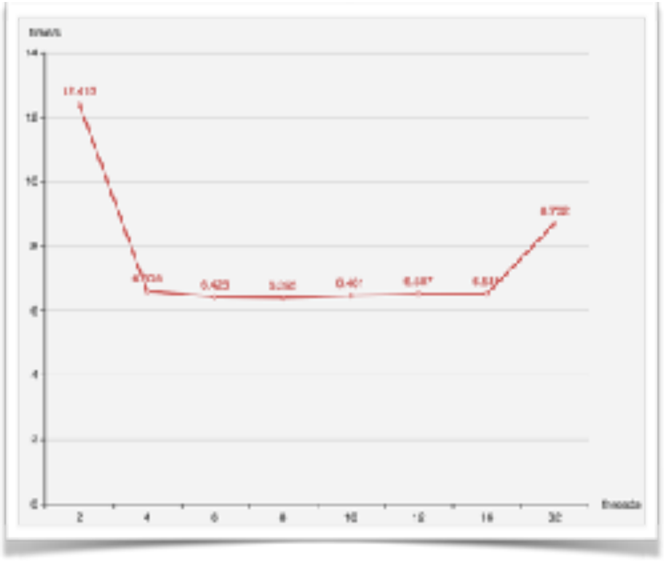
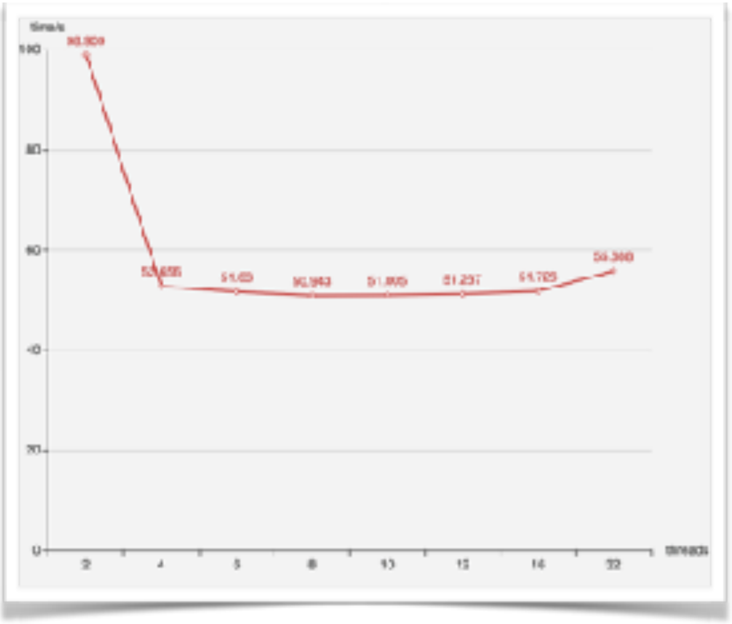
I used tbb::parallel_for and tbb::parallel_reduce to optimize the code. I also used tbb::task_scheduler_init to explicitly init threads' number.

1. parallel_for is used to eliminate the ith row from all subsequent rows. I package the original for-for loop with it and set grainsize.
2. parallel_reduce is used to find the largest value in this column. The value of this grainsize is the same as previous one.
3. I set tbb::task_scheduler_init before invoking function gauss() in main().

Firstly I added a command-line parameter about grainsize and let compiler automatically determine the threads' number and change grainsize. For 2048 matrix, I found the most efficient grainsize number is 2. And this value equals to 4 or 8 when I tested 4096 matrix.(I choose 4 to continue subsequent tests.)
Then I tested the impact of threads' number on efficiency. The most efficient number of threads of both 2048 and 4096 matrices is 8.
Below are the charts which show threads-time based on sunlab tests.



2048 matrix



4096 matrix

(Please notice that the x-axis is not continuous)

All of the data are the average of 5 trials. Variances of 2048 matrix are below 0.0001 except when threads equals to 32, whose variance is below 0.001. As for 4096 matrix, variances are below 0.5.
We can tell from charts that the efficiency is really similar from 4 to 16. time-consuming is high at 2, because only two threads don't efficiently use the CPU resources. I think the CPU on sunlab can only manage four threads. If there are more threads, they have to wait until CPU has chance to wake them up. As threads' number increase, they take more time on threads switch, so the program takes more time.