Computational Astrophysics HW 6

R08244002 蔡欣蓉

- ## SOR Relaxation Method with OpenMP

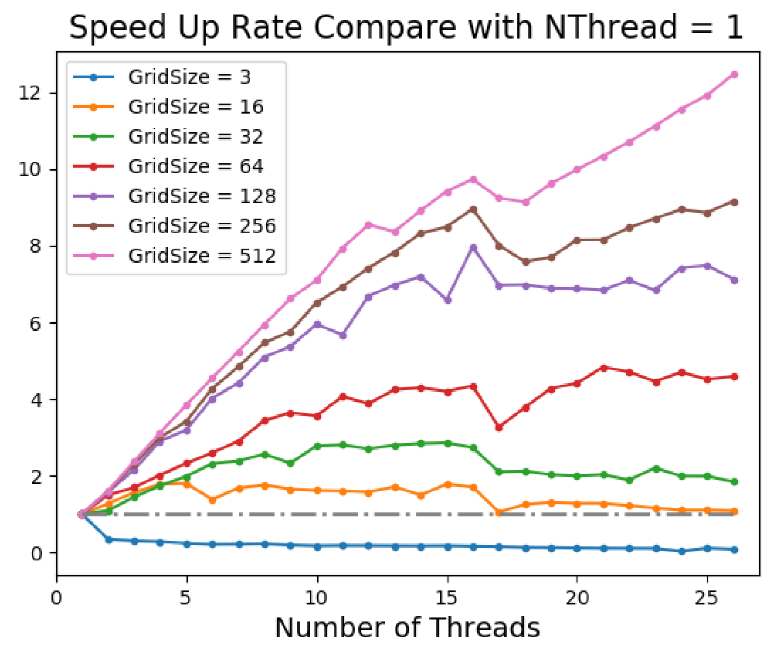
  Compile and run *SOR_parallel.cpp*, then loop through square grid size $N = \{16,32,64,128,256,512\}$ and number of threads $\text{NThread} = 1\sim26$ with bash script. Finally, plot the result with *result.py.*

  The CPU I'm using is from the lab, it is really powerful!!! It has 16 cores with a total of 32 CPUs, so I guess I didn't overload the CPU with too much threads than it can carry.

  After running in serial (without OpenMP), and got the elapsed time through *clock( )* function in *<time.h>* , I get these results. And compare to the OpenMP with $\text{NThread} = 1$, whom got elapsed time through *omp_get_wtime( )*, it is slightly different. And most important of all, *clock( )* function is not that precise. So in the following discussion, I'll use OpenMP with $\text{NThread} = 1$ as the datum for all the other OpenMP runs.

Grid Size / Time (sec)	3	16	32	64	128	256	512
Serial	0.0	0.0	0.03	0.16	2.4	39.9	650.0
OpenMP NThread = 1	0.0006	0.001	0.002	0.1	2.5	40.8	663.0

Defined



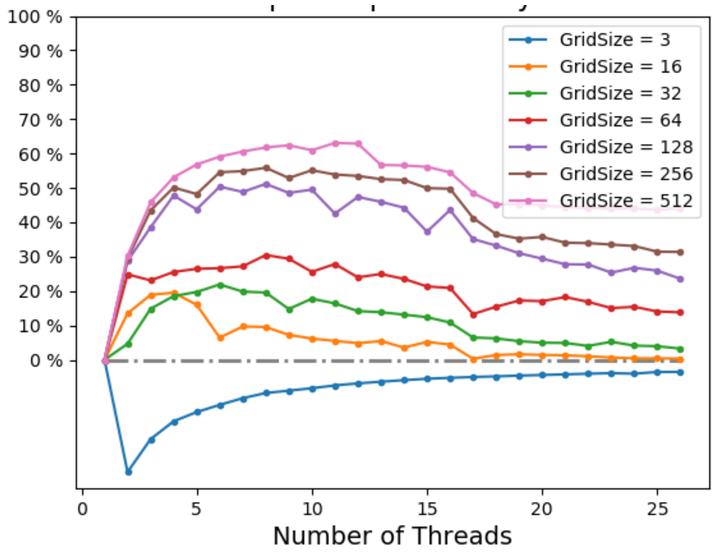
$$\text{Speed Up} = \frac{TimeUsed_{NThread=1}}{TimeUsed_{NThread}}$$

1. The overall performance is heavily determined by the workload, which is grid size N here. Since it consumes some time at handling the OpenMP constructs, and the time spent will grow with number of threads used, the speed up rate won't just grow whatever you wish when you increase the number of threads.
2. An extreme example in grid size $N = 3$, the workload is too light, that even launching a OpenMP cost even more time than run CPU serial.
   Grid size $N = 16, 32$, we can clearly see that there is an optimal number of threads used. It is a balance between time saved in distributing workload to threads, and time used on launching OpenMP threads.
   For grid size $N = 64,128,256,512$, the slope depends on the workload, since it hasn't reach the optimal number of threads used.
3. There is a drop from $NThread = 16$ to $NThread = 17$, probably because that it is 16 cores, the last thread must squeeze inside with another thread.

Define

$$\text{Improved} = \frac{Speed\ Up\ Rate - 1}{NThread}$$

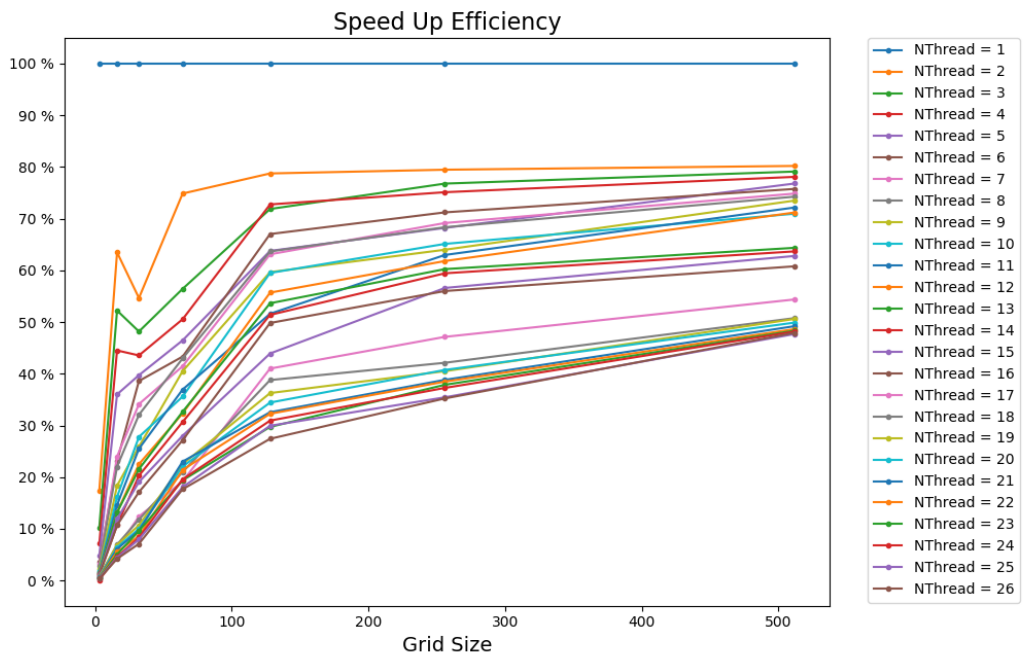
To see how much partition of Speed Up it increases of each thread adding.



1. There is a bump in each grid size, is just a fact that balance between distributed workload and launching OpenMP threads.

Define

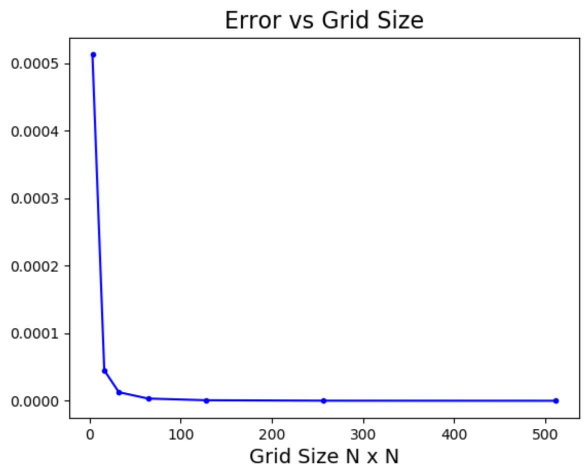
$$\text{Efficiency} = \frac{Speed\ \ Up}{NThread}$$



1. The efficiency grows with grid size, but saturates if each thread reaches its maximum workload that it can handle.

Define

$$\text{Error} = \frac{|Analytic - SOR|}{Grid\ \ Size\ \ N \times N}$$



1. Error will saturate as grid size get large, so increasing grid size to get a better result is a good ideal at initial when required errors are not that small.
2. Grid size $N = 512$ has error about 5.6×10^{-8}.