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

Lingqun Liu, University of Michigan

MISUG, Feb 2018

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.
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


"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= vars[i]*amount*cnnissions;
     end;
   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;
   ...
   ```

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

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);
```
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

---

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

```sas
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;
```

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

Similarly, this technique can be used to summarize all numeric 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;
```

• Check missing values of all variables in a data set

```
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

• Identify the new or changed records

Use DATA-MERGE
```mermaid
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';
   end
```

Use SQL-set operation
```sql
proc sql;
create table changed_or_new as
   select * from trans
   except corr
      select * from master;
quit;
```

• 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.);
```

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.
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

Optimize PROC GINSIDE

• A application: find Block for Zip code centers
  • ZIP code and CENSUS Block data sets
Optimize PROC GINSIDE

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

1. Convert Shapefile to SAS MAP data
   proc mapimport datafile="spath\&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;

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._GINSID00000000000000000000 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
Optimize PROC GINSIDE

- New algorithm step 1

```sql
create table block_cfp_limit
as
select geoid0, 
    max(y) as max_y, 
    min(y) as min_y, 
    max(x) as max_x, 
    min(x) as min_x 
from map.cfp_limit 
group by geoid0;
quit;
```

```
create table ginside_t_cfp 
as 
select a.*, 
    b.geoid0, 
    c.countdistinct geoid0 as ct_match 
from zip.cfp where ct_match=1 
left join block_cfp_limit b 
    on a.geoid0 between b.min_y and b.max_y 
and a.x between b.min_x and b.max_x 
group by zip;
quit;
```

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

Optimize PROC GINSIDE

- New algorithm step 2

```sql
/* Algorithm step 2: use proc ginside for ZIP codes with multiple matched */
macro ginside(zip);
proc gnl;
    create table map Giles as 
    select a.*, 
    from map.cfp.where ct_match=1 
    where b.zip=zip and a.geoid0 b.geoid0;
quit;
proc ginside 
    data=map Giles (where=(zip=zip)); 
    map=map Giles 
    out=ginside Giles; 
    id Giles;
run;
end ginside;
```

Each ZIP/GINSIDE took only 0.02 – 0.04 seconds.
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

<table>
<thead>
<tr>
<th></th>
<th>records</th>
<th>runtime - PC</th>
<th>runtime - Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o optimization</td>
<td>310 ~ 780 ZIP codes in Texas</td>
<td>– –</td>
<td>31~33 hours</td>
</tr>
<tr>
<td></td>
<td>2600 ZIP in Texas</td>
<td>&gt; 3 days, job killed</td>
<td>&lt;30 minutes</td>
</tr>
<tr>
<td></td>
<td>4356 ZIP codes</td>
<td>– –</td>
<td>135 hours</td>
</tr>
<tr>
<td>optimization</td>
<td>2600 ZIP codes</td>
<td>&lt; 6 minutes</td>
<td>3 minutes</td>
</tr>
<tr>
<td></td>
<td>41k ZIP codes</td>
<td>&lt; 8 hours, 1 hour w reuse of the limits files</td>
<td>&lt; 30 minutes</td>
</tr>
</tbody>
</table>
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