

For this project I rewrite the code of linear interpolation.

First I used `#pragma omp parallel` with shared memory `vector<double> yVal` is where to store the output information, and `NPTS` which is the size of `xVal` and `yVal` also the number to do iterations.

```
(base) Grants-MacBook-Pro:MP WanderD0D0$ pwd
/Users/WanderD0D0/Documents/CSULB/Distribute/MP
(base) Grants-MacBook-Pro:MP WanderD0D0$ clang++ -std=c++11 MP_linear_interpolation.cpp -o MP_linear_interpolation -Xpreprocessor -fopenmp -lomp
(base) Grants-MacBook-Pro:MP WanderD0D0$ clang++ -std=c++11 Non_MP.cpp -o Non_MP
(base) Grants-MacBook-Pro:MP WanderD0D0$ time ./MP_linear_interpolation

real    1m17.325s
user    4m37.887s
sys      0m0.696s
(base) Grants-MacBook-Pro:MP WanderD0D0$ time ./Non_MP

real    2m27.072s
user    2m25.690s
sys      0m0.449s
(base) Grants-MacBook-Pro:MP WanderD0D0$
```

We can find out by using multi-thread, the real processing time decrease because the work split to the CPUs. And the total user time increases due to the total processing time of CPUs increased, to explain clearly in about 1m17 the CPUs did the work that required 4m37 to process which means multi-threads can use CPU more efficiently.

Then I try the different `omp_set_num_threads` from 16, 8 to 4.

```
(base) Grants-MacBook-Pro:MP WanderD0D0$ clang++ -std=c++11 MP_linear_interpolation.cpp -o MP_linear_interpolation -Xpreprocessor -fopenmp -lomp
(base) Grants-MacBook-Pro:MP WanderD0D0$ time ./MP_linear_interpolation

real    1m15.589s
user    4m43.975s
sys      0m0.607s
(base) Grants-MacBook-Pro:MP WanderD0D0$ clang++ -std=c++11 MP_linear_interpolation.cpp -o MP_linear_interpolation -Xpreprocessor -fopenmp -lomp
(base) Grants-MacBook-Pro:MP WanderD0D0$ time ./MP_linear_interpolation

real    1m17.760s
user    4m29.608s
sys      0m0.675s
(base) Grants-MacBook-Pro:MP WanderD0D0$ clang++ -std=c++11 MP_linear_interpolation.cpp -o MP_linear_interpolation -Xpreprocessor -fopenmp -lomp
(base) Grants-MacBook-Pro:MP WanderD0D0$ time ./MP_linear_interpolation

real    1m25.810s
user    3m31.197s
sys      0m0.348s
(base) Grants-MacBook-Pro:MP WanderD0D0$
```

By decreasing the threads, the real processing time increased because each thread need to serve more work.

One step ahead, I tried to let my program do more multithreading process. Now I use multi-thread to initialize my vector variables.

```
sys      0m0.274s
(base) Grants-MacBook-Pro:MP WanderD000$ clang++ -std=c++11 MP_linear_interpolation.cpp -o MP_linear_interpolation -Xpreprocessor -fopenmp -lomp
(base) Grants-MacBook-Pro:MP WanderD000$ time ./MP_linear_interpolation

real    1m18.438s
user    4m36.946s
sys      0m1.164s
```

But unfortunately, the performance does not speed up. For the reason I concluded is because the vector/array initialization is more about memory accessing/storing than the computation, so even using multi-thread to accelerating the computation part doesn't affect the real processing time much.