Paper 107-27

Passing Values to a Remote Multi-Process SAS/CONNECT® Session

John E. Bentley, Wachovia Bank, Charlotte, North Carolina

Abstract

SAS® Software Version 8 supports parallel processing on symmetrical multiprocessor (SMP) systems. Part of SAS/CONNECT®, Multi-Process Connect® (MP CONNECT) exploits multiple processors on a host system by a process similar to the RSUBMIT/ENDRSUBMIT approach to remote compute services. Within a single program, it allows the user to start multiple independent SAS sessions on each processor on the system, submit code to each of those remote sessions, and then gather the results back into the originating SAS session. Although conceptually clear, a major sticking point can be how to pass macro values between the independent sessions. This paper introduces parallel processing and MP CONNECT and presents one solution to the problem of passing values between sessions.

Disclaimer: The views and opinions expressed here are those of the author and not those of Wachovia Bank. Wachovia does not endorse, recommend, or promote any of the computing architectures, platforms, software, programming techniques or styles referenced in this paper.

MP CONNECT

In today's disk-based data warehouse and data mart environments, we have easier access to massive amounts of data. It's quite common to work with several million or tens of millions of records. Unless highly specialized software is used, however, processing time is still measured in hours. SAS Version 8 changes that, though.

Version 8 SAS/CONNECT includes the Multi-Process Connect facility (MP CONNECT) to exploit multi-processor capabilities of symmetrical multiprocessor (SMP) and massively parallel processor (MPP) systems "by allowing parallel processing of self-contained tasks and the coordination of all the results in the original SAS session". (SAS/CONNECT User's Guide, On-line version.) MP CONNECT runs in all parallel environments, but this author has experience with MP CONNECT only in UNIX so all examples and references presented here will be UNIX-oriented. Because of SAS's multi-vendor architecture, however, solution should be platform independent.

Without MP CONNECT, SAS Software, including Versions 8 and 8.1, executes *sequentially* one step or procedure at a time on a single processor. (SPDS is an exception.) Even when extracting from parallel relational database engines, the results of the SQL query must be returned to a single processor where the remaining SAS code executes sequentially.

MP CONNECT is one way that SAS users can enter the world of parallel processing. With it, the DATA step, PROC SQL, and some PROCs such as SORT can execute in parallel on SMP UNIX servers like Sun Enterprise Servers, the HP-9000 Servers, Compaq's AlphaServers, and IBM's RS/6000s as well as MVS and CMS mainframe systems. Parallelization results in increased in program efficiency. As programs run faster, more timely information is provided to decision makers and that directly affects profitability. Return on investment in the computer system is increased because the computer that's running parallel SAS programs will be able to handle more work in the same amount of time.

Parallel Processing

Consider a simple SAS program that reads in a flat file to create a SAS data set, transforms the variables, sorts the data set, and then calculates grouped summary statistics. PROC SORT doesn't begin until the DATA step ends, and PROC MEANS won't run until the sort finishes. The steps run *sequentially*, and the total execution time for the job is the sum of the times for each of the separate steps.

MP CONNECT provides *independent parallelism*. This occurs when there are no dependencies between tasks and they can therefore be run concurrently. Suppose our example has two flat files that need to be read, manipulated, sorted, and then merged before calculating group summary statistics.

If we read in and manipulate both data sets at the same time and have the output data sets sorted at the same time, then we've implemented independent parallelism for this part of the processing. The processes of gathering the sorted data sets into a single data set and calculating the statistics are done sequentially. Execution time is calculated by adding the time required for the lengthiest series of parallel processes plus the time needed for sequential processing.

Program structure is critical in parallel programming. In our example the merge step and summary statistics calculations were be done sequentially. But if the business user was comfortable using grouped statistics, the statistics could have been calculated in parallel, the results output to a data set, and then those data sets merged and used as input for final calculations. In another case, we could have calculated the summary statistics in parallel if none of the variables for which statistics are needed were created in the merge step and none were shared between the data sets

Probably the most widely used parallel-capable hardware architecture today is the *symmetrical multiprocessor* (SMP) system, an incredible advance over single CPU systems. In an SMP box, multiple CPUs and associated resources run under the control of a single instance of the operating system. Memory and disk resources are shared. All major computer manufacturers make UNIX-based SMP machines, but SAS users may be most familiar with those from Sun and Hewlett-Packard.

Decision support systems (DSS) are well suited for parallel processing on SMP platforms because of the nature of the data flow. DSS refers to the extraction, analysis, and presentation of data from historic databases to enable knowledge-based decision-making. OLAP, on-line analytical processing, is a subtype Decision Support. System An example is examining weekly sales over the past quarter by product, city, and state. SAS/EIS® is software for building Executive Information Systems, also a type of DSS.

In a DSS application, data is read, manipulated, analyzed, and output. The data may be read from multiple source data sets, relational databases, or flat files. Business rules may require that different manipulations and transformations be applied to different data sets. The analysis, reporting, or presentation requirements may specify detail based on location, time, product or some other categorical or grouping variables.

When any of these situations exist, we can often process in parallel by splitting the program into independent pieces for simultaneous execution, thereby reducing overall execution time. Ideally, and not considering system overhead, the execution time for a parallel program will equal the execution time for the same program run sequentially divided by the number of processors being used. That is, if a program takes an hour to run sequentially then splitting it among four processors will reduce the run time to 15 minutes.

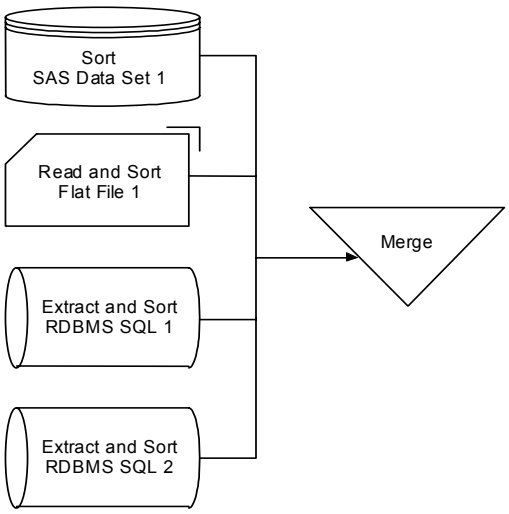
MP CONNECT and Parallel Processing

MP CONNECT is part of the SAS/CONNECT product. But instead of running a SAS session on a remote host, it runs a separate SAS session on a different CPU on the same host as the local SAS session. This allows a SAS program to exploit multi-processor capabilities of symmetrical multiprocessor (SMP) and massively parallel (MPP) systems. It eliminates the need for executing a spawner program or a TELNET session on the remote host, and it also eliminates the need for a script file on the local host and the need to configure the remote access method.

Although MP CONNECT's syntax is simple and straightforward, the programs must be structured with parallel and sequential code sections—encapsulated—to take advantage of parallel architectures. Because the parallel sections must be independent and self-contained, converting complex production jobs this may require a significant redesign and redevelopment effort. The ancient art of flowcharting can be a big help.

MP CONNECT is limited physically only by the number of CPUs available, but logically there are no limits on the number of SAS sessions that can be spawned. Figure 5 illustrates one usage that combines data from multiple data sources: SAS data set, flat file, and relational database. For each, the data must be sorted before merging. On a four-way system, MP CONNECT allows all four data sets to be read (or extracted) and sorted simultaneously. The merge won't execute until all the previous processing completes.

Figure 1. The Fast Way to Merge Four Data Sets



MP CONNECT – Considerations

At SUGI 25 in "Multiprocessing with Version 8 of the SAS System", Cheryl Garner of the SAS Institute listed three questions that need to be answered to determine if a particular program or application can benefit from MP CONNECT. This author adds a fourth question.

1. Can the data source(s) be split so that they can be processed separately?
2. Can the program be split into multiple SAS sessions that can run at the same time and not depend on each other or access non-shared resources, such as output data sets?
3. Can you get a good return on your investment of programmer time needed to write a parallel program vs. the machine time needed to run the program sequentially?
4. What is the impact to other users if one program uses multiple processors simultaneously?

There are no easy answers to these questions because there are many variables that must be considered. For questions 1 and 2, the go/no-go decision is based on the degree of difficulty for doing it, which in turn is based in large part on the expertise of the programmer. The degree of difficulty is also based on whether we are modifying an existing program or writing a new program—it may be more difficult to modify an existing program, determined in part by the presence or absence of program documentation and the use of macro programming. Another factor is whether or not the original programmer is doing the work and how long ago the program was written, that is, how well the programmer remembers why things were done the way they were.

If we assume that the answer is "yes" to the first two questions, then question 3 is the key. It may not be a good return on investment to spend six hours writing (or modifying), testing and debugging a sequential program run monthly if the speed-up is only 20 minutes. If, however, the usage load on the computer is great enough, the resulting scale up of the box may warrant the time and energy.

The Problem of Passing Macro Values

A Remote MP CONNECT session is completely independent from the local session that spawned it. It has no knowledge of any librefs, filerefs, or system options assigned in the local session, uses a different WORK directory, and doesn't share macro variables created in the local session. In effect, it runs in isolation from the local session (and all other remote sessions as well.) This can sometimes be a hindrance.

As an example, let's say that we have a program that extracts records from a relational data warehouse table one field at a time, produces one-way frequencies for character fields and descriptive statistics for numeric fields, and writes the output to a separate data sets for each field. (We're using "field" and "variable" interchangeably.) To speed it up, we want to use two processors—the local session will process the numeric variables and a remote session will process the character variables at the same time.

In a local session, field names and types are pulled from a system table in the database and written to a SAS data set. Each session uses a macro do-loop within Proc SQL to run a separate extract and create a data set for each variable.

In the local session, the following values are assigned to macro variables during the "pre-processing":

- Informix table name
- Date criteria for selection (needed by the SQL Where clause)
- Other selection criteria (needed by the SQL Where clause)
- The number of character variables
- The number of numeric variables
- The target directory for the output data set
- SAS system options

All of these values except for number of numeric variables are needed by the remote session. But because the remote session is completely independent of the local session, it has no knowledge of the existence of these macro variables. So how do we get them there?

One Solution to the Problem

Both sessions have access to the same directory structures, and that's the key to this solution. By converting the macro variables to "regular" SAS variables in the sending session and then writing a permanent data set containing them to a common directory before another session needs them, the values will be available. The receiving session can then easily read the data set and convert the variables back into macro variables. The only caveat is that the SAS librefs and/or data set name holding the values must be hard coded in each sessions.

Code Sample 1. Writing Macro Values to a Permanent SAS Data Set

```
** Count number of record types in the table and assign
to macro vars. Variable names pulled earlier from
systables and separated by type into data sets. (Shows
one method of getting number of obs in a data set.);

data _null_;
  dsid=open("riskData.char");
  call symput ('_numChar',attrn(dsid,"nobs"));

  dsid=open("riskData.num");
  call symput ('_numNum',attrn(dsid,"nobs"));

  dsid=open("riskData.other");
  call symput ('_numOther',attrn(dsid,"nobs"));
run;

** Write the macro variables into a SAS data set so they
are available to remote processes. Use a fully-defined
SAS data set name. The data set has one observation
and 9 variables. Without a LENGTH statement and using
formats, all variables will character, length 200. ;

data " /risk_dm_validation/data/macroVars.sas7bdat";
 riskTable=symget('_riskTable');
 lib=symget('_remDir');
 cnt1=symget('_numChar');
 cnt2=symget('_numNum');
 cnt3=symget('_numOther');
 opt=symget('_options');
 filter=symget('_riskWhere');
 period=symget('_riskPeriod');
 sess=symget('_session');
run;
```

Code Sample 2. Retrieving Macro Values by a Remote Session and Using Them

```
/*****************************************************************
 Run the Character Variable extract and analysis as a
Remote Session using MP Connect
*****************************************************************/
options autosignon=yes sascmd='/usr/local/sas8/sas' ;

rsubmit process=remote1 wait=no;

 ** Read the data set containing the macro variables and
recreate them ;

 data _null_;
   set "/ risk_dm_validation/data/macroVars.sas7bdat";
   call symput('_riskTable',riskTable);
   call symput('_numChar',cnt1);
   call symput('_numNum',cnt2);
   call symput('_numOther',cnt3);
   call symput('_remDir',lib);
   call symput('_options',opt);
   call symput('_riskWhere',filter);
   call symput('_riskPeriod',period);
   call symput('_session',sess);
 run;

** Use a macro to extract and summarize each variable in
the data set. (Shows how the passed values are used.) ;

%macro Freqs;
     options "&_options";
     libname riskData v8 "&_remDir";
     %local n;

     ** Create series of macro variables containing the
    variable names;

    %do n=1 %to %eval(&_numChar-0);
      data _null_;
        set riskData.char (firstobs=&n obs=&n);
        call symput("_field&n",riskColName);
        call symput("_outdsn&n",'riskData.'||
                    trim(lowcase(riskColName))||
                    '_risk_freqs');
      run;
   %end;

    ** Use Proc SQL to produce a data set of counts for
each variable;

   proc sql exec;
     %riskset;
     %do n=1 %to %eval(&_numChar-0);
       execute (
         select
           &&_field&n, count(*) as numRecs
         from
           &_riskTable
         %if %str(&_riskWhere) ne  and
             %str(&_riskPeriod) ne %then
         where %str(&_riskWhere)

<snip>
```

Tips and Clues about MP CONNECT

The difficulty of writing a parallel program or modifying an existing program for parallel execution is partially dependent on the degree of programming "elegance" being sought. MP CONNECT syntax and implementation is simple and straight-forward, but without using macro-programming techniques, a complex parallel program can turn into some of the worst spaghetti code ever seen.

It's temptingly easy to copy and paste existing code segments and then encapsulate them with RSUBMIT-ENDRSUBMIT and hardcode values instead of using macro variables, but then a program of a few hundred lines with numerous DATA steps, LINKs, GOTOs, and PROCs can quickly balloons to a thousand lines of what looks like repetitious code with dozens of hard coded variables and references, depending on the number of remote sessions being used. That creates very serious problems for testing and debugging and program modification and maintenance.

Documentation, especially program comments, takes on added importance. As was mentioned earlier, flowcharting makes parallel programming much easier to plan and implement. A visual representation of the process clearly identifies which sections must be sequential and which can be parallel. Keep in mind that just because a section can be run in parallel doesn't mean that it must be run in parallel.

Using macros and %INCLUDEs are key coding techniques for reducing the number of lines of code that must be written and maintained. Good programming style, resulting in improved readability, is also critical.

Things to remember:

- A program can move into and out of parallel execution any number of times, not just once per CPU.
- Local and remote sessions exist independently of each other. System options, librefs, filerefs, and macro variables must be specified for each session; each session has it's own work directory.
- Each time a remote session is started, it is a completely new session and options, librefs, etc. must be specified.
- Manage repetitive code by using macros. Use the SASAUTOS=<directory> NOMPRINT system options to specify the macro directory and prevent macro code from printing in the log. Alternatively, use %INCLUDE with the NOSOURCE2 option to keep the log readable.
- Macros variables are session-specific, even global macro variables defined in the local session before any remote processing is started.
- Use a permanent SAS data set to pass macro values between the local and a remote session.
- Remote sessions do not persist. The default is an implicit SIGNOFF when ENDRSUBMIT is encountered. Using the SIGNOFF command, though, makes the code more readable.
- Use long file names to establish a naming convention for variables and data set names.
- Test and debug each remote session one at a time in the local environment with a small test data set. After it's working properly, encapsulate it for remote processing.

Summary and Conclusions

SAS's MP CONNECT provides SAS users with the ability to exploit the power of multiple processor systems. By encapsulating programs into parallel and sequential sections, execution time can be dramatically reduced. Speeding up a program reduces the time it takes to put information into the hands of decision makers. The productivity of the computer system itself is improved because it can handle more work in the same amount of time, thereby improving return on investment.

MP CONNECT is simple to code and implement. The concept is the same as using RSUBMIT to take advantage of remote computer services. The only difference is that the remote SAS session is really a different CPU on the same box that the local host is running on.

With SAS, there is almost always more than way to accomplish a data manipulation task or to solve a coding problem. Passing macro values from one SAS session to another session running simultaneously may first appear challenging, but at least two solutions exist. The solution presented here is to use the SYMGET function to write the values before they are needed by the other session to a permanent data set in a common directory. Then any session that needs the values can read the data set and recreate the macro variables by using the CALL SYMPUT command.

Question: What's another solution? Email the author if you can't figure it out.

References and Resources

Bentley, John (2001) "SAS Multi-Process Connect: What, When, Where, How, and Why" in SUGI26 Conference Proceedings. Cary, NC: SAS Institute.

"SAS/CONNECT User's Guide" in SAS Online Doc, Version 8. Cary, NC: SAS Institute.

Stephen Morse and David Isaac (1998). Parallel Systems in the Data Warehouse. Upper Saddle River, NJ: Prentice-Hall, Inc.

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.

About the Author

John E. Bentley has used SAS Software for fifteen years in the healthcare, insurance, and banking industries. For the past three years he has been with the Corporate Data Management Group of Wachovia Bank with responsibilities of supporting users of the bank's data warehouse and data marts and managing the development of SAS client-server applications to extract, manipulate, and present information from them. John regularly presents at national, regional, and special interest SAS User Group Conferences and local SAS User Group meetings. He is co-chairing the SUGI27 Data Warehousing and Enterprise Solutions section and was Program Chair for the 2000 and 2001 Data Mining SAS User Group Conferences.

Contact Information

John E. Bentley
Wachovia Bank
201 S. College Street, NC-1025
Charlotte, NC 28288
704-383-2686
John.Bentley2@FirstUnion.Com