Optimizing Your SAS programs – Tips and Techniques

Craig Kasper, Manitoba Health

Winnipeg SAS User Group

October 13th, 2011

What is optimization?

- **Optimization** is refining your SAS programs to run in a way which is somehow “better”.
- There are several kinds of optimizing which a programmer may wish to do:
  - tailoring a program to use fewer resources
  - ensuring it is understandably written
  - ensuring it takes as little time to run as possible
- This presentation focuses on the most common kind of optimization: reducing program run times.

Why Worry About Optimization?

- “Optimizing is unimportant until it becomes important, and then it is very important.”
  - I ran into a situation this summer where optimization suddenly became very important.
  - In this case, optimization was the difference between a program that could finish running and one that couldn’t!

Design and Implementation

- This presentation contains both design principles and specific programming techniques.
- The design principles are discussed at a relatively high level here.
  - For more technical detail around these design principles, please see my blog posting at the SAS Canada community site.

Optimization: Design Tips

1: Memory is faster than disk.

- This is a simple matter of distance: data in memory is effectively closer to the processor chip which actually runs the program.
  - Memory on the processor itself is even closer than system memory, but this difference has much less of an effect.

(On the other hand…)

- While reading from and writing to disk can really slow a program down, modern systems handle it a lot better than older systems did.
  - Modern hard drives are blazingly fast compared to storage in older systems.
  - Modern computers are better at switching between tasks.
- Accordingly, the performance of many programs will often be "good enough" without worrying about reducing disk activity.

If disk accesses are a problem…

- If disk accesses are slowing your program down too much the solution is to load data into memory.
  - If the amount of free memory permits it, the SASFILE statement instructs SAS to load a dataset directly into memory before running a DATA step which uses it.
  - As of SAS 9.1, programmers can also use SAS's hash tables feature to load tables into memory during a DATA step.
    - Hash tables are very flexible, and can help even when other optimizations are difficult or impossible to apply because access to multiple tables at once is needed.
  - For more on reducing disk-related delay, look up “System Performance, Optimizing I/O” in SAS’s online help index.

2: Reading data is faster than writing

- This is because writing is actually slower, …
- … but also because writing has additional overhead involved.
  - e.g. allocating disk locations for the data.
- Accordingly, writing data to disk less often and in larger chunks can often boost speed.
  - SAS's BUFSIZE option can be used to set the size of the data chunks written to (or read from) a file, improving performance.

3: Certain disk accesses are faster

- Reading or writing data in the order it is stored on disk is normally faster than accessing the same number of records in random order.
  - This is because of how the data is stored on disk.
- It is normally faster to update a large data set in place on the disk (whether using a DATA step or PROC SQL) than to create a completely new one.
  - This will always reduce the overhead involved.
  - If there is a relatively small amount of data being updated, this will often reduce the amount of actual writing to disk as well.

Disk reads and writes vs Hash Tables

- If you're wondering when it would be most beneficial to use hash tables with your data, using them to replace the slower types of disk access is often the best place to start.
  - Use hash tables to replace high-volume random-access reads from a file. Not only is the memory access faster, random access to a hash table row is implemented more efficiently than it is possible for random disk access to be.
  - Gains can also be made by replacing high volume random-access writes with a process of loading data into a hash table, updating it and saving it back to disk.
- Gains can often be made using hash tables instead of other types of reading or writing data as well, but these gains will vary with the specific situation.

4: Numeric is faster than character

- Numeric values and variables are faster within SAS because they are more naturally suited to the processors which run SAS.
  - They use the full power of the processor.
  - They don’t require the type of indirect storage that character values do.
  - If they fit in the temporary storage on the computer’s processor, they may not need to be shuttled back and forth between the processor and memory as often.

5: Use 4-byte or 8-byte numeric variable lengths

- These are the lengths that the most recent modern computers and processors have been designed for.
  - Modern microprocessors normally handle them most efficiently.
  - They are the best fit for both on-processor memory and system memory, and for transferring between the two.
  - If math co-processing is available (as a design feature of the computer or of the main processor, say), one of these variable sizes will be expected.
- SAS's numeric variables are 8-byte by default.

6: Character functions are much slower than numeric functions

- There are several reasons for this:
  - Numeric values can be handled as one value while character strings must be handled in pieces.
  - Character strings usually need to be offloaded to memory, while numeric values sometimes do not.
  - Character strings use relatively slower instructions in your computer's processor.

Why this matters…

- Numbers can be stored as text in the real world, depending on the application.
  - This is not uncommon on legacy systems.
  - This technique may be used to avoid issues when sharing data between SAS and current systems.
- Depending on the nature of the processing involved, it may be faster to store the data in SAS as numeric data, converting to or from text only as needed.

Optimization: Techniques

About these techniques

- Please be advised that these strategies are mostly specific to DATA steps and similar types of SAS programming.
  - PROC SQL processing is normally fairly well optimized by default.
  - Optimization with other PROCs is sometimes not even possible, and is beyond the scope of this presentation.

7: IF… THEN… ELSE all you can

- Put as much code as you possibly can into IF ... THEN… ELSE blocks. This will make your program run faster by allowing it to achieve the same thing by executing less code.
  - This is where you will often find the “low-hanging fruit” of program optimization.
- Remember to use IF… THEN … ELSE instead of two separate IF… THEN… statements where appropriate.
  - The separate statements require an extra comparison when compared to using the ELSE.

8: Make the “common case” fast

- Choose the technique that will be faster for 80% of the inputs and slower for 20%, rather than the one which is faster for 20% of the inputs and slower for 80%.
  - You may have to analyze your input data or your program code to determine which technique is which.
- The most common application of this idea is re-arranging chained IF ... THEN … ELSE conditions to move the IF ... THEN conditions which are most often true to the earliest places in the chain.

9: Pre-compute where possible

- Pre-computing optimizations may involve:
  - Moving any computation which doesn't absolutely have to be done inside a loop outside the loop.
  - Creating a dataset or format that will be used to look up the results of processing rather than repeating processing itself.
  - Sorting data before processing to enable other optimizations.
  - Using sort-and-merge techniques when working with multiple dimensions of data

10: Have a low resource footprint

- If you have it, and you don't need it, it could very well be slowing your program down.
- You may be able to trim some unnecessary use of resources, if…
  - You are reading or writing data you don't actually need.
    - … or processing more dimensions of data at once than you need to.
  - You are formatting your data for output before you are finished processing it.
  - You are leaving code which was used for debugging in the production version of your program.

11: Profile your code

- Profiling your program code is simply investigating how it runs in order to find out where most of the run time is spent.
  - When you know which parts of the code take up the most run time, you know which parts of the code to focus your optimization efforts on.
- On a high level, this is relatively straightforward to do. Simply ending your DATA steps and PROCs with RUN or QUIT will ordinarily result in SAS printing their run times to the log.

Profiling, and code within a DATA step

- Because a DATA step is structured to do the same things repeatedly, it can at times be tricky to figure out where DATA step code is bogging down.
- You can estimate relative percentages of run-time by creating copies of your data step with specific portions of the program code removed, then comparing their run time to the run time of the entire DATA step.
  - You can also use a similar approach to estimate which functions and methods are faster than others by enclosing them in otherwise identical DATA _null_ steps.

12: Other Optimizations are Possible

- Sometimes there isn't a better way to do something, but often there is.
- Keep learning… the more you know, the more tools you'll have to help you make your programs work better.

- Any questions?