The %Distribute System for Large-Scale Parallel Computation in the SAS® System

Cheryl Doninger, Manager of SAS/CONNECT® Software R & D
Randy Tobias, Manager of Linear Models R & D

Introduction

Parallel computation allows large data processing tasks critical for science, business, and industry to be accomplished in a fraction of the usual time. The new MP CONNECT tools in SAS/CONNECT software enable you to perform parallel computation by coordinating the data processing power of the SAS System running simultaneously on multiple servers.

The %Distribute macro is the workhorse of a system of SAS macros using the new MP CONNECT capabilities to distribute a fairly general class of computational problems across an arbitrarily large number of SAS/CONNECT servers. Features of the system include

- a load balancing scheme to improve the scaling efficiency of the parallel computation
- a simple but informative real-time display of the state of all the servers
- automatic generation and management of independent random number streams on the servers

This paper describes the underlying approach used in %Distribute and demonstrates its application for Monte Carlo simulation. The approach has also been used to address open problems in statistical analysis of variance (Westfall et al. 2001, O'Brien and Tobias 2001). The accuracy and completeness of these simulation approaches would have been infeasible without the distributed parallel processing capabilities in MP CONNECT. For the work of Westfall et al. in particular, MP CONNECT made it possible to condense three years' worth of computing time into just a month's worth of desktop time.

Distributing SAS with MP CONNECT

To begin, let's step through what's required for distributing a large SAS job with MP CONNECT. In outline, all you need to do is

- Connect to the hosts.
- Give them tasks to do in parallel.
- Wait for them to complete.

To be more specific, suppose you have a macro

```
%Simulate(Output,1000,seed=64382);
```

that simulates a complicated random process 1000 times and puts the resulting sample in the data set `Output`. In theory, you could do this in half the time if you did it in parallel on two machines instead of one. What are the bare-bones steps you need to take to do this with MP CONNECT? First, you need to sign on to the hosts

```
%let Host1 = <<IP address of Host 1>>;
filename Script "<<Path to script for logging into Host 1>>";
```

```
signon remote=Host1 script=Script;

%let Host2 = <<IP address of Host 2>>;
filename Script "<<Path to script for logging into Host 2>>";
signon remote=Host2 script=Script;
```

and remotely submit the code to do half the problem on each one:

```
rsubmit remote=Host1 wait=no;
   %Simulate(Output,500,seed=64382);
endrsubmit;

rsubmit remote=Host2 wait=no;
   %Simulate(Output,500,seed=31584);
endrsubmit;
```

The WAIT=NO option on the RSUBMIT statements is crucial. It specifies that the two halves of the problem are to run simultaneously, and thus enables the anticipated speed-up. So finally, you just wait for the jobs to finish:

```
waitfor _all_ Host1 Host2;
```

But this leaves out some crucial details:

- How do you get the definition for the %Simulate macro from the client to the hosts? You need to run the code to define it on each of the hosts. One way to do this is to nest the definition within another macro that remotely submits it:

  ```
  %macro RInit;
     rsubmit remote=Host&iHost;
        %macro Simulate(ds,n,seed=0); << Simulation code >>; %mend;
     endrsubmit;
  %mend;
  ```

  and run this for each host

  ```
  %let iHost = 1; %RInit;
  %let iHost = 2; %RInit;
  ```

- How do you do you retrieve the results on the hosts and put them back together on the client? If the results can be condensed to a few macro variables, then you can pass these macro variable values back through the MP CONNECT-ion using `%sysrput`. However, if your results are more complicated, you'll need to use data sets. SAS/CONNECT *Remote Library Services* enable you to connect the client to the WORK library of each host

  ```
  libname RWork1 slibref=WORK server=Host1;
  libname RWork2 slibref=WORK server=Host2;
  ```

  and you can use these to combine the two host Output data sets into a single one on the client:

  ```
  data Output; set RWork1.Output RWork2.Output; run;
  ```

Load Balancing

The last section presented a general scheme for distributing a task of size *N* across *k* hosts (*N*=1000 and *k*=2 above). If your hosts are all guaranteed to have equivalent horsepower, then this scheme will likely work well enough. However, if some of the *k* hosts are faster than others, then simply giving each of them a job of size *N/k* and waiting for them all to complete will make it seem like they are all as slow as the slowest one. What you need to do is to *balance* the load, giving faster hosts more of the job and slower ones less.

If you know the relative speeds of your hosts, then you can determine the proper load balance up front in such a way that all hosts finish their part of the job in about the same amount of time. However, when you are accessing many hosts on a network, you often cannot determine host speed ahead of time, since it is a function of jobs other users are running in real time. In this case, a reasonable approach is to break the size *N* task into small chunks of size $n < N/K$. As hosts finish up their chunk, give them another until the entire problem has been parceled out. This interactive form of load balancing will give faster hosts more chunks and thus more of the entire problem.

What is the optimal choice for the size *n* of the chunks? There are two things to consider: how much time it takes on the hosts to run each chunk, and how much time it takes on the client to manage submitting chunks. The distributed processing will be efficient when the hosts are given enough work so that they don't spend too much time waiting for the client to manage them, but not so much that the entire process ends up waiting for slow hosts to finish their task.

The %Distribute Macro System

The previous two sections discuss how to distribute a large SAS job efficiently over several hosts. Using MP CONNECT, you connect to and initialize your hosts, remotely submit pieces of the job, and then collect the results. Furthermore, you should balance the load so that fast hosts do more of the work; breaking the problems into more chunks than there are hosts and giving fast hosts new chunks to work on as they finish can be effective.

The system built around the %Distribute macro implements the approach to distributed SAS processing outlined above. In general, %Distribute addresses problems that consist of many replicate runs of a fundamental task. For example,

- Global integer optimization problems are often solved by steepest ascent (the fundamental task) with random restarts (the replicate runs).

- Monte Carlo methods in statistics rely on a set of pseudo-random observations (the replicate runs) of one or more difficult-to-compute test statistics (the fundamental task).

- Mining web data for e-intelligence applications often involves running queries and summaries (the fundamental task) over a massive number of different groups of data (the replicate runs).

If your problem can be factored in this way, then %Distribute may offer a useful approach.

Besides running the specified fundamental task the required replicate number of times, %Distribute offers two additional services:

- Applications of %Distribute often require a stream of random numbers, and %Distribute creates and manages such a stream for each host.

- In order to monitor your program, you would like to know where your fundamental processing time is spent. So %Distribute keeps track of how much time each host spends working on its fundamental tasks and waiting for the client to give it more work.

In order to use the %Distribute system, you must first tell it how to connect to and initialize the hosts and what fundamental task to perform. In order to do this, you need to create the following input for the system:

1. **_Hosts data set** - contains information about the hosts you want to run on
2. **%RInit macro** - defines the programs to be run on each host

Detailed descriptions of the input follow:

_Hosts Data Set

Define a data set named _Hosts in the current WORK library with an observation for each host you want to distribute the computation to. For each observation, set two (character) variables:

Variable Name	Contents
`Host`	Name of the host
`Script`	Signon script for host

These variables are used on the client, so make sure they are valid names and paths in this environment. The value of the `Host` variable should be either a name defined in your HOSTS file, a name defined to your name server, or the actual IP address of a machine to which you want to connect. The value of the `Script` variable is the unquoted full pathname of a script file to be used in a SIGNON statement for the specified HOST. If the script file exists in the directory from which SAS is invoked, you can specify only the file name instead of the full path.

%RInit Macro

This is what really defines the distributed computation. It needs to wrap (at least) two macro definitions

`%FirstRSub` - initialization on that host
`%TaskRSub` - the fundamental unit of distributed computation

within an RSUBMIT block, like so

```
%macro RInit;
   rsubmit remote=Host&iHost wait=yes macvar=Status&iHost;

      %macro FirstRSub;
         << Initialization for this host >>;
      %mend;

      %macro TaskRSub;
         << Fundamental unit of distributed computation >>;
      %mend;

      << Optional additional initializations >>;

   endrsubmit;
%mend;
```

It is your job to write the macros %FirstRSub and %TaskRSub so that they define the fundamental task. Note that while anything between %macro ...; and %mend; in the RSUBMIT block will be submitted on the host as expected, additional macro initializations on the host that are not between %macro ...; and %mend; may be intercepted by the client's macro processor by default. For example, if you want to initialize the macro variable HostMac to 1 for the host inside the RSUBMIT block but outside of %FirstRSub, you need to use the macro quoting function %nrstr() to keep the %let statement from being intercepted by the client, like so:

```
%macro RInit;
   rsubmit remote=Host&iHost wait=yes macvar=Status&iHost;
      %macro FirstRSub; ...; %mend;
      %macro TaskRSub;  ...; %mend;

      %nrstr(%%)let HostMac = 1;
      .
      .
      .
   endrsubmit;
%mend;
```

For more information on interaction between remote submitting and the macro language, see Doninger (2001).

As parameters for these macros, you can assume the following global macro variables to be available within the host SAS session:

Predefined host global macro	Contents
Rem_Host	Name of this host
Rem_iHost	Index of this host in _Host data set
Rem_NIter	Size of chunk
Rem_Seed	Random seed, rerandomized for each chunk

If you need other macro values (say, &Mac1, &Mac2, and &Mac3) to be passed from the client to the host, you should specify them in a %Macros macro, like so:

```
%macro Macros;
   %syslput rem_Mac1=&Mac1;
   %syslput rem_Mac2=&Mac2;
   %syslput rem_Mac3=&Mac3;
%mend;
```

The %Distribute macro will take care of running this macro for each of the hosts, so that you can use the macros &rem_Mac1, &rem_Mac2, and &rem_Mac3 in your definitions of %FirstRSub and %TaskRSub.

Once you have created the _Hosts data set and the %RInit macro, all you need to do to distribute the fundamental task across all the hosts is:

1. Use the %SignOn macro to sign on to the hosts.
2. Set the macro variables &NiterAll and &Niter to the size of the entire problem and the size of each chunk, respectively.
3. Use the %Distribute macro to perform the distribution.

As chunks of the fundamental task are parceled out to the hosts, a status line will be printed to the SAS log on the client, displaying

- **'.'** for hosts that are currently working,
- **'!'** for hosts that are currently waiting to be assigned work, and
- **'X'** for hosts that are unable to do work for some reason.

Additional status information includes the number of fundamental tasks submitted and completed so far. When the distribution is complete, the system will display a breakdown of how much work was done on each host, together with an over-all summary of the efficiency of the distribution. Finally, you can

4. use the `%RCollect` or `%RCollectView` macros to collect the results from each host into a data set or DATA step view that concatenates them all.

Further analysis of the collected results can be performed on the client.

Example: Distributing a Monte Carlo Simulation

In linear models, an experimental design can be concisely represented as a matrix *X*, and the eigenvalues of *X'X* give information about the efficiency of the design. In particular, the so-called E-efficiency of a design is related to the smallest eigenvalue of *X'X*. Suppose you are interested in the E-efficiency of a set of *k* normally distributed points in R^k, for different values of *k*. Creating random normal data and computing the minimum eigenvalue requires only a couple of lines of SAS/IML code

```
x = rannor((1:&k)`*(1:&k));
e = eigval(x`*x)[><];
```

but it takes a large sample to estimate the expected value of *e* with precision, and if you want to look at the expected value for many values of *k*, it can take a while. In order to distribute the job, you first need to wrap the above code so that it can be run by %Distribute on the hosts.

```
%macro RInit;
   rsubmit remote=Host&iHost wait=yes macvar=Status&iHost;

      /*
      / This macro initializes the host, preparing it to run
      / the fundamental task multiple times.  In this case,
      / all you need to do is initialize the data set in which
      / all the results for this host are to be collected.
      /-----------------------------------------------------*/
      %macro FirstRSub;
         options nonotes;
         data Sample;       /* Create an empty dataset        */
            if (0);
         run;
      %mend;

      /*
      / This macro defines the fundamental task.  SAS/IML is
      / used to perform the basic simulation the number of
      / times (&rem_NIter) required.  Each chunk of results
      / is created in a work data set called Sample, and all
      / the results for this host are collected in
      / Results.Sample&rem_iHost.
      /-----------------------------------------------------*/
      %macro TaskRSub;
         proc iml;

            /*
            / Initialize random numbers
```

```
            /------------------------------------------------*/
            x = rannor(&rem_Seed);

            /*
            / For each job in this chunk, create random data,
            / compute min eigenvalue, and save it.
            /------------------------------------------------*/
            free data;
            do i = 1 to &rem_Niter;
               x = rannor( (1:&rem_Dimen)`*(1:&rem_Dimen));
               e = eigval(x`*x)[><];
               data = data // e;
               end;

            /*
            / Save whole sample for this chunk.
            /------------------------------------------------*/
            create iSample var {E1};
            append from data;
         quit;

         /*
         / Append this sample to the accumulated set of all
         / samples created on this host.
         /------------------------------------------------*/
         data Sample; set Sample iSample;
         run;
      %mend;

   endrsubmit;
%mend;
```

Define this macro on the client and make a _Hosts data set as described above. One final bit of set-up is required: you need to tell %Distribute to pass the dimension of X down to the hosts. Note that we have assumed in `%TaskRSub` above that this is available in a macro `&rem_Dimen`. Assuming its value is associated with a macro variable `&Dimen` on the client, the following definition for `%Macros` will do the trick.

```
%macro Macros;
   %syslput rem_Dimen=&Dimen;
%mend;
```

Once these three components (`_Hosts`, `%Rinit`, and `%Macros`) have been set up, the following code will run a 10-dimensional simulation 1,000,000 times over the hosts in chunks of 2,000.

```
%inc "Distribute.sas";

/*
/  Arguments for the distribution.
/-------------------------------------------------------------*/
%let NIter    =    2000;
%let NIterAll = 1000000;

%SignOn;

/*
/  This performs the distributed computation for a
```

```
/  10-dimensional problem.
/*------------------------------------------------------------*/
%let Dimen = 10;
%Distribute;
```

For 25 hosts, the first few status lines look like this:

```
Stat: .!!!!!!!!!!!!!!!!!!!!!!: (2000,0)/1000000
Stat: ..!!!!!!!!!!!!!!!!!!!!!: (4000,0)/1000000
Stat: ...!!!!!!!!!!!!!!!!!!!!: (6000,0)/1000000
Stat: ....!!!!!!!!!!!!!!!!!!!: (8000,0)/1000000
Stat: .....!!!!!!!!!!!!!!!!!!: (10000,0)/1000000
```

The bottom line here, for example, indicates that the first five hosts are working on their tasks and the others are waiting to be given tasks. After a while, the tasks on the first hosts will begin to finish up, and they'll be given more to do:

```
Stat: !.........!!!!!!!!!!!!!: (22000,2000)/1000000
Stat: .!.........!!!!!!!!!!!!!: (26000,4000)/1000000
Stat: ..!!.........!!!!!!!!!!!!: (30000,8000)/1000000
```

Eventually, all hosts will have been given tasks and the status line will indicate which ones finish theirs while other hosts are being serviced by the client:

```
Stat: ..!.........!!.....!!...: (682000,642000)/1000000
Stat: ....!!...!...........!..: (692000,650000)/1000000
Stat: ...!...!........!......!.: (702000,660000)/1000000
Stat: ..........!!...!.........: (712000,668000)/1000000
```

Toward the end of the run, most hosts will be waiting while the last chunks to finish up, and a status line will be printed as each one does:

```
Stat: !.!!!!.!!!!!!!.!!!!!..!.!: (1000000,988000)/1000000
Stat: !.!!!!.!!!!!!!!!!!!!!!.!: (1000000,994000)/1000000
Stat: !.!!!!.!!!!!!!!!!!!!!!!!: (1000000,996000)/1000000
Stat: !!!!!!.!!!!!!!!!!!!!!!!!: (1000000,998000)/1000000
Stat: !!!!!!!!!!!!!!!!!!!!!!!!: (1000000,1000000)/1000000
```

For the 25 UNIX hosts that the above test was run on, the timing results look like this:

i Host	Host	No. Iter	Work Time/2000 Iter	Estimated Time for Entire Problem	Distribution Efficiency	Wait Time/2000 Iter
20	**UNIX68**	**30000**	**0:00:04**	**0:35:28**	**92%**	**0:00:01**
8	UNIX73	42000	0:00:03	0:27:50	72%	0:00:01
1	UNIX49	48000	0:00:03	0:27:19	71%	0:00:00
5	UNIX16	44000	0:00:03	0:27:08	70%	0:00:01
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
22	UNIX71	36000	0:00:03	0:25:05	65%	0:00:01
12	UNIX79	40000	0:00:03	0:25:02	65%	0:00:01

Total elapsed time: **0:01:33**

```
                    Cumulative working time:  0:26:22
                    Cumulative waiting time:  0:07:59
```

(The names of the machines have been changed to protect the innocent.) Most of the machines took about three seconds for each chunk of 2000 samples, but UNIX68 (in bold) took about four seconds and was consequently given fewer chunks to do. The bottom line is that a job that would have taken about a half-hour on one machine took just a minute and a half distributed across 25 machines.

You can use a DATA step view to collect the results from all 25 hosts into one data set and analyze them:

```
%RCollect(Sample,Sample / view=Sample);
proc means data=Sample n mean stderr lclm uclm;
   var e1;
run;
```

The results indicate that the minimum eigenvalue for a 10-dimensional, 10-point random normal design is about 0.0725 with about four digits of precision.

```
                      The MEANS Procedure

                    Analysis Variable : E1

                                          Lower 95%     Upper 95%
      N          Mean    Std Error     CL for Mean   CL for Mean
-------------------------------------------------------------------
1000000    0.0724593  0.000106880      0.0722498     0.0726688
-------------------------------------------------------------------
```

The original problem was to examine the relationship of the minimum eigenvalue to the dimension for a range of dimensions. Once all the set-up has been done, it is a simple matter to wrap a macro around the distribution and collection code:

```
%macro MinEVal;
   options nonotes;

   data MinEval; if (0); run;
   %do Dimen = 1 %to 30;
      %Distribute;
      %RCollectView(Sample,Sample);
      proc summary data=Sample;
         var e1;
         output out=_temp mean=Mean StdErr=StdErr
                          LCLM=LCLM UCLM=UCLM;
      data _temp; set _temp; Dimen = &Dimen;
      data MinEval; set MinEval _temp;
      run;
      %end;

   options notes;

%mend;
%MinEval;
```

This simulation would require 40 hours to run on a single machine, but it took less than two hours when distributed across 25 UNIX hosts. The results indicate that the expected value of the minimum eigenvalue is inversely related to the dimension.

Conclusion

The %Distribute system described in this paper, for distributing many replicates of a fundamental task across multiple hosts, has been applied to a variety of real-world problems. It enabled Westfall et al. (2001) and O'Brien and Tobias (2001) to use simulation to address important problems in statistical testing for analysis of variance for which analytical results were difficult or even impossible. The accuracy and completeness of these simulation approaches would have been infeasible without the distributed parallel processing capabilities in MP CONNECT. In particular, for the work of Westfall et al. (2001), MP CONNECT made it possible to condense three years' worth of computing time into just a month's worth of desktop time.

Global optimization using steepest ascent with random restarts is also amenable to distribution by this method. Thus, for example, it is straightforward to use this system with the OPTEX procedure in SAS/QC™ software to perform extensive searches for optimal experimental designs.

The system has been tested on a variety of configurations, using PCs running Windows NT and HP/UX UNIX boxes for both the client and the hosts. The system successfully used MP CONNECT to distribute a single computation between the U.S. and Germany, connecting through the Internet using only the IP addresses of the German machines. As many as 150 servers have been employed simultaneously.

The %Distribute system is very much a work in progress; we are still working to make it easier to configure as well as more robust. For example, although the current version of the system works well as long as all server connections stay active, handling for connection problems needs to be improved, to ensure that work is salvaged when some of the servers go down. Moreover, developing the %Distribute system has inspired areas for future work in MP CONNECT, including

- remote library services that make it easy for the servers to transfer data from and back to the client
- easy ways to pass macro definitions from the client down to the servers

The goal is to make it as easy as possible for you to use MP CONNECT to multiply by many times the power of SAS for solving complex problems.

Acknowledgment

Dr. Frank Bretz of the University of Hannover implemented and tested various versions of %Distribute and also made his machines available to us for testing. His insights on issues (i.e. bugs) have significantly improved both the software and this article, and we heartily thank him for his encouraging help.

References

Doninger, C. (2001), "Interaction Between Macro Facility and RSUBMIT," unpublished manuscript.

O'Brien, R., and Tobias, R. (2001), " Improving the Levene/Brown-Forsythe Test for Heterogeneity of Spread," presented at the Spring Meeting of the International Biometric Society, Eastern North American Region, March 25–28,2001, Charlotte, NC.

Westfall, P., Tobias, R., and Bretz, F. (2000), "Estimating Directional Error Rates of Step-wise Multiple Comparisons Methods Using Distributed SAS Computing and Variance Reduction," submitted for publication; draft available at http://www.sas.com/rnd/app/papers/directional.pdf.

Appendix - The %Distribute System

```
 /*************************************************************/
 /*                                                           */
 /*    NAME: DISTRIBUTE                                       */
 /*   TITLE: Distributed parallel processing in SAS using MP  */
 /*          CONNECT                                          */
 /*  SYSTEM: ALL                                              */
 /*   PROCS: TEMPLATE                                         */
 /* PRODUCT: SAS/CONNECT                                      */
 /*    DATA:                                                  */
 /*                                                           */
 /* SUPPORT: sasrdt                      UPDATE:  26FEB01     */
 /*     REF: See "Large-Scale Parallel Numerical Computation in  */
 /*          the SAS System", Cheryl Doninger & Randy Tobias.    */
 /*************************************************************/

 /*-------------------------------------------------------------------
The macros defined in this file are centered around the %Distribute
macro, which uses MP CONNECT tools from SAS/CONNECT software to employ
distributed processing for problems that consist of many replicate
runs of a fundamental task.  For example, global integer optimization
problems are often solved by steepest ascent (the fundamental task)
with random restarts (the replicate runs).  Similarly, Monte Carlo
methods in statistics rely on a set of pseudo-random observations (the
replicate runs) of one or more difficult-to-compute test statistics
(the fundamental task).  If your problem can be factored in this way,
then %Distribute may offer a useful approach.

Besides running the fundamental task you specify the required
replicate number of times, %Distribute offers two additional
services:

   - Problems of the type that %Distribute can handle often require
     a stream of random numbers, and %Distribute creates and
     manages such a stream for each host.

   - In order to monitor your program, you would like to know
     where your fundamental processing time is spent.  So
     %Distribute keeps track of how much time each host spends
     working on its fundamental tasks and waiting for the client
     to give it more work.

In order to use %Distribute system, you must first create the
following:
   1. _Hosts data set - containing information about the hosts you
      want to run on
   2. %RInit macro - defining the programs to be run on each host
(1) says how to connect to and initialize to the hosts; (2) defines
the fundamental task to be performed on each host.  Detailed
descriptions of the input is as follows:

   _Hosts data set
      Define a data set named _Hosts in the current WORK library with
      an observation for each host you want to distribute the
      computation to.  For each observation, set two (character)
      variables:

         Variable Name   Contents
```

```
         Host            Name of the host
         Script          signon script for host

      These variables are used on the client, so make sure they are
      valid names and paths in this environment

   %RInit macro
      This is what really defines the distributed computation.  It
      needs to wrap of (at least) two macro definitions

         %FirstRSub - initialization on that host
         %TaskRSub  - the fundamental unit of distributed computation

      within an RSUBMIT block, like so

         %macro RInit;
            rsubmit remote=Host&iHost wait=yes macvar=Status&iHost;

               %macro FirstRSub;
                  << Initialization for this host >>;
               %mend;

               %macro TaskRSub;
                  << Fundamental unit of distributed computation >>;
               %mend;

               << Optional additional initializations >>;

            endrsubmit;
         %mend;

      It is your job to write the macros %FirstRSub and %TaskRSub so
      that they define the fundamental task.  As parameters for these
      macros, you can assume the following global macro variables to
      be available within the host SAS session:

         Predefined host   Contents
         global macro
         Rem_Host          Name of this host
         Rem_iHost         Index of this host in _Host data set
         Rem_NIter         Size of chunk
         Rem_Seed          Random seed, rerandomized for each chunk

      If you need other macro values (say, &Mac1, &Mac2, and &Mac3) to
      be passed from the client to the host, you should specify them
      in a %Macros macro, like so:

         %macro Macros;
            %syslput rem_Mac1=&Mac1;
            %syslput rem_Mac2=&Mac2;
            %syslput rem_Mac3=&Mac3;
         %mend;

      The %Distribute macro will take care of running this macro for
      each of the hosts, so that you can use the macros &rem_Mac1,
      &rem_Mac2, and &rem_Mac3 in your definitions of %FirstRSub and
      %TaskRSub.

Once you have created the _Hosts data set and the %RInit macro, you can
distribute the fundamental task across all the hosts with just the
following four lines of code:
   %SignOn;
   %let NIterAll = <Size of entire problem>;
   %let NIter    = <Size of each chunk>;
   %Distribute;

As chunks of the fundamental task are parceled out to the hosts, a
status line will be printed to the SAS log on the client, displaying
   '.' for hosts that are currently working,
   '!' for hosts that are currently waiting to be assigned work, and
   'X' for hosts that are unable to do work for some reason.
Additional status information includes the number of fundamental tasks
```

```
submitted and completed so far.  When the distribution is complete,
the system will display a breakdown of how much work was done on each
host, together with an over-all summary of the efficiency of the
distribution.  Finally, you can
   - use the %RCollect or %RCollectView macros to collect the
     results from each host into a data set or DATA step view that
     catenates all of them.
Further analysis of the collected results can be performed on the
client.

-------------------------------------------------------------------------

 DISCLAIMER:

       THIS INFORMATION IS PROVIDED BY SAS INSTITUTE INC. AS A SERVICE
 TO ITS USERS.  IT IS PROVIDED "AS IS".  THERE ARE NO WARRANTIES,
 EXPRESSED OR IMPLIED, AS TO MERCHANTABILITY OR FITNESS FOR A
 PARTICULAR PURPOSE REGARDING THE ACCURACY OF THE MATERIALS OR CODE
 CONTAINED HEREIN.


-------------------------------------------------------------------------*/

/*
/  Name the files to which to write any log and listing produced by
/  the servers.
/-------------------------------------------------------------------------*/
%let DistLog = distribute.log;
%let DistLst = distribute.lst;

/*
/  Options to control distribution-monitoring information:
/-------------------------------------------------------------------------*/
%let DIST_DETAIL = 1;     /* Print timing info from servers?          */
%let DIST_STATUS = 1;     /* Print periodic status lines for servers? */
%let DIST_DEBUG  = 0;     /* Print debugging information              */




%*********************************************************************;
%*  MACRO: SignOn                                                    *;
%*  USAGE: %SignOn;                                                  *;
%*  DESCRIPTION:                                                     *;
%*    This macro signs on to the hosts, using the information in the *;
%*    _Hosts data set.  See comments above for a description.        *;
%*  NOTES:                                                           *;
%*    Assumes _Hosts data set has been created correctly.            *;
%*                                                                   *;
%*********************************************************************;
%macro SignOn;

%local notesopt; %let notesopt = %sysfunc(getoption(notes));
options nonotes;


/*
/  Retrieve global host information from the _Hosts data set and set
/  up Seed and Status variables for each host.
/-------------------------------------------------------------------------*/
%global NumHosts;

data _Hosts; set _Hosts;
   Status = '!';
   iHost = _N_;
   call symput('NumHosts',trim(left(_N_)));
run;

%do iHost=1 %to &NumHosts;
   %global Host&iHost Status&iHost;
   %end;
```

```
data _null_; set _Hosts;
   call symput('Host'||trim(left(iHost)),trim(left(Host))       );
   call symput('Seed'||trim(left(iHost)),round(100000*ranuni(1)));
run;

data _null_; set _Hosts;
   select (Status);
      when ('!') call symput('Status'||trim(left(iHost)),0);
      when ('X') call symput('Status'||trim(left(iHost)),1);
      when ('.') call symput('Status'||trim(left(iHost)),2);
      end;
run;

/*
/  For each host, retrieve script, sign on, and set up a RLS libref
/  to the host's work library.
/------------------------------------------------------------------*/
%do iHost = 1 %to &NumHosts;
   %put Starting up Host&iHost = &&host&iHost;

   data _null_; set _Hosts(where=(iHost=&iHost));
      call symput('Script' ,trim(left(Script )));
   run;

   options notes;

   proc printto log="&DistLog" print="&DistLst"; run;

   filename Script "&Script";
   signon remote=Host&iHost script=Script;
   filename Script;

   libname RWork&iHost slibref=WORK server=Host&iHost;

   %let Status&iHost = -1;
   %RInit;

   %let stimeropt = %sysfunc(getoption(stimer));
   options nostimer;
   proc printto; run;
   options &stimeropt;

   options nonotes;

%end;

/*
/  Print the initial status of each host.
/------------------------------------------------------------------*/
   %let StatLine =;
   data _null_; set _Hosts;
      call symput('StatLine',symget('StatLine')||trim(left(Status)));
   run;
   %put Stat: &StatLine;

options &notesopt;

%mend;



%*******************************************************************;
%*  MACRO: SignOff                                                 *;
%*  USAGE: %SignOff;                                               *;
%*  DESCRIPTION:                                                   *;
%*    This macro signs off the hosts.                              *;
%*******************************************************************;
%macro SignOff;

%local notesopt; %let notesopt = %sysfunc(getoption(notes));
options nonotes;
```

```
data _null_; set _Hosts;
   call symput('Host'  ||trim(left(iHost)),trim(left(Host))       );
   call symput('NumHosts'                ,trim(left(_n_))        );
   select (Status);
      when ('!')     call symput('Status'||trim(left(iHost)),0);
      when ('X')     call symput('Status'||trim(left(iHost)),1);
      when ('.','?') call symput('Status'||trim(left(iHost)),2);
      end;
run;


%do iHost = 1 %to &NumHosts;
   %put Stopping Host&iHost = &&host&iHost;

   options notes;

   proc printto log="&DistLog" print="&DistLst"; run;

   %let Status&iHost = -1;
   signoff remote=Host&iHost macvar=Status&iHost;

/*
   rsubmit remote=Host&iHost wait=yes macvar=Status&iHost;
   ;
   endrsubmit;
*/

   %let stimeropt = %sysfunc(getoption(stimer));
   options nostimer;
   proc printto; run;
   options &stimeropt;

   options nonotes;

%end;

%PrintStatus;

options &notesopt;

%mend;




%*****************************************************************;
%*  MACRO: _TaskRSub                                             *;
%*  USAGE: Internal macro only                                   *;
%*  DESCRIPTION:                                                 *;
%*    This macro submits a chunk of the fundamental task to the host *;
%*    with index nHost.                                          *;
%*  NOTES:                                                       *;
%*    Assumes the macro %TaskRSub has been previously defined on the *;
%*    host.  Do this in the RInit macro defined on the client.   *;
%*****************************************************************;
%macro _TaskRSub;
%global DIST_DEBUG;

   %if (^&DIST_DEBUG) %then %do;
      proc printto log="&DistLog" print="&DistLst"; run;
      %end;

   %let Status&nHost = -1;

   /*
   /  Keep track of what's been submitted to each host *on* the
   /  hosts themselves, to ensure wires don't get crossed.
   /-----------------------------------------------------------------*/
   rsubmit remote=Host&nHost wait=yes;
      %nrstr(%%)let TaskNSubm = %eval(&TaskNSubm + &rem_niter);
```

```
      %nrstr(%%)sysrput NSubm&rem_iHost = &TaskNSubm;
   endrsubmit;
   %syslput GlobalNSubm = &NSubm;


   /*
   /  Submit not only the task to this host, but also the random
   /  seed and time management code.
   /-------------------------------------------------------------------*/
   rsubmit remote=Host&nHost wait=no macvar=Status&nHost
           SYSRPUTSYNC=yes;
      data _null_;
         old_Seed = &rem_Seed;
         new_Seed = round(100000*ranuni(&rem_Seed));
         call symput('rem_Seed',trim(left(new_Seed)));
      run;

      data _TimeBetween;
         Host = "&rem_Host";
         now = datetime();
         call symput('_TimeStart_',now);
         TimeBetween = now - symget('_TimeEnd_');
      run;

      data _null_; call symput('_TimeStart_',datetime()); run;

      %TaskRSub;

      data _TimeEnd;
         Host = "&rem_Host";
         now = datetime();
         call symput('_TimeEnd_',now);
         Time = now - symget('_TimeStart_');
      run;

      data _Time; merge _TimeBetween _TimeEnd;
      data _TimeAll; set _TimeAll _Time;
      run;

      %nrstr(%%)let TaskNDone = %eval(&TaskNDone + &rem_niter);
      %nrstr(%%)sysrput NDone&rem_iHost = &TaskNDone;
   endrsubmit;
   %let NSubm = %eval(&NSubm + &NIter);

   %if (^&DIST_DEBUG) %then %do;
      options nostimer; proc printto; run; options stimer;
      %end;
%mend;



%*********************************************************************;
%*  MACRO: PrintStatus                                               *;
%*  USAGE: Internal macro only                                       *;
%*  DESCRIPTION:                                                     *;
%*    This macro prints a line summarizing the current status of     *;
%*    each server:                                                   *;
%*       '.' for hosts that are currently working,                   *;
%*       '!' for hosts that are currently waiting to be assigned     *;
%*           work, and                                               *;
%*       'X' for hosts that are unable to do work for some reason.   *;
%*    Additional status information includes the number of           *;
%*    fundamental tasks submitted and completed so far.              *;
%*********************************************************************;
%macro PrintStatus;

%local notesopt; %let notesopt = %sysfunc(getoption(notes));
options nonotes;

   data _Hosts; set _Hosts;
      xStatus = 1*symget('Status'||trim(left(iHost)));
```

```
      select (xStatus);
         when (0)  Status = '!';
         when (1)  Status = 'X';
         when (2)  Status = '.';
         otherwise do; Status = '?'; put iHost= xStatus=; end;
         end;
      NSubm = 1*symget('NSubm'||trim(left(iHost)));
      NDone = 1*symget('NDone'||trim(left(iHost)));
   run;
   %let StatLine =;
   data _null_; set _Hosts;
      call symput('StatLine',symget('StatLine')||trim(left(Status)));
   run;
   proc summary data=_Hosts;
      var NSubm NDone;
      output out=_SHosts sum=NSubm NDone;
   data _null_; set _SHosts;
      call symput('StatLine',symget('StatLine')
                  ||': ('||trim(left(put(NSubm    ,best20.)))
                  ||','  ||trim(left(put(NDone    ,best20.)))
                  ||')/' ||trim(left(put(&NIterAll,best20.))));
   run;
   %put Stat: &StatLine;

options &notesopt;

%mend;



%*********************************************************************;
%*  MACRO: Distribute                                                *;
%*  USAGE: %Distribute                                               *;
%*  DESCRIPTION:                                                     *;
%*    This macro performs the actual distribution, assuming that you *;
%*    have first created the _Hosts data set and the %RInit macro,   *;
%*    as described in the header comments above.                     *;
%*********************************************************************;
%macro Distribute;

%global DIST_DETAIL DIST_STATUS DIST_DEBUG;

%do iHost=1 %to &NumHosts;
   %global Host&iHost Seed&iHost Status&iHost NSubm&iHost NDone&iHost
           TimeWork&iHost TimeWait&iHost TimeFreq&iHost _ElapsedTime;
%end;

%if (^&DIST_DEBUG) %then %do;
%local notesopt;    %let notesopt = %sysfunc(getoption(notes));
   options nonotes;
   %end;
%else %do;
   options mprint mtrace;
   %end;

/*
/  Start the timer.
/------------------------------------------------------------------*/
data _null_; call symput('TimeStart',trim(left(datetime()))); run;

/*
/  Set up macro variables with the names and random number seeds for
/  all the hosts, and initialize monitoring information.
/------------------------------------------------------------------*/
data _null_; set _Hosts;
   call symput('Host'  ||trim(left(iHost)),
               trim(left(Host))       );
   call symput('Seed'  ||trim(left(iHost)),
               trim(left(round(100000*ranuni(1)))));
   call symput('NumHosts'                ,
               trim(left(_N_))        );
   select (Status);
```

```
      when ('!') call symput('Status'||trim(left(iHost)),0);
      when ('X') call symput('Status'||trim(left(iHost)),1);
      when ('.') call symput('Status'||trim(left(iHost)),2);
      end;
   call symput('NSubm' ||trim(left(iHost)),'0');
   call symput('NDone' ||trim(left(iHost)),'0');
run;

%let NSubm = 0;

/*
/  Start-up phase: submit the initialization task to each host, and
/  also the first fundamental task.
/------------------------------------------------------------------*/
%do iHost = 1 %to &NumHosts;
   %if (^&DIST_DEBUG) %then %do;
      proc printto log="&DistLog" print="&DistLst"; run;
      %end;

   options remote=Host&iHost;

   %syslput rem_Host  =&&host&iHost;
   %syslput rem_Seed  =&&seed&iHost;
   %syslput rem_niter =&niter;
   %syslput rem_iHost =&iHost;

   %Macros;

   rsubmit remote=Host&iHost wait=yes;

      %FirstRSub;

      data _null_;
         now = datetime();
         call symput('_TimeEnd_',now);
      run;

      data _TimeAll; if (0); run;

      %nrstr(%%)let TaskNSubm = 0;
      %nrstr(%%)sysrput NSubm&rem_iHost = &TaskNSubm;
      %nrstr(%%)let TaskNDone = 0;
      %nrstr(%%)sysrput NDone&rem_iHost = &TaskNDone;
   endrsubmit;
   %let nHost = &iHost; %_TaskRSub;

   %if (^&DIST_DEBUG) %then %do;
      options nostimer; proc printto; run; options stimer;
      %end;

   %if (&DIST_STATUS) %then %PrintStatus;
   %let NSubm = 0;
   %let NDone = 0;
   %do jHost = 1 %to %eval(&iHost);
      %let NSubm = %eval(&NSubm + &&NSubm&jHost);
      %let NDone = %eval(&NDone + &&NDone&jHost);
      %end;

   %do jHost = 1 %to %eval(&iHost);
      %let LastStat&jHost = &&Status&jHost;
      %end;


   /*
   /  We also recycle through previous hosts here, resubmitting tasks
   /  to them if they're ready for more: this saves a little time
   /  when some hosts finish their first task before the start-up
   /  phase is complete.
   /------------------------------------------------------------------*/
   %do jHost = 1 %to %eval(&iHost);
      %if ((&&LastStat&jHost = 0) & (&NDone < &niterall)) %then %do;
         %let nHost = &jHost; %_TaskRSub;
```

```
         %end;
      %end;

   %end;

%if (&DIST_STATUS) %then %PrintStatus;
%let NSubm = 0;
%let NDone = 0;
%do jHost = 1 %to %eval(&NumHosts);
   %let NSubm = %eval(&NSubm + &&NSubm&jHost);
   %let NDone = %eval(&NDone + &&NDone&jHost);
   %end;


/*
/  Monitor the MACVARs Status1, Status2, etc. to watch the jobs
/  finish.  As they do, resubmit new tasks for free hosts to work on.
/------------------------------------------------------------------------*/
%let Running = 1;
%do %while(%length(&Running));
   waitfor _any_
      %do iHost = 1 %to &NumHosts;
         %if (&&LastStat&iHost = 2) %then %do;
            Host&iHost
            %end;
         %end;
      ;

   %if (&DIST_STATUS) %then %PrintStatus;
   %let NSubm = 0;
   %let NDone = 0;
   %do jHost = 1 %to %eval(&NumHosts);
      %let NSubm = %eval(&NSubm + &&NSubm&jHost);
      %let NDone = %eval(&NDone + &&NDone&jHost);
      %end;

   %let Running   =;
   %do iHost = 1 %to &NumHosts;
      %let LastStat&iHost = &&Status&iHost;
      %if (&&LastStat&iHost = 2) %then %let Running  =&Running &iHost;
      %end;

   %do iHost = 1 %to &NumHosts;
      %if ((&&LastStat&iHost = 0) & (&NSubm < &niterall)) %then %do;
         %let nHost = &iHost; %_TaskRSub;

         %let LastStat&iHost = &&Status&iHost;
         %if (&&LastStat&iHost = 2) %then %let Running  =&Running &iHost;

         %let NSubm = 0;
         %let NDone = 0;
         %do jHost = 1 %to %eval(&NumHosts);
            %let NSubm = %eval(&NSubm + &&NSubm&jHost);
            %let NDone = %eval(&NDone + &&NDone&jHost);
            %end;
         %end;
      %end;
   %end;


/*
/  All tasks are finished: retrieve the timing information from each
/  of the hosts, ...
/------------------------------------------------------------------------*/
%do iHost = 1 %to &NumHosts;
   %let LastStat&iHost = &&Status&iHost;
   %if (&&LastStat&iHost = 0) %then %do;

      %if (^&DIST_DEBUG) %then %do;
         proc printto log="&DistLog" print="&DistLst"; run;
         %end;
```

```
      rsubmit remote=Host&iHost wait=yes;
         proc summary data=_TimeAll;
            var Time TimeBetween;
            output out=_TimeSumm mean=Work Wait;
         run;

         data _null_; set _TimeSumm;
            call symput('_TimeWork',trim(left(Work)));
            call symput('_TimeWait',trim(left(Wait)));
            call symput('_TimeFreq',trim(left(_FREQ_ )));
         run;
         %nrstr(%%)sysrput TimeWork&rem_iHost = &_TimeWork;
         %nrstr(%%)sysrput TimeWait&rem_iHost = &_TimeWait;
         %nrstr(%%)sysrput TimeFreq&rem_iHost = &_TimeFreq;
      endrsubmit;

      %if (^&DIST_DEBUG) %then %do;
         options nostimer; proc printto; run; options stimer;
         %end;

      %end;
   %end;


/*
/  ... add it to the _Hosts data set, ...
/------------------------------------------------------------------------*/
   data _null_;
      Elapse = datetime() - &TimeStart;
      call symput('_ElapsedTime',trim(left(put(Elapse,best.))));
   run;

   data TimeSumm; set _Hosts(keep=Host iHost);
      keep iHost Host NIter TimeWork EstElapsed Eff TimeWait TotalWork TotalWait;

      NIter    = &NIter*symget('TimeFreq'||trim(left(iHost)));
      TimeWork =      1*symget('TimeWork'||trim(left(iHost)));
      TimeWait =      1*symget('TimeWait'||trim(left(iHost)));

      EstElapsed = (&NIterAll/&NIter)*TimeWork;
      Eff        = (EstElapsed/&_ElapsedTime)/&NumHosts;
      TotalWork  = (NIter/&NIter)*TimeWork;
      TotalWait  = (NIter/&NIter)*TimeWait;
   run;


/*
/  ... and report it.
/------------------------------------------------------------------------*/
   %if (&DIST_DETAIL) %then %do;
   proc sort data=TimeSumm out=TimeSumm; by descending TimeWork;
   proc print data=TimeSumm label noobs;
      format TimeWork  time.;
      format TimeWait  time.;
      format EstElapsed time.;
      format Eff       percent.;
      label  NIter      = "No. Iter";
      label  TimeWork   = "Work Time/&Niter Iter";
      label  TimeWait   = "Wait Time/&Niter Iter";
      label  EstElapsed = "Estimated Time for Entire Problem";
      label  Eff        = "Distribution Efficiency";
      var iHost Host NIter TimeWork EstElapsed Eff TimeWait;
   run;
   %end;

   proc summary data=TimeSumm;
      var TotalWork TotalWait;
      output out=TotalTime Sum=TotalWork TotalWait;
   data _null_; set TotalTime;
      Elapsed = &_ElapsedTime;
      Eff     = (TotalWork/Elapsed)/&NumHosts;
      put " ";
```

```
      put "                      Total elapsed time:      " Elapsed   time.;
      put "                      Cumulative working time: " TotalWork time.;
      put "                      Cumulative waiting time: " TotalWait time.;
      put "                      Scaling efficiency:      " Eff percent8.2;
      put " ";
      put " ";
   run;

%if (^&DIST_DEBUG) %then %do;
   options &notesopt;
   %end;

%mend;



%*********************************************************************;
%*  MACRO: RCollect                                                  *;
%*  USAGE: %RCollect(<<Remote DS prefix>>,<<Client DS>>);            *;
%*  DESCRIPTION:                                                     *;
%*    This macro collects similarly named data sets from &n          *;
%*    libraries named RWork1, ..., RWork&n, into a single data set.  *;
%*  NOTES:                                                           *;
%*    In order to use a view to virtually collect the data sets, use *;
%*    'dsname / view=dsname' as the client data set name.            *;
%*********************************************************************;
%macro RCollect(from,to,n=);
%global NumHosts;

%if (^%length(%left(%trim(&n)))) %then %let n=&NumHosts;
%do iHost=1 %to &NumHosts;
   %global Host&iHost Status&iHost;
   %end;

data &to;
   set
   %do i=1 %to &N;
      %if (&&Status&i = 0) %then %do;
         RWork&i..&from
         %end;
      %end;
      ;
run;
%mend;


/*
Example

In linear models, an experimental design can be concisely represented
as a matrix X, and the eigenvalues of X'X give information about the
efficiency of the design.  In particular, the so-called E-efficiency
of a design is related to the smallest eigenvalue of X'X.  Suppose you
are interested in the E-efficiency of a set of k normally distributed
points in R^k, for different values of k.  Creating random normal data
and computing the minimum eigenvalue requires only a couple of lines
of SAS/IML code

   x = rannor((1:&k)`*(1:&k));
   e = eigval(x`*x)[><];

but it takes a large sample to estimate the expected value of e with
precision, and if you want to look at the expected value for many
values of k, it can take a while.  In order to distribute the job, you
first need to wrap the above code so that it can be run by %Distribute
on the hosts.

   %macro RInit;
      rsubmit remote=Host&iHost wait=yes macvar=Status&iHost;

         %macro FirstRSub;
            options nonotes;
```

```
            data Results.Sample&rem_iHost;
               if (0);
            run;
         %mend;

         %macro TaskRSub;
            proc iml;

               x = rannor(&rem_Seed);

               free data;
               do i = 1 to &rem_Niter;
                  x = rannor( (1:&rem_Dimen)`*(1:&rem_Dimen));
                  e = eigval(x`*x)[><];
                  data = data // e;
                  end;

               create Sample var {E1};
               append from data;
            quit;

            data Results.Sample&rem_iHost;
               set Results.Sample&rem_iHost Sample;
            run;
         %mend;

   endrsubmit;
%mend;
```

Define this macro on the client and make a _Hosts data set as described above. One final bit of set-up is required: you need to tell %Distribute to pass the dimension of X down to the hosts. Note that we have assumed in %TaskRSub above that this is available in a macro &rem_Dimen. Assuming its value is associated with a macro variable &Dimen on the client, the following definition for %Macros will do the trick.

```
   %macro Macros;
      %syslput rem_Dimen=&Dimen;
   %mend;
```

Once these three components (_Hosts, %Rinit, and %Macros) have been set up, the following code will run a 10-dimensional simulation 1,000,000 times over the hosts in chunks of 2,000.

```
   %inc "Distribute.sas";

   %let NIter    =    2000;
   %let NIterAll = 1000000;

   %SignOn;

   %let Dimen = 10;
   %Distribute;
```

For 25 hosts, the first few status lines look like this:

```
   Stat: .!!!!!!!!!!!!!!!!!!!!!!!: (2000,0)/1000000
   Stat: ..!!!!!!!!!!!!!!!!!!!!!!: (4000,0)/1000000
   Stat: ...!!!!!!!!!!!!!!!!!!!!!: (6000,0)/1000000
   Stat: ....!!!!!!!!!!!!!!!!!!!!: (8000,0)/1000000
   Stat: .....!!!!!!!!!!!!!!!!!!!: (10000,0)/1000000
```

The bottom line here, for example, indicates that the first five hosts are working on their tasks and the others are waiting to be given tasks. After a while, the tasks on the first hosts will begin to finish up, and they'll be given more to do:

```
   Stat: !.........!!!!!!!!!!!!!!: (22000,2000)/1000000
   Stat: .!.........!!!!!!!!!!!!!: (26000,4000)/1000000
   Stat: ..!!.........!!!!!!!!!!!: (30000,8000)/1000000
```

Eventually, all hosts will have been given tasks and the status line will indicate which ones finish theirs while other hosts are being serviced by the client:

```
Stat: ..!.........!!.....!!...: (682000,642000)/1000000
Stat: ....!!...!...........!..: (692000,650000)/1000000
Stat: ...!...!........!......!.: (702000,660000)/1000000
Stat: ..........!!...!.........: (712000,668000)/1000000
```

Towards the end of the run, most hosts will be waiting while the last chunks to finish up, and a status line will be printed as each one does:

```
Stat: !.!!!!.!!!!!!!.!!!!!..!.!: (1000000,988000)/1000000
Stat: !.!!!!.!!!!!!!!!!!!!!!!.!: (1000000,994000)/1000000
Stat: !.!!!!.!!!!!!!!!!!!!!!!!!: (1000000,996000)/1000000
Stat: !!!!!!.!!!!!!!!!!!!!!!!!!: (1000000,998000)/1000000
Stat: !!!!!!!!!!!!!!!!!!!!!!!!!: (1000000,1000000)/1000000
```

For the 25 UNIX hosts that the above test was run on, the timing results look like this:

i Host	Host	No. Iter	Work Time/2000 Iter	Estimated Time for Entire Problem	Distribution Efficiency	Wait Time/2000 Iter
20	UNIX68	30000	0:00:04	0:35:28	92%	0:00:01
8	UNIX73	42000	0:00:03	0:27:50	72%	0:00:01
1	UNIX49	48000	0:00:03	0:27:19	71%	0:00:00
5	UNIX16	44000	0:00:03	0:27:08	70%	0:00:01
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
22	UNIX71	36000	0:00:03	0:25:05	65%	0:00:01
12	UNIX79	40000	0:00:03	0:25:02	65%	0:00:01

```
Total elapsed time:       0:01:33
Cumulative working time:  0:26:22
Cumulative waiting time:  0:07:59
```

(The names of the machines have been changed to protect the innocent.) Most of the machines took about 3 seconds for each chunk of 2000 samples, but UNIX68 took a bit longer and was consequently given fewer chunks to do. The bottom line is that a job that would have taken about a half-hour on one machine took just a minute and a half distributed across 25 machines.

You can use a DATA step view to collect the results from all 25 hosts into one data set and analyze them:

```
%RCollect(Sample,Sample / view=Sample);
proc means data=Sample n mean stderr lclm uclm;
   var e1;
run;
```

The results indicate that the minimum eigenvalue for a 10-dimensional, 10-point random normal design is about 0.0725 with about 4 digits of precision.

The MEANS Procedure

Analysis Variable : E1

N	Mean	Std Error	Lower 95% CL for Mean	Upper 95% CL for Mean
1000000	0.0724593	0.000106880	0.0722498	0.0726688

The original problem was to examine the relationship of the minimum eigenvalue to the dimension for a range of dimensions. Once all the

set-up has been done, it is a simple matter to wrap a macro around the distribution and collection code:

```
   %macro MinEVal;
      options nonotes;

      data MinEval; if (0); run;
      %do Dimen = 1 %to 30;
         %Distribute;
         %RCollect(Sample,Sample / view=Sample);
         proc summary data=Sample;
            var e1;
            output out=_temp mean=Mean StdErr=StdErr
                             LCLM=LCLM UCLM=UCLM;
         data _temp; set _temp; Dimen = &Dimen;
         data MinEval; set MinEval _temp;
         run;
         %end;

      options notes;

   %mend;
   %MinEval;
```

This simulation would require 40 hours to run on a single machine, but it took less than two hours when distributed across 25 UNIX hosts. The results indicate that the expected value of the minimum eigenvalue is inversely related to the dimension.
*/