MORE EFFICIENT USE OF SAS

(OR THE EFFECTS OF INDULGING SAS)

Robert L. Anderson

Deering Milliken, Inc.

MORE EFFICIENT USE OF SAS

The purpose of this paper is not to expose any previously
unknown or highly sophisticated techniques for making more efficient
use of SAS. The author's intent is to discuss a collection of
very simple methods he uses for much greater efficiency. Hopefully,
this will work as a catalyst to stimulate discussion and thinking
among other users so that the total effect will be great savings to all.

Many users of SAS are not systems analysts or even programmers
and are not aware of the effect that various uses (or abuses) of SAS
have on an Operating System. I have heard complaints from many of
my associates in academia, government, and in business about the
inefficiencies of SAS. SAS is not inefficient at doing what it is
designed to do and with proper caution it can be used as a valuable
tool for many non-statistical endeavors. Hopefully, through more
consciousness in our use of SAS we can generate a much wider
acceptance of it as the valuable tool that it is.

My remarks will be directed at some simple ways of reducing
run time, conserving on line storage, and reducing abuse of the CPU.
Since I work in a business environment, my examples will be most
directly applicable to business situations.
A. PRE-EDITING INPUT DATA

One of the greatest abuses of SAS is in its use to edit large data files to obtain a relatively small amount of data. For example, in my environment we quite frequently wish to draw on information from some large database or to select a small amount of data for analysis from a large file. Now the INPUT facility in SAS is stream (edit) oriented and, as such, is very slow at reading a large data file. A strip program in PL/I or COBOL could quickly and efficiently edit the data for input to SAS.

In the appendix an example is given of a typical strip program (in PL/I) that I use for pre-editing data. As an example of savings, suppose we wish to look at a quality file that contains about 200,000 records (each 250 bytes long). We would like to do some quality analysis on a particular product, PROD = 'XXXXXX'. There are about 1,000 records in the file containing quality information relative to PROD. A typical abuse of SAS would be to select the data for analysis by use of the following statements:

```
DATA QUAL;
INPUT DDNAME=QUALITY PROD $4-9 ...;
IF PROD='XXXXXX';
```

This would achieve our objective of creating an input data set named QUAL containing only those 1,000 records pertaining to the product 'XXXXXX'. The run time on an IBM 370 under VS at our location would be approximately 60-80 minutes. If the system should be heavily loaded and cycling times are long, this could be 2-3 hours. If the simple strip program in the appendix (record oriented transmission) were used to strip the data for SAS, the total run time would be 10-15 minutes. Such a simple strip program can be prepared in a few minutes by anyone and can be used over and over again with simple modifications.
B. KEEPING EDITED DATA

Another real savings comes from keeping (or storing) the edited data until our analysis is finished. Very seldom is one finished with the analysis after a first try. To avoid spending valuable hours restriping data every time one makes another run, it is advisable to store the edited data on a disc pack, tape, or some other convenient place until the analysis is finished. Even if the original data is on cards, it is a good idea to store the data on some other convenient device rather than to run the risk of losing or misplacing cards.

Pre-editing and storing data not only saves valuable hours (and $) on computer time but there are many additional benefits. Shorter jobs usually mean much faster turnaround. The personnel at your computer location are less likely to post a "bad" tag on your jobs and curse you and SAS. Eventually they can be trained to trust a SAS job as just another run.

C. REPETITION OF STEPS

One point at which many analysts are wasteful is in the repetition of previously successful steps. Suppose an analysis consists of three independent steps. If the first two steps are successful but the third step fails, then only the third step should be resubmitted. Many people would rather use an additional thirty minutes of computer time (to keep their card deck intact) than to spend one minute removing a few cards which could easily be replaced later. A good rule of thumb in making a decision in this case is to ask oneself, "If I were paying for the computer time from my pocket, what would I do?"
D. CONSERVATION OF ON-LINE STORAGE

Conservation of on-line storage is more important than most users realize. It can actually determine whether a job will run. There are two ways to achieve this savings.

First, SAS has a "built-in" facility for conserving space. Although many users are aware of this, it is still worth mentioning here.

Within a given program it is often necessary to modify a data set several times. For example, we often see the following sequence of statements:

```
DATA BILL1; INPUT ...;....
DATA BILL2; SET BILL1;....
```

This sequence of statements creates two separate data sets BILL1 and BILL2. Often the program can be organized so that when BILL2 is created we are finished with BILL1. In that case we can replace BILL1 by BILL2 and save only the second data set using the following statements:

```
DATA BILL1; INPUT ...;....
DATA BILL1; SET BILL1;....
```

Note that the second set is named BILL1 so that it replaces the old BILL1.

In some cases use of multiple job steps can be helpful in SAS. These can be used most effectively in long SAS runs if the job can be separated easily into individual steps where core requirements are different and where there are a large number of data sets to be handled. Consider the following:

```
//STEP1 EXEC SAS
DATA FILE1; INPUT ...;....
.
.
DATA FILE2; INPUT ...;....
```
Suppose that the procedures after the statement DATA FILE2 require only a region size of 120K whereas the core requirements of the earlier procedures is 200K. Then we could separate the job into two steps as follows:

```sas
//STEP1 EXEC SAS, REGION=200K
DATA FILE1;
.
.
//STEP2 EXEC SAS, REGION=120K
DATA FILE2;
```

Note this also frees the space occupied by the files used in STEP1 when we go to STEP2.

Quite frequently, on-line storage requirements can determine whether a job will run in a given location. A job that ties up a lot of on-line storage or a lot of tape drives for a long period of time endangers one's friendship with personnel in the computer department as well as robs other users of the system. Organizing a job to conserve on-line storage requirements also provides for more simplicity in the program and helps avoid confusion when we look at the program at some later time.

E. WRAP-UP

Being frugal in our use of the SAS leads to an obvious valuable savings in computer requirements. Large core requirements for a short period of time, for example, are much less noticeable than for long periods of time. (Similarly for large amounts of on-line storage.) The real benefit is that such savings can permit fast turnaround on a job that might otherwise be totally infeasible.
APPENDIX

This is an example of a short PL/I strip program that could be used for stripping all records of a given type, PROD='XXXXXX' from a large file containing 250 byte records. PROD is a six character product description beginning in position four on the record.

```
STRIP: PROC OPTIONS (MAIN);
DCL
  RECORD CHAR(250), PROD CHAR(6) DEFINED RECORD POS(4),
  (INPUT, OUTPUT) RECORD SEQL FILE, NO_OUT FIXED BIN(31) INIT(0),
  EOF BIT(1) INIT('0'B); /*END OF FILE SWITCH*/
ON ENDFILE(INPUT) EOF='1'B;
OPEN FILE(INPUT) INPUT, FILE(OUTPUT) OUTPUT;
READ FILE(INPUT) INTO (RECORD);
DO WHILE (~EOF);
  IF PROD='XXXXXX' THEN DO;
    WRITE FILE(OUTPUT) FROM (RECORD);
    NO_OUT=NO_OUT + 1; END;
  READ FILE(INPUT) INTO (RECORD);
END;
PUT SKIP LIST ('NO OF RECORDS OUT=', NO_OUT);
END STRIP;
```