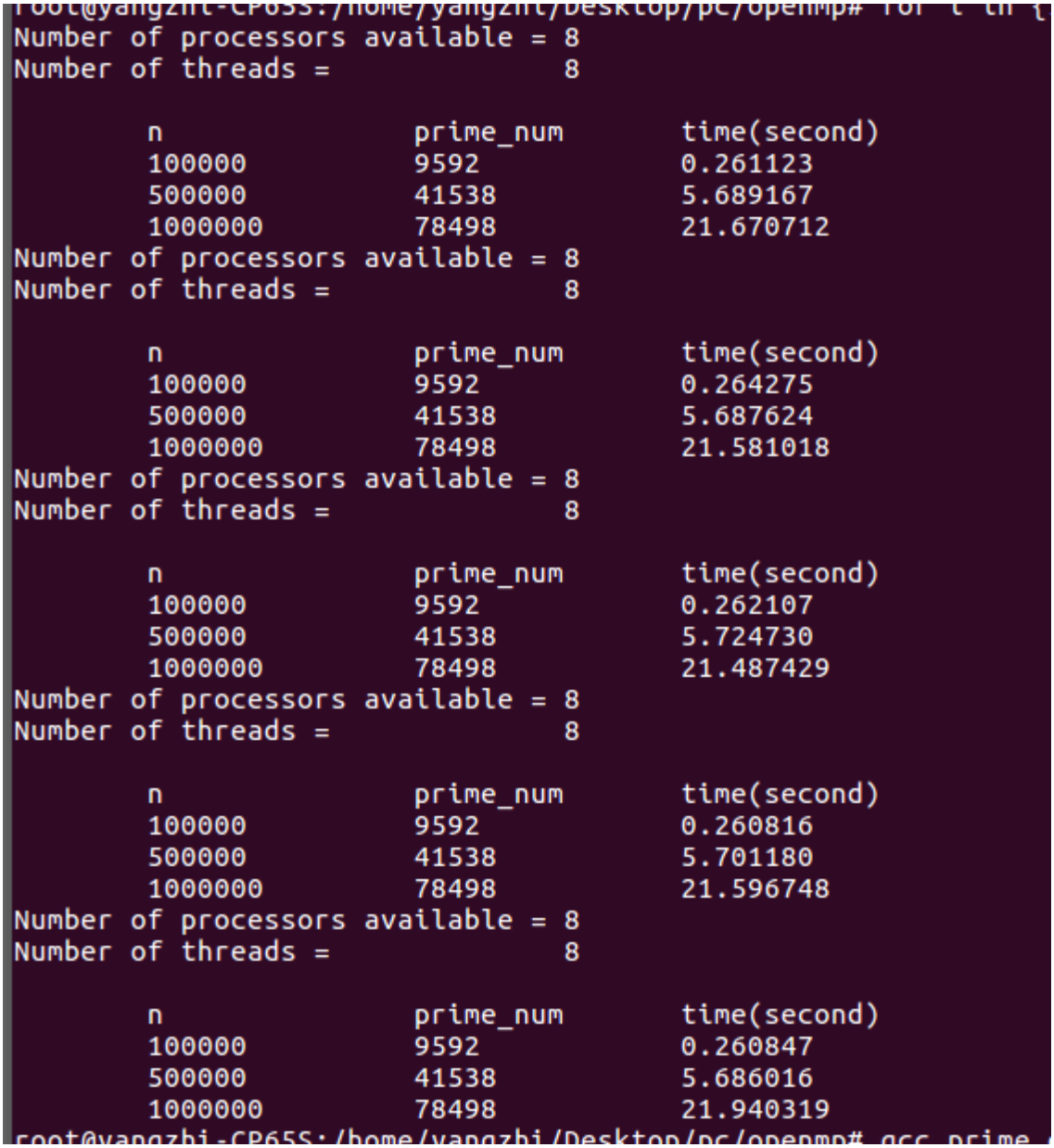
Find prime number

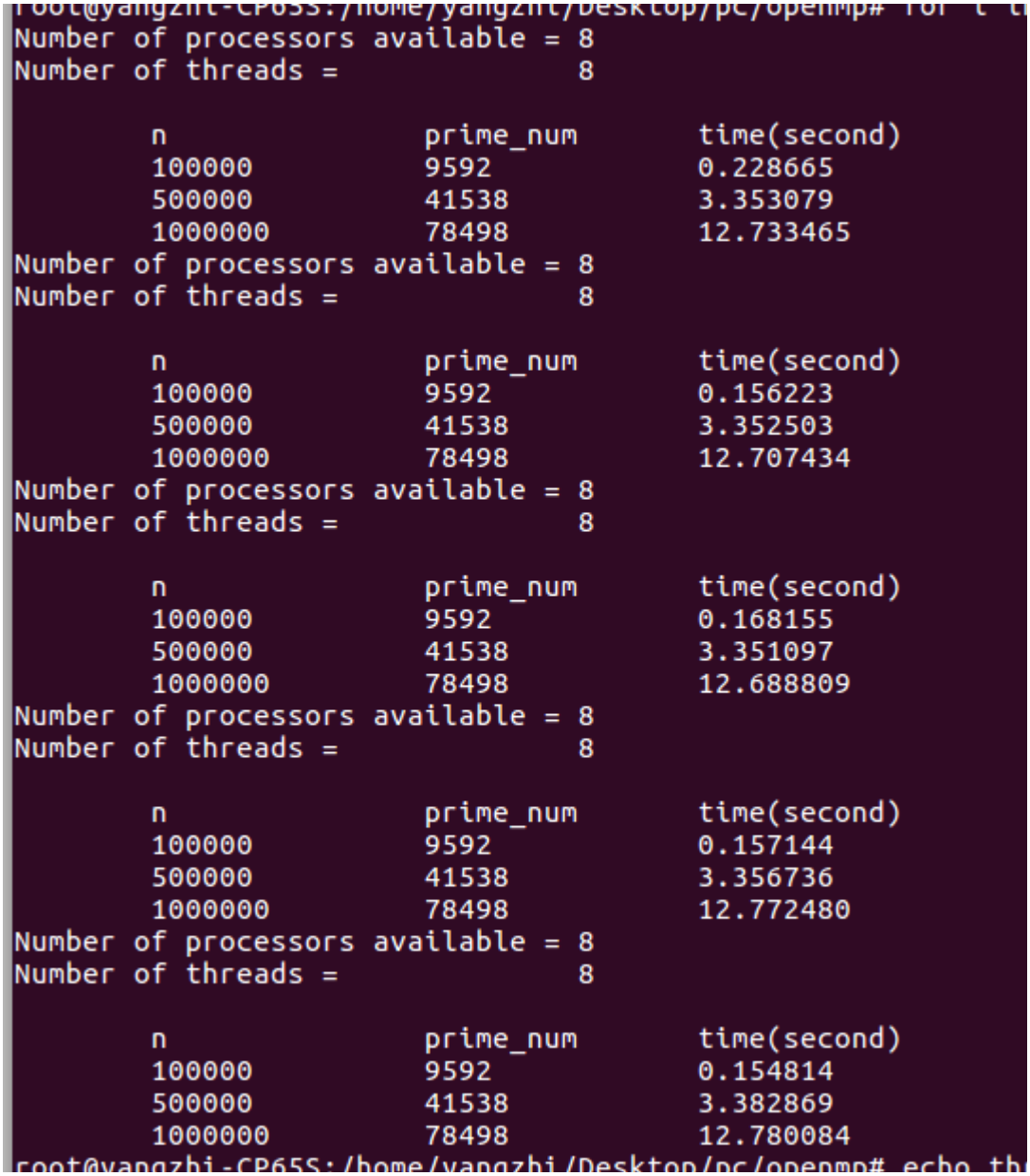
1 thread

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# ./a.out
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.446454
        500000          41538           7.660347
        1000000         78498           30.097349
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}
> do
> ./a.out
> done
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.361774
        500000          41538           7.768906
        1000000         78498           29.728132
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.366366
        500000          41538           7.907921
        1000000         78498           29.732908
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.360453
        500000          41538           7.766636
        1000000         78498           29.703908
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.352182
        500000          41538           7.933141
        1000000         78498           29.650899
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.355514
        500000          41538           7.861869
        1000000         78498           29.757864
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# ibus-daemon -drx
```

2 threads


root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for t in {
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.261123
500000 41538 5.689167
1000000 78498 21.670712
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.264275
500000 41538 5.687624
1000000 78498 21.581018
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.262107
500000 41538 5.724730
1000000 78498 21.487429
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.260816
500000 41538 5.701180
1000000 78498 21.596748
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.260847
500000 41538 5.686016
1000000 78498 21.940319
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc prime

4 threads


root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for t in
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.228665
500000 41538 3.353079
1000000 78498 12.733465
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.156223
500000 41538 3.352503
1000000 78498 12.707434
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.168155
500000 41538 3.351097
1000000 78498 12.688809
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.157144
500000 41538 3.356736
1000000 78498 12.772480
Number of processors available = 8
Number of threads = 8
n prime_num time(second)
100000 9592 0.154814
500000 41538 3.382869
1000000 78498 12.780084
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# echo th

8 threads

```
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.138568
        500000          41538           2.523613
        1000000         78498           8.955768
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.113492
        500000          41538           2.008447
        1000000         78498           9.046535
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.106332
        500000          41538           2.604765
        1000000         78498           7.799303
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.108049
        500000          41538           2.352422
        1000000         78498           8.720399
Number of processors available = 8
Number of threads =              8

        n               prime_num       time(second)
        100000          9592            0.104888
        500000          41538           2.183251
        1000000         78498           8.772462
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp#
```

cell automata

1 thread

```
cell400100threads1
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 100 400; done
okoktime: 0.181557 s
okoktime: 0.137733 s
okoktime: 0.140470 s
okoktime: 0.140039 s
okoktime: 0.136913 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 200 1000; done
okoktime: 1.669909 s
okoktime: 1.689056 s
okoktime: 1.681951 s
okoktime: 1.683491 s
okoktime: 1.692072 s
```

2 threads

```
cell400100threads2
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc cell_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 100 400; done
okoktime: 0.138963 s
okoktime: 0.138105 s
okoktime: 0.139928 s
okoktime: 0.139729 s
okoktime: 0.139456 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 200 1000; done
okoktime: 1.686001 s
okoktime: 1.712972 s
okoktime: 1.693973 s
okoktime: 1.712479 s
okoktime: 1.690500 s
```

4 threads

```
cell400100threads4
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc cell_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 100 400; done
okoktime: 0.125594 s
okoktime: 0.112103 s
okoktime: 0.075976 s
okoktime: 0.072341 s
okoktime: 0.072400 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 200 1000; done
okoktime: 1.523523 s
okoktime: 1.325799 s
okoktime: 1.084326 s
okoktime: 0.879717 s
okoktime: 0.884039 s
```

8 threads

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# echo cell400100threads8
cell400100threads8
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc cell_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 100 400; done
okoktime: 0.093682 s
okoktime: 0.076622 s
okoktime: 0.084106 s
okoktime: 0.071834 s
okoktime: 0.084092 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 200 1000; done
okoktime: 0.864817 s
okoktime: 0.852762 s
okoktime: 0.857915 s
okoktime: 0.845805 s
okoktime: 0.849830 s
```

plus:

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 200 5000; done
okoktime: 42.693989 s
okoktime: 42.276780 s
okoktime: 42.328090 s
okoktime: 43.069778 s
okoktime: 42.981298 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# echo cell400100threads8_plus
cell400100threads8_plus
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 200 5000; done
okoktime: 48.632455 s
^C
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc cell_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 200 5000; done
okoktime: 20.015132 s
okoktime: 23.204818 s
okoktime: 23.312928 s
okoktime: 25.530121 s
okoktime: 22.139462 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp#
```

matrix multiplication

1 thread

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc mult_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 500; done
time: 0.753727 s
time: 0.751846 s
time: 0.762616 s
time: 0.750345 s
time: 0.759727 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 1000; done
time: 7.375343 s
time: 7.013140 s
time: 7.964062 s
time: 7.647459 s
time: 6.778351 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 5000; done
time: 1817.400360 s
^C
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp#
```

2 threads

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc mult_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 500; done
time: 0.489789 s
time: 0.480724 s
time: 0.488285 s
time: 0.487596 s
time: 0.485102 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 1000; done
time: 3.931108 s
time: 3.882801 s
time: 3.868394 s
time: 3.892351 s
time: 3.852715 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 5000; done
^C
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 5000; done
time: 892.287384 s
```

4 threads

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 500; done
time: 0.314287 s
time: 0.416915 s
time: 0.259665 s
time: 0.257030 s
time: 0.296007 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 1000; done
time: 2.360910 s
time: 3.656342 s
time: 2.450993 s
time: 2.350229 s
time: 2.029989 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 5000; done
time: 508.190227 s
^C
```

8 threads

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 500; done
time: 0.344824 s
time: 0.288741 s
time: 0.276068 s
time: 0.279793 s
time: 0.274803 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 1000; done
time: 2.154473 s
time: 2.179978 s
time: 2.223179 s
time: 2.152042 s
time: 2.172611 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out 5000; done
time: 400.844721 s
^C
```

LU fatrorize

1 thread

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc lu_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out; done
time = 0.121617 s
time = 0.121184 s
time = 0.120090 s
time = 0.120801 s
time = 0.122144 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc lu_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out; done
time = 0.996376 s
time = 1.005487 s
time = 0.992971 s
time = 0.987156 s
time = 1.042458 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc lu_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out; done
time = 126.230616 s
time = 125.753975 s
time = 125.553665 s
time = 125.474679 s
time = 125.270768 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp#
```

2 threads

```
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc lu_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out; done
time = 0.063232 s
time = 0.065614 s
time = 0.062927 s
time = 0.063432 s
time = 0.064211 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc lu_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out; done
time = 0.524896 s
time = 0.513657 s
time = 0.534390 s
time = 0.517159 s
time = 0.519008 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# gcc lu_openmp.c -fopenmp
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp# for i in {1..5}; do ./a.out; done
time = 69.068050 s
time = 70.501650 s
time = 69.462477 s
time = 70.308563 s
time = 70.428515 s
root@yangzhi-CP65S:/home/yangzhi/Desktop/pc/openmp#
```

4 threads

```
yangzhi@yangzhi-CP65S:~/Desktop/pc/openmp$ for i in {1..5}
> do
> ./a.out
> done
time = 0.037244 s
time = 0.068104 s
time = 0.039011 s
time = 0.037718 s
time = 0.033443 s
yangzhi@yangzhi-CP65S:~/Desktop/pc/openmp$ gcc lu_openmp.c -fopenmp
yangzhi@yangzhi-CP65S:~/Desktop/pc/openmp$ for i in {1..5}; do ./a.out ; done
time = 0.285921 s
time = 0.301180 s
time = 0.282022 s
time = 0.288983 s
time = 0.284199 s
yangzhi@yangzhi-CP65S:~/Desktop/pc/openmp$ gcc lu_openmp.c -fopenmp
yangzhi@yangzhi-CP65S:~/Desktop/pc/openmp$ for i in {1..5}; do ./a.out ; done
time = 69.329573 s
time = 67.379222 s
time = 67.059076 s
time = 68.702966 s
time = 69.075347 s
yangzhi@yangzhi-CP65S:~/Desktop/pc/openmp$
```

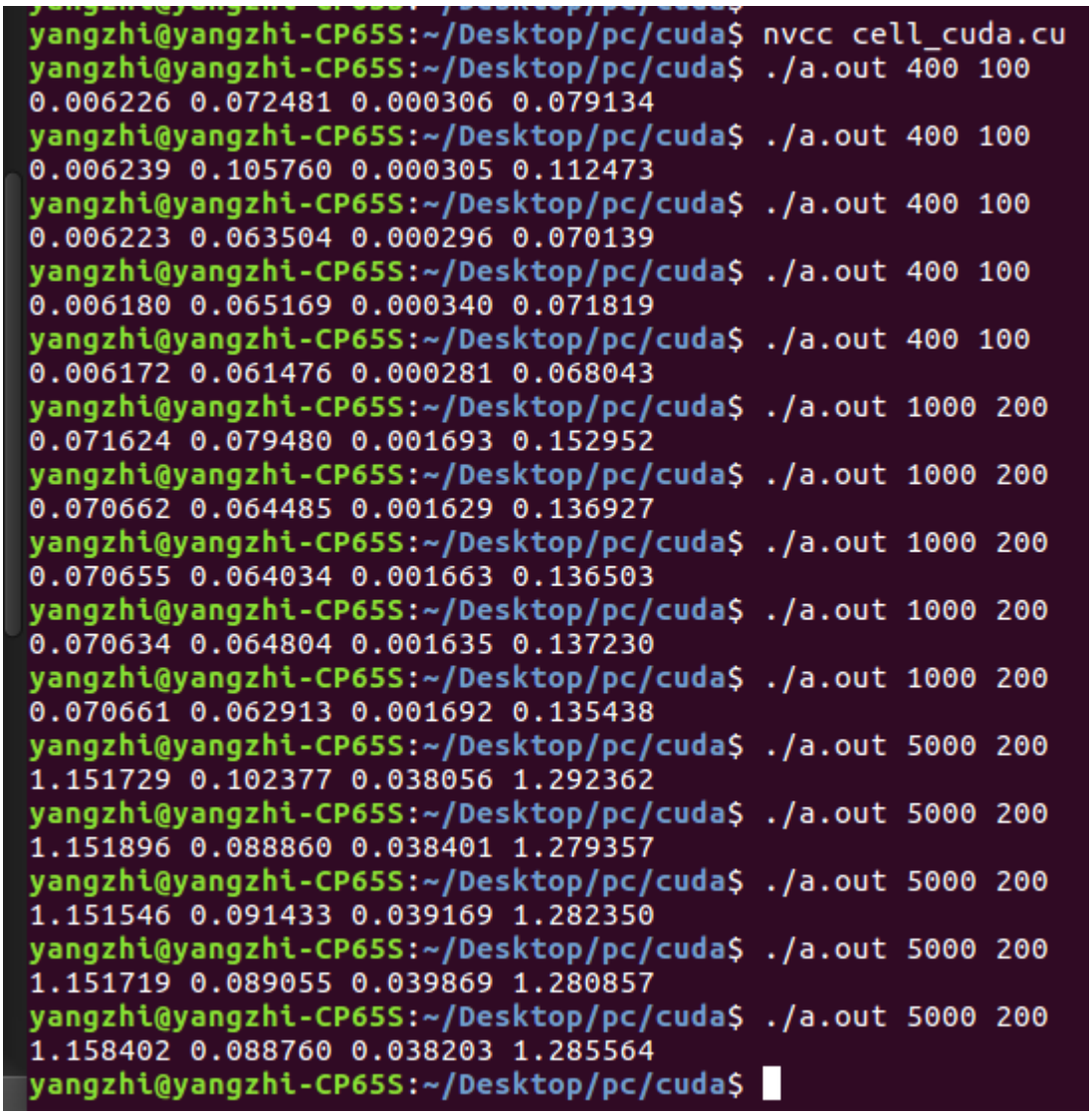
8 threads

CUDA

prime_number

```
yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ nvcc prime_cuda.cu
yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ for i in {1..5}
> do
> ./a.out 100000
> done
Max prime number 99991
input size 100000
Total time 0.000366 secs
prime number 9592
Max prime number 99991
input size 100000
Total time 0.000366 secs
prime number 9592
Max prime number 99991
input size 100000
Total time 0.000352 secs
prime number 9592
Max prime number 99991
input size 100000
Total time 0.000381 secs
prime number 9592
Max prime number 99991
input size 100000
Total time 0.000360 secs
prime number 9592
yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ for i in {1..5}; do ./a.out 500000; done
Max prime number 499979
input size 500000
Total time 0.002958 secs
prime number 41538
Max prime number 499979
input size 500000
Total time 0.002956 secs
prime number 41538
Max prime number 499979
input size 500000
Total time 0.002922 secs
prime number 41538
Max prime number 499979
input size 500000
Total time 0.002911 secs
prime number 41538
Max prime number 499979
input size 500000
Total time 0.002958 secs
prime number 41538
yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ for i in {1..5}; do ./a.out 1000000; done
Max prime number 999983
input size 1000000
Total time 0.008154 secs
prime number 78498
Max prime number 999983
input size 1000000
Total time 0.008126 secs
prime number 78498
Max prime number 999983
input size 1000000
Total time 0.008120 secs
prime number 78498
Max prime number 999983
input size 1000000
Total time 0.008107 secs
prime number 78498
Max prime number 999983
input size 1000000
Total time 0.008088 secs
prime number 78498
yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$
```

cell automata



matrix multiplication

yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ for i in {1..5}; do ./a.out 500; done

sequential matrix multiplication: 0.410527sec

parallel matrix multiplication without using Tiles: 0.067675sec

parallel matrix multiplication using Tiles: 0.00512sec

speedup without using Tiles: 6.06615

speedup using Tiles: 80.1811

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

sequential matrix multiplication: 0.38543sec

parallel matrix multiplication without using Tiles: 0.066485sec

parallel matrix multiplication using Tiles: 0.005193sec

speedup without using Tiles: 5.79725

speedup using Tiles: 74.2211

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

sequential matrix multiplication: 0.385542sec

parallel matrix multiplication without using Tiles: 0.066248sec

parallel matrix multiplication using Tiles: 0.005183sec

speedup without using Tiles: 5.81968

speedup using Tiles: 74.3859

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

sequential matrix multiplication: 0.38589sec

parallel matrix multiplication without using Tiles: 0.065829sec

parallel matrix multiplication using Tiles: 0.005184sec

speedup without using Tiles: 5.86201

speedup using Tiles: 74.4387

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

sequential matrix multiplication: 0.396794sec

parallel matrix multiplication without using Tiles: 0.06636sec

parallel matrix multiplication using Tiles: 0.005124sec

speedup without using Tiles: 5.97942

speedup using Tiles: 77.4383

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ for i in {1..5}; do ./a.out 1000; done

sequential matrix multiplication: 3.11095sec

parallel matrix multiplication without using Tiles: 0.105837sec

parallel matrix multiplication using Tiles: 0.036715sec

speedup without using Tiles: 29.3938

speedup using Tiles: 84.7325

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

sequential matrix multiplication: 3.04812sec

parallel matrix multiplication without using Tiles: 0.093727sec

parallel matrix multiplication using Tiles: 0.038376sec

speedup without using Tiles: 32.5212

speedup using Tiles: 79.4277

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

sequential matrix multiplication: 3.15009sec

parallel matrix multiplication without using Tiles: 0.094051sec

parallel matrix multiplication using Tiles: 0.036795sec

speedup without using Tiles: 33.4934

speedup using Tiles: 85.6118

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

sequential matrix multiplication: 3.16158sec

parallel matrix multiplication without using Tiles: 0.093972sec

parallel matrix multiplication using Tiles: 0.036678sec

speedup without using Tiles: 33.6439

speedup using Tiles: 86.1983

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

sequential matrix multiplication: 3.10774sec

parallel matrix multiplication without using Tiles: 0.093633sec

parallel matrix multiplication using Tiles: 0.036669sec

speedup without using Tiles: 33.1907

speedup using Tiles: 84.7512

check parallel result without using Tiles:

the sequential result and parallel result are equal

check parallel result using Tiles:

the sequential result and parallel result are equal

```
yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ for i in {1..5}; do ./a.out 5000; done
^C
yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ nvcc mult_cuda.cu
mult_cuda.cu(162): warning: variable "elapsedsequential" was declared but never referenced

mult_cuda.cu(162): warning: variable "optimizationP" was declared but never referenced

mult_cuda.cu(162): warning: variable "optimizationT" was declared but never referenced

mult_cuda.cu(162): warning: variable "elapsedsequential" was declared but never referenced

mult_cuda.cu(162): warning: variable "optimizationP" was declared but never referenced

mult_cuda.cu(162): warning: variable "optimizationT" was declared but never referenced

yangzhi@yangzhi-CP65S:~/Desktop/pc/cuda$ for i in {1..5}; do ./a.out 5000; done
parallel matrix multiplication without using Tiles: 3.339sec

parallel matrix multiplication using Tiles: 3.28648sec

parallel matrix multiplication without using Tiles: 3.33834sec

parallel matrix multiplication using Tiles: 3.27242sec

parallel matrix multiplication without using Tiles: 3.34364sec

parallel matrix multiplication using Tiles: 3.28701sec

parallel matrix multiplication without using Tiles: 3.33857sec
```

parallel matrix multiplication using Tiles: 3.27249sec

parallel matrix multiplication without using Tiles: 3.34113sec

parallel matrix multiplication using Tiles: 3.28357sec

LU decomposition