AD-A132 312 BENCHMARKING THE JOIN OPERATIONS OF RELATIONAL DATABASE MACHINES(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA M D CROCKER JUN 83 1/1

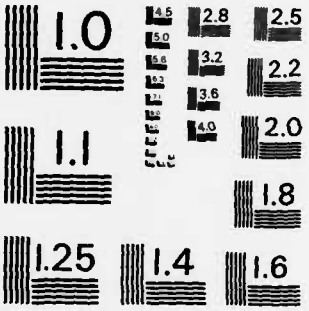
UNCLASSIFIED F/G 9/2 NL

END

DATE FILMED

9 - 83

DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

BENCHMARKING THE JOIN OPERATIONS OF RELATIONAL DATABASE MACHINES

by

Michael D. Crocker

June 1983

Thesis Advisor: David K. Hsiao

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A132312	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Benchmarking the Join Operations of Relational Database Machines		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June, 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Michael D. Crocker		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1983
		13. NUMBER OF PAGES 59
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

join, benchmarking, relational database machines, backend computers

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Over the past several years benchmarking has been developed into an effective technique for performance analyses of computer systems. Relational database machines are relatively new computer systems for which a benchmarking techniques does not yet exist. The benchmarking of relational database machines involves the identification and design of test programs through which relevant performance data can be gathered and interpreted. (Continued)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

1 SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (Continued) Block # 20

All features of relational database management must be considered when designing these test programs. The join operations are an important feature of relational database management.
The test programs for the join operations necessarily include the repetition of certain queries during which specific join parameters are varied. These parameters include: tuple size, relation size, disk placement, and the use of indices. A number of join operations have been benchmarked. These operations are equality joins, inequality joins, three-way joins, and virtual joins (i.e., views). In addition, a number of relational database machine configurations have been utilized for benchmarking the join operations.
The highlights of the thesis can be found in its contribution to a benchmarking technique for the join operations and its conclusions on the performance analyses of various relational machines in operating joins.

Approved for public release; distribution unlimited.

Benchmarking the Join Operations of Relational Database Machines

by

Michael D. Crocker
Lieutenant, United States Navy
B.S., Auburn University, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1983

Author: Michael D. Crocker, LT, USN

Approved by: David K. Hsiao
Thesis Advisor

Second Reader

David K. Hsiao
Chairman, Department of Computer Science

Kneale T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

Over the past several years benchmarking has been developed into an effective technique for performance analyses of computer systems. Relational database machines are relatively new compter systems for which a benchmarking technique does not yet exist.

The benchmarking of relational database machines involves the indentification and design of test programs through which relevant performance data can be gathered and interpreted. All features of relational database management must be considered when designing these test programs. The join operations are an important feature of relational database management.

The test programs for the join operations necessarily include the repetition of certain queries during which specific join parameters are varied. These parameters include: tuple size, relation size, disk placement, and the use of indices. A number of join operations have been benchmarked. These operations are equality joins, inequality joins, three-way joins, and virtual joins (i.e., views). In addition, a number of relational database machine configurations have been utilized for benchmarking the join operations.

The highlights of the thesis can be found in its contribution to a benchmarking technique for the join operations and its conclusions on the performance analyses of various relational machines in operating joins.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

The experiments described in this thesis are the results of the coordinated efforts of many individuals. Foremost of these individuals is certainly Ms. Paula Strawser of The Ohio State University and the Naval Postgraduate School. Ms. Strawser's diligence, dedication, and guidance have proven invaluable in both the research prior to and subsequent preparation of this thesis.

Likewise, Ms. Doris Mleczko and her staff at the Data Processing Service Center West at Point Mugu, California, have provided much assistance, and they have proven flexible enough to accomodate our sometimes inflexible requirements. Gratitude is also expressed to Mr. Ben Torres and his staff at the Computer Operations Center at Point Mugu.

Finally, to Commander Tom Pigoski of the Naval Security Group Command is offered a special thanks for his continued support in providing the necessary assistance both to keep this project going and to enable the results of this research to be presented at the International Workshop on Database Machines in Munich, September, 1983.

I. INTRODUCTION

A. WHAT IS BENCHMARKING?

The term "benchmark" has its origin in the field of geographical surveying. A benchmark is a permanent geographic feature which serves as a landmark for surveying. The term has evolved into defining a standard or criterion associated with a particular type of system or product. This standard serves as a point of reference to which functionally similar systems or products can be compared.

In the realm of computer science a benchmark consists of a standard set of instructions or programs. The execution of the set on one system provides measurements that can be used to compare with measurements obtained by running the same set on another system. This is the essence of computer system benchmarking: the process of conducting controlled experiments to collect indicators of comparative performance of different computer systems.

B. THE "GIBSON MIX"

Comparisons of computer systems were prompted by the increasing application of the systems in business and other situations in a cost-effective way. This interest in comparative performance of systems had resulted in the controlled experiments of the systems. In 1970, J.C. Gibson introduced a system of programs sets or "mixes" by which variable types of workloads could be compared. The "Gibson Mix" approach to comparing systems is based on testing several sets of applications in both business and science. The results, execution times, of these tests were published. The problem of selecting a particular computer system could be

reduced tc establishing workloads as multiples of the mix. By properly balancing execution times and mix multiples, system evaluators could produce comparative estimates for total computer systems.

C. BENCHMARK DESIGN AND OBJECTIVES

Benchmarking as a technique for comparisons of computer performance has enjoyed increasing popularity over the past decade. This approach is appealing both to producers and consumers. Basic guidelines have been developed for the proper use of benchmarks. The benchmark must be representative of real-world workloads, and the mix of instructions should be inclusive enough to provide as much relevant data as possible. Additionally, the relevance of benchmark content must be justifiable. The benchmark should be carefully designed, and objectives should be specifically stated so that the proper sequence of steps in the benchmark progression can be set down. Objectives may include evaluation towards procurement, design analysis, component certification, quality determinations, load analysis, improvement of performance, or other objectives as determined by those requiring the benchmark. The benchmark should be tailored to the objective and deal with those demands or applications which initially formed the basis of and the requirement for the benchmarking. The benchmark must be controlled from design through implementation and throughout the interpretation of the results.

II. THE BENCHMARKING ENVIRONMENT

The experiments described in this paper have been conducted on several configurations of an RDM 1100 at the Data Processing Service Center West, Naval Air Station, Point Mugu, California. The RDM 1100 and its various configurations are relational database machines, each of which is designed to be the backend of UNIVAC 1100 series computers.

A. THE HOST COMPUTER

The host computer system of which a relational database machine is used as the backend is the UNIVAC 1100/42. No modifications have been required of the UNIVAC operating system. Specially designed host-resident software has been installed in the UNIVAC.

B. THE HOST COMPUTER/DATABASE MACHINE INTERFACE

Figure 2.1 depicts the presently available methods for interfacing between the host and the backend. The first method, the relational query language, is a command interface. The second method allows the user to execute a series of queries by referring to a set of stored commands. The third method is via user programs written in high-level programming languages such as COBOL and FORTRAN in which a subroutine is provided for accessing data stored in the backend machine.

In the RDM, host interfacing is accomplished by both parallel and serial interface modules (processors) (see Figure 2.2). Each interface module can support up to 8 host systems.

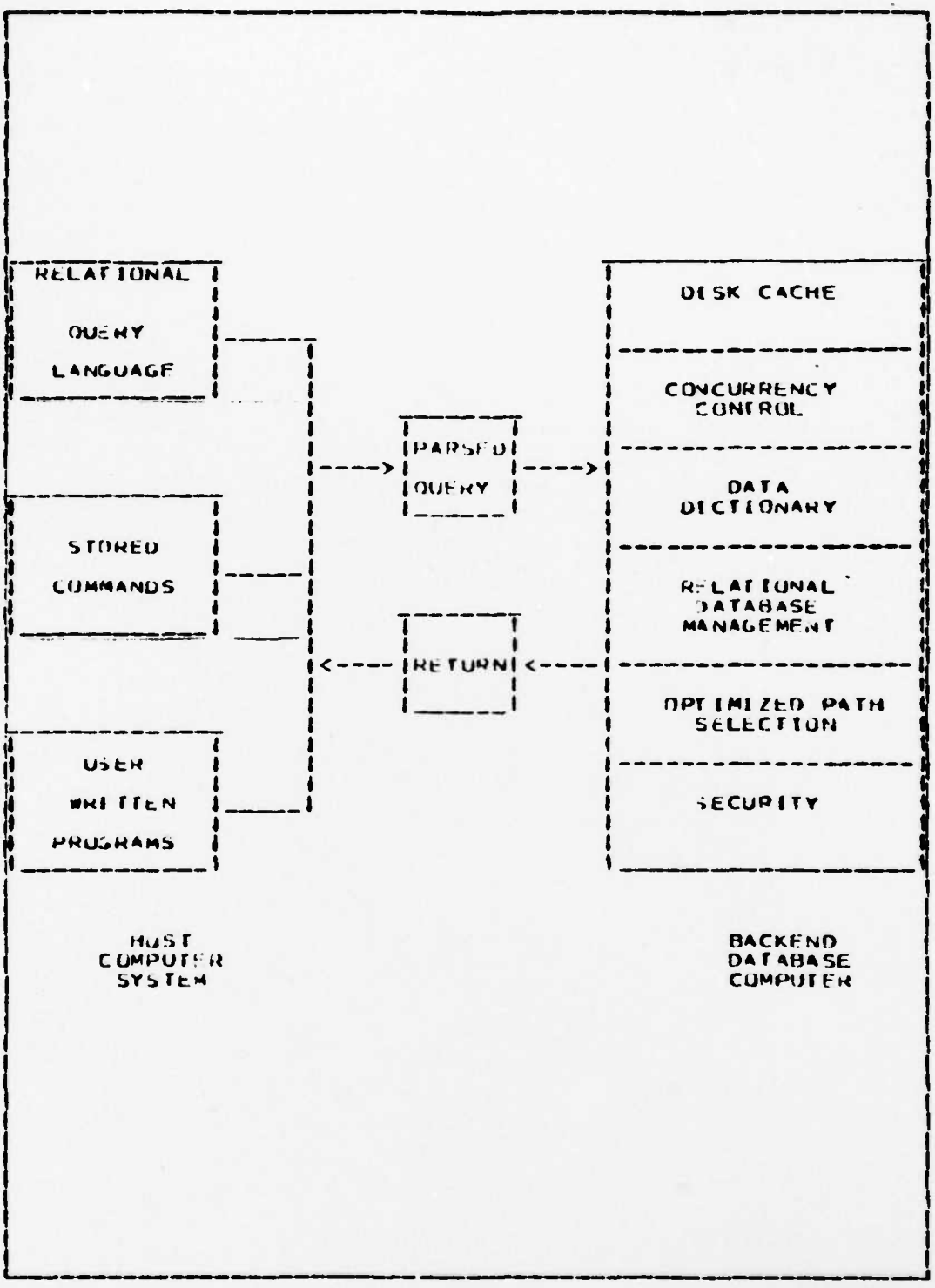


Figure 2.1 The Host/Backend Interface.

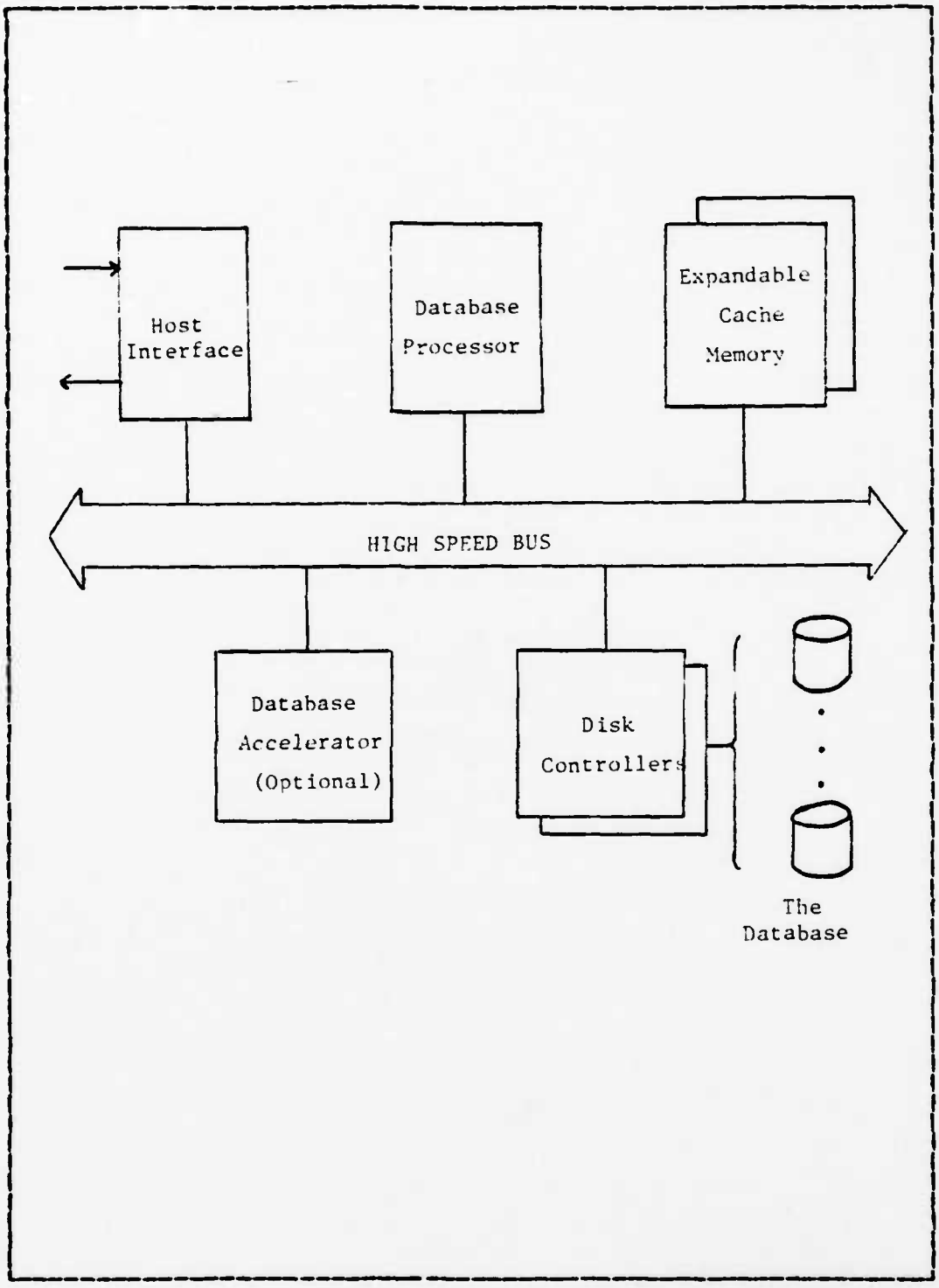


Figure 2.2 The Database Machine Architecture.

C. THE BENCHMARKED RELATIONAL DATABASE MACHINE

The basic relational database machine on which the benchmarking experiments have been conducted is a modularly designed, microprocessor-based database computer. The modules are organized around a single high-speed bus (see Figure 2.2 again).

1. Technology and Functionality of Modules

a. The Database Processor

This-Z8000 series microprocessor controls the flow of data by translating user queries into procedures. Additionally, this processor supervises system resources, coordinates hardware monitoring, and performs bus arbitration. The processor contains approximately 99% of C-codes and operates at 1/2 MIP. If the database accelerator (described below) is available, the database processor senses its availability and issues calls for its services.

b. The Accelerator

This high-speed, auxilary processor which executes instructions at 10 MIPS is built from ECL logic. It has a three-stage pipeline and is designed to optimize a well-defined collection of often used database management subroutines. The accelerator can filter data at disk transfer rates.

c. The Cache

This main memory is composed of 64K dynamic ram chips and is expandable up to 6 megabytes. System information and code occupy approximately 360K of this memory. Cache is allocated in 2K blocks, contiguously whenever possible. The paging algorithm is basically Least-Recently-Used, and the system code is never paged out.

d. Disk Drives and the Secondary Storage

The disk controller module performs burst error detection and correction and retry without intervention by the database processor. This controller can manage from one to four disk drives with each drive having a capacity of one to four disks. Presently there are two disks available with each disk capable of storing approximately 600 megabytes of data.

2. Different Accelerator and Cache Configurations Tested

The benchmarking experiments have been conducted on the following different machine configurations;

a. 1/2-megabyte cache without the database accelerator

b. 2-megabytes cache with the database accelerator

c. 2-megabytes cache without the database accelerator

D. THE DATABASES

The relational database machine handles data in 2K byte blocks. With this in mind a syntesized database has been designed. Tuple (record) lengths of 100 bytes, 200 bytes, 1000 bytes, and 2000 bytes have been chosen, thereby providing a range of 1 to 20 tuples per block. It has been sought through experimentation to contrast the same operations performed on relations with different numbers of tuples per block. It is felt that this approach may provide some measure of processor-overhead time versus I/O time.

1. Database Generation

Standard templates for each of the four different tuple lengths have been designed. Table I describes these templates. Note that within each template there are attributes (fields) that are common to all four templates: sequential integers and random integers. The attributes of sequential and random integers can be used to enforce different orderings of the same data. Each template also contains attributes specified with values uniformly distributed over a number of enumerated values. By ensuring specified distribution, the reliability of equality joins can be assured.

The actual relations for the experimental databases have been generated on an IBM 3033 system in batch mode and have been transferred to tape for transport to the UNIVAC system. For each of the four tuple lengths, relations have been generated with 500, 1000, 2500, 5000, and 10000 tuples.

2. Database Creation, Loading, and Disk Placement

In the environment of the database machine, the number of 2K-byte blocks assigned to a database is specified with the CREATE DATABASE command in the query language (Section II.E further describes the query language). Since database allocations are made in the whole number of cylinders, the number of blocks specified will be rounded up to the first whole number of cylinders. Once the allocation is made, the number of blocks actually allocated is returned to the user. The syntax for database creation in the query language is:

CREATE DATABASE (name) WITH (options)

TABLE I

Standard Tuple Templates

100-BYTE RELATION		200-BYTE RELATION		1000-BYTE RELATION		2000-BYTE RELATION	
FIELD	TYPE	FIELD	TYPE	FIELD	TYPE	FIELD	TYPE
KEY	I4	KEY	I4	KEY	I4	KEY	I4
MIRROR	C11	MIRROR	C11	MIRROR	C11	MIRROR	C11
RAND	I4	RAND	I4	RAND	I4	RAND	I4
UNIORAND	I4	UNIORAND	I4	CHARS	C63	CHARS	C79
CHARS	C4	CHARS	C14	P5	C9	P5	C9
LETTER	C1	LETTER	C1	P10	C9	P10	C9
P5	C9	P5	C9	P20	C9	P20	C9
P10	C9	P10	C9	P25	C9	P25	C9
P20	C9	P20	C9	P30	C9	P30	C9
P25	C9	P25	C9	P35	C9	P40	C9
P35	C9	P30	C9	P40	C9	P50	C9
P50	C9	P35	C9	P45	C9	P60	C9
P75	C9	P40	C9	P50	C9	P70	C9
P80	C9	P45	C9	P60	C9	P75	C9
		P50	C9	P65	C9	P80	C9
		P55	C9	P70	C9	P90	C9
		P60	C9	P75	C9	P100	C9
		P65	C9	P80	C9	UP10	UC255
		P70	C9	P85	C9	UP20	UC255
		P75	C9	P90	C9	UP25	UC255
		P80	C9	P100	C9	UP50	UC255
		P85	C9	UP10	UC255	UP75	UC255
		P90	C9	UP25	UC255	UP80	UC255
		P100	C9	UP50	UC255	UP100	UC255

FIELD TYPES

C- COMPRESSED CHARACTER STRING
(MAXIMUM OF 255 CHARACTERS)

UC - UNCOMPRESSED CHARACTER STRING
(MAXIMUM OF 255 CHARACTERS)

I4 - FOUR-BYTE INTEGER
THIS FIELD MAY CONTAIN ANY INTEGER VALUE
BETWEEN
-2,147,483,648 AND +2,147,483,647

options:
demand - the number of blocks to allocate
disk - the disk on which allocation is desired

For example,

CREATE DATABASE NPSTEST with demand = 1000 on "DSK001",
demand = 2000 on "DSKSYS"

would set aside 1000 blocks on the disk named "DSK001" and 2000 blocks on the disk named "DSKSYS" for the database "NPSTEST".

Once the database has been assigned disk space, relations in that database may be created as follows:

CREATE relation-name ((field name) = (format),...,
(field name) = (format))

The above command would set up an empty relation in the database to which tuples could then be appended. A database is opened by simplying entering: "OPEN (database name)".

In order to bulkload records into relations in specified databases, utility programs have been provided. The experimental relations that have been generated on the IBM 3033 system and subsequently loaded into the UNIVAC system have been translated into the backend machine using these utility programs.

Initially, we have attempted to manipulate the placement of relations in a database. That is, once a database has been allocated with disk space by the CREATE command, we have tried to force a specific placement of a

relation on a particular disk. We have assumed that for a join, optimization can be achieved if the relations to be joined are physically located on different disks. However, our attempts at placement have proven futile. The designers of the backend machine utilized certain placement algorithms. These algorithms are proprietary and are, therefore, unavailable for our modification.

The query language for the machine allows the creation of indices for quicker data access. The creation of these indices and their use is described in the following section.

3. Indices

Simply stated, indices are designed to provide more direct access to stored data. The query language for the relational database machine allows for the creation of two different types of indices. A "clustered" index is one for which the tuple is physically in the order of the value in the specified field. A "nonclustered" index is one that is created for a field or group of fields for which the tuple is not clustered.

Note that in NPSTEST all of the relations have been created with clustered indices. Also, as they are described below, indices for certain relations in other experimental databases may be created, destroyed, and then recreated during the course of the run stream for a particular join experiment.

4. The Experimental Databases

Table II describes the experimental databases. As they are explained more fully below in individual experiment descriptions, the size of the databases, the number of relations in the databases, and the indices employed are all factors in the measurements obtained.

TABLE II

The Experimental Databases

DATABASE NAME	NUMBER OF RELATIONS	NUMBER OF BLOCKS	REMARKS
NPS1	2	75	8 JOINS RUN UNDER CLUSTERED,
NPS2	2	375	NONCLUSTERED, AND NO INDEX
NPS3	2	750	SITUATIONS. 6 JOINS RUN WITH QUALIFIED SELECTIONS. 3 JOINS RUN WITH QUALIFICATION PREDICATES REVERSED-DATA SORTED ON UNIQRAND FIELD-NONCLUSTERED INICES. THESE 3 DATABASES ARE SPREAD ACROSS THE TWO DISKS.
NPS4	2	150	8 JOINS RUN UNDER CLUSTERED,
NPS5	2	750	NONCLUSTERED, AND NO INDEX
NPS6	2	1500	SITUATIONS. THESE 3 DATABASES ARE SPREAD ACROSS THE TWO DISKS.
NPS11	2	75	8 JOINS RUN UNDER CLUSTERED,
NPS12	2	375	NONCLUSTERED, AND NO INDEX
NPS13	2	750	SITUATIONS. EACH OF THESE 3 DATABASES IS LOCATED ON A SINGLE DISK.
NPS3WAY	3	200	1 THREE-WAY JOIN. THE DATABASE IS LOCATED ON A SINGLE DISK.
NPSTEST	24	31350	24 JOINS FOR EACH OF THE THREE DIFFERENT MACHINE CONFIG-URATIONS. THE DATABASE IS SPREAD ACROSS THE TWO DISKS. EACH RELATION HAS A CLUSTERED INDEX.

E. THE QUERY LANGUAGE FOR THE DATABASE MACHINE

Incorporated as an integral part of the relational database system is the query language (RQL in the case of the RDM 1100). This query language is designed to be both a definition language and manipulation language for the data stored in the machine.

1. Semantics and Syntax

The use of the CREATE command for both databases and relations has previously been discussed. The following discussion seeks to describe those features of the query language that are essential to an understanding of the nature of experiments that have been conducted on the join operation.

a. BEGIN (transaction name)

This command is used whenever multiple RQL commands are to be treated as a single command.

b. END (transaction name)

This command is used at the end of the group of RQL commands under BEGIN.

c. CREATE VIEW (view name)

This command is used to set up a virtual relation within a database.

d. DEFINE (stored command name)

This command is used to define a stored command for a particular database. The command so defined can be referenced simply by its name.

e. DESTROY (object name)

This command is used to eliminate databases, relations, indices, views, stored commands, or other constructs from the system.

f. RANGE of (range variable) is (relation name)

Range variables are used to allow the user to establish a synonym for a relation name. Once this synonym is established it can be used in lieu of the relation name.

g. RETRIEVE (target list) WHERE (qualification)

This is the most essential command for performing join operations. Relations or portions of relations are pulled from storage and displayed for the user. The data retrieved depends on the user supplied qualification which may include singular or multiple equalities and inequalities. Up to 22 fields can be specified in the target list.

h. RETRIEVE (variable name = GETTIME ())

GETTIME is a function in RQL that allows the user to retrieve a time statement from the RDM clock. The time integer retrieved is in 1/60 seconds. As will be described below, we used these times for our computations.

2. The Experimental Queries

Queries have been designed utilizing those features of RQL described above. The query streams have been designed as sets of transactions, and the joins have been designed as stored commands so that the commands could be pre-parsed in order that parsing time would be eliminated from the join time measurements. The number of fields targeted for a join is described below in individual experiment descriptions.

III. THE BENCHMARKING

A. GOALS

The experiments described in this paper are directed towards the development of a procedure by which database machines may be benchmarked. The efforts described here represent only a portion of the research. Interested readers are directed to [Ref. 1], [Ref. 2], and [Ref. 3] for additional research on selection and projection, database administration, and database generation.

The goal of these experiments is not to make a definitive pronouncement on the performance of the various configurations of the RDM 1100. Rather, the goal is to learn how to design benchmarks and interpret the results of the benchmarking experiments. Towards this end, the methodology must be machine independent, and the workload model must be based on a mix of database management statements.

B. THE METHODOLOGY

The workload has been modeled as a collection of queries in the relational query language (RQL). The primary benchmark kernel for the join operations is the RETRIEVE statement with associated qualifications. In designing this workload, classes of queries have been identified. These include data-intensive and overhead-intensive classes. The workload has been constructed as a combination of queries from each class. The query language has functioned as the primary tool for performance measurement since neither sorftware nor hardware probes have been available for use in conducting these experiments. Using the functions provided in the query language, elapsed times are measureable from

the database machine clock in seconds. Since the goal of these experiments is to learn the effects of varying parameters on machine performance and not absolute machine performance, this "rough" measurement technique is acceptable.

The operating system of the host machine allows the use of pre-defined commands and queries known as scripts which has eliminated the fluctuation of terminal time. Additionally, the fluctuation of the parse time has been eliminated by using pre-parsed commands stored in the database. However, some fluctuation is introduced by the query-post processor which formats data for screen display, but this is not significant within the query sets.

The initial approach to defining relevant queries has been to concentrate on the repetition of certain operations. During this repetition, given factors have been varied to ascertain effects on performance. For the join operations, tuple sizes, database sizes, index structure, disk placement, and machine configuration have been varied.

By and large, the query streams have been run in a controlled environment. To offset the workload variability of the host machine, runs have been conducted during times of minimal activity on the host. Likewise, use of the database machine has been restricted to a single user.

C. THE JOIN OPERATIONS

Several groups of experiments have been conducted during which certain parameters have been varied during repetitions of the same experiment. These experiments have been designed to obtain measurements on one particular aspect of relational database management: the join operations.

1. A Formal Definition of a Join

Simply stated, a join is a composition of two or more relations. In relational algebra, a join can be expressed as follows: the θ-join of column x of table R and column y of table S is a table whose rows are in the Cartesian Product of R and S, such that, for the mathematical operator θ, the row element x of R and the row element y of S hold true for θ.

2. The Join in the Benchmarked Query Language

In the benchmarked query language, RQL, a join is accomplished using the RETRIEVE, RANGE, and qualifier WHERE commands. For example:

RELATION: PERSONNEL

LASTNAME	AGE	DEPT
BROWN	25	OPS
WHITE	25	OPS
SMITH	31	ADMIN

RELATION: DEPARTMENT

NAME	PHONE	OFFICE
OPS	261	141
ADMIN	265	142
ENG	259	144

Given the above relations, a typical join query in RQL could be:

```
RANGE of P is Personnel
RANGE of D is Department
RETRIEVE(P.lastname, D.phone,D.office)
WHERE P.dept = D.name
```

This query would return:

LASTNAME	PHONE	OFFICE
BROWN	261	141
WHITE	261	141
SMITH	265	142

D. EQUALITY JOINS

1. The Definition and Examples

An equality join is one in which θ is defined as the mathematical equaltiy (i.e., =). That is, the statement following the qualifier WHERE in RQL contains either a singular or multiple equalities. For example, using the relations described above, the following retrievals represent two different equality joins:

```
RETRIEVE(P.lastname,D.phone)
WHERE P.Dept = D.name and P.age = "25"
```

or

```
RETRIEVE(P.lastname,D.phone)
WHERE P.dept = D.name and D.name = "OPS"
      and P.age = "25"
```

2. The Databases Used

Equality joins represent the vast majority of experiments conducted during this research. Equality joins have been conducted on all of the databases listed in Table II.

3. Queries Used

Equality joins have been run with both singular and multiple qualifications (i.e., singular or multiple equalities in the WHERE clause). The majority of the joins have been conducted on singular qualifications, and the discussions below focus primarily on those experiments. The multiple-qualification joins will be discussed separately. The singularly-qualified joins are equated on the KEY field of each relation.

4. Results

As previously explained, the methodology emphasizes varying parameters throughout the repetition of the same group of experiments. The queries and their results are presented below, and they are grouped by the parameter that has been varied.

a. Variability of Relation Size and Tuple Size

Figure 3.1 depicts three joins of relations whose tuple size is of 100 bytes. The first equality join involves a relation of 500 tuples and another relation of 1000 tuples. The second equality join involves a relation of 2500 tuples and another of 5000 tuples. The third equality join involves a relation of 5000 tuples and another of 10000 tuples. It is clearly evident that the join times increase linearly as the number of tuples being joined increases linearly.

We now vary the tuple size for all three relations. Thus, we benchmark the three relations whose tuple size is of 200 bytes. This is depicted in Figure 3.2. The benchmark of the relations whose tuple size is of 1000 bytes is depicted in Figure 3.3, and the benchmark of the relations whose tuple size is of 2000 bytes is depicted in Figure 3.4. The linearity demonstrated earlier in Figure 3.1 is again evident in these joins.

Figure 3.5 is a compilation of Figures 3.1, 3.2, 3.3, and 3.4 in which the slopes (or the rates) of linearity may be compared. It is important to note that, the bigger the tuple size there is; the steeper the slope will be.

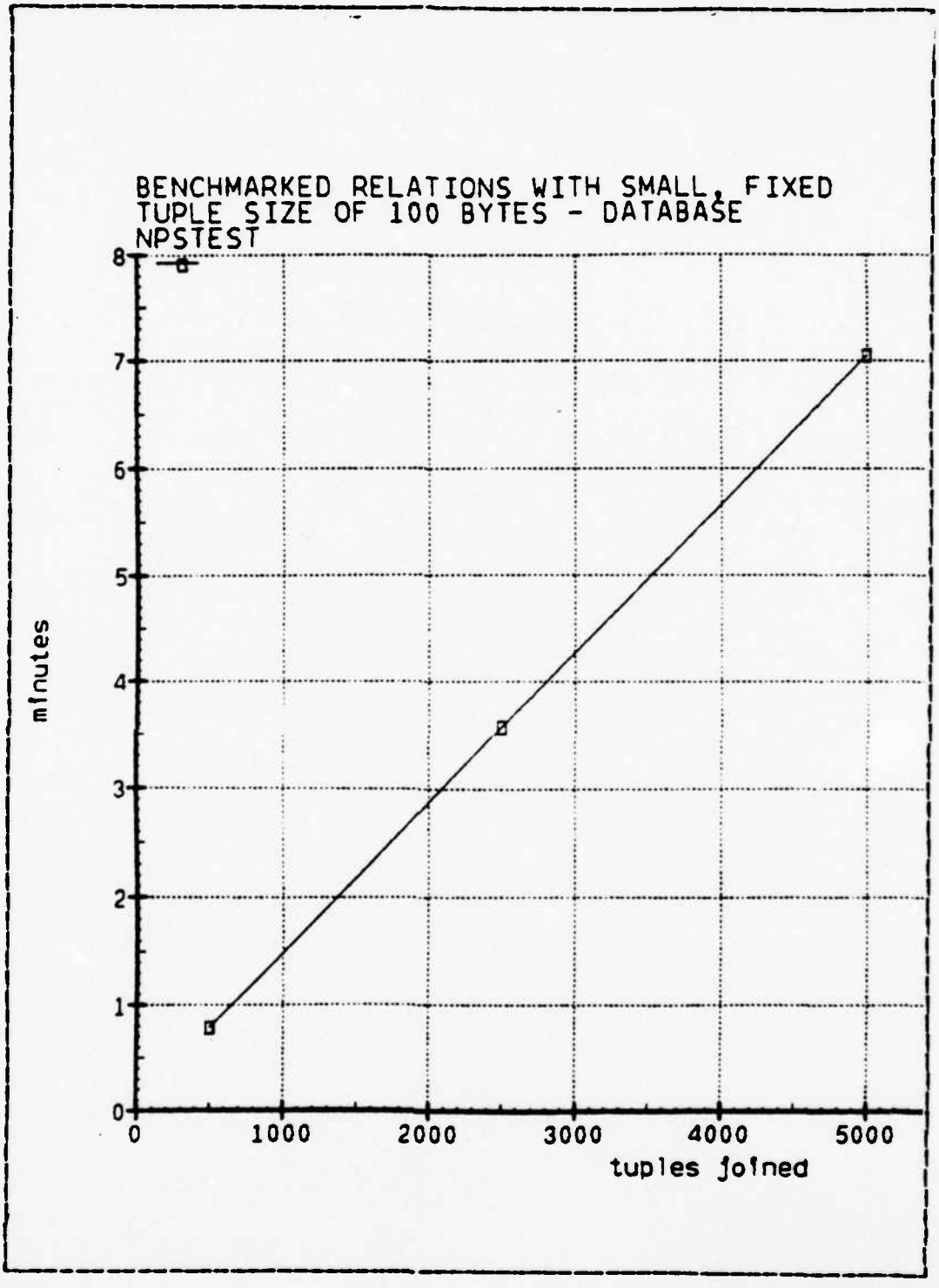


Figure 3.1 Benchmarked Relations - Small Tuple Size.

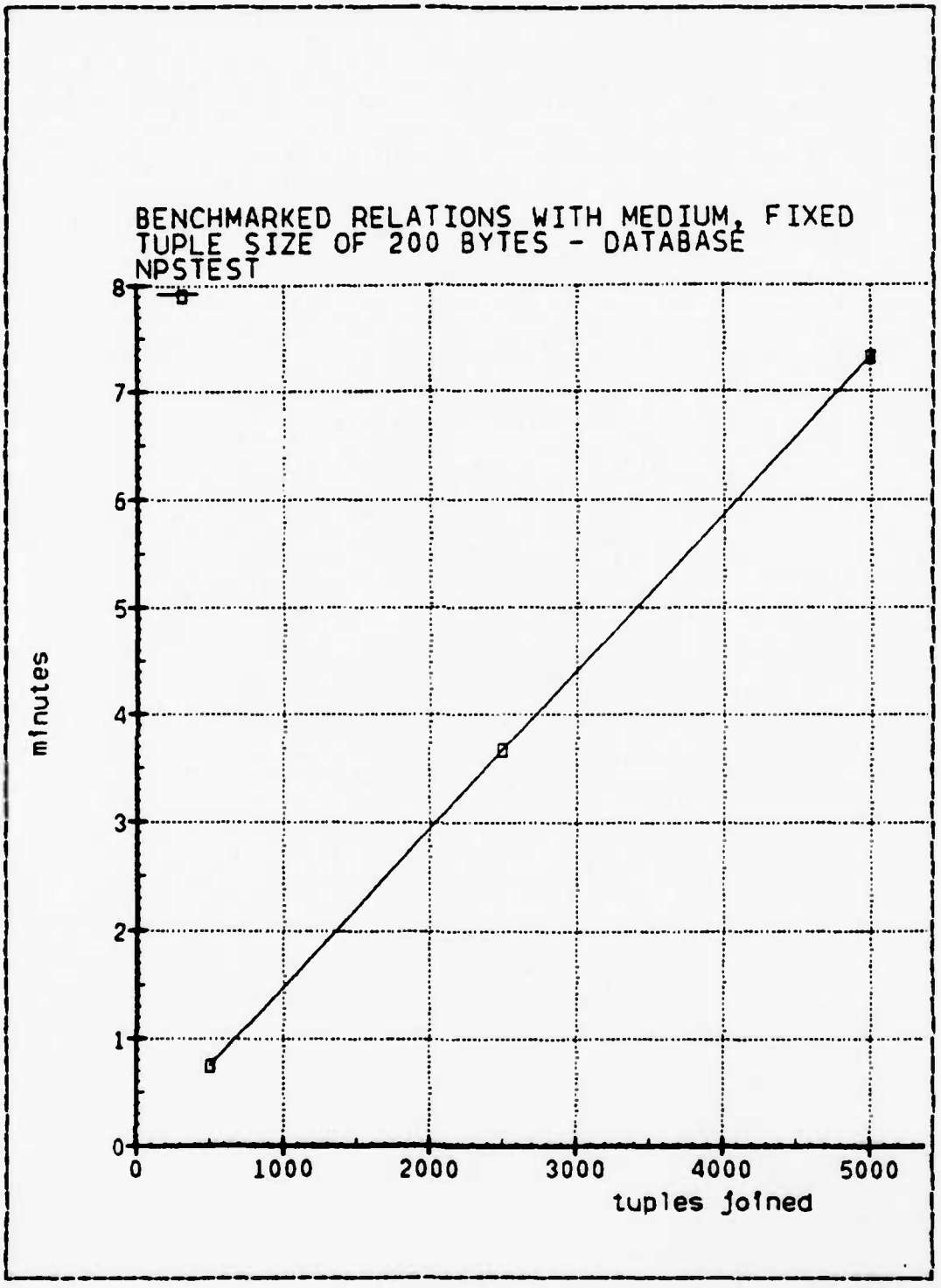


Figure 3.2 Benchmarked Relations - Medium Tuple Size.

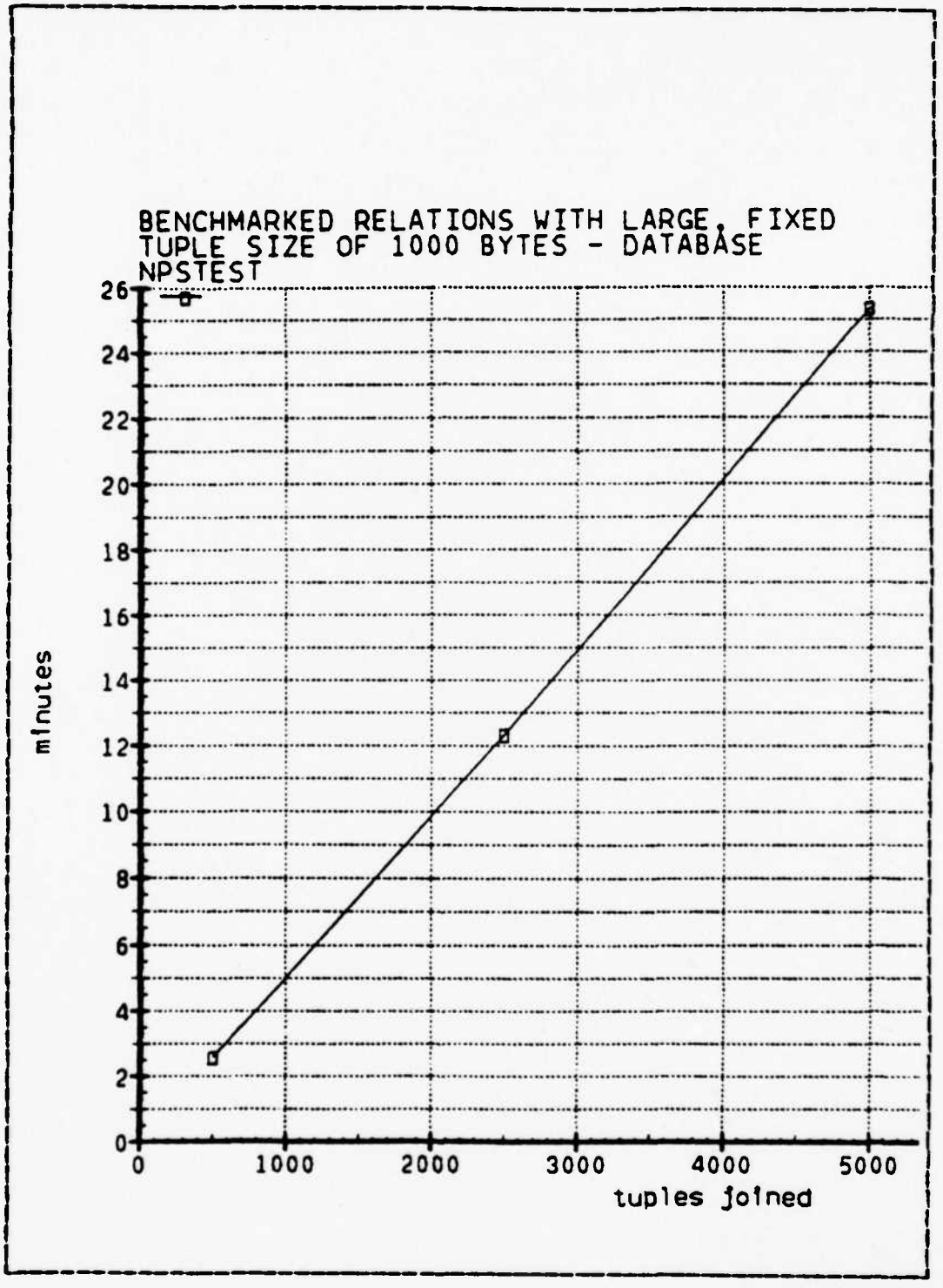


Figure 3.3 Benchmarked Relations - Large Tuple Size.

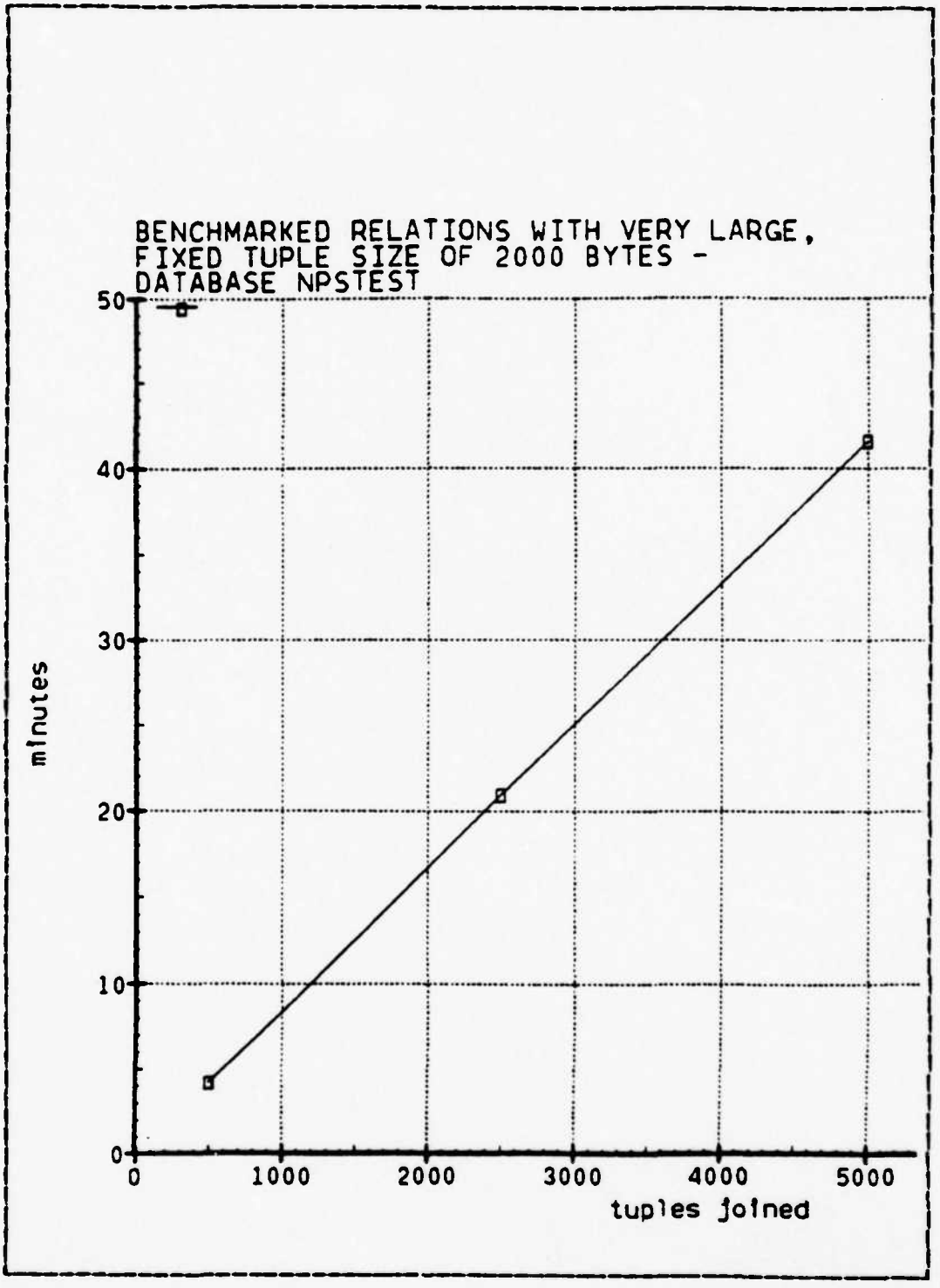


Figure 3.4 Benchmarked Relations - Very Large Tuple Size.

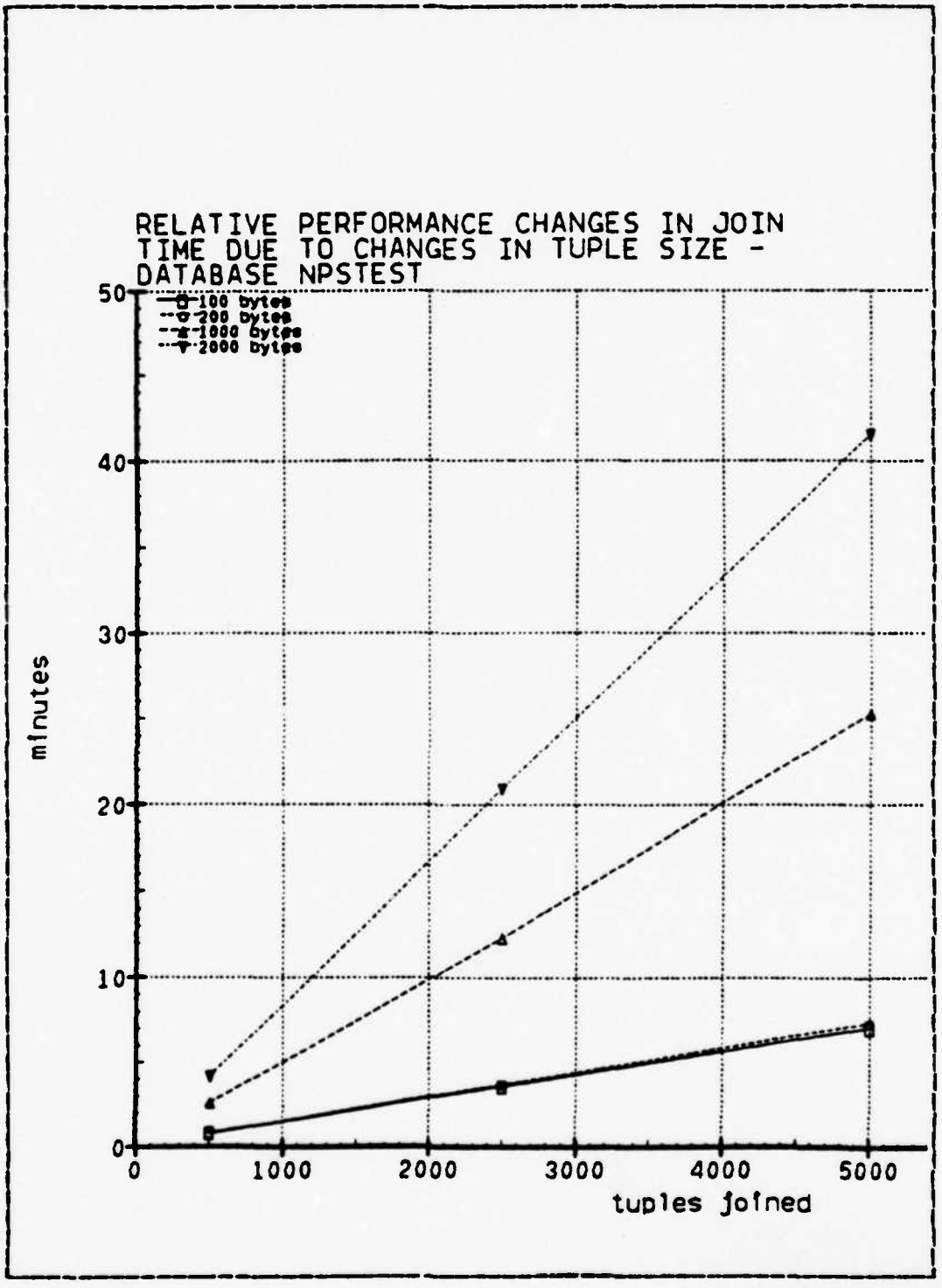


Figure 3.5 Relative Performance - Changes in Tuple Size.

b. Variability of Database Size in Terms of Number of Blocks

Figure 3.6 depicts three join operations over a large database of 31350 blocks, NPSTEST. Additionally, represented in Figure 3.6 are the same three joins over three small databases, NPS4 of 150 blocks, NPS5 of 750 blocks, and NPS6 of 1500 blocks. The results of these queries reveal that the block size is not a significant factor in join time.

c. Variability of Database Disk Placement

Every database, namely, NPS1, NPS2, NPS3, NPS11, NPS12, or NPS13 contains only two relations. NPS1, NPS2, and NPS3 have been created on the same disk. NPS11, NPS12, and NPS13 have been created separately, each of which occupies two disks. Figure 3.7 depicts the time for joins on NPS1, NPS2, and NPS3 versus the same joins conducted on NPS11, NPS12, and NPS13. The results strongly suggest that database disk placement, especially for relatively small databases, is not a major factor in join time.

d. Variability of Index Structure

A query stream has been run on NPS11, NPS12, and NPS13. During the run the index structure on the relations in the databases has been modified from clustered to nonclustered and then eliminated. Figure 3.8 depicts the join times in each situation. From the results obtained, it can be reasonably assumed that for relations of this size there is no significant difference between join times on clustered and nonclustered indices. However, the join times for those relations with no indices have exhibited increases exponentially.

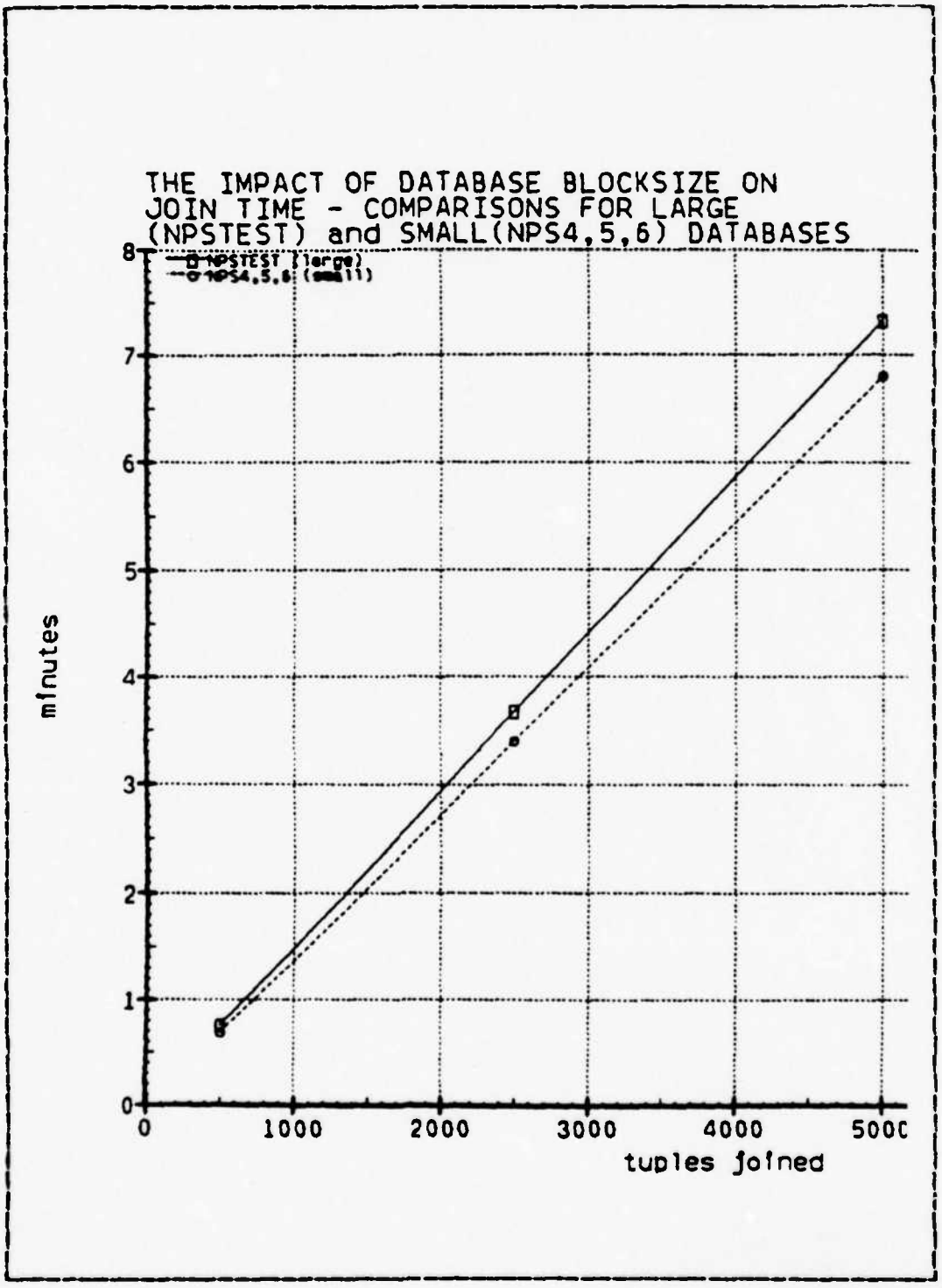


Figure 3.6 The Impact of Database Block Size on Joins.

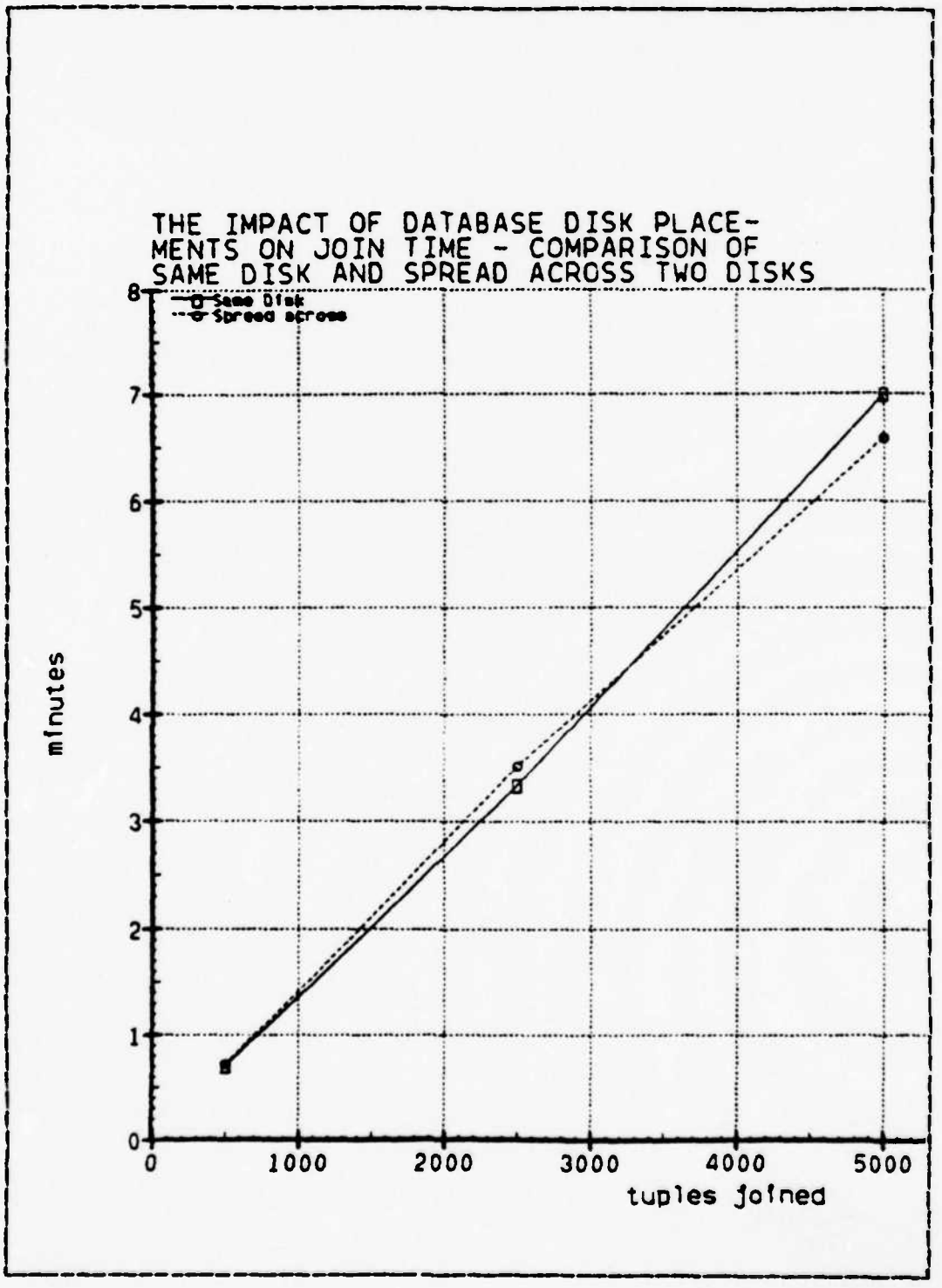


Figure 3.7 The Impact of Disk Placements on Joins.

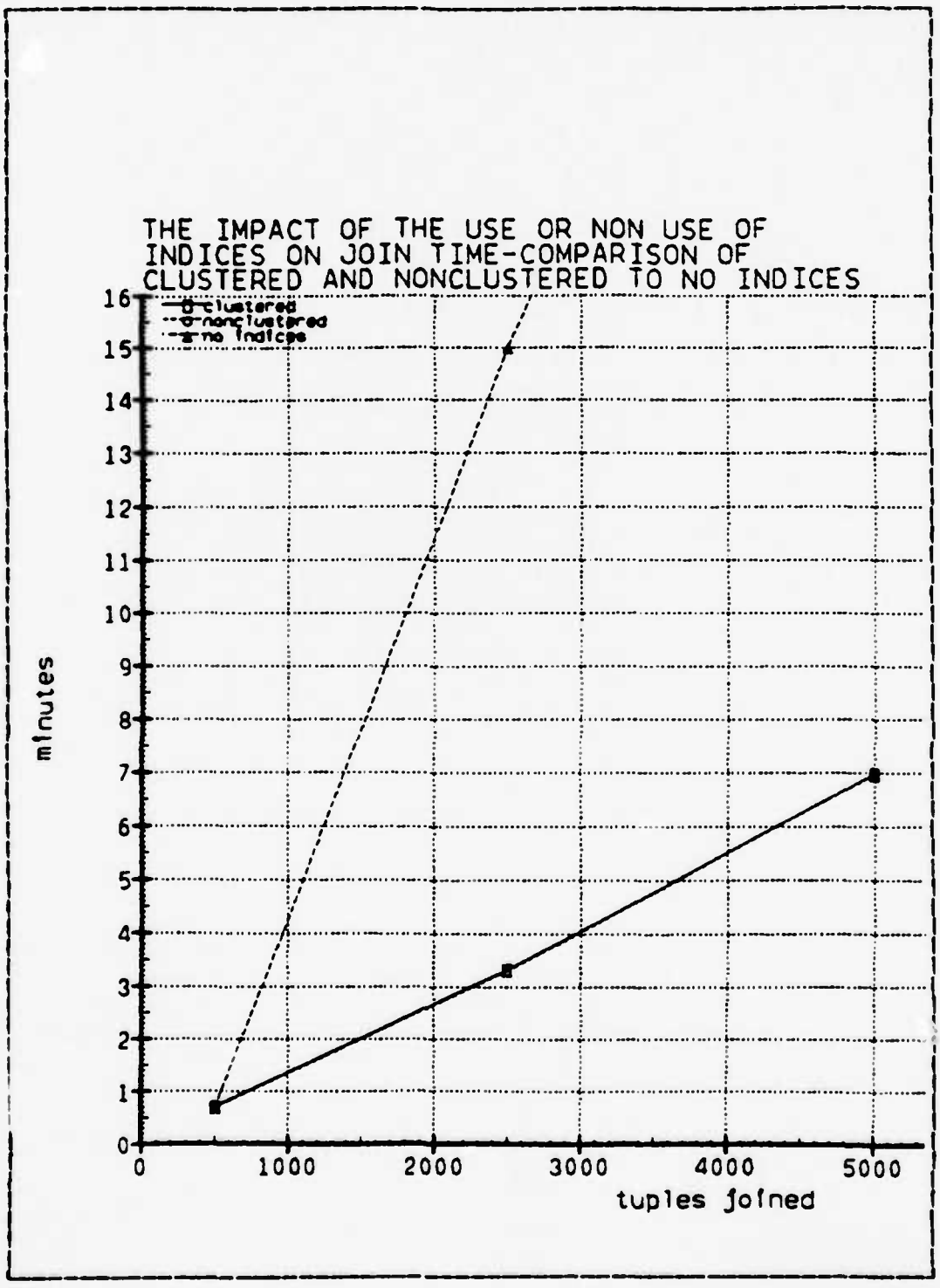


Figure 3.8 The Impact of Indices on Joins.

e. Variability of Machine Configuration

As stated earlier, during the course of these experiments, the database machine being benchmarked has operated under three different conifgurations: 1/2-megabyte cache memory without a database accelerator, 2-megabyte cache memory without a database accelerator, and 2-megabyte cache memory with a database accelerator. Joins over relations with the fixed tuple size of 100 bytes in the database, NPSTEST, have been conducted on all three configurations. The comparative results of these joins are depicted in Figure 3.9. These results show that an increase in cache memory size from 1/2 to 2 megabytes improved join time by a factor of 27% to 31%. The addition of the database accelerator to the 2-megabyte cache improved the join time by a factor of 6% to 12% only. These results would seem to clearly indicate that, for the join operation, a larger cache memory is much more effective than the addition of a database accelerator.

5. Selection Experiments

In addition to the equality joins described so far, there has been an additional qualification designed to select only a certain portion of the joined tuples for display. The number of tuples to be displayed is to be 5 % of the number of tuples in the smaller relation of the two relations in each join. To accomplish this objective for the join of the 500-tuple relation and the 1000-tuple relation, the additional qualification is to impose a "< 25" restriction on the KEY attribute. That is, the relations have been joined on the equality of the KEY field in each relation, and there has been the additional qualifier that those tuples to be displayed must have a KEY value that is less than "25". For the join of the 2500-tuple relation and

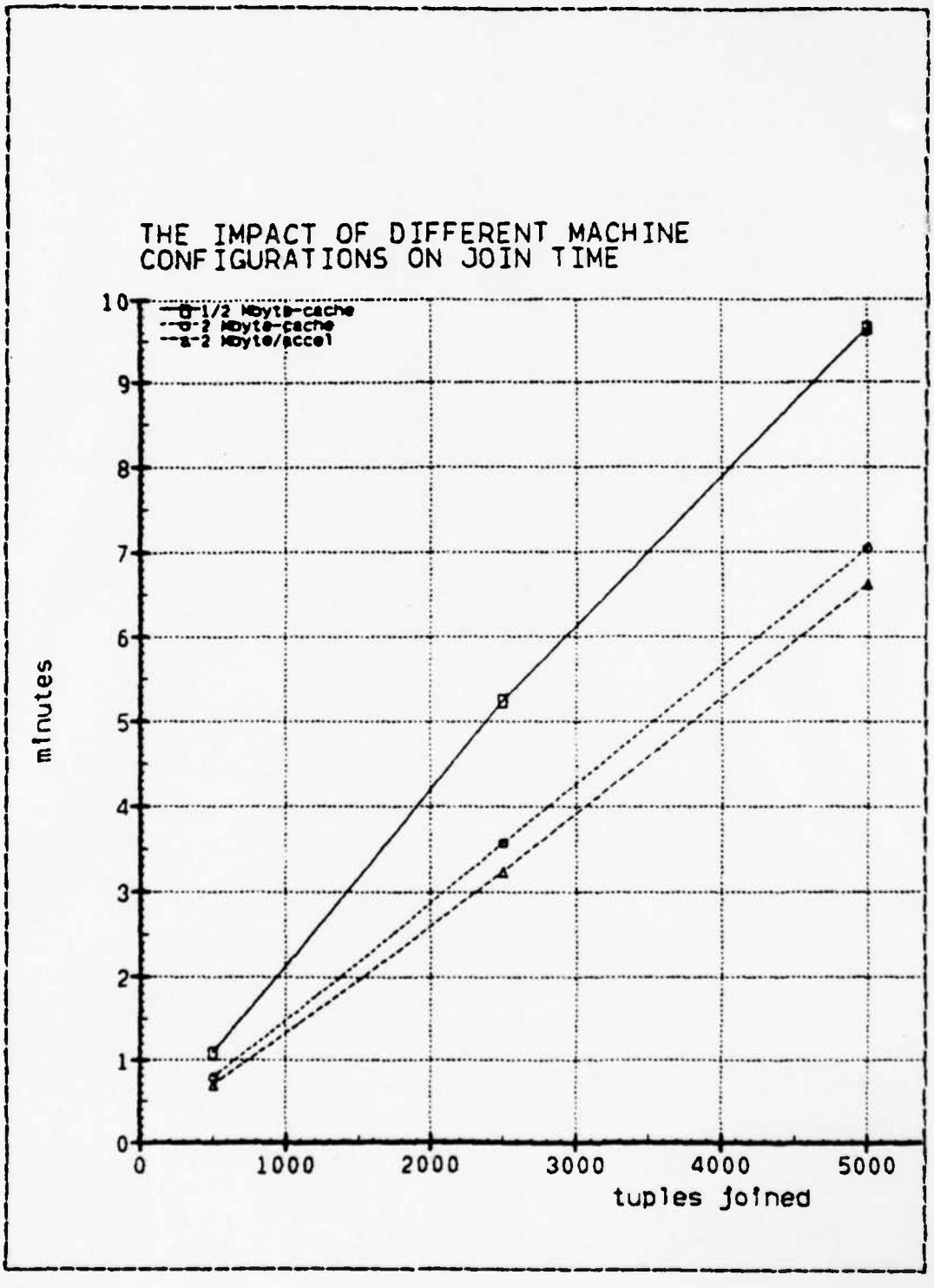


Figure 3.9 The Impact of Machine Configurations on Joins.

the 5000-tuple relation the restriction is "< 125", and for the join of the 5000-tuple relation and the 10000-tuple relation the restriction is "< 250".

Figure 3.10 depicts the response times for these join selections. Figure 3.11 depicts the response times for the same joins for which there is no 5%-selection restriction. A comparison of the results of each join reveals that especially for the join of the larger relations the difference in response time is proportionally greater. These significant differences are likely due to at least two prevalent factors. First of all, there is an I/O overhead that undoubtedly comprises a major portion of the difference. Secondly, it is highly probable that for this type of join the select operation is performed first, and then the actual join is performed. A comparison of Figures 3.10 and 3.11 would support this hypothesis.

6. Other Equality-Join Experiments

Figure 3.12 depicts a comparison between two sets of three joins on the same relations with nonclustered indices. The first set requires no relations to be sorted. The second set requires the relations to be sorted on an attribute other than the KEY attribute on which the index is based. The comparative results of the runs for these joins are close. The plotted curves for the response times cross themselves. This may indicate that the sorting of relations on the basis of a non-key attribute does not improve the join time.

Figure 3.13 depicts a comparison between two sets of the same three joins for which the expression of the equality predicate has been reversed. For these particular joins the reversal of the expression of the equality predicate appears to be insignificant as a factor in join time.

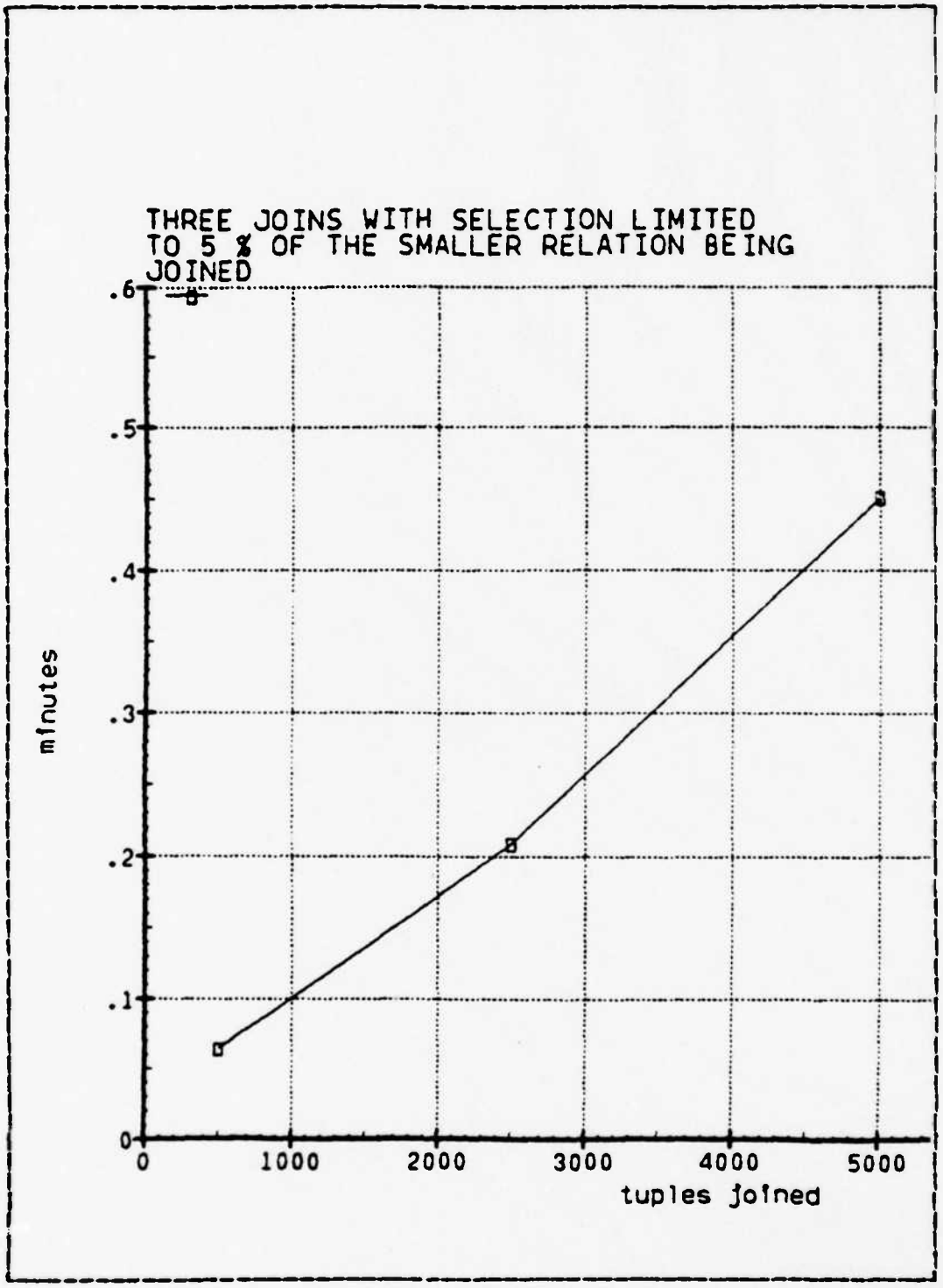


Figure 3.10 Three 5%-Join Selections.

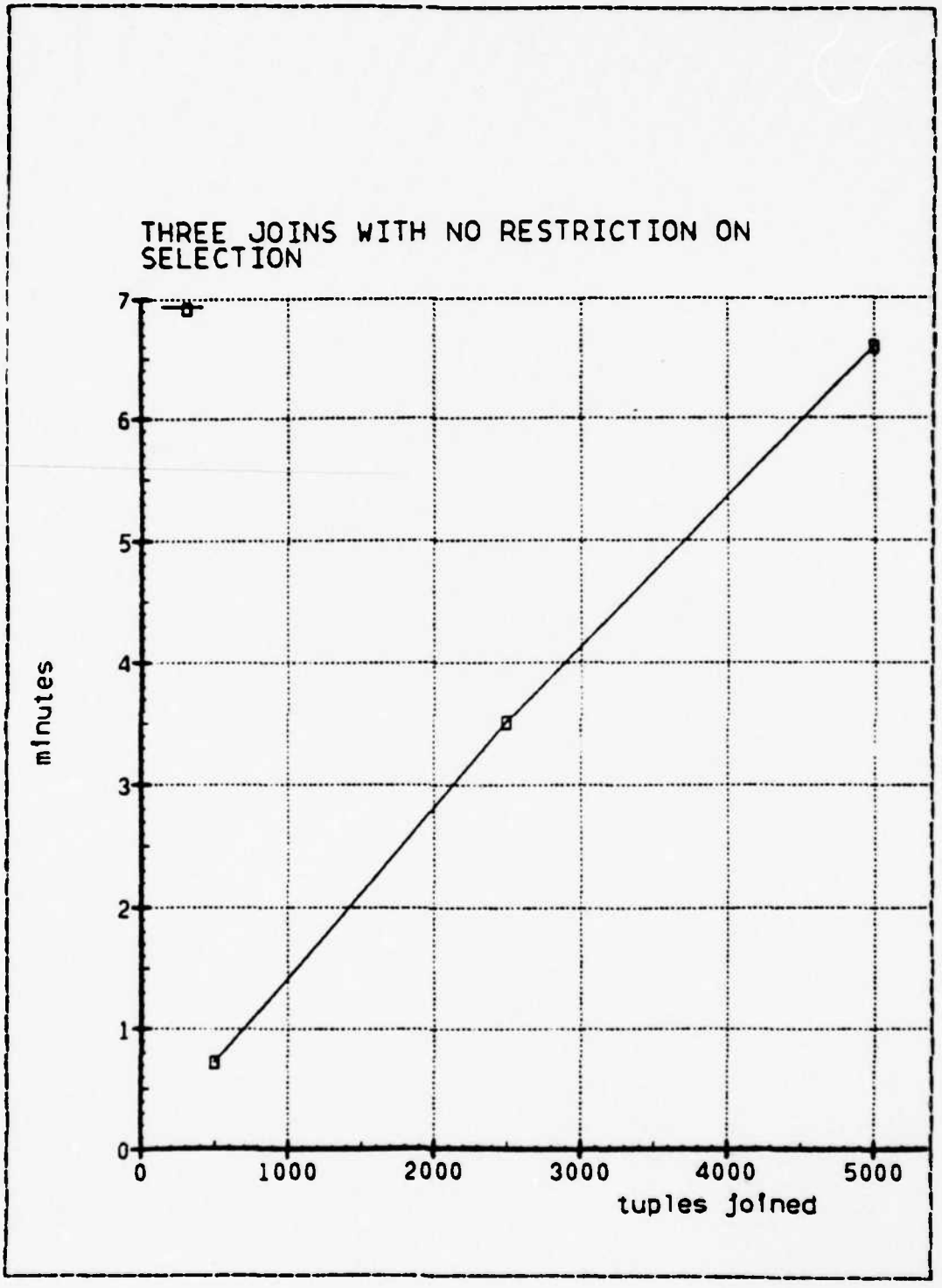


Figure 3.11 Three Joins Without Selection.

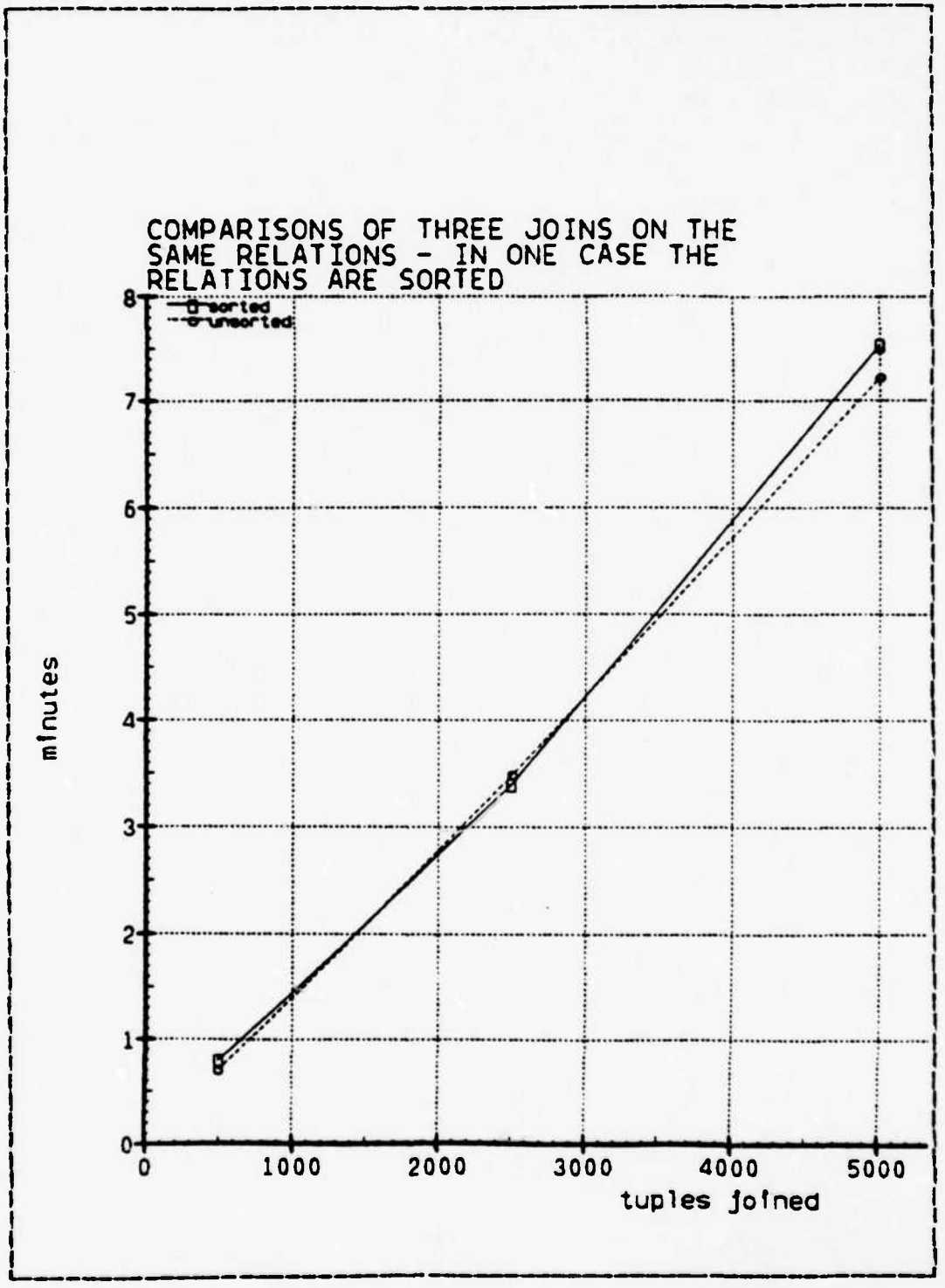


Figure 3.12 Joins on Sorted and Unsorted Relations.

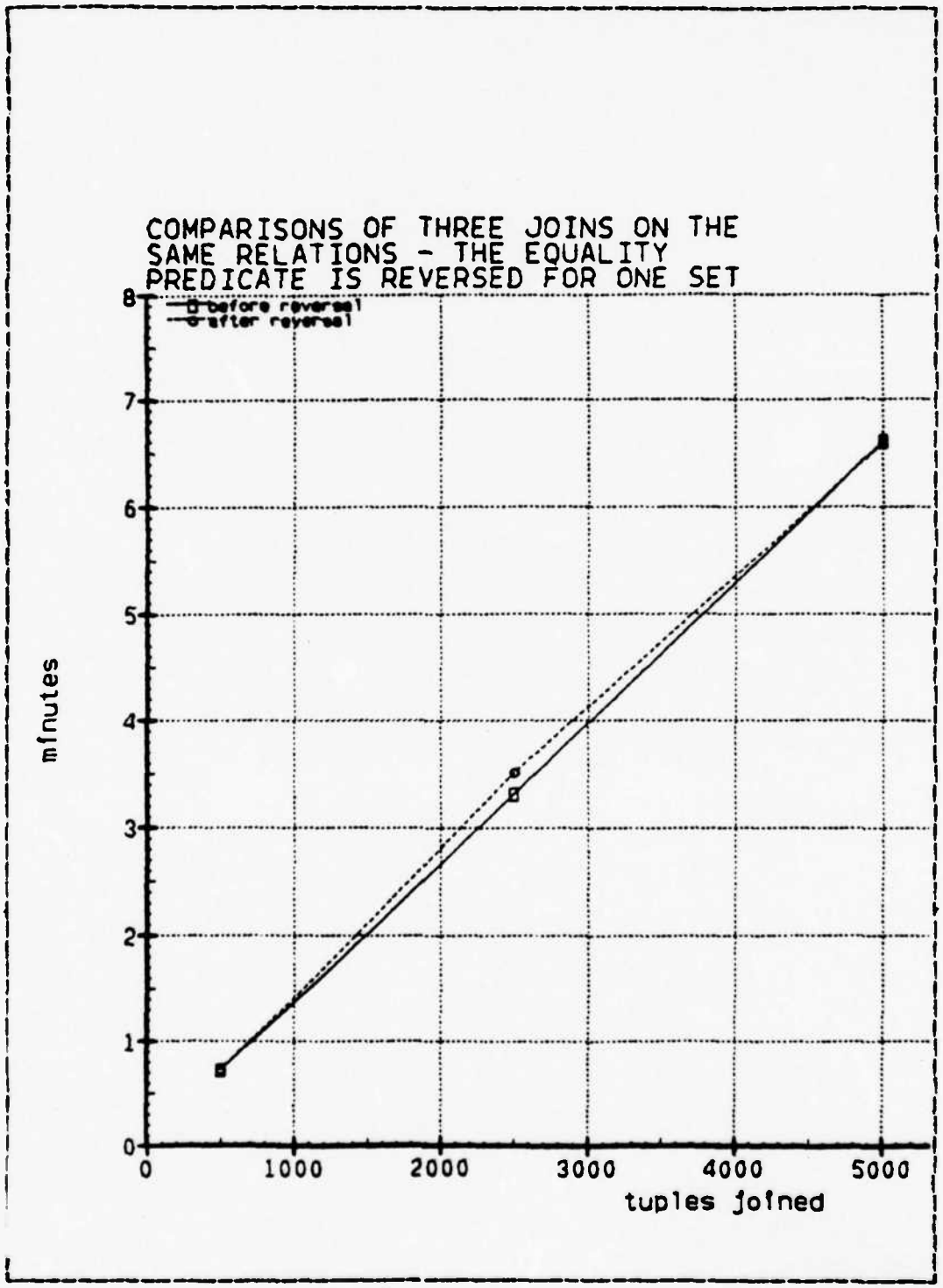


Figure 3.13 Joins With Predicates Reversed.

E. INEQUALITY JOINS

A limited number of inequality joins has been conducted during the course of these experiments.

1. A Definition

For these experiments an inequality join is one in which θ is defined as a mathematical inequality. That is, the statement following the qualification WHERE in RQL contains either "!" or "<" or ">". This qualification has been imposed on the KEY attribute.

2. Experiments

Inequalities have been applied to the join of a 500-tuple relation and a 1000-tuple relation and to the join of a 2500-tuple relation and a 5000-tuple relation.

3. Disastrous Results

The results of these joins have proven to be disastrous. For even the smaller join of the 500-tuple relation and the 1000-tuple relation, the response time has run into hours. This long response time has jeopardized the integrity of the experiments, since during the course of the run the status of the host machine has experienced significant fluctuations in load conditions. Obviously, it may prove the point that the inequality joins cannot be supported by the machine with any reasonable response time.

F. THE THREE-WAY JOIN

1. A Definition and Example

For these experiments a three-way join is simply a composition of three relations via equality joins. The three relations have been joined on the equality of the KEY

attribute of each relation. That is, for relations A, B, and C, the join has been accomplished WHERE A.KEY = B.KEY and B.KEY = C.KEY.

2. Experiments

The three relations that have been joined are a 500-tuple relation, a 1000-tuple relation, and a 2500-tuple relation. No selection restriction has been imposed on the join.

The response time for this query is .8114 minutes. A two-way join, under similar conditions, of the same 500-tuple and 1000-tuple relations has been accomplished in .7011 minutes. The small increase of the response time from the two-way join to the three-way join of .1103 minutes (15.7%) would appear to further demonstrate the significance of the one-time I/O overhead in joins. In other words, regardless of the number of ways a join is to be conducted, the one-time I/O overhead would consume a substantial portion of the join time. In this case, the overhead consumes about 65% of the three-way join time.

G. JOINS VERSUS VIEWS

1. The View in the Benchmarked Query Language

In RQL the CREATE VIEW command is used to set up a virtual relaton which is composed of attributes of one or more relations. The VIEW is not physically a relation. Rather, its definition is stored in the database. The following example creates a new virtual relation, LOCATOR:

```
RANGE of P is Personnel
RANGE of D is Department
CREATE VIEW LOCATOR(P.name,D.name,D.office,D.phone)
WHERE P.dept = d.name
```

2. Experiments on Views

The views have been defined and stored in the appropriate databases, before their use for comparison to join operations. For both the views and the joins, projection has been limited to five attributes, but no restriction has been imposed on selection.

The views have been created from a 500-tuple relations and a 1000-tuple relation; a 2500-tuple relation and a 5000-tuple relation; and a 5000-tuple relation and a 10000-tuple relation. These relations exist in databases NPS11, NPS12, and NPS13, respectively. Likewise, the joins have been accomplished on these same relations and databases.

Figure 3.14 depicts the comparative response time for each of the three situations. The remarkable similarity in response times between views and joins for these experiments would seem to point out that the views are no more expensive and inefficient to use than the joins. In certain situations, however, the views could be of greater value, since they require very little disk space as compared to the physical space needed by the tuples of the joins. Additionally, the view appears to provide the user greater flexibility for controlling access to the database.

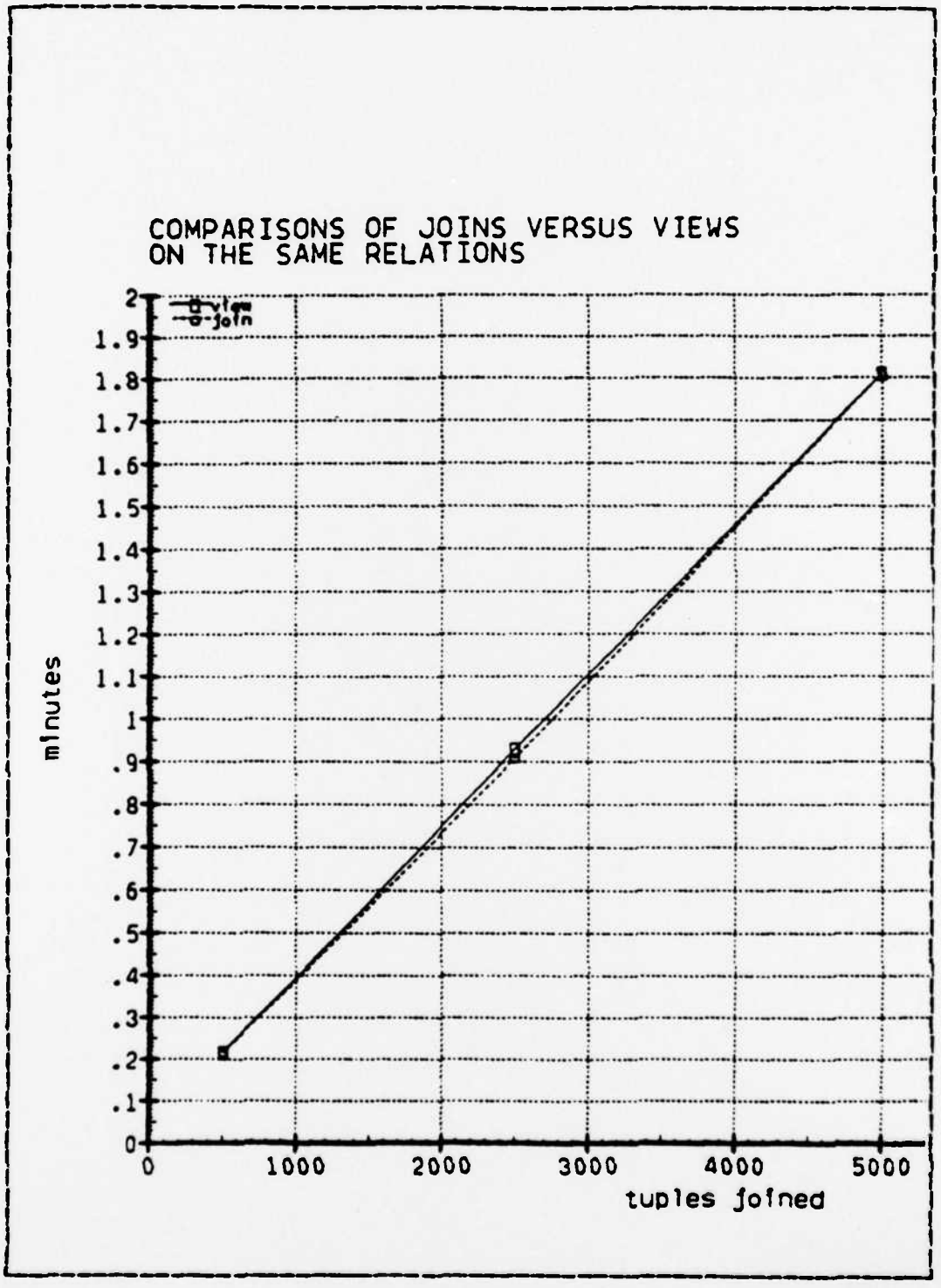


Figure 3.14 Joins Versus Views.

IV. CONCLUDING REMARKS

A. GENERAL COMMENTS

The experiments discussed above have revealed several interesting results, notably the consistent linearity in join times and the apparent significant join overhead undoubtedly resulting from bus contention. Figure 4.1 illustrates both of these charactersistics.

More specifically, Figure 4.1 depicts the total join time for various numbers of blocks of joined data. The inherent overhead is clearly evident for access to less than 1000 blocks while those joins involved with 1000 or more blocks clearly demonstrate the consistent linearity as previously discussed.

As also previously discussed, the GETTIME function in RQL has been the only measurement tool employed. Although no hardware or software probes have been available, the experiments that have been run using GETTIME have provided enough information so that some statement concerning the mean of attainable block access time can be made. Figure 4.2 depicts the average block access time for each tuple template and the effects on this average as the join has been repeated over increasingly larger relations (in the number of tuples).

In Figure 4.2 it is evident that the overhead of the initial access is being absorbed as the size of the relations being joined increases. By repeating the same join for increases in both blocks size and number of tuples accessed, some representative mean access times can be ascertained. That is, the access time curves will approach some asymptotic lower bound.

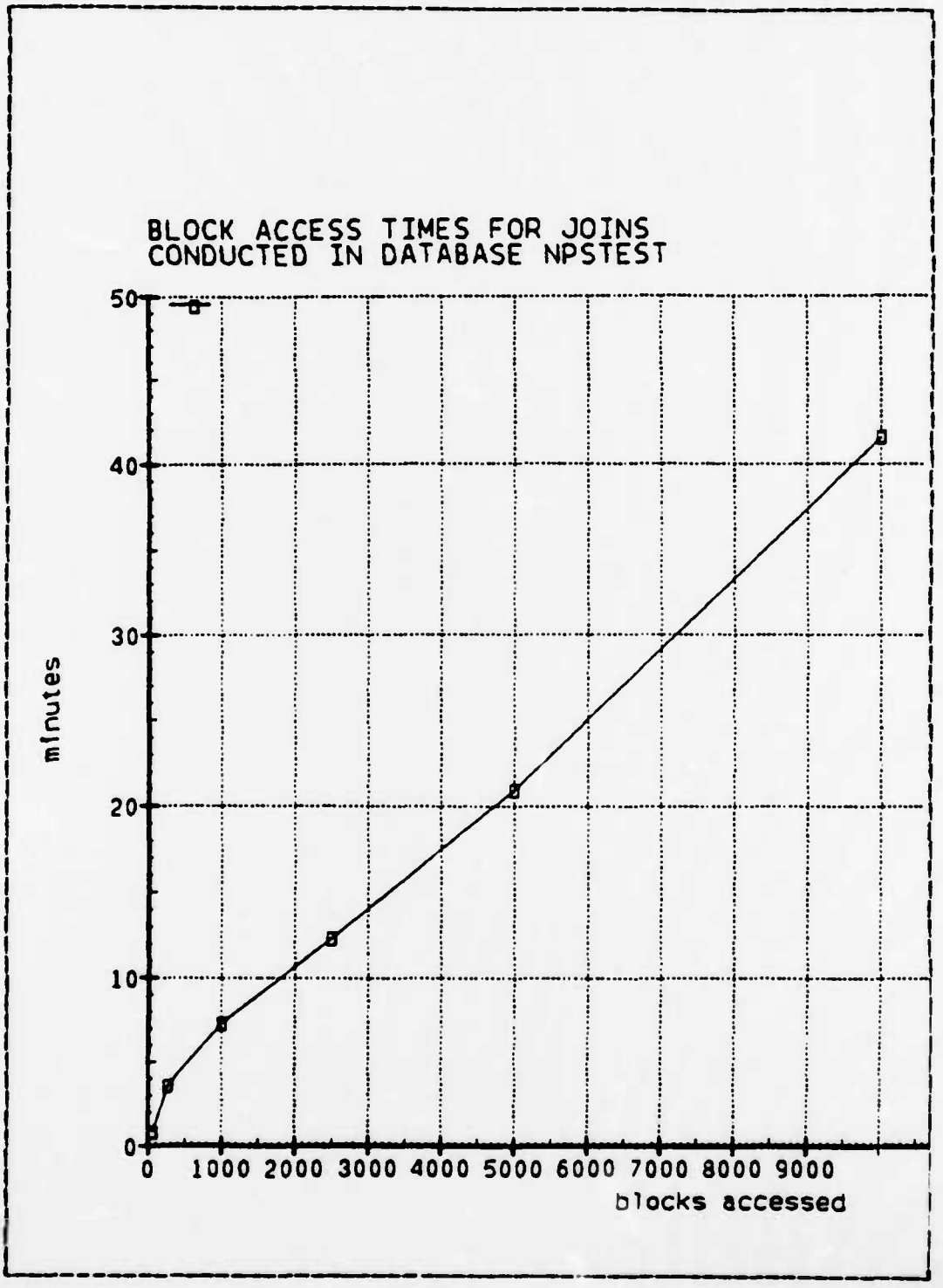


Figure 4.1 Block Access Times.

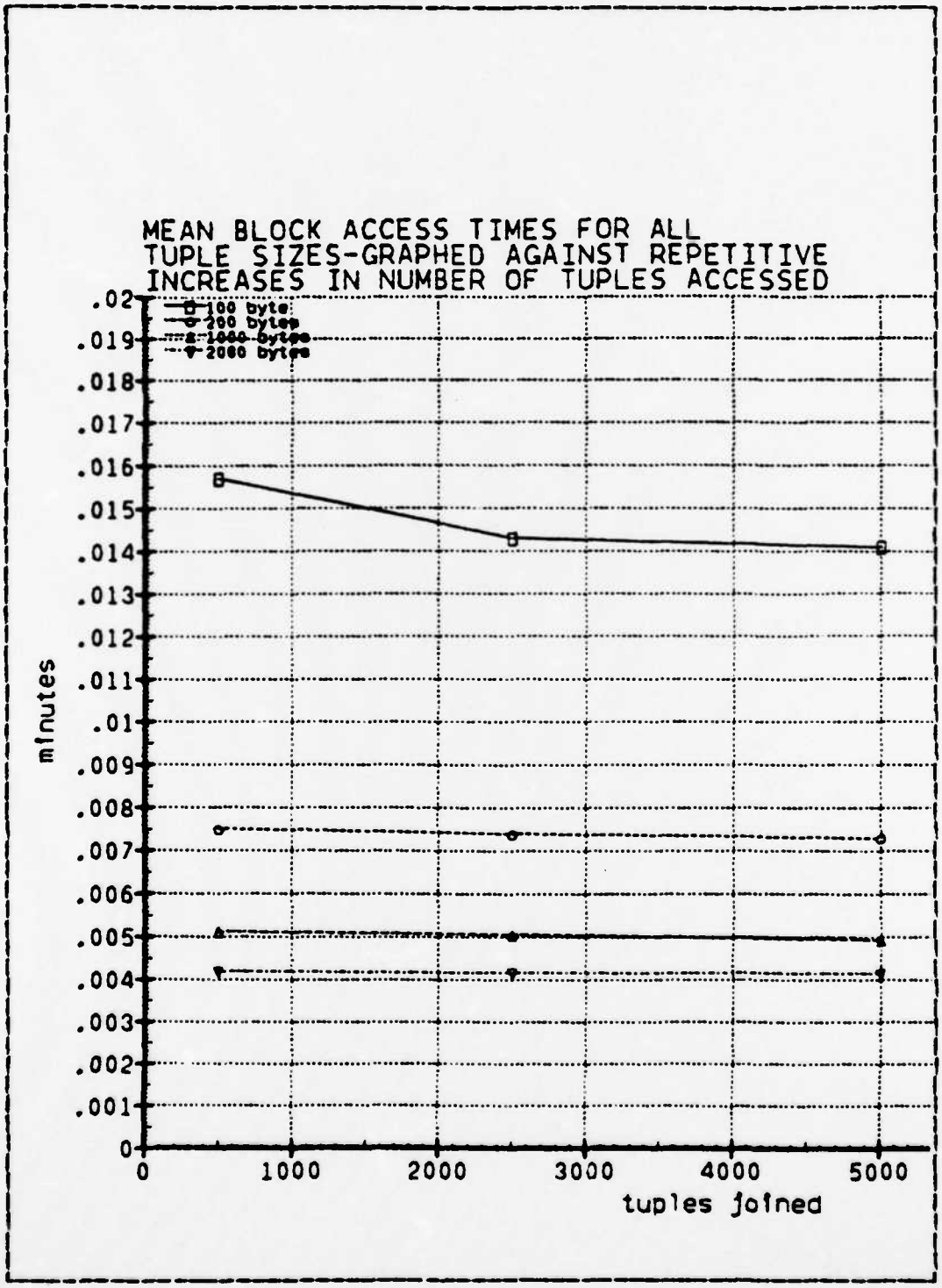


Figure 4.2 Mean Access Times.

Figure 4.2 also reveals that for this particular database machine that it is more efficient (or profitable) to perform joins on larger relations. The access times for the smaller relations are much higher than the access times for the larger relations. As the size of the relation increases, the mean access time demonstrates a convergence to a representative number. This number, the mean access time, can be considered an important charactersitic of this particular benchmarking experiment.

B. A COMPARISON OF DIFFERENT ACCELERATOR/CACHE CONFIGURATIONS

This benchmarking experiment has not been designed as an analysis of several differently configured RDM 1100s. However, while this benchmarking is making progress, the availability of more cache and the database accelerator has stimulated much interest in the performance differences for the different machine configurations. Therefore, considerable time has been expended towards accumulating comparable data for each of the three configurations on which experiments have been run.

In Chapter III there is a brief discussion of the differences in join times for the relations of 100-byte tuples. The following discussion focuses on the 24 joins conducted on the database, NPSTEST, for each of the three configurations.

Table III summarizes the average percentage decrease in join time for each join as the amount of cache is increased from 1/2 megabyte to 2 meagbytes. Table III also summarizes the further decrease in join time as the database accelerator is added to the 2-megabyte cache configuration. This summary reveals larger decreases in the join time as the sizes of the relations being joined increase. In other

TABLE III

Comparison of Jcins Conducted on Different Machine Configurations

TUPLE SIZE NUMBER OF TUPLES IN RELATION A X NUMBER OF TUPLES IN RELATION B	% DECREASE IN JOIN TIME 1/2 MEGABYTE TO 2 MEGABYTE-CACHE	% DECREASE IN JOIN TIME WITH ADDITION OF ACCELERATOR
100 BYTES		
500 X 1000	27.6	12.3
2500 X 5000	31.8	9.1
5000 X 10000	27.0	6.0
200 BYTES		
500 X 1000	49.7	0.0
2500 X 5000	49.8	3.5
5000 X 10000	47.4	4.7
1000 BYTES		
500 X 1000	51.0	5.3
2500 X 5000	52.0	1.3
5000 X 10000	51.0	5.1
2000 BYTES		
500 X 1000	59.2	0.2
2500 X 5000	58.3	3.0
5000 X 10000	57.9	3.3

words, as the initial join overhead is absorbed, the additional cache increasingly decreases the join time by a percentage of approximately 59%. Correspondingly, it appears that the effects of adding a database accelerator to the 2-megabyte cache are less significant for the larger relations, although in all cases there is some improvement.

C. THE METHODOLOGY AND ITS LIMITATIONS

The methodology that has been discussed in this paper has fundamentally sound origins, and the experimental approach of varying join parameters has and should continue to provide relevant information from which insight can be drawn. However, as discussed above, benchmarking is a relatively new area of research in computer science, and certainly the techniques that have been applied throughout the course of these experiments can be improved and refined.

A definitive performance pronouncement on the RDM 1100 has not been the ultimate goal due to the use of the GETTIME function of RQL. Despite its "coarseness" in getting performance measurements, the GETTIME function has been deemed accurate enough for the purposes of our experiments. Actually, this function has been considered sufficiently accurate in view of the lack of other more accurate measurement tools. Probes have not been available, and software packages for performance data collection have been delayed and are unavailable for these experiments. Future attempts to benchmark such a system should utilize additional methods for determining relevant performance data.

The benchmarking of the RDM 1100 is a project of seemingly low priority at the command which houses the host UNIVAC system. Existing workloads demand vast amount of the system's resources, and in reality it has been quite difficult to "control" the environment in which these experiments

have been conducted. Thus the load conditions of the host system may have compromised the integrity of some results. Additionally, the majority of the experiments have been conducted from a remote terminal which has probably further degraded the experimental results. Obviously, on site, strictly controlled experimentation is the ideal practice for benchmarking experiments.

Our inability to control the host environment raises yet another issue. The goal of the experiments has been to collect measurements on joins for which certain parameters are varied. However, a major parameter has not been varied. That parameter is the load condition of the host. As described above, attempts have been made to run experiments at times of minimal host activity. In actual practice, the database machine is likely to be benchmarked during periods of peak host activity. Future benchmarking efforts should take this into consideration, and attempts should be made to control and vary host load conditions as part of the mix of query scripts. In view of the minimal host activity, the results we have obtained may be considered as the optimal performance of the RDM 1100 for join operations.

As the deadline for submission of this thesis has drawn near, planned experiments have been cancelled from the testing agenda. A "time crunch" has resulted from a variety of sources. Primary of these sources has been the continuing requirements to correct software deficiencies that have been identified as a result of the experiments that have been conducted. Likewise, the changing of the database machine configuration has also severely cut into the time available to run the full set of planned experiments. In essence, although a great deal of relevant data has been collected, the consistency of some data may be questionable since a limited number of experiments has been conducted in each area of experimentation.

Besides these limitations and deficiencies, the experiments that have been conducted have provided enough relevant information from which valuable conclusions can be drawn. The results of the join experiments described here, when combined with those results of selection and projection experiments, comprise a substantial starting point for the comparison of similar database machine architectures. They provide a solid framework for benchmarking relational database machines.

LIST OF REFERENCES

1. Bogdanowicz, Robert A., Benchmarking the Selection and Projection Operations and Ordering Capabilities of Relational Database Machines, M.S. Thesis, Naval Postgraduate School, Monterey, California, September, 1983.

2. Ryder, Curtis J., Benchmarking Relational Database Machines' Capabilities in Supporting the Database Administrator's Function and Responsibilities, M.S. Thesis, Naval Postgraduate School, Monterey, California, September, 1983.

3. Stone, Vincent C., Design of Relational Database Benchmarks, M.S. Thesis, Naval Postgraduate School, Monterey, California, June, 1983.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Camerca Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Mcnterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Mcntery, California 93940	1
4.	Curricula Officer, Code 37 Ccmputer Technolcgy Naval Postgraduate School Mcnterey, California 93940	1
5.	Dr. D.K. Hsiao, Code 52 Ccmputer Science Department Naval Postgraduate School Mcnterey, California 93940	2
6.	Ms. Paula R. Strawser, Code 52 Computer Science Department Naval Postgraduate School Mcnterey, California 93940	1
7.	LT Michael D. Crocker, USN, Code 52 Ccmputer Science Department Naval Postgraduate School Mcnterey, California 93940	1
8.	Ccmmanding Officer Naval Air Station ATTN: MS. Doris Mleczko, DPSC West (Code 0340) Point Mugu, California 93042	1
9.	LCDR Curtis J. Ryder, USN, Code 52 Computer Science Department Naval Postgraduate School Mcnterey, California 93940	1
10.	LT Robert A. Bogdanowicz, USN, Code 52 Ccmputer Science Department Naval Postgraduate School Monterey, California 93940	1
11.	Ccmmanding Officer NAVGMSCCL Damneck ATTN: LCDR Vincent C. Stone, USN (Code 513) Virginia Beach, Virginia 23461	1
12.	Commander Naval Security Group Command ATTN: CDR T.M. Pigoski, USN (Code G30D) 3801 Nebraska Avenue, N.W. Washington, D.C. 20390	1

13. Miss Penelope F. Crocker
P.O. Box 459
Demopolis, Alabama 36732

1


DATE
ILME

Figure 4.2 Mean Access Times.

sizes of the relations being joined increase. In other