Paper 124-29

Parallel Processing Hands-On Workshop

M. Michelle Buchecker, SAS Institute Inc., Chicago, IL

ABSTRACT

This hands-on workshop teaches you how to implement your SAS® applications to execute code in parallel and significantly reduce overall execution time. This underutilized SAS, Version 8, technology enables your SAS jobs to take advantage of multiple processors on a machine to perform parallel processing. This capability, called MP CONNECT, enables you to perform disjoint units of work in parallel and to coordinate all results into your original SAS session for the purpose of reducing the total elapsed time necessary to execute a particular application.

INTRODUCTION

Parallel processing (also known as multiprocessing or asynchronous processing) is the dividing of an application into smaller units of work that can be executed simultaneously. Parallel processing can occur on the same machine or on different machines.

Parallel processing can be used to

- execute independent tasks in parallel
- overlap execution of dependent tasks
- take advantage of multiple processors on a *symmetric multiprocessing* (*SMP*) single machine
- take advantage of each processor on a network of machines
- complete a job in less total elapsed time than it would take to execute the same job serially
- increase usage of CPUs.

Asynchronous execution is available in SAS, Version 8, and later and requires SAS/CONNECT® software.

CONDENSED VERSION AND WORKSHOP INSTRUCTIONS

To keep you from having to flip multiple pages in this paper, the major topic areas are condensed and listed below with the workshop instructions. Expanded discussions of the following topics are also included in the remaining sections of this paper (the letters next to each topic map to the corresponding expanded discussion).

Open the existing program STARTER.SAS. This program reads from the raw data files labeled “Equipment” and “Outdoors”, sorts each data set, and then interleaves them. Your goal is to run as many tasks as you can concurrently.

A 

To tell SAS that you want the option to start multiple SAS sessions, you need to code the following statement at the top of the program.

```
options sascmd='c:\progra~1\sasins~1\sas\v8\sas.exe -nosyntaxcheck'
autosignon;
```

B 

To execute code asynchronously:

1. Use the RSUBMIT statement to specify that the task is to run in a separate SAS session.
2. Specify the name of the task (the name must have no more than 8 characters).
3. Use the WAIT=NO option to specify that the task is to process asynchronously.
4. Use the ENDRSUBMIT statement to end the task.

```
①↓      ②↓    ③↓
rsubmit equiptsk wait=no;
data equipment;
   infile 'c:\workshop\winsas\mpdp\equip.csv' dsd;
   input order_id order_item_num
         product_id quantity
         total_retail_price : comma7.
         costprice_per_unit : comma7.
         discount :percent7. product_name :$45.
         supplier_id product_level
         product_ref_id;
run;
proc sort data=equipment;
   by costprice_per_unit;
run;

endrsubmit; ←  ④
```

```
①↓     ②↓    ③↓
rsubmit outtsk wait=no;
data outdoors;
   infile 'c:\workshop\winsas\mpdp\outdoors.csv' dsd;
   input order_id order_item_num
         product_id quantity
         total_retail_price : comma7.
         costprice_per_unit : comma7.
         discount :percent7. product_name :$45.
         supplier_id product_level
         product_ref_id;
run;
proc sort data=outdoors;
   by costprice_per_unit;
run;

endrsubmit; ←  ④
```

! Notice that the order of the code changed. The PROC SORTs have been moved, so they are in the same RSUBMIT block as the corresponding DATA step.

! Notice that the raw data file name now must be fully qualified because the current working directory cannot be assumed when starting a new SAS session.

C 

The next piece of the original code is the DATA step that interleaves the data sets named OUTDOORS and EQUIPMENT. The EQUIPTSK and OUTTSK tasks must complete before this code executes. SAS must finish processing both tasks before starting this DATA step. To ensure this, add the following WAITFOR statement just above that DATA step.

```
waitfor _all_ equiptsk outtsk;
```

(NOTE: If you were unable to complete the previous workshop steps, you can open the PGM1.SAS program and then perform this step.)

D 

Because each task starts a separate SAS session, each task has its own WORK library. The data sets being created need to be processed in the "local" or parent SAS session. There are many ways to make the "remote" WORK libraries known to the parent SAS session, and they are discussed later in this paper. The most efficient technique in SAS, Version 8, is to

- determine the physical path of the WORK library
- put that information into a macro variable
- transfer the macro variable to the parent SAS session
- assign a libref in the parent session to point to the physical location obtained by the transferred macro variable. (I said it was efficient, I didn't say it was easy!)

To make all this happen:

1. Add the **sysrputsync=yes** option to each RSUBMIT statement.
2. Add the following statement before each ENDRSUBMIT statement:

```
%sysrput pathequip=%sysfunc(pathname(work));
```

3. Add the following statements after the WAITFOR statement:

```
libname equipwrk "&pathequip";

libname outwrk "&pathout";
```

4. Change the existing SET statement in the DATA step to the following:

```
data combine;
   set outwrk.outdoors equipwrk.equipment;
   by CostPrice_per_unit;
run;
```

(NOTE: If you were unable to complete the previous workshop steps, you can open the PGM2.SAS program and then perform this step.)

E 

By default, the SAS sessions that were established with the RSUBMIT statement are still running. To terminate a remote SAS session, you can issue the SIGNOFF statement. Add the following SIGNOFF statements after the DATA step:

```
signoff outtsk;

signoff equiptsk;
```

(NOTE: If you were unable to complete the previous workshop steps, you can open the PGM3.SAS program and then perform this step.)

F 

The LISTTASK statement displays information such as the task name and its current state of execution. Add the following LISTTASK statement before the WAITFOR statement:

```
LISTTASK _ALL_;
```

(NOTE: If you were unable to complete the previous workshop steps, you can open the PGM4.SAS program and then perform this step.)

G 

The SAS/CONNECT Monitor window displays information about the tasks in two columns: Name and Status. To open the SAS/CONNECT Monitor window, activate the SAS Explorer window, and select **View** ⇨ **SAS/CONNECT Monitor**. If you are running SAS, Version 8, **View** ⇨ **Details** must be turned on in order to see the status value. Submit the program and view the progress in the SAS/CONNECT Monitor window. You might want to comment out the SIGNOFF statements before submitting.

H 

The NOTIFY= option displays a message window that indicates the completion of a task. Add the **NOTIFY=** option to the first RSUBMIT statement.

That concludes this workshop. If you want to save your code, this would be a good time to do it. If you prefer, you can copy the completed program that is named COMPLETE.SAS. Please make sure that you have submitted the following statements:

```
signoff outtsk;

signoff equiptsk;
```

If you commented them out to terminate any existing SAS sessions, then close your main SAS session.

If you want to learn more about parallel processing, including how to perform it on multiple machines, as well as piping parallelism and threading, SAS offers a 2-day course titled “Distributed and Parallel Processing with the SAS System”! Course dates and locations are May 19-20 in Chicago, IL and June 3-4 in Cary, NC.

A SYMMETRIC MULTIPROCESSING ENVIRONMENT

A symmetric multiprocessing environment

- is a hardware and software architecture that can improve the speed of I/O and processing
- has multiple CPUs that share the same memory and has a thread-enabled operating system
- can spawn and process multiple threads, simultaneously, using multiple CPUs
- coordinates threads from the same process to share data very efficiently.

MP CONNECT enables SAS tasks or sessions to execute in parallel and coordinates and synchronizes all results into a single parent (or client) session. This simultaneous processing can occur on

- a single SMP (symmetric multiprocessor) machine
- multiple remote/networked machines with each machine having a single processor
- a combination of the above.

MP CONNECT is a feature of SAS/CONNECT software. MP CONNECT enables grid computing. Research and development at SAS is on-going to include features such as

- load balancing
- job scheduling and restart
- user interface for load balancing/job scheduling/resource allocation.

CONSIDERATIONS

When working with multiprocessing applications, consider

- overhead needed to start the separate SAS sessions
- load factor – an already busy machine might not have much extra processing time for additional requests
- additional requirements on the I/O subsystem
- interdependencies of data between steps
- dividing the data into smaller data sets that can be worked on simultaneously and then brought back together.

SCALING UP

Scaling Up enables SAS to fully use *symmetric multiprocessing* (*SMP*) hardware by running multiple concurrent SAS sessions on **one** machine.

BUSINESS SCENARIO

The company Orion Star (a sporting and retail store) needs to

1. read the outdoors and equipment data from separate raw data files
2. sort each data set by **costprice_per_unit**
3. interleave the two data sets by **costprice_per_unit**.

The data set that is named OUTDOORS can be read and sorted at the same time that the data set that is named EQUIPMENT is read and sorted.

Now that the pieces of concurrently executable code are identified, you must determine how to execute the pieces in parallel to obtain faster results.

When developing an application to run asynchronously, first run your code synchronously to make sure that your SAS code is error free (see the next section).

```
data equipment;
   infile 'equip.csv' dsd;
   input order_id order_item_num
         product_id quantity
         total_retail_price : comma7.
         costprice_per_unit : comma7.
         discount :percent7. product_name :$45.
         supplier_id product_level
         product_ref_id;
run;
data outdoors;
   infile 'outdoors.csv' dsd;
   input order_id order_item_num
         product_id quantity
         total_retail_price : comma7.
         costprice_per_unit : comma7.
         discount :percent7. product_name :$45.
         supplier_id product_level
         product_ref_id;
run;

proc sort data=equipment;
   by costprice_per_unit;
run;
proc sort data=outdoors;
   by costprice_per_unit;
run;
data combine;
   set outdoors equipment;
   by CostPrice_per_unit;
run;
```

In scaling up, the same machine acts as both a client (local) and a server (remote), and sends requests to itself. For each task that executes simultaneously, a new SAS session is invoked, which involves overhead. MP CONNECT is used for time-consuming programs where the overhead to start a new SAS session is a minimal amount of the overall process time.

In order to perform asynchronous processing on the same machine, you must specify the options

- SASCMD
- AUTOSIGNON (or use a SIGNON statement).

The SASCMD option specifies the command that invokes another SAS session on the same machine.

```
options sascmd='sas -nosyntaxcheck';
```

A common option is -NOSYNTAXCHECK. Otherwise, any syntax errors that are encountered set OBS=0, which prevents additional code from executing properly.

A

If multiple versions of SAS exist on the same machine, fully qualify the executable name with the path name.

For example, to ensure that the Version 8 executable is run, submit the following:

```
options sascmd='c:\progra~1\sasins~1\sas\v8\sas.exe
                -nosyntaxcheck';
```

In SAS System 9, you can use the same executable to invoke a new SAS session as the one that was used to invoke the current SAS session. To do this, substitute the *SAS-invocation-command* with the value !SASCMD.

```
options sascmd='!sascmd -nosyntaxcheck';
```

If you scale up on an OS/390 machine, the syntax for the SASCMD option uses a colon before options .

```
SASCMD=':SAS-system-options' |
    '!SASCMD :SAS-system-options'
```

For example, to set the page number to start at 12, use the following statement:

```
options sascmd=':pageno=12';
```

The pagesize and linesize options cannot be changed using SASCMD.

The AUTOSIGNON option enables the SAS session to automatically invoke a new SAS session whenever a request is made. The default is NOAUTOSIGNON.

```
options autosignon sascmd='!sascmd -nosyntaxcheck';
```

B

To execute code asynchronously:

1. Use the RSUBMIT statement to specify that the task is to run in a separate SAS session
2. Specify the name of the task (the name must be 8 characters or less)
3. Use the WAIT=NO option to specify that the task is to process asynchronously
4. Use the ENDRSUBMIT statement to end the task.

```
①↓      ②↓   ③↓
rsubmit equiptsk wait=no;
data equipment;
   infile 'c:\workshop\winsas\mpdp\equip.csv' dsd;
   input order_id order_item_num
         product_id quantity
         total_retail_price : comma7.
         costprice_per_unit : comma7.
         discount :percent7. product_name :$45.
         supplier_id product_level
         product_ref_id;
run;

proc sort data=equipment;
   by costprice_per_unit;
run;

endrsubmit; ← ④
```

```
①↓    ②↓   ③↓
rsubmit outtsk wait=no;
data outdoors;
   infile 'c:\workshop\winsas\mpdp\outdoors.csv' dsd;
   input order_id order_item_num
         product_id quantity
         total_retail_price : comma7.
         costprice_per_unit : comma7.
         discount :percent7. product_name :$45.
         supplier_id product_level
         product_ref_id;
run;
```

```
proc sort data=outdoors;
   by costprice_per_unit;
run;

endrsubmit; ← ④
```

Actual asynchronous (parallel) processing cannot occur on any machine that has only one CPU. These demonstrations and exercises only mimic asynchronous processing.

C The next piece of the original code is the DATA step that interleaves the data sets named OUTDOORS and EQUIPMENT. SAS must complete processing the EQUIPTSK and OUTTSK tasks before starting this DATA step. To ensure this, use the WAITFOR statement.

WAITFOR _ALL_ | _ANY_ *task1 task2 … taskn*;

The WAITFOR statement tells the local SAS session not to execute any additional code until all of the listed tasks have completed (if you use the _ALL_ option) or until at least one of the tasks has completed (if you use the _ANY_ option).

D **WORK WORK WORK**

Because each task starts a separate SAS session, each task has its own **WORK** library.
The data sets that are being created need to be processed in the "local" or parent SAS session. To make the data sets in the WORK library in the "remote" SAS session available to the "local" or parent SAS session you can:

- Use PROC DOWNLOAD to transfer the data from the remote SAS sessions to the local SAS session.
  - Drawbacks:
    - makes a copy of the data on the same machine
    - increases I/O
    - increases CPU time.
- Use a permanent library that all SAS sessions on that machine can access in place of the WORK library.
  - Drawbacks:
    - you are responsible for cleaning up after yourself.
- Use the INHERITLIB option to enable remote sessions to write directly to the parent WORK library.
  - Drawbacks:
    - requires SAS 9
    - code must be modified to use two unique librefs
    - might create I/O bottleneck if used in many remote tasks.
- Use Remote Library Services to transparently point to the remote WORK libraries from the local SAS session.
  - Drawbacks:
    - adds a layer of communications on the machine, which decreases efficiency.
- Create a macro variable that contains the name of the physical path of the WORK library and transfer that to the local SAS session.
  - Drawbacks:
    - adds a layer of complexity.

We are going to use the last option and create a macro variable that contains the name of the physical path because it is the most efficient technique if you are not using SAS 9.

To identify the physical location of the WORK library, use the PATHNAME function. **-- PATHNAME**(*libref*).

To transfer the information of the physical location to your local SAS session, use macro variables. A *macro variable* is a variable that is stored in an area of memory that is called the *symbol table*. Each SAS session (task) has its own symbol table. One way to create a macro variable is with a %LET statement.

Traditionally, what is on the right side of the equal sign in a %LET statement is assigned as text exactly the way it appears. However, using a macro function called %SYSFUNC, you can supply a DATA step function, and the results of that function are assigned to the macro variable as text.
For example:

```
%let path=%sysfunc(pathname(work));
```

The macro variable PATH is still stored on the logical remote SAS session. To transfer the value of the

macro variable on the remote session to your local session, use the %SYSRPUT statement. The %SYSRPUT statement takes the value of the macro variable on the remote session (identified by the macro variable reference *&remotemvar*) and places the value into a macro variable in the local session (identified by the macro variable *localmvar*).

By default, macro variables that are created with the %SYSRPUT statement are not transferred to the local session during asynchronous execution until specific execution points are reached.

To transfer the macro variable that is created with %SYSRPUT, you can specify either

- the SYSRPUTSYNC=YES option in the existing RSUBMIT statement
- the WAIT=YES option in a new RSUBMIT statement.

Beginning in SAS 9, you can use the global option SYSRPUTSYNC in an OPTIONS statement to activate SYSRPUTSYNC for all occurrences of an asynchronous RSUBMIT during that SAS session. The default is NOSYSRPUTSYNC.

```
rsubmit equiptsk wait=no sysrputsync=yes;
data equipment;
   .
   .
   .
run;
proc sort data=equipment;
   by costprice_per_unit;
run;
/* Retrieve the path location of EQUIPTSK WORK library*/
%sysrput pathequip=%sysfunc(pathname(work));
endrsubmit;
rsubmit outtsk wait=no sysrputsync=yes;
data outdoors;
   .
   .
   .
run;
proc sort data=outdoors;
   by costprice_per_unit;
run;
/* Retrieve the path location of OUTTSK WORK library  */
%sysrput pathout=%sysfunc(pathname(work));
endrsubmit;
waitfor _all_ equiptsk outtsk;
```

There are now two macro variables in the local session, **pathequip** and **pathout**, that point to locations on the local machine where the data sets are stored.

To process these data sets in the local SAS session, issue a LIBNAME statement that points to each physical location.

```
/* execute in "local" session */
libname equipwrk "&pathequip";
libname outwrk "&pathout";
data combine;
   set outwrk.outdoors equipwrk.equipment;
   by CostPrice_per_unit;
run;
```

E TERMINATING TASKS

By default, the SAS sessions that were established with the RSUBMIT statement are still running. To terminate a remote SAS session, you can issue the SIGNOFF statement. Each task must have its own SIGNOFF statement.

```
signoff outtsk;
signoff equiptsk;
```

MANAGING TASKS

Frequently your tasks are executing for a long time and you might want to

- display a list of executing asynchronous tasks and their status
- monitor the list of executing asynchronous tasks interactively by using the SAS/CONNECT Monitor window
- be notified when a task has completed.

F **LISTTASK STATEMENT**

The LISTTASK statement lists information about any or all active connections.

LISTTASK <_ALL_ | *taskname*>;

The LISTTASK statement displays information such as the task name and its current state of execution.

A task can be in one of the following states of execution:

- READY
- RUNNING ASYNCHRONOUSLY
- COMPLETE.

G **SAS/CONNECT MONITOR WINDOW**

The SAS/CONNECT Monitor window displays information about the tasks in two columns: Name and Status. To open the SAS/CONNECT Monitor window, activate the SAS Explorer window, and select **View** ⇨ **SAS/CONNECT Monitor**. If you are running SAS, Version 8, **View** ⇨ **Details** must be turned on in order to see the status value.

H **NOTIFY= OPTION**

The NOTIFY= option displays a message window that indicates the completion of a task. The NOTIFY= option can be placed in the RSUBMIT statement if AUTOSIGNON=YES, or in the SIGNON statement. The value of NOTIFY= is either YES or NO. If NOTIFY=YES is in effect, then when the task completes a notification message window is displayed

- when performing an asynchronous RSUBMIT (ie WAIT=NO)
- for all SAS session execution modes (Display Manager, and batch) to which a terminal is attached by setting the TERMINAL option.

CONCLUSION

Performing tasks in parallel on the same machine can dramatically reduce overall elapsed time for your program. SAS/CONNECT provides this capability starting in SAS, Version 8. Managing your tasks can easily be performed by using the SAS/CONNECT Monitor window and the NOTIFY= option.

TRADEMARKS

CONTACT INFORMATION

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

M. Michelle Buchecker
SAS Institute Inc.
180 N. Stetson Ave, Suite 5200
Chicago, IL 60613
Michelle.Buchecker@sas.com