Examples

1. simple

This program demonstrates the use of `sprng` on one processor without explicit initialization.

Note that we have *included* the SPRNG *header file* We have also defined the *macro* `SIMPLE_SPRNG` before including the SPRNG header file in order to invoke the simple interface.

When `sprng` is called the first time, it initializes a default stream using default parameters, since no explicit initialization has been performed. It then returns double precision random numbers in [0,1).

Code:

Simple Interface	
C++	FORTRAN

Compilation: Example compilation on Intel

```
g++ -O3 -DLittleEndian -DINTEL -Wno-deprecated -g -ansi -I../include -o simple-simple simple-simple.cpp -L../lib -
lsprng
```

Note: `g++` is the C++ compiler.

Output:

```
[ren@phoenix EXAMPLES]$ ./simple-simple
 Printing 3 random numbers in [0,1):
0.014267
0.749392
0.007316
```

---

2. sprng

This program demonstrates the use of `sprng`, `isprng`, `init_sprng` and `print_sprng` on one process.

Note that we have *included* the SPRNG *header file*.

First we initialize a random number stream. The value of the *seed* is defined to be `985456376`. We call `print_sprng` immediately after initialization so that we can record the particular stream we obtained, for later reference.

`sprng` is next called to return double precision random numbers in [0,1) and then `isprng` is called to return random integers in [0,2^{31}).

Code:

Simple Interface		Default Interface	
C++	FORTRAN	C++	FORTRAN

Compilation: Example compilation on Intel

```
g77 -lc -lfrtbegin -lg2c -DINTEL -I. -I../include -o sprngf sprngf.F -L../lib -lsprng -lstdc++
```

Note:

```
     f77 is the FORTRAN 77 compiler.
```

Output:

```
[ren@phoenix EXAMPLES]$ ./sprngf
 Available generators; use corresponding numeral:
    lfg     --- 0
    lcg     --- 1
    lcg64     --- 2
    cmrg     --- 3
    mlfg     --- 4
    pmlcg     --- 5
 Type in a generator type (integers: 0,1,2,3,4,5):
1
 Print information about new stream:

48 bit Linear Congruential Generator with Prime Addend

  seed = 985456376, stream_number = 0 parameter = 0

 Printing 3 random numbers in [0,1):
0.707488
0.664048
0.005616
 Printing 3 random integers in [0,2^31):
      1873949618
      1484742006
       602016304
```

---

3. sprng_mpi

This program demonstrates the use of sprng with multiple processes.

Note that we have *included* the SPRNG and MPI *header files* at the beginning.

First we initialize MPI.

We then initialize a random number stream. The value of the *seed* is defined to be 985456376. We call print_sprng immediately after initialization so that we can record the particular stream we obtained, for later reference.

sprng is next called to return double precision random numbers in [0,1).

Code:

Simple Interface		Default Interface	
C++	FORTRAN	C++	FORTRAN

Compilation: Example compilation on Intel

mpiCC -I../include -o sprng-simple_mpi sprng-simple_mpi.cpp -L../lib -lsprng *MPI_Compile_Flags MPI_Link_Flags*

where *MPI_Compile_Flags* is the output of the command mpiCC --showme:compile
On our system, mpiCC --showme:compile produced the following ouptput
-I/usr/local/openmpi/gcc/include -I/usr/local/openmpi/gcc/include/openmpi -pthread

and *MPI_Link_Flags* is the output of the command mpiCC --showme:link
On our system, mpiCC --showme:link produced the following output
-pthread -L/usr/local/openmpi/gcc/lib -lmpi_cxx -lmpi -lorte -lopal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl

Note: mpiCC is the MPI C++ compiler.

Output:

```
[ren@phoenix EXAMPLES]$ mpirun -np 2 sprng-simple_mpi
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
1
Type in a generator type (integers: 0,1,2,3,4,5):

Process 0, print information about stream:

48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 0     parameter = 0

Process 0, random number 1: 0.70748766363408
Process 0, random number 2: 0.66404841879710
Process 0, random number 3: 0.00561565119362


Process 1, print information about stream:

48 bit Linear Congruential Generator with Prime Addend
```

```
        seed = 985456376, stream_number = 1     parameter = 0

Process 1, random number 1: 0.99144237784703
Process 1, random number 2: 0.15098918187292
Process 1, random number 3: 0.09757740755008
```

Note that the order in which the output is printed may differ.

---

4. fsprng_mpi

This program demonstrates the use of the single precision version of sprng with one *stream* per process.

Note that we have *included* the SPRNG and MPI *header files* sprng.h and mpi.h respectively at the beginning. Before including the SPRNG header file, we have defined the *macro*FLOAT_GEN, which results in single precision random numbers being generated by calls to sprng.

First we initialize MPI.

We then initialize a random number stream. The value of the *seed* is defined to be 985456376. We call print_sprng immediately after initialization so that we can record the particular stream we obtained, for later reference.

sprng is next called to return single precision random numbers in [0,1).

Code:

Simple Interface		**Default Interface**	
C++	**FORTRAN**	**C++**	**FORTRAN**

Compilation: Example compilation on Intel

```
g77 -lc -lfrtbegin -lg2c -DSPRNG_MPI -DINTEL -I/opt/openmpi/include -I/opt/openmpi/include/openmpi -pthread -I. -I../include -o fsprngf_mpi fsprngf_mpi.F -L../lib -lsprng -I/opt/openmpi/include -I/opt/openmpi/include/openmpi -pthread -pthread -L/opt/openmpi/lib -lmpi_cxx -lmpi -lorte -lopal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl -lstdc++
```

Note: g77 is the FORTRAN 77 compiler.

Output:

```
[ren@phoenix EXAMPLES]$ mpirun -np 2 fsprngf_mpi
 Available generators; use corresponding numeral:
    lfg     --- 0
    lcg     --- 1
    lcg64     --- 2
    cmrg     --- 3
    mlfg     --- 4
    pmlcg     --- 5
 Type in a generator type (integers: 0,1,2,3,4,5):
1
Process 0: Print information about stream:
Process 1: Print information about stream:

48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 1     parameter = 0


Process 1, random number 1: 0.991442
Process 1, random number 2: 0.150989
Process 1, random number 3: 0.097577
48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 0     parameter = 0

Process 0, random number 1: 0.707488
Process 0, random number 2: 0.664048
Process 0, random number 3: 0.005616
```

Note that the order in which the output is printed may differ.

---

5. seed

This program demonstrates the use of make_sprng_seed on one processor.

Note that we have *included* the SPRNG *header file* at the beginning.

First we initialize a random number stream. The value of a *seed* is obtained from a call to make_sprng_seed and is based on system date and time information. We call print_sprng immediately after initialization so that we can record the particular stream we obtained, for later reference.

sprng is next called to return double precision random numbers in [0,1).

Code:

Simple Interface		Default Interface	
C++	FORTRAN	C++	FORTRAN

Compilation: Example compilation on Intel

```
g++ -O3 -DLittleEndian -DINTEL -Wno-deprecated -g -ansi -I../include -o seed-simple seed-simple.cpp -L../lib -lsprng
```

Output:

```

Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
Type in a generator type (integers: 0,1,2,3,4,5):  1
 Printing information about new stream

48 bit Linear Congruential Generator with Prime Addend

  seed = 897339232, stream_number = 0 parameter = 0

 Printing 3 random numbers in [0,1):
0.008849
0.368579
0.217235
```

---

6. seed_mpi

This program demonstrates the use of make_sprng_seed with the MPI version of SPRNG in order to obtain the same *seed* in each process.

Note that we have *included* the SPRNG and MPI *header files* at the beginning. Before including the SPRNG header file, we have defined the *macro*USE_MPI, which enables SPRNG to make MPI calls to ensure that the same seed is obtained on each process.

First we initialize MPI.

We then initialize a random number stream. The value of the *seed* is obtained from a call to make_sprng_seed. This function makes MPI calls to ensure that the same seed is returned in each process, assuming that the MPI version of SPRNG has been installed. We call print_sprng immediately after initialization so that we can record the particular stream we obtained, for later reference.

sprng is next called to return double precision random numbers in [0,1).

Code:

Simple Interface		Default Interface	
C++	FORTRAN	C++	FORTRAN

Compilation: Example compilation on INTEL

```
mpiCC -O3 -DLittleEndian -DSPRNG_MPI -DINTEL -I/opt/openmpi/include -I/opt/openmpi/include/openmpi -pthread -Wno-deprecated -g -ansi -I../include -o seed-simple_mpi seed-simple_mpi.cpp -L../lib -lsprng -I/opt/openmpi/include -I/opt/openmpi/include/openmpi -pthread -pthread -L/opt/openmpi/lib -lmpi_cxx -lmpi -lorte -lopal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl
```

Output:

```

[ren@phoenix EXAMPLES]$ mpirun -np 2 seed-simple_mpi
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
1
Type in a generator type (integers: 0,1,2,3,4,5):



Process 1: seed =        958872146
Process 1: Print information about stream:

48 bit Linear Congruential Generator with Prime Addend

        seed = 958872146, stream_number = 1     parameter = 0

process 1, random number 1: 0.445218
process 1, random number 2: 0.289561
process 1, random number 3: 0.590956
Process 0: seed =        958872146
Process 0: Print information about stream:

48 bit Linear Congruential Generator with Prime Addend

        seed = 958872146, stream_number = 0     parameter = 0

process 0, random number 1: 0.702144
process 0, random number 2: 0.918198
process 0, random number 3: 0.791728


Note that the order in which the output is printed may differ.
```

7. message_mpi

This program demonstrates the use of pack_sprng and unpack_sprng for passing the state of a stream from one processor to another.

Note that we have *included* the SPRNG and MPI *header files* at the beginning.

First we initialize MPI.

We then initialize a random number stream on process 0. The value of the *seed* is defined to be 985456376. We call print_sprng immediately after initialization so that we can record the particular stream we obtained, for later reference.

sprng is next called by process 0 to return double precision random numbers in [0,1).

Process 0 then packs the state of the stream into an array and sends it to process 1, which unpacks it and prints a few more random numbers from that stream. In this process, the original default stream on process 1 is over-written by the unpacked stream.

Code:

Simple Interface		Default Interface	
C++	FORTRAN	C++	FORTRAN

Compilation: Example compilation on Intel

```
mpiCC -O3 -DLittleEndian -DSPRNG_MPI -DINTEL -I/opt/openmpi/include -I/opt/openmpi/include/openmpi -pthread -Wno-deprecated -g -ansi -I../include -o message-simple_mpi message-simple_mpi.cpp -L../lib -lsprng -I/opt/openmpi/include -I/opt/openmpi/include/openmpi -pthread -pthread -L/opt/openmpi/lib -lmpi_cxx -lmpi -lorte -lopal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl
```

Output:

```
[ren@phoenix EXAMPLES]$ mpirun -np 2 message-simple_mpi
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
1
Type in a generator type (integers: 0,1,2,3,4,5):

Process 0: Print information about stream:

48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 0     parameter = 0
```

```
Process 0: Print 2 random numbers in [0,1):
Process 0: 0.707488
Process 0: 0.664048
 Process 0 sends stream to process 1
 Process 1 has received the packed stream
Process 1: Print information about stream:

48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 0     parameter = 0

 Process 1 prints 2 numbers from received stream:
Process 1: 0.005616
Process 1: 0.872626


Note that the order in which the output is printed may differ.
```

---

8. 2streams_mpi

This program demonstrates the use of multiple streams on each process.

Note that we have *included* the SPRNG and MPI *header files* at the beginning.

First we initialize MPI and determine the number of processes and the rank of the local process.

We then initialize a random number stream. The value of the *seed* is defined to be 985456376. We shall have two streams per process, but is only one of these streams will be distinct. The other stream will be identical on each process. This is useful in situations in which we would like all the processes to take the same decision based on a random number. Thus the number of distinct streams is one more than the total number of processor, resulting in nstreams = nprocs + 1. For the distinct stream on each process, we initialize it with stream number as the rank of the local process, myid. The common stream is initialized with the stream number nprocs.

sprng is next called for each stream. It returns a double precision random numbers in [0,1) with the stream ID as its argument.

Finally, we free the memory used to store the states of the two streams by calls to free_sprng.

Code:

Simple Interface	**Default**	
Not Available	C++	FORTRAN

Compilation: Example compilation on Intel

```
mpiCC -O3 -DLittleEndian -DSPRNG_MPI -DINTEL -I/usr/local/openmpi/gcc/include -
I/usr/local/openmpi/gcc/include/openmpi -pthread -Wno-deprecated -g -ansi -I../include -o 2streams_mpi
2streams_mpi.cpp -L../lib -lsprng -I/usr/local/openmpi/gcc/include -I/usr/local/openmpi/gcc/include/openmpi -pthread
-pthread -L/usr/local/openmpi/gcc/lib -lmpi_cxx -lmpi -lorte -lopal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl
```

Output:

```

[ren@phoenix EXAMPLES]$ mpirun -np 2 2streams_mpi
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
1
Type in a generator type (integers: 0,1,2,3,4,5):  Process 0: Print information about new stream

48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 0     parameter = 0

Process 0: This stream is identical on all processes

48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 2     parameter = 0

Process 0, random number (distinct stream) 1: 0.707488
Process 0, random number (distinct stream) 2: 0.664048
Process 0, random number (shared stream) 1: 0.700906
Process 0, random number (shared stream) 2: 0.602204
Process 1: Print information about new stream
```

```
48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 1     parameter = 0

Process 1: This stream is identical on all processes

48 bit Linear Congruential Generator with Prime Addend

        seed = 985456376, stream_number = 2     parameter = 0

Process 1, random number (distinct stream) 1: 0.991442
Process 1, random number (distinct stream) 2: 0.150989
Process 1, random number (shared stream) 1: 0.700906
Process 1, random number (shared stream) 2: 0.602204
```

Note that the order in which the output is printed may differ.

---

9. spawn

This program demonstrates the use of spawn_sprng.

Note that we have *included* the SPRNG *header file* at the beginning.

First we initialize a random number stream. The value of the *seed* is defined to be 985456376. Since there is only one stream, the number of streams nstreams = 1 and the stream number streamnum = 0. stream stores the returned *ID* of the initialized stream. We call print_sprng immediately after initialization so that we can record the particular stream we obtained, for later reference.

sprng is next called to return double precision random numbers in [0,1), with the stream ID as its argument.

We next spawn two streams from stream. Their *ID's* are stored in an array allocated by sprng. The variable new points to this array. We print information about the newly spawned streams, and then print a few random numbers from the second stream spawned.

Finally, we free the memory used to store the states of the streams by calls to free_sprng.

Code:

Simple Interface	**Default Interface**	
Not Available	C++	FORTRAN

Compilation: Example compilation on Intel

```
g++ -O3 -DLittleEndian -DINTEL -Wno-deprecated -g -ansi -I../include -o spawn spawn.cpp -L../lib -lsprng
```

Output:

```
[ren@phoenix EXAMPLES]$ ./spawn
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
Type in a generator type (integers: 0,1,2,3,4,5):  1
 Print information about stream:

48 bit Linear Congruential Generator with Prime Addend

  seed = 985456376, stream_number = 0 parameter = 0

 Printing 2 random numbers in [0,1):
0.707488
0.664048
 Spawned two streams
 Information on first spawned stream:

48 bit Linear Congruential Generator with Prime Addend

  seed = 985456376, stream_number = 1 parameter = 0

 Information on second spawned stream:

48 bit Linear Congruential Generator with Prime Addend

  seed = 985456376, stream_number = 2 parameter = 0
 Printing 2 random numbers from second spawned stream:
0.700906
0.602204
```

---

10. convert

This program demonstrates converting user code to call SPRNG instead of the original random number generator myrandom.

Note that we have *included* the SPRNG *header file* at the beginning. We have also defined the *macro*SIMPLE_SPRNG before including the SPRNG header file in order to invoke the simple interface. We define the macro myrandom to sprng **after** including the SPRNG header. This replaces the former function call by calls to SPRNG instead.

We then add statements to initialize SPRNG.

Code:

Simple Interface		Default Interface
C++	FORTRAN	***Not Available***

Compilation: Example compilation on Intel

```
g++ -O3 -DLittleEndian -DINTEL -Wno-deprecated -g -ansi -I../include -o convert convert.cpp -L../lib -lsprng
```

Note: In the code provided with SPRNG, users need to define the macro CONVERT in order to get this converted code.

Output:

```
[ren@phoenix EXAMPLES]$ ./convert
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
Type in a generator type (integers: 0,1,2,3,4,5):  1
Print information about random number stream:

48 bit Linear Congruential Generator with Prime Addend

  seed = 985456376, stream_number = 0 parameter = 0

Printing 3 random numbers in [0,1):
0.707488
0.664048
0.005616
```

---

11. subroutine

This program demonstrates the use of SPRNG in subroutines in FORTRAN programs.

Note that we have *included* the SPRNG *header file* sprng_f.h at the beginning. We have also included this header file in the subroutine sub1. If we wished to define a macro for use by SPRNG, we need only do it once in any file, before the first time a SPRNG header is included.

Subroutine sub1 is then called, which makes calls to sprng.

Code:

Simple Interface	Default Interface
Not Available	FORTRAN

Compilation: Example compilation on Intel

```
g77 -lc -lfrtbegin -lg2c -DINTEL -I. -I../include -o subroutinef subroutinef.F -L../lib -lsprng -lstdc++
```

Output:

```
sprng/EXAMPLES:sif% a.out
[ren@phoenix EXAMPLES]$ ./subroutinef
 Available generators; use corresponding numeral:
    lfg     --- 0
    lcg     --- 1
    lcg64     --- 2
    cmrg     --- 3
    mlfg     --- 4
    pmlcg     --- 5
 Type in a generator type (integers: 0,1,2,3,4,5):
1
 Printing information about new stream

48 bit Linear Congruential Generator with Prime Addend

  seed = 985456376, stream_number = 0 parameter = 0

 Printing 3 double precision numbers in [0,1):
     1   0.7074876636340832
     2   0.6640484187971047
     3   0.0056156511936152
```

---

12. pi-simple

This is a serial program that demonstrates the use of several SPRNG features in a simple example application involving the computation of *PI* through a Monte Carlo method. We generate random points on a square of side 2 which circumscribes a circle of unit radius. We determine the proportion of points that fall within this circle. Since the ratio of the area of the circle to the area of the square is *PI/4*, the estimated value of *PI* is four times the proportion of points that lie within the circle.

The user specifies the number of samples to be generated on the square. The final state of the computations is stored in a file specified by the user. The user can restart the computations by reading from this file in subsequent runs. The user should also specify whether the current run is a new one (in which case a new random number stream is initialized) or an old one (in which case the state of the random number stream is read from a file).

Note that we have *included* the SPRNG *header file* at the beginning. We have also defined the *macro*SIMPLE_SPRNG before including the SPRNG header file in order to invoke the simple interface.

The function initialize is then called to take suitable actions based on the information input by the user. A new run involves initializing a random number stream. In a continuation of a previous run, the state of the random number stream is read from a file whose name is input by the user. The final state of the computations is also written into this file. The user should also input the number of samples to be generated in this run. In a new run, make_sprng_seed is called to produce a seed based on system date and time information.

Next the function count_in_circle is called, which generates points on a square of side 2. The number of points that fall on a circle of unit radius inscribed within this square is returned by this function.

We then estimate *PI* as four times the proportion of points that fall within the circle inscribed in the square.

Finally we call the function save_state to save the state of the random number stream, the cumulative number of sample points generated in all the runs, and the cumulative number of sample points that lie within the inscribed circle in all the runs.

Code:

C++	FORTRAN

Compilation: Example compilation on Intel

```
g++ -O3 -DLittleEndian -DINTEL -Wno-deprecated -g -ansi -I../include -o pi-simple pi-simple.cpp -L../lib -lsprng -lm
```

Output:

```
[ren@phoenix EXAMPLES]$ ./pi-simple
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
Type in a generator type (integers: 0,1,2,3,4,5):  1
Enter 9 for a new run, or 2 for the continuation of an old run:
9
Enter name of file to store/load state of the stream:
temp
Enter number of new samples:
10000

48 bit Linear Congruential Generator with Prime Addend

  seed = 1082937184, stream_number = 0 parameter = 0

pi is estimated as 3.1644000000000001 from        10000 samples.
 Error = 0.022807346, standard error = 0.016421834
```

```
[ren@phoenix EXAMPLES]$ ./pi-simple
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
Type in a generator type (integers: 0,1,2,3,4,5):  1
Enter 9 for a new run, or 2 for the continuation of an old run:
2
Enter name of file to store/load state of the stream:
temp
Enter number of new samples:
5000
pi is estimated as 3.1557333333333335 from        15000 samples.
 Error = 0.01414068, standard error = 0.013408371
```

Note: **Results may vary due to the use of a random seed.**

---

13. pi-simple_mpi

This is a parallel version of is a parallel version of pi-simple in which we divide the work of generating the samples among the available processes.

First we initialize MPI and call the function initialize to initialize a random number stream. If this is the continuation of a previously checkpointed run, then process 0 reads the previous state and passes to to each process.

Next we divide the number of samples to be generated among all the processes and then call the function count_in_circle as in the sequential case. We then sum the number of points that fall within the circle across all the processes by a call to MPI_REDUCE and estimate *PI* as four times the proportion of points that fall within this circle inscribed in the square.

Finally we call the function save_state to save the state of the random number stream, the cumulative number of sample points generated in all the runs, and the cumulative number of sample points that lie within the inscribed circle in all the runs. Each process passes the state of its stream to process 0, which then saves it to a file.

Code:

C++

Compilation: Example compilation on Intel

```
mpiCC -O3 -DLittleEndian -DSPRNG_MPI -DINTEL -I/opt/openmpi/include -I/opt/openmpi/include/openmpi -pthread -Wno-deprecated -g -ansi -I../include -o pi-simple_mpi pi-simple_mpi.cpp -L../lib -lsprng -I/opt/openmpi/include -I/opt/openmpi/include/openmpi -pthread -pthread -L/opt/openmpi/lib -lmpi_cxx -lmpi -lorte -lopal -ldl -Wl,--export-dynamic -lnsl -lutil -lm -ldl
```

Output:

```
[ren@phoenix EXAMPLES]$ mpirun -np 2 pi-simple_mpi
Available generators; use corresponding numeral:
   lfg     --- 0
   lcg     --- 1
   lcg64     --- 2
   cmrg     --- 3
   mlfg     --- 4
   pmlcg     --- 5
1
Type in a generator type (integers: 0,1,2,3,4,5):  Enter 9 for a new run, or 2 to continue an old run:
9
Enter name of file to store/load state of the stream:
temp
Enter number of new samples:
5000


48 bit Linear Congruential Generator with Prime Addend

        seed = 455562018, stream_number = 1     parameter = 0

48 bit Linear Congruential Generator with Prime Addend

        seed = 455562018, stream_number = 0     parameter = 0

pi is estimated as 3.1648000000000001 from         5000 samples.
        Error = 0.023207346, standard error = 0.02322398
```

Note: **Results may vary due to the use of a random seed.**

Jane Ren

ren@scs.fsu.edu
Last modified: 27 Oct, 2006