# TPC Benchmark B
# <u>Full Disclosure Report</u>

**for the**

**DECsystem 5100**

**Using**

**ULTRIX 4.2 and INFORMIX-OnLine 4.10**

| Company Name | System Name | Database Software | Operating System Software |
|---|---|---|---|
| **Digital Equipment Corporation** | **DECsystem 5100** | **INFORMIX-OnLine 4.10** | **ULTRIX 4.2** |

| Total System Cost | TPC-B Throughput | Price Performance |
|---|---|---|
| -Hardware<br>-Software<br>-5 years Maintenance | Sustained maximum throughput of system running TPC Benchmark B expressed in transactions per second. | Total system cost/ TPC-B throughput ($66,130/28.2 tpsB) |
| $66,130 | 28.2 tpsB | $2,345 per tpsB |

TM

Submitted for Review: 12/3/91

**First Printing December, 1991**

## Abstract

This report documents the compliance of testing performed on a DECsystem 5100 server running INFORMIX-OnLine 4.10, in conformance to the Transaction Processing Performance Council Benchmark B Standard Specification.

Two standard metrics, transactions per second (TPS) and price per TPS ($/TPS), are reported. Throughout this report, TPS refers to the tpsB performance metric, in accordance with the TPC Benchmark B Standard. The independent auditor’s report by KPMG Peat Marwick is included.

# Table of Contents

## Preface

This report documents the compliance of the Digital TPC Benchmark B testing on a DECsystem 5100 with the *TPC Benchmark B Standard Specification*[1]. The TPC Benchmark B Standard represents an effort by Digital Equipment Corporation, Informix Software Inc., and other members of the Transaction Processing Performance Council (TPC) to create an industry-wide benchmark for evaluating the performance and price/performance of transaction processing systems.

These tests were run using the INFORMIX-OnLine relational database running under the Digital ULTRIX operating system.

### Document Structure

The *TPC Benchmark B Full Disclosure Report* is organized as follows:

- The main body of the document lists each item in Clause 10 of the TPC Benchmark B Standard and explains how each specification is satisfied.
- Appendix A contains the source code of the application program used to implement the TPC Benchmark B transaction and related programs and scripts.
- Appendix B contains the INFORMIX-OnLine database definitions.
- Appendix C contains the source code used to populate the database.
- Appendix D contains samples of contents of the database files used in the tests.
- Appendix E contains a description of the physical disk partitions.
- Appendix F contains the operating system parameters and options.
- Appendix G contains the Independent Auditor’s Report by KPMG Peat Marwick.

### Additional Copies

To request additional copies of this report, write to the following address:

Digital Equipment Corporation
Administrator, TPC Benchmark Reports
Transaction Processing Systems Group
151 Taylor Street (TAY1)
Littleton, MA 01460-1407
U.S.A.
FAX number: (508) 952-4197

[1] *TPC Benchmark B Standard Specification,* Transaction Processing Performance Council, August 23, 1990, and addenda as of September 20, 1991.

# TPC Benchmark B Full Disclosure

The *TPC Benchmark™ B Standard Specification* requires test sponsors to publish, and make available to the public, a full disclosure report in order for the results to be considered compliant with the standard. The required contents of the full disclosure report are specified in Clause 10.

This report is intended to satisfy the TPC Benchmark B standard's requirement for full disclosure. In the *TPC Benchmark™ B Standard Specification*, the main headings in Clause 10 are keyed to the other standard clauses. The headings in this report use the same sequence, so that they correspond to the titles or subjects referred to in Clause 10.

Each section in this report begins with the text of the corresponding item from Clause 10 of the *TPC Benchmark™ B Standard Specification,* printed in italic type. The plain type text that follows explains how the tests comply with the TPC Benchmark B requirement. In sections where Clause 10 requires extensive listings, the section refers to the appropriate appendix at the end of this report.

## 1 - General Items

### 1.1 Sponsor

*A statement identifying the sponsor of the benchmark and any other companies who have participated.*

This benchmark test was sponsored by both Digital Equipment Corporation and Informix Software, Inc. The results were attested to by KPMG Peat Marwick.

### 1.2 Application Code and Definition Statements

*Program listing of application code and definition language statements for files/tables.*

- Appendix A contains the C source code of the application program used to implement the TPC Benchmark B transaction and related programs and scripts.
- Appendix B contains the INFORMIX-OnLine database definitions.
- Appendix C contains the source code used to populate the database.
- Appendix D contains samples of contents of the database files used in the test.
- Appendix E contains a description of the physical disk partitions.
- Appendix F contains the operating system parameters and options.
- Appendix G contains the Independent Auditor's Report by KPMG Peat Marwick.

## 1.3 Parameter Settings

*Settings for all customer-tunable parameters and options that have been changed from the defaults found in actual products; including but not limited to:*

- *Database options*
- *Recovery / commit options*
- *Consistency / locking options*
- *System parameters, application parameters, and configuration parameters*

*Test sponsors may optionally provide a full list of all parameters and options.*

A listing of all parameters and options is provided.

Appendixes A, B, E, and F contain the application, database configuration, partition, and operating system parameters used in the TPC Benchmark B tests.

## 1.4 Configuration Diagrams

*Configuration diagrams of both benchmark configuration and the priced system, and a description of the differences.*

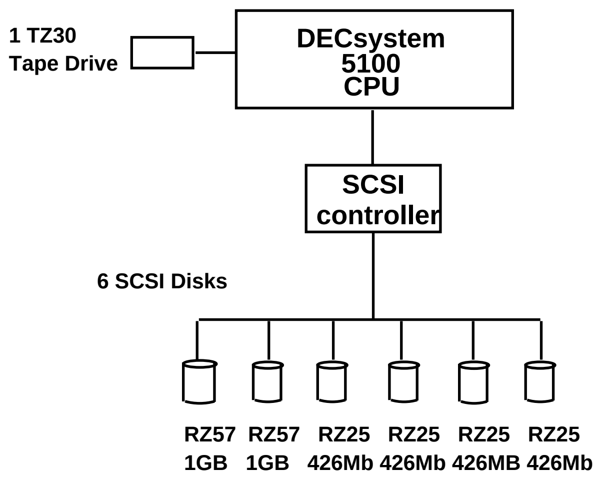
The configurations used for the benchmark and the priced system were the same.

The configuration consisted of a DECsystem 5100 with 32 Megabytes (Mbytes) of main memory and one embedded SCSI controller supporting 2 RZ57 disk drives (1 Gbyte each) and 4 RZ25 disk drives (426 Mbytes each). The TZ30 tape drive is included with the DECsystem 5100.

We enabled continuous archiving of the logical logs. The logical logs were backed up to an archive device. One 1 Gbyte RZ57 disk drive was used for this purpose. This disk drive provided the necessary storage capacity so that 8 hours of log data could be kept on-line. Informix transaction logging was at all times set to unbuffered mode.

### Benchmark and Priced System Configuration

The diagram that follows represents the benchmark configuration and priced system configuration.



## 2 - Clause 2 Related Items

### ACID Properties

*Results of the ACIDity test (specified in Clause 2) must describe how the requirements were met.*

Clause 2 of the TPC Benchmark B Standard lists specific tests to ensure the atomicity, consistency, isolation, and durability (ACID) properties of the SUT (System Under Test). The following subsections show how the tests required in Clause 2 were performed. All mechanisms needed to ensure full ACID properties were enabled during both the measurement and test periods. A fully-scaled database was used for the atomicity, consistency, and isolation tests.

## 2.1 Atomicity Tests

### Atomicity of Completed Transaction

*Perform the standard TPC Benchmark B transaction for a randomly selected account and verify that the appropriate records have been changed in the Account, Branch, Teller, and History files/tables.*

The following test was performed and verified the atomicity of completed transactions:

1. Select a random Branch record.
2. Select a random Teller record.
3. Select a random Account record.
4. Count the History records.
5. Using the randomly selected records, perform the following steps:
   A. Update the Branch record.
   B. Update the Teller record.
   C. Update the Account record.
   D. Insert the History record.
   E. Commit the transaction.
   F. Select the Branch record.
   G. Select the Teller record.
   H. Select the Account record.
6. Count the History records. Verify that the History record count reflects the committed transaction.

### Atomicity of Aborted Transaction

*Perform the standard TPC Benchmark B transaction for a randomly selected account, substituting an ABORT of the transaction account for the COMMIT of the transaction. Verify that the appropriate records have not been changed in the Account, Branch, Teller, and History files/tables.*

The following test was performed, and verified the atomicity of aborted transactions:

1. Select a random Branch record.
2. Select a random Teller record.
3. Select a random Account record.
4. Count the History records.

5. Using the randomly selected Branch, Teller and Account records from above, do the following:
   - A. Update the Branch record.
   - B. Update the Teller record.
   - C. Update the Account record.
   - D. Insert the History record.
   - E. Abort the transaction and perform a rollback recovery.
   - F. Select the Branch record.
   - G. Select the Teller record.
   - H. Select the Account record.
6. Count the History records. Verify that the History record count has not changed.

## 2.2 Consistency Tests

*Consistency is the property of the application that requires any execution of the transaction to take the database from one consistent state to another.*

The following tests were performed and verified the consistency property of transactions:

1. Consistency of Branch and Teller records before transactions
   - A. Select Branch balances for each Branch record.
   - B. Select the sum of Teller balances for each Branch record.
   - C. Verify that the balance for each Branch record is equal to the balance of its Teller records.
2. Consistency of Branch and Teller records after transactions
   - A. For the entire History file, count the History records and sum their delta values.
   - B. Perform the standard TPC Benchmark B test and record the number of committed transactions.
   - C. Repeat step 1.
3. Consistency of History files
   - A. For the entire History file, count the History records and sum their delta values.
   - B. Verify that this History record count equals the sum of History record count taken in step 2A plus the number of committed transactions.
   - C. Verify that the difference between the final History delta sum and the initial History delta sum equals the difference between the final and initial Branch record balances.

## 2.3 Isolation Tests

*Operations of concurrent transactions must yield results which are indistinguishable from the results which would be obtained by forcing each transaction to be serially executed to completion in some order.*

The following tests were performed and verified the isolation property of the transactions for conventional locking used by the database system:

### Isolation of Completed Transactions

1. Select the Branch balance for a Branch record (Branch B).
2. Start transaction 1.
   A. Update the Branch B record with delta(1).
   B. Stop just prior to committing transaction 1.
3. Start transaction 2.
   A. Attempt to update Branch B with delta(2).
   B. Transaction 2 hangs.
4. Resume transaction 1.
   A. Update the Teller record.
   B. Update the Account record.
   C. Insert the History record.
   D. Commit transaction 1.
5. Resume transaction 2.
   A. Update the Teller record.
   B. Update the Account record.
   C. Insert the History record.
   D. Commit transaction 2.
6. Select the Branch balance for Branch B. The balance should equal the previous balance plus delta(1) and delta(2).

### Isolation of Aborted Transactions

1. Start transaction 1.
   A. Update Branch B with delta(3).
   B. Stop just prior to committing transaction 1.
2. Start transaction 2.
   A. Attempt to update Branch B with delta(4).
   B. Transaction 2 hangs.
3. Resume transaction 1.

   A. Update the Teller record.
   B. Update the Account record.
   C. Insert the History record.
   D. Abort transaction 1 and perform a rollback recovery.
4. Resume transaction 2.
   A. Update the Teller record.
   B. Update the Account record.
   C. Insert the History record.
   D. Commit transaction 2.
5. Select the Branch balance for Branch B. The balance should equal the previous balance plus delta(4).

The preceding isolation tests were repeated for Teller and Account files (that is, by generating a conflict on a Teller and then an Account record).

## 2.4 Durability Tests

*The tested system must guarantee the ability to preserve the effects of committed transactions and ensure database consistency after recovery from any one of the failures listed below:*

- *Permanent irrecoverable failure of any single durable medium containing database, ABTH (Accounts, Branch, Teller, and History) files/tables, or recovery log data.*
- *Instantaneous interruption (system crash/system hang) in processing which requires system reboot to recover.*
- *Failure of all or part of memory (loss of contents).*

The following test was performed for each of the preceding types of failures to verify the durability property of the SUT:

- For the entire History file, count the History records.
- Perform the standard TPC Benchmark B test and record the committed transactions in the success file.
- Cause one of the preceding types of failure.
- Restart the system under test for this failure as required in Clause 2.5.3.
- Verify that every record in the success file has a corresponding record in the History file.
- For the entire History file, count the History records. Verify that the number of records in the History file is greater than or equal to the original count obtained in step 1 plus the number of records in the success file. If they are different, the new History file must contain additional records and the difference must be less than or equal to the number of terminals (tellers) simulated.

The durability tests were run on a database sized at 10% of the fully loaded database, i.e., 3 TPS, in order to comply with future, expected TPC requirements.

In addition, the durability test "failure of all or part of memory  (loss of contents)," i.e. complete power failure, was applied to the fully loaded database under full load conditions.  This  too was done to comply  with future, expected TPC requirements.

*In addition, the test sponsors must guarantee that, to the best of their knowledge, a fully-loaded system would pass the durability tests.*

To the best of the test sponsor's knowledge, a fully-loaded and fully-scaled system would pass the durability tests.

## 3 - Clause 3 Related Items

### 3.1 ABTH Data Storage Distribution

*The distribution across storage media of ABTH (Accounts, Branch, Teller, and History) files/tables and all logs must be explicitly depicted.*

The following diagram shows how the databases were distributed on disk media on the DECsystem 5100 test system for both the benchmark and priced system configurations.

The physical log was placed on a 426 Mbyte RZ25 disk drive using 210 Mbytes.  The rootdbs space containing the logical logs and catalog were located on another 426 Mbyte RZ25 disk drive using 230 Mbytes for the partition.  The rootdbs mirror (logical log) was located on another 426 Mbyte RZ25 disk drive.

**ABTH Data Storage Distribution Diagram**

| | Partition | File/Data | Kbytes Used (000's) | % of Data |
|---|---|---|---|---|
| RZ57 | rz1a | /root | 15.6 | |
| | rz1b | swap dump | 87.9 | |
| | rz1g | /usr | 107.4 | |
| | rrz1h | account | 71.6 | 20% |
| | | history | 694.5 | 50% |
| RZ57 | rrz0c | log archive | 945.4 | 100% |
| RZ25 | rrz2h | teller | .032 | 100% |
| | | branch | .0032 | 100% |
| | | history (active) | 146.5 | 10% |
| | rrz2g | account | 71.6 | 20% |
| | | history | 198.1 | 14% |
| RZ25 | rrz3g | rootdbs (logical log) | 230 | 100% |
| | rrz3h | account | 71.6 | 20% |
| | | history | 114.6 | 8% |
| RZ25 | rrz4g | rootdbs mirror (logical log) | 230 | 100% |
| | rrz4h | account | 71.6 | 20% |
| | | history | 114.6 | 8% |
| RZ25 | rrz6g | physical log | 210 | 100% |
| | rrz6h | account | 71.6 | 20% |
| | | history | 134.6 | 10% |

The distribution of the database is further evidenced and illustrated by the Informix tbstat utility  tbstat_d.

**tbstat_d Listing**

```
 RSAM Version 4.10.UE1P1 -- On-Line -- Up 05:29:03 -- 12016 Kbytes

Dbspaces
address number flags fchunk nchunks flags    owner     name
80b47c     1      2      1       1      M     informix   rootdbs
80b4ac     2      1      2       1      N     informix   physdbs
80b4dc     3      1      3       1      N     informix   tbhdbs
80b50c     4      1      4       5      N     informix  acctdbs
 4 active, 20 total

Chunks
address chk/dbs offset    size     free      bpages flags   pathname
8084fc    1  1   500      115000  100178           PO-    /dev/rrz3g
809cbc    1  1   500      115000  0                MO-    /dev/rrz4g
808594    2  2   500      115000  8842             PO-    /dev/rrz6g
80862c    3  3   500      75000   72522            PO-    /dev/rrz2h
8086c4    4  4   45000    35800   0                PO-    /dev/rrz2g
80875c    5  4   20000    35800   0                PO-    /dev/rrz3h
8087f4    6  4   20000    35800   297              PO-    /dev/rrz4h
80888c    7  4   20000    35800   297              PO-    /dev/rrz6h
808924    8  4   125000   35800   297              PO-    /dev/rrz1h
 8 active, 40 total
```

## 3.1.1 History Storage and Recovery

*Within the priced system, there must be sufficient on-line storage to support 8 hours of recovery log data, if required to recover from any single point of failure, plus any other expanding system files (see Clause 7.1) and durable history records/rows for 30 eight-hour days at the published tpsB rate (i.e., 30 x 8 x 60 x 60 = 864,000 records/rows per tpsB).*

The history and log file storage calculations are shown below:

History File Storage

The following calculations were used to determine the aggregate size of the history file.

| | |
|---|---|
| INFORMIX-OnLine Page Size | 2048 bytes |
| Overhead per Page | 32 bytes |
| Overhead per Row | 4 bytes |
| History Table Row Size | 50 bytes |

History Rows per Page = (Page Size - Page Overhead)/(Row Size + Row Overhead)
truncated to lower integer value = 37 History Rows per Page

History Rows Needed = (tpsB * 3600 * 8 * 30) = 25,920,000 Rows

History Space = (Rows Needed/37) * 2048 = 1,434,707,027 bytes =
1,401,081.081 Kbytes

Logfile Storage

INFORMIX-OnLine records a timestamp for every completed logical log archived, we used this timestamp to calculate the average time to archive one logical log during the steady state run. The average time to fill a 5 Mbyte logical log was approximately 145 seconds which equates to a logfile requirement of 993,100,000 bytes.

|  |  |
|---|---|
| Total Logfile Space Needed: | 993,100,000 bytes |
| Active Logfile Space Supplied: | - 25,000,000 bytes |
|  | -------------- |
|  | 968,100,000 bytes |

Additional 8 hour log space was required. A 1 Gbyte RZ57 disk drive was used to accommodate this requirement.

Informix tbstat output for the logical logs and part of the message log follow.

**tbstat_l listing**

```
 RSAM Version 4.10.UE1P1 -- On-Line -- Up 05:29:03 -- 12016 Kbytes

Physical Logging
Buffer      bufused   bufsize    numpages   numwrits   pages/io
P-1         15        16         36351      2272       16.00
            phybegin  physize    phypos     phyused    %used
            200486    105000     84030      4175       3.98

Logical Logging
Buffer     bufused bufsize   numrecs  numpages  numwrits     recs/pages pages/io
L-1        0       16        331542   22530     12259        14.7       1.8

address     number     flags      uniqid     begin      size   used  %used
884408      1          F------    0          100b02     2500   0     0.00
884424      2          U-B---L    32         1014c6     2500   2500  100.00
884440      3          U---C--    33         101e8a     2500   31    1.24
88445c      4          F------    0          10284e     2500   0     0.00
884478      5          F------    0          103212     2500   0     0.00
```

### Message Log File Listing

```
 RSAM Version 4.10.UE1P1 -- On-Line -- Up 05:29:04 -- 12016 Kbytes

Message Log File: /usr/users/informix/online.log
11:56:20 Logical Log 24 Complete
11:56:55 Logical Log 24 Backed Up
11:58:45 Logical Log 25 Complete
11:59:23 Logical Log 25 Backed Up
12:01:13 Logical Log 26 Complete
12:01:53 Logical Log 26 Backed Up
12:03:40 Logical Log 27 Complete
12:03:51 Checkpoint Completed
12:04:17 Logical Log 27 Backed Up
12:06:05 Logical Log 28 Complete
12:06:47 Logical Log 28 Backed Up
12:08:30 Logical Log 29 Complete
12:09:13 Logical Log 29 Backed Up
12:10:57 Logical Log 30 Complete
12:11:43 Logical Log 30 Backed Up
12:13:23 Logical Log 31 Complete
12:13:34 Checkpoint Completed
12:14:06 Logical Log 31 Backed Up
12:15:46 Logical Log 32 Complete
12:16:19 Logical Log 32 Backed Up
```

Appendix E contains a complete listing of  the disk devices to support the test.

## 3.2 Database Contents and Method of Population

*A description of how the database was populated, along with sample contents of each ABTH file/table to meet the requirements described in Clause 3.*

### Database Contents

Appendix C contains the database population program and Appendix D contains samples of the contents of the database files used in the tests.

### 3.3 Type of Database

*A statement of the type of database utilized, e.g., relational, Codasyl, flat file, etc.*

These TPC Benchmark B tests used INFORMIX-OnLine, a relational database management system.

## 4 - Clause 4 Related Items

There are no Clause 4 Related Items in the checklist for TPC-B.

## 5 - Clause 5 Related Items

### 5.1 Method of Verification of Random Number Generator

*The method of verification of the random number generator should be described.*

Branch, Teller, and Account IDs were generated by the random number generation routines, random() and srandom() in the bench.h code. Random()/srandom() use a non-linear additive feedback random number generator, employing a default table size of 31 long integers to return successive random numbers in the range from 0 to (2**31)-1. These routines produce a more random sequence than earlier subroutines such as rand(). Random() and srandom() are well known random number generation routines. Randomness of the generated values are further verified by observing the 85/15 distribution rule, which showed that approximately 85% of the transactions submitted to a Branch had the Account belong to that Branch.

### 5.2 Horizontal Partitioning Disclosure

*Vendors must clearly disclose if horizontal partitioning is used. Specifically, vendors must:*

- *Describe textually the extent of transparency of the implementation*
- *Which tables/files were accessed using partitioning*
- *How partitioned tables/files were accessed*

Horizontal partitioning, i.e. the partitioning of a table according to some logical order, was not used. The account relation records were randomly distributed over multiple (5) disk drives.

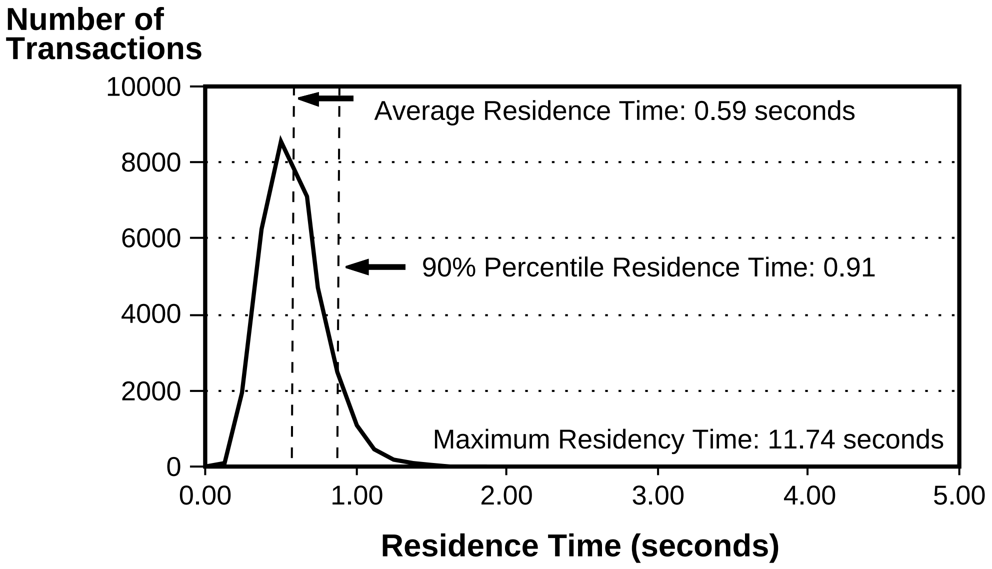
## 6 - Clause 6 Related Items

*Report all the data specified in Clause 6.6, including maximum and average residence time, as well as performance curves for number of transactions vs. residence time (see Clause 6.6.1) and throughput vs. level of concurrency for three data points (see Clause 6.6.5).*

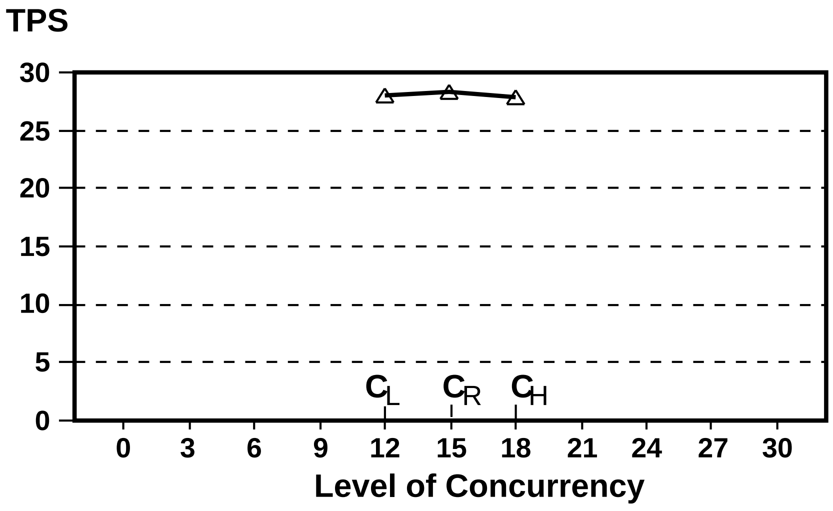
The graphs and tables in this section show the response time performance results.

Please note that for all performance runs the database was scaled for 30 TPS.

**Residence Time Frequency Distribution for All Transactions**



**Throughput Versus Level of Concurrency**



### Concurrency Legend

| Measured Points | Level of Concurrency | TPS | Average Residence Time (seconds) |
|---|---|---|---|
| $C_L$ | 12 | 27.9 | 0.49 |
| $C_R$ | 15 | 28.2 | 0.59 |
| $C_H$ | 18 | 27.8 | 0.69 |

### Profile of Executed Transactions

| Description | Result |
|---|---|
| Remote Transactions (see Clause 6.6.2) | 15.00% |
| Home Transactions | 85.00% |
| Transactions started and not completed during measurement interval (see Clause 6.6.3) | 0.05% |
| Number of transaction started but not completed | 15 |
| Total number of transactions | 33,058 |
| Average residence time for all transactions | 0.59 seconds |
| Maximum residence time for all transactions | 11.74 seconds |
| Percent of all transactions qualified within 2 second response time constraint | 99.90 |
| Maximum qualified throughput | 28.2 tpsB |

# 7 - Clause 7 Related Items

## 7.1 Determining Steady State

*The method used to determine that the SUT had reached a steady state prior to commencing the measurement interval should be described.*

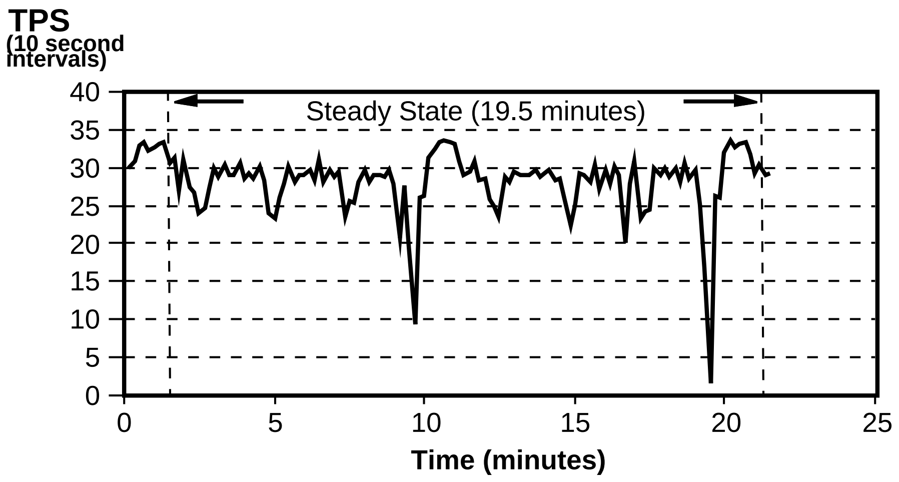
Confirmation that the SUT has reached steady state prior to the beginning of the data collection measurement interval is based on a visual inspection of the plot of TPS versus time.

The design of the benchmark driver program was such that all processes wait to be signaled to commence ramp-up work. During ramp-up, the processes begin executing identical TPC-B transactions as they do during the steady state run.

During the ramp-up, which lasted for 120 seconds, all processes began executing the identical TPC-B transaction that they do during the timed steady state run. At the end of the ramp-up period, each process independently kept track of the numbers and characteristics of its committed transactions that started during the steady state interval. The audited benchmark steady state period lasted for 19.5 minutes. When the run was done, the processes individually and independently reported their accumulated transactions and residence time results. The driver then calculated the required numbers to report.

To confirm that steady state was reached, the history table was examined. The graph title "TPS Versus Time" (below), indicates the number of transactions completed in each 10 second interval. The steady state portion is labeled on the graph. Note the pronounced dips (checkpoints) in transaction rate that occurred 2 times during the steady state for the run. Also, note the less pronounced dip that occurred as each inactive 5 Mbyte logical log is backed up.

**TPS Versus Time**



## 7.2 Work Performed During Steady State

*A description of how the work normally performed during a sustained test (for example, checkpointing, writing redo/undo log records, etc., as required by Clause 7.2), actually occurred during the measurement interval.*

When INFORMIX-OnLine receives a SQL statement from the application, it determines how to best access the data. Using an index (B-tree), INFORMIX-OnLine determines the page number from the database that the record is located on, and searches for the page in shared memory.

If the page is not in shared memory, INFORMIX-OnLine chooses a LRU Buffer in shared memory and reads the page from the database into the buffer. Typically this will take two disk reads. The first read acquires the bottom level of the B-tree index, the second disk read actually acquires the data.

When a transaction starts, a BEGIN WORK is written to the logical log buffers. When the application issues a SQL UPDATE statement (Account, Teller, and Branch) to modify a record, the copy of the record, if it is already in shared memory, is locked and updated. A transaction record is written to the logical log buffer.

At the same time, if the page in shared memory has not previously been written to, a copy of the before image of the page is written to the physical log buffer in shared memory. In addition, the before and after images of the record are written to the logical log buffer in shared memory. So, the physical log buffer contains a copy of the page that a record is on, as it looked prior to making any modification.

When the application issues a SQL COMMIT WORK statement, the logical log buffer is flushed from shared memory to the logical log on disk in a single I/O. The database pages remain in shared memory and are not written to the database at that time. Any locks that were placed by the transaction are released. This means that when an application commits a transaction to the database, the logical log buffer is written to a corresponding logical log on disk with a single I/O, and successful completion code is returned to the application.

Periodically INFORMIX-OnLine will automatically write all modified pages in shared memory to their appropriate locations in the database during a checkpoint. A checkpoint is preceded by a write of the physical log buffer to the physical log on disk. Checkpoints occur periodically during the run. With INFORMIX-OnLine there are several ways of controlling when a checkpoint occurs. For our benchmark, checkpointing occurs every time INFORMIX-OnLine starts the last logical log. We configured INFORMIX-OnLine with three logical logs. Thus, every time two logs were filled and the third started, a checkpoint would occur. In our benchmark run of 19.5 minutes, 2 checkpoints occurred.

When the checkpoint occurs, one or more background processes called page cleaners "wake up" and write all the modified pages from shared memory to the database on disk. A checkpoint record is written to the logical log buffer. A checkpoint message is written to the message log.

The page reading and writing activity to the individual chunk partitions in the database are reflected in the Informix utility tbstat_D and tbstat_p.

**tbstat_D**

```
 RSAM Version 4.10.UE1P1 -- On-Line -- Up 05:29:03 -- 12016 Kbytes

Dbspaces
address   number  flags  fchunk  nchunks  flags   owner      name
80b47c    1       2      1       1        M       informix   rootdbs
80b4ac    2       1      2       1        N       informix   physdbs
80b4dc    3       1      3       1        N       informix   tbhdbs
80b50c    4       1      4       5        N       informix   acctdbs
 4 active, 20 total
```

```
Chunks
address    chk/dbs    offset    page Rd       page Wr    pathname
8084fc     1     1    500       22503         22534      /dev/rrz3g
809cbc     1     1    500       0             22534      /dev/rrz4g
808594     2     2    500       0             36337      /dev/rrz6g
80862c     3     3    500       2686          1073       /dev/rrz2h
8086c4     4     4    45000     14271         6795       /dev/rrz2g
80875c     5     4    20000     14287         6766       /dev/rrz3h
8087f4     6     4    20000     14159         6669       /dev/rrz4h
80888c     7     4    20000     14216         6732       /dev/rrz6h
808924     8     4    125000    14356         6767       /dev/rrz1h
 8 active, 40 total
```

**tbstat_p**

```
 RSAM Version 4.10.UE1P1 -- On-Line -- Up 05:29:05 -- 12016 Kbytes

Profile
dskreads pagreads bufreads %cached dskwrits pagwrits bufwrits %cached
74462    96468    1116087  93.33   61337    116205   149548   58.99

isamtot  open start  read   write  rewrite delete commit rollbk
1434831 99   110554 147753 36867 110479 0      37094  0

ovtbls ovlock ovuser ovbuff  usercpu syscpu numckpts flushes
0      0      0      0       610.92  446.47 2        4

bufwaits lokwaits lockreqs deadlks  dltouts lchwaits ckpwaits compress
4638     8480     947296   0        0       22261    30       1
```

## 7.3 Determining Reproducibility

*A description of the method used to determine the reproducibility of the measurement results.*

Experiments were repeated at least 3 times at the maximum targeted TPS level to ensure reproducibility. The results are shown in the following table. The variation in TPS was less than 1%.

**DECsystem 5100 TPC-B Benchmark Runs**

| Run # | Processes | CPUs | tpsB | Percent < 2 sec. | Transactions | db Size | Duration |
|---|---|---|---|---|---|---|---|
| 1 | 15 | 1 | 28.2 | 99.90 | 33,058 | 30 tps | 19.5 mins |
| 2 | 15 | 1 | 28.2 | 99.90 | 32,997 | 30 tps | 19.5 mins |
| 3 | 15 | 1 | 28.0 | 99.90 | 32,855 | 30 tps | 19.5 mins |

## 7.4 Duration of Measurement Period

*A statement of the duration of the measurement period for the reported tpsB (it should be at least 15 minutes and no longer than 1 hour).*

The steady state measurement period for each experiment was 19.5 minutes and began approximately 2 minutes after all servers had begun executing transactions.

## 8 - Clause 8 Related Items

## 8.1 Description of the Driver

*If the driver is commercially available, then its inputs should be specified. Otherwise, a description of the driver should be supplied.*

The driver used was an "internal driver" (i.e., the driver software resides on the system under test, not on a remote driver machine) that controls transaction processing and performance data collection for the TPC Benchmark B runs. The driver was comprised of two parts: a control csh script and a set of identical ESQL/C transaction programs that submitted the TPC Benchmark B transactions for execution.

The control script performs the following operations:

1. forks and execs the desired number of transaction programs, passing ramp-up and measurement interval parameters as command line arguments.
2. waits for a short period of time (30 seconds) to ensure that each driver has started up and opened the test database.
3. sends a SIGUSR1 signal to each transaction process to synchronize the start of transaction processing.
4. waits until all transaction processes have completed the benchmark run.
5. invokes a program called sumrun to sum the performance statistics collected by the transaction processes involved in the benchmark run.

   After each transaction program completes a benchmark run, the transaction program stores residence time counts, incomplete transaction counts, and other performance statistics in a database table named "results". The sumrun program sums all "results" records for a run and inserts aggregate run values into a table named "runs".

Each transaction program performs the following operations:

1. examines its command line arguments to determine the ramp-up and measurement intervals to use.
2. waits until it receives a SIGUSR1 signal before initiating transaction processing.
3. continuously submits TPC-B transactions, with 0 sleep time. The transaction program collects response time statistics in internal program data structures, but does not begin collecting them until the ramp-up period has completed.
4. inserts its collected performance statistics into a "results" table record once the measurement interval has completed. It is the contents of these "results" records that are summed by the sumrun program.

"Success files" were implemented through the tpc.ec application program by writing synchronously using fsync() and flushing the confirmation of transactions to standard output. This was captured into a file nohup.out running under the Korn shell.

## 9 - Clause 9 Related Items

### 9.1 Hardware and Software Components

*A detailed list of hardware and software used in the priced system. Each item must have vendor part number, description, and release/revision level, and either general availability status or committed delivery date. If package pricing is used, contents of the package must be disclosed.*

#### 9.1.1 Priced System Configuration Tables

The following tables show the hardware and software components in the priced DECsystem 5100 system:

| Component | Product | Quantity |
|---|---|---|
| Processor | DECsystem 5100 | 1 |
| Memory | | 32 Megabytes |
| Tape drive | TZ30 | 1 |
| Disk controller | SCSI | 1 |
| Disks | RZ57 | 2 |
| | RZ25 | 4 |
| Operating system | ULTRIX 4.2 | |
| Database | INFORMIX-OnLine V4.10 | |
| | INFORMIX ESQL/C | |

### 9.1.2 Package Pricing

| Package Description | Model # |
|---|---|
| DECsystem 5100<br>MS44-DA 32MB memory<br>ThinWire/ThickWire Ethernet<br>Four asynchronous lines<br>One SCSI controller<br>2 RZ25 426MB SCSI disks<br>ULTRIX license 1-4 users<br>UWS Server license<br>Hardware Documentation<br>One TZ30 95MB Tape Drive<br>With One Year System Warranty | DU-51MT4-B9 |

| Package Description | Model # |
|---|---|
| 2xRZ57 1GB Disks with BA42 Box | SZ12C-CA |

## 9.2 Total Price of System Configuration

*The total price of the entire configuration is required including: hardware, software and maintenance charges. Separate component pricing is recommended. The basis of all discounts used shall be disclosed.*

This section lists the separate components in the priced system and their associated purchase and maintenance costs. All items are currently available. All prices were taken from the Digital Standard Pricing System (DSPS) on November 21, 1991. A description of the packages used in the pricing is contained in Section 9.1.2.

Informix prices were taken from Informix’s price list, titled "Informix Americas Price List, Advanced Products, Release 4.0 or Greater, Class D", dated August 1, 1991.

### 9.2.1 Hardware Pricing

The Digital TPC Benchmark B DECsystem 5100 test used packaged hardware systems whenever possible to simplify configurations to the fewest number of line items. Disks were connected using one SCSI controller. The system used a TZ30 tape drive to load the software and back up the database.

The purchase price of all systems includes one year of hardware warranty service. Post-warranty hardware service is configured for an additional four years.

The following levels of post-warranty hardware service are used in the system pricing:

DECsystem Support 9x5 (DS9) 2-4 hours response time.
Basic monthly charge (BMC) warranty level is the same as the DS9 to which the hardware is directly attached.
Basic System Support (BSS) with a warranty upgrade to DS9.

### 9.2.2 Software Pricing

The priced system uses the following software products:

- ULTRIX V4.2 Operating System
- INFORMIX-OnLine relational database management system
- INFORMIX-ESQL/C

The ULTRIX license purchase includes one year of warranty service. Post-warranty service is configured for additional four years. The software warranty and service level are the same as the service level for the hardware system on which the software operates. The level of post-warranty service is Software Support Service (SSS).

### 9.2.3 Price Discounts

Digital's five (5) years warranty pricing is as follows:

- the unit price carries one (1) year warranty.
- the price of year 2-5 warranty adder is calculated according to this formula:
  - (warranty/month)*12*(1+1+1.07+$(1.07)^2$)=(1.053725)*48*(warranty/month)

The pre-payment maintenance (warranty) discount is calculated at 25% of the year 2-5 warranty price.

Informix's five-year prepaid maintenance option consists of five years of maintenance for four times the price of standard maintenance.

## 9.2.4 System Pricing Summary

DECsystem 5100 TPC-B = 28.2 TPS

| DESCRIPTION | MODEL # | US LIST PRICE | QTY | TOTAL PRICE | SERVICE LEVEL | UNIT $/MO. | DELT # MO | 5 YR TOTAL SERVICE | PRICE+SRVC 5 YR COST |
|---|---|---|---|---|---|---|---|---|---|
| Digital Price (21 November 1991) | | | | | | | | | |
| Host and Database | | | | | | | | | |
| DS5100 32 MB | DU-51MT4-B9 | $19,831.00 | 1 | $19,831.00 | BSS | $0.00 | 48 | $0.00 | $19,831.00 |
| Warranty Upgrade to DS9 | FM-DECUP-12 | $252.00 | 1 | $252.00 | DS9 | $174.00 | 48 | $8,352.00 | $8,604.00 |
| 2 RZ57 1.0 GB Disk in Exp. Box | SZ12C-CA | $16,324.00 | 1 | $16,324.00 | DS9/BMC | $152.00 | 48 | $7,296.00 | $23,620.00 |
| RZ25 | RZ25-EF | $2,550.00 | 2 | $5,100.00 | DS9/BMC | $25.00 | 48 | $2,400.00 | $7,500.00 |
| ULTRIX-32 V4.2 Media & Doc. | QA-VYVAA-H5 | $3,240.00 | 1 | $3,240.00 | NA | $.00 | 48 | $.00 | $3,240.00 |
| | | | | $44,747.00 | | | | $18,048.00 | $62,795.00 |
| Years 2-5 Warranty Adder=5.3725% | | | | | | | | $969.63 | $969.63 |
| Digital Subtotal | | | | $44,747.00 | | | | $19,017.63 | $63,764.63 |
| Prepayment Maintenance Discount =(25%) | | | | | | | | ($4,754.41) | ($4,754.41) |
| Digital Total | | | | $44,747.00 | | | | $14,263.22 | $59,010.22 |
| | | | | | | $/YEAR | YR | | |
| Informix Price (1 August 1991) | | | | | | | | | |
| INFORMIX-OnLine (Class "D" License) | Full Dev./Run T | $3,300.00 | 1 | $3,300.00 | SSS | $590.00 | 4 | $2,360.00 | $5,660.00 |
| INFORMIX-ESQL/C | Full Dev./Run T | $660.00 | 1 | $660.00 | SSS | $200.00 | 4 | $800.00 | $1,460.00 |
| Informix Total | | | | $3,960.00 | | | | $3,160.00 | $7,120.00 |
| CONFIGURATION TOTALS | | | | $48,707.00 | | | | $17,423.22 | $66,130.22 |
| | | | | | | TPS- | | | 28.2 |
| | | | | | | $/TPS | | | $2,345 |

### 9.3 Performance and Price/Performance

*A statement of the measured tpsB, and the calculated price/tpsB.*

The following table shows measured tpsB and price/tpsB results for the tested system:

| CPU Model | Software | TPS (tpsB) | Price per TPS ($/tpsB) |
|---|---|---|---|
| DECsystem 5100 | ULTRIX 4.2 and INFORMIX-OnLine 4.10 | 28.2 | $2,345 |

## 10 - Clause 10 Related Items

None.

## 11 - Clause 11 Related Items

### 11.1 Independent Auditor's Report

*If the benchmark has been independently audited, then the auditor's name, address, phone number, and a brief audit summary report indicating compliance must be included in the full disclosure report. A statement should be included, specifying when the complete audit report will become available and who to contact in order to obtain a copy.*

Appendix G contains the complete independent auditor's report by KPMG Peat Marwick for the tests described in this report.

# Appendix A

# Application Code

This appendix contains the source code of the application programs that implement the TPC Benchmark B transaction.

## A.1 tpc.ec source code

```
#include <stdio.h>
#include <sys/signal.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include <math.h>
$include sqlca ;
#include "bench.h"

$long acct_bal, cntr, seconds, intvl, startsec, tot_response ;
$int branch_num, teller_num, acct_num, delta, acct_branch, run, procnum ;
$int notdone, tmslot[BUCKETS+1] ;
int rampup, runtime, timing, thru, measure, bucketval, transactions, verbose ;
int longest_tran;

settimer() {  timing = ~timing ; }

setmeasure()
{
                intvl = (measure) ? rampup : runtime ;
                thru++ ;
                measure = ~measure ;
                 startsec = time(0) ;
}

main(argc,argv)
                int argc ;
                char **argv ;
{
                int i, *rnum, do_trans() ;

                 runtime = rampup = intvl = 0 ;
                transactions = -1 ;
                procnum = atoi(argv[1]) ;
                i = 1 ;

                 while (++i < argc) {
                                if (strcmp(argv[i], "-s") == 0)
                                runtime += atoi(argv[++i]) ;
                                else if (strcmp(argv[i], "-m") == 0)
                                runtime += (60 * atoi(argv[++i])) ;
                                else if (strcmp(argv[i], "-h") == 0)
                                runtime += (3600 * atoi(argv[++i])) ;
```

```
                                        else if (strcmp(argv[i], "-t") == 0)
                                        transactions = atoi(argv[++i]) ;
                                        else if (strcmp(argv[i], "-r") == 0)
                                        rampup = atoi(argv[++i]) ;
                                        else if (strcmp(argv[i], "-v") == 0)
                                        verbose = 1 ;
                                        else {
                                                    fprintf(stderr,"usage: tp1 <proc #> [-t <n>] [-r <n>]
[-h <n> -m <n> -s <n>]\n") ;
                                        exit(1) ;
                                        }
 }
                    RandSeed(getpid()) ;

                    if (runtime == 0)
                                        runtime = (transactions == -1) ? 300 : 30000 ;

                    printf("process %d: procnum=%03d runtime=%d seconds / %d transac
                                        tions\n", getpid(),procnum,runtime,transactions) ;

                    cntr = tot_response = run = measure = timing = thru = intvl = notdone = 0 ;
                    for (i=0; i < 50; i++)
                                        tmslot[i] = 0 ;

                    $ database tpc ;
                       SqlErr("attach to database") ;

                    $ select max(number) into $run from results ;
                       SqlErr("select from results") ;
                    if (run < 0)
                                        run = 0 ;
                     ++run ;

                     do_prepare() ;

                     bucketval = RPTINTVL * 1000 / BUCKETS ;
                    intvl = rampup ;
                    signal(SIGUSR1,settimer) ;
                    sigpause(0) ;

                    printf("%d starting\n",procnum) ;  do_trans() ;  testend() ;

}

do_prepare()
{
                    $char s[512];

                         sprintf(s,"%s %s %s%d%s %s commit work",
                         "update account set balance = balance + ? where current of sel_acct;",
                         "update teller set balance = balance + ? where number = ?;",
                         "insert into history",procnum % HISTORY,
                         " values(?,?,?,?,CURRENT YEAR TO SECOND,'the rest is history');",
                         "update branch set balance = balance + ? where number = ?;") ;
                     $ prepare tpc_trans from $s;
                       SqlErr("prepare updall");

                     $ prepare bwork from "begin work" ;
                       SqlErr("prepare begin work") ;

                     $ declare sel_acct cursor for
                                        select balance into $acct_bal from account
                                                    where number = $acct_num
                                                    for update of balance ;
                       SqlErr("declare cursor") ;

                     $ set isolation to cursor stability ;
                       SqlErr("set isolation") ;

                     $ set lock mode to wait;
                       SqlErr("set lock mode");
}

do_trans()
{
                    long timediff ;
                    char s[100] ;
                    struct timeb clk_beg,clk_end ;

                                        startsec = time(0) ;
                    if (rampup == 0)
                                        setmeasure() ;
                    else
                                        thru++ ;

                     while (timing && (cntr != transactions)) {

                            /*
                             * select a random branch, a random teller at that branch, and
                             * 85% of the time a random account at that branch, and 15% of
                             * the time a random acccount at a different branch.
                             */

                    teller_num = RandVal() % T_RECS ;
                    branch_num = teller_num / T_PERB ;
                    acct_num = RandVal() % A_PERB ;

                    if ((RandVal() % 100) < 85)
                                        acct_branch = branch_num ;
                    else {
                                        do              /* endless loop when TPS_SIZE=1 */
                                                    acct_branch = RandVal() % B_RECS ;
                                     while (acct_branch == branch_num) ;
                    }

                    acct_num = acct_branch * A_PERB + acct_num ;
                    delta = RandVal() % 1999999 - 999999 ;
                    if (measure)
                                     notdone++ ;
```

```
ftime(&clk_beg) ;

$ execute bwork ;

$ open sel_acct ;
   SqlErr("open cursor") ;

$ fetch sel_acct ;
   SqlErr("fetch cursor") ;

if (sqlca.sqlcode == 0) {
 $ execute tpc_trans using
                    $delta,
                    $delta, $teller_num,
                    $acct_num, $teller_num, $branch_num, $delta,
                    $delta, $branch_num ;
}

ftime(&clk_end) ;

if (sqlca.sqlcode != 0) {
         sprintf(s,"in transaction %d acc#: %d branch#: %d teller#: %d",
                                        cntr, acct_num, branch_num, teller_num) ;
 SqlFatal(s) ;
 /*
 $ rollback work ;
 */
}

timediff = clk_end.time - startsec ;
if (timediff > intvl) {
                    if (thru == 2)
                                        settimer() ;
                     else
                                        setmeasure() ;
}

if (measure) {
        timediff = (clk_end.time - clk_beg.time) * 1000
                                        + clk_end.millitm - clk_beg.millitm ;
 if(timediff > longest_tran)
        longest_tran = timediff;
 tot_response += timediff ;
 timediff /= bucketval ; /* 0-.124, .125-0.249, etc. seconds */
 if (timediff > BUCKETS)
         timediff = BUCKETS ;
 tmslot[timediff]++ ;
 cntr++ ;
 if(verbose)
 {
        printf("procnum %3d: tran %d completed!\n",procnum,cntr);
        fflush(stdout);
        fsync(1);
 }
```

```
                     notdone-- ;
                    }

             }

             seconds = (transactions > 0) ? (time(0)-startsec) : runtime ;

}

testend()
{
             int hrs, min, sec ;

                    hrs = seconds / 3600 ;
                    min = (seconds - hrs * 3600) / 60 ;
                    sec = seconds - hrs * 3600 - min * 60 ;
                    printf("procnum %3d completed %6d transactions in %4d:%02d:%02d, long-
est=%d msec.\n",
                                        procnum, cntr, hrs, min, sec, longest_tran) ;

                    $ insert into results values(
                                         $run, $procnum, $seconds, $cntr, $notdone, $tot_response,
                                         $tmslot[0],$tmslot[1],$tmslot[2],$tmslot[3],$tmslot[4],
                                         $tmslot[5],$tmslot[6],$tmslot[7],$tmslot[8],$tmslot[9],
                                         $tmslot[10],$tmslot[11],$tmslot[12],$tmslot[13],$tmslot[14],
                                         $tmslot[15],$tmslot[16],$tmslot[17],$tmslot[18],$tmslot[19],
                                         $tmslot[20],$tmslot[21],$tmslot[22],$tmslot[23],$tmslot[24],
                                         $tmslot[25],$tmslot[26],$tmslot[27],$tmslot[28],$tmslot[29],
                                         $tmslot[30],$tmslot[31],$tmslot[32],$tmslot[33],$tmslot[34],
                                         $tmslot[35],$tmslot[36],$tmslot[37],$tmslot[38],$tmslot[39],
                                         $tmslot[40]) ;
                                         SqlErr("insert into results") ;

                    $ close database ;
                      SqlErr("close database") ;
 }
```

## A.2 createdb.ec source code

```
#include <stdio.h>
#include "bench.h"
 $include sqlca ;

/*
* FILE: createdb.ec (for OnLine)
*
* Creates the database and related tables, except result-consolidation
* tables. It is possible to place the tables on different drives by
* adding location options to the CREATE TABLE statements.
*
* You can also decide to place logging on the database by adding it
* to the CREATE DATABASE statement. However, the loading programs
* provided assume no transaction logging, so you should turn on logging
* afterward via archiving and changing the database logging mode.
*
* The configuration here accommodates scaling to 100 TPS.
*
*/

main()
{

        $ create database tpc in TBHDBS ;
          SqlErr("create database") ;

        $ grant dba to public ;
           SqlErr("grant dba") ;

        printf("Database created, permission granted\n") ;

        $ create table branch      (
                number numeric(2,0),
                balance numeric(10,0),
                fillstr char(92)
        )
        lock mode row
        ;
          SqlErr("create branch") ;
        printf("Branch created\n") ;

        $ create table teller
        (
                number numeric(4,0),
                balance numeric(10,0),
                branch numeric(2,0),
                fillstr char(89)
        )
        extent size 200
        next size 100
        lock mode row
```

```
                    ;
                      SqlErr("create teller") ;
                    printf("Teller created\n") ;

                    $ create table account      (
                    number         numeric(8,0),
                    balance        numeric(10,0),
                    branch         numeric(2,0),
                    fillstr         char(87)
                    )
                    in acctdbs
                    extent size 5000
                    next size 1000
                    ;
                      SqlErr("create account") ;
                    printf("Account created\n") ;

                    $ close database ;
                      SqlErr("close database") ;

                     exit(0) ;
}
```

## A.3 createhist.ec source code

```
#include <stdio.h>
#include "bench.h"
 $include sqlca ;

/*
* FILE: createhist.ec (for OnLine)
*
* Creates the history tables. Number of tables is HISTORY in "bench.h".
*
* The configuration here accommodates scaling to 100 TPS.
*
*/

main()
{
            $char dstr[200] ;
            $int cnt, i ;

             $ database tpc ;
               SqlErr("connect to database") ;

            $ select count(*) into $cnt from systables
                        where tabname matches "hist" ;
              SqlErr("test for history tables") ;

            $ select count(*) into $cnt from systables
                        where tabname matches "hist*" ;
              SqlErr("test for history tables") ;

            if (cnt) {
                        printf("Dropping History tables...\n") ;
                        for (i=0; i < cnt; i++) {
                                    sprintf(dstr,"drop table history%d",i) ;
                                    $ prepare drop_tab from $dstr ;
                                      SqlErr("prepare drop") ;
                                    $ execute drop_tab ;
                                      SqlErr(dstr) ;
                        }
             }

            for (i=0; i < HISTORY; i++) {

                        sprintf(dstr, "%s%d (%s,%s,%s,%s,%s,%s) %s %s %s",
                                    "create table history", i,
                                                "account integer",
                                                "teller integer",
                                                "branch integer",
                                                "delta char(11)",
                                                "tstamp datetime year to second",
                                                "fillstr char(22)",
                                                "extent size 1000",
                                                "next size 1000",
                                                    "lock mode row"
                                          ) ;
                                 $ prepare make_tab from $dstr ;
                                    SqlErr("prepare create") ;
                                 $ execute make_tab ;
                                    SqlErr("execute history") ;
                                 printf("History%d table created\n",i) ;

 }

 $ close database ;
    SqlErr("close database") ;

 exit(0) ;
}
```

## A.4 createruns.ec source code

```
#include <stdio.h>
#include "bench.h"
$include sqlca ;

/*
 * FILE:  createruns.ec
 *
 * Creates the results tables for cumulative reporting
 *
 */

main()
{
   $int cnt ;

                 $ database tpc ;
                   SqlErr("open database") ;

                $ select count(*) into $cnt from systables
                                where tabname = "runs" ;
                   SqlErr("test for runs table") ;

                if (cnt) {
                                $ drop table runs ;
                                  SqlErr("drop table runs") ;
                }

                $ select count(*) into $cnt from systables
                                where tabname = "results" ;
                  SqlErr("test for results table") ;

                if (cnt) {
                                $ drop table results ;
                                  SqlErr("drop table results") ;
 }

   $ create table runs
                (
                 num           serial,
                 numprocs      smallint,
                 test_intvl    integer,
                 total_xact    integer,
                 total_inc     integer,
                 resp_time     integer,
                 cpus          smallint,
                 test_size     smallint,
                 tslot01       smallint,
                 tslot02       smallint,
                 tslot03       smallint,
                 tslot04      smallint,
                 tslot05      smallint,
                 tslot06      smallint,
                 tslot07      smallint,
                 tslot08      smallint,
                 tslot09      smallint,
                 tslot10      smallint,
                 tslot11      smallint,
                 tslot12      smallint,
                 tslot13      smallint,
                 tslot14      smallint,
                 tslot15      smallint,
                 tslot16      smallint,
                 tslot17      smallint,
                 tslot18      smallint,
                 tslot19      smallint,
                 tslot20      smallint,
                 tslot21      smallint,
                 tslot22      smallint,
                 tslot23      smallint,
                 tslot24      smallint,
                 tslot25      smallint,
                 tslot26      smallint,
                 tslot27      smallint,
                 tslot28      smallint,
                 tslot29      smallint,
                 tslot30      smallint,
                 tslot31      smallint,
                 tslot32      smallint,
                 tslot33      smallint,
                 tslot34      smallint,
                 tslot35      smallint,
                 tslot36      smallint,
                 tslot37      smallint,
                 tslot38      smallint,
                 tslot39      smallint,
                 tslot40      smallint,
                 tslot41      smallint
                ) ;
   SqlErr("create runs") ;
printf("Runs table created\n") ;

   $ create table results     (
                 number       smallint,
                 procnum      smallint,
                 seconds      smallint,
                 xactcnt      smallint,
                 notdone      smallint,
                 response     smallint,
                 tslot01      smallint,
                    tslot02         smallint,
                    tslot03         smallint,
                    tslot04         smallint,
                    tslot05         smallint,
                    tslot06         smallint,
                    tslot07         smallint,
                    tslot08         smallint,
                    tslot09         smallint,
                    tslot10         smallint,
                    tslot11         smallint,
                    tslot12         smallint,
                    tslot13         smallint,
                    tslot14         smallint,
                    tslot15         smallint,
                    tslot16         smallint,
                    tslot17         smallint,
                    tslot18         smallint,
                    tslot19         smallint,
                    tslot20         smallint,
                    tslot21         smallint,
                    tslot22         smallint,
                    tslot23         smallint,
                    tslot24         smallint,
                    tslot25         smallint,
                    tslot26         smallint,
                    tslot27         smallint,
                    tslot28         smallint,
                    tslot29         smallint,
                    tslot30         smallint,
                    tslot31         smallint,
                    tslot32         smallint,
                    tslot33         smallint,
                    tslot34         smallint,
                    tslot35         smallint,
                    tslot36         smallint,
                    tslot37         smallint,
                    tslot38         smallint,
                    tslot39         smallint,
                    tslot40         smallint,
                    tslot41         smallint
                   ) ;
    SqlErr("create results") ;
 printf("Results table created\n") ;

 $ close database ;
    SqlErr("close database") ;

 exit(0) ;
}
```

## A.5   createidx.ec source code

```
#include <stdio.h>
#include "bench.h"
$include sqlca ;

/*
 * FILE: createidx.ec
  *
  * Creates the indices for the main database tables. This is a separate
  * process in case loads without indices are desired.
  *
  */

main()
 {

                    $ database tpc ;
                       SqlErr("open database") ;

                    $ create unique index ibranch on branch(number) ;
                       SqlErr("create branch index") ;  printf("Branch index created\n") ;

                    $ create unique index iteller on teller(number) ;
                       SqlErr("create teller index") ;  printf("Teller index created\n") ;

                    $ create unique index iaccount on account(number) ;
                       SqlErr("create account index") ;
                     printf("Account index created\n") ;

                    $ close database ;
                       SqlErr("close database") ;

                     exit(0) ;
}
```

## A.6 config.scr code

```
echo Going into Quiescent mode
tbmode -uy
echo Creating physdbs...
tbspaces -c -d physdbs -p /dev/rrz6g -o 1000 -s 230000
echo Creating tbhdbs...
tbspaces -c -d tbhdbs -p /dev/rrz2h -o 1000 -s 150000
echo Creating acctdbs...
tbspaces -c -d acctdbs -p /dev/rrz2g -o 90000 -s 71600
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz3h -o 40000 -s 71600
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz4h -o 40000 -s 71600
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz6h -o 40000 -s 71600
echo Adding chunk to acctdbs...
tbspaces -a acctdbs -p /dev/rrz1h -o 250000 -s 71600
echo Moving Physical Log
tbparams -p -s 210000 -d physdbs -y
echo Going back On-Line tbmode -m
echo Configuration done
```

## A.7 bench.h code

```
/*
*
* PURPOSE: to set up the sizing of the TPC database
*
* the scale factors for TPC per 1 TPS are:
*              1 Branch, 10 Tellers, 100000 Accounts
*
* Modify the TPS_SIZE to the desired rating.
* DO NOT modify any but the first 4 lines.
*
 */
#define TPS_SIZE        30
#define HISTORY         1
#define RandVal         random
#define RandSeed        srandom
#define BUCKETS         40
#define RPTINTVL        5

#define T_PERB          10
#define A_PERB          100000

#define B_RECS          TPS_SIZE
#define T_RECS          (T_PERB * B_RECS)
#define A_RECS          (A_PERB * B_RECS)

#define IsqlCode        sqlca.sqlcode
#define IsamCode        sqlca.sqlerrd[1]
#define SqlErr(x)       if (IsqlCode) Sqlmsg(x)
#define SqlErrNF(x)     if (IsqlCode && IsqlCode != SQLNOTFOUND) Sqlmsg(x)
```

# Appendix B

# Database Definitions

```
#*********************************************************************************************
#
#                                  INFORMIX SOFTWARE, INC.
#
# Title:                           tbconfig.std
# Sccsid:                          @(#)tbconfig.std 7.2 11/20/90 11:06:55
#Description:                      INFORMIX-OnLine Configuration Parameters
#
#*********************************************************************************************

# Root Dbspace Configuration

ROOTNAME              rootdbs       # Root dbspace name
ROOTPATH              /dev/rra2c    # Path for device containing root dbspace
ROOTOFFSET            1000          # Offset of root dbspace into device (Kbytes)
ROOTSIZE              230000        # Size of root dbspace (Kbytes)

# Disk Mirroring Configuration Parameters

MIRROR                1             # Mirroring flag (Yes = 1, No = 0)
MIRRORPATH            /dev/rrz4g    # Path for device containing mirrored root
MIRROROFFSET          1000          # Offset into mirrored device (Kbytes)

# Physical Log Configuration

PHYSDBS               physdbs       # Location (dbspace) of physical log
PHYSFILE              210000        # Physical log file size (Kbytes)

# Logical Log Configuration

LOGFILES              5             # Number of logical log files
LOGSIZE               1000          # Logical log size (Kbytes)

# Message Files

MSGPATH               /usr/informix/online.log       # System message log file path
CONSOLE               /usr/informix/console.log      # System console message path

# System Archive Tape Device

TAPEDEV               /dev/null     # Tape device path
TAPEBLK               16            #  Tape block size (Kbytes)
TAPESIZE              90000         #  Maximum amount of data to put on tape
                                    #   (Kbytes)

# Log Archive Tape Device

LTAPEDEV              /dev/rrz0c    # Log tape device path
LTAPEBLK              16            # Log tape block size (Bytes)
```

```
LTAPESIZE           1000472    # Max amount of data to put on log tape (Kbytes)

# System Configuration

SERVERNUM           0          # Unique id corresponding to an OnLine instance
SERVERNAME          dectpc     #
DEADLOCK_TIMEOUT    30         # max time to wait of lock in distributed env.
RESIDENT            0          # Forced residency flag (Yes = 1, No = 0)

# Shared Memory Parameters

USERS               50         #  Maximum  number  of  concurrent  users  (proc
                               #   esses)
LOCKS               5000       # Maximum number of locks
BUFFERS             5000       # Maximum number of shared buffers
TBLSPACES           1200       # Maximum number of open tblspaces
CHUNKS              40         # Maximum number of chunks
DBSPACES            20         # Maximum number of dbspaces
PHYSBUFF            32         # Physical log buffer size (Kbytes)
LOGBUFF             32         # Logical log buffer size (Kbytes)
LOGSMAX             3          # Maximum number of logical log files
CLEANERS            8          # Number of buffer cleaner processes
SHMBASE             0x800000   # Shared memory base address
CKPTINTVL           720        # Check point interval (in sec)

# System Page Size

BUFFSIZE            2048       # Page size (do not change!)

# System LRU Parameters

LRUS                8          # Number of LRU’s
LRU_MAX_DIRTY       60         # Start page cleaning
LRU_MIN_DIRTY       50         # Stop page cleaning
LRU_SEARCH          70         # First Level search for free buffers
```

# Appendix C

# Code to Populate Database

This appendix contains the program used to populate the database used in the TPC Benchmark B tests.

## C.1 Database Population Program

The following program was used to populate the database:

```
#include <stdio.h>
#include <math.h>
#include <sys/types.h>
#include <sys/wait.h>
#include "bench.h"
$include sqlca ;
/*
 * FILE: load_db.ec
 *
 * PURPOSE: load the Branch and Teller tables, and kick off the Account
 *              table load procedures. The Account table is loaded by
 *              dividing the key range into equal parts (according to the
 *              number of load processes), and the "load_act" program is
 *              forked off for each process. The program then waits for
 *              them to finish and reports the total load time.
 *
 * NOTE: The type "pid_t" may be system-dependent. Under Ultrix it's
 * equivalent to an "int".
 *
 */

FILE *flog,*fopen() ;
int logfile ;

main(argc,argv)
                int argc ;
                char *argv[] ;
{
                int i, load_procs, skip, freespace ;
                char begnum[15], endnum[15], log_fname[40], rpt_str[80] ;
                long load_accts, startacct, acct_hunk, beg_time, end_time, totsecs ;
                pid_t pid ;
                union wait wait_status ;

                $int branch, teller, branch_idx ;
                $char filler[100] ;

                                RandSeed(getpid()) ;
                load_procs = 1 ;
                i = logfile = skip = freespace = branch_idx = 0 ;

                while (++i < argc) {
                        if (strcmp(argv[i], "-p") == 0)
                                load_procs = atoi(argv[++i]) ;
                        else if (strcmp(argv[i], "-s") == 0)
                                skip = atoi(argv[++i]) ;
                        else if (strcmp(argv[i], "-f") == 0)
                                freespace = atoi(argv[++i]) ;
                        else if (strcmp(argv[i], "-l") == 0) {
                                strcpy(log_fname,argv[++i]) ;
                                logfile = 1 ;
                        }
                        else {
                                printf("usage: load_db -p <#> -s <#> -
f <#> -l <file>\n") ;
                        exit(0) ;
                        }
                 }

                if (load_procs && ((A_PERB % load_procs) != 0)) {
                        printf("Cannot split up load of accounts evenly. Try
again.\n") ;
                        exit(0) ;
                }

                load_accts = (B_RECS - skip) * A_PERB ;
                        if (load_procs) acct_hunk = load_accts /
load_procs ;
                for (i=0; i < 10; i++)
                        bycopy("1234567890",&filler[i*10],10) ;
                if (logfile) {  if ((flog=fopen(log_fname,"w")) == NULL) {
                        perror("on opening log file") ;
                        logfile = 0 ;
                }
        }

        $ database tpc ;
         SqlErr("database open") ;

        if (freespace) {
                $ select count(*) into $branch_idx from sysindexes
                        where idxname = "ibranch" ;
                 SqlErr("load_db -- select branch index") ;
                if (branch_idx) {
                        $ drop index ibranch ;
                         SqlErr("load_db -- delete branch index") ;
                }
        }

        for (branch=skip; branch < B_RECS; branch++) {
                $ insert into branch values($branch, 0, $filler) ;
```

```
                    SqlErr("load_db -- branch insert") ;
                 for (i=0; i < freespace; i++) {
                                  $ insert into branch values(0, -1, $filler) ;
                                     SqlErr("load_db -- branch free insert") ;
                 }
}
if (freespace) {
                 $ delete from branch where balance < 0 ;
                    SqlErr("load_db -- delete branch records") ;
 }
if (branch_idx) {
                 $ create unique index ibranch on branch(number) ;
                    SqlErr("load_db -- create branch index") ;
}
 print_log("branch table loaded") ;
for (teller=T_PERB*skip; teller < T_RECS; teller++) {
                 branch = teller / T_PERB ;
                 $ insert into teller values($teller, 0, $branch, $filler) ;
                    SqlErr("load_db -- insert into teller") ;
 }
 print_log("teller table loaded") ;
$ update statistics for table branch ;
  SqlErr("load_db -- update stats on branch") ;
$ update statistics for table teller ;
  SqlErr("load_db -- update stats on teller") ;
$ close database ;
  SqlErr("load_db -- close database") ;
sqlexit() ;
if (load_procs) {
                 beg_time = time(0) ;
                 startacct = skip * A_PERB ;
                 for (i=0; i < load_procs; i++) {
                                  sprintf(begnum,"%d",startacct) ;
                                  startacct += acct_hunk ;
                                  sprintf(endnum,"%d",startacct-1) ;
                                  pid = fork() ;
                                   if (pid == -1) {
                                                   perror("on fork of loadact process") ;
                                                   exit(1) ;
                 }
                 if (pid == 0)
                 if (logfile)
                  execl("load_act","load_act",begnum,endnum,"1",log_fname,0) ;
                 else
                  execl("load_act","load_act",begnum,endnum,"0"," ",0) ;
```

```
                                        }
                                        while (i--) {
                                                    pid = wait(&wait_status) ;
                                                    if (pid == -1) {
                                                                perror("on return from loadact") ;
                                                                exit(1) ;
                                                    }
                                                    end_time = time(0) ;
                                                    totsecs = end_time - beg_time ;
                                                    sprintf(rpt_str,"\nprocess %d completed in ",pid) ;
                                                    report_time(rpt_str,totsecs) ;
                                                    if (i > 0)
                                                                sprintf(rpt_str,"%s;   %d   procs   still
working",rpt_str,i) ;
                                                     print_log(rpt_str) ;
                                        }
                                        sprintf(rpt_str,"\nAll          processes          finished          at
%s",ctime(&end_time)) ;
                                        print_log(rpt_str) ;
                                        sprintf(rpt_str,"loaded %d account records in ",load_accts) ;
                                        report_time(rpt_str,totsecs) ;
                                        sprintf(rpt_str,"%s = %d rows/sec\n",rpt_str,load_accts/totsecs)
;
                                        print_log(rpt_str) ;
                                        $ database tpc ;
                                          SqlErr("Open Database") ;
                                        $ update statistics for table account ;
                                          SqlErr("Update Statistics on account") ;
                                        $ close database ;
                                          SqlErr("Close Database") ;
                    }
                    if (logfile)    fclose(flog) ;
                    exit(0) ;
}
report_time(s,secs)
                    char s[] ;
                    long secs ;
{
                    int hrs, mins, slen ;
                    hrs = secs / 3600 ;
                    secs = secs % 3600 ;
                    mins = secs / 60 ;
                    secs = secs % 60 ;
                    slen = strlen(s) ;
                    sprintf(&s[slen],"%2d:%02d:%02d",hrs,mins,secs) ;
}
```

```
print_log(s)
        char *s ;
{
        if (logfile) {
                fprintf(flog,"%s\n",s) ;
                fflush(flog) ;
 }
else {
                printf("%s\n",s) ;
                fflush(stdout) ;
        }
}
```

# Appendix D

# Database Contents Samples

This appendix contains the database contents samples for the TPC Benchmark B run on the DECsystem 5100.

## D.1 Branch Table

Following is a sample of the Branch File contents:

| | |
|---|---|
| number | 14 |
| balance | 0 |
| fillstr | 12345678901234567890123456789012345678901234567890123456789012345678 9012345678901234567890123456789012 |
| number | 15 |
| balance | 0 |
| fillstr | 12345678901234567890123456789012345678901234567890123456789012345678 9012345678901234567890123456789012 |
| number | 16 |
| balance | 0 |
| fillstr | 12345678901234567890123456789012345678901234567890123456789012345678 9012345678901234567890123456789012 |

## D.2 Teller Table

Following is a sample of the Teller table contents:

| | |
|---|---|
| number | 28 |
| balance | 0 |
| branch | 2 |
| fillstr | 12345678901234567890123456789012345678901234567890123456789012345678 9012345678901234567890123456789 |
| number | 29 |
| balance | 0 |
| branch | 2 |
| fillstr | 12345678901234567890123456789012345678901234567890123456789012345678 9012345678901234567890123456789 |
| number | 30 |
| balance | 0 |
| branch | 3 |
| fillstr | 12345678901234567890123456789012345678901234567890123456789012345678 9012345678901234567890123456789 |

## D.3 History Table

Following is a sample of the History table contents:

| | |
|---|---|
| account | 2564244 |
| teller | 255 |
| branch | 25 |
| delta | 808156 |
| tstamp | 1991-09-17 11:37:37 |
| fillstr | the rest is history |

| | |
|---|---|
| account | 1222608 |
| teller | 121 |
| branch | 12 |
| delta | 750865 |
| tstamp | 1991-09-17 11:37:37 |
| fillstr | the rest is history |

| | |
|---|---|
| account | 2732829 |
| teller | 275 |
| branch | 27 |
| delta | 580118 |
| tstamp | 1991-09-17 11:37:37 |
| fillstr | the rest is history |

## D.4 Account Table

Following is a sample of the Account table contents:

| | |
|---|---|
| number | 37 |
| balance | 0 |
| branch | 0 |
| fillstr | 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567 |

| | |
|---|---|
| number | 38 |
| balance | 0 |
| branch | 0 |
| fillstr | 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567 |

| | |
|---|---|
| number | 39 |
| balance | 0 |
| branch | 0 |
| fillstr | 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567 |

# Appendix E

# Device Configurations

This appendix contains a description of the physical disk configurations tested for the DECsystem 5100 configuration.

```
/dev/rrz0c
No partition table found in superblock... using default table from device driver.
Current partition table:
partition     bottom      top         size        overlap
 a            0           32767       32768       c
 b            32768       217087      184320      c
 c            0           1954049     1954050     a,b,d,e,f,g,h
 d            831488      1130495     299008      c,h
 e            1130496     1429503     299008      c,h
 f            1429504     1954049     524546      c,h
 g            217088      831487      614400      c
 h            831488      1954049     1122562     c,d,e,f
```

```
/dev/rrz1c
Current partition table:
partition     bottom      top         size        overlap
 a            0           32767       32768       c
 b            32768       217087      184320      c
 c            0           1954049     1954050     a,b,d,e,f,g,h
 d            831488      1130495     299008      c,h
 e            1130496     1429503     299008      c,h
 f            1429504     1954049     524546      c,h
 g            217088      831487      614400      c
 h            831488      1954049     1122562     c,d,e,f
```

```
/dev/rrz2c
Current partition table:
partition     bottom      top         size         overlap
 a            0           32767       32768       c,g
 b            32768       163839      131072      c,g
 c            0           832526      832527      a,b,d,e,f,g,h
 d            163840      386735      222896      c,g
 e            386736      609631      222896      c,g,h
 f            609632      832526      222895      c,h
 g            0           499999      500000      a,b,c,d,e
 h            500000      832526      332527      c,e,f
```

```
/dev/rrz3c
Current partition table:
partition     bottom        top           size           overlap
 a            0             32767         32768          c,g
 b            32768         163839        131072         c,g
 c            0             832526        832527         a,b,d,e,f,g,h
 d            163840        386735        222896         c,g
 e            386736        609631        222896         c,g,h
 f            609632        832526        222895         c,h
 g            0             499999        500000         a,b,c,d,e
 h            500000        832526        332527         c,e,f

/dev/rrz4c
Current partition table:
partition     bottom        top           size           overlap
 a            0             32767         32768          c,g
 b            32768         163839        131072         c,g
 c            0             832526        832527         a,b,d,e,f,g,h
 d            163840        386735        222896         c,g
 e            386736        609631        222896         c,g,h
 f            609632        832526        222895         c,h
 g            0             499999        500000         a,b,c,d,e
 h            500000        832526        332527         c,e,f

/dev/rrz6c
Current partition table:
partition     bottom        top           size          overlap
 a            0             32767         32768         c,g
 b            32768         163839        131072        c,g
 c            0             832526        832527        a,b,d,e,f,g,h
 d            163840        386735        222896        c,g
 e            386736        609631        222896        c,g,h
 f            609632        832526        222895        c,h
 g            0             499999        500000        a,b,c,d,e
 h            500000        832526        332527        c,e,f
```

# Appendix F

# System Parameter Settings

This appendix contains the operating system parameters and database options in the TPC Benchmark B test system.

## F.1 System Parameters

ULTRIX version 4.2 system parameters were configured as shown below. In all instances default values were used except for

- MAXUSERS was set to 128
- MAXUPRC was set to 128
- SMMAX was set to 1024
- SMSEG was set to 120

Additionally, two semaphore constant values were changed in the ULTRIX IPC Semaphore Facility sem.h (/usr/sys/h/sem.h). The value SEMMNI, the number of semaphore identifiers, was set to 40, and the SEMMNS, the number of semaphores in the system, was set to 120. A copy of sem.h appears in this appendix.

```
ident                   "PENNYS"
machine                 mips
cpu                     "DS5100"
maxusers                128
processors              1
maxuprc                 128
physmem                 32
timezone                5 dst 1
smmax                   1024
smseg                   120

options                 QUOTA
options                 INET
options                 NFS
options                 RPC
options                 DLI
options                 UFS
options                 NETMAN
options                 LAT
options                 CDFS
options                 PACKETFILTER
options                 AUDIT
options                 SYS_TPATH
options                 DECNET

makeoptions             ENDIAN="-EL"
```

```
config                  vmunix      root on rz1a swap on rz1b dumps on rz1b

adapter                 ibus0       at nexus?
controller              sii0        at ibus?      vector sii_intr
 disk         rz0       at sii0 drive 0
disk          rz1       at sii0 drive 1
disk          rz2       at sii0 drive 2
disk          rz3       at sii0 drive 3
disk          rz4       at sii0 drive 4
disk          rz5       at sii0 drive 5
disk          rz6       at sii0 drive 6
disk          rz7       at sii0 drive 7
tape          tz0       at sii0 drive 0
tape          tz1       at sii0 drive 1
tape          tz2       at sii0 drive 2
tape          tz3       at sii0 drive 3
tape          tz4       at sii0 drive 4
tape          tz5       at sii0 drive 5
tape          tz6       at sii0 drive 6
tape          tz7       at sii0 drive 7
device        ln0       at ibus?    vector lnintr
device        mdc0      at ibus?    vector mdcintr

scs_sysid      1

pseudo-device           nfs
pseudo-device           rpc
pseudo-device           dli
pseudo-device           pty
pseudo-device           loop
pseudo-device           ether
pseudo-device           ufs
pseudo-device           netman
pseudo-device           inet
pseudo-device           lat
pseudo-device           lta
pseudo-device           scsnet
pseudo-device           msdup
pseudo-device           cdfs
pseudo-device           packetfilter
pseudo-device           sys_tpath
pseudo-device           audit
pseudo-device           decnet
```

## F.2   IPC Semaphore Facility

```
/* @(#)sem.h                4.1 (ULTRIX)              7/2/90 */
/************************************************************************
 *                                                                     *
 *              Copyright (c) 1986, 1988 by                            *
 *              Digital Equipment Corporation, Maynard, MA             *
 *              All rights reserved.                                   *
 *                                                                     *
 * This software is furnished under a license and may be used and      *
 * copied only in accordance with the terms of such license and        *
 * with the inclusion of the above copyright notice. This              *
 * software or any other copies thereof may not be provided or         *
 * otherwise made available to any other person. No title to and       *
 * ownership of the software is hereby transferred.                    *
 *                                                                     *
 * This software is derived from software received from the            *
 * University of California, Berkeley, and from Bell                   *
 * Laboratories. Use, duplication, or disclosure is subject to         *
 * restrictions under license agreements with University of            *
 * California and with AT&T.                                           *
 *                                                                     *
 * The information in this software is subject to change without       *
 * notice and should not be construed as a commitment by Digital       *
 * Equipment Corporation.                                              *
 *                                                                     *
 * Digital assumes no responsibility for the use or reliability        *
 * of its software on equipment which is not supplied by Digital.      *
 *                                                                     *
 ************************************************************************/
/*  *  * Modification history:
 *
 * 19 Mar 90 -- burns
 *              Added ifdef kernel around SMP lock imbedded in
 *              a user visable data structure (msqid_ds).
 *
 * 13 Dec 89 -- scott
 *              xpg compliance changes
 *
 * 16 Aug 88 -- miche
 *              Add support for SMP
 *
 * 02 Apr 86 -- depp
 *              Moved sizing constants from /sys/h/param.h to here.
 *
 * 01 Mar 85 -- depp  *     New file derived from System V IPC
 *
 */
```

```
/*
 **            IPC Semaphore Facility.
*/

#ifndef KERNEL
#include <sys/smp_lock.h>
extern int semctl();
extern int semget();
extern int semop();
#endif /* KERNEL */

#if !defined(_POSIX_SOURCE)
/*
**             Implementation Constants.
*/

#define        PSEMN        (PZERO + 3)        /* sleep priority waiting for greater value */
#define        PSEMZ        (PZERO + 2)        /* sleep priority waiting for zero */

/*
**             Permission Definitions.
*/

#define        SEM_A        0200        /* alter permission */
#define        SEM_R        0400        /* read permission */

#endif /* !defined(_POSIX_SOURCE) */
/*
**             Semaphore Operation Flags.
*/

#define        SEM_UNDO010000        /* set up adjust on exit entry */

/*
**             Semctl Command Definitions.
*/

#define        GETNCNT   3           /* get semncnt */
#define        GETPID    4           /* get sempid */
#define        GETVAL    5           /* get semval */
#define        GETALL    6           /* get all semval's */
#define        GETZCNT   7           /* get semzcnt */
#define        SETVAL    8           /* set semval */
#define        SETALL    9           /* set all semval's */

/*
**             Structure Definitions.
*/

/*
**             There is one semaphore id data structure for each set of semaphores
**                          in the system. The ipc_perm structure must be first and
**                          the lock must be last.
*/
```

```
struct semid_ds {
            struct ipc_perm             sem_perm;   /* operation permission struct */
            struct sem     *sem_base;  /* ptr to first semaphore in set */
            unsigned short              sem_nsems;/* # of semaphores in set */
            time_t                      sem_otime;  /* last semop time */
            time_t                      sem_ctime;  /* last change time */
#ifdef KERNEL
            struct __lock_t sem_lk;     /* SMP lock for the semaphore queue */
#endif /* KERNEL */ };

/*
**          There is one semaphore structure for each semaphore in the system.
*/

struct sem {
            unsigned short semval;      /* semaphore text map address */
            pid_t sempid;               /* pid of last operation */
            unsigned short semncnt;     /* # awaiting semval > cval */
            unsigned short semzcnt;     /* # awaiting semval = 0 */
            unsigned short semnwakup;/* wake up those waiting on semncnt */
};

#if !defined(_POSIX_SOURCE)

/*
**          There is one undo structure per process in the system.
*/

struct sem_undo {
            struct sem_undo             *un_np;     /* ptr to next active undo structure */
            short                       un_cnt;     /* # of active entries */
            struct undo {
                        short           un_aoe;     /* adjust on exit values */
                        short           un_num;     /* semaphore # */
                        int             un_id;      /* semid */
            }           un_ent[1];      /* undo entries (one minimum) */
};

/*
** semaphore information structure
*/
struct      seminfo     {
            int         semmap,     /* # of entries in semaphore map */
                        semmni,     /* # of semaphore identifiers */
                        semmns,     /* # of semaphores in system */
                        semmnu,     /* # of undo structures in system */
                        semmsl,     /* max # of semaphores per id */
                        semopm,     /* max # of operations per semop call */
                        semume,     /* max # of undo entries per process */
                        semusz,     /* size in bytes of undo structure */
                        semvmx,     /* semaphore maximum value */
                        semaem;     /* adjust on exit max value */
};
```

```
/*
**          User semaphore template for semop system calls.
*/

struct sembuf {
          unsigned short sem_num;            /* semaphore # */
          short       sem_op;                /* semaphore operation */
          short       sem_flg;               /* operation flags */
};

/*
 * Sizing constants
 */

#define SEMMAP 10
#define SEMMNI 40
#define SEMMNS 120
#define SEMMNU 30
#define SEMMSL 25
#define SEMOPM 10
#define SEMUME 10
#define SEMVMX 32767
#define SEMAEM 16384

#endif /* !defined(_POSIX_SOURCE) */
```

# Appendix G

# Independent Auditor’s Report