SAS Programming Efficiency: Tips, Examples, and PROC GINSIDE Optimization

Lingqun Liu, University of Michigan

MISUG, Feb 2018

1

Outline

- This paper first explores the concepts of efficiency.
- Then reviews some relevant materials and tips available online.
- Examples of efficient programming.
- PROC GINSIDE optimization.

2

Efficiency: Concepts

- Computer resources
- Human resources
- SAS OPTIONS: STIMER, FULLSTIMER
- Time, time, time:
  - Less I/O time,
  - Less CPU time,
  - Less human time
- Two principals/strategies: Do the right thing, and do it right.

Programming Efficiency Tips

- Google search for "SAS efficiency"
- Presentations at MSUG meetings

"Leave Your Bad Code Behind: 50 Ways to Make Your SAS Code Execute More Efficiently",

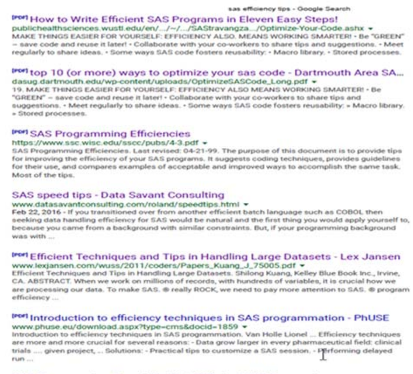
by William Benjamin, June 2017 one-day conference.

"SAS Advanced Programming with Efficiency in Mind: A Real Case Study",

by Lingqun Liu, Feb 2017 meeting.

"Quick Hits - My Favorite SAS Tricks", by Marje Fecht, May 2013 one-day conference.

"Utilizing SAS for Efficient Coding", by Michelle Gayari, November 2009 meeting.



Programming Efficiency Tips

New Tips

1) Use array to reduce coding

```
...
set sample;
array vars {3} vara varb varc;
do i=1 to 3;
        if vars[i]>1 then do;
                A= var[i]*amount*cmmisions;
                ...
end;
...
```

2) Use IF ELSE to avoid wasting extra CPU time

```
...
set sample;
if varc>1 then do; …  end;
else if varb>1 then do; … end;
else If vara>1 then do; …  end;
...
```

3) Use temp variable to reduce coding

```
...
set sample;
if varc > 1 then _temp = varc;
else if varb>1 then _temp = varb;
else if vara>1 then _temp=vara;
if _temp>1 then do;
        ...
end;
...
```

4) Use temporary variable and IFN() function to reduce coding

```
...
temp=ifn(varc>1,varc,ifn(varb>1,varb,ifn(vara>1,vara,.))); ...
```

Original tips #16-#17 presented at MISUG June 2017 one-day conference

#	This works:	This is uses less code:
16	**Computers are good at doing the same thing over and over again. But, programmers do not like to code the same thing over and over again. Take this code for an example.** Data test; Set sample; If vara > 1 then do; A = vara * amount * commission; B = A / 2; C = A + B; End; If varb > 1 then do; A = varb * amount * commission; B = A / 2; C = A + B; End; If varc > 1 then do; A = varc * amount * commission; B = A / 2; C = A + B; End; Run;	**One way to rewrite the code on the left is to use subroutines. The following is one example.** Data test; Set sample; Temp =vara; If vara >1 then link test; Temp =varb; If varb >1 then link test; Temp =varc; If varc >1 then link test; Return; * end of the main datastep; * this subroutine code is executed three times; Test: A = Temp * amount * commission; B = A / 2; C = A + B; Return; * end of the subroutine; Run;
17	**This is the same code as the previous example, but a second solution is presented that makes the code**	**The use of a macro to do the same task over and over again works well too.**

Programming Efficiency Tips

New tips

1) Use built-in function REPEAT() to simplify the code.

```
flag=repeat('0',31);
```

1) Use array to simplify the code

```
array conds {31}  cond_1 – cond_31;
do i=31 to 1 by -1;
        if conds[i] = i then substr (flag,32-i,1) ='1';
end;
```

1) Use LENGTH statement and CATS() to simplify the code

```
length flag $31;
array conds {31}  cond_1 – cond_31;
do i=31 to 1 by -1;
        flag = cats(flag,conds[i]=i);;
end;
```

or

```
do i=1 to 31;
    flag=cats(conds[i]=i,flag);
end;
```

What if the conditions `conds[i]=i are` changed and there is no pattern?

A TEMPORARY array will do the trick.

```
array _cs {31} _temporary_ (1 2 3 …. 4 5 6);
do i=1 to 31;
    flag=cats(conds[i]=_cs[i],flag);
end;
```

Original tip #48 presented at MISUG June 2017 one-day conference

```
Data Master_file;

   flag = '0000000000000000000000000000000';   * 31 zeros;

   Set my_sas_file;

   If (cond_1 = 1) then substr(flag,31,1) = '1';
   If (cond_2 = 2) then substr(flag,30,1) = '1';
   If (cond_3 = 3) then substr(flag,29,1) = '1';

   *. . . more conditions . . . ;

   If (cond_29 = 4) then substr(flag,3,1) = '1';
   If (cond_30 = 5) then substr(flag,2,1) = '1';
   If (cond_31 = 6) then substr(flag,1,1) = '1';


   Conditions_flag = input(flag,ib4.) ; * convert the flags to a real numeric variable;
                                        * IB4. informat is Integer Binary for 4 bytes.;
                                        * It knows there were 31 binary digits not 32;
 run;

 proc sort data = Master_file
             (where=(Conditions_flag=input('1010000000000000000000000000010',ib4.)))
         Out = New_subset_file;
 by key;
 run;
```

Programming Efficiency Examples

- How to check missing values of all variables in a data set
- How to identify the new or changed records
- How to identify common variables in multiple data sets
- Use the right SAS built-in functions

7

Programming Efficiency Examples

- Check missing values of all variables in a data set

```
proc format;
        value $miss
        '',' '='c_missing'
        other='c_non-missing'
        ;
        value miss
        .='n_missing'
        other='n_non-missing'
        ;
run;
```

```
ods output OneWayFreqs = _checking_missing_
     (keep=table frequency percent cumfreq: cumpercent f_:);
proc freq data= _test_;
     table _all_/missing;
     format _numeric_ miss. _CHARACTER_  $miss.;
run;
ods output close;

data _missing_;
     length table $32;
     set _checking_missing_ ;
     length formated $30;
     formated = cats(of f_:);
     table=substr(table,7);
     drop f_:;
run;
```

8

Programming Efficiency Examples

- Check all variables in a data set w/o using %macro loop

Similarly, this technique can be used to summarize all numeric variables in a data set.

```
proc means data=_test_
noprint;
    var _numeric_;
    output
out=_all_mean_ n=
nmiss= mean=/autoname;
run;
```

Create Frequency table for all variables in a data set.

```
ods output OneWayFreqs = _freq_all_ (keep=table
frequency percent cumfreq: cumperc: F_:);
proc freq data=sashelp.class;
     table _all_/missing;
run;
ods output close;

data freq_all (keep = varname value freq: cum:);
     set _freq_all_ (rename = (table=varname) );
     value=cats( of F_:);
     varname=substr(varname,7);
run;
```


Programming Efficiency Examples

- Check missing values of all variables in a data set

	Name	Sex	Age	Height	Weight	var	cvar
1	Alfred	M	14	69	.	.	
2		F	13	56.5	84	.	
3		F	13	65.3	98	.	
4	Carol	F	14	62.8	.	.	
5	Henry	M	14	63.5	.	.	
6	James	M	12	57.3	83	.	
7	Jane	F	12	59.8	84.5	.	
8	Janet	F	15	62.5	112.5	.	
9		M	13	62.5	84	.	
10	John	M	12	59	99.5	.	
11	Joyce	F	11	51.3	50.5	.	
12	Judy	F	14	64.3	.	.	
13	Louise	F	12	56.3	77	.	
14	Mary	F	15	66.5	112	.	
15	Philip	M	16	72	150	.	
16	Robert	M	12	64.8	128	.	
17	Ronald	M	15	67	133	.	
18	Thomas	M	11	57.5	85	.	
19	William	M	15	66.5	112	.	

	Table	Freq	Percent	CumFreq	CumPerc	F_Name	F_Sex	F_Age	F_Height	F_Weight	F_var	F_cvar
1	Table Name	3	15.79	3	15.79	c_missing						
2	Table Name	16	84.21	19	100.00	c_non-missing						
3	Table Sex	19	100.00	19	100.00		c_non-missing					
4	Table Age	19	100.00	19	100.00			n_non-missin				
5	Table Height	19	100.00	19	100.00				n_non-missing			
6	Table Weight	4	21.05	4	21.05					n_missing		
7	Table Weight	15	78.95	19	100.00					n_non-missing		
8	Table var	19	100.00	19	100.00						n_missing	
9	Table cvar	19	100.00	19	100.00							c_missing

	varname	N	NMiss	Mean
1	Age	19	0	13.315789474
2	Height	19	0	62.336842105
3	Weight	15	4	99.533333333
4	var	0	19	.

	table	Frequency	Percent	CumFrequency	CumPercent	formated
1	Name	3	15.79	3	15.79	c_missing
2	Name	16	84.21	19	100.00	c_non-missing
3	Sex	19	100.00	19	100.00	c_non-missing
4	Age	19	100.00	19	100.00	n_non-missing
5	Height	19	100.00	19	100.00	n_non-missing
6	Weight	4	21.05	4	21.05	n_missing
7	Weight	15	78.95	19	100.00	n_non-missing
8	var	19	100.00	19	100.00	n_missing
9	cvar	19	100.00	19	100.00	c_missing

Programming Efficiency Examples

- Identify the new or changed records

Use DATA-MERGE

```
data compare;
    merge master ( in=a )
          trans  ( in=b
rename=(var1=_var1 var2=_var2));
    by id;
    if b and not a then flag_new = '1';
    else flag_new = '0';

    if flag_new = '0';
    if var1=_var1  then flag1= '0';
else flag1='1';
            …
```

Use SQL-set operation

```
proc sql;
    create table changed_or_new as
            select  * from trans
            except corr
            select  * from master
;
quit;
```


Programming Efficiency Examples

- Identify common variables in multiple data sets

```
proc sql;
   create table _common as
    select * from a  where 0
    union corr
    select * from b  where 0
    union corr
    select * from c  where 0
    ;
quit;
```

SQL set operation overlays columns that have the same name in the tables, when used with EXCEPT, INTERSECT, and UNION, CORR (CORRESPONDING) suppresses columns that are not in all of the tables.

Programming Efficiency Examples

- Use the right SAS built-in functions

Old:

```
text = TRANWRD(TRANWRD(TRANWRD(TRANWRD(TRANWRD(htmltext,'>','&gt;'),'<','&lt;'),
'&','&amp;'), '"', '&quot;'), "'", '&apos;') ;
```

New:

```
text = HTMLDECODE(htmltext);
```

Old:

```
initial = substr(first_name,1,1)||substr(last_name,1,1) ;
```

New:

```
initial = first(first_name)||first(last_name);
```

Old:

```
cdate = put(year(datepart(datetime())), f4.) || put(month(datepart(datetime())), z2.);
```

New:

```
cdate = put(today(),yymmn.);
```

13

Optimize PROC GINSIDE

- PROC GINSIDE overview
- An application: find Blocks for Zip code centers
- PROC GINSIDE performance
  - Large data sets
  - Intensive computations
- Optimize PROC GINSIDE
  - Reduce map data sizes – SELECT statement
  - Preliminary search – Block limits of XY coordinates
  - Search within the selected Blocks only – %macro Loop to create ZIP specific map data set and run PROC GINSIDE for each ZIP.

14

Optimize PROC GINSIDE

- PROC GINSIDE overview

PROC GINSIDE was first introduced in SAS 9.2. "The new GINSIDE procedure determines which polygon in a map data set contains the X and Y coordinates in your input data set. For example, if your input data set contains coordinates within Canada, you can use the GINSIDE procedure to identify the province for each data point."

PROC GINSIDE is a application of the **point-in-polygon** (**PIP**) problem.

- An application: find Blocks for Zip code centers
  - CENSUS Block

https://www.census.gov/newsroom/blogs/random-samplings/2011/07/what-are-census-blocks.html

15

Optimize PROC GINSIDE

- A application: find Block for Zip code centers
  - ZIP code and CENSUS Block data sets

Column Name	Type	Length	Format	Label
ZIP	Number	8	Z5.	The 5-digit ZIP Code
X	Number	8	11.6	Longitude (degrees) of the center (centroid) of ZIP Code.
Y	Number	8	11.6	Latitude (degrees) of the center (centroid) of ZIP Code.

16

Optimize PROC GINSIDE

- Texas has about 2600 ZIP codes and 914,231 Census Blocks

1. **Convert Shapefile to SAS MAP data**

```
proc mapimport datafile="&path\&shpfile\&shpfile..shp"
                        out     = map.map_&fp._block ;
              select geoid10;
run;
```

MAP.MAP_48_BLOCK has 43,353,186 observations and 4 variables.

2. **ZIP code data**

```
data zip_48;
    set sashelp.zipcode(keep= x y zip state);
    where state=48;
run;
```

17

Optimize PROC GINSIDE

- Performance w/o optimization

```
proc ginside   data= zc_48   map = map.block_48   out = work.zc_48_block      ;
  id geoid10 ;  run ;

NOTE: Compressing data set WORK._GINSID000000000000000000000 decreased size by
      2.53 percent.
      Compressed is 716243 pages; un-compressed would require 734806 pages.
NOTE: The data set WORK.ZC_48_BLOCK has 310 observations and 19 variables.
NOTE: Compressing data set WORK.ZC_48_BLOCK decreased size by 16.67 percent.
      Compressed is 5 pages; un-compressed would require 6 pages.
NOTE: PROCEDURE GINSIDE used (Total process time):
      real time           30:59:38.86
      cpu time            30:58:15.82
```

18

Optimize PROC GINSIDE

- New algorithm step 1

```
*********************************************************************
* Algorithm step 1: use limits of each geoid10 to match zip codes;
*********************************************************************
proc sql;
   create table block_&fp._limit
   as
   select geoid10
       ,max(y) as max_y
       ,min(y) as min_y
       ,max(x) as max_x
       ,min(x) as min_x
   from  map.block_&fp
   group by geoid10;
quit;

proc sql;
   create table  gzinside_1_&fp
   as
   select a.*
         ,b.geoid10
         ,count(distinct geoid) as ct_match
   from  zip.zipcode (keep=zip x y state where=(state=&fp)) a
          left join block_&fp._limit b
         on     a.y between b.min_y and b.max_y
            and a.x between b.min_x and b.max_x
   group by zip;
quit;
```

ct_match	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	3	0.12	3	0.12
1	952	36.62	955	36.73
2	958	36.85	1913	73.58
3	417	16.04	2330	89.62
4	166	6.38	2496	96.00
5	63	2.42	2559	98.42
6	23	0.88	2582	99.31
7	11	0.42	2593	99.73
8	6	0.23	2599	99.96
9	1	0.04	2600	100.00

Some states, like OK, can have more than 55% matched with ct_match=1.

19

Optimize PROC GINSIDE

- New algorithm step 2

```
*********************************************************************;
* Algorithm step 2: use proc ginside for zip codes with multiple matched;
*********************************************************************;

%macro ginside(zip);

proc sql;
   create table map_&zip
   as
   select a.*
   from  map.map_&fp._block a, map.gzinside_1  b
   where b.zip=&zip and a.geoid10=b.geoid10;
quit;

proc ginside
     data= map.zip_&fp (where=(zip=&zip))
     map = map_&zip
     out = ginside_&zip;
     id geoid10;
 run;

%mend ginside;
```

Each ZIP/GINSIDE took only 0.02 ~ 0.04 seconds.

```
data _null_;
   set map.gzinside_1;
   where ct_match>1;
   call execute(catt('%ginside(', zip,');'));
run;

data map.gzinside_&fp;
   set ginside_:
       map.gzinside_1 (in=a where=(ct_match<2));
       if a then rc=1; else rc=2;
       if a and geoid10='' then rc=9;
  run;

proc freq data=map.gzinside_&fp;
  table rc;
run;

* ***** END * ********************************;
```

rc	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	952	36.62	952	36.62
2	1645	63.27	2597	99.88
9	3	0.12	2600	100.00

20

Optimize PROC GINSIDE

New algorithm step 2:

- %macro Loop: create and process ZIP specific data

Applications that break and process data sets chunk by chunk are not efficient **if the data sets can be processed as a whole,** because it increases I/O operations. An example can be found in the paper presented at MISUG Feb 2017 meeting. Here the situation is different. %macro loop is efficient.

- Data-oriented

Instead of searching among about 914,231 polygons in Texas Block data set (43,353,186 observations), the new algorithm search only among 2 to 9 polygons for each zip code. It runs much faster since it reduces lots of CPU time and I/O time.

Optimize PROC GINSIDE

- Results and improvement

	records	runtime - PC	runtime - Linux
w/o optimization	310 ~ 780 ZIP codes in Texas	_ _	31~33 hours
	2600 ZIP in Texas	> 3 days, job killed	<30 minutes
	4356 ZIP codes	_ _	**135 hours**
optimization	2600 ZIP codes	< 6 minutes	3 minutes
	41k ZIP codes	< 8 hours, 1 hour w reuse of the limits files	**< 30 minutes**

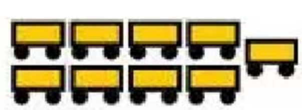
Optimize PROC GINSIDE

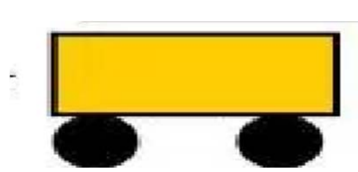
Summary of the optimization

1. Use the **select** statement to reduce map data file size.
2. Use Block **limit** data sets (that have way much less observations than the original Block data sets) to perform first match.
3. Use **ZIP specific map data** sets to enormously reduce the search range of GINSIDE procedure. Instead of searching within 914,231 blocks, GINSIDE only searches within about 5000 blocks overall.
4. In short, it reduces a large number of the processed records; therefore, it reduces I/O and CPU time. The improvement is significant.

Another Optimization Example

1. Medicare Part D claim data and patient data
2. Code is shorter, easier to understand (user friendly)
   Less than 80 lines. (original one has 185 lines)
3. Run faster
   One-drug job run time less than 1 hour (original one took 13~33+ hour)
   Three-drug job run time less than 3 hours (original one took 72~96 hours)
4. Algorithm: simplified, all in one batch/bunch process.
   Avoid %macro loop. Only 5 steps (original one has around 1000 steps)





Questions and Comments

THANK YOU!

Contact: lqliu@umich.edu

25