SALIH KILICLI – STAT 624 HOMEWORK 3

—— SETUP ——

module load intel/2019

I have written a C file that generates 1 billion exponentially distributed random numbers with alpha=2.0, beta=3.0 using different number generators and options. Lastly, I made the C file executable and then run it on my Terra account.

chmod u+x hw3.c

Problem 1) For this problem I have run the hw3.c file given below with Intel MKL using the commands

icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl

time ./hw3.sh

Problem 2) I have compiled, link and run the code with default GNU C Compiler using the commands

gcc -o hw3 hw3.c -O2 -m64 -I${MKLROOT}/include /

-L${MKLROOT}/lib/intel64 -Wl,–no-as-needed -lmkl_rt -lpthread -lm -ldl

time ./hw3.sh

Problem 3) I have switched between options VSL_BRNG_{250, MCG59, MT19937} and run them using the same commands in problem 1.

1) **vslNewStream(&stream, VSL_BRNG_R250, 777);** // **Generalized feedback shift register**
2) **vslNewStream(&stream, VSL_BRNG_MCG59, 777);** // **59-bit multiplicative congruential**
3) **vslNewStream(&stream, VSL_BRNG_MT19937, 777);** // **Mersenne Twister pseudorandom**

Problem 4) I have switched the fixed variables between NVECTOR={10, 10^3, 10^6} and REPS={10^8, 10^6, 10^3} to fix the # of random numbers 10^9 (1 billion).

1) **#define NVECTOR 1000L** **(10^3)**
   #define REPS 1000000L **(10^6)**
2) **#define NVECTOR 10L** **(10^1)**
   #define REPS 100000000L **(10^8)**
3) **#define NVECTOR 1000000L** **(10^6)**
   #define REPS 1000L **(10^3)**

The outputs for the problems and comparisons between different options is given in Final Remarks.

SALIH KILICLI – STAT 624 HOMEWORK 3

—— HW3.C FILE ——

```
#include <stdio.h>

#include "mkl_vsl.h"

#include "mkl.h"

#define NVECTOR 1000L                   //10L                   //1000000L

#define REPS 1000000L                   //100000000L            //1000L


// icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl                  # to compile in Intel

// gcc -o hw3 hw3.c -O2 -xHost -m64 -I${MKLROOT}/include \                  # to compile in GNU GCC

// -L${MKLROOT}/lib/intel64 -Wl, –no-as-needed -lmkl_rt -lpthread -lm -ldl

// time ./hw3                                                               # to run and time

int main() // hw3.c

{

/* Initializing r and average s */

double r[NVECTOR], s=0.0;

VSLStreamStatePtr stream;

int i, j;

//vslNewStream( &stream, VSL_BRNG_R250, 777 );        // A generalized feedback shift register generator

//vslNewStream( &stream, VSL_BRNG_MCG59, 777 );      // 59-bit multiplicative congruential generator

vslNewStream( &stream, VSL_BRNG_MT19937, 777 );      // Mersenne Twister pseudorandom num generator

/* Generating random numbers*/

for ( i=0; i<REPS; i++ ) {

        vdRngExponential(VSL_RNG_METHOD_EXPONENTIAL_ICDF, stream, NVECTOR, r, 2.0, 3.0 );

for ( j=0; j<NVECTOR; j++ ) s += r[j];

}
```

```
/* Deleting the stream */

vslDeleteStream( &stream );

printf( "Sample mean = %f (n=%ld)\n", s/(REPS*NVECTOR), REPS*NVECTOR );

return 0;

}
```

—— FINAL REMARKS ——

Running hw3.c executable file in Terra and collecting time information, I have created tables below in order to compare random generators and different options. I have also attached the outputs.

Problem 1: Output 1 – NVECTOR = 10^3 – REPS = 10^6 – Intel MKL with Exponential Distribution

```
[math3mantic@terra2 ~]$ ls
hw1  hw2  hw3  perl5
[math3mantic@terra2 ~]$ cd ~/hw3
[math3mantic@terra2 hw3]$ ls
hw3  hw3.c
[math3mantic@terra2 hw3]$ module load intel/2019
[math3mantic@terra2 hw3]$ icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl
[math3mantic@terra2 hw3]$ time ./hw3
Sample mean = 4.999862 (n=1000000000)

real    0m2.704s
user    0m2.478s
sys     0m0.031s
```

Problem 2: Output 2 – NVECTOR = 10^3 – REPS = 10^6 – GNU GCC

```
[math3mantic@terra1 hw3]$ gcc -o hw3 hw3.c -O2 -m64 -I${MKLROOT}/include -L${MKLROOT}/lib/intel64 -Wl,--no-as-needed -lmkl_rt -lpthread -lm -ldl
[math3mantic@terra1 hw3]$ time ./hw3
Sample mean = 4.999862 (n=1000000000)

real    0m4.015s
user    0m3.320s
sys     0m0.026s
```

Random Number Generator	Intel MKL	Linked GNU GCC
Mersenne Twister	real 0m2.701s user 0m2.684s sys 0m0.016s	real 0m4.015s user 0m3.320s sys 0m0.026s

It is clear from the outputs and the table given above that Intel MKL is almost 2x faster than default GNU C compiler in terms of speed of generating random numbers.

Problem 3: Output 3 – NVECTOR = 10^3 – REPS = 10^6 – Intel MKL with different generators

Outputs for 1) Generalized feedback shift operator, 2) 59-bit multiplicative congruential generator and the 3) Mersenne Twister pseudorandom number generator for Intel MKL given below.

```
[math3mantic@terra2 hw3]$ icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl
[math3mantic@terra2 hw3]$ time ./hw3
Sample mean = 4.999996 (n=1000000000)

real    0m3.322s
user    0m3.303s
sys     0m0.018s
[math3mantic@terra2 hw3]$ icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl
[math3mantic@terra2 hw3]$ time ./hw3
Sample mean = 4.999999 (n=1000000000)

real    0m2.390s
user    0m2.372s
sys     0m0.017s
[math3mantic@terra2 hw3]$ icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl
[math3mantic@terra2 hw3]$ time ./hw3
Sample mean = 4.999862 (n=1000000000)

real    0m2.724s
user    0m2.702s
sys     0m0.020s
```

Table below compares the speed of different random Intel MKL number generators:

Random Number Generator	Intel
59-bit multiplicative congruential generator VSL_BRNG_MCG59 Sample Mean = 4.999999	real 0m2.390s user 0m2.372s sys 0m0.017s
The Mersenne Twister pseudorandom generator VSL_BRNG_MT19937 Sample Mean = 4.999862	real 0m2.724s user 0m2.702s sys 0m0.020s
Generalized feedback shift register generator VSL_BRNG_R250 Sample Mean = 4.999996	real 0m3.322s user 0m3.303s sys 0m0.018s

It is evident from the table that for Intel RNG, fastest method is MCH59 which also gives the best estimate for sample mean.

Problem 4: Output 4 – Intel MKL number generator using Mersenne Twister is tested for different. NVECTORs. Outputs for NVECTOR = {10^3, 10, 10^6} and REPS = {10^6, 10^8, 10^3}, respectively:

```
[math3mantic@terra2 hw3]$ module load intel/2019
[[math3mantic@terra2 hw3]$ icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl
[[math3mantic@terra2 hw3]$ time ./hw3
Sample mean = 4.999862 (n=1000000000)

real    0m2.704s
user    0m2.478s
sys     0m0.031s
[[math3mantic@terra2 hw3]$ icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl
[[math3mantic@terra2 hw3]$ time ./hw3
Sample mean = 4.999862 (n=1000000000)

real    0m21.274s
user    0m21.078s
sys     0m0.030s
[[math3mantic@terra2 hw3]$ icc -o hw3 hw3.c -O2 -xHost -lmkl_rt -lpthread -lm -ldl
[[math3mantic@terra2 hw3]$ time ./hw3
Sample mean = 4.999862 (n=1000000000)

real    0m2.970s
user    0m2.946s
sys     0m0.022s
```

Intel MKL RNG	N=10	N=1000	N=1000,000
Mersenne Twister Sample Mean=4.999862	real 0m21.274s user 0m21.078s sys 0m0.030s	real 0m2.704s user 0m2.478s sys 0m0.031s	real 0m2.970s user 0m2.946s sys 0m0.022s

When the number of random numbers generated at a time is 10, the time to run increases dramatically. However, when the number of random numbers generated at a time is 1000 it takes less time to run than 1000,000 at a time. It is apparently because of the CPU architecture and neither generating very small nor very big numbers at a time yield the best result in terms of speed.