ABSTRACT
This paper explores a number of SAS System data management and coding techniques for improving efficiency. After efficiency is defined and some basic efficiency techniques are explained, some SAS Version 6.07+ features are discussed from an efficiency viewpoint. These features include PROC SQL, SAS data set compressing, the Stored Program Facility, and SAS Data Views.

INTRODUCTION - Definition of Efficiency
Efficient coding is more than being able to code a program that will run the quickest. Efficiency can be broadly defined as completing a task while utilizing the minimum resources.

Since there are many different resources used in completing a programming task, there are many aspects to efficiency. These include elapsed time, CPU time, memory usage, data storage, and I/O operations. Also very important, and too frequently ignored, is the expenditure of programmer time for design, coding, and maintenance. All these factors should be considered when measuring efficiency.

Various SAS System Options are can be used to obtain performance statistics. Often used options are indicated below. Some options are platform specific, as noted.

OPTIONS NOTES
MPRINT SOURCE SOURCE2 /* Log source code */
STIMER FULLSTIMER /* Usage statistics */
MSGLEVEL=I /* PC Only */
MEMRPT FULLSTATS /* MVS only */ ;

All of these options utilize resources too, so be sure to use them only when needed.

Efficiency Goals
The efficiency techniques that you use will depend upon your specific goals. Think of efficiency goals as progressive goals, where basic efficiency goals should to be met before progressing to more specialized goals. Some sample goals are listed below:

Learn Basic Efficient Techniques
Fortunately, the biggest savings in SAS coding are obtained from the most basic and, usually, intuitive techniques. These techniques are summarized in a SAS Institute manual entitled "SAS Programming Tips: A Guide to Efficient SAS Processing" and are mainly a combination of good programming practice and common sense.

Minimize Costs
By determining your costs, you can address issues that directly affect your organization's budget. For example, if your time (including benefits) is valued at $20 per hour and you are charged 25¢ per CPU second, one hour of your time is worth 80 CPU seconds. This amount of CPU reduction can easily be obtained by making a large, inefficient job more efficient. On the other hand, a huge loss could occur by re-writing an application using techniques that will save only a minimal amount of resources.

If any of your costs are variable, attempt to take advantage of this variability (eg. lower job costs that may be offered for off-peak job submissions, better response time during the lunch hour).

Minimize Utilization of Critical Resources
Data storage may be cheap and abundant at your site, making CPU usage a focus of this goal. Memory constraints may be more of a concern for smaller machines. By identifying critical resources on your platform you can code to decrease usage of these resources, usually at the expense of other resources.

Optimize an Application
Is your application a batch job or an interactive system? Speed is especially critical for on-line applications. Does your application use large data sets or small? Very large data sets may require that you attempt to optimize data storage and I/O considerations at the expense of other factors.

Lower Costs In an Already Efficient Application
By the time you have progressed to this goal, your code will be so efficient that squeezing those last few cents from an application will be difficult. The best situation would be when working with large systems or implementing a new feature of the SAS System.

With the SAS System now widely available on multiple platforms in an organization, we are now faced with not only learning generalized efficiency techniques but also utilizing the relative efficiencies of different platforms in order to achieve our goals. Thus, while basic efficiency techniques are applicable to all platforms, each of the other goals may be achieved differently for each SAS platform.

Test Data Sets and Testing Platforms
The results of test runs are not presented in this paper due to space limitations. Rest assured, though, that there were hundreds of tests run to establish this synopsis of techniques that showed a significant advantage. Feel free to contact the author for the actual code or results of tests performed.

These tests used 12 different SAS 6.07 data sets, each containing a mixture of randomly generated numeric and character variables. They were run in Version 6.07 on an Hitachi 9021 mainframe under MVS. The characteristics of these data sets are shown on the following page.
Large 150,000 obs 25, 75, and 150 variables
Medium 25,000 obs 25, 75, and 150 variables
Small 6,000 obs 25, 75, and 150 variables
Very Small 1,000 obs 25, 75, and 150 variables

(Record Lengths=191, 670, and 1340)

The Small, and Very Small data sets were also tested via a LAN based SAS, Version 6.08, on an IBM PC with 8 megabytes of memory under Windows 3.1 and a math co-processor.

This PC configuration is no trivial matter. Version 6.08 runs about 40% faster than Version 6.04 with the aforementioned configuration. However, an Intel 80386 with 4 meg of memory and no math coprocessor takes twice as long to run 6.08 as it does 6.04.

As previously mentioned, there will usually be different costs and resources associated with different SAS platforms. Because of this, different statistics were used when measuring efficiency on the two testing platforms.

Mainframe charges are billed back according to CPU usage and tape EXCP's. There are also costs associated with the amount of data storage that is used. Thus, efficiency techniques should be measured according to the usage of these resources.

Costs associated with PC usage are quite different. Assuming that most applications are run from a SAS Display Manager System session, the main concern with this platform is speed of execution. Memory constraints are also a consideration, and, by utilizing a network server, I/O operations become more important.

The next few sections of this paper simply list techniques that have been shown to be effective.

Programmer Time - A Big Saver

- Familiarity with your data saves time coding and re-work.
- Uses of the data will impact the design of an application as well as the elements in the data sets

- Design Before Coding
  - Design the flow of data through your application.
  - Flowcharting helps (really!)

- Develop and Follow Coding Conventions
  - Make your jobs easy to read
  - Place a descriptive header on each job
  - Code only one statement per line
  - Use lots of blank lines
  - Proper indentation
  - Terminate every DATA step and PROC with a RUN
  - Use lots of source code comments

- Define File References in your SAS Source Code
  - LIBNAME and FILENAME statements can create and allocate SAS libraries and external files, eliminating the need for host commands. This keeps job code in the SAS LOG, instead of a host job control log.

- FILE and INFILE statements allow the specification of the full external file name.

- Documentation
  - An important but often ignored part of the job. This is especially important if your code will be used or maintained by others, or if the code is tricky or infrequently executed.

- Develop Job Skeletons / Copy Files
  - Having source code files that are designed to be copied and modified saves you from re-typing often used code. These copy files should be expertly coded and documented, allowing you to quickly delete unneeded code, keeping only what is applicable to the task at hand.

For example, when beginning a new program, I may copy an MVS JCL skeleton, an input file specification for an external file, and a PROC TABULATE outline into the job I am building. This gives a bug-free foundation from which an application may be developed.

- Create a Set of Flexible Tools
  - Develop your own utility library to allow you to quickly and accurately perform routine tasks. SAS macros are especially useful for creating utilities.

Using SAS Effectively

- Know How SAS Works
  - Understand the macro resolution, compile, and execution phases of a SAS job (Henderson, et. al., 1991). Many inefficiencies result from a lack of understanding of the SAS System data management and procedural methods.

- Know SAS Capabilities
  - Familiarize yourself with SAS PROCS, functions, and OPTIONS.
  - Aside from experience, a good way to increase your SAS knowledge is to browse SAS manuals & SUGI Proceedings.
  - Get continual training for you and your staff.

- Use Character Variables
  - Numeric variables are much more CPU intensive than character variables. Only analysis variables need to be numeric.

- Use a LENGTH Statement
  - This not only saves space and I/O, but also CPU utilization as well when used to reduce storage space for numeric integer variables. Length of 4 bytes is sufficient for SAS dates while the minimum acceptable length is 2 bytes.

Consult the SAS Companion manual for your platform for a table showing the "Significant Digits and Largest Integer by Lengths".

Lengths for non-integer variables can also be shortened, but beware that precision can be lost if the length is too short. Also be careful when porting data between platforms.

- Use Formats and Informats
  - Save storage space.
  - Saves CPU when used to perform complex assignments.
Effectively use SAS Libraries
- Be familiar with concepts in Chapter 6 ("SAS Files") of the SAS Language manual.
- Explicitly defining the data engine on a LIBNAME statement saves CPU.

Keep the WORK library clean
- Use PROC DATASETS to delete unneeded data sets from the SAS WORK library. SAS works harder when reading or writing to a full SAS library.
- Minimize the number of DATA steps.
- Create multiple data sets with a single DATA step.
- Do not perform unnecessary sorting.

Use SAS OPTIONS effectively
OPTIONS OBS=10 ERRORABEND;
Some SAS options are good for use when developing and testing an application. The above options will prevent a job from using excessive resources during testing. OBS can be set to 0, but I prefer to pass a few observations so analyses can be performed and output generated.

OPTIONS BUFNO=8; /* I/O for memory tradeoff */
Other options are more specialized and can be used to improve the SAS System's performance with respect to various resources. The above option was especially useful at saving CPU and I/O on the mainframe. Tests on the PC with differing values showed no improvement over the default value of 1.

Pay attention to your SASLOG
- The SASLOG should be your best friend. Use it!
- Do not ignore SAS warnings. Understand why they are generated and implications for your processing.

Know Costs of SAS
- Don't be afraid to use non-SAS jobs. Forgive this blasphemy, but processing large external files may be best done with non-SAS software.

SAS DATA Step
- Read only the data that is needed
- Input only the fields you need from external files.
- Keep only variables you need from SAS data sets.

Know How the DATA Step is Processed
- Realize that the DATA step is a loop that is executed for each observation in a data set.
- Know the difference between executable and non-executable statements. Group each type together in your DATA steps. While this doesn't save resources per se, it shows that you have an understanding of SAS that allows you greater insight into efficient SAS DATA step programming.

Consider the SAS Program Data Vector (PDV)
- The PDV is a set of buffers that includes all variables referenced, either explicitly or implicitly, in a DATA step (Henderson, et. al., 1991). It is initialized at the top of the DATA step and then loaded with values for each observation processed.
- Note that since the WHERE = data set option and the WHERE statement will cause only desired observations to be loaded into the SAS Program Data Vector, any FIRST. and LAST. variables are set for only the observations that meet the WHERE condition. A subsetting IF statement would set the FIRST. and LAST. variables according to all observations in a data set.

Testing
- Use RUN CANCEL; for incremental testing. This will compile the code and check for syntax errors without executing a DATA step or procedure.
- Another technique for testing a SAS application involves using the SAS macro language to provide intermediate output for verification when an application is executed in the "test mode". For example:

```sas
%macro job(test = N)
  %if &test = Y %then OPTIONS MPRINT;;
  ....SAS code......
  %if &test = Y %then %do;
  TITLE3 'Intermediate data set...';
  PROC PRINT DATA = TEMP;
  RUN;
  %end;
  ....more SAS code.....
  %mend job;
  %job;
```

Pay attention to your SASLOG
- The SASLOG should be your best friend. Use it!
- Do not ignore SAS warnings. Understand why they are generated and implications for your processing.

Know Costs of SAS
- Don't be afraid to use non-SAS jobs. Forgive this blasphemy, but processing large external files may be best done with non-SAS software.

SAS DATA Step
- Read only the data that is needed
- Input only the fields you need from external files.
- Keep only variables you need from SAS data sets.

Know How the DATA Step is Processed
- Realize that the DATA step is a loop that is executed for each observation in a data set.
- Know the difference between executable and non-executable statements. Group each type together in your DATA steps. While this doesn't save resources per se, it shows that you have an understanding of SAS that allows you greater insight into efficient SAS DATA step programming.

Consider the SAS Program Data Vector (PDV)
- The PDV is a set of buffers that includes all variables referenced, either explicitly or implicitly, in a DATA step (Henderson, et. al., 1991). It is initialized at the top of the DATA step and then loaded with values for each observation processed.
- Note that since the WHERE = data set option and the WHERE statement will cause only desired observations to be loaded into the SAS Program Data Vector, any FIRST. and LAST. variables are set for only the observations that meet the WHERE condition. A subsetting IF statement would set the FIRST. and LAST. variables according to all observations in a data set.
When nesting loops, place the loop with the fewest iterations outermost. This can provide excellent CPU savings because SAS is jumping around less.

### Using arrays

It has been suggested (Hardy, 1992) that users avoid using SAS arrays because SAS requires much more CPU to process arrays than if each statement is explicitly coded. This is due to the fact that array indices are floating point numbers and array elements can be of varying length (Squillace, 1988).

My testing has shown that array efficiency differs by platform. Using arrays on the mainframe in Version 6.07 cost 30% to 100% more CPU than explicit coding, while the same code on the PC, in Version 6.04 as well as Version 6.06, cost only between 0% and 10% more in elapsed time.

Thus, while arrays still use more resources, they appear to be vastly more efficient on the PC platform.

Although arrays can be easier to code and read, I try to avoid their use on the mainframe. For larger data sets I often use macros to generate statements that would otherwise be processed using arrays.

### SAS PROCs

**Use PROCs for their intended purpose**

- PROC SUMMARY uses somewhat less CPU than PROC MEANS because PROC MEANS was designed to generate printed output while SUMMARY does not.

**Use SAS PROCs to process your data**

- Do not waste time writing a DATA step that replicates the actions of a PROC.
- The following PROCs are good data manipulators:
  - APPEND
  - COMPARE
  - COPY
  - DATASETS
  - EXPAND
  - SQL
  - SUMMARY
  - TRANSPOSE
- Use the CNTLIN = data set with PROC FORMAT.

**Use output data sets created by SAS PROCs**

- Most SAS PROCs produce an output data set that can be creatively used. Some of my favorites are:
  - CNTLOUT = data set with PROC FORMAT.
  - OUT = data set with PROC CONTENTS.
  - OUTEST = data sets on modeling PROCs

**Use the NOEQUALS option with PROC SORT**

This option specifies that observations with identical BY variables do not need to retain their same relative positions in the output data set as in the input data set. While the results were uneven on the mainframe Version 6.07, the NOEQUALS case never performed worse. On the PC Version 6.08, however, this option consistently gave good elapsed time savings.

**Use PROC SQL**

When processing SAS data sets, PROC SQL can perform multiple operations on multiple SAS data sets that would normally require the use of numerous other SAS DATA steps and procedures. This consolidation of multiple DATA steps and PROCs can be very efficient.

For example, to merge and summarize two data sets would require two PROC SORTs, a DATA step, and a PROC SUMMARY. These actions can all be accomplished with one PROC SQL with as much as a 50% reduction in CPU and elapsed time.

While PROC SQL is good at performing multiple operations on data sets, it is not generally efficient when performing a single function on a single data set.

Unfortunately, PROC SQL requires a fair amount of programmer time to learn due to the unfamiliar syntax (unless you know SQL). However, testing indicates that time spent learning this procedure would be a useful investment.

**Use the Pass-Through Facility in PROC SQL**

- When used to access a non-SAS data structure (DB2 table, ORACLE database, etc.), PROC SQL can be used to pass logic directly to a target database management system (DBMS), allowing the evaluation of the "subquery" to be done by the database system. This is extremely efficient because the data can be evaluated in its natural state, realizing the inherent efficiencies of the database system.

The following example shows a sample query using the SQL Pass-Through Facility to retrieve data from a DB2 table:

```
PROC SQL;
/* CONNECT TO establishes a connection to DB2 */
CONNECT TO DB2(SSID=DSN2);
SELECT *
/* CONNECTION TO retrieves data from DB2 */
FROM ADT.VMES_IUS_DATA
WHERE...... --
/* The following "subquery" is processed by DB2 */
SELECT *
/* DB2 Table */
FROM ADT.VMES_IUS_DATA
WHERE......
QUIT;
```

**Unfounded claims of efficiency**

The SAS Institute has published a guide of efficient coding techniques called "SAS Programming Tips: A Guide to Efficient SAS Processing". In this guide there are many techniques recommended for CPU savings in this SAS manual. In my testing of some of these techniques under Version 6.07, I have found many of these recommendations to be of no help for CPU savings.

Unfortunately, this manual was benchmarked by the SAS Institute using SAS Version 6.04. Beginning with SAS Version 6.07, an optimizing compiler was implemented. This new compile routine appears to have obviated many techniques recommended for CPU savings in this SAS manual. In my testing of some of these techniques under Version 6.07, I have found many of these recommendations to be of no help for CPU savings.

This does not in any way, however, diminish the usefulness of this manual as a source of efficient coding techniques.
Data Set Compressing

Compressing large data sets saves physical storage space and, therefore, I/O. In a compressed data set, SAS treats each observation as a single string of bytes and compresses multiple occurrences of the same byte without regard for variable boundaries. SAS Version 6 can create these variable-length observations only for direct access data sets (Engines V607 or V608).

Because of this compressing algorithm, data set compressing works best for data sets with a large record length (large # number of variables), especially those with lots of missing and/or repeated values. Space reduction can typically range from 5% to 35%.

Surprisingly, elapsed time on the PC decreased slightly for most operations when using compressed data sets. This was true whether the data being used was resident on the network server or on the PC's hard drive. Evidently, the PC file structure lends itself rather well to the SAS data compressing routine.

On the mainframe, however, a completely different story emerged. CPU usage increased by a substantial amount. Therefore, you should first gather information to aid in determining whether a mainframe data set is best stored as compressed or un-compressed. How often the data set will be processed or updated is a key consideration.

I use the following formula as a first guess to evaluate whether compressing a data set might save money:

\[
\text{(Monthly Space Charge)} \times \left( \frac{\% \text{ of Storage Reduction}}{\text{(Monthly CPU Charge to process the data set)}} \right) \times 3
\]

When this condition is true, the amount saved on the mainframe by compressing will likely be greater than the increase in CPU expense that will be incurred. The factor of 3 is somewhat high, as average overall CPU charges rarely increase by more than 300% when working with compressed data sets.

Again, this estimate is just a first guess. This is a difficult estimation because the overhead of compressing and decompressing depends on many factors, including the % reduction in space achieved, the proportion of numeric vs character variables, etc.

To create a compressed data set, use the COMPRESS= data set option when creating the data set. This option only needs to be specified when first creating the data set. For example:

```sas
DATA SASLIB.NEW_DS(COMPRESS=YES);
PROC SORT DATA=OLD OUT=NEW(COMPRESS=YES);
```

NOTE: Observations in a compressed data set cannot be accessed using the POINT= option or by observation number in SAS/FSP procedures.

Stored Program Facility

Use the Stored Program Facility to store compiled SAS DATA steps for large, frequently run jobs. "Stored Program Facility" is a bit of a misnomer since only single DATA steps may be stored. SAS PROCs cannot be stored as compiled code.

```sas
/* Compile and store a data step */
DATA NEW / PGM=SASLIB.STORED;
SET IN_DS;
KG = LBS *.4536;
...many more SAS data step statements...
RUN;

/* Execute a Stored Program */
DATA PGM=SASLIB.STORED;
RUN;
```

The optional REDIRECT statement is available when executing the stored DATA step to change the name of the input or output SAS data sets. Unfortunately, SAS data set options may not be passed via the REDIRECT statement.

```sas
/* Execute the stored DATA step */
DATA PGM=SASLIB.STORED;
REDIRECT INPUT NEW = NEWLIB.NEW_DS;
REDIRECT OUTPUT NEW = NEWLIB.NEW_OS;
RUN;
```

For a job with many DATA steps, each DATA step must be stored as a separate compiled member in a SAS library. Thus, early planning is advisable to implement this feature in an application due to the complexity of converting an application that contains many DATA steps.

This technique saves the most when large DATA steps are stored, especially when they are used to process small data sets. This is because the savings come from avoiding the compile time for the DATA step. This is usually a small proportion of a SAS job cost.

The DATA step used to test this feature contained 320 SAS DATA step statements and showed that there is very little overhead associated with this feature of the SAS System. Use of the Stored Program Facility, then, should be encouraged for the proper situations.

SAS Data Views

Use data VIEWS to avoid storing data elements in numerous data sets. A data view does not contain any data values. Instead, it simply contains a compiled DATA step program or SQL query that defines data or describes data that is stored somewhere else. Because of this, data cannot be modified in a view.

```sas
/* Create a DATA step VIEW */
/* The VIEW name must appear twice */
DATA SASLIB.CA_MALE; VIEW=SASLIB.CA_MALE;
SET SOCAL(WHERE=(SEX='M')) NORCAL(WHERE=(SEX='M'));
...other SAS data step statements....
RUN;
```
/* Create an SQL VIEW */
PROC SQL;
CREATE VIEW SASLIB.CA MALE AS ...
QUIT;

Views can also be created for external files. Once the view of the external file is created, jobs may reference the view, eliminating the need to know the file layout. For example:

DATA SASLIB.HOSP VIEW=SASLIB.HOSP;
INFILE 'HOSP.DOR.HOSP.YRS7992';
INPUT @1 ID $7.
@8 GENDER $1.
@9 DOB MMDYYYS.
......
AGE = (TODAY()-DOB)/365.25;
RUN;

This DATA step creates a view of an external file. When data is needed from this file in future jobs, it can simply be referenced by the SAS data view name SASLIB.HOSP.

Like the Stored Program Facility, code that defines a SAS data view does not execute. It is simply compiled and stored for future use. Once stored, the view is referenced like a SAS data set.

Each time the view is referenced it must execute the underlying compiled SAS code to create the data before performing any requested actions. So, for example, running a PROC PRINT on a SAS data view is like executing a DATA step (or SQL Query, depending upon the type of view) followed by a PROC PRINT.

Since this overhead is associated with a SAS view each time it is referenced, SAS data views are very costly if frequently used. If a view is going to be referenced more than once during a SAS session, you are well advised to create a SAS data set that contains the data described by the view and process the data set rather than the view.

Notes for Stored Program Facility & SAS Views
1) Be sure to keep a copy of the original, uncompiled, SAS code. Source code cannot be retrieved from the compiled SAS code.

2) A stored DATA step can take from 1.5 to 8 times more disk space than an uncompiled SAS program (Reynolds, 1992).

3) Global statements (TITLE, FOOTNOTE, FILENAME, LIBNAME, OPTIONS) and most host-specific options and host-specific data set options cannot be included in the stored DATA step. Macro code cannot be stored either, since the stored code is compiled SAS code.

Conclusion
When it comes to efficiently coding in SAS, it is important to start with the basics. Although the SAS System has many appealing features, many of these pleasantries come with a price tag in terms of CPU usage. Therefore you should use these bells & whistles only for specialized circumstances where their use is warranted.

My testing has found that efficient coding techniques are rarely useful with data sets containing fewer than 1,000 observations on a mainframe.

On the PC side, I was pleasantly surprised at the speed of Version 6.08 over the previous PC Version 6.04. However, this increased speed will only materialize on PC's with ample resources (486, math coprocessor, 8 megabytes memory).

Finally, remember that while one technique may be very efficient on one platform there is no guarantee that it will work as well on another platform. As shown, different results are obtained depending upon platform.

References


Author's Address
Steve Wilson
Kaiser Permanente, Division of Research
3451 Piedmont Avenue
Oakland, California 94611-5400
(510) 450-2211

SAS and SAS/FSP are registered trademarks of the SAS Institute Inc., Cary, NC, USA

SQL and DB2 are registered trademarks of International Business Machines Corporation.

Oracle is a registered trademark of Oracle Corporation.