Paper 279-28

Developing Client/Server Applications to Maximize SAS 9 Parallel Capabilities

Cheryl Doninger, SAS Institute Inc.

ABSTRACT

Parallel SAS processes, multiple threads within a single SAS process, parallel SAS servers, distributed computing, grid computing, cluster computing – how might all of these choices affect the way that you develop your client/server SAS applications? This paper will address some of these areas and what you should understand when you develop or run client/server SAS applications in these environments. Details will be given on how MP CONNECT can be used in a grid environment as well as in conjunction with the new Scalable Performance Data Engine (SPDE engine), the SAS 9 threaded procedures, and other parallel features to maximize performance and take advantage of each of these environments.

INTRODUCTION

SAS 9 is built around four cornerstones:

- Usability
- Interoperability
- Manageability
- Scalability

Scalability is the focus of this paper. For the purposes of this paper scalability is defined as improving the performance of a SAS application by leveraging computing resources as they are added.

In order to achieve better performance of your SAS applications through scalability, the following requirements must be met:

- your application must have pieces that can be overlapped or executed in parallel,
- you must have both compute and i/o resources that can run these pieces in parallel, and
- you must implement your application with software that can leverage the available hardware resources.

SAS 9 has the capability to leverage the available hardware resources to both scale up and scale out your applications. SAS provides scalability in two ways:

- parallel SAS processes
- parallel threads within a SAS process.

A ***SAS process*** is made up of many pieces (execution units, data structures, resources, etc.). These are collectively referred to as a SAS process or session. A ***process*** corresponds to an operating system process. It has a largely private address space. It is scheduled by the operating system and its resources are managed by the operating system at the lowest level. Multiple SAS processes make use of multiple processors on a symmetric multi-processing (SMP) machine but can also be run across any number of remote single or multi-processor machines on a network. When running multiple SAS processes on an SMP machine, SAS does not participate in scheduling a specific process to a specific processor; that is controlled by the operating system.

A process consists of one or more ***threads.*** A thread is also scheduled by the operating system, though the running process may influence the behavior of threads using synchronization techniques. All threads in a process share an address space, and must cooperatively share the resources of the process among themselves. Multiple threads make use of multiple processors on an SMP machine but cannot be executed across machines. When running multiple threads within a SAS process, SAS does not participate in scheduling a specific thread to a specific processor; that is controlled by the operating system.

Scaling up, from a hardware perspective, means increasing the number of processors, disk drives, I/O channels, etc. on a single server machine. From a software perspective, scaling up means being able to leverage the multiple processors, disk drives, I/O channels, etc. on a single server machine. ***Scaling out***, on the other hand, means adding more machines, not bigger machines. From a software

perspective, scaling out means being able to reach out across a network and run portions of an application across the hardware available on the network. When you scale out, the size and speed of an individual machine does not limit the total capacity of your network.

With SAS 9 multiple threads are used to scale up and make use of multiple processors in SMP hardware. Multi-threading has been incorporated into much of SAS 9 including many of the SAS servers, several performance critical SAS procedures, and many of the SAS engines. Multi-threading is used for both compute intensive portions as well as I/O intensive portions in order to get data into SAS as quickly as possible and reduce the total elapsed execution time as much as possible.

Multiple SAS processes are used to both scale up and scale out. By running multiple processes on an SMP machine the operating system can schedule the processes to different processors to make use of all of the hardware resources on the machine. In addition, by running multiple SAS processes across the machines available on a network you can utilize idle remote processors and put any number of slow, inexpensive machines to work in parallel on a job, turning them into a very valuable, powerful, but still inexpensive computing resource. MP CONNECT is the facility available within SAS that provides the interface for spawning, synchronizing, and managing multiple SAS processes.

Multi-threading and multiple SAS processes (MP CONNECT) are not mutually exclusive and in fact, for some applications, the biggest gains in performance come from applying a solution that incorporates both. Provided you have the hardware resources to support it, you can use MP CONNECT to run multiple SAS processes with each process making use of multi-threading. When doing this on an SMP machine it may be necessary to set some options in each of the SAS processes to tune the amount of threading done by each process to avoid the sum of the processes and threads from overloading your system. The amount of tuning will depend on your individual application, data, and hardware configuration. On the other hand, when doing this across multiple remote machines and each SAS process is running on a physically separate machine, it could be desirable to let the threading within the process fully utilize the individual machines.

The remainder of this paper will cover three different scenarios that involve using MP CONNECT in a client server environment in conjunction with the multi-threaded SAS 9 features to maximize the performance of three different applications. First, a brief introduction will be given for MP CONNECT to highlight why and how you can use it to scale some of your own applications. Then the implementation details and results of the following scenarios will be described:

1. using MP CONNECT in a grid environment
2. using MP CONNECT and threaded I/O
3. using MP CONNECT and threaded SAS procedures.

MP CONNECT

Multi-process CONNECT (MP CONNECT) allows you to perform work in parallel and coordinate the results into your original SAS session for the purpose of reducing the total elapsed time necessary to execute a particular application. Initially with SAS 8 MP CONNECT addressed ***independent parallelism*** by enabling you to syntactically identify independent units of work in a SAS job stream and execute those units of work in parallel. In SAS 9, MP CONNECT also addresses ***pipeline parallelism*** by providing a convenient way to pipe data from one SAS data step or SAS procedure to another to allow overlapping execution of dependent SAS steps. By dividing time-consuming tasks into multiple units of work and executing these units of work in parallel, a job can be performed in substantially less time than if each task is performed sequentially.

MP CONNECT provides a convenient syntax for spawning *n* SAS sessions to simultaneously execute *n* tasks and coordinate the execution and results of all *n* tasks into the original SAS session. The *n* SAS sessions or processes can all execute on the same machine with each session or process running on a separate processor. In addition, MP CONNECT has the flexibility of being able to spawn multiple SAS sessions to run on any number of remote machines across a network. The remote machines can have either single or multi-processor capabilities.

MP CONNECT is useful for either distributing multiple independent tasks to execute in parallel as well as to "divide and conquer" a large problem by breaking it up into many smaller pieces and repeatedly distributing the pieces to multiple processes until the problem is solved. In addition,

MP CONNECT can execute a SAS process running one SAS data step or procedure which pipes its output as input to another SAS data step or procedure running in another process. This execution overlap can reduce the time to solution as well as reduce disk space requirements by eliminating the write to disk of intermediate temporary SAS data sets.

MP CONNECT is part of SAS/CONNECT® and was initially available with SAS 8. It works at the process level creating multiple SAS sessions or processes to execute in parallel. Because the interface to MP CONNECT is syntactic, it does require that you analyze your application to identify parallel tasks and then insert the appropriate syntax to cause these tasks to execute in parallel.

MP CONNECT AND GRID COMPUTING

Grid computing is not a new concept but one that has gained recent renewed interest and activity for a variety of reasons:

- IT costs continue to spiral and grid computing offers a less expensive alternative to large server platforms.
- The sheer volumes of data that need to be processed can have very large computing resource requirements.
- Collaboration and sharing of data and computing resources is becoming a necessity in many industries including but not limited to: computer manufacturing, financial services, life sciences, industrial manufacturing, and government.

It is difficult to give a single complete definition of grid computing. For the purposes of this paper, we will define ***grid computing*** as transparent, secure, coordinated computing resource sharing and collaboration. The ***grid*** of resources that are shared could be inter-department sharing within the same organization or across multiple organizations. SAS is able to leverage a grid environment through the functionality available in MP CONNECT.

Description of the Scenario

In this first scenario MP CONNECT will be used in a grid environment for the purpose of parallelizing data analysis of a toxicogenomics microarray study. In this example we utilize data from microarrays, a popular new scientific instrumentation platform enabling the assessment of the expression levels of thousands of genes simultaneously. The data arise from an investigation conducted at the Microarray Center of the National Institute of Environmental Health Sciences (NIEHS), Research Triangle Park, NC, designed to reveal the chemical-specific profiles of two classes of toxic compounds (peroxisome proliferators and enzyme inducers). These compounds were introduced into a set of genetically and environmentally identical rats, and their effects on 1700 genes were measured in livers at 1-day and 14-day time points (Hamadeh et al, 2002a, 2002b).

After suitably normalizing the data from the microarrays, a common initial analysis is to fit a statistical model to data from each of the 1700 genes separately. While this kind of basic analysis is well-suited for parallelization, it can usually be completed on a single CPU in a few minutes using BY processing. Here, though, we consider models for (potentially all) possible pairs of genes, models which are both more complex and whose number in this case exceeds 1.4 million. Completing this analysis in a reasonable amount of time on a single CPU is impossible at today's clock speeds.

We approach this problem with the %Distribute macro, using CPUs from both SAS and NIEHS. (Complete details of the %Distribute macro can be found in the papers link from the Scalability and Performance Community www.sas.com/rnd/scalability). The particular statistical model is a mixed linear model (Wolfinger et al, 2001), which we fit to each pair in succession using BY group calls to Proc Mixed. We use CONTRAST statements to test various hypotheses of scientific interest and ODS statements control the large amount of output. The central computing code is as follows:

```
ods exclude all;
ods noresults;

proc mixed data=gp;
   by gene1 gene2;
   class cloneindex trt days animal hyb1
dye;
   model log2en = cloneindex*trt*days /
fullx noint;
   random int hyb1 / subject=animal
group=cloneindex;
   repeated cloneindex / type=un
subject=dye(hyb1 animal);
   contrast 'Both Genes: CGW minus P day 1'
cloneindex*trt*days
      1 0 0 1 0 -3 0 1 0 0 0
      0 0 0 0 0 0 0 0 0 0 0,
cloneindex*trt*days
      0 0 0 0 0 0 0 0 0 0 0
      1 0 0 1 0 -3 0 1 0 0 0;
```

```
   contrast 'Both Genes: CGW minus P day 14'
cloneindex*trt*days
      0 1 0 0 1 0 -3 0 1 0 0
      0 0 0 0 0 0 0 0 0 0 0,
cloneindex*trt*days
      0 0 0 0 0 0 0 0 0 0 0
      0 1 0 0 1 0 -3 0 1 0 0;
   contrast 'Both Genes: CGW minus P day
avg' cloneindex*trt*days
      1 1 0 1 1 -3 -3 1 1 0 0
      0 0 0 0 0 0 0 0 0 0 0,
cloneindex*trt*days
      0 0 0 0 0 0 0 0 0 0 0
      1 1 0 1 1 -3 -3 1 1 0 0;
   lsmeans cloneindex*trt*days;
   ods output fitstatistics=ft covparms=cp
estimates=es
      contrasts=co lsmeans=ls;
run;

ods exclude none;
ods results;
```

Results

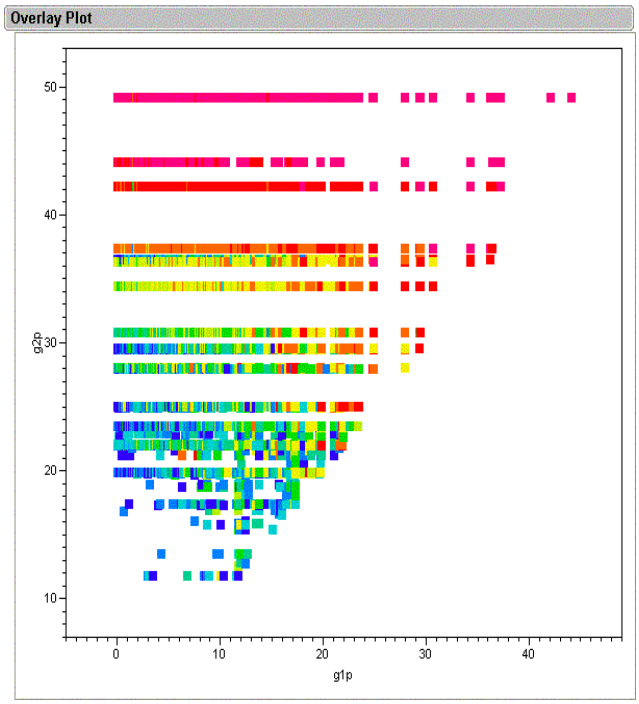

Figure 1.

A key scientific question of interest is whether or not there are pairs of genes that discriminate between the two kinds of compounds in a way that is synergistic; that is, the statistical significance of the test using both genes is much higher than either individual test. The graph in Figure 1 provides some insight into this question. The x- and y- coordinates are the negative log10 p-values from the individual tests (paired so that y >= x), and the color indicates the significance of the joint pair-wise tests (ranging from 80-pink to 40 blue). The red and yellow spots near the bottom indicate potential synergies.

We continue to analyze additional models for this study and this paper will be updated with additional results as they are obtained.

MP CONNECT AND THREADED I/O

The next scenario involves running several SAS steps against a single very large data source in order to produce several different reports. The first step in implementing this scenario is to identify the independent pieces of the whole SAS job that are needed for each specific report. MP CONNECT can then be used to spawn multiple SAS processes, each of which executes an individual report. However, in this scenario, the implementation of parallel sessions may not be enough. Once we parallelize the compute portion of our application we may very likely bottleneck with I/O in two different areas:

- single input data source

Because a single input data source is being read to produce the multiple reports, all of the parallel SAS sessions will be trying to read their input from a single disk and single I/O channel. One way to reduce this bottleneck would be to create copies of the original data source, or subsets of it, on different disk drives. Another solution would be to enable multi-user access to the single large data source by using the new ***Scalable Performance Data Engine (SPDE engine)*** available in SAS 9. The purpose of this engine is to speed the processing of large data sets by accessing data that has been partitioned into multiple physical files called partitions. The SPDE engine initiates multiple threads with each thread having a direct path to a partition of the data set. Each partition can then be accessed in parallel (by a separate processor) which allows the application to analyze data in parallel, as fast as the data is read from disk. In addition, if the partitions are spread across multiple mount points then the multiple SAS processes can each access the partition(s) that they need without conflicting with each other. This can effectively reduce any I/O bottlenecks and substantially decrease the elapsed time to process data.

- I/O activity in WORK library of each SAS session.

The I/O activity in the WORK library for a typical SAS process can be very high. When you use MP CONNECT to spawn multiple SAS sessions on the same SMP machine each session has its own WORK library. Further, each WORK library for each SAS process is created in the same temporary file directory by default. So you end up with multiple SAS processes performing a lot of I/O to their respective WORK libraries but all of these WORK libraries exist on the same physical disk. This is another potential I/O bottleneck. This bottleneck can be minimized in one of two ways:

- Use the WORK invocation option on each of the MP CONNECT processes to direct each process to create its work directory on a separate disk.
- Use the SPDE engine to create a temporary library to be used instead of work and point the USER= option to this temporary library. With SPDE the datasets that are created can be partitioned over multiple file systems. Utility data sets that are created by SAS procedures continue to be stored in the WORK library. However, any data sets that have one-level names and that are created by your SAS programs will be stored in the USER library.

Note: When using MP CONNECT on multiple remote machines the WORK library of the remote sessions exists on the individual machines so this bottleneck does not occur.

Description of the Scenario

The input dataset contains 20,000,000 observations and is 4.8Gig in size. The application creates four outputs: two subsets of the data with data steps using WHERE clauses and two data sets created with frequency reports of certain elements of the data. We'll look at 6 different ways to accomplish this using combinations of the standard base engine, the new SPDE engine, and MP CONNECT to run tasks in parallel.

1) In the first method (BASE/SEQ in Figure 2), the standard base engine is used and the two data steps and two proc steps are run in sequence.

2) The second method (BASE/MP in Figure 2) still uses the base engine to access the data. But since the two data steps and the two proc steps are independent, MP CONNECT is used to run the four of them in parallel.
This is 58% faster than method 1.

With four processes accessing the same data library contained on a single disk, the possibility exists for bottle necks due to I/O limitations. One way to minimize this bottleneck is to use the new SPDE engine. With SPDE, the dataset can be partitioned over multiple file systems, allowing parallel access to the data.

3) The third method (SPDE/SEQ in Figure 2) takes advantage of the SPDE engine, but still runs the steps sequentially.
This change alone is a 15% improvement over method 1.

4) The fourth method (SPDE/MP in Figure 2) combines the parallel capabilities of both the SPDE engine and MP CONNECT.
This is a 51% improvement over the previous method and a 59% improvement over the original method 1. However, it is not much of an improvement over method 2 - the MP CONNECT scenario with the base engine.

Because we feel like there are further performance improvements that can be made, we continue to look for bottlenecks. Another possibility for an I/O bottleneck is that all four processes are writing their datasets to work directories on the same file system. The new SPDE engine can be used to create a temporary library to be used instead of work. With SPDE, the datasets created can be partitioned over multiple file systems.

5) The fifth method (SPDE_OUT/SEQ in Figure 2) uses SPDE for access to both the original dataset and a temporary storage location for output. It is run sequentially.
This result is 25% faster than the original method 1.

6) The sixth method (SPDE_OUT/MP in Figure 2) combines the use of the SPDE engine from method 5 with the parallel capabilities of MP CONNECT.
This results in a 54% improvement over method 5, a 16% improvement over method 4,
which used MP CONNECT and SPDE only for accessing the original dataset, and a 65% improvement over the original method 1. Appendix 1 contains the SAS source code used to implement this scenario.

Results

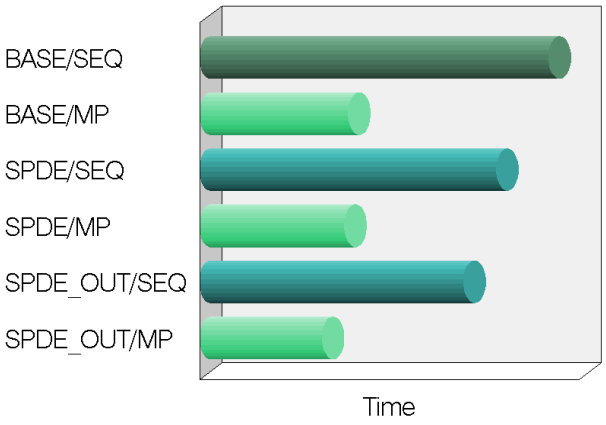

Figure 2

Figure 2 above summarizes the results of the 6 different implementations. The top bar on the graph "BASE/SEQ" represents the use of the standard base engine with all steps of the application running sequentially. The bottom bar on the graph "SPDE_OUT/MP" represents the combined use of MP CONNECT with the SPDE engine being used for both the input data set as well as the output of temporary SAS data sets. This implementation gives the best gains in performance with a 65% improvement in execution time over the original method.

MP CONNECT AND THREADED PROCEDURES

From both in-house as well as customer experience, several SAS procedures have been identified as performance critical, especially for long-running applications. In SAS 9 some of these procedures have been modified to take advantage of SMP hardware by supporting multi-threading. With the appropriate hardware configuration and these enhanced procedures, bottlenecks in your long-running applications can be minimized. The procedures that have been modified to date include:

- in BASE SAS
  - PROC SORT
  - PROC SQL (group by, order by)
  - PROC SUMMARY/MEANS
- in SAS/STAT
  - PROC REG
  - PROC GLM
  - PROC LOESS
  - PROC ROBUSTREG
- in Enterprise Miner
  - PROC DMREG
  - PROC DMDB
  - PROC DMINE.

For these selected SAS procedures, internal modifications have been made to take advantage of threads in order to parallelize some or all of the computationally intensive portions of the procedures to reduce the real time of execution. It is important to note that performance gains on SMP machines with these procedures are not limited to SPDE data sources. Assuming your hardware provides sufficient I/O bandwidth, the threading changes incorporated into these procedures will scale against standard base data sets. Our testing of these procedures using existing base data sets has shown excellent scalability, especially on machines that provide hardware striping.

MP CONNECT can be used to create multiple SAS sessions in which one or more of the SAS sessions runs a threaded procedure. This is a good way to take full advantage of all of the CPU's on an SMP machine with both parallel processes and parallel threads all running at the same time. It is important to note that it is often necessary to use the appropriate options to tune the amount of threading occurring in the individual SAS sessions so that an individual threaded session does not consume all of the CPU's on the machine.

Description of the Scenario

In the final scenario we will process two raw data files:

- Goals.txt - containing sales goals for the year.
- Sales.txt - containing actual sales figures for the year.

Two data steps will be required to read each of the raw files and calculate the projected and the actual revenue and profit for the year. PROC SUMMARY is then run to summarize the data by region and classify the data by employee number. It then computes the total profit that each employee has made for the year and saves the three top performing employees. A final step merges the projected and actual sales by region to determine those employees that reached or exceeded their sales goal for the year.

Results

It is easy to see that there are multiple steps in this application that can be done in parallel:

- the two data steps that read the two raw files and do various calculations for each file, and
- the two PROC SUMMARY procedures that summarize the output of the two prior data steps.

So it makes sense to implement this scenario using MP CONNECT to spawn parallel sessions. The two sessions that execute the PROC SUMMARY will take advantage of the parallel threading of PROC SUMMARY to maximize cpu usage and minimize total elapsed time.

There is still one additional feature that fits perfectly into this scenario and that is the piping capability of MP CONNECT. Since there is no need to persist the temporary data sets created by the initial data steps and then the subsequent PROC SUMMARY procedures, we can use piping to pipe this data from one step to the next. This not only allows us to overlap the execution of all of the steps (the initial data steps, the PROC SUMMARY procedures, as well as the data step merge) but it also eliminates the need to write the temporary data sets to disk. This minimizes our disk space requirements.

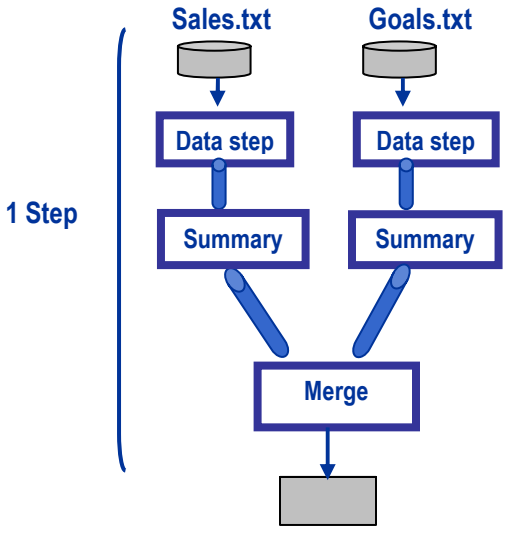

Figure 3.

Figure 3 illustrates the execution flow for this implementation. By making use of MP CONNECT and piping we are able to run ALL of these steps in parallel. The execution of the data steps, the PROC SUMMARY procedures and the final data step to do the merge are all overlapped because piping allows for the output of one phase to be piped as input to the next phase as it is created, instead of waiting until all of the output has been created.

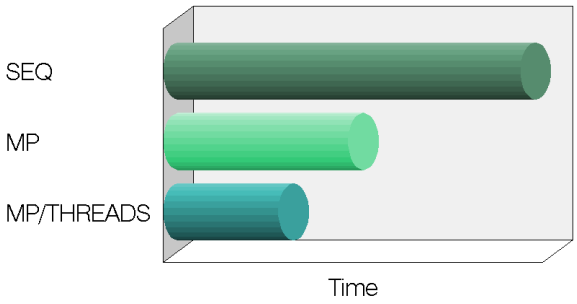

Figure 4.

Figure 4 summarizes the scalability and performance gains achieved by combining the parallel processing and piping capabilities of MP CONNECT with the parallel threading of PROC SUMMARY. The bottom bar labeled “MP/THREADS” represents a 70% improvement in the total elapsed time compared to the original single-threaded serial SAS job!

SUMMARY

SAS 9 is software that can definitely leverage the hardware resources in your environment to deliver a scalable solution. Through the use of parallel SAS sessions and/or parallel threads you can now achieve scalable performance with your SAS applications. As pointed out in the Introduction, not all applications are good candidates for increasing their performance through scalability. But for the applications that do take a substantial amount of time to run and do have pieces that can be overlapped or run in parallel SAS 9 gives you many ways to increase performance through scalability. These ways include:

- threaded procedures
- threaded I/O with new SPDE engine
- threaded servers such as the OMR server, OLAP server, etc.

- parallel sessions with MP CONNECT
- piping with MP CONNECT

In addition, many of our vertical SAS solutions are built on top of these threaded servers and procedures and have incorporated MP CONNECT so that our solutions scale as well.

Future updates to this paper as well as more details about the products and features discussed in this paper can be found at our new Scalability and Performance community on our external website:

www.sas.com/rnd/scalability

APPENDIX 1

The following is the SAS code for the final implementation in the "MP CONNECT AND THREADED I/O" section combining MP CONNECT and the SPDE engine (method #6). Note: The SPDE libname statements are given in detail for the first "rsubmit task1" block but collapsed to save space in the subsequent three "rsubmit" blocks. They are exact duplicates of the libname statements contained in the first "rsubmit" block. The RSUBMIT blocks have been highlighted to help identify the parallel SAS sessions.

```
signon task1 sascmd="!sascmd";
signon task2 sascmd="!sascmd";
signon task3 sascmd="!sascmd";
signon task4 sascmd="!sascmd";

rsubmit task1 wait=no;
  libname user spde '/DATA09/jokenl/testspde'
       datapath=('/DATA01/jokenl/testspde'
                 '/DATA02/jokenl/testspde'
                 '/DATA03/jokenl/testspde'
                 '/DATA04/jokenl/testspde'
                 '/DATA05/jokenl/testspde'
                 '/DATA06/jokenl/testspde'
                 '/DATA07/jokenl/testspde'
                 '/DATA08/jokenl/testspde')
       indexpath=('/IDX1/jokenl/testspde')
       partsize=512 temp=yes;
  libname testspde spde '/DATA09/jokenl/testspde'
       datapath=('/DATA01/jokenl/testspde'
                 '/DATA02/jokenl/testspde'
                 '/DATA03/jokenl/testspde'
                 '/DATA04/jokenl/testspde'
                 '/DATA05/jokenl/testspde'
                 '/DATA06/jokenl/testspde'
                 '/DATA07/jokenl/testspde'
                 '/DATA08/jokenl/testspde')
       indexpath=('/IDX1/jokenl/testspde')
       partsize=512;
  data temp;
    set testspde.big;
    where (un50>25 and un500<300);
  run;
endrsubmit;

rsubmit task2 wait=no;
  libname user spde … temp=yes;
  libname testspde spde …;
  data temp2;
    set testspde.big;
    where (un1000<600 or (un10=10 and
             un100=100));
  run;
endrsubmit;

rsubmit task3 wait=no;
  libname user spde … temp=yes;
  libname testspde spde …;
  proc freq data=testspde.big noprint;
    weight un10;
    tables un100 un50 un100*un50/out=freqcnt
           outexpect sparse;
  run;
endrsubmit;

rsubmit task4 wait=no;
  libname user … temp=yes;
  libname testspde spde …;
  proc freq data=testspde.big noprint;
    weight un1000;
    tables un500 un10 un500*un10/out=freqcnt2
           outexpect sparse;
  run;
endrsubmit;

waitfor _ALL_ task1 task2 task3 task4;

signoff task1;
signoff task2;
signoff task3;
signoff task4;
```

APPENDIX 2

The following is the SAS code for the final implementation in the 'MP CONNECT AND THREADED PROCEDURES" section combining MP CONNECT and threaded PROC SUMMARY. The RSUBMIT blocks have been highlighted to help identify the parallel SAS sessions.

```
options autosignon=yes;

rsubmit task1 sascmd="sas  -WORK
                  /path0/WORK" wait=no;
libname out1 sasesock ":pipe1" TIMEOUT=500000;

data out1.sales ;
  infile '/WORK/salesSumSmall.txt' dlm=':';

  input region sStore EmpID  sItemCode
  sNumOfUnitsSold sUnitPrice sUnitCost;
  sTotalRevenue = sNumOfUnitsSold * sUnitCost;
  sTotalMargin  = (sNumOfUnitsSold * sUnitPrice) -
  sTotalRevenue;
  output out1.sales;
run;
endrsubmit;

rsubmit task2 sascmd="sas -WORK
                  /path1/WORK" wait=no;
libname out2 sasesock ":pipe2" TIMEOUT=500000;

data out2.goals ;
  infile '/d2/WORK/goalsSumSmall.txt' dlm=':';

  input region gStore EmpID  gItemCode
  gNumOfUnitsSold gUnitPrice gUnitCost;
  gTotalRevenue = gNumOfUnitsSold * gUnitCost;
  gTotalMargin  = (gNumOfUnitsSold * gUnitPrice)
   -  gTotalRevenue;
  output out2.goals;
run;
endrsubmit;

rsubmit sumTsk1 sascmd="sas –WORK
                     /d2/WORK" wait=no;
libname in1  sasesock ":pipe1" TIMEOUT=500000;
libname out3 sasesock ":pipe11"
                                 TIMEOUT=500000;
proc summary data=in1.sales noprint nway
       sumtrace=2 THREADS;
       by region;
      class EmpID;
      var sNumOfUnitsSold sUnitCost sUnitPrice
      sTotalMargin sTotalRevenue;
      output out=out3.sumSales
      sum(sTotalRevenue)=sTotalRevenueSum
         idgroup(max(sTotalMargin)
         out[3](sItemCode
        sTotalMargin)=sTotalMargin sTotalRevenue);
run;
endrsubmit;

rsubmit sumTsk2 sascmd="sas -WORK /WORK"
                     wait=no;
libname in2  sasesock ":pipe2" TIMEOUT=500000;
libname out4 sasesock ":pipe12"
                                 TIMEOUT=500000;
proc summary data=in2.goals noprint nway
  sumtrace=2 THREADS;
  by region;
  class EmpID;
  var gNumOfUnitsSold gUnitCost gUnitPrice
  gTotalMargin gTotalRevenue;
  output out=out4.sumGoals
  sum(gTotalRevenue)=gTotalRevenueSum
       idgroup(max(gTotalMargin)
       out[3](gItemCode
       gTotalMargin)=gTotalMargin gTotalRevenue);
run;
endrsubmit;

rsubmit mrgTsk sascmd="sas -WORK
                  /path0/WORK" wait=no;
libname in3a sasesock ":pipe11" TIMEOUT=500000
;
libname in3b sasesock ":pipe12" TIMEOUT=500000
;
libname out5 "/WORK";

data out5.mergeData;
  merge in3a.sumSales in3b.sumGoals;
  by region;
  if sTotalRevenueSum >= gTotalRevenueSum then
  output;
run;
endrsubmit;

waitfor _ALL_ sumTsk1 sumTsk2 mrgTsk;
signoff task1;
signoff task2;
signoff sumTsk1;
signoff sumTsk2;
signoff mrgTsk;
```

REFERENCES

Hamadeh, H.K., Bushel, P., Jayadev, J., Martin, K., DiSorbo, O., Sieber, S., Bennett, L., Tennant, R., Stoll, R., Barrett, J.C., Blanchard, K., Paules, RS, Afshari, C.A. (2002a) "Gene Expression Analysis Reveals Chemical-Specific Profiles". *Toxicological Sciences* 67: 219-231.

Hamadeh, H.K., Bushel, P.R., Jayadev, S., Martin, K., DiSorbo, O., Sieber, S., Bennett, L., Tennant, R., Stoll, R., Barrett, J.C., Blanchard, K., Paules, R.S., Afshari, C.A. (2002b) "Gene Expression Analysis Reveals Chemical-Specific Profiles". *Toxicological Sciences* 67: 219-231.

Olson, D. and Ray, R. (2001), "Version 9: Scaling the Future," *Proceedings of the Twenty-sixth Annual SAS Users Group International Conference.*

Wolfinger, R.D., Gibson, G., Wolfinger, E.D., **Bennett, L., Hamadeh, H., Bushel, P., Afshari, C., Paules, R.S.** (2001) "Assessing gene significance from cDNA microarray expression data via mixed models". *Journal of Computational Biology* 8(6): 625-637.

ACKNOWLEDGEMENTS

Many people have contributed to the products and features described in this paper. They include but are not limited to:

MP CONNECT
Cheryl.Doninger@sas.com
Renee.Palmer@sas.com

SPDE Engine
Billy.Clifford@sas.com

PROC SORT
Robert.Ray@sas.com

In addition I would like to thank Russ Wolfinger, John Kenline and Renee Palmer for their help in running the applications detailed in this paper.

AUTHOR CONTACT INFORMATION

Cheryl Doninger
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
(919) 531-7941
Cheryl.Doninger@sas.com