CS3210 lab2

1.

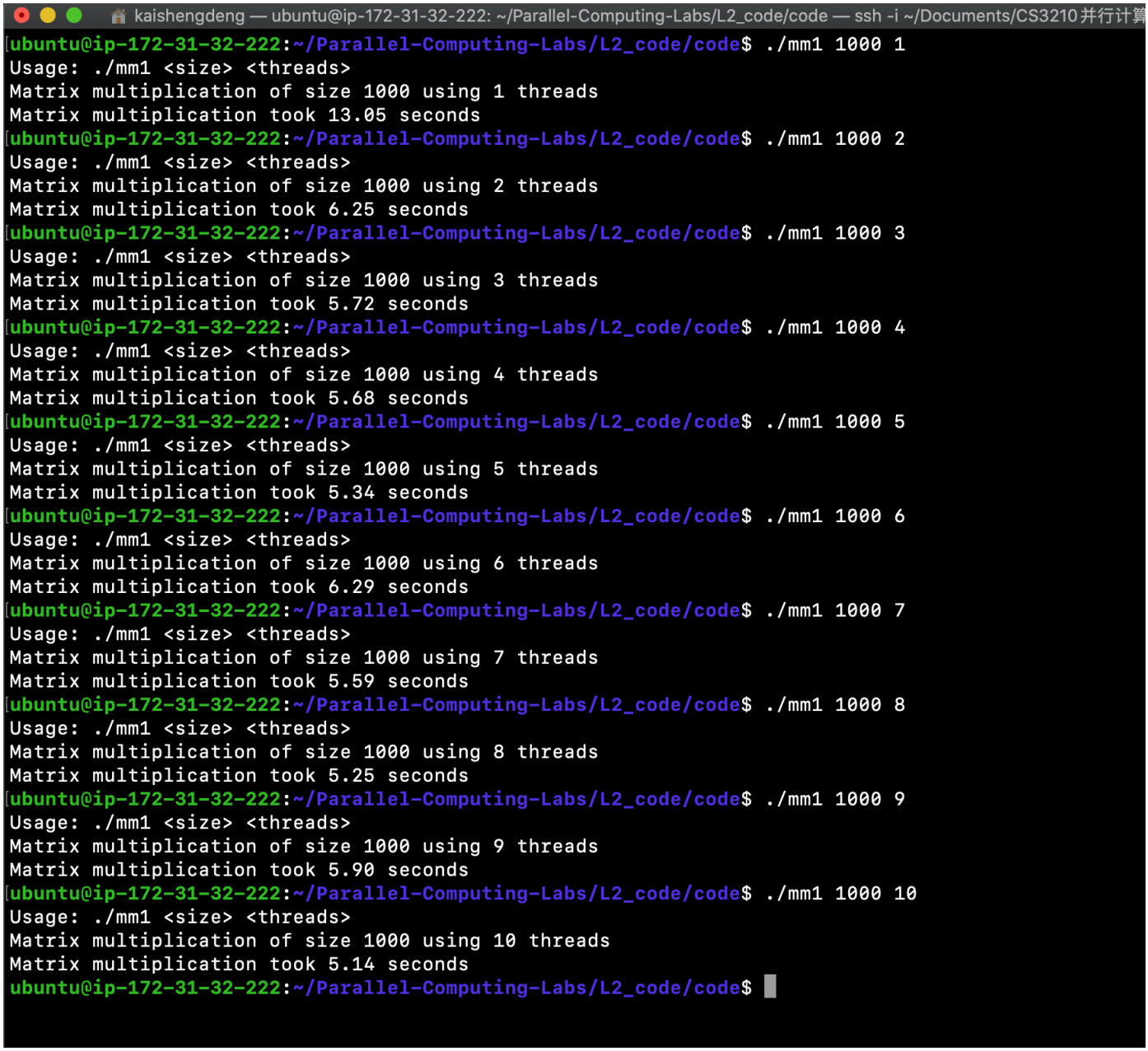
```
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$ ./mm0 10
Usage: ./mm0 <size>
Sequential matrix multiplication of size 10
Matrix multiplication took 0.000 seconds
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$ ./mm0 20
Usage: ./mm0 <size>
Sequential matrix multiplication of size 20
Matrix multiplication took 0.000 seconds
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$ ./mm0 40
Usage: ./mm0 <size>
Sequential matrix multiplication of size 40
Matrix multiplication took 0.000 seconds
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$ ./mm0 80
Usage: ./mm0 <size>
Sequential matrix multiplication of size 80
Matrix multiplication took 0.003 seconds
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$ ./mm0 160
Usage: ./mm0 <size>
Sequential matrix multiplication of size 160
Matrix multiplication took 0.021 seconds
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$ ./mm0 320
Usage: ./mm0 <size>
Sequential matrix multiplication of size 320
Matrix multiplication took 0.209 seconds
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$ ./mm0 640
Usage: ./mm0 <size>
Sequential matrix multiplication of size 640
Matrix multiplication took 1.982 seconds
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$ ./mm0 1280
Usage: ./mm0 <size>
Sequential matrix multiplication of size 1280
Matrix multiplication took 24.547 seconds
ubuntu@ip-172-31-32-222:~/Parallel-Computing-Labs/L2_code/code$
```

As size of matrix increases by 2 times, the execution time of sequential matrix multiplication increase by roughly 7-12 times

2.

CPU info of my machine:

There are 2 processors, each has 2 cores.



When the number of threads increase from 1 to 2, there is a dramatic decrease of the execution time.

However when the number of threads keeps increasing, there is very minor change of the execution time.

So the conclusion is,

The number of processors determines the ideal number of threads to use.

(**I wonder why it is not the number of cores rather than the number of processors...**)

3.

To give access to performance monitoring and observability operations, run this command,

```
sudo sysctl -w kernel.perf_event_paranoid=-1
```

Below is the output of command `perf stat -- ./mm1 1000 n` where n is from 1 to 6,

```
Usage: ./mm1 <size> <threads>
Matrix multiplication of size 1000 using 1 threads
Matrix multiplication took 13.14 seconds

 Performance counter stats for './mm1 1000 1':

          13178.09 msec task-clock                #    1.000 CPUs utilized

                50      context-switches          #    3.794 /sec

                 0      cpu-migrations            #    0.000 /sec

              3027      page-faults               #  229.699 /sec

   <not supported>      cycles

   <not supported>      instructions

   <not supported>      branches

   <not supported>      branch-misses


      13.179128553 seconds time elapsed

      13.170172000 seconds user
       0.007986000 seconds sys
```

```
Usage: ./mm1 <size> <threads>
Matrix multiplication of size 1000 using 2 threads
Matrix multiplication took 6.12 seconds

 Performance counter stats for './mm1 1000 2':

          12301.64 msec task-clock                #    1.991 CPUs utilized
```

```
                68      context-switches          #    5.528 /sec

                 1      cpu-migrations            #    0.081 /sec

              3030      page-faults               #  246.309 /sec

   <not supported>      cycles

   <not supported>      instructions

   <not supported>      branches

   <not supported>      branch-misses


       6.179111370 seconds time elapsed

      12.297753000 seconds user
       0.004001000 seconds sys
```

```
Usage: ./mm1 <size> <threads>
Matrix multiplication of size 1000 using 3 threads
Matrix multiplication took 5.52 seconds

 Performance counter stats for './mm1 1000 3':

          11045.49 msec task-clock                #    1.980 CPUs utilized

               706      context-switches          #   63.918 /sec

                44      cpu-migrations            #    3.984 /sec

              3033      page-faults               #  274.592 /sec

   <not supported>      cycles

   <not supported>      instructions

   <not supported>      branches

   <not supported>      branch-misses
```

```
       5.579005904 seconds time elapsed

      11.037777000 seconds user
       0.007944000 seconds sys
```

```
Usage: ./mm1 <size> <threads>
Matrix multiplication of size 1000 using 4 threads
Matrix multiplication took 5.24 seconds

 Performance counter stats for './mm1 1000 4':

          10536.91 msec task-clock                #    1.989 CPUs utilized

              1317      context-switches          #  124.989 /sec

                 8      cpu-migrations            #    0.759 /sec

              3034      page-faults               #  287.940 /sec

   <not supported>      cycles

   <not supported>      instructions

   <not supported>      branches

   <not supported>      branch-misses


       5.297357940 seconds time elapsed

      10.533156000 seconds user
       0.003998000 seconds sys
```

```
Usage: ./mm1 <size> <threads>
Matrix multiplication of size 1000 using 5 threads
Matrix multiplication took 4.98 seconds

 Performance counter stats for './mm1 1000 5':

           9977.46 msec task-clock                #    1.982 CPUs utilized
```

```
             1286      context-switches          #  128.891 /sec

               87      cpu-migrations            #    8.720 /sec

             3036      page-faults               #  304.286 /sec

   <not supported>      cycles

   <not supported>      instructions

   <not supported>      branches

   <not supported>      branch-misses


       5.034824556 seconds time elapsed

       9.969813000 seconds user
       0.007957000 seconds sys
```

```
Usage: ./mm1 <size> <threads>
Matrix multiplication of size 1000 using 6 threads
Matrix multiplication took 5.15 seconds

 Performance counter stats for './mm1 1000 6':

          10340.36 msec task-clock                #    1.987 CPUs utilized

             1379      context-switches          #  133.361 /sec

               12      cpu-migrations            #    1.161 /sec

             3037      page-faults               #  293.704 /sec

   <not supported>      cycles

   <not supported>      instructions

   <not supported>      branches

   <not supported>      branch-misses
```

```
       5.203710538 seconds time elapsed

      10.332684000 seconds user
       0.007979000 seconds sys
```

Terms,

- task-clock means CPU time used to finish the task
- time elapsed means how much time the user has to wait for the task to finish, usually,

> time elapsed = task-clock / number of CPUs utilized

- context-switch, **I'm not sure wheather it means switch between threads or between processes,** but I guess it's the former one.
- cpu-migrations,

> Migration is *when a thread, usually after a context switch, get scheduled on a different CPU than it was scheduled before*.

See https://stackoverflow.com/questions/45368742/linux-difference-between-migrations-and-switches

Obviously to promote efficiency, we need to reduce the number of cpu-migrations.

- page-fault means accessing a virtual page that has not yet been loaded into physical memory, this will cause an exception and the OS will then load the target page into memory. This is also what we want to avoid to get better efficiency.

Several conclusions can be drawn from the output,

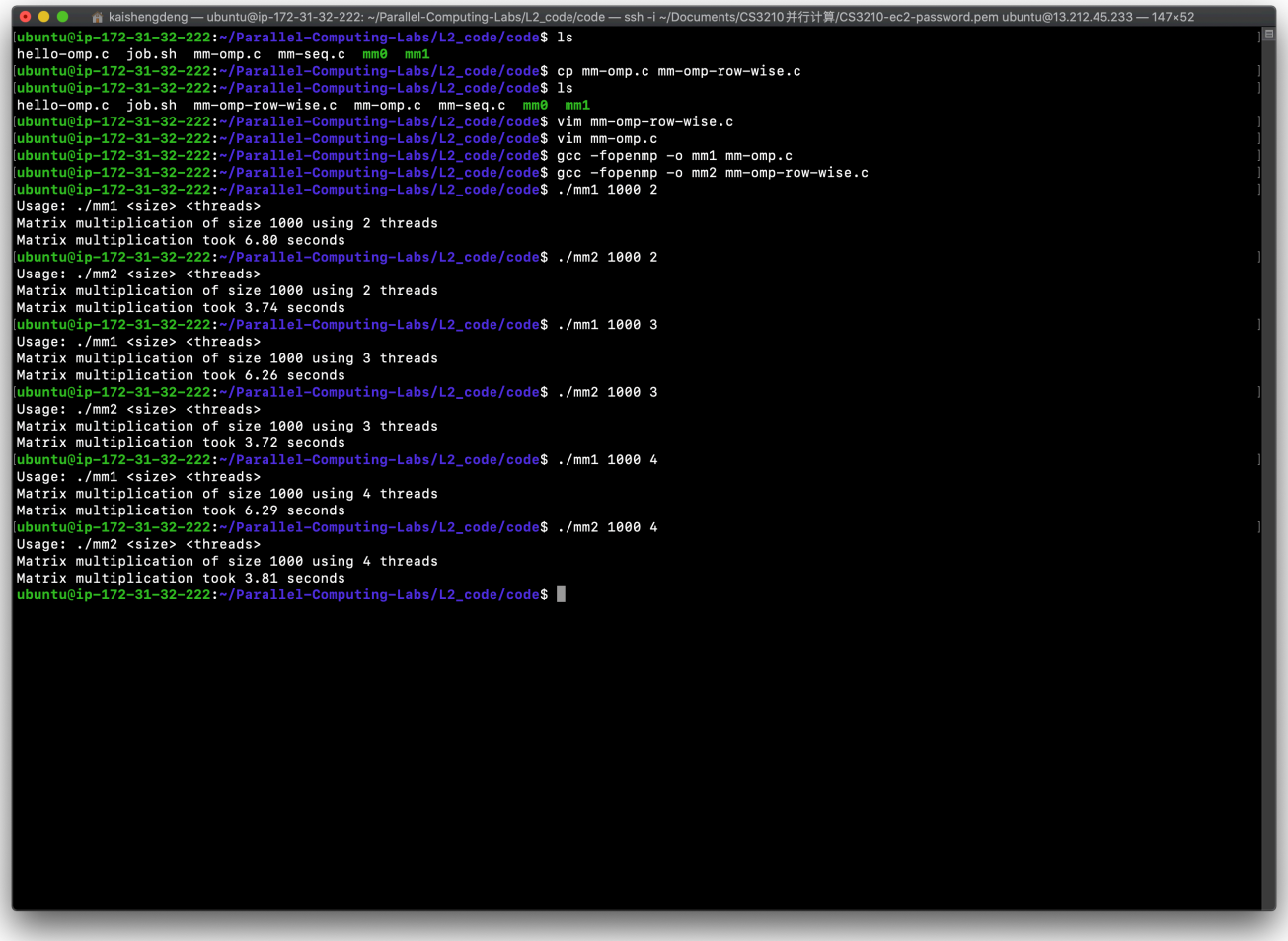
- When the number of threads is equal to or larger than the number of processers, the CPU utilization will reach maximum and the time elapsed will be reduced.
- When the number of threads is greater than the number of processors, the number of context switch will increase dramatically.
- When the number of threads is not an integer multiple of the number of processors, the number of CPU migrations will increase dramatically. I guess this is due to the fact that in this case the threads can not be distributed evenly to the processors, so the probability of migration will increase.
- The number of page faults is not influenced by the number of threads.

Putting together 2 and 3, I think the best choice of number of threads is exactly the number of processers, because in this case we can get the highest CPU utilization, lowest context-switch and lowest CPU-migration

7.

modify to access matrix `B` in row-wise

```
void mm_row_wise(matrix a, matrix b, matrix result)
{
    int i, j, k;
#pragma omp parallel for shared(a, b, result) private(i, j, k)
    for (i = 0; i < size; i++)
        for (k = 0; k < size; k++)
            for (j = 0; j < size; j++)
                result.element[i][j] += a.element[i][k] * b.element[k][j];
}
```



A increase of around 40%