A TALE OF TWO RELEASES:

BENCHMARKING THE PERFORMANCE OF THE SAS SYSTEM

RELEASE 6.06 AGAINST RELEASE 5.18

Michael A. Raithel, Marriott Corporation

INTRODUCTION

Last year, at SUGI 15, the SAS Institute formally announced Release 6.06 of the SAS System. Release 6.06 was the culmination of years of staff development effort and millions of dollars in development costs. Many new concepts and structures were introduced along with the language improvements normally associated with a new release. SAS Engines, Indexing of SAS data sets, SAS Access software, and SAS data set compression are only a few of the many new features that were presented.

As SUGI 15 attendees listened to descriptions of the new concepts and features, an air of anticipation was evident. Many of the new constructs promised greater flexibility for designing SAS code to meet the data processing needs of parent organizations. It was easy to visualize how they could be exploited in one's own environment. Many were anxious to have the new release installed so they could begin to use it.

However, some of the enthusiasm was tempered by some very real questions and concerns. How did Release 6.06 of the SAS System perform? Did it consume more or less of the computer's resources than the previous release? Did the new features and flexibility of Release 6.06 come at a price of higher resource consumption? These were hard questions that had to be addressed. The answers were important not only for new applications, but especially for currently running production systems.

Those were the major questions the author explored upon returning to his organization after SUGI 15. The author designed and executed over two-hundred benchmarks to determine how conversion to Release 6.06 of the SAS System would affect application performance. This paper details the findings of the benchmarks and offers practical performance strategies.

The paper begins by discussing the goals and scope of the benchmarks. It details the environment the benchmarks were executed in and the base test job used to measure performance changes. It examines a simple conversion to Release 6.06 from 5.18 and the performance consequences. Then it discusses the new data set performance tools and how they can be used to modify resource consumption. The paper concludes with some recommendations.

BENCHMARK GOALS

When the author first began benchmarking the differences between the two releases, he wrote a lot of code that looked like this:

```
DATA FILE01;
SET IMSFILE.IMS01;
RUN;
```

The code was executed under Release 5.18 and then executed under Release 6.06. The resource consumption of the two jobs was compared and notes were jotted down. This was done for about two weeks before the author realized the basic flaw inherent in this approach. The basic flaw was: Nobody (except, perhaps a fledgling SAS programmer, or a very burned-out experienced SAS programmer) writes code like this. It is highly artificial and does nothing except burn up computer resources.

This realization led to the larger question: Why run benchmarks? Were they simply esoteric

exercises in intellectual curiosity or was there a specific goal for executing them? The latter was certainly true. First, the author wanted to determine how Release 6.06 resource consumption differed from Release 5.18. Then he had the following goals in mind:

- o Determine the best performance options
- o Set installation performance option standards
- o Convert production systems
  - Convert SAS libraries
  - Convert SAS code

The benchmarks were being run to determine the best performance options to use with Release 6.06. "Best" performance options were those which acted to reduce computer processing overhead. Once these options had been identified, installation standards could be set. Dissemination of this information would help the applications community to best exploit the power of the new release.

The information gained from the benchmarks would also be used to aid the conversion of the author's production systems to the new release. Installation performance option standards would be used to convert SAS libraries and, if necessary, SAS code.

BENCHMARK SCOPE

This is the best of benchmarks; this is the worst of benchmarks. This is the best of benchmarks, because the author selected a production job currently running under Release 5.18, converted it to Release 6.06, and benchmarked various performance options. That is what many SAS users will do, to some degree, when they convert their production systems. This is the worst of benchmarks, because, like all scientific inquiry, it is biased by the environment in which it was conducted. The processor size, the type of DASD used, the application selected, and other factors all exerted influences on the benchmark findings. Therefore, the reader is urged to carefully consider the author's environment and benchmark application in judging the applicability of the benchmark conclusions.

With the scores of new features available in Release 6.06, it was not practical to benchmark everything. Constraints on time and system availability made it necessary to tailor the scope of the benchmarks. Since the author's organization is a heavy SAS batch environment, he concentrated on benchmarking Release 6.06 in batch. Furthermore, he decided to limit the benchmarks to testing the efficiencies of the new SAS data set performance tools.

To ensure the benchmarks were "real world" in nature, a single production job was selected to be the "base test job". Statistics from the last production run of the job were recorded as the "baseline". These figures were used to measure the performance gains or losses during subsequent benchmarks. A copy of the production job and its SAS data library was created for the tests. Various aspects of the data library and the job's code were converted to Release 6.06 and the benchmarks were run. The resulting performance measurements were noted and recorded.

BENCHMARK ENVIRONMENT

All benchmarks were run on an IBM 3090/400 under MVS/XA SP 2.2.0. The SAS data library resided on IBM 3380 DASD and the default data set page size was 23040. Release 5.18 of the SAS System used entry SASLPA and Release 6.06 used entry SASXA1. A computer chargeback system based on charging for CPU time and disk I/O's was used to compute job costs.

The baseline job was a report job with three steps. Each of the steps processed a single SAS data library containing eight data sets. Each step read all eight of the data sets, extracted records meeting specific selection criteria, and then formatted and printed a report.

The library contained the following data sets:

MEMBER	LENGTH	OBSERVATIONS
DATACIC	200	3611
DATADB2	200	220359
DATAIMS	200	461337
DATAPRI	200	121912
DATASCT	200	26
DATAVCC	200	167714
DATAVMC	200	910481
INMARS	200	40939

The baseline run of the job consumed 99 CPU seconds, completed 36,668 disk I/O's and cost $38.50 to run.

SIMPLE CONVERSION

The simplest way to begin using Release 6.06 is to process Release 5.18 libraries under Release 6.06. No SAS code changes are needed to do this and existing production batch jobs can be run. However, most SAS programmers will recognize the inevitability of the conversion effort. They will undoubtably opt for converting Release 5.18 libraries to 6.06 and then running their production jobs.

Figure 1 (at the end of the paper) shows the CPU time and disk I/O's of three runs of the base test job. The first column on each chart is the baseline; the SAS 5.18 library processed under SAS 5.18. The second column is the SAS 5.18 library processed under SAS 6.06. This produced a minor drop in CPU seconds, but a rise in disk I/O's. The final column on both graphs shows Release 6.06 libraries running under Release 6.06. This brought a rise in CPU time, but a significant drop in disk I/O's.

These benchmarks are summed up by the following:

- Baseline: 5.18 library under Release 5.18
  - 99 CPU seconds
  - 37,668 disk I/O's
  - $38.50 cost
- 5.18 library under Release 6.06
  - 96 CPU seconds
  - 40,789 disk I/O's
  - $38.31 cost
- 6.06 library under Release 6.06
  - 103 CPU seconds
  - 26,960 disk I/O's
  - $37.89 cost

After converting production SAS libraries to Release 6.06, many programmers will be tempted to move on to building new applications. They might believe that dropping several thousand disk I/O's is the best performance gain they can get from the new release. This would be a wrong conclusion. Better performance gains can be had for very little effort.

PERFORMANCE TOOLS AND METRICS

With Release 6.06 of the SAS System, the SAS Institute has provided programmers with a number of performance tools. These tools, when used properly, can significantly reduce job processing overhead. The four data set performance tools used in the benchmarks were: Buffer Number, Indexing, Buffer Size, and Compression. They are all well documented in the SAS Language Reference 6 First Edition. (See References at end of paper).

Three metrics were used to determine the effectiveness of the performance tools: CPU time, disk I/O's, and memory usage. These metrics are interconnected in an inverse relationship of trade-offs. When memory is increased to hold more data, I/O's and CPU decrease. When memory is constrained and can hold less data, the I/O's and CPU increase. The author's goal was to balance these metrics within the context of his installation. CPU and disk I/O's had to be reduced without bringing the memory consumption up to too high a level.

These metrics and other performance data can be gathered from a variety of sources. PROC CONTENTS provides information about page size, indexes, compression, and block size. The SAS log provides CPU and memory consumption statistics when the STIMER and

FULLSTATS options are turned on. The MVS job log IEF374I and IEF376I messages give CPU and memory for the step and job, respectively. Finally, performance management packages such as MXG and MICS can provide CPU, memory, and I/O information.

BUFFER NUMBER

The first performance tool that was examined was Buffer Number. Buffer Number is the number of data set page buffers to allocate for a SAS data set. These buffers are allocated in main memory and are set to the page size of the SAS data set being processed. Buffer Number is not a permanent attribute of a SAS data set and can be modified by the programmer.

In Release 5.18 the value of Buffer Number could only be 1 or 2. With Release 6.06, any practical number of buffers can be coded. However, when high values (over twenty) are used, the performance gains begin to diminish as more buffers are added.

Buffer Number can be coded on either the OPTIONS statement or on the DATA step as in the following examples.

```
OPTIONS BUFNO=10;

DATA AUDIT.CIC1(BUFNO=10);
```

Buffer Number is a powerful performance tool. When used, it will greatly reduce disk I/O's. CPU time will be reduced very slightly, and there will be a small rise in memory usage.

Figure 2 shows the effects of increasing the number of buffers in the base test job. In each graph, the leftmost column represents the baseline job of a Release 5.18 library running under Release 5.18. The succeeding columns show the effects of increasing the number of buffers by multiples of five after the library has been converted to Release 6.06. When five or more buffers are used, there is only a slight decrease in CPU time. However, the number of disk I/O's is halved when five buffers are declared. The graph on the right shows a dramatic decline in disk I/O's as more buffers are used. There is a parabolic curve in the decrease of disk I/O's. After twenty buffers, the reduction is only about a hundred or so I/O's for every five additional buffers.

From the graphs in Figure 2, it is aparent that Buffer Number should be used to improve the efficiency of SAS batch jobs. Even low numbers of buffers (ie. five) have a significant impact on disk I/O's. It is important to remember why disk I/O's should be reduced. While other internal events in the life of a batch job take nanoseconds to complete, the data transfer from DASD takes milliseconds. Reductions in disk I/O's can lead to better job turnaround (wallclock) time.

The optimum benchmark with Buffer Number was:

- OPTIONS BUFFNO=20;
- CPU to 96 seconds from 99
- Disk I/O's to 7,066 from 37,669
- Cost to $31.50 from $38.50

INDEXING

The second performance tool to be benchmarked was Indexing. With Release 6.06, the ability to index a SAS data set was introduced for the first time. The programmer is able to build an index data set based on certain field(s) in a SAS data set. The index contains variable values and pointers to the observations containing those values. The index can be accessed by programs to efficiently return observations meeting certain field value criteria.

There are two types of indexes available: Simple and Composit. A Simple index is an index built from a single field in a SAS data set. A Composit index is an index built from two or more fields in a SAS data set. Either type of index can be invoked in a DATA step through the use of the WHERE expression. This means the programmer has to not only build the index, but to modify code to facilitate its use.

Below, is an example of how a SAS data set index is built. In the example, PROC DATASETS is executed to build a Simple index for the CIC1 data set. The index is based on variable COST5. Notice BUFNO=20 is used in the index creation. Buffering an index creation job will cut some CPU time and many disk I/O's.

```
OPTIONS BUFNO=20;
PROC DATASETS LIBRARY=AUDIT;
    MODIFY CIC1;
    INDEX CREATE COST5;
RUN;
```

Below is an example of how the WHERE expression is coded to exploit the index. In the example, the WHERE expression has replaced a self-limiting IF statement. It will return all observations in which COSTCTR5 equals "R601005".

```
DATA FILE01;
SET AUDIT.CIC1(WHERE=(COST5='R601005'));
RUN;
```

Figure 3 illustrates the CPU and disk I/O reductions gained by using Indexing with Buffer Number. In both graphs, the leftmost column is the baseline job of Release 5.18 libraries run under Release 5.18. The next columns represent the base test job run after indexes have been built and the code has been modified to use WHERE expressions. Notice the dramatic drop in both CPU and disk I/O's. Indexing is clearly a very powerful performance tool.

However, Figure 3 does not represent the whole performance picture related to Indexing. What about the resources expended to build the index? In Figure 4, the overhead CPU and disk I/O's from the index creation job have been added to the runs of the base test job from Figure 3. As with the previous graphs, the leftmost column is the baseline job of Release 5.18 libraries under Release 5.18 software. When the index creation overhead is added, the CPU time is three and one half times the baseline CPU time. The disk I/O's are also higher than the baseline.

The point being made here is that the index creation resource expenditure has to be seriously weighed when making a decision to use Indexing. If the base test job will be run daily once the indexes have been created, and other jobs will also use the indexes, the overhead can be justified. If this job is run only one time after the indexes have been created, and no other jobs utilize the indexes, the overhead cannot not be justified.

The optimum benchmark with Indexing was:

- OPTIONS BUFNO=20;
- CPU to 36 seconds from 99
- Disk I/O's to 7,409 from 37,668
- Cost to $12.85 from $38.50

BUFFER SIZE

The third performance tool tested in the benchmarks was Buffer Size. Buffer Size is used to set the SAS data set page size. This is also the number of bytes set aside in main memory for each buffer of a SAS data set. The Buffer Size is declared when a data set is first created and cannot be modified during the life of the data set. Buffer Size is declared on either the OPTIONS or DATA statements as in the examples, below.

```
OPTIONS BUFSIZE=46080;

DATA AUDIT.CIC1(BUFSIZE=46080);
```

Buffer Size is another powerful SAS performance tool. Increasing Buffer Size beyond 23040 (half-track blocking on IBM 3380's) results in large decreases in disk I/O's. CPU time is also decreased, but only slightly. Memory utilization rises dramatically as greater amounts of data are kept in main storage.

Figure 5 shows the results of increasing the Buffer Size from 23040 bytes to 230400 bytes in even multiples of 23040. The leftmost column of both graphs represents the baseline of Release 5.18 libraries running under Release 5.18. There is a slight decrease in CPU time, but nothing

very significant. The rightmost graph shows the decline in disk I/O's from simply declaring a larger buffer size at data set creation time. With a page size of 92160 (four times the default size of 23040), the disk I/O's have been cut well below half of those in the baseline benchmark.

When Buffer Size is used with Buffer Number, the results are even better. Figure 6 shows a varying number of buffers used with BUFSIZE=115200. This figure is different from the others because the left-hand graph shows memory utilization instead of CPU seconds. Notice how the memory size rises as the number of buffers rises. Conversely, the right hand graph shows the drastic reduction in disk I/O's as the number of buffers rises. In this example, the break-even point is at about 15 buffers. After that, memory continues to rise with very little appreciable reduction in disk I/O's.

Installations with memory constraints should be very careful about the use of large Buffer Sizes. If large sizes are used, and high Buffer Numbers are declared, memory shortages could occur and jobs could abend. Readers should consider consulting a systems programmer at their organization to understand the memory limitations of their environment.

The optimum benchmark with Buffer Size was:

- OPTIONS BUFSIZE=230400 BUFNO=15;
- CPU to 94 seconds from 99
- Disk I/O's tp 5,748 from 37,668
- Cost to $30.83 from $38.50
- Memory to 5951K from 2128

COMPRESSION

The final performance tool the author benchmarked was Compression. Compression changes the fields of a SAS data set to variable length and removes redundent data by converting it to representations using fewer bytes. This results in a reduction of the total size of the SAS data set. Compression can be initiated by coding COMPRESS=YES on either a DATA or a PROC step.

Below, is an example of how to compress a SAS data set. After this code has been executed, a PROC CONTENTS listing will reveal that the data set has been compressed.

```
DATA OUTFILE.IMSDATA(COMPRESS=YES);
SET INFILE.IMSWORK;
RUN;
```

Figure 7 shows the effects of compressing the SAS data sets used in the base test job. Again, the leftmost columns in each graph are the baseline of Release 5.18 libraries running under Release 5.18 of the SAS System. The graph on the left reveals CPU time has been dramatically raised to three and one-half times the CPU of the original baseline. The disk I/O's have been drastically reduced as in other benchmarks.

The size of the base test SAS data library was reduced by 26% when all of the data sets within it were compressed. However, as has been illustrated, this led to extremely high CPU times. (It should also be noted that CPU time was even higher when Indexing was used with Compression). Readers should seriously weigh the CPU overhead against the DASD savings. Perhaps this trade-off will make sense in some DASD constrained environments. Compression has been turned off at the author's installation.

The optimum benchmark with Compression was:

- Library to 450 cylinders from 611
- CPU to 356 seconds from 99
- Disk I/O's to 5,987 from 37,668
- Cost to $113.10 from $38.50

CONCLUSIONS

Release 6.06 of the SAS System has provided programmers with a number of performance tools. If users simply convert from Release 5.18

to 6.06 and do not exploit these tools, they will not reap the full benefits of the new release. A little effort on the user's part can significantly impact the processing overhead of batch jobs.

Buffer Number is a powerful tool that can significantly reduce disk I/O's. It should be used. The author's installation has set BUFNO=10 as the default in the SAS system options. Readers unsure of the setting in their own environment can code BUFNO on the OPTIONS statement. This will override the SAS default options setting.

Indexing can greatly reduce CPU time and disk I/O's. However, the user must weigh the costs of building the index against the number of times it will be used. If the subsequent performance returns are worth the resources invested in building the index, then Indexing should be used.

Buffer size is another performance tool that cuts disk I/O's. By choosing larger page/buffer sizes, more data is kept in memory. Used in conjunction with Buffer Number, the reduction in disk I/O's can be phenomenal. The user must be careful the attendent rise in memory usage does not cause batch jobs to abend.

Compression can reduce library size and disk I/O's. However, this is at a high cost in CPU time. DASD constrained shops may consider this to be a valid option. Because the author's environment is CPU constrained, this option has been turned off at the SAS system level.

The excitement and anticipation SUGI 15 attendees felt about Release 6.06 of the SAS System was more than justified. 'Tis a far far better release of SAS we have today than we have ever had before!

ACKNOWLEDGEMENTS

SAS is a registered trademark of SAS Institute, Inc., Cary, NC, USA.

MICS is a registered trademark of Legent Corporation.

IBM is a registered trademark of International Business Machines Corporation, Armonk, NY, USA.

The author would like to thank Mary Krobath for her consumate skill in reformatting this paper to SUGI specifications. Thanks are also due to Lynne Warburton for her skillful layout of the graphics.

REFERENCES

SAS Institute Inc., SAS Companion for the MVS Environment, Version 6, First Edition, Cary, NC: SAS Institute Inc., 1990. 589pp.

SAS Institute Inc., SAS Language: Reference, Version 6, First Edition, Cary NC: SAS Institute Inc., 1990. 1042 pp.

CONTACT INFORMATION

Michael A. Raithel
Marriott Corporation
Department 996.58
1 Marriott Drive
Washington, D.C. 20058

Telephone: (301) 380-2215

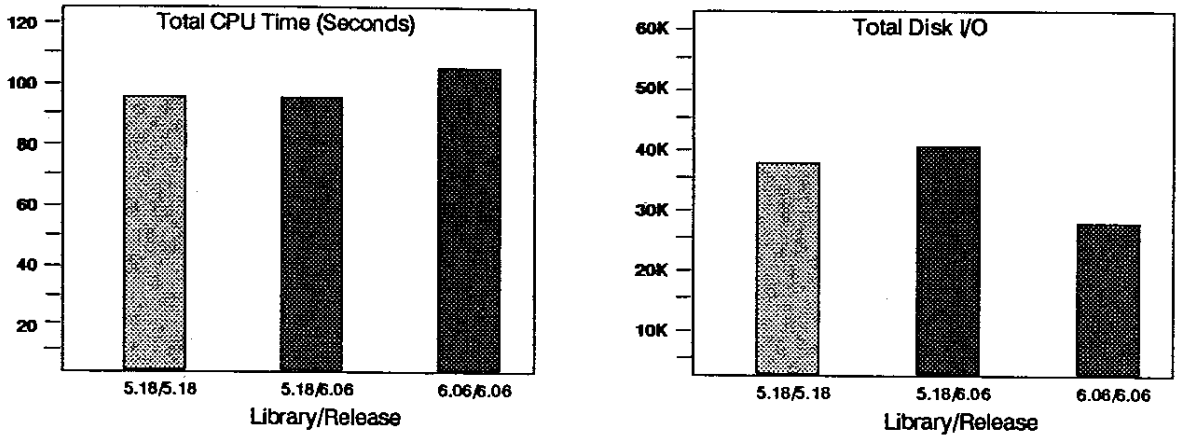

Figure 1. Simple Conversion Benchmark

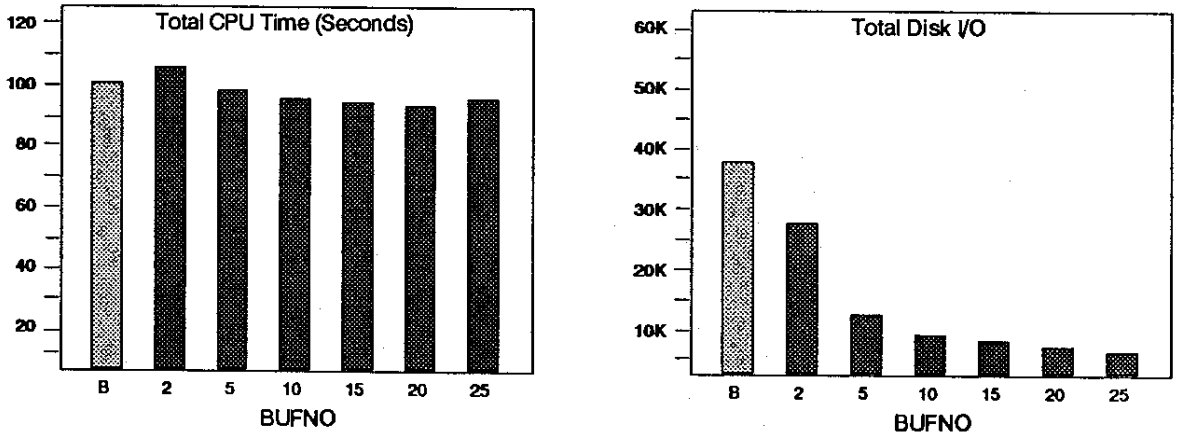

Figure 2. Benchmarking Buffer Number

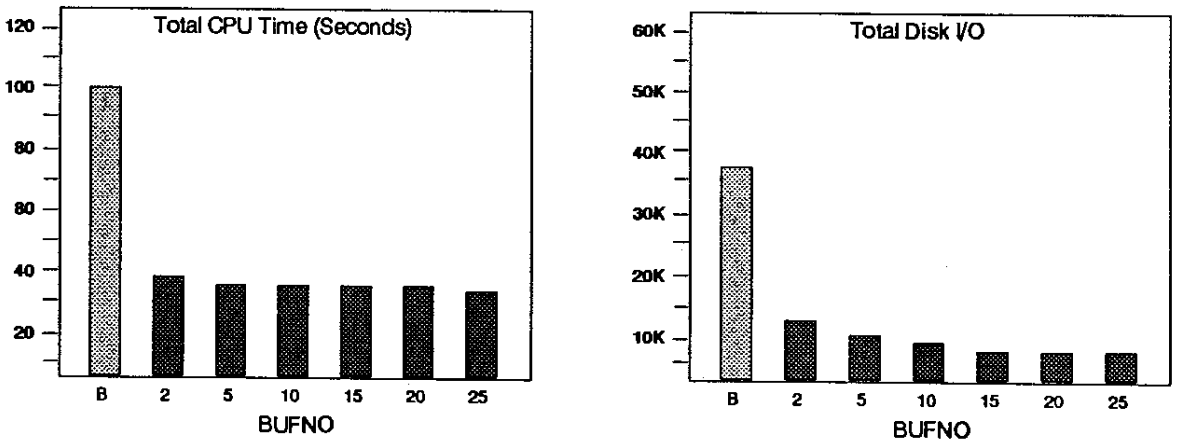

Figure 3. Benchmarking Indexing

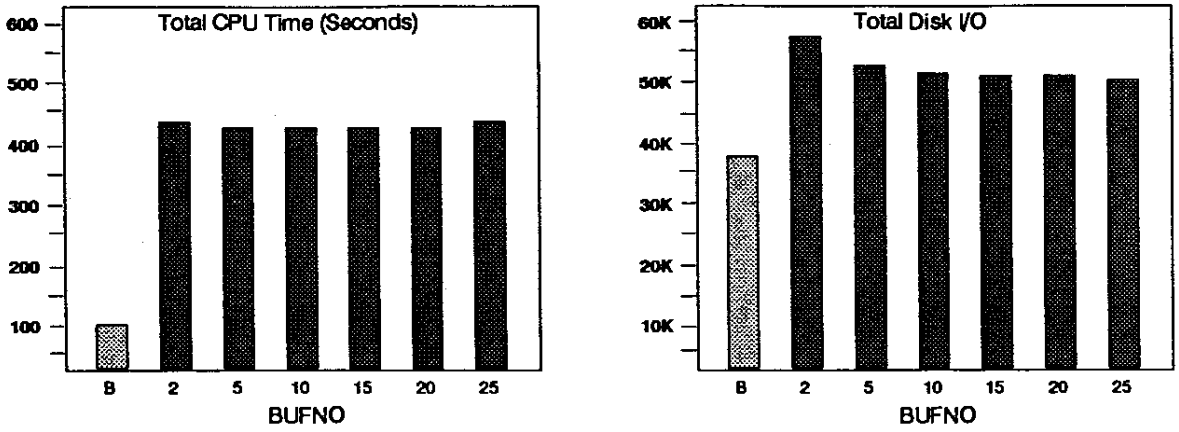

Figure 4. Indexing with Index Creation Overhead

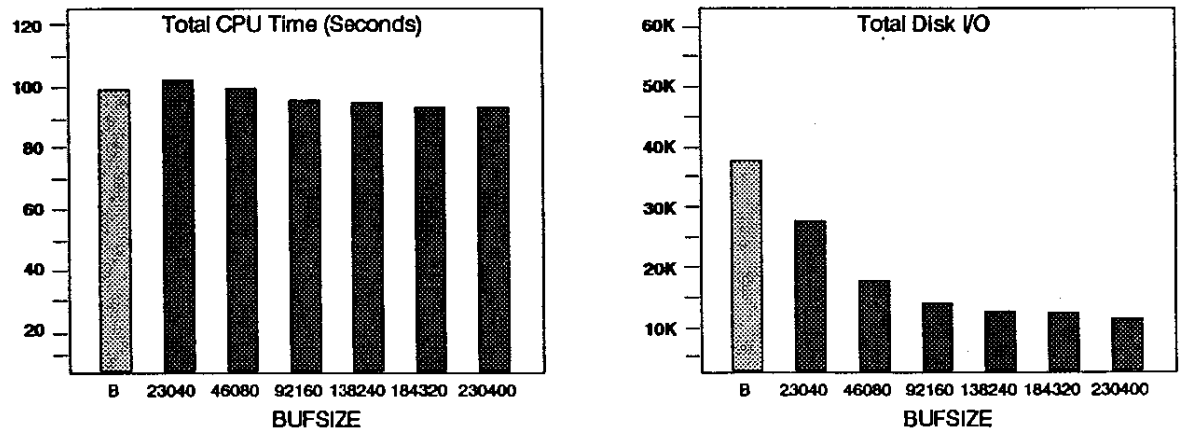

Figure 5. Benchmarking Buffer Size

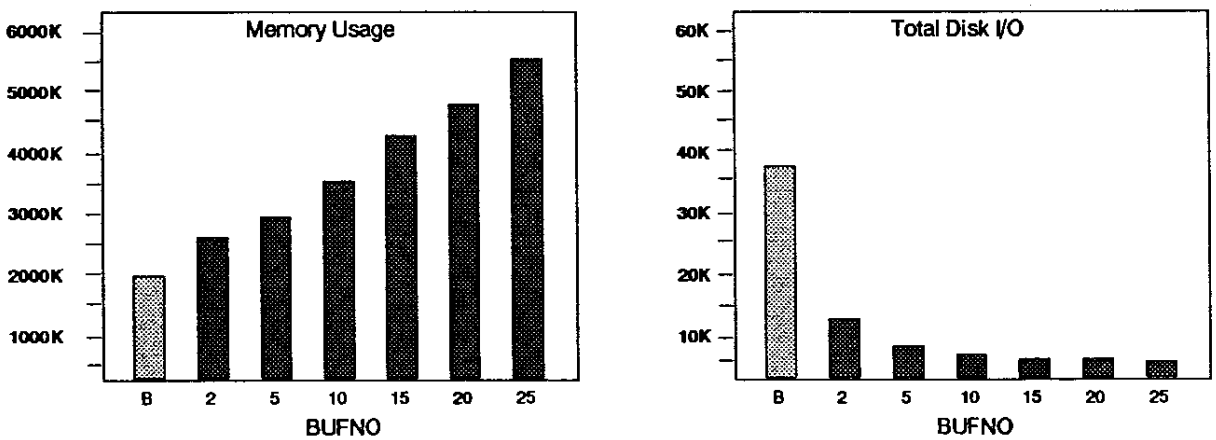

Figure 6. Benchmarking Buffer Size with Buffer Number (BUFSIZE=115200)

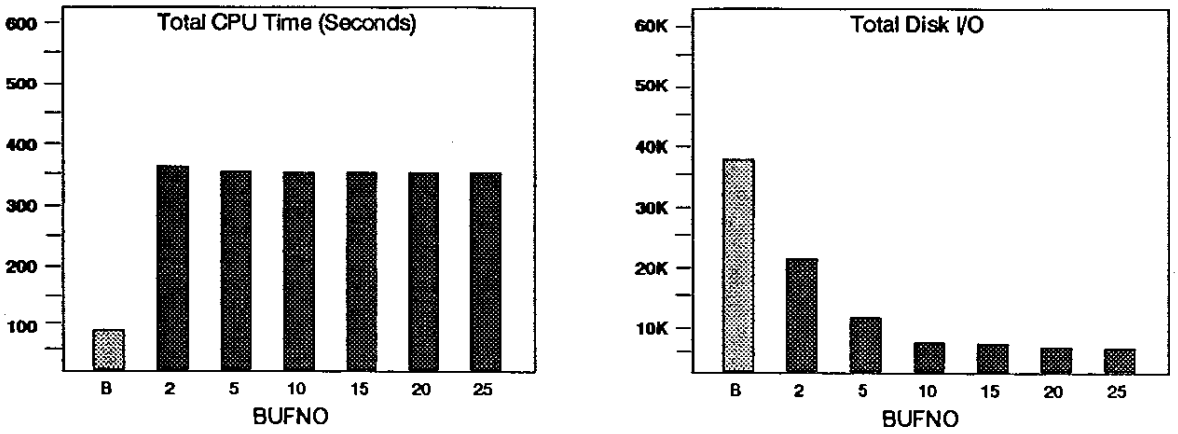

Figure 7. Benchmarking Compression