 is significantly faster than the HP735 on the POP code. The rates obtained for the RIOS-2 and HP735 are both significantly below peak, but much more so for the HP735.

A significant difference between RIOS-2 and the Alpha processor is that the Alpha uses a secondary, "off-chip" cache to back up its small, primary data cache, whereas neither the HP735 nor the RIOS-2 employs a secondary cache. Thus, the two memory latencies listed in Table 7 for the AXP system are for data fetched from the primary and secondary caches, and for RIOS-2 they are for data fetched from the cache and from the main memory. The secondary cache helps mitigate the effect of large numbers of cache misses that might

Table 7. Comparison of Floating Point Operation Times

	Add	Multiply	Divide*	Load
AXP 3000/500X	6 CP	6 CP	61 CP	3 CP/7 CP†
	30 ns	30 ns	305ns	15 ns / 35 ns
HP735	2 CP	2 CP	15 CP	2 CP
	20 ns	20 ns	151.1ns	20 ns
RIOS-2	2 CP	2 CP	17–19 CP	1 CP / 12 CP†
	30 ns	30 ns	255–285 ns	15 ns / 180 ns
CRAY C90	6 CP	6 CP	10 CP	23 CP
	25 ns	25 ns	41.7 ns	96 ns

* The divide operation for the C90 is actually a reciprocal approximation.
† Cache hit/cache miss.

Table 8. Performance for Stencil Operations: Rates (in MFLOPS) and Percentages of Peak

Stencil	HP735 (%)	RIOS-2 (%)	C90 (%)
5-point	11.7 (6.9)	84.9 (32)	640.9 (64)
9-point	13.7 (6.9)	98.4 (37)	673.5 (67)

occur because of strided access to arrays (HYDRO is an example) or because of poor mapping of very large arrays to the primary data cache (NEUT is an example). HYDRO and NEUT are the two vectorizable codes on which DEC AXP performance is best relative to the RIOS-2.

Data from a more recent version of the AXP workstation containing the same 200-MHz Alpha processor with a larger secondary cache further substantiate the effect of cache misses on performance of vectorizable codes. The DEC model 3000/800, with a 2-Mbyte external cache, was announced in October 1993. With the larger cache, the AXP and RIOS-2 times are about equal for HYDRO and now the AXP is considerably faster than the RIOS-2 on NEUT.

4.5 Comparison with a Single Processor of the CRAY C90

Figure 5 presents a comparison of RIOS-2 performance with that of a single CRAY C90 processor, but now the vertical axis shows how much faster the C90 is on each code. The single C90 processor and the RIOS-2 yield equivalent performance on MCNP. Two other codes with low levels of vectorizability run faster on the C90 but the ratio of performance for these two codes is *less* than the ratio of the machines' CPs. The remaining codes run 6–30 times faster on the C90.

The relatively poor performance of the RIOS-2

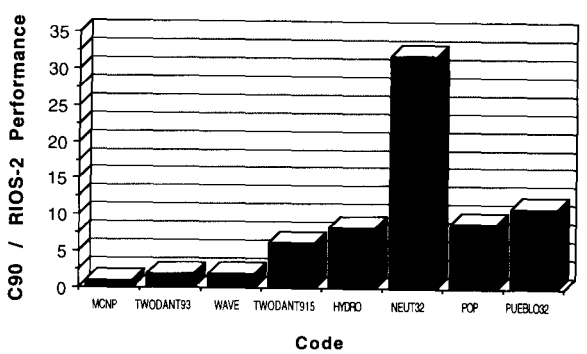

FIGURE 5 Relative performance of IBM RISC System/6000 model 590 and a single processor of the CRAY C90. The vertical axis shows how much faster the C90 is on each code.

on NEUT is most likely due to excessive cache misses on the RIOS-2. The time shown for NEUT on the RIOS-2 actually represents a tuned version of the code. NEUT was originally written for the Thinking Machines CM-2 and then translated from CM Fortran, which is similar to Fortran 90, into Fortran 77 [14]. The code vectorizes nearly completely, and the predominant vector length is about 32,000. On the RIOS-2, the code initially ran in 6666.6 seconds, nearly 60 times slower than the single C90 processor. The routine-level profile on the RIOS-2 showed the most time-consuming routine to be one entirely based on relatively straightforward, stride-1 computation. However, the four loops in this routine each calculate a series of array-based intermediates, each of which is stored to memory, through the cache, each loop iteration. By replacing these array intermediates with scalar temporaries, the time to run the code decreased by almost a factor of two, to 3787 seconds. In the restructured version of the code this same routine consumes 50% of the execution time. When measured separately on the RIOS-2, the loops of the type found in this routine run at about 100 MFLOPS at short vector lengths but performance degrades to about 8 MFLOPS at vector length 32769.

The results for POP [12, 13] also demonstrate the tuning process for codes ported from a vector architecture to a cache-based architecture such as the RIOS-2. POP was also written initially for the CM-2 and then translated into Fortran 77. In the CM Fortran code, longitude and latitude are distributed across processors and depth is an "in-processor" dimension. The CM Fortran compiler required that this in-processor array dimension be the first of the three dimensions for what is the majority of arrays in the code. When this code is translated to Fortran 77, inner loops running over latitude and longitude are thus nonunit stride loops. This version of POP, which is not shown in the tables in this article ran in 155 seconds, about 17 times slower than the single-processor CRAY C90 time. POP was then converted into an implementation in which the serial depth direction was the last of the three. Performance on the RIOS-2 increased by a factor of two, to the time shown in the tables. Of course, there is no difference in performance on the C90 between these two versions of the code because C90 performance of nonunit stride vectors at asymptotic vector lengths is the same as contiguous vector performance.

This converted version of POP still runs about nine times faster on the C90 than on the RIOS-2,

a factor much larger than the fourfold difference in the CPs of the two machines. The primary reason for this is still cache interference, even with the stride 1 version of the code. The stencil operations shown in Table 7 above run seven times faster on the C90, and this again is likely to be a conservative estimate because of the way the stencils are timed. The five-point stencil requires 11 arrays at about 32,000 words each, which just exceeds the RIOS-2 cache capacity.

Note that the benchmark version of POP included here uses a 255 X 127 grid. Actual production runs of POP on parallel machines such as the Thinking Machines CM-5 utilize much larger grids (e.g., 1024 X 512) [12]. The 255 X 127 grid problem uses about 430 Mbytes on the RIOS-2 and is therefore the largest POP problem that can be run without using virtual memory. Only one code requiring virtual memory on the RIOS-2 was timed. The larger version of PUEBLO (using a 64-cubed grid) ran in about 1400 seconds, compared with 52 seconds for the 32-cubed problem.

Finally, it is interesting to see how RISC System/6000 performance relative to CRAY processors has improved over the two generations of RIOS chip sets. Table 9, which lists relative CRAY Y-MP/RIOS-1 and CRAY C90/RIOS-2 performance, suggests that there has not been overwhelming improvement, although note that the benchmark data are subject to factors other than just processor differences. In particular, improvements in compiler technology on both systems probably play an important role.

5 DISCUSSION

The IBM RISC System/6000 model 590 (RIOS-2) processor includes several key architectural improvements over the older RIOS-1 systems. The RIOS-2 is a well-balanced pipelined processor that generally yields better performance on our application benchmarks than other high-performance microprocessor-based workstations. The superscalar nature of the RIOS-2 allows it to perform better than microprocessors operating at three times its CPU clock speed. Certainly, this shows why CPU clock speed alone is ineffective at estimating the performance of a given processor on real codes.

The design of microprocessors in the last two or three years seems to be proceeding along two rather different strategies, one that could be called "fast/simple," the other "slow(er)/complex." The DEC Alpha is an example of the former because it runs at very high clock speeds but does not allow concurrent execution of more than one floating point operation. The IBM RIOS is an example of the latter; its superscalar nature, i.e., its instruction-level parallelism, with a maximum of four FLOPS/CP, allows it to achieve high processing rates at relatively slower clock speeds. For the time being, the RIOS-2 approach seems to be best overall, although as shown above, the results are highly sensitive to cache effects. Again, White et al. [5] provide for a more detailed comparison of the RIOS-2 and Alpha architectures.

The RIOS-2, and, indeed, the other microprocessor-based workstations we tested, performed as well as a single processor of the CRAY C90 on an important Monte Carlo code that does not vectorize at all. On codes that are partially vectorizable, the C90 processor is faster, but the RIOS-2 is able to achieve performance that is greater than the ratio of the CPs of the two machines. This is probably due to lower overhead for short vector loops on the RIOS-2.

On highly vectorizable codes with long vector

Table 9. Relative Performance of Two Generations of IBM RISC System/6000 and CRAY Vector Processors*

Code	Performance of CRAY Y-MP/1 Relative to RIOS-1 (Model 560)	Performance of CRAY C90/1 Relative to RIOS-2 (Model 590)
MCNP	1	1
TWODANT93	1.5	1.9
WAVE	2.6	2.6
TWODANT915	10.0	6.1
HYDRO	7.8	8.4
POP	11.6	8.8
PUEBLO32	9.9	10.9

* Single processor data.

lengths, the single CRAY C90 processor can be 10 to 30 times faster than the IBM RIOS-2 processor. On such codes, the RIOS-2 experiences low overall memory throughput due to poor data cache reuse. All microprocessors we have tested suffer from this problem to one extent or another. Vectorizable codes with nonunit strides through memory, with random memory access (scatter/gather), or with uniform, stride-one access to very large arrays can reduce floating point performance on the RIOS-2 to nearly 10% of its optimal, cache-resident performance.

This article also suggested some of the changes that must be made in existing vectorizable code to better optimize for cache-based architectures. The results for NEUT suggest that in some cases, efforts devoted towards vectorization of codes over the last few years must now be put towards "dismantling" of vectorized code for optimal performance on cache-based systems. It has been reported that "stripmining" or "blocking" optimization techniques may be used to improve cache performance, and preprocessing tools now attempt to apply these procedures automatically. However, it appears at the present time that automatic restructuring may be limited to programs involving relatively simple loops, such as matrix multiplication. The benchmarks employed here contain very little of this.

The results from the DEC AXP systems also show how using a hierarchical cache system can improve data bandwidth. Presumably, users can expect that the size of secondary caches will continue to grow (at least in workstation-based systems; in massively parallel processing systems, which often use microprocessors as computational building blocks, this will not necessarily be the case). However, for some codes, even this multimegabyte cache is unable to supply data fast enough to the 5-ns CP CPU.

It is interesting to compare the relative performance of the C90 and the RIOS-2 by comparing their efficiencies, their performance relative to their respective peak (per-processor) rates. Such a comparison eliminates differences in CPU clock speed. To do this we assume that both machines are carrying out the same number of FLOPS for each code and we use the C90 hardware performance monitor to count the FLOPS. These results are shown in Table 10, where rates are given in MFLOPS. For the C90, the ratio of its maximum to minimum efficiency is about 30, and as expected, its efficiency varies directly with vectorization level (see Table 1). The RIOS-2 efficiency is not as predictable. It reaches a maximum for a moderately vectorizable code but is not much lower for a highly vectorizable, stride-1 code. Neglecting the result for NEUT32, its range is about an order of magnitude lower than the C90, suggesting that microprocessors provide a more consistent level of performance than vector processors over a range of vectorizability. For vectorization levels of about 65% or lower, the RIOS-2 achieves a higher percentage of peak performance than the C90. For one long vector code, the C90 achieves nearly 50% efficiency, more than twice the level of the RIOS-2 on this code. Finally, less than 2% efficiency is achieved on the RIOS-2 on NEUT because of poor data cache performance.

ACKNOWLEDGMENTS

The IBM RISC System/6000 model 590 on which these studies were performed was made available through a beta-test agreement between Los Alamos National Laboratory (LANL) and International Business Machines Corporation. We thank Shirley Grider and Tung Nguyen of IBM for instituting this agreement and for many helpful techni-

Table 10. Comparison of RIOS-2 and C90 Processing Efficiency

Code	Estimated RIOS-2 Rate (MFLOPS)	Observed C90 Rate (MFLOPS)	RIOS-2 % Peak	C90 % Peak
MCNP	12.3	11.6	4.6	1.5
TWODANT93	29.9	54.3	11.2	5.4
WAVE	53.8	77.5	20.2	7.7
TWODANT915	18.7	96.9	7.0	9.7
HYDRO	20.5	177.7	7.7	17.8
NEUT	4.6	278.1	1.7	27.8
POP	39.1	362.1	14.7	36.2
PUEBLO	43.0	458.4	16.2	45.8

cal discussions. We also thank Ann Hayes, Jerry Delapp, Dave Rich, and Andy White of the LANL Advanced Computing Laboratory for making generous availability of ACL resources. We thank Giao Dau and John Shakshober of Digital Equipment Corp. for making generous availability of AXP workstation time and for their own efforts in helping us run codes and explain the results. We thank Stan Barr of Hewlett Packard for his efforts in optimizing the codes for the HP735, Margaret Simmons of LANL for measuring the time for MCNP on the C90 and providing a detailed description of RIOS-2 architecture, and we thank Olaf Lubeck and Jim Moore (LANL) for many helpful discussions. Finally, we thank Mr. Samuel Kortas of LANL for providing us with his version of the POP code.

APPENDIX

Description of Benchmark Codes

POP: A global ocean model developed on the Thinking Machines Inc. CM-2 and translated into Fortran 77 [12, 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.

NEUT: A highly vectorizable Monte Carlo neutron transport code [14]. Two problem sizes may be run, one starting with 32K neutrons, the other with 64K neutrons. NEUT represents a Fortran 77 version of Eldon Linnebur's (LANL Group X-7) Connection Machine Fortran code.

HYDRO: A two-dimensional Lagrangian hydrodynamics code based on an algorithm by W. D. Schulz [15]. 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 timesteps. An important characteristic of the code is that most arrays are accessed with a stride equal to the length of the grid.

WAVE: A two-dimensional, relativistic, electromagnetic particle-in-cell simulation code used to study various plasma phenomena [16]. 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 run-time.

TWODANT: A two-dimensional discrete ordinates particle transport code used for neutral particle transport [17]. 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 source multiplication tests. TWODANT915 runs a "k-calc" computation and TWODANT93 runs a source multiplication for a fixed value of k.

MCNP: A general purpose Monte Carlo particle transport code widely used at LANL and elsewhere [18]. 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 5000 source particles.

PUEBLO: A three-dimensional Lagrangian hydrodynamics code used to model point explosions in space [19]. 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.

REFERENCES

[1] 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, March 1990.

[2] M. L. Simmons and H. J. Wasserman, *Proceedings of Supercomputing '90.* Los Alamitos, CA: IEEE Computer Society Press, 1990, pp. 142–152.

[3] IBM RISC System/6000 Technology, International Business Machines Corp Book SA23-2619, IBM, Austin, TX, 1990.

[4] R. Sites, ed. *Alpha Architecture Reference Manual.* Burlington, MA: Digital Equipment Corp. Press, 1992, Document No. EY-L520E-DP.

[5] S. W. White, P. D. Hester, J. W. Kemp, and G. J. McWilliams, "How does processor MHz relate to end-user performance?" *IEEE Micro,* pp. 8–16, 1993.

[6] M. Forsyth, S. Magelsdorf, E. DeLano, C. Gleason, and J. Yetter, *Compcon Spring '91 Digest of Papers.* Los Alamitos, CA: IEEE Computer Society.