IOBENCH: A System Independent IO Benchmark[1]

Barry L. Wolman
Thomas M. Olson

Prime Computer, Inc.
500 Old Connecticut Path
Framingham, MA 01701

Abstract

IOBENCH™ is an operating system and processor independent synthetic input/output (IO) benchmark designed to put a configurable IO and processor (CP) load on the system under test. It is meant to stress the system under test in a manner consistent with the way in which Oracle™, Ingres™, Prime INFORMATION™ or other data management products do IO. The IO and CP load is generated by background processes doing as many "transactions" as they can on a specified set of files during a specified time interval. By appropriately choosing and varying the benchmark parameters, IOBENCH can be configured to approximate the IO access patterns of real applications. IOBENCH can be used to compare different hardware platforms, different implementations of the operating system, different disk buffering mechanisms, and so forth. IOBENCH has proven to be a very good indicator of system IO performance. Use of IOBENCH has enabled us to pinpoint operating system bugs and bottlenecks.

IOBENCH currently runs on PRIMOS™ and a number of UNIX™ systems; this paper discusses the UNIX versions. IOBENCH can be ported to a new platform in a few days. Prime proposes that IOBENCH and a standard spectrum of runs be adopted as an industry standard for measuring IO performance. Sources and documentation for IOBENCH will be made available free of charge.

Background

The performance of a computer system is often characterized by citing its CP performance, e.g., 10 mips. There is a hidden assumption that two systems with comparable CP ratings will perform comparably. Often, this is not the case.

While the performance of the arithmetic processor is often a significant factor, for many applications, the overall performance of the system is determined by the performance of the memory and IO subsystems. The performance of the IO subsystem has not kept pace with the rapid growth in performance of arithmetic processors and the development of high speed memory and caching mechanisms. High end systems used to be CP limited; today they are often IO limited.

There are a large number of benchmarks that test the performance of the arithmetic processor. These range from purely synthetic, relatively small benchmarks, such as Whetstone and Dhrystone, to larger benchmarks derived from real problem spaces, such as Linpack or Livermore Loops, to real applications such as Spice [1]. In many cases,

[1]Submitted to Computer Architecture News, August 1989. Send comments or questions to barry@s66.Prime.COM or olson@s66.Prime.COM.

however, such benchmarks are a better test of the quality of the compiler or of system cache size than of the fundamental performance of the processor.

There are relatively few benchmarks that test the IO subsystem. This may be due to the fact that IO performance can't be characterized by a single number. IO performance depends on a large number of factors, including number of controllers, number of drives, number of users, number and size of files, access patterns, etc. Single user benchmarks such as IOSTONE [2] provide only a single sample. IO performance must be characterized by a spectrum of runs, not by a single data point.

The TP1 debit/credit benchmark being standardized by the Transaction Processing Performance Council [3] (TPPC Benchmarks A and B) provides a controlled benchmark that approximates a real problem of commercial interest. However, TP1 requires significant resources such as terminal emulator system to drive the system under test. While useful, TP1 is too cumbersome to be used in the lab.

Over the years, a number of IO intensive benchmarks had been written at Prime. However, these made extensive use of PRIMOS features not available in other operating systems, and so could not be ported easily. In addition, these benchmarks had fixed configurations and were hard to modify.

Since we couldn't find any appropriate IO benchmarks in the public domain, we decided to write our own and make it available to others. The primary requirements for IOBENCH were:

- We wanted to be able to simulate a number of different data management applications. We did not want to depend on having the same data management subsystem available on all operating systems.
- We wanted to be able to vary the benchmark parameters, e.g. from 1 to 1000+ users.
- We wanted one configurable benchmark, rather than a family of different benchmarks using common components.
- We wanted a benchmark that would be easy to install and use.
- We wanted a benchmark that could be ported easily to new hardware and to new operating systems that supported a small set of key features.

Benchmark Characteristics

IOBENCH operates by applying a repeatable, configurable work load to a set of files. Both the IO and CP components of the work load can be specified.

IO Load

All IO is done on a set of **F** files. Each file consists of **N** fixed length records **B** bytes long. **N** and **B** should be chosen so that the total size of all files (**N** * **B** * **F**) is substantially larger than the available disk cache. The IO load comes from a set of **U** users (background processes) doing as many IO intensive transactions (not to be confused with TP1 transactions) as they can in a specified time interval subject to constraints imposed by the benchmark configuration.

Each file has **L** locks associated with it; the **N** records in the file are mapped on to the **L** locks. By varying the number of locks the effect of file locking (L=1) or group locking (L=very large) can be simulated. Locks are kept in shared memory and are managed by a lock manager that is part of the benchmark. A sleep parameter **S** determines whether process failing to get a lock spins (S=0) or sleeps for S seconds; **S** = -1 means no locking is done.

Each IO transaction consists of **R** reads and **W** writes, where **W** must be less than or equal to **R**. **R-W** reads are done followed by **W** alternating reads and writes. **R** and **W** should be chosen to model a relevant work load. For example, **R** = **W** = 4 corresponds to the debit/credit benchmark. **R** = 10 and **W** = 2 might correspond to an application that does a query followed by an update.

CP Load

There is a "work" parameter **CP** that can be used to configure the benchmark as pure IO (**CP** = 0), CP limited (**CP** = "large" value), or anywhere in between. **CP** units of work consists of a null loop of CP*1000 iterations. The processor time needed to implement **CP** units of work will depend on the processor on which IOBENCH is run. **CP/R** units of work are performed for each read in the transaction.

Benchmark Parameters

The parameters that control IOBENCH can be specified by default, from the UNIX environment, or via an optional control file. Values specified in the control file override those specified in the environment; values specified in the environment override the default values. The control file need not be provided if all benchmark parameters have values established by defaults or in the environment.

Each parameter has an associated keyword, which is used in both the environment and the control file. The keywords, their meaning, and the default values are shown in the following table:

Keyword	**Meaning**	**Default**
RUN	identification of run	0
USERS	U, number of users	4
FILES	F, number of files	4
NRECS	N, number of records in files	1024
RECLEN	B, number of bytes in each record of files	2048
READS	R, number of reads per transaction	1
WRITES	W, number of writes per transaction	1
LOCKS	L, number of locks per file	1000
SLEEP	S, lock sleep time (seconds)	0
START	DS, start delay (minutes)	1
END	DE, end delay (minutes)	1
DURATION	M, run duration (minutes)	1
WORK	CP, work parameter	0
FILE0	pathname of file number 0	(none)
FILE1	pathname of file number 1	(none)
...		

The control file, which may be empty, consists of free form keyword assignments of the form

```
FILES = 16; RECLEN = 4096; USERS = 100;
```

Each assignment is terminated by a semi-colon. "White-space" is generally ignored; however, spaces cannot occur in the middle of a keyword name or value.

If the first character in a keyword is "@", the remainder of the input line is interpreted as the pathname of a secondary input file. The file currently being read is closed and subsequent input is taken from the new file. This may be convenient if multiple runs are being done where most of the parameters are the same, e.g. file pathnames. The common parameters can be put in the secondary control file and the parameters that change for each run can be put in the primary file. For example,

```
RECLEN = 2048; USERS = 200;
@FILES
```

Benchmark Components

There are three major components in IOBENCH:

- **iomain** is the control process. It collects the parameters that define the run to be made (see below); spawns **ioserver**, sets up for the run; spawns and synchronizes **U** instances of **iouser** as background processes to do the actual benchmark; and collects results after the background processes have finished.

- **ioserver** is a server process that optionally can be used to perform operations such as locking or file access.

- **iouser** is the background process, **U** of these are created by **iomain**. Each instance of **iouser** loops for **M** minutes doing the following operations:

    - **R-W** times: randomly pick a file, randomly pick a record in that file, lock the record, read the record, optionally burn CP time, and unlock the record.

    - **W** times: randomly pick a file, randomly pick a record in that file, lock the record, read the record, optionally burn CP time, write the record, and unlock the record.

  After running for the specified interval, each background process writes to the results file the number of cycles it was able to complete and the CP and IO times. Each instance of **iouser** also accumulates statistics on times taken to read a record.

Running IOBENCH

A run of IOBENCH is initiated by

```
iomain [options] [control]
```

A variety of options are provided to control creation and naming of output files, to cause system usage data to be collected, to cause a transaction histogram to be printed, and to request that specific operations be performed in **ioserver**.

A series of IOBENCH runs can be initiated by either preparing a set of control files or by using a Shell file that appropriately varies the run parameters. IOBENCH is fairly robust. **iomain** is able to recover from most errors and terminate cleanly. Thus, a group of IOBENCH runs can be left to run over night.

Benchmark Output

IOBENCH generates a results file, a summary of results on the terminal, and, if the **usage** flag is set, generates one or more files with system usage statistics.

Results File

Figure 1 shows an example of the results file generated when IOBENCH was run on a Prime EXL 316 system with options that requested locking in **ioserver**. The starting time of the run and the version of IOBENCH that was used is shown at the top of the output file. The next group of lines (through "`CP load/cycle`") provide a summary of the parameters defining the specific run. The rest of the output file contains the results of the run:

- The lock summary shows the number of locks taken for each file and the maximum that were active simultaneously. The number of locks taken for each file should be uniformly distributed. A non-uniform distribution indicates a problem in the system under test.

- The user summary shows for each user the number of transactions (cycles) performed by that user, the CP time consumed, the IO time consumed (this will be 0 in UNIX), the number of times files were opened or closed *for each user* because there were insufficient file units available, the time the user process started, and the number of seconds the user process ran. It is unreasonable to expect all users to complete the same number of cycles; however, the number of cycles and the CP and IO times should be evenly distributed. A non-uniform distribution indicates a problem in the system under test.

- The IO summary table shows for each file: the number of times the file was opened or closed because there were insufficient file units available, the number of uses of the file (should match the number of locks taken for the file), the average response time for a read request, the standard deviation of the read time, the maximum observed read time, and the record and user for which the maximum time was observed. All times are shown in seconds and milliseconds.

- The per user figures are aggregated into a total for cycles, megabytes of data processed, CP seconds, and IO seconds. The CP and IO time consumed by **ioserver** is also reported.

- The transaction statistics show the average transactions per second and average response time. These figures are produced by averaging over the number of users the average transactions per second and average response time of each user.

```
Started test-v1 at Fri Jun  2 13:47:08 1989
IOBENCH version 5.0 of 25 May 1989
Summary will be written to file summary
Run id is 0
Variant is 1
Number of files = 3
Number of records = 10000
Record length = 1024
Number of users = 8
Locks per file = 10000
Lock sleep time = 0
Start delay = 0
Number of minutes = 5
End delay = 0
Work parameter = 0
Reads = 1
Writes = 1
path[0] = FILE0
path[1] = /d21/FILE1
path[2] = /d31/FILE3
CP load/cycle = 0.000 secs

 FILE     LOCKS TAKEN     MAX ACTIVE
    0            5202              8
    1            5181              8
    2            5243              8

 USER CYCLES   CP TIME    IO TIME    OPENS   CLOSES   ----START TIME----    DURATION
    1   1973    22.800      0.000        0        0   Fri Jun  2 13:47:09        301
    2   1969    22.020      0.000        0        0   Fri Jun  2 13:47:09        301
    3   1919    20.870      0.000        0        0   Fri Jun  2 13:47:09        304
    4   1972    22.300      0.000        0        0   Fri Jun  2 13:47:09        303
    5   1933    21.270      0.000        0        0   Fri Jun  2 13:47:09        303
    6   1985    21.740      0.000        0        0   Fri Jun  2 13:47:09        304
    7   1972    22.210      0.000        0        0   Fri Jun  2 13:47:09        304
    8   1903    21.630      0.000        0        0   Fri Jun  2 13:47:09        301

FILE    OPENS  CLOSES     USES  AVG TIME    STD DEV     MAX T  RECORD  USER
   0        0       0     5202     0.141      0.528     9.000    6519     4
   1        0       0     5181     0.087      0.350     9.000     220     8
   2        0       0     5243     0.104      0.413     8.000    3857     4

*** Total cycles = 15626
*** Total MB of data = 32.002

*** User   CP seconds = 174.840
*** User   IO seconds = 0.000

*** Server CP seconds = 41.340
*** Server IO seconds = 0.000

*** Total  CP seconds = 216.180
*** Total  IO seconds = 0.000
```

Figure 1: IOBENCH Results File

- The per user figures show the average transaction response time observed for each user as well as the standard deviation and the maximum number.

- A response time distribution table provided if the "`-t`" option was used shows a histogram of the transaction response times.

- A line prefixed "===" is a summary of the run; it contains the number of files, the number of users, the record size, the work parameter, the total number of cycles completed, the total megabytes of data, the total CP time, the total IO time, and the number of user processes that did not complete normally. Use of the "===" prefix makes it easy to find this line with `grep`.

Summary File

The **summary** option causes **iomain** to append a single line to the specified summary file. This line consists of the following fields separated by horizontal tab characters:

- input values: number of files, number of users, record length, number of records, number of reads in transaction, number of writes in transaction, and work parameter.

- output values: megabytes processed, CP time, IO time, total average response time, transactions per second, and number of failed users.

It should be possible to use the summary file as input to a spreadsheet or any other analysis program that supports a tab-delimited input format.

Usage File

In addition to generating the primary results file shown above, IOBENCH can also generate one or more additional output files with detailed system metering data. If the "`-u`" option was used, **iomain** spawns an additional background process to run the system metering command(s) contained in the shell file `usage`. The specific metering commands used will depend on which variant of UNIX is being used.

Benchmark Results

Over the past year, IOBENCH has been used at Prime to analyze the performance of different multiprocessor UNIX systems, to evaluate the performance impact of a series of changes to the disk IO subsystem in UNIX, to evaluate the effect of disk cache size on system throughput, to evaluate performance of RAM disks, to assess the performance of disk array prototypes, and to study the effect of using a central server in a loosely coupled multiprocessor with limited shared memory. The latter two investigations are described in companion papers in this issue. This section reviews the use of IOBENCH in the multiprocessor comparisons and UNIX disk IO changes.

Multiprocessor Comparisons

Figure 2 shows the profile of a series of runs performed on two different multiprocessor systems. The top graph in Figure 2 shows how the number of files varied from run to run, the middle graph shows how the number of users varied, and the bottom graph shows how the processor work load varied. Each of the 32 points on the **Run** axis represents a specific combination of files, users, and work load. There is no particular significance to these values other than the fact that they were chosen to stress the systems in a number of dimensions. It is easier to see the pattern if the run parameters are shown as a graph rather than as a bar chart.

Figures 3-5 show the results from the runs profiled in Figure 2. The top graph in each figure shows the total MB of data processed for each run configuration. The bottom graph shows the percentage of CP transactions (cycles) completed in the run.

Although they used completely different processors, the two systems had comparable configurations. Both systems supplied 24 aggregate mips and were configured with 32MB of memory, 10 MB of disk cache, eight disk controllers, and two disks per controller. Since the identity of the two systems is not important, we've called them A and B.

Because both systems had the same aggregate computing power, a naive evaluation based solely on CP performance would predict that A and B provide comparable throughput. Experienced engineers would be suspicious of such a comparison The results confirm these suspicions.

A total of 96 runs were performed on 20MB files, 32 each with record lengths of 2048 (shown in Figure 3), 4096 (shown in Figure 4), and 8192 (Figure 5). Runs 1-8 in these figures correspond to **CP** = 0; runs 9-16 correspond to **CP** = 50, a light work load; runs 17-24 correspond to **CP** = 100, a moderate work load; and runs 25-32 correspond to **CP** = 200, a heavy work load. The runs with 0 work are all IO limited. For each of these work loads, the following eight combinations of (files, users) were used: (4,50), (4,100), (8,50), (8,100), (8,150), (8,200), (16,50), and (16,100).

Consider the results shown in Figure 3 (record length of 2048):

- The two system provide nearly similar throughput. B is significantly better in the 16 file cases (runs 7-8 and 15-16) when the CP work load is light.

- The unit cost of doing IO is higher on A when 8 or 16 files are used. Runs 3-7, which have no CP load associated with them, consume almost 20+% to 40+% of the CP cycles on A. The unit cost of IO is quite low on B, which never gets more than 20% busy.

- Going from 16 files and 50 users (run 7) to 16 files and 100 users (run 8) caused a big drop in throughput on A while throughput increased on B. Since CP utilization on A dropped from 44% to 27%, there is a bottleneck, e.g. locking in the operating system, that throttles performance.

- A becomes CP limited in runs 17-32 for nearly all combinations of files and users. However, B remains IO limited in all of the 4 file and most of the 8 file cases.

- There's something wrong with time accounting on A, which reports more CP time used than is available for runs 6, 14, 22, and 30! Since these were the 200 user runs, this may be due to accumulated errors in times reported by A's operating system to each of the **iouser** processes.

Now consider the results shown in Figure 4 (notice that scale on top graph is different):

- Both systems provided increased throughput using 4096 byte records. However, A improved only slightly while B did almost twice as much work.
- CP utilization on A decreased in almost every run, while CP utilization on B increased slightly in runs 1-16 and decreased slightly in runs 17-32. Performance of A appears to be controller limited.

Now consider the results shown in Figure 5 (notice that scale on top graph is different):

- Throughput of B has doubled again, while throughput of A has stayed about the same. B provided about three times the throughput of A for 8 file cases and four times the throughput for 16 file cases.
- CP utilization of A decreased significantly, particularly in 16 file runs. CP utilization of B increased slightly in runs 1-16 (CP $\leq$ 50) and increased significantly in runs 17-32 (CP $\geq$ 100). A is IO limited except for runs 29-30, which correspond to 8 files with 150 and 200 users. A definitely appears to be controller limited.

The overall conclusions that we can draw from these runs are:

- Throughput on system A improved only marginally as record length increased from 2048 to 8192 bytes, while throughput on system B increased by nearly a factor of four. There is no advantage to be gained from using a large block size on A.
- The IO code path length is longer on A than on B, i.e. the IO operating system code on B is more efficient.
- There appear to be no major bottlenecks in the IO subsystem of B. There seems to be an operating system bottleneck on A that affects runs with 200 users and a controller bottleneck that limits throughput of runs with large record lengths.

When the same series of runs was done on a third multiprocessor, C, we were surprised to see that CP utilization was very high, even in runs 1-8 where there is no CP load. For example, in run 2 (4 files, 100 users), CP utilization was 56% as compared to 16% used by A and 4% used by B. CP utilization on C grew as number of users increased, reaching 80% for runs where B was only 20% busy. An investigation revealed that there was a bug in the operating system on C that became a factor when a large number of users tried to access the same file. When user accessing the file released the internal lock maintained by UNIX, **all** users trying to use the file were rescheduled; one user got the lock, and the remaining users became blocked again. When this bug was fixed, CP utilization on C became comparable to that of B. This experience shows the value of the comparative tests that IOBENCH makes possible. Without the kind of methodical test that can be done with IOBENCH, this bug would not have been found.

Disk IO Improvements

IOBENCH has been used extensively within Prime to measure the impact of system changes. By doing runs before and after the changes, the impact of the changes are easily identified. Because IOBENCH is used with a spectrum of runs, the performance improvements identified by using it tend to be quite conservative. For example, one system change caused a 300% improvement in the results for IOSTONE but only a 20% increase in results from IOBENCH.

Further Work

In the year since it was first written, IOBENCH has proven to be very useful in assessing system performance and in pinpointing areas in the operating system or hardware environment that limit performance. Nevertheless, there are a number of areas where IOBENCH can be improved:

- We plan to provide tools to post process IOBENCH results, e.g. EXCEL™ or 123™ spreadsheets. Currently, too much manual processing of run data is required.

- We plan to make the definition of a transaction more flexible, e.g. let user define the sequence of reads, writes, and re-writes.

- We plan to improve the level of statistical analysis, which is currently rudimentary. Thus far, we've used five minute run durations and have depended on the length of the run to swamp statistical variations.

In addition, we plan to extend **ioserver** to include additional options such as a variant to do all IO in **ioserver**, which would let us model server-based data management systems.

Porting IOBENCH

IOBENCH is available on PRIMOS, the operating system for Prime's 50 series super-minicomputers, where it was first implemented, and on a variety of UNIX systems. IOBENCH consists of approximately 2800 lines of system independent code, 250 lines of UNIX specific code, and 370 lines of PRIMOS specific code.

Porting IOBENCH to a new platform may require changes for a new processor or for a new operating system. IOBENCH has been written to make both of these changes as easy as possible. Experience has shown that moving to a new version of UNIX on an already supported processor requires only a few hours; moving to UNIX on a new processor requires at most a few days work. Porting to an entirely new operating system might require a few weeks work.

Nearly all of IOBENCH is written in C. The only processor specific code occurs in a module that implements low level locking primitives in assembly language. These can be done either as a ".s" file that defines routines called from high level locking routines written in C or as a header file that defines assembly language macros to be inserted inline. In either case, only about one page of code is required.

Most of IOBENCH is independent of the host operating system. Usually, where operating system features are used, UNIX routines are called and, if necessary, a set of interface routines maps UNIX calls to equivalent routines in the host operating system. IOBENCH requires the standard UNIX IO and process creation mechanisms. System V shared memory calls are used to access shared memory. If the optional server mechanism is used, System V message queues (or a reasonable facsimile) must be present.

IOBENCH as a Standard Benchmark

Prime proposes that IOBENCH be adopted as an industry standard benchmark for evaluating the IO performance of multiuser systems. We will make the sources available without charge and act as a clearing house for any changes requested or made by the user community. In addition to the sources for IOBENCH, using IOBENCH as a standard will require defining standard test configurations and run profiles to be used with those configurations. Contact the authors for more information or to obtain a copy of IOBENCH.

References

[1] Paolo Antognetti and Giuseppe Massohbrio, *Semiconductor Device Modeling with Spice*, McGraw-Hill, New York, 1987.

[2] Arvin Park and Richard Lipton, *IOSTONE: A Synthetic File System Performance Benchmark*, Department of Computer Science, Princeton University, January 1987.

[3] TPPC Benchmark™: A Draft Proposed Standard, Transaction Processing Performance Council, 1989.

```
*** Transaction statistics ***

*** 15626 transactions consisting of 1 reads and 1 writes
*** Transactions per Second = 51.914
*** Average response time = 0.155 Seconds
*** Average CP time per transaction = 13.835 milliseconds
*** Average IO time per transaction = 0.000 milliseconds

 USER  AVG TIME   STD DEV  MAX TIME
    1     0.152     0.458     7.000
    2     0.152     0.480     7.000
    3     0.158     0.513     7.000
    4     0.152     0.503     9.000
    5     0.156     0.484     6.000
    6     0.153     0.489     7.000
    7     0.151     0.471     8.000
    8     0.156     0.478     9.000

*** Response time distribution table

Seconds |   0   |   1   |   2   |   3   |   4   |   5   |   6   |   7   |   8   |
--------+-------+-------+-------+-------+-------+-------+-------+-------+-------
Count   |13572  | 1912  |   64  |   27  |   16  |   10  |   15  |    6  |    2  |
--------+-------+-------+-------+-------+-------+-------+-------+-------+-------
Percent |86.8%  |12.2%  | 0.4%  | 0.2%  | 0.1%  | 0.1%  | 0.1%  | 0.0%  | 0.0%  |

Seconds |   9   |  10   |  11   |  12   |  13   |  14   |  15   |  16   |  17   |
--------+-------+-------+-------+-------+-------+-------+-------+-------+-------
Count   |    2  |    0  |    0  |    0  |    0  |    0  |    0  |    0  |    0  |
--------+-------+-------+-------+-------+-------+-------+-------+-------+-------
Percent | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  |

Seconds |  18   |  19   |  20   |  21   |  22   |  23   |  24   | >24   |
--------+-------+-------+-------+-------+-------+-------+-------+-------+
Count   |    0  |    0  |    0  |    0  |    0  |    0  |    0  |    0  |
--------+-------+-------+-------+-------+-------+-------+-------+-------+
Percent | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  | 0.0%  |

===    3       8     1024      0     15626      32.002      216.180        0.000      0
```

Figure 1 (continued): IOBENCH Results

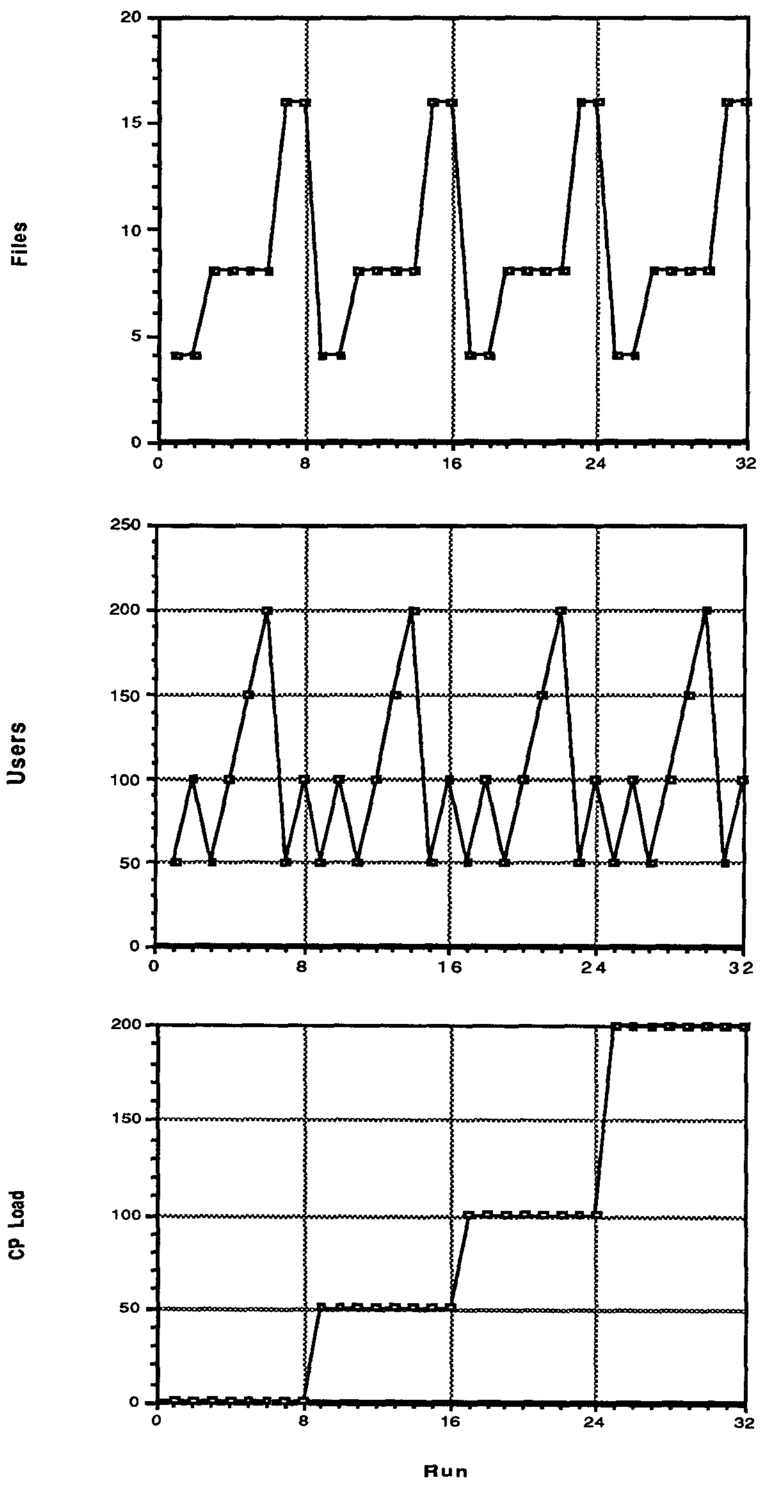

Figure 2: Multiprocessor Run Profile

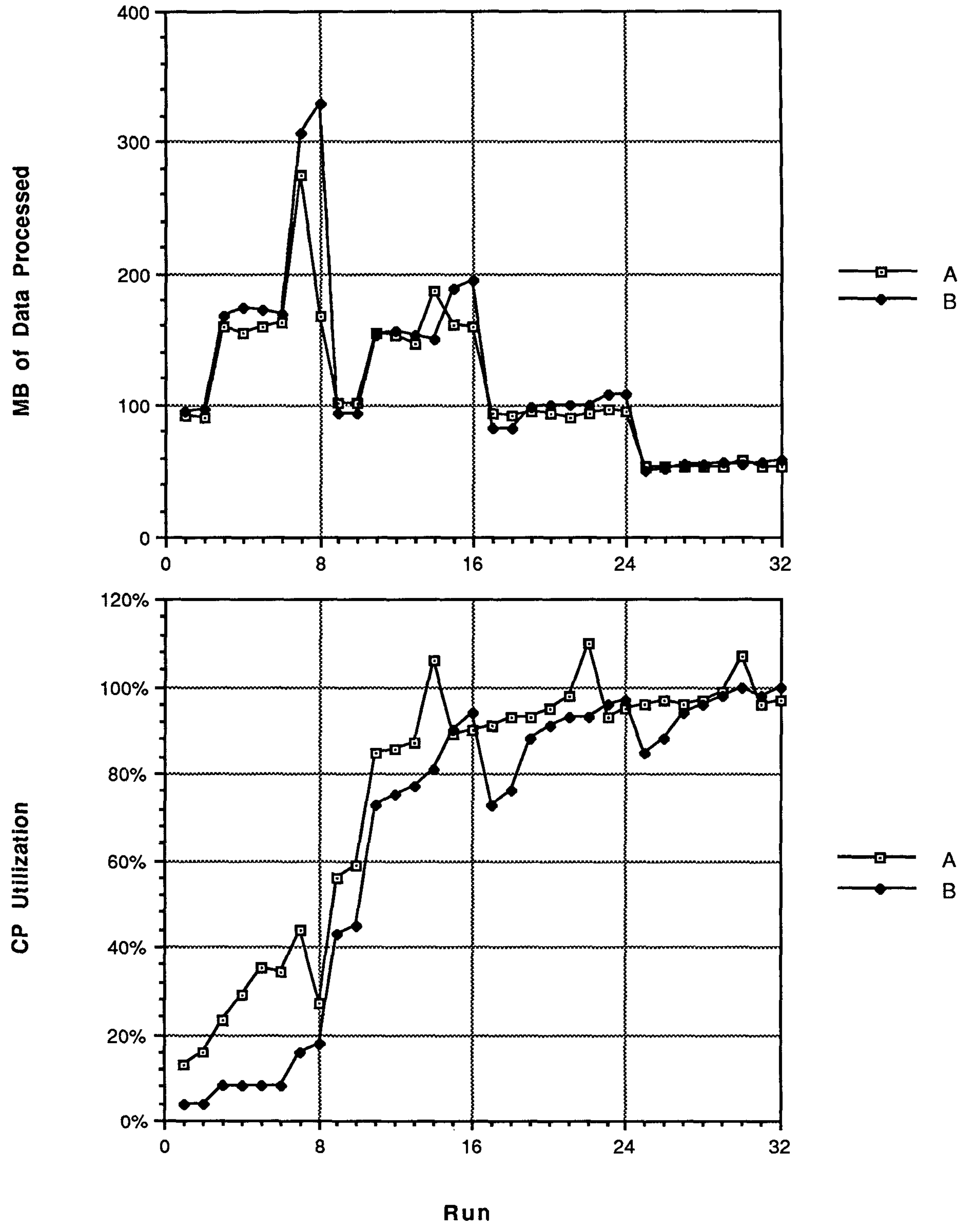

Figure 3: Multiprocessor Runs with Record Length = 2048

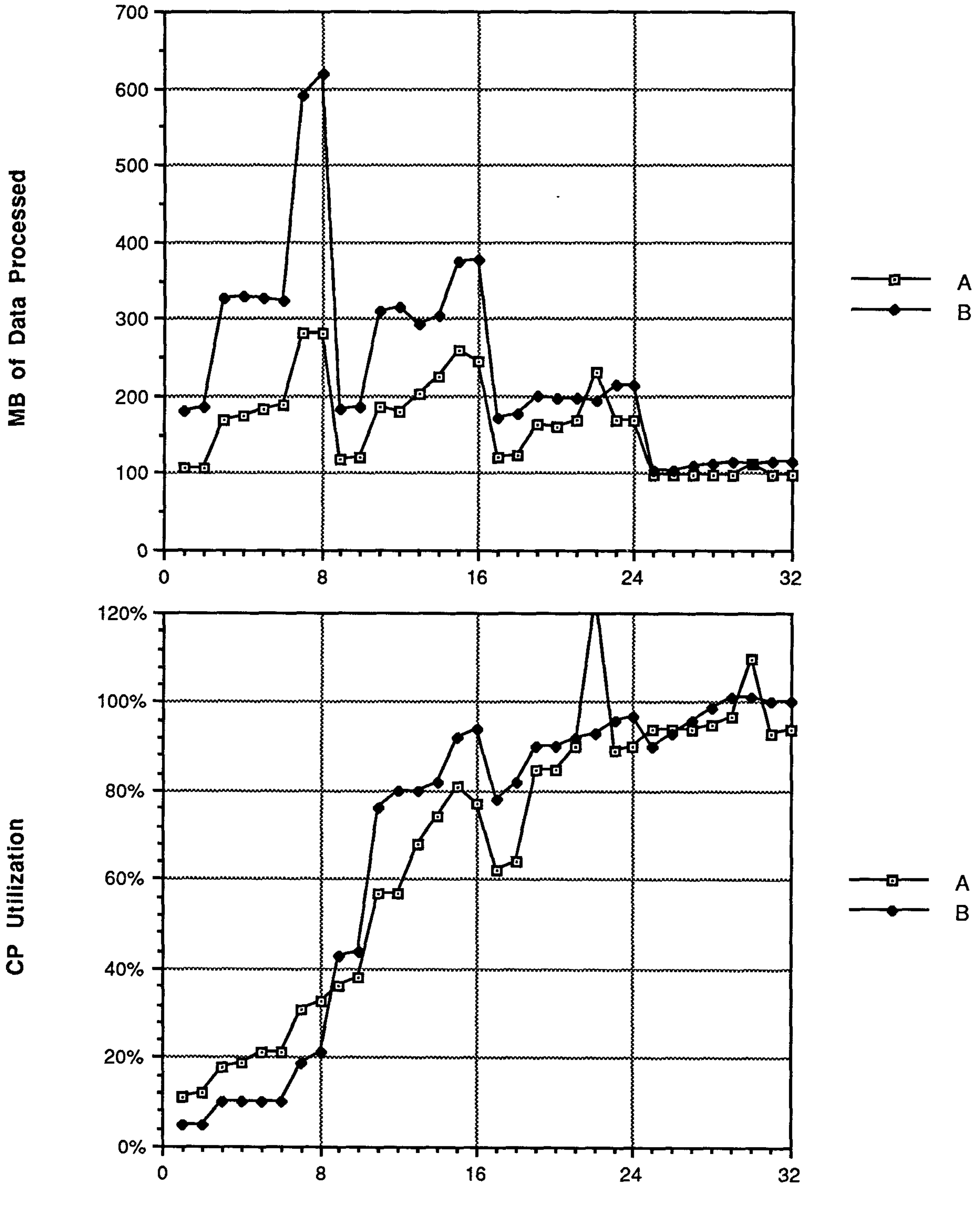

Figure 4: Multiprocessor Runs with Record Length = 4096

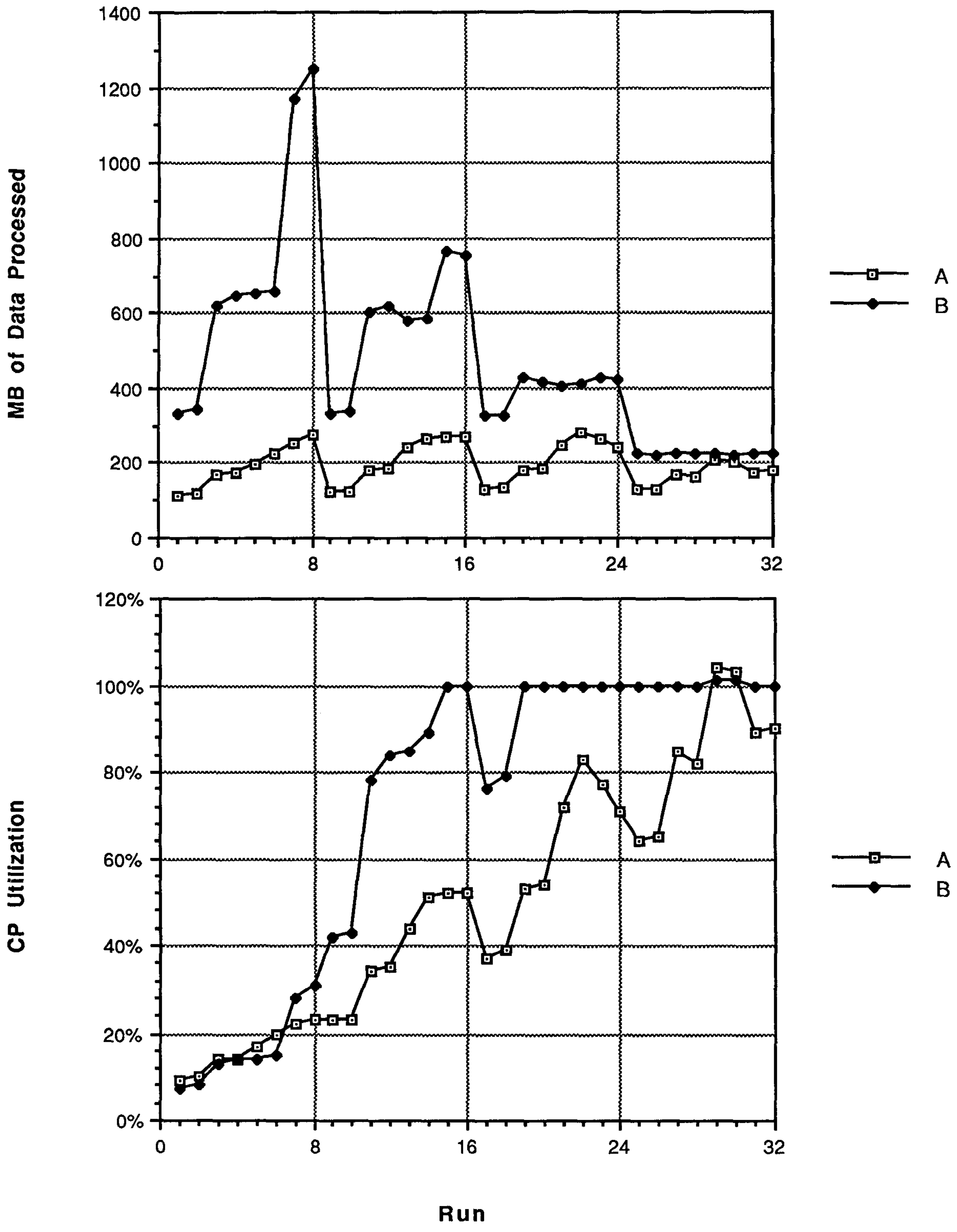

Figure 5: Multiprocessor Runs with Record Length = 8192