Performance Analysis Using Very Large Memory on the 64-bit AlphaServer System

Tareef S. Kawaf
D. John Shakshober
David C. Stanley

Optimization techniques have been used to deploy very large memory (VLM) database technology on Digital's AlphaServer 8400 multiprocessor system. VLM improves the use of hardware and software caches, main memory, the I/O subsystems, and the Alpha 21164 microprocessor itself, which in turn causes fewer processor stalls and provides faster locking. Digital's 64-bit AlphaServer 8400 system running database software from a leading vendor has achieved the highest TPC-C results to date, an increased throughput due to increased database cache size, and an improved scaling with symmetric multiprocessing systems.

Digital's AlphaServer 8400 enterprise-class server combines a 2-gigabyte-per-second (GB/s) multiprocessor bus with the latest Alpha 21164 64-bit microprocessor.[1] Between October and December 1995, an AlphaServer 8400 multiprocessor system running the 64-bit Digital UNIX operating system achieved unprecedented results on the Transaction Processing Performance Council's TPC-C benchmark, surpassing all other single-node results by a factor of nearly 2. As of September 1996, only one other computer vendor has come within 20 percent of the AlphaServer 8400 system's TPC-C results.

A memory size of 2 GB or more, known as very large memory (VLM), was essential to achieving these results. Most 32-bit UNIX systems can use 31 bits for virtual address space, leaving 1 bit to differentiate between system and user space, which creates difficulties when attempting to address more than 2 GB of memory (whether virtual or physical).

In contrast, Digital's Alpha microprocessors and the Digital UNIX operating system have implemented a 64-bit virtual address space that is four billion times larger than 32-bit systems. Today's Alpha chips are capable of addressing 43 bits of physical memory. The AlphaServer 8400 system supports as many as 8 physical modules, each of which can contain 2 CPUs or as much as 2 GB of memory.[2] Using these limits, database applications tend to achieve peak performance using 8 to 10 CPUs and as much as 8 GB of memory.

The examples in this paper are drawn primarily from the optimization of a state-of-the-art database application on AlphaServer systems; similar technical considerations apply to any database running in an Alpha environment. As of September 1996, three of the foremost database companies have extended their products to exploit Digital's 64-bit Alpha environment, namely Oracle Corporation, Sybase, Inc., and Informix Software, Inc.

The sections that follow describe the TPC-C workload and discuss two database optimizations that are useful regardless of memory size: locking intrinsics and OM instruction-cache packing. (OM is a post–link time optimizer available on the Digital UNIX operating system.)[3] VLM experimental data is then presented in the section VLM Results.

TPC-C Benchmark

The TPC-C benchmark was designed to mimic complex on-line transaction processing (OLTP) as specified by the Transaction Processing Performance Council.[4] The TPC-C workload depicts the activity of a generic wholesale supplier company. The company consists of a number of distributed sales districts and associated warehouses. Each warehouse has 10 districts. Each district services 3,000 customer requests. Each warehouse maintains a stock of 100,000 items sold by the company. The database is scaled according to throughput (that is, higher transaction rates use larger databases). Customers call the company to place new orders or request the status of an existing order.

Method

The benchmark consists of five complex transactions that access nine different tables.[5] The five transactions are weighted as follows:

1. Forty-three percent—A new-order transaction places an order (an average of 10 lines) from a warehouse through a single database transaction and updates the corresponding stock level for each item. In 99 percent of the new-order transactions, the supplying warehouse is the local warehouse and only 1 percent of the accesses are to a remote warehouse.
2. Forty-three percent—A payment transaction processes a payment for a customer, updates the customer's balance, and reflects the payment in the district and warehouse sales statistics. The customer resident warehouse is the home warehouse 85 percent of the time and is the remote warehouse 15 percent of the time.
3. Four percent—An order-status transaction returns the status of a customer order. The customer order is selected 60 percent of the time by the last name and 40 percent of the time by an identification number.
4. Four percent—A delivery transaction processes orders corresponding to 10 pending orders for each district with 10 items per order. The corresponding entry in the new-order table is also deleted. The delivery transaction is intended to be executed in deferred mode through a queuing mechanism. There is no terminal response for completion.
5. Four percent—A stock-level transaction examines the quantity of stock for the items ordered by each of the last 20 orders in a district and determines the items that have a stock level below a specified threshold. This is a read-only transaction.

The TPC-C specification requires a response time that is less than or equal to 5 seconds for the 90th percentile of all but the delivery transaction, which must complete within 20 seconds.

In addition, the TPC-C specification requires that a complete checkpoint of the database be done. A checkpoint flushes all transactions committed to the database from the database cache (memory) to nonvolatile storage in less than 30 minutes. This important requirement is one of the more difficult parts to tune for systems with VLM.[6]

Results

Table 1 gives the highest single-node TPC-C results published by the Transaction Processing Performance Council as of September 1, 1996.[4]

For a complete TPC-C run, a remote terminal emulator must be used to simulate users making transactions. For performance optimization purposes, however, it is convenient to use a back-end-only version of the benchmark in which the clients reside on the server. The transactions per minute (tpm) derived in this environment are called back-end tpm in Table 2 and cannot be compared to the results of audited runs (such as those given in Table 1). However, when a performance improvement is made to the back-end-only environment, performance improvements are clearly seen in the full environment.

Tuning for the system is iterative. For each data point collected, clients were added to try to saturate the server; then the amount of memory was varied for the database cache. A trade-off between database memory, system throughput, and checkpoint performance required us to tune each data point individually. The system was configured with a sufficient number of disk drives and I/O controllers to ensure that it was 100-percent CPU saturated and never I/O limited. The experiments reported in this paper use database sizes of approximately 100 GB, spread over 172 RZ29 spindles and 7 KZPSA adapter/HSZ40 controller pairs, with each HSZ40 controller using 5 small computer systems interface (SCSI) buses.

Tuning Specific to Alpha

UNIX databases on Digital's Alpha systems were first ported in 1992. For database companies to fully use the power of Alpha's 64-bit address space, each database vendor had to expand the scope of its normal 32-bit architecture to make use of 64-bit pointers. Thus, each database could then address more than 2 GB of physical memory without awkward code segments or other manipulations to the operating system to extend physical address space.

By 1994, most vendors of large databases were offering 64-bit versions of their databases for Digital's Alpha environment. As a group chartered to measure database performance on Alpha systems, Digital's Computer Systems Division (CSD) Performance Group worked with each database vendor and with the Digital System Performance Expertise Center to improve performance.

Table 1
TPC-C Results

System	Throughput	Price/ Performance	Number of CPUs	Date
AlphaServer 8400 5/350, Oracle Rdb7 V7.0, OpenVMS V7.0	14,227 tpmC	$269/tpmC	10	May 1996
AlphaServer 8400 5/350, Sybase SQL Server 11.0, Digital UNIX, iTi Tuxedo	14,176 tpmC	$198/tpmC	10	May 1996
AlphaServer 8400 5/350, Informix V7.21, Digital UNIX, iTi Tuxedo	13,646.17 tpmC	$277/tpmC	10	March 1996
Sun Ultra Enterprise 5000, Sybase SQL Server V 11.0.2	11,465.93 tpmC	$191/tpmC	12	April 1996
AlphaServer 8400 5/350, Oracle7, Digital UNIX, iTi Tuxedo	11,456.13 tpmC	$286/tpmC	8	December 1995
AlphaServer 8400 5/300, Sybase SQL Server 11.0, Digital UNIX, iTi Tuxedo	11,014.10 tpmC	$222/tpmC	10	December 1995
AlphaServer 8400 5/300, Oracle7, Digital UNIX, iTi Tuxedo	9,414.06 tpmC	$316/tpmC	8	October 1995
SGI CHALLENGE XL Server, INFORMIX-OnLine V7.1, IRIX, IMC Tuxedo	6,313.78 tpmC	$479/tpmC	16	November 1995
HP 9000 Corporate Business Server, Sybase SQL Server 11, HP-UX, IMC Tuxedo	5,621.00 tpmC	$380/tpmC	12	May 1995
HP 9000 Corporate Business Server, Oracle7, HP-UX, IMC Tuxedo	5,369.68 tpmC	$535/tpmC	12	May 1995
Sun SPARCcenter 2000E Oracle7, Solaris, Tuxedo	5,124.21 tpmC	$323/tpmC	16	April 1996
Sun SPARCcenter 2000E, INFORMIX-OnLine 7.1, Solaris, Tuxedo	3,534.20 tpmC	$495/tpmC	20	July 1995
IBM RS/6000 PowerPC R30, DB2 for AIX, AIX, IMC Tuxedo	3,119.16 tpmC	$355/tpmC	8	June 1995
IBM RS/6000 PowerPC J30, DB2 for AIX, AIX, IMC Tuxedo	3119.16 tpmC	$349/tpmC	8	June 1995

Table 2
Amount of Memory versus Back-end tpm, Database-cache Miss Rate, and Instructions per Transaction

Database Memory (GB)	Back-end (Normalized tpm)	Relative Database-cache Miss (Percentage)	Relative Instructions per Transaction
1	1.0	1.0	1.0
2	1.3	0.73	0.75
3	1.5	0.58	0.63
4	1.6	0.50	0.57
5	1.7	0.42	0.50
6	1.8	0.40	0.45

Two optimizations generally realized 20 percent gains on Alpha systems.[7] These were

1. Optimization of spinlock primitives supported now by DEC C compiler intrinsics
2. OM profile-based link optimization, which performs instruction-cache packing during the final link of the database

In addition, the Digital UNIX operating system version 3.2 and higher versions have optimized I/O code paths and support advanced processor affinity and other scheduling algorithms that have been optimized for enterprise-class commercial performance. With these optimizations, database performance on Digital's Alpha systems has been significantly improved.

Lock Optimization

Locks are used on multiprocessor systems to synchronize atomic access to shared data. A lock is either unowned (clear) or owned (set). A key design decision leading to good multiprocessor performance and scaling is partitioning the shared data and associated locks. The discussion of how to partition data and associated locks to minimize contention and the number of locks required is beyond the scope of this paper.

The implementation of locks requires an atomic test-and-set operation. On a particular system, the implementation of the lock is dependent on the primitive test-and-set capabilities provided by the hardware.

Locks are used to synchronize atomic access to shared data. A shared data element that requires atomic access is associated with a lock that must be acquired and held while the data is modified. On multiprocessing systems, locks are used to synchronize atomic access to shared data. A sequence of code that accesses shared data protected by a lock is called a critical section. A critical section begins with the acquisition of a lock and ends with the release of that lock. Although it is possible to have nested critical sections where multiple locks are acquired and released, the discussion in this section is limited to a critical section with a single lock.

To provide atomic access to shared data, the critical section running on a given processor locks the data by acquiring the lock associated with the shared data. In the simplest case, if a second processor tries to acquire access to shared data that is already locked, the second processor loops and continually retries the access (spins) until the processor owning the lock releases it. In a complex case, if a second processor tries to acquire access to shared data that is already locked, the second processor loops a few times and then, if the lock is still owned by another processor, puts itself into a wait state until the processor owning the lock releases it.

The Alpha Architecture Reference Manual specifies that "…the order of reads and writes done in an Alpha implementation may differ from that specified by the programmer."[8] Therefore, process coordination requires a special test-and-set operation that is implemented through the load-locked/store-conditional instruction sequence. To provide good performance and scaling on multiprocessor Alpha systems, it is important to optimize the test-and-set operation to minimize latency. The test-and-set operation can be optimized by the following methods:

- Use an in-lined load-locked/store-conditional sequence through an embedded assembler or compiler intrinsics.
- Preload a lock using a simple load operation prior to a load-locked operation.
- If a lock is held, spin on a simple load instruction rather than a load-locked instruction sequence.

The basic hardware building block used to implement the acquisition of a lock is the test-and-set operation. On many microprocessors, an atomic test-and-set operation is provided as a single instruction. On an Alpha microprocessor, the test-and-set operation needs to be built out of load-locked (LDx_L) and store-conditional (STx_C) instructions. The LDx_L … STx_C instructions allow the Alpha microprocessor to provide a multiprocessor-safe method to implement the test-and-set operation with minimal restrictions on read and write ordering. The load-locked operation sets a locked flag on the cache block containing the data item. The store-conditional operation ensures that no other processor has modified the cache block before it stores the data. If no other processor has modified the cache block, the store-conditional operation is successful and the data is written to memory. If another processor has modified the cache block, the store-conditional operation fails, and the data is not written to memory. Optimizing the test-and-set sequence on Alpha systems is a complex task that provides significant performance gains.

Figure 1 shows code sequences that Digital's CSD Performance Group has given to database vendors to improve locking intrinsics in the Alpha environment. These code sequences can be used to implement spinlocks in the DEC C compiler on the Digital UNIX operating system.

Using OM Feedback

As previously mentioned, OM is a post–link time optimizer available on the Digital UNIX operating system. It performs optimizations such as compression of addressing instructions and dead code elimination through the use of feedback. The performance improvement provided by OM on Alpha 21164 systems is dramatic for the following two reasons.[3]

- The 21164 microprocessor has an 8-kilobyte (KB) direct-mapped instruction cache, which makes code placement extremely important. In a direct-mapped cache, the code layout and linking order maps one for one to its placement in cache. Thus a poorly chosen instruction stream layout or simply unlucky code placement within libraries can alter performance by 10 to 20 percent. Routines are frequently page aligned, which can increase the likelihood of cache collisions.
- The high clock rate of the Alpha 21164 microprocessor (300 to 500 megahertz [MHz]) requires a cache hierarchy to attempt to keep the CPU pipelines filled. The penalty of a first-level cache miss is 5 to 9 cycles, which means that an

```
//TEST_AND_SET implements the Alpha version of a test and set operation using
//the load-locked .. store-conditional instructions.  The purpose of this
//function is to check the value pointed to by spinlock_address and, if the
//value is 0, set it to 1 and return success (1) in R0.  If either the spinlock
//value is already 1 or the store-conditional failed, the value of the spinlock
//remains unchanged and a failure status (0,2, or 3) is returned in R0.
//
//The status returned in R0 is one of the following:
//  0 - failure (spinlock was clear; still clear, store-conditional failed)
//  1 - success (spinlock was clear; now set)
//  2 - failure (spinlock was set; still set, store-conditional failed)
//  3 - failure (spinlock was set; still set)
//
#define TEST_AND_SET (spinlock_address) asm( "ldl_l     $0,($16);  "    \
                                             "or        $0,1,$1;   "    \
                                             "stl_c     $1,($16);  "    \
                                             "sll       $0,1,$0;   "    \
                                             "or        $0,$1,$0   ",   \
                                             (spinlock_address));

// BASIC_SPINLOCK_ACQUIRE implements the simple case of acquiring a spinlock. If
// the spinlock is already owned or the store-conditional fails, this function
// spins until the spinlock is acquired. This function doesn't return until the
// spinlock is acquired.
//
#define BASIC_SPINLOCK_ACQUIRE(spinlock_address)                          \
{        long status = 0;                                                 \
                                                                          \
         while (1)                                                        \
         {                                                                \
                  if (*(spinlock_address) == 0)                           \
                  {                                                       \
                            status = TEST_AND_SET (spinlock_address);     \
                            if (status == 1)                              \
                           {                                              \
                                     MB;                                  \
                                   break;                                 \
                           }                                              \
                  }                                                       \
         }                                                                \
}
```

Figure 1
Code Sequences for Locking Intrinsics

instruction-cache miss rate of 10 to 12 percent can effectively stall the CPU 70 to 80 percent of the time. Conversely, decreasing the miss rate by 2 percent can increase throughput by 10 percent.

OM performs profile-based optimization. A program is first partitioned into basic blocks (that is, regions containing only one entrance and one exit), and instrumentation code is added to count the number of times each block is executed. The instrumented version of the program is run to create a feedback file that contains a profile of basic block counts. OM then uses the feedback to rearrange the blocks in an optimal way for the first-level caches on the Alpha chip. The details of the procedure for using OM may be found in the manpage for cc on the Digital UNIX operating system but can be summarized as follows:

- Build executable with -non_shared -om options, producing prog.
- Use pixie to produce prog.pixie (the instrumented executable) and prog.Addrs (addresses).
- Run prog.pixie to produce prog.Counts, which records the basic block counts.
- Now build prog again with -non_shared -om -WL, om_ireorg_feedback.

VLM Results

Figure 2 shows the increase in throughput realized when using VLM. Note that throughput nearly doubles as the amount of memory allocated to the database cache is varied from 1 GB to 6 GB. Of course, the overall system requires additional memory beyond the database cache to run UNIX itself and other

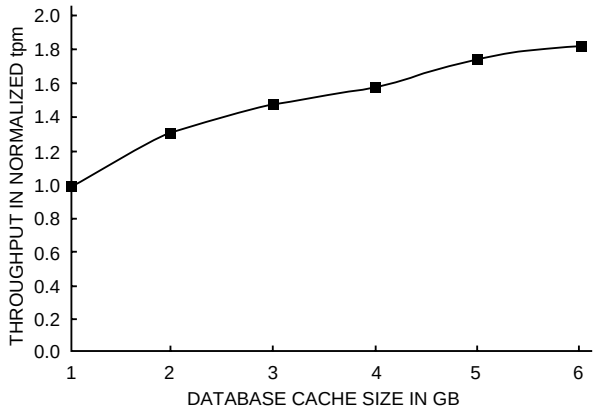

Figure 2
Database Cache Size versus Throughput

processes. For example, an 8-GB system allows 6.6 GB to be used for the database cache.

Performance Analysis

Why does the use of VLM improve performance by a factor of nearly 2? Using statistics within the database, we measured the database-cache hit ratio as memory was added. Figure 3 shows the direct correlation between more memory and decreased database-cache misses: as memory is added, the database-cache miss rate declines from 12 percent to 5 percent. This raises two more questions: (1) Why does the database-cache miss rate remain at 5 percent? and (2) Why does a small change in database-cache miss rates improve the throughput so greatly?

The answer to the first question is that with a database size of more than 100 GB, it is not possible to cache the entire database. The cache improves the transactions that are read-intensive, but it does not entirely eliminate I/O contention.

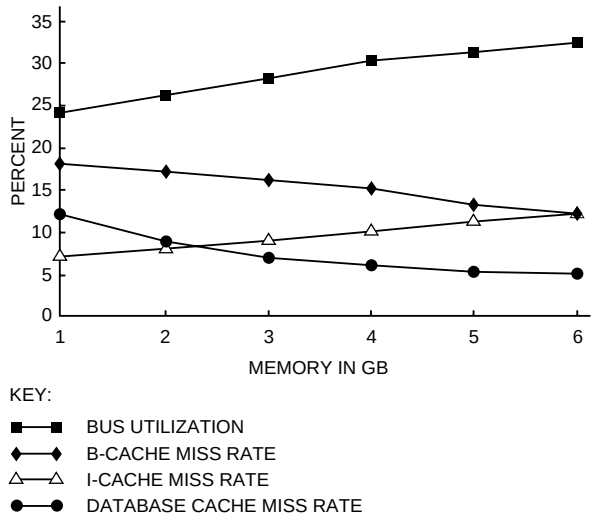

Figure 3
Cache Miss Rates and Bus Utilization

To answer the second question, we need to look at the AlphaServer 8400 system's hardware counters that measure instruction-cache (I-cache) miss rate, board-cache (B-cache) miss rate, and the bandwidth used on the multiprocessor bus. With an increase in throughput and memory size, the VLM system is spanning a larger data space, and the bus utilization increases from 24 percent to 32 percent. Intuitively, one might think this would result in less optimal instruction-and data-stream locality, thus increasing both miss rates. As shown in Figure 3, this proved true for instruction stream misses (I-cache miss rate) but not true for the data stream, as represented by the B-cache miss rate. The instruction stream rarely results in B-cache misses, so B-cache misses can be attributed primarily to the data stream.

Performance analysis requires careful examination of the throughput of the system under test. The apparent paradox just related can be resolved if we normalize the statistics to the throughput achieved. Figure 4 shows that the instruction-cache misses per transaction declined slightly as the memory size was increased from 1 GB to 6 GB—and as transaction throughput doubled. Furthermore, the B-cache works substantially better with more memory: misses declined by 2X on a per-transaction basis. Why is this so?

Analysis of the system monitor data for each run indicates that bringing the data into memory helped reduce the I/O per second by 30 percent. If the transaction is forced to wait for I/O operations, it is done asynchronously, and the database causes some other thread to begin executing. Without VLM, 12 percent of transactions miss the database cache and thus stall for I/O activity. With VLM, only 5 percent of the transactions miss the database cache, and the time to perform each transaction is greatly reduced. Thus each thread or process has a shorter transaction latency. The shorter latency contributes to a 15-percent reduction in system context switch rates. We attribute the measured improvement in hardware miss rates per transaction when using VLM to the improvement in context switching.

The performance counters on the Alpha microprocessor were used to collect the number of instructions issued and the number of cycles.[9] In Table 2, the relative instructions per transaction results are the ratios of instructions issued per second divided by the number of new-order transactions. (In TPC-C, each transaction has a different code path and instruction count; therefore the instructions per transaction amount is not the total number of new-order transactions.) The relative difference between instructions per transaction for 1 GB of database memory versus 6 GB of database memory is the measured effect of eliminating 30 percent of the I/O operations, satisfying more transactions from main memory, reducing context switches, and reducing lock contention.

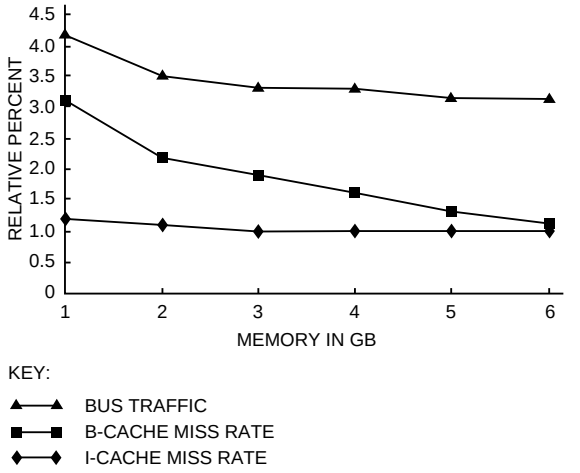

Figure 4
Normalized Cache Miss Rates and Bus Traffic

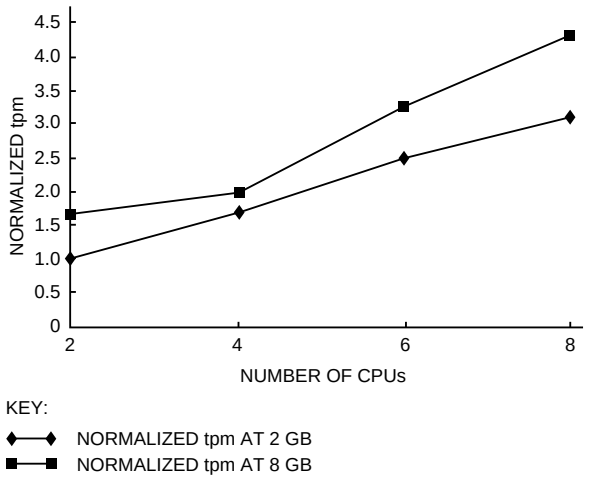

Figure 5
CPU Scaling versus Memory

Improved CPU Scaling—More Efficient Locking

A final benefit of using VLM is improved symmetric multiprocessing (SMP) scaling. Because the TPC-C workload has several transactions with high read content, having the data available in memory, rather than on disk, allows an SMP system to perform more efficiently. More requests can be serviced that are closer in cycles to the CPU. Data found in memory is less than a microsecond away, whereas data found on disk is on the order of milliseconds away.

We have shown how this situation improves the overall system throughput. In addition, it improves SMP scaling. Figure 5 shows the relative scaling between 2 CPUs and 8 CPUs with only 2 GB of system memory (1.5 GB of database cache) compared to the same configurations having 8 GB of system memory (6.6 GB of database cache).

We used the performance counters on the Alpha 21164 microprocessor to monitor the number of cycles spent on the memory barrier instruction.[9] Memory barriers are required for implementing mutual exclusion in the Alpha processor. They are used by all locking primitives in the database and the operating system. With VLM at 8 GB of memory, we measured a 20-percent decline in time spent in the memory barrier instruction. Larger memory implied less contention for critical disk and I/O channel resources and thus less time in the memory barrier instruction.

Conclusions

Open system database vendors are expanding into mainframe markets as open systems acquire greater processing power, larger I/O subsystems, and the ability to deliver higher throughput at reasonable response times. To this end, Digital's AlphaServer 8400 5/350 system using VLM database technology has demonstrated substantial gains in commercial performance when compared to systems without the capability to use VLM. The use of up to 8 GB of memory helps increase system throughput by a factor of 2, even for databases that span 50 GB to 100 GB in size.

The Digital AlphaServer 8400 5/350 system combined with the Digital UNIX operating system to address greater than 2 GB of memory has made possible improved TPC-C results from several vendors. In this paper, we have shown how VLM

- Increased the throughput by a factor of nearly 2
- Increased the database-cache hit ratios from 88 percent to 95 percent

By using monitor tools designed for the Alpha platform, we have measured the effect of VLM in issuing fewer instructions per transaction on the Alpha 21164 microprocessor. When transactions are satisfied by data that is already in memory, the CPU has fewer hardware cache misses, fewer memory barrier processor stalls, faster locking, and better SMP scaling.

Future Digital AlphaServer systems that will be capable of using more physical memory will be able to further exploit VLM database technology. The results of industry-standard benchmarks such as TPC-C, which force problem sizes to grow with increased throughput, will continue to demonstrate the realistic value of state-of-the-art computer architectures.

Acknowledgments

Many people from a variety of groups throughout Digital helped tune and deliver the TPC-C results. In particular, we would like to thank Lee Allison, Roger Deschenes, Joe McFadden, Bhagyam Moses, and Cheryl O'Neill (CSD Performance Group); Jim Woodward (Digital UNIX Group); Sean Reilly, Simon Steely, Doug Williams, and Zarka Cvetanovic (Server

Engineering Group); Mark Davis and Rich Grove (Compilers Group); Peter Yakutis (I/O Performance Group); and Don Harbert and Pauline Nist (project sponsors).

References and Notes

1. D. Fenwick, D. Foley, W. Gist, S. VanDoren, and D. Wissell, "The AlphaServer 8000 Series: High-end Server Platform Development," *Digital Technical Journal,* vol. 7, no. 1 (1995): 43–65.
2. At the time this paper was written, 2 GB was the largest size module. Digital has announced that a 4-GB option will be available in January 1997.
3. L. Wilson, C. Neth, and M. Rickabaugh, "Delivering Binary Object Modification Tools for Program Analysis and Optimization," *Digital Technical Journal,* vol. 8, no. 1 (1996): 18–31.
4. Transaction Processing Performance Council, *TPC Benchmark C Standard Specification, Revision 3.0,* February 1995.
5. W. Kohler, A. Shah, and R. Raab, *Overview of TPC Benchmark: The Order Entry Benchmark,* Technical Report (Transaction Processing Performance Council, December 1991).
6. More information about the TPC-C benchmark may be obtained from the TPC World Wide Web site, http://www.tpc.org.
7. J. Shakshober and B. Waters, "Improving Database Performance on Digital Alpha 21064 with OM and Spinlock Optimizations" (CSD Performance Group, Digital Equipment Corporation, July 1995).
8. R. Sites, ed., *Alpha Architecture Reference Manual* (Burlington, Mass.: Digital Press, 1992).
9. B. Wibecan, *Guide to IPROBE* (Digital Equipment Corporation, December 1994).

Biographies

Tareef S. Kawaf
Tareef Kawaf received a B.S. in computer science (magna cum laude) from the University of Massachusetts at Amherst. He is a member of Phi Beta Kappa. Tareef joined Digital in 1994 to work on performance enhancements and tuning of high-end systems and is a senior software engineer in the CSD Performance Group. He worked on attaining the world record-setting TPC-C results on the AlphaServer 8400 5/300 and 5/350 systems and the four-node AlphaServer 8400 5/350 cluster system running a state-of-the-art database application. Tareef has received two excellence awards from Digital for his work in TPC-C performance measurement on the AlphaServer 8000 series.

D. John Shakshober
John Shakshober is the technical director of the CSD Performance Group. The Computer Systems Division Performance Group evaluates Digital's systems against industry-standard benchmarks such as those of the Transaction Processing Performance Council (TPC) and the Standard Performance Evaluation Corporation (SPEC). In this function, John has been responsible for integrating Digital's state-of-the-art software technologies with Digital's Alpha-based products since their introduction in 1992. Prior to joining the CSD Performance Group, John modeled the performance of the 21064 and 21164 Alpha 64-bit VLSI microprocessors and was a member of the VAX 6000 Hardware Group. He joined Digital in 1984 after receiving a B.S. in computer engineering from the Rochester Institute of Technology. John also received an M.S. in electrical engineering from Cornell University in 1988.

David C. Stanley
Dave Stanley joined Digital in 1984. He is a principal software engineer in the CSD Performance Group and was the project leader for the TruCluster system that achieved a world-record result for the TPC-C benchmark. Dave has also led several TPC-C audits on the AlphaServer 8000 series running a state-of-the-art database application. He is a secondary representative at the TPC General Council and a member of the TPC-C Maintenance Subcommittee. Prior to these responsibilities, he was a microprocessor application engineer at Digital Semiconductor, where he ran competitive benchmarks on the MicroVAX II processor chip versus the Motorola 68020. Dave received a B.S.E.E. from the State University of New York at Buffalo (1981).