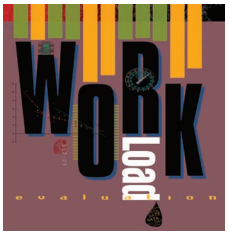
Benchmarking Internet Servers on Superscalar Machines



The authors compared three popular Internet server benchmarks with a suite of CPU-intensive benchmarks to evaluate the impact of front-end and middle-tier servers on modern microprocessor architectures.

Yue Luo
Juan Rubio
Lizy Kurian John
University of Texas at Austin

Pattabi Seshadri
Alex Mericas
IBM

Today's demanding Internet applications typically execute on superscalar microprocessors that fetch, decode, and execute multiple instructions in each clock cycle. These microprocessors contain multiple functional units, generally employ large caches, and tend to execute instructions in an order different from the instruction sequence fed to them.

To finish the job as soon as possible, they look deep into the instruction stream and execute instructions from places where sequential execution flow has not yet reached. With the aid of sophisticated branch predictors, they identify the program flow's potential path to find instructions that can be executed in advance. At times, however, a processor makes wrong predictions and must nullify the extra work it performed speculatively.

Developers designed most of the microprocessors that execute today's Internet applications before the advent of these emerging workloads. CPU-intensive benchmarks have been widely used in processor performance evaluation but differ in functionality from emerging commercial applications, which contain several Web server and e-commerce software packages, interfaces, and standards. An end-to-end e-business transaction typically involves at least a dozen different software modules, including the front end or portal, shopping carts, network communication, credit card or electronic check processing, and security.

Many of these applications involve a Web-based interface to an underlying database that stores the data relating to the user inquiry or transaction. Modern servers use a three-tier approach in which the back-end tier handles the database access while the middle tiers and front end implement much of the business logic and user interface. Some researchers have studied large database applications, which are usually used as an Internet server back end.[1-4] Although these studies have revealed much about the back end, we still do not fully understand how the front and middle tiers of server-side workloads behave.

To help fill that knowledge gap, we examined how the front-end and middle-tier workloads of Web and Java servers have affected modern microprocessor architectures. We also compared a number of server benchmarks with CPU-intensive benchmarks, such as the Standard Performance Evaluation Corporation's SPECint2000, to get a perspective on their behavior compared to more traditional and better-understood workloads.

SUPERSCALAR PLATFORMS

We evaluated Internet workloads on three significantly different microarchitectures. The IBM RS64-III[5] is a 64-bit, superscalar, in-order speculative-execution processor, while the IBM POWER3-II[6] is a 64-bit, superscalar, out-of-order speculative-

Published by the IEEE Computer Society
0018-9162/03/$17.00 © 2003 IEEE

execution processor. Both have reduced-instruction-set computing (RISC) architectures. The Intel Pentium III is an out-of-order superscalar processor with a complex-instruction-set computing (CISC) instruction set, but it converts CISC-style instructions into simple RISC-style micro-operations before execution.

The IBM processors retire up to four instructions per cycle, while the Pentium III can retire three micro-operations per cycle. The POWER3-II and Pentium III employ dynamic branch prediction; the RS64-III prefetches up to eight instructions from the branch target into a branch target buffer during normal execution, predicts the branch not taken, continues to fetch from the instruction stream, then—once the branch is resolved—either continues fetching from the current instruction stream with no penalty or flushes the instructions after the branch occurs.

The RS64-III has 128-Kbyte level-one instruction and data caches and a 4-Mbyte level-two cache. The POWER3-II has 64-Kbyte L1 instruction and data caches and 8-Mbyte L2 caches. The Pentium III has 16-Kbyte L1 instruction and data caches and 512-Kbyte L2 caches.

The Pentium III we used ran Windows NT Workstation 4.0, while the IBM platforms ran AIX 4.3.3. Our experiments used Apache 1.3.23 as the Web server. We used the IBM Java Development Kit 1.1.8 and Sun JDK 1.3.0 with HotSpot Server as the Java virtual machines on the IBM and Intel platforms, respectively.

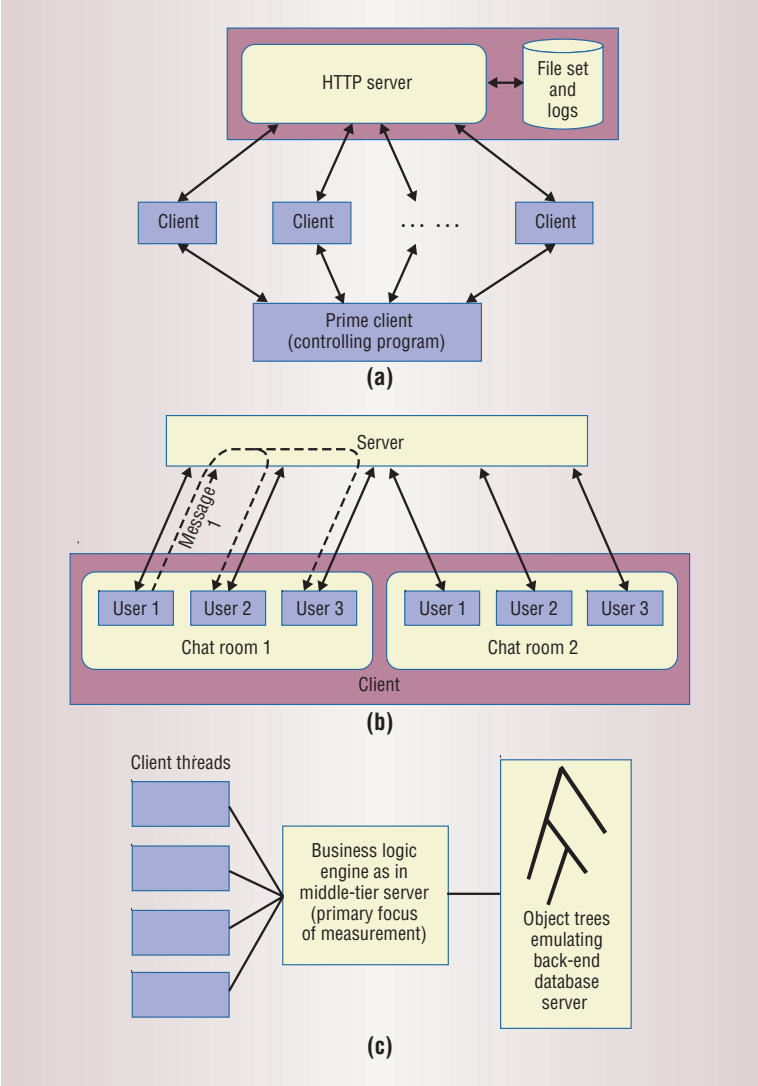
EXPERIMENTAL ENVIRONMENT

Using these three platforms, we ran three server benchmarks that represent various front-end and middle-tier Internet services: SPECweb99, VolanoMark 2.1.2, and SPECjbb2000. To compare them to traditional and better-understood applications, we also experimented with the integer programs in the SPEC CPU2000 suite (www.spec.org).

Web server

SPECweb99 (www.spec.org/web99/) provides SPEC's current benchmark for measuring Web server performance. The SPECweb99 workload, shown in Figure 1a, simulates accesses to an Internet service provider, wherein the server supports Web pages for several different organizations.

Each homepage in the benchmark contains a collection of files that range from small icons to large documents and images. The workload simulates dynamic operations such as rotating advertisements on a Web page and customized Web page creation.



The file accesses closely match today's real-world Web server access patterns. SPECweb99 supports the HTTP1.0 and HTTP1.1 protocols.

Internet chat server

VolanoMark (www.volano.com/benchmarks.html), a pure Java chat server benchmark, has long-lasting network connections and high thread counts. It is based on VolanoChat, a commercial chat server application deployed in several countries. Although provided as a single package, it can be divided into a server and a client. The server accepts connections from the chat client, as shown in Figure 1b. This client simulates many chat rooms, with multiple users in each room. The client continuously sends messages to the server and waits for the server to broadcast them to users in the same chat room. The VolanoMark server creates two threads for each client connection.

E-commerce application

SPECjbb2000 (www.spec.org/jbb2000/), SPEC's first e-business benchmark, also provides the first

Figure 1. Three server benchmarks. (a) SPECweb99 models a Web server, (b) VolanoMark models an Internet chat server, and (c) SPECjbb2000 models an electronic commerce application. Both VolanoMark and SPECjbb2000 are written in Java.

On-Chip Performance Monitoring Counters

Most high-performance microprocessors incorporate on-chip performance monitoring counters. Developers can use these counters to understand how microprocessors perform when running complex, real-world workloads.

Previously, workload analysis used simulators and profilers, which either ignored operating system activity or resulted in prohibitively long slowdowns. Using on-chip performance counters now makes it possible to evaluate and monitor complex runtime systems involving multiple software applications at execution speed. Performing this type of measurement on binaries helps developers analyze the behavior of proprietary software for which no source code is available.

Various performance events can be measured using counters. Usually, they include

- total cycles,
- instructions fetched,
- instructions retired,
- number of loads and stores,
- cache misses at various levels,
- number of branch instructions,
- number of mispredicted branches, and
- number of branches not predicted.[1]

Counters can be read with special instructions and configured to measure user and kernel activity in combination or isolation. For design simplicity, most microprocessors severely limit the number of events that can be measured simultaneously. At times, certain events can be accessed only through a particular counter. Microprocessor developers have found such steps to reduce the overhead associated with on-chip performance monitoring. Performance counters consume on-chip real estate and, if not carefully implemented, can detrimentally affect processor cycle time.

Among the many available software tools to access performance counters are VTune (developer.intel.com/software/products/vtune/vtune60/vtune_oview.htm) and PMON (www.ece.utexas.edu/projects/ece/lca/pmon) on Windows/x86, the Compaq Continuous Profiling Infrastructure (www.research.compaq.com/SRC/ dcpi/) on Alpha, the perf-monitor utility (www.sics.se/~mch/perf-monitor/index.html) on UltraSparc, Brink/Abyss (www.eg.bucknell.edu/~bsprunt/emon/brink_abyss/brink_abyss.shtm) on Pentium 4, and Rabbit for Intel and AMD processor on Linux systems (www.scl.ameslab.gov/Projects/Rabbit).

Developers using these counters must be sure to have as few undesired processes running during an experiment as possible to minimize the interference from these processes.

Reference

1. L.K. John, "Performance Evaluation: Techniques, Tools and Benchmarks," *The Computer Engineering Handbook*, V.G. Oklobdzija, ed., CRC Press, 2001, pp. 8-21 to 8-37.

benchmark for evaluating the performance of server-side Java. The Java Business Benchmark (JBB) emulates an electronic commerce workload in a three-tier system. It contains business logic and object manipulation, primarily representing the activities of the middle tier in an actual business server.

The benchmark models a wholesale company with warehouses that serve several districts. Customers initiate a set of operations, such as placing new orders and checking the status of existing orders. The benchmark generates additional operations within the company, such as processing orders for delivery, entering customer payments, and checking stock levels.

Written in Java, SPECjbb2000 is an adaptation of IBM's portable business-oriented benchmark, pBOB. Although it emulates business transactions, SPECjbb2000 differs significantly from the Transaction Processing Council benchmarks (www.tpc.org). For example, as shown in Figure 1c, it substitutes driver threads for actual clients. Similarly, instead of providing actual database access, it stores data as binary trees of objects. Memory-resident and thus without inherent disk I/O, SPECjbb2000 assigns one active customer per warehouse, each of which involves 25 Mbytes of data.

Hardware performance monitors

We used the hardware performance monitors built into the microprocessors to measure processor performance. The "On-Chip Performance Monitoring Counters" sidebar describes performance monitors in detail. For the IBM machines, we used the IBM-supplied performance monitor API and IBM's pmcount—both AIX kernel extensions—to interface with the PowerPC performance monitors. On the Intel processor, we used PMON to access these counters. We measured the server only, and only when it was handling client requests, thus excluding startup and shutdown operations.

WORKLOAD COMPARISON

We compared the three server benchmarks with SPECint on the three different microarchitectures. Table 1 shows the results from VolanoMark runs with 10 and 30 chat rooms as volano10 and volano30, and SPECjbb runs with 10 and 25 warehouses as SPECjbb10 and SPECjbb25. For VolanoMark, we used the default configuration of 20 users per room.

Throughout, we strove to make the metrics common across all three platforms. However, not all the platforms can monitor all the microarchitecture events. Therefore, we compared some related metrics when identical metrics could not be collected from all three. Further, the differences in the processors and platforms forced us to focus on the performance differences between server and traditional CPU-intensive benchmarks on each platform, rather than comparing the platforms directly.

As Table 1 shows, on all three platforms the server applications generally exhibit higher cycles per instruction than most SPECint programs. CPI reflects the execution efficiency of the processor pipelines and their ability to extract instruction-level parallelism. Our results indicate that the platforms have more difficulty exploiting ILP in the server benchmarks than in the SPECint benchmarks. On the IBM platforms, we found that for more than half the execution cycles in the server applications the system dispatched no instructions. Similarly, no operations can be retired in more than 60 percent of the Pentium III's execution cycles.[7]

Operating system activity

Database and file servers reportedly devote a higher percentage of their execution time to the privileged operating system (kernel) mode than technical workloads do,[8] an observation we confirmed for Web servers as well, as Table 1 shows. The SPECweb and VolanoMark programs spend 30 to 65 percent of their execution cycles in OS mode. In contrast, most programs in the SPECint2000 suite spend negligible time in this mode.

Processors typically enter the OS mode when the user application invokes a system call requesting the operating system to perform some task on its behalf, such as creating another process, requesting synchronization with another kernel-level thread, or sending a packet to the network. VolanoMark spends most of its time receiving and sending network messages, which is mainly the OS's task. Also, to handle simultaneous client connections, a server usually spawns multiple threads. Scheduling and synchronizing these threads, in addition to network communication, result in VolanoMark spending more than half its execution time in OS mode.

Network communication is a major part of SPECweb. In addition, its execution incurs many disk accesses. SPECjbb, on the other hand, lacks the network communications and disk accesses typical of servers. It thus does a poor job of representing server applications in this respect. For example, it spends less than 0.7 percent of the total execution time in OS mode, which is little different from SPECint.

Cache performance

Modern processors devote much of their real estate to on-chip caches designed to capture the instruction and data working sets and reduce the average memory access time.

L1 instruction cache. Figure 2 shows the L1 instruction cache misses per 1,000 instructions for the various workloads. The server applications exhibited poorer instruction cache performance than SPECint programs on all three machines. The instruction translation lookaside buffer (ITLB) misses also exhibited a similar trend.[7]

The use of dynamically linked libraries (DLLs) strongly influences servers' instruction access behavior in emerging applications. To streamline the development of complex software, developers now adhere to the principle of modularity at both the source code and binary levels. Most Web server functions are implemented as DLL modules on the Windows platform.

For example, consider the Apache server in the SPECweb benchmark. This server has a main exe-

Table 1. Workload cycles per instruction and percentage of cycles spent in OS mode.

Benchmark*	Cycles per instruction			Average kernel cycle percentage on all three platforms
	RS64-III	POWER3-II	Pentium III	
SPECweb	1.45	1.19	2.10	33.53
volano30	1.76	1.44	3.03	55.50
volano10	2.17	1.59	3.72	60.85
SPECjbb25	1.52	1.27	2.31	0.46
SPECjbb10	1.45	1.25	2.29	0.48
vortex	1.45	0.64	1.27	0.59
twolf	1.41	1.23	2.25	0.24
gcc	1.07	0.78	2.25	0.92
eon	1.27	1.04	1.36	0.23
crafty	0.77	0.63	1.22	0.20
perlbmk	1.16	0.85	1.13	0.53
parser	1.04	0.93	1.64	0.26
gap	1.19	0.82	1.32	0.38
bzip2	0.98	0.97	1.36	0.75
vpr	1.32	1.29	1.79	0.28
mcf	4.66	3.08	6.65	0.36
gzip	0.80	0.68	1.24	0.67

*Server benchmarks appear in roman type, SPECint2000 benchmarks in italic.

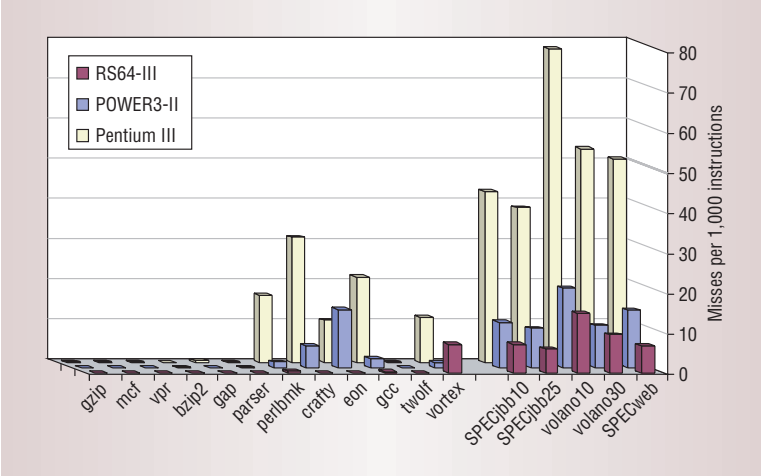


Figure 2. Level-one instruction cache misses per 1,000 instructions, for various workloads. The server applications exhibited poorer instruction cache performance than SPECint programs on all three machines.

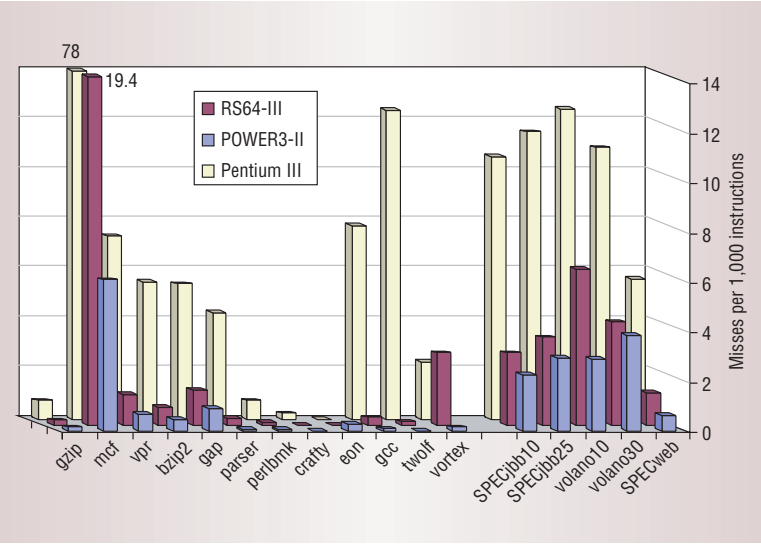


Figure 3. Level-two cache misses per 1,000 instructions, for various workloads. The high server miss rates on the IBM platforms indicate that the data footprint of server applications is hard to capture with ordinarily large caches.

cutable of only about 20 Kbytes, and its core functions—implemented in the ApacheCore.dll—comprise 320 Kbytes. The SPECint programs, on the other hand, are compiled into self-contained standalone executables. In addition, SPECint programs seldom request services from the operating system or other applications, except for reading some input files at the beginning and printing the results at the end. Thus, few DLLs become involved in the execution of SPECint programs.

The dynamic invocation and loading of libraries affects the server program's instruction footprint and access nature. Because each shared library loads to a different memory page, calling a function in another library causes the control flow to transfer to another memory page, resulting in poor ITLB performance. Similar behavior has been observed in Windows desktop applications.[9]

Further, many server applications or components are written in Java and compiled by just-in-time compilers at runtime. This dynamically compiled code for consecutively invoked methods may not occupy contiguous address spaces.[10] All of these effects explain the difference in instruction access behavior between server applications and the SPECint suite.

High instruction cache miss rates have been observed in traditional database server applications as well.[1,2]

L1 data cache. The miss rates in the level-one data cache do not differ significantly enough between the server and the SPECint applications to warrant presentation here.

L2 cache. Figure 3 shows level-two cache misses per 1,000 instructions. On the Pentium III, the server applications' L2 cache misses are comparable to SPECint's. On the PowerPC machines, which have much larger L2 caches, the servers' L2 cache miss rates are higher than SPECint's, except for *mcf*, which has extraordinarily high L2 cache miss rates on all platforms.

The high server miss rates on the IBM platforms, even with relatively large L2 caches, indicate that the data footprint of server applications is hard to capture with ordinarily large caches. Servers usually manage large data sets, as in the case of SPECjbb, in which each warehouse holds 25 Mbytes of data. Because each client thread accesses a different warehouse, the large number of clients creates a large and scattered data footprint.

Branch predictor behavior

State-of-the-art high-performance microprocessors employ speculative execution to enhance performance. The Pentium III and POWER3-II employ dynamic branch prediction to predict branch direction and the branch target at runtime based on execution history. Although branch misprediction rates are similar for both server and SPECint workloads on the Pentium III, branch target buffer misses run higher for server applications than for SPECint2000. The POWER3-II behaves similarly, although the difference between SPECint and server branch prediction performance is less pronounced than on the Pentium III.

Further, one SPECint program, *eon,* that shows poor target prediction performance is written in C++ and makes heavy use of virtual functions, a common feature of object-oriented programming. Java programs have been seen to demonstrate poor branch target predictability, often because of indirect branches that result from virtual function calls and code interpretation.[11]

WHERE DO THE CYCLES GO?

We used the processor performance counters to shed more light on the various CPI components and their individual contributions. Given the differences in the processors' counters, we used slightly different approaches.

Pentium III

On the Pentium III, we measured cycles with instruction stream stalls and resource stalls. Instruction stream stalls are mainly the result of instruction cache misses and ITLB misses. Resource stalls indicate the number of cycles in which resources such as reorder buffer entries, memory buffer entries, or execution units are unavailable.[12]

We then estimated the computation CPI using a tested methodology.[3] As Figure 4a shows, the server applications and SPECint programs experienced resource stalls in similar ranges, while the former's instruction stream stalls are much higher than the latter's.

Determining the contribution of each stall component to the CPI is difficult, especially in an out-of-order superscalar processor like the Pentium III, which tolerates latencies well by overlapping them. Therefore, we conducted a linear regression analysis and statistically isolated the contributions of resource and instruction stream stalls to the CPI.

We found that for server applications, instruction stream stalls contribute from 33 to 62 percent to the CPI, while for SPECint such stalls contribute less than 33 percent overall and for half the suite are negligible—less than 1 percent. Further analysis showed that resource stalls highly correlate with data cache misses.

RS64-III

Because the POWER3-II performance monitor lacks the stall events necessary to perform a detailed CPI component analysis, we explored only the RS64-III's CPI components. The RS64-III offers a single countable event that indicates the nonoverlapped total amount of storage-related stalls—in effect, it counts multiple storage-related stalls in one cycle as a single stall. Figure 4b compares CPI components of the server and SPECint benchmarks on the RS64-III.

To put these results in perspective, we first estimated an approximate *ideal CPI*—one with a perfect memory system—using the measured overall CPI and storage-related stalls. We then constructed a CPI stack with this ideal CPI and major storage-stall components such as instruction cache, data cache, and L2 cache stalls.

Our results clearly show that the server benchmarks incur significantly more storage-related stalls than the SPECint benchmarks. Storage-related stalls also have a high impact on CPI in the server programs and a few of the SPEC programs, but only a negligible effect on half of the SPECint programs. Despite the large number of storage stall cycles for the server benchmarks, their CPIs are lower than the sum total of the CPI components, which indicates the effectiveness of the RS64-III's pipelined architecture in hiding some of the storage latency.

Given that the RS64-III instruction cache is eight times as large as the Pentium III's, we can expect instruction stream stalls to be a less serious contributor to overall CPI in the RS64-III. The contrast in these two platforms' performance suggests that server applications can benefit from large instruction caches. L2 cache miss stalls constitute a major stall component of CPI on the RS64-III.

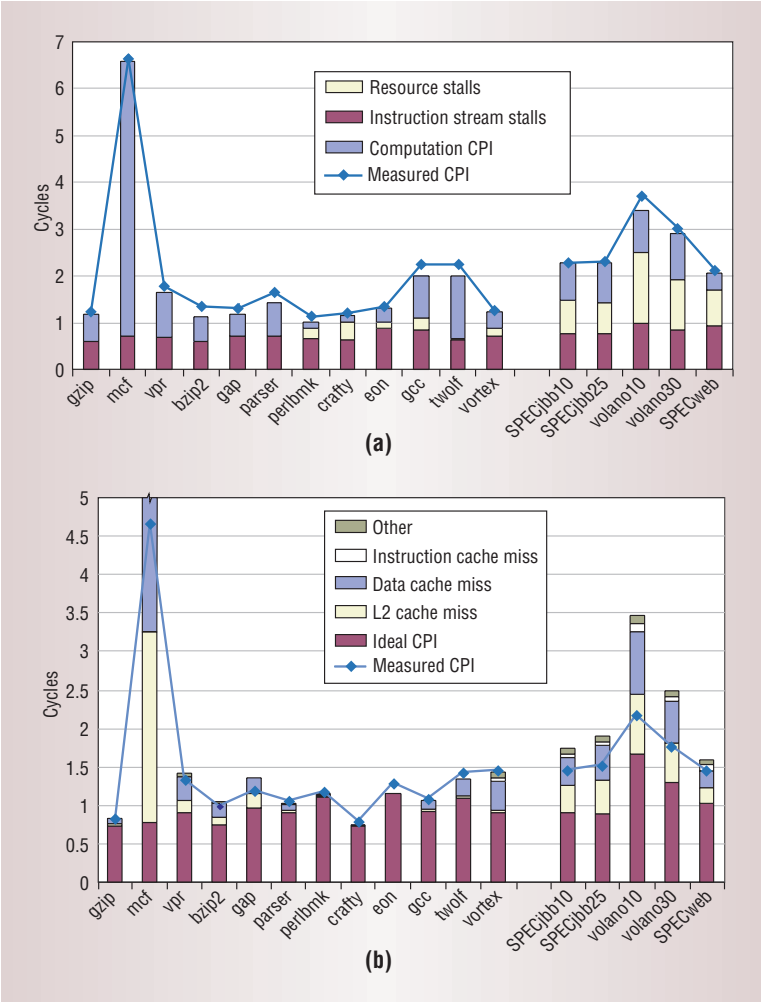


Figure 4. Comparing cycles per instruction between server benchmarks and SPECint on different processors. (a) CPI stack for the Pentium III, detailing the major CPI components. (b) RS64-III CPI components for the server and SPECint benchmarks.

Our study demonstrates that many of the benchmarking results hold across all three architectures, irrespective of their different designs. This leaves no doubt that, to maximize performance on Internet server applications, modern processor architectures need further enhancements and optimizations, particularly in memory system design. Increased memory requirements coupled with interacting software packages will lead to more complex memory access stream behavior for instruction and data memory access. While increased cache sizes may help, concerns about large memory access times, increased wire delays, and increased power consumption could arise. Designing microprocessors for emerging Internet servers will continue to be a challenge. ■

Acknowledgments

This research is supported in part by the National Science Foundation under grants 9807112 and 0113105; the Defense Advanced Research Projects Agency under contract F33615-01-C-1892; the IBM Center for Advanced Studies (CAS) project; and by the IBM, Intel, Tivoli, and Microsoft corporations.

References

1. A. Ailamaki et al., "DBMSs on a Modern Processor: Where Does the Time Go?" *Proc. 25th Int'l Conf. Very Large Databases* (VLDB 99), Morgan Kaufmann, 1999, pp. 15-26.
2. L.A. Barroso, K. Gharachorloo, and E. Bugnion, "Memory System Characterization of Commercial Workloads," *Proc. 25th Ann. Int'l Symp. Computer Architecture* (ISCA 98), IEEE CS Press, 1998, pp. 3-14.
3. K. Keeton et al., "Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads," *Proc. 25th Ann. Int'l Symp. Computer Architecture* (ISCA 98), IEEE CS Press, 1998, pp. 15-26.
4. P. Ranganathan et al., "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors," *Proc. 8th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 98), ACM Press, 1998, pp. 144-156.
5. J.M. Borkenhagen et al., "A Multithreaded PowerPC Processor for Commercial Servers," *IBM J. Research and Development*, vol. 44, no. 6, 2000, pp. 885-894.
6. F.P. O'Connell and S.W. White, "POWER3: The Next Generation of PowerPC Processors," *IBM J. Research and Development*, vol. 44, no. 6, 2000, pp. 873-884.
7. Y. Luo et al., *A Case Study of 3 Internet Benchmarks on 3 Superscalar Machines,* tech. report, Laboratory for Computer Architecture, University of Texas at Austin, 2001; www.ece.utexas.edu/projects/ece/lca/ps/UT_LCA_TR-020817-01.pdf.
8. A.M.G. Maynard, C.M. Donnelly, and B.R. Olszewski, "Contrasting Characteristics and Cache Performance of Technical and Multi-user Commercial Workloads," *Proc. 6th Int'l Conf. Architectural Support for Programming Languages and Operating Systems* (ASPLOS 94), ACM Press, 1994, pp. 145-156.
9. D.C. Lee et al., "Execution Characteristics of Desktop Applications on Windows NT," *Proc. 25th Ann. Int'l Symp. Computer Architecture* (ISCA 98), IEEE CS Press, 1998, pp. 27-38.
10. R. Radhakrishnan et al., "Architectural Issues in Java Runtime Systems," *Proc. 6th Int'l Conf. High-Performance Computer Architecture* (HPCA 00), IEEE CS Press, 2000, pp. 387-398.
11. T. Li and L. John, "Understanding Control Flow Transfer and its Predictability in Java Processing," *Proc. 2001 IEEE Int'l Symp. Performance Analysis of Systems and Software* (ISPASS 01), IEEE Press, 2001, pp. 65-76.
12. D. Bhandarkar and J. Ding, "Performance Characterization of the Pentium Pro Processor," *Proc. 3rd Int'l Symp. High-Performance Computer Architecture* (HPCA 97), IEEE CS Press, 1997, pp. 288-297.

Yue Luo *is a PhD student in the Electrical and Computer Engineering Department at the University of Texas at Austin. His research interests include microprocessor modeling, computer architecture simulation, and performance analysis. Luo received an MS in electronics from Peking University. He is a student member of the IEEE. Contact him at luo@ece.utexas.edu.*

Juan Rubio *is a doctoral candidate in the Electrical and Computer Engineering Department at the University of Texas at Austin. His research interests include microprocessor and computer system architecture and operating systems. Rubio received an MS in electrical and computer engineering from the University of Texas at Austin. He is a student member of the IEEE Computer Society and the ACM. Contact him at jrubio@ece.utexas.edu.*

Lizy Kurian John *is an associate professor in the Department of Electrical and Computer Engineering at the University of Texas at Austin. Her research interests include computer architecture, microprocessor design, performance evaluation, workload characterization, reconfigurable architectures, the performance impact of contemporary programming paradigms, and compiler support for innovative architectures. John received a PhD in computer engineering from Pennsylvania State University. She is a member of the IEEE Computer Society, the IEEE, and the ACM. Contact her at ljohn@ece.utexas.edu.*

Pattabi Seshadri *is a hardware performance engineer at IBM. His research interests include workload characterization, novel microprocessor architectures, and memory systems. Seshadri received a BS in computer engineering from the University of Texas at Austin. Contact him at p_seshadri@us.ibm.com.*

Alex Mericas *is a senior engineer at IBM. His research interests include workload characterization, performance verification, and hardware performance monitors. Mericas received an MCE from National Technological University. Contact him at mericas@us.ibm.com.*