FIRST WORKSHOP ON COMPUTER ARCHITECTURE EVALUATION USING COMMERCIAL WORKLOADS

Las Vegas, Sunday Feb. 1, 1998

FINAL PROGRAM

8:00 am - 8:15 am
Registration

8:15 am - 8:25 am
Introductory Comments

8:25 am - 10:05 am
Session 1 : Workload Behavior

Commercial Applications on Shared-Memory Multiprocessors
Zheng Zhang and Sekhar Sarukkai
Hewlett-Packard Labs

Importance of Proper Configuration in Architectural Evaluations using Database Workloads
Kimberly Keeton and David A. Patterson
University of California - Berkeley

System Design Considerations for a Commercial Application Environment
Luiz Andre Barroso and Kourosh Gharachorloo
Western Research Laboratory
Digital Equipment Corporation

Exploiting Caches Under Database Workloads
Pedro Trancoso and Josep Torrellas
University of Illinois at Urbana Champaign

10:05 am - 10:25 am -- Coffee Break

10:25 am - 12:05 pm
Session 2: Tracing and Characterization

Optimizing UNIX for OLTP on CC-NUMA
Darrell Suggs
Data General Corporation

Tracing and Characterization of NT-based System Workloads
Jason Casmira, David Kaeli - Northeastern University
David Hunter - DEC Software Partners Engineering Group

Analysis of Commercial and Technical Workloads on AlphaServer Platforms
Zarka Cvetanovic
Digital Equipment Corporation

Characterizing TPC-D on a MIPS R10K Architecture
Qiang Cao, Pedro Trancoso, and Josep Torrellas
University of Illinois at Urbana Champaign

12:05 pm - 1:30 pm -- Lunch

1:30 pm - 3:10 pm
Session 3: Design Tradeoff Analysis

Characterization, Tracing, and Optimization of Commercial I/O Workloads
H. Huang, M. Teshome, J. Casmira and D.R. Kaeli
Northeastern University Computer Architecture Research Laboratory
Brian Garrett and William Zahavi
EMC Corporation

Trace-driven Performance Exploration of a PowerPC 601 OLTP Workload on Wide Superscalar Processors
J.H. Moreno, M. Moudgill, J.D. Wellman, P.Bose, L. Trevillyan
IBM T.J. Watson Research Center

Performance Analysis of Shadow Directory Prefetching for TPC-C
Dan Friendly - University of Michigan
Mark Charney - IBM Research

Evaluating Branch Prediction Methods for an S390 Processor using Traces from Commercial Application Workloads
Rolf B. Hilgendorf, IBM Entwicklung GmbH, Boeblingen, Germany
Gerald J. Heim, Wilhelm Schichard-Institut fuer Informatik, Universitaet Tuebingen, Tuebingen, Germany

3:10 pm - 3:30 pm -- Coffee Break

3:30 pm - 4:45 pm
Session 4: Simulation Environments

Multiprocessor Architecture Evaluation using Commercial Applications
Ashwini K. Nanda
IBM TJ Watson Research Center

CASE - A Computer System Simulation Environment for Commercial Workloads
Jim Nilsson, Fredrik Dahlgren, Magnus Karlsson, Peter Magnusson* and Per Stenstrom
Department of Computer Engineering, Chalmers University of Technology
*Swedish Institute of Computer Science

VPC and IADYN - Project Overviews
Rich Uhlig and Todd Austin
Intel Corporation

4:45 pm - 5:00 pm -- Coffee Break

5:00 pm - 6:00 pm
Panel Session

"Do academics require access to DBMS source code in order to do effective research in the area of computer architecture for commercial workloads?"
Participants:

Josep Torrellas, University of Illinois
Ken Dove, Sequent Computer Systems
Kourosh Gharachorloo, Digital Equipment Corporation
Darrell Suggs, Data General Corporation

ABSTRACTS:

Session 1 : Workload Behavior

Commercial Applications on Shared-Memory Multiprocessors
Zheng Zhang and Sekhar Sarukkai
Hewlett-Packard Labs

Many current studies often use scientific and engineer applications as benchmarks to study shared-memory multiprocessors. This, however, contrast sharply with the reality where most high-end systems only run commercial applications such as database engine and web servers. Despite the pressing need of understanding these applications, the progress has been relatively slow and little. This is largely due to the fact that these applications are often of prohibitive complexity and proprietary.

In this talk we will discuss some commercial applications such as database, web server and multi-tier server complex. We will discuss where and when they might differ greatly with popular scientific and engineering applications. We want to point out that, in spite of the many differences that we see, some of the architectural highlights have already been addressed. For example, commercial applications would benefit a great deal from better handling of coherent misses, migratory data and false sharing. Their synchronization mechanisms are also ready to take advantage of the relaxed consistency model. There are, however, some features that clearly distinguish this group of applications from the rest. For example, we find that there are no workingset knees often observed elsewhere; some of the applications such as web servers may not be of primary interest at all, since OS spends most of the execution time. The IO, at both network and storage front, can no longer be ignored. What's more, the strong interaction of IO with memory/cache hiearchies indicate that it will be near-sighted if either of the IO and memory subsystem is designed in isolation.

Importance of Proper Configuration in Architectural Evaluations using Database Workloads
Kimberly Keeton and David A. Patterson
University of California - Berkeley

Databases are complex systems, with many (e.g., O(10)) configuration knobs to turn in order to achieve reasonable performance. The Transaction Processing Council (TPC) benchmarks are more difficult to run than other commonly used benchmarks like SPEC, because of the complexity of the database server application. TPC benchmarks, such as TPC-C [1], provide a well-defined workload with quantitative criteria for scaling datasets. The inclusion of large datasets and their associated disk IO component also complicates the benchmarking effort. Finally, the networking component of the client-server benchmarks also adds complexity. Because of resource constraints or lack of understanding of the underlying software, researchers often improperly configure their benchmarking systems. They may underpopulate the disk

subsystem, use too little memory, scale down the data set so that it fits entirely into memory, etc. Our full presentation will survey the validity of system configurations described in the literature.

All of these departures from a well-balanced system have the potential to affect the system under test in adverse ways, creating anomalies that may be unwittingly and unintentionally observed as being correct. The danger is that computer and operating systems designers might include optimizations that make no sense for properly configured systems, or might miss opportunities for improvements by not testing the machine under real load.

In this presentation, we will discuss some of the hardware and software factors that can affect the validity ofa configuration. We will discuss several ways that systems can be poorly configured, what potential performance anomalies may result from these bad configurations, and how to detect the situation. Finally, we will provide some rules of thumb for configuring well-balanced systems for using database workloads as benchmarks.

1.0 Factors that Impact the Validity of Configuration

A number of hardware and software factors may impact the validity of a configuration for measuring architectural performance for database workloads. We assume that the number of processors will be chosen to match the size of memory and the I/O subsystem. These additional factors may be grouped into four categories: I/O system hardware factors, memory-related hardware factors, database software system configuration parameters, and benchmark configuration parameters, and are discussed in Table 1. These factors may impact several key architectural and operating system characteristics, such as the breakdown of user/system time, number of disk I/Os, and CPI.

Category	Example Factors
I/O system hardware	Number and speed of disks, number and speed of I/O controllers, speed of disk I/O (e.g., SCSI) bus, speed of processor I/O (e.g., PCI) bus, amount of caching provided by I/O controllers, data layout to avoid hot spots
Memory-related hardware	Amount of physical memory impacts size of buffer cache, and memory bandwidth delivered by system
Database software parameters	Database buffer pool size, buffer management strategy parameters (e.g., high and low water marks for background writes of dirty pages)
Benchmark configuration parameters	Database size (e.g., TPC-C's number of warehouses), number of simulated clients

TABLE 1. Factors Affecting Configuration Validity for Architectural Performance Studies

2.0 Detecting a Poorly Balanced Configuration

Fortunately, the TPC-C benchmark provides a quantitative measure for verifying that a system is reasonably configured. The benchmark specification requires that the ratio of transaction throughput, transactions per minute C (tpmC), to database scale factor, the number of warehouses, resides in the range from 9.0 to 12.7. tpmC:warehouses ratios outside this range indicate that the system isn't properly balanced, the data set isn't properly scaled, or both.

In Table 2, we present a comparison of selected architectural and operating system behavior for systems

with tpmC:warehouses ratios within the acceptable 9.0 to 12.7 range, below 9.0, and above 12.7. These measurements were performed using a commercial database running on a four-processor Pentium Pro-based SMP running Windows NT 4.0, with 512 KB L2 processor caches, 2 GB of main memory, and 90 database data disks.

Architectural Behavior	Properly Configured	Low Memory	Low Warehouses
tpmC:warehouse ratio	10.1	8.0	27.8
Relative transaction throughput	100%	79%	115%
User/system breakdown	78% / 21%	70% / 30%	83% / 16%
Disk reads/sec	1550	2460	1070
Disk writes/sec	1280	1235	1100
Database cycles per instruction (CPI)	2.91	3.25	2.67
Database computation CPI	0.96	0.96	0.95
Database resource stalls CPI	0.57	0.74	0.49
Database instruction-related stalls CPI	1.36	1.61	1.24

TABLE 2. Effects of Invalid Hardware/Software Configurations on Architectural and OS Behavior

The "low memory" configuration yields a tpmC:warehouses ratio lower than the acceptable range. There is a loss in throughput due to the additional disk I/O required to compensate for the insufficient memory buffer pool. The increased I/O request rate results in an increased percentage of time spent in the operating system. In addition, we observe an increased database CPI, due to an increase in the resource and instruction-related stalls. Low tpmC:warehouses ratios may also occur when disk I/O takes too long, for example, when there is contention for the I/O bus, the I/O controller, or disks in the system. Alternately, this behavior could be exhibited by an extremely small dataset where there is artificial contention in data access, or when the offered transaction request load is too small because the number of client request generators is too low.

The "low warehouses" configuration yields a higher than appropriate tpmC:warehouses ratio. The throughput for this configuration is higher than it should be, because there is a reduced I/O rate due to the artificially high degree of locality in the workload. In this case, the data set isn't big enough to fully exercise the I/O system, as prescribed by the benchmark specification. This reduction in I/O request rate results in a higher percentage of time spent at user level. In addition, because more of the dataset is resident in memory, we see a lower database CPI, due to a decrease in resource and instruction-related stalls. Higher than acceptable tpmC:warehouses ratios may also occur for entirely memory-resident datasets.

From these simple examples, it is clear that the configuration of the system can impact several key architectural and operating system characteristics, such as the breakdown of user and system time, the disk I/O rates, and the cycles per instruction. To ensure that computer and operating system designers optimize for the correct behavior, performance analysts need to measure properly configured systems. In the full talk, we will conclude by articulating several rules of thumb for finding a reasonable configuration.

3.0 References

[1] Jim Gray. The Benchmark Handbook for Database and Transaction Processing Systems. Morgan Kaufmann
Publishers, Inc., 2nd edition, 1993.

System Design Considerations for a Commercial Application Environment

Luiz Andre Barroso and Kourosh Gharachorloo
Western Research Laboratory
Digital Equipment Corporation

We have been studying numerous commercial workloads during the past two years, with a special focus on their memory system behavior. This talk describes our extensive experience with three important classes of workloads: online transaction processing (OLTP), decision support systems (DSS), and web index search. We use a popular commercial database engine for running our OLTP and DSS workloads, and the AltaVista search engine for our web index search workload.

Given the large scale of commercial workloads, scaling the size of these workloads is critical for enabling a broad range of monitoring and simulation studies. However, such scaling requires a deep knowledge of the workloads and is rather sensitive to the set of architectural issues under study. We briefly describe our experience in scaling these workloads to allow for manageable architectural studies without compromising representative memory system behavior.

Our memory system characterization effort is based on a large number of architectural experiments on Alpha multiprocessors, augmented with full system simulations (using our Alpha port of SimOS) to determine the impact of architectural trends. We have also studied the impact of processor architecture on the memory system behavior of these workloads using trace-driven simulations. Specifically, we discuss the effectiveness of dynamic scheduling and simultaneous multithreading for hiding memory latencies in such applications.

Overall, our studies have allowed us to identify and evaluate critical system design parameters that impact the performance of commercial workloads.

Exploiting Caches Under Database Workloads

Pedro Trancoso and Josep Torrellas
University of Illinois at Urbana Champaign

Since databases have traditionally been bottlenecked by I/O, the cost functions used to select what operations and algorithms to use are dominated by the number of I/O operations. As DRAM becomes denser and cheaper, however, machines come equipped with more memory, and more databases are able to keep a large fraction of their data permanently in memory. Furthermore, memory systems are typically organized in cache hierarchies. Exploiting these cache hierarchies well is crucial to high performance. In this talk, we discuss a range of software and hardware supports for databases to run efficiently in machines with caches. We design a model of cache misses for database algorithms, and use it to build new cost functions for a cache-oriented query optimizer. We also examine enhancing the cache use of database operations with software prefetching, blocking algorithms, data layout restructuring, and cache bypassing. The combination of all these optimizations has a very high impact.

Session 2: Tracing and Characterization

Optimizing UNIX for OLTP on CC-NUMA

Darrell Suggs
Data General Corporation.

Our product goals challenged us with the need to identify software scaling issues for commercial OLTP workloads in a UNIX environment for a CC-NUMA system under design. The lack of prototype hardware that even remotely approximated the new design required that all analysis be done in a modeling and simulation environment. Our goal was to using tracing and simulation to provide sufficient lead time to make the required software modifications. As part of the process, we evolved a technique for obtaining architecture independent address traces for OLTP (TPC-C) workloads. We then used these traces to drive a detailed hardware simulation model. Through this technique we identified software scaling issues, advised reasonable changes, and predicted the impact of the changes. As the product became real, we were able to verify our predictive techniques against actual behavior. The results indicated that our analysis techniques are quite sound.

In the presentation, we will discuss machine architecture specifics, general software scaling issues, trace techniques, and comparative results from simulation and actual execution.

Tracing and Characterization of NT-based System Workloads
Jason Casmira, David Kaeli - Northeastern University
David Hunter - DEC Software Partners Engineering Group

Commercial applications are beginning to rely more on the services and APIs provided by the hosting operating system. This is particularly true in web browsers and database oriented workloads. Current trends in computer architecture have been driven by the characteristics of benchmark programs (e.g., SPEC, SPLASH, Bytemark). We argue that results using these simple benchmarks may be misleading, especially since most of these benchmarks make little use of operating system services.

In an effort to address this deficiency, we have developed the PatchWrx Dynamic Execution Tracing Tool Suite. PatchWrx operates on a DEC Alpha NT platform and has the ability to capture the entire execution of commercial applications, including the NTOS kernel, DLLs and system services. Trace capture is performed with a slowdown factor of 2x-4x. This low overhead is only possible by using the PALcode interface provided on the Alpha architecture. With this ability to capture complete application behavior on a system, it is possible to more accurately project performance of commercial applications on future architectures.

We have been able to capture the workload characteristics of several applications run under Microsoft NT Workstation, hosted on a DEC AXP Alpha 21064 workstation. We have also compared these results to traces of the bytemark benchmark captured in the same environment. To date we have studied the characteristics of the BYTE magazine benchmark and have compared these characteristics to Microsoft Internet Explorer (IE) and Microsoft CD Player (MCD). The amount of operating system interaction in the BYTE benchmark is less than 1% of the total execution, while for the two commercial applications we encounter overheads of 23% for IE and 78% for CD. We have also observed as much as a 53.6% decrease in basic block size when considering an application running with operating system behavior included versus the application viewed as independent of any operating system activity. This will have a dramatic effect on a branch prediction design.

Not only does the basic block size differ, but there is a marked difference in the total number of load and store instructions executed on the system. We found the total number of loads and stores to increase by as much as 72% when considering the operating overhead present in the application, versus the execution of the application code alone. This suggests that we may need to rethink some of our assumptions about

memory reference characteristics and memory hierachy design.

Using the PatchWrx Dynamic Execution Tool Suite, we can not only address questions related to the performance of commercial workloads, but we can also identify the inherent differences between application and operating system execution. We are currently gathering traces on Oracle databases running TPC-D workloads. We will be reporting on the characteristics of these workloads and their interaction with the NT kernel in our presentation.

Analysis of Commercial and Technical Workloads on AlphaServer Platforms
Zarka Cvetanovic
Digital Equipment Corporation

This talk will identify major hardware components that have crucial impact on the performance of commercial workloads on AlphaServer platforms. We will contrast the characteristics and requirements of technical and commercial workloads. The technical workloads used include the set of SPEC95, SPEC95-parallel, Linpack, and NAS Parallel (decomposed). The commercial workloads include: TPC-C, SPECweb96, and Laddis. The AlphaServer platforms evaluated include: mid-range server AlphaServer 4100 (Rawhide), and high-end server AlphaServer 8400 (TurboLaser).

We will contrast performance and SMP scaling of parallel/multistream technical and commercial workloads. We will analyze single, dual, triple, and quad issuing time on AlphaServer platforms. We will compare issuing and stall time and identify major components that caused CPU stalls. We will analyze cache misses thru several levels of cache/memory hierarchy. Included will be data and analysis of traps. TB misses, branch mispredicts and their effect on performance of technical/commercial workloads. The analysis of bus traces with Read, Victim, and Shared traffic will be presented. We will include analysis of the performance effects of Memory Barriers, Locks, and Shared-Dirty data. A breakdown of the total stall time to different stall components will be presented. The evaluation of performance benefits from larger caches will be included. We will present guidelines for optimizing future hardware architectures and implementations that will allow for high performance in both technical and commercial applications.

We will conclude the presentation with the discussion of the effect of I/O subsystem efficiency on the commercial performance.

Characterizing TPC-D on a MIPS R10K Architecture
Qiang Cao, Pedro Trancoso, and Josep Torrellas
University of Illinois at Urbana Champaign

In this work, we use the MIPS R10K counters to measure statistics on the instruction execution and cache performance of the TPC-D benchmark queries running on an SGI Origin 200 machine. The database is memory resident. We classify the different queries from the TPC-D benchmark according to their absolute number of instruction and data misses and relative weight of instruction to data misses. This classification allows us to identify queries with similar behavior and therefore select the queries that are representative of the whole benchmark. For those representative queries we study the behavior of each of their operations. In addition we also study the impact of the access methods to the data, namely the use of index or sequential scan. Finally, we study the scalability of the benchmark by running different data set sizes.

The results show that the time lost to cache misses accounts for a significant portion of the query execution time. In addition, instruction misses have a high weight. Finally, additional index structures do not seem to help in improving the scan performance for some of the queries.

Session 3: Design Tradeoff Analysis

Characterization, Tracing, and Optimization of Commercial I/O Workloads

H. Huang, M. Teshome, J. Casmira and D.R. Kaeli
Northeastern University Computer Architecture Research Laboratory
Boston, MA

Brian Garrett and William Zahavi
EMC Corporation
Hopkington, MA

RAID systems have become the industry standard for providing efficient, fault-tolerant mass storage [1,2]. Many of these RAID systems provide large (multi-gigabyte), integrated cache storage. To provide scalable I/O storage access, improved storage system caching strategies must be provided. This research attempts to improve the efficiency of the disk array subsystem by characterizing the reference patterns generated by commercial workloads. Then new cache management algorithms are explored to improve the performance of these commercial systems. This work is in conjunction with researchers at EMC Corporation, the leading manufacturer of cached disk array subsystems.

Our research is targeted at storage intensive commercial applications such as On-Line Transaction Processing (OLTP). This paper presents our work on the capture, characterization, sythesis and use of these workloads, detailing new prefetching and organizational issues related to providing scalable storage performance. We also present a new hardware mechanism which limits the effects of errant prefetching called a Prefetch Buffer Filter (PBF) [3]. For OLTP workloads the PBF can increase the effective disk cache hit ratio by as much as 52%, while reducing the amount of disk traffic by 49%.

References:

[1] R.H. Katz, G.A. Gibson and D.A. Patterson,
``Disk System Architectures for High Performance Computing,"
Proc. of the IEEE, Vol. 77, pp. 1942-1958, Dec. 1989.

[2] P.M. Chen and E.K. Lee,
``Striping in a RAID Level 5 Disk Array,"
in Proc. of Sigmetrics '95, Vol. 23, No. 1, pp. 136-145, May 1995.

[3] J. Casmira and D.R. Kaeli, ``Modeling Cache Pollution,"
to appear in the Journal of Modeling and Simulation,
Vol. 19, No. 2, May 1998.

Trace-driven Performance Exploration of a PowerPC 601 OLTPWorkload on Wide Superscalar Processors

J.H. Moreno, M. Moudgill, J.D. Wellman, P.Bose, L. Trevillyan
IBM T.J. Watson Research Center

We describe an investigation of the potential performance of PowerPC-based wide superscalar processors on a standard on-line transaction processing (OLTP) benchmark, using a PowerPC 601 instruction and data reference trace containing 170 million instructions. In this study, we have explored instruction-level parallelism as a function of the policy for issuing instructions, the processor width, the size of the cache memory, and the branch predictor. We summarize the characteristics of our exploration approach, describe

the features of the processor model used in the exploration, the configurations explored, and the results from the experimentation. Results include the average cycles per instruction (CPI) obtained in each configuration, details on degradation factors that limit performance, and sensitivity to some selected microarchitecture features.

The simulation results validate common wisdom regarding the degrading effects of the memory subsystem on workloads such as the one considered, behavior encountered regardless of the width of the processor; the results also give insight into the utilization of the various resources in the processor. For the trace and processor organizations considered, the simulation data show that increasing the processor issue width to eight operations per cycle is advantageous whereas wider organizations provide diminishing performance improvement. For example, doubling the features of an out-of-order processor whose issue width is four by doubling the number of units and the various widths, while preserving the size of caches and prediction accuracy, produces about 20% overall performance improvement. Doubling also the size of the caches in the same configuration produces an additional 10% improvement. In addition, the simulation results show that wide-issue out-of-order superscalar processors are potentially capable of continuing delivering performance, albeit at the cost of complexity, thereby helping lay the grounds for the evaluation of the corresponding trade-offs in the design of such systems.

Performance Analysis of Shadow Directory Prefetching for TPC-C
Dan Friendly - University of Michigan
Mark Charney - IBM Research

It is a widely accepted attribute of commercial workloads that they place greater demands on the memory subsystem than the standard benchmark suites. In this work we take a close look at the impact servicing memory requests has upon performance. To mitigate the effects of the larger memory footprint of commercial code we suggest an aggressive prefetching technique for both instruction and data caches.

Shadow directory prefetching is a recently proposed hardware prefetching mechanism that attempts to maintain an accurate history of the pattern of level 1 cache misses. By mapping an L2 cache request to a previously requested line, shadow directory prefetching can often anticipate subsequent requests of the data. When a line is requested the L2 responds with both the demand miss and provides a number of prefetches by following the mappings in the shadow directory. In doing so it able to reduce the average latency of fetch requests to the memory subsystem.

In this work we evaluate the effectiveness of shadow directory prefetching on a Power-PC 604 trace of the Oracle TPC-C code. The prefetching scheme is assessed through a number of metrics -- CPI, coverage, accuracy, latency, bus traffic and bus utilization. Variations of the shadow directory prefetching algorithm are examined including the use of confirmation and altering the mapping scheme.

Our preliminary results show that the utilization of the shared L2 to L1s reply bus places a significant limitation on the effectiveness of the prefetching algorithm. As shadow directory prefetching is initiated at the L2 cache it becomes imperative that the L2 maintain inclusion information so that the prefetch engine can inhibit the sending of redundant prefetches. Furthermore, we have found that inhibiting the bus transfers of prefetches when demand requests are being processed increases the effectiveness of the prefetching. Using both techniques, shadow directory prefetching is shown to improve performance by close to 10%. This represents a significant portion of the performance lost to the effects of having a finite memory subsystem.

Evaluating Branch Prediction Methods for an S390 Processor using Traces from Commercial Application Workloads

Rolf B. Hilgendorf, IBM Entwicklung GmbH, Boeblingen, Germany
Gerald J. Heim, Wilhelm Schichard-Institut fuer Informatik, Universitaet Tuebingen, Tuebingen, Germany

For modern superscalar processors, branch prediction is a must. There has been significant progress in this field during recent years, but it is not so clear which if any of the currently advocated schemes is superior.

The quality of the branch prediction algorithms has been measured by using the SPECmark test suite. Different programs from this suite show different behavior with respect to branch prediction. So some often the average of the prediction rate for each member of the suit is taken while in other publications only selected programs are used to stress certain aspects.

For the IBM System 390 environment exists a set of traces representing different areas of commercial workloads. They include operating-system interactions, too. We used four of these traces to evaluate a big variety of branch prediction algorithms in order to give assistance for the design trade-offs to be made. The traces used differ significantly from traces extracted form the SPECmark test suit mostly in respect to the

- total number of instructions
- distribution of branches
- presence of task switching (including address-space switches)
- influence of interrupt service routines.

In order to evaluate the different branch algorithms, a cycle precise, high level model of a hypothetical superscalar S390 processor was constructed, which could use the traces as input. For most of the results discussed here only the I-fetch, I-decode and branch resolution units of this model were used. As for mispredicted paths, no information is available from the trace, dummy instruction sequences were generated and presented to the I-fetch unit.

One speciality of the S390 architecture is that for most of the branches target address calculation involves the use of values stored in general purpose registers. Therefore not only branch directions but target addresses have to be predicted. When performing prefetch-time prediction commonly a branch target buffer (BTB) is used to provide/predict the target address. All prediction methods evaluated are combined with such a BTB. Evaluating the needed size for the BTB suggested it to be significantly larger compared to existing designs sized based on SPECmark traces. Reasons are discussed based on existing trace differences. The method of priming large data structures prior to executing relative short traces is introduced, too.

Bound to a particular size for the BTB algorithms for determining the branch direction are examined and compared. These algorithms include Local branch history methods like saturation counter and 2 level adaptive predictors as well as Global History and Path History procedures. Finally combinations of some of this methods known as hybrid-predictors were tested. For all this methods design parameters were varied to find the trade-off between the hardware needed and the prediction quality achieved.

An other speciality of the S390 architecture, the absence of specific subroutine call and return instructions led into the investigation of hardware for self detecting of call return pairs. Their effect on the prediction rate and buffer-size trade-offs will be shown.

As mentioned above all methods use the BTB. Such a BTB is good if a certain branch has always the same target. This is not true for all the branches in the traces, about 5% of them have changing target addresses. Very recently an algorithm was proposed for treating this branches using a modification to the BTB approach. We could achieve some enhancements in prediction correctness using this new method in

combination with the other methods investigated.

Finally it can be said that combining several of the newly developed schemes are able to increase branch prediction correctness in a commercial environment. However it remains to be seen if the tremendous increase in hardware needed for their implementation can be justified.

Session 4: Simulation Environments

Multiprocessor Architecture Evaluation using Commercial Applications
Ashwini K. Nanda
IBM TJ Watson Research Center

In this talk I will discuss (1) the design of a new simulation environment called COMPASS, and (2) how COMPASS is used to evaluate the memory behavior of three important commercial applications, namely, TPCC (transaction processing), TPCD (decision support) and SPECWeb (web server) on multiprocessor systems.

Shared memory multiprocessors have become de facto choice as server platforms for commercial applications such as transaction processing, decision support and web servers. Therefore, it has become desirable to study the performance impact of these commercial applications on the shared memory server platforms in order to make the right design decisions. However, most of the contemporary simulators used to evaluate shared memory multiprocessors both in industry and academia are not suitable for running commercial parallel programs. As a result most architecture studies are confined to scientific applications such as the Stanford SPLASH2 benchmarks.

Scientific applications on shared memory machines usually spend very little time in the operating systems. Therefore, not simulating any OS activity does not result in any significant loss of accuracy for these applications. However, many commercial applications heavily depend on operating system services, and some of the applications spend a significant portion of their CPU time in the operating systems. This is because commercial applications tend to use sophisticated inter-process communication mechanisms and I/O functions that operating systems provide. For example, our profiling data show that on both Unix and Windows NT systems, Web Servers spend 70-85% of their CPU times on OS kernels. On-Line-Transaction-Processing (OLTP) applications such as TPCC and decision support applications such as TPCD spend about 20% of their time in the operating systems. These applications, moreover, generate a significant amount of I/O activity. Therefore, in order to study commercial application behavior with reasonable accuracy, one has to simulate the OS functions where these applications spend a significant amount of their execution time.

COMPASS (COMmercial PArallel Shared Memory Simulator) was developed keeping these requirements in mind. COMPASS uses the basic instrumentation and execution driven simulation mechanism in a PowerPC version of the Augmint simulator. We carefully designed mechanisms to simulate only important OS functions that affect the performance of our target applications, yet to support virtually all of the OS functions that are potentially used by these applications. One can use the COMPASS environment to study the behavior of future shared memory multiprocessor systems and optimize them for commercial applications, as well as to optimize these applications for future server platforms.

CASE - A Computer System Simulation Environment for Commercial Workloads
Jim Nilsson, Fredrik Dahlgren, Magnus Karlsson, Peter Magnusson* and Per Stenstrom
Department of Computer Engineering, Chalmers University of Technology

*Swedish Institute of Computer Science

INTRODUCTION

We recognize an increasing interest in identifying, and finding remedies for, performance bottlenecks for large commercial applications for future computer systems. This requires an experimental platform that is able to support the exploration of hardware/software interaction that includes system software as well as the I/O system. Furthermore, such a platform should be able to take new architectural designs into consideration as well as support the adaptation of software to take full advantage of the target system. The CASE simulation environment is such a platform.

CASE is built around SimICS -- a program driven Sparc V8 instruction set simulator. Together with precise simulated models of hardware devices that comprise the Sun4m architecture, and the ability to accurately model an arbitrary memory hierarchy, we posess a simulator platform that is able to boot and run unmodified commercial operating systems, such as Solaris. This simulator provides the means for examining the underlying principles of contemporary and future multiprocessor computer architectures, and their interaction with operating systems and applications.

In this abstract we present the CASE platform, and its underlying principles of operation, as well as a representative case study of the caching behavior of a database application, including impact from system overhead.

SIMULATOR PLATFORM

The CASE platform consists of three components: the SimICS instruction-set simulator developed at SICS, the Sun4m kernel architecture simulator developed at SICS and Chalmers, and a set of memory system simulators developed at Chalmers. SimICS supports multiple Sparc V8 processors, multiple physical address spaces, and system-level code.

SimICS follows in the tradition of a family of fast instruction set simulators that make use of threaded code and/or native code generation. SimICS today runs approximately 25 times slower than the host platform when running SPECint95 programs and gathering only a minimum of statistics, and approximately 80 times slower when gathering detailed information, including TLB misses, data cache and instruction cache misses, and executed instructions, all at the granularity of individual culprit instructions.

The system level support in SimICS allows a user to write separate modules to simulate devices, memory management units, and memory hierarchy simulators. These modules can then be dynamically loaded to extend SimICS. SimICS then provides generic support for passing memory operations to the correct device, as well as providing a range of general services such as extendible command-line interface, event queue, access to process registers, access to physical memory, and a framework for communication among dynamically-loaded extensions.

Using this generic support, we have developed a set of devices to model the Sun4m kernel architecture, which includes the Sparcstation 5, 10, and 20. A kernel architecture means that the same set of operating system binaries are particular to one kernel architecture. These devices include a SCSI chipset with disk, console, ethernet, interrupt chips, I/O memory management unit (I/O MMU), on-chip MMU, timers and counters, DMA, and the boot prom. Properly configured with these devices, CASE today boots either Linux 2.x or Solaris 2.6, and run as a virtual workstation on the local network.

The kernel architecture model is hardware-specific in order to allow us to run completely unmodified operating systems, but the cache model can be changed since it need not be visible to the program. This allow us to study the impact of architectural alternatives on the performance of the application program.

SimICS supports a simple timing model where instructions take one ‘‘cycle’’, but memory operations are allowed to stall the processor several cycles, thus allowing realistic interleaving of memory operations in a multiprocessor.

This work is similar to the SimOS platform, except that the SimICS interpreter is faster when gathering information on cache performance, and the kernel architecture model can run completely unmodified operating system binaries; i.e. the exact same binaries will boot on target hardware.

CASE STUDY
One example of how the CASE platform can be used, is to use it to characterize the cache performance of database applications. Both to quantitatively determine figures such as cache miss ratios, but also (and more important) to relate the obtained numbers to events in the application, operating system, and hardware.

Undertaking such a characterization, statistics are collected using the fact that the processor simulator exports all memory references from the running operating system and binaries. The references are fed into the simulated memory hierarchy, which allows us to evaluate different design alternatives.

In order to exemplify how CASE can be used, we will describe a small and simple scenario. When measuring hit ratios for first and second level caches with PostgreSQL running query 6 from the TPC-D benchmark suite on Linux2.0.30, we recognized that there was a large amount of misses in the second level (L2) cache. By making histograms over all cache blocks during database queries, we found that in the first level cache, a substantial amount of blocks (88%) was cached only once during the entire run. Tracing all accesses to cached-once blocks, we saw that over 95% of them originated from only three instructions in the kernel. These instructions in turn, were responsible for moving and initializing memory objects, in many cases generated by a sequential scan operation in the database application. These blocks could have a tendency to sweep parts of the L2 cache, producing unnecessary high conflict miss ratios. The analysis above clearly shows how powerful CASE is as a tool for studying the interactions between application, operating system, and architecture.

Consequently, to detect the effect of not caching the cached-once blocks originating from the sequential scan, we measured miss ratios in the L2 cache for two scenarios, called All and Selective. All means that caching was enabled for all blocks, whereas in the Selective scenario, blocks arising from data structures used in a sequential scan were not cached. For a 1 Mbyte L2 cache, the miss ratio decreased from 5.1% to 2.8%, and from 3.5% to 1.3% when a 2 Mbyte L2 cache was used. This means that about half of the cache misses are caused by a single function in the application. By using selective caching during sequential scan operations, the amount of conflict misses can be reduced.

In this abstract we have shown how we can use CASE to characterize the behavior of an application and operating system. We have also shown how we can alter the underlying hardware mechanisms in a what-if methodology to propose performance increasing actions, and to analyze the interaction between architecture and applications for different design alternatives.

VPC and IADYN - Project Overviews
Rich Uhlig and Todd Austin
Intel Corporation

Despite their importance in the commercial marketplace, systems based on PC hardware architecture and operating systems remain, for the most part, unstudied in the research literature. Part of the reason has been lack of good, publically-available analysis tools that work in the PC realm; many excellent tools have

long existed for RISC-based systems (SimOS, Embra, Shade, ATOM, Qpt, EEL, Pixie, PatchWrx, etc.), but only recently have similar tools become freely available for x86-based systems, and even these new tools are limited in their ability to consider complete system behavior. In this talk we will describe on-going work in Intel’s Microcomputer Research Lab (MRL) to build new simulation and performance-analysis tools for PC-based systems. We will focus, in particular, on characteristics of PC systems that make them inherently harder to study than RISC-based systems.

The first part of the talk will discuss the VPC project, an effort to build a fast functional PC platform simulator. The VPC design emphasizes completeness and extensibility over raw simulation speed, and we'll discuss the factors we considered in making this design tradeoff. Completeness means that VPC is able to simulate the full execution of unmodified OS and application binaries, including both their kernel-mode and user-mode components. Extensibility means that the VPC simulator kernel has been designed to ease the development of new platform device models, and to make it possible for similar device models (e.g., IDE and SCSI disk interfaces) to share common simulation functions. Our VPC prototype is able to simulate a complete boot of unmodified NT system binaries in under 10 minutes. We will discuss work-in-progress that applies VPC to the analysis of popular office productivity applications (e.g., Office97), server workloads (e.g., Database, Web and File Servers), and games.

The second part of the talk will discuss the IADYN project, an ongoing effort to build a family of iA32 ISA simulation components. With over 15,000 semantically distinct variants of more than 700 instructions, the iA32 ISA presents a unique challenge in ISA simulator design. At the same time, varied user requirements ranging from fast functional simulation for workload positioning and light tracing to detailed functional simulation with support for micro-operation generation and arbitrary speculation serve to further amplify complexity. To help manage this development challenge we've built DEFIANT, a simulator development framework based on a formal model of the iA32 ISA. During this part of the talk, we will highlight the components being constructed as part of the IADYN project, lend some insights into the complexity of the iA32 ISA, and briefly describe our DEFIANT simulator development environment.

Session 1 : Workload Behavior

Commercial Applications on Shared-Memory Multiprocessors
Zheng Zhang and Sekhar Sarukkai
Hewlett-Packard Labs

Importance of Proper Configuration in Architectural Evaluations using Database Workloads
Kimberly Keeton and David A. Patterson
University of California - Berkeley

System Design Considerations for a Commercial Application Environment
Luiz Andre Barroso and Kourosh Gharachorloo
Western Research Laboratory
Digital Equipment Corporation

Exploiting Caches Under Database Workloads
Pedro Trancoso and Josep Torrellas
University of Illinois at Urbana Champaign

Commercial Applications on Shared-Memory Multiprocessors

Zheng Zhang and Sekhar Sarukkai

HP Labs,
Page Mill Road, Palo Alto, CA

Acknowledgements

Thanks are due to: Gheith Abandah (U. Michigan),
and members of our groups at HPL (Tom Rokicki, Milon Mackey and Josep Ferrandiz)

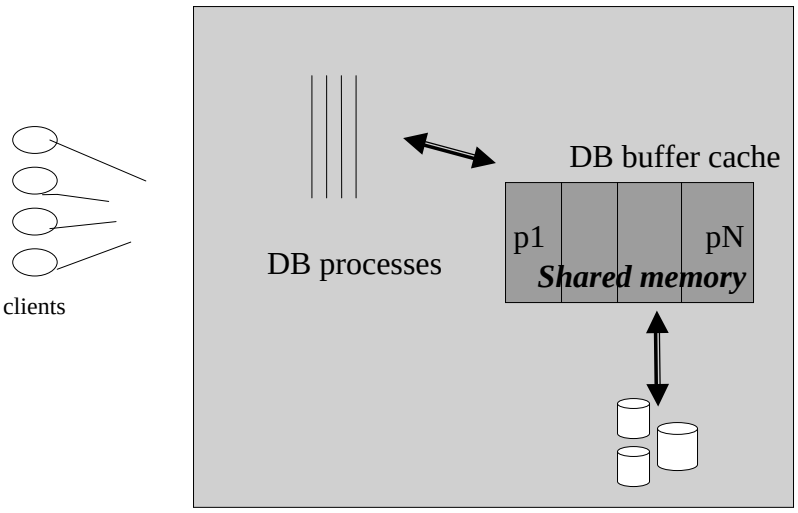
Commercial Applications

- > 80% of server market
- Of this, database servers ~40%
- Application landscape changing with internet technologies, multi-tier applications and multimedia
- Hard to get source code; harder still to work independent of myriad software developers
- Difficult to publish any insightful solutions (direct impact on bottom-line$$ and effects multiple companies)

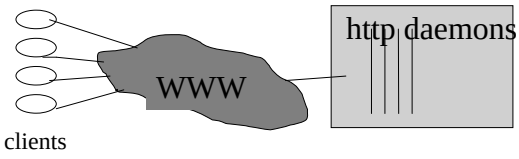
Example Commercial Applications

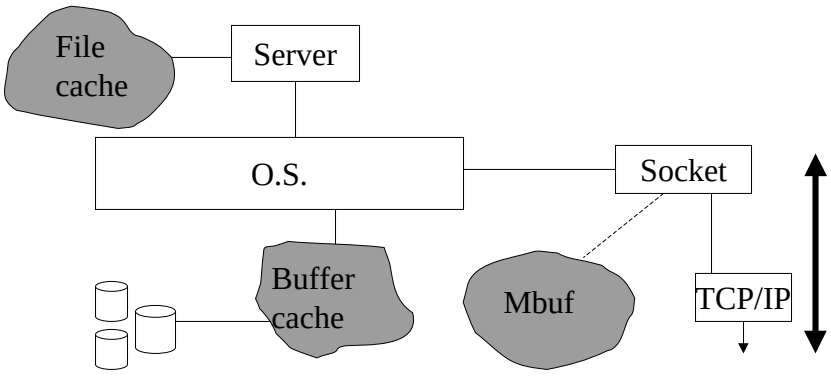
- Database
  - On-line transaction processing (OLTP)
  - Decision support (DSS)
- Internet
  - Web servers
  - JDBC, Web enabled OLTP
  - Multi-tier applications, E-commerce
- *Results from traces of these applications and simulations of different architectural choices*

Database Servers

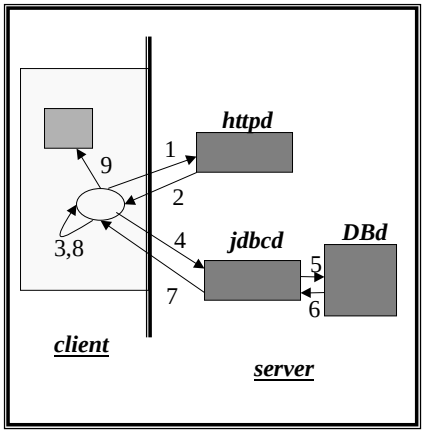


Web Servers

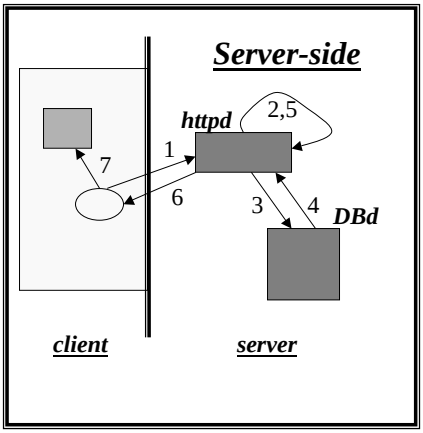




Web-enabled Database Access



Client-side DB access



Server-side DB access

System Requirements for Commercial Apps

- Performance is not the only criteria; requirements include:
  - Performance
  - Manageability / Administration
  - Availability
  - Software deployment (ex: multiple vs. single instance, etc.)
  - Legacy integration
  - Cost, etc...
- *Here we concentrate only on the performance aspects*

Analysis Methodology Issues

- Issues to consider:
  - Access to and monitoring of realistic and properly configured systems
  - Tracing overhead, tools
  - Size of traces (> 3GB compressed for four sample transaction of a reasonable multi-tier application on processors); sampling not a cure all.
  - Assumptions about OS choices and application-specific features in simulations (when scaling # of processors)
  - Projection of future workloads

Key Performance Indices

- OS Impact
- Memory-hierarchy performance
- Communication

OS Impact on Applications

	System	User
OLTP	10-20%	80-90%
OLAP	10-15%	85-90%
JDBC	0-5%	95-100%
Web serve	75-85%	15-25%
E-comm	30-40%	60-70%

OS Impact

- Implications for System Analysis
  - Need OS traces or SimOS-type approach for some critical applications
  - Even when OS contribution (in terms of time) is not much, there are many opportunities for misusing trace with improper assumptions about OS choices (examples: scheduling, TLB performance,etc.)
  - Scalability is seriously affected due to shared data structures in kernel. Example: specWEB is almost embarrassingly parallel, yet has very poor scalability.

Communication Patterns

- OLTP
  - Moderate to low miss-rates
  - Coherent misses account for a significant fraction of misses.
  - Lots of migratory data, accounts for approximately 70% of all 3 hop misses.
  - Pre-fetching opportunities to tolerate latency
  - False sharing could be an issue for un-tuned code and due to programmer inexperience

Communication Patterns

- E-commerce
  - Database servers exhibit similar patterns as for OLTP applications
  - Web and application servers have much lesser communication
  - Communication between application and database servers through the use of cursors can be performed with TCP/IP or shared-memory
    - Use of shared-memory for communication between the two tiers results in a two-fold increase in throughput.
    - This difference could be reduced with low-latency user-space communication protocol for message passing.

Synchronization

- Rare (if any) use of barrier synchronization
- Sophisticated relaxed, application-level locking mechanisms
  - could potentially capitalize on relaxed memory models

Caching Behavior

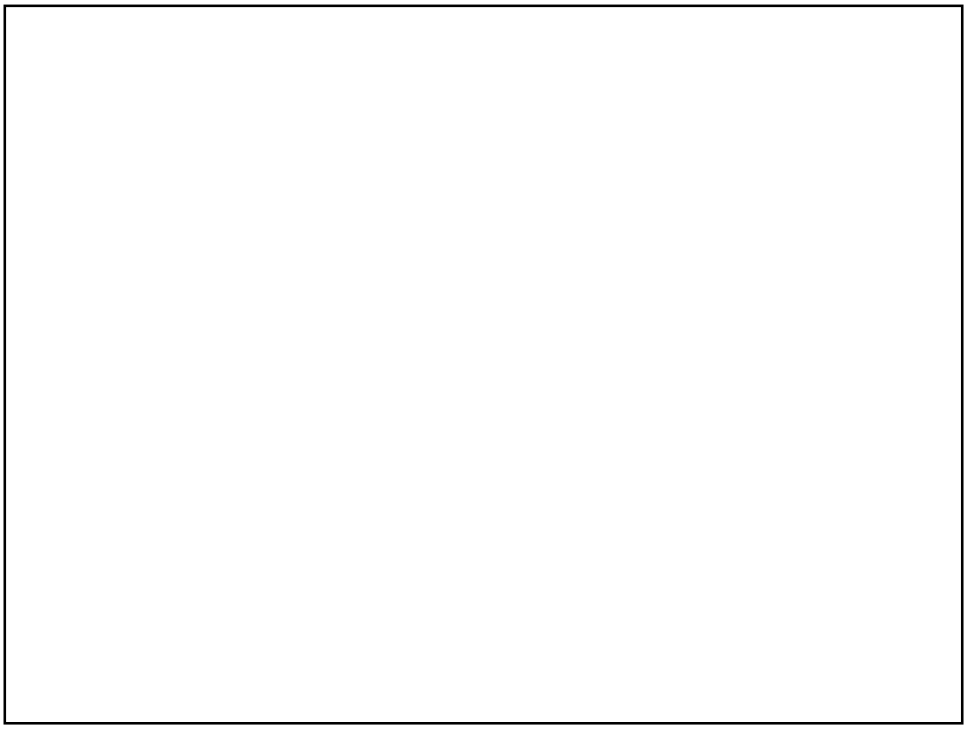
- No distinct working set knee
- Miss rates: varies from 10 to 20 per 1000 instructions (cache size up to 1MB).
- Spatial locality: granularity is pretty low (< 64 bytes)

Memory<-> IO Subsystem

- Size of memory directly impacts performance of OLTP applications
- Most applications are memory intensive; In general, larger the memory lesser the IO rate.
- IO is significant contributor to bus traffic. For example, most cache misses are due to IO in many TPC-D queries.
- Do not consider IO subsystem in isolation from the memory subsystem (can lead to wrong conclusions; ex: temporal locality in TPC-D)
- Huge amount of copies from user to system space for specWEB due to network IO.

Conclusions

- Commercial applications are diverse and rapidly evolving; will the naturally distributed nature of the internet impact application design?
- OS is a critical component
- Web servers: Network IO, OS scalability, shared-memory not that critical
- Database servers: IO, coherence misses are significant and migratory in nature
- Multi-tier servers: Inter-tier communication can make use of shared-memory in some cases, interpretive code in application servers
- Don't forget other requirements for commercial servers



The Impact of Hardware and Software Configuration on Computer Architecture Performance Evaluation

Kimberly Keeton and David A. Patterson

UC Berkeley Computer Science Division
{kkeeton,patterson}@cs.berkeley.edu
http://http.cs.berkeley.edu/~kkeeton/Talks/

Workshop on Computer Architecture Evaluation
Using Commercial Workloads
February 1, 1998

Motivation: Potential Errors in Database Benchmarking

- ✶ Database benchmarks more complex than other benchmarks (e.g., SPEC)
  - Database server complexity
  - Large data sets and associated disk I/O
  - Client/server networking
- ✶ Potential mistakes in architecting experimental systems
  - Examples:
    - Underpopulate disk system
    - Use too little memory
    - Scale down data set to fit entirely in memory
  - Why?
    - Resource constraints
    - Lack of understanding of underlying software

2

Motivation: Cautionary Tale

- ✶ Potential danger, Will Robinson!
  - Departures from well-balanced system affect behavior adversely
  - Anomalies may be unknowingly observed as correct
  - Designers may go down wrong path
    - Include optimizations meaningless for well-balanced system
    - Miss opportunities for true improvements
- ✶ Careful attention paid to building well-balanced systems
- ✶ Configuration information should be reported with performance results

3

Outline

- ✶ Motivation
- ✶ Factors impacting validity of configuration
- ✶ Detecting poorly configured systems
- ✶ Rules of thumb for building well-balanced systems
- ✶ Conclusions

4

Factors Impacting Configuration Validity

- ✶ I/O system hardware
  - # and speed of disks, I/O controllers, disk I/O bus, processor I/O bus
  - Caching provided by I/O controllers
  - Data layout to avoid hot spots
- ✶ Memory-related hardware
  - Amount of physical memory
    - Size of buffer cache
    - Memory bandwidth delivered by system
- ✶ Database software parameters
  - Buffer pool size and management strategy
  - Number of threads/processes
- ✶ Benchmark configuration parameters
  - Database size (e.g., TPC-C's number of warehouses)
  - Number of simulated clients

5

Detecting Poorly Balanced Configurations

Architectural Behavior	**Properly Configured**	**Low Memory (Relative)**	**Low Warehouses (Relative)**
TpmC:warehouse ratio	10.1	8.0	27.8
Relative transaction thruput	100%	79%	115%
% User time	78%	90%	106%
Disk reads/sec	1550	159%	69%
Disk writes/sec	1280	96%	86%
Database CPI	2.91	112%	92%
Database Computation CPI	0.96	100%	99%
Datbase resource stalls CPI	0.57	130%	86%
Database instruction stalls CPI	1.36	118%	91%

6

Rules of Thumb for Configurations

- ✶ Few hard-and-fast rules
- ✶ "Well-balanced" == getting maximum useful work out of each of system components
  - No way to get more useful work from existing system
- ✶ System tuning combination of:
  - Qualitative measures
  - Quantitative measures
  - Arbitrary decisions
- ✶ Cost is often-forgotten metric
  - May be okay to sacrifice 5 - 10% throughput for $ savings
- ✶ Can only understand component behavior in context of larger system
  - Big performance gain from new feature in prototype may be in the noise in larger configuration
  - => Build the larger system, if possible

7

Rules of Thumb (cont.)

- ✶ Qualitative measures
  - Develop guess-timates of performance from previous studies and TPC results (http://www.tpc.org/)
  - Get rid of system bottlenecks
    - Ex: spread I/O over many disks, so no hot spots
- ✶ Quantitative measures
  - TPC-C:
    - Valid tpmC:warehouses ratio: 9.0 - 12.7
    - 5-sec response time constraints
    - ~0.5 I/Os per second per tpmC
  - CPU utilization: upper 90%s
    - Nearly no idle time
- ✶ Arbitrary decisions
  - Cost-motivated decisions
    - Ex: how much memory?

8

Report from the Trenches: TPC-C System Configuration

- ✶ Size of database α offered transaction load
  - How many transactions per minute can processors support?
- ✶ Determining I/O requirements as f(memory size):
  - Determine %age of data accesses that hit memory buffer cache
    - May be complicated by preferential caching scheme
  - Determine I/Os required for offered load and mem. config.
- ✶ Determine I/O system configuration
  - Disks in system for I/Os / sec, not capacity
  - Determine disk random I/O rate, including bus xfers, s/w overhead
    - Avoid queueing at disks
  - TPC-C has skewed distribution of I/O accesses
    - Some disks hotter than others
    - Ensure hot disks don't exceed comfortable I/O request rate
- ✶ Complications: tension between steady-state and checkpoints

9

Summary and Conclusions

- ✶ Database workloads are complex
  - O(10) configuration knobs to turn
  - Mistakes are easily made
- ✶ Improper H/W and S/W configurations adversely impact observed architectural performance
  - May lead designers to make wrong choices
- ✶ Quantitative guidelines for detecting poor configurations
  - TPC-C: tpmC:warehouses, I/O rates, CPU utilization
- ✶ Rules of thumb for designing proper configurations
  - Qualitative measures
  - Quantitative measures
  - Arbitrary decisions

10

Conclusions: What Should Computer Architects Do?

- Continue measuring databases and other commercial workloads!!
- Until scaled-back configs shown to correctly emulate full-scale configs, build the complete system
- Apply rules of thumb to build well-balanced system
- Use quantitative guidelines to detect poor configurations
- Report configuration info with performance results

11

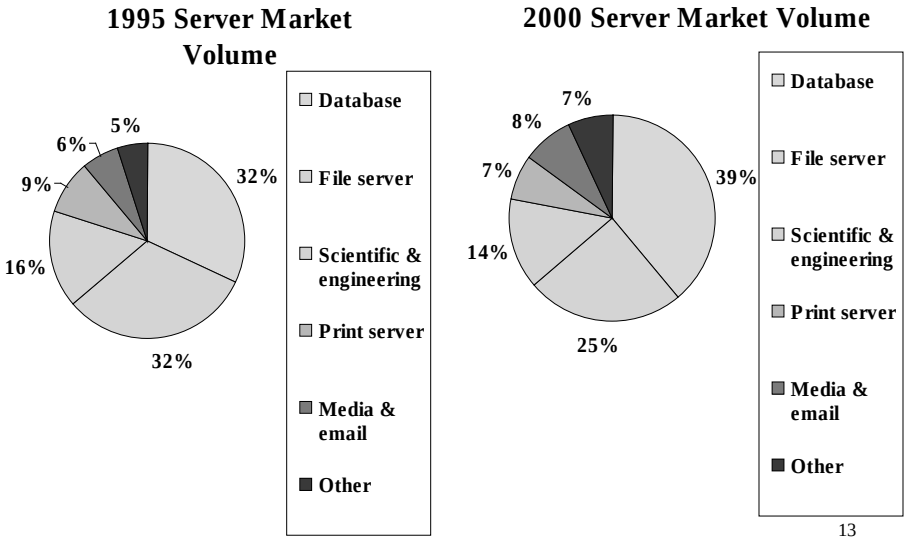
Backup Slides

(These slides used to help answer questions.)

12

Server Market Breakdown

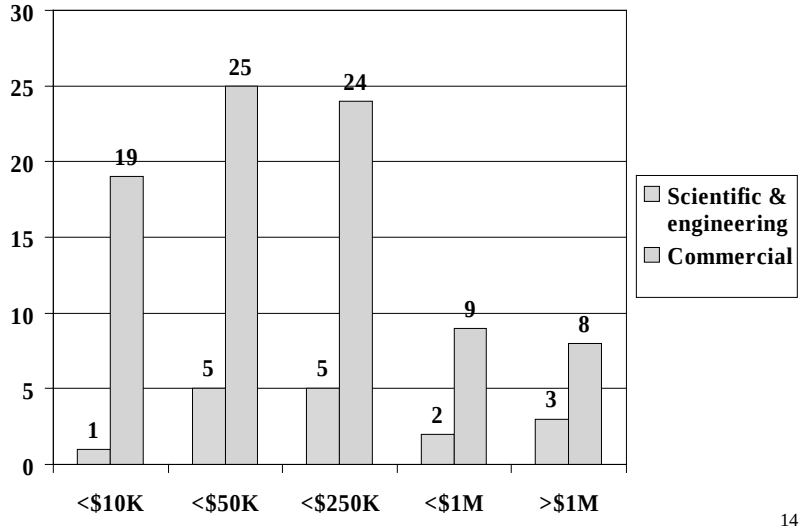
Source: Stenstrom, et al., *IEEE Computer*, December 1997



13

1995 Market Volume by Machine Price

Source: Stenstrom, et al., *IEEE Computer*, December 1997



14

Database Workload Characteristics

Characteristic	**Decision Support**	**Transaction Process.**
Business question	Historical: support for forming business decisions	Operational: day-to-day business transactions
Industry benchmark	TPC-D	TPC-C
Query complexity	Long, very complex queries	Short, moderately complex queries
Portion of DB accessed per query	Large	Small
Type of data access	Read-mostly	Read-write
How are updates propagated?	Periodic batch runs or background "trickle" streams	Through most transaction types

15

TPC-C Overview

- ✶ Moderately complex online transaction processing (OLTP)
- ✶ Order-entry for a wholesale supplier
  - Users and database size scale linearly with throughput
  - Throughput at or below a 5-sec response
- ✶ 5 transaction types
  - New-Order, Payment, Order-Status, Delivery, Stock-Level
  - New-Order and Payment dominate
- ✶ Performance metrics
  - New-Order transaction rate (tpmC)
  - Price/Performance ($/tpmC)

16

TPC-C Rules of Thumb

- Source: Catharine van Ingen, Microsoft Research, 12/97

- 1.2 tpmC per user/terminal
- 10 terminals per warehouse
- 65-70 MB/tpmC minimum priced disk capacity
- 0.5 I/O / sec / tpmC
- 2 KB - 8 KB random I/Os (DBMS-specific)
- 300 - 700 KB memory/tpmC
- Benchmarking tends to configure storage system, then regenerate and balance keeping CPU saturated

17

TPC-D Overview

- Complex Decision Support
- Pseudo-ad hoc queries (parameterized)
- Queries and updates
  - 17 read-only queries; 2 update streams
  - quantized database size (1, 10, 20, 100, 300, 1000, 3000 GB)
- Performance metrics
  - Power (Qppd)
  - Throughput (QthD)
  - Price/Performance ($/QphD)

18

TPC-D Performance Metrics

- Power (QppD)
  - single user query processing power
  - (1 * 3600 * SF) / geometric_mean(Q1…Q17,UF1,UF2)
  - SF = scale factor (e.g., 1, 10, 30, 100, 300, 1000, 3000 GB)
- Throughput (QthD)
  - multi-user query throughput
  - (S * 17 * 3600 * SF) / Ts
  - S = # concurrent users, each executing all 17 read-only queries
  - Ts = total elapsed time
- Price per Queries per hour ($/QphD)
  - QphD = square_root(QppD * QthD)

19

TPC-D Rules of Thumb

- Source: Catharine van Ingen, Microsoft Research, 12/97

- Full storage requirements about 3-5x database size
- Typical operations
  - Scan, aggregates (e.g., min, max, count) & 1-pass, 2-pass sort/join
  - Avg. 500-2000 instructions/record (DBMS-, query-specific)
  - 200 B/record; 1-3 tables/query
  - Records are read 1-2x, written 0-1x
- I/O pattern tends to be sequential
  - 8KB - 4MB reads; 8KB - 64KB writes (DBMS-specific)
  - Index scan may be more random (DBMS-specific)
- Random updates 0.1% of data per query stream

20

Detecting Poorly Balanced Configurations

Architectural Behavior	**Properly Configured**	**Low Memory**	**Low Warehouses**
TpmC:warehouse ratio	10.1	8.0	27.8
Relative txn thruput	100%	79%	115%
User/system breakdown	78% / 21%	70% / 30%	83% / 16%
Disk reads/sec	1550	2460	1070
Disk writes/sec	1280	1235	1100
Database CPI	2.91	3.25	2.67
Database Computation CPI	0.96	0.96	0.95
Datbase resource stalls CPI	0.57	0.74	0.49
Database instruction stalls CPI	1.36	1.61	1.24

21

System Design Considerations for a Commercial Application Environment

Luiz André Barroso and Kourosh Gharachorloo
Western Research Laboratory

Contributions from: Edouard Bugnion, Jack Lo, and Parthas Ranganathan

Myths About Database Applications

"I/O performance is all that really matters"

"Most of the time is spent in the OS"

"You need a $1M+ system to study it"

"Applications are too complex for a simulation environment"

Shift of Focus

- DB applications used to be I/O bound
- These days I/O performance matters, but…
  - I/O bandwidth/latency/architecture has improved
  - DB engine software has evolved to tolerate I/O latency
- Most applications can be made CPU bound
- Today's challenges:
  - **Memory system!**
  - Processor architecture

Outline

- Introduction
- Scaling down DB workloads
- Tools and methods
- Memory system performance
- Processor architecture
- Summary

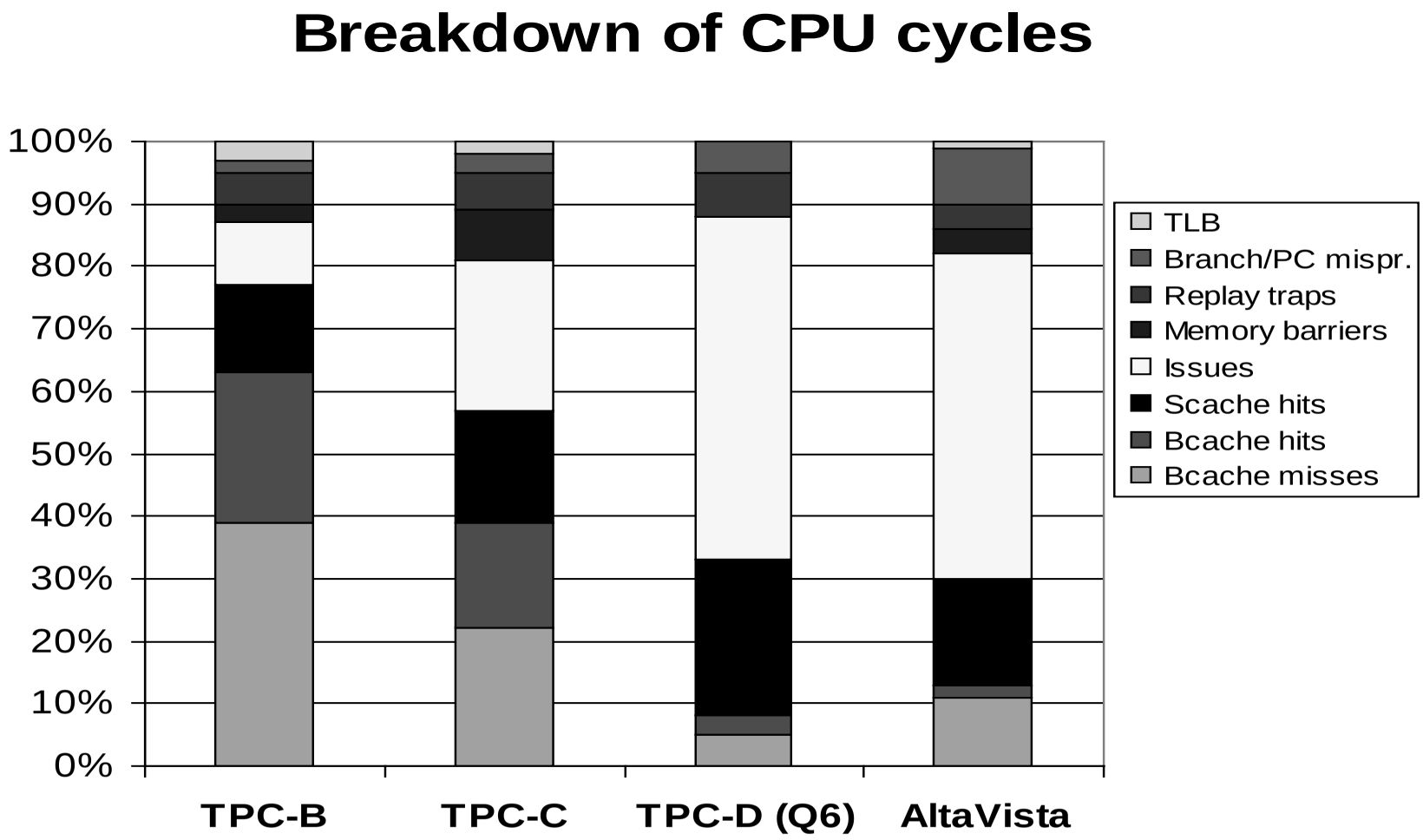
Scaling Down DB Workloads

- Not a trivial task, but a feasible one
  - Requires deep understanding of the workload behavior
  - Depends on the target of the study
  - Critical for enabling simulation studies
- Extensive monitoring of the native application is required to verify scaling assumptions before simulation
- In-memory runs can be used for memory system studies
- Important to exclude idle time from calculated statistics

Tools and Methods

- Good hardware event counters (as in the 21164) are key
- Rich set of performance tools (ATOM, DCPI, IPROBE) enable:
  - Careful tuning and detailed profiling of the code
  - Detailed breakdown of the causes of processor stalls
  - Trace generation
  - Validation of scaling assumptions
- Powerful simulation infrastructure (SimOS-Alpha)
  - Enable the study of future designs
  - Account for both user and system behavior
  - Study complex applications “out-of-the-box”
  - Access to events not visible in a running system

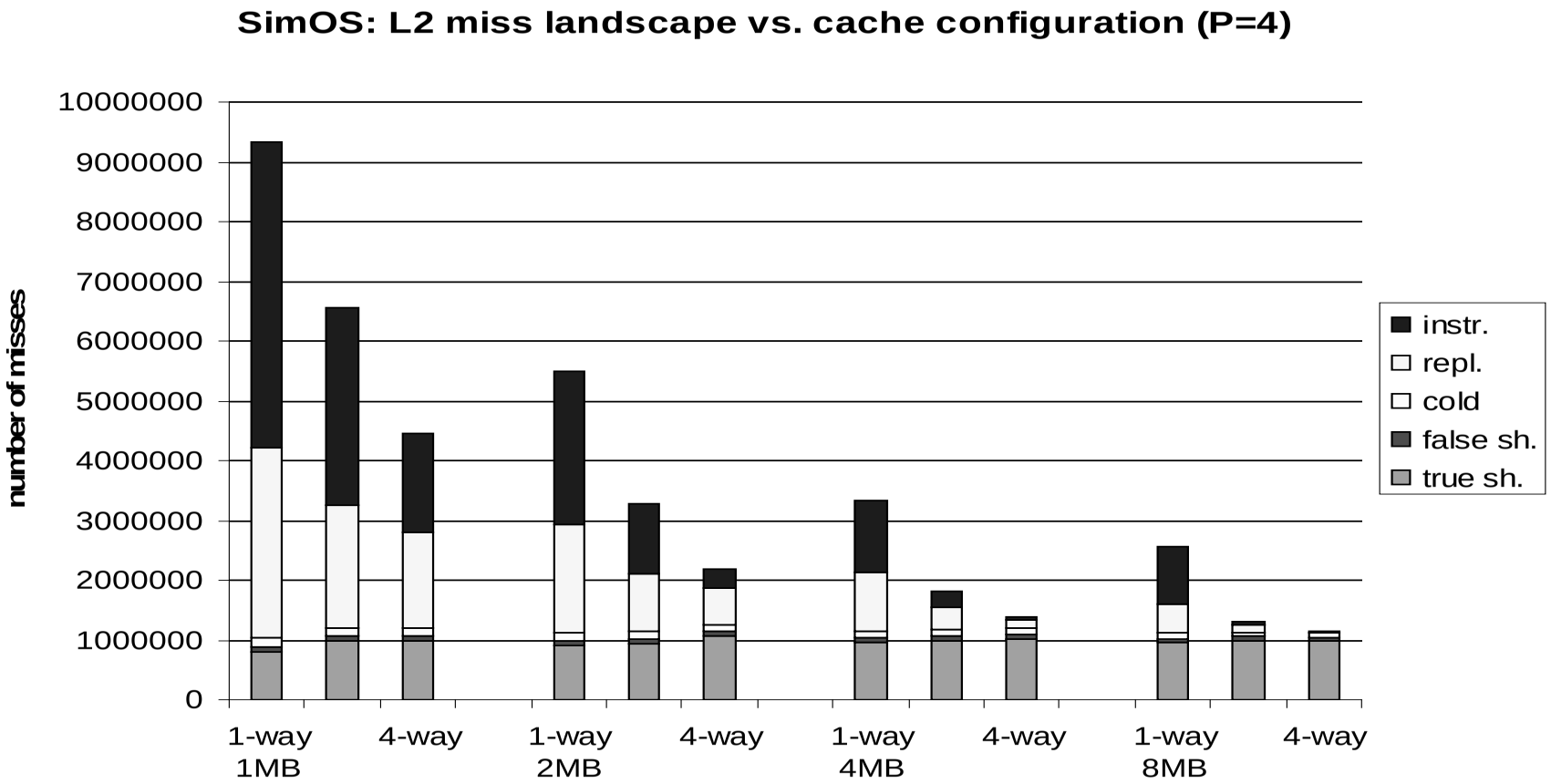
Memory System Performance



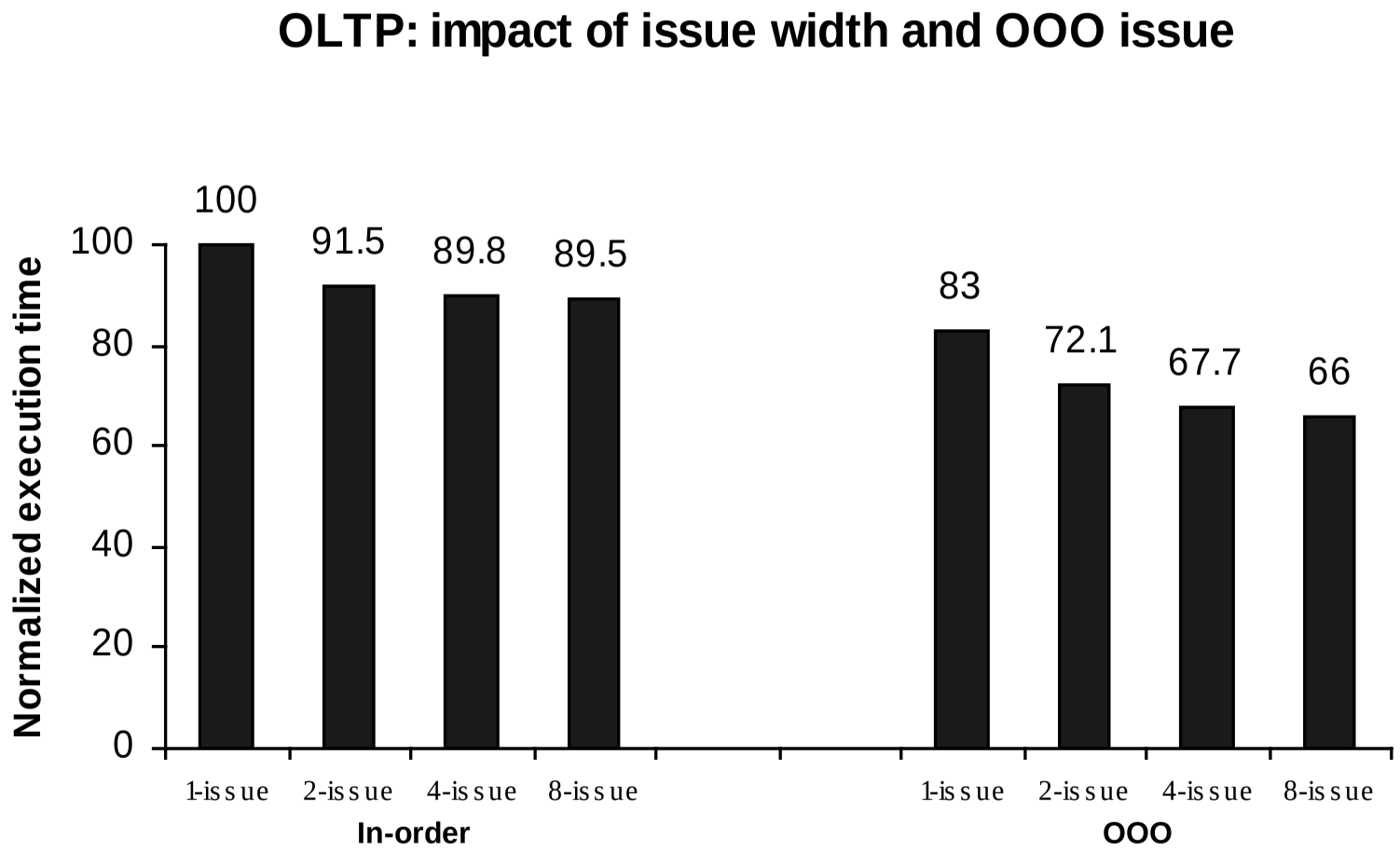
Memory System Performance

- OLTP:
  - Very large instruction and data footprints
  - Over 50% of misses will require a cache-to-cache transfer (e.g., dirty miss)
  - Critical design issue: large/fast secondary caches and support for fast cache-to-cache transfers
- DSS and AltaVista
  - Smaller instruction and primary data working sets
  - Huge secondary data working set, but with good spatial locality
  - No significant sharing
  - Critical design issue: fast and large enough (64-128KB) on-chip caches

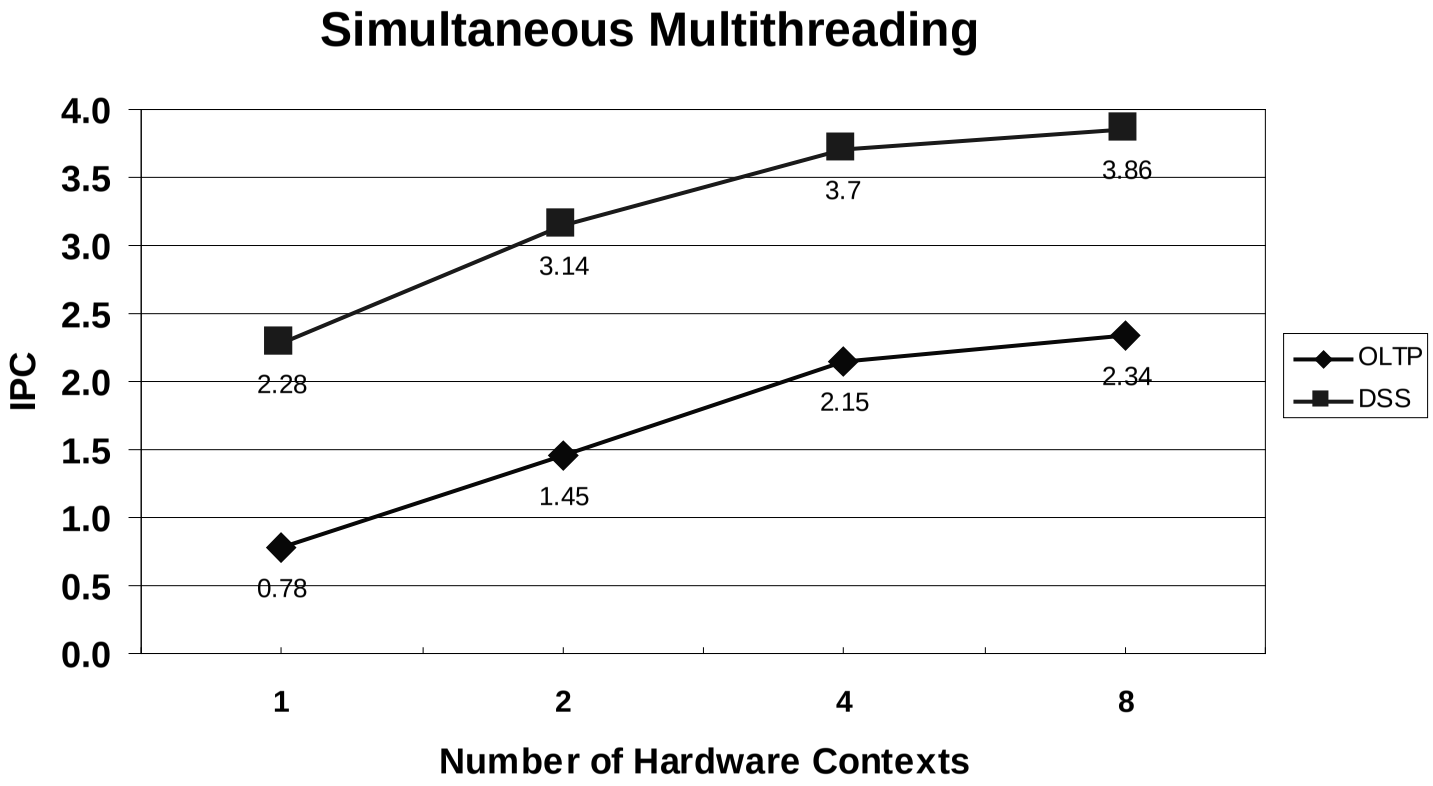
OLTP: Effect of Cache Organization



Processor Architecture



Processor Architecture



Summary

- Memory system is the next challenge for DB performance
- Problem size can be scaled down, but very carefully
- Combination of monitoring and simulation is very powerful

- Diverging memory system designs:
  - OLTP: use large/fast secondary caches and optimize the dirty miss case
  - DSS, AltaVista: use large/fast on-chip caches; may perform better without a secondary cache
- Both out-of-order and wider issue (up to 4-way) help
- SMT could improve OLTP performance significantly

Exploiting Caches Under Database Workloads

Pedro Trancoso and Josep Torrellas

[trancoso,torrella]@cs.uiuc.edu
Department of Computer Science
University of Illinois at Urbana-Champaign

Motivation

- Architectures are changing:
  - larger memories and deeper hierarchies
- Database systems are still I/O oriented
- Main-memory database systems are not good:
  - Uniform memory access assumption

- Target workload:
  - DSS applications with large part of data memory resident
- Proposed solution: Cache-Oriented Optimizations

Overview

- Motivation
- Database background
- Optimizations
- Results
- Conclusions

Database Background

- Query Execution:
  - Parser, Decomposer, **Optimizer**, and Executor
- Query Tree:
  - bushy, left-deep
- Operations:
  - scan, join, sort, group, aggregate
- Algorithms:
  - scan: sequential and index
  - join: nested loop, index nested loop, hash, sort merge

Cache-Oriented Optimizations

- Blocking / Tiling Algorithms
- Query Optimization
  - Modeling Read Misses
  - Cost Functions
  - Optimizer
- Data Layout
  - Data and Index structures
- Data Prefetching

Blocking

- Pipelined nested loop join:

```
join_n()
  while( TmpTuple_n = join_n-1() )
    while( InTuple_n = scan( Input_n ) )
      if match( TmpTuple_n, InTuple_n )
        TmpTuple_n+1 = project( TmpTuple_n, InTuple_n )
        return TmpTuple_n+1
```

- Pipelined blocked nested loop join:

```
join_n()
  while( TmpBlock_n = join_n-1() )
    while( InTuple_n = scan( Input_n ) )
      for( BSize_n )
        TmpTuple = get_tuple( TmpBlock_n )
        if match( TmpTuple, InTuple_n )
           add project( TmpTuple, InTuple_n ) to TmpBlock_n+1
    return TmpBlock_n+1
```

Modeling Read Misses

- Nested loop join:

```
Misses = read_outer_input +
         outer_size x read_inner_input +
         copy_output
```

- Blocked nested loop join:

```
Misses = read_outer_input +
         nblocks x read_inner_input +
         copy_output +
         cross +
         self
```

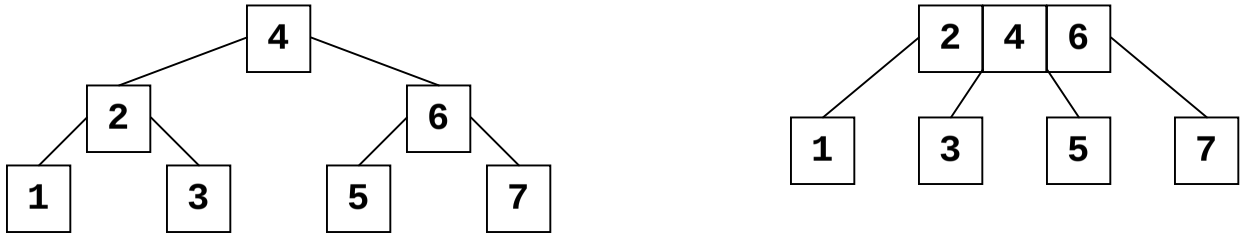
Cost Functions

- Total Misses
  - $\mathbf{cost_{total} = \sum misses(result\_size)}$
- First Misses
  - $\mathbf{cost_{first} = \sum misses(output\_size)}$
- Combined Misses
  - $$\mathbf{cost_{combined} = \sum \left(f \times \frac{misses}{cost_{first}} + (1 - f) \times \frac{misses}{cost_{total}} \right)}$$

Data Layout

- Database Data

- Index Structures



Data Prefetching

- Sequential Scan

```
for( i = index; i < TableSize; i++ )
  PREFETCH( Table[ i + PrefOffset.SelectAttr )
  PREFETCH( Table[ i + PrefOffset.JoinAttr )
  if match( Table[ i ].SelectAttr )
    addToMatches( Table[ i ] )
    for( j = i + 1; j < i + PrefOffset; j++ )
      if match( Table[ j ].SelectAttr )
        addToMatches( Table[ j ] )
    return Matches
```

- Index Scan

```
low = 0; high = IndexSize
while( not leaf )
  mid = low + ( high - low ) / 2
  PREFETCH( Index[ low + ( mid - low ) / 2 ] )
  PREFETCH( Index[ mid + ( high - mid ) / 2 ] )
  if( value = Index[ mid ] )
    return tuple( Index[ mid ] )
  else if( value > mid ) low = mid
  else high = mid
```

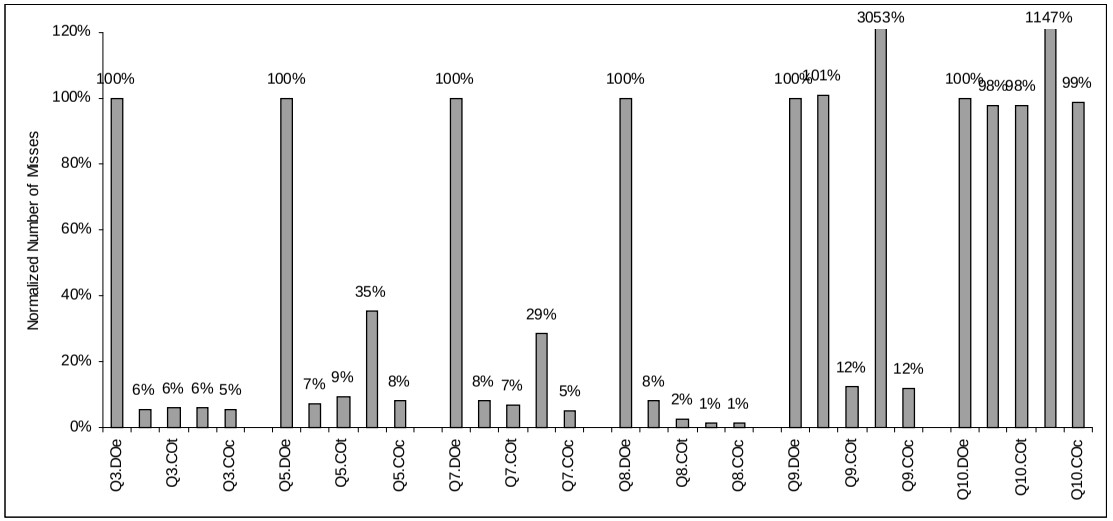
Query Plans

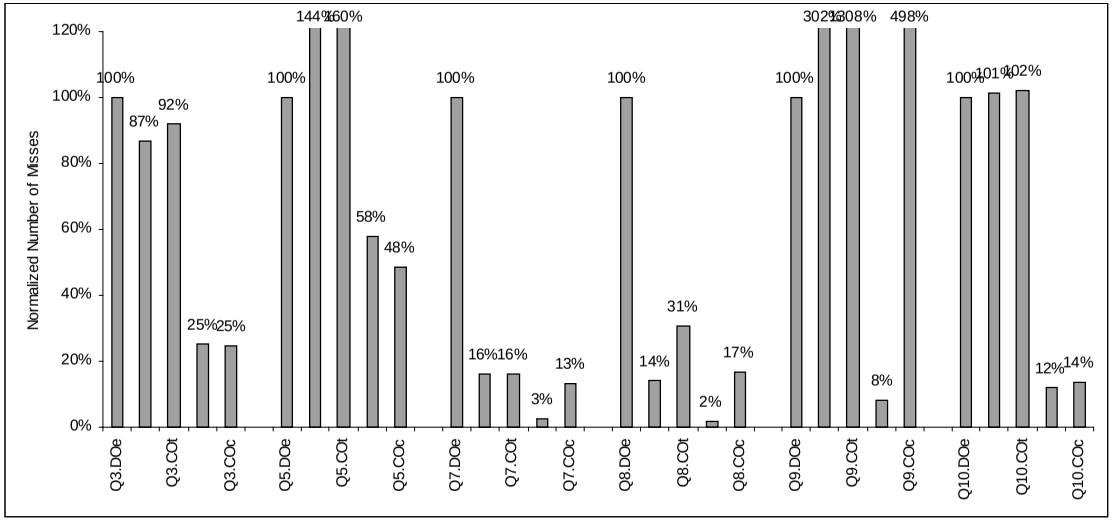
Disk Oriented Enhanced (*DOe*)	Generated by Postgres95 (enhanced with pointers)
Cache Oriented eXchange (*COx*)	*DOe* using blocked algorithms instead of traditional ones
Cache Oriented Total (*COt*)	Generated by optimizer with *Total Misses* cost function
Cache Oriented First (*COf*)	Generated by optimizer with *First Misses* cost function
Cache Oriented Combined (*COc*)	Generated by optimizer with *Combined Misses* cost func
COc + Data Layout (*COc+dt*)	*COc* plus data layout optimization
COc + Prefetch (*COc+p*)	*COc* plu prefetch optimization
COc + Pref + Bypass (*COc+pb*)	*COc* plus prefetch and bypass optimizations

Evaluation

- Queries and query plans
  - 6 queries from TPC-D: Q3, Q5, Q7, Q8, Q9, Q10
  - Data generated with dbgen for SF=0.01
  - plans: DOe, COx, COt, COf, COc, COc+dt, COc+p, COc+pb
- Simulated architecture
  - dynamic superscalar processor (4 issue)
  - 4-Kbyte D-cache, 16-byte line and 2-way set associative
  - 1 cycle cache hit; 30 cycles memory hit
  - MINT based

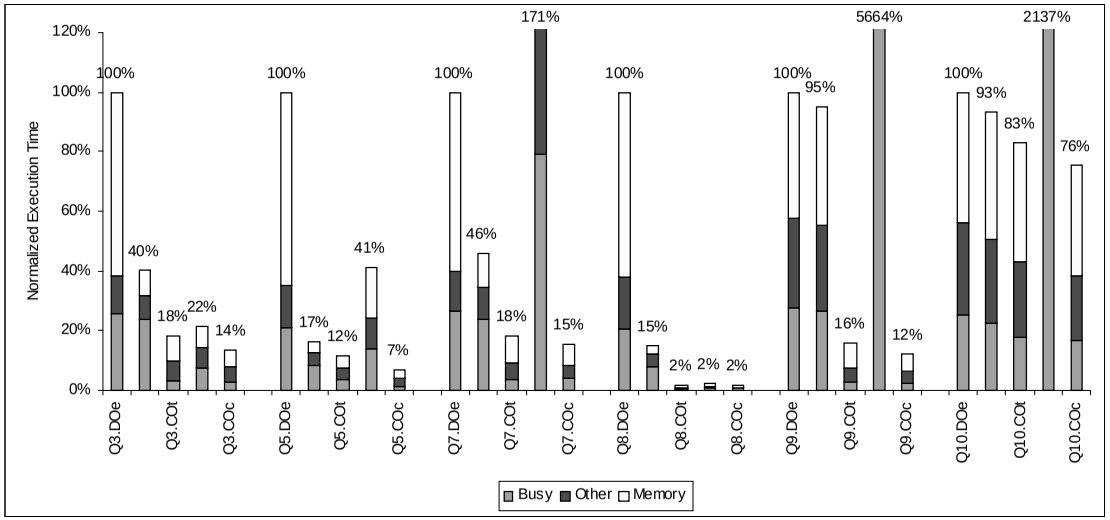
Results - Number of Misses

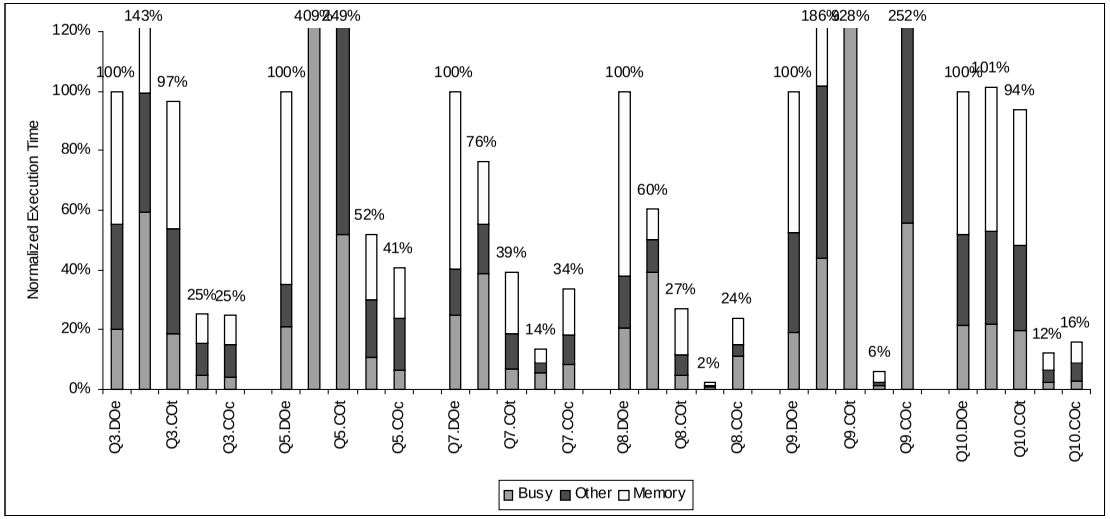




- Total Query
  - *COx*: maybe
  - *COt*: good
  - *COf*: bad
  - *COc*: good
- First Result
  - *COx*: maybe
  - *COt*: maybe
  - *COf*: good
  - *COc*: good

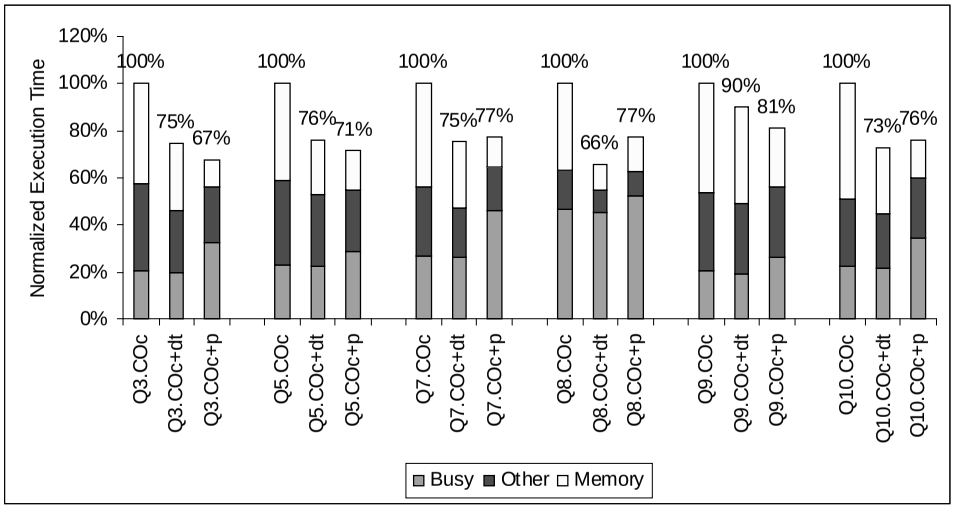
Results - Execution Time

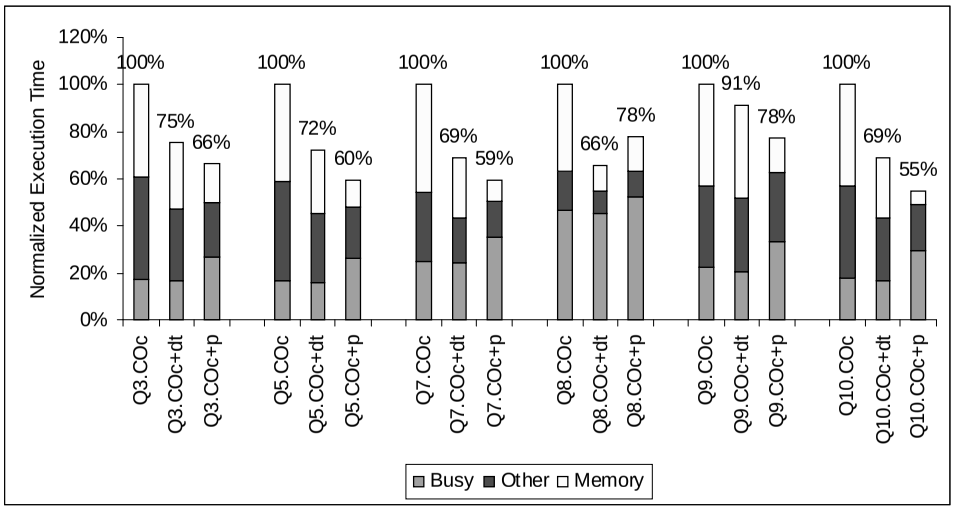




- Total Query
- First Result
- For Both:
  - Same trends
  - Changes in *Busy*
  - `COC`: overall best

Results - Optimizations





- Total Query
  - both optimizations effective
  - *COc+p* reduces *Memory* but *Busy* penalty
- First Result
  - *COc+p* also good for first result

Conclusions

- Optimization work:
  - Cache-oriented algorithms are effective but no guarantees
  - Cache-oriented optimizer guarantees best performance
  - Cache-oriented optimizations give additional improvements
- Good Improvements:
  - Miss reduction of 80%
  - Execution time reduction of 85%