Large Scale Itanium® 2 Processor OLTP Workload Characterization and Optimization

Gerrit Saylor
Intel Corporation
Suite 300, 2700 156th Ave NE
Bellevue, WA 98007
(408) 765-8080
gerrit.saylor@intel.com

Badriddine Khessib
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052
(425) 882-8080
bkhessib@microsoft.com

ABSTRACT
Large scale OLTP workloads on modern database servers are well understood across the industry. Their runtime performance characterizations serve to drive both server side software features and processor specific design decisions but are not understood outside of the primary industry stakeholders. We provide a rare glimpse into the performance characterizations of processor and platform targeted software optimizations running on a large-scale 32 processor, Intel®[1] Itanium® 2[1] based, ccNUMA platform.

Categories and Subject Descriptors
B.8.2 [**Performance and Reliability**]: Performance analysis

C.4 [**Performance Of Systems**]: Measurement Techniques, Performance attributes

H.2.4 [**Database Management**]: Systems – *Transaction processing*

General Terms
Measurement, Performance, Experimentation

Keywords
OLTP, Itanium, ccNUMA, software optimization, profile guided optimization, cache coherency, data partitioning, and performance characterization

1. INTRODUCTION
On-line transaction processing (OLTP) workloads are widely used by hardware and software vendors as a basis for design decisions; they provide a convenient metric for self-evaluation and a method to target specific server market segments. Despite concerns about the validity of synthetic benchmarking, customers commonly rely on OLTP performance to aid purchasing decisions. Not surprisingly, vendors expend considerable effort in characterizing OLTP behavior and minimizing performance constraints in order to maximize system throughput.

These constraints occur in many contexts, but we present a set of processor level performance indicators and the effects of processor specific software optimizations. It is a general overview of the performance constraints of a large-scale, enterprise class, Itanium2 based, 32-processor server. We briefly describe OLTP workloads, Itanium 2 processor performance analysis support, Itanium 2 microarchitectural characterizations under an OLTP load, and the system under test. We compare the OLTP load on an optimized software stack against a set of measurements made by individually disabling the following performance features: profile guided optimization (PGO), data partitioning, Superlatching, and large virtual page support. More broadly, these comparisons are general indicators of the necessity of considering hardware design in software performance and scalability work, especially with regard to enterprise class machine architectures, large executables, and non-linear bus transaction growth under heavy data contention.

Proceedings of the Second International Workshop on Data Management on New Hardware (DaMoN 2006), June 25, 2006, Chicago, IL, U.S.A.

2. PRIOR WORK
Prior work considers multiple facets of OLTP performance evaluation, but characterizations of servers with more than four processors are absent in the literature due to the great expenditures required to assemble large-scale configurations. General performance evaluation is covered by [1], [3], [6], and [9]; [6] offers a rich description of OLTP workloads at the database server level outside the space of synthetic benchmarks and [1] provides microarchitectural characterizations of loads including OLTP. In hardware space, coherency effects are discussed in [2], cache design OLTP performance effects in [10] and [14], and simultaneous multithreading in [12]. A synopsis of scalable locking algorithms is covered in [13]. The performance effects of targeted software design are covered in [11]. And finally, the compiler effects on OLTP workloads are discussed in [4] and [15].

3. OLTP WORKLOADS
An OLTP workload, such as the synthetic TPC-C Benchmark[2] [16], describes the transactional database support required to support on-line customer transaction requests. OLTP workloads, with the exception of an initial ramp-up period, execute in a homogenous, steady state where the system behavior over short periods is fully representative of an entire run. This significantly eases analysis and makes such workloads an attractive study target. Public benchmark specifications also contribute to workload longevity and allow long-term analysis relevance.

Transaction processing is characterized at the processor level by long, per-transaction instruction counts (IPX) of several million instructions and large instruction cache footprints. Issued instructions are predominately integer calculations with branch

[1] Intel and Itanium 2 are registered trademarks of Intel Corporation

[2] The TPC-C Benchmark is a trademark of the Transaction Processing Performance Council

instruction issue quite common: occurring as often as one to two dozen other instructions. The data references under OLTP involve random, single row database index lookups and serialized, chained pointer dereferences of complex, database server data structures. Overall performance is highly dependant on the cache and memory hierarchy; row lookups are dependent on memory latencies, and critical internal data structures must remain cache resident. At the application level, the high IPX rates reflect the complexity of implementing fail-safe, transactional support on top of large-scale memory management, scalable I/O support, mutual exclusion guarantees, query optimization, and other features. At the system level very large configurations are required for high-end results. For example, the top, non-clustered TPC-C Itanium performance result (as of March 2006) from Hewlett-Packard [5] required a full terabyte of memory and 1742 disks. The total system cost (including support and software) is quoted in excess of $5.9 million.

4. CONFIGURATION AND WORKLOAD

All measured results of Section 5 and Section 6 were collected with the server configuration and workload described in the following section. We used a large server configuration in order to provide a realistic approximation of an enterprise customer operating under an extremely high transaction load with a very large database. 32 Intel Itanium 2 processors (1.50GHz with 6MB L3 cache, 256KB L2) were installed on our server which implemented an 8 node cache coherent NUMA (ccNUMA) architecture and ran a mainstream, commercial operating system. 256GB of main memory was configured for use under our OLTP load. The system and OLTP load parameters were configured to impose a heavy I/O load during data collection in order to sustain a kernel overhead of 20% to 25% of the total execution time in each test. Data storage was supplied by a total of 1260 15K rpm disks, and I/O bandwidth was roughly 960 MB/sec under load. Table 1 outlines a set of basic system metrics recorded when establishing a baseline result.

The OLTP load itself was implemented as 2-tier, client-server configuration modeling a warehouse distribution system. The database was populated with synthetic data in a TPC-C schema, and the five TPC-C defined transactions were used to implement a generalized OLTP workload on a database size in excess of 5 terabytes. A total of approximately 1200 emulated clients issued a defined ratio of the transactions from 8 dual-processor servers. The workload parameters remained fixed for each performance evaluation run. In other words, the I/O requirements per transaction remained fixed, and the data foot print per transaction was inversely proportion to total throughput. Audited OLTP results, on the other hand, require linear scaling of the database size as throughput increases.

While a configuration of this size is difficult and costly to maintain given the short mean time to failure (MTTF) of a large disk array, it is necessary in order to impose the effects of disk and network I/O on individual processors. These effects, in turn, impose user to kernel transitions, interrupts, asynchronous procedure calls, and context switches. Stresses beyond the processor pipeline and memory bus are exhibited (interrupt distribution for example), and these require attention to all facets of system and software design. Looking forward, as multicore processor usage becomes commonplace, it will not be surprising to find large-scale OLTP performance work serve as a model for desktop software development.

Table 1. Basic system metrics on the 32 processor server

Metric	Rate
Total user processor time	74.8%
Total system processor time	23.4%
Total system idle time	1.8%
Total I/Os per second	123.2K
Write I/Os per OLTP transactions executed	9.7
Read I/Os per OLTP transactions executed	14.9

5. ITANIUM OLTP CHARACTERIZATION

Native Intel Itanium 2 processor support is available from major database vendors, and the top Itanium TPC-C results [17] occupy slots in the top 10 throughput results, demonstrating solid, scalable database performance capabilities. As such, we present a processor level characterization of the current state-of-the-art.

5.1.1 Performance Characterization Support

In OLTP work, an Itanium processor is particularly valuable due to its ability to characterize the workload at the microarchitectural level, using a robust set of sampled performance events [7]. We use stall event accounting to describe the causes of those cycles where instructions fail to be retired by the processor, i.e. stall cycles. Instruction issue is initiated in-order on the Itanium and a stalling instruction prevents progress of all subsequent instructions in the processor pipeline. This is in comparison to out-of-order execution where multiple, independent instructions may be in flight or simultaneously experiencing unrelated stall events. In the in-order case, execution stall cycles may be directly associated with specific stages and events in the processor pipeline. The Itanium pipeline is decoupled into a front-end and a back-end; we break total back-end cycle time down into the following categories:

1. Front-end stalls (FE): cycles spent by the back-end waiting for front-end instruction fetch and decode. This includes instruction cache miss overhead.
2. General register load stalls (GR): cycles typically resulting from data (not instruction) cache and memory hierarchy latencies. Could include scoreboard stalls.
3. TLB stalls (TLB): cycles required for virtual to physical address lookups in the translation look-aside buffer (TLB) or via the hardware page walker (HPW); may include L2 to L1 communication stall time.
4. Other stalls (OTH): cycles attributable to register stack engine misses, branch misprediction, exceptions, or interrupts. Least significant under OLTP loads.
5. Not stalled (NOT): cycles retiring instructions.

5.1.2 OLTP Characterization

Under an OLTP steady state workload, stall cycle accounting provides a processor level perspective of the performance

constraints imposed by a system, and Figure 1 and Table 2 illustrate the cycle accounting of a measurement made on the large-scale, 32 processor system and database described in Section 4. In this case, the database server implemented a full set of software performance enhancements which will be used as a baseline for comparative purposes in Section 6. The cycle accounting presents the average cycle breakdown across all 32 processors.

Overall, 61% of total time during OLTP steady state on our system is the result of data load latencies from the cache and memory hierarchy (GR), but the kernel time impact versus user time is much more pronounced. Kernel data is much less likely to be cache resident: typically the effect of remapping kernel defined I/O management structures in the cache after long lasting (from the processor perspective) I/O events. Kernel time GR cycles are also representative of inter-processor synchronization required at the kernel level to protect shared data structures. Table 3 uses cache miss statistics and the cache latencies specified in [7] combined with proprietary platform latencies of our system under test to roughly estimate the percentage of time spent waiting for data delivery from different sources in the cache and memory hierarchy. This estimate actually includes cycle components for both GR and TLB stall sources since page translations are ultimately located in the cache and memory hierarchy and are included in the cache events. In the large-scale server case of this study, roughly 52% of total stalls are spent waiting for memory, but the L1 and L2 effects should not be dismissed as these account for more than 40% of overall data latencies. We explore approaches to alleviate L3 miss overheads using platform specific insight in Section 6.

Although the front-end and the TLB stall cycles individually account for only 9% of the total stall time in Table 2, this remains noteworthy, as described in Section 6. Very large memory configurations impose considerable overhead without careful attention to memory locality concerns, and the results reflect good behavior of highly optimized code. Large executable image size and complex code paths also present difficulties when instruction cache space is not considered. Finally, although time spent freely executing without stalls is preferred to stall time, this component may actually reflect superfluous instruction issue, for example, the unnecessary overhead of prolog and epilog code in very short functions. Overall, however, the cycle time accounting provides a precise way to quantify the effects of the optimizations described in Section 6.

Table 2. The contribution percentages of Figure 1.

Source	Kernel	User	Total
FE	5%	9%	8%
GR	69%	57%	61%
TLB	12%	8%	9%
OTH	5%	6%	5%
NOT	9%	20%	17%
Total	**100%**	**100%**	**100%**

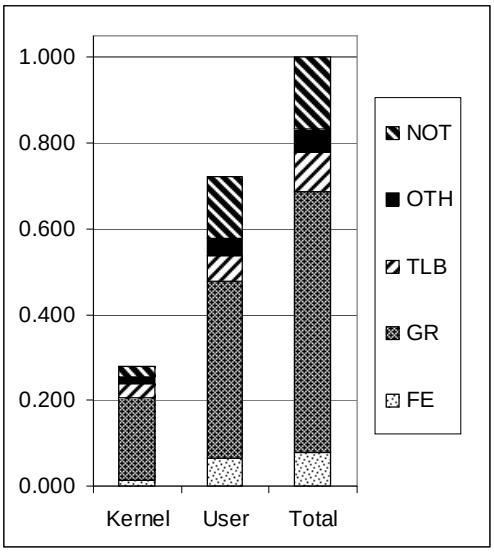

Figure 1. Cycle time per optimized OLTP transaction.

Table 3. Estimated cache and memory hierarchy data delivery cycle breakdown by data source.

Data Source	% of GR Cycles
L1 hit latency	20%
L2 hit latency	21%
L3 hit latency	7%
L3 miss latency	52%
Total	**100%**

6. OPTIMIZATIONS AND RESULTS

A set of performance optimizations were implemented during a long development process in our DBMS based on processor level performance feedback. For each optimization we'll briefly describe the problem as identified in stall cycle accounting and then characterize the performance degradation of explicitly disabling the optimization. We quantify the effects by drawing attention to specific stall cycle accounting events and reporting the rates of more specialized performance counters.

Table 4 describes the stall cycle accounting for each optimization, normalized by the base stall cycles of the fully optimized server. The table allows one to quickly identify those areas in which the most significant effects of a performance optimization are exhibited. For instance, when comparing the base results (ALL) and the results with disabled profile guided optimization (-PGO), we see 11% more absolute cycles spent in front-end (FE) stalls, and a 123% (1.23X) increase in the total cycles per transaction is exhibited. Note that throughput is approximately the inverse of the normalized cycle growth, and minor sample-to-sample processor event data variances exist. In this case, because PGO directly affects embedded branch prediction and code ordering, we expect to see the most significant effects associated with the front-end. Secondary effects, however, are also exhibited across the other stall sources and are not insignificant and account for half of the difference, as indicated in the "-PGO" row of Table 4.

Table 4. Back-end cycle accounting differences for disabling either large page allocation (-LP), superlatches (-SL), profile-guided optimization (-PGO), client-side data partitioning (-CS), or server-side data partitioning (-SS). All are normalized by the fully optimized baseline (ALL) execution cycles. The relative total cycle time (Total column) is expressed per general OLTP workload transaction during steady state. See Section 5.1.1 for stall descriptions.

	Stall Sources					
Opt.	**OTH**	**FE**	**TLB**	**NOT**	**GR**	**Total**
ALL	5%	8%	9%	17%	61%	**100%**
-LP	6%	9%	14%	17%	63%	**109%**
-SL	5%	8%	9%	16%	71%	**110%**
-PGO	9%	19%	12%	19%	64%	**123%**
-CS	6%	9%	10%	17%	77%	**118%**
-SS	5%	8%	10%	18%	65%	**106%**

6.1 Profile Guided Optimization

6.1.1 Problem Description

As briefly described in Section 5.1.1, when a stall occurs on the Itanium 2 processor, the entire pipeline stops execution. Our stall cycle accounting includes a front-end instruction fetch and decode component, the area where a compiler has direct impact on performance by optimizing machine code to reduce front-end stalls. A compiler does this in several ways. First, it must emit efficient and concise instruction sequences. This limits the instruction cache footprint, reduces the overall IPX rate, and reduces secondary instruction footprint effects in the L2 and L3 caches which contain data and instructions, both contending for space. Second, the compiler should recognize and predict the favored control path of a workload scenario. This limits the effects of branch misprediction by enabling Itanium instruction predication and the use of branch hints. Third, the compiler should further reduce the IPX rate by using a handbag of optimization options including inlining and both data and control speculation.

Profile guided optimization (PGO) uses image instrumentation and run-time profiling to identify critical paths and allow the compiler to generate efficient code, reduce the overall instruction footprint, and therefore reduce the number of fetch and decode stall cycles. For the system under test, despite the long memory latencies exhibited on the ccNUMA platform, front-end pressures remain a principal performance concern. These pressures include instruction cache pressure, instruction TLB pressure, and branch misprediction pressure.

6.1.2 Results

Table 4 outlines the effects of PGO optimization in the comparison of the "ALL" and "-PGO" rows. The data was collected by running an engine compiled without using PGO. There is a 1.23X increase in the number of stall cycles and a 20% loss in transactional throughput. This is primarily reflected in the FE stall time associated with instruction decoding, along with an increase in GR stalls resulting from the effects of a larger instruction footprint resident in the unified L2 cache. These secondary effects are significant and are primarily the result of reducing the cache resident instruction working set and freeing up space for data and page tables.

Table 5 includes some key metrics exhibited at run-time under the OLTP load by disabling PGO. Code generation is clearly improved with the help of the profile feedback. Without feedback, the IPX rate is 1.21X of the fully optimized results, and the processor cycles required per instruction (CPI) is up 1.08X, reflecting new constraints imposed on the processor. The secondary metrics in the table indicate a reduction in correct branch prediction, and a larger instruction footprint when running without PGO. Inlining rates are reflected by the number of IA-64 *alloc* instructions issued during runtime. This instruction allocates space in the register stack engine (RSE) for general register use and occurs at the beginning of a function. Without PGO feedback during optimization, roughly 1.6X the number of explicit function calls are executed during the workload.

Table 5. Disabled PGO metrics versus full optimization

Basic Metrics	**Versus Full Opt.**
User IPX	1.21X
User CPI	1.08X
Throughput	0.80X
Secondary Metrics	
Instructions Between Mispredicted Branches	0.69X
L3 Instruction References	2.09X
***ALLOC* Instruction Issue**	1.61X

6.2 Data Partitioning

6.2.1 Problem Description

NUMA based architectures exhibit divergent latencies between local and remote memory as well as local and remote cache-to-cache (C2C) transfers. Given the long remote latencies on ccNUMA architectures, we expect that increasing localized data references will have a significant effect, especially since Table 3 indicates over 50% of data stall time is the result of misses at the L3 cache level. In this section we show that it is not enough for the DBMS to be NUMA aware. Huge gains can also be achieved if the workload is partitioned on the client side. The database server under test supports locality in two different senses that we explore: database page buffer allocation locality and internal DBMS data structure allocation locality.

When a page buffer allocation request is issued, the buffer manager allocates a buffer from the node where a thread is running. Since most databases ensure there is only a single copy of each page in the buffer pool, this scheme breaks if page references are not co-located on the same node with page allocation. To reduce cross-node buffer accesses we affinitize each client to a node. Furthermore, each client is assigned an exclusive range of warehouses.

Internal DBMS data structures include all non-database page data used in the database server during query execution. These uses may include network connection contexts, internal locking structures, memory management bookkeeping, and query plans. It's not unreasonable to expect a majority of references to be to

these data structures. Our DBMS automatically localizes such structures to local memory when NUMA architectures are detected.

6.2.2 Results

We collected two sets of results by first modifying the client-side transaction parameters by increasing the size of the applicable, per-client warehouse range so that all clients had non-overlapping ranges. This is a common and allowable practice under TPC-C specification, and was trivially accomplished since the number of configured clients matched the number of NUMA nodes on the database server. We then separately disabled a server side feature that enabled a specific class of data structure localization. Table 4 demonstrates the effects of disabling client-side and server-side data partitioning in the "-CS" and "-SS" rows respectively. Not unexpectedly, there are significant impacts in GR stall time due to an overall increase in memory latencies when client and server side partitioning are disabled and local node memory reference rates decrease compared to remote node reference rates. Table 6 shows the runtime effects: average memory latencies are up by 1.08X for client-side partitioning and 1.05X for server-side partitioning. There are smaller changes in IPX rates since transactions are doing essentially the same amount of work, but the user CPI reductions again reflect the benefits of limiting remote accesses.

Table 6. Metrics for disabling client-side (CS) and server-side (SS) data partitioning.

Basic Metrics	-CS vs. Full Opt.	-SS vs. Full Opt.
User IPX	0.98X	1.03X
User CPI	1.20X	1.04X
Throughput	0.84X	0.94X
Secondary Metrics		
Memory Latency	1.08X	1.05X

6.3 Cache Coherency and Superlatches

6.3.1 Problem Description

In the context of thousands or tens-of-thousands transactions executed per second, mutual exclusion is an absolute necessity of database server design. In modern microprocessors, mutual exclusion is enforced by the use of atomically executed instructions which determine exclusive cache line ownership. The instructions often incur a bus transaction due to cache coherency protocols that contribute to the total C2C traffic. As shown in Table 3, an estimated 52% of GR stall cycles result from L3 miss latencies, the manifestation of both memory and C2C transfer latencies. On platforms where C2C latencies are considerably higher than memory latencies, cumulative L3 miss latency may be comprised primarily of C2C transfers. While not all C2C activity is the result of locking and mutual exclusion (for instance, read-only, shared copies of global data transferred between processors), for this workload it is the significant portion. Optimizations to reduce the impact of C2C latencies can take several forms. The simplest involves isolating locks or commonly modified data onto separate cachelines to reduce false sharing, i.e. unnecessary coherency traffic. Algorithmic design may also be possible to reduce lock contention, for instance by completely eliminating a lock or by partitioning data and protecting it by multiple locks instead of a single lock.

As shown in Figure 2, the top 10 cache lines account for 17% of the total stall cycles. In this section we will explain why those cache lines are hot and we will propose a new mechanism (superlatches) that will reduce their latency. Our analysis showed that those cache lines are associated with the latches (i.e. multiple readers, single writer semaphores) that protect the root pages in the database index b-trees. B-tree data structures lay out index records in a balanced binary tree, and exhibit little record modification at the root level. Root page access under latch control, however, is required for any record lookup. Because most access to the root pages are read only, latches do not scale well due to the C2C transfers incurred by exclusively accessing and maintaining a list of latch owners. Superlatches are a solution to this problem. The DBMS dynamically looks for high reference count latches at run time. When a hot latch is detected and is primarily read only, it is promoted into a superlatch. The promotion process involves tagging the original latch as a superlatch and then cloning it into per-processor latch copies. From this point only the per-processor latch is required for read operations. For write operations, though, all the local latches must be acquired and the superlatch demoted to a "normal" latch again. Obviously this is an expensive operation, but it rarely happens.

Table 7. Disabled superlatch metrics versus fully optimized results.

Basic Metrics	Versus Full Opt.
User IPX	0.95X
User CPI	1.19X
Throughput	0.91X
Secondary Metrics	
C2C Transfers	1.10X
Read & invalidate line	1.52X
Memory latency	0.88X

6.3.2 Results

Table 4 is again the source for comparing of the benefits of utilizing superlatches. Overall there's a 1.10X increase in the total number of stall cycles comparing the "ALL" and the "-SL" rows. This time is fully attributed to GR time and an increase in coherency traffic, as expected. Table 7 represents the general coherency traffic effects; disabling the read only lock implementations increases the user level CPI by 1.19X, a reflection of the additional bus transactions and cacheline invalidations despite along with a decrease in the instruction complexity in acquiring normal latches. The secondary metrics in Table 7 reflect this too: read and invalidate line requests are up tremendously, 1.52X. An interesting side effect from this is that the average memory latency actually decreases when superlatches are disabled because the platform exhibits better C2C latencies versus memory. Although the data latencies are down, total accesses to and cumulative latencies from the most hotly contended lines are up. Figure 2 represents this effect. Itanium 2 processor performance counter support provides support for data

address and access latency sampling, and the figure represents the cumulative latencies of the top 500 cachelines recorded during the OLTP workload on the machine described in Section 4. Read-only locking effectively eliminates the high costs associated with the top 10 cachelines that directly map to the root page latches of major b-tree structures when read/write locking is used.

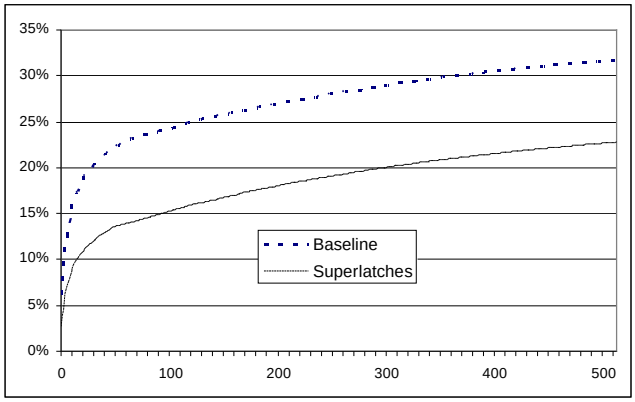

Figure 2. Cumulative data latencies for the top 500 cachelines for a baseline (standard) latch versus superlatch implementations expressed as a percentage of the total L3 miss latency.

Table 8. Disabled large virtual page allocation metrics

Basic Metrics	Versus Full Opt.
User IPX	0.99X
User CPI	1.08X
Throughput	0.91X
Secondary Metrics	
L2 Data TLB	2.25X
L2 to L1 Data TLB Transfers	1.01X
L3 Data Read Miss	1.12X
Privileged Level Change	1.94X
Kernel IPX	1.09X

6.4 Large Pages

6.4.1 Problem description

256GB is a huge memory space, especially for OLTP loads which exhibit random, non-local data references to database rows. The 2nd level DTLB on the Itanium 2 processor contains 128 entries, covering only a megabyte of virtual space when using 8KB virtual pages. On the other hand, the processor supports up to a 4GB virtual page, and most operating systems support large virtual page allocation (1MB+) in both memory management APIs, static/global data allocation, and instruction page allocation. For applications operating in very large memory spaces, depending on the workload, there can be a large translation overhead which may be alleviated by taking advantage of larger virtual page support.

6.4.2 Results

Table 8 describes the effects of disabling 16MB large-page allocations of dynamic data, static data, and instruction pages in the "ALL" and "-LP" rows. As expected, the largest effects are from the TLB stall source. A small, secondary effect is exhibited as GR stalls in Table 4 because fewer page translations are cache resident. The IPX rate remains constant; CPI is up when standard page size is used since these references incur higher translation probability. Since the L1DTLB is a fixed 4KB size, there is little change in the number of L2 to L1DTLB transfers. L3 data read misses are up by 1.12X, reflecting space consumption from cache resident translations. The total number of privileged level transitions is up 1.94X from explicit page table lookups by the OS.

7. CONCLUSION

We used cycle time accounting to outline the performance constraints of a general OLTP workload on a large ccNUMA platform and describe targeted optimization on the platform. Obviously, memory and C2C transfer latencies must be taken into account when designing for performance and scalability, but surprisingly, the compiler still has a tremendous impact on overall performance due to the size and complexity of modern database server executable images. Ultimately the software, both on the server and client sides along with the compiler, must operate with knowledge of platform constraints in order to achieve optimal performance. Although at present the constraints explored above are exhibited on expensive, enterprise class machines, multi-core processor availability signals further deepening of cache and memory hierarchies on commodity parts and platforms. Scalable database performance on tomorrow's commodity parts will require approaches similar to those explored here.

8. ACKNOWLEDGMENTS

Our thanks go to Martin Redmond of Intel for supporting the machine configuration and necessary data collection in this work.

9. REFERENCES

[1] Barroso, L.A.; Gharachorloo, K.; Bugnion, E. *Memory system characterization of commercial workloads*. Computer Architecture, 1998. Proceedings. The 25th Annual International Symposium on, 27 June-1 July 1998 Page(s):3 - 14

[2] Black, J.E.; Wright, D.F.; Salgueiro, E.M. *Improving the performance of OLTP workloads on SMP computer systems by limiting modified cache lines*. Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on 27 Oct. 2003 Page(s):21 - 29

[3] Hankins, R.; Diep, T.; Annavaram, M.; Hirano, B.; Eri, H.; Nueckel, H.; Shen, J.P. *Scaling and characterizing database workloads: bridging the gap between research and practice*. Microarchitecture, 2003. MICRO-36. Proceedings. 36th Annual IEEE/ACM International Symposium on, 2003 Page(s):151 - 162

[4] Hoflehner, Gerolf; Kirkegaard, Knud; Skinner, Rod; Lavery, Daniel; Lee, Yong-Fong; Li, Wei *Compiler Optimizations for Transaction Processing Workloads on Itanium Linux Systems.* Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture MICRO 37, December 2004.

[5] Hewlett-Packard Corporation, *HP Integrity TPC-C Results*. Transaction Processing Performance Council website,

reported November 29, 2005. http://www.tpc.org/results/individual_results/HP/hp_orca1tb_win64_ex.pdf

[6] Hsu, W.; Smith, A.; Young, H. *Characteristics of Production Database Workloads and the TPC Benchmarks.* IBM Systems Journal 40, No. 3, 2001

[7] Intel Corporation, *Intel® Itanium® 2 Processor Reference Manual For Software Development and Optimization*. ftp://download.intel.com/design/Itanium2/manuals/25111003.pdf, May 2004, Section 11.

[8] Intel Corporation, *Intel Itanium Software Developer's Manual, Volume 1, 2, and 3.* *http://developer.intel.com/design/Itanium/manuals/iiasdmanual.htm*

[9] Keeton, Kimberly; Patterson, David A.; He, Yong Qiang; Raphael, Roger C.; Baker, Walter E. *Performance characterization of a Quad Pentium Pro SMP using OLTP* workloads. ACM SIGARCH Computer Architecture News, Proceedings of the 25th annual international symposium on Computer architecture ISCA '98, Vol. 26(3)

[10] Kundu, Partha; Annavaram, Murali; Diep, Trung; Shen, John *A Case for Shared Instruction Cache on Chip Multiprocessors running OLTP*. ACM SIGARCH Computer Architecture News, Vol. 32, No. 3, June 2004 Pages: 11-18

[11] Kunkel, S.; Armstrong, B.; Vitale, P. *System optimization for OLTP workloads.* Micro, IEEE, Volume 19, Issue 3, May-June 1999, Page(s):56 - 64

[12] Lo, J. L; Barroso, L.A.; Eggars, S.J.; Garachorloo, K.; Levy, H. M.; Parekh, S. S. *An analysis of database workload performance on simultaneous multithreaded processors*. ACM SIGARCH Computer Architecture News, Proceedings of the 25th annual international symposium on Computer architecture ISCA '98, Vo. 26(3). April 1998.

[13] Mellor-Crummey, John M.; Scott, Michael L. *Algorithms for scalable synchronization on shared-memory multiprocessors.* ACM Transactions on Computer Systems (TOCS), Vol. 9(1), Feb. 1991.

[14] Nurvitadhi, Eriko; Chalainanont, Nirut; Lu, Shih-Lien *Characterizations of L3 Cache Behavior of SPECjAppServer2002 and TPC-C*. Supercomputing ICS'05, Proceedings of the 19th annual international conference on. June 2005.

[15] Ramirez, Alex; Barroso, Luiz André; Gharachorloo, Kourosh; Cohn, Robert; Larriba-Pey, Josep P.; Lowney, Geoffrey; Valero, Mateo *Code layout optimizations for transaction processing workloads*. ACM SIGARCH Computer Architecture News , Proceedings of the 28th annual international symposium on Computer architecture ISCA '01, Volume 29 Issue 2 May 2001

[16] Transaction Processing Performance Council *TPC Benchmark C, Standard Specification, Revision 5.6*. http://www.tpc.org/tpcc/spec/tpcc_current.pdf, December 2005.

[17] Transaction Processing Performance Council, *Top 10 Non-clustered TPC-C by Performance Version 5 Results.* http://www.tpc.org/tpcc/results/tpcc_perf_results.asp?resulttype=noncluster. As of April 2004