Paper 325-2013

Not dividing by zero: last of the low hanging efficiency fruit

Bruce Gilsen, Federal Reserve Board, Washington, DC

ABSTRACT

As SAS ® Institute has improved the efficiency of its code, some of the old ways for users to improve efficiency, such as using WHERE or WHERE= instead of IF in the DATA step, no longer make much difference. Current user efforts to improve efficiency tend to focus on more sophisticated techniques such as indexes or hashing. However, one of the classic methods, not dividing by zero in the DATA step, can still offer a large efficiency improvement. This paper shows some sample code that divides by zero and some benchmark results from changing the code to test for zero denominators to avoid dividing by zero.

Efficiency in "the old days" and now

Longtime SAS users can recall that there were some fairly easy ways to improve SAS code efficiency. Perhaps most common was to use a WHERE statement or WHERE= clause instead of a subsetting IF statement, which was extolled frequently in conference papers, including Gilsen (1999). The informal, commonly used description of these statements makes it seem obvious that WHERE or WHERE= is more efficient than IF:

- A subsetting IF statement requires that the observation be read into the Program Data Vector (PDV) to be evaluated and then discarded if the selection criteria are not met.
- WHERE and WHERE= "pre-process" the observation before it is read into the PDV. Observations not meeting the selection criteria are never read, improving efficiency.

However, SAS Institute has improved the efficiency of its code, and as described in Langston (2005), some of the old ways to improve efficiency, including using WHERE or WHERE= instead of IF in the DATA step, no longer make much difference. In fact, as noted on the SAS-L e-mail list, a WHERE statement or WHERE= clause could even run slightly slower than a subsetting IF statement if a DATA step function is being processed.

Current user efforts to improve efficiency tend to focus on more sophisticated techniques such as indexes or hashing. But, some recent benchmark tests I conducted at the Federal Reserve Board confirmed that at least one of the classic methods, testing for zero denominators to avoid dividing by zero, can still offer a large efficiency improvement.

Code that divides or does not divide by zero

Data set ONE contains 1,000,000 observations and 50 variables, with values set to 0, 1, 2, 3, and 4 in equal amounts. Here are the first few observations for selected variables; see the appendix for code that creates this data set.

Obs	v1	v2	v3	v4	v5	v6	...	v46	v47	v48	v49	v50
1	0	1	2	3	4	0		0	1	2	3	4
2	1	2	3	4	0	1		1	2	3	4	0
3	2	3	4	0	1	2		2	3	4	0	1
4	3	4	0	1	2	3		3	4	0	1	2
5	4	0	1	2	3	4		4	0	1	2	3
6	0	1	2	3	4	0		0	1	2	3	4

Here are two examples of code that accomplish the same task: setting the variables V1 - V50 equal to 10 divided by the existing value of the variable. There are 50 million potential division operations, and in 20% of them (10 million), the denominator is zero.

Version 1: Do the division in all cases.

```
data two;
  set one;
  drop i;  * index variable;
  array vall (*) v1-v50;
  do i = 1 to dim(vall);
    vall(i) = 10 / vall(i);
  end;
run;
```

Version 2: Test if the denominator is missing, and if so, do not do the division.

```
data two;
  set one;
  drop i;  * index variable;
  array vall (*) v1-v50;
  do i = 1 to dim(vall);
    if vall(i) = 0
    then vall(i) = .;               * denominator is 0, do not divide ;
    else vall(i) = 10 / vall(i);    * denominator is not 0, so divide ;
  end;
run;
```

Note that both versions of the code generate the same result. In Version 1, attempts to divide by zero return a missing value. In Version 2, when the denominator is zero, we just assign a missing value as the result.

Here are the results using Version 9.2 on the Linux, Windows, and z/OS mainframe operating systems. Results are shown in the form real time / cpu time. Not dividing by zero improves performance on all platforms, most significantly on Linux.

	Linux	Windows	Mainframe
Divide by zero	43.43/43.40	1:01.28/1:01.06	1:17.52/49.49
Avoid divide by zero	2.82/2.83	28.68/5.21	57.45/34.37

Explanation of the results

According to SAS Institute, dividing by zero involves the operating system receiving a floating point hardware exception that is then forwarded to the SAS system. This operating system intervention explains why always attempting to divide was slower.

Divide by missing

I replaced zeros with missing values in the original data set and re-ran the benchmark tests. Dividing by missing and testing for a missing denominator to avoid divides by missing are about equally efficient. This is because unlike checking for zero, SAS does check if the denominator is missing before dividing.

CONCLUSION

Unless the number of attempts to divide by zero is very rare, you can improve program efficiency by checking if the denominator is zero and avoiding divides by zero. The amount of improvement and the necessary frequency of zero denominators to improve efficiency vary by operating system. And, based on tests run with previous SAS releases, results also vary by SAS release, and these results could change in a future release of SAS.

REFERENCES

Gilsen, Bruce (1999), "SAS ® Program Efficiency For Beginners," Proceedings of the Twenty-Fourth Annual SAS Users Group International Conference. <http://www2.sas.com/proceedings/sugi24/Begtutor/p72-24.pdf>

Langston, Rick (2005), "Efficiency Considerations Using the SAS® System," Proceedings of the Thirtieth Annual SAS Users Group International Conference. <http://www2.sas.com/proceedings/sugi30/002-30.pdf>

ACKNOWLEDGMENTS

The following people contributed extensively to the development of this paper: Heidi Markovitz and Donna Hill at the Federal Reserve Board, and Rick Langston and Jason Secosky at SAS Institute. Their support is greatly appreciated.

CONTACT INFORMATION

For more information, contact the author, Bruce Gilsen, by mail at Federal Reserve Board, Mail Stop N-122, Washington, DC 20551; by e-mail at bruce.gilsen@frb.gov; or by phone at 202-452-2494.

TRADEMARK INFORMATION

APPENDIX

Here is the DATA step used to create data set ONE.

```
    /* Create a data set in which 20% of values are 0 */
  data one;
    drop i ii vtemp;  /* index and temporary variables */
    array vall (*) v1-v50;
      /* In 1st observation,
         v1=0 v2=1 v3=2 v4=3 v5=4 v6=0 v7=2..... */
    do i=1 to 50 by 5;
      vall(i)=0;
      vall(i+1)=1;
      vall(i+2)=2;
      vall(i+3)=3;
      vall(i+4)=4;
    end;
```

```
    /* 1,000,000 obs, equal number of values of 0, 1, 2, 3, and 4 */
  do i = 1 to 1000000;
    output;
    vtemp = vall(1);
    do ii = 1 to 49;
      vall(ii) = vall(ii+1);
    end;
    vall(50) = vtemp;
  end;
run;
```