PharmaSUG China 2016 - Paper 68

Big Data Processing Techniques

Eduard Joseph Siquioco, PPD, Manila, Philippines

ABSTRACT

Oftentimes programmers will encounter big data; these data comes with many observations and variables. These are mostly used in conjunction with complex programming tasks and will result in long run times and programming. Different SAS functions, options, and techniques can help in reducing the run time when used correctly. This paper will explore the basic techniques like using SAS options and go in-depth with beginner friendly techniques that programmers most often times overlook.

INTRODUCTION

The industry has increased the enrollment of patients thru the help of technology. In turn, there would be more data to process and more complex derivations that make the dataset processing longer. These run times of these datasets would cause a loss in revenue for the companies and longer production times.

ENVIRONMENT SETUP

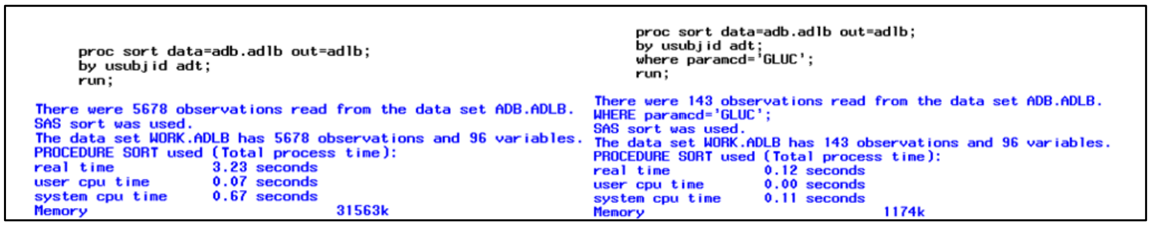
1. Close all applications that is not SAS
   Other applications also use the system's resources. If there are applications that use these resources heavily SAS will have less than what should be available, leading to slower processing speed.

2. Use SAS Options to make processing faster
   ```
   options compress=Y;
   ```
   The compress option stores the SAS datasets in compressed forms. This option is not recommended to small amount of SAS datasets as there is additional CPU time needed to decompress the dataset. Alternatively this options can be used as a dataset option for greater control

3. Remove macro processing displays.
   ```
   options nomlogic nomprint nomrecall nosymbolgen;
   ```
   After completely debugging the macro codes, the options stated above should be applied to turn off the printing of irrelevant information to the log for production. Looping macro runs with these options turned on will run much slower because SAS has to write statements repeatedly .log file – this will clog up the disk write process. If the file becomes large enough it might also crash the program.

EFFICIENT CODING - STATEMENTS

1. Using WHERE statements: The where options saves resources by eliminating the number of records processed. In an example shown below, the programmer wants to sort the dataset by Subject ID and Analysis date but only needs the records for a specific parameter code.



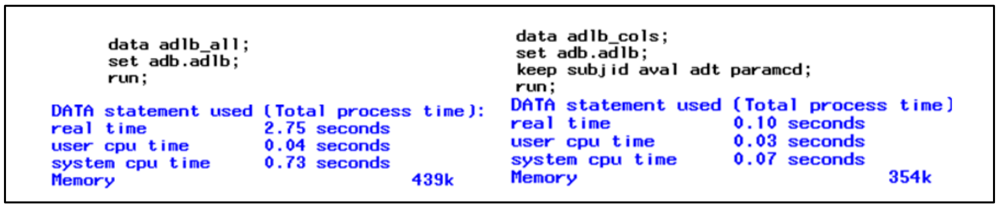
```
     proc sort data=adb.adlb out=adlb;
     by usubjid adt;
     run;

There were 5678 observations read from the data set ADB.ADLB.
SAS sort was used.
The data set WORK.ADLB has 5678 observations and 96 variables.
PROCEDURE SORT used (Total process time):
real time           3.23 seconds
user cpu time       0.07 seconds
system cpu time     0.67 seconds
Memory                               31563k
```

```
     proc sort data=adb.adlb out=adlb;
     by usubjid adt;
     where paramcd='GLUC';
     run;

There were 143 observations read from the data set ADB.ADLB.
WHERE paramcd='GLUC';
SAS sort was used.
The data set WORK.ADLB has 143 observations and 96 variables.
PROCEDURE SORT used (Total process time):
real time           0.12 seconds
user cpu time       0.00 seconds
system cpu time     0.11 seconds
Memory                                1174k
```

Display 1: CPU Time Difference for Where statement

Notice that there is a difference in the Memory usage and CPU time of both runs. The code with the where statement ran much faster and used less memory than its counterpart. It is best to apply the where statement as early as possible in the code before further processing is made.

2. DROP KEEP statements: The drop and keep statements can save resources by dropping the unnecessary variables for further processing. These statements also lowers the consumed space of your dataset.

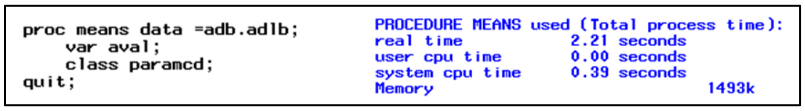


```
        data adlb_all;
        set adb.adlb;
        run;
DATA statement used (Total process time):
real time              2.75 seconds
user cpu time          0.04 seconds
system cpu time        0.73 seconds
Memory                                   439k
```

```
 data adlb_cols;
 set adb.adlb;
 keep subjid aval adt paramcd;
 run;
DATA statement used (Total process time)
real time              0.10 seconds
user cpu time          0.03 seconds
system cpu time        0.07 seconds
Memory                                   354k
```

Display 2: CPU Time Difference for Drop/Keep Statement

3. Class Statements: The Class statements can save resources by eliminating the need to create staging datasets to feed in the proc statement. Although not all procs are compatible with the by, it is worth nothing that the proc means statements is compatible and widely used in these scenarios



```
proc means data =adb.adlb;
     var aval;
     class paramcd;
quit;
```

```
PROCEDURE MEANS used (Total process time):
real time              2.21 seconds
user cpu time          0.00 seconds
system cpu time        0.39 seconds
Memory                                 1493k
```

Display 3: CPU Time of Proc Means with a class statement

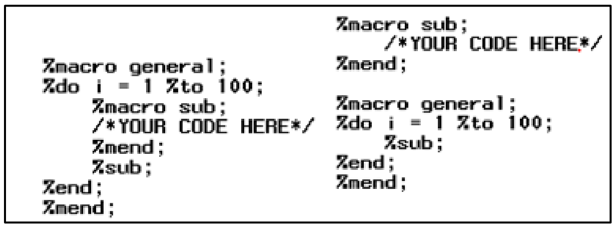
In display 1, it is shown that creating a subset of the data of a PARAMCD would require .11 seconds of CPU time, while the *proc means* with a class statement only runs .39 seconds of CPU time. The processing it takes and effort to code different data subsets is not efficient in these cases. Programmers must know when the class statement is available to the proc.

EFFICIENT CODING – TECHNIQUES

1. Create a subset of the large dataset with only the required observations and variables needed
   Reading the large source dataset multiple times would require more time for the code to run, by creating the data subset we eliminate this step and proceed with our further processing using only this data set with subset.

   If there is a need to create multiple datasets from one source, programmers should invoke the source dataset only once and use explicit output statements.

2. Macro compilation independence
   Macros within macros should be compiled independently of each other so the macro compilation will only occur once and not every loop of the macro. The left side panel of Display 4 shows a code that will compile the macro %sub for each loop of the macro %general. The right side panel shows the correct way of compiling macros, the macro %sub will not be recompiled each time the loop of the macro %general is invoked.



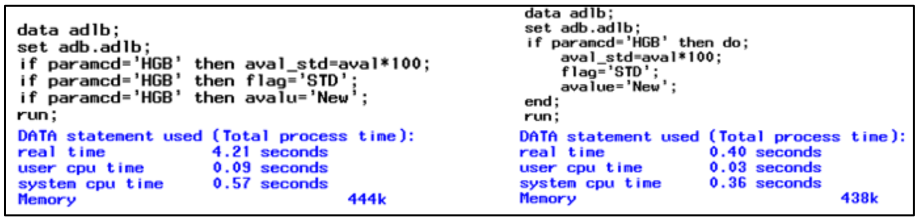
```
%macro general;
%do i = 1 %to 100;
     %macro sub;
     /*YOUR CODE HERE*/
     %mend;
     %sub;
%end;
%mend;
```

```
%macro sub;
     /*YOUR CODE HERE*/
%mend;

%macro general;
%do i = 1 %to 100;
     %sub;
%end;
%mend;
```

Display 4: Inefficient Macro compilation vs Efficient Macro compilation

3. *If then do* nests in data steps
   When a set of functions/derivation/computation needs only to be applied in specific observations, it would be more efficient to place them in a Do nest. The nest would only process codes if the observation satisfies the condition specified by the programmer.

   Placing these if statements on each processing code will require SAS to process if statement repeatedly for all the function in the supposed to be nested code.

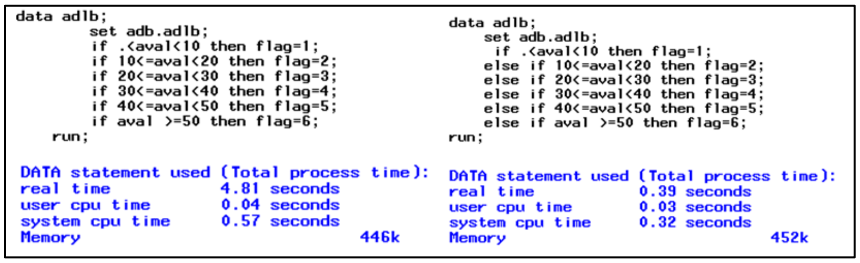


```
data adlb;
set adb.adlb;
if paramcd='HGB' then aval_std=aval*100;
if paramcd='HGB' then flag='STD';
if paramcd='HGB' then avalu='New';
run;
DATA statement used (Total process time):
real time           4.21 seconds
user cpu time       0.09 seconds
system cpu time     0.57 seconds
Memory                             444k
```

```
data adlb;
set adb.adlb;
if paramcd='HGB' then do;
    aval_std=aval*100;
    flag='STD';
    avalue='New';
end;
run;
DATA statement used (Total process time):
real time           0.40 seconds
user cpu time       0.03 seconds
system cpu time     0.36 seconds
Memory                             438k
```

Display 5: CPU Time difference for nested code

4. *Else if statements*
   Using the else if statement eliminates the processing of observations which has passed the condition of the previous if functions.

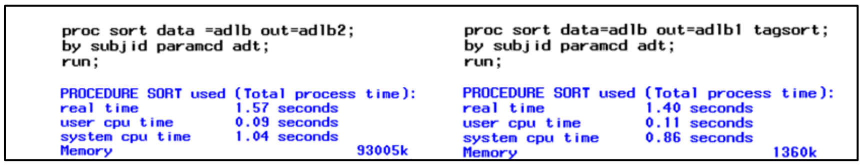


```
data adlb;
        set adb.adlb;
        if .<aval<10 then flag=1;
        if 10<=aval<20 then flag=2;
        if 20<=aval<30 then flag=3;
        if 30<=aval<40 then flag=4;
        if 40<=aval<50 then flag=5;
        if aval >=50 then flag=6;
    run;

DATA statement used (Total process time):
real time           4.81 seconds
user cpu time       0.04 seconds
system cpu time     0.57 seconds
Memory                             446k
```

```
data adlb;
    set adb.adlb;
     if .<aval<10 then flag=1;
    else if 10<=aval<20 then flag=2;
    else if 20<=aval<30 then flag=3;
    else if 30<=aval<40 then flag=4;
    else if 40<=aval<50 then flag=5;
    else if aval >=50 then flag=6;
run;

DATA statement used (Total process time):
real time           0.39 seconds
user cpu time       0.03 seconds
system cpu time     0.32 seconds
Memory                             452k
```

Display 6: CPU Time difference of Else if Statement

5. *Sort with tagsort option*
   The tagsort option works differently and uses significantly less disk space at the cost of CPU Time. Time savings for very large datasets can go up to 15 minutes or more.



```
proc sort data =adlb out=adlb2;
by subjid paramcd adt;
run;

PROCEDURE SORT used (Total process time):
real time           1.57 seconds
user cpu time       0.09 seconds
system cpu time     1.04 seconds
Memory                             93005k
```

```
proc sort data=adlb out=adlb1 tagsort;
by subjid paramcd adt;
run;

PROCEDURE SORT used (Total process time):
real time           1.40 seconds
user cpu time       0.11 seconds
system cpu time     0.86 seconds
Memory                             1360k
```

Display 7: CPU Time difference for Tagsort Sorting

CONCLUSION

It is very important to always code in efficiently. This would ease the production process of the datasets and will cause more revenue for the company. Coding efficiently should always be accompanied with programmer's comments in order to ease in debugging if another programmer will take over. This should be instilled as early as we can, i.e. in the training process so it will become natural for the programmer.

REFERENCES

SAS 9.2 Online product documentation. Available at http://support.sas.com/documentation/92/index.html

Lafler, Kirk Paul, 2012, "Top Ten SAS® Performance Tuning Techniques." Available at http://support.sas.com/resources/papers/proceedings12/357-2012.pdf

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Eduard Joseph Siquioco
Enterprise: PPD, Inc.
Address: 22nd Floor Net Park Building, 5th Avenue, Bonifacio Global City
City, ZIP: Taguig City, Metro Manila 1634
Work Phone: +632 464 7675
E-mail: EduardJoseph.Siquioco@ppdi.com
Web: www.ppdi.com