

Dear Miss SASAnswers: A Guide to SAS® Efficiency
Linda Jolley and Jane Stroupe, SAS Institute Inc., Cary, NC

ABSTRACT

What if you could send your dilemmas about SAS programming efficiency to an advice columnist? Well...Here's Miss SASAnswers! She's received some questions from curious users—questions such as "What's up with I/O?" and "Where did my memory go?" She would love to share her answers to get you started solving the efficiency problems that plague your programs.

THE QUESTIONS AND THE ANSWERS

**Dear Miss SASAnswers,
My program is taking forever to run. What's the problem?
Signed,
Newbie**

Dear Newbie,

I'm not sure I understand exactly what you mean by "taking forever to run," but here are a couple of simple hints that might help, so give them a try:

1. Reduce Input/Output (I/O) by using WHERE statements or DROP | KEEP statements on the output side. Keep only the variables and observations that you absolutely must have in order to do further processing.
2. Minimize the CPU time that SAS uses by executing the minimum number of statements in the most efficient order. If you are creating a new variable for subsetting, test it as soon as the variable is created. If you are subsetting on an existing variable, use a WHERE statement instead of a subsetting IF statement.
3. When you use SAS repeatedly to analyze or manipulate any particular group of data, create a SAS data set instead of reading the raw data each time. This can be a temporary SAS data set in the WORK library or a permanent SAS data set in some other library, depending on your needs for the data.
4. Several procedures enable you to create multiple reports by invoking the procedure only once. Procedures that use this technique include the SQL, DATASETS, FREQ, and TABULATE procedures. BY-group processing can also minimize procedure invocation, because multiple reports are created from one read of the input data. Using a CLASS statement in the procedures that support it can eliminate presorting the data.

Happy Programming,
Miss SASAnswers

**Dear Miss SASAnswers,
I'm having problems with memory usage. What can I do?
Signed,
Having a Senior Moment**

Dear Having a Senior Moment,

Here are a few suggestions:

1. Use KEEP= and DROP= data set options so that only relevant variables consume memory during processing. Remember that where you place these data set options determines their impact on memory. Use these options on the input SAS data set to save memory.

2. Use a small data set page size to minimize wasted disk space when creating a small SAS data set or a SAS data set that will be accessed in a sparse random pattern using an index or the POINT= SET statement option. When you use a small data set page size, fewer observations are read or written in each I/O operation.
3. Use a small value for BUFNO= when the data will be accessed randomly instead of sequentially, such as when an index or the POINT= SET statement option is used. BUFNO= controls how many data set pages are read or written in one I/O operation.
4. Create a small copy of a large data file with only the observations and variables that will be used by subsequent reporting or analysis steps. This copy can be a WORK data set if your need for it is only temporary.

Just remember that the techniques that reduce CPU and I/O can increase memory usage. So benchmark carefully to balance the need to conserve memory with the need to reduce CPU and I/O.

Happy Programming,
Miss SASAnswers

**Dear Miss SASAnswers,
What is parallel processing, and is it more efficient?
Signed,
Running in a Circle**

Dear Running in a Circle,

Parallel processing refers to processing that is handled by multiple CPUs simultaneously. This type of processing takes advantage of Simultaneous Multi-Processing (SMP) machines that have multiple CPUs and provide threaded I/O and threaded application processing using a thread-enabled operating system.

Threading takes advantage of multiple CPUs by dividing processing among the available CPUs, but some types of threading can be performed using a single CPU.

For SAS[®]9, certain procedures, such as SORT and SUMMARY, have been modified so that they can thread the processing through multiple CPUs if they are available. In addition, the MEANS procedure, parts of the SQL and REPORT procedures, and some of the analytical and data mining procedures have also been thread-enabled in SAS[®]9.

For example, if you submit a SORT procedure step on a data set with 1 million rows and have four CPUs to do the sort, the data is partitioned into “chunks” of data, each containing about 250,000 rows. Each chunk of data is read by one of the four CPUs and sorted individually. The sorted data from each thread is then interleaved and written to the output data set as shown in Figure 1.

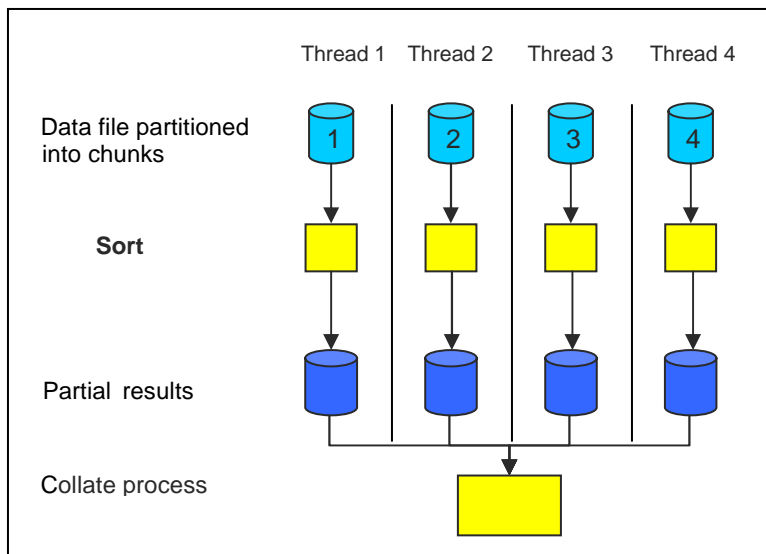


Figure 1. Example Showing SORT Procedure Processing Threaded through Multiple CPUs

Of course, your SAS session might not use four threads; the number of threads depends on many factors. Using multiple threads is actually the default when you submit a PROC SORT step. To disable threading, you need to use the NOTHEADS option on the PROC SORT statement or the NOTHEADS SAS system option.

But just remember that if you use threaded procedures, CPU, I/O, and memory usage might increase. Threading addresses wall clock time—the time from submitting the code until the results are returned. Jobs that take 30 or more minutes of wall clock time usually make better candidates for threading.

Also, threading helps with processes that are CPU-bound, not I/O-bound. CPU-bound processes can receive data faster than they can actually process the data. I/O-bound processes have CPUs that are waiting on data to be retrieved.

Finally, long, skinny SAS data sets are better candidates for threading than are short, wide ones. The threaded operating system needs a lot of observations to effectively use multiple threads.

Happy Programming,
Miss SASAnswers

**Dear Miss SASAnswers,
How can I save programmer time?**

**Signed,
Tired of Typing**

Dear Tired of Typing
Here are a few suggestions:

1. Use tools such as SAS® Enterprise Guide®, SAS® Data Integration Studio, or SAS® Enterprise Miner™ to simplify many of your programming tasks. Not only do these tools organize your work, they also write code behind the scenes.
2. Use SQL for code simplification. A single SQL statement is generally easier to code and understand than a long series of SORT and DATA steps.
3. Use a procedure for report writing instead of writing DATA _NULL_ steps. If you must have greater control over the output, consider using the REPORT or TABULATE procedures, which both offer many ways to customize the output.

4. Document, document, document. There is nothing more frustrating than trying to figure out what the SAS code is supposed to do. Documenting your programs will save precious time for you and anyone else who needs to look at the programs.
5. Use macros for redundant code. Any time you find yourself copying a complete SAS step or a portion of a SAS step and changing only one or two pieces of code, you have a candidate for a macro program to do it for you.
6. Use macro variables for text substitution in programs that you are editing often. You can change the variable value in one place and have the changes propagate through your code. This reduces the chance for typing errors.
7. Assign descriptive and meaningful variable names. It's difficult to know what the values of the variable "X" represent. It's easy to know what the values of the variable "LastName" represent.
8. Start each statement on a separate line and indent statements that belong to a step. Although SAS doesn't care how you physically type your code, following these conventions will make it easier to read your code and understand your code logic.
9. Type `RUN;`, especially if you are using the Output Delivery System (ODS). If you omit the end-of-step boundary when using ODS, you will not get the expected results. Get in the habit of typing explicit step boundaries to prevent misinterpretation of your code by SAS.

Happy Programming,
Miss SASAnswers

**Dear Miss SASAnswers,
Which is more efficient, the DATA step MERGE or PROC SQL?
Signed,
Put Them Together**

Dear Put Them Together,

Well, that depends on your data and what you mean by "efficient." You have to know the relationship between the tables, the sparseness or density of matches, the size of the tables, and the availability of an index or sort flag.

When data sets are large and unsorted, the SQL inner join might outperform SORT and MERGE. If you have a long series of SORT and DATA steps, the SQL inner join might be easier to code and read. In most cases, a DATA step MERGE statement generally outperforms an SQL outer join, even taking sort resources into account. One exception is a very sparse match join when you only want the observations with matching key values. Also, keep in mind that the SQL procedure and the DATA step MERGE do not provide the same results if you have a many-to-many match.

Because there are no hard and fast rules about the efficiency of MERGE versus SQL, you'll just have to benchmark them to find out how they perform with your data. Keep in mind that benchmarking involves multiple submissions of the two sets of code against the data, each submission running in a separate SAS session. Throw out any outliers in terms of resources and average the 3 to 5 submissions that are left.

Happy Programming,
Miss SASAnswers

Dear Miss SASAnswers,

I need to eliminate duplicates for a data set based on an ID value. I've been sorting the data and then using a DATA step with FIRST. and LAST. processing because I need to save the duplicates in another data set. Is there an easier way?

**Signed,
De Dup**

Dear De Dup,

Yes. In SAS[®]9, there's a new DUPOUT= option for the SORT procedure that will write the duplicates to a SAS data set. This saves reprocessing of the sorted data with a DATA step.

```
proc sort data = alldata nodupkey
  out = unique
  dupout = duplicates;
  by y;
run;
```

The effect of this option is identical to the processing you've been doing in the DATA step: one observation is written to ALLDATA, and all other duplicate observations are written to DUPLICATES.

Happy Programming,
Miss SASAnswers

Dear Miss SASAnswers,

My data sets are huge! Is there any help for me?

**Signed,
Overweight from Data**

Dear Overweight from Data,

Take heart, you are not alone! There are some simple ways to reduce the size of your data.

1. Compress the data by using COMPRESS=CHAR|YES for data sets that have long character data that contain many blanks and COMPRESS=BINARY for data that has long observation length (over 1000 bytes). As usual, benchmark to make sure which is appropriate. Remember that the cost of compression is the CPU time necessary to uncompress the data when it is read into a DATA or PROC step, and that BINARY compression takes considerably longer to uncompress than CHAR compression.
2. Reduce the length of numeric integer variables, as long as they are not large integer values. Do *not* reduce the length of decimal values, as the length reduction actually truncates the length of the mantissa that is stored. As with compression, the cost of reducing the length of numeric integer values is the CPU time necessary to re-expand them to 8 bytes when they are read.
3. Store the data using the SAS Scalable Performance Data (SPD) Engine. The SAS SPD Engine delivers data to applications rapidly because it organizes the data into a streamlined file format that takes advantage of multiple CPUs and I/O channels to perform parallel I/O processing. The file format is such that the data set and the indexes are stored in chunks to enable parallel WHERE selections, parallel I/O data delivery to applications, and implicit sorting on BY statements. The SAS SPD Engine is available in Base SAS[®]9 on selected operating systems.

- And one more idea. The SAS Scalable Performance Data Server, which you can license for selected UNIX or Windows platforms, combines the capabilities of the SAS SPD Engine (which is part of Base SAS) with concurrent update access by multiple users. In addition, the SAS SPD Server extends the SQL procedure to provide the functionality of SQL Pass-Through similar to that available with the SAS/ACCESS products. When SQL Pass-Through is used, the SAS SPD Server can optimize queries—more than the SAS SPD Engine or any procedure can! In addition, the SAS SPD Server provides additional user- and access-security features.

Happy Programming,
Miss SASAnswers

**Dear Miss SASAnswers,
I spend a lot of time merging, and I mean a *lot* of time. Is there an alternative that would be faster?**

**Signed,
Muggy Merger**

P.S. These are one-to-many merges, if that makes a difference.

Dear MM,

Your PostScript helps a lot in giving you some suggestions. Depending on the type of your BY variable and the amount of memory available, here are alternatives to a MERGE:

- An array
- A DATA step HASH object
- A format

To explain these, let's call the data set with the multiple observations for each key value the *detail* table and the data set with the one observation for each key value the *lookup* table.

- You can load the *lookup* table into an array if the key values are numeric. Then you can use the numeric key from the *detail* table to do the lookup. Here's a simple example:

LookUp	
ID	Type
1	Dog
2	Cat
...	
...	
20	Bird

Detail		
ID	Amount	Purchase
1	.50	Biscuit
1	.75	Toy
1	1.00	Food
...		
20	5.00	Seed

```
data combine;
  keep ID PetType Amount Purchase;
  array pet{20} $ 9 _temporary_;
  if _n_=1 then do i=1 to numobs;
    set lookup nobs=numobs;
    pet{id}=type;
  end;
  set detail;
  PetType=Pet{id};
run;
```

2. Have you not heard of a HASH object in the DATA step? Well, that's because it is new in SAS[®]9. It is a second technique of storing the *lookup* table in memory, but the advantage of the HASH object over the array is that the key value can be character or even consist of more than one variable. Here's the previous example using the HASH object.

```
data combine;
  length Type $ 9;
  keep ID Type Amount Purchase;
  if _n_=1 then do;
    declare hash pet(dataset:'lookup');
    pet.definekey('id');
    pet.definedata('Type');
    pet.definedone();
  end;
  set detail;
  if pet.find()=0 then output;
run;
```

3. Familiarity is a great advantage when it comes to using formats! And you can load the format from the *lookup* table. Let's do the example again, this time using a format.

```
data fmt;
  set lookup(rename=(id=start type=label));
  retain fmtname 'Pet';
run;

proc format cntlin=fmt library=sasuser;
run;

options fmtsearch=(sasuser);
data combine;
  keep ID PetType Amount Purchase;
  set detail;
  PetType=put(id,Pet.);
run;
```

If you can use an array for the table lookup, it is the most efficient. If your key is character or composite, then using the HASH object should be your choice. On the other hand, both techniques require that you repeat the appropriate SAS code every time you use the lookup table. A format can be stored permanently, as in the example, and that makes using the lookup table much simpler, because the code to create the lookup table needs to be run only one time.

Happy Programming,
Miss SASAnswers

CONCLUSION

There are many considerations to make your programming in SAS more efficient. Although it seems as though the answer to your efficiency questions is always "It depends," the truth is that it really does depend on what resource you are trying to save. Remember that lowering the use of one resource frequently increases the use of another resource. You have to balance the use of these scarce resources depending on your particular situation.

We hope we have given you some ideas to start you thinking about efficient SAS programming techniques. The more SAS code you write and submit, the better you will get at recognizing which techniques work with your data on your system.

Miss SASAnswers may not exist in real life, but there are many places that you can find assistance. When you have questions, help is just a phone call, e-mail, or Web search away. Contact SAS Technical Support with questions, or check out SAS training that is offered on the Web or in the classroom.

Happy efficient coding!

ACKNOWLEDGMENTS

The authors thank Kent Reeve, Kay Alden, Richard Bell, and Donna Bennett for their help in reviewing this paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors:

Linda Jolley, Technical Training Specialist
SAS Institute Inc.
Kansas City Regional Office
9401 Indian Creek Parkway
Overland Park KC 66210
Phone: (913) 491-1166
Email: linda.jolley@sas.com

Jane Stroupe, Technical Training Specialist
SAS Institute Inc.
Chicago Regional Office
Two Prudential Plaza, Suite 1600
Chicago IL 60601
Phone: (847) 367-7216
Email: jane.stroupe@sas.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.