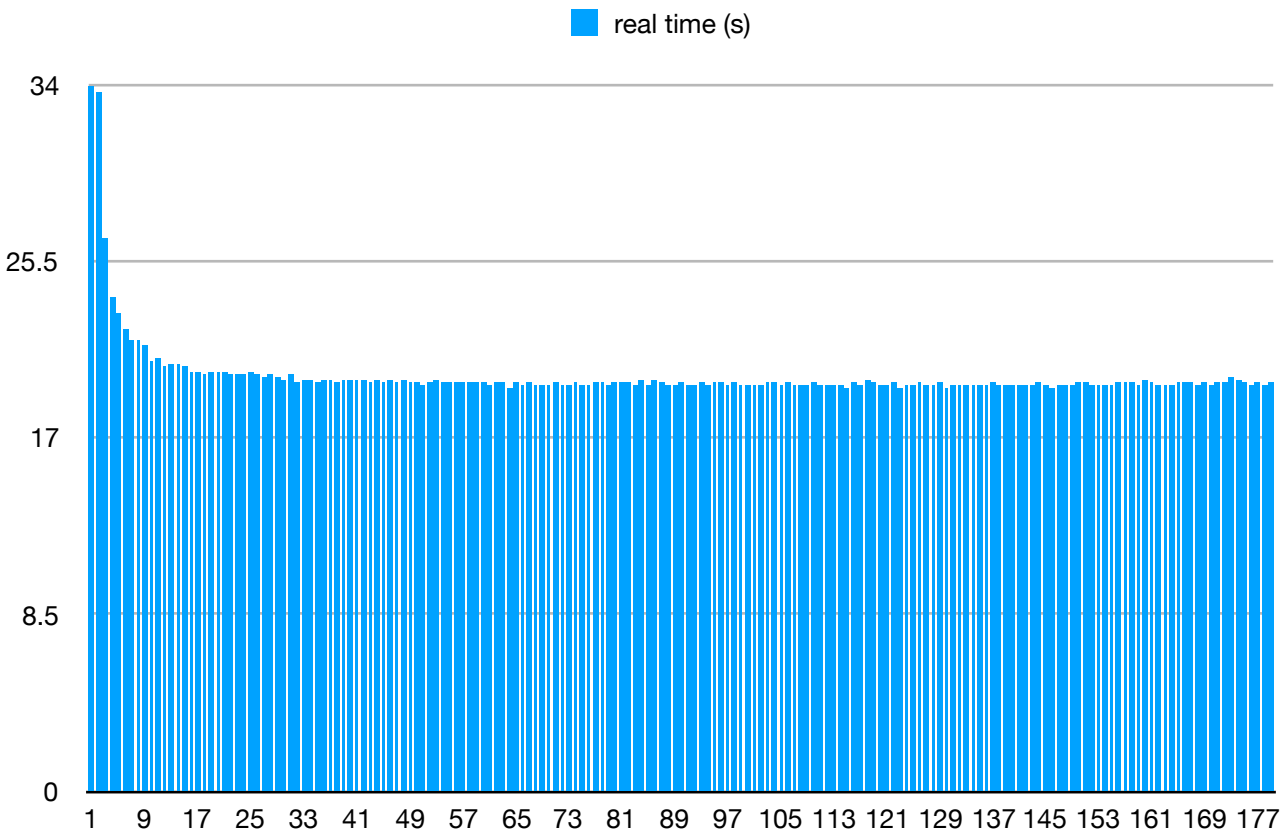
SYSTEM PROGRAMMING PROGRAMMING ASSIGNMENT #4 REPORT

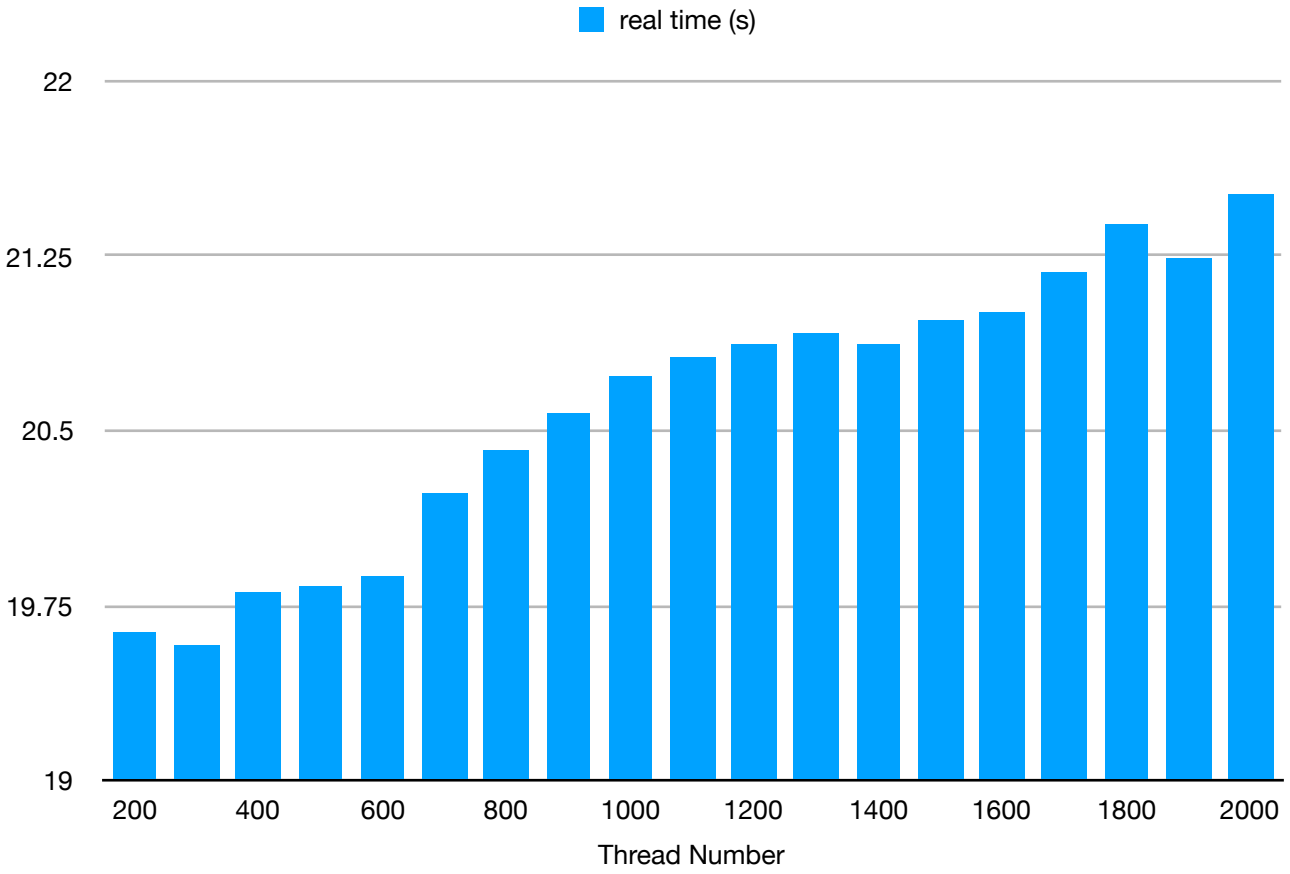
B07902143 陳正康

Time

When thread number is small, the efficiency gain with multi-threading is obvious. When thread number is above around the core number of the machine (the workstation has 24 cores), the time decreasing rate as thread number increases is quite small, but still > 0. So we can conclude that approximately when thread number < 200, more threads brings more efficiency.

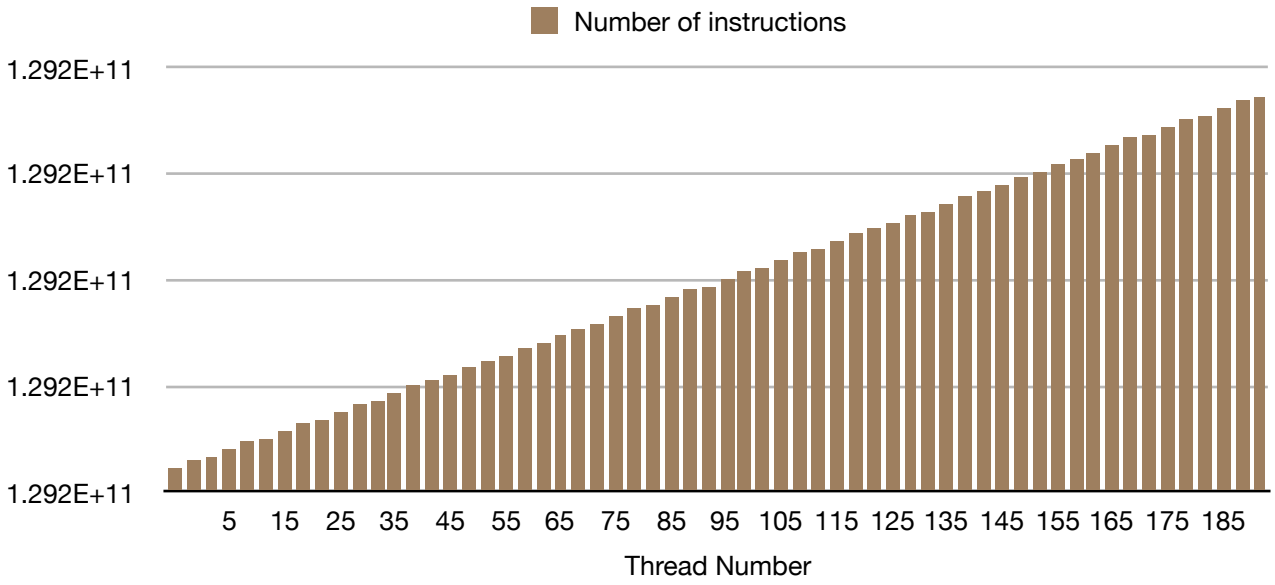


The next chart indicates that this tendency continues until around 300 threads. Then since it costs many time to create so many threads, the efficiency get worse after thread number is above 300.

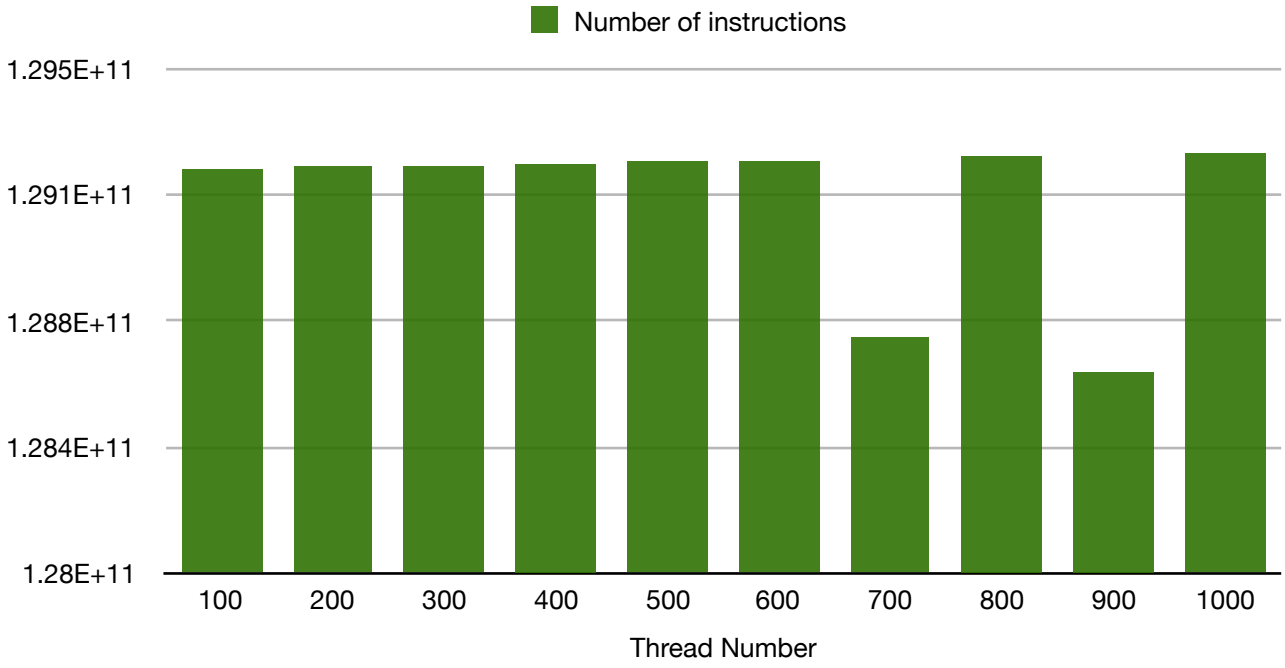


Number of Instructions

This number grows linearly as thread number increases.



However, it comes to a weird drop at 700 and 900 threads. My guess is that it's due to race condition, where some thread failed to be created when workload is high.



Optimization Approaches

1. The way to stored large matrices in memory:
   Multiplication of matrices is the main bottleneck of efficiency. We've observed that sometimes it's preferable to store matrix **transposed**, but use it as not transposed. It's more cache-friendly and easier for compiler to optimize. For example, in step (4) we stored the y_hat matrix (60000*10) as (10*60000), since it's easier to move through columns than rows.
2. Tools that helps analyzing time consumption in each step of the program:
   https://hackmd.io/@twzjwang/BJkx6f6Fe?type=view
   https://www.ibm.com/developerworks/cn/linux/l-gnuprof.html
   https://www.itread01.com/p/168437.html