Paper 113-28

Run Time Comparison Macro

Robert Patten, Lands' End, Dodgeville, WI

Abstract

Programming is part logic and part art; there are as many ways to solve problems as there are people that work on them. SAS® allows for this flexibility which is one of the beauties of the language. However this flexibility can also be troublesome. When you are confronted with two competing coding algorithms, how do you decide between them? This decision can become very important when dealing with large datasets in production code where resources are tight. Even if you do not live in this scenario, choosing the right algorithm is the right thing to do. Because I frequently need to compare code and/or determine the impact of coding changes, I developed a macro that facilitates the creation of average run time statistics. In addition, the macro can be a useful aid in identifying code bottlenecks by modifying code and comparing before and after average run times. SAS product: BASE. Audience: Beginner to Intermediate.

Introduction

Deciding between two competing coding algorithms, testing for code bottlenecks, or determining code change impact is not a glamorous job. This is a task that most of us do not want to think about and, fortunately, most of the time, we do not need to. There are other times however, when we do need to at least have some idea of the impact of our decisions. These decisions have the greatest potential impact under the following circumstances:

- Dealing with large datasets
- Developing code for a production environment
- Resources are tight

Thoroughly testing code requires more than a single run. Basing coding decisions by comparing run times from single runs can lead to erroneous decisions. Generally, taking several program runs and then averaging the resultant run times is best. This is because run times are influenced by operating system load and other factors. Averaging run time helps control for these effects.

Next, lets take a look at the macro.

The Macro and discussion

```
%*------------------------------------------;
%* Macro AlgTest                            ;
%* Purpose: Repeatedly runs SAS code, combining;
%* run times and reports average usage.     ;
%* Parms:                                   ;
%*  TestPGM - Location of SAS code to cycle.  ;
%*    This code should be stand-alone code -  ;
%*    that is complete steps only.          ;
%*  Cycles  - Number of times to cycle over   ;
%*            code.                         ;
%*  Log     - Location for log files.  There is;
%*            one log file per cycle.       ;
%* Example Call:                            ;
%*  AlgTest(TestPGM=C:\Method1.txt,cycles=2,  ;
%*          Log=C:\log\)                    ;
%* Operating System:                        ;
%*   The SAS System for PCs Version 8       ;
%*------------------------------------------;
%macro AlgTest(TestPGM=,Cycles=,Log=);
  %*-------------------------------;
  %* file path for program to test  ;
  %* and for log files             ;
  %*-------------------------------;
  filename TestPGM "&TestPGM";
  Libname  OutCPU  "&Log";

  %*---------------------------------------;
  %* Cycle over desired number of test runs. ;
  %*   - redirect log files with printto    ;
  %*   - load code to test with %include    ;
  %*---------------------------------------;
  options stimer notes;
  %do i=1 %to &Cycles;
    proc printto log="&Log.log&i..tst" new;
    run;
    %include TestPGM;
    proc printto;
    run;

    %*---------------------------------------;
    %* Read test log and parse for CPU time   ;
    %* - Step is the current SAS step         ;
    %* - Note is the Note for the current step;
    %* - cpu is the cpu for the current step  ;
    %* - real is the wall time for the current;
    %* - step                                 ;
    %*---------------------------------------;
    data Cpu (keep=Step Note Cpu Real);
      length code $5
             Note $200;
      retain Note;

      %* point to log file created with       ;
      %* printto above;
      infile "&Log.log&i..tst" missover;

      %* input code and check for NOTE: so we ;
      %*can set Note variable                 ;
      input @1 code $Upcase5. @;
      if code='NOTE:' then input @7 Note & $ @;

      %* now check for cpu/real code and input;
      %* time.  note that the first step is   ;
      %* associated with printto so we skip it;
      input @7 code $Upcase4. @;
      if code = 'REAL' then do;
        input @16 Real 15.2 /
              @16 Cpu  15.2;
        step+1;
        if Real =. Then Real=0;
        if Cpu =. Then Cpu=0;
        if step>1 then output;
      end;
    run;

    %*---------------------------------------;
    %* Append to new dataset                  ;
    %*  we must delete if already exists      ;
    %*---------------------------------------;
    %if &i=1 %then %do;
      proc delete data=OutCPU.FinalCPU;
      run;
    %end;
    proc append data=Cpu base=OutCPU.FinalCPU;
    run;

  %end;

  %*---------------------------------------;
  %* Finally, process for final report      ;
  %*---------------------------------------;
  proc means data=OutCpu.FinalCPU mean noprint
                  nway;
    var Cpu Real;
    id Note;
    class step;
    output out=Test(drop=_type_ _freq_)
           mean=;
  run;
  proc print data=Test noobs;
    title1 "SUGI 28 paper";
    title2 "Average CPU times (Seconds)";
```

```
    title3 "Cycles=&cycles";
    title4 "For => &TestPGM";
    sum Cpu Real;
  run;

%mend AlgTest;
```

The code above is well documented and straight forward. I am not going to spend a lot of time going over it, however, I would like to go over some key points. First, the parameters are as follows:

- TestPGM – points to a file that contains the SAS code you want to develop run time statistics for. The file is %included in the program. It must contain complete working SAS code steps. The macro does not perform any error checking. It is up to you to make sure that you are providing proper code.
- Cycles – The number of times you want to run the included code. The final report contains the average of these individual runs.
- Log – Each run creates its own separate log file that is parsed in the subsequent section of the macro. If you ask for 30 cycles then you will get 30 log files.

PROC PRINTTO is used to direct the SAS log to a file. Note the use of the NEW option that rewrites any existing log file.

The Cpu datastep is where the individual log files are parsed extracting the Cpu and Real run times as well as the Step Note. What must be pointed out here, is that the code (especially the code surrounding the INPUT statements) is specific to the SAS system for PCs version 8. If you want to port to another environment, then you must modify these statements to match that environment.

The final dataset (FinalCPU) is an accumulation of the individual parsed log files information. If this dataset already exists, then we should delete it first, as we are using PROC APPEND which would append to the already existing dataset and the resultant run time statistics would be in error.

Finally, a PROC MEANS is used to roll the data up to the step level and a PROC PRINT is used to produce the report.

Using this macro to check code that involves large datasets might take forever. In these cases it would be wise to develop smaller sample datasets that are identical to the originals and use them instead. This way, you can loop several times and still end up with meaningful statistics.

Next, lets take a look at a specific example. Keep in mind, that the example itself is not important here. What I am trying to show is how you might take two separate coding solutions, both producing the same results, and compare their performance.

Example

Lets say that you want to compare alternative methods of producing a report (both produce similar reports). Method 1 is what I would call a traditional approach that involves reading in external data, matching this data with an already existing dataset, using PROC MEANS to summarize the data, then using PROC PRINT to produce the final report. Method 2 uses the data step to perform all processing including merging the SAS dataset and the flat file directly, summarizing the data using temporary arrays, then printing the report using PUT statements. You would like to implement Method 2 as you suspect is quite a bit more efficient. However, to be thorough, you need to test these methods head to head. The actual code for the two methods follows:

Method 1:

```
*-------------------------------------------;
* Raw data from outside vendor             ;
*   Assumption: Contains customer key as well ;
*   as additional demographic data.         ;
*-------------------------------------------;
Data demo;
  infile 'C:\Demo.txt';
  input @1 key    7.
        @9 income 8.2
  ;
run;
LibName TestData 'C:\Data';
*-------------------------------------------;
* Traditional merging of these files.      ;
*-------------------------------------------;
Data final;
  merge demo
        Testdata.customer;
  by key;
  if income >700 then level=1;
  else
  if income >500 then level=2;
  else
  if income >200 then level=3;
  else                level=4;
run;
*----------------------;
* Report               ;
* Sales by income level ;
*----------------------;
proc means data=Final sum noprint nway;
  var sales;
  class level;
  output out=Rpt1
         sum=;
run;
proc print noobs split='*';
  var level sales _freq_;
  title1 "Total Sales by Income level";
  label
    sales='Total*Sales'
    level='Income*Level'
    _freq_='Number*of*Customers'
  ;
run;
```

Method 2

```
*-------------------------------------------;
* Method 2                                 ;
* Doing all the work in one pass of the data! ;
*  - Non-traditional merging of these files.  ;
*  - create reports                         ;
*-------------------------------------------;
Libname testdata 'C:\Data';
data _null_;
  set testdata.customer end=lastSAS;

  * merge SAS dataset and flat file;
  retain done 0;
  if not done then do;
    retain rawkey;
    advance=0;
    infile 'C:\Demo.txt' end=lastraw;
    do until (lastraw or rawkey>=key);
      if advance then input;
      input @1 rawkey 7. @@;
      advance=1;
    end;
    if rawkey=key then input @9 income 8.2;
    if lastraw and rawkey<=key then done=1;
  end;

  * Setup arrays to hold aggregates;
  array IncomeSales{4} _temporary_;
  array IncomeCust{4}  _temporary_;

  * Assign income level sales and count;
  * customer;
  if income >700 then level=1;
  else
  if income >500 then level=2;
  else
```

```
  if income >200 then level=3;
  else                level=4;
  IncomeSales{level}+sales;
  IncomeCust{level}+1;

  *-----------------------------------;
  * Create report after last record    ;
  * is processed                      ;
  *-----------------------------------;
  if lastSAS then do;
    file print;
    title;

    * Report 1;
    put @34 "Total Sales by Income level";
    put @37 "Report 1 - all in one";
    put;
    put @56 'Number';
    put @34 'Income'
        @46 'Total'
        @58 'of';
    put @34 'Level'
        @46 'Income'
        @56 'Customers';
    put;
    do i=1 to 4;
      put @34 i 2.
          @44 IncomeSales{i}
          @56 IncomeCust{i}
      ;
    end;
  end;
run;
```

You would like to understand the impact of choosing Method 2 over Method 1. Using the following macro call:

```
%AlgTest(TestPGM=C:\Method1.txt,
         cycles=10,
         Log=C:\sugi28\);
```

Produces the following report:

```
                SUGI 28 paper
          Average CPU times (Seconds)
                  Cycles=10
             For => C:\Method1.txt

step  Note                    Cpu      Real

2     DATA statement used:    0.569    0.853
3     DATA statement used:    1.287    5.213
4     PROCEDURE MEANS used:   0.232    0.497
5     PROCEDURE PRINT used:   0.008    0.043
                              =====    =====
                              2.096    6.606
```

Alternatively, running the following macro call:

```
%AlgTest(TestPGM=C:\Method2.txt,
         cycles=10,
         Log=C:\sugi28\);
```

Produces this report:

```
                SUGI 28 paper
          Average CPU times (Seconds)
                  Cycles=10
             For => C:\Method2.txt

step  Note                    Cpu      Real

2     DATA statement used:    0.643    1.415
                              =====    =====
                              0.643    1.415
```

A comparison of the two reports shows that Method 2 produces substantially faster CPU and Real time as you suspected. Having this information allows you to confidently move forward with this coding alternative. Using the macro has put you in a better position to make a more informed decision when choosing between these two alternatives.

Conclusion

Choosing between competing code implementations, making and testing coding changes, and testing for code bottlenecks are not always the easiest or glamorous tasks involved in programming. However, these tasks can be very important especially when involved in situations where finding the most efficient way to perform a task is a priority. But even in our day to day programming, we should be striving to produce the most efficient code possible. Using the macro introduced here, can aid in making more informed decisions when faced with these tasks.

Contact Information

Robert Patten
Lands' End
5 Lands End Way
Dodgeville, WI 53533
608-935-4832
robert.patten@landsend.com