Parallel Programming

J.B. Boisvert

Parallel programming with Open MP requires an additional four lines of code and parallel do loops can be implemented to significantly reduce the run times of GSLIB style programs. Rather than rewrite all the code written at the CCG to take advantage of parallel programming, a wrapper was written that calls an executable multiple times in parallel. This allows for the parallelization of any GSLIB style program with very little modification to the original code. In addition to presenting this `script like` wrapper, developing parallel do loops are discussed in the Microsoft Visual Studio 2008 environment.

1. Introduction

Geostatistical programs are naturally amenable to parallel programming. Within virtually every geostatistical algorithm there are possibilities for significant speed gains if the programmer is fluent in parallel programming. Any independent calculation can be run in parallel; this could represent multiple realizations, the same algorithm applied to multiple variables, kriging various grid blocks in parallel, etc. In each one of these examples, the realizations/variable/grid locations could be run on a separate processor. Clearly, the ability to program in parallel would significantly reduce run times for typical geostatistical algorithms.

Programming in parallel is very easy. There are an additional four lines of code required to identify blocks of code that are written in parallel and Open MP automatically distributes the indices of the loop to the available processors, consider the simple loop in Figure 1.

The *!$omp parallel NUM_THREADS(nthr)* command indicates a section of code that is to be run in parallel with the number of threads to use specified by *nthr*; alternatively the command could simply be *!$omp parallel* and the number of processors would be selected automatically. The block of code that is run in parallel must be ended by the *!$omp end parallel* statement. A parallel do loop is indicated by the *!$omp do* and *!$omp end do* statements. In this simple example, the program `exefl` is run on multiple processors using the system call *sys_call_out.* A more typical example would be the generation of independent realizations in parallel (Figure 2).

In this example a subroutine "sgs" is called *nnn* times in parallel. The array *sgs_reals* is filled by multiple processors according to different INPUTS for each realization, such as a different random number for each call. In addition to the syntax indicated in Figure 2 there are compiler flags that must be set to indicate that the compiler should compile in parallel, these flags are /Qopenmp /threads /dbglibs /fpp. If you are not compiling from the command line where you would use these flags, the options can be found in the "Project Properties" menu, then Configuration Properties/Fortran/Languages/Process OpenMP Directives and Configuration Properties/ Fortran/ Languages/ Debug Multithreaded and Configuration Properties/ Fortran/ Preprocessor/ Preprocess Source File (see Figure 3 and Figure 4). The runtime library option should also be set to Multithreaded in Configuration Properties/Fortran/Libraries/Runtime Library. If you would like your application to run on other computers, you also have to indicate that the OMP libraries are static incase the target computer does not have the same dll's installed. Set the flag /Qopenmp-link:static which is not an option in Microsoft Visual studio, thus it must be added as an "Additioal Option" in Configuration Properties/Fortran/Libraries/Command Line. This location also indicates all of the flags that you have selected, to check your configuration, the configurations for the compiled `make_parallel` program are given in Figure 5.

The difficult part of programming in parallel is ensuring that everything within the parallel do loop is independent (note there are more advanced programming techniques that consider calculations that are not independent, but they will not be considered here). In a typical GSLIB implementation of sequential Gaussian simulation (SGS) there are search parameters, covariance lookup tables, data arrays, temporary arrays, etc. that

are used and written to during the construction of each realization. Each realization in SGS is mathematically independent but the implementation/code may not be independent.

At this point the generation of parallel do loops with Open MP (OMP) in the Microsoft Visual Studio 2008 environment has been covered. It is this authors' hope that future CCG code will take advantage of the simplicity of the OMP language and be incorporated in a meaningful way; however, it would not be efficient to rewrite all CCG code to date to incorporate parallel programming. Rather, a single program `make_parallel` has been written that can call any executable multiple times in parallel. Consider running SGS, each realization is independent and could be run by independent executables executing simultaneously. For 10 realizations, 10 separate SGS.exe executions would generate the realizations. `make_parallel` is essentially a script that calls the same executable multiple times with a slightly different parameter file. In the case of SGS the parameter file for each SGS.exe run would be identical save for the random number that would be incremented for each call.

2. Make_parallel.exe Program

A wrapper/script like parallel program was written as an alternative to modifying all previous GSLIB source code to consider parallel loops. `Make_parallel` is a program which does nothing but call another ".exe" program in parallel (this program will be referred to as the "main program"). The necessary inputs to `make_parallel` are the program name, number of processors to use, number of times to call the main program, and an output option (Figure 6).

Line 1: Set the number of threads to use. If set ==0, the program will detect the number of threads on the computer and use all of them. If set ==-1 the program will detect the number of threads on the computer and use nthreads-1. This is to allow one free processor for the user to continue to use the computer while make_parallel is running.

Line 2: Name of the main program to be called. This program must take its parameter file name as a command line argument rather than as an input from the screen (see below for more details on how to perform this modification).

Line 3: Name of the parameter file for the main program in Line 2.

Line 4: in the parameter file (Line 3) the text string from this line should appear (i.e. RAND). The `Make_parallel` program will call the main program (Line 2) a number of times while replacing the RAND string with incremented values.

Line 5: Number of times to call the main program. For efficiency this should be the same as the number of processors from Line 1; however, this is not a requirement. The second number is the starting value to substitute in the parameter file (Line 3) for the input character string (Line 4).

Line 6: a temporary directory that will be created (and then deleted) that will store all of the intermediate files.

Line 7: For each output file that the main program (Line 2) generates, a duplicate must be made to ensure that multiple processors do not write to the same file.

Line 8-10: name of the output files that will be duplicated. These names must be the same as the name in the main program parameter file (Line 3).

Line 11: Once the main program has been run multiple times there will be multiple output files (one for each main program call). This option will merge them as columns or append the files. Note that the length of each file must be the same. Selecting the option to "not merge" is the most efficient (see below).

Line 12: Option to delete the temp folder. Note that if you select the option to not merge the files, all the output files will be generated in the temp folder and should not be deleted.

The `make_parallel` program can be used to call any GSLIB style program with very minimal modifications to the original code. The only change to the original code is the manner in which the main program source code reads in the parameter file name. The system call (Figure 1) requires the name of the parameter file

to be a command line argument. The majority of GSLIB programs accept this type of input argument; however, the modifications are minor and are shown in Figure 7.

3. Example Call for SGSIM.exe and Time Trials

SGSIM.exe will be used to demonstrate the speedup of the `make_parallel` program. The parameters used for the trial run are:

- Windows 7 with Intel(R) Xeon(R) CPU X5677@3.47GHz
- 8 processors (dual quad core)
- 96 realizations (multiple of 8, 6, 4, 2 for running multiple processors)
- 150x150x150 grid (3.375M cells)
- 25 random data points
- Max of 10 original data and 20 previously simulated nodes
- Multigridding option (3 grid searches)
- 50 50 50 search radius

The parameter files for `SGSIM` and `Make_parallel` are shown in Figure 8 and Figure 9 respectively. Note that in this example for each call to `SGSIM` a different random number will be used. The 10 random numbers 69756, 69757, 69758, 69759, 69760, 69761, 69762, 69763, 69764 and 69765, as indicated in Figure 9, will replace the RAND string (Figure 8). `Make_parallel` will call SGSIM a total of 10 times, each time generating 10 realizations with different initial random number seeds. The output will be ten sgs_dbg.out, sgs.out and sgsim.trn files in the temporary directory "temp_dir798".

Figure 10 shows the run times for the scenario described above. Note that there are two options with the program, if the outputs need not be merged into a single file (no merging output) then the speedup realized is very close to the ideal speedup. Ideal speed up is determined by reducing the single processor time of 72.1min by the number of processors (ideal time for 8 processors is 72.1/8). The additional time for the "merging output" option comes from reading all output files and writing a single output with all realizations. The reading and writing speed is dependent on the disk read/write speed rather than the number of processors and requires approximately 12min for this example regardless of how many processors are used.

4. Conclusions

For non-expert FORTRAN programmers the idea of programming in parallel seems daunting when most are simply trying to master do loops, if statements and other basic programming tools. Once a programmer reaches a certain level they begin to try to make programs run better. With the reduction in price of multiple processor personal computers, parallel programing is a logical option for most geostatistical algorithms as they are naturally parallelizable. This paper discussed how openMP can be used to create simple independent parallel do loops. The modification to the original code is the addition of four lines of code (one above/below the do loop and one above/below the parallelized section of code). There are a number of complier flags that have to be set, otherwise parallel programming is identical to serial programming.

For those programmers or practitioners who do not wish to modify existing code or do not have the tools to develop the code as discussed in this paper, a program `make_parallel` has been created to parallelize any GSLIB program that can accept a command line argument. The drawback to using a program like `make_parallel` is that the file manipulation to generate a single output file is costly.

Figures

```
!$omp parallel NUM_THREADS(nthr)
    print *, ' We are using',nthr,' thread(s)'
    !$omp do
        do i=1,n2paralize
            command = trim(adjustl(exefl))//' '//trim(adjustl(par_list(i)))
            sys_call_out = systemqq(command)
        end do
    !$omp end do
!$omp end parallel
```

Figure 1: Example parallel code to write a parallel “do loop” to use the system call.

```
!$omp parallel

    !$omp do
        do i=1,nnn
            write(*,*)  'running in parallel, realization: '  , i
            call sgs(sgs_reals(:,i),INPUTS)
        end do
    !$omp end do

!$omp end parallel
```

Figure 2: Simple parallel code to build sgs realizations.

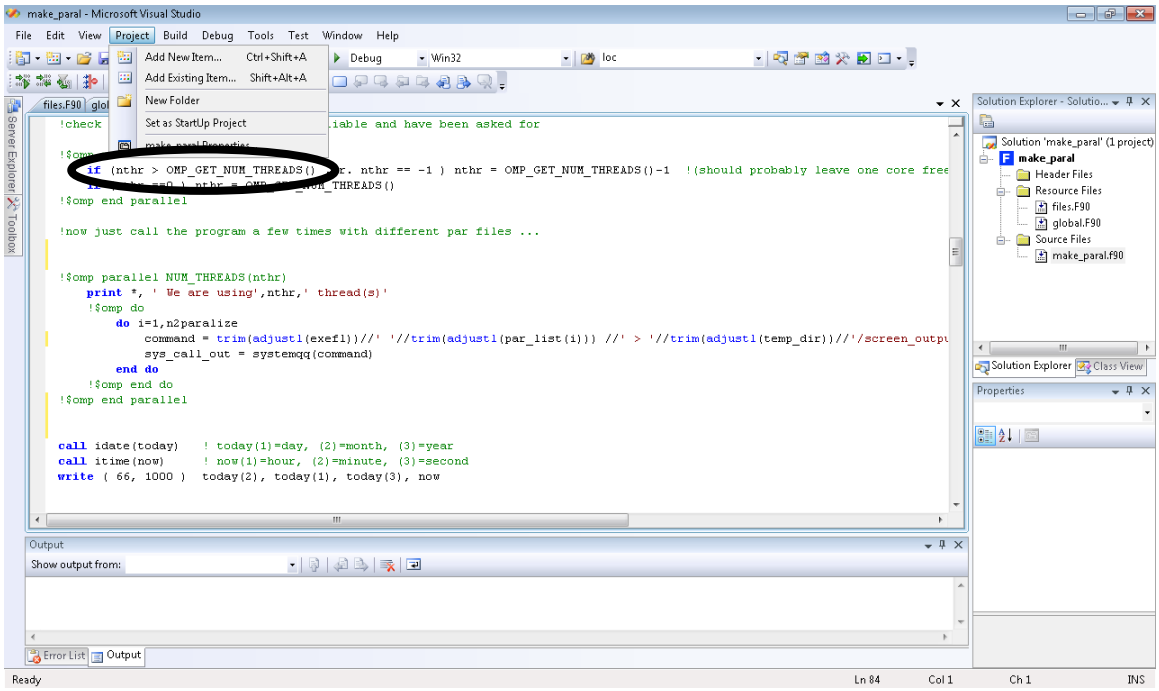


Figure 3: Location of project properties.

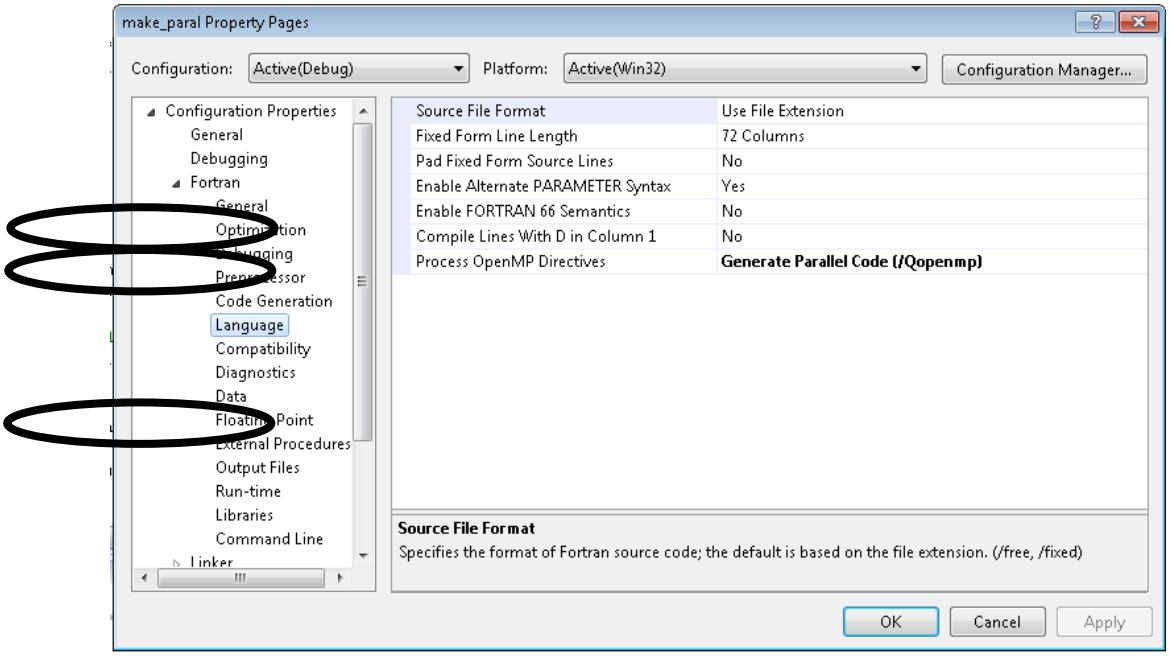


Figure 4: Location of the /Qopenmp flag.

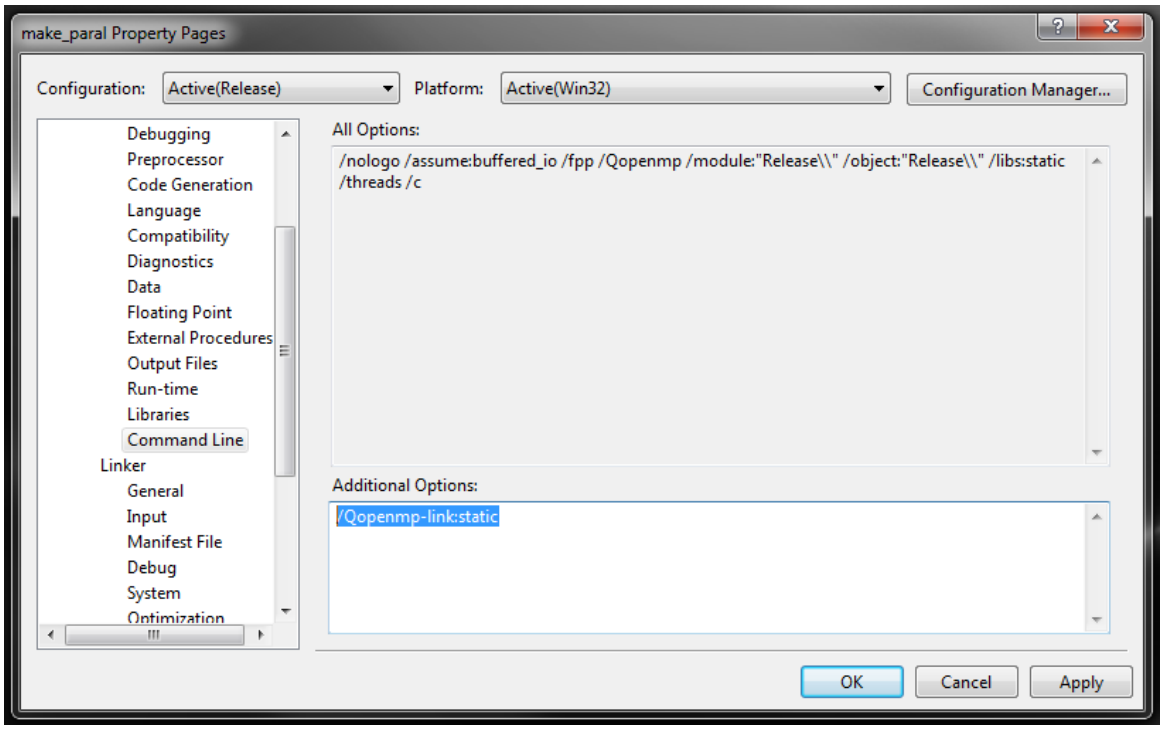


Figure 5: Make_parallel.exe configuration. All the flags are indicated under ¨All Options¨.

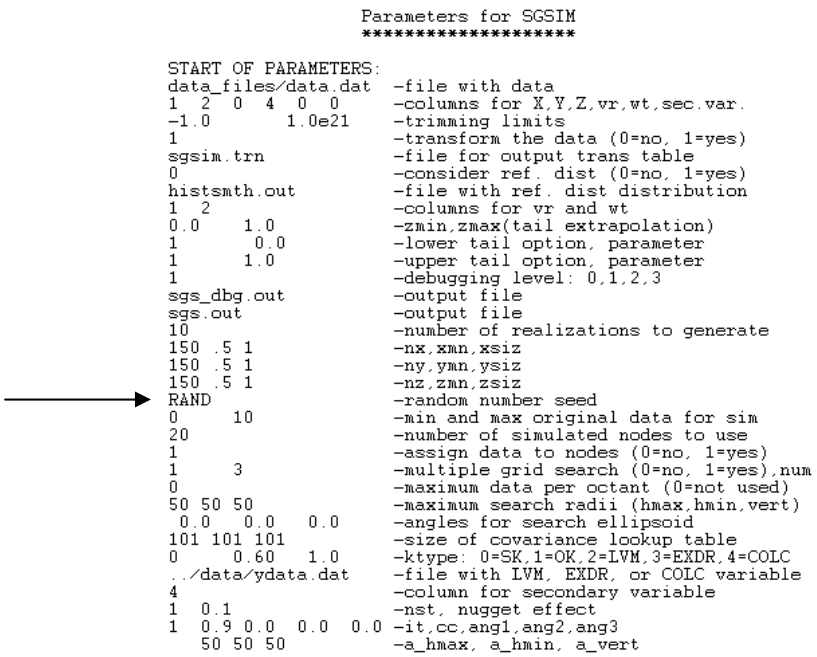
```
          Parameters for make_paral
          START
Line1:    1                          -number of threads to use (==0 to detect available, ==-1 to detect nthreads and then use nthreads-1
Line2:    sgsim_command_line.exe     -executable, this must be in the current directory, MUST HAVE PAR FILE AS A COMMAND LINE ARGUMENT .
Line3:    sgsim_paral.par            -par file to duplicate
Line4:    RAND                       -character string to find in the par file and increment
Line5:    10 69756                   -number of values to try, starting value
Line6:    temp_dir798                -temp directory to make all temp files, CAREFUL ALL FILES AND SUBFOLDERS HERE WILL BE DELETED !!!!
Line7:    3                          -number of output files, need to duplicate so each process does not write to the same file
Line8:    sgs.out                    -output file
Line9:    sgs_dbg.out                -output file
Line10:   sgsim.trn                  -output file
Line11:   -1                         -<0 do not merge files, 0 - append the first output file, 1=merge the first output file as columns
Line12:   1                          -delete the temp folder when done (0 =yes, 1=no)
```

Figure 6: Make_parallel.exe parameter file.

```
      subroutine readparm(MAXNOD,MAXCXY,MAXCTX,MAXCTY,MAXCTZ,MAXSBX,
     +                    MAXSBY,MAXSBZ)
c-----------------------------------------------------------------------
c
c                  Initialization and Read Parameters
c                  **********************************
c
c The input parameters and data are read in from their files. Some quick
c error checking is performed and the statistics of all the variables
c being considered are written to standard output.
c
c
c
c-----------------------------------------------------------------------
      use       msflib
      use       geostat
      include  'sgsim.inc'
      real      var(500)
      real*8    p,acorni,cp,oldcp,w
      character transfl*512,smthfl*512,tmpfl*512,datafl*512,outfl*512,
     +          dbgfl*512,lvmfl*512,str*512
      logical   testfl,testind
c
c Note VERSION number:
c
      write(*,9999) VERSION
 9999 format(/' SGSIM Version: ',f5.3/)
c
c Get the name of the parameter file - try the default name if no input:
c
      do i=1,512
            str(i:i) = ' '
      end do
      call getarg(1,str)
      if(str(1:1).eq.' ')then
            write(*,*) 'Which parameter file do you want to use?'
            read (*,'(a)') str
      end if
```

Getarg command to read in parameter file name

Figure 7: Original sgsim.for GSLIB code.



```
                  Parameters for SGSIM
                  ********************

START OF PARAMETERS:
data_files/data.dat  -file with data
1  2  0  4  0  0     -columns for X,Y,Z,vr,wt,sec.var.
-1.0       1.0e21    -trimming limits
1                    -transform the data (0=no, 1=yes)
sgsim.trn            -file for output trans table
0                    -consider ref. dist (0=no, 1=yes)
histsmth.out         -file with ref. dist distribution
1  2                 -columns for vr and wt
0.0    1.0           -zmin,zmax(tail extrapolation)
1       0.0          -lower tail option, parameter
1      1.0           -upper tail option, parameter
1                    -debugging level: 0,1,2,3
sgs_dbg.out          -output file
sgs.out              -output file
10                   -number of realizations to generate
150  .5 1            -nx,xmn,xsiz
150  .5 1            -ny,ymn,ysiz
150  .5 1            -nz,zmn,zsiz
RAND                 -random number seed
0     10             -min and max original data for sim
20                   -number of simulated nodes to use
1                    -assign data to nodes (0=no, 1=yes)
1     3              -multiple grid search (0=no, 1=yes),num
0                    -maximum data per octant (0=not used)
50 50 50             -maximum search radii (hmax,hmin,vert)
 0.0   0.0   0.0     -angles for search ellipsoid
101 101 101          -size of covariance lookup table
0     0.60   1.0     -ktype: 0=SK,1=OK,2=LVM,3=EXDR,4=COLC
../data/ydata.dat    -file with LVM, EXDR, or COLC variable
4                    -column for secondary variable
1  0.1               -nst, nugget effect
1  0.9 0.0  0.0  0.0 -it,cc,ang1,ang2,ang3
   50 50 50          -a_hmax, a_hmin, a_vert
```

Figure 8: Parameters for SGSIM example.

```
Parameters for make_paral
START
2                          -number of threads to use (==0 to detect available, ==-1 to detect nthreads and then use nthreads-1
sgsim_command_line.exe     -executable, this must be in the current directory, MUST HAVE PAR FILE AS A COMMAND LINE ARGUMENT .
sgsim_paral.par            -par file to duplicate
RAND                       -character string to find in the par file and increment
10 69756                   -number of values to try, starting value
temp_dir798                -temp directory to make all temp files, CAREFUL ALL FILES AND SUBFOLDERS HERE WILL BE DELETED !!!!
3                          -number of output files, need to duplicate so each process does not write to the same file
sgs.out                    -output file
sgs_dbg.out                -output file
sgsim.trn                  -output file
0                          -<0 do not merge files, 0 - append the first output file, 1=merge the first output file as columns
1                          -delete the temp folder when done (0 =yes, 1=no)
```

Figure 9: Parameters for `Make_parallel` example.

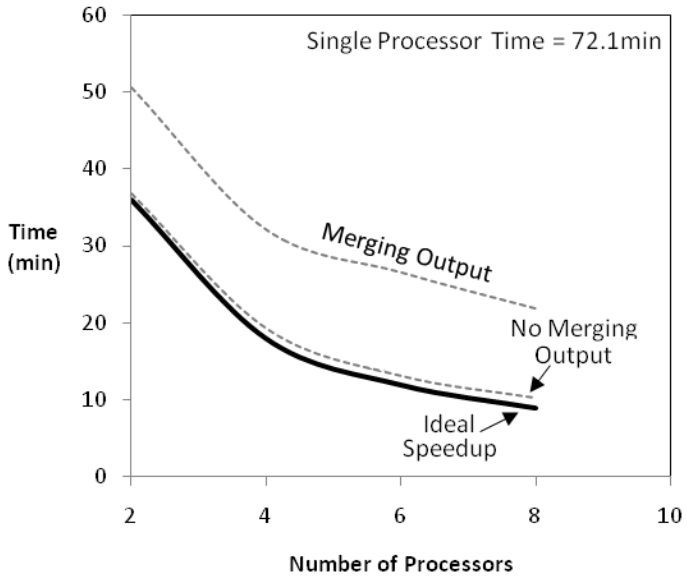


Figure 10: Time trials running SGSIM.exe to generate 96 realizations.