Lab 2 report

6314061 – Quang Tuan Le

February 2023

1 Task 1: OpenMP hello world

Here is the results:

- When I ran the code without a clause num_threads(4), multiple threads are run the code and print "Hello World."

- This is the result when I ran the code with num_threads(4), then only 4 threads run the code and print "Hello World."

```
quangle — ssh qle010@onyx.cs.fiu.edu — 80×24
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
qle010@onyx:~/COP4520/Lab2/Task1 122% vi openmp_simple.c
qle010@onyx:~/COP4520/Lab2/Task1 123% g++ -fopenmp -o openmp_simple openmp_simpl
e.c
qle010@onyx:~/COP4520/Lab2/Task1 124% ./openmp_simple
Hello World
Hello World
Hello World
Hello World
qle010@onyx:~/COP4520/Lab2/Task1 125%
```

2 Task 2: Hello world extension

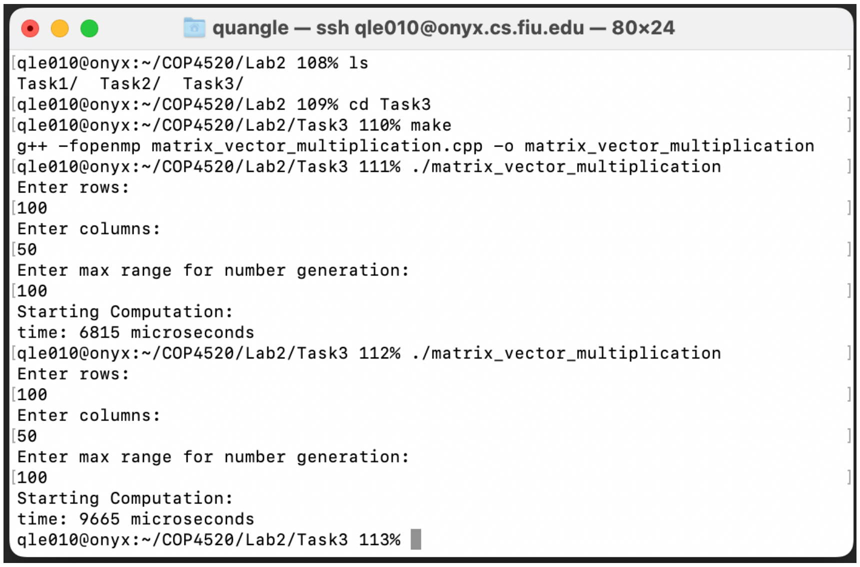
- This is the result when I ran the code parallel, and using the function omp_get_num_threads() to get number of threads and the function omp_get_thread_num() to get a thread's id.
- The result shows 112 threads, and every thread prints Hello World with its id number.

```
quangle — ssh qle010@onyx.cs.fiu.edu — 80×24
Hello World from thread = 5
Hello World from thread = 97
Hello World from thread = 76
Hello World from thread = 90
Hello World from thread = 37
Hello World from thread = 101
Hello World from thread = 71
Hello World from thread = 67
Hello World from thread = 40
Hello World from thread = 94
Hello World from thread = 45
Hello World from thread = 34
Hello World from thread = 52
Hello World from thread = 41
Hello World from thread = 104
Hello World from thread = 36
Hello World from thread = 0
Number of threads = 112
Hello World from thread = 100
Hello World from thread = 30
Hello World from thread = 9
Hello World from thread = 103
Hello World from thread = 1
qle010@onyx:~/COP4520/Lab2/Task2 135%
```

3 Task 3: Parallelization of matrix vector multiplication

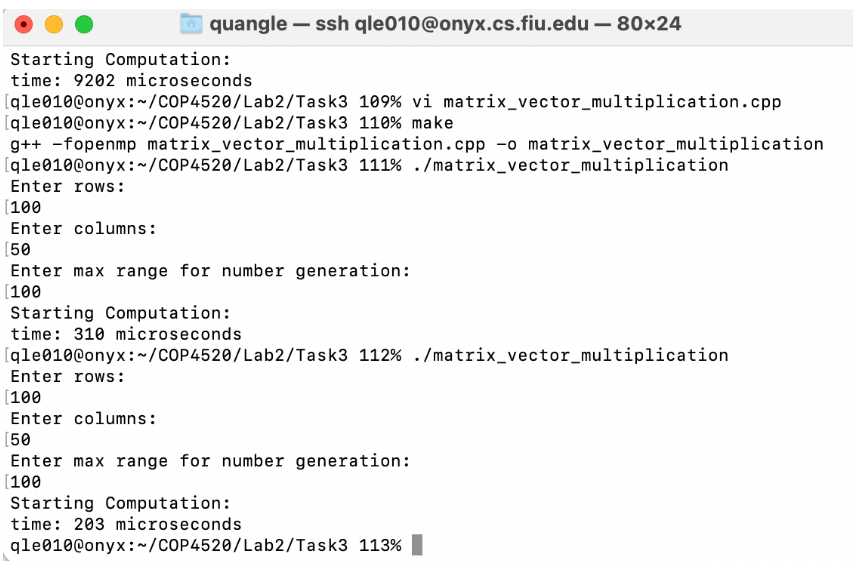
3.1

- This is the result of using OpenMP directives to make it run in parallel.



```
quangle — ssh qle010@onyx.cs.fiu.edu — 80×24
qle010@onyx:~/COP4520/Lab2 108% ls
Task1/  Task2/  Task3/
qle010@onyx:~/COP4520/Lab2 109% cd Task3
qle010@onyx:~/COP4520/Lab2/Task3 110% make
g++ -fopenmp matrix_vector_multiplication.cpp -o matrix_vector_multiplication
qle010@onyx:~/COP4520/Lab2/Task3 111% ./matrix_vector_multiplication
Enter rows:
100
Enter columns:
50
Enter max range for number generation:
100
Starting Computation:
time: 6815 microseconds
qle010@onyx:~/COP4520/Lab2/Task3 112% ./matrix_vector_multiplication
Enter rows:
100
Enter columns:
50
Enter max range for number generation:
100
Starting Computation:
time: 9665 microseconds
qle010@onyx:~/COP4520/Lab2/Task3 113%
```

- This is the result of using the above requirements with changing the number of threads to 4. The time is improved a lot.

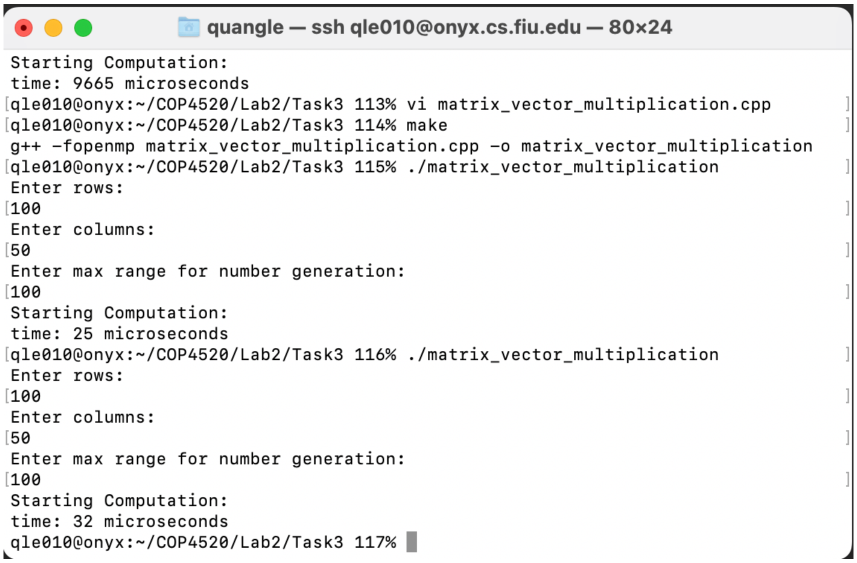


```
quangle — ssh qle010@onyx.cs.fiu.edu — 80×24
Starting Computation:
time: 9202 microseconds
qle010@onyx:~/COP4520/Lab2/Task3 109% vi matrix_vector_multiplication.cpp
qle010@onyx:~/COP4520/Lab2/Task3 110% make
g++ -fopenmp matrix_vector_multiplication.cpp -o matrix_vector_multiplication
qle010@onyx:~/COP4520/Lab2/Task3 111% ./matrix_vector_multiplication
Enter rows:
100
Enter columns:
50
Enter max range for number generation:
100
Starting Computation:
time: 310 microseconds
qle010@onyx:~/COP4520/Lab2/Task3 112% ./matrix_vector_multiplication
Enter rows:
100
Enter columns:
50
Enter max range for number generation:
100
Starting Computation:
time: 203 microseconds
qle010@onyx:~/COP4520/Lab2/Task3 113%
```

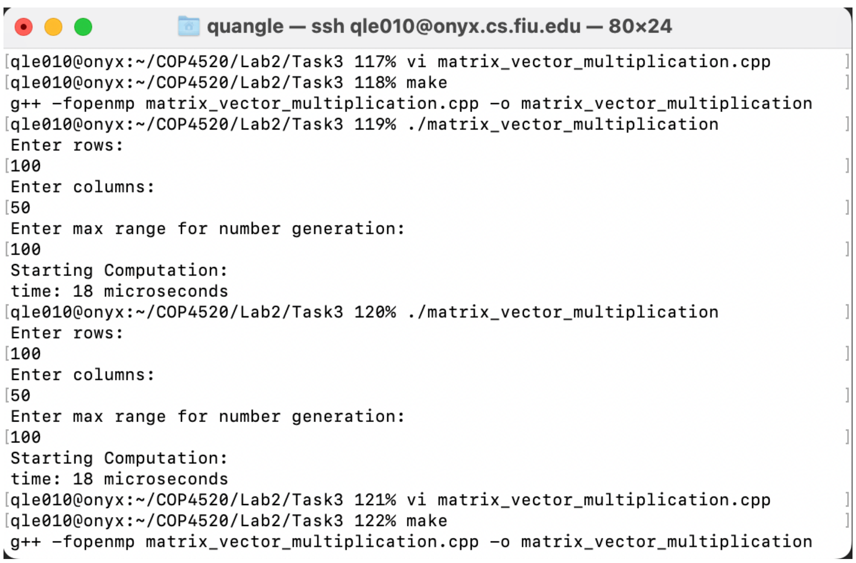
3.2 Fix the number of threads = 4

I tested all cases with the same inputs

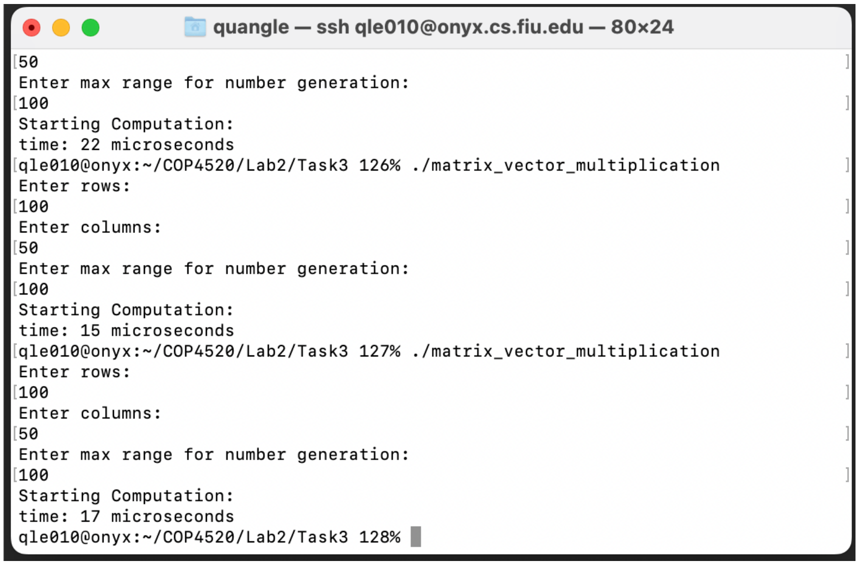
- This is the result of using static scheduling strategy.



- This is the result of using dynamic scheduling strategies.
  - Guided



  - Dynamic




```
50
Enter max range for number generation:
100
Starting Computation:
time: 22 microseconds
qle010@onyx:~/COP4520/Lab2/Task3 126% ./matrix_vector_multiplication
Enter rows:
100
Enter columns:
50
Enter max range for number generation:
100
Starting Computation:
time: 15 microseconds
qle010@onyx:~/COP4520/Lab2/Task3 127% ./matrix_vector_multiplication
Enter rows:
100
Enter columns:
50
Enter max range for number generation:
100
Starting Computation:
time: 17 microseconds
qle010@onyx:~/COP4520/Lab2/Task3 128%
```

This table shows my comparation among these parallel performance:

	#pragma omp parallel for	#pragma omp parallel for num_threads(4)	#pragma omp for schedule(static, 4)	#pragma omp for schedule(guided, 4)	#pragma omp for schedule(dynamic, 4)
Rows: 100 Columns: 50	4500 – 9200 microseconds	200 – 300 microseconds	15 – 20 microseconds	18 – 20 microseconds	18 – 19 microseconds
Rows: 1000 Columns: 500	10000 – 10500 microseconds	500 – 700 microseconds	1200 – 1300 microseconds	1230 – 1240 microseconds	1230 – 1270 microseconds

- So, from my observation in the above cases, the performance among static and dynamic scheduling strategies ("dynamic" and "guided") is quite equal because the workload is predictable, so they offer a balance between load balancing and minimizing overhead. However, it is totally different between (#pragma omp parallel for) loop and the same loop with the "num_threads(4)" clause because of several such as: overhead of dynamic thread management, when using the "parallel for" directive without the "num_threads" clause, the OpenMP runtime system dynamically manages the number of threads, which can incur additional overhead and unbalanced workload. In contrast, specifying the number of threads using the "num_threads" clause can help reduce this overhead, and ensure a balanced workload.