Rochester Institute of Technology
RIT Digital Institutional Repository

Theses

1989

Performance issues in mid-sized relational database machines

Larry Sullivan

Follow this and additional works at: https://repository.rit.edu/theses

Recommended Citation
Sullivan, Larry, "Performance issues in mid-sized relational database machines" (1989). Thesis. Rochester Institute of Technology. Accessed from

This Thesis is brought to you for free and open access by the RIT Libraries. For more information, please contact repository@rit.edu.

Rochester Institute of Technology
School of Computer Science and Technology

Performance Issues in Mid-Sized Relational Database Machines

by
Larry Sullivan

A thesis, submitted to
The Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Henry A. Etlinger
Thesis Committee Chairperson

Jeff Lasky

Stanislaw Radziszowski

February 14, 1989

Title of Thesis: Performance Issues in Mid-Sized Relational Database Machines

I Larry Sullivan hereby grant permission to the Wallace Memorial Library, of R.I.T., to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

February 14, 1989

TABLE OF CONTENTS

PERFORMANCE ISSUES IN MID-SIZED DATABASE MACHINES

By Larry Sullivan

ABSTRACT

Relational database systems have provided end users and application programmers with an improved working environment over older hierarchial and networked database systems. End users now use interactive query languages to inspect and manage their data. And application programs are easier to write and maintain due to the separation of physical data storage information from the application program itself. These and other benefits do not come without a price however. System resource consumption has long been the perceived problem with relational systems. The additional resource demands usually force computing sites to upgrade existing systems or add additional facilities. One method of protecting the current investment in systems is to use specialized hardware designed specifically for relational database processing. 'Database Machines' provide that alternative. Since the commercial introduction of database machines in the early 1980's, both software and hardware vendors of relational database systems have claimed superior performance over competing products. Without a *STANDARD* performance measurement technique, the database user community has been flooded with benchmarks and claims from vendors which are immediately discarded by some competitors as being biased towards a particular system design.

This thesis discusses the issues of relational database performance measurement with an emphasis on database machines, however; these performance issues are applicable to *both* hardware and software systems. A discussion of hardware design, performance metrics, software and database design is included. Also provided are recommended guidelines to use in evaluating relational database systems in lieu of a standard benchmark methodology.

1.0. Objective

The objective of this research is to describe and evaluate the performance benefits of currently available back-end database machines operating in conjunction with VAX/780 class processors.

1.2.0. Introduction

Database management systems (DBMS) have been commercially available for over a decade. These systems were developed out of the need to manage the information explosion that resulted from industrial and academic utilization of computer systems during the sixties. As the popularity of these systems grew and the sizes of the databases increased, an enormous amount of effort was expended to maintain application software which process the information contained in these databases. This was one shortcomming of DBMS which led to the development of the relational database model first proposed by E.F. Codd in 1970 [1]. Several relational database systems were developed in the early 1970's and were fairly well received. Users soon found however, that for complex queries on large databases, the system's response time and general performance degraded dramatically. A proposed solution to these performance problems was to implement many of the time consuming software functions in hardware. Such specialized hardware came to be known as a "database machine".

1.2.1. Database Machines

Before discussing what a database machine is, an understanding of what it is not is helpful. A database machine is not a stand-alone, self-sufficient computer system. It has no language compilers, so in this context, does not "run" programs. It does not support direct terminal I/O with a "dumb" terminal nor does it contain a conventional operating system to support general functions such as editing and printing/plotting. A database machine is a specialized hardware system designed specifically for relational database processing. It's primary function is to off-load database management overhead from a host computer and efficiently execute database commands quickly in hardware. The overall goal of this solution is to independently perform all database functions and recover host system resources, thereby substantially improving host performance.

There are five classifications of database machines. These classifications focus on the specific hardware strategy used. All strategies involve communications to and from a host computer so all designs include communications support. Some systems support a variety of communication strategies to promote flexibility and host independence. All strategies place the database machine between the host computer and the database storage facility. The time-consuming task of reading large amounts of unnecessary data to the host computer and subsequently locating the pertinent data to satisfy a query is performed by the database machine. High speed searching and

sorting of data is performed in hardware by the database machine. Only the required data to satisfy the query is returned to the host computer and to the user.

Most database machine designs support security, data integrity, concurrency and backup apart from the host computer. This further off-loads the host system of database administrative tasks and returns host resources for use by other computing jobs.

1.2.2. Relational Database Processing

The adoption of Codd's Relational Model was the result of the computing communities' attempt to resolve three major concerns surrounding non-relational DBMS [2]. First, data independence was provided in the relational model which freed the applications programmer from data storage issues. Non-relational systems required that data be retrieved through specific access paths. The relational system removed this requirement by providing a logical to physical mapping of data on behalf of the programmer. Second, error-prone code to implement looping or recursive processing was required to manage blocks of data in a non-relational system. By providing high level language concepts in the relational system, whole blocks of related data (or tables) were manipulated thereby replacing the iterative loop processing. Third, a query capability was provided for use by the end user. By providing easy, direct interaction with the database, database management systems became more productive

by removing delays in access program development and by promoting wide-spread use.

A price is paid, however, for the more flexible relational database. Data is now stored in discrete tables (relations) and an attempt to locate and manipulate data within these tables generally results in degraded system performance. The continual increase in the price/performance ratio of hardware together with the development of more efficient relational processing algorithms has resulted in decreasing processing costs per transaction. Although these cost improvements have been substantial, the increased usage of relational database systems and the dramatically increasing size of application databases have prolonged and emphasized the performance issues related to system overhead and response [3]. Database managers are finding that as the size and usage of the database increases, additional hardware resources are required. Also, as users become more familiar with databases, the level of user interest and sophistication also increases. This then imposes a monotonically increasing demand on system resources. When the system response degrades to an unacceptable level, the Database Administrator or Data Processing Manager must respond to improve the system's capabilities.

1.2.3. Cost of Expansion

Most large computer systems are designed so that they may be expanded in terms of main memory (to some addressable

maximum), disk capacity and in some cases CPU upgrades through the addition of co-processors. These options are usually available to the Database Administrator, but may not always resolve the problem which is responsible for database processing performance degradation. For example, if database activity has saturated the disk I/O channels (ie. bottleneck because of data transfer rates) then the addition of more main memory will do little to improve performance, (assuming, of course, that the maximum memory capacity is not on the same order of magnitude as the database size). The point here is that there will come a time when a system is at its hardware maximum in terms of main memory, I/O channels etc. At this time either a new and probably compatible machine is purchased or a back-end processor can be acquired to protect the substantial investment in the existing hardware. Which solution to be adopted is highly dependent on the particular situation. Examples of specific situations will be included in the thesis.

1.2.4. Value of Database Machine Technology

Database machine technology represents the state-of-the-art in database systems methodology. Most database machines are designed to support the more recently developed relational database model. This is one reason why the database machine has been slow in gaining acceptance in the user community. Existing non-relational database installations represent substantial amounts of invested

capital and labor resources. There currently exists no effective method of transforming both the existing database structures and application programs, which access the database, into a relational configuration. In effect, these databases would have to be re-created and all access software re-written to utilize the database machine solution.

In new database installations, however,the database machine offers an attractive solution for the anticipated additional workload. In those cases where support of a database application on existing systems would degrade system performance to an unacceptable level, the database machine represents a cost/performance solution. The relational model, considered by many to possess desirable characteristics in database processing, is gaining universal acceptance as the model of choice in new installations. Generally, the database machine supports this model and is cost competitive to the software database management solution.

1.2.5. First Commercial Machine

By the late 1970's, VLSI improvements, software developments, improvements to the relational model and the persistent performance issues, prompted David Britton and Geoffrey Lee to begin work on a dedicated back-end computer system [4]. Encouraged by continued wide-spread interest in relational database technology, Britton Lee Inc. introduced, in 1981, the IDM/500. The design of the IDM (Intelligent Database Machine) is modular in the sense that it is easily

configured to match the specific application being addressed. It is not unlike a basic minicomputer minus the CPU and all other components not required for a database management system. In place of these standard components are highly specialized hardware modules designed specifically to support the execution of relational database processing.

Before founding Britton Lee in 1978, Geoffrey Lee and David Britton marketed the industry's first 8-inch Winchester disk drive through the IMI company they founded in 1977. They subsequently established, for BASF, an 8-inch Winchester disk drive manufacturing plant in Germany. Being entrepreneurs and believing that the relational database model represented an area of growing interest and opportunity, they contacted the head of the Ingres project at the University of California at Berkeley. Through their discussions and research on the subject, they decided that a hardware and software system together, could solve many of the inherent problems in a relational system based on a software-only approach. In 1978, they founded Britton Lee Inc., in Los Gatos, California. Their company is generally credited with the introduction of the first commercial database machine (the IDM/500). Competition at the high-end of database applications has come in the form of the DBC/1012 machine, marketed by Teradata Corporation. This machine, available since 1984, is specifically designed for very large database applications and now controls this area of the market. Today, the database machine is being commonly recognized as a real alternative in

the relational database systems arena.

1.3.0. Database Machine Taxonomy

Reasearch in the study of database machine architecture has been proceeding since the early 1970's. The common goal of this research is to optimize the dedicated hardware design for the most efficient database functionality possible. This functionality is centered around three primary concerns. First, and foremost, is the recovery of general purpose computing capacity on the host computer by off-loading most of the database overhead to a dedicated, back-end computer. Second, efficient database algorithms for the sorting and searching of data may be implemented in hardware. Third, I/O methods may be used which are specifically designed for database management thereby improving on the data storage and retrieval bottleneck.

1.3.1. Five DBM Classifications

In the attempt to optimize the design of database machines, the following five classifications in the design of database machines have emerged [7].

1. Back-end computer dedicated to database management.

There are really two strategies that comprise this classification. The first strategy represents some of the earlier designs of database machine methodology. The goal here was to demonstrate that database management represented a

large, intensive computer application which warranted the use of a dedicated system. This was exactly how this classification was implemented. A general-purpose computer was linked between a host computer and the storage facility. This "back-end" computer ran the DBMS and performed all the database tasks, thereby relieving the host computer of this overhead. This strategy successfully demonstrated the advantage of dedicated, back-end systems but was not commercially adopted due to its high cost of implementation.

The second strategy is very similar to the first except that the general-purpose, back-end computer system was replaced with a tailored computer system, specifically designed for database management. This tailored system ran the database application which freed the host. Historically, as the price of hardware came down, this design was favored with many of the DBMS functions implemented directly in hardware. This strategy addressed both the performance issues as well as the cost problems which resulted in the quick abandonment of the first strategy.

2. Intelligent I/O controllers or associative processors.

This strategy grew out of the realization that a large percentage of the total time required to satisfy a query was used to locate (ie. read/write) data in disk storage. By doing high speed searches on behalf of the host, the net response time could be improved which also reduced the host computer workload. The design here was to place an

associative processor(s) between the host computer, running the DBMS, and the storage facility. This controller was dedicated to the task of logical/physical mapping of data used in the database. Several alternative designs of this controller were attempted. They ranged from a single microprocessor performing all the data mapping to a series of microprocessors each designated to an area of disk storage, all working together to satisfy a database query. The latter is also referred to as "parallel content-addressable" database machine design. The theory behind this design is straightforward. A supervisor microprocessor issues the same database request to all area microprocessors in parallel. The expectation was an increase in throughput since the area of the disk searched by each microprocessor was reduced as the number of area processors was increased. The result was sent to the supervisor which, in turn, transmited it back to the host.

3. Back-end computer plus associative processor.

This classification is really a combination of the first two classifications discussed thus far. This strategy placed a back-end computer and intelligent I/O controller between the host computer and the disk storage area. This design off-loads the DBMS task overhead to the dedicated, back-end computer and provides the required high-speed disk storage and retrieval operations through the associative processor. The benefits here are the same benefits attained through the first

two classifications.

4. Storage hierarchies capable of easily migrating data along the hierarchy.

In this strategy, a storage hierarchy system is appended to the host computer with the goal of minimizing disk I/O activity. This hierarchy is based at the mass storage level with slow access disks containing the large database. Above this level is faster, and more expensive, disk storage which holds more recently used pages of data. Above this is a disk cache memory which holds the disk pages most often used by the host computer running the DBMS. The host computer manages a standard memory hierarchy of core storage and processor cache. In this strategy, the response time is directly influenced by the management and amount of high-speed memory capacity; however, the DBMS overhead is still supported by the host computer.

5. Multiple-hosts with back-end DBM.

This classification is a culmination of the previous four classifications. In this strategy, the back-end database machine is placed between a network of hosts and a database storage facility. The back-end machine may be a general-purpose computer or a tailored machine. The hosts may be general-purpose systems running their own DBMS, a general-purpose machine and a back-end interface or even a smart terminal directly querying the database. The data

storage facility may utilize an intelligent I/O controller, contain a storage hierarchy system or simply contain the database on disk.

1.3.2. Design Implementations

The preceeding five classifications imply that actual database machine design might make use of the characteristics of one or more of the classifications. This is indeed evident in the following machines. First, Intel Corp., in the early 1980's introduced the SYSTEM 2000-FAST 3805 Data Base Assist Processor (DBAP). This associative processor is intended to be used between a host computer running the DBMS and the database storage facility. This system exemplifies a machine of classification type 2 (above). It is augmented, however, with a disk cache which is characteristic of a classification type 4 machine using a storage hierarchy system. The intelligent I/O controller emulates a standard IBM large disk except that through data mapping, provides faster access and transfer rates of data. All database inquiries are processed through the intelligent I/O controller prior to any actual disk access. The most recently used pages of data are maintained in the disk cache and the access time to these pages is improved by several orders of magnitude.

The Britton Lee IDM/500 is a back-end computer specifically designed for DBMS applications and runs between a host computer and the database storage facility. This design is an expample of a type 1 classification. The IDM/500

design, however, includes a disk cache which is used to store the most recently used pages of data for future use. This feature again exemplifies a characteristic of a type 4 classification.

A machine called RAP [8] developed in 1975, at the University of Toronto, is another example of a type 2 classification machine. RAP (Relational Associative Processor) consists of a hardwired controller tied to a chain of cells. Each cell is comprised of a processor and block addressable memory. The cells are responsible for searching and sorting data in the database and, through the controller, return the required data to the host computer. All operations are implemented in a very low level language. In 1978, an enhanced RAP architecture was constructed [9]. In place of the hardwired controller, a D.E.C. PDP-11 was incorporated to redistribute the workload from the cells to the controller. This changed the classification of the RAP machine from type 2 to type 3. The use of the PDP-11 raised the level of interface between the database and the host computer. It also provided a means to decouple the timing between the chain of cells from each other and the controller.

A slightly different concept in database machines is found in the SABRE system (Systeme d' Accese a des Bases Relationneles) [10]. This system is designed to be portable to any host system and is implemented in Pascal. The system is designed to be connected to multiple host processors which

represents a type 5 implementation. The previously discussed database machines all used hardware to implement database processing functions. SABRE implements these same functions in software by creating "virtual processors" which perform the required tasks. Parallelism is supported by decomposing each request into a set of functional steps. Each step is then executed by a virtual processor or processors collaborating together such as in the case of performing a join. SABRE is very close to being catagorized as a software relational database system since it is implemented in software. It is included in this discussion, however, since its implementation simulates an architecture which is based on the concepts of a hardware implementation. It is evident that current and future database machine design will be influenced by the performance/cost balance of features from all the classifications discussed thus far.

1.4.0. Addressing the Problem

Support of a relational processing capability entirely in software constitutes the primary cause of performance degradation on conventional computers. By implementing optimized pattern matching and sorting operations in firmware on dedicated processors this performance concern is addressed. Database machines rely on limited support from the host machine. Typically, the host is required to run a user interface or query language as well as support the user terminal I/O. The maintenance of the database,(ie.

error-checking, security and integrity issues, data retrieval and storage) is supported entirely by the database machine. The ability to interface with minimal effort to virtually any host system or network and operate as a back-end system appears attractive to the database management community.

1.4.1. Available Solutions

In the same way that general purpose computers are available in many different architectures to solve various computing needs, so there are different architectures of dedicated database machines. Databases exist in sizes ranging from very large (gigabytes) down to quite small (kilobytes). To process and manage these databases, it is quite reasonable to expect that there are available systems designed to accommodate these various applications. Designed for very large database applications, Teradata Inc., of Los Angeles CA., markets the DBC/1012. The theoretical limit of the DBC/1012 is a one tera byte (trillion) sized database. At the time of this writing, one of the largest database applications managed by the DBC/1012 is at Citibank N.A., New York. Their 70 Gbyte database application required more capacity than the largest available host configuration could provide (partly due to speed requirements). The DBC/1012 is required to access, in parallel, 136 515-Mbyte Winchester disk drives.

Designed for the very small database application, Cogent Data Technologies, Bellevue, Washington, in late 1983 announced a $1500 database machine for personal computers.

Actually, the machine is a replacement card for the fixed disk controller and is accessed in the same way as if used as a file-server on a network. Cogent claims that the machine is much more than a file-server since it supports concurrent requests from multiple users, record-level locking, accesses 1.5 Mbytes/sec, supports distributed database activity and implements database query primitives in hardware. Internally, the machine utilizes the Intel 80286 micro-processor in combination with Western Digital's fast 1010 hard-disk controller chip which provides adequate speed to read an entire track of data in one disk revolution. Cogent feels that much of the product success will be related to this high speed I/O [5].

Supporting the medium sized database application, Britton-Lee Inc, Los Gatos CA., pioneered the database machine market place with the IDM/500 series product line. This product was the first commercially available machine designed solely to operate as a back-end relational database computer. Since 1981, the IDM product line has evolved into three models plus a low-end Relational Server (RS300) series targeted for the integrated office environment. The IDM series begins with the 500X system configured with 2 Mbyte RAM, 680 Mbyte (2*y*340 Mbyte) mirrored disk capacity, a 500 Mbyte tape drive, the standard 500/1 database processor and standard disk controller. The mirrored disk design provides immediate database redundancy by maintaining two identical images on separate storage devices. If one database image becomes

corrupted, the other is used until the corrupted database can be corrected [6]. This system, and the upgraded systems, provide backplane expansion slots for a mix of communication and disk storage interfaces.

1.5.0. Performance Analysis

Much controversy has surrounded the issues in database machine performance. The technology is new and varied enough that no accepted metrics or measurement methods have been adopted as standard by the computing community. Many single-user and multiple-user benchmark studies have been performed on database machines, but the performance issues here are quite different from the well understood issues that are important on general computing systems. For example, most sophisticated operating systems provide utilities which permit monitoring and tuning of system resources. These resources include the CPU time consumed by a program, the memory space required by the program, the number of disk files used and number of I/O operations, the number of lines printed and many more [7]. The operating system usually provides adjustable parameters which allow a system administrator to alter the allocation of these resources to optimize the system utilization and performance. This is a delicate "give-and-take" balancing operation, but one which is supported by the operating system. This is not as true with database machines. While the host machine may schedule batch jobs at different times or assign a different priority level

to a user's process, little control is available over the database machine. One may improve disk I/O by adding more disks or take advantage of improved parallelism by increasing the number of interfaces to multiple hosts, but this is increasing capacity, not explicitly tuning existing capacity.

1.5.1. Database Machine Performance

Database machines are designed for a highly specific function. On the database machine, most database processing functions are implemented in hardware which can neither be accessed or changed by the machine user. The "operating system" on the database machine is also highly specialized and generally lacks the resource allocation utilities which are available on general computing systems. Since little control is available to the machine user over resource management, the machine designers are responsible for the most effective design to support any possible workload. It is the design of the database system workload and the monitoring of system response and resource utilization metrics which govern database machine performance analysis. And since system tuning is not an option with the database machine, most machine benchmark studies are really a comparison of results with software database management systems and other database machines.

1.6.0. Open Issues

Being a new technology, there are many questions

surrounding database machines. These questions address the claimed performance benefits as well as other pertinent issues.

Performance Issues

- What benchmark tests were used to evaluate the performance benefits of a DBM vs general purpose DBMS?
- Why haven't standard methods to evaluate database systems been developed?
- What current work in performance analysis methodologies is being conducted?
- How are operational issues handled (backup, recovery, integrity, concurrency, security)?
- If processor designs improve, will disk I/O ever become the constraint on performance (which DBM's have no control over)?

Other Related Issues

- How reliable have the systems been?
- Are there known bugs in the existing systems?
- Since no industry giants are marketing their own DBMs (yet), is there a commitment to the long-term client?
- If bad database designs can effect performance, how does one guarantee an optimum design?
- Are current users satisfied?

1.7.0. Thesis Plan

This thesis research is directed towards the issues

surrounding the performance benefits of database machine technology. Chapter 1 discusses the value of the relational model and the subsequent problems which emerged with its implementation. Some possible solutions to these problems are presented in the form of database machines. The classifications of such machines are discussed and several real hardware implementations are briefly reviewed. Several open issues surrounding the current database machine technology are then listed.

Chapter 2 is a detailed discussion of one commercial database machine. The IDM (Intelligent Database Machine) first marketed by Britton Lee Inc, Los Gatos California, is presented. The machine's unique hardware architecture is discussed followed by a description of the software which is required to run the relational database system. This includes both the host software (the user interface) and the IDM software performing the database processing.

The paper continues with a detailed description of performance measurement and analysis in chapters 3 and 4. The studies here focus on the IDM system as compared to traditional software systems and other database machines. Chapter 5 then provides a summary of the research and lessons learned as a result of that research.

Chapter 2

The IDM/500 Hardware and Software System

This chapter is a detailed discussion of one particular database machine system. The IDM (Intelligent Database Machine) marketed by Britton Lee Inc., is presented. The machine's hardware architecture is discussed followed by a description of the IDM relational software system. This includes both the host software system supporting the user as well as the IDM machine software performing the actual database processing.

2.1.0. The IDM Hardware

Several different models make up the Britton Lee family of database machines. Three models, in particular, are targeted for the VAX size system application. These models are similar in design and are fully upward and downward compatible as well as field upgradable. Designed for the small application (less than 50 users), the IDM/500X is configured with 2 Mbytes of RAM, 2 C.D.C. 340 Mbyte disk drives, implemented as mirrored disks (with SMD controller),a 500 Mbyte cartridge tape drive (used for data archival) and the 500/1 database processor.

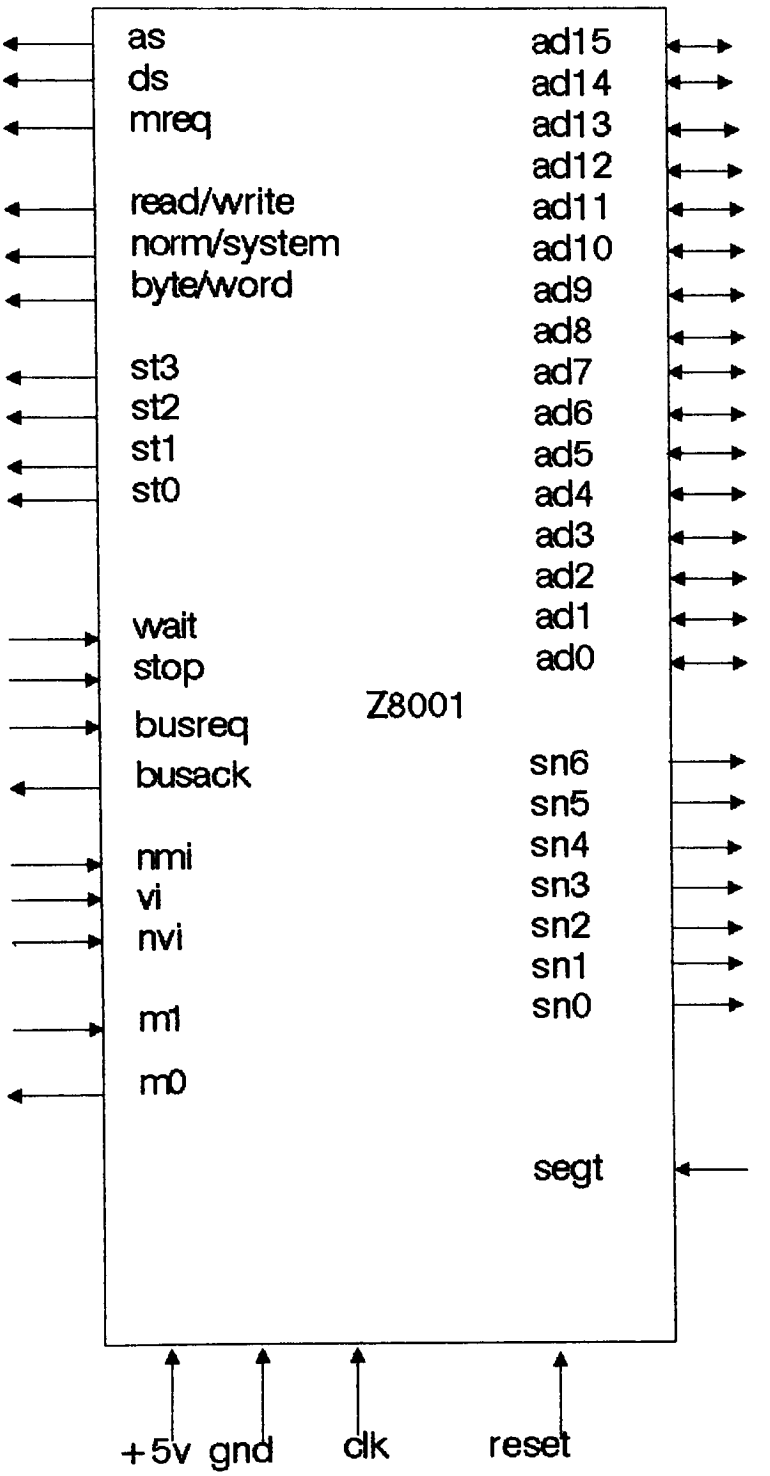
Medium sized applications (50-300 users) are candidates for the IDM/500XL database machine. This machine is shipped with 3 Mbytes of RAM, 2 515 Mbyte Winchester disks (with SMDE controller) again, in a mirrored implementation, the 500 Mbyte cartridge tape unit and the 500/2 database processor. This system is advertised to offer twice the performance of the IDM/500X machine. At the high end application (300-400 + users), Britton Lee offers the IDM/500XLE database machine. Its base configuration has 4 Mbytes of main memory, 4 515 Mbyte Winchester drives (on 2 SMDE controllers), the 500 megabye tape drive and the 500/4 database processor. This system claims a 40 percent improvement in performance over the IDM/500XL machine [6].

A comment is needed here concerning the size of the database applications mentioned above. Obviously, many factors influence the "size" of a database system. Physical size of the database(s), system response requirements, redundancy, complexity of database queries, number of supported host computers, etc., all influence the "size" classification of a database application. All things considered, the number of users is offered simply as an approximation of the size of a database application. The author is not stating that this is the determining factor when designing a system to support a relational database system.

2.1.1. The Database Processor

At the heart of the database machine is the database processor. The database processor is responsible for managing all system resources as well as executing most of the software in the machine. System upgrade options are very modular in their design and the database processor first identifies all available system resources at startup and utilizes those resources during operation. The standard database processor, 500/1, is a 6 mhz processor and is centered around a Zilog Z8001 microprocessor. The Zilog Z8001 is a 48 pin device and can address a total of 8 Mbytes of storage (see figure 2.1). The Z8000 family of microprocessors contain sixteen 16-bit internal registers which can be addressed as 8-bit, 16-bit, 32-bit or 64-bit register pairs. All cooperating devices are connected to a high speed, parallel bus which multiplexes data and address information between the Z8001 and the device [11].

Britton Lee offers 2 options to the 500/1 processor. The 500/2 database processor, which comes with the IDM/500XL, is essentially the same database processor but with a high clock speed of 10 mhz. This is advertised to provide a doubling of system performance. The second option marketed by Britton Lee is the Database Accelerator or DAC. This slave processor is advertised to improve performance by a factor of 2 to 10 depending upon the type and load of database queries. It operates using a specialized, proprietary instruction set specifically designed for relational database operations. This high speed (8 MIPS) processor can search memory far more quickly than the database processor, but works under the



Zilog 8001 Microprocessor by Zilog Inc.

fig 2.1

direction of the database processor. The DAC and the 500/2 are packaged together and sold as a third processor type, the 500/4 database processor. The 500/4 comes standard with the IDM/500XLE machine.

2.1.2. IDM Cache Memory

Memory, on the IDM, is managed by the Memory Timing and Control (MTC) module. This board manages up to 6 Mbytes of RAM memory (in 1 Mbyte increments) and is pipelined to supply data at the processing speed of the DAC. The MTC provides error correction for data transfers while the main memory is used to hold the system software, the most frequently used database pages and stored database commands. To minimize data read/write delays due to disk I/O, a portion of the IDM main memory may be dedicated to disk caching. When a CPU request for disk data results in a "hit" from cache memory, the MTC satisfies this request in 1-2ms. If the disk must be accessed for the requested data, a typical response time is 20-25ms. The same is true for data written out to the database. The IDM standard page size is 2k bytes although this may be modified by the database administrator in 1k byte increments [12].

2.1.3. IDM Disk Controllers

Data is supplied to the IDM memory through the disk controller(s) from the disk(s). Two types of disk controllers

are available on the IDM system; one for SMD (Storage Module Drive) disks and one for Extended SMD (SMDE) disks. One controller can manage up to four disk drives and a maximum of four controllers (16 disks) may be configured. A mirrored disk design provides on-line database redundancy. When data is written out to one disk, the controller simultaneously writes the data to the second disk. Disk shadowing may also be implemented on the IDM through the addition of another disk controller. When data is written or read from the database, one controller instructs it's disk(s) head to begin the seek starting from the outside track while the other controller starts from the inside track. This still provides for database redundancy, but also reduces data retrieval time since two disk heads starting at different locations will locate the required data in one-half the time of a single disk head [13]. The IDM also provides for off-line data storage through its 500 Mbyte cartridge tape drive unit.

2.1.4. Host/IDM Communications

A significant advantage in database administration is realized in a centrally located database system. The IDM manages the entire database at one location thereby insuring data integrity and security while providing concurrent access to users on many different host computer systems. Host independence is accomplished through the appropriate configuration of the IDM's host interface module(s). This module (board) performs the host/IDM communication error

checking and transfers host commands into the IDM main memory for processing. Host interface modules are available with up to 64 RS-232 ports, 4 IEEE 488 channels, 4 Ethernet LANs or 4 IBM block multiplexers. Eight host interface modules may be configured into the IDM 500 to promote parallelism. Host operating system dependent software packages are available for almost any vendor's computer system. These multi-user, parallel drivers perform the translation of a user's request into IDM's internal format which is sent through the host interface module for processing. To support ad hoc queries, Britton Lee has developed IDL, a data query language, but SQL (developed by IBM) is also available.

2.2.0. Database Machine Software

Relational database systems running on database machines have a unique software structure to attain the benefits that database machines offer. The physical separation of the host computer and the database machine dictates that the software system have a design which is different from the more traditional software-only database management systems. The design differences are evident primarily in the lower levels of the database processing software which are hidden from the user's perspective. Not required in the software-only systems, additional software is needed to support the host to back-end communications which adds to the design complexity of the database machine software system. An additional processing delay is imposed as a result of this communications

overhead. However, if this delay is small when compared to the time savings achieved through the processing speed of the database machine, then the overall system response and performance may be improved.

The Briton Lee software system is made up of two distinct but cooperating software modules [14]. One module, the BLI (Britton Lee Interface) IDM, resides entirely on the database machine while the other module, the BLI Host software system, resides on the host computer. A third component, though not unique to database machines, is the user's interface. While extremely important to a successful database implementation, the user's interface does not impact the performance issues addressed in this paper.

2.2.1. Britton Lee Host Software

Residing on the host computer, the BLI Host software system is responsible for a variety of functions including the host to IDM communications and management of the user environment. The hardware architecture of the IDM series supports four different IDM to host communication protocols. Britton Lee offers a long list of communications drivers to run on the particular host/operating system combination required for the application [14]. These communication drivers are very important to the overall performance of the IDM system. They are required to efficiently transfer data and database requests between the host computer and the IDM.

To accomplish this, these drivers must properly interface to the operating system of the host computer, the communications hardware port on the host and to the IDM itself. Since the IDM may be configured with multiple host interfaces, the IDM is maintained as a host independent machine by keeping the required communications drivers in the BLI Host software on the host computer.

Another component of the BLI Host software is a run-time library which provides a set of tools to interface the user with the IDM. This library is composed of two major subsets of routines. The first set provides the program to IDM communication routines. These allow any user application program to pass data and commands directly to the IDM. Whether written in a high-level language such as SQL (Structured Query Language) or in a language more closely related to the internal command structure of the IDM (IDL - Interactive Database Language), the library offers an efficient and useful set of routines for program development. The second set of routines were developed by Britton Lee for user convenience. It quickly became apparent to the BLI software designers that many different database applications had mutual need for some common data manipulation and conversion functions. These functions, such as date/time stamping, data type transformation, error condition handling and others, are made available in the run-time library. Applications programmers may call the run-time library routines within programs written in the "C" language.

However, Britton Lee is developing a set of language interface programs to extend the run-time library capabilities to other programming languages running on the variety of host computer systems supported by the IDM series.

A method of supporting database queries without the need for writing applications programs is provided by the Interactive Session Monitor. This part of the BLI Host software system is designed to support ad hoc database queries in the command language (SQL or IDL). IDL is Britton Lee's query language designed specifically for the IDM system. The I.S.M. provides a full on-line help utility to the user and contains examples and explanations of user commands. The I.S.M. may also be used by the database administrator to manage database security, transaction audits and other management functions.

The database administrator has need of other database management functions and these are included in the Utilities module of the BLI Host software system. Included in this set of pre-written programs, are a variety of required operational tasks. These include file conversion routines from host structured files into IDM format, backup and recovery utilities and utilities to roll-back, re-create and re-start the database due to system problems such as disk failure or system crash.

The Britton Lee Host software system is designed to provide two primary functions. First, it provides the

necessary tools for a user and database administrator alike, to access and manage the database. Second, it provides the host dependent interface for a particular system to communicate with the IDM thereby maintaining host independence for the database machine. The latter function then, allows the database to be shared by multiple hosts, maintained at a single location and operated on by fast, highly specialized hardware and software to attain the processing speed benefit of the database machine.

2.2.2. Britton Lee IDM Software

Performing the actual database processing and management, the BLI IDM software resides entirely on the backend database machine. This software is responsible for the many tasks required to run a relational database system. The database machine receives parsed database commands from the host system to store or retrieve data from a database. In addition to handling the data storage and retrieval, the IDM software also maintains the database. Security, search path optimization, concurrency, managing views and indices, error handling, transaction logging and audit trails represent some of the database management tasks performed by the BLI IDM software.

The database administrator and the general user are provided with an extensive relational database command set, IDL [6]. Since the relational model is supported on the IDM, a database can be constructed as a group of one or more,

two-dimensional tables. To create a database, the user issues the "create database" command along with the database name. If permitted, the user can specify the amount of storage to be allocated to the new database or default to the first available space located by the IDM. Initially, the database is empty with the exception of several relations automatically generated by the system and maintained in the system data dictionary. The data dictionary relations contain information about all of the existing databases. This information includes the name and description of each table in the database, attribute information for each relation, index information for the database, user access priviliges and various other parameters.

Once created, the database must be explicity opened by a user to gain access. To do this, the "open" command is issued along with the database name. Once opened, the user may create relations for the database by issuing the "create table" command and specifying the table name, attribute list and attribute types. The IDM supports nine data types as well as conversion between data types. After the relation has been created, the user may use two different methods of putting data into the database. The first method provides for direct addition of data into the relation. Issuing the "insert into" command and specifying both the table name and list of attribute values causes the table to be modified on-line. The second method available to the user, is in the form of a bulk loading of data from a file. The user issues the "copy in"

command, specifying the table name and file name which causes the data in the file to be loaded into the table. The IDM software also provides for a method of writing out entire tables to files by issuing the "copy out" command.

To retrieve data from the database, the user issues the "select... from" combination, specifying the needed attribute(s) and the table(s) to search. The selection may be refined by augmenting the search with qualifier options. The IDM supports the comparison operators (ie. =. >, <, !=. etc) as well as the boolean operators and aggregates of both. This, in effect, provides the user with the relational operators "project", "select" and "join". Also supported with the select command are three text string manipulation functions. These are substring, concatenation and pattern matching. Selection functions such as "group by", "count", "order by", "MAX", and others are also supported by the IDM software system.

To modify the data within an existing database, the user would issue the "update" command specifying both the table name and the new value to be assigned to the specific attribute. Again, in specifying the precise attribute to be changed, the user may qualify the update command with the previously mentioned operators. To remove a specific record from the database, the user issues the "delete from" command. Here also, the user must specify the table name and identify which record(s) are to be deleted. To remove an entire table,

the user simply issues the "drop" command along with the table name.

All database manipulation commands may be issued interactively by the user or embedded into applications programs. Further, a repetitive group of commands may be defined to the IDM software system and called as a type of macro instruction. This group of instructions is pre-compiled and stored by the database machine. A call to this macro results in its immediate execution, thereby removing the overhead of repetitive parsing done by the host computer. Another benefit of using "stored commands" is in the reduction of the number of commands sent from the host to the IDM since it may be called by reference.

Operational functions are also provided as part of the IDM software system. These are included to permit accurate maintenance of the database as well as assist in developing efficient database designs. The user of a database may, at times, wish to review the results of a database operation before permanently altering the database with that operation. If the "set autocommit off" command is issued, all following operations are temporary, in the sense that the user has the option of aborting the alterations made to the database. If the user is satisfied with the temporary results, he may issue the "commit work" commmand to permanently update the database. If he is not, however, he may issue the "rollback work" command to ignore the temporary alterations made since the

last "set autocommit off" command.

To fully exploit the benefits of the IDM, the database user may elect to assist in data manipulation and management by providing indices into the database relations. The IDM supports both clustered and non-clustered indices. Issuing the "create clustered index" command and specifying the table name and attribute, causes the IDM to sort the table by the attribute and build a b* -tree index table that contains pointers to the memory locations of the indexed attribute range values. Indices may be removed by issuing the "drop clustered index" command (for clustered indices).

At times it becomes necessary for the database administrator to restrict the access of certain users from some of the data in the database. The use of "views" is a method to create a logical table with attributes from one or more physical tables. In this way, sensitive data may be "hidden" from a user without the user's cognizance. To create a view, the "create view" command along with the view name and table attributes making up the view must be issued. Another method of securing data in a database is through the granting and revoking of privileges to the user. The database administrator may grant or revoke read, write and execute privileges to a user as well as the creation and deletion of relations, indices and databases. These access privileges may be specified on a single attribute in a record or the entire database. Particular users or all users may be specified for

the privilege.

Just as important as securing the database is the integrity of the data. There are several methods within the IDM software system to validate data. Using "stored commands" to check the range and sign of data is one method to inspect data entered on-line. The "select distinct" command allows a user to create a new table from an old table in which duplicate entries may exist. Additionally, the "create distinct index" command can be used to ensure uniqueness of a key field.

Occasionally computer systems fail. This might be the result of a disk head crash (hard-failure) or a power surge, network failure, etc. (soft-failure). These problems must be expected and database integrity must be maintained. The IDM transaction log is the tool by which the database is recovered in the event of a system failure. Every command that is sent to the IDM for processing is considered to be one complete transaction unless the "auto commit off" command has been previously issued and no "commit work" command has yet been received. In this case, the entire group of commands is considered as one transaction. To maintain the consistency of the database, a transaction is run to completion or is not started. The transaction log holds sufficient information to return the database to it's previous condition in the event of a system failure. Enough change information is recorded in the transaction log to both reconstruct the change or to back

it out of the database. The IDM transaction log uses a signaling mechanism with the storage device to assure a successful update of the database. The transaction log is first written from the IDM's memory to disk before the transaction is committed from memory to the database. If the transaction is successfully written to the database, a "done" token is written to the transaction log. This guarantees that the update to the database has been performed to completion and the next transaction may be started.

In the event of a "soft-failure", the transaction logs will be held intact and the database may be recovered. After the IDM is rebooted, a recovery program executes which interrogates the transaction logs to see which transactions were only partially completed. These transactions are then backed-out of the database and re-issued for processing. To maintain its high speed, the updated transaction blocks are held in memory and written out in bulk at specific time intervals called "check points". The recovery program proceeds from the most recent checkpoint since all earlier transactions were completed. This scheme breaks down, however, in the event of a hard failure. When this happens, the transaction logs may be lost or corrupted. The IDM system encourages occasional backup of the entire database and frequent backups of the transaction log. If a hard failure occurs, the latest database dump is loaded and the transaction logs are rolled forward to the point when the failure occurred.

CHAPTER 3

Relational Database Performance Evaluation Methodology

The comparison of both software and hardware relational database systems' performance is of current interest to the computer science community. No standard comparison methodology has been adopted in this area of study. Current researchers in this field are attempting to establish a set of standardized benchmarks to permit the fair and accurate comparison of relational database systems. This task, to say the least, is difficult. Any developed benchmark test must be application independent and capable of running on any computer system supporting a database. The latter part of the previous statement is quite obvious. A database system must be executable on a computer or the comparison of the system is made impossible. But why must a benchmark be application independent? Researchers know that if their studies are based on a specific size of database, or type of database or mix of database queries, etc., the study will be criticized for losing generality and thus credibility. Satisfying this requirement however, is quite difficult. Well understood and accepted benchmarks exist for the more traditional "component" studies (ie. CPU speed, disk I/O bandwidth, etc.). These benchmarks meet the above requirements and are proven to provide a meaningful and fair comparison of component

performance. The problems arise when the benchmark attempts to fairly describe a "system" of components for comparison.

This chapter will discuss relational database performance measurement methodology. This will include workload considerations, query classifications and hardware implementation considerations. Also included in this chapter is a discussion of various database machine architectures and how specific application considerations effect machine design.

3.1.0. Database Workload Characterization

Many working parameters effect the performance of a database system. Most of the work in database machine benchmarking is dedicated to the identification and understanding of these parameters. The goal of many researchers in this field, is to find a set of tests and metrics which permit a general and unbiased comparison of relational systems. This can theoretically be accomplished if benchmarks can be developed while maintaining machine and application independence. This means that comparisons of database systems can be made without regard to hardware architectures or database structure or content. While this generality can be maintained, several decisions must be made concerning the scope of the benchmark study. The most fundamental decision in workload characterization is whether the study includes multiple-user or single-user job loads. Other decisions must address such issues as which performance indices are important to the study, whether to use a real or

artificial database, the use of dedicated or shared computing facilities with other non-database applications and many others.

3.1.1. Single vs. Mutiple User Benchmarks

A fundamental decision is made at the start of database benchmarking activity. This decision confines the study to single-user job loads or extends the scope to include the most general study of multiple-user job loads. There are benefits and penalties in the selection of either scope. Single-user studies provide the researcher with the most controlled environment possible when studying relational database systems. Inferences of performance based on observations of the system may be made with more confidence. Variations in the workload can be made with the subsequent effects made clearly available to the researcher. This makes deciphering of performance effects based on various queries possible since system parameters such as elapsed time, CPU utilization and I/O bottlenecks are directly available and attributable to the query. Much useful information has been collected about specific performance effects due to hardware configurations, operating system characteristics and database processing algorithms in single-user benchmark studies [15] [16].

Single-user benchmark studies provide a stable environment for the researcher but do they describe the performance of "real life" database applications? Probably not. Database systems are generally used in an application which must

support more than one concurrent user. If what is observed in a single-user environment may be translated and applied to the more general case of multiple-user systems, then the single-user case provides the most convenient arena for performance evaluation. This is not typically the case, however. Performance of multiple-user systems depends upon many interrelated parameters. Unlike single-user studies, multiple-user benchmarks must consider the varying degree of complexity in the query mix, consumption of system resources used by "background" jobs (ie. disk contention, processor queues), operating system differences (ie.process scheduling, priority) as well as the interpretation of the results. Single-user tests provide a direct cause and effect environment for performance analysis. Multiple-user tests confound this causal relationship since the performance effects are usually not directly attributable to one specific process.

If multiple-user tests are so difficult to interpret, why use them? The answer is obvious. Performance tests that do not evaluate a system in the environment that it is intended, are not credible. Currently, the bulk of performance benchmark research is being expended to design tests which are useful in multi-user systems and apply to all relational database systems.

3.1.2. Using A Real Or Constructed Database

Ideally, when one is working on the development of a standardized test, the use of some actual test data is preferred. The users of a standard test method would obviously prefer viewing the results of the benchmark based on their own specific data sets rather than those of some obscure and not understood data sets. This, like so many ideal situations is not practical. To maintain the requirement of "application independence," researchers purposely construct test databases which resemble no "real" application database at all. If users are permitted to use any random workload while running a benchmark, inaccurate results may be derived. For example, consider that a user's sample database is not designed to make use of non-clustered indices. Consider also, that the benchmark includes queries that test the database system for efficient use of non-clustered indices. One possible result that may be listed by the benchmark analysis is that the database system in question does not utilize the benefits of non-clustered indices! This type of inaccurate conclusion must be guarded against when using a standard benchmark. The developers of a benchmark, then, require that a "synthetic" database be created which accurately tests the capabilities of the system by providing valid statistics which can be used for comparison.

Many workload databases have been developed for various benchmark studies. One of the first and most notable is the Wisconsin Database proposed for single-user benchmarking [15]. Although this is one of the earliest synthetic databases

designed for relational database system comparisions, it proved to be quite elegant in design and served as a model for many other single and multiple user benchmark studies [17] [18] [19]. The Wisconsin Database is designed such that the relations and attributes are easily understood. Integers and character strings make up the data types of attributes which can be easily utilized as indices or control selectivity levels in aggregate computations.

The database consists of four fundamental relations [15]. These are named the "thoustup", "twothoustup", "fivethoustup" and "tenthoustup" relations. They contain 1000, 2000, 5000 and 10000 tuples respectively. These relations all contain the same tuple width when created. For example, the Ingres specification to create the thoustup relation follows:

create thoustup (uniquelA=i2, unique2A=i2, twoA=i2, fourA=i2, tenA=i2, twentyA=i2, hundredA=i2, thousandA=i2, twothousA=i2, fivethousA=i2, tenthousA=i2, odd 100A=i2, even 100A=i2, string U1A=c52, stringU2A=c52, string4A=c52).

This specification identifies 16 attributes for a 182 byte total tuple width. The first attribute, "uniquel", contains unique integer values for the entire relation. That is, for the thoustup relation (1000 tuples), uniquel has as its values, the integers 0,1,...999. The second attribute "unique2" has the same range as "uniquel" and so both are primary key candidates. While a random generator is used

during the creation of the relations, "unique2" is generally used for sorting operations. The next attributes are integer fields with varying ranges. For example, the "two" attribute contains only the values "0" and "1", the "hundred" attribute contains only the value 0...99, and so forth. These attributes, although randomly distributed, contain an even distribution of values. This provides for a convenient method of controlling the selectivity level in selection queries. This is demonstrated in the following SQL example.

```
select *
   from twothoustup
   where unique2B < 200;
```

In this example, the primary key "unique2B", which contains the values 0,1,...1999, satisfies the query for 10% (ie. 200) of the the 2000 tuples. In this way, it becomes convenient to measure the system's performance based on controlled variation in the amount of returned data.

The last three attributes in each relation are strings, each 52 bytes long. These strings vary only in that the first, middle and last character will contain the values "A", "B",...,"V" while the remaining 49 characters contain only "X". The first two strings, "stringU1" and "stringU2" contain unique values, so each may be used as a primary key candidate. The range of the character values include "A" thru "V" because this provides 22*22*22 = 10,648 unique strings which is larger than the cardinality of the largest relation (ie. 10,000). The last string "string4" contains only 4 unique values

(Axx..Axx..A, Hxx..Hxx..H, Oxx..Oxx..O and Vxx..Vxx..V). This string provides for different selectivity and partitioning similar to the integer attribute "four". As an example of a string used as a primary key, consider the following SQL example which selects 1% of the tuples from the "tenthoustup" relation.

```
select *
   from tenthoustup
   where stringuD > "Bxx...xxxGxxx...xxxE"
   and   stringuD < "Bxx...xxxLxxx...xxxA";
```

In these queries, system performance indices may be monitored to provide a comparison of systems in a single-user environment. What must be accomplished to extend the comparison to the more general multiple-user environment? Principly, the addition of a known, controlled background workload provides the extension. One method to provide this background job mix is to select a "type" of query set that provides the desired workload. Then, by issuing concurrent copies of the job mix, a controlled level of multi-programming is provided [20] [21]. The concurrent "users" can be simulated by using a terminal emulator [22]. This is very useful since strict control over the user processes is required in addition to the fact that some query sets may take hours to complete! Another performance issue arises when the system workload includes non-database related jobs sharing the host computer resources.

3.1.3. Performance Effects In A Shared Computing Environment

One of the advertised advantages of database machine performance is the indifference of the database machine to the host processor workload. Many non-database functions may be running on the host system while the database machine processes database queries independent of the other workload. This independence is not complete, however, since the database machine requires parsed database queries from the host as well as support for data I/O between the two systems.

While the database machine retains this separation from the host system workload, software DBMS systems can be quite influenced by this host system activity. This is an important consideration if comparative benchmark analysis is to be provided between active software and machine database systems. Proponents of software DBMS solutions might be quick to point out that the non-database host workload is independent from the real capabilities of the software DBMS system. Afterall, the performance statistics comparing a database machine and a software DBMS should not be clouded by extraneous and uncontrollable background noise. However, unless the software DBMS is actually used on a dedicated host (which turns it into a database machine), the measurement of non-database activity and its effect on the software DBMS could prove to be very informative. For instance, consider the database installation which shares system resources with a scientific computing application. In this case, the prospective database user would be quite interested in performance statistics of the

software DBMS while a CPU intensive background workload was executing. Similarly, a host system supporting a large file server activity requiring heavy I/O service may provide interesting performance statistics while concurrently supporting the software DBMS. While this discussion crosses the line into "application dependent" benchmarking, the questions that arise are very pertinent to actual database system performance.

3.2.0. Query Classifications

A strategy to create a test database and provide various levels of multiprogramming has been discussed thus far. Now, the benchmark methodology must provide a set of "representative" queries which allow the researcher to compare different relational database systems. The questions now arise as to which performance indices are important and how one measures these indices.

3.2.1. Performance Measurement

Database systems, like other computing applications, tend to consume two primary resources; CPU cycles and disk I/O bandwidth. The principle concern to the database user is the amount of time required by the system to accept, process and return database information. With this in mind, it is reasonable to assert that the time required by the system to process both CPU intensive and I/O intensive queries would be important considerations in the benchmark methodology. Time,

however, is both a difficult parameter to define and a difficult parameter to measure. What time window do we measure? Is total elapsed time per query important? If it is, then, in the case of a database machine, we are appending the host system overhead time to the processing time of the database machine. For the software database solution this may be appropriate; but, is it appropriate to include this into the performance evaluation of a database machine? The answer is yes, but only partially. Let us look at the steps in processing a query for both systems using a time reference. First, the query must be received by the host from a user. This is required for both systems. Second, the query must be parsed and verified by the system. Again, this is required for both solutions. Third, the query is passed to the database application for execution. For the database machine, this requires an I/O operation to the backend communication link and transfer time of the parsed query to the database machine. This in not an incurred delay with the software system. This is a delay for the database machine and may become a large delay if the host is busy. Fourth, both systems must process the query. Fifth, the result must be returned to the user. Again, for the database machine, this involves an I/O operation to the host computer which must accept and possibly deblock or reformat the result prior to returning it to the user. This transfer time back to the host computer may get appreciably large if the query resulted in a large amount of participating data and if the host is very

busy. This example indicates that the "total elapsed time" metric may not be directly comparable for both database systems.

A performance metric which has gained popularity among performance researchers is "query execution time". This metric includes only the required time to process the query. This purposely excludes the terminal I/O overhead as well as any contention and transmission delays included with communications overhead. A related metric, "queries per second", has also gained popularity and appears in many studies. These are predominently used in this paper.

With metrics defined, how are they measured? This is the second problem with time based performance analysis. Most operating systems provide a facility to monitor the system clock. In this way, a software DBMS may record the time that it received a query from a user's process and compare that to the time when the query was completed by the DBMS as an estimate of the query execution time. If the database machine provides a monitor for this purpose, as in the case of the IDM/500, similar statistics will be available. If this utility is not available, then this metric will not be directly comparable across the relational systems in question.

With the development of performance metrics, the benchmark methodology must include "representative" queries to tax the system in specific and repeatable ways. Through past experimentation [16] [18] [21], four types of query

classifications have emerged:

Type 1 : Low CPU intensive, Low Disk intensive
Type 2 : Low CPU intensive, High Disk intensive
Type 3 : High CPU intensive, Low Disk intensive
Type 4 : High CPU intensive, High Disk intensive.

These query classifications, coupled with varying multi-processing levels, provides a fairly comprehensive environment for comparative studies of relational database systems.

3.2.2. Type 1 Query Classification

The type 1 query is included in the benchmark since it provides a measure of the "best case" performance of a relational database system.. For example:

```
select odd100A
   from thoustup
   where uniquelA < 10;
```

This simple query selects a small number of tuples (1%) from the smallest relation (thoustup). In this instance, neither CPU cycles nor disk I/O bandwidth should influence the response time. In this manner, a base line performance metric is provided to the benchmark user.

3.2.3. Type 2 Query Classification

The type 2 query provides a mechanism to measure the effect of high disk I/O activity on system performance. This query requires little CPU duty and returns a relatively small number of participating tuples (10%). For example:

```
select *
   from tenthoustup
   where tenD=5;
```

This query searches the largest relation in the database and returns data qualified on a non-indexed attribute (tenD). This inefficient searching activity on the entire relation (tenthoustup) also provides the benchmark user with information related to the benefits of using indexed attributes when searching the database.

3.2.4. Type 3 Query Classification

This type of query is used to provide the user with information about the effects of CPU intensive queries on system performance. Disk I/O activity is purposely kept to a minimum so that only CPU duty, and thus pure system speed, is measured. This class of query is different from the preceding two in that a series of queries is required. The strategy here is to sequentially add predicates to the query string in an attempt to locate the "breakpoint" at which the query becomes CPU bound. For example:

```
select twentyC
   from fivethoustup
   where twoC=0 AND
         fourC=3 AND
                .
                .
                . ;
```

In this query scheme, a retrieval of data is performed on the first predicate. A second query is then submitted

containing the first and second predicates. This is continued until the effect of CPU duty is observed. In this case, it would be advantageous to create another relation with very narrow tuple width. This would provide a maximum of tuples per I/O block and keep disk I/O activity to a minimum while the CPU deals with increasingly complex data qualification.

3.2.5. Type 4 Query Classification

This query type is included to provide a "worse-case" measurement of system performance. As in case 3, the degree of complexity may easily be modified in this query strategy. The attempt is to tax both the CPU power as well as disk I/O capacity to measure the overall capability of the system. This type of query usually involves a join operation of large relations, predicated to heavily tax the CPU and disk channels. For example:

```
select *, *,...
   from twothoustup, fivethoustup,...
   where unique2B = unique3C AND
         stringu2B = stringu3C AND
                                    .
                                    .
                                    .;
```

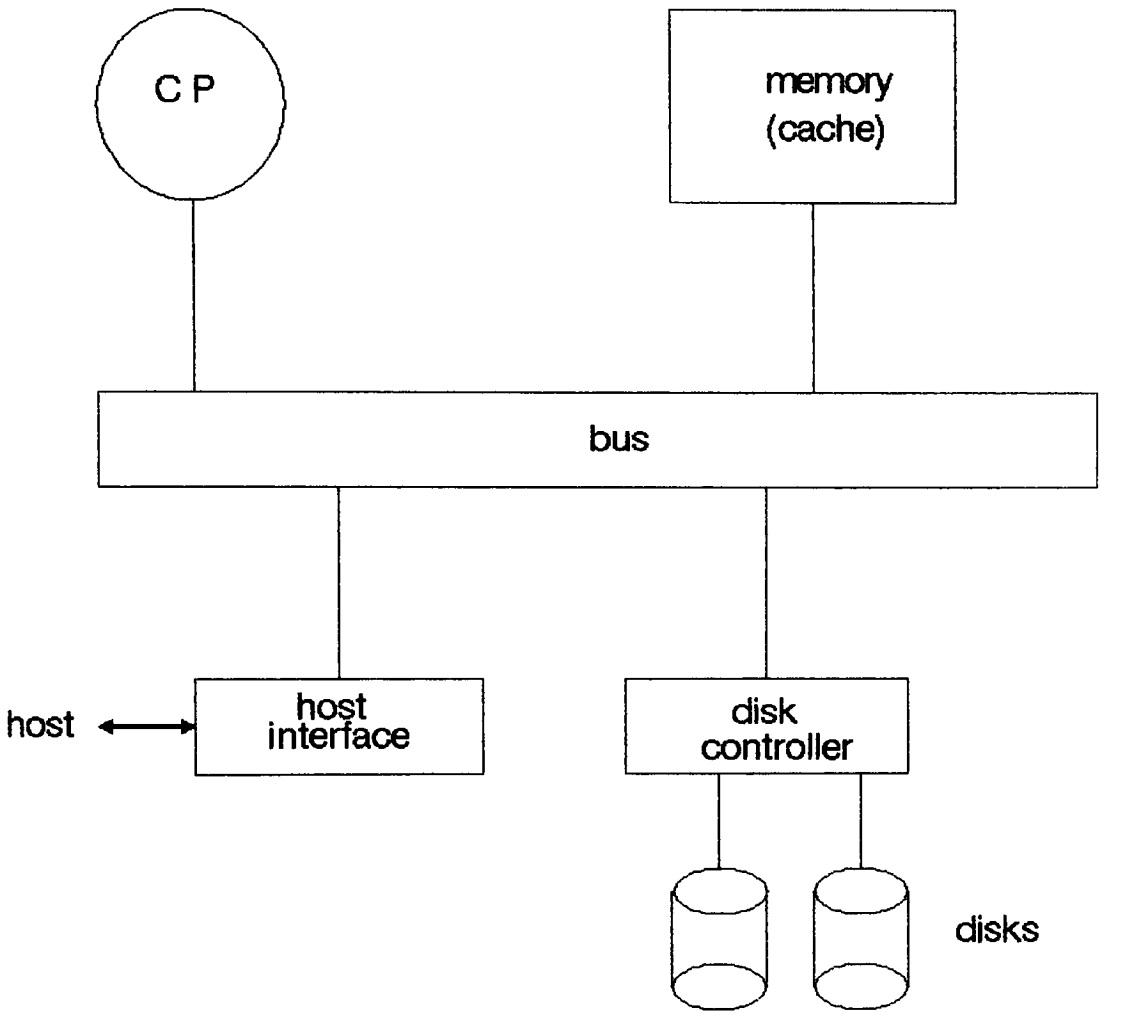
This query is using primary and secondary key attributes to perform the join. Modifications to the query could easily include join operations on non-key attributes or a three-way or higher join. These considerations are selectable by the benchmark user and should match the claimed capabilities of the system under scrutiny.

3.3.0. Hardware Implementation Considerations

A related requirement in the understanding of performance results is associated with the actual hardware design. Differences in system performance may be explained through the understanding of a system's physical configuration. Performance benefits through the implementation of relational database operations in hardware or firmware should also be understood to further enhance the understanding of benchmark results. Researchers have found that the various database machine designs (section 1.3.1.) tend to execute different query mixes with varying degrees of efficiency. This observation leads to the discussion of application dependent database machines.

3.3.1. Database Performance and Machine Design Considerations

The observed effects of database machine design on database performance has resulted in several basic machine architectures. These architectures demonstrate both strengths and weaknesses as the query mix varies. A close inspection of these architectures reveals that the primary differences reside in three areas of the machine. The first, and most obvious, difference is the number of processors in the database machine. Current designs incorporate from one to many processors. All designs include at least one processor which is commonly referred to as the "control processor". Figure 3.1 shows this most basic machine configuration. In this configuration, the control processor must manage all



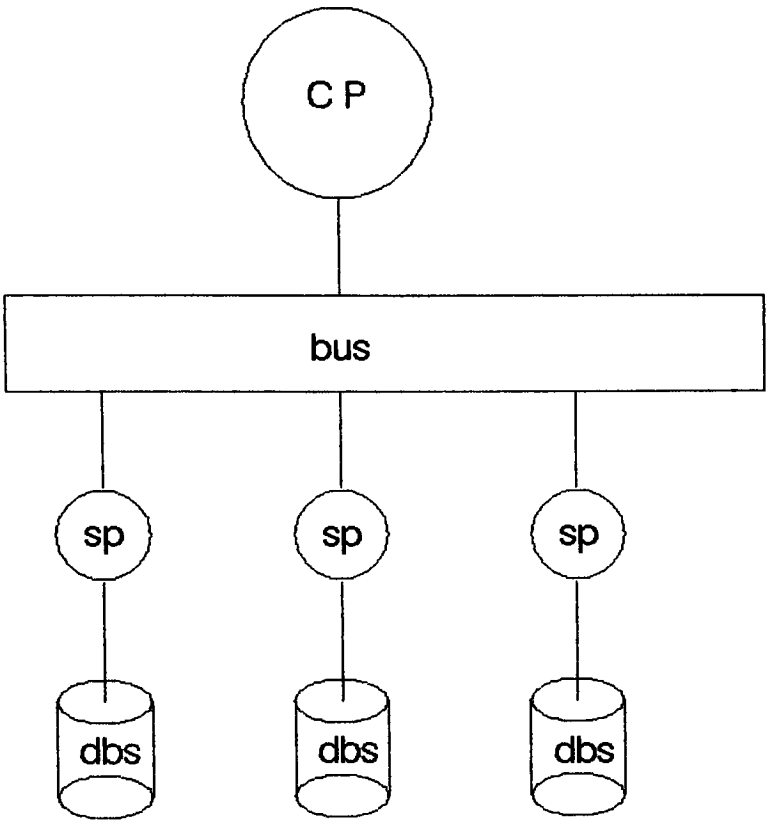
Single Instruction Single Data Stream
(SISD) Architecture

fig 3.1

machine resources as well as perform all the relational database operations. This architecture is limited to handling a single query at a time and is known as a SISD (single instruction single data stream) architecture. The Britton Lee IDM/500, even with the DAC slave processor, is a SISD machine. This is due to the fact that the DAC performs only some of the relational operations on behalf of the control processor [23].

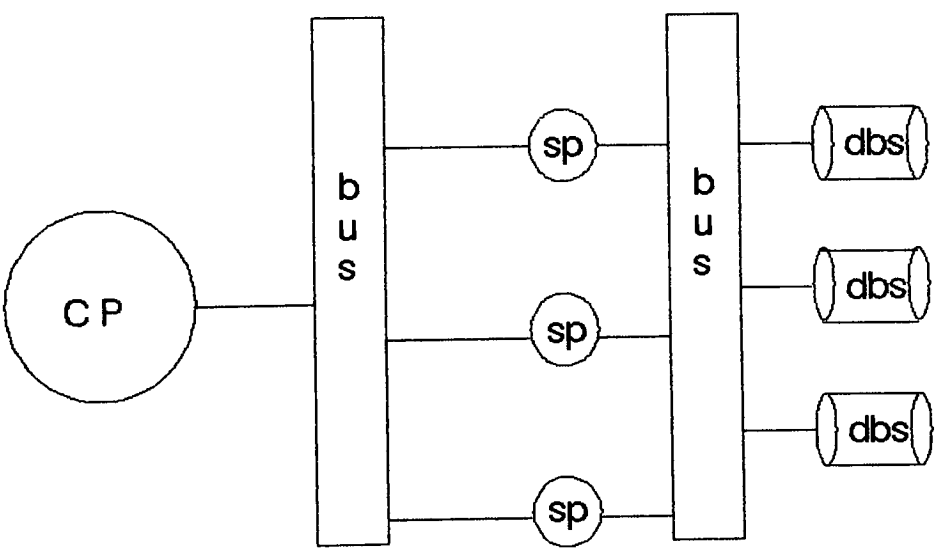
The second most notable difference in current machine design appears with the addition of multiple slave processors communicating with the database store. Figure 3.2 shows multiple slave processors (under the direction of the control processor (cp)) accessing the database store (dbs).

In this configuration, a single query is processed by all the slave processors simultaneously. The slave(s) find the qualified data and return the data to the control processor. The benefit, here, resides in the fact that the database may be accessed in 1/n time, as n increases to some practical limit (where n = number of slave processors). RAP [8] and CASSM [24] are examples of this architecture known as SIMD (single instruction multiple data stream). RAP and CASSM differ in this second design variation from DIRECT [25], DBMAC [26] and MDBS [27], in that the latter use an interconnection bus with the database store. Figure 3.3 shows this additional bus. In this configuration, the control processor may issue multiple query/instructions to the individual slave processors which can access the entire database store. In DIRECT, the



Single Instruction Multiple Data Stream
(SIMD) Architecture

fig 3.2



Multiple Instruction Multiple Data Stream
(MIMD) Architecture

fig 3.3

slave processors perform functional primitives such as a join or projection. In this way, the slaves cooperate together to satisfy a query. Machines of this design are known as MIMD (multiple instruction multiple data stream) architectures since they support the concurrent execution of multiple queries or instructions.

The third, and quite subtle, design difference of current database machines is in the exterior communication interface with the host computer(s). Erroneous or misleading results may be surmised about the machine performance which may actually be the effect of slow host to back-end communication lines. Consider an example where a query resulted in a block of returned data of 1 Mbyte. If the host to back-end line was an Ethernet line capable of approximatly 1 Mbyte/second data transfer rates, the elapsed time to return the data to the host system would require about 1 second. If, however, the host to back-end link is an RS-232 serial line running at 9600 baud (1200 bytes/second), the elapsed time to return the data to the host would span about 14 minutes! Clearly, the transfer time of any query which returns a high volume of data to the user would be heavily biased irregardless of what benchmark the query was designed to test.

3.3.2. Implementation of Database Operations

One of the primary benefits of database machines is the performance improvement gained through firmware/hardware implementation of database operations. Obviously, all

processor operations could be implemented in hardware but the current cost of such a machine would be so high that it will not be discussed here. The first question then becomes; what database operations are candidates for firmware/hardware implementation? The following five functions represent the basic database operations to be considered [10]:

1. Index search and scan operations
2. Bit-map operations
3. Sorting operations
4. Address translation
5. Tuple fetch

1. The database index entries are commonly implemented as a B-tree. B-tree nodes are typically a fixed sized page and entries within the page are sorted in the order of some key values for the entry. The operation of locating the next entry sequentially in the index is called an "index scan operation".

After the index is loaded into memory, a sequential scan of the index tree is required to locate the desired index. The speed of this "scan and compare" operation is closly tied to the memory cycle speed and consequently is memory cycle bound. A microcoded implementation running on an associative hardware chip can offer an improved scan time. This is due to the microcode instruction cycle time being approximately 3 times faster than the memory cycle time [28]. However, the index search operation is still heavily memory cycle bound and so a hardware implementation would offer little improvement over the firmware solution.

2. The bit map operations are used for logical AND, OR and bit stream comparisons. This type of bit comparison is at the crux of hardware logic. Very little logic and therefore software or microcode is required to perform these operations. This is then a candidate for a hardware implementation. In an actual implementation [10], a 64 bit stream was implemented using a tree of encoder chips of height two. This resulted in an execution time of 1.2 micro seconds which was 4 times faster than a firmware solution and over 16 times faster than a software implementation!

3. Sorting operations may be implemented under many algorithms. This implies that execution times vary depending upon which algorithm is implemented. Generally, sorting operations, like index searching operations, are heavily memory cycle bound. For the same reasons as stated in the index search and sort operations, the firmware solution offers an advantage over software implementations but hardware solutions are avoided.

4. Address translation efficiency is essential to most database systems operation. Databases, by nature, are stored on secondary memory devices, therefore logical to physical address mapping is frequently used. Many memory management methods are available to current designers but "hashing" techniques are very common. Hashing algorithms can be very complex and software solutions will be very memory cycle bound. The algorithm can be microcoded and implemented in

firmware for an improved execution time however, a hardware solution would be preferred for this repetitive and frequently used operation.

5. Tuple fetch is the time required to retrieve the tuple from secondary memory after its' address has been translated. Being fetched to primary memory, makes this operation dependent on memory cycle speed. Firmware solutions offer minimal advantages and hardware solutions, again, are avoided due to this memory speed constraint.

There are several factors, other than faster cycle time, which determine why firmware solutions have an advantage over software solutions. First, microcoded programs run in cooperation with specific hardware. This leads to fewer instruction fetches, and in addition, some operand fetches are avoided. Second, firmware solutions allow concurrent or piplined operations to be performed. Finally, microcoded programs are ususally more carefully written than a conventional program. This fact is an observation rather than a function of the technology.

3.3.3. Application Dependent Machine Design

Database machines are being utilized in a wide variety of applications. It is reasonable to assume that one machine design may perform better than another design in a particular application. Although performance benchmark testing has been decreed to be machine and application independent, let us look

at the expected performance of the machine designs of section 3.3.1. in different applications.

Let us look at three different application categories called the business system, the bibliographical search system and the statistical analysis system [29]. The business system is characterized by short, simple queries to highly structured data, for example; the employee records of several company departments. In this type of database application, it is likely that little manipulation of the requested data would be needed. If this is the case, the SIMD system (fig 3.2) would perform no better than SISD (fig 3.1) since a query is likely to return little data. If the data resides on one disk, only one slave processor would be active and the rest would be idle. The MIMD system (fig 3.3) would probably offer poorer performance due to the CP's increased overhead from system complexity. One could contend, that if the users of a business system need only simple query support, then the purchase of a SIMD or MIMD machine would, at best, offer the same performance as the (cheaper) SISD system.

In a bibliographical search system, queries are characterized by long searches through large relations, only to return small data sets to the user. This type of system might be used for library catalogues, telephone directory look-up, etc. This type of system typically has poorly indexed relations since the index tables are too large to maintain or too time consuming to use. Applications running

on this type of system usually require a projection and/or restriction operation. Since a large amount of data is inspected, it is not expected that a SISD design would be adequate since multiple processors could search the data store much more quickly. In the MIMD system, the data must first be "bused" for the slave processor to gain access. This can indicate that the bus could become a bottleneck. The SIMD design looks to be the optimum for this application since the slave processors can perform the select/project operations before returning any data across the bus to the CP.

In the statistical analysis application, the queries are characterized by the high amount of manipulation over large amounts of data. Applications using this type of system might be medical databases, national sales figures (economics), summary census databases etc. Complex queries to join, project and select data from many data relations would be typical for this application. The complexity of the queries separates this application from the bibliographical search system. In a SISD design, the CP would quickly become the system bottleneck due to the query complexity. SIMD designs issue the same query to all slave processors which could remove some of the work from the CP (ie. projections, selections). The MIMD seems to be the best choice here since a query may be parsed into descrete relational operations which could be executed concurrently by the slave processors. The additional overhead in this system would be compensated for by the improved query throughput.

This discussion indicates that application dependence may be an integral part of machine design. Does this infer application dependent benchmarking? At the beginning of this chapter, a point is made about the requirement for application independent benchmarking. This requirement is asserted to maintain the generality and credibility of the benchmark design. The chapter then proceeds with discussions of hardware architectures and the ability of one architecture to better execute certain queries then the others. This infers that instead of application specific benchmark testing, one must include representative queries in the benchmark's design to rigorously test the DBMS. This is to say, that query work loads which simulate the three mentioned database applications must be included in the benchmark to provide the researcher with sufficient information to evaluate the DBMS system being tested. In this way, an intelligent selection of a DBMS may be made which satisfies the type of application(s) to be supported.

Chapter 4

Comparative Analysis of Several Benchmark Studies

This chapter presents a comparative analysis of several published benchmark studies. These studies principally compare the Britton Lee IDM/500 database machine to other database machines and to software-only database management systems. The studies are logically presented in two groups. First, the single-user benchmarks are described in detail followed by a description of the multiple-user studies. A summary and comparison of the benchmark results is then presented.

A word of caution is noted here concerning the comparison of benchmark results across these studies. These benchmarks use different types of host computers with varying degrees of resources and networking capability. One must be very careful when comparing performance, in general, across multiple systems. The intent is to provide a conceptual appreciation for database machine performance analysis and a tangible report of several published benchmarks.

The information reported for each study will follow a common format. First, the hardware/software system, as a whole, will be described. Next, a summary of the database queries and context will follow. Finally, a summary of the performance statistics will be presented in both tabular and

graphical forms.

4.1.0. Single-User Benchmark Performance

This section will describe the single-user benchmark studies. Many reports include the single-user case when developing the more complex multiple-user study. The single-user part of those studies will be reported here. This is done to give the reader an appreciation for the goals of the benchmark while deferring the more complex issues of multiple-user studies to section 4.2.0.

4.1.1. Benchmark - I

The first study to be reviewed [20], deals with both the single-user and multiple-user benchmark job loads. The single-user case is presented here. Database machines of several sizes are benchmarked in this study along with a software DBMS. This paper deals with mid-sized database machines so only those cases from the study will be included.

4.1.1.1. Software D.B.M.S. Configuration

A Digital Equipment Corporation VAX 11/750 is used as the host for the software-only database system. This VAX is configured with 2 Mbyte of main memory, a RL02 (20Mbyte) system disk and a CDC 9766 (300Mbyte) mass storage device. The ORACLE Database System version 3.1.1 was installed on the VAX which ran the VMS version 3.0 operating system. When ORACLE was installed, all default settings provided by ORACLE were used except for the "ORACLE $BI" file. This "before

image" file is used to retain an unaltered version of the database being used prior to a "commit" command. This file would be restored as the latest accurate version of the database following a system crash and has to be configured with sufficient size to hold the largest database (10 Mbyte).

4.1.1.2. IDM/500 System Configuration

A Britton Lee IDM/500 database machine was used as the hardware DBMS for this study. The IDM is configured with 1Mbyte of primary memory and operated without the "database accelerator" board. A C.D.C. 9766 (300Mbyte) disk drive operates as the mass storage device for the system. Release 24 of the IDM software is used and communication to the "host" computer is through a 9600 baud RS-232 interface. The VAX-11/750 is used as the host computer for this test. Berkeley Unix 4.1 is used as the operating system for the VAX.

4.1.1.3. Database Context and Queries

The records for the database were provided by the Department of Commerce. The data contains personnel information and individual records were randomly selected to form three (3Mbyte, 6Mbyte and 10Mbyte) databases. Each database contains 12 relations comprised of records with varying length from 8 bytes to 85 bytes. The 3Mbyte database contains 10,500 records. The 6Mbyte database contains 20,000 records and the 10Mbyte database contains 33,000 records.

The queries for this study were written in SQL and were designed to perform a wide variety of retrievals and updates.

All query sets were run with three levels of indices. These include no indices, "level-1" indices (unique, clustered index) and "level-2" indices (non-clustered index plus level-1 index). Ten sets of queries were developed with increasing complexity. Query sets 1 through 5 test single relation retrieval. The size of the relation separates the query groups. Small relations were tested with query set "q-1". Medium size relations were tested with query sets "q-2" and "q-3" and large relations with sets "q-4" and "q-5". Query sets "q-6" through "q-10" test multiple relation retrieval. Query sets "q-6,7,8" test two relation retrieval while "q-9" tests three relation retrieval and "q-10" tests four relation retrieval. Other queries were run to test ordered sorts, insertions, deletions, query predicate order and others.

Of the approximate 100 queries used in the study, four (4) have been selected to represent each of the four classifications of queries developed in 3.2.1. This is done to provide the reader with a representative set of queries which thoroughly tax the speed and disk I/O capacity of the DBMS.

Type I Query - (q1-2):

```
Select ssn, ret.grade, ret.pay.plan
  from retain.data
  where ret.pay.plan = "WG";
```

Type II Query - (q4-2):

```
Select agency, subelement
  from agency.desc
  where agency = "BD" or agency = "AF";
```

Type III Query - (q6-4):

```
Select retain.data.ssn, ret.grade, barg.unit
  from retain.data, job.detail
  where retain.data.ssn = job.detail.ssn
     and (((Patco = "T" or patco = "0")
     and barg.unit = "7777"
     and ret.grade < "08")
     or ret.pay.plan = "WG");
```

Type IV Query - (q 9-4)

```
Select pers.data.ssn, educ.level, birth.date, vet.pref
  from pers.data, education, pers-misc
  where pers.data.ssn = education.ssn
     and education.ssn = pers.misc.ssn;
```

4.1.1.4. Software DBMS Benchmark Results

The focus of this study is "query execution time". As described in section 3.2.1, this metric contains only the time needed to process the query. The time to parse and transmit the query is omitted. Many time based metrices are included in the analysis, but "time to first" and "time to last" are the primary statistics considered. The difference between "time to first" and "time to last" computes the "query execution time". The study presents several groups of queries designed to tax different capabilities of the DBMS. The study only provides, however, detailed data describing the effect of indexing levels used within the queries. The analysis here will include only these statistics due to the limited data.

The queries were run under the three different levels of indexing. First, "level-0" queries which include NO index for the queries were executed. Without the use of an index, no query that ran against the 10Mbyte database completed before the 30 minute time limit usually imposed by the researchers.

Also, type 3 and type 4 queries could not complete when run against any of the databases. Table 4.1.1 shows the published statistics for the "level-0" index queries.

The next level of indexing provides a more interesting case. Level-1 indexing provides that each relation in the database be equipped with a unique, clustered index. In this case, all queries were allowed to run to completion. Table 4.1.2 shows the "time to first" statistics for all three databases and all query types. Table 4.1.3 shows the "time to last" statistics for these same conditions. These intermediate statistics are included here to provide the reader with an appreciation for the speed of the systems being tested. The "time to first" statistic shows how quickly (or slowly) the DBMS finds and returns the first record satisfying the query. This information is not directly provided with the "query execution time" statistic. These statistics will only be provided in this benchmark study. Table 4.1.4 and figure 4.1.4 show the total "query execution" time for level-1 indexing.

Finally, "level-2" indexing is imposed for the study. In this level of indexing, all relations are equipped with a non-clustered index in addition to the unique, clustered index of level-1. Table 4.1.5 shows the "time to first" statistics for the three databases and all query types. Table 4.1.6 shows the "time to last" statistics for these same conditions. Table 4.1.7 and figure 4.1.7 show the query execution times

for all 3 databases and query types using level-2 indices.

4.1.1.5. IDM/500 Benchmark Results

The same query sets were run on the IDM/500 system as were run on the software only DBMS. The researchers verified that the identical records were retrieved with each query (Table 4.1.0). Again, the thrust of this study is to identify the effects of indexing on the database systems. Thus, the same strategy of index levels is used on the IDM/500 system as was described in section 4.1.1.3.

Table 4.1.8 shows the available data for level-0 indexing for the "time to first" and "time to last" response on the IDM/500 system. Only the data for the 6 Mbyte database was available. Table 4.1.9 shows the "time to first" data for level-1 indexing and Table 4.1.10 shows the "time to last" data for level-1 indexing. Table 4.1.11 and figure 4.1.11 show the "query execution time" for all 3 databases and query types for level-1 indexing. Figure 4.1.15 shows the results of level-1 indexing for both the software only DBMS and the IDM/500 system.

The last test in the benchmark includes level-2 indices. Table 4.1.12 shows the "time to first" data using level-2 indexing while Table 4.1.13 shows the "time to last". Table 4.1.14 and figure 4.1.14 show the query execution time for level-2 indexing on the IDM/500 database machine. Figure 4.1.16 shows the results of level-2 indexing for both the software only DBMS and the IDM/500 system.

4.1.1.6 Benchmark - I Summary

The data tables and figures in this benchmark study show some interesting effects of indexing on query execution times. The following observations are drawn from the data.

* On the Software-Only Database System, the level of indexing had almost no effect on the query execution time over all query types.

* On the IDM/500 Database Machine, the time to receive the first record increased as the index level increased. This may be expected as the overhead increases due to the use and maintenance of additional index tables.

* On the IDM/500 system, minimal improvement is obseved in level-2 indexing over level-1.

* On the IDM/500 System, the type of query has a large effect on query execution time. Query types I and III (high CPU activity) complete in a fraction of the time required for types II and IV (high disk activity). This infers that the disk I/O bottleneck may be a limiting factor in the IDM/500 system.

* On the IDM/500 System, the "time to first" is almost always longer then on the Software-Only system. This may be partially due to the time required to receive the data from the IDM/500 through the (slow) 9600 baud communications line.

* No type IV query run against the 10 Mbyte database

completed on the IDM/500 in the allotted time. Complex queries require and appreciable amount of time to execute on both systems. In all other cases, the IDM/500 system out performed the Software-Only system, sometimes significantly.

NUMBER OF RECORDS
RETURNED

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	46	72	78
2 Type II	10500	19948	19948
3 Type III	46	72	79
4 Type IV	11152	21349	

Table 4.1.0

LEVEL 0 INDEX
RETRIEVAL TIME (seconds)
SOFTWARE-ONLY DATABASE SYSTEM

Query Type	3 Mbyte First	3 Mbyte Last	6 Mbyte First	6 Mbyte Last
1 Type I	0.39	1.54	0.30	2.39
2 Type II	0.37	190.18	0.31	373.82
3 Type III				
4 Type IV				

Table 4.1.1

LEVEL 1 INDEX
TIME TO FIRST (seconds)
SOFTWARE-ONLY DATABASE SYSTEM

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	0.34	0.53	0.36
2 Type II	0.40	0.31	0.38
3 Type III	2.82	3.13	3.18
4 Type IV	1.74	1.22	1.16

Table 4.1.2

LEVEL 1 INDEX
TIME TO LAST (seconds)
SOFTWARE-ONLY DATABASE SYSTEM

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	1.47	2.21	2.75
2 Type II	191.04	363.21	517.44
3 Type III	380.39	826.77	1355.40
4 Type IV	1473.72	2862.69	5884.11

Table 4.1.3

LEVEL 1 INDEX
QUERY EXECUTION TIME (seconds)
SOFTWARE-ONLY DATABASE SYSTEM

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	1.13	1.68	2.39
2 Type II	190.64	362.90	517.06
3 Type III	377.57	823.64	1352.22
4 Type IV	1471.98	2861.47	5882.95

Table 4.1.4

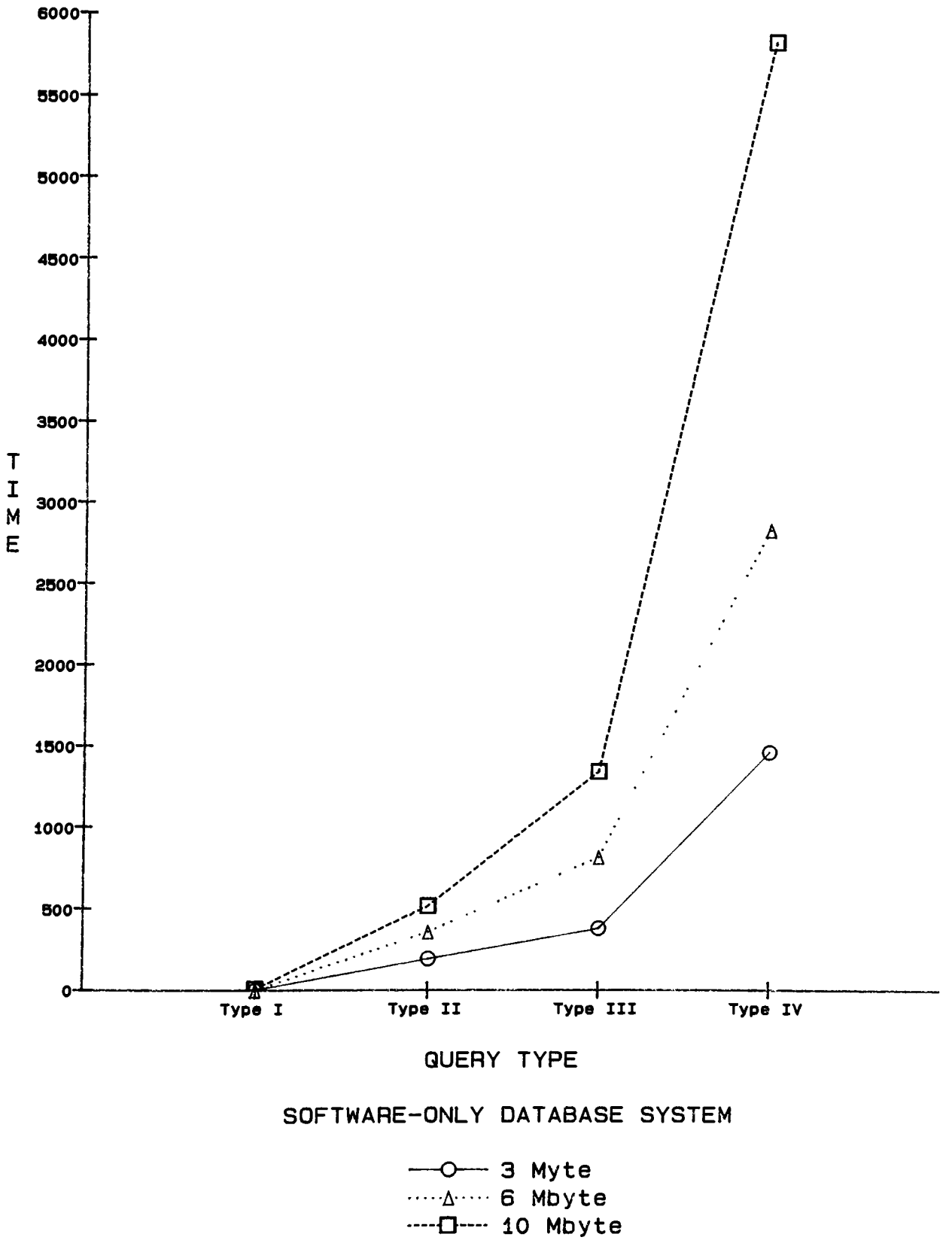

LEVEL 1 INDEX
QUERY EXECUTION TIME (seconds)
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
TIME
Type I
Type II
Type III
Type IV
QUERY TYPE
SOFTWARE-ONLY DATABASE SYSTEM
3 Myte
6 Mbyte
10 Mbyte

Figure 4.1.4

LEVEL 2 INDEX
TIME TO FIRST (seconds)
SOFTWARE-ONLY DATABASE SYSTEM

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	0.54	0.59	0.50
2 Type II	0.35	0.31	0.39
3 Type III	2.83	3.32	3.13
4 Type IV	1.05	1.07	1.84

Table 4.1.5

LEVEL 2 INDEX
TIME TO LAST (seconds)
SOFTWARE-ONLY DATABASE SYSTEM

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	1.52	2.24	2.34
2 Type II	190.95	376.88	533.25
3 Type III	391.81	825.40	1348.59
4 Type IV	1503.57	2845.86	5873.57

Table 4.1.6

LEVEL 2 INDEX
QUERY EXECUTION TIME (seconds)
SOFTWARE-ONLY DATABASE SYSTEM

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	0.98	1.65	1.84
2 Type II	190.60	376.57	532.86
3 Type III	388.98	822.08	1345.46
4 Type IV	1502.52	2844.79	5871.73

Table 4.1.7

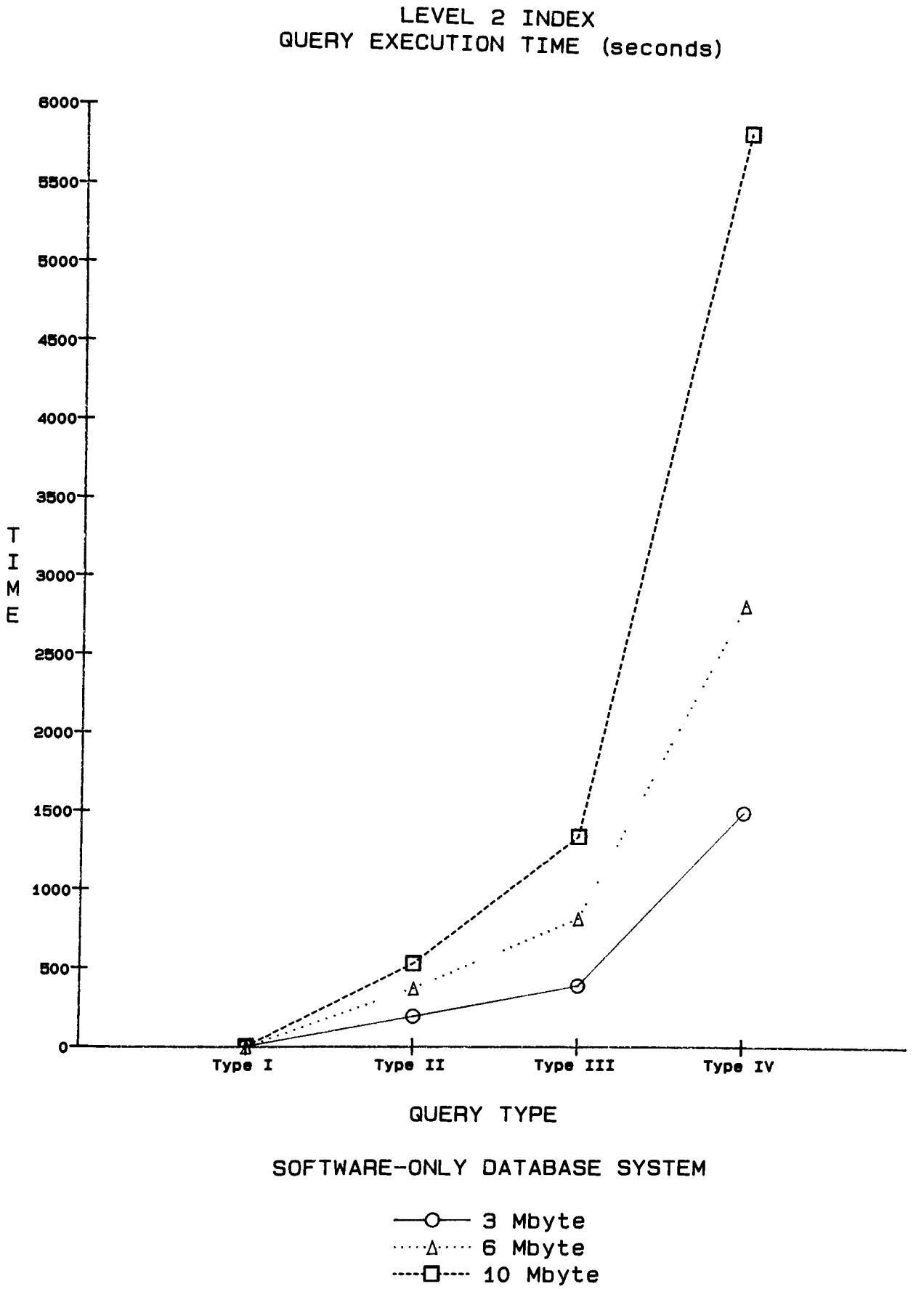

LEVEL 2 INDEX
QUERY EXECUTION TIME (seconds)
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
TIME
Type I
Type II
Type III
Type IV
QUERY TYPE
SOFTWARE-ONLY DATABASE SYSTEM
3 Mbyte
6 Mbyte
10 Mbyte

Figure 4.1.7

LEVEL 0 INDEX (seconds)
TIME TO FIRST AND LAST FOR 6 Mbyte DATABASE
IDM/500 DATABASE MACHINE

Query Type	6 Mbyte Time to First	6 Mbyte Time to Last
1 Type I	0.87	1.80
2 Type II	1.10	194.63
3 Type III	145.10	693.25
4 Type IV		

Table 4.1.8

LEVEL 1 INDEX
TIME TO FIRST (seconds)
IDM/500 DATABASE MACHINE

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	1.08	1.02	1.08
2 Type II	1.05	1.10	1.05
3 Type III	1.80	2.08	107.28
4 Type IV	2.03	1.75	

Table 4.1.9

LEVEL 1 INDEX
TIME TO LAST (seconds)
IDM/500 DATABASE MACHINE

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	1.63	1.92	2.07
2 Type II	102.32	192.52	204.17
3 Type III	2.75	4.23	626.52
4 Type IV	528.73	1028.57	

Table 4.1.10

LEVEL 1 INDEX
QUERY EXECUTION TIME (seconds)
IDM/500 DATABASE MACHINE

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	0.55	0.90	0.99
2 Type II	101.27	191.42	203.12
3 Type III	0.95	2.15	519.24
4 Type IV	526.70	1026.82	

Table 4.1.11

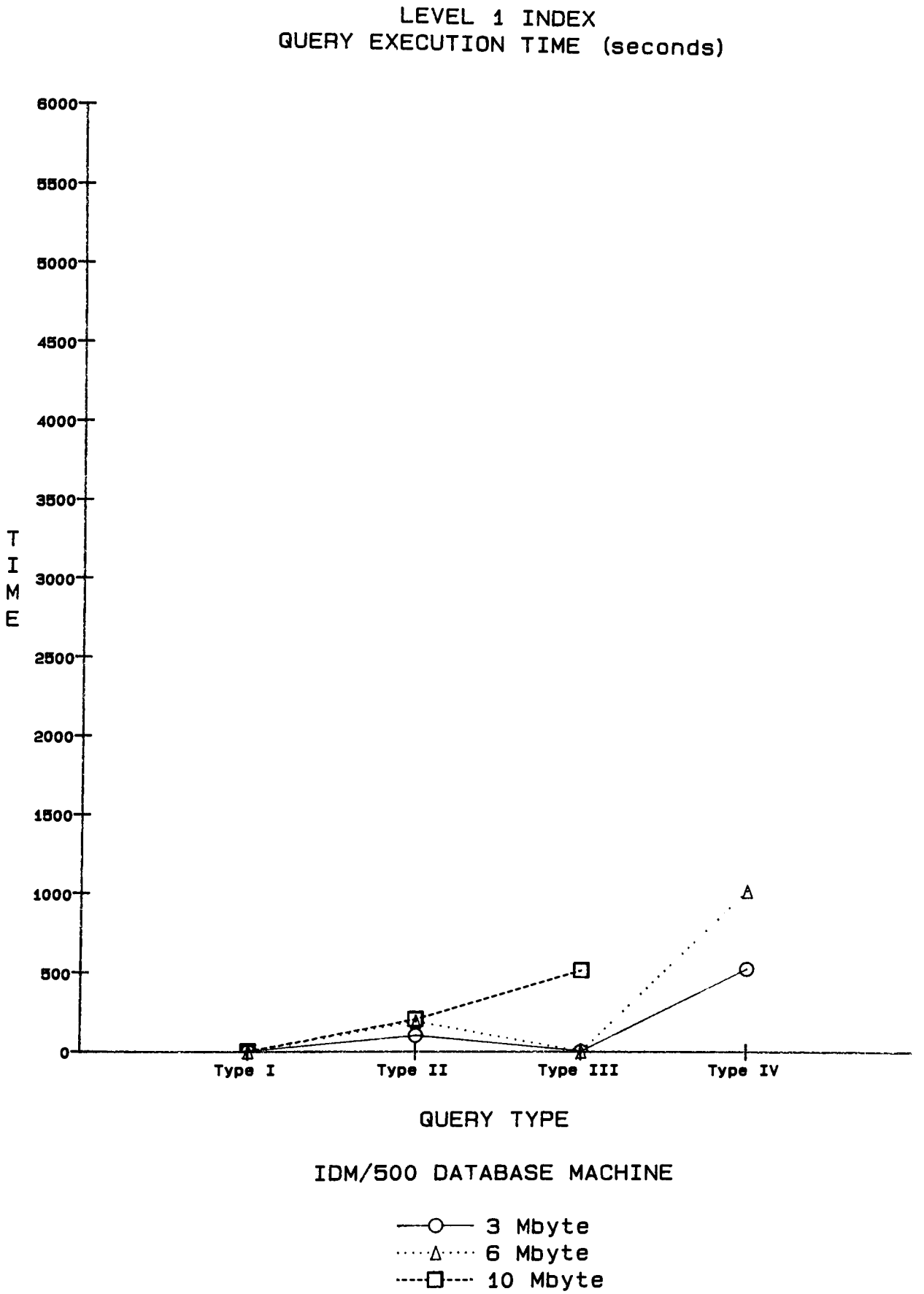

LEVEL 1 INDEX
QUERY EXECUTION TIME (seconds)
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
TIME
Type I
Type II
Type III
Type IV
QUERY TYPE
IDM/500 DATABASE MACHINE
3 Mbyte
6 Mbyte
10 Mbyte

Figure 4.1.11

LEVEL 2 INDEX
TIME TO FIRST (seconds)
IDM/500 DATABASE MACHINE

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	1.87	1.42	1.15
2 Type II	1.08	1.13	0.98
3 Type III	1.76	1.75	107.27
4 Type IV	2.12	2.07	

Table 4.1.12

LEVEL 2 INDEX
TIME TO LAST (seconds)
IDM/500 DATABASE MACHINE

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	2.82	2.32	2.13
2 Type II	102.95	193.57	204.20
3 Type III	2.32	3.93	186.45
4 Type IV	528.35	851.62	

Table 4.1.13

LEVEL 2 INDEX
QUERY EXECUTION TIME (seconds)
IDM/500 DATABASE MACHINE

Query Type	3 Mbyte	6 Mbyte	10 Mbyte
1 Type I	0.95	0.90	0.98
2 Type II	101.87	192.44	203.22
3 Type III	0.56	2.18	79.18
4 Type IV	526.23	849.55	

Table 4.1.14

LEVEL 2 INDEX
QUERY EXECUTION TIME (seconds)

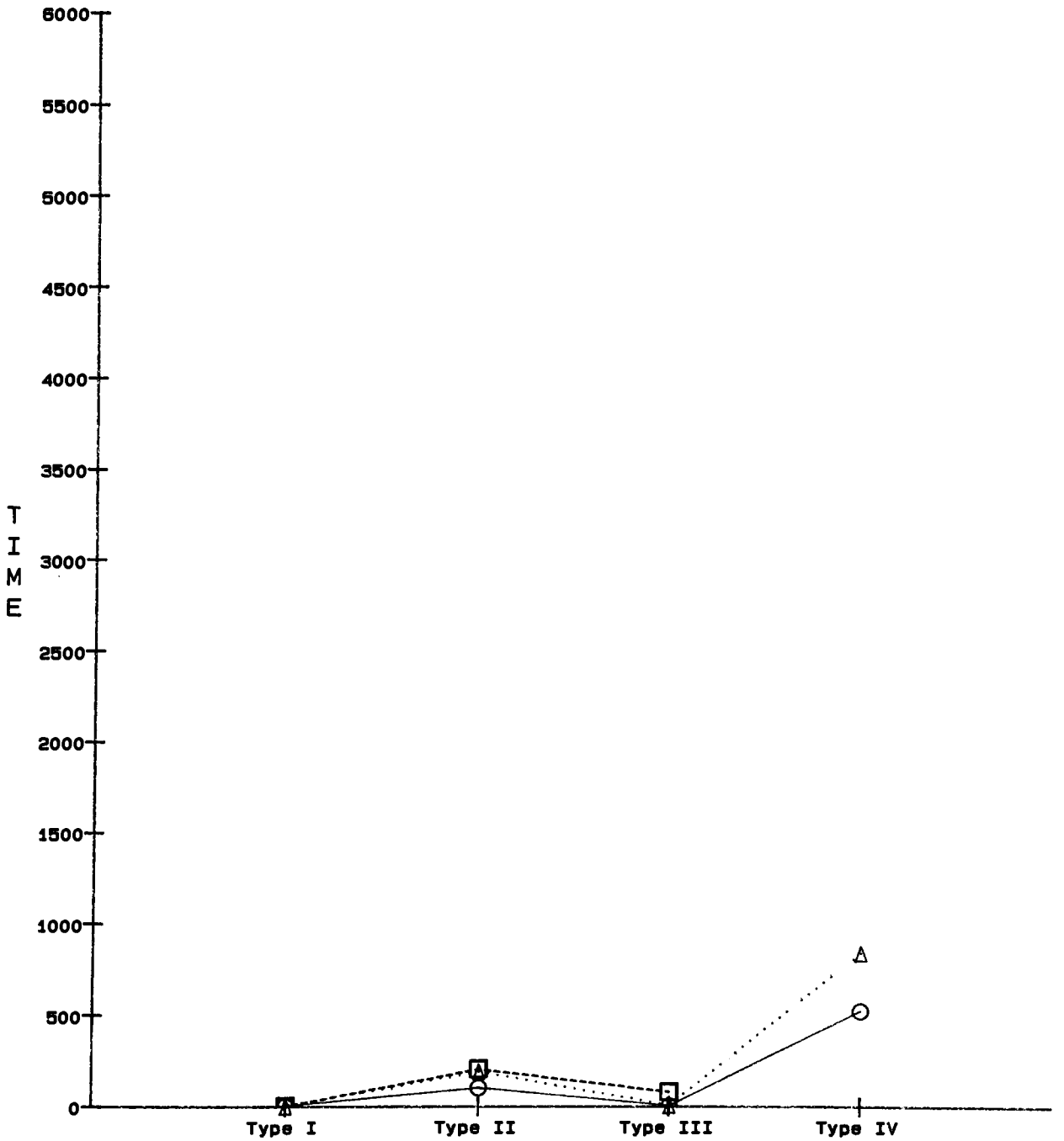


Figure 4.1.14

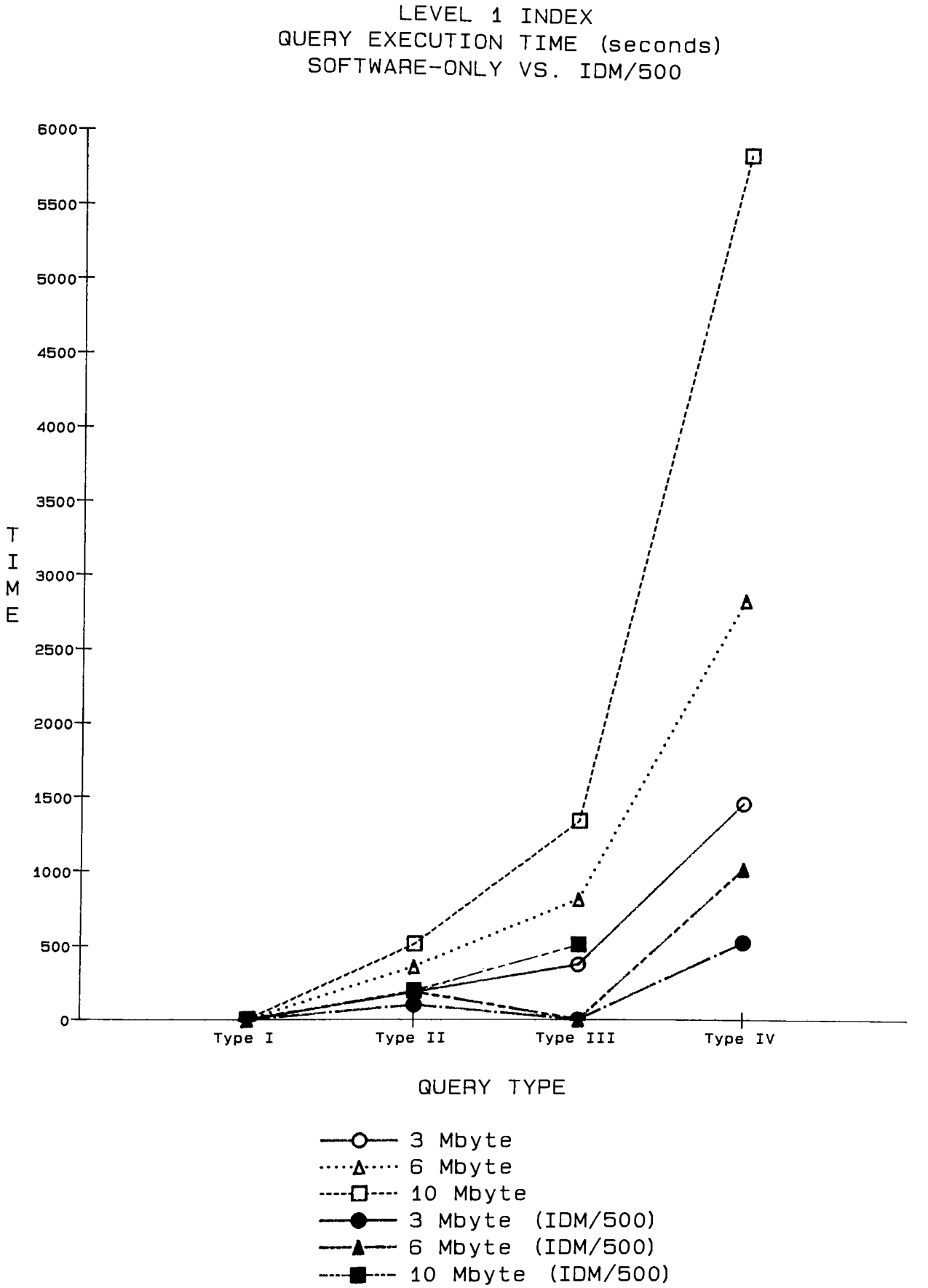

LEVEL 1 INDEX
QUERY EXECUTION TIME (seconds)
SOFTWARE-ONLY VS. IDM/500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
TIME
Type I
Type II
Type III
Type IV
QUERY TYPE
3 Mbyte
6 Mbyte
10 Mbyte
3 Mbyte (IDM/500)
6 Mbyte (IDM/500)
10 Mbyte (IDM/500)

Figure 4.1.15

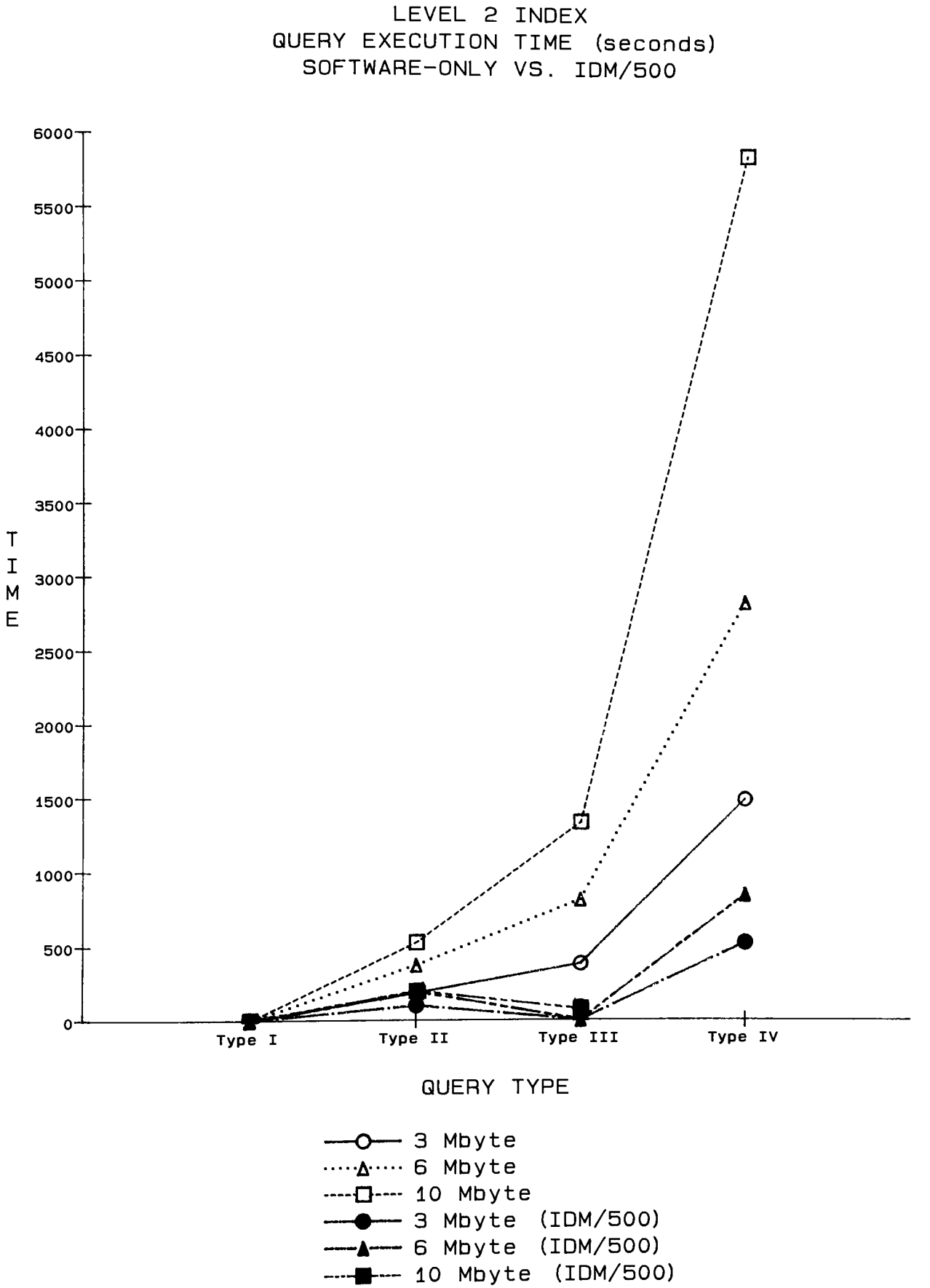

LEVEL 2 INDEX
QUERY EXECUTION TIME (seconds)
SOFTWARE-ONLY VS. IDM/500
6000
5500
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
TIME
Type I
Type II
Type III
Type IV
QUERY TYPE
3 Mbyte
6 Mbyte
10 Mbyte
3 Mbyte (IDM/500)
6 Mbyte (IDM/500)
10 Mbyte (IDM/500)

Figure 4.1.16

4.1.2. Benchmark - II

The next benchmark for consideration is another single-user study that was performed at the University of Wisconsin [15]. In this study, five database management systems are benchmarked. These systems are made up of three software only systems and two database machines. The three software systems are ORACLE, U-INGRES (university version) and C-INGRES (commercial version). The two database machines are the IDM/500 and the DIRECT database machine developed at the University of Wisconsin [25].

4.1.2.1. University - Ingres Database System Configuration

This software database system was loaded onto a VAX 11/750 running the Berkley 4.1 Unix operating system. The VAX was configured with 2 Mbyte of primary memory and the database was stored on a Fujistu Eagle drive (474 Mbyte). This version of INGRES has no buffer management of its own and therefore relies on the operating system to provide the buffer management function. Unfortunately, as discussed in [30], the LRU (least recently used) data pages become candidates for swapping in BSD 4.1 Unix. The LRU algorithm for buffer management happens to be the worst algorithm for repeated access to the inner relation during a "join" operation.

4.1.2.2. Commercial - Ingres Database System Configuration

This system (version 2.0 of commercial INGRES) was loaded

onto a VAX 11/750 configured with 6 Mbyte of primary memory. The VMS version 3.0 operating system was used and a Fujistu Eagle drive contained the database. Several performance enhancements, not available in U-INGRES, are included in C-INGRES. A data page buffer management system, tuned for relational operations, is part of the C-INGRES system. The data page size is doubled (2k byte vs 1k byte pages in 4.1 Unix) and caching of queries is included to permit the re-execution of a query thereby avoiding re-parsing.

4.1.2.3. ORACLE Database System Configuration

In this system, version 3.1 of the ORACLE software was loaded onto a VAX 11/750. The VAX contained 4 Mbyte of primary memory and a Fujustu Eagle drive was used for database storage. Berkeley 4.2 Unix was the operating system. ORACLE is somewhat different then INGRES, from the user's perspective, since ORACLE implements an SQL interface.

4.1.2.4. The IDM/500 Database Machine Configuration

The IDM/500 used in this benchmark was configured with 2 Mbyte of primary memory, a 675 Mbyte CDC drive, a parallel channel (IEEE-488) communications interface to the host and a DAC (database accelerator board) which could be turned on and off as required. Release 25 of the IDM software was used during this test. The host computer used in the benchmark is a DEC PdP 11/70 running a variant of the Unix 2.8 operating system.

4.1.2.5. The DIRECT Database Machine

DIRECT is a multi-processor database machine designed to implement parallelism of relational database operations. In this benchmark, a VAX 11/750 is used as the host computer and also used as the back-end controller. The function of the back-end controller is to select the optimal number of processors to work in parallel to satisfy a given query. The "parsed packets" of machine language instructions are passed to the selected processors for execution. The back-end machine is comprised of 4-LSI 11/23 processors, each configured with 128 Kbyte of primary memory. All 4 processors share a 1/2 Mbyte disk cache which is used to hold records comming from and going to the database which is stored on a Fujistu Eagle drive. Berkeley 4.1 Unix was selected as the operating system to run on the VAX 11/750.

4.1.2.6. Database Context and Queries

The Wisconsin database, detailed in section 3.1.2., is the database used in this study. The benchmark queries are divided into five groups of relational database operations. These are Selection, Projection, Join, Aggregates and Updates. For continuity of analysis, four queries are selected which approach the "representative" query types discussed in 3.2.1. The provided tables of data are made up of the average query execution time for repeated queries. The values themselves represent the total elapsed time to complete the query and write the result to disk. This is done, as in Benchmark - I,

to eliminate the effects of any parsing and terminal I/O delay.

As in Benchmark - I, the queries in this study were repeated using three levels of indexing. These include no index, a primary clustered index, and secondary non-clustered index. The available data for each query type will be included. The queries are written in SQL format.

Query Type - I

```
select *
  from onektupl
  where unique 2A = 2001;
```

Query Type - II

```
select *
  from tenktupl
  where unique 2D > 902
    and unique 2D < 1903;
```

Query Type - III

```
select * , *
  from twoktupl, twoktup2
  where unique 2B = unique 2C
    and unique 2C < 1000;
```

Query Type - IV

```
select * , * , *
  from onektup, tenktupl, tenktup2
  where unique 1A = unique 1D
    and unique 1D = unique 1E
    and unique 1E < 1000
    and unique 1D < 1000;
```

4.1.2.7. Benchmark - II Results

The following data tables contain the query execution time for all five systems. Since this paper deals primarily with the comparison of the IDM/500 with other systems, figures

4.2.1 through 4.2.4 show this comparison over all query types. Each query type was run with varying levels of indexing. As the figures illustrate, sometimes drastic effects are seen through this variation of indexing. The most notable are ORACLE and the IDM/500. Figures 4.2.2, 4.2.3(a) and 4.2.4(a) show that ORACLE and the IDM/500 perform poorly and even fail when operating without the use of any index. The other figures indicate, however, that the IDM/500 outperforms all others when provided with some type of index. The data for these figures are included in the provided tables. Table 4.2.1 contains the data for Query Type I. This differs with Query Type II only in that high disk I/O is avoided through the use of indices in Type I. Table 4.2.2 contains the data for Query Type II using no index. Table 4.2.3 contains the data for Query Type III and the data for Query Type IV is in Table 4.2.4.

The following observations are drawn from the data.

* For both U-INGRES and C-INGRES, several instances are observed where query execution time increases with the use of indices over the use of no index. This means that the query optimizer did not recognize that the query would be better executed without the use of the index.

* In DIRECT, it is evident from tables 4.2.1, 2 and 3, that it's use of parallel processors is not a replacement for indexing. However, as the complexity of the query increases (Type IV), the relative preformance of DIRECT improves.

* For Join queries (Type III and Type IV) C-INGRES is the only system that always provided adequate performance. In these cases, the IDM/500 (and ORACLE) systems failed when no index for the join was available. If available, however, the IDM/500 performed significantly better than any of the other systems.

* ORACLE consistently performed poorer than the other systems in the study.

QUERY EXECUTION TIME (seconds)
QUERY TYPE I

SYSTEM	CLUSTERED INDEX	NON-CLUSTERED INDEX
1 U-INGRES	7.7	59.2
2 C-INGRES	3.9	11.4
3 ORACLE	16.3	17.3
4 DIRECT	43.0	43.0
5 IDM/500	1.5	3.3

Table 4.2.1

QUERY EXECUTION TIME (seconds)
QUERY TYPE II

SYSTEM	NO INDEX
1 U-INGRES	53.2
2 C-INGRES	38.4
3 ORACLE	194.2
4 DIRECT	43.0
5 IDM/500	21.6

Table 4.2.2

QUERY EXECUTION TIME (seconds)
QUERY TYPE I
CLUSTERED INDEX

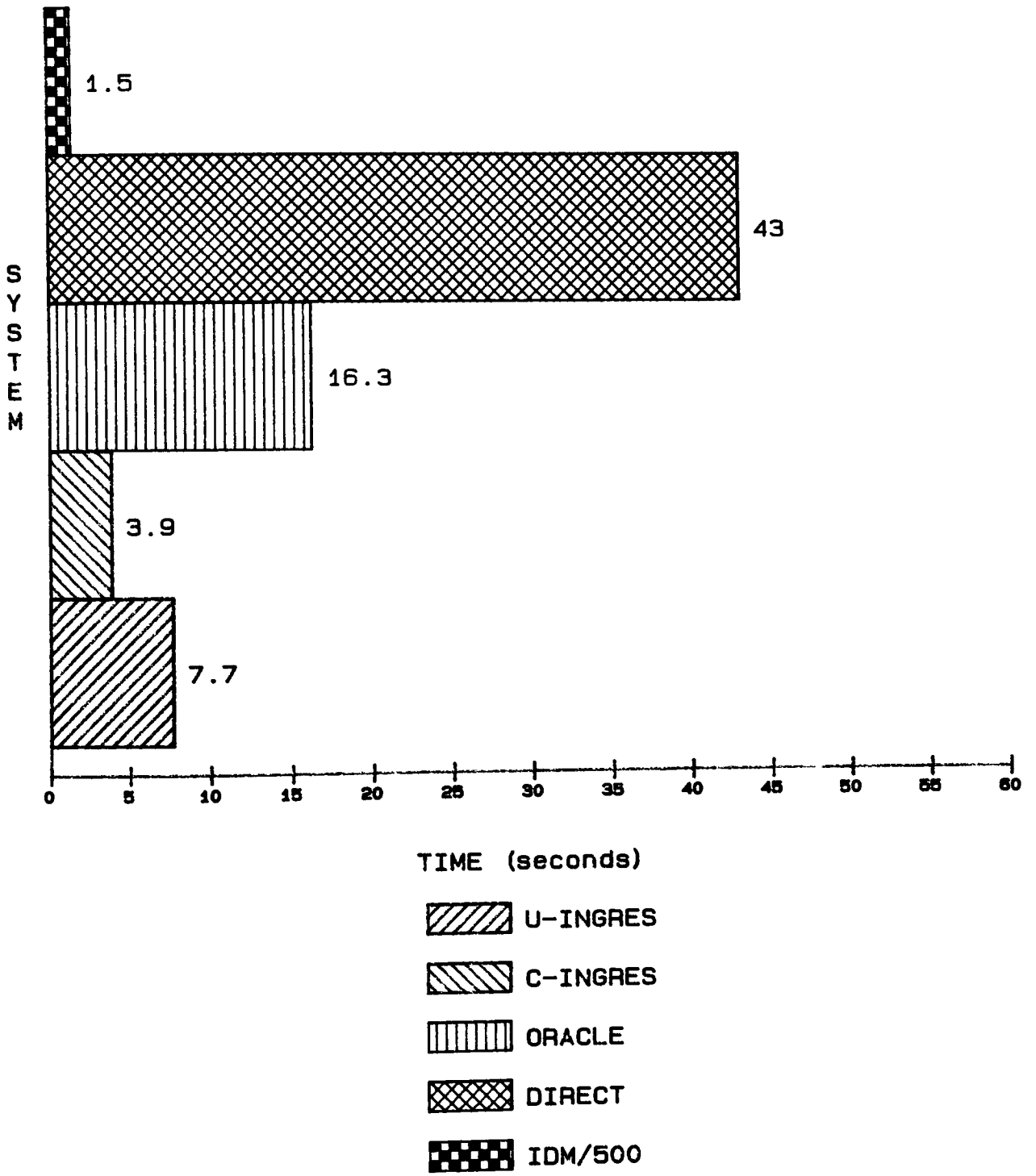


Figure 4.2.1 (a)

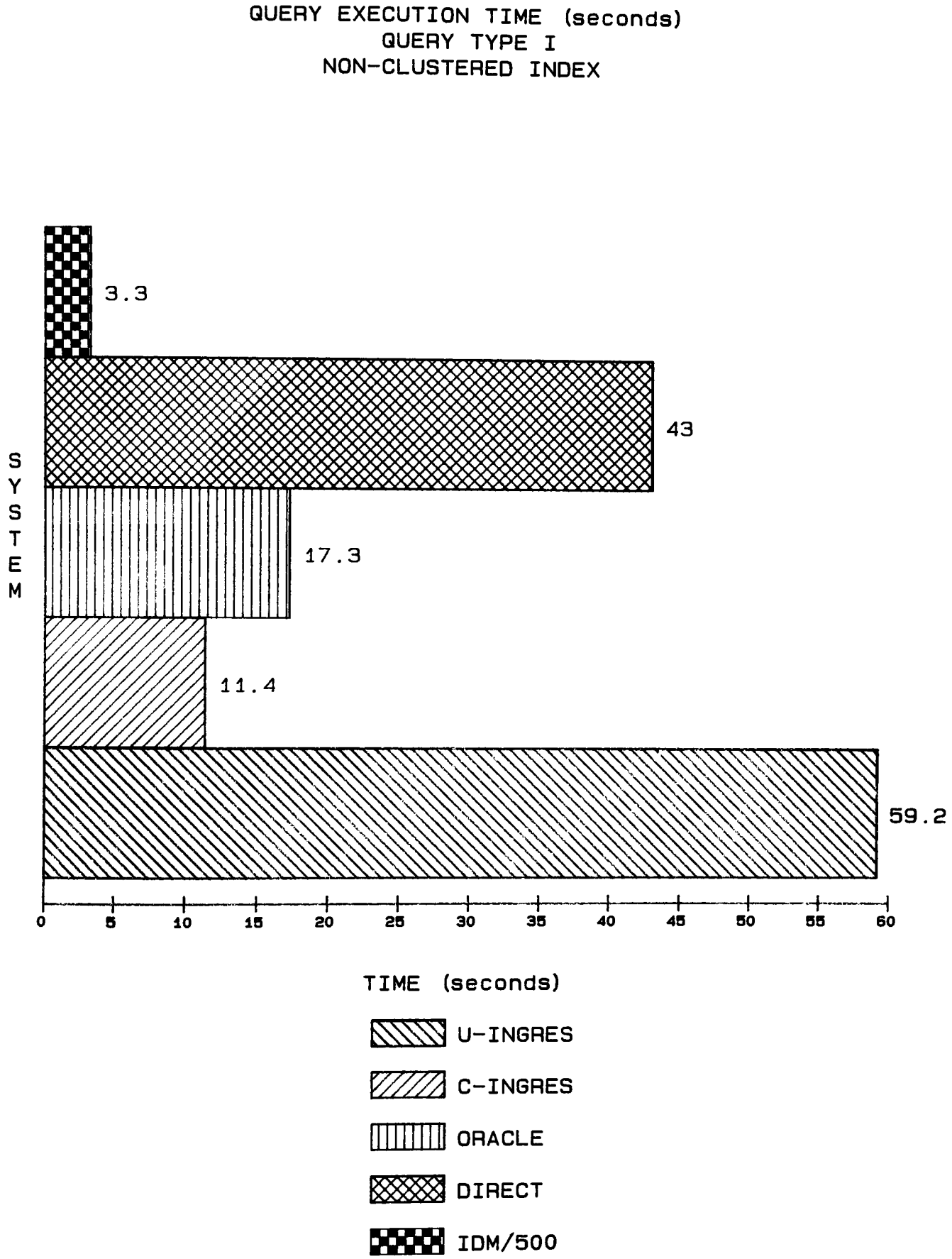

QUERY EXECUTION TIME (seconds)
QUERY TYPE I
NON-CLUSTERED INDEX
3.3
43
17.3
11.4
59.2
SYSTEM
0
5
10
15
20
25
30
35
40
45
50
55
60
TIME (seconds)
U-INGRES
C-INGRES
ORACLE
DIRECT
IDM/500

Figure 4.2.1 (b)

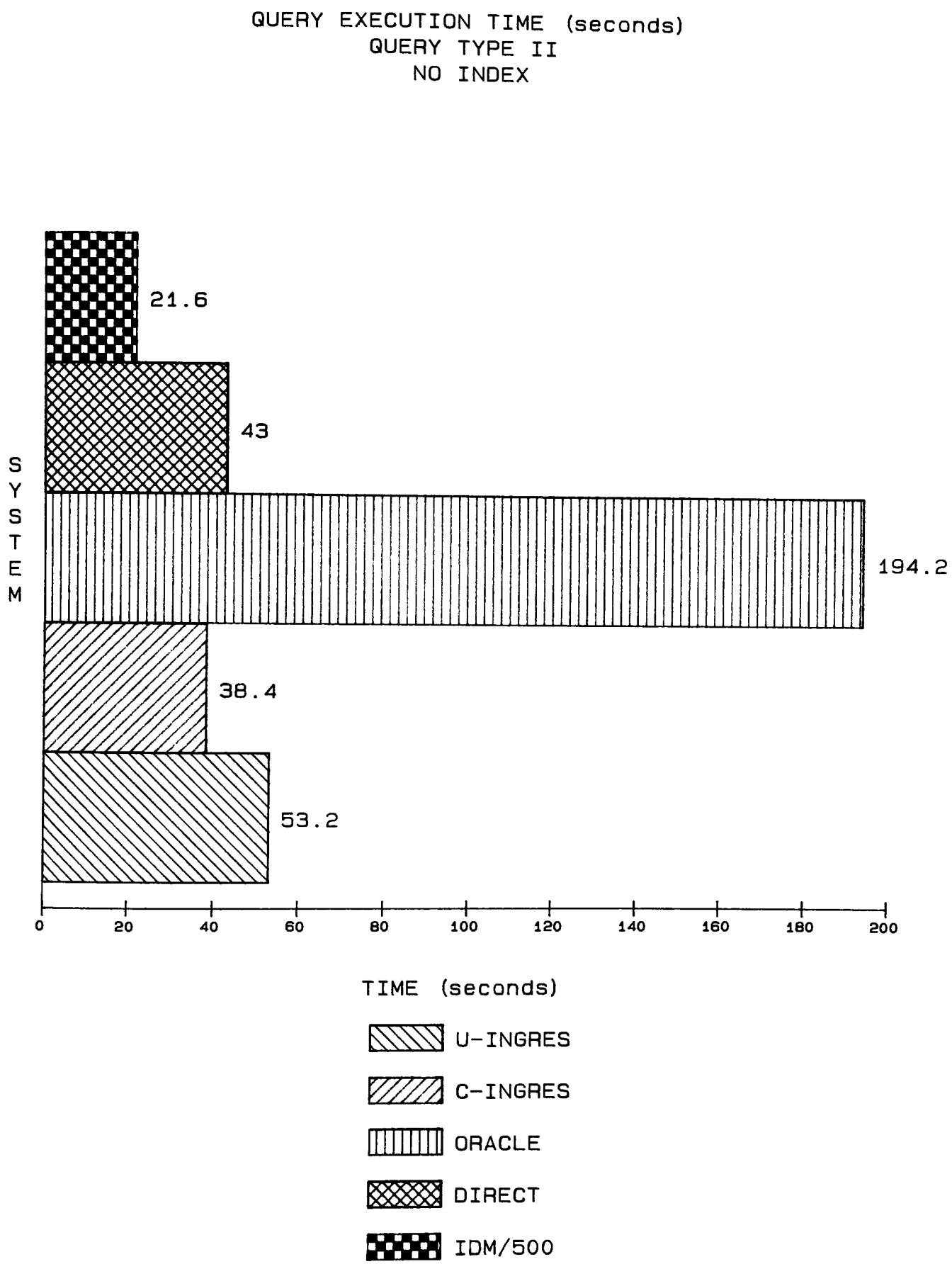

QUERY EXECUTION TIME (seconds)
QUERY TYPE II
NO INDEX
SYSTEM
21.6
43
194.2
38.4
53.2
0
20
40
60
80
100
120
140
160
180
200
TIME (seconds)
U-INGRES
C-INGRES
ORACLE
DIRECT
IDM/500

Figure 4.2.2

QUERY EXECUTION TIME (minutes)
QUERY TYPE III

SYSTEM	NO INDEX	NON-CLUSTERED INDEX	CLUSTERED INDEX
1 U-INGRES	10.20	4.49	2.11
2 C-INGRES	1.80	1.97	0.98
3 ORACLE	> 300	8.52	7.94
4 DIRECT	10.21	10.21	10.21
5 IDM/500	> 300	1.19	0.39

Table 4.2.3

QUERY EXECUTION TIME (minutes)
QUERY TYPE IV

SYSTEM	NO INDEX	NON-CLUSTERED INDEX	CLUSTERED INDEX
1 U-INGRES	9.40	10.55	9.07
2 C-INGRES	2.10	2.41	1.07
3 ORACLE	> 300	18.85	13.78
4 DIRECT	5.62	5.62	5.62
5 IDM/500	> 300	1.47	0.58

Table 4.2.4

QUERY EXECUTION TIME (minutes)
QUERY TYPE III
NO INDEX

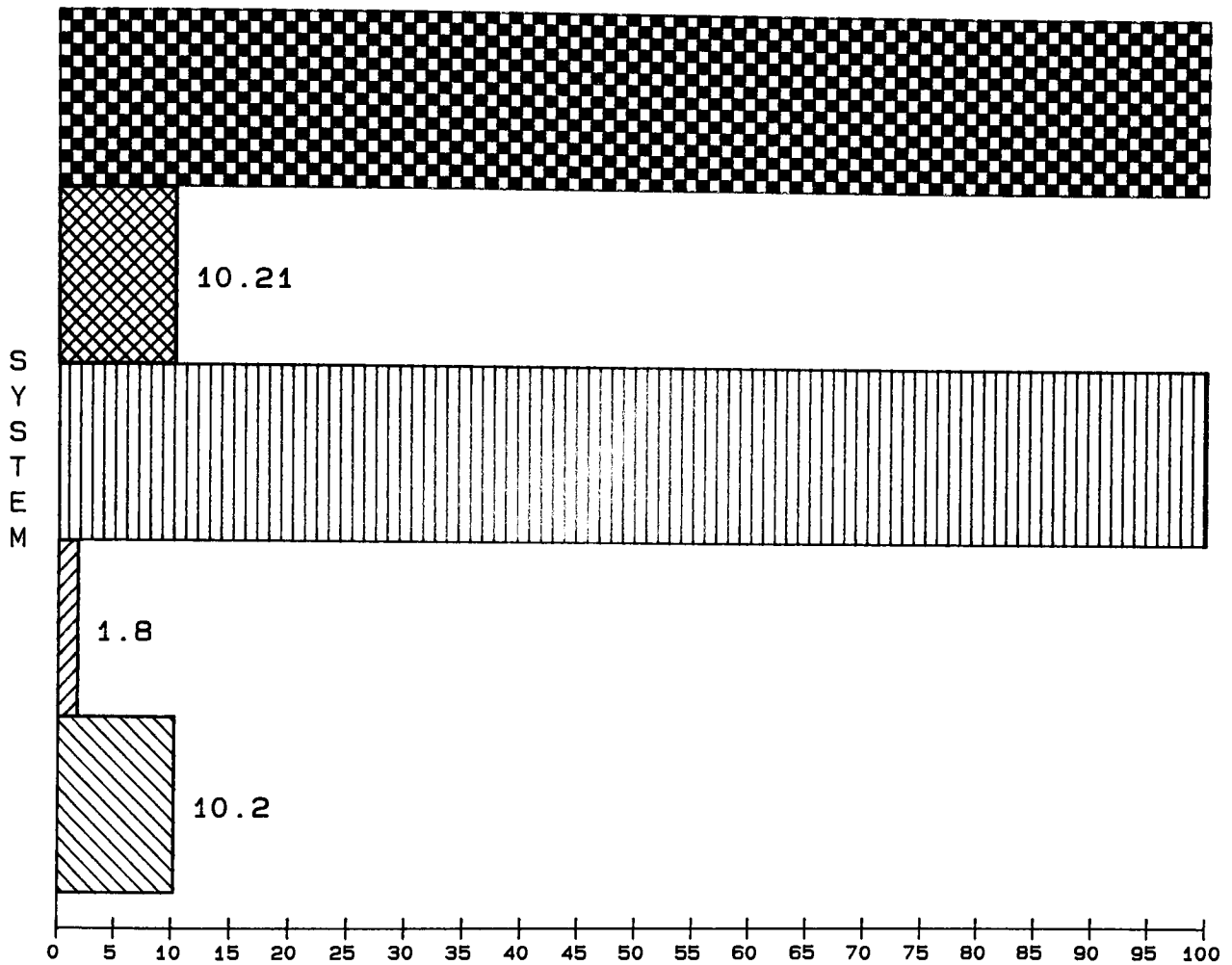


Figure 4.2.3 (a)

NOTE: Query NOT Completed by IDM/500 or ORACLE

QUERY EXECUTION TIME (minutes)
QUERY TYPE III
NON-CLUSTERED INDEX

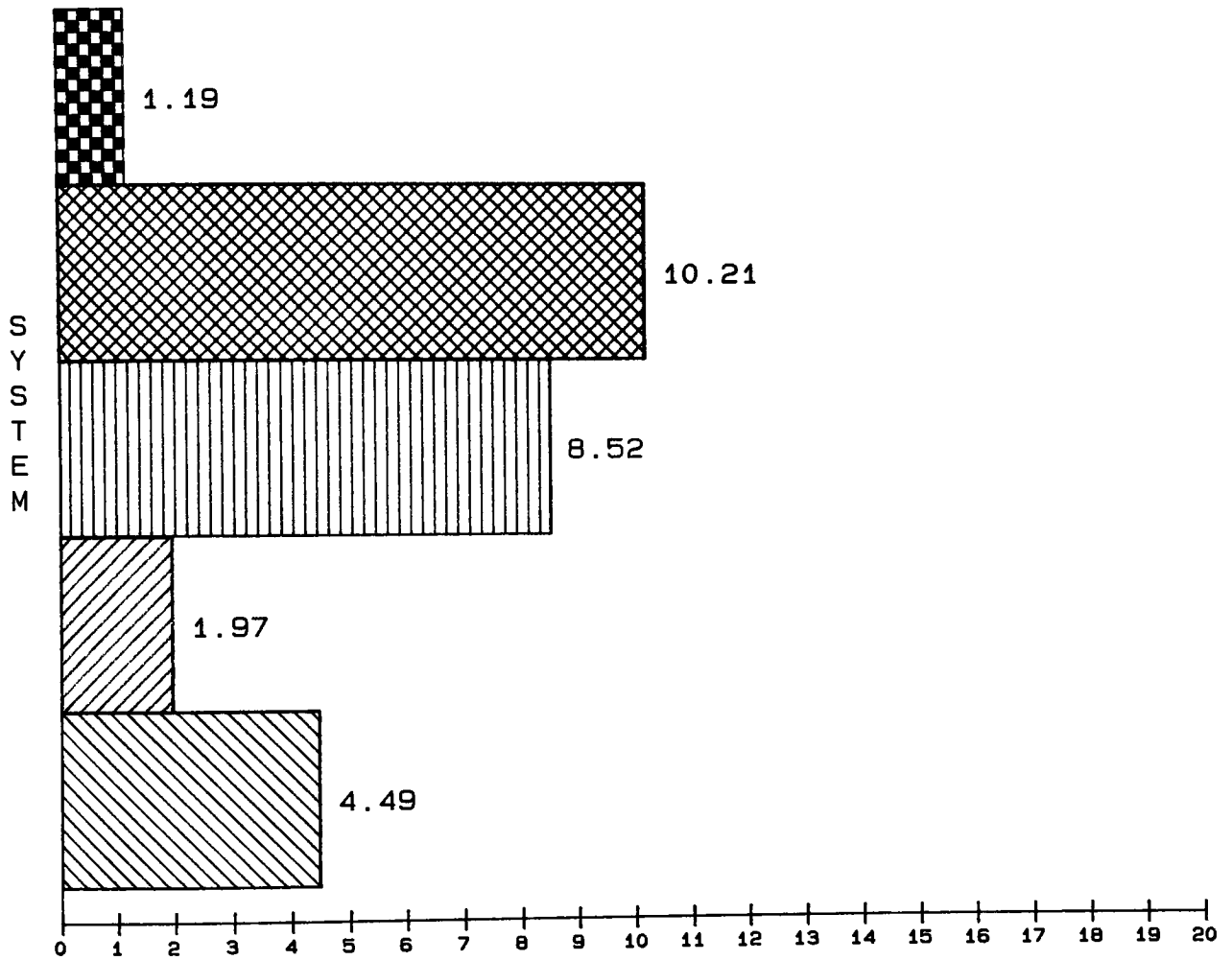


Figure 4.2.3 (b)

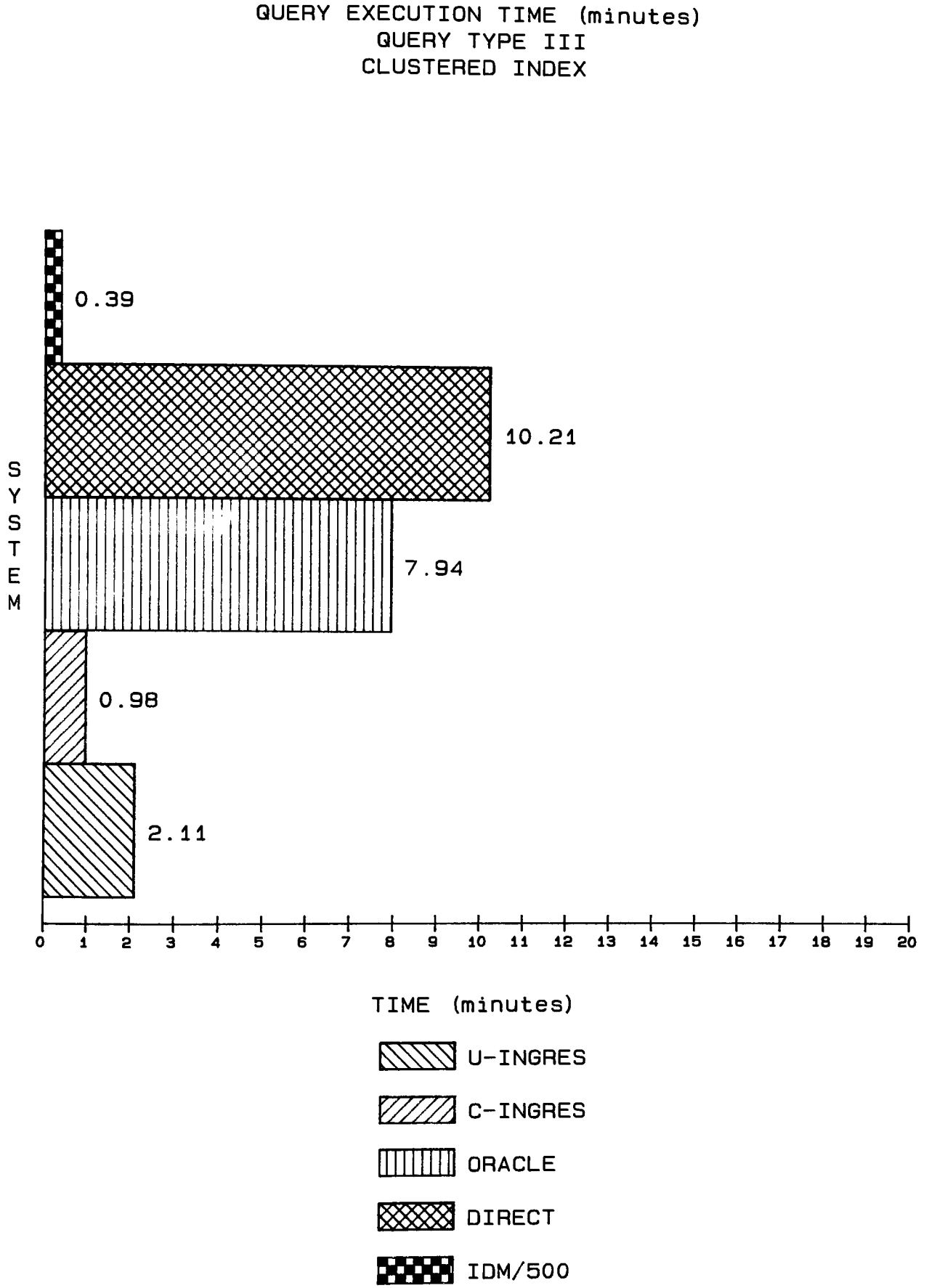

QUERY EXECUTION TIME (minutes)
QUERY TYPE III
CLUSTERED INDEX
0.39
10.21
7.94
0.98
2.11
SYSTEM
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
TIME (minutes)
U-INGRES
C-INGRES
ORACLE
DIRECT
IDM/500

Figure 4.2.3 (c)

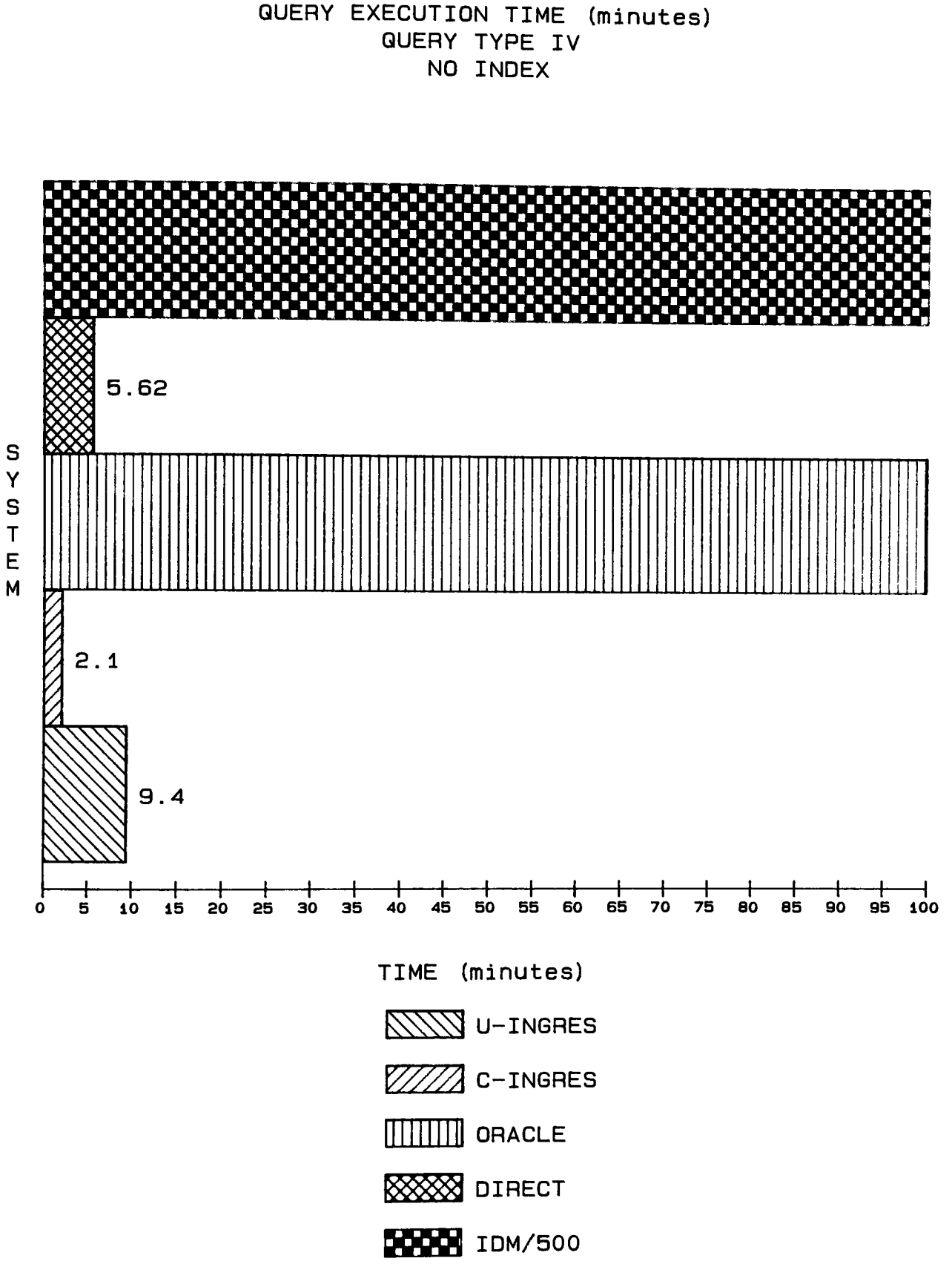


Figure 4.2.4 (a)

NOTE: Query NOT Completed by IDM/500 or ORACLE

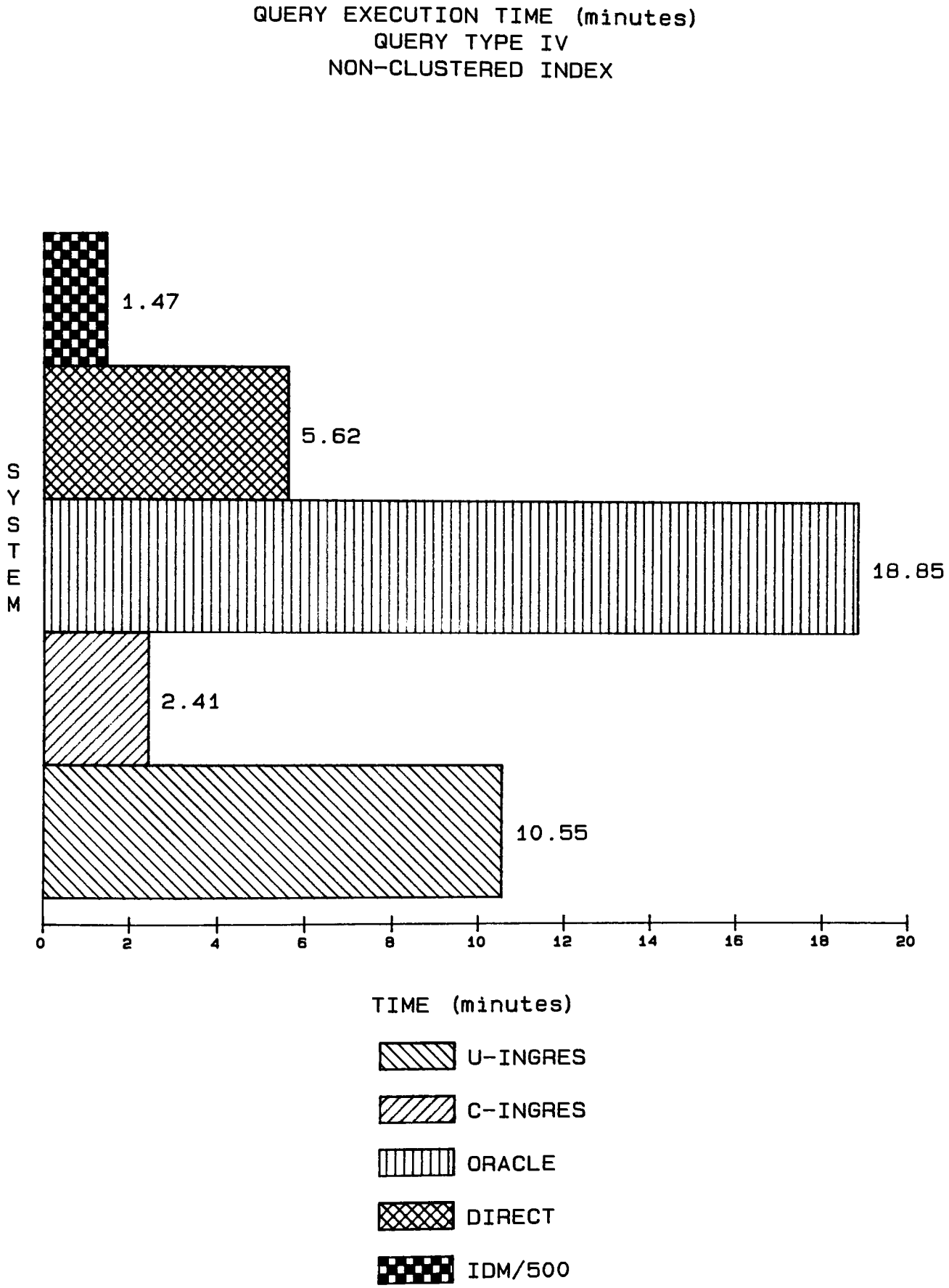

QUERY EXECUTION TIME (minutes)
QUERY TYPE IV
NON-CLUSTERED INDEX
1.47
5.62
18.85
2.41
10.55
SYSTEM
0
2
4
6
8
10
12
14
16
18
20
TIME (minutes)
U-INGRES
C-INGRES
ORACLE
DIRECT
IDM/500

Figure 4.2.4 (b)

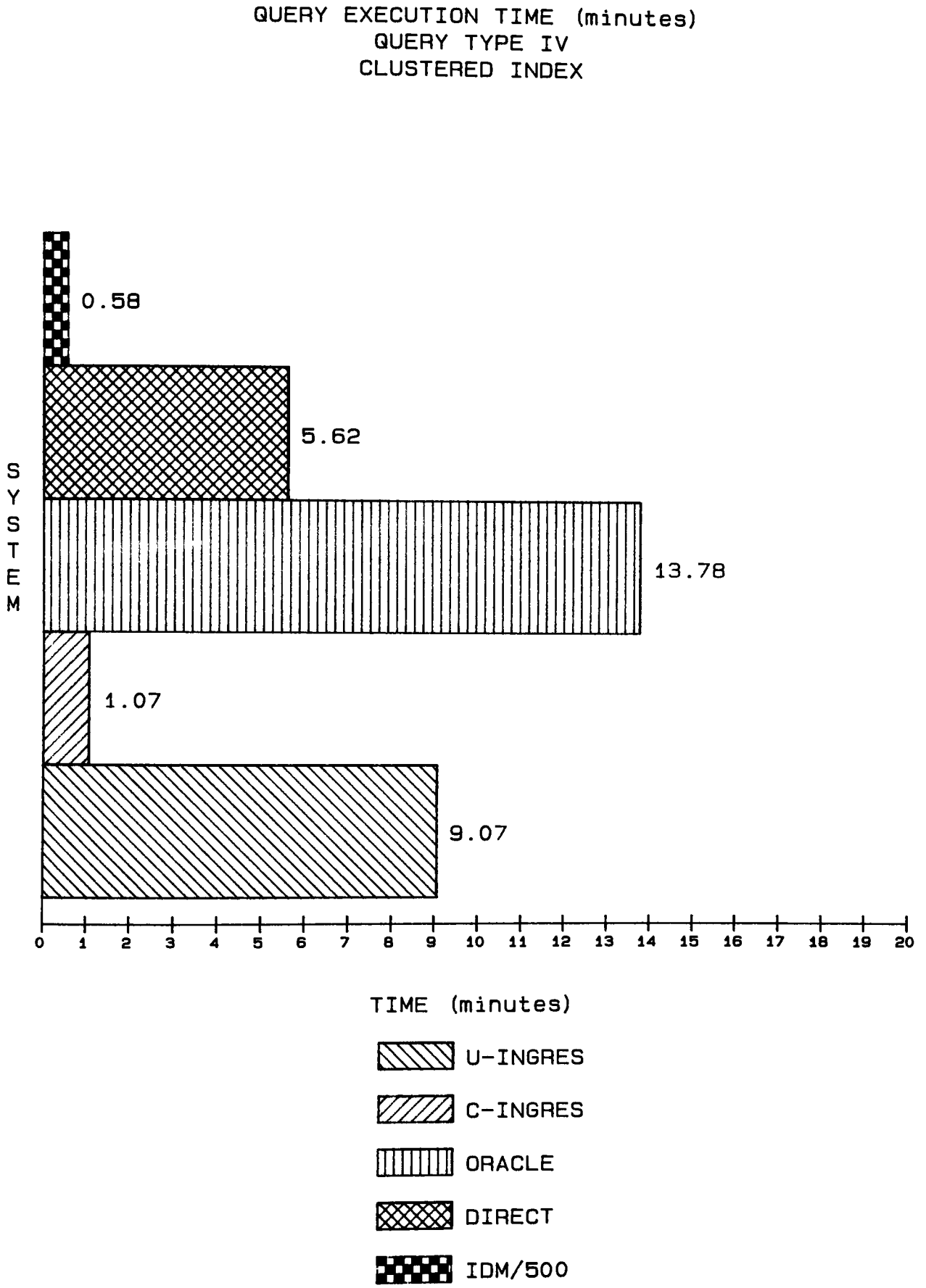

QUERY EXECUTION TIME (minutes)
QUERY TYPE IV
CLUSTERED INDEX
0.58
5.62
13.78
1.07
9.07
SYSTEM
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
TIME (minutes)
U-INGRES
C-INGRES
ORACLE
DIRECT
IDM/500

Figure 4.2.4 (c)

4.2.0. Multiple-User Benchmark Performance

This section of the paper deals with performance benchmarks of relational database systems under multiple user job loads. Here, the simple stand-alone benchmark query sets are replaced with groups of queries operating concurrently against the database store. By extending the scope of these benchmark queries, the baseline queries which primarily measure speed indices in single-user transactions, now address performance issues dealing with the utilization of system resources. These extended query sets may now provide the researcher with information concerning resource sharing, CPU utilization, disk I/O utilization, communications to/from the back-end computer (in the database machine instance) and other database system performance issues.

4.2.1. Benchmark - III

This multiple-user benchmark was run at Cornell University in 1984 [19]. The benchmark tests the capabilities of the software DBMS INGRES and the IDM/500 database machine. Two issues of database performance are developed in this study. First, the issue of measuring performance indices while varying the level of database multiprogramming is presented. Next, the issue of software DBMS performance degradation as a function of resource sharing with non-DBMS programs is discussed.

4.2.1.1. Database Systems Configuration

Very little specific information describing either database system is available. The only configuration information provided is that a DEC VAX 11/782 is used as both the host system for the IDM/500 and as the software DBMS system. The operating system is VAX VMS version 3.3 and the database was stored on similar type disks on both the host and back-end systems.

4.2.1.2. Database Context and Queries

The Wisconsin database is used in this benchmark study (section 3.1.2.). In the single-user benchmarks, retrieval queries which tested the effects of indices were run. In this multiple-user study, "representative" queries which simulate "real-life" interactive queries are substituted. These queries, numbered QRY1 to QRY9, are described below. Only the result of the query is available. No SQL code is included in the benchmark report.

QRY1: Select 1 tuple, clustered index available.
QRY2: Select 10 tuples, clustered index available.
QRY3: Select 10 tuples, secondary index available.
QRY4: Select 100 tuples, clustered index available.
QRY5: Find minimum (simple aggregate), clustered index available.
QRY6: Find minimum (complex aggregate), no indices available.
QRY7: Selection followed by join, clustered index available.
QRY8: Selection followed by join, secondary index available.
QRY9: Complex join, 2 selections followed by 2 joins, clustered indices available.

4.2.1.3. Benchmark - III Results

The first part of this benchmark deals with the effect of varying a "background" workload while running concurrently with the 9 basic queries. This is accomplished by simulating user processes with an "emulator" [22]. This emulator will submit duplicate query "scripts" for all user processes and also change the operands in each script to eliminate the possibility of data sharing among the users. Table 4.3.1 and figure 4.3.1 show the baseline execution times for the 9 queries. These times include only the execution time. All query transmission, parsing and returned data transmission times have been removed. Table 4.3.2 shows the effects of increasing the multiprogramming level from 1 to 4, 8 and 12. Each "user" process is started before the measured "foreground" process and stopped after the foreground process has completed. Table 4.3.2 shows that the INGRES system suffers an 87% degradation in performance at a high multiprogramming level while the IDM/500 suffers only a 37% degradation in performance. Table 4.3.3 shows the total number of queries processed for each "user" in the time required to complete the 9 "foreground" queries. The "user" processes executed a simple selection query. This table indicated that both systems equally share resources among its users.

The second part of the study deals with the effect of non-DBMS jobs running concurrently with the 9 basic queries,

or a subset of each. There are 3 non-DBMS programs used in this investigation. First, the I/O Program reads two 80,000 byte arrays and then writes them back out again. Second, the CPU Program runs a CPU intensive Fortran program (no page faults occur). Third, the Page Fault Program, incurrs 795 page faults for every CPU second that it runs. A disk seek is needed for every page read whereas, in the I/O program, 80,000 bytes are read/written in each seek. Interesting observations can be made when running these non-DBMS jobs concurrently with mixes of the 9 basic queries. For example, a mix of queries can be determined to be I/O or CPU bound if run first with the I/O program in one test, and then with the CPU program in another test and comparing the performance statistics of each. This study is performed with the INGRES software-only database system since query execution time on the IDM/500 is uneffected by non-DBMS job loads.

Six different job mixes are executed as the "user" processes in this test. The total elapsed time to complete the 9 queries is shown in Table 4.3.4 along with the percent degradation in throughput (queries/second) as compared to mix 2. The mixes are comprised of selection queries (QRYS), join queries (QRYJ) and the non-DBMS programs IO, CPU, and PAGEF. Mix 3 and 5 show an interesting statistic. Mix 3 which most effects the total elapsed time, had the least effect on throughput. Mix 5 which most effects throughput had the least effect on total elapsed time. One possible explanation involves the frequency at which the I/O and PAGEF programs

cause context switches. The I/O program reads large blocks of data to memory and probably never issued a context switch prior to using up its' quantum allotted by the operating system. This interferes with the frequency with which the foreground process is issued a quantum. The PAGEF program issues a context switch after 1 page of data is read to memory since a disk seek is required for each read. This does not delay (as much) the awarding of a quantum to the foreground process but does reduce the systems throughput due to the frequency of context switching.

The following observations are drawn from the data.

* Resource sharing was handled well by both systems as the level of multiprogramming increased.

* The IDM/500 outperformed INGRES by a factor of 3.4 in query execution time.

* Non-DBMS program execution seriously degraded INGRES's performance.

BASELINE TIMES FOR THE 9 QUERIES
USING INGRES AND IDM/500 (seconds)

QUERY	INGRES	IDM/500
1 query 1	1.44	0.70
2 query 2	1.25	0.70
3 query 3	36.95	1.08
4 query 4	1.32	1.04
5 query 5	37.60	21.85
6 query 6	226.22	49.32
7 query 7	2.71	1.04
8 query 8	76.25	2.05
9 query 9	4.95	21.28

Table 4.3.1

EFFECT OF MULTIPROGRAMMING LEVEL ON
TOTAL ELAPSED TIME (seconds)

SYSTEM	1 CONCURRENT USER	4 CONCURRENT USERS	8 CONCURRENT USERS	12 CONCURRENT USERS
1 INGRES	530.45	633.21	762.62	973.55
2 IDM/500	206.92	217.12	243.90	283.00

Table 4.3.2

BASELINE EXECUTION TIME FOR THE 9 QUERIES
USING INGRES VS IDM/500

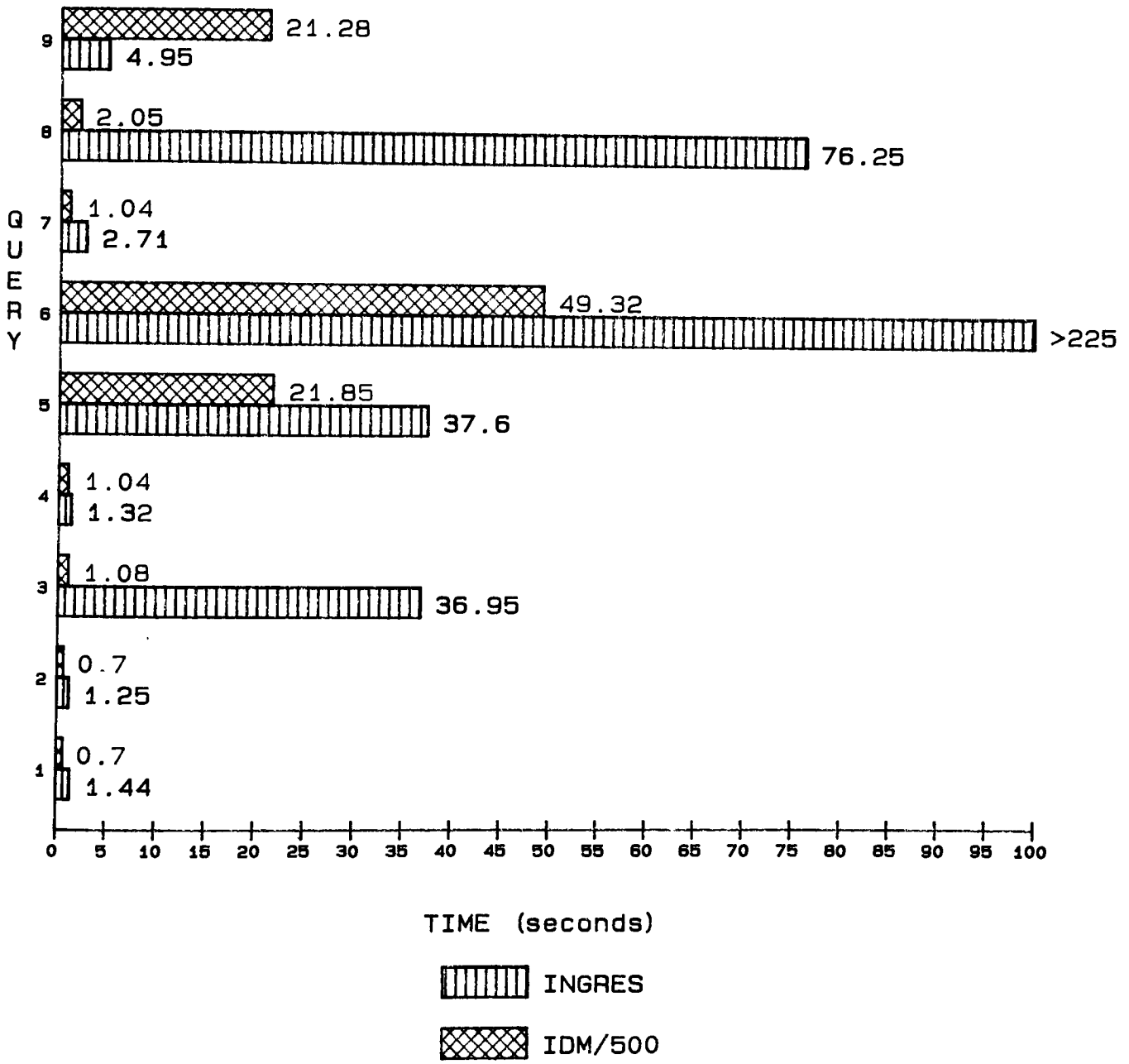


Figure 4.3.1

NUMBER OF QUERIES EXECUTED BY EACH 'USER'
(Multiprogramming Level of 8)

SYSTEM	USER 1	USER 2	USER 3	USER 4	USER 5	USER 6	USER 7	USER 8
1 IDM/500	9	20	20	20	20	20	20	20
2 INGRES	9	56	54	56	54	54	55	56

Table 4.3.3

TOTAL ELAPSED TIME (seconds) FOR THE 9 QUERIES
TO EXECUTE WITH VARYING TYPES OF BACKGROUND
WORKLOADS USING INGRES

QUERY MIX	BACKGROUND WORKLOAD	ELAPSED TIME	THROUGHPUT DEGRADATION
1	7 QRYS	762.62 (-5%)	.51 (-113%)
2	7 QRYJ	805.20 (0%)	.24 (0%)
3	6 QRYJ, 1 I/O	936.10 (16%)	.22 (8%)
4	6 QRYJ,1 CPU	862.76 (7%)	.21 (13%)
5	6 QRYJ, 1 PAGEF	833.62 (4%)	.20 (16%)
6	Diverse Mix	1226 (52%)	.15 (38%)

Table 4.3.4

4.2.2. Benchmark - IV

This benchmark study was performed in 1985 and compares the software DBMS INGRES with the Briton Lee IDM database machine [30]. This study, unlike the previous three, was performed by a corporate IMS group (at Citicorp Corporation) and provides a recommendation based on initially defined requirements and price/performance. This study is included in this paper because it provides the reader with an opportunity to focus on "real life" considerations in selecting a database management system. The previous studies are more theoretical in nature and attempt to develop and define a methodology for benchmarking a DBMS. This benchmark, on the other hand, is designed for a very specific application and therefore it's results must be viewed in that context.

The designers of this benchmark had some interesting requirements for the DBMS to be selected. First, the database system must operate within a VAX environment. Citicorp obviously has an investment in VAX architecture. Also, the system must support high level programming languages and contain a "powerful" report writer. This suggests that normal operations will require batch type jobs to periodically summarize and report information from the database. Finally, the DBMS must support a large number (50-100) of simultaneous users operating with databases in the gigabyte (10^9) range. This is especially interesting since none of the previous

studies approached even half of this workload.

4.2.2.1. Software-Only System Configuration

The software system used in this benchmark is run on a DEC VAX 11/785. The VAX has 12 Mbytes of main memory and runs the VMS version 3.6 operating system. Four DEC disk drives, for a total of 1.3 Gbytes, serve as the secondary memory devices. INGRES version 3.0 serves as the DBMS for this configuration. INGRES satisfies all of the initial requirements set down by the benchmark designers except one. Prior to this test, no documented experience with gigabyte size databases existed. This will prove to be a major point in the designers recommendation.

4.2.2.2. Database Machine System Configuration

A Britton Lee IDM-500/2 is used as the back-end database machine for this benchmark (note: the IDM-500/2 contains the faster (10mhz) database processor and 2 Mbytes of primary memory). An additional Mbyte of memory brings the total to 3 Mbytes. Two disk controllers are needed to handle the 5 CDC 9766 (300 Mbyte) disk drives. The IDM user interface is replaced, in this study, by Signal Technology Inc. OMNIBASE version 3.31. Apparently, the need for a more friendly user interface is also a requirement for this benchmark.

4.2.2.3. Database Queries and Context

The database consists of relations with equal cardinality. A primary key attribute and four non-key attributes make up

the 25 byte record. The primary key is 12 bytes long and the non-key attributes are 5,4,2, and 2 bytes long respectively. Two large relations, TAB2 and TAB3 each contain 98,246 records. Relation USER "n" (where "n" is the user number) is the result relation modified by the "user" process. Up to 40 users are simulated in this benchmark. Relations USER1, USER2,..., USER40 all contain 335 records. To raise the multiprogramming level, additional "users" are introduced into the job stream. For example, say USER 20 is to be added. A join of TAB2 and TAB3 is performed and then a selection based on some primary key value not contained in USER20 is made. The resultant tuple (record) is first appended to relation USER20 and then deleted. All user processes act in the same way. All user processes perform 10 join, append and delete operations before completing. The benchmark contains results of 10,20,30 and 40 "user" processes.

The user processes are submitted through a batch queue with a job limit = 40. Although submitted sequentially, the processes are allowed to execute concurrently. Since all user processes perform the same join and selection query string, an average query execution metric can be calculated. The average, maximum and minimum query execution times are provided in the benchmark summary.

4.2.2.4. Benchmark - IV Summary

The VAX system clock is used to monitor and record the elapsed clock time and elapsed CPU time for the user

processes. It should be noted that the IDM/500 values contain the delay times to transfer the query to the IDM and the time to transmit the data back to the VAX. Table 4.4.1 shows the average, minimum and maximum time (both clock and CPU) for the software-only database system. The missing values, for the 40 user level, reflect a problem encountered by the benchmark group. When attempting to run INGRES with 40 simultaneous users, the system crashed. This turned out to be a system resource limitation when running a high number of concurrent processes with INGRES. The failure occured in INGRES and not in OMNIBASE because INGRES spawns a second process for every interactive or batch process on the system. The result is a lack of memory to create process PCBs. This can be corrected by increasing a system resource parameter, however, this will cause other non-DBMS jobs to page fault more frequently (especially at 100 users!). Figures 4.4.1 and 4.4.2 show the minimum and maximum CPU and clock times for both the IDM/500 and INGRES systems.

Table 4.4.2 shows the elapsed clock and CPU times for the IDM/OMNIBASE system. The IDM, working independently from the VAX operating system, has no problem with 40 concurrenct users. The data indicates that the database machine significantly outperforms the software-only system at all multiprogramming levels. Figure 4.4.3 shows the average elapsed CPU times for both the IDM/500 and INGRES systems.

The following conclusions and recommendations are included

in the benchmark study.

* Both systems could have been tuned for the benchmark tests. It is assumed that both could have been equally improved through this tuning so running the "untuned" systems was considered equitable.

* INGRES and OBMIBASE support the same data types except OMNIBASE also supports packed decimal fields. This feature will save huge amounts of storage space for the application which is to be supported.

* The costs of both systems are almost the same.

* Both systems share resources equally at all multiuser levels.

* The IDM/500 with OMNIBASE is recommended because of its successful government installations and because of its performance compared to the INGRES solution.

INGRES - AVG,MIN AND MAX TIME (CLOCK & CPU)
TO COMPLETE THE USER PROCESSES
(seconds)

# OF USERS	AVG. CLOCK	MIN. CLOCK	MAX. CLOCK	AVG. CPU	MIN. CPU	MAX. CPU
10	10.43	7	15	0.81	0.59	1.13
20	19.68	5	32	0.83	0.61	1.60
30	29.84	7	55	0.82	0.44	1.28
40						

Table 4.4.1

IDM/500 - AVG,MIN AND MAX TIME (CLOCK & CPU)
TO COMPLETE THE USER PROCESSES
(seconds)

# OF USERS	AVG. CLOCK	MIN. CLOCK	MAX. CLOCK	AVG. CPU	MIN. CPU	MAX. CPU
10	2.64	0.56	4.76	0.18	0.14	0.22
20	7.11	0.88	15.19	0.19	0.15	0.24
30	6.68	0.56	17.58	0.19	0.14	0.55
40	10.17	0.61	38.12	0.18	0.07	0.23

Table 4.4.2

MINIMUM & MAXIMUM CLOCK TIME (seconds)
OF INGRES AND IDM/500 SYSTEMS

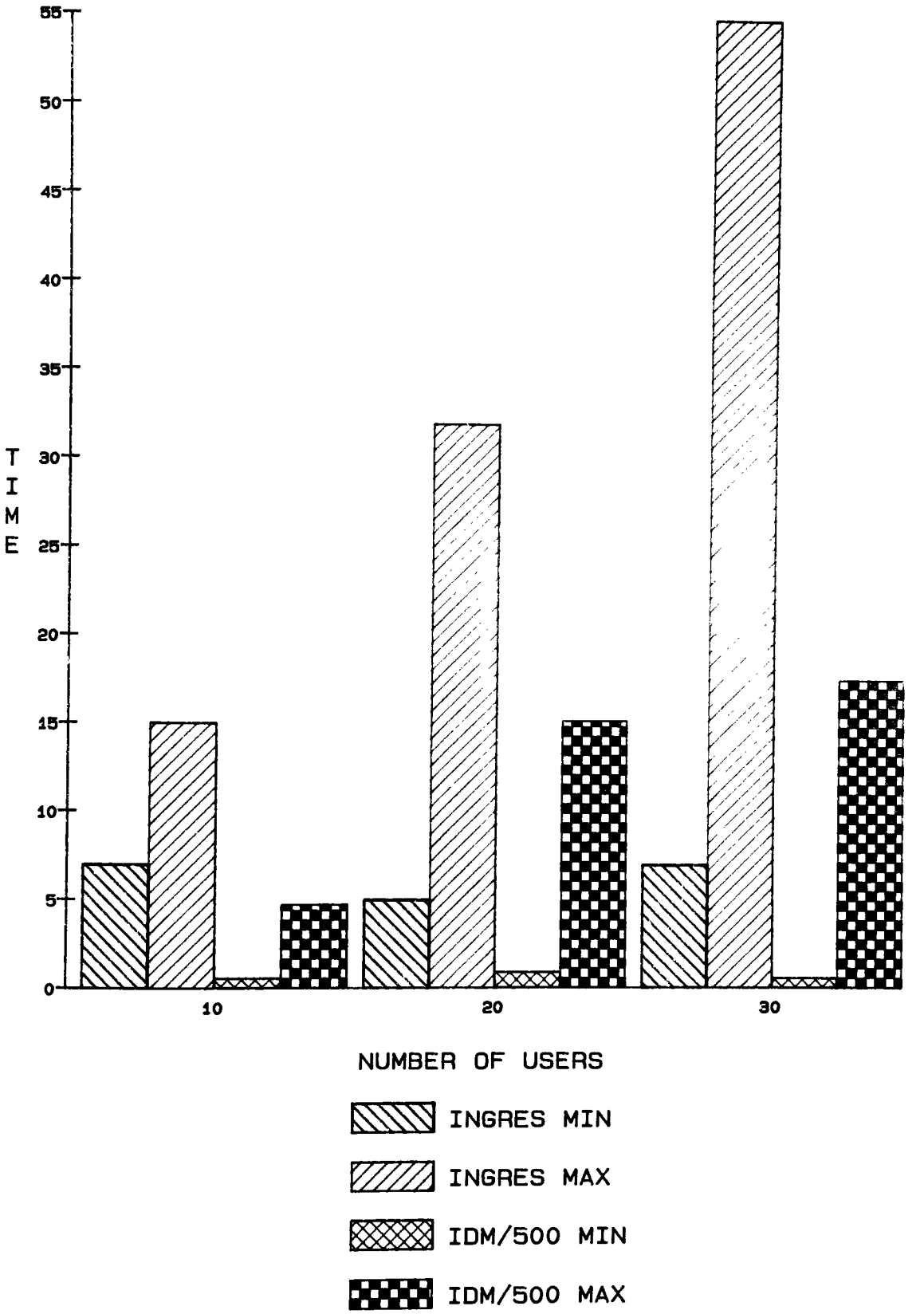


Figure 4.4.1

MINIMUM & MAXIMUM CPU TIME (seconds)
OF INGRES AND IDM/500 SYSTEMS

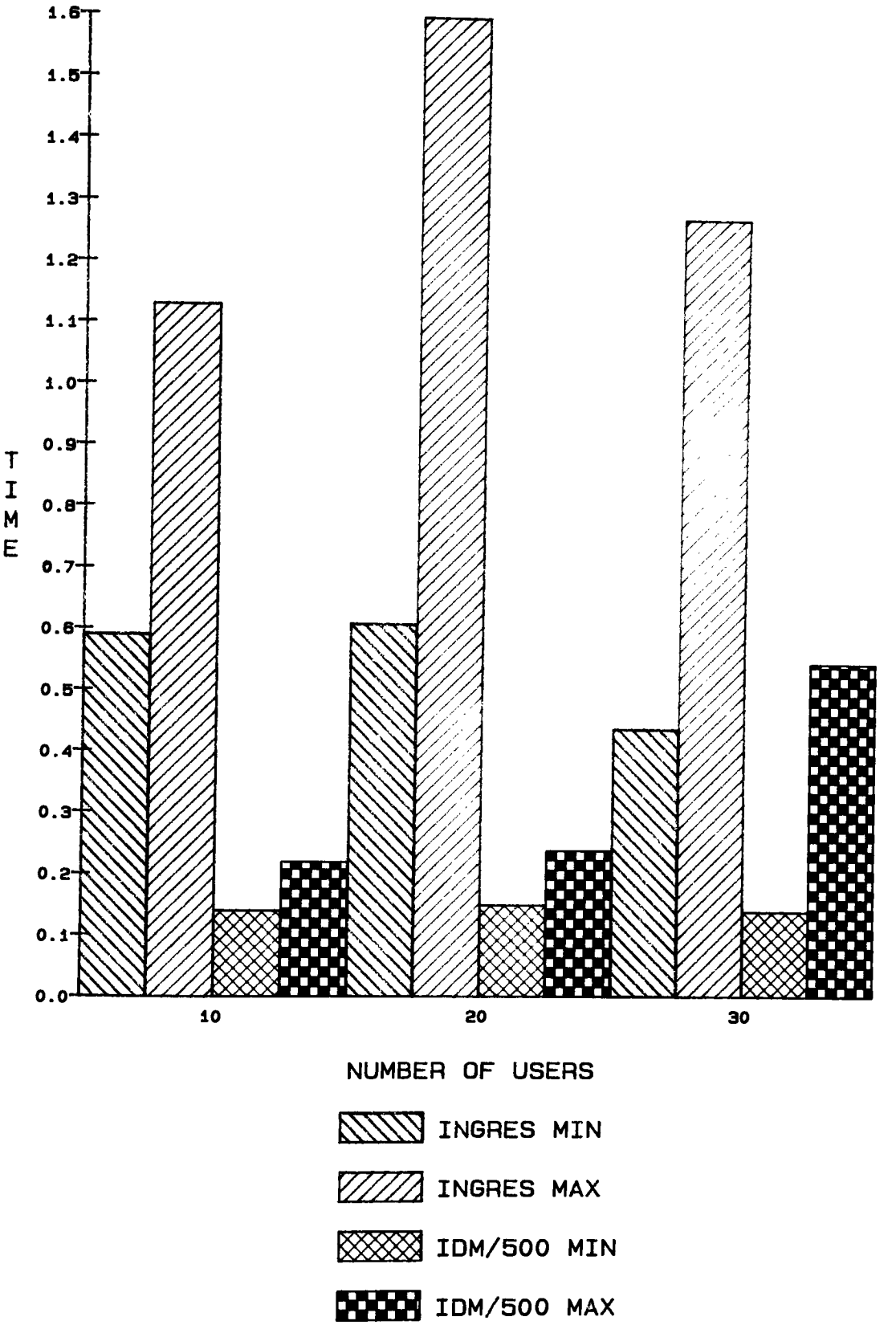


Figure 4.4.2

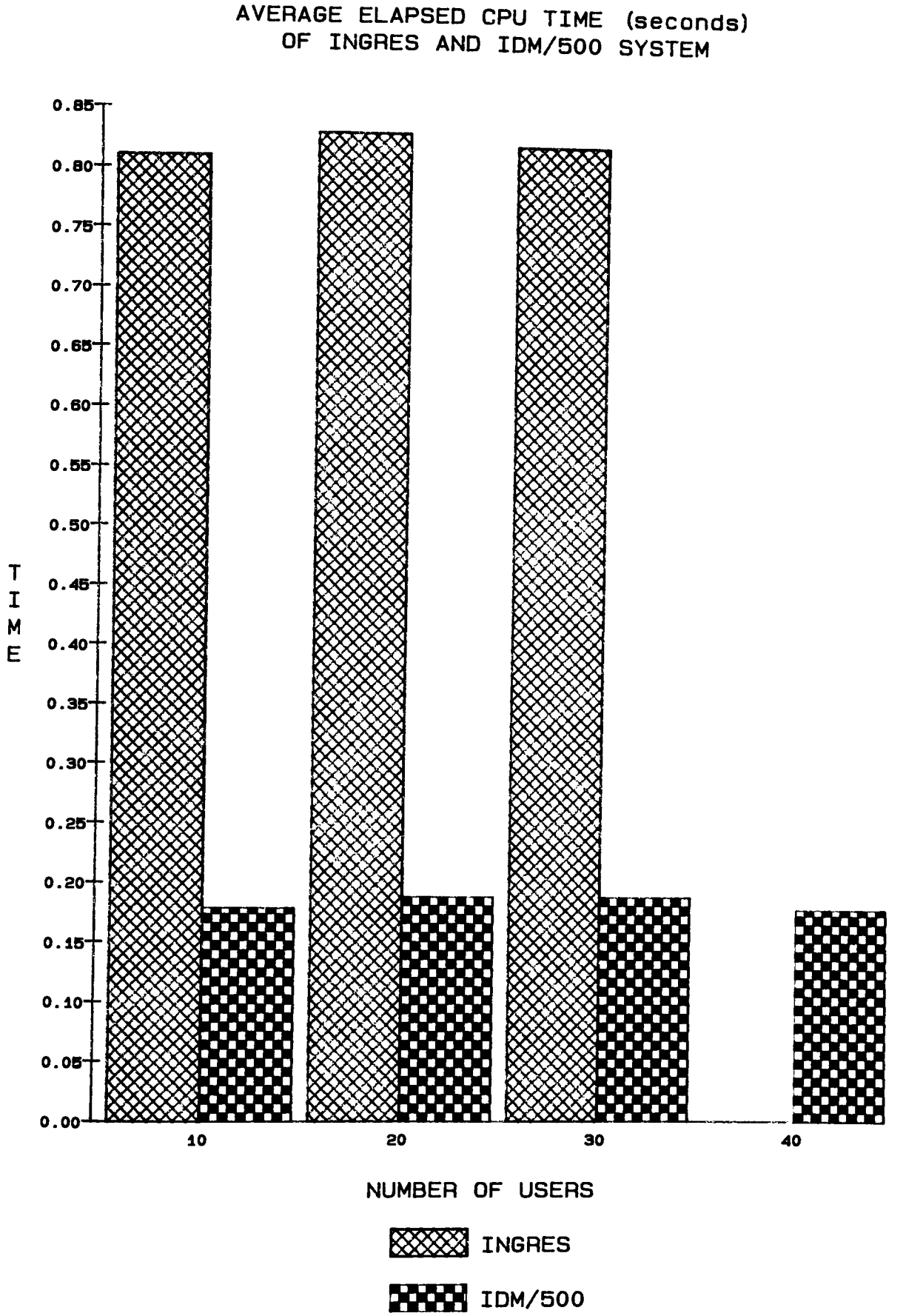

AVERAGE ELAPSED CPU TIME (seconds)
OF INGRES AND IDM/500 SYSTEM
TIME
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
10
20
30
40
NUMBER OF USERS
INGRES
IDM/500

Figure 4.4.3

4.3.0. Benchmark Analysis Summary

Across all benchmarks considered in this paper, the IDM/500 database machine was clearly the top performer. In benchmarks - I and II where system speed is of primary interest, the IDM/500 showed itself to be significantly superior to the software systems. These benchmarks tested the effects of providing different levels of indexing within the benchmark query sets. When some type of index was available for use by the IDM system, acceptable performance was observed. There were cases of failure however. When the IDM executed queries without the benefit of any index, a minimal improvement over the software system was gained. In fact, if the query is complex (Type III, Type IV) the system failed to complete the task! This indicates that if a "reasonable" database design is followed, the IDM system will outperform the software-only systems. Of the software systems tested in these benchmarks, the Commercial version of INGRES was the best performer. One improvement made in the Commercial version of INGRES over the University version was in the buffer management of data. One shortcomming in U-INGRES was in its reliance on the operating system to manage data for the query sets. In benchmark - II, Unix was the operating system running U-INGRES which implements an LRU paging algorithm. When queries performing joins were executed a significant improvement is observed in C-INGRES over U-INGRES due to it's improved buffer management capability.

The other database machine included in these benchmarks is DIRECT. This machine was consistently slower than the IDM/500 except in the case when no index was available in the query. DIRECT does not use indices. Instead, as a replacement to indices, a group of processors working in parallel are used to execute the query. DIRECT was capable of completing the complex queries when both the IDM/500 and INGRES systems failed.

In the multi-user benchmarks - III and IV, the primary emphasis changes from that of speed to workload. Here, groups of query sets are run against a database to measure the effects of resource sharing among users and non-database jobs running concurrently with database jobs. In these benchmarks the queries vary in complexity from simple selections to multiple joins. Again, in almost all cases, the IDM/500 system executed the queries faster and supported more users than the software systems being tested. Also indicated in these studies is the ability of both the software and IDM/500 systems to share resources among users. The results indicate that the average elapsed CPU time remained about the same for each user's process regardless of the number of processes running.

The benchmarks included in this paper show a clear advantage in performance with the IDM/500 over other widely used software DBMS systems. It is also clear that no consistent method of measuring performance is evident within

these studies. There do seem to be similarities between the benchmarks however. This indicates that perhaps the DBMS benchmarking community is closer to a standard performance methodology than realized. This point is persued in the next chapter.

Chapter 5

Summary of the Research and Lessons Learned

This chapter will discuss, in summary, the observations and results of this thesis research. Additionally, a discussion of possible benchmark guidelines, developed through this research, is presented.

5.1.0. Summary of Thesis Research

This thesis has reviewed various performance issues surrounding relational database technology. In chapter 1, the inherent benefits and liabilities of the relational model is discussed and the database machine, which is one possible solution to these performance liabilities, is introduced. In chapter 2, one such database machine, the IDM/500 system from Britton Lee Inc., is discussed in detail. This discussion included both a hardware description, including peripheral devices, and a software description of both the host software and the database machine software operations.

The issues of database performance measurement methodologies and database machine architectures is discussed in chapter 3. It is in this chapter that specific definitions of performance metrics and methodologies are presented. At this point in the thesis, a concern in the performance

benchmarking methodologies is identified. No standard method of benchmarking relational database systems exists. Several topics in benchmarking methodologies are discussed such as single-user tests, multiple-user tests, test database design and context and other related topics. Also introduced in this chapter, are the concepts of application dependent benchmark testing and application dependent database machine design. The thesis then turns to a discussion of results from several actual benchmarks in chapter 4. A description of the benchmarks further supports the concern raised in chapter 3 on the lack of a standard benchmarking methodology for relational database systems. The summary of results from these benchmark studies indicates that the IDM/500 database machine provides some definite benefits in relational database processing but a clear need in developing a standard benchmarking methodology is evident.

5.1.1. Results of Thesis Research and Lessons Learned

The hope of developing an accepted relational DBMS benchmark methodology was evident in many of the benchmark reports encountered in this thesis research. Similarities between benchmark reports quickly became apparent. The effect of indexing, for example, is a very common test among the benchmark methodologies. This suggests that the benchmarking community agrees that this is an important characteristic of a relational DBMS being tested. Most benchmark reports also acknowledged that the multiple-user benchmark is ths most realistic approach in benchmarking methodologies. This infers

that a multiple-user simulation would offer the best results as a standard methodology. Another important characteristic which is common to many of the benchmark reports, is the performance measurement metric. Query execution time, which eliminates terminal I/O overhead, parse time and transmission delays, seems to be an agreed upon technique for calculating a system's speed. In the multiple-user case, this not only includes the pure system speed but also includes a measure of the ability of a relational DBMS to share system resources across different multiprogramming job loads. The last frequently encountered characteristic in these benchmark reports is the database itself. The Wisconsin database [15], which is one of the earlier designed databases, is frequently used as a synthetic, general purpose database. It's design allows for the easy control over selectivity levels and types of needed data (ie. numeric or text) to test a given query.

The single most variable element in the benchmarks encountered, is the list of queries to be executed. Many benchmark reports included a vast list of different queries designed to measure the speed of specific relational operations. While these lists were very extensive and thoroughly tested the capabilities of a relational DBMS, they were also overwhelming. Too many queries were directed at specific relations in the database or tested hardware capabilities of memory expansion or additional disk I/O channels. These topics, although pertinent to the specific benchmark study, are much too specific to be considered as a

standard methodology. A promising approach is outlined in [29]. Database applications can be categorized into three groups. These are primarily compute bound, I/O bound and a mixture of both. If three methodologies were developed to address each of these categories, a useful and manageable benchmark could be devised for the majority of all relational database applications. This is similar to the benchmarking techniques used for conventional computer systems. To intelligently rank an entire system based on one test would be difficult if not impossible. Rather, a series of accepted benchmarks are run to collectively evaluate a given computer system. A perspective user of a computer system can determine, based on her/his own criterion, if a system is adequate for her/his application. In a like manner, a relational DBMS could also be evaluated and selected based on the required support for an application. This could be taken one step further by also developing a standard methodology of matching a user's application into one of the three standard groups. This methodology could include a checklist of required support items as well as sets of queries and programs (such as PAGEF and I/O [19]), which could be selected and run as "representative" of the actual application. In addition, the number of anticipated users could be simulated based on the user's selection. In this manner, the methodology could quickly categorize the actual application into one of the three standard groups. At this point, the appropriate benchmark could be run yielding informative statistics to the

user.

5.2.0. Proposed DBMS System Evaluation Guidelines

The benchmark studies included in this thesis were designed to measure the performance of a relational DBMS. These studies included software only systems and database machines. Any generally acceptable benchmark methodology should be designed to be applicable to any software or hardware implementation of a relational DBMS. The benchmark then, should test the functionality of the DBMS and not the systems' implementation strategy. In this way, any relational system may be fairly compared to any other relational system. This policy will be followed in this section addressing relational database system evaluation.

5.2.1. Topics to be Addressed in the Guidelines

A vast list of specific topics can be generated when considering the evaluation of database implementations. This is not the preferred strategy within these guidelines. These guidelines are directed at an end user evaluation of performance so only critical issues are important here. These issues include the performance metric to be used, the level of multiprogramming to be considered, the effects of indexing on the query sets, the classification of a perspective application and the database to be used. Another group of related topics should also be considered when evaluating a relational DBMS. This group of issues includes those topics

which are not directly related to performance but are very important when selecting a DBMS. Database security, maintenance, conformance to the relational model and others will be included in this discussion.

5.2.2. Query Execution Time

The time based metric to evaluate a systems' performance in these guidelines is Query Execution time. As described in 3.2.1., this metric includes only the time for the DBMS to process a given query. Query transmission, parsing, and data return time are intentionally omitted from the measurement. It is assumed that the DBMS host computer as well as any database machine being considered provides a mechanism to monitor just the query execution time apart from the other delays. If not, the fair comparison of relational DBMS systems is made extremely difficult.

5.2.3. Multiprogramming Level

This guideline considers only the multiuser case in relational DBMS performance evaluation. It is recommended that a method to control the initiation and termination of concurrent processes on the DBMS be utilized. One method, discussed in [22], is a software program which emulates DBMS users. This emulator provides a convenient way to vary the concurrent workload on the DBMS while measuring system performance during the execution of a given query or query set. If duplicate queries are submitted by multiple

processes, the emulator also provides a method of measuring the ability of the system to share system resources. It also removes the need for actual users on the system. This is especially useful if the queries require a long time to complete.

Another important use of the emulator is in the determination of a query set's effect on the system. This benchmark guideline will provide test examples of usable query sets to evaluate a relational DBMS. However, the benchmark user is encouraged to develop his/her own queries to more closely simulate the actual anticipated workload of their own application. A good way to determine if a query or query set is, for example, I/O intensive, is to submit it using the emulator along with a known I/O bound query and monitor the performance. If little degradation in the known I/O query is realized, then the test query is probably not I/O intensive and is more CPU intensive. The same method may be used to determine CPU intensive queries. This will be very important when classifying user applications.

5.2.4. The Effects of Indexing

The use of indices is very important in relational database design. A large improvement in system performance may generally be seen when issuing a query with an index over the same query without the benefit of the index. All of the published benchmarks encountered in this thesis research discussed the use and effect of indices.

This guideline includes the topic of indexing. Test query sets will be included which make use of no indices as well as the use of a primary index. The use of secondary indices will not be included due to previous benchmark reports that indicate a minimal improvement using secondary indices if a primary index is available. The point here is to find out if the DBMS in question uses indices to its advantage. DIRECT, discussed in 4.1.2.5., is one example of a DBMS that does not use indices at all.

5.2.5. User Application Classification

The types of queries issued by a DBMS user changes depending upon the application being supported. Statistical applications, for example, tend to be CPU intensive when compared to a bibliographical application which is more I/O intensive. A perspective user of a statistical system, then, is probably more interested in the performance measurement of a relational DBMS while executing CPU intensive queries. There are also applications which require a mix of queries to support the workload. These three general classifications, as discussed in 3.3.3., will serve as the guideline categories for user applications. Queries which more closely simulate the expected workload for all three classifications will be provided. It is reasonable to assume that a DBMS user would know the common types of queries to be issued in his/her own application without knowing if their application is CPU or I/O intensive or a mixture of both. The following methods should

aid in the determination of their application category.

As discussed in 4.2.1.3., programs known to be CPU or disk I/O intensive may be used to facilitate the classification of a user's application. These programs offer the user an opportunity to evaluate the effects of non-DBMS jobs on the database host computer. They also provide a means of determining the workload characteristics of a given query or query set. The user of the benchmark strategy should be able to provide "representative" queries which would be common to his/her own application. These may include simple single queries, complex queries or groups of queries which would logically be run together (ex. standard report generation). Once the type of query or query sets are defined, they could be run against the database to establish a baseline timing index to be used for comparison. After the baseline is established, the query(s) would then be re-run concurrently with the known CPU intensive job [31]. If the baseline is "considerably" shorter than the new execution time, then indications point to the query as being CPU intensive. If minimal effect is observed on the baseline time, then the query is probably more I/O intensive. To verify which, the query(s) may be run again concurrently with the known I/O intensive program. Again, if the new execution time is not much different from the baseline time, then the query is verified as CPU intensive. Otherwise, if the new execution time is considerably longer than the baseline time, then the query is verified as more I/O intensive. If neither the CPU

nor I/O programs have much of an impact, then the query may be considered equally CPU and I/O intensive.

In this way, a user of these benchamrk guidelines may determine if his/her application is more disk I/O or CPU intensive. If, after identifying representative queries, the user determines that his/her application is CPU intensive, the classification of the application is "Statistical Analysis System" [29]. If the application is more I/O intensive, then the classification of the application is "Bibliographical Search System". And if the user determines that the queries are neither more CPU or I/O intensive, then the classification of the application is "Business System".

After the classification of the application has been established, the user will need to determine the number of concurrent users which are expected for his/her own application. The user may also, at this point, re-run the above experiment with multiple users to verify the determined classification. This is easily done by using the terminal emulator and submitting duplicate copies of the query(s) in combination with the CPU (or I/O) programs. With the application classified and the number of users determined, the user may select the appropriate benchmark model for his/her application. After a brief discussion of the recommended database, the statistical, bibliographical and business systems benchmark models will be discussed.

5.2.6. Recommended Benchmark

To fairly compare any software and hardware DBMS under consideration, the user must have first classified their application, determined the number of expected concurrent users and have a database in which to process their queries. Since all applications must fall within the three categories discussed in 5.2.5., the database used must have sufficiently large relations to adequately test the application queries. This is why the Wisconsin Database [15] is chosen for these guidelines. This database provides for easy indexing, controlled selectivity levels and integer and character string manipulation. This database has proven successful and is recommended for these guidelines.

5.3.0. Application Classification Models

This section contains the three possible application models for use by the benchmark user. Included with each model are representative queries which the perspective user could use directly or use as templates of similiar queries created by the user. These guidelines encourage the user to create his/her own queries as was done when classifying their application category in 5.2.5. Remember, that the user may always test whether a query or query set is CPU or I/O intensive through the afore mentioned categorization method. This permits the user to create and test queries which most closely approximate his/her own application.

These models include queries which use a primary index, no

index and various multiuser levels. Only the query execution time discussed in 5.2.2. should be recorded for comparison.

5.3.1. Bibliographical Search System Model

If the user has classified their application in this category, queries which test large relation searching should be used. Such queries typically look through large relations (many times un-indexed) to return a small amount of qualified data. Typical applications might include insurance company databases, telephone directory look-up databases, city taxation and others. Since the Wisconsin Database is being used in all three models, the actual query for a specific application must be thought of in terms of the real database being used. The models, then, will include a typical application query for the category stated in English. The equivalent SQL formatted query, to represent the English query, would be run against the Wisconsisn Database. The resultant query execution time would then be recorded.

In this example, a user is interested in finding some taxation information of a property at a paticular address.

English Form: Find the amount of property taxes paid at 115 Cherry Road.

SQL Form:
```
select hundredD
  from tenthoustup
  where stringUD = 'BxxxxxxxxGxxxxxxxxxU';
```

The attribute "hundredD" represents the tax dollar value returned. "StringUD" represents the address (which is unique). The tenthoustup relation represents a city database

of 10,000 unique addresses. This would be equivalent to a community with a population of approximately 40,000 (assumming 4 people per household). This query has no indexed attributes of which it could use to improve the query execution time. This is typical of the bibliographical systems. The data in the database is generally unique in value (ie. address, phone#) and therefore an index table would be too large to be useful.

The other issue here is number of users. If, for example, the user wishes to simulate 50 other concurrent users, one method would be to create a set of I/O intensive queries and submit 50 copies of the query set repeatedly until the query of interest has completed. In this way a controlled workload executing in the background may be employed.

The next query example uses a clustered index to select a small number of tuples (1%).

English Form: Find the telephone numbers of all people with their last name beginning with "Y".

SQL Form:
```
select uniquelD
  from tenthoustup
  where hundredD = 5;
```

The attribute "uniquelD" represents the returned telephone numbers. "HundredD" controls the selectivity level (1%) which might be a reasonable estimate of all lastnames beginning with "Y". Again, tenthoustup represents ten thousand entries in a telephone directory.

Many I/O intensive queries could be created and used to

measure the performance of a relational DBMS in a bibliographical system. If the user is not sure if a particular query qualifies as I/O intensive, he/she may always check by using the previously discussed method (5.2.5.). In this way a user of these guidelines may fairly compare the performance of multiple DBMS systems by using his own representative query sets.

5.3.2. Statistical Analysis System Model

If a user has classified their application in this category, CPU intensive queries become important. Queries in this application category are typically responsible for returning highly qualified data from multiple relations. This is commonly disjoint factual information contained in many smaller relations in the database. This type of application may be found in product service databases, a component database (ie. bill of materials) or national sales figures databases.

In the first example, a user is interested in finding the address and phone number of any product site where a particular model product (model 10) has had 3 or more service calls.

English Form: Find all site ids with addresses and phone numbers where machine model 10 has had more than 2 service calls.

```
SQL Form:  select unique thousandA, stringUA, stringUD
             from thoustup, tenthoustup
             where thousandA = thousandD and
             thousandA =
               (select thousandB
```

```
from twothoustup
where fourB > 2 and
hundredB = 10);
```

In this query, "thousandA" represents the id of 1,000 possible sites in a "product sites" (thoustup) relation where a product line is sold. "stringUA" represents the phone numbers and "stringUD" the addresses. The tenthoustup relation represents the addresses (thousandD) of all product sites and their phone numbers (stringUD). The "thousandB" attribute represents site ids where a product service call has been placed. The "hundredB" attribute represents qualified product models (model=10) with more than 2 service calls (fourB > 2) from the "product service calls" relation (twothoustup). So the site ids for product model 10, with more than 2 service calls is returned. Of these, only model 10 sites are joined with the entire product site relation to return those qualified sites (thousandA) along with their addresses (stringUD) and telephone numbers (stringUA).

This query could be re-run with the benefit of indexing. For example one could create a clustered index on the product model number in the "product service calls" relation which should aid in the loaction of the qualified sites with model = 10. So, by creating the index:

```
create index Xhundred on twothoustup (hundredB);
```

and re-running the query, one can measure if any benefit results from the use of the index.

5.3.3. Business System Model

The user of this category system would typically issue short simple queries to highly structured data. Systems such as these may be commonly found in business areas such as employee records, distribution or support services. As mentioned, these queries are probably not too complex and generally return small amounts of data.

As an example, consider a query which would return salary information for a divison of employees.

English Form: Find the total of all salaries paid in department 55.

```
SQL Form:  select sum (thousandA)
             from thoustup
             where hundredA = 55;
```

In this query, "thousandA" represents the dollar value of salaries. The thoustup relation represents the "workforce" relation of 1,000 employees. The "hundredA" attribute qualifies the employees as being in department 55 (10%). Again, this query could be re-run with the benefit of an index. One could build an index on the department (hundredA) attribute and compare the performance benefits.

```
Create index Xhundred on thoustup (hundredA);
```

5.3.4. Guideline Model Summary

In the previous section, the idea of application dependent benchmarking is presented with a slightly different slant. It is asserted that the specific queries to test a DBMS are not

as important as the type of query selected. The first objective in benchmarking a relational DBMS is to classify the application on which the system will support. When classifying the application, the user has defined the type of workload which must be supported and in doing so has specified the most critical parameter in the DBMS selection process. Now, the user is free to create his/her own workload (queries) which most closely approximates the real application and can issue those queries in a controlled environment. This method does not, nor is meant to, identify causes for degraded DBMS performance. And these issues are of primary importance to the DBMS developer/vendor. This method does however, provide a user with a strategy to fairly compare 2 or more relational DBMS systems within his/her own known application environment.

5.4.0. Related DBMS Selection Topics

Other issues, than performance, become important to the perspective user of a relational database management system. The system performance is paramount in the selection process but maintanability, security and associated issues are important as well. This section is included in the thesis for completeness and represents real issues confronting a user community when evaluating a relational DBMS. Many such features may be included in a particular system but only the critical features are discussed here.

5.4.1. Database Security

The fact that a particular system is the fastest is of little concern to the manager that discovers that his/her database has been deleted accidentally or otherwise. Databases must be protected. Many systems invoke security to varying degrees. Setting up and maintaining the database may become time consuming if security (ie. view, copy, update, delete, modify) privileges are assignable down the the attribute level of a relation. If problems arise, however, one may be pleased to have that security enforced. Most systems provide security options at least to the relation level and any time invested in using these options is time well spent.

5.4.2. Backup and Recovery

The integrity of the data in the database must never be in question. Users who have no confidence in the accuracy of the data will not use the data. Transaction logs are a common method to trace the "completeness" of a transaction and thus the data integrity of the database. Uncommitted transactions are recorded and in the event of a system failure, the transactions are "re-run" to place the database back to its original state prior to the failure. This method, or one like it, is required for the database administrator to accurately maintain the database and is a characteristic of the DBMS that must be provided.

5.4.3. Query Language and Relational Algebra

SQL (Structured Query Language) developed by IBM has been identified as the relational DBMS standard query language. Many vendors include their own query language into thier particular relational system. Using such a "custom" language may prove to be troublesome. Some operations which one would expect to be available (such as a projection on a relation) may not be included in the language. Other problems may become more prominent as time passes. Local expertise of the non-standard language may become scarce, especially as more vendors convert to the SQL standard. Large applications then, may become very difficult to maintain.

Language selection and use is a personal choice however sufficient care should be taken during the system selection process. Non-SQL systems are becomming more scarce and a committment to such a system may very well cause problems in the future.

5.5.0. Areas of Future Research

Some general comments can be made about software versus hardware DBMS systems. Both seem to have good environments in which each performs optimally. In those cases when a user is faced with large amounts of data (gigabyte size databases) and high throughput is required for many users, the database machine seems to be the top performer. Indications from the benchmarks, encountered in this thesis, point to marginal or poor performance from software-only DBMS systems in this

environment. If smaller sized databases are in use by a smaller group of users, then the software-only DBMS can be a solid performer. This might be one reason for the slow sales encountered by mid-sized database machine manufacturers.

Parallel processor design has had a large influence in the top-end database machine environment. On line transaction processing applications (OLTP) such as banking (automatic teller machines), airline reservation, brokerage systems etc., require very high speed processing for a huge number of users. Here, the database machine solution has performed very well. This environment seems to be the current "nitch" for database machine technology.

Another area of future work involves those issues addressed in this thesis surrounding performance measurement. Many claims have been made by both software and hardware system proponents. This has steadily confused the issues of real and reliable system performance measurement. There is a great need on the part of relational database system technologists to develop and propogate a standard method to measure relational DBMS systems. Fortunatly, these issues are legitimate for both the software DBMS and database machine solutions. With a standardized method of system performance measurement developed, a fair and accurate comparison of both software had hardware relational DBMS systems may be made.

Bibliography

[1] Codd, E.F. A relational model of data for large shared data banks. CACM, 13, 6, (June 197Ø) 377-387.

[2] Codd, E.F. Relational Database: A Practical Foundation for Productivity. CACM, 25, 2, (Feb 1982) 1Ø9-117.

[3] Malabarba, F.J. Understanding Database Machines. Government Data Systems (June/July 1985) 33-35.

[4] Humphrey, S.R. The Relational Database Machine, HARDCOPY, (Aug 1985) 21-37.

[5] Rauch-Hinden, W., First Database Machine Available for PCs', Systems & Software, (Jan 1984) 3Ø-34.

[6] The Intelligent Database Machine - Product Description, Britton-Lee Inc., Los Gatos, California.

[7] Ferrari, D., Serazz, G., Ziegner, A. Measurement and Tuning of Computer Systems, Prentice-Hall, Englewood Cliffs, N.J., 1983.

[8] Ozkarahan, E.A, Schuster, S.A. and Smith, K.C., RAP - An Associative Processor for Database Management, Proc. AFIPS NCC, vol. 45, 1976, 855-862.

[9] Schuster, S.A., Nguyen, H.B., Smith, K.C. RAP.2 - An Associative Processor for Databases. Proc. of Fifth Symposium on Computer Architecture., (April 1978) 52-59.

[1Ø] Hsiao, D.K., Advanced Database Machine Architecture, Prentice Hall, Englewood Cliffs, N.J., 1983.

[11] Cluley, J.C., Interfacing to Microprocessors, McGraw-Hill, Hong Kong, 1983.

[12] Heath, K. Notes from telephone conversation with Britton Lee Advisory Sales System Engineer, October 1Ø, 1986.

[13] Fischer, M., and Heath, K. Notes from meeting to discuss Britton Lee product line, November 18, 1986.

[14] Berlind, G. Britton Lee RDBMS software - An Overview of Its Structure, BLI Product Marketing, August 1985.

[15] Bitton D, D.J. DeWitt, C. Turbyfill, "Benchmarking Database Systems A Systematic Approach". University of Wisconsin - Madison, Comp. Sci. Dept Rept. #526, Dec. 1983.

[16] Bogdanowica, R.A., "Experiments in Benchmarking Relational Database Machines," Database Machines, H.O. Leilich and M. Missikoff, editors, Springer Verlag, 1983.

[17] Simon, E., "IDM vs Rdb Benchmarks. IDM Performance Report #1", Performed by GEI and GTE, published by Britton Lee Inc., 1985.

[18] Boral, H., D.J. Dewitt, "A Methodology for Database System Performance Evaluation." University of Wisconson - Madison, Comp. Sci. Dept Rept. #532, Jan. 1984.

[19] Bitton D, C. Turbyfill, "Design and Analysis of Multi-User Benchmarks for Database Systems," Cornell University, Dept. of Computer Science Rept. #84-589, Jan. 1984.

[20] Yao, S.B, A.R. Hevner, D.R. Benigni, "Benchmark Analysis of Database Architectures: A Case Study". National Bureau of Standards, Garthersburg, MD., Report No.: NBS /sp-500/132, Oct. 1985.

[21] Strawser, P.R., "A Methodology for Benchmarking Relational Database Machines," Naval Postgraduate School, Montarey CA., Report 4. Number: NPS 52-84-004, Jan. 1984.

[22] Martinez, R., O. Lubeck, "A PDP 11/34 Based Remote Terminal Emulation System," Proceedings of the Digital Equipment Computer Users Society, California, Dec. 1982.

[23] Epstein, R. and P. Hawthorn, "Design Decisions for the Intelligent Database Machine," Proc. of the 1980 National Computer Conference, pp. 237-241.

[24] Lipovski, G.J., "Architectural Features of CASSM: A context addressed segment sequential memory," Proc. 5th Annv. IEEE Symp. Comput. Arch., Apr. 1978.

[25] Dewitt, D.J., "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems," IEEE Trans. on Comp, June 1979, pp. 395-406.

[26] Missikoff, M., M. Terranova, "The Architecture of DBMAC, A Relational Database Computer," in Advanced Database Machine Architectures, D.K. Hsiao ed., Prentice Hall, 1983.

[27] He, Xin-Gui, M. Higashida, D.K. Hsiao, D.S. Kerr, A. Orouji, Z. Shi, P. Strawser, "The Implementation of a Multi-Backend Database System (MDBS)," in Advanced Database Machine Architectures, D.K. Hsiao ed., Prentice Hall, 1983.

[28] Eckhouse, R., J. Estabrook, "Operating System Enhancement through firmware", Proc. 10th Annual Workshop on Microprogramming, 1977, pp. 119-128.

[29] Hawthorn, P., "The Effect of Target Applications on the Design of Database Machines," Comp. Sci. and Math. Dept., Lawrence Berkeley Laboratory, Berkeley Ca., Report #7405-Eng-48, 1982.

[30] Corporate Systems Group, "The Performance Evaluation of a Database Management System, Software vs Hardware/Software", Citicorp Corp., May 16, 1985.

[31] Knuth, D.E., "An Empirical Study of FORTRAN Programs, Software Practice and Experience," Vol. 1, 105-133.