# Database Servers on Chip Multiprocessors: Limitations and Opportunities

Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju G. Mancheril,
Stavros Harizopoulos, Anastassia Ailamaki, Babak Falsafi

CMU-CS-06-153
September 2006

**School of Computer Science**
Carnegie Mellon University
Pittsburgh, PA 15213-3890

**Acknowledgements**: Acknowledgements to funding agencies and collaborators may appear on the first page at the bottom margin in justified, 9 pt Times or equivalent text.

**Keywords:** Database Servers, Chip Multiprocessors, Performance Characterization, Staged Database Systems, Cordoba, Fat Camp Cores, Lean Camp Cores, Operator Level Parallelism, Online Transaction Processing (OLTP), Decision Support Systems (DSS).

**Abstract**

With the advent of chip multiprocessors, high-performance data management software is an imminent technical and research challenge for the database community. In this paper we characterize the performance of a commercial database server on top of the emerging chip multiprocessor technologies. Using both simulations and experiments on real hardware we find that the major bottleneck of current software is data cache stalls, and briefly describe techniques to address the problem. Finally, we derive a list of features for future database designs and outline a preliminary design and implementation of an adaptable multi-core optimized database engine, called *Cordoba*.

# 1 Introduction

In the past decades, microprocessor designs primarily focused on techniques to exploit instruction-level parallelism through wide-issue out-of-order processors, and overlap both computation and memory accesses to improve single-thread performance across a variety of workloads. Unfortunately, because of the diminishing returns on exploiting instruction-level parallelism and the prohibitive levels of power dissipation in wide-issue processors, with the continued increase in chip integration levels, most vendors have begun to integrate multiple such processors into chip multiprocessor (CMP) to exploit both instruction-level and workload thread-level parallelism at the same time.

Much recent work in the database community focuses on characterizing commercial DBMS performance on modern wide-issue, out-of-order processors. These studies identify data cache stalls as a fundamental single-thread performance bottleneck for database workloads when running on modern hardware [2, 3, 20]. To address the data stall bottleneck, architects have recently designed CMPs that incorporate several multithreaded narrow-issue, in-order processors that can effectively hide data stalls across threads. For a fixed area budget, chip multiprocessors can integrate a much larger number of in-order, narrow-issue processors than out-of-order, wide-issue ones, at the cost of single-thread performance. Therefore, when executing database workloads in which threads are not abundant, performance may dramatically suffer.

Unfortunately, database workload performance has not been thoroughly characterized across such diverse spectrum of current and future chip designs. A recent study [9] focuses on throughput as the primary performance metric to compare server workload performance across chip multiprocessors with varying processor granularity, but has stopped short of a detailed performance characterization and breakdown of where time is spent during execution.

In this paper, we present a taxonomy of processor designs and DBMS workloads to distinguish the various combinations of workload and system configuration. We divide chip multiprocessors into two “camps”: (1) the fat camp employs conventional wide-issue out-of-order or simultaneously multithreaded processors (e.g., Intel Core Duo [1], IBM Power 5 [15]), and (2) the lean camp employs multithreaded narrow-issue in-order processors (e.g., Sun Niagara [16]). We further divide database applications into (1) saturated workloads, in which idle processors always find an available thread to run, and (2) unsaturated workloads, in which processors may not always find threads to run, thereby exposing memory latencies from executing threads. We characterize the performance of each database workload and system configuration pair within the taxonomy. For this study we use a cycle-accurate full-system chip multiprocessor simulator called Flexus [10], and run OLTP (TPC-C) and DSS (TPC-H) benchmarks on a commercial DBMS. Our results indicate that:

- Conventional DBMS hide stalls only in one out of four combinations of chip designs and workloads. Despite the chip-level parallelism, the fat camp still spends 46% – 64% of execution time on data cache stalls. The lean camp efficiently overlaps data stalls when executing saturated workloads, but unsaturated workloads perform up to 41% worse than the fat camp.

- To hide stall time when executing DBMS across the entire spectrum of workloads and systems, the software must improve both data reuse/locality and exhibit high thread-level parallelism across and within queries and transactions. Data locality helps eliminate stalls in both processor camps independently from workload type. Parallelism helps exploit the abundance of thread execution resources in the lean camp even when the workload is not saturated.

To our knowledge, this is the first study to provide an analytic taxonomy of the behavior of database workloads in the various emerging classes of chip multiprocessors. Through a series of simulations and experiments on real hardware we find that the behavior of database systems varies as a function of hardware and workload, and that conventional database systems fail to provide high performance in the whole spectrum of them. There is enough space for improvement and the presented taxonomy enables us to concentrate in each segment separately and identify the features a database system should support. The results help us outline a DBMS design that naturally provides locality as well as aggressive parallelism, given the underlying hardware and running workload.

The rest of the paper is organized as follows. Section 2 proposes a taxonomy of processor core technologies and workloads. Section 3 analyzes DBMS behavior as a function of hardware and workloads. Section 4 discusses techniques to reduce the impact of data stalls. Section 5 outlines a list of features a database server should incorporate to maximize performance in chip multiprocessor environments. Section 6 presents *Cordoba*, an adaptable database design optimized for multi-core chips. Section 7 discusses possible extensions in hardware. Finally, Section 8 presents related work, and Section 9 concludes.

# 2 Core Camps and Workloads

In this section we propose a taxonomy of processor core technologies and workloads and analyze their characteristics.

## 2.1 Fat Camp vs. Lean Camp

Hardware vendors adopt two distinct approaches to processor (*core*) design. One approach targets maximum single-thread performance through sophisticated out-of-order cores and aggressive speculation (fat-camp or FC cores). Representative processors from this camp include Intel Core Duo [1] and IBM Power5 [15]. The other approach favors much simpler designs that support many thread contexts in hardware (lean-camp or LC cores). Such cores attempt to overlap stalls in a given thread with useful computation by other threads. Sun Niagara [16] and Compaq Piranha [4] fall into this camp. Table 1 summarizes the typical characteristics of each technology camp.

| Core Technology | Fat Camp (FC) | Lean Camp (LC) |
|---|---|---|
| Issue Width | Wide (4+) | 1 or 2 |
| Execution Order | Out-of-order | In-order |
| Pipeline Size | Long (14+ stages) | Short (5-6 stages) |
| HW Threads per Core | Few (1-2) | Many (4+) |
| Core size | Large (3xLCsize) | Small (LCsize) |

**Table 1: Processor Technology Camp Characteristics**

Because LC cores are heavily multithreaded, we expect them to efficiently hide stalls due to misses and provide high and scalable throughput when there is enough parallelism in the workload. Thus, we expect them to outperform FC cores when supported by the same memory hierarchy and run the same workload (which implies similar instruction and data cache miss rates), even with much lower single-thread performance than that of a FC core. In addition to the performance differences when comparing single cores, an LC CMP can typically fit three times more cores in one chip than an FC CMP, resulting in roughly an order of magnitude more hardware contexts in the same space. In this paper, we study the increasing performance differences between fat and lean camps when running database workloads on CMPs.

## 2.2 Unsaturated vs. Saturated Workloads

Database performance varies with the number of requests serviced. For simplicity we assume that the database system assigns one worker thread per query (or transaction) it receives. We observe that the performance of a database application for a given hardware platform falls within one of two regions, and characterize the workload as unsaturated or saturated.

A workload is unsaturated when processors do not always find threads to run. As the number of concurrent requests increases, performance improves by utilizing otherwise idle hardware contexts. Figure 1 illustrates throughput as a function of the number of concurrent requests in the system when running TPC-H queries on a commercial DBMS on a real 4-core Power5 (FC) machine. The workload is unsaturated initially, but increasing the number of concurrent requests eventually results in a saturated workload, where there are always available threads for idle processors to run. Peak performance occurs at the beginning of the saturated region; increasing the number of concurrent requests too far overwhelms the hardware, reducing the amount of useful work performed by the system and lowering performance.

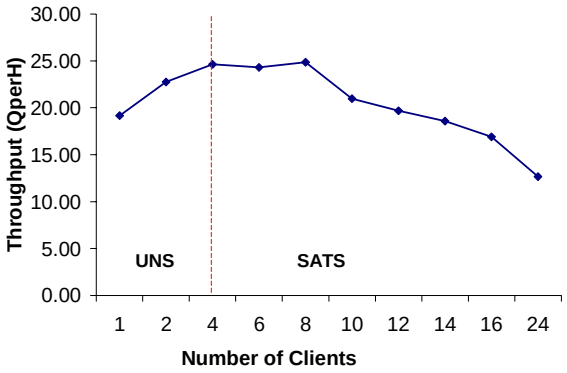


**Figure 1. Throughput of a commercial DBMS as a function of the number of concurrent clients on a Fat Camp (FC) CMP**

# 3 DBMS Performance on CMPs

In this section we study the performance of a commercial DBMS running both unsaturated and saturated workloads on FC and LC CMP systems.

## 3.1 Experimental Methodology

We use FLEXUS [10] to accurately simulate applications running on chip multiprocessors. We characterize the performance of database workloads on an LC CMP and an FC CMP. The LC CMP employs four 2-way superscalar in-order cores. The LC cores are 4-way multithreaded, for a total of 16 contexts on the LC CMP. The FC CMP employs 16 single-threaded 8-wide out-of-order cores. The two CMP designs have identical memory subsystems and a shared 26MB L2 cache.

We run OLTP (TPC-C) and DSS (TPC-H) workloads on a commercial DBMS. The saturated OLTP workload consists of 64 clients submitting transactions on a 100-warehouse database. The saturated DSS workload consists of 16 clients running four queries, each with random predicates. We select the queries as follows [22]: Queries 1, 6 and 13 are scan-dominated, and Query 16 is join-dominated. To achieve practical simulation times we run the queries on a 1GB database. Recent research has shown that varying the database size does not incur any microarchitectural behavior changes [22]. Unsaturated workloads use the above methodology running only a single client.

## 3.2 Workload Characterization

Figure 2 compares throughput, execution time breakdown and normalized data miss rates for all CMPs on a commercial DBMS running unsaturated and saturated DSS and OLTP workloads. Figure 2a presents the throughput of the LC CMP we use in our studies normalized to FC throughput. The FC CMP achieves up to 41% higher throughput than LC when running unsaturated (single-thread) workloads, corroborating prior results [20]. Even though FC exhibits higher single-thread performance than LC, the LC CMP achieves 70% higher throughput than its FC counterpart when running saturated workloads.

Figure 2b shows execution time breakdown for each camp and workload combination. Data stalls dominate execution time in three out of four cases. While FC spends 46% – 64% of execution time on data stalls, LC spends at most 13% of execution time on data stalls when running saturated workloads, while spending 76% – 80% of the time on useful computation. The multiple hardware contexts in LC efficiently overlap data stalls with useful computation, thereby allowing LC to significantly outperform its FC counterpart on saturated workloads, even though both camps suffer similar L2 data miss rates.

Figure 2c presents the L2 data miss rates of the LC CMP normalized to the FC CMP. The L2 data miss rates of both camps are identical when running saturated OLTP and unsaturated DSS. Even though the LC CMP on the saturated OLTP exhibits 36% more L2 data misses than FC, both are low (1.3% and 1.6% respectively). The L2 data miss rate when running saturated DSS is higher by 32% on FC due to misspeculated instructions, a side effect of the aggressive optimizations in FC cores (as opposed to the simple LC cores)

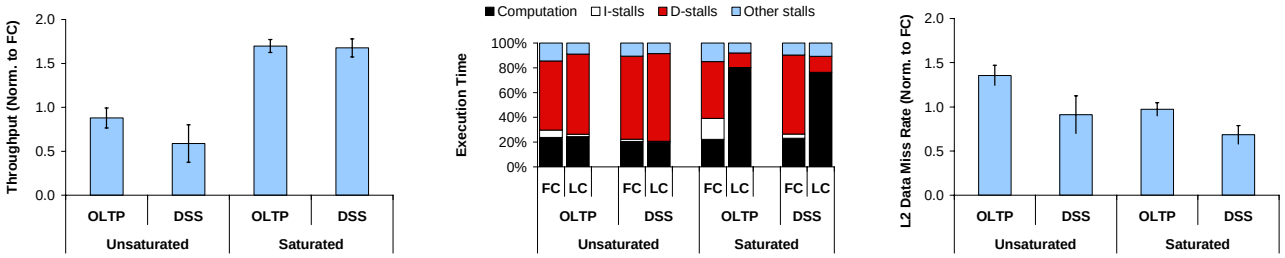


**Figure 2. (a) Normalized throughput, (b) Execution time breakdown, and (c) Normalized L2 data miss rates of LC CMP running OLTP (TPC-C) and DSS (TPC-H) workloads with 1 (Unsaturated) and 64 (Saturated) concurrent clients**

Despite prior work [3] showing that instruction stalls often dominate memory stalls when running database workloads, our CMP experiments indicate that data stalls dominate the memory access component of the execution time for all workload/camp combinations. Both camps employ instruction stream buffers [14], a simple hardware mechanism that automatically initiates prefetches to successive instruction cache lines following a miss. Our results corroborate prior research [20] that demonstrates instruction stream buffers efficiently reduce instruction stalls. Because of their simplicity, instruction stream buffers can be employed easily by the majority of chip multiprocessors thus we do not analyze further instruction cache performance.

We conclude that the abundance of threads in saturated workloads allows LC CMPs to hide data stalls efficiently. We expect, however, that the FC CMP will still claim a significant percentage in the market due to the low performance of LC CMPs on unsaturated workloads. Thus, database systems must be designed to perform well on both CMP camps.

# 4 Reducing Data Stalls

The previous section shows that data stalls are the dominant reason for underutilization and low throughput when database workloads run on CMP. Three high-level approaches can help alleviate data stalls: (1) Reducing the number of data accesses. (2) Reducing data access cost (latency), and (3) Overlapping (or hiding) data accesses with useful computation.

The number of data accesses depends on algorithm selection and compilation technology. This section briefly discusses the latter two techniques and their effect on CMPs of each camp (details are beyond the scope of this paper).

## 4.1 Reducing Access Cost (Latency)

The architecture community has developed several techniques that reduce the latency of data accesses. Most of theese techniques employ some sort of caching. Caches allow the processor to keep frequently-used data nearby to greatly reduce access costs. Modern processors employ extensive cache hierarchies where each level is larger and slower than the one above. Data misses at one level fall through to the level below, or to main memory, incurring higher miss latency than cache hits.

To reduce the absolute number of data misses at a given cache level we must either (a) increase the cache size, or (b) increase the spatial and temporal locality when accessing data.

### 4.1.1 Data Cache Size

To investigate the effect of data cache size when running commercial DBMS, we run saturated TPC-H and TPC-C workloads on an LC CMP. Figure 3 plots L2 user data miss rate as a function of the cache size. We observe that the rate quickly stabilizes below 5%. Despite the low miss rates of the 26MB simulated L2 cache, however, the data stall component remains high at 64% and 70% for FC and LC CMPs, respectively, indicating diminishing returns. As the L2 data miss rate is independent of the database size [22] increasing cache size alone is not sufficient to alleviate data stalls.

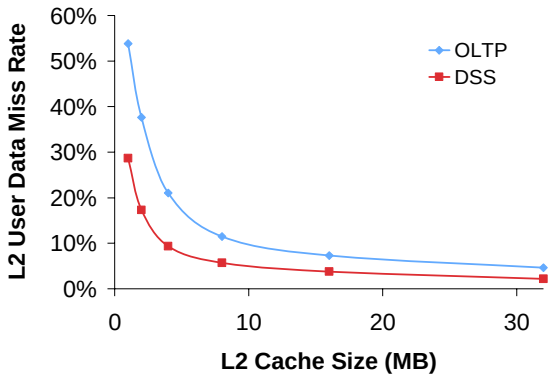


**Figure 3. L2 Data miss rate in TPC-H and TPC-C saturated workloads in LC CMP, as the cache size varies**

#### 4.1.2 Data Locality

The second technique for reducing the number of cache misses is to increase the data locality and reduce the reuse distances of cache lines. For instance, a database designer could (a) implement cache-conscious algorithms and execution, (b) restructure the data layout, and (c) improve locality through scheduling.

**Cache-conscious Algorithms and Execution**: A growing number of proposals for cache-conscious algorithms [23] and execution have arisen out of the database community. Their goal is to provide cache-friendly access patterns and data structures. For example, MonetDB/X100 [26] uses vectorized processing and column-wise storage to improve spatial and temporal data locality.

**Data Layout**: Another approach is to restructure the data layout in the disk and memory to reduce the number of cache misses. PAX [2] is a cache-friendly data disk placement algorithm. Similarly, [21] proposes a modified (read-only) B+ tree that improves cache behavior. Chilimbi [7] proposes the use of compilers to automatically produce cache-conscious structures.

**Scheduling**: The database engine can also manage worker threads explicitly to provide better cache performance. For example STEPS [12] is a technique that achieves high instruction locality by carefully scheduling threads in transactional workloads. Similar techniques can be employed to increase data locality. For example, binding producer/consumer threads together on the same core can improve the data locality. More aggressively, when producer and consumer threads are bound on the same core, the producer can yield to the consumer whenever it fills L1D completely, allowing the consumer to use the data before it is evicted from the L1D. This would especially benefit FC cores because they may not provide enough hardware contexts to run the producer and consumer simultaneously.

### 4.2 Hiding Data Accesses

To hide the stalls caused by data cache misses we must increase parallelism. There are three flavors of parallelism: (a) Instruction Level, (b) Memory Level, and (c) Thread Level.

#### 4.2.1 Instruction Level Parallelism (ILP)

Instruction level parallelism is a measure of how many operations of a specific computation can be performed simultaneously without violating dataflow dependencies. The small instruction issue width of LC CMPs prevents them from exploiting potential ILP. FC cores have the means to exploit ILP but still fail to hide data stalls because database workloads exhibit limited ILP [20].

#### 4.2.2 Memory Level Parallelism (MLP)

Memory-level parallelism hides memory latency by issuing multiple memory requests in parallel. Techniques that exploit MLP include prefetching [6] and/or predicting access patterns of the application [24]. Some techniques, such as run-ahead execution [19], use additional helper threads to perform prefetching, often speculatively.

#### 4.2.3 Thread Level Parallelism (TLP)

Database applications inherently provide thread-level parallelism. A workload consisting of multiple queries or transactions causes the creation of a large number of worker threads to serve the requests. That is, database workloads naturally exhibit significant inter-transaction and inter-query parallelism.

Though database workloads are inherently parallel there are ways to increase the thread-level parallelism further. Some methods include partitioning, pipelining, and intra-query parallelism.

**Partitioning**: This approach distributes data into independent pieces that can be processed in parallel. Partitioning is usually applied in the table level in databases. Tables can be partitioned either vertically or horizontally, depending on the expected access patterns (i.e. queries). A query that accesses a partitioned table may be divided into a number of independent sub-queries operating on part of the partitioned data. This approach is very efficient, especially when the database server is tuned correctly and the workload does not exhibit fluctuations. Unfortunately partitioning is static and there always exist queries that a chosen partitioning scheme cannot handle efficiently.

**Operator Level Parallelism (Pipelining)**: A recent proposal suggests an operator-centric execution of the database operations [11, 13]. Under this design philosophy each database query is decomposed by operation each of which is assigned to a different worker thread. The worker threads communicate with each other through pipelining. This is a new parallelism paradigm in databases called Operator-level parallelism (or OLP). OLP requires redesign of the database server, it produces many additional threads and exposes opportunities for work sharing.

**Intra-Xaction/Query Parallelism**: This category contains all techniques that try to exploit parallelism opportunities within a single transaction or query. For example, in [8] the loops inside TPC-C transactions execute speculatively and in parallel, increasing the parallelism and throughput.

FC CMPs have limited means to exploit TLP. The large size of FC cores limits the number that can fit in one chip. Additionally, their complexity limits the number of hardware contexts that one core can support. On the other hand, LC cores are simple enough to allow the implementation of multiple hardware contexts. They are also small enough to fit many cores in one chip. In both cases, power is a constraint, but many more LC hardware contexts can fit on one chip for a given power and area budget.

# 5 High Performance DBMS on CMP

DBMS workloads running on CMPs maneuver through the tradeoff between parallelism and locality. For example, consider a producer-consumer thread pair participating in a CMP workload. If the workload is saturated, peak performance may be obtained when the producer-consumer pair is bound on the same core, thus increasing locality through cache sharing. However, if the workload is unsaturated, maximizing parallelism at the expense of data locality by running each thread on a different core and allowing data to pipeline naturally from the producer to the consumer may yield higher performance overall. An adaptive database manager can increase the performance of the workload through policies that exploit the parallelism/locality tradeoff.

Conventional database systems fail to achieve high performance in the entire spectrum of chip multiprocessor technologies and workloads because (1) their execution time is typically dominated by data stalls, (2) they do not exhibit sufficiently high thread-level parallelism, and (3) they cannot adapt to different hardware configurations and workloads. To achieve peak performance a database system should (1) preserve data locality and provide parallelism, (2) provide the flexibility to favor parallelism or locality given the runtime workload, and (3) is conscious of the underlying hardware characteristics.

# 6 Cordoba: an Adaptive DB Design

In this section we briefly present *Cordoba*, a prototype adaptive database engine designed for CMPs. *Cordoba* primarily focuses on the aforementioned challenges. We outline the prototype implementation and show preliminary performance results.

## 6.1 Design of Cordoba

*Cordoba* is a database engine that follows the principles of staged database systems [11, 13]. In its operator-centric architecture, each conventional database operator becomes an independent component (called *stage*) with its own pool of worker threads that serve requests from a stage-local queue. Every incoming query is decomposed into a number of smaller requests (called *packets*) that can execute in parallel. The packets are routed to the queue of their corresponding operator, and communicate with the other packets of the same query through intermediate buffers that reside in shared memory.

Central to *Cordoba* are two components: (1) The Core Wizard (CW) and (2) The Micro-Request Dispatcher (MRD).

**The Core Wizard (CW)**: The Core Wizard maintains runtime information about the system. It uses a Stage Matrix that keeps track of the binding of packets to cores, as well as other useful information (e.g., number of client connections). CW uses profilers to maintain a complete overview of the running workload and hardware utilization.

**The Micro-Request Dispatcher (MRD)**: The Dispatcher performs look-ups to the Stage Matrix, decides which routing policy is the most appropriate given the running workload and hardware configuration, and routes the incoming requests accordingly. The MRD has the flexibility to favor (a) locality (by routing all the requests that operate on the same resources to the same core on LC, or additionally forcing thread throttling on FC) or (b) pipelined parallelism (by routing all the requests that compose a particular query or transaction to different cores and pipelining the intermediate results).

To summarize, *Cordoba* (1) aggressively tries to maximize data locality, (2) decomposes requests for work into simpler, independent tasks that can execute in parallel, and (3) consults the running workload status and the underlying architecture to route worker threads accordingly.

### 6.2 Implementation of Cordoba

The current *Cordoba* prototype is a CMP implementation of a staged database engine. It is written in C++ and uses BerkeleyDB as its storage manager. The Stage Matrix contains information about the active packets and the core they are bound to. We are in the phase of implementing and evaluating a number of routing policies for the MRD. The policies belong to two categories, those that consult the runtime information provided by the CW, and those that are static.

In the implementation of *Cordoba* we used a number of optimizations for increased performance. As an example, the communication between the various packets of a query is done through intermediate buffers of pages of tuples, rather than simple tuple buffers. The data layout is cache-friendly and the access patterns are predictable.

### 6.3 Experimental Results

*Cordoba* is still a work in progress. However, preliminary experimental results show that *Cordoba* provides high performance, and can adapt to the workload and hardware configuration. Figure 5 presents the throughput with varying number of concurrent clients for various dispatching policies. The experimental platform consists of a memory-resident TPC-H database on a dual-processor Power5 SMT server with a total of 8 hardware contexts. The dispatching policies are as follows: *RR* assigns packets to available cores in a round-robin fashion. *RR-MOD* routes requests of the same query in a round-robin fashion to the cores that belong to the same chip. *QUERY* routes all packets of the same query to the same core. Finally, *Cordoba* uses runtime information and switches between RR and QUERY when the number of concurrent clients exceeds the number of available hardware contexts. The ability of *Cordoba* to adapt to changing workloads allows it to outperform the RR variants of dispatching policies by up to 33%, while it performs similarly to the *QUERY* dispatching policy.

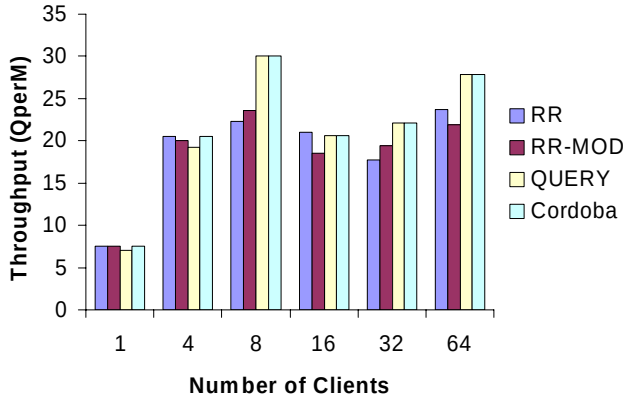


**Figure 5. Throughput of Cordoba on a FC CMP with increasing concurrent clients**

## 7 Hardware Support

This section presents possible extensions in hardware that may enhance the performance of database servers on CMPs.

**Streaming:** Hardware support for streaming data between cores can increase performance by hiding the communication latency. However, the software must know a priori what data to stream, to which consumer, and when streaming should start. *Cordoba* maintains explicit knowledge of producer/consumer pairs and the cores they run on, what data they will be sharing (intermediate results generated by the producer), and when will the consumer start consuming.

**Queuing operations**: A large fraction of the execution time in *Cordoba* is spent in queuing operations, such as monitoring a stage queue and synchronizing the access to it. Hardware support for queuing operations may minimize the impact of queuing operation on execution time, further increasing performance.

## 8 Related Work

Barroso et al. [5] conduct a performance study of OLTP and DSS workloads on a distributed shared memory multiprocessor and conclude that instruction and data locality on OLTP can be captured effectively only by large caches. Our study shows that data stalls dominate the execution time of modern chip multiprocessors even with very large caches. Barroso et al. [4] compare the performance of the Piranha chip multiprocessor against a single out-of-order processor with similar resources, but do not consider FC core designs for the Piranha chip.

Ranganathan et al. [20] examine the performance of database workloads on shared-memory multiprocessors and identify simple optimizations that improve performance when employed by aggressive out-of-order processors. However this study does not consider lean cores, which outperform their aggressive out-of-order counterparts on saturated workloads despite their low single-thread performance. Lo et al. [18] study the performance of database workloads on simultaneous multithreaded (SMT) processors and show that aggressive wide-issue out-of-order SMT processors can outperform their single-threaded out-of-order counterparts. However, wide-issue out-of-order processors with an abundance of hardware contexts are complex designs, and their area and power overhead render them unsuitable for CMP designs that target database workloads. In our study we show that even simple in-order multithreaded processors can outperform aggressive out-of-order ones when the database workload exhibits significant thread-level parallelism.

Staged computation is an emerging proposal for software server design. *Cordoba* follows the principles of staged database servers, as described in [11]. Larus et al. [17] propose a staged server programming paradigm that divides computation into stages and schedules requests within each stage to improve the cache performance of a Web server and a Publish-Subscribe server. Welsh et al. [25] propose a staged event-driven architecture for internet services that prevents resource over-commitment when demand exceeds the server's capacity.

## 9 Conclusions

Chip multiprocessors will be the dominant hardware technology for future database servers. The database community is not ready for this shift in hardware design. In this study we present a taxonomy of processor designs and DBMS workloads to distinguish the various combinations of workload and system configuration. Based on this taxonomy we characterize the performance of a commercial database server and identify performance bottlenecks. Our results indicate that data stalls limit performance in memory-resident databases, and we briefly describe techniques to address them. We outline the requirements for a high-performance database server on chip multiprocessors and present *Cordoba*, a staged database engine optimized for multi-core chips.

## 10 Acknowledgments

We cordially thank Debabrata Dash for his feedback and technical support throughput this research effort.

## 11 References

[1] Intel Corporation, "Intel Core Duo Processor and Intel Core Solo Processor on 65nm Process Datasheet", June 2006.

[2] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis. "Weaving Relations for Cache Performance". In Proceedings of 27th International Conference on Very Large Data Bases (VLDB), Rome, Italy, 2001.

[3] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. "DBMSs on a Modern Processor: Where Does Time Go?" In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB), Edinburgh, Scotland, 1999.

[4] L. A. Barroso, et al. "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing". In Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA), Vancouver, Canada, 2000.

[5] L. A. Barroso, K. Gharachorloo, and E. Bugnion "Memory System Characterization of Commercial Workloads". In Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA), Barcelona, Spain, 1998.

[6] S. Chen, A. Ailamaki, P. B. Gibbons, and T. C. Mowry. "Improving Hash Join Performance Through Prefetching". In Proceedings of the 20th International Conference on Data Engineering (ICDE), 2004.

[7] T. M. Chilimbi, M. Hill, and J. R. Larus. "Cache-Conscious Structure Layout". In Proceedings of SIGPLAN Conference on Programming Language Language Design and Implementation (PLDI), 1999.

[8] B. Colohan, A. Ailamaki, J. G. Steffan, and T. C. Mowry. "Optimistic Intra-Transaction Parallelism on Chip Multiprocessors". In Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway, 2005.

[9] J. D. Davis, J. Laudon and K. Olukotun, "Maximizing CMP Throughput with Mediocre Cores". In Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques (PACT), 2005

[10] N. Hardavellas, et al. "SIMFLEX: A Fast, Accurate, Flexible Full-System Simulation Framework for Performance Evaluation of Server Architecture". ACM SIGMETRICS Performance Evaluation Review, March 2004.

[11] S. Harizopoulos, and A. Ailamaki. "A Case for Staged Database Systems". In Proceedings of the 1st International Conference on Innovative Data Systems Research (CIDR), Asilomar, California, 2003.

[12] S. Harizopoulos, and A. Ailamaki. "STEPS towards cache-resident transaction processing". In Proceedings of the 30th International Conference on Very Large Data Bases (VLDB), Toronto, Canada, 2004.

[13] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki. "QPipe: A Simultaneously Pipelined Relational Query Engine". In Proceedings of the 24th ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, 2005.

[14] N. P. Jouppi. "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers". In Proceedings of the 17th Annual International Symposium on Computer Architecture (ISCA), Seattle, Washington, 1998.

[15] R. Kalla, B. Sinharoy, and J. Tendler. "IBM Power5 chip: A dual-core multithreaded processor". In IEEE MICRO, 2004.

[16] P. Kongetira, K. Aingaran, and K. Olukotun. "Niagara: A 32-Way Multithreaded SPARC Processor". In IEEE MICRO, 2005.

[17] J. R. Larus, and M. Parkes. "Using Cohort Scheduling to Enhance Server Performance". In Proceedings of the General Track: 2002 USENIX Annual Technical Conference, 2002.

[18] J. Lo, et al. "An Analysis of Database Workload Performance on Simultaneous Multithreaded Processors". In Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA), Barcelona, Spain, 1998.

[19] O. Mutlu, J. Stark, C. Wilkerson, and Y. N. Patt. "Run Ahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors". In Proceedings of the 9th International Symposium on High-Performance Computer Architecture (HPCA), 2003.

[20] P. Ranganathan, K. Gharachorloo, S. V. Adve, and L. A. Barroso. "Performance of Database Workloads on Shared-Memory Systems with Out-of-Order Processors". In Proceedings of the 8th International Conference of Architectural Support for Programming Languages and Operating Systems (ASPLOS), San Jose, California, 1998.

[21] J. Rao, and K. A. Ross. "Making B+-Trees Cache Conscious in Main Memory. In Proceedings of the 19th ACM SIGMOD International Conference on Management of Data, Dallas, Texas, 2000.

[22] M. Shao, A. Ailamaki and B. Falsafi. "DBmbench: Fast and Accurate Database Workload Representation on Modern Microarchitecture" In Proceedings of the IBM Center for Advanced Studies Conference, 2005.

[23] A. Shatdal, C. Kant, and J. F. Naughton. “Cache Conscious Algorithms for Relational Query Processing”. In Proceedings of the 20th International Conference on Very Large Data Bases (VLDB), Santiago de Chile, Chile, 1994.

[24] S. Somogyi, et al. “Spatial Memory Streaming”. In Proceedings of the 33rd Annual International Symposium on Computer Architecture (ISCA), Boston, Massachusetts, 2006.

[25] M. Welsh, D. Culler, and E. Brewer. “SEDA: An Architecture for Well-Conditioned, Scalable Internet Services”. In Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), Alberta, Canada, 2001.

[26] M. Zukowski, P. Boncz, N. Nes, and S. Heman. “MonetDB/X100 – A DBMS in the CPU Cache”. In IEEE Data Engineering Bulletin, June 2005.