Paper 223-30

The Rolling Stones were Right (Time is On Our Side): An Application of SAS® Multiprocessing

Trang Lance, Department of Veterans Affairs, OQP PACE, Morrisville, NC

ABSTRACT

Previously, the SAS software is a single-threaded process, where a thread is defined as a list of instructions to perform a certain task. This process means that each individual SAS session thinks that it is running on only one CPU. In this single-CPU system, the processor is forced to assist in the input/output operations, which means it cannot perform calculations when I/O processing is taking place and vice versa. SAS® Scalable Architecture (SSA) Version 9 now allows a single SAS session to leverage multiple I/O channels and multiple CPUs to drastically cut run-time for large jobs. SAS SSA is a tool necessary to leverage symmetric multiprocessor (SMP) architectures. Thus, allowing multiple process threads to execute in parallel on separate CPUs while sharing a common memory space. In addition, modern operating systems use symmetrical multiprocessing, where the system can load multi-threaded applications among different processors, it would be advantageous to optimize all of the processors to reduce time-to-solution. As a result, it will try to balance the computational load rather than having certain processors do specific types of processes. Using SAS and the operating system simultaneously, we hope to use all resources available on a machine and maximize benefits gained on machine with multiple CPUs, I/O channels and disks where there are large amounts of data to be manipulated. This paper presents applications of this process using Veterans Health Administration SHEP data, which uses administrative VHA data, some of which contains over 70 million patient records.

INTRODUCTION

Over time, the amount of data being collected, stored, and analyzed has increased exponentially. In conjunction with this increase, there has been a rise in the demand for turning the volumes of data collected into useful and actionable information in a timely way so that organizations can be responsive to changes in their area of concern. Organizations are confronted with the dilemma of how to systematically approach both of these issues within a framework of limited time and resources, with the overarching goal being to reduce the time-to-solution cycle. To reduce the time-to-solution cycle one must begin by seeking out and identifying what portion of the applications takes up the most time and determining if those parts are computer intensive or if they are I/O bound.

Hardware vendors have aided in this endeavor by supplying faster CPU speeds, higher I/O rates and multi-processor architecture. These machines are called symmetric multiprocessing (SMP) systems because they have multiple CPUs and an operating

environment that can spawn and manage multiple pieces of executable code called threads. By definition, a thread is a single, independent flow of control through a program or within a process that is scheduled by the operating environment. The threaded reads only augment performance when the data set is large and optimal performance occurs when partitions are similar in size. As a result, it takes advantage of the multiple CPUs by dividing processing among the available CPUs, which results in the reduction of elapsed time of the SAS data step.

SAS Version 9 now supports parallel processing, which takes advantage of these SMP (symmetric multiprocessor) machines and provides performance gains for threaded I/O and threaded application processing. Parallel processing is the division of an application into modules of work that can be executed simultaneously on the same machine or different machines. It is also referred to as multiprocessing or asynchronous processing. In threaded I/O, some applications can process data quicker than the data can be delivered to the application. As a result, the application becomes I/O-bound, which means the application cannot keep the available CPUs in use. The solution for these CPU-bound applications is to increase processing power.

It is also known that the ability to process data in threads depends on the process itself. There are some types of processing that are not suited to threading, while other types of processing can benefit from it greatly. For example, procedure of sorting can be performed in multiple threads. This is accomplished by dividing the sorting process into separately executable sorting processes. One or more of these threads will perform a process in each CPU. The sorted data from each thread is then set together and written to the output data set. Although sorting can be performed in threads, the joining process and the outputting processes are non-threadable.

VHA DATA

One of the charges of our office is to obtain valid and reliable evaluations of VHA inpatient and ambulatory care and services from veterans through a standardized questionnaire. The use of consistent methodology nation-wide allows trend over time analysis within VHA. The office sends out monthly surveys where approximately 12,000 inpatient cases and 36,000 outpatient cases are selected. The administrative database used in creating the sample frames has numerous files that range from 500 thousand to over 70 million observations and with an average of 50 variables.

In addition to monthly sample selections, there are monthly and quarterly analysis and reporting that are performed on a national, network, and facility level. Veteran Healthcare Service Standard (VHSS) scores are calculated and risk-adjusted at the patient level to compensate for the effect of any baseline differences among the reporting stratum on those characteristics know to have the greatest influence on patient's rating of medical care.

THREADS and CPUs

Given the volume of data coming in and going out and the numerous analyses, it would be extremely beneficial for our office to reduce the run-time to solution. In order to see how much time is saved when using the multiprocessing, various programs were run using both SAS version 8 and SAS version 9. When using SAS version 9, the following options were used to invoke the multi-threading multiprocessing options:

```
OPTIONS THREADS=YES CPUCOUNT=4;
```

These two global options do not necessarily control the behavior of the threaded procedure but will influence it. Each procedure is free to interpret the settings of these options as appropriate.

The THREADS option turns on or off the threading procedure. When on, it notifies procedures capable of threading that threading can be used. The processes that are suitable for threading consists of procedures involved with sorting, grouping and summarizing. These procedures are:

- MEANS/SUMMARY
- REPORT
- SORT
- SQL
- TABULATE

The CPUCOUNT option controls the number of CPUs to utilize but it does not control the number of threads. It can be set to ACTUAL instead of an integer, which means that the SAS system will determine how many CPUs are available to its process space and set CPUCOUNT to that value. It can be set low or high to alter the behavior of the threaded procedure.

DISTRIBUTED PROCESSING

Recall that parallel processing is the division of an application into modules of work that can be executed simultaneously on the same machine or different machines. Some benefits of parallel processing include taking advantage of multiple processors on a SMP single machine or a network of machines; completes the processing of a program in less total elapsed time than it would to execute the same job sequentially; and augmented usage of underutilized CPUs. In addition, Version 8 allows for execution of independent tasks in parallel and Version 9 allows for execution of select dependent tasks in parallel. However, there are overhead costs when separate SAS sessions are started and there would be additional requirements on the I/O subsystem.

There are a number of extensions of parallel processing. For instance, there is independent parallelism, which is when the execution of separate tasks does not have any interdependencies. Another type is pipeline parallelism, which is when multiple steps depends on each other but the execution can overlap and the output of one step is streamed as input to the next step. And finally, there is distributed processing, which is where one process (a client or local host) is requesting services or data from another

process (a server or a remote host) to execute on a different machine. It maximizes all computing resources by dividing applications into tasks to be performed on different computers. This last process has been instrument to our office in reducing the time-to-solution cycle.

SIGNON PROCEDURE

The SAS spawner (Spawner.exe) is provided with SAS Base and is used to facilitate connections from workstations running SAS. It is an alternative method to signing on to a TELNET Daemon on a remote host to a remote machine. The spawner program resides on a SAS/CONNECT remote host and it waits for requests to connect to the remote machine. When it receives a signon request, it will invoke the remote SAS session.

There are two methods to connect a spawner on a remote machine. The first is to issue the spawner command in the subdirectory where SAS is installed. The general form of the spawner command is:

```
SPAWNER –C TCP,
```

where –c option specifies the communication access method.

The other method is to automatically start the spawner whenever the remote machine is started or rebooted. Specific instructions for installing the spawner as a windows service are detailed in SAS TS638 for Windows NT/XP/2000. Note that the SAS TS does not mention Windows 2003 compability, yet we were able to successfully install and utilize the service on Windows 2003 server.

To invoke the spawner, the following %LET statement is used to name the remote host:

```
%LET mvar = remote-name;
```

Then the following SIGNON statement is needed:

```
SIGNON <fileref | ‘file-specification’ | remote-session-ID>;
```

Here is an example of an invocation that was used in our analysis:

```
%LET ANALYSIS=ANALYSISSERVER;
SIGNON ANALYSIS;
```

Note, when submitting jobs to the spawner service on a server that that also houses the data sets, the above code will work. However, when the spawner service is on a server that does not house the data sets and it is attempting to access the data sets through libnames, the following error will occur:

```
AUTHENTICATION IS REQUIRED TO ACCESS THE NETWORK SHARE
```

```
IN THE WINDOWS ENVIRONMENT OR USER DOES NOT HAVE
APPROPRIATE AUTHORIZATION LEVEL FOR LIBRARY.
```

By adding your windows username and requesting SAS to prompt you for your password, you can bypass this error.

```
%LET ANALYSIS=ANALYSISSERVER;
SIGNON ANALYSIS USER=DOMAIN\USERNAME'
        PASSWORD=_PROMPT_;
```

When the remote processing is complete, the following SIGNOFF statement is needed:

```
SIGNOFF <remote-session-id>;
```

Here is an example of the SIGNOFF statement:

```
SIGNOFF ANALYSIS;
```

SUBMITTING JOBS REMOTELY

After a connection to a remote machine has been established, code is sent to execute on the remote machine by enclosing the code in an RSUBMIT block.

```
RSUBMIT <remote-machine-name>;
        code to be processed remotely
ENDRSUBMIT;
```

One important process that is used in the remote coding is the data transfer service. It enables data to be transfers from a local machine to a remote machine and vice versa. The procedures that will enable this are transfer of data is UPLOADS and DOWNLOADS, respectively.

Here is an example of the RSUBMIT, UPLOADS, and DOWNLOADS statements:

```
RSUBMIT ANALYSIS WAIT = NO;
OPTIONS NOFMTERR THREADS=YES CPUCOUNT=4;
LIBNAME SE04 "C:\\SHEP\PDW_ADM\AUSTINDATA\SE\2004\DATA";
LIBNAME LIBRARY
"C:\\SHEP\PDW_ADM\AUSTINDATA\STAK_FORMATS";

PROC UPLOAD DATA = VHA;
RUN;

DATA SE04;
  MERGE SE04.SE04 (KEEP=SCRSSN STA5A VIZDAY CL CLC)
     VHA (KEEP=SCRSSN IN=SHEP);
```

```
  BY SCRSSN;
  STAKOLD = PUT(STA5A,$STAK6L.);
  IF SHEP THEN OUTPUT;
RUN;

PROC DOWNLOAD DATA = SE04;
RUN;
ENDRSUBMIT ANALYSIS;
```

SYNCHRONIZING TASKS

If multiple independent remote sessions are being processed before a dependent task is performed, the WAITFOR statement is needed. It tells the local SAS session to not execute any additional code until at least one of the tasks or all tasks has completed. The general form of the WAITFOR statement is:

```
WAITFOR _ALL_ | _ANY_ task1 task2 …taskn;
```

Here's an example of the WAITFOR statement;

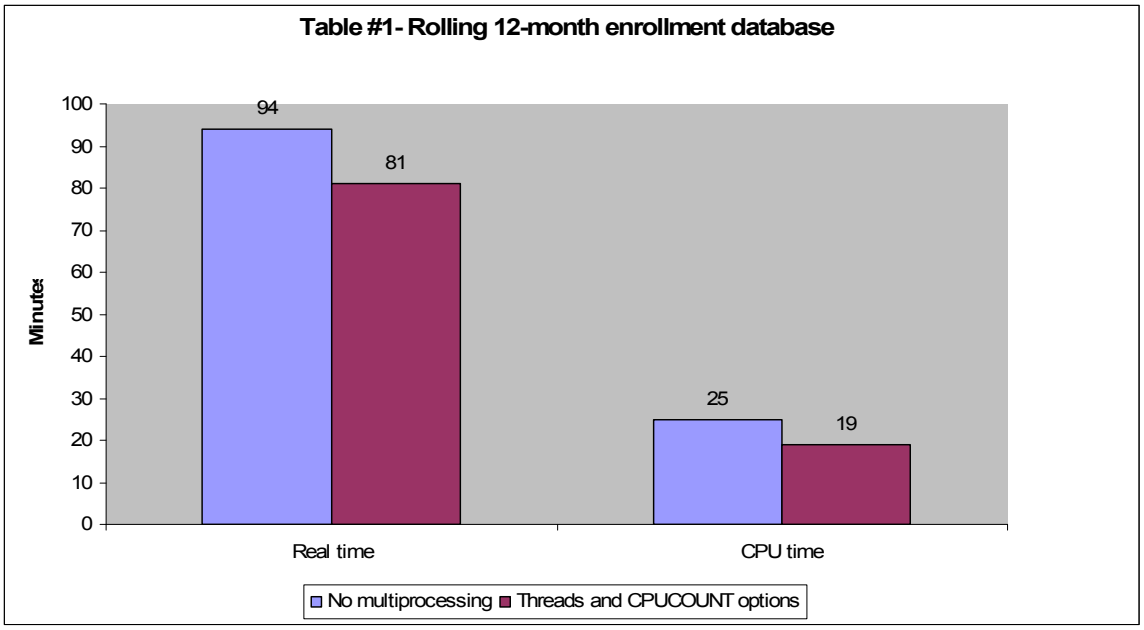
```
WAITFOR _ALL_ CARTER HAMMOND ANALYSIS;
```

RESULTS

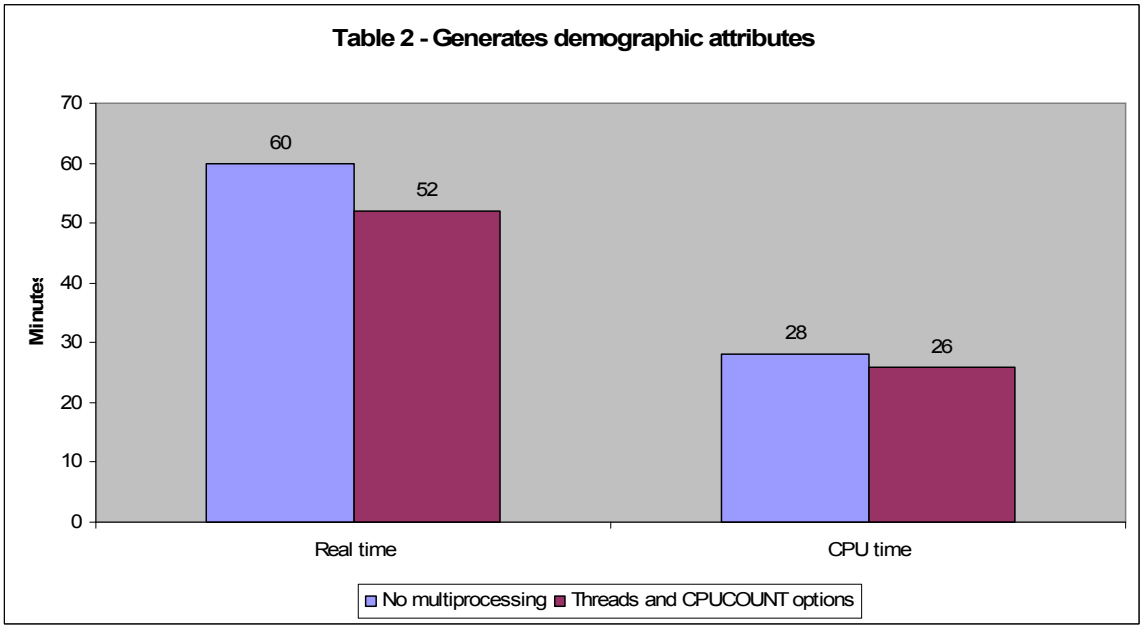
All other factors were held constant to reduce instrumental bias. For instance, the comparison programs were run in the evenings to ensure that no other programs were running, which would compete for CPU time. In addition, all comparisons were performed on the server, a Dell PowerEdge 6650 with 4 x86 Family 15 Model 2 Stepping 5 GenuineIntel ~1993 Mhz processors. The operating system is Microsoft® Windows ® Server 2003, Enterprise Edition. The total physical memory is approximately 4.1 GB with 8.85 GB total virtual memory.

The first four tests looks at how threads and the use of more than one CPU affects the real time and CPU time. The last test incorporates the distributed processing through remote sessions.

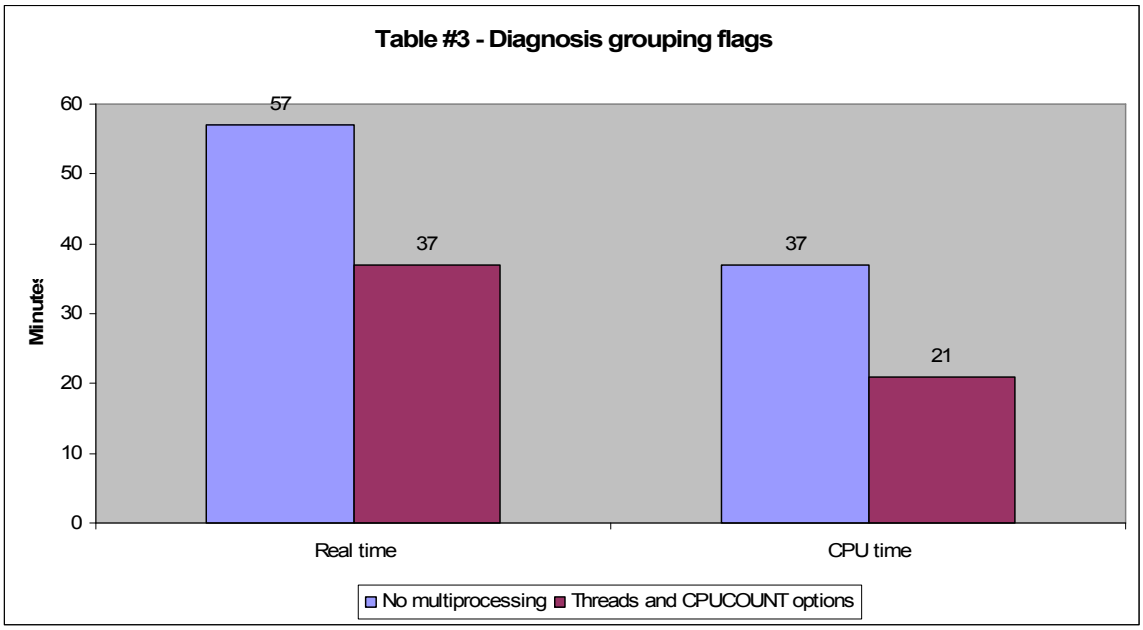
The first comparison creates a rolling 12-month enrollment database with approximately 7.5 million observations and 82 variables. The program is mainly based on data steps, proc sort, proc cimport, proc sort, and proc contents. The real time processing was 1:34:13.62 for SAS 8 and 1:21:21.81 for SAS 9. As a result, the real time processing decreased by almost 13 minutes or approximately 16% using SAS 9 and its multithreading capabilities. The CPU time was 25:27.21 for SAS 8 and 19:46.32 for SAS 9. Thus, the CPU time decreased by approximately 5.7 minutes or almost 29%. (See Table 1.)



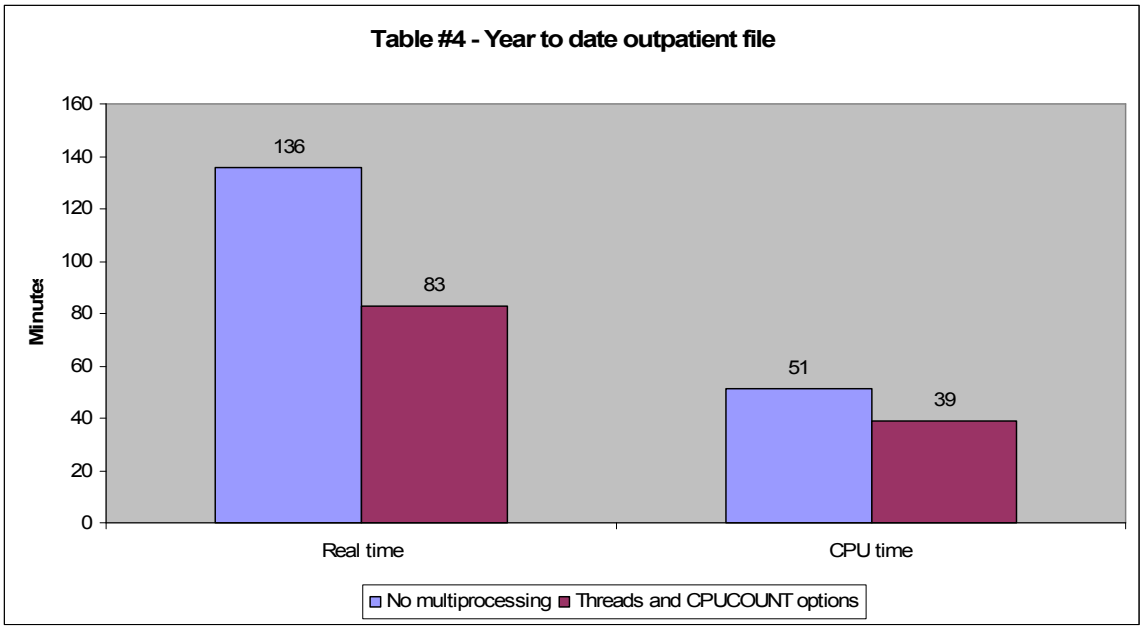
In the second comparison, the program generates demographic attributes (race, gender, date of birth, and marital status by analyzing 3 years of inpatient and outpatient data which contains over 500,000 and 46 million observations, respectively in each year's data. The program mainly consists of a number of proc sorts and data steps. The real time processing was 1:00:46.65 for SAS 8 and 52:31.65 for SAS 9. As a result, the real time processing decreased by almost 8 minutes or approximately 15% using SAS 9 and its multithreading capabilities. The CPU time was 26:09.71 for SAS 9 and 28:36.81 for SAS 8. Thus, the CPU time decreased by approximately 2 minutes or almost 7%. (See Table 2.)



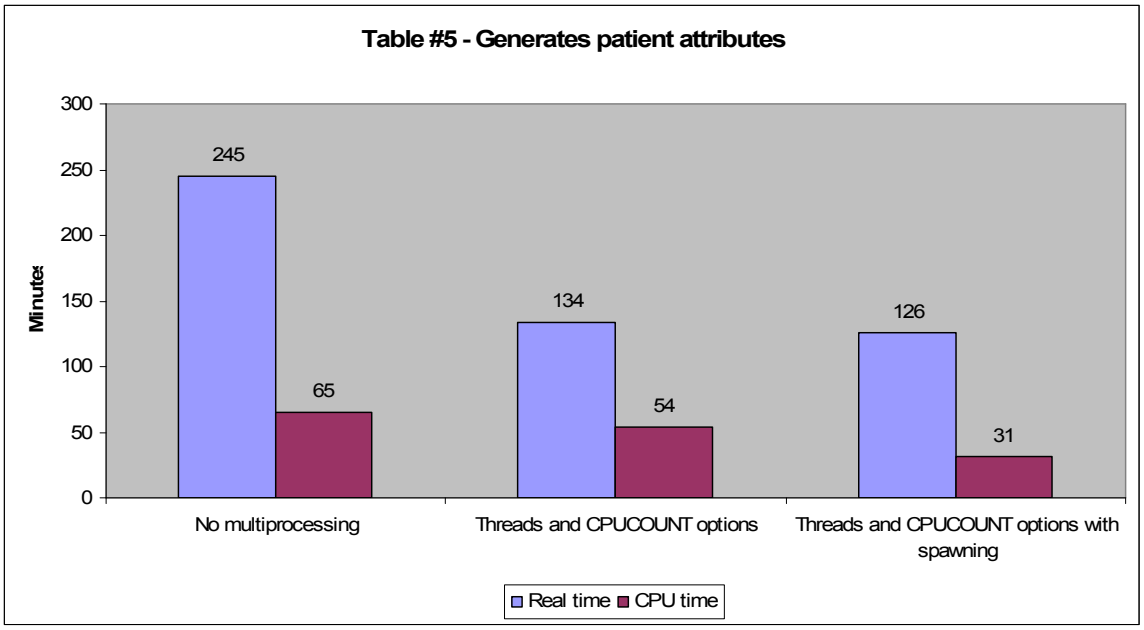
In the next comparison, the program creates 36 diagnosis grouping flags reading 2 years of inpatient and outpatient data which contains approximately 200,000 and 70 million observations, respectively in each year's data. This program consists mainly of one large data step and one proc sort. The real time processing was 57:33.46 for SAS 8 and 37:17.21 for SAS 9. As a result, the real time processing decreased by almost 20 minutes or approximately 35% using SAS 9 and its multithreading capabilities. The CPU time was 36:50.92 for SAS 8 and 21:30.26 for SAS 9. Thus, the CPU time decreased by approximately 16 minutes or almost 43%. (See Table 3.)



In the fourth comparison, a year to date outpatient file with approximately 50 million records and 80 variables is created. This program consists mainly of one large data step and one proc sort. The real time processing was 2:16:27.63 for SAS 8 and 1:22:48.76 for SAS 9. As a result, the real time processing decreased by almost 44 minutes or approximately 32% using SAS 9 and its multithreading capabilities. The CPU time was 50:34.92 for SAS 8 and 38:53.06 for SAS 9. Thus, the CPU time decreased by approximately 12 minutes or almost 24%. (See Table 4.)



In the final comparison, patient attributes based on three years of data are created. The program consists mainly of proc sort, proc cimport, proc freq, and data steps. When looking at how threads and additional CPU usage, the real time processing was 4:05:57.82 for SAS 8 and 2:14:07.45 for SAS 9. As a result, the real time processing decreased by almost 111 minutes or approximately 83% using SAS 9 and its multithreading capabilities. The CPU time was 1:05:58.23 for SAS 8 and 53:50.21 for SAS 9. Thus, the CPU time decreased by approximately 11 minutes or almost 20%. When adding remote sessions, the real time processing was 2:05:57.67. As a result, the real time processing decreased by almost 119 minutes or approximately 94%. The CPU time was 31:24.17, which is a decrease by approximately 23 minutes or almost 43%. (See Table 5.)



DISCUSSION

It is evident that the threading and CPUCOUNT options will greatly decrease real time and CPU time. However, there is great variation in benefit. This is due to the fact that certain processes are suitable for threading. These processes consist of procedures involved with sorting, grouping and summarizing. In addition, the number of data steps and procs in a program will make difference. In the programs where there were only one or two data steps and proc sorts, the improvement was much higher.

Spawning in conjunction with the threading and CPUCOUNT options will decrease real time and CPU time even more by utilizing additional computers/servers available on a network. However, it would be best to write the program sequentially to ensure the program is running correct and producing what the programmer intended before trying to spawn the code. Also, some consideration is needed to see if the benefits of spawning are reduced by the uploading and downloading of files and the management of a more difficult style of programming.

CONCLUSION

SAS Version 9 has provided many opportunities for improvement in real time and CPU time. It along with the new hardware technology could help organizations who are confronted with the dilemma of how to work within a framework of limited time and resources but with the overarching goal to reduce the time-to-solution cycle. The use of SAS multiprocessing and multithreading has been a great benefit for our office. It has greatly reduced our processing time but at the same time increased the actual usage of our multiple processors. We are still in the early learning phase of what Version 9 has to offer but what we have learned and used so far has been extremely beneficial.

REFERNCES and RESOURCES

Bentley, J. E., (2000) "An Introduction to Parallel Computing", in SUGI 25 Conference Proceedings, Cary, NC: SAS Institute.

Ray, R., (2003) "An Inside Look at Version 9 and Release 9.1 Threaded Base SAS ® procedures", in SUGI28 Conference Proceedings, Cary, NC: SAS Institute.

Doninger, C., (2002) "Up and Out: Where We're Going with Scalability in SAS ® Version 9, in SUGI27 Conference Proceedings, Cary, NC: SAS Institute.

Distributed and Parallel Processing with the SAS System Course Notes

ACKNOWLEDGEMENTS

I would like to recognize the contributions of the following:

- Jim Schaefer, MPH
- Thomas Bullock
- Eric Cohen, CISSP
- Stacey Campbell, MPH

CONTACT INFORMATION

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

Trang Lance, MPH
Statistician
Department of Veterans Affairs, OQP PACE
601 Keystone Park Drive, Suite 800
Morrisville, NC 27560
919-993-3035 x. 227
919-993-3053
Trang.Lance@med.va.gov