PhUSE EU Connect 2018

Paper CT05

Do Loops in Proc DS2

Fred Ross, Quanticate International Ltd, Manchester, United Kingdom
George Allerston, Quanticate International Ltd, Manchester, United Kingdom
Rachel Linacre, Quanticate International Ltd, Hitchin, United Kingdom

ABSTRACT

PROC DS2 is a new SAS® proprietary programming language with full release in version 9.4. It has many features but in the paper focus will be on Object Oriented Programming (OOP) and multithreading. Multithreading and greater efficiency in the use of your system can be an exciting prospect, but the daunting task of learning OOP can slow down or block attempts to fully learn and utilise this exciting new procedure. With its relatively recent addition to SAS, DS2 has few existing examples and tutorials, and even fewer Pharmaceutical specific learning resources. We will take you through a basic overview of OOP, followed by simple Do Loops in PROC DS2 and finish with some real world applied examples. This abstract/paper will provide a starting point for programmers and statisticians, new to the procedure, to begin their journey of utilising DS2 fully and effectively in their day-to-day programming activities.

INTRODUCTION

DS2 is a SAS proprietary programming language that is appropriate for advanced data manipulation. DS2 is included with SAS 9.4 and intersects with the SAS DATA step. Its advantages over data step programming include, ANSI SQL types, programming structure elements, the capability to write user-defined methods and packages and multithreading. [1]

Multithreading is, in our use, the process of breaking up the input dataset and performing the operation on these split datasets then combining them back together. Every one of these operations is run in parallel, spread and processed across multiple cores on a single processor simultaneously. Each of these can be described as a thread.

DS2 is an intriguing procedure to SAS programmers; however the depth of the procedure can prove to be intimidating. In this paper we are going to set out some basic first steps in the form of performing an iterative Do Loop in PROC DS2 and also giving a brief introduction on how to multithread in DS2 to speed up the real time taken to perform Do Loops.

BASIC OVERVIEW AND RELEVANT CONCEPTS OF OBJECT-ORIENTED PROGRAMMING

OOP is an alternate technique to write computer programs compared to procedural programming, i.e. traditional SAS data steps and procedures. In OOP data and related procedures are contained together within “objects”. Each object is instantiated based on the parent class. This means that any given object will inherit all methods and properties of that class. Simultaneously, methods and properties may be instances of other classes and may be nested further. Therefore, OOP can be considered as a system consisting of many levels of nested connections.

There are a number of advantages of using OOP over procedural programming. Generally object-oriented programs are easier to interpret because the underlying code can map directly to real-world concepts. The code used is easier to modify because modification tends to involve updates to individual and centralized objects as opposed to the re-writing of, or changing of, segments of code throughout the entire program. Within this paper we will be making use of OOP so it is important to include a basic overview and the differences to the procedural programming you would be used to in SAS DATA step. It should be noted that there are disadvantages to OOP too, as it is not always as straightforward compared to procedural programming.

Our paper’s focus is on PROC DS2, a new programming feature first present in SAS 9.3 with full release in SAS 9.4. DS2 introduces concepts of Object-Oriented principles that go beyond existing data step programming and allows advanced data manipulation. Two DS2 statements we will cover briefly are the DECLARE statement and the METHOD statement.

DS2 CODE BLOCK STRUCTURE

The basic structure of the DS2 code block is shown below. The procedure starts with PROC DS2; and is concluded with a RUN; and a QUIT; statement. The contents of this code will be covered in greater detail later in the paper:

```
proc ds2;
    data dsetout2/overwrite=yes;
      dcl char(2) anl01fl;
      method run();
        dcl int i;
          set subjvisitdata;
            do i=0 to &visits by 10;
              if avisitn=i then anl01fl='Y';
            end;
      end;
    enddata;
run;
quit;
```

DECLARE STATEMENT

For DS2, all variables must be declared, in this paper in various illustrations of SAS code we have explicitly declared our variables by using the DCL statement or the equivalent DECLARE statement. DCL associates a data type with each variable. For example here we associate our Analysis Flag 1 variable (ANL01FL) as a character variable of length 2.

```
dcl char(2) anl01fl;
```

The next thing to bear in mind is that where the DCL statement is positioned within the code also determines the scope of the variable. If used outside a METHOD statement, a global variable is created. Variables that we want to keep in the output must have this so called global scope. If DCL is used within a METHOD, i.e. after METHOD RUN(); but before the corresponding END; statement, a local variable is created. A local variable can be used to manipulate the data but ultimately will not be kept in the final dataset. Within a METHOD, DCL statements must precede METHOD statements or an error will be encountered.

In the code below the first DCL statement is of global scope as it is outside of the METHOD. The second DCL statement is within a METHOD so is local to that METHOD:

```
proc ds2;
    data dsetout2/overwrite=yes;
      dcl char(2) anl01fl;
      method run();
        dcl int i;
          set subjvisitdata;
            do i=0 to &visits by 10;
              if avisitn=i then anl01fl='Y';
            end;
      end;
    enddata;
run;
quit;
```

DATA TYPES

There is an expanded range of data types available in DS2 for us to specify within a DCL statement and some of the more useful ones are displayed below. It should be noted that when outputting to a .sas7bdat dataset all variables must be converted to the traditional character and numeric data types. In the table below CHAR(n) and INTEGER data types are detailed:

Data Type	Description
CHAR(n)	Stores a fixed-length character string, where n is the maximum number of characters to store. The maximum number of characters is required to store each value regardless of the actual size of the value. If char (10) is specified and the character string is only five characters long, the value is right padded with spaces.
INTEGER or INT	Stores a regular size signed, exact whole number, with a precision of ten digits. The range of integers is -2,147,483,648 to 2,147,483,647. Integer data types do not store decimal values; fractional portions are discarded. Note: Integer division by zero does not produce the same result on all operating systems. It is recommended that you avoid integer division by zero.

[2]

METHOD STATEMENT

In the second DS2 statement we will make use of is the METHOD statement. All data-processing code in DS2

(initialisation, derivation, outputting, etc.) must reside within a METHOD statement; there are three system-defined methods: RUN, INIT and TERM. It is also possible to create user-defined methods. We will be making use solely of the METHOD RUN. In Base SAS, the entire DATA Step program is included in the implicit loop. In DS2, the implicit loop is represented by the METHOD RUN, with the METHODs INIT and TERM (outside the scope of this paper)providing initialization and finalization code, respectively. [1]

When a system-defined METHOD is explicitly stated it must be defined without any parameters and without a return value, as we have done below with METHOD RUN(), if we add parameters and/or a returning value this will result in a compile error. Each METHOD statement must have a corresponding END as shown here:

```
proc ds2;
     data dsetout2/overwrite=yes;
       dcl char(2) anl01fl;
       method run();
         dcl int i;
           set subjvisitdata;
             do i=0 to &visits by 10;
               if avisitn=i then anl01fl='Y';
             end;
       end;
     enddata;
run;
quit;
```

SIMPLE DO LOOPS IN PROC DS2

For use throughout our examples we first create a dummy dataset of 10000 subjects each with 201 visits. Though not a common example of a dataset that might be encountered it is large and simple enough for us to illustrate the differences between a basic data step and a similar process in DS2.

The dummy dataset SUBJVISITDATA to be used throughout this paper is created with the following code and though this contains a Do Loop we will not be converting this to DS2:

```
%let subjects=10000;
%let visits=1000;
data subjvisitdata;
     do usubjid=1 to &subjects;
       do avisitn=0 to &visits by 5;
       output;
       end;
     end;
run;
```

This creates the below dataset, SUBJVISITDATA:

usubjid	avisitn
1	0
1	5
1	10
...	...
1	995
1	1000
2	0
2	5
2	10
...	...

DO LOOP IN REGULAR DATA STEP

The purpose of our simple loop will be to flag each visit that is a multiple of 10 and also visit 0. To start with we will do this without using DS2 and instead use a traditional data step to create DSETOUT1:

```
data dsetout1(drop=i);
     set subjvisitdata;
     do i=0 to &visits by 10;
       if avisitn=i then anl01fl='Y';
     end;
run;
```

```
NOTE: There were 2010000 observations read from the data set WORK.SUBJVISITDATA.
NOTE: The data set WORK.DSETOUT1 has 2010000 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           1.68 seconds
      cpu time            1.69 seconds
```

This produces the following dataset, DSETOUT1:

usubjid	Avisitn	anl01fl
1	0	Y
1	5	
1	10	Y
...	...	
1	995	
1	1000	Y
2	0	Y
2	5	
2	10	Y
…	…	

The real time can be reduced and the cpu time increased by converting to DS2 and then implementing multithreading. It is important to note that the real and cpu times will be the measure used to compare the efficiency. Shorter real time indicates the most important efficiency gain: less waiting time. Higher cpu time is only of secondary importance as a measure for efficiency: it might indicate that a higher percentage of (available) computing capacity has been used, i.e. more efficient use of resources; however, an increase in cpu time with unchanged real time actually would be a loss in efficiency. Also using more capacity of a shared CPU might actually lead to loss of efficiency for co-users of the CPU.

DO LOOP IN DS2

First we create a new dataset, DSETOUT2. We will not multithread in this example and instead demonstrate a straight conversion of the data step into DS2 to highlight aforementioned features of the procedure.

Here we create our required dataset. The DCL statements declare the new variables, their types and length. We declare the character variable ANL01FL globally outside the METHOD as this is a variable we want to keep. We can declare the i variable as INTEGER, within the METHOD locally as we do not want this variable in our output dataset:

```
******************DS2 without multithreading;
proc ds2;
    data dsetout2/overwrite=yes;
      dcl char(2) anl01fl;
      method run();
        dcl int i;
          set subjvisitdata;
            do i=0 to &visits by 10;
              if avisitn=i then anl01fl='Y';
            end;
      end;
    enddata;
run;
quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           2.84 seconds
      cpu time            2.23 seconds
```

Comparing these numbers we can see that a straight conversion into DS2 increases the real time and cpu time spent crafting exactly the same result. This is because calling PROC DS2 in SAS generates additional work for the CPU when compared to DATA step, even when performing the same core function, and so an increase in both cpu and real time is expected. We can decrease the real time taken by making use of other techniques such as multithreading.

SIMPLE DO LOOP WITH MULTITHREADING

Usage of multithreading can reduce the real time spent on the same operation at the expense of increased cpu time and hence increased utilisation of your system. Multithreading sections the data into the number of threads, executes the split data simultaneously across multiple processors or a distributed system and then sets the processed data back together.

DO LOOP WITH 4 THREADS

First we need to turn the data step into a thread program. To do this we take the previous conversion of the data step into DS2 and then simply change DATA to THREAD and ENDDATA to ENDTHREAD we then apply the thread to our original dataset and specify the number of threads to be used.

The code contained within the THREAD and ENDTHREAD statements is the same as that contained within the previous PROC DS2. We have changed DSETOUT2 to THREAD1, this will still create a dataset called THREAD1 but this contains information about the thread for processing.

```
**********Do Loop with Multithreading with 4 threads;
proc ds2;
  thread thread1/overwrite=yes;
    dcl char(2) anl01fl;
    method run();
      dcl int i;
        set subjvisitdata;
          do i=0 to &visits by 10;
            if avisitn=i then anl01fl='Y';
          end;
    end;
  endthread;
```

Within the same PROC DS2 we then want to create the dataset dsetout3 using the previously defined thread. We must declare the THREAD, THREAD1 and the object, SUBJVISITDATA.

```
  data dsetout3/overwrite=yes;
    dcl thread thread1 subjvisitdata;
    method run();
      set from subjvisitdata threads=4;
    end;
run;
quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           0.65 seconds
      cpu time            2.57 seconds
```

We can see from the reduced real time and increased cpu time that we successfully multithreaded our Do Loop.

DO LOOP WITH 18 THREADS

We can now try increasing the number of threads from 4 to 18 to add another point of comparison. All we need to change is the number after THREADS= as below:

```
******************Do Loop with Multithreading with 18 threads;
proc ds2;
  thread thread1/overwrite=yes;
    dcl char(2) anl01fl;
    method run();
      dcl int i;
        set subjvisitdata;
          do i=0 to &visits by 10;
            if avisitn=i then anl01fl='Y';
          end;
    end;
  endthread;

 data dsetout3/overwrite=yes;
    dcl thread thread1 subjvisitdata;
    method run();
      set from subjvisitdata threads=18;
    end;
run;
```

```
quit;

NOTE: PROCEDURE DS2 used (Total process time):
      real time           0.62 seconds
      cpu time            3.12 seconds
```

We can see that using 18 threads and DS2 to complete the operation reduces the real time taken to run the procedure. The increased cpu time can be accounted for as the time taken to split the data, process in respective threads (roughly the cpu time taken with basic data step divided by the number of threads), and set back together.

However if we try to increase the number of threads to 100 the reduction below in real time is significantly less than the jump from 4 to 18 threads but the cpu time is again increased. This can be illustrated with the following approximations of real and cpu time:

Approximation of Real Time when using DS2	**Approximation of CPU Time when using DS2**
=	**=**
Time taken to call PROC DS2	Time taken to call PROC DS2
+	+
Time taken to split data into threads	Time taken to split data into threads
+	+
(Time taken to process METHOD/Number of threads)	Time taken to process METHOD across all threads
+	+
Time to merge data back together	Time to merge data back together

	Real time	CPU time
Data step	1.68	1.69
Data step conversion to DS2	2.84	2.23
DS2 + Multithreading 4 threads	0.65	2.57
DS2 + Multithreading 18 threads	0.62	3.12

Above we can see the differences in real time and cpu time between the different techniques we employed, showing an increased efficiency of using DS2 over a standard data step.

LIMITATIONS WITH PROC DS2

Although DS2 can be a useful tool in some circumstances, and using it with multithreading can save time in performing some tasks, it does have its limitations and drawbacks. Whilst experimenting within the procedure we encountered plenty of scenarios where using other SAS functions or procedures were more efficient than PROC DS2.For example:

CONDITIONAL DO LOOPS

Iterative Do Loops are where multithreading can increase cpu time as the iterations can be split between threads and thus decrease real time. Conditional Do Loops cannot be sped up with multithreading however Do Loops which are both Iterative and Conditional can be.

SYSTEM LIMITATIONS

If the system you are running SAS in has only 1 core in the CPU then multithreading will not be of any use to you. A SAS program will still run however the real time will not be decreased compared to a data step.

RECREATING OTHER PROCEDURES IN DS2

We attempted to recreate PROC FREQ in DS2 making use of multithreading, starting with a basic PROC FREQ, then transforming this into DS2. Though we managed to recreate the results of the PROC FREQ in DS2, comparing the results showed that both the real time and cpu time were higher in the DS2 version. This had shown us that although we had multithreaded the process we had failed to decrease real time compared to the PROC FREQ we were trying to replicate. This seems to confirm the robustness and efficiency of SAS Procedures.

CONCLUSION

PROC DS2 offers SAS programmers a more advanced way to manipulate data over the traditional data step and multithreading to speed up some operations within SAS, should their system allow it. Using the very basic first steps

outlined in this paper, DS2 will provide a rich learning ground with new tools and new ways to work for programmers looking to expand their repertoire of SAS techniques.

REFERENCES

[1] SAS 9.4 DS2 Programmers Guide
https://documentation.sas.com/api/docsets/ds2pg/9.4/content/ds2pg.pdf?locale=en

[2] Expansion of Opportunities in Programming: DS2 Features and Examples of Usage Object Oriented Programming in SAS. PharmaSUG 2017, BB09. Serhii Voievutkyi e.a.
https://www.pharmasug.org/proceedings/2017/BB/PharmaSUG-2017-BB09.pdf

ACKNOWLEDGEMENTS

Big thanks to Will Greenway and Jon Henshaw for performing technical review of the paper. Also thanks to Piotr Karasiewicz, Charles Wright, Shrishaila Patil and John van Bemmelen for performing further review and comment.

CONTACT INFORMATION

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

Fred Ross
Quanticate International Ltd
Blackbox Business Centre,
Beech Ln,
Wilmslow,
SK9 5ER
Email: Fred.Ross@quanticate.com
Web: www.quanticate.com