

Evaluating Efficiency Techniques Used in the Processing of Large Data Files

Alfred W. Ey
United Services Automobile Association

Abstract

The processing of large data files can often strain system resources. There are several techniques suggested by SAS Institute which can help to improve processing efficiency and resource utilization. These techniques often call for a tradeoff between time and space and may be efficient only under certain conditions. This paper evaluates some of the methods used to optimize CPU time, I/O, main memory, and disk storage. Each method of optimization is tested under differing conditions and analyzed as to the amount of efficiency gained versus the resources needed.

Introduction

Organizations working with large data files should be concerned with efficiency since large files are notorious for draining a system's resources. CPU time, I/O, main memory, and data storage are all dramatically increased when processing large files. Techniques which realize minor gains in efficiency for small files can result in significant savings of resources when used on large files. In addition, under some circumstances SAS version 6.06 is less efficient in processing than its predecessor, version 5.18 (Raithel, 1991). The increase in resources needed by the SAS system has generated a renewed interest in the use of efficiency techniques.

The book, SAS Programming Tips: A Guide to Efficient SAS Processing, provides tips and examples of efficient SAS code. The tips are focused on optimizing CPU time, I/O, memory, data storage, and programming time. While the book provides examples of efficient coding, it does not measure the efficiency gained by using the routines. It is important to measure the efficiency of the techniques since the measurement will quantify the consequences of inefficient programming. Users and programmers may not realize how much of a resource savings can be obtained until a measurement is made. A measured resource savings can also be an effective argument toward changing long standing inefficient programming practices.

Efficiency can be measured on both the amount of the system's resources used and the amount of programmer time needed to create and maintain the code. A valid argument can be made that efficiency techniques should be measured on both human and machine resources, especially for small programming jobs. The cost of programmer time is weighed against the cost savings of using different techniques. For small programming jobs it can be more cost effective to sacrifice technique for quick output. However, it can also be argued that even small programs must be

created with efficiency in mind when large data files are being used. When processing large files, regardless of program size, it is often the case that there is a heavier cost of machine resources. Therefore, cost of resource efficiency may outweigh the cost of programming time even with small programs. The system's resources which can be optimized are CPU time, I/O, main memory, and data storage.

This paper will evaluate some of the techniques suggested by the SAS Programming Tips: A Guide to Efficient SAS Processing with respect to the amount of efficiency gained when processing large data files. A measurement of the resources used by certain efficiency techniques is made with respect to CPU time, I/O, main memory, and data storage. The programming time needed to implement the techniques that are evaluated is negligible and not addressed in this paper.

Methodology

The techniques provided by SAS Institute (1991) have examples of both acceptable and more efficient code. Performance was measured by running both versions of code and recording the subsequent runtime statistics. The computer used was an IBM 3090 using SAS 6.06 on the MVS operating system. The performance statistics were obtained by using the SAS FULLSTATS option which provides information on CPU time, I/O accesses, task and total memory usage. The I/O usage is measured by the number of Executed Channel Programs (EXCP) recorded. An EXCP is an I/O call for data. A large number of EXCPs usually indicates that a program is reading or writing a large amount of data. The task memory statistic was used to describe the technique's memory requirements. Due to the large size of the files and the amount of resources used during the experiments, each test was run only once. The size and shape of the files used as well as the different techniques evaluated are described below.

File Descriptions

File size and shape was varied in order to gauge performance under different conditions. File size refers to the number of observations, file shape refers to the length of each observation. To gauge the performance of the techniques, six different sized and shaped files were used. File dimensions ranged from many observations and few variables (long and narrow) to few observations and many variables (short and wide) (Mei-Mei Ma, 1991). The files were obtained by taking three random samples from an insurance company's most recently updated research database SAS file. The research file was chosen because of its large number of records and record sizes. The samples were drawn by using the SAS UNIFORM random number generator. Each observation was assigned a random number and the observation was chosen if the number fell within the selection range. Two sets of files were created using the sample files. Each set was composed of the three sample files with one set containing a

limited number of variables. A description of the test files is shown in Table 1.

Table 1

Description of Files Used in Efficiency Technique Evaluations

File Attributes			
File	Number Observations	Observation Length	Number Variables
A1	10,000	209	53
A2	100,000	209	53
A3	500,000	209	53
B1	10,000	4547	1541
B2	100,000	4547	1541
B3	500,000	4547	1541

The definition of a large file is highly dependent on the size of the environment. A company with a small amount of computing resources combined with a large number of users may define file A1 as being large. However, an installation with a large number of resources may consider a file the size of B3 to be medium or small. For the purposes of this paper a large file will be defined based upon the number of observations, the length of the observations, and the size of the computing environment. To compensate for the computing environment and to gauge differences among differing degrees of size, all six files will be considered large.

Efficiency Techniques

Five techniques were evaluated using the above six files. Three of the techniques dealt with saving CPU time, one technique focused on saving I/O and memory, and the last technique was designed to save disk storage and I/O. An evaluation of each technique was made by using a prototypical approach (Howard 1991). A prototype of the technique was created using the efficiency example code presented in SAS Programmer Tips: A Guide to Efficient SAS Processing. The efficiency code was compared to the less efficient acceptable code also present in the book's examples. The two programs (acceptable and efficient) were executed separately and performance statistics were compared on a percent change basis. The formula for determining percent change is as follows:

$$[(\text{Acceptable Code}) - (\text{Efficient Code})] / (\text{Acceptable Code}).$$

A negative percentage indicates a decrease in resources used, a positive percentage indicates an increase. Any changes made to

the example code are noted. Listed below is a brief description of each technique and how it was evaluated. The actual SAS code used for each technique can be found in Appendix A.

Write conditional statements in order of descending probability. When SAS evaluates a series of conditions it stops the evaluation when a true condition is met. This logic implies that the most efficient IF/SELECT statements will have the condition that occurs most often placed first, the next likely placed second and so on. Two different sets of conditions were checked. One set of code had conditions which occurred 39.9%, 31.0%, and 12.9% of the time. The second set of code had conditions occurring 39.9%, 31.0%, and 0.3% of the time. The technique was evaluated by comparing the performance of conditions placed in ascending and descending order.

Use the IN operator rather than logical OR operators. The IN operator is new in version 6.06. It is similar to the OR statement in that it checks to see if a variable is present in a set of values. For example,

```
IF X IN (1,2,3) THEN OUTPUT;
```

If X has the value of 1, 2, or 3 then the observation will be output to the DATA step. The version 5.18 representation of the above code would be an IF statement with three OR operators present. The logic to this technique is that once a match is found using an IN operator, SAS will execute the THEN code without checking the rest of the list. The logical OR will evaluate all conditions even after a true condition is met. Two programs were constructed using the same variables from each of the six test files. A numeric variable was compared to 12 values using a conditional statement with an IN and an OR operator.

Use a series of conditional statements rather than the logical AND operator. This technique suggests that instead of using the AND operator:

```
IF (var1=1) AND (var2=2) AND (var3=3) THEN OUTPUT;
```

the conditions should be separated into different IF statements and linked together. For Example:

```
IF var1 = 1 THEN  
  IF var2 = 2 THEN  
    IF var3 = 3 THEN OUTPUT;
```

Like the previous technique, this efficiency tip suggests that the multiple IF statements will discontinue the evaluation when a false condition occurs; however, the logical AND operator will evaluate all the conditions regardless of what is found. SAS

suggests that this technique is only useful for a large number of conditions. Both versions were tested using the same 8 variables, creating 8 different conditions.

Use WHERE conditions in procedures. The WHERE clause is a new addition to SAS. In version 5.18 when a user ran a procedure on a subset of their data they had to follow these steps: read the data into a temporary data set, subset the data, and then pass the resulting data set to the procedure. The WHERE clause allows users to subset their data within the procedure, thereby eliminating the need to read the data twice, once to subset the data and once to analyze it. Two procedures were evaluated using two different WHERE statements. A PROC FREQ and a PROC UNIVARIATE were run to examine the effects of the WHERE clause on different procedures. To test the effects of different conditions, each procedure was evaluated using two different subset conditions. To gauge the efficiency, each procedure was also run by subsetting the data by using a temporary DATA step and then passing the resulting data set to the procedure. The system's resources were accumulated by adding the CPU, I/O and Memory used in the DATA step to those resources used in the procedure. The variables used in the WHERE conditions were not index variables.

Compress large SAS data files. Another addition to SAS version 6.06 is the data compression option. Data compression reduces the size of a file by compressing redundant values to 2 or 3 bytes of data. Five of the six data sets were copied into new files with the compress option on. One DATA step and two procedures were run separately against the compressed and non-compressed files to measure any increases or decreases in resources used. The B3 file was not compressed due to disk space limitations.

Results and Discussion

Conditional Efficiencies

Perhaps one of the most commonly used programming constructs of the SAS DATA step is the IF statement. Three techniques for improving the efficiency of IF statements were chosen and evaluated. All three conditional techniques had the same underlying concept: once a condition is met, the SAS system stops evaluating the criteria and continues on with other processing. The techniques examined claim to prevent the SAS system from wasting time by evaluating all of the criteria even after the decision is clear.

The first conditional technique examined was the placing of conditional criteria in descending order of probability. As can be seen in Table 2, two sets of conditions were placed in ascending and descending order of occurrence. The descending order of probability did not show a remarkable improvement over conditions placed in ascending order for either set of

conditions. It was expected that the difference between the two different sets of probabilities tested would have been pronounced. The small differences that are shown could have been caused by the system, rather than the differences in the programs. All the differences between the ascending and descending conditions, with the exception of file B3, were less than 1 CPU second (See Appendix B). Another cause of the inconsistency of the results could be the uniformity of the probabilities chosen. Perhaps a better test may be constructed with conditions that have probabilities of 90%, 2%, and 1%. These probabilities placed in ascending order should execute the two conditions (1% and 2%) 90% percent of the time; the same probabilities placed in descending order would only check the two lower probabilities 10% of the time.

The results suggest that for the conditions chosen, the technique is really no more efficient than placing the conditions in ascending order.

Table 2

Percentage Change in Resources for Placing
Conditions in Order of Descending
Probability

File	CPU Time	
	Set 1	Set 2
A1	-3.33	1.79
A2	-1.89	- .26
A3	0.00	- .05
B1	-1.09	.83
B2	1.23	.28
B3	.12	-1.37

The results in Table 3 show that the technique suggesting that the IN operator be used instead of the logical OR operator was inappropriate for the files tested. All of the files had some percentage increase in the amount of CPU time when using the IN operator. It is actually more efficient in all cases to use the OR operator instead of the IN. This is in agreement with Defoor (1991), who also found the IN statement to be less efficient. While the use of the IN statement is not being discouraged, it should be noted that the OR operator is as efficient and in some cases more efficient than the IN operator.

The technique prescribing the use of multiple IF statements over the AND operator did prove to be more efficient on all six of the files. Table 4 shows small differences in CPU time between the same size files. This would indicate that the number

Table 3

Percentage Change in CPU Time for
IN Operator Versus OR Operator

File	CPU Time
A1	10.45
A2	12.35
A3	11.21
B1	.58
B2	2.76
B3	7.25

of observations and not the observation length is influencing CPU savings. The technique saves a small amount of CPU per observation, therefore, more CPU time is saved as the number of observations in the file increases. An increase in the number of IF statements used would also raise the CPU savings by increasing the CPU time saved per observation. Regardless of the size of the observations, the technique is more efficient for large numbers of observations with large numbers of IF statements.

Table 4

CPU Time Used for Multiple IF Statements
Versus the And Operator

File	CPU Time (Seconds)		
	And Operator	Multiple IF	Difference
A1	.47	.43	.04
A2	2.99	2.68	.31
A3	14.03	12.54	1.49
B1	4.36	4.31	.05
B2	40.29	39.95	.34
B3	196.52	194.96	1.56

WHERE Condition Efficiencies

Using the WHERE condition in a procedure saves the time and space needed to read the data into a temporary data set, subset the data, and then pass the data to the procedure. The technique was purported to save I/O and memory, however, the results show that CPU time was also greatly decreased. Table 5 shows the percentages of resources saved. A small amount of CPU time is

used each time an observation is read, therefore savings gained in EXCPs translates into a savings in CPU time. It is unclear as to why the univariate procedure was actually less efficient in using CPU for file A2.

The percentage savings of memory and I/O was consistent for each procedure using the same conditions. It was expected that the procedures using the two conditional statements would result in smaller increases in efficiency than the same procedures using only one subsetting condition. The smaller the subset of data the less the percentage of savings. The savings of resources is realized by processing the data only once using a WHERE clause in a procedure; however, if when using a temporary DATA step the subset of data is very small, the second reading of the data by the procedure will not be significant. The two conditionals reduced the size of the data set to be analyzed so the overall efficiency gained is lower.

The files with long observations have a substantial percentage decrease in the amount of resources used. There is a certain amount of overhead associated with processing SAS variables; the B files have over 1400 variables compared to the 53 variables present in the A files. Procedures using a WHERE statement to subset data only process variables once, this infers that a greater savings is realized for files which contain a large amount of variables. The savings would be realized in the CPU time and memory categories. For large files the WHERE statement is more efficient than a temporary DATA step for subsetting data to be used in a procedure. The amount of efficiency gained is related to the length of the observation and the size of the subset data. A large subset of data produces a larger savings of system's resources. Likewise, the more variables the observation has, the greater the percentage savings of CPU time and memory.

Data Compression

Table 6 shows the amount of storage space saved due to data compression. Approximately 90% of the variables in all of the data files are numeric. Over half of the variables in the B files are blank or zero. Because of the data redundancy present, the B files were expected to have a high percentage of disk storage savings.

The amount of disk savings is quite impressive, however, the amount of CPU time needed to use these compressed files can increase from 50% to over 400%. Table 7 shows both the increase in CPU time and the decrease in the number of EXCPs. The larger the file the larger the increase in CPU and the larger the decrease in EXCPs. The large tradeoff as suggested by SAS, is CPU time. The increase in CPU time is caused by the added overhead of the decompression algorithm. The B files took up to 2 to 3 times more CPU than their non-compressed counterparts. In comparison, the procedures took longer for the B files where as the DATA step used more CPU for the A files.

Table 5

Percent Change of Systems Resources for Using the WHERE Condition in Procedures

File	System Resources		
	CPU Time	EXCP	Memory
Frequency Distribution with 1 Condition			
A1	-39.06	-68.10	-15.29
A2	-25.81	-46.15	-15.20
A3	-27.32	-17.21	-13.43
B1	-85.69	-64.55	-44.01
B2	-72.33	-13.36	-33.00
B3	-72.50	-11.09	-33.11
Frequency Distribution with 2 Conditions			
A1	-26.23	-64.19	-14.72
A2	-11.11	-41.75	-14.65
A3	-12.75	-14.41	-12.87
B1	-83.97	-62.85	-43.81
B2	-70.05	-10.86	-32.66
B3	-70.34	-08.56	-32.76
Univariate with 1 Condition			
A1	-11.11	-68.46	-17.43
A2	23.69	-35.59	-17.35
A3	-26.67	-17.25	-16.99
B1	-81.82	-65.23	-46.10
B2	-72.67	-15.33	-35.63
B3	-73.51	-11.07	-35.63
Univariate with 2 Conditions			
A1	-09.38	-77.18	-17.43
A2	31.00	-34.58	-17.35
A3	-12.87	-14.69	-17.12
B1	-81.08	-65.19	-46.15
B2	-70.82	-12.56	-35.63
B3	-69.97	-08.58	-35.63

Note. Memory was measured using the total memory used during the SAS session.

Table 6

File Sizes for Non-Compressed and Compressed Data

File	Non-Compressed (tracks)	Compressed (tracks)	% Saved
A1	49	28	42.86
A2	824	254	69.17
A3	2021	1260	37.65
B1	972	257	73.56
B2	9616	2268	76.41

Note. The tracks used were IBM 3380 disk tracks, they hold 47,475 bytes of data. Files A3 and B2 were on tape, the non-compressed disk space was estimated using the SAS Companion for the MVS Environment.

The compression reduces the I/O by reducing the amount of data that needs to be read. The I/O savings was nominal for files with only 10,000 observations. This is one technique where the tradeoffs are straightforward and obvious. If there is a large amount of compression savings then there will be a large increase in CPU needed to use the files. The increase is especially noticeable in files with observations of 100,000 or more. The decision to be made is whether the savings realized in disk storage and I/O is worth the cost of CPU time.

Table 7

Percentage Change of Resources for Procedures
Using Non-Compressed and Compressed Files

System Resources		
File	CPU Time	EXCP
Frequency Distribution Without Conditionals		
A1	51.52	-3.82
A2	66.93	-36.88
A3	66.08	-79.27
B1	397.12	-18.82
B2	220.16	-83.52
Frequency Distribution with 2 Conditions		
A1	71.43	-3.68
A2	96.13	-38.27
A3	104.94	-79.40
B1	477.01	-22.33
B2	254.60	-83.59
DATA Step		
A1	72.92	-.84
A2	107.17	-26.29
A3	111.57	-70.93
B1	77.60	-6.21
B2	71.63	-76.00

Conclusion

The need to measure and quantify an efficiency technique can be seen in the results of this study. Two of the five proposed efficiency techniques were found to be inefficient for the data tested. The compression technique was found to save up to 75% of the disk space for one file at an increase of 250% for CPU time. Programmers, analysts, and users all need to be aware of the techniques that are available and how these techniques work. Version 6.06 of SAS has offered some new and exciting additions to the SAS language, however, these and other more proven techniques should be measured against the data. The measurement of a proposed efficiency technique is even more important for large data files which often have a voracious appetite for system's resources.

References

- Defoor, J. (1991). A comparison of data step performance in versions 6.06 and 5.18, SAS Users Group International Conference Proceedings: SUGI 16 (p 52), Cary, NC: SAS Institute Inc.
- Howard, N. (1991). Efficiency techniques for improving i/o and processing time in the data step, SAS Users Group International Conference Proceedings: SUGI 16 (p 285), Cary, NC: SAS Institute Inc.
- Mei-Mei, M. (1991). Efficiency revisited: large files and release 6.06, SAS Users Group International Conference Proceedings: SUGI 14 (p 390), Cary, NC: SAS Institute Inc.
- Raithel, M. (1991). A tale of two releases: benchmarking the performance of the sas system release 6.06 against release 5.18, SAS Users Group International Conference Proceedings: SUGI 14 (pp 498-500), Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (1990). SAS Companion for the MVS Environment, Version 6, First Edition (pp 537-539), Cary, NC: SAS Institute Inc.
- SAS Institute Inc. (1990), SAS Programming Tips: A Guide To Efficient SAS Processing, Cary, NC: SAS Institute, Inc., 1990. 155 pp.

Appendix A.

SAS Code Used in Evaluations

I. Place Conditional Statements in Order of Descending Probability

Acceptable Code

```
SELECT (X);

    WHEN (4) OUTPUT;    /* Probability is 12% */
    WHEN (2) OUTPUT;    /* Probability is 31% */
    WHEN (6) OUTPUT;    /* Probability is 39% */

    OTHERWISE;
END;
```

Efficient Code

```
SELECT (X);

    WHEN (6) OUTPUT;    /* Probability is 39% */
    WHEN (2) OUTPUT;    /* Probability is 31% */
    WHEN (4) OUTPUT;    /* Probability is 12% */

    OTHERWISE;
END;
```

Acceptable Code

```
SELECT (X);

    WHEN (3) OUTPUT;    /* Probability is 03% */
    WHEN (2) OUTPUT;    /* Probability is 31% */
    WHEN (6) OUTPUT;    /* Probability is 39% */

    OTHERWISE;
END;
```

Efficient Code

```
SELECT (X);

    WHEN (6) OUTPUT;    /* Probability is 39% */
    WHEN (2) OUTPUT;    /* Probability is 31% */
    WHEN (3) OUTPUT;    /* Probability is 03% */

    OTHERWISE;
END;
```

II. Use IN Operator Rather Than Logical OR Operator

Acceptable Code

```
IF A = 13 OR
  A = 12 OR
  A = 14 OR
  A = 15 OR
  A = 21 OR
  A = 22 OR
  A = 24 OR
  A = 26 OR
  A = 33 OR
  A = 34 OR
  A = 40 OR
  A = 48 THEN B = 'AA';
ELSE
  B = 'BB';
```

Efficient Code

```
IF A IN (13,12,14,15,21,22,24,26,33,34,40,48) THEN
  B = 'AA';
ELSE
  B = 'BB';
```

III. Use a Series of Conditional Statements Rather than the Logical AND Operator

Acceptable Code

```
IF (A = 2)      AND
  (16<=B<=19)   AND
  (C = 'BB')    AND
  (D NE 5)      AND
  (E > 2)       AND
  (F > 0)       AND
  (G < 1)       AND
  (H LE 50)     THEN OUTPUT;
```

Efficient Code

```
IF (A = 2)
  IF (16<=B<=19)
    IF (C = 'BB')
      IF (D NE 5)
        IF (E > 2)
          IF (F > 0)
            IF (G < 1)
              IF (H LE 50) THEN OUTPUT;
```

IV. Use WHERE Conditions in Procedures

Subset Data Using 1 Conditional

Acceptable Code

```
DATA TEMP1;  
  SET IN1.FILEA1;  
  
  IF (X = 2);  
  
PROC FREQ DATA=TEMP1;  
  TABLES X Y Z;  
  
PROC UNIVARIATE DATA=TEMP1 NORMAL;  
  VAR W;
```

Efficient Code

```
PROC FREQ DATA=IN1.FILEA1;  
  WHERE (X = 2);  
  TABLES X Y Z;  
  
PROC UNIVARIATE DATA=IN1.FILEA1;  
  WHERE (X = 2);  
  TABLES X Y Z;
```

Subset Data Using 2 Conditionals

Acceptable Code

```
DATA TEMP2;  
  SET IN1.FILE;  
  
  IF (X = 2) and (Y = 1);  
  
PROC FREQ DATA=TEMP1;  
  TABLES X Y Z;  
  
PROC UNIVARIATE DATA=TEMP1 NORMAL;  
  VAR W;
```

Efficient Code

```
PROC FREQ DATA=IN1.FILE;  
  WHERE (X = 2) AND (Y = 1);  
  TABLES X Y Z;  
  
PROC UNIVARIATE DATA=IN1.FILE;  
  WHERE (X = 2) AND (Y = 1);  
  TABLES X Y Z;
```

V. Compress Large SAS Data Files

This code was run on both non-compressed and compressed data files.

```
PROC FREQ DATA=IN1.FILE;  
  TABLES A B C D E F;
```

```
PROC FREQ DATA=IN1.FILE;  
  WHERE (A = 2);  
  TABLES A B C D E F;
```

```
DATA TEMP;  
  SET IN1.FILE;
```

```
  IF A = 2;
```

```
  IF (B = 1) AND (C = 2) THEN D = 1;  
  ELSE D = 0;
```

Appendix B

RESULTS OF EFFICIENCY TECHNIQUE EVALUATIONS

	A1	A1E	A1%	A2	A2E	A2%	A3	A3E	A3%	B1	B1E	B1%	B2	B2E	B2%	B3	B3E	B3%
CONDITIONS PLACED IN DESCENDING PROBABILITY (39%,30%,12%)																		
CPU	0.6	0.58	-3.33	4.23	4.15	-1.89	19.9	19.9	0	6.41	6.34	-1.09	60.01	60.75	1.23	198.43	198.68	0.12
CONDITIONS PLACED IN DESCENDING PROBABILITY (39%,30%,3%)																		
CPU	0.56	0.57	1.79	3.92	3.91	-0.26	18.73	18.72	-0.05	6.06	6.11	0.83	56.89	57.05	0.28	196.44	193.74	-1.37
IN OPERATOR VS. OR OPERATOR																		
CPU	0.67	0.74	10.45	4.94	5.55	12.35	23.65	26.3	11.21	6.58	6.89	0.58	65.12	66.92	2.76	66.62	71.45	7.25
MULTIPLE IF STATEMENTS VS. LOGICAL AND OPERATOR																		
CPU	0.47	0.43	-8.51	2.99	2.68	-10.37	14.03	12.54	-10.62	4.36	4.31	-1.14	40.29	39.95	-0.84	196.52	194.96	-0.79
WHERE STATEMENT - PROC FREQ (1 Condition)																		
CPU	0.64	0.39	-39.06	4.03	2.99	-25.81	19.07	13.86	-27.32	5.17	0.74	-85.69	48.25	13.35	-72.33	234.01	64.16	-72.58
EXCP	326	104	-68.1	858	462	-46.15	7089	5869	-17.21	1100	390	-64.55	16297	14120	-13.36	78731	70001	-11.09
MEMORY	2511	2127	-15.29	2527	2143	-15.2	2592	2244	-13.43	4108	2300	-44.01	3697	2477	-33	3697	2473	-33.11
WHERE STATEMENT - PROC FREQ (2 Conditions)																		
CPU	0.61	0.45	-26.23	3.78	3.36	-11.11	17.88	15.6	-12.75	4.99	0.8	-83.97	46.17	13.83	-70.05	220.07	65.27	-70.34
EXCP	310	111	-64.19	800	466	-41.75	6871	5881	-14.41	1047	389	-62.85	15863	14141	-10.86	76564	70011	-8.56
MEMORY	2507	2138	-14.72	2519	2150	-14.65	2588	2255	-12.87	4104	2306	-43.81	3693	2487	-32.66	3693	2483	-32.76
WHERE STATEMENT - PROC UNIVARIATE (1 Condition)																		
CPU	0.63	0.56	-11.11	4.01	4.96	23.69	19.12	14.02	-26.67	5.17	0.94	-81.82	48.34	13.21	-72.67	234.15	62.03	-73.51
EXCP	298	94	-68.46	829	534	-35.59	7059	5841	-17.25	1070	372	-65.23	16646	14094	-15.33	78701	69986	-11.07
MEMORY	2495	2060	-17.43	2507	2072	-17.35	2584	2145	-16.99	4104	2212	-46.1	3693	2377	-35.63	3693	2377	-35.63
WHERE STATEMENT - PROC UNIVARIATE (2 Conditions)																		
CPU	0.64	0.58	-9.38	3.71	4.86	31	17.71	15.43	-12.87	5.02	0.95	-81.08	46.33	13.52	-70.82	217.01	65.17	-69.97
EXCP	390	89	-77.18	775	507	-34.58	6849	5843	-14.69	1063	370	-65.19	16118	14093	-12.56	76553	69984	-8.58
MEMORY	2495	2060	-17.43	2507	2072	-17.35	2588	2145	-17.12	4108	2212	-46.15	3693	2377	-35.63	3693	2377	-35.63
DATA COMPRESSION - PROC FREQ (No Conditions)																		
CPU	0.66	1	51.52	5.14	8.58	66.93	25.65	42.6	66.08	1.04	5.17	397.12	15.48	49.56	220.16			
EXCP	157	151	-3.82	507	320	-36.88	5918	1227	-79.27	425	345	-18.82	14170	2335	-83.52			
MEMORY	2167	2167	0	2183	2179	-0.18	2304	2203	-4.38	2323	2527	8.78	2516	2523	0.28			
DATA COMPRESSION - PROC FREQ (1 Condition)																		
CPU	0.49	0.84	71.43	3.62	7.1	96.13	17	34.84	104.94	0.87	5.02	477.01	13.55	48.05	254.6			
EXCP	136	131	-3.68	486	300	-38.27	5887	1213	-79.4	412	320	-22.33	14149	2322	-83.59			
MEMORY	2151	2151	0	2175	2175	0	2276	2179	-4.26	2323	2511	8.09	2516	2523	0.28			
DATA COMPRESSION - DATA STEP																		
CPU	0.48	0.83	72.92	3.21	6.65	107.17	15.38	32.54	111.57	5.09	9.04	77.6	46.45	79.72	71.63			
EXCP	238	236	-0.84	700	516	-26.29	6611	1922	-70.93	966	906	-6.21	15565	3736	-76			
MEMORY	2487	2487	0	2495	2495	0	2568	2491	-3	4084	4280	4.8	3669	4280	16.65			

Note: E indicates file run with efficiency technique. % is percent change.