A Case Study of 3 Internet Benchmarks on 3 Superscalar Machines

Yue Luo[+], Pattabi Seshadri[*], Juan Rubio[+], Lizy Kurian John[+] and Alex Mericas[*]

[+]*Laboratory for Computer Architecture*
The University of Texas at Austin
Austin, TX 78712
{luo,jrubio,ljohn}@ece.utexas.edu

[*]*IBM Corporation*
11400 Burnet Road
Austin, TX 78758
mericas@us.ibm.com

The phenomenal growth of the World Wide Web has resulted in the emergence and popularity of several information technology related computer applications. These applications are typically executed on computer systems that contain state-of-the-art superscalar microprocessors. Superscalar microprocessors can fetch, decode, and execute multiple instructions in each clock cycle. They contain multiple functional units and generally employ large caches. Most of these superscalar processors execute instructions in an order different from the instruction sequence that is fed to them. In order to finish the job as soon as possible, they look further down into the instruction stream and execute instructions from places where sequential execution flow has not reached yet. With the aid of sophisticated branch predictors, they identify the potential path of program flow in order to find instructions to execute in advance. At times, the predictions are wrong and the processor nullifies the extra work that it speculatively performed.

Most of the microprocessors that are executing today's internet workloads were designed before the advent of these emerging workloads. The *SPEC CPU* benchmarks (See sidebar on *SPEC CPU* benchmarks) have been used widely in performance evaluation during the last 12 years, but they are different in functionality from the emerging commercial applications. Whether the difference in functionality results in key differences in the exploitation of architectural and microarchitectural features of the processor, is the subject of this article. In order to answer this question one has to identify appropriate workloads and obtain performance metrics that indicate the execution characteristics.

Emerging workloads contain several software packages, interfaces and standards that were triggered by the proliferation of web servers and the arrival of electronic commerce. An end-to-end e-business transaction typically involves a dozen or more different software layers, including the front end/portal, shopping carts, network communication, credit card or electronic check transaction, security software layers, etc. Literally all enterprises including airlines, banks, stock brokerage firms, and most consumer product vendors nowadays use their web servers to deal with a significant part of their business. Many of these applications involve a web based interface to an underlying database that stores the data relating to the user enquiry or transaction. Modern servers use a 3-tier approach in which the backend tier handles the database accessing and the front end and the middle tiers implement much of the user interface and portals. Some researchers have studied large database applications, which are usually used as the backend of Internet servers [1, 2, 5, 12]. While these studies have revealed much about the behavior of backend applications, the behavior of the front and middle tiers of server side workloads is still not fully understood.

We attempt with this study to fill some of that knowledge gap by characterizing the impact of the front and middle tier of web servers and Java servers on modern processor microarchitectures. We also compare server benchmarks with CPU intensive benchmarks such as *SPECint2000* to get a perspective on their behavior in comparison to more "traditional" and better understood workloads.

Platforms

We gathered data on three drastically different platforms: the IBM RS64-III (an in-order superscalar RISC machine), the IBM POWER3-II (an out-of-order superscalar RISC machine), and the Intel Pentium III (an out-of-order superscalar CISC machine). They are all current microarchitectures, but they differ in many significant ways.

The **RS64-III** [4] is a 64-bit, superscalar, in order, speculative execution processor and is targeted specifically for commercial applications. It has one single cycle integer unit, one multiple cycle integer unit, one four stage pipelined floating point unit, one branch unit, and one load/store unit. The RS64-III can fetch, dispatch, and retire up to four instructions per cycle and has a five stage pipeline. It does not predict branches dynamically like the POWER3-II, but rather 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 and then, once the branch is resolved in the dispatch stage, either continues fetching from the current instruction stream with no penalty or flushes the instructions after the branch and begins fetching from the branch target buffer, with a penalty of at most one and often zero cycles. The RS64-III has a 128KB, two way set associative L1 instruction cache, a 128KB, two way set associative data cache, and a 4MB, four way set associative unified L2. It also has a 512 entry four way set associative unified TLB and a 64 entry instruction effective to real address translation buffer (IERAT) that allows fast address translation without the use of the TLB. . The RS64-III system we use is the IBM M80, configured as a uniprocessor system. The system has

2 GB of main memory.

The **POWER3-I**I [9] is a 64-bit, superscalar, out of order, speculative execution processor. It has two single cycle integer units, one multiple cycle integer unit, one branch/condition register unit, two load/store units, and two three stage pipelined floating point units. It can fetch, dispatch, and retire up to four instructions in the same cycle. It has a 256-entry branch target address cache (BTAC), which works like a branch target buffer, and a 2048 entry, 2 bits per entry branch history table for dynamic branch prediction. The POWER3-II has a 64KB, 128-way set associative, four way interleaved L1 instruction cache, a 64KB, 128 way set associative, four way interleaved L1 data cache, and a 8MB, four way set associative unified off-chip L2. It also has a 256-entry two way set associative instruction TLB and two 256 entry two way set associative data TLBs. The POWER3-II is designed with separate buses to memory and L2 for greater memory bandwidth. The POWER3-II also employs a data prefetching mechanism that detects sequential data access patterns and prefetches cache lines to match these patterns. . For our experiments, we use the IBM 44p-170, which contains the POWER3-II processor and 2 GB of main memory.

The **Pentium III** processor is an out-of-order superscalar CISC processor. It first converts the CISC-style instructions into simple RISC style operations called uops. The Pentium III is capable of issuing up to 5 uops and retiring up to 3 uops in one cycle. The processor has a 40 entry reorder buffer to facilitate retirement of instructions in order. The processor employs speculative execution using a two level branch predictor and a 512-entry branch target buffer (BTB). The processor has a separate L1 data cache and L1 instruction cache. Each cache is 16 Kbytes in size, 4-way set associative with a 32-byte block size and employs an LRU replacement algorithm. The data cache is write-allocate, non-blocking and dual-ported. The processor also has a unified 512KB 4-way set associative non-blocking L2 cache with a 32-byte block size. In our study, we use a Dell Precision 410 PC with one Pentium III processor and 1 GB of physical memory.

Experimental Environment

The IBM systems run AIX 4.3.3 and the IBM JDK version 1.18. We use Apache 1.3.23 as our web server. Admittedly, it is not the highest performance web server, but it is very popular. According to a Netcraft survey (http://www.netcraft.com), about 55% of all web sites on the Internet are using Apache. The operating system on the Pentium III system under measurement is Windows NT Workstation 4.0 with Service Pack 6a. We use the Sun JDK 1.3.0 with Hotspot

Server (build 2.0fcs-E, mixed mode) as the Java virtual machine. The Web server is Apache 1.3.23 for Windows with mod_perl and ActivePerl 5.6.1.

Three server benchmarks: *SPECweb99*, *VolanoMark* 2.1.2 and *SPECjbb2000*, which represent various front end and middle-tier Internet services are run on the 3 platforms described above. These benchmarks are described in detail in the sidebar on Internet benchmarks. To compare these benchmarks to "traditional" and better understood applications, we also experiment with the integer programs in the *SPEC CPU2000* suite. (See sidebar on *SPEC CPU2000*). We use the hardware performance monitors built into the microprocessors under study to make performance measurements. (See sidebar for a description on performance measurement using on-chip counters.) The Pentium III processor has two performance monitoring counters, and the IBM PowerPC processors have eight performance counters that can be programmed to count a variety of events. Events in unprivileged user code (user mode) and privileged operating system code (OS mode) can be counted separately on all 3 machines. The list of countable events differs between the two IBM machines, but many important events can be counted on both. We interface with the PowerPC performance monitors using the IBM-supplied performance monitor API and IBM's pmcount (a utility that allows the user to interface with the performance monitor), both of which are AIX kernel extensions. On the Intel processor, we use PMON [10] to access these counters. PMON consists of two parts, a device driver and a control program. The driver reads the performance counters of the Pentium III processor while the control program controls the measurement process and logs the results. Since we developed the whole tool ourselves, we have better control over it than any other performance counter tools like Intel's P6Perf. The overhead of PMON is extremely small because it does not have GUI displays and does not write results to disks during measurements. Thus the tool incurs no disk I/O activity given enough memory. The low overhead associated with the tool allows us to perform the measurements in a non-invasive fashion. The operation of the tool was verified by several test cases and by comparing it with VTune and P6Perf.

Though it is desirable to have quick starting and shutdown processes, the most important aspect of server performance is how the server responds to client requests. Therefore, in all our measurements we skip the starting and shutdown period and measure the server only when it is busy handling client requests. For this purpose, two timers are setup in the monitoring program to measure *SPECweb*. Since the *SPECweb* client is controlled by the warmup time, test time, etc. specified in the configure file, the first timer is used to skip the warmup time and the second timer to stop the measurement just before the client closes connections. However, *VolanoMark* is not controlled by a time parameter. Therefore, to synchronize our measurements with the client connections in the *VolanoMark* test, we add a wrapper to the client program. The wrapper sets up an extra connection to the server to trigger the monitoring program immediately before it starts the actual client. The monitoring program ends the measurement as soon as the wrapper closes the extra connection, which signals the end of the client program. Since *SPECjbb2000* does not have a separate client program, it is impossible to isolate the server transaction activity from data initialization and report generation without instrumentation of the benchmark itself. The benchmark program has the ability to measure itself for reporting benchmark scores. We modify Company.java file to send signals to the monitoring program so that our measurement is synchronized with the benchmark's own measurement interval. To minimize the effect of instrumentation we only recompile Company.java and leave all other class files untouched. As can be seen from our measurement method, the JIT compiling part should be negligible in the results because we skipped the starting of the program, where most compilation is done, and if any compilation slipped into our measurement, it would only account for a very small part of the long running of the benchmarks.

For *VolanoMark* and *SPECweb99* the servers are run on machines with high-speed network connections to the client machines. Only events on the server machines are counted. Apache on Windows is a multithreaded application, while on AIX it creates multiple processes to handle concurrent client connections. Apache's Perl module (mod_perl) is used to speedup the generation of dynamic web pages. But because the script provided by SPEC could not perform CAD Gets correctly using mod_perl, the CAD Get is implemented through traditional CGI resulting in some perl processes being created dynamically. Therefore, the number of processes/threads changes with time on all platforms. For *VolanoMark*, we use the default 20 users per room configuration. We vary the number of chat rooms from 1 to 30 resulting in a connection number range of 20 to 600. *SPECjbb2000* is a data intensive application with 25M bytes data for each warehouse/thread. The maximum number of threads that our systems can afford without significant memory swapping is 25. Therefore, we increase the number of warehouses from 1 to 25 in our experiments.

Comparison

In this section, we present data comparing the three server benchmarks with *SPECint* on our three different microarchitectures. We present results from *VolanoMark* runs with 10 and 30 chat rooms (indicated as volano10 and volano30 in the graphs and tables) and *SPECjbb* runs with 10 and 25 warehouses (indicated as SPECjbb10 and SPECjbb25). The metrics collected are similar to those collected by Bhandarkar et.al. [3]. We have made the best effort to make the metrics common across all three platforms. However, not all events are available on all platforms. Therefore, some related metrics are compared where the exact same metrics cannot be collected on all platforms. We focus on the difference between the server and SPEC integer (*SPECint*) benchmarks.

Operating System Component

Modern desktop and server processors have at least two modes: unprivileged user mode and privileged operating system (OS) or kernel mode. To ensure security, some instructions can only be executed in the OS mode. The processor enters the OS mode when the user application invokes a system call requesting the operating system to perform some task on its behalf (e.g. create another process, request synchronization with another kernel-level thread, or send a packet to the network). The processor also switches to OS mode when it responds to a hardware interrupt. For example, when the network adaptor receives a packet it will raise an interrupt to inform the operating system, which will, in turn, read the packet into a buffer in main memory. Switches to OS mode can also be caused by software exceptions during execution (e.g. page fault, divided by zero). We measure the operating system activity caused by the different applications under study. Table 1 compares the server benchmarks (*SPECweb*, *VolanoMark* and *SPECjbb*) to the *SPECint* benchmarks on the three different machines.

Table1. Percentage of OS mode cycles

Benchmark		RS64-III	POWER3-II	Pentium III
Servers	SPECweb	36.50%	32.87%	31.21%
	volano30	47.06%	54.11%	65.32%
	volano10	64.47%	59.83%	58.25%
	SPECjbb25	0.37%	0.33%	0.67%
	SPECjbb10	0.41%	0.39%	0.63%
SPECint 2000	vortex	0.41%	0.29%	1.06%
	twolf	0.17%	0.18%	0.37%
	gcc	0.87%	0.86%	1.04%
	eon	0.15%	0.29%	0.24%
	crafty	0.16%	0.18%	0.27%
	perlbmk	0.39%	0.40%	0.81%
	parser	0.22%	0.15%	0.41%
	gap	0.32%	0.35%	0.47%
	bzip2	0.69%	1.11%	0.44%
	vpr	0.28%	0.19%	0.38%
	mcf	0.43%	0.32%	0.34%
	gzip	0.74%	0.78%	0.48%

As the table indicates, *SPECweb* and *VolanoMark* spend a high proportion of their execution cycles in operating system mode. In contrast, most programs in the *SPECint2000* suite spend negligible time in the kernel mode. *VolanoMark* spends most of its time in receiving and sending network messages, which is mainly the task of the operating system. The number of threads in *VolanoMark* is large. To handle simultaneous client connections, a server usually spawns multiple threads or processes. The studied server creates two threads to manage each client connection. Thus scheduling and synchronizing these threads constitutes a major task of the operating system. Adding to this is the relatively simple operation of distributing messages in user code. Consequently, more than half of the execution time of *VolanoMark* is in OS mode. Network communication is also a major part of *SPECweb*. In addition, its execution has a large portion of disk accesses. Therefore 30% of the time is spent in OS mode. *SPECjbb*, on the other hand, spends less than 0.7% of the total execution time in OS mode, which is not very different from *SPECint*. In this respect, *SPECjbb* is not representative of server applications because it lacks the network communications and disk accesses common in all servers. Servers have been reported to have a higher percentage of execution time in kernel mode than technical workloads [8].

Instruction Level Parallelism profile

All three processors investigated in this study are superscalar processors capable of extracting instruction level parallelism (ILP). Both the RS64-III and the POWER3-II can dispatch up to four instructions per cycle (dispatch for these machines meaning the stage in which the instruction is sent to the reservation station of the execution unit). From Figure 4 it is clear that the machines have more difficulty in exploiting ILP in the server benchmarks than in the *SPECint* benchmarks. For all of the server benchmarks on the RS64-III, zero instructions are dispatched for over 50% of the execution cycles. Only three *SPECint* benchmarks has zero instructions dispatched for over 50% of the execution cycles. On the POWER3-II, the dispatch profile is similar (we show only the percentage of cycles with zero instructions dispatched because the other counts were not available on this machine). All of the server benchmarks on the POWER3-II have zero instructions dispatched for more than 60% of the execution cycles, while only *twolf* , *vpr* and *mcf* cross this threshold among the *SPECint* workloads. The profile is almost identical for the percentage of zero instructions-retired cycles on the POWER3-II, which is reasonable given that stalls are being created in the dispatch stage.

The same phenomenon is observed on the Pentium III processor. In the Pentium III, most instructions are first converted into simpler RISC-type operations called uops. Some are decoded into two to four uops, and the more complex instructions require microcode. The processor has 3 decoders that can handle up to 3 instructions every cycle. Up to 5 uops can be issued every clock cycle to the various execution units and up to 3 uops can be retired every cycle. But in the server applications, instruction level parallelism is seen to be limited: most of the time (more than 60%), the decoders are idle when executing the server benchmarks (Figure 4a) and no uops can be retired in more than 60% of the execution cycles (Figure 4b). Only *gcc* and *twolf* in *SPECint2000* shows a similarly poor decode and retirement profile. (*Mcf* is considered to be an exception. Its poor performance is solely due to its extraordinarily high data cache misses, which makes it somewhat easier to understand than the server applications.)

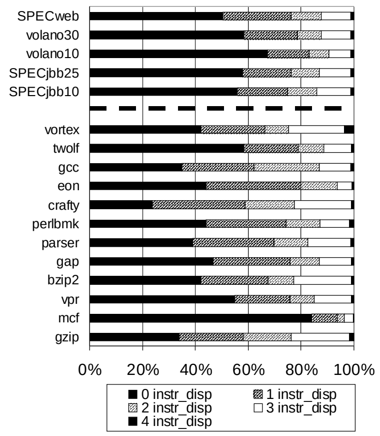

Figure 4A Instruction dispatch profile on RS64-III

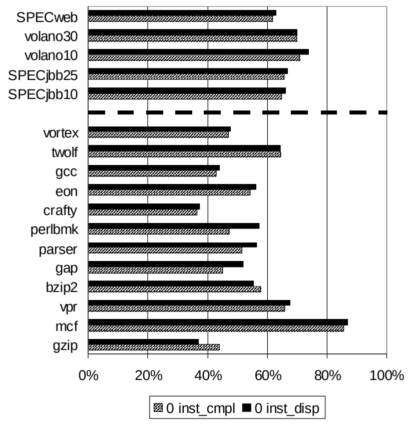

Figure 4B Instruction dispatch/completion profile on POWER3-III

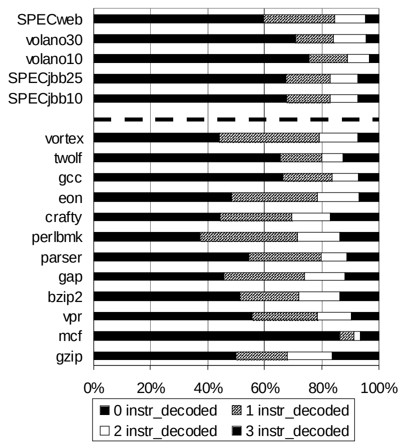

Figure 4C Instruction decode profile on Pentium III

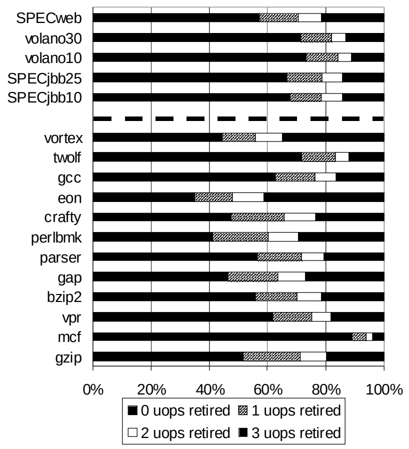

Figure 4D Instruction retirement profile on Pentium III

The difficulty in exploiting the instruction level parallelism also influences the average cycles per instruction (CPI) of the applications. Server applications generally exhibit a higher CPI than most *SPECint* programs as shown in Table 2.

Table2. Cycle Per Instruction (CPI)

Benchmark		RS64-III	POWER3-II	Pentium III
Servers	SPECweb	1.45	1.19	2.10
	volano30	1.76	1.44	3.03
	volano10	2.17	1.59	3.72
	SPECjbb25	1.52	1.27	2.31
	SPECjbb10	1.45	1.25	2.29
SPECint 2000	vortex	1.45	0.64	1.27
	twolf	1.41	1.23	2.25
	gcc	1.07	0.78	2.25
	eon	1.27	1.04	1.36
	crafty	0.77	0.63	1.22
	perlbmk	1.16	0.85	1.13
	parser	1.04	0.93	1.64
	gap	1.19	0.82	1.32
	bzip2	0.98	0.97	1.36
	vpr	1.32	1.29	1.79
	mcf	4.66	3.08	6.65
	gzip	0.80	0.68	1.24

Cache and TLB performance

Modern processors spend a significant part of their real estate on on-chip caches. These caches capture the instruction and data working sets and thus reduce the average memory access time. This section provides a comparison of the instruction and data cache behavior of the web server workloads and *SPECint2000* on the 3 machines.

L1 instruction cache: Figure 5 and Figure 6 show the L1 instruction cache misses and instruction TLB misses per 1000 instructions. The server applications exhibit poorer instruction cache and TLB performance than *SPECint* programs on all three machines. This indicates poor instruction locality at both the cache-line size level and page-size level. Comparing results on the POWER3-II and RS64-III, we can see that the instruction cache miss rates are higher on the POWER3-II for most of the workloads (since its instruction cache is 64KB as opposed to 128KB for the RS64- III), but for *vortex* and *crafty* the instruction cache miss rates are higher on the RS64-III. This indicates that, for the server workloads and the other *SPECint* benchmarks, size is more important than associativity for instruction cache performance, while for *vortex* and *crafty* associativity (2 for the RS64-III and 128 for the POWER3-II) is more important than size for performance.

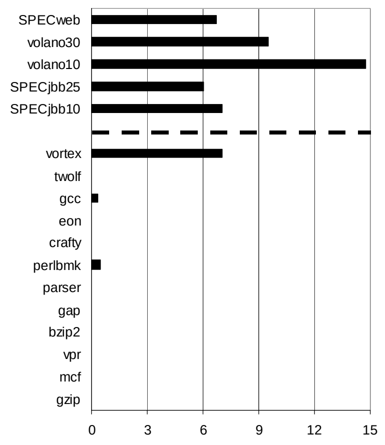

Figure 5A Icache misses per 1000 instructions on RS64-III

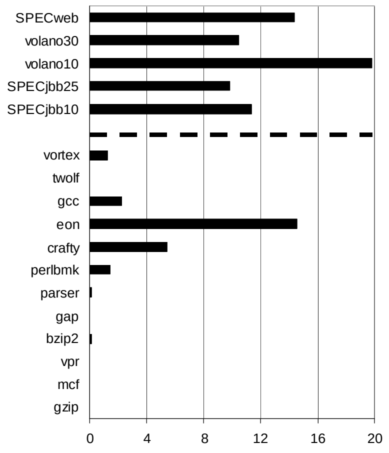

Figure 5B Icache misses per 1000 instructions on POWER3-II

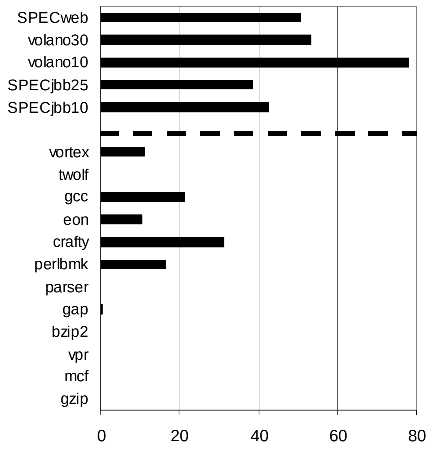

Figure 5C Icache misses per 1000 instructions on Pentium III

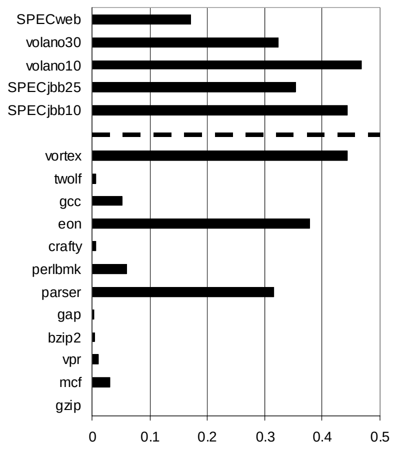

Figure 6A ITLB misses per 1000 instructions on RS64-III

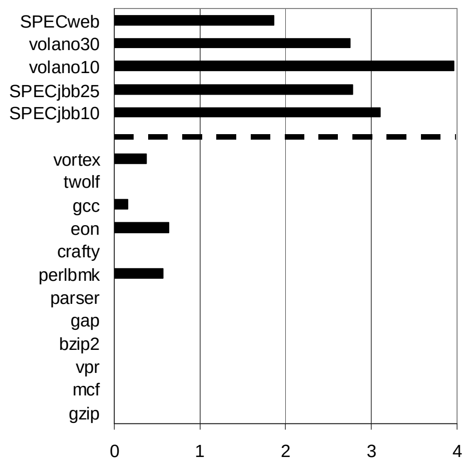

Figure 6B ITLB misses per 1000 instructions on Pentium III

To help understand the difference in the instruction access behavior, Table 3 compares the memory map for the code segments of various modules (user mode only) involved in the execution of *gcc* from the *SPECint* suite and the Apache web server running SPECweb benchmark. A noticeable difference between *Gcc* and Apache is that the web server invokes many more Dynamically Linked Libraries (DLLs). Real-world server applications are complex pieces of software. To streamline software development, nowadays, developers adhere to the principle of modularity at both source code level and binary level. Most functions in the web server are implemented as modules, which are DLLs on the Windows platform. The main executable of Apache server (apache.exe) is only about 20KB, while its core functions implemented in the DLL ApacheCore.dll is 320KB. One heavily used DLL is mod_perl.so, which handles the cgi perl scripts. Mod_perl in turn calls perl56.dll, a library from ActivePerl package to actually interprets and executes the perl script. To complete its job, perl56.dll needs services from other modules in the perl package, such as fcntl.dll. Besides, server programs usually require more services from the OS. For example, to communicate through the network, the winsock library is called. To access the Windows Registry, functions in advapi32.dll are invoked. *SPECint* programs, on the other hand, are compiled into self-contained stand-alone programs. Libraries are usually statically linked, resulting in compact executables. In addition, they seldom request services from OS or other applications except reading some input files at the beginning and printing the results at the end. As a result, few DLLs are involved in the execution of *SPECint* programs.

Table 3A. Memory map of code segments of gcc from the SPECint2000 suite

Starting address (hex)	code segment size (decimal)	module
00401000	958464	cc1_base
77f01000	241664	kernel32.dll
77f61000	245760	Ntdll.dll

Table 3B. Memory map of code segments of Apache web server

Starting address (hex)	code segment size (decimal)	module
00401000	4096	apache.exe
0d861000	8192	fcntl.dll
10001000	163840	mod_perl.so
1c0f1000	4096	Win9xConHook.dll
28001000	516096	perl56.dll
6ff61000	217088	ApacheCore.dll
77661000	36864	msafd.dll
77691000	12288	wshtcpip.dll
776a1000	8192	ws2help.dll
776b1000	49152	ws2_32.dll
776d1000	4096	wsock32.dll
77dc1000	172032	advapi32.dll
77e11000	286720	rpcrt4.dll
77e71000	266240	user32.dll
77ed1000	147456	gdi32.dll
77f01000	241664	kernel32.dll
77f61000	245760	ntdll.dll
78001000	204800	msvcrt.dll

The dynamic invocation and loading of the libraries affect the instruction footprint and the instruction access nature of the server program. Since each shared library is loaded to a different memory page, calling a function in another library causes the control flow to transfer to another memory page, resulting in poor instruction TLB performance. Similar behavior has also been observed in Windows desktop applications by Lee et. al. [6]. In addition, many server applications are written in Java and Just-In-Time (JIT) compilers compile Java code at runtime. Dynamically compiled code for consecutively invoked methods may not be located in contiguous address spaces [11]. All the aforementioned effects explain the difference in instruction access behavior between server applications and the *SPECint* suite. Considering the significant OS activity observed in the server execution also, the whole instruction stream of the Apache server is definitely much more complex and larger than *Gcc*. It might be noted that high instruction cache miss rates have been observed in traditional database server applications also [1, 2].

L1 data cache: The trends of the data cache miss rates in the level one data cache are not very different between the server and the *SPECint* applications and hence are not shown in detail. The *SPECint* applications have miss rates in a wide range, with *mcf* miss rates being very high. The miss rates of the server applications fall within the range exhibited by the *SPECint* applications. On the Pentium III, the number of L1 data misses per instruction in the *SPECint* applications range from 0.03 to 0.10, while the range for the server applications is 0.012 to 0.024. The number of misses per instruction is slightly lower on PowerPC machines: 0 to 0.094 for *SPECint*, and .005 to 0.016 for the server benchmarks. The SPEC program *mcf* has the highest miss rate on all the 3 platforms.

L2 cache: L2 cache misses per 1000 instructions are shown in Figure 7. The *mcf* program is exceptional on all platforms because of its high L2 cache miss rate. On Pentium III, which has a 512K byte L2 cache, the L2 cache miss rates of the server applications are comparable to those of *SPECint*. On the PowerPC machines, which have much larger L2 caches, the L2 cache miss rates of the servers are higher than those of *SPECint*. Comparing the L2 performance of the POWER3-II and RS64-III shows that when the cache size increases from 4M bytes to 8M bytes, the miss rates for the servers change very little, while the miss rates for *SPECint* are almost halved. This indicates that the data footprint of server applications is usually harder to capture with ordinarily large caches. Servers usually manage large data sets, as in the case of *SPECjbb*, in which each warehouse is 25MB of data. Since each client thread accesses a different warehouse, a large number of clients create a large and scattered data footprint.

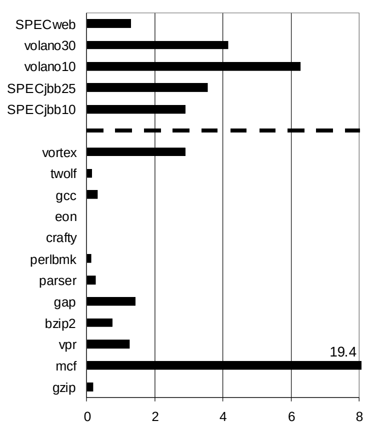

Figure 7A L2 cache misses per 1000 instructions on RS64-III

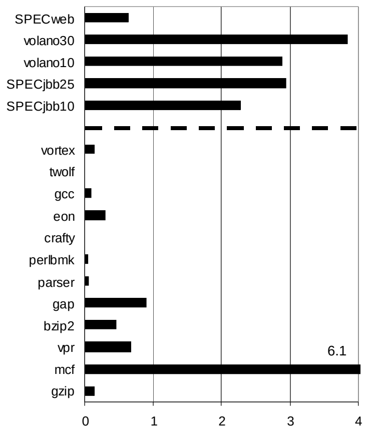

Figure 7B L2 cache misses per 1000 instructions on POWER3-II

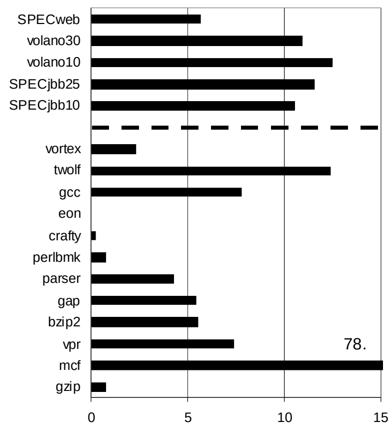

Figure 7C L2 cache misses per 1000 instructions Pentium III

Branch behavior

State-of-the-art high performance microprocessors employ speculative execution as a means of enhancing performance. Control path prediction is the most common form of speculation implemented on modern superscalar processors. As shown in Figure 8A, the two-level adaptive branch prediction scheme of Pentium III does a fairly good job in terms of misprediction rate, but the BTB miss rates for the server applications are higher compared to *SPECint2000*. If a BTB miss occurs, the static branch predictor will be used, which is generally not as accurate as the 2-level dynamic predictor. Even if the static predictor makes correct predictions, the processor may still suffer from the latency because the static predictor is rendered much deeper in the pipeline than the dynamic one. While we do not isolate the causes of the BTB misses and mispredictions, it is clear that the current BTB architecture on the Pentium III does not work very satisfactorily for server code.

Figure 8B indicates that the POWER3-II's branch direction predictor works as well for the server applications as for the *SPECint* benchmarks. However, the server benchmarks (with the exception of vol30) exhibit, on the average, worse BTAC (Branch Target Address Cache) performance than all *SPECint* programs but *eon*, *crafty*, and *gap*. This could indicate that the BTAC of the POWER3-II, which caches branch target addresses and does not store any target instructions, does not work very well for server code, though the difference between *SPECint* and server BTAC performance is not as pronounced as on the Pentium III. Further, *eon*, which shows BTAC performance similar to the server benchmarks, is written in C++ and makes heavy use of virtual functions, which are also widely used in Java. Java programs are known to have poor branch target predictability due to indirect branches resulting from virtual function calls and code interpretation [11]. Branch prediction numbers for RS64-III are not shown because it does not employ dynamic branch prediction.

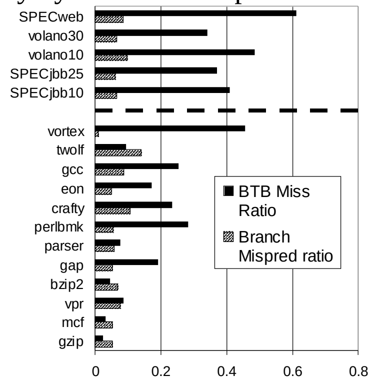

Figure 8A. Branch behavior on Pentium III

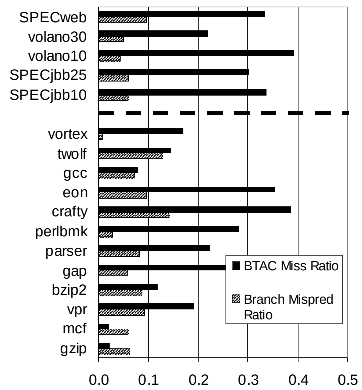

Figure 8B. Branch behavior on POWER3-II

Understanding components of CPI

It is very beneficial and interesting to find out the various components of the CPI and their individual contributions. There are some performance counters that shed some light on this aspect. On Pentium III two major types of stalls can be counted: I-stream stalls and resource stalls, measured in terms of the cycles in which the stall conditions occur. I-stream stalls are caused mainly by instruction cache misses and ITLB misses. Resource stalls show the number of cycles in which resources such as reorder buffer entries, memory buffer entries, or execution units are not available [3]. Figure 9A and 9B show the I-stream stalls and resource stalls per instruction respectively. Clearly, the resource stalls of the servers are within the range of *SPECint* while the I-stream stalls of the former are much higher than those of the latter. To determine the contribution of each stall component to the CPI is hard, especially on an out-of-order superscalar processor like the Pentium III, which is good at tolerating latencies by overlapping them. Therefore, we conducted a linear regression analysis in a top down fashion to determine statistically the importance of each stall factor. (See sidebar on regression analysis).

The top linear model is

$$CPI=b_2*RS+b_1*IS+b_0 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

where

RS is the number of resource stalls per instruction, and

IS is the number of I-stream stalls per instruction.

The regression analysis gives the following results:

b_2=1.00 with 95% confidence interval of [0.947, 1.06]

b_1=1.23 with 95% confidence interval of [1.10, 1.36]

b_0=0.785 with 95% confidence interval of [0.680, 0.891]

The coefficient of determination R^2 is 0.991. This means that this regression model explains 99.1% of the CPI variation, indicating high goodness of the linear model. The correlation between resource stalls and I-stream stalls is small (-0.156). Therefore, the regression model does not suffer from the problem of multicollinearity. Neither the confidence interval of b_2 nor that of b_1 includes 0. Thus both resource stalls and I-stream stalls are statistically important factors in determining the CPI. Based on this regression model, the contributions of resource and I-stream stalls to the CPI are shown in Table 4. For server applications, I-stream stalls contribute 33% to 62% to the CPI while the contributions of I-stream stalls for *SPECint* are all below 33%

and are negligible (less than 1%) for half of the suite. In all cases, the I-stream stalls make much larger absolute contribution to CPI for server applications than for *SPECint*.

Table 4. Contributions of resource stalls and I-stream stalls to CPI

Benchmarks		Absolute contribution		Relative contribution of I-stream stalls to CPI (as %)
		Resource stalls	I-stream stalls	
Server	SPECweb	0.35068	0.93785	45.22%
	volano30	0.97476	1.30841	42.64%
	volano10	0.89170	1.86214	52.61%
	volano01	0.59106	2.22142	61.74%
	SPECjbb25	0.86600	0.81121	32.94%
	SPECjbb10	0.80074	0.89546	36.08%
	SPECjbb01	0.74111	0.94555	38.25%
SPECint 2000	vortex	0.36342	0.20713	15.28%
	twolf	1.37280	0.00713	0.33%
	gcc	0.89576	0.32962	16.39%
	eon	0.29467	0.17404	13.88%
	crafty	0.13662	0.45766	33.17%
	perlbmk	0.12145	0.26453	22.58%
	parser	0.69689	0.00643	0.43%
	gap	0.45838	0.01048	0.84%
	bzip2	0.52759	0.00197	0.15%
	vpr	0.95453	0.00358	0.21%
	mcf	5.88349	0.00830	0.12%
	gzip	0.57135	0.00214	0.16%

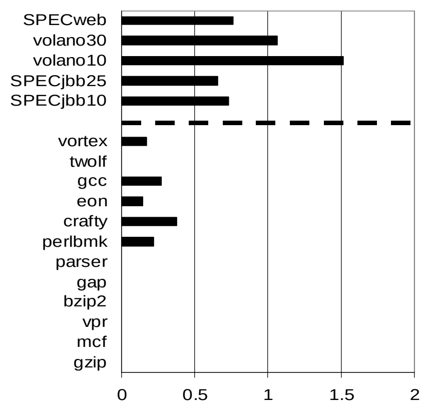

Figure 9A I-stream stalls per instruction on Pentium III

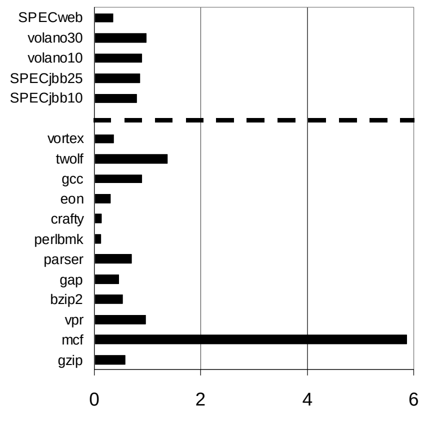

Figure 9B Resource stalls per instruction on Pentium III

Resource stalls encompass the conditions where register renaming buffer entries, reorder buffer entries, memory buffer entries, or execution units are full. In addition, serializing instructions (e.g., CPUID), interrupts, and privilege level changes may spend considerable cycles in execution, forcing the decoder to wait and incrementing the resource stalls counter. Stalls due to data cache misses are not explicitly included in resource stalls; however, if some other resource becomes oversubscribed due to a long data cache miss, the resource stall counter will be incremented [5]. Though resource stalls consist of various stall events, the linear regression analysis shows that the major component of resource stalls is caused by data cache misses, as in equation 2.

Resource stalls=54.10(L2 cache misses)+14.12*(L1 data cache misses)+0.0443…….(2)*

The coefficient of determination R^2 is 0.991. The single large resource stalls of *mcf* may result in unduly good R^2. Therefore, another analysis was performed without *mcf*. R^2 drops to 0.867, which still indicates strong correlation between resource stalls and data cache misses.

I-stream stalls count the number of cycles that instruction fetch is stalled for any reason, including L1 instruction cache misses, ITLB misses, ITLB faults, and other minor stalls. The influence of instruction cache misses on I-stream stalls can be understood using the following linear regression model:

$$IS=b1*L1+b0 \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

where

IS = I-stream stalls per instruction, and

$L1$ = L1 instruction cache misses per instruction.

Regression analysis yields that

b_1 = 18.1 with 95% confidence interval [16.9,19.4]

b_0 = -0.0330 with 95% confidence interval [-0.0822, 0.0161]

The coefficient of determination R^2=0.986, which means that L1 instruction cache misses alone statistically account for 98.6% of the variation in I-stream stalls.

In summary, CPI is statistically determined by resource stalls and I-stream stalls on the Pentium III processor. The I-stream stalls make larger contributions to the CPI for the server applications than for the *SPECint* suite. The high I-stream stalls of the servers can be attributed to their high L1 instruction cache misses.

Next, we explore the CPI components of the IBM platforms. Stall related events are not countable on the POWER3-II, and hence the analysis is limited to the RS64-III. Figure 10 compares CPI components of the server benchmarks to the *SPECint* benchmarks on the RS64-III. With good performance counter support and in-order-issue, the CPI stack of RS64-III is a good indication of the relative importance of various stalls. The stalls in the figure do not comprise a comprehensive list, but they are the most significant storage related stalls on the machine. "Ideal CPI" refers to (total execution cycles – storage latency)/instructions executed. "Storage latency" is a single countable event on the RS64-III performance monitor that indicates the (nonoverlapped) total amount of storage related stalls (i.e. multiple storage related stalls in one cycle are counted as one stall). Thus "Ideal CPI" is an approximation of CPI in the absence of all storage related stalls. "Isync" and "Other sync" stalls are caused by various synchronizing PowerPC instructions. It is clear that the server benchmarks incur significantly more instruction cache stalls, data cache stalls, data TLB stalls, and L2 cache stalls than the *SPECint* benchmarks, and further, that these stalls dominate the memory related stall components of CPI. (In contrast, the *SPECint* benchmarks suffer from very little, if any, of the storage related stalls included in the figure. However, despite the large number of storage stall cycles for the server benchmarks, the CPIs of the benchmarks 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.

Data cache stalls and L2 stalls are the most significant storage stalls for the server applications. Figure 11 shows that the vast majority of the L2 misses on the RS64-III are due to loads as opposed to instructions, and Figure 10 shows that the data cache stall components of the server applications are nearly equal to the L2 stall components, indicating that most of the data cache stalls are in fact L2 stalls (L2 stalls are, of course, included in instruction and data cache stall times.). Therefore, L2 performance, not data cache performance, is the real performance bottleneck with these server applications. The high proportion of L2 stalls agrees with our earlier speculation that the large data sets, characteristic of server applications, cause poor L2 behavior in comparison to *SPECint*. Though instruction cache behavior also has a significant impact on server performance, the poor L2 performance overshadows the poor instruction cache performance. Observing Figure 10 and 11 instruction cache stalls on the RS64-III do not seem to adversely affect performance to the same degree that Istream stalls do on the Pentium III. This is most likely because the RS64-III has an instruction cache eight times as large (128 KB vs. 16 KB). Further, the high ITLB miss rates of the server applications combined with the fact, illustrated by Figure 11, that hardly any L2 misses arise from instruction accesses suggest that most of the instruction cache misses on the RS64-III are due to a widely scattered instruction footprint rather than one large and contiguous footprint, which agrees with the earlier discussion in Section 4.3 about function call addresses and dynamically compiled JIT code and their affect on instruction stream behavior. The contrast in instruction steam stalls between the RS64-III and the Pentium III shows that a larger instruction cache can improve server performance, despite the scattered nature of the instruction footprint. Still, instruction cache stalls do adversely affect server performance on the RS64-III.

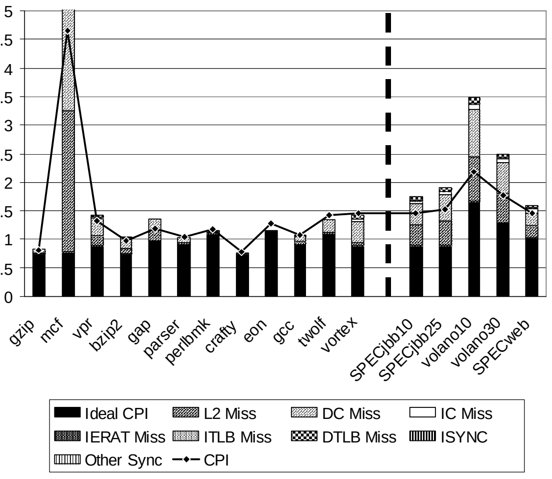

Figure 10 CPI stack for RS64-III

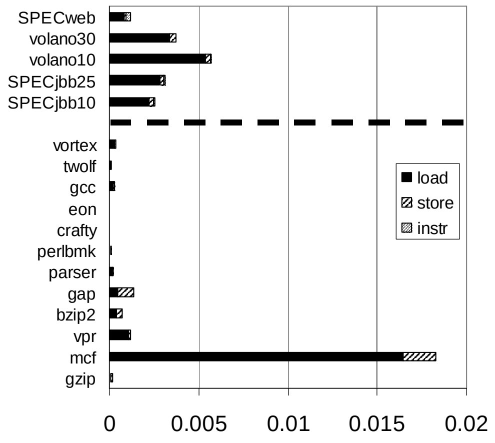

Figure 11 Breakdown of L2 Misses per Instruction on RS64-III

Summary

Detailed characterization of three Internet server benchmarks, *SPECweb99*, *VolanoMark* and *SPECjbb2000*, and their comparison with *SPECint*2000 on three different architectures, i.e. IBM RS64-III, IBM POWER3-II and Intel Pentium III, demonstrate that the server applications differ from *SPECint* in several ways:

1. Server benchmarks show worse instruction stream behavior than *SPECint2000*. Higher instruction cache miss rates, higher ITLB miss rates, higher BTB miss rates and consequently, higher I-stream stall cycles are observed on all three machines for the server applications in comparison to the *SPECint* suite. The linear regression model shows that statistically, the instruction cache misses make a larger contribution to CPI for the server applications than for *SPECint* on the Pentium III. On the PowerPC architectures, which have much larger instruction caches than the Pentium III, instruction cache stalls still make up a significant component of the CPIs of server workloads, while they are near negligible in the *SPECint* workloads. Nevertheless, with the RS64-III instruction cache being four times as large (128 KB vs. 16 KB) as that of the Pentium III, instruction stream behavior degrades performance to a lesser degree on the former than on the latter, demonstrating the advantage that large instruction caches can provide for these server applications.
2. Due to the large data set and network I/O the servers manage, the data footprint of servers is harder to capture than that of *SPECint* even with large L2 caches as in the PowerPC machines. L2 cache misses per instruction are significantly higher than those for

SPECint2000 (except for *mcf*), and L2 cache miss stalls constitute the major stall component of CPI on the RS64-III. Poor L2 performance is another major factor in overall performance for the server workloads.

3. In general, the server workloads have a higher percentage of cycles in which no instructions are decoded/dispatched/retired, suggesting that it is somewhat difficult to exploit ILP in these workloads.

It is interesting to note that the observations generally hold on all the three architectures irrespective of them being drastically different, the RS64-II being an in-order execution RISC machine, the POWER3-II being a highly aggressive out-of-order execution RISC machine, and the Pentium III being an out-of-order execution CISC machine, all with varying cache and TLB sizes and branch prediction schemes. There is no doubt that, to maximize performance on Internet server applications, modern processor architectures need further design enhancements and optimizations, particularly in instruction stream and L2 cache performance.

References

[1] A. Ailamaki, D. J. DeWitt, M. D. Hill and D. A. Wood. DBMSs on a Modern Processor: Where Does the Time Go? In *Proceedings of the 25th VLDB Conference*, Edinburgh, Scotland, 1999, pp. 15-26

[2] L. A. Barroso, K. Gharachorloo and E. Bugnion. Memory System Characterization of Commercial Workloads. In *Proceedings of the 25th International Symposium on Computer Architecture*, 1998, pp. 3-14

[3] D. Bhandarkar and J. Ding. Performance Characterization of the Pentium Pro Processor. In *Proceedings of The third International Symposium on High-Performance Computer Architecture*, 1997, pp. 288-297

[4] J.M. Borkenhagen, R. J. Eickemeyer, R. N. Kalla, and S.R. Kunkel. A Multithreaded PowerPC Processor for Commercial Servers. *IBM Journal of Research and Development*, Vol. 44, No. 6, 2000, pp. 885-894.

[5] K. Keeton, D. A. Patterson, Y. Q. He, R. C. Raphael, and W. E. Baker. Performance Characterization of a Quad Pentium Pro SMP Using OLTP Workloads. In *Proceedings of the 25th International Symposium on Computer Architecture*, Barcelona, Spain, June 1998, pp. 15-26

[6] D. C. Lee, P. J. Crowley, J. Baer, T. E. Anderson, and B. N. Bershad. Execution Characteristics of Desktop Applications on Windows NT. In *Proceedings of the 25th International Symposium on Computer Architecture*, Barcelona, Spain, June 1998, pp. 27-38

[7] T. Li, L. K. John, N.Vijaykrishnan, A. Sivasubramaniam, A.Murthy, and J. Sabarinathan, Using Complete System Simulation to Characterize SPECjvm98 Benchmarks. In *Proceedings of International Conference on Supercomputing*, 2000, pp. 22-33

[8] A. M. G. Maynard, C. M. Donnelly and B. R. Olszewski. Contrasting characteristics and cache performance of technical and multi-user commercial workloads. In *Proceedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems*, San Jose, October 1994, 145-156.

[9] F.P. O'Connell and S.W. White. POWER3: the Next Generation of PowerPC Processors. *IBM Journal of Research and Development*, Vol. 44, No. 6, 2000, pp. 873-884, 873-884.

[10] PMON webpage. http://www.ece.utexas.edu/projects/ece/lca/pmon.

[11] R. Radhakrishnan, N. Vijaykrishnan, L. K. John, and A. Sivasubramaniam. Architectural Issues in Java Runtime Systems. In *Proceedings of the Sixth International Conference on High Performance Computer Architecture*, January 2000, pp. 387-398.

[12] P. Ranganathan, K. Gharachorloo, S. V. Adve and L. A. Barroso. Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*. October 1998, pp. 144-156.

Sidebar 1

SPEC CPU Benchmarks The System Performance Evaluation Cooperative (SPEC) is one of the early efforts in benchmark gathering and performance evaluation. SPEC was founded in 1988 by a small number of workstation vendors who realized that the marketplace was in desperate need of realistic, standardized performance tests. The basic SPEC methodology is to provide the benchmarker with a standardized suite of source code based upon existing applications that has already been ported to a wide variety of platforms by its membership. The benchmarker then takes this source code and compiles it for the system in question.

SPEC started its efforts by releasing the SPEC89 suite of CPU intensive benchmarks in 1989. This was followed by SPEC92, SPEC95 and SPEC2000. The suites contain integer and floating point benchmarks. SPEC creates its suites by gathering source code from real user applications. They give importance to portability of code so that the benchmarks can be used across a wide range of hardware. These benchmarks measure the performance of the processor, memory and compiler on the tested system. The current CPU suite, SPEC2000 suite contains 14 floating point programs written in C/Fortran and 12 integer programs (11 written in C and 1 in C++). The programs in the SPEC 2000 suite are listed in Table below.

Programs in the SPEC Cint2000

Benchmark name	Language	Application details
Gzip	C	Compression
Vpr	C	FPGA Circuit Placement and Routing
Gcc	C	Programming Language Compiler
Mcf	C	Combinatorial Optimization
Crafty	C	Game Playing: Chess
Parser	C	Word Processing
Eon	C++	Computer Visualization
Perlbmk	C	PERL Programming Language
Gap	C	Group Theory, Interpreter
Vortex	C	Object-oriented Database
Bzip	C	Compression
twolf	C	Place and Route Simulator

Programs in SPEC Cfp2000

wupwise	Fortran 77	Physics / Quantum Chromodynamics
swim	Fortran 77	Shallow Water Modeling
mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
applu	Fortran 77	Parabolic / Elliptic Partial Differential Equations
mesa	C	3-D Graphics Library
galgel	Fortran 90	Computational Fluid Dynamics
art	C	Image Recognition / Neural Networks
equake	C	Seismic Wave Propagation Simulation
facerec	Fortran 90	Image Processing: Face Recognition
ammp	C	Computational Chemistry
lucas	Fortran 90	Number Theory / Primality Testing
fma3d	Fortran 90	Finite-element Crash Simulation
sixtrack	Fortran 77	High Energy Nuclear Physics Accelerator Design
apsi	Fortran 77	Meteorology: Pollutant Distribution

For more information, please see http://www.spec.org.

Sidebar 2
Internet/Server Benchmarks

The phenomenal growth of the World Wide Web has resulted in the emergence and popularity of several information technology applications. The proliferation of web servers and arrival of electronic commerce has brought in several new software packages, interfaces and standards into the picture. Literally all enterprises including airlines, banks, stock brokerage firms, and most consumer product vendors nowadays use their web servers to deal with a significant part of their business. Many of these applications involve a web based interface to an underlying database that stores the data relating to the user enquiry or transaction. Driven by the business need for decision-support and powerful on-line transaction processing, organizations are implementing high-volume databases with terabytes of data. Commercial workloads include Enterprise Resource Planning (ERP) applications, on-line transaction processing (OLTP), decision support systems (DSS), web-index search and many similar e-commerce workloads. ERP applications integrate all aspects of an enterprise such as manufacturing, finance, sales, distribution, and human resources. Business decisions can be made by DSS software running on top of the enterprise database. Market research, customer buying patterns, etc can be studied using business intelligence and data mining software. An end-to-end e-business transaction typically involves a dozen or more different software layers, including the front end/portal, shopping carts, network communication, credit card or electronic check transaction, security software layers, etc. These layers may be implemented using cgi-script, Java servlets, Java Beans, XML, CORBA, JINI, etc. Thus e-commerce involves the integration of a variety of software, protocols, and standards: GUIs, Java, cgi, perl, php, SQL, TCP/IP, HTTP, and HTML are just a few of them.

Modern servers use a 3-tier approach in which the backend tier handles the database accessing and the front end and the middle tiers implement much of the user interface and portals. Table 1 provides examples of benchmarks in the different tiers. The Transaction Processing Council (TPC) benchmarks model the backend. (See related article XXX for a description of the TPC Benchmarks.) Examples of the front and middle tier benchmarks are SPECweb99, VolanoMark and SPECjbb2000. In the accompanying article, we describe the characteristics of these 3 benchmarks on 3 different superscalar machines.

Table 1. Popular benchmarks for different internet/server workloads

Server Backend /Database	On-Line Transaction Processing	TPC-C, TPC-W
	Decision Support Systems	TPC-H, TPC-R
Server front and middle tier	Web Server	SPECweb99
	Chat Server	VolanoMark
	E-commerce server	SPECjbb2000, TPC-W

SPECweb99 [1] is SPEC's current benchmark for measuring the performance of web servers. The SPECweb99 workload (Figure 1) simulates accesses to a web service provider, where the server supports web pages for a number of different organizations. Each home page is a collection of files ranging in size 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 are made to closely match today's real-world web server access patterns. The benchmark supports HTTP1.0 and HTTP1.1 protocols. The benchmark's metric is SPECweb99, which represents the number of simultaneous connections the web server can support using the predefined workload.

VolanoMark [2] is a pure Java server benchmark with long-lasting network connections and high thread counts. It can be divided into two parts: a server and a client, although they are provided in one package. It is based on a commercial chat server application, the VolanoChat, which is used in several countries world-wide. The server accepts connections from the chat client as in Figure 2. The chat client simulates many chat rooms and many users in each chat room. The client continuously sends messages to the server and waits for the server to broadcast the messages to the users in the same chat room. The VolanoMark server creates two threads for each client connection. VolanoMark can be used to test both speed and scalability of a system. In the speed test, it is run on a local loopback connection with the server and client on a single machine. In the scalability test, the server and client are run on separate machines with high speed network connections.

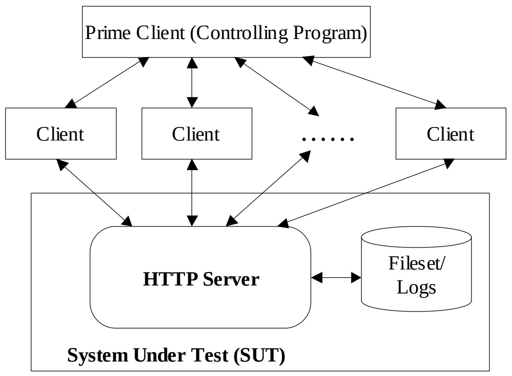

Figure 1 SPECweb99 Environment

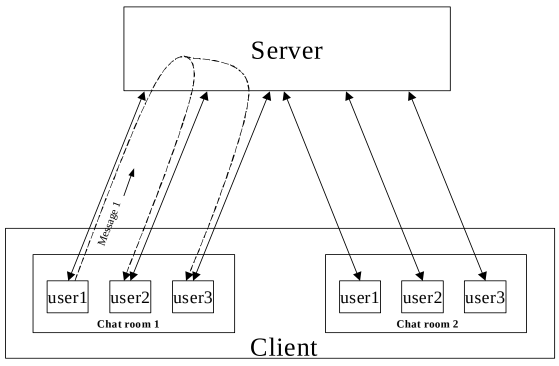

Figure 2 VolanoMark Environment

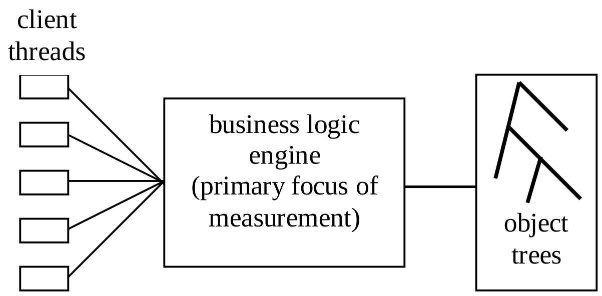

Figure 3 SPECjbb2000 Environment

SPECjbb2000 [3,4] is SPEC's first e-business benchmark and the first benchmark for evaluating the performance of server-side Java. JBB stands for Java Business Benchmark. The benchmark emulates an electronic commerce workload in a 3-tier system (Figure 3). The benchmark contains business logic and object manipulation, primarily representing the activities of the middle tier in an actual business server. It models a wholesale company with warehouses serving a number of districts. Customers initiate a set of operations such as placing new orders and checking the status of existing orders. Additional operations are generated within the company, such as processing orders for delivery, entering customer payments, and checking stock levels. It is written in Java, adapting a portable business oriented benchmark called pBOB written by IBM. Although it is a benchmark that emulates business transactions, it is very different from the Transaction Processing Council (TPC) benchmarks. There are no actual clients, but they are replaced by driver threads. Similarly, there is no actual database access. Data is stored as binary trees of objects. The whole benchmark is implemented within a single Java Virtual machine (JVM). The emulation of the other tiers isolates the middle tier and simplifies the benchmark by not requiring user emulation or a database. SPECjbb2000 assigns one active customer per warehouse, which is a 25MB data set stored in binary trees (Btrees). The benchmark is memory resident without inherent disk I/O. One warehouse maps directly to one Java thread. As the number of warehouses increases during the full benchmark run, so does the number of threads.

References:

1. Standard Performance Evaluation Cooperative SPECweb99 Benchmark. http://www.spec.org/osg/web99/ (current July 2002).
2. Volano LLC. VolanoMark benchmark. http://www.volano.com/benchmarks.html (current July 2002).
3. Standard Performance Evaluation Corporation. SPECjbb2000 Benchmark. http://www.spec.org/osg/jbb2000/ (current July 2002).
4. Standard Performance Evaluation Corporation. Architecture schematic of the SPEC JBB2000 benchmark process. http://www.spec.org/osg/jbb2000/images/arch.jpg (current July 2002).

Sidebar 3
Performance Monitoring Using Microprocessor On-chip Performance Monitoring Counters

All state-of-the-art high performance microprocessors including Intel's Pentium III and Pentium 4, IBM's POWER 3 and POWER 4 processors, AMD's Athlon, Compaq's Alpha, and Sun's UltraSPARC processors incorporate on-chip performance monitoring counters which can be used to understand the performance of these microprocessors while they run complex, real-world workloads. Studies to understand workloads used to use simulators and simulator based program profilers but they often could not execute complex workloads. The inability to run complete workloads including their operating system component used to be a major limitation. Now, complex run time systems involving multiple software applications can be evaluated and monitored very closely. All microprocessor vendors nowadays release information on their performance monitoring counters.

For illustration of on-chip performance monitoring, let us look at the Intel P6 family processors (Pentium Pro, Pentium II and III). The microprocessors in the Intel P6 family contain two performance monitoring counters. These counters can be read with special instructions (eg: RDPMC) on the processor. The counters can be configured to measure user and kernel activity in combination or in isolation. A variety of performance events can be measured using the counters [1]. For illustration of the nature of the events that can be measured, Table 1 lists a small subset of the events that can be measured on the Pentium III. While more than 200 distinct events can be measured on the Pentium III, only 2 events can be measured simultaneously. For design simplicity, most microprocessors limit the number of events that can be simultaneously measured to a small number. Table 2 lists the number of performance monitoring counters on several processors. At times, certain events are restricted to be accessible only through a particular counter. These steps are necessary to reduce the overhead associated with on-chip performance monitoring. Performance counters do consume on-chip real estate. Unless carefully implemented, they can detrimentally affect the processor cycle time.

Table 1. Examples of events that can be measured using performance monitoring counters on an Intel Pentium III processor

EVENT	Description of Event	Event Number in Hex
DATA_MEM_REFS	All loads and stores from/to memory	43H
DCU_LINES_IN	Total lines allocated in the data cache unit	45H
IFU_IFETCH	Number of instruction fetches (cacheable and uncacheable)	80H
IFU_IFETCH_MISS	Number of instruction fetch misses	81H
ITLB_MISS	Number of Instruction TLB misses	85H
IFU_MEM_STALL	Number of cycles instruction fetch is stalled for any reason	86H
L2_IFETCH	Number of L2 instruction fetches	28H
L2_LD	Number of L2 data loads	29H
L2_ST	Number of L2 data stores	2AH
L2_LINES_IN	Number of lines allocated in the L2	24H
L2_RQSTS	Total number of L2 requests	2EH
INST_RETIRED	Number of instructions retired	C0H
UOPS_RETIRED	Number of micro-operations retired	C2H
INST_DECODED	Number of instructions decoded	D0H
RESOURCE_STALLS	Number of cycles in which there is a resource related stall	A2H
MMX_INSTR_EXEC	Number of MMX Instructions Executed	B0H
BR_INST_RETIRED	Number of branch instructions retired	C4H
BR_MISS_PRED_RETIRED	Number of mispredicted branches retired	C5H
BR_TAKEN_RETIRED	Number of taken branches retired	C9H
BR_INST_DECODED	Number of branch instructions decoded	E0H

BTB_MISSES	Number of branches for which BTB did not predict	E2H

Table 2. Number of Performance Monitoring Counters on Microprocessors

Microprocessor	Number of Counters
Intel Pentium II/III	2
Pentium 4	18
IBM Power3-II, IBM RS64-III	8
AMD Athlon	4
Compaq Alpha 21164	3
Compaq Alpha 21264	2
MIPS R10000	2
MIPS R12000	4
UltraSPARC I, II, III	2

There are several tools available to measure performance using performance monitoring counters. Table 3 lists some of the available tools. Intel's *VTune* software may be used to perform measurements using the Intel processor performance counters [2]. The *P6Perf* utility is a plug in for Windows NT performance monitoring [3]. The Compaq DIGITAL Continuous Profiling Infrastructure *(DCPI)* is a very powerful tool to profile programs on the Alpha processors [4,5]. The performance monitor *perf-mon* uses the on-chip counters on UltraSPARC-I/II processors to gather statistics [6]. Packages like VTune perform extensive post-processing and present data in graphical forms. However, some times, extensive post-processing can result in tools that are somewhat invasive. *PMON* [7] is a counter reading software developed at the Laboratory for Computer Architecture at the University of Texas. It provides a mechanism to read specified counters with minimal or no perceivable overhead. All these tools measure user and operating system activity. Since everything on a processor is counted, effort should be made to have minimal or no other undesired process running during experimentation. This type of performance measurement can be done on binaries, and no source code is needed.

Table 3. Example software packages for performance counter measurement

Tool	**Platform**	**Reference**
VTune	IA-32	http://developer.intel.com/software/products/vtune/vtune60/vtune_oview.htm.
P6Perf	IA-32	http://developer.intel.com/vtune/p6perf/index.htm
PMON	IA-32	http://www.ece.utexas.edu/projects/ece/lca/pmon
DCPI	Alpha	http://www.research.digital.com/SRC/dcpi/ http://www.research.compaq.com/SRC/dcpi/
Perf-mon	UltraSPARC	http://www.sics.se/~mch/perf-monitor/index.html

References:

[1] D. Bhandarkar and J. Ding, "Performance Characterization of the Pentium Pro Processor", Proceedings of the 3rd High Performance Computer Architecture (HPCA) Symposium, 1997, pp. 288-297.

[2] Vtune profiling software,
http://developer.intel.com/software/products/vtune/vtune60/vtune_oview.htm.

[3] P6perf utility, http://developer.intel.com/vtune/p6perf/index.htm
[4] DCPI Tool home page, http://www.research.compaq.com/SRC/dcpi/

[5] J. Dean, J. E. Hicks, C. A. Waldspurger, W. E. Weihl, and G. Chrysos, "Profile Me: Hardware Support for Instruction Level Profiling on Out of Order Processors", Proceedings of the 1997 International Symposium on Microarchitecture (MICRO-30), 1997, pp. 292-302.

[6] Perf-monitor for UltraSparc, http://www.sics.se/~mch/perf-monitor/index.html
[7] PMON http://www.ece.utexas.edu/projects/ece/lca/pmon

Sidebar 4

Linear Regression

Given a set of related measures, the term "regression" is used to characterize the manner in which one of the measures changes as the other measures change. Thus one estimates or predicts one variable as a function of several other variables. The estimated variable is called the **response variable**, and the variables used to predict the response are called **predictor variables**, **predictors**, or **factors**. In order to construct a regression model, the information on response variables and predictor variables are obtained from a sample of objects, events, or individuals. Although regression techniques can be used to develop a variety of linear and nonlinear models, their most common use in linear models. Such models are called **linear regression models**. A **simple linear regression model** has only one predictor variable as in

$$\hat{y} = b_0 + b_1 x$$

where x is the predictor variable, y is the response variable, and the parameters b_0 and b_1 are fixed regression parameters to be determined from the data. Given n observation pairs $\{(x_1, y_1), \ldots , (x_n, y_n)\}$, the estimated response $\hat{y}_i$ for the *i*th observation is

$$\hat{y}_i = b_0 + b_1 x_i$$

The error between the predicted value and the actual value for the *i*th observation is

$$e_i = y_i - \hat{y}_i$$

If one sums up the errors to find the goodness of the regression, positive and negative errors can cancel each other yielding a deceiving metric. Hence goodness is judged based on a Least Squares Criterion, the **Sum of Squared Errors (SSE**) given by

$$\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - b_0 - b_1 x_i)^2$$

with the constraint that the mean error is zero.

The regression parameters that produce minimum error can be calculated as

$$b_1 = \frac{\sum_{i=1}^{n} x_i y_i - n\overline{x}\overline{y}}{\sum_{i=1}^{n} x_i^2 - n(\overline{x})^2} \qquad b_0 = \overline{y} - b_1 \overline{x}$$

where $\overline{x}$ is the mean of the predictor variable and $\overline{y}$ is the mean response.

If there was no regression model and one used the mean value of *y* as the estimated value for all values of the predictor variable, then the sum of squared errors would be

$$\sum_{i=1}^{n} (y_i - \overline{y})^2 ,$$

which is defined as **Total Sum of Squares** (**SST**).

The difference between SST (no regression) and SSE (with regression) is called **SSR** or the sum of squares explained by regression. Thus SSR indicates the variation that can be explained by the regression, and SST is the total variation. SSE indicates the variation that cannot be explained by the regression.

The fraction of the variation that is explained by the regression determines the goodness of the regression and is called the **coefficient of determination**, R^2:

$$R^2 = \frac{SSR}{SST} = \frac{SST - SSE}{SST} = 1 - \frac{SSE}{SST}$$

A perfect regression model has a value of 1 for R^2.

The basic idea of simple linear regression can be extended to **multiple linear regression model**, in which more than one predictor variable is used. When doing multiple linear regression, a predictor variable that has linear dependence on other predictor variables should not be used. Otherwise, various significance tests may show contradictory results, known as the problem of **multicollinearity.** For a detailed treatment of regression models, the reader is referred to [1].

Reference:

[1] Raj Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc. 1991

Sidebar 5 – TPC Benchmarks

We can write a Sidebar on TPC Benchmarks if no other article is describing TPC Benchmarks

Glossary:

We can prepare a glossary of terms if required.