Our purpose of this assignment is to make code run faster. Here is some methods that I considered and discussed with my classmates. Some of them are successful to achieve the goal, but some of them are fail to make code run faster. But I still want to add it to my write-up because they gave me more ideas and make me better in this field.

The process of my project is:

1. Configure the runtime environment of c++ in the laptop and add tbb
2. Use the parallel_for function in tbb to optimize the Gaussian elimination algorithm and study the optimization efficiency.
3. Use the parallel_reduce in tbb to optimize the function and study the optimization efficiency.
4. Use init to show performance for 2048 and 4096 matrices, and make it into graphs.
5. Inclusion.

1.In the first step, I downloaded the Clion and added the following code to the makelist.txt:

```
include_directories(/usr/local/Cellar/tbb)
link_directories(/usr/local/Cellar/tbb)
add_executable(untitled main.cpp)
target_link_libraries(untitled tbb)
```

So we read the tbb package and just declare the header file:

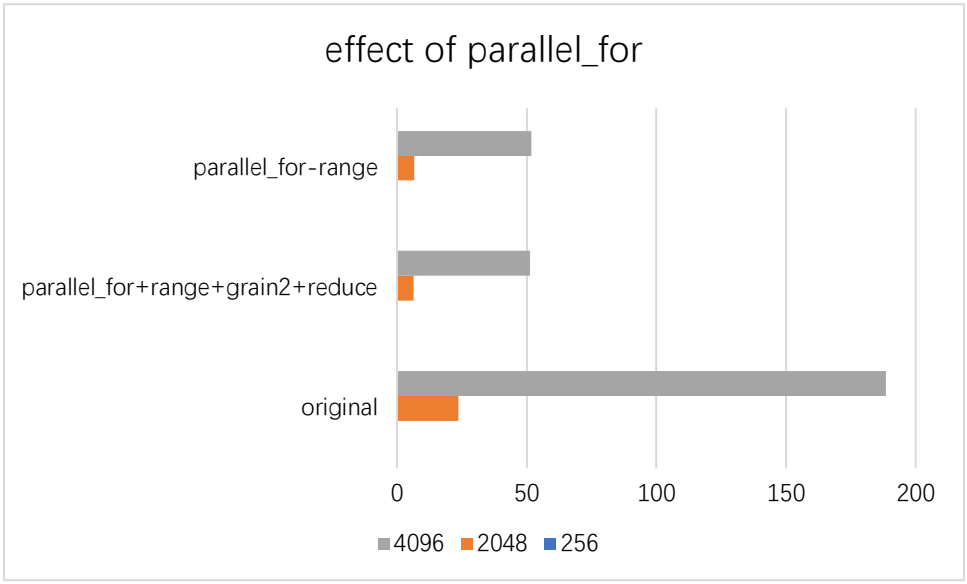
```
#include <tbb/tbb.h>
```

2.we use the parallel_for function in tbb to optimize the guass algorithm. Parallel_for is a function commonly used in the tbb parallel computing library. Its function is to traverse each space in parallel. I add it to the second layer of the Gaussian elimination algorithm (ie, for a given matrix row i, eliminate the ith coefficient of each row after i). The reason for adding the second layer loop is because the operations performed in this loop conform to the principle that parallel computing can be performed. Each loop is independent of each other, that is, for i+1 to A.getsize() lines, each line perform operations with ith line. Here we have two ways to add a parallel_for operation to the second layer of the loop, one is to add the block_range part, you can set the granular size, which means a "suitable size" block, this block will be processed in a loop, if The array is larger than this grain, and parallel_for will split it into separate blocks and then schedule them separately (possibly by multiple threads).

The second method is I capture the address of k directly in the lambda expression.

```
parallel_for(i+1,int(A_getSize(),[&](int k
```

Both methods are fine, but obviously the first method works better according to the following chart.

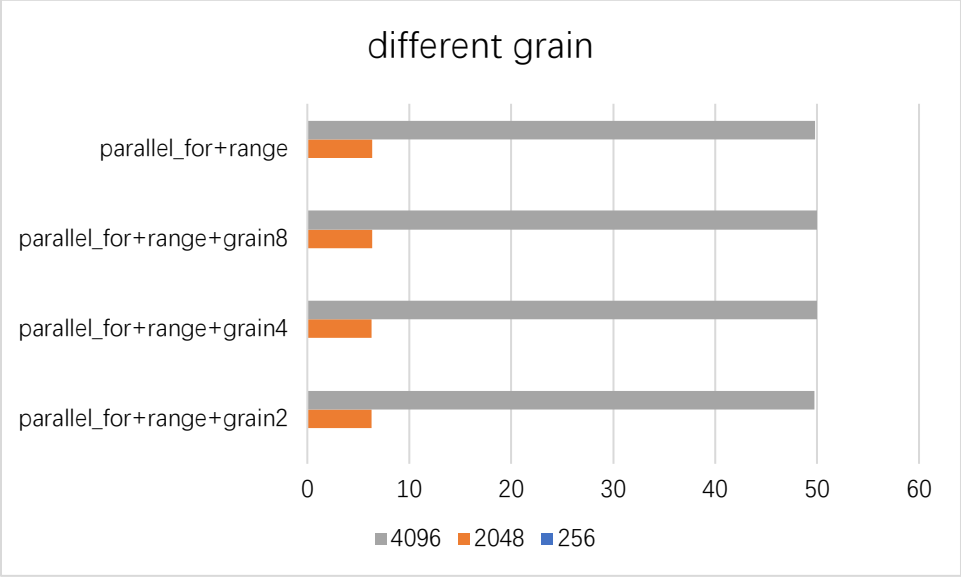


Parallel_for-range: the second method.

Parallel_for+range: the first method.

Original: original code.

So we choose the first method. Then we compare the method of different grain.



There is no big different in these. So we choose the grain 2.

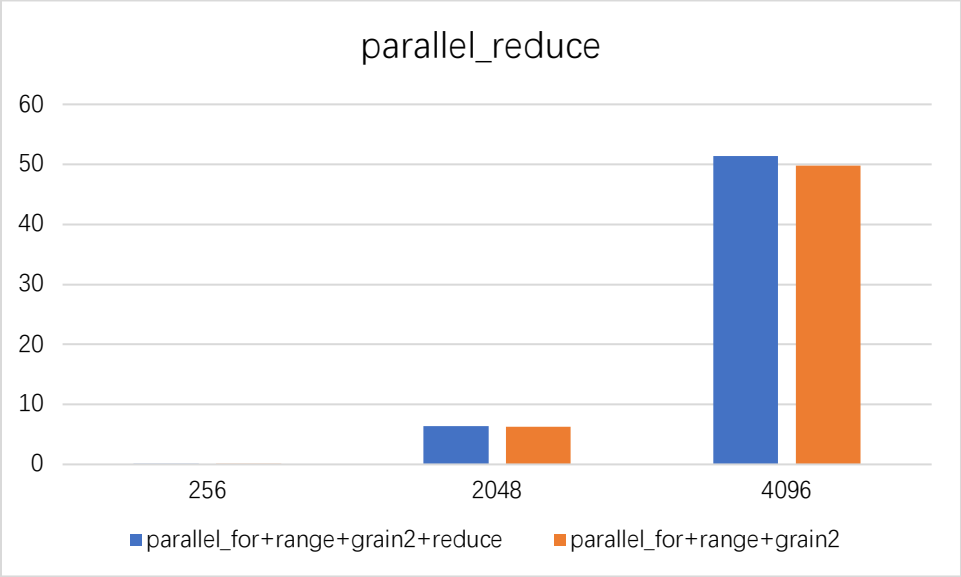
Note：we need to set the ratio to make verification succeeded.

```
ratio >1.000001 || ratio <0.99999
```

3.parallel_reduce :It automatically groups the intervals, accumulates each group, gets one result for each group, and finally aggregates the results of each group.

In the algorithm, the number of rows containing the maximum value of the i-column coefficient is found in each i-cycle. The loop that finds the maximum value can be parallelized. So I added the parallel_reduce function here to try. But when range = 256, The results are not ideal. I guess because reduce itself needs to create a lot of processes, this

process takes more time to save the matrix in this way.So I try to do it when range=2048 and range=4096. But the matrix is not big enough, so the parallel_reduce is not work effectly.



parallel_for+range+grain2+reduce: use grain=2 and use parallel_reduce

parallel_for+range+grain2: use grain=2 but don’ t use parallel_reduce

Finally, I don’ t use this method.

4.result of init

parallel_for+range	2048	4096
2	12.524	98.84
4	6.655	52.66
8	6.288	50.518

16	6.441	50.657

parallel_for+range+grain2	2048	4096
2	12.508	98.72
4	6.686	52.688
8	6.3	49.76
16	6.432	50.507

parallel_for+range+grain4	2048	4096
2	12.568	98.935
4	6.671	52.454
8	6.338	49.778
16	6.86	50.401

	256	2048	4096
original	0.046	23.689	188.441

5.inclusion

In this code, I decide that I use the parallel_for +range +grain2,and the init = 8, and do not use parallel_reduce.