

Main Memory DBMS on Modern Processors, a Simulation Approach for Database Performance Characterization

Xiang Xiao and Jaehwan John Lee

ECE Department, Purdue School of Engineering and Technology

Indiana University-Purdue University Indianapolis

Abstract

Database applications are an important type of workloads that is very different from other types of application workloads such as SPEC benchmarks. In the past, much research has been devoted to evaluate performance characteristics of database workloads on various processor architectures. While most of the previous performance evaluations have been done using hardware counters, many recently developed full-system simulators have provided another method for performance characterization. With the simulation paradigm, some architectural features of processors that were hard to examine using hardware counters are now accessible. However, results of simulations have to be validated against real machine executions to ensure their fidelity. In this paper, we extend the Simics full-system simulator to measure performance of architectural features of a simulated Pentium 4 alike PC running TPC-H workload on MonetDB (a main memory database management system). Then, we compare our simulation results with some well-established results published in literature previously. Our contributions are: (i) a flexible simulation methodology to characterize database performance; (ii) a close study of TPC-H benchmark performance including bandwidth measurement between memory and CPU, which has not been revealed in previously published research; (iii) validation of the simulation result against real machine measurement. Specifically, we find that our simulation results are mostly consistent with previously published results. In our simulation, approximately 83% of query execution time in the database is spent on various stalls. Memory stalls and resource-related stalls are the most prominent ones for all TPC-H queries under test.

Index Terms

Full system simulation, out-of-order execution processor, main memory database, TPC-H benchmark.

I. INTRODUCTION

Database applications are a very important type of workloads that is very different from other workloads such as SPEC benchmarks [1]. For example, database applications have high context switch rates [15]. In addition, database applications often execute fewer loops and use non-looping branch instructions [15]. In contrast, SPEC programs usually have small instruction working sets and tight loops [15]. Much research has been done in the past to characterize performance of database applications for the purpose of designing better systems for database servers. Many of these studies have used hardware counters as their tools for performance characterization mainly because hardware counters are convenient and since results generated by them are trustworthy. However, architectural parameters of the systems are fixed when measurements are made. Thus, it is difficult to explore different architectural parameters and study their effect on the performance. Furthermore, information on the system performance is provided through a limited number of hardware counters. Any other information, even if desired, is not accessible. On the contrary, performance characterization using simulation provides great flexibility in system configurations. Moreover, there is virtually no limitation to what can be measured. Recent advancement in full system simulators makes it feasible to measure database performance using simulation approach within acceptable time and quality. Nevertheless, simulation results need to be validated against real machine execution.

In this study, we use the Simics full system simulator [11] to measure performance of architectural features of a Pentium 4 alike simulated PC running MonetDB [2] executing TPC-H benchmark [3]. Then, we validate our simulation results against real machine execution by comparing our results with previously published findings. Much of previous research has shown that modern processor advances, such as out-of-order execution, branch prediction, speculative execution and superscalar issue and retire, have not improved performance for database workloads as effectively as for SPEC benchmarks. It is reported that processors spent a large portion of the execution time on stalls. We also have similar results from our study with a few differences. Overall, we find that in average more than 80% of query execution time is spent on various stalls.

Several past studies have examined architectural impacts on performance characteristics of transaction processing database workloads on symmetric multiprocessors (SMP). Most of the studies have focused on benchmarks such as Transaction Processing benchmark One (TP1), Transaction Processing Performance Council benchmarks such as TPC-A, TPC-B and TPC-C [6], [8], [15]–[19]. In particular, [6] and [17] explore the effectiveness of out-of-order execution for the TPC-B workload. [17] finds that the processor

spends over 50% of its execution on cache miss stalls. In [15], TPC-A and TPC-C workloads were studied. It is found that these database workloads exhibit large instruction footprints with different branch behavior than many scientific workloads such as SPEC benchmark. In [18], how to assign processes to different processors on an SMP system to avoid bandwidth bottlenecks is explored. Some other researchers have studied On-Line Transaction Processing (OLTP) workloads [10], [13], [14] and Decision Support System (DSS) workloads such as TPC-D [20]. These studies have found that DSS workloads benefit more from out-of-order processors with increased instruction-level parallelism than OLTP. For both DSS and OLTP workloads, memory stalls are a major bottleneck.

The rest of this report is organized as follows. Section II describes the measurement framework for performance characterization. Section III presents our experimental setup. Section IV elaborates on the simulation results. Section V concludes this report.

II. QUERY EXECUTION ON OUT-OF-ORDER PROCESSORS: A SIMULATION PERSPECTIVE

In this section, we first present the framework for the analysis of query execution time on database systems, which has been introduced by Ailamaki *et al.* [5]. Next, we explain how we extend Simics Micro-Architecture Interface (MAI) [4] to mimic the hardware behavior of an out-of-order processor more accurately under full system simulations. Lastly, we describe the TPC-H database workloads [3] and MonetDB [2] that we use to execute TPC-H queries.

A. Query Execution Model

A framework that models an out-of-order pipelined processor and determines different components of query execution time on the out-of-order processor was described by Ailamaki *et al.* [5]. This framework assumes an out-of-order processor to contain several pipeline stages, as illustrated in Figure 1.

When some operations stall the pipeline, the out-of-order processor utilizes techniques such as non-blocking cache, out-of-order instruction execution and speculative execution with branch prediction to overlap the stall time with other useful work. However, a processor stalls even with these techniques. Thus, the execution time of a query on such an out-of-order processor consists of useful computation time and stall time due to a variety of reasons. Ailamaki *et al.* classified stall time into several categories, as listed in Table 1.

Using Ailamaki's measurement framework and notations in Table 1, the execution time of a query (T_Q) can be calculated by Equation 1.

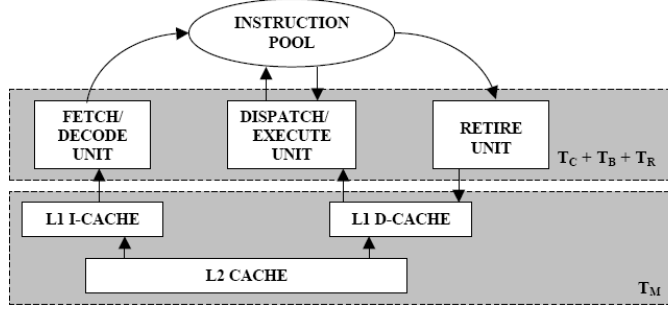


Figure 1. Simplified block diagram of the interactions of an out-of-order pipelined processor (Courtesy of [5]).

Table 1. Breakdown of query execution time.

T_C		computation time
T_M	T_{L1D}	stall time due to L1 D-cache misses
	T_{L1I}	stall time due to L1 I-cache misses
	T_{L2D}	stall time due to L2 data misses
	T_{L2I}	stall time due to L2 instruction misses
	T_{DTLB}	stall time due to DTLB misses
	T_{ITLB}	stall time due to ITLB misses
T_B		branch misprediction penalty
T_R	T_{FU}	stall time due to functional unit unavailability
	T_{DEP}	stall time due to dependencies among instructions
	T_{MISC}	stall time due to platform specific characteristics

$$T_Q = T_C + T_M + T_B + T_R - T_{OVL} \quad (1)$$

where T_C, T_M, T_B and T_R denote useful computation time, stall time due to memory stalls, branch misprediction overhead time and resource-related stall time, respectively. Since out-of-order execution is able to overlap some of these stalls with useful work, some amount of overlapped stall time (T_{OVL}) is subtracted from the sum of aforementioned four types of stalls to obtain the actual query execution time. Note that in [5], T_{OVL} was not measured. The authors approximate T_C by ignoring T_{OVL} in Equation 1. In the next subsection, we will describe how we use the Simics full system simulator [11] to measure the various types of stall time listed in Table 1.

B. Simics MAI Extension for Out-of-order Processor Simulations

We use Simics [11] with its Micro-Architectural Interface (MAI) [4] to simulate an out-of-order pipelined processor. Being a full system simulator, Simics allows us to run unmodified MonetDB [2] and

TPC-H database benchmark suite [3] on a Linux operating system. To enable more accurate simulation of an out-of-order processor, we extend Simics MAI such that different functional units take different numbers of CPU cycles to finish their computation jobs. In original Simics MAI, each instruction spends one cycle during its execution phase no matter what type of instruction it is. In our simulation, we set the execution latencies (in CPU cycles) specified in Table 2 for different types of functional units. Table 2 also shows the number of same type of functional units that are present in the simulated out-of-order pipelined processor. In addition to the extension for supporting multi-cycle functional unit execution, we implant various measuring instruments in MAI source code to calculate the stall time listed in Table 1.

Table 2. Functional unit composition.

Functional Unit	Latency (CPU cycles)	Number of this type of units
Integer ALU	1	4
Memory Load/Store	2	2
Floating point adder	3	2
Floating point/Integer multiplier	6	1
Floating point/Integer divider	24	1

C. Database Workload

The workload used in this study is the TPC-H benchmark suite [3], which consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates a decision support system that examines large volumes of data, executes queries with a high degree of complexity, and gives answers to critical business questions. The TPC-H benchmark consists of eight tables, namely CUSTOMER, LINEITEM, NATION, ORDERS, PART, PARTSUPP, REGION and SUPPLIER. Each of these tables scales linearly as described by the scaling factor except the table LINEITEM whose size is prescribed by the benchmark. Table 3 shows the relative sizes of database items for scaling factor 0.01 and 1. In addition, TPC-H provides tools such as DBGEN and QGEN to create databases and queries for various scaling factors. We have generated all the database information for the scaling factor of 0.01 for this study.

We use Monet database management system (MonetDB) [2] to execute TPC-H queries. MonetDB is an open-source database system used for high-performance applications in data mining, On-Line

Table 3. Table cardinality and database size.

Table Name	Scaling Factor (SF) = 0.01		Scaling Factor (SF) = 1.00	
	Cardinality(Rows)	Size(MB)	Cardinality(Rows)	Size(MB)
CUSTOMER	1,500	0.267	150,000	26.7
LINEITEM	60,175	8.342	6,001,215	641
NATION	25	0.005	25	< 1
ORDERS	15,000	1.733	1,500,000	173.3
PART	60	0.256	200,000	25.6
PARTSUPP	8,000	1.335	800,000	135
REGION	5	0.002	25	< 1
SUPPLIER	100	0.016	10,000	< 2
Total Size	-	11.6 MB	-	Approx. 1 GB

Analytical Processing (OLAP), Geographic Information Systems (GIS), XML querying and text as well as multimedia retrieval. We install MonetDB on Fedora Core 5 Linux, which runs on a Simics x86 simulated machine with a Pentium 4 alike processor. The diagram of the execution layers of TPC-H queries using Simics is illustrated in Figure 2.

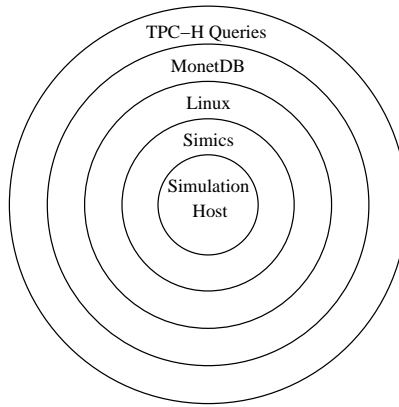


Figure 2. Diagram of layers of the simulation setup.

III. EXPERIMENTAL SETUP

We use an x86-compatible PC as our simulation host, which has an AMD Athlon 64 X2 dual core processor 5200+ and 2GB DDR2 main memory. Simics 3.0 with a modified MAI module is used to perform simulations of an unmodified Linux operating system as well as the underneath out-of-order processor and other hardware units. The system and software specifications of the simulated machine are

listed in Table 4, and detailed cache specifications of the simulated machine are listed in Table 5. Under this experimental setup, we first randomly select a few queries to run on the simulated machine in order to warm up its caches. After the warm-up, we execute each of TPC-H queries from No.1 to 22 (excluding query No.15) ten times. Query No. 15 is excluded because its simulation takes too long time to finish. During the execution, we measure the number of cycles per instruction (CPI) for each query under OS mode and user mode. Furthermore, we collect various types of stall time and calculate their percentage to the entire query execution time. In Section IV, the average CPI and average percentage of various types of stall time are discussed in detail.

Table 4. System and software specifications of the simulated machine

Simics Target	Tango, Pentium 4 processor, 256MB memory, one 19GB IDE disk and one IDE CDROM
OS	Fedora Core 5, Linux kernel 2.6.15
MonetDB	MonetDB server4 version 4.21.0, MonetDB client version 1.21.0

Table 5. Cache specifications of the simulated machine

Characteristic	L1 Data or Instruction	L2 (unified)
Cache size	64 KB	256 KB
Cache line size	64 bytes	64 bytes
Associativity	2	16
Index	Physical	Physical
Tag	Physical	Physical
Write allocate	no	yes
Write policy	write-through	write-back
Replacement policy	LRU	LRU
Non-blocking	yes	yes
Maximum transactions	16	16
Hit latency	4 cycles	7 cycles
Main memory latency	240 cycles	

IV. SIMULATION RESULTS

We executed the workload described in Section II-C on MonetDB, which runs on an x86 Simics simulated machine. In this section, we first present the average CPI of each query and an overview of the execution time breakdown. Then, we focus on each of the important stall time components and analyze it further to determine the implications from its behavior.

A. Average CPI and Execution Time

We executed each query ten times on the simulated machine. However, we observed that the number of instructions executed during the 1st runs for all queries are approximately 60% more than other nine runs on average. This bizarre behavior may be due to initial database startup operations. Thus, we took the average of the last nine runs, which we believe represent the normal behavior of the queries. Therefore, in this report, the average always refers to the average of the last nine runs. Figures 3 and 4 show the average number of cycles executed in user and OS mode of each query and the percentage of the contribution of user and OS code to the total execution time, respectively. Figure 5 shows the average user and OS cycle-per-instruction (CPI) of each TPC-H query.

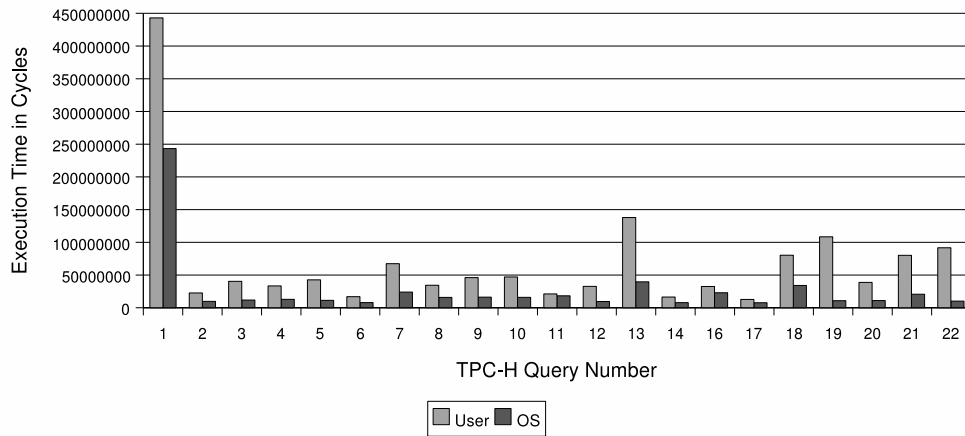


Figure 3. Execution time in cycles of each TPC-H query under test.

It is observed that on average a query spends 73% of its execution time in user database code, which has an average CPI of 3.22. The remaining time is spent in the operating system at an average CPI of 6.2, nearly two times the user database CPI. These results are roughly consistent with those reported in [13]. Keeton *et al.* [13] reported a user CPI of 1.37 and an OS CPI of 2.87 for Informix Online Dynamic Server [7] running OLTP on a quad Pentium Pro SMP. They found roughly 76% to 80% of the execution time was spent on user database code. To better understand the characteristics of the execution of TPC-H queries, we further decompose the execution time into computation and stall cycles, which is discussed in detail from Section IV-B to Section IV-E.

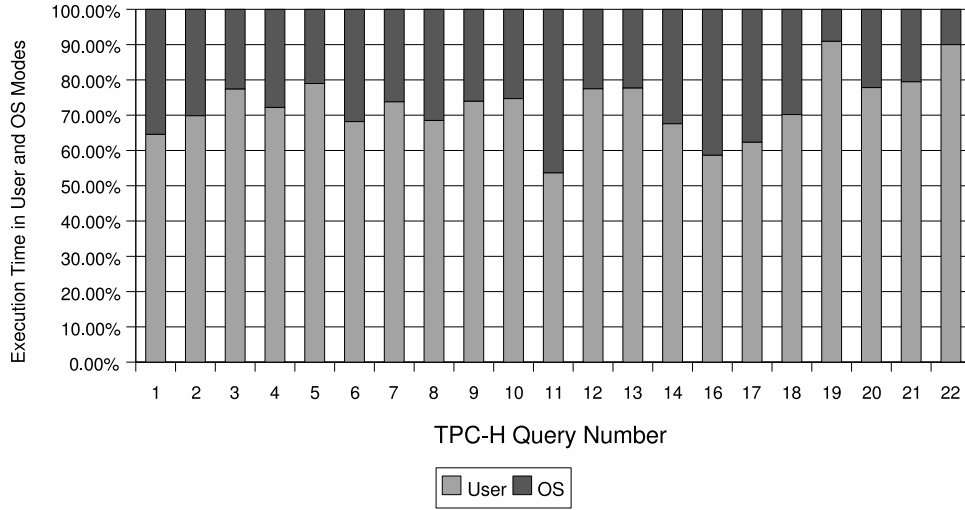


Figure 4. Percentage of user and OS execution time of each TPC-H query under test.

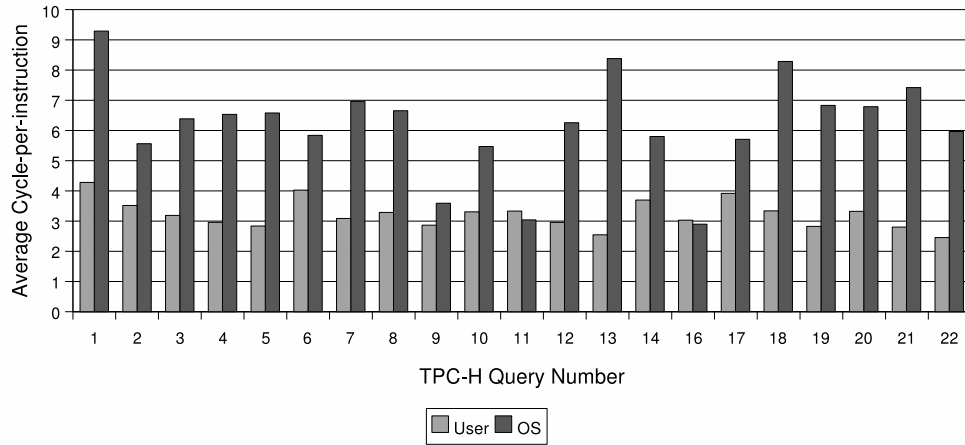


Figure 5. Average user and OS cycle-per-instruction (CPI) of each TPC-H query under test.

B. Query Execution Time Breakdown

Figure 6 summarizes the average execution time breakdown for each of the 21 TPC-H queries. Each bar shows the contribution of the four time components (T_C , T_M , T_B and T_R) as a percentage of the total query execution time. The computation time (T_C), the memory stall time (T_M), the branch misprediction stall time (T_B), and the resource stall time (T_R) across all 21 queries comprised on average of 16.7%, 37.3%, 13% and 33% of the entire query execution time, respectively. Similar results have been presented in the literature [9], [13]. Cvetanovic and Donaldson [9] reported that for roughly 80% of the time the processor was stalled, in which resource stalls comprised of 49%. Keeton *et al.* [13] also reported that

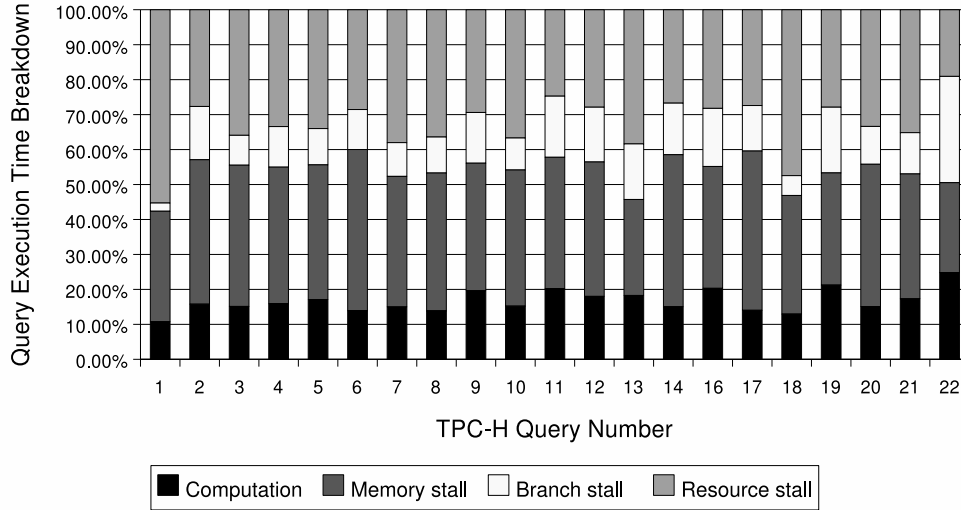


Figure 6. Query execution time breakdown into the four time components.

for 256 KB L2 cache size, stall time made up to nearly 75% to the entire query execution time. These results indicate that although minimizing memory stalls (T_M) is important for performance improvement, solely reducing memory stalls may not be sufficient. It is because the other two components (T_B and T_R) are also significant. Thus, optimizing for only one kind of stalls may shift the bottleneck to the other kinds. Techniques to improve database performance should reduce all three kinds of stalls to effectively decrease the execution time.

C. Memory Stalls

Many efforts have been made by researchers to minimize the stall time due to memory hierarchy. Although these techniques are successful within the context in which they were proposed, a closer look at the execution time breakdown shows that there is significant room for improvement. To further investigate the causes of memory stall time, memory stall time (T_M) is broken down into six components: T_{L1D} (L1 D-cache miss stalls), T_{L1I} (L1 I-cache miss stalls), T_{L2D} (L2 cache data miss stalls), T_{L2I} (L2 cache instruction miss stalls), T_{ITLB} (ITLB miss stalls) and T_{DTLB} (DTLB miss stalls). Due to the limitation of Simics, we do not measure T_{ITLB} and T_{DTLB} . Figure 7 shows the breakdown of T_M into four components (i.e., T_{L1D} , T_{L1I} , T_{L2D} and T_{L2I}) while Figure 8 shows various cache miss rates. In addition, the overall L2 cache miss rate (including both data and instruction misses) for each query is shown in Figure 9. In Figure 7, each type of stall time is calculated as “the number of cache misses \times

miss penalty”. In Figure 8, cache miss rate is equal to “the number of cache misses / the number of cache requests”. Thus, stall time can also be calculated as “cache miss rate \times the number of cache requests \times miss penalty”. Note that the reason for query No.1 L2-I stall time portion is tiny although L2-I miss rate is 36% is that the number of L2-I requests is small. Although L2-I miss rates are relatively high (37.4% on average across all queries), the L2 cache, as an unified cache, has an overall miss rate of 2.9% on average. Therefore, the L2 cache performs well in terms of its overall miss rate. Thus, the high L2-I miss rate may be due to (but not limited to) two reasons: (i) The instructions are cached very successfully in the L1-I cache (0.3% miss rate on average). the L2 cache cannot provide extra caching very effectively on instructions that are not contained in the L1-I cache. (ii) Because instructions are seldom missed in the L1-I cache, they are less frequently accessed in the L2 cache. Hence, instruction blocks in the L2 cache are more likely to be replaced by data blocks due to the LRU replacement policy of the L2 cache. Therefore, when a instruction request is missed in the L1-I cache, this request is less likely fulfilled in the L2 cache.

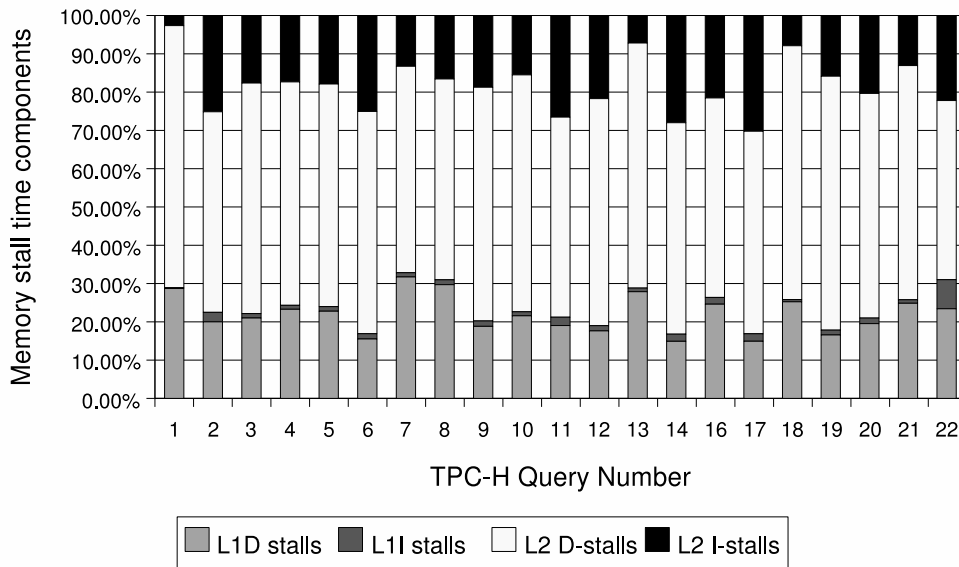


Figure 7. Contribution of the four memory components to the memory stall time (T_M).

As seen from Figure 7, T_{L2D} is the most significant portion of memory stalls across all queries (ranging from 47% to 58%). T_{L1D} and T_{L2I} also contributes important parts of memory stalls for all queries. T_{L1I} only contributes a very small portion of memory stalls except for query No. 22, for which T_{L1I} accounts

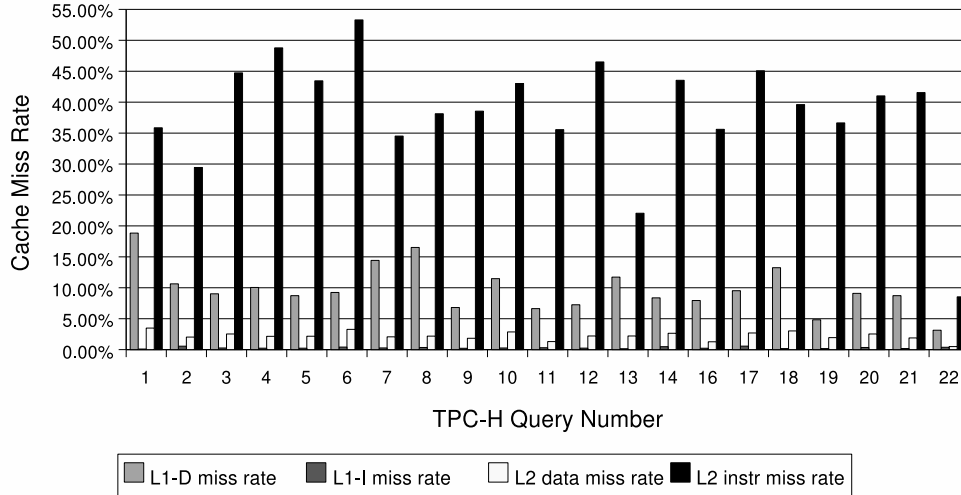


Figure 8. Cache miss rates.

for 8% of its memory stalls. For others, T_{L1I} accounts for from 1% to 2%. The average percentages of all components contributing to memory stalls across all 21 TPC-H queries are listed in Table 6.

Table 6. The average percentages of all memory stall components.

	T_{L1D}	T_{L1I}	T_{L2D}	T_{L2I}
Percentage in T_M	22.0%	1.6%	58.1%	18.3%

Although it is commonly found that memory stalls contribute to a large portion of the query execution time, other researchers have found different breakdown of memory stalls from ours. Ailamaki *et al.* [5] reported that T_{L2D} and T_{L1I} were the largest and second largest components of memory stalls. The discrepancy of the impact of T_{L1I} on memory stalls may be due to the difference in the sizes of L1 instruction (L1I) caches used between Ailamaki’s work and our study. We used a 64KB L1I cache for our simulated system while the machine used in Ailamaki’s study had a 16KB L1I cache. Ailamaki *et al.* also reported the L1 D-cache miss rate around 2% and never above 4%. In our study, we observed that the average L1 D-cache miss rate of all 21 queries is 10%, as shown in Figure 8. As a result, Ailamaki *et al.* found T_{L1D} insignificant while we found T_{L1D} accounts for 22% of memory stalls on average across all 21 queries. One of the reasons that could cause the difference in results between Ailamaki’s and our study is that TPC-H workload used in our study consists of more complex data access pattern

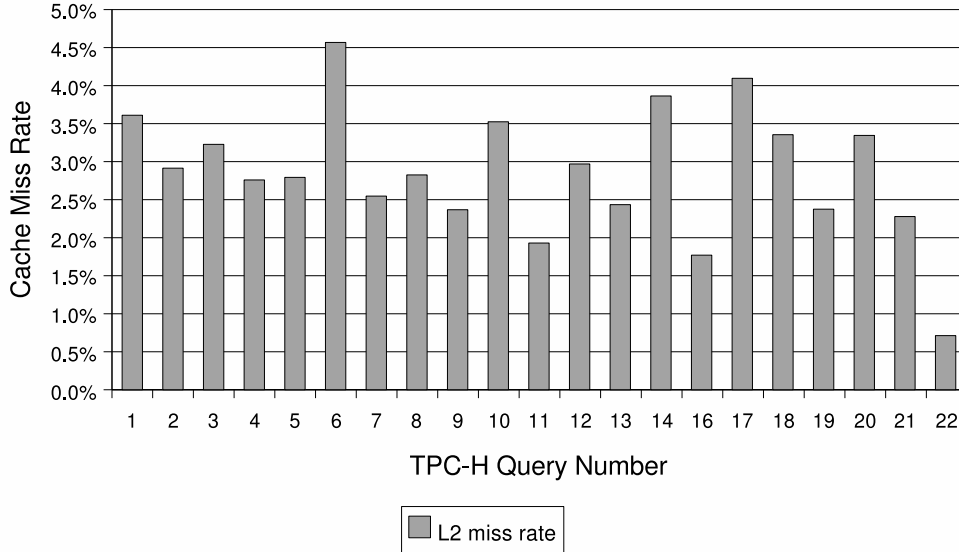


Figure 9. L2 cache overall miss rates.

and inquiry of larger data access volume, which the simplified workload used in Ailamaki’s study does not have.

D. Branch Mispredictions

The branch predictor used in the simulated machine has a 256-entry buffer to store both branch instruction PCs and target PCs. Branch miprediction stall (T_B) accounts for from 2% to 30% of query execution time for different queries, as shown in Figure 6. The average percentage of T_B in query execution time is 13%. Ailamaki *et al.* [5] reported similar results. In [5], branch misprediction stalls were found to account for 10-20% of the query execution time. It has also been shown that increasing the size of the buffer storing branch targets can reduce branch misprediction stalls [12].

E. Resource Stalls

Resource-related stall time is the time during which the processor must wait for a resource to become available. Such resources include functional units in the execution stage, registers for handling dependencies between instructions and other platform-dependent resources. We found that the contribution of resource stalls to the overall execution time varies from 19% to 55% for different queries. In our study, resource-related stall time (T_R) is further split into dependency stall time (T_{DEP}) and functional unit

availability stall time (T_{FU}). Their contributions to the overall query execution time are illustrated in Figure 10.

Dependency stalls are caused by low instruction-level parallelism opportunity in the instruction pool. Dependency stall occurs when an instruction depends on the results of one or more other instructions that have not yet completed execution. The processor must wait for the dependencies to be resolved in order to continue. Functional unit availability stalls are caused by bursts of instructions that create contention in the execution units. Ailamaki *et al.* [5] reported that T_{FU} is usually very small compared to T_{DEP} except for one commercial database system (its name not revealed) in their study. However, in our study, T_{FU} exceeds T_{DEP} for many queries. In the cases that T_{DEP} is smaller than T_{FU} , mostly they are comparable. This may be due to a limitation of Simics MAI [4], which allows only one instruction to execute in the pipeline for some instructions, such as CALL, PUSH, POP, MOV (see [4] for a complete list), because these instructions modify architectural states immediately after their executions. In those cases, many ready instructions cannot be executed because they are prohibited to enter the pipeline. Thus, the cycles in which these instructions spend on waiting are counted as a part of T_{FU} .

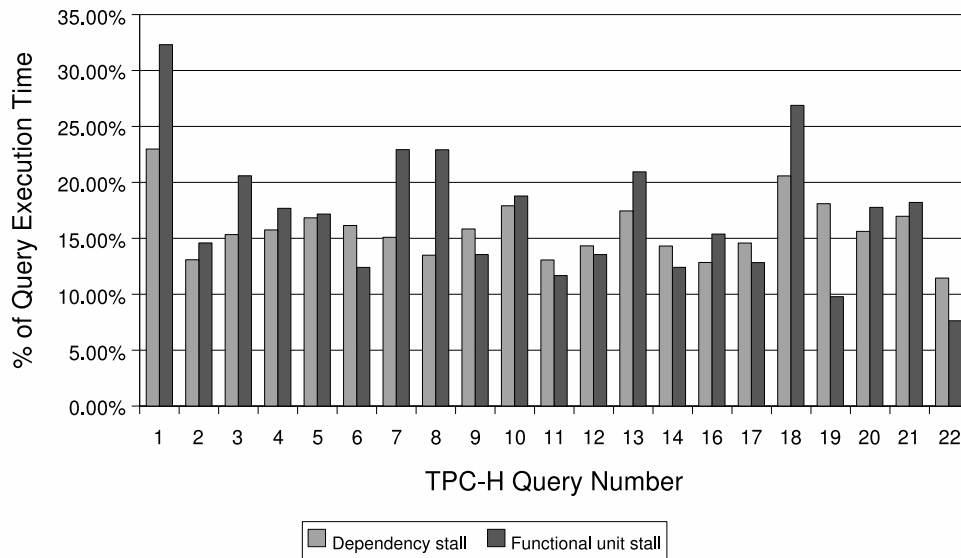


Figure 10. T_{DEP} and T_{FU} contributions to the overall query execution time for 21 TPC-H queries.

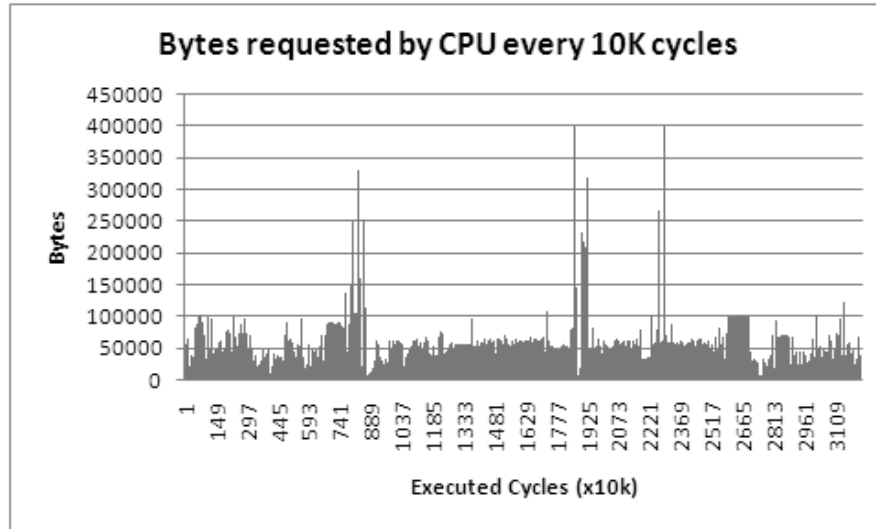


Figure 11. Data request of CPU for TPC-H query No.2 in bytes.

F. Bandwidth Estimation in Query Processing

In addition to the performance characteristics of database workloads, we are also very interested in the bandwidth requirement between memory and CPU. Since it is assumed that a main memory database completely resides in the memory, insufficient bandwidth between memory and CPU may starve CPU and thus degrade query processing performance. To address such concerns, we use Simics to measure the number of bytes that CPU requests from main memory every 10,000 CPU cycles throughout the processing of a query. Figure 11 shows such a data request profile of the processing of TPC-H query No. 2 in our simulation. As seen in Figure 11, although the peak value of data request is 400,000 bytes in 10,000 CPU cycles, the average request is only 40,611 bytes every 10,000 CPU cycles. Thus, considering a CPU with clock frequency of 2GHz for instance, the bandwidth between memory and CPU needs to be 4GB/sec or more in order to sustain CPU's data processing.

V. CONCLUSION

Full system simulators have matured enough in recent years to provide an alternative way to characterize performance of database applications. In this report, we use the Simics full system simulator to run TPC-H queries on MonetDB and evaluate its performance characteristics. We find that nearly 80% of query execution time is spent on various stalls, which is similar to several published findings in the literature. Among these stalls, memory stalls, branch misprediction stalls and resource-related stalls account for

37.3%, 13% and 33%, respectively. Moreover, among various types of memory stalls, L2 cache data miss stalls are the most prominent across all queries under test. We also find that L1-D cache miss stalls and L2 cache instruction miss stalls degrade the performance of TPC-H benchmark by a noticeable amount. These two types of stalls have been reported to be insignificant for simple queries in [5]. Furthermore, we profiled the data request from CPU to memory for TPC-H query No. 2, which reveals that bandwidth of 4GB/sec or more is required between memory and CPU. This provides a lower bound on the required bandwidth between memory and CPU in order to sustain CPU's data processing. In the future, more in depth experiments on database workloads using simulation approaches are going to be carried out. One of the future experiments is to simulate and characterize the performance of TPC-H benchmark with scaling factor 1, which is more often used in performance characterization on real machines.

VI. ACKNOWLEDGMENT

This research utilizes the Simics full system simulator and the MonetDB system. We would like to thank RSFG internal grant that enabled the research performed here.

REFERENCES

- [1] SPEC benchmark suite, Standard Performance Evaluation Corporation, <http://www.spec.org/benchmarks.html>, visited in Dec. 2007.
- [2] MonetDB, <http://monetdb.cwi.nl/>, visited in Dec. 2007.
- [3] TPC-H, <http://www.tpc.org/tpch/>, visited in Dec. 2007.
- [4] *Simics Micro-Architectural Interface, Simics Version 3.0*. 2007.
- [5] A. Ailamaki, D. Dewitt, M. Hill and D. Wood, "DBMSs on a modern processor: Where does time go?" In *The VLDB Journal*, pages 266–277, 1999.
- [6] L. Barroso and K. Gharachorloo, "System design consideration for a commercial application application environment," In *First Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW'98)*.
- [7] Informix Corporation, *Informix OnLine Dynamic Server Administrator's Guide, Vol. 1 and Vol 2*.
- [8] Z. Cvetanovic and D. Bhandarkar, "Performance characterization of the Alpha 21164 microprocessor using TP and SPEC workloads," In *Proc. of HPCA-2*, 1996.
- [9] Z. Cvetanovic and D. Donaldson, "Alphaserver 4100 performance characterization," *Digital Technical Journal.*, 8(4):3–20, 1996.
- [10] R. Eickemeyer, R. Johnson, S. Kunkel, M. Squillante and S. Liu, "Evaluation of multithreaded uniprocessors for commercial application environments," In *Proc. of the 23rd Int'l Symposium on Computer Architecture (ISCA)*, 1996.
- [11] Peter Magnusson et al, "Simics: A full system simulation platform," *IEEE Computer*, 35(2):50–58, 2002.

- [12] R. Hilgendorf and G. Heim, "Evaluating branch prediction methods for an S390 processor using traces from commercial application workloads," In *Presented at CAEVW'98 in conjunction with HPCA-4*, 1998.
- [13] K. Keeton, D. Patterson, Y. He, R. Raphael and W. Baker, "Performance characterization of a quad Pentium Pro SMP using OLTP workloads," In *Proceedings of the 25th ISCA*, pages 15–26, 1998.
- [14] J. Lo, L. Barroso, S. Eggers, K. Gharachorloo, H. Levy and S. Parekh, "An analysis of database workload performance on simultaneous multithreaded processors," In *Proc. of the 25th Int'l Symposium on Computer Architecture (ISCA)*, pages 39–50, 1998.
- [15] A. Maynard, C. Donnelly and B. Olszewski, "Contrasting characteristics and cache performance of technical and multi-user commercial workloads," In *Proc. of the 6th Int'l Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 145–156, 1994.
- [16] S. Perl and R. Sites, "Studies of Windows NT performance using dynamic execution traces," In *Proc. of the Second USENIX Symposium on Operating Systems Design and Implementation*, pages 169–184, 1996.
- [17] M. Rosenblum, E. Bugnion, S. Herrod, E. Witchel and A. Gupta, "The impact of architectural trends on operating system performance," In *Proc. of the 15th ACM SOSP*, pages 285–298, 1995.
- [18] S. Thakkar and M. Sweiger, "Performance of an OLTP application on symmetry multiprocessor system," In *Proc. of the Int'l Symposium on Computer Architecture (ISCA)*, 1990.
- [19] J. Torrellas, A. Gupta and J. Hennessy, "Characterizing the cache performance and synchronization behavior of a multiprocessing operating system," In *Proc. of the 5th ASPLOS*, pages 162–174, 1992.
- [20] P. Trancoso, J. Larriba-Pey, Z. Zhang and J. Torellas, "The memory performance of DSS commercial workloads in shared-memory multiprocessors," In *Proc. of HPCA-3*, 1997.