Characterizing the Performance of Data Management Systems on the Pentium 4 Hyper-Threaded Architecture

Wessam Hassanein, Layali Rashid, Maryam Mehri, Moustafa Hammad*

Department of Electrical & Computer Engineering

*Department of Computer Science
University of Calgary

{hassanein@enel., lrashid@, mmehride@, hammad@cpsc.}ucalgary.ca

ABSTRACT

As the information acquisition and processing applications take greater roles in our everyday life, database management systems are growing in importance. Database management systems have traditionally exhibited poor cache performance and large memory footprints, therefore performing only at a fraction of their ideal execution and exhibiting low processor utilization. Previous research has studied the memory system of database management systems (DBMSs) on research-based SMT processors. Recently, several differences have been noted between the real hyper-threaded architecture implemented by the Intel Pentium 4 and the earlier SMT research architectures. Therefore, it is becoming important to study and analyze the performance of DBMS on a real SMT processor. This paper characterizes the performance of a prototype open-source DBMS running benchmark queries on an Intel Pentium 4 Hyper-Threading processor. We use the performance hardware counters provided by the Pentium 4 to evaluate the micro-architecture and study the memory system behavior of each query running on the data management system.

1. INTRODUCTION

Information Management Systems are gaining importance as various types of data acquisition and information processing applications play larger roles in our everyday life. Database management system (DBMS) is a typical information management system that continuously evolves to meet the new application demands and hardware capabilities. Therefore, re-evaluating the DBMS's performance is significant to both the computer architecture and database communities. Previous studies have shown that DBMS exhibits high cache miss rates and low CPU utilization [13]. Simultaneous Multithreading (SMT) [24] is a form of hardware multithreading where a single physical multithreaded processor can execute multiple threads concurrently. The resources of a multithreaded processor are either shared between the threads or duplicated for each thread compared to a non multithreaded processor. Recently, SMT microprocessors have appeared in commercial machines (e.g., the Intel® Pentium® 4 Hyper-threading processor is a dual thread general purpose SMT processor). This has allowed SMT research to move from simulation-based studies to real hardware studies. Furthermore, the availability of real SMT processors allows for validating previous simulation results against new experimental results on real hardware. At the data management side, recent studies in hardware-aware data management focus on specific data operations such as sorting [10], main memory data structures [26], and data mining algorithms [11]. These studies provide insightful comments to improve various data management components by utilizing new hardware features. However, to the best of our knowledge, no characterization study exists of a complete DBMS system using SMT processors. Such study is inevitable to realistically evaluate a sophisticated and an integral system as DBMS.

This paper studies the performance of an open source DBMS, namely PostgreSQL [3], on the Intel Pentium 4 Hyper-threading processor. We analyze and compare the performance of several queries from the Wisconsin Benchmark [8] on PostgreSQL. Wisconsin Benchmark is gaining popularity among parallel database management systems and widely used in open-source DBMS such as PREDATOR and MySQL [8][1][2]. The benchmark queries are tuned to utilize index structures such as clustered and unclustered indexes. We compare the effects of hyper-threading on the DBMS performance by switching the hyper-threading feature of the processor on and off through the BIOS. The characterization study showed that for benchmark queries that benefit the least from the index structures either because of accessing large portions of an unclustered index or avoiding index structures altogether, hyper-threading improves the performance. For the rest of the workload queries that took advantage of index processing, hyper-threading negatively affected the performance.

The rest of the paper is organized as follows: Section 2 describes the experimental methodology. In Section 3, we present the experimental results. Section 4 discusses related work. Finally, conclusions are provided in Section 5.

2. EXPERIMENTAL METHODOLOGY

2.1 Experimental Platform

All experiments were run on a single 3.4GHz Pentium 4 Hyperthreading processor with a 2MB on chip L2 cache,

and a 1GB 533MHz DDR2 SDRAM memory. The operating system used is the Scientific Linux version 4.1 (Based on Redhat Linux Enterprise version 4.0) and running the Linux 2.6.14 kernel. The Pentium 4 uses a 12K micro-operation (uops) L1 instruction trace cache and a 64KB L1 data cache. The machine has an 800MHz front side bus.

2.2 Benchmarks

All experiments were run on the PostgreSQL open-source database management system (DBMS) version xx and using Wisconsin Benchmark [8]. PostgreSQL is a widely used DBMS in both research and industry with over 15 years of development effort. Specifically, we use PostgreSQL for the following two main reasons: First, PostgreSQL includes optimized implementations of various data management functionalities such as query processor, index manager, buffer manager, and storage manager. Furthermore, PostgreSQL SQL (Structured Query Language) implementation strongly conforms to the ANSI-SQL 92/99 standards, which is the latest SQL standard that is currently implemented in industrial-strength DBMSs. Second, being an open-source system facilitates problem tracking, identification, and resolution at both the architecture and data management sides.

Various Benchmarks are used to measure the performance of DBMSs (refer to [8] for further reference). We choose Wisconsin Benchmark as our workload queries because of its wide-applicability in evaluating the performance of ad hoc queries on parallel databases [8]. Second, in addition to its ease of implementation and publicity, the Wisconsin Benchmark is adopted for many open-source DBMS such as PREDATOR [1] and MySQL [2]. One argument against the Wisconsin Benchmark lies primarily in being not so popular among industrial-strength DBMSs because of targeting single-user queries. Although this argument is not really justified [8], we believe that using industrial-strength DBMS at this stage of the research will lack flexibility to trace performance bottlenecks at hardware and software levels. Furthermore, studying the effect of hyper-threading on a single user environment is an integral component of studying multi-user systems, which we plan to study in future research.

Wisconsin Benchmark. Wisconsin benchmark consists of data generation and workload queries. The data generation is based on the algorithm developed by Susan Englert and Jim Gray [9]. The workload queries use primarily three database tables (10MTUPLES, 1MTUPLES, and 100KTUPLES) and a set of temporary tables. The number of records in 10MTUPLES, 1MTUPLES, and 100KTUPLES is ten millions, one million, and a hundred thousand records, respectively. Table sizes are in the range from two gigabytes to two megabytes. None of the benchmark queries can access tables that fit entirely in the available memory-space. The tables have the same description (i.e., schema) as descried in [8] as given in Table 1. A table consists of a set of integer and string fields. The record size is 208 bytes. Table 2 gives the workload queries while omitting the INSERT INTO part for space limitations. All queries are using a mix of clustered and unclustered indexes to speed up the access of database tables. Clustered indexes have the records physically stored in the same order as the keys of the clustered index. Because of the physical records proximity, using a clustered index to retrieve records with equal or consecutive values reduces significantly the query execution time. However, at most one clustered index can be built on a single table. On the other hand, an unclustered index will speed up search. However, the index significantly increases query execution time as we retrieve large portions of the stored data from the indexed table. The percentage of retrieved records is related to the selectivity of the query.

Table 1: The description (schema) of tables used by our queries

Attribute Name	Range of Values	Order	Comment
unique1	0-(MAXTUPLES-1)	random	unique, random order
unique2	0-(MAXTUPLES-1)	sequential	unique, sequential
two	0-1	random	(unique1 mod 2)
four	0-3	random	(unique1 mod 4)
ten	0-9	random	(unique1 mod 10)
twenty	0-19	random	(unique1 mod 20)
onePercent	0-99	random	(unique1 mod 100)
tenPercent	0-9	random	(unique1 mod 10)
twentyPercent	0-4	random	(unique1 mod 5)
fiftyPercent	0-1	random	(unique1 mod 2)
unique3	0-(MAXTUPLES-1)	random	unique1
evenOnePercent	0,2,4,...,198	random	(onePercent * 2)
oddOnePercent	1,3,5,...,199	random	(onePercent * 2)+1
stringu1	-	random	candidate key
stringu2	-	random	candidate key
string4	-	cyclic	

2.3 Tools for Performance Measurements

The Pentium 4 is equipped with 18 performance counters that allow us to monitor 48 events at the architecture level. These events include cache misses, TLB misses, branch mispredictions, etc. We use the Intel vtune performance analyzer version 3.0 for Linux to take our measurements unless otherwise noted. We use the Event Based Sampling feature of the vtune performance analyzer to collect our data. We use the system BIOS to turn hyperthreading on and off.

3. RESULTS

This section examines the effect of hyperthreading on the performance of database benchmark queries (Table 2)

running on the PostgreSQL database. The Linux operating system considers a hyperthreaded machine with hyperthreading turned on as a two processor machine. Therefore, a single hyperthreaded (dual threads) SMT physical processor is considered by the Linux operating system as two logical processors. "Processor 0" in our results refers to the first thread (logical processor) on the hyperthreaded Pentium 4 while "Processor 1" refers to the second thread. In a hyperthreaded-off machine the physical processor is seen as only a single logical processor "Processor 0".

Table 2: Wisconsin Benchmark Queries used in our analysis

Query Name	Query
Q1(clus. index 1% selection)	SELECT * FROM 10MTUPLES WHERE unique2 BETWEEN 0 AND 99999;
Q2(clus. index 10% selection)	SELECT * FROM 10MTUPLES WHERE unique2 BETWEEN 792 AND 1000791;
Q3(unclus. index 1% selection)	SELECT * FROM 10MTUPLES WHERE unique1 BETWEEN 0 AND 99999;
Q4(unclus. index 1% selection)	SELECT * FROM 10MTUPLES WHERE unique1 BETWEEN 792 AND 100791;
Q5(clus. index)	SELECT * FROM 10MTUPLES, 1MTUPLES WHERE(10MTUPLES.unique2 = 1MTUPLES.unique2) AND (1MTUPLES.unique2 < 100000);
Q6(clus. index)	SELECT * FROM 10MTUPLES, 1MTUPLES, 1OOKTUPLES WHERE (1OOKTUPLES.unique2 = 1MTUPLES.unique2) AND (1MTUPLES.unique2 =10MTUPLES.unique2) AND (10MTUPLES.unique2 < 1000000)
Q7(unclus. index)	SELECT * FROM 10MTUPLES, 1MTUPLES WHERE (10MTUPLES.unique1 = 1MTUPLES.unique1) AND (10MTUPLES.unique2 < 1000000);
Q8(unclus. index)	SELECT * FROM 10MTUPLES, 1MTUPLES, 1OOKTUPLES WHERE (1OOKTUPLES.unique1 = 1MTUPLES.unique1) AND(1MTUPLES.unique1 = 10MTUPLES.unique1) AND (10MTUPLES.unique1 < 1000000)
Q9(clus. index)	SELECT MIN(10MTUPLES.unique2) FROM 10MTUPLES;
Q10(clus. index)	SELECT MIN (10MTUPLES.unique2) FROM 10MTUPLES GROUP BY 10MTUPLES.onePercent;
Q11(clus. index)	UPDATE 10MTUPLES SET unique2=10000001 WHERE unique2=1491;
Q12(unclus. index)	UPDATE 10MTUPLES SET unique1=10000001 WHERE unique1=1491;

The PostgreSQL database generates a new postgres process for each query run. This postgres process contains several modules including the postgres module as well as other operating system modules such as vmlinux, etc. To measure the effects of the benchmark query apart from other factors in the operating system, the results presented in this paper are of the postgres module within the postgres process.

- **Speedup.** Figure 1 shows the actual run time of each of our benchmark queries running on the PostgreSQL database. The run time is measured using the PostgreSQL database timing option for each query. We compare the run time when hyperthreading is turned on verses turned off. Our results show performance variations between benchmarks ranging from a speedup of 12% to a slowdown of 13% with the majority of benchmarks showing slowdowns when hyperthreading is turned on. Our results also show that Q3, Q4 and Q5 which are our longest running queries exhibit speedups due to hyperthreading.

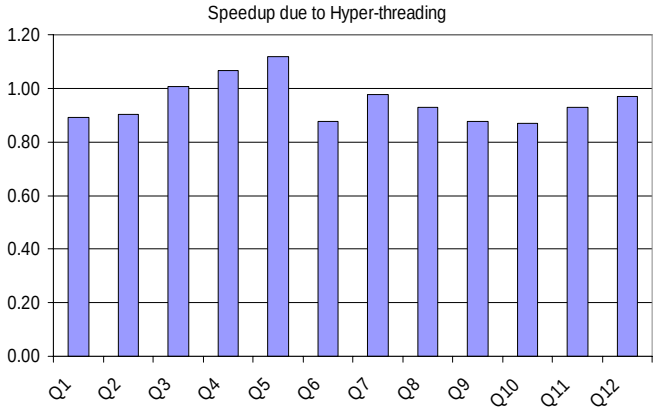

Figure 1: Speedup due to hyprethreading

- **Instructions Retired**

The Intel Pentium 4 decodes each instruction into several micro-operations (uops). The ratio of uops to instructions ranges from 1.28 to 1.96 uop/instruction in our query bencharmks. Figure 2 compares the instructions retired for hyperthreading turned on versus off. Our results show equivalent number of instructions with hyperthreading on compared to off. Figure 2 also shows that the majority of instructions are executed on the 1st thread. With Q2 and Q5 being the only benchmarks with a significant number of instructions executing on the 2nd thread.

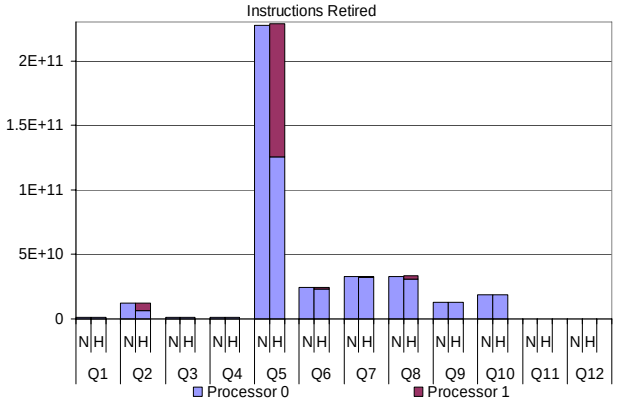

Figure 2: Instructions retired with hyperthreading on vs. off. (N= HT off, H = HT on.). Q11 and Q12 have a small number of instructions retired.

Cache Behavior. In SMT architectures the cache hierarchy is shared between all the thread contexts. This is also the case in the Intel Pentium 4 hyperthreaded architecture where both threads share the L1 instruction trace cache, the L1 data cache, and the unified L2 cache. This sharing can be either beneficial, if for example one thread prefetches data for the other, or detrimental if one thread conflicts with the other causing a large number of cache misses. In [16],

SMT introduces severe cache conflicts for On-line transcation processing (OLTP) database queries using conventional virtual memory management. Both beneficial and detrimental patterns have been reported on the Intel Pentium 4 hyperthreaded processor L1 and L2 caches for Java Applications [12]. Benefical patterns for L2 cache and detrimental patterns for L1 cache in network servers [22] have been reported on the Intel Xeon hyperthreaded processor. Figure 3 shows the L1 data cache miss rate for load instructions. We compare the miss rates with hyperthreading turned on versus off showing the percentage of misses experienced by each processor thread. It can be seen that Q11 (which is a short running query) benefits from sharing the L1 data cache, while very small benefits are experienced by Q6, Q7, Q9 and Q10. All other benchmark queries experience a detrimental effect some up to a 4% increase in cache misses. Our results also show that the majority of benchmarks that experience negative effects due to hyperthreading are experiencing some of these cache misses on the 2nd thread (processor 1).

Figure 3 shows the L2 cache miss rates. Our results show that Q3, Q4 and Q5 experience a significantly smaller number of L2 cache misses. These three benchmarks experience the largest drops in L2 cache misses compared to their L2 misses with hyprethreading off. This partially explains the resulting speedup in these benchmarks. The majority of our queries benefit at the L2 cache level from hyperthreading. This can be explained by our large L2 cache (2MB) that could reduce potential conflicts between threads. Figure 4 shows the L1 instruction trace cache miss rates. The Intel Pentium 4 processor has a 12Kuops instruction trace cache. Both Q3 and Q4 show reduced instruction trace cache miss rates due to hyprethreading. While Q5 experiences a slight increase in instruction cache misses with a large portion of this increase experience by the 2nd thread. Both Q11 and Q12 also benefit from hyperthreading at the instruction cache level. However, all other benchmarks show larger instruction cache miss rates due to hyperthreading.

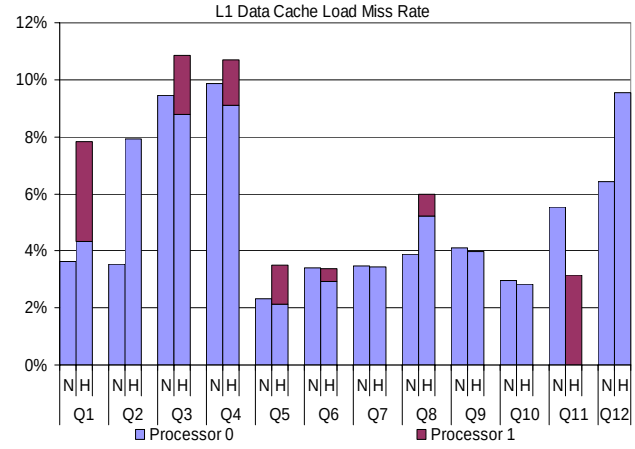

Figure 2: L1 data cache miss rate (N= HT off, H = HT)

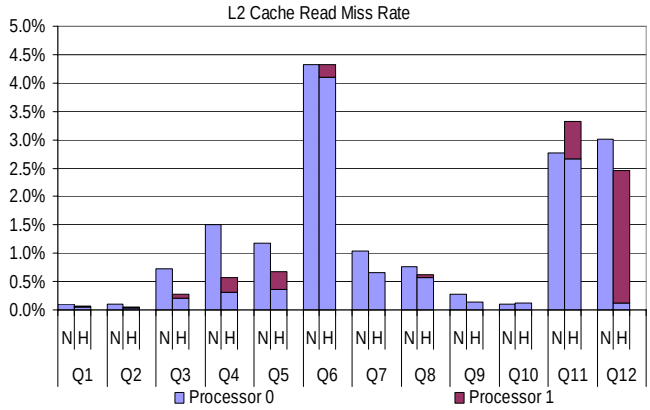

Figure 3: L2 cache miss rate (N= HT off, H = HT)

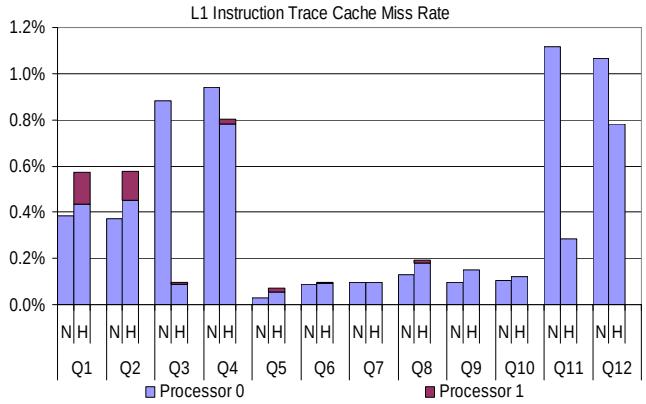

Figure 4: L1 instruction trace cache miss rate (N= HT off, H = HT)

- **TLB Misses**

The DTLB is used to translate the data logical addresses into physical addresses. The DTLB is shared in the Intel Pentium 4. Our results in Figure 5 show both beneficial and detrimental patterns in the DTLB misses with some benchmark queries benefiting from hyperthreading while others showing larger numbers of misses. Q4 benefits from hyperthreading DTLB sharing while Q3 and Q5 show larger DTLB misses with large percentages from the 2nd thread indicating that both threads are conflicting at the DTLB level.

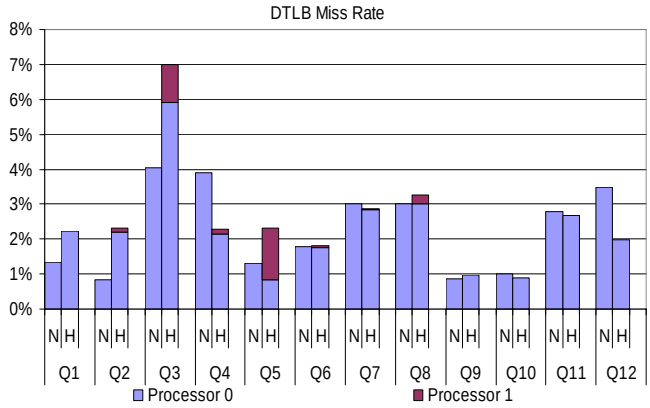

Figure 5: DTLB miss rates (N= HT off, H = HT)

- **Mispredicted Branches**

The branch misprediction rates in our query benchmarks shown in Figure 6 are up to 7.5%. A mispredicted branch has a 20 cycle penalty. Q3, Q4, Q5 and Q7 all benefit from the sharing at the branch history table level and result in significantly lower misprediction rates compared to hyperthreading off. All other benchmarks either experience slight degredation or improvement in branch performance due to hyperthreading.

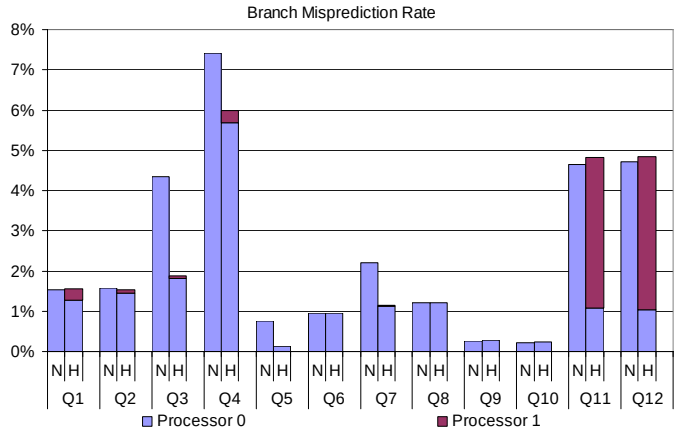

Figure 6: Branch misprediction rate (N= HT off, H = HT)

- **Performance Impact**

To further understand the importance of the different processor parts on the performance of our benchmarks, we plot the non-overlapped CPI stall component in Figure 7. This stall component represents the non-overlapped stall cycles per instruction and does not take into account cycle overlapping during execution which can affect the measured performance. Using LMbench tool [19] we measured our memory hierarchy latencies to be 1.18ns, 8.26ns, 137.9ns for L1 cache, L2 cache and memory latencies respectively. Our results show that the major stall cycle components are those of the DTLB and L2 cache misses.

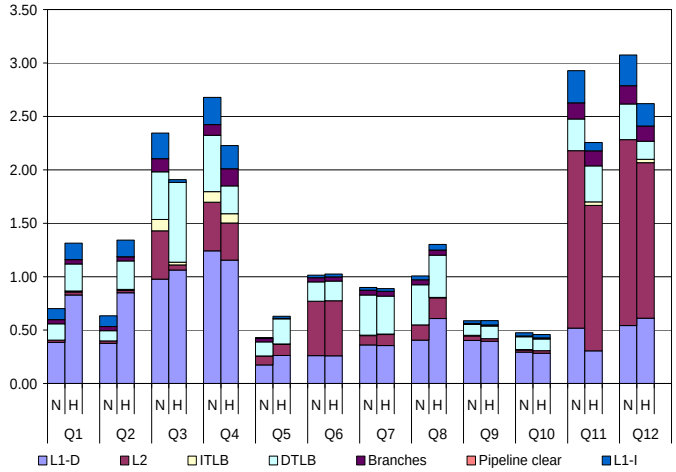

Figure 7: Non-overlapped CPI stall component (N= HT off, H = HT)

4. RELATED WORK

This section presents an overview of several related work on workload characterization of various applications. The behavior of Oracle DBMS using OLTP and DSS has been studied by Lo et al. [16] on a simulated SMT processor. They concluded that severe cache conflicts occurs using conventional virtual memory management and proposed a new page placement policy. A new technique for evaluating database workloads "microbenchmark" is proposed in [15].

Tuch *et al.* [23] measured the performance of SPEC2000 on Intel Pentium 4 processors. Their results showed both multi-programmed and parallel speedups. A related work has also been done in [6].

Chen *et al.* [7] evaluated the performance of multi-media applications on SMT processors and compared SMT versus SMP systems. Their analysis includes memory-bounded kernels and computational bounded functions. In [22] the impact of SMT on the Intel Xeon processor of network servers shows several resource bottlenecks.

Raasch and Reinhardt studied the impact of resource partitioning on SMT processors by focusing on the execution portion of the pipeline [21] . The performance of Pentium pro systems has been characterized in [13] and [1].

Wei Haung *et al.* [12] characterized the performance of Java applications on Intel Pentium 4 hyper-threading processors. Blackburn studied the performance of garbage collection on multithreading by using some of the Pentium 4 performance counters [5]. To the best of our knowledge no previous work has characterized the PostgreSQL DBMSs on a real hyperthreaded Pentium 4 processor.

5. CONCLUSION

In this paper we characterized the performance of an open-source database management system on an Intel Pentium 4 Hyper-Threading processor. The presented performance characterization is based on real measurements of a widely-adopted database benchmark of parallel databases. The database benchmark queries are tuned to utilize both clustered and unclustered indexes. The detailed performance study focused primarily on the effect of hyper-threading on the total execution time of the benchmark queries. The characterization study showed that for complex benchmark queries that benefit the least from the index structures either because of accessing large portions of an unclustered index or avoiding index structures altogether, hyper-threading improves the performance. For the rest of the workload queries that took advantage of index processing, hyper-threading negatively affected the performance. To the best of our knowledge no other characterization study has analyzed DBMS on SMT processors and only a limited amount of work has studied various data management operations generally on SMT architectures.

6. REFERENCES

[1] http://www.distlab.dk/predator/

[2] http://www.mysql.com/

[3] http://www.potgresql.org

[4] D. Bhandarkar and J. Ding. “Performance characterization of the Pentium Pro processor”. In Proc. of the 3rd IEEE Symp. on High-Performance Computer Architecture (HPCA) ’97, Feb. 1997.

[5] S. Blackburn, P. Cheng, and K. McKinley. “Myths and realities: The performance impact of garbage collection”. In Proc. of the SIGMETRICS ’04, June 2004.

[6] James R. Bulpin and Ian A. Pratt. “Multiprogramming performance of the Pentium 4 with Hyper-Threading”. In Proc. of the 3rd Annual Workshop on Duplicating, Deconstructing, and Debunking (WDDD), Munich, Germany, June 2004.

[7] Y. K. Chen, E. Debes, R. Lienhart, M. Holliman, and M. Yeung. “Evaluating and improving performance of multimedia applications on simultaneous multi-threading”. In 9th Intl. Conf. on Parallel and Distributed Systems, Dec. 2002.

[8] D.J.DeWitt. “The Wisconsin Benchmark: Past, Present, and Future”. In the Benchmark Handbook for Database and Transaction Systems (2nd Edition). Morgan Kaufmann 1993, ISBN 1-55860-292-5.

[9] Englert, S. and J. Gray, "Generating Dense-Unique Random Numbers for Synthetic Database Loading," Tandem Computers, January, 1989.

[10] P. Garcia and H. Korth. “Multithreaded Architectures and The Sort Benchmark". In Proceedings of the first international workshop on data management on new hardware, June 2005.

[11] A. Ghoting, G. Buehrer, S. Parthasarathy, D. Kim, Y.Chen, A. Nguyen, and P. Dubey, "Cache-Conscious Frequent Pattern Mining on a Modern Processor", In Proceedings of the International Conference on Very Large Data Bases (VLDB), 2005.

[12] Wei Huang; Jiang Lin; Zhao Zhang; Chang, J.M. “Performance Characterization of Java Applications on SMT Processors”, International Symposium on Performance Analysis of Systems and Software (ISPASS), March, 2005

[13] K. Keeton, D. Patterson, Y. He, R. Raphael, and W. Baker. “Performance charaterization of a Quad Pentium Pro SMP using OLTP workloads”. In Proc. of the 25th International Symposium on Computer Architecture (ISCA).

[14] K. Keeton, A. Veitch, D. Obal, and J. Wilkes. "I/O Characterization of Commercial Workloads," presented at Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW)-00, before High-Performance Computer Architecture (HPCA)-6, January 2000

[15] K. Keeton and D. Patterson. "Towards a Simplified Database Workload for Computer Architecture Evaluations," presented at the Workshop on Workload Characterization, Austin, Texas, October 1999. In Workload Characterization for Computer System Design, edited by L. K. John and A. M. Maynard, Kluwer Academic Publishers, 2000, ISBN 0-7923-7777-x.

[16] Jack Lo, Luiz Barroso, Susan Eggers, Kourosh Gharachorloo, Henry Levy, and Sujay Parekh. “An analysis of database workload performance on simultaneous multithreaded processors”. In Proc. of the 25th International Symposium on Computer Architecture (ISCA), Barcelona, Spain, June 1998.

[17] Yue Luo and Lizy K. John. “Workload characterization of multithreaded Java servers”. In Proceedings of 2001 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Tucson, Arizona, November 2001.

[18] D.Marr, F. Binns, D. Hill, G. Hinton, D. Koufaty, J. A.Miller, and M. Upton. “Hyper-threading technology architecture and microarchitecture”. Intel Technology Journal,

[19] L. McVoy and C. Staelin. lmbench. “Portable tools for performance analysis”. In USENIX 1996 Annual Technical Conference.

[20] PostgresSQL Documentation available at http://www.postgresql.org/docs/8.1/interactive/history.html

[21] Steven E. Raasch , Steven K. Reinhardt, The Impact of Resource Partitioning on SMT Processors, Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques, p.15, September 27-October 01, 2003

[22] Y. Ruan, V. S. Pai, E. Nahum, John M. Tracey. ”Evaluating the Impact of Simultaneous Multithreading on Network Servers Using Real Hardware”. In theProc. of the SIGMETRICS '05, June, 2005.

[23] N. Tuck and D. Tullsen. “Initial observations of a simultaneous multithreading processor”. In 12th Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT’)03, Sept. 2003.

[24] D. Tullsen, S. Eggers, and H. Levy. “Simultaneous multithreading: Maximizing on-chip parallelism”. In Proc. of the 22th International Symposium on Computer Architecture (ISCA), 1995.

[25] Intel Corp. VTune performance analyzer. Available at http://www.intel.com/software/products/vtune/.

[26] J. Zhou, J. Cieslewicz, K. A. Ross, and M. Shah. “Improving database performance on simultaneous multithreading processors”. In Proceedings of the International Conference on Very Large Data Bases (VLDB), 2005.