Optimize SAS/IML Codes for Big Data Simulation

Chao Huang, Yu Fu and Goutam Chakraborty
Oklahoma State University. Stillwater, OK. 74075

Introduction

SAS/IML is the matrix language module in SAS, and also a common toolbox for scientific simulation. Together with DATA Step and other SAS procedures, the IML procedure provides SAS users with a dynamic and interactive environment for matrix operation. Through our practice, we find that using some optimization techniques including rewriting codes in SAS/IML.to avoid loops will make the codes faster and simpler, and even achieve dramatic speedup. In addition, recently SAS/IML introduces a few new vector-wise subscripts, operators and functions for code optimization [1]. In this paper, we introduce our exploration toward the memory management in SAS/IML, and three examples about vectorization versus looping in the pursuit of computation efficiency increase. The examples are run on a desktop computer installed with Windows XP and SAS 9.3. And this PC has an Inter® Core 2 Duo CPU and 3GB memory.

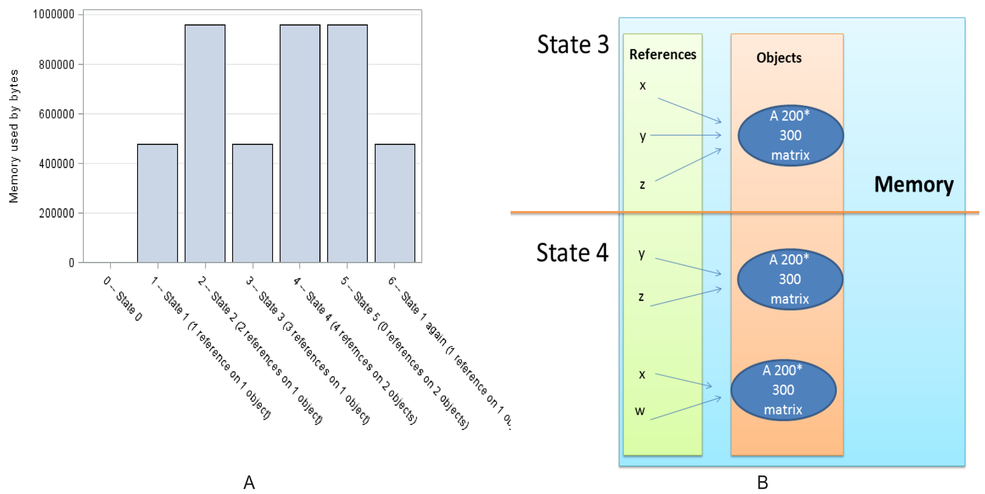
More details including codes are available on the proceeding **paper 256-2013**.

Methods and Results

Memory management in SAS/IML

We design an experiment to check the behavior of SAS/IML's memory manager. In SAS/IML, usually a matrix of 200 rows and 300 columns occupies about 400kB memory. However, at the beginning, we only request 1MB memory from the system by specifying the WORKSIZE option. Thus, given 1MB memory, three matrices of such size would cause a stack overflow and therefore the failure of PROC IML. Firstly we assign two (State 2) or three references (State 3) toward a single random matrix, and secondly change the value of only one element in this matrix (State 4). Finally we use the CLEAR statement to clear all of the objects (State 5), and re-generate another matrix with the same size (State 6).

The memory manager checks the size available of the memory allocated to IML. If the memory usage doesn't pass the alarming line, it will generously make copies. Otherwise, it will become very aggressive and compress the memory frenetically (Figure 1A). SAS/IML follows the rule of copy-on-change. At last, SAS/IML's FREE statement kills the references instead of the objects themselves (Figure 1B).



A B

Figure 1. Memory usage by PROC IML at different stages. (A) The memory costs by byte at each of the seven stages. (B) The comparison between 3 references on 1 object and 4 references on 2 objects

Strategies that optimize SAS/IML

VECTORIZATION VERSUS DO LOOP TO SQUARE NUMBERS

By the looping codes and the vectorized IML codes, we apply PROC IML to square every element in a vector from 1 to 50,000, and calculate the time spent by this operation. Two user-defined modules are created in PROC IML and repeatedly used to square those numbers for 100 times. System time spent is recorded by PROC IML's TIME function. Overall, the vectorizatized codes perform much faster than a DO loop in PROC IML (Figure 2A).

CALCULATE RUNNING TOTAL

An example to generate a new variable for the running total by either a DO loop or the CUSUM function, by a serial of random vectors from 1 million to 10 million elements with the step size of 1 million. The CUSUM function is at least 100 times faster than the looping structure. With the increase of the vector size from 1 million to 10 million, the gap between the time costs of the two methods turns even wider (Figure 2B).

COUNT ODD NUMBERS FOR A RANDOM VECTOR

An experiment to count how many odd numbers a random vector contains by 6 different methods. These methods include the SUM function, the "+" subscript, the SUM and CHOOSE functions, the NCOL and LOC functions, the COUNTMISS and LOC functions, a DO looping structure as benchmark. In general, the more straightforward the method is, the faster the computation is. Most importantly, all vector-wise methods beat the DO loop, especially for the vector that has many elements. In general, de-looping the IML codes would significantly improve the efficiency (Figure 2C).

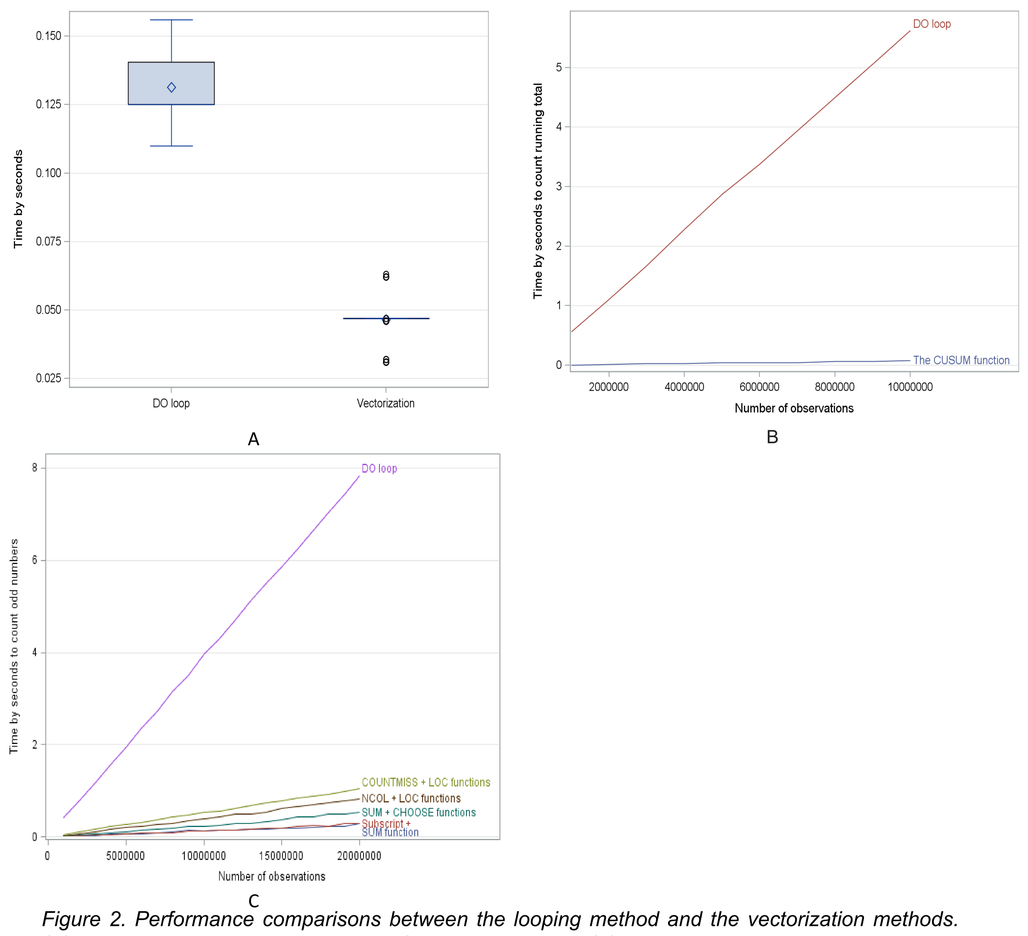


Figure 2. Performance comparisons between the looping method and the vectorization methods. (A) The example that squares number for random vectors. (B) The example that calculates running total for random vectors. (C) The example that counts odd numbers for random vectors.

Conclusion

We could make better use of SAS/IML as followed:

- Avoid unnecessary looping
- Apply the vector-wise functions and operators
- Check memory usage by the SHOW SPACE statement
- Test the codes' efficiency by its built-in TIME function

Reference

1. Rick Wicklin, "Statistical Programming with SAS/IML Software". SAS Publishing. 2011
2. Rick Wicklin. "SAS/IML tip sheets". The DO Loop. October 10, 2011