High Performance Computing

Jiarui XIE

Semester: 2018/2019
Superviser: David Hill

Catalogue

Introduction

In this exercise we will practice with a scientific library used for high performance computing in high energy physics. We will try to use parallel optimization to reduce the code running time.

1 Basics

1.1 Test the speed of MT with and without optimization

We used the following commands to compile、link and run the code.

```
g++ -c mt19973ar-cok.c –o2
g++ -o myMt mt19937ar-cok.c
time ./myMt
```

The time used showed in the following Figure1.1, there are not obvisely different, maybe due to my one-core computer.

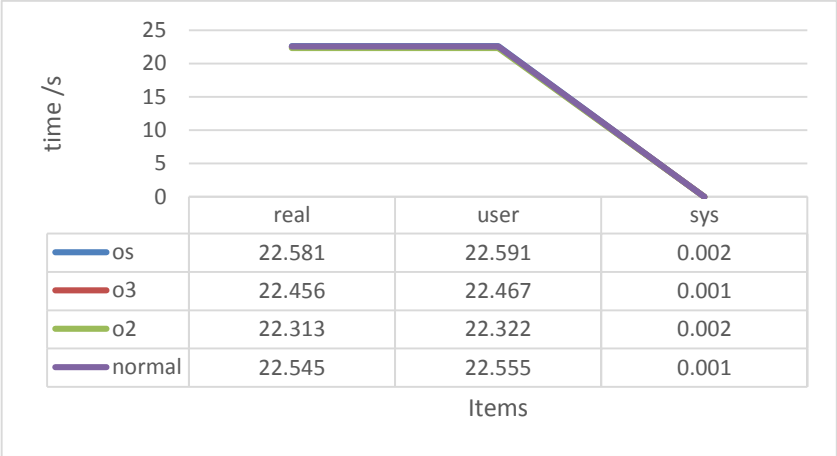

Figure1.1 Time of different parameter

2 CLHEP

2.1 Installation

We used the *time make* and *time make –j32* to install the CLHEP. The time they consume as the following Figure2.1. The sys time approximatively same, but the real time reduced as well as user time.

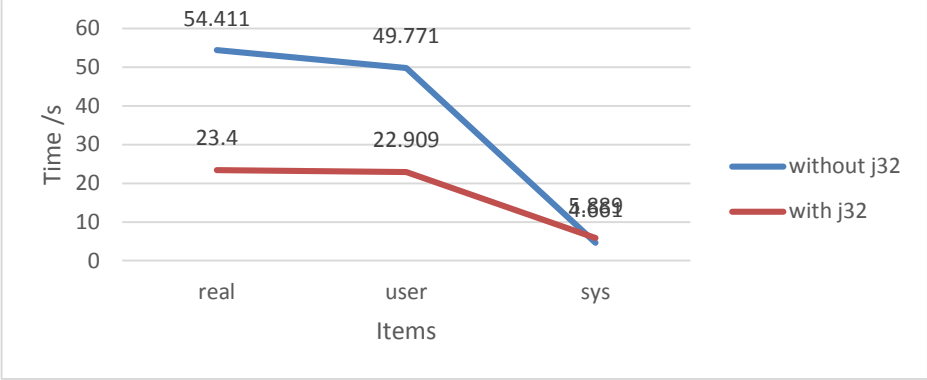

Figure2.1 Time of different parameter

2.2 Test with the code proposed at the end of TP file

We used the following commands to compile、link and run the code.

```
g++ -c testRand.cpp –I./include
g++ -o myExe testRand.cpp –I./include –L./lib –lCLHEP-Random-2.1.0.0
export LD_LIBRARY_PATH=./lib:$LD_LBRARY_PATH
./myExe
```

In addtation, we changed the code to test a parallelization technique of random streams, form the Figure2.2, we can see it has a obvisuly devation. It will work but the number generated will be the same with *restore()* Method. Finally, we worded the same but with 1000000 numbers the result , we can see it is stabler than 10 numbers.

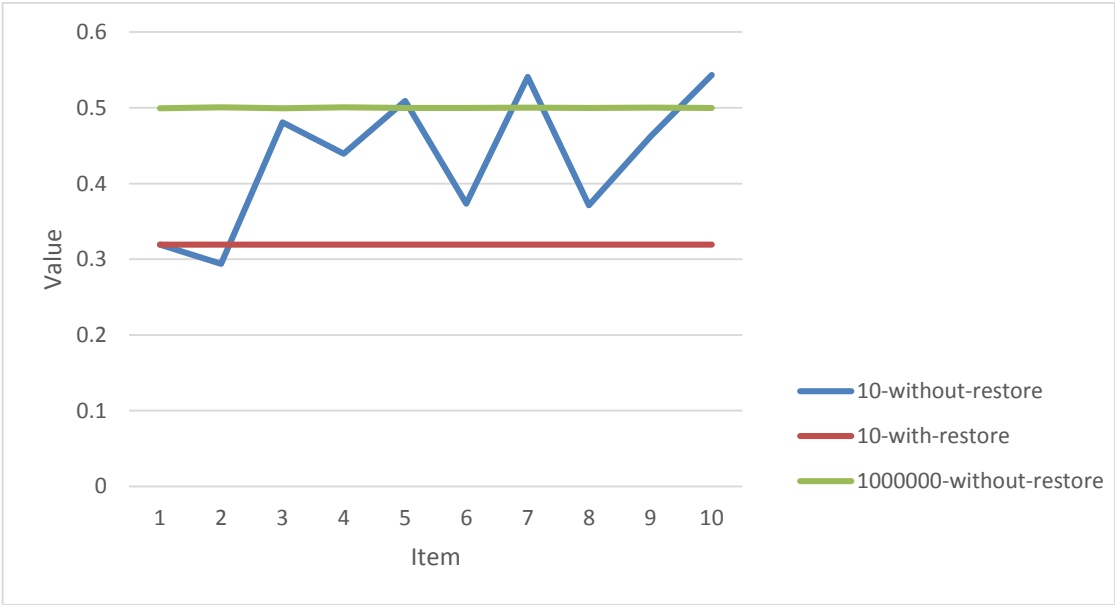

Figure2.2 without restoreStatus method

The import code is:

```
for (int j = 1; j <= 10; j++)
        {
                sum = 0.0;
                string nameOfFile = "status" ;
                nameOfFile.append(to_string(j));
                rs->saveStatus(nameOfFile.c_str());
                for(int i = 1; i < numberToDraw; i++)
                {
                        temp = rs->flat();
                        sum += temp;
                }
                cout << endl << nameOfFile << ":"<< sum / numberToDraw << endl;
                        //rs->restoreStatus(nameOfFile.c_str());  }
```

2.3 Test with the main test

We used the following commands to compile、link and run the code.

```
g++ -c testRandom.cc –I../include
g++ -o myTestRandom testRandom.cc –I../include –L../lib –lCLHEP-Random-2.1.0.0
export LD_LIBRARY_PATH=../lib:$LD_LBRARY_PATH
./myTestRandom
```

The result as Figure2.3

```
---------------------------- Random shooting test -----------------------------
                             --------------------
 >>> Random Engines available <<<

   > HepJamesRandom (default)
   > Rand
   > DRand48
   > Ranlux
   > Ranlux64
   > Ranecu
   > Hurd160
   > Hurd288
   > MTwist
   > Ranshi
   > DualRand
   > TripleRand

                  -----  Press <ENTER> to continue  -----
```

Figure2.1 Time of different parameter

3 Parallel Monte Carlo simulation

3.1 Xmtc.c

There are some errors in Xmtc.c, after correct we run it and got the result as following Figure3.1.

132: ul → pi 113: comment line 113 95: posterGraphic → posterGraphics 93: Ev.Type → Ev.type 65: 0.0 → 0,0 50: void main → int main

```
[xie@localhost lab2]$ g++ -c Xmtc.c
[xie@localhost lab2]$ g++ -o myXmtc Xmtc.c
[xie@localhost lab2]$ ./myXmtc
PI Calculation Part
500000 points, Approx of PI =3.142600, Approx Err:-0.001010
```

Figure3.1 Result of Xmtc.c

3.2 Vcd.c

There are some errors in Vcd.c, after correct we run it and got the result as following Figure3.2.

88: MirrorPuissances → MirrorPowers 119: it → long long it

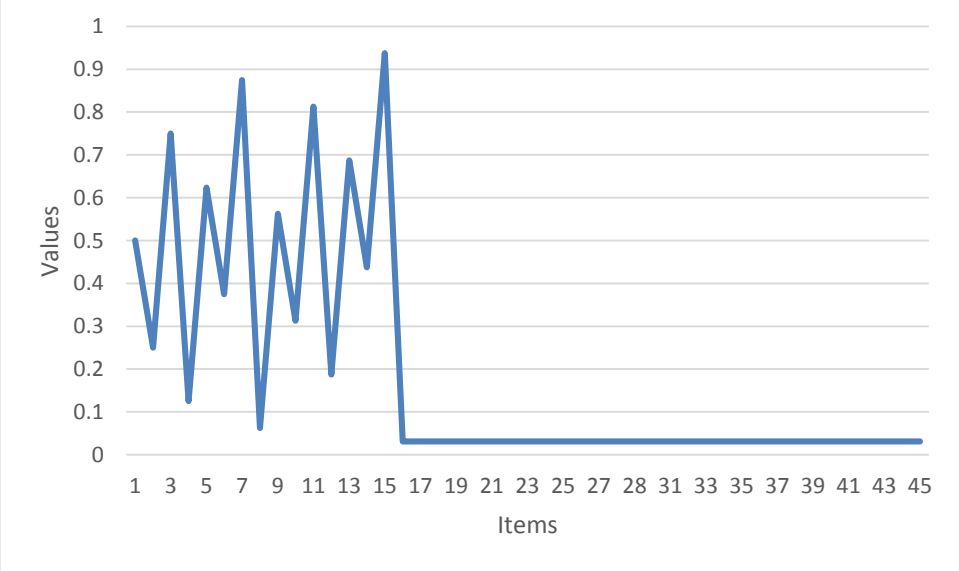

Figure3.2 result of Vcd.c

Conclusion

Parallelization technique can improve the speed of the code. But there are two challenges, first the parallelization need optimize, second we should put much time on designing the parallel code.