

# COMPUTER SYSTEM EVALUATION WITH COMMERCIAL WORKLOADS

JIM NILSSON, FREDRIK DAHLGREN, MAGNUS KARLSSON, PETER MAGNUSSON<sup>†</sup>, and PER STENSTRÖM

{j,dahlgren,karlsson,pers}@ce.chalmers.se  
Department of Computer Engineering  
Chalmers University of Technology  
SE-412 96 Göteborg, Sweden

<sup>†</sup>psm@sics.se  
Swedish Institute of Computer Science  
Box 1263  
SE-164 28 Kista, Sweden

## Abstract

*Instruction-level simulation techniques are the predominant approach to evaluate the impact of architectural design alternatives on the performance of computer systems. Previous simulation approaches have not been capable of executing unmodified system as well as application software at an acceptable performance level. Commercial applications, such as databases, constitute a particular challenge.*

*The SIMICS/sun4m platform has been designed to efficiently execute completely unmodified software binaries, such as databases and operating systems. Moreover, it is possible to flexibly model a variety of computer system architectures. We describe in this paper how the platform is used (i) for software development and application performance tuning for a given hardware architecture, (ii) for evaluation of performance increasing architectural modifications with focus on a particular application, and (iii) for combined performance increasing measures of both the hardware architecture and the software system.*

## 1 Introduction

We recognize an increasing importance in identifying, and finding remedies for, performance bottlenecks for complex commercial applications such as databases, on future single and multiprocessor systems. In order to do this, suitable tools for system evaluation are needed. Such a tool must be able to cover the software/hardware interactions, including the effects of operating system and I/O. It should also provide the basis for architectural modifications with the purpose to increase application performance, as well as the ability to examine application and operating system adaptations for increased performance on a particular target architecture. The SIMICS/sun4m simulation environment is such a tool.

To correctly model interactions of system software and underlying hardware, several possibilities exist. We believe that detailed *simulation* yields the most promising alternative. Other commonly used methods to study complex systems (such as running the workload directly on target hardware), all lack the observability and versatility of simulation. They also do not exhibit the accuracy and level of detail that simulation can provide. Several simulation techniques exist, but we argue that *instruction-set simulation* [2] allows for the accuracy and observability needed to accommodate system software interactions.

The SIMICS/sun4m platform is built around SIMICS [15]

– a program driven SPARC<sup>1</sup> V8 instruction-set simulator. Together with precise simulated models of hardware devices that comprise the sun4m<sup>2</sup> architecture, and the ability to accurately model detailed implementations of this architecture, we possess a simulator platform that is able to boot and run unmodified operating systems, such as Linux and Solaris. This simulator provides the means for evaluating the performance of contemporary and future multiprocessor computer architectures, and their interaction with operating systems and applications, such as databases.

The goal of the SIMICS/sun4m platform has been to accommodate detailed system models of e.g. memory and I/O systems and at the same time provide a reasonable slowdown to make it feasible to execute large applications. While the slowdown depends on the level of detail of the system model, a slowdown of about 25 is achieved for simple memory system simulations.

While the underlying principle of SIMICS/sun4m is presented in an accompanying paper [7], we focus in this paper on how one can use it to evaluate performance of commercial workloads, taking the complete system behavior into account. In our studies, we have used the Linux 2.0.30 operating system, with the PostgreSQL database handler from Berkeley.

The rest of this paper is organized as follows. Section 2 identifies which system aspects are important to track performance bottlenecks in database applications. It also identifies the weaknesses of previous evaluation methods so as to understand the approach taken in SIMICS/sun4m. Section 3 then describes how SIMICS/sun4m addresses these weaknesses and Section 4 presents how SIMICS/sun4m has been used to identify performance bottlenecks in different parts of the complete system, before we conclude in Section 5.

## 2 Background

Historically, performance evaluations of computer systems have mainly targeted scientific and engineering application domains, utilizing benchmark suites such as SPEC for uniprocessors and SPLASH/SPLASH-2 [16] for multiprocessors. Although trends show a substantial increase in the market revenue for using high-end multiprocessor systems for database applications [12], performance evaluation methods have not been geared towards this application domain.

<sup>1</sup>SPARC is a trademark of SPARC International, Inc.

<sup>2</sup>Sun4m is a trademark of Sun Microsystems, Inc.

In Section 2.1, we give a brief introduction to database systems, while in Section 2.2, we point out the specific concerns that need to be taken when making performance evaluations of database systems. Finally in section 2.3, we list available methods that are used to do these evaluations and their shortcomings.

## 2.1 Database System Overview

A database system consists of a number of data tables, and often some relations between entries in them. Each entry often has multiple fields, such as name, social security number, etc. There are two common application domains of database systems, transaction processing (TP) or decision support (DSS). In a TP system, queries arrive to the database server requesting certain updates of database entries, for example a decrement of the number of parts in an inventory database. In a DSS, the database is mainly used in a read-only manner, and the queries instead ask for more complex compilations of information from multiple tables such as “What is the total value of all orders received after June 12 that have not yet been shipped?”.

In a client-server database management system (DBMS), a query arrives at the database server process, which often starts a new process to handle the query. Such a system typically can handle multiple different queries in parallel as different processes, i.e. *inter-query parallelism*, but some database handlers also support one query to be executed in parallel on multiple processors in a multiprocessor, i.e. *intra-query parallelism*. The fact that different processes potentially access the same database tables concurrently requires more or less complex locking mechanisms to guarantee data consistency.

## 2.2 Evaluation of Database Systems

In order to understand what happens when running database management systems on high-end (multiprocessor) computers, other considerations need to be taken than when evaluating scientific/engineering applications. The reason for this is that database system behavior differs from that of scientific/engineering applications in some major aspects [9]. They have

- much larger code and data sizes;
- a higher I/O intensity; and
- more interaction with the operating system

Consequently, a deeper knowledge is needed of what happens at all levels of the system. These levels include (in a coarse distinction): application, compiler, operating system, and hardware architecture with its I/O system, conceptually shown in Figure 1. Database system behavior is highly dependent on the properties of the processor, memory, and the I/O system for its operation. These components should therefore be included when making evaluations of these systems.

Let us review the differences between database systems, and scientific/engineering applications, as well as point out specific attributes of database behavior.

Scientific/engineering types of applications often have simple data structures with regular operations performed on them, with high temporal and spatial code/data locality.

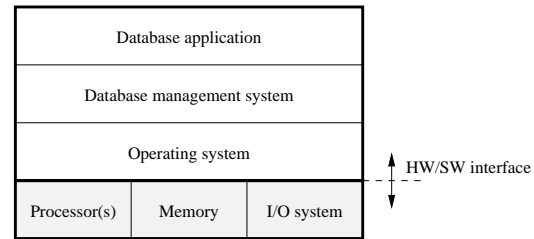


Figure 1: Conceptual overview of a database system

Databases on the other hand, have highly irregular data structures and more complex operations performed on the data, with less locality [9]. This ultimately leads to more conflicts in the memory system, and a detrimental impact on performance.

As mentioned, the performance of a DBMS largely depends on the performance of the I/O subsystem, both regarding virtual memory handling, and the granularity and organization of data structures, e.g. *records* in the database. As database operations spend much time reading and writing records to disk, these operations need to take into consideration how the data is actually laid out on disk and how the virtual memory is exploited to cache reused data and code. With disk operations being magnitudes slower than memory accesses, their influence on performance is high.

Inter-query parallelism and intra-query parallelism are fundamental performance increasing techniques, which puts higher demands on scheduling, and an increase in synchronization usage on the operating system. These techniques can thereby not be studied with a system model that lacks ability to include them. Especially with multiprocessor architectures, these techniques have crucial influence on performance.

In summary, the evaluation methodology used when identifying performance bottlenecks, or estimating gains from architectural modifications on performance for an application, needs to model the system in high detail and include the operating system as well as the underlying hardware architecture. It should also be able to relate events in the source code of the application and operating system to effects in the architecture, and vice versa.

## 2.3 Performance Evaluation Approaches

The methodologies used today in evaluating the performance of database applications varies heavily. All methods presented, conceptually belongs to one of a few basic techniques for performance evaluation.

**Evaluating a real system** – By the use of hardware probing or rudimentary instrumentation of object code, conclusions of the running application can be drawn [1, 3]. Although this approach has the benefit of being highly realistic, it yields poor instrumentation capabilities, and it is confined to be used on existing architectures. It also has low observability in terms of relating system events to application source code.

**Analytical modeling** – With a statistical/probabilistic approach based on rough estimations on the memory system, execution times, etc., dynamic effects such as sharing of data, contention at synchronization variables, hit-ratios and con-

flicts in the cache hierarchy, are very hard to contemplate. Even if valid assumptions can be made on several important system aspects and parameters, modeling their combined behavior is difficult. This method is subsequently restricted to confined aspects of the application and not the overall system performance. An example of this approach can be found in [4], where the performance of parallel join algorithms in a memory mapped I/O environment are evaluated using quantitative analytical models.

**Trace-driven simulation** – An existing system is probed when executing a particular workload. The memory references extracted are used in a simulator that allows for manipulations on architectural features such as memory organization, processor design, etc. While this simulation technique has been successful in describing the behavior of complex workloads [9], it exhibits some essential drawbacks. As the trace is produced on an existing system, no conceptual modifications can be made to that system, without jeopardizing the validity of the trace. Neither can any useful insights be drawn on internal system behavior due to the inherent “black-box” view of the system that this method provides.

**Instruction-set simulation** – This method is based on the concept of letting binaries, possibly modified, run on a simulator that produces the same effect on a system state that the binary would, being executed on a real architecture. Traditionally, instruction-set simulators have been unable to provide enough system realism to allow for system-level code (i.e. operating systems), and effects of hardware interactions [14]. The SimOS [10, 11] platform from Stanford, as well as SIMICS/sun4m [5–7, 15] platform from Chalmers/SICS, model a hardware platform in enough detail to execute an operating system with arbitrary workloads. However, SimOS lacks the ability to run completely unmodified operating system binaries because of its simplified devices, while restricted to a proprietary operating system (Irix) with publicly unavailable source code.

As we are interested in an evaluation methodology that supports modifications of the underlying hardware as well as software, instruction-set simulation is the most viable alternative. Additionally, binary compatibility, i.e. capability to run completely unmodified binaries on the platform, motivates the usage of SIMICS/sun4m, which we describe in the next section.

### 3 Simulation Platform

The SIMICS/sun4m platform, depicted by the shaded area in Figure 2, is an instruction-set simulator environment, capable of executing unmodified binaries of operating systems as well as applications, as it would on target hardware. In this section we describe how SIMICS/sun4m is constructed, as well as give some examples of its applicability.

SIMICS/sun4m 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. A more in-depth description of the SIMICS/sun4m simulator can be found in [7].

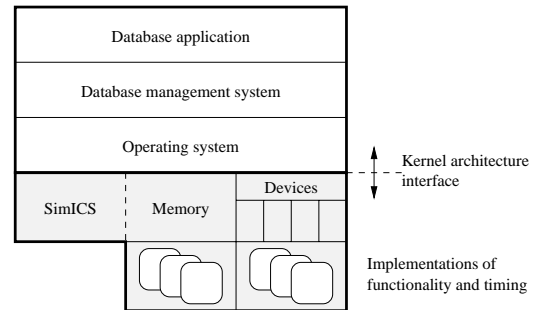


Figure 2: The SIMICS/sun4m simulator platform

#### 3.1 SIMICS

SIMICS is a highly optimized, program-driven, SPARC V8 instruction-set architecture simulator, supporting multiple processors, multiple physical address spaces, and system-level code. It extends the design principles of fast instruction-set simulators that make use of threaded code and/or native code generation [2, 5]. 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. Additionally, SIMICS can perform extensive profiling and debugging of the running workload, as shown in [8] for application and architecture, although not including operating system interaction.

#### 3.2 Kernel Architecture Support

Extensions to SIMICS in the form of devices, provide a hardware/software interface equivalent to their hardware counterparts. SIMICS provides generic support for passing memory operations to the correct device, which in turn implements the required functionality. A *kernel architecture* is the minimal collection of such devices, together with an instruction-set architecture and a basic memory model, that provides binary compatibility with an existing architecture, e.g. the sun4m architecture, allowing unmodified system binaries to be run as they would on target hardware. Currently, the kernel architecture of SIMICS/sun4m provides essentially complete sun4m functionality; SCSI, console, interrupt, timers, EEPROM, and Ethernet devices.

#### 3.3 Memory System Simulators

SIMICS implements the basic memory model of the kernel architecture, such as handling interleaving of memory operations. The memory system simulator is not part of the kernel architecture, as it is not visible by any executed code. Instead, it provides the processors with a latency to be associated with any memory operation. This latency depends on the organization and characteristics of the simulated memory hierarchy, which can have arbitrary instrumentation.

The memory system simulator is fed with every memory reference stemming from the executed program(s), resulting in a possible processor stall at the granularity of one instruction. A memory reference is represented by a data structure containing detailed context information about the operation, e.g. the physical as well as the virtual address of the operation, what processor and what process issued the reference,

is it a read, write, or synchronization operation, etc. This allows collection of statistics and detailed profiling of the running workload.

As long as we uphold the kernel architecture properties of the simulator, we have total freedom to modify the underlying functional and temporal implementations of the memory system as well as devices, i.e. we can evaluate different system characteristics in terms of timing and behavior. This will be further described in the next section. These implementations are symbolically depicted in the bottom row of Figure 2.

In the next section we present how we use SIMICS/sun4m for hardware and software evaluations.

## 4 Using SIMICS/sun4m

We will in this section illustrate how SIMICS/sun4m can be used for the following tasks; (i) with a given hardware architecture, support software development and application performance tuning, (ii) for a given application, enable evaluations of performance increasing architectural modifications, and finally (iii) being a platform for performance increasing measures of both the hardware architecture and the software system. This will be shown with some simplified, easily understood examples.

### 4.1 Evaluating Software Design

It is of great importance to understand how applications should be constructed to make the most use of hardware resources such as processors and memory. Larger modifications to existing applications with the purpose of enhancing performance, i.e. *performance tuning*, are often impaired by insufficient understanding of the total outcome of such modifications.

An example of a situation where the overall behavior of an application is difficult to contemplate, is the effect of *false sharing*, arising in a multiprocessor. False sharing occurs when e.g. a data structure contains elements which are individually private to a processor, but located within the same cache block. This phenomenon could lead to unnecessary cache block invalidations and a potential performance problem.

With SIMICS/sun4m, statistics on cache behavior can identify false sharing behavior and also, due to the fact that both physical and virtual addresses are known, show which data structures in the source code that are involved. An application programmer can use this information for restructuring or rewriting the application in a manner that reduces the amount of false sharing.

Here, we present another example of software development support where a performance bottleneck in a database system is identified. When measuring hit ratios for first and second level caches with PostgreSQL [13] running query #6 from the TPC-D benchmark suite on Linux 2.0.18, using a database scaling factor of 0.01, 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. Although this example showed an inherent attribute of the application, it still pointed out the important aspect of identifying performance problems.

The analysis above clearly shows how powerful the SIMICS/sun4m simulation environment is as a tool for identifying application or operating system bottlenecks. A clear line of events could be drawn from the initial discovery of high conflict miss ratios in the second level cache, to a line of source code in the operating system kernel and further to where it is used in the application program. It remains to be shown what can be done to remedy this performance bottleneck.

### 4.2 Evaluating Hardware Design

In this section we give an example of how SIMICS/sun4m for a given application, enables evaluations of performance increasing architectural modifications. The same methodology can be used for any commercial workload in the form of unmodified binaries, including operating systems, even without available source code.

In our example, we modified the cache sizes to reflect different memory system configurations. We repeated the database query for a range of different sizes of L1 and L2 caches and we found that the problem persisted for a range of cache sizes. The miss ratios ranged from approximately 50% for a 64 kbyte L2 cache, to 3.5% for a 2 Mbyte L2 cache.

The experiment was done using SIMICS/sun4m in a uniprocessor configuration. When evaluating a multiprocessor architecture with a more complex memory organization, additional issues would be of interest. These include the cache coherence protocol for keeping multiple cached copies of data consistent, mechanisms for moving large quantities of data, and interconnection network parameters.

### 4.3 Modifying the Hardware/Software Interface

To increase the performance of a system, modifications to the hardware/software interface is sometimes of importance. An example is *software prefetching*, where new instructions to fetch data prior to its usage are added to the architecture. To evaluate a system employing e.g. software prefetching, we need a methodology supporting both modifications of hardware and the corresponding changes in software to use the feature. Next, we describe how SIMICS/sun4m was utilized to perform this type of modifications to our system example.

After having identified high conflict miss ratios as a possible performance problem in the previous section, we sought a means to limit its impact on the performance of the application. We therefore introduced a feature in the architecture which lets a memory operation select whether to bypass the second level cache or not. Subsequently, we modified our application to consider this alternative and evaluated the resulting effects on the second-level cache miss ratios.

To detect the effect of not caching the cached-once blocks

originating from the sequential scan, we measured miss ratios in the L2 cache before and after the modification to the application that exploited the possibility of bypassing the second level cache. With the use of special instructions, inserted at appropriate points in code that performed the sequential scan, measurements were made with all other parameters fixed. 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. A valuable insight of this, is that by using selective caching during sequential scan operations, the amount of conflict misses can be reduced.

This example shows the strength of SIMICS/sun4m when it comes to modifications in the hardware/software interface, a very important property for evaluating new architectural features and for understanding how the applications can benefit from them.

In summary, we have shown the potential of SIMICS/sun4m as 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.

## 5 Conclusions

In this paper we have described how to use the SIMICS/sun4m platform as a complete system simulation environment for commercial workloads. SIMICS/sun4m consists of the instruction-set simulator SIMICS, a sun4m kernel architecture simulator, and a set of memory system simulators. Combined, these simulators constitute a platform which we intend to use for performance evaluation purposes, with the main focus being commercial workloads, e.g. database applications.

SIMICS/sun4m is currently able to run completely unmodified binaries, developed for the SPARC/sun4m hardware architecture. This includes the Linux 2.x and Solaris 2.6 operating systems, which are booted directly from dumps of real disk partitions, exactly in the same manner as they would on real hardware.

Additionally, we have shown how SIMICS/sun4m; (i) for a given hardware architecture, supports software development and application performance tuning, (ii) for a given application, enables evaluations of performance increasing architectural modifications, and also (iii) is a platform for performance increasing measures of both the hardware architecture and the software system.

## 6 Acknowledgements

We would like to thank Fredrik Larsson, Andreas Moestedt, and Bengt Werner at SICS, Fredrik Lundholm at Chalmers, and Håkan Grahn at the University of Karlskrona/Ronneby, for their contributions to the work presented in this paper. This work was supported by Sun Microsystems and the Swedish National Board for Industrial and Technical Development (NUTEK) under grants 96-04044 and 96-11255.

## References

- [1] J. Anderson, L. Berc, J. Dean, S. Ghemawat, M. Henzinger, S.-T. Leung, R. Sites, M. Vandevoorde, C. Waldspurger, and W. Weihl. Continuous Profiling: Where Have All the Cycles Gone? In *Proc. of the 16th Symp. on Operating System Principles*, October 1997.
- [2] R. Bedichek. Some Efficient Architecture Simulation Techniques. In *Proc. of Winter '90 USENIX Conf.*, pp. 53–63, January 1990.
- [3] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Parallel Processing of Spatial Joins Using R-trees. In *Proc. of the 12th IEEE Int. Conf. on Data Engineering*, pp. 258–265, February 1997.
- [4] P. Buhr, A. Goel, N. Nishimura, and P. Ragde. Parallel Pointer-Based Join Algorithms in Memory-Mapped Environments. In *Proc. of the 12th IEEE Int. Conf. on Data Engineering*, pp. 266–275, February 1997.
- [5] P. Magnusson. Simulation of Parallel Hardware. In *Proc. of MASCOTS'97*, January 1993.
- [6] P. Magnusson and B. Werner. Efficient Memory Simulation in SIMICS. In *Proc. of the 28th Annual Simulation Symposium*, April 1995.
- [7] P. Magnusson, F. Dahlgren, H. Grahn, M. Karlsson, F. Larsson, F. Lundholm, A. Moestedt, J. Nilsson, P. Stenström, and B. Werner. SIMICS/sun4m: A Virtual Workstation. In *Proc. of the 23rd Annual USENIX Technical Conference*.
- [8] J. Montelius and P. Magnusson. Using SIMICS to Evaluate the Penny System. In *Proc. of ILPS*, 1997.
- [9] A. Maynard, C. Donnelly, and B. Olszewski. Contrasting Characteristics and Cache Performance of Technical and Multi-User Commercial Workloads. In *Proc. of ASPLOS IV*, pp. 145–155, October 1994.
- [10] M. Rosenblum, E. Bugnion, S. Devine, and S. Herrod. Using the SimOS Machine Simulator to Study Complex Computer Systems. *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 1, pp. 78–103, January 1997.
- [11] M. Rosenblum, S. Herrod, E. Witchell, and A. Gupta. Complete Computer System Simulation: The SimOS Approach. *IEEE Parallel and Distributed Technology*, pp. 34–43, Fall 1995.
- [12] P. Stenström, E. Hagersten, D. Lilja, M. Martonosi, and M. Venugopal. Trends in Shared-Memory Multiprocessing. *IEEE Computer*, pp. 44–50, December 1997.
- [13] M. Stonebraker, L. Rowe, and M. Hirohama. The Implementation of POSTGRES. In *IEEE Transactions on Knowledge and Data Engineering*, vol.2, pp. 125–142, March 1990.
- [14] P. Trancoso, J. Larriba-Pey, Z. Zhang, and J. Torrellas. The Memory Performance of DSS Commercial Workloads in Shared-Memory Multiprocessors. In *Proc. of HPCA-3*, February 1997.
- [15] B. Werner and P. Magnusson. A Hybrid Simulation Approach Enabling Performance Characterization of Large Software Systems. In *Proc. of MASCOTS*, 1997.
- [16] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proc. of the 22nd Annual Symposium on Computer Architecture*, pp. 24–36, July 1995.