Tian Tang (A20284409)

For the experiments, I made use both of the Pthreads and OpenMP to write parallel program of Gaussian elimination. I tried to parallelize the inner loop (every column) firstly, but it was very slow for both tools. I guess creating and joining threads in every row leads to too much overhead. And I also noticed that the two dimension array is stored by rows in C. So accessing the data in the close columns in same row continually would save a lot of time. Then I parallelized the outer loop and it worked. For the convenient, I used a struct pointer as the arguments when creating threads in Pthreads. And after all the programming work done, I did different experiments using different numbers of threads and different sizes of matrices. The bigger size of the matrix, the better performance was observed compared with serial codes. And more threads gave me better performance especially with the big numbers of matrix size. But for OpenMP the speedup is relatively low not matter how many threads there are. I guess its overhead is relatively high and it would show advantages when scaling up to bigger size. The result of experiments is displayed below.

# Matrix Size	# Threads	Method	Elapsed Time (ms)	Speedup
100	1		2.716	
100	2	OpenMP	1.776	1.53
100	4	OpenMP	1.758	1.54
100	8	OpenMP	1.357	2.00
100	2	Pthreads	8.651	0.34
100	4	Pthreads	11.484	0.23
100	8	Pthreads	20.503	0.13
500	1		152.11	
500	2	OpenMP	157.139	0.97
500	4	OpenMP	158.198	0.96
500	8	OpenMP	152	1.00
500	2	Pthreads	147.253	1.03
500	4	Pthreads	123.894	1.23
500	8	Pthreads	161.393	0.94
1000	1		1248.03	
1000	2	OpenMP	1213.35	1.03
1000	4	OpenMP	1221.37	1.02
1000	8	OpenMP	1217.02	1.03
1000	2	Pthreads	922.061	1.35
1000	4	Pthreads	487.36	2.56
1000	8	Pthreads	659.407	1.89
2000	1		9927.05	
2000	2	OpenMP	9639.41	1.03
2000	4	OpenMP	9861.45	1.01
2000	8	OpenMP	8968.47	1.11
2000	2	Pthreads	5751.56	1.73
2000	4	Pthreads	3342.86	2.97
2000	8	Pthreads	3584.27	2.77