Commercial workload performance in the IBM POWER2 RISC System/6000 processor

by M. T. Franklin
W. P. Alexander
R. Jauhari
A. M. G. Maynard
B. R. Olszewski

We describe features of the POWER2™ processor and memory subsystem that enhance RISC System/6000® performance on commercial workloads. We explain the performance characteristics of commercial workloads and some of the common benchmarks used to measure them. Our own analysis methods are also described.

Introduction

Data-intensive applications such as transaction processing and file servers form a major market segment for computer systems. These applications are collectively called "commercial" applications, as most were initially used primarily by commercial enterprises such as banks, airlines, and insurance companies. Early microprocessor-based systems simply could not handle such data-intensive applications because they did not have enough processing power and I/O connectivity to accommodate large numbers of data processing users concurrently. In recent years, the dramatic increase in the processing power available from microprocessors has made it possible for systems such as RISC-based workstations to compete in markets that were once exclusively the domain of mainframes. In fact, commercial applications represent one of the most rapidly growing market segments for RISC-based UNIX® workstations today [1].

In this paper, we examine some of the features of the POWER2™-based RISC System/6000® that make it suitable for the commercial workload arena. We begin by describing the performance characteristics of commercial workloads. We then discuss our analysis methodology and some of the common benchmarks used by the industry to compare commercial performance. This is followed by a discussion of the POWER2 hardware features that contribute to the enhanced commercial performance of the RISC System/6000.

Characteristics of commercial workloads

Commercial workloads include a wide variety of applications; some of the more prominent include on-line transaction processing, other database management

Table 1 Characteristics of commercial benchmarks.

Benchmark	*Branches* (%)	*Branches taken* (%)	*Average sequential block size* (instructions)	*Percentage of instructions in OS*
TPC-A da	18.8	66.4	8.0	67
TPC-A cs	16.9	67.1	8.8	40
TPC-C	18.9	65.1	8.1	43
LADDIS	18.9	68.7	7.7	100
KENBUS	16.3	64.9	9.5	23
SDET	17.8	66.5	8.4	50

services such as batch transaction processing and decision support, and file servers.

While different commercial applications stress the system in different ways, most share some common characteristics. These include

- Many (up to hundreds or thousands of) concurrent users.
- Long path lengths over a large set of instructions, with a substantial part of the path length in the operating system code.
- Fewer repetitive loops and more non-loop-closing branches than in scientific applications (see [2] for an analysis of the branching characteristics of some scientific benchmarks).
- Extensive manipulations of data structures via pointers, requiring integer arithmetic for address resolution.
- Relatively little floating-point arithmetic; data manipulation consists primarily of string or integer comparison, updates, or insertions.
- High random I/O rates, with data spread over many megabytes or gigabytes of disk; disk I/Os are primarily short (4 or 8 KB), and successive disk I/Os are often randomly distributed over the total disk space used.

Note that the characteristics of commercial applications differ dramatically from those of scientific or numerically intensive programs, which often have small instruction working sets with tight loops, use floating-point arithmetic extensively, and often do sequential rather than random I/O.

Table 1 shows some characteristics of certain commercial benchmarks. The benchmarks themselves, as well as the methodology used to determine these characteristics, are discussed in the next section.

The first two numeric columns show the percentage of instructions executed that are branches, and of these the percent that are taken. An analysis of branching behavior helps to explain one of the reasons why the I-cache miss rates for commercial workloads are higher than those for scientific applications. Consider a typical scientific workload dominated by short-to-medium-length loops. For such a workload, where most branch instructions are executed to return to the head of a loop, the percentage of branches that are taken would be much higher than the percentages shown in Table 1. Also, if the largest instruction loop for this workload fits in the I-cache, the I-cache miss rate for the scientific application would be very low. In contrast, the branching data in Table 1 show that, for commercial applications, the percentage of taken branches is relatively low, indicating that there are relatively few loops executed in these applications. This scarcity of loops implies that the application has a lower probability of re-executing recent instructions present in the I-cache, leading to a higher I-cache miss rate. Other characteristics of commercial applications that lead to high I-cache miss rates include large numbers of processes and a high context switch rate. The number of processes is often proportional to the number of concurrent users; frequent context switches are the result of frequent short I/Os and interprocess synchronization. Commercial applications tend to have higher data cache (D-cache) miss rates as well, because their data exhibit less locality and sequentiality than scientific applications.

The column labeled *Average sequential block size* in Table 1 shows in a different way the tendency of commercial workloads to cause branching. In this paper we define the term "sequential block" to be the sequence of instructions executed between two taken branches (including the second branch instruction). (This is a dynamic measure, not something obtainable from a static analysis of the code.) These sequential block sizes are much shorter than in the scientific and engineering applications we have analyzed.

The last column in Table 1 shows the percentage of the total number of instructions executed in the AIX® operating system code, as opposed to the instructions executed by the user application and in shared library code. These numbers indicate that much of the work in commercial applications is actually done by the operating system. One reason for this relatively high usage of the operating system code is that, in these applications, there is frequent movement of small amounts of data between different levels of the system, and little arithmetic computation on the data. (For example, queries must be sent from an on-line user's terminal to a database server, and responses must be sent back; this involves the use of a lot of operating system communication code, and very little arithmetic.) This is in contrast to many scientific applications, where once data are brought to the application space by the operating system, extensive arithmetic manipulation is performed before the data are handed back to the operating system for storage. Operating system code typically has a high incidence of branches and few loops.

Clearly, architectural features such as the number of integer arithmetic units, the sizes, organization, and

number of levels of instruction and data caches, and the latency to caches and to memory, all have a strong bearing on the degree to which commercial applications perform well on a system. The I/O subsystem has an important effect on the performance of most applications, but in this paper we focus on processors and memory subsystems, and I/O is not discussed further.

Until recently, many RISC-based UNIX workstations focused on the scientific application environment, but as this paper and others in this publication [3] show, the POWER2-based workstation is designed to provide superior performance in both scientific and commercial environments.

Performance methodology and benchmarks

The results presented in this paper come from one of three sources: direct measurement on POWER2-based systems, analysis of traces taken on POWER-based systems, or simulations using these traces.

• *POWER2 performance measurement*

The POWER2 processor has a built-in hardware performance monitoring capability that collects measurements which allow not only detailed performance analysis, but also detailed workload characterization and analysis of system behavior [4].

• *Traces and simulation*

The AIX performance group has a tool that collects instruction traces. These traces record the sequence of instructions executed (for both system code and application code) along with the virtual address of each instruction and data reference. The traces can be postprocessed to reveal workload characteristics such as those shown in Table 1. They can also drive simulators of the processor and memory subsystem. From the simulations we can determine many of the same system performance characteristics as from hardware measurements, including cache miss rates.

Determining these quantities from a software trace plus simulations is less accurate than direct hardware measurements, but it has the advantage of applying to system configurations that do not actually exist; for example, we can create a trace once and use it to investigate many different cache configurations. It also allowed us to begin analyzing POWER2 performance before the processor was actually available, and to analyze workload characteristics on systems that do not have hardware counters.

We have validated our trace-based simulation methodology against direct hardware measurements on both POWER- and POWER2-based systems, and the results almost always agree to within 10%.

• *Commercial benchmarks*

In the intensely competitive marketplace of open systems, performance comparisons provide a key differentiator in making purchasing decisions.

While some customers have developed their own, often proprietary, benchmarks to help them choose among different systems, several benchmarks have been accepted by the industry as good indicators of the performance capabilities of systems aimed at different parts of the commercial market. These include the set of TPC benchmarks for database applications, the LADDIS benchmark for file server applications, and SPEC™ SDM benchmarks such as SDET and KENBUS for multi-user software development applications. The traces used in this paper are of these benchmark programs.

The TPC benchmarks

The Transaction Processing Performance Council (TPC) benchmarks are probably the best known and most frequently cited measures of commercial performance. The TPC is a collection of representatives from many of the computer systems vendors and several of the database vendors competing in the commercial arena throughout the world. At the time of writing, the TPC has released three benchmarks, called TPC-A, TPC-B, and TPC-C, along with rules for running them and reporting results that are intended to produce fair comparisons among possibly dissimilar commercial computing systems.

TPC-A simulates a banking application where users submit simple debit or credit transactions to the system from automated teller machines. (TPC-B is logically very similar to TPC-A; it differs mainly in that it is easier and cheaper to configure a system to run TPC-B than to configure it for TPC-A. We do not discuss TPC-B further in this paper.)

The trace called TPC-A da in this paper was collected while the TPC-A benchmark was running in direct-attach mode using a commercially available relational database management system (RDBMS) on a RISC System/6000 Model 530. In the direct-attach mode, the workstation executes the entire TPC-A benchmark code, including the user interface, the application code, and the database server.

The trace called TPC-A cs was collected while the TPC-A benchmark was running in client-server mode using a different commercially available RDBMS on a RISC System/6000 Model 530 as a server. In the client-server mode, the user interface part of the TPC-A benchmark runs on one or more client machines, while the rest runs on a server. Since TPC-A cs captures only server code, it provides a different view of the workload than the TPC-A da trace, and the two traces are not directly comparable; we include both in our analysis because both modes of operation are fairly common in the marketplace.

Table 2 TPC-C instructions by unit.

Execution unit	*Percentage of instruction mix*
FXU	78.0
ICU	21.8
FPU	0.2

TPC-A transactions are relatively short and uniform. In fact, TPC-A transactions are so simple and so short that this benchmark is becoming obsolete as a measure of high-end commercial performance; the demand from industry for more realistic and more complex benchmarks has led to the development of TPC-C. TPC-C is intended to simulate an order-entry environment, with a mix of read-only and update-intensive transactions.

The TPC-C trace was collected on the server portion of the TPC-C benchmark in a client-server configuration running on yet another type of commercially available RDBMS on a RISC System/6000 Model 550.

The LADDIS benchmark

The LADDIS benchmark measures a system's file serving capability [5]. One or more client workstations submit a mix of Network File Server (NFS)™ requests to the server. LADDIS is the basis of a new system-level file server (SFS) benchmark suite recently announced by the Standard Performance Evaluation Corporation (SPEC) [6]. The LADDIS trace was collected on a RISC System/6000 Model 530H.

The KENBUS and SDET benchmarks

KENBUS and SDET are two of the benchmarks that make up the SPEC SDM (System Development Multitasking) suite [7]. Both KENBUS and SDET represent multi-user software development environments with large numbers of concurrent users executing common UNIX commands such as edits, compiles, and binds. KENBUS represents a UNIX/C research and development environment, while SDET represents a C-based commercial software development environment. The KENBUS and SDET traces were collected on a RISC System/6000 Model 530.

System performance

The performance metric that is published for each of the benchmarks described in the previous section is a measure of throughput: transactions per second or minute for the TPC benchmarks and LADDIS, and scripts per hour for the KENBUS and SDET benchmarks. Systems being tested against these benchmarks are always configured so that the processor is the bottleneck by, for example, adding enough disks to ensure that this is so. In these configurations, the latencies for I/O are entirely overlapped by the processing of instructions, and the processor is fully utilized, while I/O devices are not. I/O performance and system response time are important considerations, but are beyond the scope of this paper.

When the processor is the bottleneck, system throughput is determined by three factors: the average number of instructions (NI) in the path length for the transaction or operation, the average number of cycles it takes to execute an instruction (CPI, or "cycles per instruction") for that benchmark, and the number of processor cycles per second (usually expressed in megaHertz, or MHz). Throughput (TP) in transactions or operations per second is expressible as a function of these three variables as

$$\mathrm{TP} = \frac{\mathrm{MHz} \times 1{,}000{,}000(\mathrm{cycles/s})}{\mathrm{NI(instr/trans)} \times \mathrm{CPI(cycles/instr)}}.$$

Operating system and application developers can improve throughput by reducing the average path length. MHz is influenced by processor size, design, implementation technology, and cost. In the next section, we describe features of POWER2 that are aimed at raising throughput by reducing CPI.

POWER2 features for high performance

Three features of the POWER2 design improve its performance on commercial workloads compared to that of the POWER processor:

- Second integer instruction unit.
- Larger caches.
- Longer cache lines.

We now discuss the effect of each of these features in more detail.

- *Second integer instruction unit*

The POWER2 processor is superscalar; i.e., it is capable of executing more than one instruction per processor cycle. As described in [3], the POWER2 has two floating-point instruction execution units (FPUs), two fixed-point (integer) instruction execution units (FXUs), and an instruction cache unit (ICU) that handles branch instructions. The POWER2 processor is capable of executing up to six instructions simultaneously in one cycle. In practice, the number of instructions that can be executed in any given cycle is constrained by which instructions are actually present in the section of code being executed, and by data dependencies among the instructions.

Multiple floating-point instruction units, which allow multiple floating-point instructions to be executed simultaneously, are fairly common in processors aimed at

Table 3 L1 cache miss rates for three cache configurations.

Workload	*Configuration*	*POWER2*		*Hypothetical A*		*Hypothetical B*	
	size type	*32KB I*	*256KB D*	*16KB I*	*16KB D*	*8KB I*	*8KB D*
	assoc/ln sz	*2/128*	*4/256*	*4/64*	*4/64*	*2/32*	*2/32*
TPC-A da		2.4	0.5	4.5	2.1	9.0	3.5
TPC-A cs		4.0	1.2	6.9	4.1	12.7	6.7
TPC-C		2.7	0.4	5.2	2.6	10.3	4.6
LADDIS		2.5	0.4	5.0	3.5	9.9	6.6
KENBUS		0.9	0.4	1.7	1.6	4.4	3.1
SDET		1.5	0.5	2.7	1.8	6.2	3.3

scientific/engineering workloads. However, commercial workloads have relatively few floating-point instructions; **Table 2** shows POWER2 hardware measurements of the fractions of all instructions executed by the ICU, the FXUs, and the FPUs for the TPC-C benchmark, run under the same RDBMS type as that used to produce our TPC-C trace.

Our instrumentation also indicates that almost all of the ICU instructions are branch instructions. Taken together, the data in Tables 1 and 2 show that most instructions in commercial applications are executed by integer units; about 15–20% of the instructions executed are branches, and only a small fraction of these are floating-point instructions.

Measurements taken on the POWER2 processor running TPC-C show that of all the FXU instructions executed, one FXU executed 68%, while the second FXU executed the remaining 32%. Generally, instructions are routed to the second FXU only if there is also an instruction to dispatch to the first FXU; this implies that the presence of the second FXU resulted in a reduction of up to 32% in execution time for the integer instructions. Analysis of other hardware measurements confirms that, in fact, the second FXU was busy 48% of the time that the first FXU was busy. Put another way, the presence of the second FXU allows the POWER2 to increase integer instruction execution throughput for TPC-C by a factor of about one half. Note that the theoretical maximum performance improvement (a doubling of the integer instruction execution throughput by two equally utilized FXUs) is difficult to attain because of dependencies between instructions, which require one unit to hold off execution until the other completes the preceding operation. (The exact speedup provided by the second FXU also depends on how many of the instructions executed by it could have been scheduled in cycles when the first FXU would otherwise have been idle. As integer instructions account for about three quarters of the total instructions, the presence of the second FXU leads to a significant performance gain for the POWER2 processor.

- *Larger caches*

The POWER2 processor has separate level 1 (L1) instruction and data caches. The I-cache holds 32 KB and is two-way set-associative, with a line size of 128 bytes. The D-cache is 256 KB, four-way, and has a 256-byte line. The original POWER processor had an 8KB I-cache and a 64KB D-cache.

The larger caches mean fewer cache misses. In particular, they increase the execution speed of commercial workloads, which tend to have larger cache footprints than scientific/engineering workloads.

To show the effect of cache size and geometry, we compare the POWER2 L1 cache configuration with two smaller, hypothetical configurations as detailed in **Table 3**. The cache size, set associativity, and line size in bytes of the instruction and data caches are given for each of the three configurations.

For each configuration, Table 3 shows the cache miss rates as predicted by our model, using the benchmark traces described earlier. Miss rates are expressed as percentages, normalized to total instructions issued:

$$\text{miss rate} = 100 \times \frac{\text{number of misses}}{\text{total instructions}}.$$

As Table 3 shows, the larger caches reduce misses significantly, thereby speeding execution of these benchmarks.

- *Longer cache lines*

The POWER2 also has longer L1 cache line sizes than the POWER. (A "line" is the unit of transfer between memory and caches.) The I-cache has 128-byte lines, and the D-cache has 256-byte lines. Longer lines help performance in two ways.

First, longer lines increase the probability that instructions in a sequential block or sequential data references will fall within the same line rather than crossing a line boundary into a different line. When two consecutive references lie in the same cache line, the

Table 4 I-cache miss/branch characteristics with different line sizes.

Workload	*Average sequential block size* (instructions)	*Total I-cache miss rate* (%)		*Fraction of misses to first byte of line* (%)		*Branches to same line* (%)	
		32-byte line size	*128-byte line size*	*32-byte line size*	*128-byte line size*	*32-byte line size*	*128-byte line size*
TPC-A da	8.0	5.0	2.4	65.3	38.9	13.8	34.5
TPC-A cs	8.8	9.5	4.0	67.1	40.6	15.1	38.3
TPC-C	8.1	6.0	2.7	67.0	41.9	15.1	36.3
LADDIS	7.7	5.6	2.5	67.4	42.4	13.7	38.0
KENBUS	9.5	1.8	0.9	69.9	41.9	14.4	38.5
SDET	8.4	3.1	1.5	68.3	41.0	13.8	36.5

second reference is almost always found in the cache; the probability of generating a cache miss increases when consecutive references lie in different lines.

We used our traces to drive simulations of two I-cache geometries: Both I-caches held 32K total bytes, but one had a 32-byte line and the other a 128-byte line (with, of course, one fourth as many lines). We counted the fraction of I-cache misses that were references to the first byte of the new line; many of these misses are caused by sequential blocks that happened to cross line boundaries. Note that I-cache misses caused by references to all bytes of a line other than the first are caused by taken branches, and cannot be attributed to the fact that sequential blocks sometimes cross line boundaries.

Table 4 shows the average sequential block size for each workload repeated from Table 1; the total I-cache miss rate expressed as a percentage of instructions executed; of these, the percent of misses that were references to the first byte of a line; and the percentage of branches taken that were to a location within the same cache line.

Table 4 shows that longer cache lines reduce both I-cache misses to the first byte of a line and total misses; not all misses are due to sequential blocks crossing line boundaries, of course.

The second benefit of longer I-cache lines shown in Table 4 is that they increase the probability that a branch will be to an instruction that is present in the same cache line. Recall that the two I-cache configurations were the same size, 32 KB; only the line size was changed.

As can be seen, longer I-cache lines significantly reduce the fraction of branches that land outside the current line, thereby helping to reduce total cache misses as shown in Table 4. This effect should especially benefit commercial workloads, which have a high incidence of branches.

While our measurements show that they reduce cache misses, longer cache lines have the disadvantage of taking longer to fill than shorter lines. To help alleviate this effect, the POWER2 has a memory bus that is 8 words (256 bits) wide.

Summary

Commercial applications form an increasingly larger segment of the market for microprocessor-based systems such as the RISC System/6000. Commercial workloads differ substantially from scientific and engineering workloads in their demands on the system. In this paper we describe these differences and, using data obtained from widely accepted commercial benchmarks, explain how some of the features of the POWER2 processor and memory subsystem make it well suited for high performance in the commercial arena.

Acknowledgments

We would like to thank several members of the AIX and Systems Performance groups for their help in putting this paper together. We appreciate the assistance of Bob Urquhart in analyzing cache performance and Marc Stephenson in helping to get database systems running. Chris Chan-Nui developed invaluable postprocessing tools for reducing hardware monitor data, and Ed Welbon helped us properly interpret the POWER2 hardware measurements.

References

1. "A Major Step for Commercial UNIX," *AIXpert,* November 1992; Order No. G580-0012; available through IBM branch offices.

2. C. Stephens, B. Cogswell, J. Heinlein, G. Palmer, and J. P. Shen, "Instruction Level Profiling and Evaluation of the IBM RS/6000," *Proceedings of the 18th Annual International Symposium on Computer Architecture*, May 1991; IEEE Catalog No. 91CH2995-9, pp. 180–189.
3. S. W. White and S. Dhawan, "POWER2: Next Generation of the RISC System/6000 Family," *IBM J. Res. Develop.* **38,** 493–502 (1994, this issue).
4. E. H. Welbon, C. C. Chan-Nui, D. J. Shippy, and D. A. Hicks, "The POWER2 Performance Monitor," *IBM J. Res. Develop.* **38,** 545–554 (1994, this issue).
5. "LADDIS File Server Benchmark," *SPEC Newsletter* **4,** No. 1 (March 1992); Standard Performance Evaluation Corporation, Fairfax, VA.
6. "SPEC Announces System File Server Benchmark Suite—SFS Release 1.0," *SPEC Newsletter* **5,** No. 1 (March 1993); Standard Performance Evaluation Corporation, Fairfax, VA.
7. "SPEC SDM Technical Fact Sheet," *SPEC Newsletter* **3,** No. 2 (Spring 1991); Standard Performance Evaluation Corporation, Fairfax, VA.

Received August 19, 1993; accepted for publication March 22, 1994

Maurice T. Franklin *IBM RISC System/6000 Division, 11400 Burnet Road, Austin, Texas 78758 (MAURICE at AUSTIN, maurice@austin.ibm.com).* Mr. Franklin is a Staff Programmer in the Systems Architecture and Performance group at IBM Austin. He received his B.A. degree in computer sciences from The University of Texas at Austin in 1987. He then served four years as an Air Force officer, developing UNIX communications software. He joined IBM in 1991 in his current position, analyzing the performance characteristics of commercial workloads, especially database benchmarks, on the IBM RISC System/6000 system.

William P. Alexander *IBM RISC System/6000 Division, 11400 Burnet Road, Austin, Texas 78758 (BILLA at AUSTIN, wpa@perfmap.austin.ibm.com).* Dr. Alexander is a Senior Programmer in the AIX System Performance group at IBM Austin. He received the B.A. degree in philosophy from Rice University and the M.A. and Ph.D. degrees in computer sciences from the University of Texas at Austin. Dr. Alexander was an Assistant Professor at Boston University and worked at the Los Alamos National Laboratory and the Microelectronics and Computer Technology Corporation before joining IBM in 1991 in his present position. His interests include performance measurement and modeling and distributed systems. He is a member of ACM.

Rajiv Jauhari *IBM RISC System/6000 Division, 11400 Burnet Road, Austin, Texas 78758 (RAJIV at AUSTIN, jauhari@austin.ibm.com).* Dr. Jauhari is an Advisory Programmer in the AIX System Performance group at IBM Austin. He received his B.Tech. in electrical engineering from the Indian Institute of Technology, Delhi, in 1985 and his Ph.D. in computer science from the University of Wisconsin–Madison in 1990. Since 1990, he has worked in the AIX Performance area in IBM Austin. His primary interest is system performance modeling and analysis. He is a member of IEEE and ACM.

Ann Marie Grizzaffi Maynard *IBM RISC System/6000 Division, 11400 Burnet Road, Austin, Texas 78758 (ANNMARIE at AUSTIN, amg@austin.ibm.com).* Dr. Maynard received her B.S. in electrical engineering at Polytechnic Institute of New York, and her M.S. and Ph.D. in electrical and computer engineering from Carnegie Mellon University. She has worked on several research projects relating to the design and performance analysis of high-performance and multiprocessor systems at AT&T Bell Laboratories in Murray Hill, New Jersey. At NASA Langley Research Center, she worked on the validation of SIFT, a highly reliable multiprocessor system for aircraft control. Dr. Maynard currently works as an Advisory Engineer in the AIX System Performance group at IBM Austin. Her current research includes the performance analysis of future memory subsystems. She is a member of ACM.

Bret R. Olszewski *IBM RISC System/6000 Division, 11400 Burnet Road, Austin, Texas 78758 (BRETO at AUSVM6, breto@perfmap.austin.ibm.com).* Mr. Olszewski is an Advisory Programmer in the AIX System Performance group at IBM Austin. He received the B.S. degree in computer science from the University of Minnesota in 1982. Since 1989, he has worked in the AIX Performance area at IBM Austin. His current research includes hardware and software design of future systems.