

Paper PO04

Speed up your SAS® Programs

Karthikeyan Chidambaram
Cognizant Technology Solutions, Newbury Park, CA

ABSTRACT

There are some basic rules, which will enhance the efficiency of the SAS programs. This Paper will attempt to list out the rules and techniques for writing Efficient SAS programs.

INTRODUCTION

I/O is one of the most important factors for optimizing performance. Most SAS jobs consist of repeated cycles of reading a particular set of data to perform various data analysis and data manipulation tasks. To improve the performance of a SAS job, we must reduce the number of times SAS accesses disk or tape devices.

We can reduce the number of data accesses by processing more data each time a device is accessed by setting the BUFNO=, BUFSIZE=, CATCACHE=, and COMPRESS= system options. Sometimes we might be able to use more than one method, making our SAS job even more efficient. This Paper focuses on some rules that will facilitate efficient coding, thereby increasing efficiency and productivity.

RULE1. READ ONLY THE VARIABLES THAT ARE USED:

SAS program data vector allocates buffer space based on the number of variables that are being read in and the number of variables that are created during the data step processing. Hence, if we are using 4 variables, out of 10 from a dataset, the keep statement at the set statement is more efficient than the keep statement at the end of the program. This is because, the keep option, when used with the set statement, avoids reading in the unwanted columns on to the buffer.

Less Efficient Code:

```
DATA sample;
  SET source;
  Other SAS Statements...
  keep var1 var2 var3;
RUN;
```

Efficient Code:

```
DATA sample;
  SET source (keep = var1 var2 var3);
  Other SAS Statements...
RUN;
```

RULE 2. AVOID UN-WANTED STEPS:

In many of our programs, we tend to create multiple steps, thereby breaking the processing into multiple steps. For instance, when we have to write a code for printing the top 10,000 observations in a dataset, we can break the dataset into a smaller one first using the data step and then print is using the PROC PRINT. Though this approach works, there is huge trade-off in terms of efficiency. We can cut the processing time by 50%, by removing the data step and adding an "obs" option to the PROC PRINT. The same has been illustrated below.

Less Efficient Code:

```
DATA topprint;
  SET source;
  If _N_ le 10000;
RUN;
```

```
PROC PRINT data = toprint;
    Title "Top 10,000 Records";
Run;
```

Efficient Code:

```
PROC PRINT data = source (obs=10000);
    Title "Top 10,000 Records";
Run;
```

RULE 3. DATA SUBSET – IF AND WHERE STATEMENTS:

Both “if” and “where” statements can be used to subset a dataset based on the specified criteria. Though both if and where statements produce the exact same results, they have a big difference in the way they operate on the data. In case of the “if” statement, the data is read into the program data vector before the condition is verified. Thus all the records are read into the program data vector irrespective of their value and the criteria. On the contrary, the where statement checks for the criteria, even before the data is read into the PDV. Hence, the unwanted data records are not read in to the buffer space at all. Thus the “Where” statement will be a better option for data subset, especially in case of datasets with a large number of variables.

Less Efficient Code:

```
DATA subst;
    SET source;
    If sales > 1000;
RUN;
```

Efficient Code:

```
DATA subst;
    SET source;
    Where sales > 1000;
RUN;
```

RULE 4. USE THE SPECIFIC PROCEDURES FOR THE JOB:

Using the specialized procedures, in place of the data step, will improve the efficiency in most of the cases. If we consider the case of appending 2 datasets together, the simplest code would be:

```
DATA dset1;
    SET dset1 dset2;
RUN;
```

In this case, both the datasets are read into the PDV, before the final dataset is created. At the same time, if we look at the APPEND procedure for the same job, only the dataset # 2, which is in the “append=” will be read into the buffer. Hence, if we use larger dataset in the “base=” option, we can cut the processing time by more than 50%.

Efficient code for data appending:

```
PROC APPEND BASE=dset1 APPEND=dset2;
RUN;
```

RULE 5. RIGHT USE OF THE “IF/THEN ELSE” STATEMENTS:

One of the key contributors to the power of SAS data step are the conditional statements like “If else” statements. At the same time, in correct usage of these statements, can have a heavy toll on the efficiency. Instead of using multiple if statements, they can be combined together to form “if –else” blocks. This will reduce the number of times, the “if” loop is being iterated. For example let us consider the following code:

```

DATA outdata;
  SET inpdata;
  If sales ge 0 and sales le 1000 then
      SAS statements...
  If sales ge 1000 and sales le 10000 then
      SAS Statements...
  If sales ge 10000 then
      SAS Statements...
RUN;

```

In this case, the SAS program processes all the 3 “if” statements for each iteration of the data step. The number of “if” statement processing can be reduced considerably by coupling the individual if statements to a single “if then else” block. In addition, the “IF” blocks need to be arranged in the descending order of the probability being true. This will further reduce the time taken in the “if then else” processing.

Efficient Code:

```

DATA outdata;
  SET inpdata;
  If sales ge 0 and sales le 1000 then
      SAS statements...
  Else If sales ge 1000 and sales le 10000 then
      SAS Statements...
  Else If sales ge 10000 then
      SAS Statements...
RUN;

```

RULE 6. USE SAS FUNCTIONS INSTEAD OF EXPRESSIONS:

The SAS functions have been designed to perform the specific task and are more faster and sophisticated than the raw expressions. For instance, if we consider the following code, with a simple sum operation:

```

DATA outdata;
  SET inpdata;
  Total = Salary + Incentive;
Run;

```

Though the program runs fine, the problem arises when we have missing value either for Salary or for the Incentive column. SAS processing takes time to record the missing value and also assigns a missing value for total for that particular record. At the same time, if we use the following code, with a “SUM” function, it will execute faster and will perform summation of the non-missing values.

```

DATA outdata;
  SET inpdata;
  Total = sum(Salary , Incentive);
RUN;

```

RULE 7. REMOVE TEMPORARY DATASETS:

The temporary datasets should be removed as soon as they are used. This clears up a lot of space in the work directory and this could be critical when we process huge datasets. The “PROC DATASETS” can be used to accomplish this.

Sample code for removing all datasets in work directory:

```

PROC DATASETS LIBRARY = work;
  DELETE _all_;
RUN;

```

RULE 8. JUDICIAL USE OF VARIABLE LENGTHS:

Unlike relational databases, SAS does not have variable types that will allocate smart memory based on the value stored. Hence, whatever is the length defined will be allocated to the variable. In fact, most of us believe that, if we do not specify the length, we are doing well, since SAS will allocate the length by itself. In some cases, this approach may have a major impact. For instance, while we create GENDER column in a data step, if SAS allocates the length it will allocate 8. But the real value of the variable occupies only one position. Hence, there is wastage of 88% space for the gender column for all the records in the dataset.

Less-Efficient Code:

```
DATA employee;
    SET hr_data;
    Gender = substr(gender_desc,1,1);
RUN;
```

Efficient Code:

```
DATA employee;
    LENGTH Gender $1;
    SET hr_data;
    Gender = substr(gender_desc,1,1);
RUN;
```

RULE 9. OPTIMIZING THE USAGE OF SORT PROCEDURE:

The use of the sort procedure must be carefully studied. If the program requires processing at multiple levels, they must be grouped together, so that a single sort takes care of many of them. For instance, if we need to do some analysis on variable "Segment" first, and then by "Segment " and "District" then, the initial sort procedure has to be on "Segment District", so that the sort for the second analysis can be avoided.

Less Efficient Code:

```
PROC SORT DATA = sales;
    BY segment;
RUN;

DATA out1;
    MERGE sales (in=a) segmentinfo(in=b);
    BY segment;
RUN;

PROC SORT DATA = sales;
    BY segment district;
RUN;

DATA out1;
    MERGE sales (in=a) districtinfo(in=b);
    BY segment district;
RUN;
```

Efficient Code:

```
PROC SORT DATA = sales;
    BY segment district;
RUN;

DATA out1;
    MERGE sales (in=a) segmentinfo(in=b);
    BY segment;
RUN;
```

```
DATA out1;
    MERGE sales (in=a) districtinfo(in=b);
    BY segment district;
RUN;
```

In addition, for many procedures, the sorting can be avoided totally by using the "CLASS" statement in place of the "BY" statement.

RULE 10. USE RETAIN STATEMENTS FOR VARIABLE ASSIGNMENTS:

Retain statement should be used to assign values to the variables. The prime advantage of the Retain statement is that, it does the assignment operation only at the first iteration of the data step. On the contrary, using the assignment operators will result in assignment operation for every iteration of the data step.

Less Efficient Code:

```
DATA outdata;
    SET indata;
    Euro = 1.5;
RUN;
```

Efficient Code:

```
DATA outdata;
    SET indata;
    Retain Euro 1.5;
RUN;
```

CONCLUSION

This paper has highlighted the basic & common rules in writing efficient SAS programs. I hope that this serves as a good starting point for those, who wish to write efficient SAS programs.

REFERENCES

SAS Online Help

ACKNOWLEDGMENTS

The Author would like to thank his family, friends, peers and supervisors for their encouragement, support and suggestions.

CONTACT INFORMATION

Karthikeyan Chidambaram - SAS certified professional, has over 6 years of experience in SAS in a variety of roles including SAS Administration, Statistical Analysis and ETL programming. Your comments and questions are valued and encouraged. Contact the author at:

Karthikeyan Chidambaram
Cognizant Technology Solutions
1890, W Hillcrest Dr, 589
Newbury Park, CA - 91320

Phone: 805-300-0505

Email: karthihere@hotmail.com
ckarthikeyan@email.cognizant.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® Indicates USA registration.

Other brand and product names are trademarks of their respective companies.