Characterization of TPC-H Queries for a Column-Oriented Database on a Dual-Core AMD Athlon Processor

Pranav Vaidya and Jaehwan John Lee
Department of Electrical and Computer Engineering
Purdue School of Engineering and Technology
Indiana University - Purdue University, Indianapolis
{psvaidya, johnlee}@iupui.edu

ABSTRACT

Column-oriented databases are reported to perform orders of magnitude faster than traditional row-oriented databases for Decision Support System (DSS) workloads. However, there have been few investigations on the detailed hardware characterization of DSS benchmarks namely TPC-H for column-oriented databases. In this paper, we characterize the performance of the TPC-H benchmark for a popular column-oriented database called MonetDB running on a dual-core AMD Athlon machine. Specifically, we measure the performance of key microarchitectural components and provide very detailed analysis of the execution time breakdown of MonetDB during the execution of TPC-H queries. We analyze in detail the nature of the various stalls namely cache stalls, branch misprediction stalls and resource stalls in order to identify the causes of these performance impediments in column-oriented databases. We then indicate possible research directions to minimize these stalls in column-oriented databases. Finally, we compare our results with published results on the characterization of TPC-H for row-oriented databases. As opposed to the previous approaches that used system-level monitoring, we used thread-level monitoring of database threads to study the performance of the database alone in isolation from the rest of the system. This enabled us to characterize the performance of databases independent of any other executing processes.

1. INTRODUCTION

In today's global economy, database information systems have become quite pervasive and represent an important class of server applications. Policy-makers in various domains such as business, engineering, military and science leverage these information systems to analyze large amounts of data in order to understand the complex interactions between various system variables. As a result, such information systems remain extremely crucial to the policy and decision making process in various domains. However, the less than optimal performance of databases on modern computing hardware represents an impediment to this policy making process. Recent times have seen substantial academic research and corresponding commercial ventures in the area of column-oriented database systems, including MonetDB [1–4], C-Store [5] and Vertica [6]. These systems claim that their approach offers performance gains of one to two orders of magnitude on read-only (or read intensive) analytical processing workloads such as those encountered in decision support systems [1–6]. However, there have been relatively few detailed studies in analyzing the performance of TPC-H workloads [8] for column-oriented databases on modern hardware. In this paper, we present our perspective on the characterization of TPC-H workloads for a column-oriented database MonetDB run on a modern dual-core AMD Athlon processor [9]. We provide detailed execution breakdown of the time spent in useful computation, L1 and L2 cache stalls, branch misprediction and resource stalls. Thanks to the advancement in hardware and software monitoring tools, we further classify in detail the nature of the resource stalls. By comparing our results with existing published research on the characterization of TPC-H queries on row-oriented databases, we find that column-oriented databases do not dramatically outperform row-oriented databases in utilizing the underlying hardware. Hence, we indicate both current and future research directions that might assist in overcoming the existing limitations of the column-oriented databases.

The outline of this paper is as follows. Section 2 describes the related work in this field. Section 3 describes the nature of the database workload and queries. This section also describes our hardware/software platform and the performance monitoring tool used in this research. Section 4 describes in detail the results from our experiments. Finally, Section 5 presents the summary of results, the corresponding conclusions deduced from the results and the possible research directions that might assist in overcoming performance limitations of current column-oriented databases.

2. RELATED WORK

Since 1980's, the relational model [10] has been the most popular database model. In relational databases, the primary data structure is a relation (or table) that consists of records (tuples). Each tuple is further composed of attributes. Most current databases store records (or tuples) contiguously and are hence called row-oriented databases. On-Line Transaction Processing (OLTP) applications most often use row-oriented relational databases. In OLTP applications, a significant portion of queries involves short-running update queries [11]. Such queries usually scan through entire relations (or tables) looking for a small number of rows to update but using most of the attributes in the tuple [1]. On the other hand, query intensive applications such as On-Line Analytical Processing (OLAP), DSS and data mining show significant difference in their data access patterns. Such applications usually summarize data in multiple dimensional structures such as datacube [12]. Queries are usually medium or long running that access entire relations (or tables) but

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM '08 Napa, CA USA
Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

require only a few columns.

Previous research shows that current row-oriented DBMSs perform very poorly in OLAP (and even OLTP) applications on modern processors [7, 13, 14]. In fact, using hardware features like multithreading on some processors might result in degradation of performance [15]. Modern processors are multicore multiprocessors that are superscalar and simultaneously multithreaded. In superscalar processors, an average 50% of the time is spent on stalls that cannot be overlapped with computational execution for OLTP workloads [7]. Similar findings are observed for OLAP and Decision Support System (DSS) workloads [16]. This means that the processor is not performing computation nearly half the time. L2 data cache and L1 instruction cache misses account for nearly 40%-90% and 5-20% of the total memory stall time, respectively [7]. One of the reasons cited is the large size of each record (or tuple). As record size increases, the number of records stored on L1 and L2 data cache decreases [1]. Usually, most database queries refer to a few attributes. Unnecessary data in terms of non-projected attributes in the query takes up crucial space in the cache. This reduces spatial locality between required attributes within the record and between attribute values stored in neighboring records [1, 7]. There is also evidence that the large record size may cause a large number of L1 instruction cache misses due to inclusion policy of L1 cache into L2 cache [7]. Increased L2 cache misses interfere with L1 cache and thus may cause further data and instruction cache misses. Furthermore, Operating System (OS) context switches may fill the caches with OS specific code/data, resulting in refilling of the cache with database related data and instructions when DBMS code resumes [7]. Memory hierarchy stalls still remain a significant component of execution time on modern processors. With the advent of multicore multiprocessors containing large megabyte caches, data cache stalls seem to account for 46-64% of the execution time [17]. Moreover, L2 hit latencies become the dominating factor of execution time as L2 cache sizes increase. As a result, L1 data cache locality has become critical to improving database performance [17].

Additionally, branch mispredictions account for nearly 10-20% of the stall times. Out-of-order execution of instructions in modern superscalar processors does not alleviate the problem completely. Resource stalls such as dependency stalls and functional unit stalls also account for nearly 10-30% of the total stall time [18]. Dependency stalls occur when execution of one instruction depends on the value produced by the execution of another instruction. Functional unit stalls occur among instructions due to the limited number of execution units provided by the processor. In CISC instruction set, each instruction is further broken down into microinstructions. Often, a compiler is unable to overlap execution as there is no easy way to determine correlation between the instructions produced by the compiler and the microinstructions executed by the processor [19].

Hence, there has been a growing interest in column-oriented databases [1–6]. Research shows that column-oriented or Decomposed Storage Model (DSM) that partitions the relations (or tables) vertically into a collection of two-column tables called Binary Association Tables (BATs) consisting of (key, attribute) pairs might be more suited for OLAP workloads. This is because only required columns can be brought into the cache for processing. However, there have been few investigations in characterizing the performance of TPC-H queries on column-oriented databases. Although Stonebraker et al. [5, 20] presented compelling results on the run-time characterization of a column-oriented database – 'C-store', we believe that there have not been any substantial investigation in the detailed hardware characterization of TPC-H benchmark for column-oriented databases. This is precisely our goal in this paper where we perform detailed characterization of the performance of TPC-H benchmark for a popular column-oriented data-base called MonetDB running on a dual-core AMD Athlon machine. Additionally, we measure the performance of key microarchitectural components and analyze the nature of the various stalls to identify the causes of performance impediments in column-oriented databases such as MonetDB.

3. EXPERIMENTAL METHODOLOGY

This section first briefly describes the TPC-H benchmark workload, then our hardware/software platform and finally the performance monitoring tool used to measure and analyze the TPC-H benchmark performance.

3.1 Database Workload

The TPC-H benchmark [8] is a Decision Support System (DSS) benchmark consisting of complex business-oriented queries against a database schema that models real world business databases. The TPC-H benchmark simulates queries that are more complex than most OLTP transactions. The TPC-H benchmark is designed to stress the database server, and consists of twenty-two queries which must be run as part of the benchmark test. These queries are modeled after real questions that a supply chain business owner might pose to obtain information about his/her business. The queries model questions about pricing issues, part and supply chain issues, customer satisfaction and inventory management issues. Table 1 shows the various database operators exercised by the benchmark.

Table 1: Query types and corresponding database functions

Query	Sequential Scan	Index Scan	Hash Join	Nested Loop Join	Sort
Q1	√				
Q2	√				
Q3			√	√	
Q4		√		√	
Q5				√	√
Q6					√
Q7			√	√	
Q8			√	√	
Q9				√	
Q10				√	
Q11				√	
Q12			√		√
Q13			√		
Q14		√			
Q15				√	
Q16				√	√
Q17				√	
Q18				√	√
Q19				√	
Q20				√	
Q21				√	
Q22			√		

The TPC-H benchmark documentation suggests that database operations should be modeled against a 24x7 running database supporting multiple clients performing ad-hoc queries and updates on all tables except during highly infrequent maintenance sessions. Furthermore, the TPC-H documentation [8] states that the database

Table 2: Table cardinality and database size

Table Name	SF = 0.01		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	2,000	0.256	200,000	25.6
PARTSUPP	8,000	1.335	800,000	135
REGION	5	0.002	5	< 1
SUPPLIER	100	0.016	10,000	< 2
Total Size	-	11.6 MB	-	Approx. 1 GB

Table 3: Hardware platform

Hardware Feature	Description
Number of CPU Cores	2 in an AMD Athlon X2 processor
Execution Units	Three integer execution units, three floating point units, three address generation units and one load-store unit
Branch Predictor	2048 two-way set-associative branch predictor
L1 Cache	Discrete L1 cache per core, Split 64 Kbyte 2-way set-associative Instr. and Data cache
L1 Cache Line Size (for Instr. and Data Cache)	64-byte line size
L2 Cache	Discrete L2 cache per core, 512 Kbyte 16-way set-associative cache
L2 Type	Victim cache
L2 Cache Line Size	64-byte line size
Translation Lookaside Buffer (TLB)	Two-level TLB where each entry maps to 4K pages
L1 Instruction TLB (ITLB)	32 entry fully-associative TLB
L1 Data TLB (DTLB)	32 entry fully-associative TLB
L2 TLB	512 entry four-way set-associative TLB
Chipset	nVidia GeForce 6100/NF410
Memory	4GB DDR2 SDRAM
OS	Fedora Core 8 (codename Werewolf) - Linux Kernel 2.6.23.9 patched with perfmon2 kernel patch release 070725
HD size	80 GB

server may batch together a number of modifications impacting some part of the DSS database. TPC-H consists of eight tables namely *SUPPLIER, PART, PARTSUPP, LINEITEM, CUSTOMER, ORDER, NATION* and *REGION*. The database tables *REGION* and *NATION* are of constant sizes for all values of the Scaling Factor (SF). All the remaining tables scale linearly as described by the scaling factor except the table *LINEITEM* whose size is prescribed by the benchmark. Table 2 shows the relative sizes of databases for scaling factor 0.01 and 1. In addition, TPC-H [8] 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 = 1.

3.2 Hardware Platform

As part of this work, we have evaluated the performance of a column-oriented database MonetDB on an AMD Athlon 64 X2 dual-core processor with each core running at 2.2 GHz. Table 3 describes in detail the characteristics of our hardware platform.

The AMD Athlon microarchitecture consists of a nine macro-op issue, out-of-order superscalar processor pipeline. The processor has three integer units, three address generation units, three floating point units and one load-store unit. There can be as many as 72 macro-ops in flight at one time. The AMD Athlon processor chip has an integrated 64-bit, dual-ported split L1 cache with a separate snoop port and two-levels of Translation Lookaside Buffers (TLB). Each processor core has a discrete integrated 512 Kbyte L2 victim cache. The data cache has eight banks to support concurrent access by two 64-bit loads or stores [9].

The Athlon microarchitecture uses a two-level TLB to determine physical addresses from virtual addresses generated by the processor. The L1 Instruction TLB (ITLB) and Data TLB are both thirty-two entry fully-associative TLBs each. The L2 TLB is a unified 512-entry, four-way set-associative TLB. Each entry in the TLB maps to a 4K page in memory.

3.3 Software Platform

We ran our experiments using MonetDB on Fedora Core 8 (codename Werewolf) with Linux kernel version 2.6.23.9. This kernel was patched with perfmon2 patch release 070725. Perfmon2 is a generic performance monitoring tool written for Linux systems. It provides a system-call interface to enable applications to monitor their performance. Perfmon2 supports per-thread monitoring of unmodified binaries for multithreaded and multiprocess workloads. Additionally, Perfmon2 supports system-wide monitoring of all processes on the system. Perfmon2 also supports sampling modules for sampling in both system-wide and per-thread mode. In

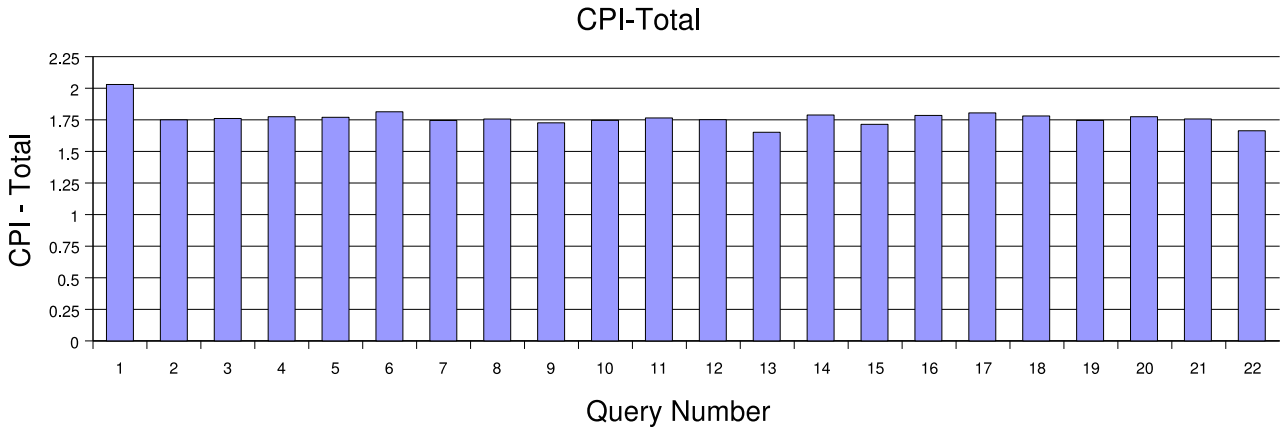
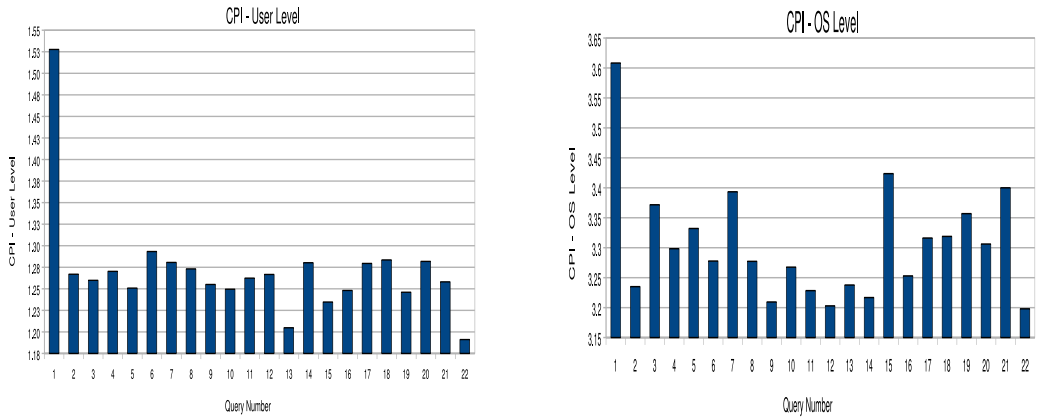


Figure 1: Overall CPI (the lower the better).



(a) CPI-User Level

(b) CPI-OS Level

Figure 2: CPI at user level and OS level.

this research, we have utilized the per-thread monitoring functionality of perfmon2 to evaluate the performance of the multithreaded database server – MonetDB version 4. Using perfmon2 we were able to measure 46 event types at both user and kernel levels. In some cases, we explicitly measured the kernel and user event types to characterize performance in more detail.

4. PERFORMANCE MEASUREMENT

4.1 Cycles Per Instruction (CPI)

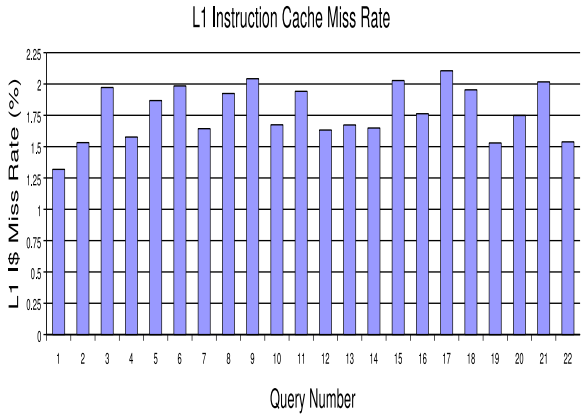
Figure 1 shows the overall CPI[1] for each of the twenty-two queries of the TPC-H benchmark. The AMD Athlon can issue up to three macro-ops per cycle. This pegs the absolute lower bound of CPI at 0.33 where three macro-ops are issued and retired in each cycle. As can be seen from the results, the lowest overall CPI is about 1.65 for Q13 and Q22 while it is highest at about 2.03 for Q1. This large difference between the best case CPI and the CPI of all TPC-H queries indicates that there are still sufficient avenues to explore in order to improve database performance. Q13 and Q22 mainly exercise hash-joins in the TPC-H query (as shown in Table 1). The CPI of Q22 is the lowest because MonetDB uses highly efficient, cache-conscious positional-join and partitioned hash-join algorithms for the join operations. The CPI of Q1 is the highest, which can be attributed to the column-oriented nature of MonetDB. This query performs aggregation operations on several columns of database table *LINEITEM*, where the large size of the data table columns results in significant cache misses as can be seen from Section 4.2.

Figure 2a shows the CPI for queries at user-level, which follows the trend similar to the overall CPI. However, the CPI at OS level shown in Figure 2b is larger than user level CPI for all queries. The CPI at OS level is maximum for Q1 at about 3.6 and minimum at about 3.2 for Q12 and Q22. A closer inspection of the kernel-level functions reveals that the OS performs a lot of complex operations such as page-walks, locking/unlocking with IRQ disable/enable as well as execution of a large number of serialization instructions. Instructions such as page-walks involve moving and copying large data blocks whereas serializing instructions cause the entire pipeline to be drained before they are issued.

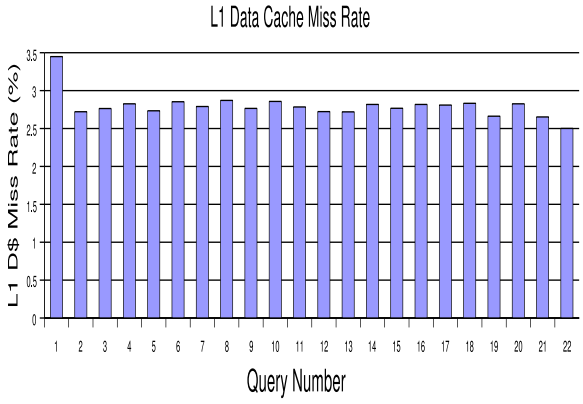
4.2 Cache Performance

Figure 3 shows the cache miss rates for L1 instruction cache (see Figure 3a) and L1 data cache (see Figure 3b). As can be seen from Figure 3a, the L1 instruction cache miss rates are quite low (<2.1% for all queries). However, queries Q3, Q5, Q6, Q15, Q17 and Q21 have instruction miss rates of nearly 2%. Q3, Q5, Q6, Q15, Q17 and Q21 exercise nested joins, and Q5 and Q6 exercise sorting in

[1]Lower CPI implies better performance

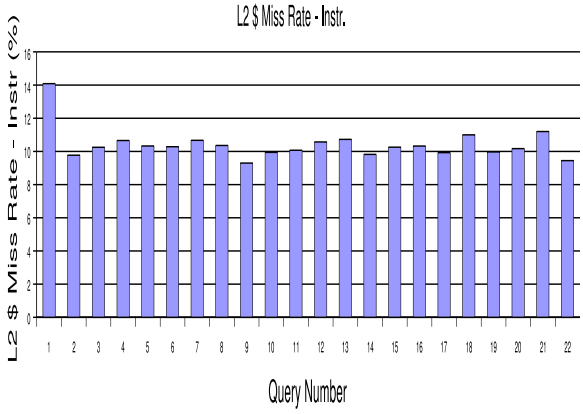


(a) L1 Instr. Cache Miss Rates

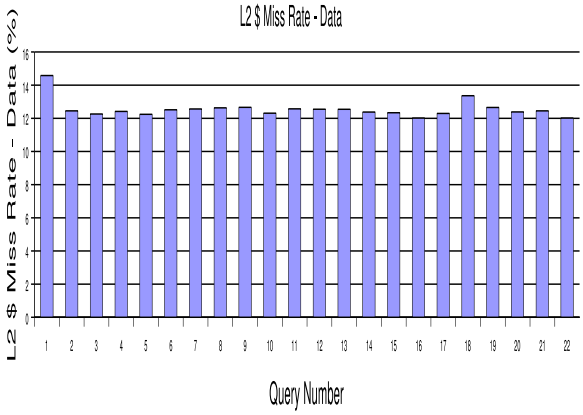


(b) L1 Data Cache Miss Rates

Figure 3: L1 instruction and data cache miss rates.



(a) L2 Cache Miss Rates - Instr.



(b) L2 Cache Miss Rates - Data

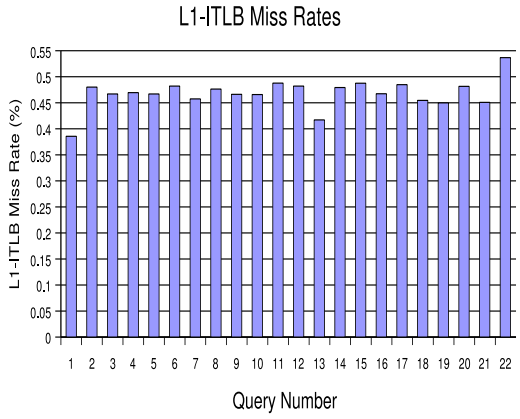
Figure 4: L2 cache miss rates for instructions and data.

database operators. From these results, we can see that MonetDB performs less than optimally with nested joins. This might be due to the fact that these queries see a lot of OS instructions that cause eviction of user-instructions from the L1 cache. As shown in Figure 3b, the L1 data cache miss rates are consistently around 2.5%-2.9%. This can be attributed to the various cache-conscious clustering and join algorithms implemented in MonetDB. The exception to this case is Q1, which has a miss rate of about 3.45%. This query performs aggregation operations on several columns of database table *LINEITEM*. As a result, the respective columns of the database table are brought into the cache. Since the size of each column is larger than the cache, Q1 incurs a lot of cache misses.

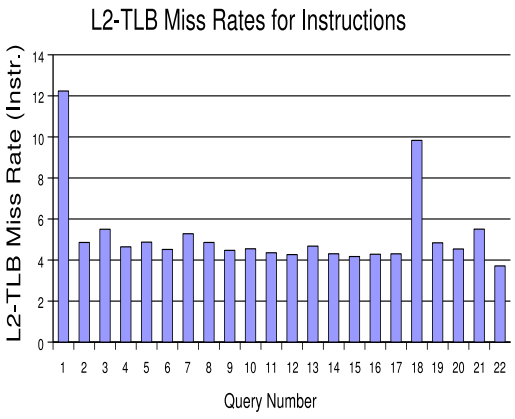
Figure 4 shows the L2 cache miss rates for instructions (see Figure 4a) and data (see Figure 4b). As can be seen from Figure 4a, the L2 cache miss rates for instructions vary from 9% to 12% for all queries, while Figure 4b shows that L2 cache misses for data are less than 15% for all queries. Previous research [7] suggested that databases should concentrate on L2 placement of data as the L2 cache miss rates were as high as 40-90%. Our result shows that MonetDB has successfully managed to achieve this goal [1]. However, interestingly (as seen from Figure 9 and Figure 10) increased processor speeds make even such small L2 cache misses formidable enough that nearly 50-60% of the total stall time can be attributed to data cache stalls. These results corroborate the key fact that simply ensuring optimal data placement in L2 cache is no longer enough and databases should strive to ensure that optimal data placement is done at L1 cache level [17]. Furthermore, this clearly indicates a shift of performance bottleneck from L2 cache misses to L2 cache hit latencies (or L1 data cache misses).

4.3 TLB Performance

Figure 5 shows the miss rates for L1 Instruction Translation Look-aside Buffers (ITLB) (see Figure 5a) and L2 TLB miss rates for instructions (see Figure 5b). The L1 ITLB miss rates are relatively low for all queries (from 0.39% to 0.54%). The L1 ITLB miss rate of Q22 is relatively higher because there are several levels of aggregation operations in this query. Furthermore, it seems that the MonetDB executable layout is not closely packed into pages for Q22, which results in larger L1 ITLB misses. The L2 TLB misses for instructions are moderately larger for most queries as compared to L1 ITLB misses. This might suggest that the working set of queries is not closely related at the L2 TLB level. Interestingly, Q1 and Q18 have a large number of L2 TLB misses for instructions. This is because these queries perform a large number of different

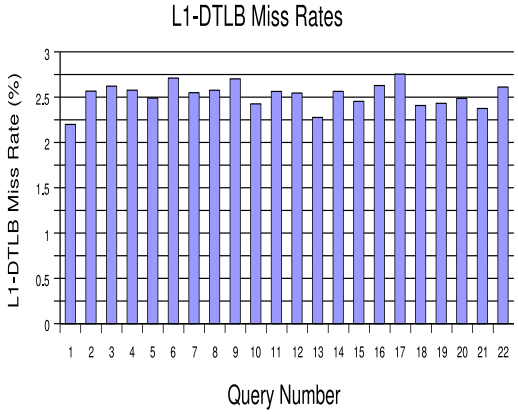


(a) L1 ITLB Miss Rates

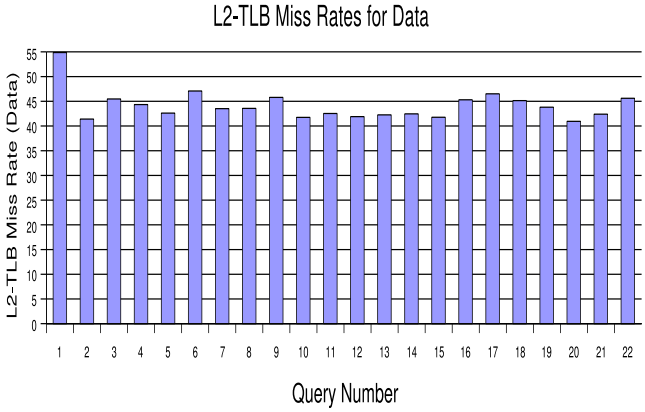


(b) L2 TLB Miss Rates - Instr.

Figure 5: L1 and L2 TLB miss rates for instructions.



(a) L1 DTLB Miss Rates



(b) L2 TLB Miss Rates - Data

Figure 6: L1 and L2 TLB miss rates for data.

aggregation operations on various columns, which result in considerable L2 TLB misses for instructions.

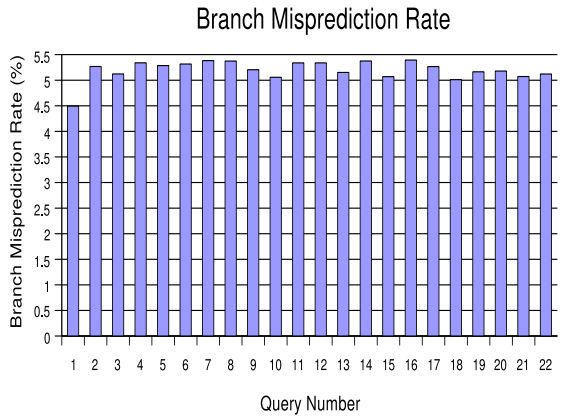
Figure 6 shows the miss rates for L1 Data Translation Look-aside Buffers (DTLB) (see Figure 6a) and L2 TLB miss rates for data (see Figure 6b). The L1 DTLB miss rates are relatively low for all queries (around 2-3%). As opposed to the L1 DTLB miss rates, the L2 TLB miss rates for data are quite large at about 40% to 55% with the maximum L2 TLB miss rate faced by Q1 and Q6. This can be attributed to two reasons. Firstly, Q1 performs different aggregation functions on the various columns of the *LINEITEM* data table. Due to the column-oriented nature of MonetDB, these columns might be stored in different pages, which resulted in a large number of L2 TLB misses. Furthermore, based on the L1 DTLB miss rates, once these pages have been fetched into the main memory, the database exploits spatial locality of the data to optimally perform the aggregation functions. Secondly, queries like Q6 perform random sorting on the data table. This results in a large number of L2 TLB misses for data since data shows relatively less spatial locality.

4.4 Branch and Near Returns

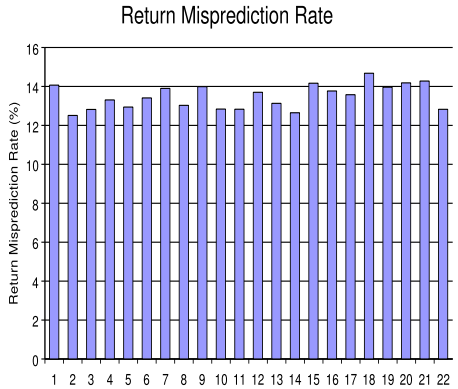
Figure 7 describes the misprediction rates for branch predictors and near return predictors. As can be seen from Figure 7a, the processor microarchitectures have improved since 1999 [7] and also 2004 [17] and only show about 4-6% branch misprediction rates as compared to 5-15% reported in [7, 17]. The AMD Athlon implements a 2048 two-way set-associative branch predictor to provide good branch prediction accuracy [9]. However, branch stalls still account for nearly 10-15% of processor stalls. This might be because of larger pipelines in AMD Athlon. Also, this can be a result of interference between Reorder Buffer (ROB) and branch prediction. As can be seen from Figure 7b, return mispredictions are quite large as compared to branch mispredictions. Most modern processors such as AMD Athlon employ a return stack which stores the last return address during any sub-routine call. AMD Athlon has a twelve-entry return stack. The return stack overflow rate is about 3-4% (see Figure 8). One reason that even this twelve-entry stack is not enough for database applications is because databases such as MonetDB use deep recursive algorithms. As a result, databases such as MonetDB make a less than optimal use of the return stack.

4.5 Detailed Analysis of Stalls

Figure 9 shows the overall breakdown of execution time while Figure 10 describes the stall profile of each of the TPC-H queries.



(a) Branch Misprediction Rate



(b) Near Return Misprediction Rate

Figure 7: Branch and near return misprediction rates.

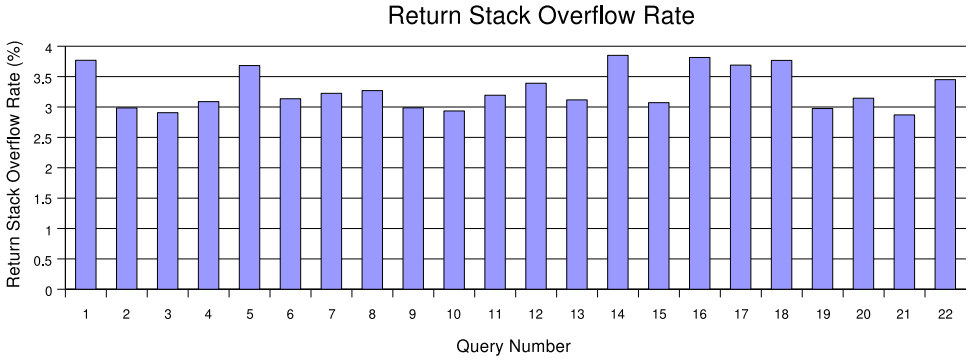


Figure 8: Return stack overflow rates.

As can be seen from these figures, LS-Full (meaning load/store buffer full) is the single most important contributor to the stall, contributing as much as 50-60% of processor stalls. This Load-Store buffer is used to hold stores waiting to retire as well as requests that missed the data cache and are waiting to refill. This condition stalls further data cache accesses, though some of these stalls might be overlapped with independent instruction execution. This result indicates that extremely tight data dependencies in databases coupled with L1 and L2 data cache misses and hit latencies are a major contributor to the performance impediment of databases. As a result, L1 data placement has become essential for database performance improvement. The second most important contributor to the stalls is RS-Full (meaning integer reservation stations full[2]). The AMD Athlon instruction reservation station can hold up to 18 integer macro-ops. This stall indicates that the integer execution reservation stations are full nearly 18-25% of the time. As most of our queries utilize integer execution units, the database system performance can be improved by increasing the depth of the reservation stations to look deeper into instruction streams. Following these stalls were ROB-Full stalls (about 9-13%) and branch abort stalls (about 10-15%). The ROB-Full event occurs when re-order buffer is full. This stall indicates that database systems show there is a mismatch between instructions in flight in the pipeline and the depth of the re-order buffer. It can also be attributed to the fact that ROB becomes full as memory intensive instructions are waiting to retire. This situation is exacerbated by the fact that the ROB might become full and then maybe flushed out due to branch mispredictions causing unnecessary instruction paths to be stored in the ROB.

[2]RS-Full stall cycles are only counted when integer reservation stations are full. The stall cycles due to floating point reservation stations being full are negligible and are represented by FPU-Stall in Figure 9 and Figure 10.

4.6 Detailed Analysis of Memory Stalls

Prior research on the performance of TPC-H queries on row-oriented databases has shown that memory stalls constitute a major part of total execution time. In this section, we investigate in more detail the major components of the memory stalls for MonetDB. Figure 11 shows the components that contribute to memory stalls, namely, L1 instruction cache stalls, L1 data cache stalls, L2 cache stalls due to instruction and data misses, L1 ITLB stalls, L1 DTLB stalls and L2 TLB stalls for instructions and data. Table 4 shows that these stall components are calculated by multiplying the misses with corresponding stall penalties. We have used the calibrator tool provided with MonetDB [1] to calculate these stall penalties. Note that in our experiments, we did not take into account the possible overlaps among stall times (in cycles) resulting from cache or TLB misses as well as overlaps with instruction execution. For example, the AMD Athlon [9] has a non-blocking cache [26] that

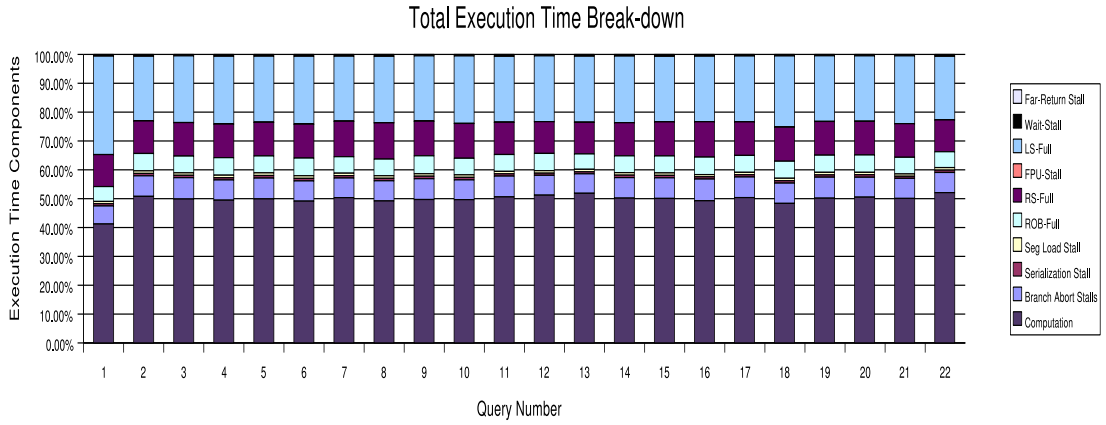


Figure 9: Execution time components.

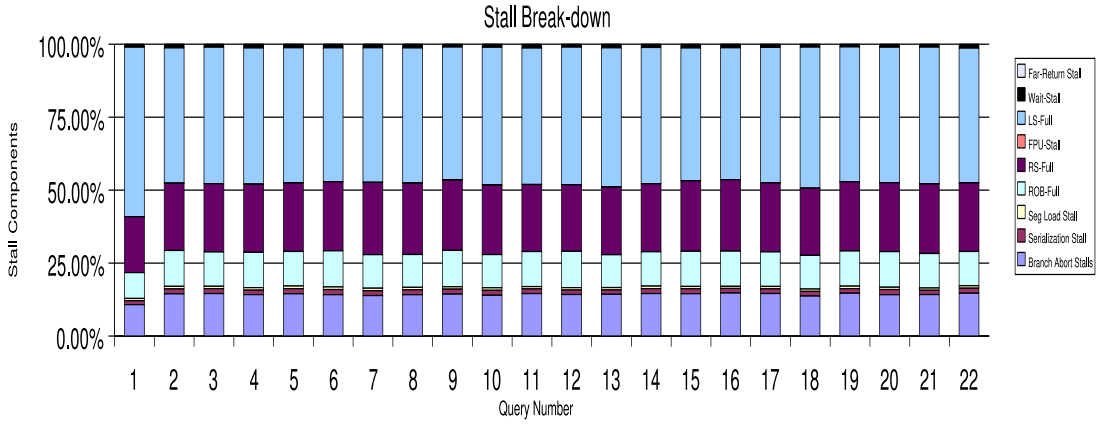


Figure 10: Components of database stalls.

can continue servicing cache requests even when there are multiple outstanding cache misses. It can happen that after a cache miss, the processor may execute some instructions out-of-order and then cause one more cache miss. At the time of the second cache miss, the servicing of the first cache miss might have been partially completed. Thus, there might be some overlap between time taken (in terms of the number of cycles) to service these two cache misses. In Table 4, the stalls due to these two events are calculated assuming there is no overlap in time taken (in terms of the number of cycles) to service two separate cache misses. The same assumption applies to the TLB misses. This may have resulted in the memory stall components shown Table 5 to be overestimated by a small amount.

Table 4: Calculation of memory stalls

Stall components	Description
L1 I-cache stalls	L1 I-cache misses * 11 cycles
L1 D-cache stalls	L1 D-cache misses * 11 cycles
L2 cache stalls	L2 cache misses * 89 cycles
L1 ITLB stalls	L1 ITLB misses * 5 cycles
L1 DTLB stalls	L1 DTLB misses * 5 cycles
L2 TLB stalls	L2 TLB misses * 42 cycles

Table 5 shows the average percentages of all memory stalls in our work. Our results for L2 cache stalls for data are similar to those found by Ailamaki et al. [7]. L2 cache stalls for data are still one of the major contributors to memory stalls. Ailamaki et al. [7] reported that L1 data cache miss stalls contribute to approximately 5-6% of total memory stalls. Our results indicate that now, L1 data cache miss stalls (or L2 cache hit stalls) have become more important than L2 cache miss stalls for data. Ailamaki et al. reported as much as 4-40% stalls due to L1 instruction cache misses. However, their results also showed that a *'System A'* having a small instruction footprint exhibited optimized instruction cache behavior that minimized L1 instruction cache stalls to at most 5% of the total memory stall time. Furthermore, they reported lower instruction TLB stalls for this system. Our results indicate that L1 instruction cache stalls are on average equal to 11.36% while L1 ITLB stalls are about 1.31%. This indicates that the real-time code expansion strategies employed by MonetDB [1] are indeed optimized for the L1 instruction cache as well as the L1 ITLB. Ailamaki et al. did not report the TLB stalls for data due to the limitation of their hardware performance counters. Our hardware platform allowed us to measure these stalls, and interestingly these stalls contribute to as much as 30% to the overall memory stall time. This can be due to the column-oriented nature of MonetDB. As columns might be stored in different pages, there can be a large number of L2 TLB misses for data when performing random scans that access multiple pages. Furthermore, based on the L1 DTLB miss rates, once these pages have been fetched into the main memory, MonetDB exploits spatial locality of the data to optimally perform the query operations.

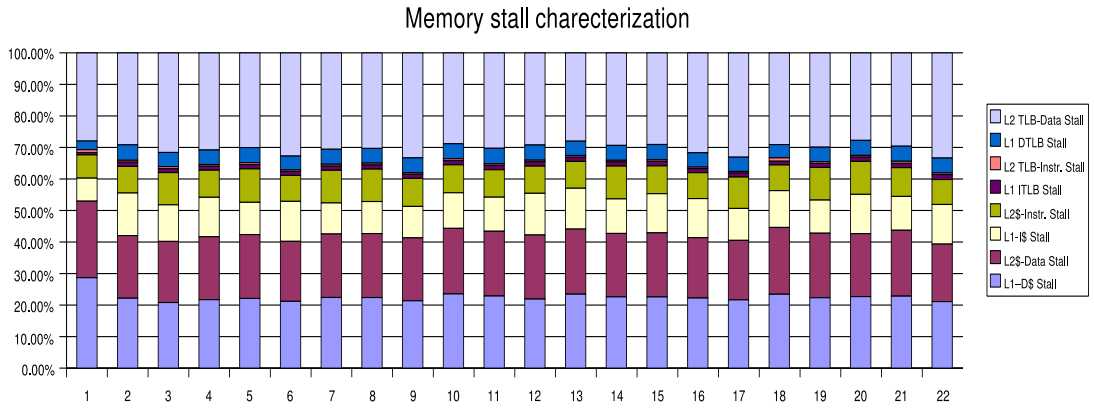


Figure 11: Components of memory stalls.

Table 5: Average percentages of all memory stall components

Stall Components	L1 D$	L2$-Data	L1 I$	L2$-Instr	L1 DTLB	L2 TLB-Data	L1 ITLB	L2 TLB-Instr	Total
Percentages (%)	22.55	20.21	11.36	9.14	4.58	30.26	1.31	0.59	100

5. CONCLUSION AND SUMMARY

Based on the results outlined in the previous section, the following comparisons can be made with published results of row-oriented databases on AMD-Athlon [21] and Intel Processors [7, 17].

1. Cycles Per Instruction (CPI): Our results indicate that the user-level CPI of the TPC-H queries when executed on MonetDB are in the same range (1.2-3 cycles per instruction) as the user-level CPI reported in [21] (for DB2) and [7, 17] (for unnamed row-oriented databases). However, our results also indicate that the average OS-level CPI for TPC-H queries on MonetDB was about 2-3 cycles per instruction as opposed to the CPI of about 10-20 cycles per instruction reported by Clark et al. [21]. This clearly shows an improvement factor of about 5-10X. One simple explanation to this decreased CPI is the difference in the operating systems used for our experimentation and that used by Clark et al. [21]. However, benchmarking results on Internet servers such as that done in [22] indicate that there is not much difference in the CPI of the operating systems under consideration. Hence, the difference in the OS-level CPI can be attributed to the privileged instructions executed by the OS kernel on behalf of the database application. A closer inspection of kernel-level functions indicated that the OS spent about 40-50% of the time in page-fault related functions such as *get_page_from_freelist, do_page_fault, find_get_page and copy_page_c* amongst others. Page faults occur when there is a miss in the L2 TLB for data or instructions. As the OS-level CPI of our experimentation is much lower than that reported in literature [21], it indicates that column-oriented databases are successful in utilizing spatial locality at page level. On the other hand, due to the column-oriented nature of MonetDB, columns may be stored in different pages, which results in a large number of L2 TLB misses when MonetDB accesses too many columns. However, based on the L1 DTLB miss rates that we reported, once these pages have been fetched into the main memory, the database exploits spatial locality of the data at page level to optimally perform the aggregation functions. To further maximize the advantages of the page level data locality, column-oriented databases can try to fit more number of columns on a single page. One of the research directions that can be followed is using optimization techniques such as column compression in order to fit more number of columns in a single page. Research has shown that such optimization techniques can contribute significantly to the performance of column-oriented databases [25].

2. Cache Performance: Our database setup performs useful computation about 40-50% of the time while the remaining time is spent in stalls (see Figure 9). This indicates that the performance bottlenecks identified in [7, 21] for row-oriented database systems still exist and further avenues should be explored to tackle these bottlenecks. L2 hit stalls have arisen to become the crucial bottleneck in modern processors. Hence, databases (both column-oriented and row-oriented) should exploit optimal placement of data at the L1 cache level. We make this conclusion in light of our results and the observation made in [17, 23] where the authors (as of 2001 and 2007) claim that L2 hit stalls have arisen to become the crucial bottleneck in the database systems.

3. Resource stalls: Resource stalls such as reservation station stalls and re-order buffer stalls contribute to as much as 20-40% of the stalls in the database. These results are also similar to the resource stalls reported in [7]. This means that modern processor manufacturers should take note of it and increase the depth of these units. As columns in a binary association table in a column-oriented database can be considered as arrays of data, another option is to employ vectorized query processing [2–4] using specialized vector processing hardware. This research direction has definitely shown promise as presented in [24].

4. Branch Prediction: Despite improved branch prediction mechanisms (that seem to have reduced the branch misprediction rates), branch abort stalls still contribute about 10-15% of database stalls. This percentage is very close to the percentage of branch stalls reported in [7, 21]. Hence, databases should try to overcome these limitations by further optimizing layout of their code. Branches are one of the artifacts of sequential Single Instruction Single Data

(SISD) machines. Using specialized vector processing hardware such as the Cell Broadband Engine (CBE) [24] may reduce the number of branch stalls as such approaches use the Single Instruction Multiple Data (SIMD) processors and predicated hardware instructions to eliminate branches.

In this paper, we characterize the performance of MonetDB for the TPC-H benchmark queries on a modern dual-core AMD Athlon processor. By comparing our results with existing published research in [7, 17, 21], we find that the nature and the percentages of the stalls faced by MonetDB is similar to that reported in literature for row-oriented databases [7, 17, 21]. Individual hardware components that we evaluated in our experiments namely, the L1 caches, L2 caches, TLBs and the processor do not show the significant amount of improvement that can contribute to the speedups reported in literature. This indicates that there is still a need to explore solutions to further accelerate column-oriented databases.

6. FUTURE WORK

In this work, we have characterized the databases on a dual-core AMD Athlon processor. The database size is set for a scaling factor of 1.0. In the future, we will extend our work on other modern processors and also evaluate the effects of varying the scaling factor on the performance characterization of TPC-H benchmark.

7. ACKNOWLEDGMENTS

This research was performed with the aid of the IUPUI RSFG internal grant. We would like to thank Peter Boncz and his team at CWI, Netherlands for their work with MonetDB. We would like to specially thank Xiang Xiao (IUPUI) for providing valuable inputs and comparing our work with his research on characterization of TPC-H benchmarks on MonetDB using hardware simulation.

8. REFERENCES

[1] P. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*, Ph.D. Thesis, Universiteit van Amsterdam, Amsterdam, Netherlands, May 2002.

[2] S. Heman, M. Zukowski, A. P. de Vries and P. A. Boncz. Efficient and flexible information retrieval using MonetDB/X100. In *Proc. of the Conference on Innovative Data Systems Research (CIDR)*, pages 96-101, Jan. 2007.

[3] S. Heman, M. Zukowski, A. P. de Vries and P. A. Boncz. MonetDB/X100 at the 2006 TREC TeraByte Track. In *Proc. of the Text REtrieval Conference*, Nov. 2006.

[4] M. Zukowski, P. A. Boncz, N. Nes and S. Heman. MonetDB/X100 - A DBMS in the CPU cache. *IEEE Data Engineering Bulletin*, 28:17-22, June 2005.

[5] M. Stonebraker, D. Abadi, A. Batkin et al. C-Store: A column oriented DBMS. In *Proc. of VLDB*, pages 553-564, 2005.

[6] Vertica Homepage, http://www.vertica.com/, visited Jan. 2008.

[7] A. Ailamaki, D. DeWitt, M. Hill and D. Wood. DBMSs on a modern processor: where does time go? In *Proc. of the VLDB*, pages 266-277, Sep. 1999.

[8] TPC-H benchmark specification, http://www.tpc.org/tpch/, visited Dec. 2007.

[9] AMD-Athlon Technical Brief, http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/22054.pdf, visited Dec. 2007.

[10] C. Date. *Introduction to Database Systems*. 8th edition, Addison-Wesley, New York, 2003.

[11] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufman, San Mateo, CA, USA 1991

[12] V. Harinarayan, A. Rajaraman and J. Ullman. Implementing datacubes efficiently. In *Proc. of ACM SIGMOD*, pages 205-216, June 1996.

[13] L. A. Barroso, K. Gharachorloo and E. Bugnion. Memory system characterization of commercial workloads. In *Proc. of the International Symposium on Computer Architecture*, pages 3-14, 1998.

[14] K. Keeton, D. Patterson, Y. He, R. Raphael and W. Baker. Performance characterization of a quad Pentium Pro SMP using OLTP workloads. In *Proc. of the International Symposium on Computer Architecture*, pages 15-26, 1998.

[15] S. Oaks. Be Aware: to hyper or not to hyper. MSDN Blog, http://blogs.msdn.com/slavao/archive/2005/11/12/492119.aspx, visited Dec. 2007.

[16] P. Trancoso, J. Larriba-Pey, Z. Zhang and J. Torrellas. The memory performance of DSS commercial workloads in shared-memory multiprocessors. In *Proc. of the International Symposium on High-Performance Computer Architecture*, pages 250-260, 1997.

[17] M. Becker, N. Mancheril and S. Okamoto. DBMSs on a modern processor: where does time go? revisited. http://www.cs.cmu.edu/People/ngm/15-721/project/final/paper.pdf, visited Dec. 2007.

[18] A. Ailamaki, N. Govindaraju, S. Harizopoulos et al. Query co-processing on commodity processors. In *Proc. of VLDB*, 2:1267, 2006.

[19] J. Cieslewicz, J. Berry, B. Hendrickson and K. Ross. Realizing parallelism in database operations: Insights from a massively multithreaded architecture. In *Proc. of the International Workshop on Data Management on New Hardware*, June 2006.

[20] M. Stonebraker et al. One size fits all ? – Part 2: Benchmarking results. In *Proc. of Conference on Innovative Data Systems Research*, Jan. 2007.

[21] M. Clark, A. Durg, K. Lienenbrugger. Characterization of TPC-H queries on AMD Athlon^{TM} microprocessors. In *IEEE International Workshop on Workload Characterization*, pages 26-35, 2001.

[22] Y. Luo, J. Rubio, L. John, P. Sheshadri, A. Mericas. Benchmarking Internet servers on superscalar machines. *Computer*, 36(2):34-40, Feb. 2003.

[23] D. Liu, S. Wang, B. Qin and W. Wong. Charecterizing DSS workloads from the processor perspective. In *International Workshop on Database Management and Application over Network*, pages 235-240, 2007.

[24] S. Heman, N. Nes, M. Zukowski and P. Boncz. Vectorized data processing on the Cell Broadband Engine. In *Proc. of the DAMON workshop*, 2007.
of China,

[25] D. Abadi, S. Madden, N. Hachem. Column-stores vs. row-stores: How different are they really? In *Proc. of ACM SIGMOD*, 2008.

[26] J. Hennessy and D. Patterson. *Computer architecture - a quantitative approach*. 4th edition, Morgan Kaufmann Inc., San Francisco, CA, 2006.