Workload Characterization of Java Server Applications on Two PowerPC Processors*

Pattabi Seshadri and Lizy K. John
Dept of Electrical and Computer Engr
The University of Texas at Austin
{seshadri,ljohn}@ece.utexas.edu

Alex Mericas
IBM Corporation
mericas@us.ibm.com

Abstract

Java has become fairly popular on commercial servers in recent years. However, the behavior of Java server applications has not been studied extensively. We characterize two Java server benchmarks, SPECjbb2000 and VolanoMark 2.1.2, on two IBM PowerPC architectures, the RS64-III and the POWER3-II, and compare them to more traditional workloads as represented by selected benchmarks from SPECint2000. We find that our Java server benchmarks have generally the same characteristics on both platforms: in particular, high instruction cache, ITLB, and BTAC (Branch Target Address Cache) miss rates. These benchmarks also exhibit high L2 miss rates due mostly to data loads. Instruction cache and L2 misses are seen to be the primary contributors to CPI.

1. Introduction

Java, originally used extensively for web client software, is an emerging paradigm for server applications because of its portability and enhanced security features. However, while Java server applications are coming into wide use, their behavior is not yet well understood. Java client applications have been studied [17,9,15], but Java server applications differ significantly from client workloads, particularly in their need to maintain many concurrent client connections. Since in the current version of Java, I/O multiplexing, polling, and signals are not available, the only method available to Java programmers to maintain a large number of client connections is threads. One or more separate threads are created to handle each client connection [12]. Therefore performance in the presence of a large number of concurrent threads is vital to a Java server application. This distinct characteristic of Java server applications could lead to differences with Java client workloads in terms of branch behavior, cache behavior, and other metrics that contribute to overall performance.

The aim of this study is to characterize the impact of multithreaded Java server applications on modern processor microarchitectures. To this end, we compare multithreaded Java server benchmarks with selected benchmarks from SPECint2000, a suite of more "traditional" workloads. We run these benchmarks on two IBM PowerPC microarchitectures, the RS64-III and the POWER3-II.

2. Related Work

Commercial workloads have been increasing in importance, and efforts have been made to understand their behavior [2,11,8,7,16,1]. Most of these studies have been focused on applications written in C or C++, in particular OLTP, DSS, and web server applications.

Java has also been a popular subject of research. The majority of Java studies use SPECjvm98 [17,9,15], which is a client benchmark suite. SPECjvm98 has been observed to have as much as 31% kernel activity due for the most part to a TLB service routine, which indicates a high TLB miss rate. SPECjvm running on an interpreter has also been observed to have poor ILP and insensitivity to wider issue width [9]. However, it has better instruction cache performance than some C/C++ applications [15].

Commercial Java servers are emerging workloads and thus research has just begun on their behavior. Most of the research in this area has been on the effect

of multithreading. Cain and Rajwar [6] studied branch prediction and cache behavior in SPECjbb2000 and TPC-W with the full-system simulation of a coarse-grained multithreaded processor. They found destructive interference between threads that degraded performance. Luo and John [10] studied the impact of multithreading in Java server benchmarks on a Pentium Pro machine. They did see constructive interference in the instruction stream and branch prediction behavior, but these benefits were eventually overcome by increasing resource stalls as the number of threads grew large.

This paper focuses on the differences between Java server applications and more "traditional" workloads (represented by SPECint2000). We use two popular IBM PowerPC platforms that represent the state of the art in microprocessor design. Several performance metrics, such as cache behavior, branch behavior, dispatch behavior, CPI components, etc., are studied.

3. Methodology

This section describes the hardware platforms and benchmarks used in this study as well as the methods used to collect performance monitor data.

3.1. Platforms

We use two IBM PowerPC microarchitectures for our study: the RS64-III and the POWER3-II. Both are current microprocessor architectures, but they differ in many significant ways.

The RS64-III [4,5] is a 64-bit, superscalar, in order, speculative execution machine 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 cache. 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 processor clock is 500Mhz.

The POWER3-II [13,14] is a 64-bit, superscalar, out of order, speculative execution machine. 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 cache. 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. The processor clock is 450 MHz.

Both of these processors are deployed in IBM p-series systems. The RS64-III system we use in the experiment is the M80 and the POWER3-II system we use is the 44p-170, both of which are configured as uniprocessor systems. Both systems have 2 GB of main memory and run AIX 4.3.3 and the IBM JDK version 1.18.

3.2. Benchmarks

In this study, we characterize VolanoMark 2.1.2 and SPECjbb2000, both of which are Java server benchmarks.

VolanoMark 2.1.2 [20] is a Java server benchmark that simulates a chat server environment, as illustrated in Figure 1. The VolanoMark server accepts connections from the chat client, which simulates a specifiable number of chat users by creating a number of chat rooms. Each chat room contains a number of users that continuously send messages to the server and wait for the server to send the messages to other users

in the room. The VolanoChat server creates two threads for each client connection.

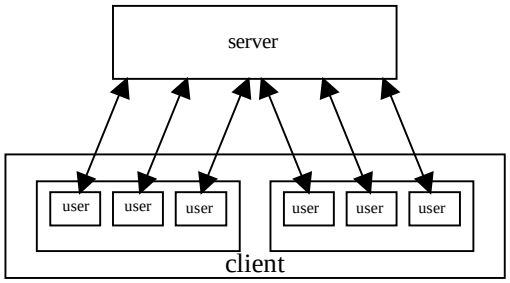


Figure 1. VolanoMark

SPECjbb2000 [19] is another Java server benchmark. As illustrated in Figure 2, it emulates a three-tier client/server system with emphasis on the middle tier, the business logic engine. The other tiers are emulated, and thus user emulation and a database are not required. SPECjbb is patterned after TPC-C in

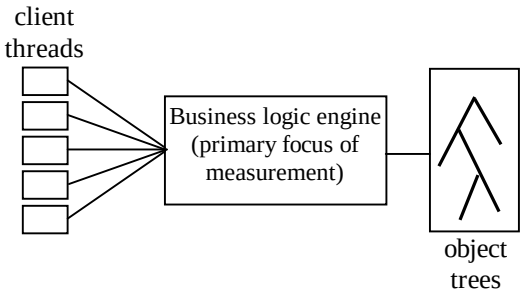


Figure 2. SPECjbb2000

that it models a wholesale company with warehouses that serve a number of districts. The transactions generated in this system include new orders and order status requests (both customer-generated transactions), as well as processing orders, entering customer payments, and checking stock levels (company-generated transactions). Each warehouse, which is represented by 25MB of data stored in binary trees, is assigned one active customer. One thread is created for each warehouse. SPECjbb is a memory resident benchmark.

In addition to these two Java server benchmarks, we run five SPECint2000 benchmarks [18] on the two platforms. This allows us to compare the multithreaded Java server applications to more traditional workloads. We use 255.vortex, 300.twolf, 176.gcc, 252.eon, and 186.crafty, which cover a wide range of application sizes and also contain the only SPECint2000 benchmark written in C++.

3.3. Measurements

We use the hardware performance monitors built into each microprocessor to make performance measurements. Each performance monitor has eight counters that can be programmed to count a variety of processor events. The list of countable events differs between the two machines, but many important events can be counted on both. We interface with the performance monitor using the IBM-supplied performance monitor API and pmcount (a utility that allows the user to interface with the performance monitor), both of which are AIX kernel extensions. Since we only want to collect performance monitor counts for VolanoMark while client connections are being made and not during server startup or shutdown, we send signals to a wrapper that makes API calls to start counting after server startup and stop counting before server shutdown. Similarly, since we only want to do performance monitoring on SPECjbb during the two-minute "measurement period," we instrument the code for SPECjbb (modifying only Company.java) to send signals to a wrapper that makes API calls to start counting at the beginning of the measurement period and stop counting at the end of the period. While pmcount is simpler to use, requiring only a list of events and the executable to count for as arguments, it does not allow this kind of selective counting. However, we do use pmcount for the SPECint benchmarks, since we count for the entire workload in those cases.

For VolanoMark, we run the client on a separate machine. Each chat room has 20 users, while the number of chat rooms is varied from 1 to 40, resulting in a number of connections ranging from 20 to 800. Since VolanoMark creates two threads for every connection, this results in a number of connection threads ranging from 40 to 1600. For SPECjbb, we vary the number of warehouses from 1 to 25. One thread is created for each warehouse.

4. Results

Table 1 and Table 2 compare the Java server benchmarks to the SPECint benchmarks on the RS64-

III and POWER3-II, respectively. VolanoMark is run with 1,10, and 30 chat rooms (indicated as vol01, vol10, and vol30)), and SPECjbb is run with 1, 10, and 25 warehouses (indicated as jbb1, jbb10, and jbb25). The metrics collected are similar to those collected by Bhandarkar et. al. [3].

Table 1. Java servers vs. SPECint2000 (RS64-III)

	bmark	os cyc %	CPI	data refs/instr	mem trans/1000 instr
Java server	vol30	47.06	1.76	0.34	5.68
	vol10	64.47	2.17	0.37	8.31
	vol01	84.68	3.68	0.35	22.93
	jbb25	0.37	1.52	0.33	4.95
	jbb10	0.41	1.45	0.34	3.84
	jbb01	0.29	1.26	0.34	1.73
SPECint2000	gcc	0.78	1.09	0.39	0.41
	crafty	0.15	0.89	0.34	0.01
	eon	0.15	1.36	0.62	0.00
	twolf	0.14	1.43	0.38	0.14
	vortex	0.26	1.07	0.41	0.51

As the tables indicate, VolanoMark spends a high proportion of its execution cycles in kernel mode (os cyc %). This phenomenon is likely due both to the fact that it spends a great deal of time sending and

Table 2. Java servers vs. SPECint2000 (POWER3-II)

	bmark	os cyc %	CPI	data refs/instr	mem trans/1000 instr
Java server	vol30	54.11	1.44	0.34	6.00
	vol10	59.83	1.59	0.38	4.43
	vol01	62.78	1.78	0.39	6.59
	jbb25	0.33	1.27	0.33	5.30
	jbb10	0.39	1.25	0.33	3.90
	jbb01	0.40	1.17	0.34	1.73
SPECint2000	gcc	0.92	0.84	0.44	0.30
	crafty	0.14	0.74	0.40	0.00
	eon	0.13	0.90	0.69	0.00
	twolf	0.14	1.21	0.39	0.01
	vortex	0.28	0.78	0.54	0.48

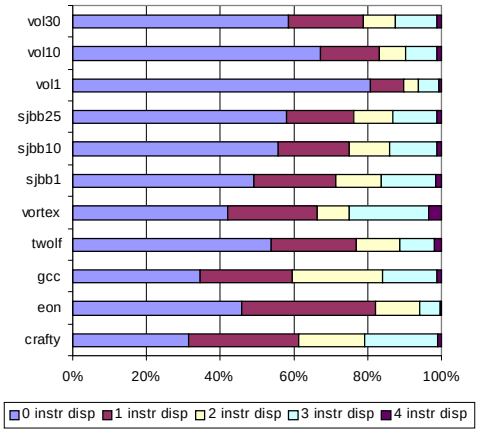
receiving messages over the network and to the fact that the number of threads in VolanoMark is very large, requiring the OS to spend a significant amount of time in thread scheduling routines. The user code is concerned mainly with distributing messages, which is a relatively simple task. We can also see that VolanoMark exhibits a higher CPI than the SPECint benchmarks, which is understandable since OS code is known to have a higher CPI than user code [8]. Since SPECjbb2000 contains no network component, has far fewer threads than VolanoMark, and is memory resident and therefore does not generate many page faults, it has a very small proportion of cycles spent in kernel mode. The same is true for the SPECint benchmarks.

Also, Table 1 and Table 2 show the data references per instruction and the memory transactions per 1000 instructions for the Java server and SPECint workloads. On the average, the Java server workloads generate less data references per instruction than the SPECint workloads, with some of the SPECint workloads far exceeding them, but the Java server workloads still generate considerably more memory transactions per instruction, by one to three orders of magnitude. This is an interesting observation that will be discussed later.

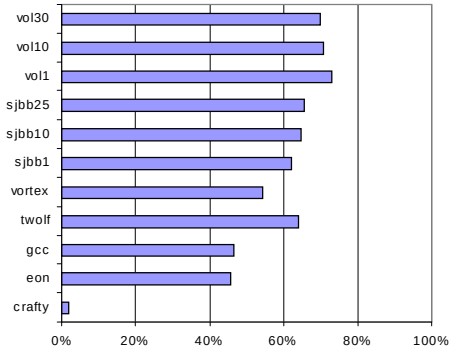
4.1. Dispatch Behavior

Both the RS64-III and the POWER3-II can dispatch up to four instructions per cycle (dispatch for the RS64-III meaning the cycle in which the instruction is sent directly to the execution unit, and dispatch for the POWER3-II meaning the cycle in which the instruction is sent to the execution unit reservation station). From Figure 3 it seems that our machines have more difficulty exploiting ILP in the Java server benchmarks than in the SPECint benchmarks. For almost all of the Java server benchmarks on the RS64-III, zero instructions are dispatched for over 50% of the execution cycles (the lone exception being sjbb1). Only one SPECint benchmark, twolf, 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 Java server benchmarks on the POWER3-II have zero instructions dispatched for more than 60% of the execution cycles, while only twolf crosses 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 pipeline delays are being created in the dispatch stage.

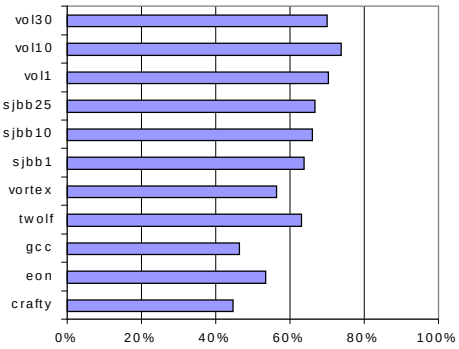
It should be noted that the dispatch stage in these machines is not the stage in which operands are read—in the POWER3-II, it is the stage in which the instructions are sent to the reservation stations, and in



(a) Dispatch profile, RS64-III



(b) Percentage of zero instructions retired cycles, POWER3-II



(c) Percentage of zero instructions dispatched cycles, POWER3-II

Figure 3. Dispatch behavior

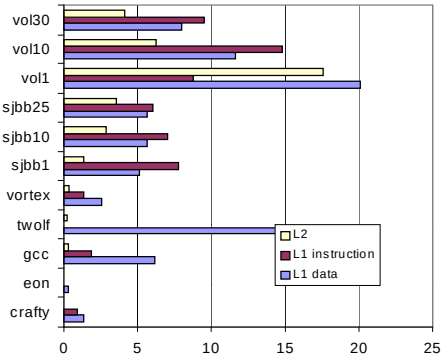
the RS64-III (which, being an in-order machine, has no reservation stations) it is the stage in which the instructions are sent directly to the execution units. In both machines, operands are read in later stages. Therefore, delays in dispatch in these machines are not necessarily due to dependencies between instructions that limit exploited ILP. Nevertheless, dispatch in the RS64-III is stalled if the operand read stage (which directly follows the dispatch stage) is stalled due to instruction dependencies. In the POWER3-II, dispatch can be stalled if the execution unit reservation stations fill, which can occur if dependencies between instructions prevent instruction issue. Therefore instruction dependencies do affect dispatch, and the above dispatch numbers are, to a degree, reflective (though more so in the RS64-III) of exploited ILP. These numbers seem to indicate that the processors cannot exploit as much ILP in the Java server workloads as they can in the SPECint workloads, which is, as mentioned above, an observed characteristic of SPECjvm running on a Java interpreter..

4.2. Cache and TLB Performance

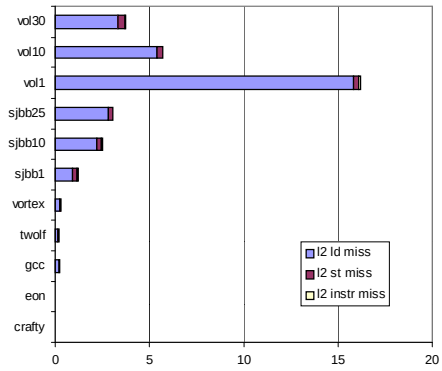
As mentioned earlier, the Java server workloads generate significantly more memory transactions per instruction than the SPECint workloads. And, as one might expect from a higher number of memory accesses per instruction, Figure 4a and Figure 4c show that the Java server workloads exhibit poorer cache performance than the SPECint workloads on both machines, particularly in the instruction cache and L2 cache.

High instruction cache miss rates have also been observed in server applications written in C or C++ [1,2]. Just-in-Time compiling (which our JDK uses) might also contribute to higher instruction cache miss rates for Java applications. With a JIT, bytecode is dynamically compiled into native code, and as a result, code for consecutively called methods may not lie in contiguous address spaces. Thus the instruction data spatial locality can be expected to be poor, causing higher instruction cache miss rates. Also, not surprisingly, 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 Java 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

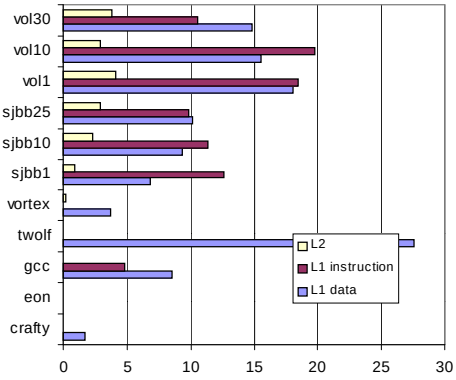
size for performance. Figure 5 shows that the Java server benchmarks cause more instruction TLB misses than the SPECint benchmarks on the RS64-III, and a similar pattern is observed for



(a) Cache misses per 1000 instructions, RS64-III

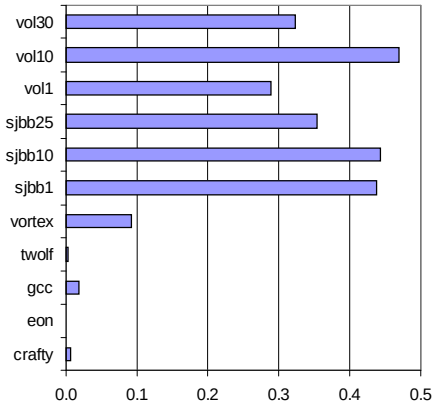


(b) L2 miss components per 1000 instructions, RS64-III

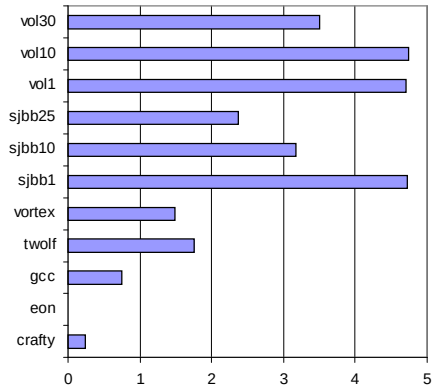


(c) Cache misses per 1000 instructions, POWER3-II

Figure 4. Cache behavior



(a) ITLB misses per 1000 instructions, RS64-III



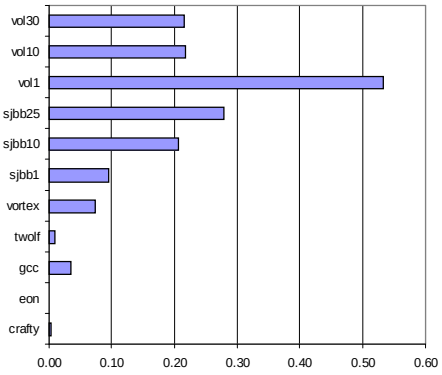
(b) TLB misses per 1000 instructions, POWER3-II

Figure 5. TLB behavior

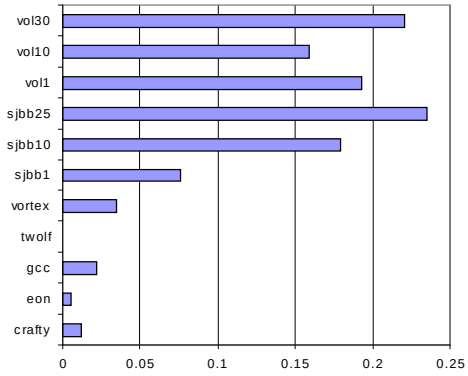
TLB misses on the POWER3-II (ITLB miss count not available on POWER3-II). This TLB performance data is further evidence that Java server benchmarks have a large/scattered instruction footprint. (Note: the RS64-III has a much smaller ITLB misses per instruction count than the POWER3-II's TLB misses per instruction count because its Instruction Effective to Real Address Table (IERAT), which caches address translations and obviates the use of the ITLB if there is a hit, seldom misses.)

Figure 4b shows the components (load misses, store misses, and instruction misses) of L2 misses for the RS64-III. (These counts were not available on the POWER3-II.) It is clear from this figure that most of the L2 misses for the Java server workloads are generated by load references. While twolf shows a

data cache miss rate comparable to Volano, its data appears to be L2 resident.



(a) RS64-III



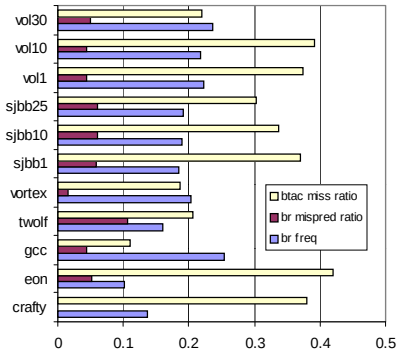
(b) POWER3-II

Figure 6. L2 miss ratio

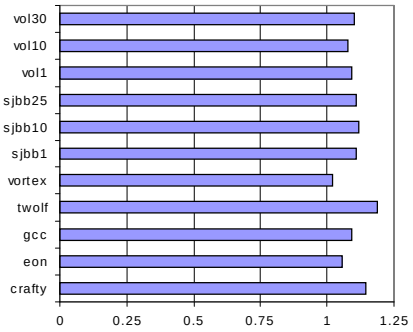
Figure 6, which shows the L2 miss ratios (as opposed to misses per 1000 instructions) on each machine, confirms that the Java server benchmarks are putting more pressure on the L2 than the SPECint benchmarks. We cannot explain this behavior with certainty, but a reasonable explanation could be that the Java server benchmarks have a much larger data footprint than the SPECint workloads (though we cannot obtain the data set size for VolanoMark, we know that each warehouse in SPECjbb uses 25MB of data, while the SPEC workloads are for the most part L1 and at worst L2 resident) and therefore generate more capacity L2 misses (hence the much higher number of memory transactions per instruction seen in Table 1).

4.3. Branch Behavior

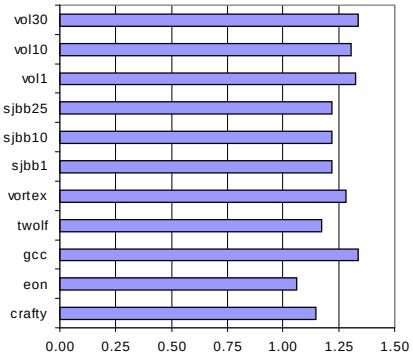
Figure 7a indicates that the POWER3-II's branch



(a) Branch prediction behavior, POWER3-II



(b) Speculation factor, POWER3-II



(c) Speculation factor, RS64-III

Figure 7. Branch behavior

prediction mechanism works as well for the Java server programs as for the SPECint benchmarks (branch prediction numbers for RS64-III not shown because it does not employ dynamic branch prediction). Figure 7b and Figure 7c show that the speculative factors (instructions dispatched/instructions executed) of the

Java server benchmarks are within the range of SPECint2000, indicating that the two sets of benchmarks have much the same effect on speculative execution. However, the Java server benchmarks (with the exception of vol30) exhibit, on the average, worse BTAC (Branch Target Address Cache) performance than gcc, twolf, and vortex. 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 Java server code. Further, eon, which shows BTAC performance similar to the Java 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 [15].

4.4. CPI Components

Figure 8 compares the Java server benchmarks to the SPECint benchmarks on the RS64-III from another perspective: CPI components per instruction. (These events are not countable on the POWER3-II.) The

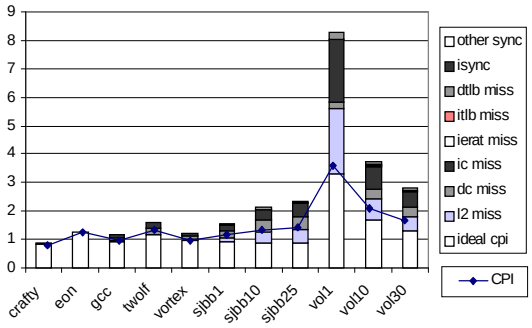


Figure 8. CPI components, RS64-III

stalls in the above figure do not comprise a comprehensive list, but they are the significant memory access 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 that indicates the non-overlapped total amount of storage related stalls (i.e. multiple storage related stalls in one cycle count 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, as could be predicted from the earlier discussion of cache misses, that the Java server benchmarks incur significantly more instruction cache stalls and L2 cache stalls than the SPECint benchmarks, and further, that these along with ideal CPI (which is determined by internal resource conflicts that we cannot count for) are responsible for most of the total CPI. For SPECjbb, data cache miss stalls also play a large role in the 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 Java server benchmarks, Figure 8 shows that the CPIs of the benchmarks are lower than the sum total of the CPI components, which indicates the effectiveness of the RS64-III's superscalar pipelined architecture in hiding some of the storage latency.

5. Conclusion

We performed a comparison of two Java server benchmarks, SPECjbb2000 and VolanoMark2.1.2, with selected benchmarks from SPECint2000 on two IBM PowerPC architectures, the RS64-III and the POWER3-II. We find that our Java server applications differ from SPECint in several ways:

- Clearly, instruction stream behavior is particularly poor for these Java server workloads. High instruction cache, ITLB, and BTAC miss rates are observed. These point toward a large or scattered instruction footprint. Instruction cache stalls make up a substantial component of the CPIs of these workloads, while they are near negligible in the SPECint workloads.
- We also see that L2 performance is a major factor in overall performance for the Java server workloads. L2 misses per instruction and per L2 reference are significantly higher than those for SPECint2000. L2 load misses make up the vast majority of the Java server benchmarks' L2 misses, due possibly to a large data footprint that causes a higher proportion of L2 capacity misses. Clearly, if one is to study the impact of Java server applications on modern processor architectures, L2 performance must not be neglected.
- In addition, these Java server workloads have a high proportion of zero dispatch cycles, suggesting that ILP is not very easily exploited in these workloads.

Given the significant differences between our two PowerPC architectures, the RS64-II being an in-order

execution machine with static branch prediction and the POWER3-II being a highly aggressive out-of-order execution machine, the fact that the above characteristics were found on both platforms suggests that they are real properties of the workload and not machine-dependent.

6. Acknowledgments

We would like to thank Steve Stevens of the IBM Austin PowerPC Performance group for his encouragement and support, Rick Eickemeyer of IBM Rochester for his assistance in calculating CPI components and advice on performance metrics, and Steve Kunkel and Frank O'Connell of IBM for their help in understanding the RS64-III and POWER3-II architectures. Thanks also go to Yue Luo of the Laboratory for Computer Architecture at the University of Texas at Austin Department of Electrical and Computer Engineering for his helpful comments and suggestions.

This study was funded by a grant from the IBM Austin Center for Advanced Studies.

7. References

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

[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] J. Borkenhagen and S. Storino. Fourth Generation 64-Bit PowerPC-Compatible Commercial Processor Design. White Paper, IBM Corporation, http://www.rs6000.ibm.com/resource/technology/nstar.html, January 1999.

[6] H.W. Cain, R. Rajwar, M. Marden, and M.H. Lipasti. An architectural Evaluation of Java TPC-W. In *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture*, 2001.

[7] Q. Cao, P. Trancoso, J.-L. Larriba-Pey, J. Torrellas, R. Knighten, Y. Won. Detailed characterization of a Quad Pentium Pro Server Running TPC-D. In *Proceedings of International Conference on Computer Design*, 1999.

[8] 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.

[9] 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.

[10] Y. Luo and L.K. John. Workload Characterization of Multithreaded Java Servers. *Technical Report TR-010815-01, Department of Electrical and Computer Engineering, University of Texas at Austin*, June 2001, http://www.ece.utexas.edu/projects/ece/lca.

[11] 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, pp. 145-156.

[12] S. Oaks and H. Wong. Java Threads, 2nd Edition, O'Reilly and Associates, January 1999.

[13] 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.

[14] M. Papermaster, R. Dinkjian, M. Mayfield, P. Lenk, B. Ciarfella, F. O'Connell, and R. DuPont. POWER3: Next Generation 64-bit PowerPC Processor Design. White Paper, IBM Corporation, 1998.

[15] 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.

[16] 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. 307-318.

[17] B. Rychik and J.P. Shen. Characterization of Value Locality in Java Programs, Workshop on

Workload Characterization, ICCD, September 2000.

[18] P. Seshadri and A. Mericas. Workload Characterization of Multithreaded Java Servers on Two PowerPC Processors. In *Proceedings of Fourth Annual Workshop on Workload Characterization*, Austin, Texas, December 2001, pp. .

[19] SPEC Benchmarks, http://www.spec.org

[20] Volano LLC. VolanoMark benchmark. http://www.volano.com/benchmarks.html