HW5

Boyao Zhu, Jiaxin Yang

1. b)

n	p	time of p2p	speedup of p2p	efficiency of p2p	time of collective	speedup of collective	efficiency of collective
1000	1	2.9e-05	1	1	2.8e-05	1	1
1000	2	2.3e-05	1.26086956521739	0.630434782608696	1.6e-05	1.75	0.875
1000	5	0.004293	0.00675518285581179	0.00135103657116236	1.2e-05	2.33333333333333	0.466666666666667
1000	10	1.8e-05	1.61111111111111	0.161111111111111	1.1e-05	2.54545454545455	0.254545454545455
1000	20	0.010692	0.00271230826786382	0.000135615413393191	0.016411	0.00170617268905003	8.53086344525014e-05
1000	100	0.028907	0.00100321721382364	1.00321721382364e-05	0.016427	0.00170451086625677	1.70451086625677e-05
1000	250	0.125222	0.000231588698471515	9.26354793886058e-07	0.021053	0.00132997672540731	5.31990690162922e-06
1e+05	1	0.002692	1	1	0.002695	1	1
1e+05	2	0.001359	1.98086828550405	0.990434142752024	0.001345	2.00371747211896	1.00185873605948
1e+05	5	0.005223	0.515412598123684	0.103082519624737	0.000583	4.62264150943396	0.924528301886792
1e+05	10	0.007583	0.355004615587498	0.0355004615587498	0.004042	0.666749134092034	0.0666749134092034
1e+05	20	0.014449	0.186310471312894	0.00931552356564468	0.006197	0.434887848959174	0.0217443924479587
1e+05	100	0.112112	0.0240117025831312	0.000240117025831312	0.025561	0.105434059700325	0.00105434059700325
1e+05	250	0.123364	0.0218216011153983	8.72864044615933e-05	0.02957	0.0911396685830233	0.000364558674332093
1e+07	1	0.269535	1	1	0.269284	1	1
1e+07	2	0.134653	2.00170066764201	1.00085033382101	0.134529	2.00167993518126	1.00083996759063
1e+07	5	0.05378	5.01180736333209	1.00236147266642	0.053943	4.99201008471906	0.998402016943811
1e+07	10	0.026997	9.98388709856651	0.998388709856651	0.026957	9.98939051081352	0.998939051081352
1e+07	20	0.025135	10.7234931370599	0.536174656852994	0.021628	12.4507120399482	0.622535601997411
1e+07	100	0.035039	7.69242843688462	0.0769242843688462	0.018083	14.8915556047116	0.148915556047116
1e+07	250	0.048019	5.61309065161707	0.0224523626064683	0.013464	20.0002970885324	0.0800011883541295

When the number of processors is small, collective and point-to-point show similar performance. As the number of processors goes up, collective version become faster than point-to-point version.

From the data above, when n is large, speed up of both point-to-point and collective increase stably. However, when n is small, the speed up is not stable and the performance is not good as expected, which may because the code is not so effective and need to be improved.

Besides, according to the formula in c), when n is large, the time is decided by O(n/p). So the time decrease by p. Also, when n is small, O(log(p)) decide the time and the time increase by p.

c)

```
# input data
n <- c(rep(1000,7), rep(100000, 7), rep(10000000, 7))
p <- rep(c(1,2,5,10,20,100,250),3)
p2p <- c(0.000029,0.000023,0.004293,0.000018,0.010692,0.028907,0.125222,
         0.002692,0.001359,0.005223,0.007583,0.014449,0.112112,0.123364,
         0.269535,0.134653,0.053780,0.026997,0.025135,0.035039,0.048019)
clct <- c(0.000028,0.000016,0.000012,0.000011,0.016411,0.016427,0.021053,
          0.002695,0.001345,0.000583,0.004042,0.006197,0.025561,0.029570,
          0.269284,0.134529,0.053943,0.026957,0.021628,0.018083,0.013464)
```

```
x1 <- n/p
x2 <- log2(p)

mydata <- data.frame(cbind(x1,x2,p2p,clct))
# fit the linear model
fit1 <- lm(p2p~x1+x2-1, mydata)
summary(fit1)
fit2 <- lm(clct~x1+x2-1, mydata)
summary(fit2)
```

For p2p, we have the formula

$$T_{p2p}(n,p) = 2.595e^{-8} \times \frac{n}{p} + 8.641e^{-3} \times \log_2(p)$$

For collective, we have the formula

$$T_{collective}(n,p) = 2.667e^{-8} \times \frac{n}{p} + 2.395e^{-3} \times \log_2(p)$$

We can observe that the collective version has smaller scale of log(p). Because MPI_Reduce is an all-to-one command, which is effective than point-to-point and can reduce the time of communicate. Besides, they have similar scale of n/p, which may due to the same computing processes.

2.

(a) parallelized code

(b) By using block partition, scalability can be nondeterministic. The speedup can be sub-linear, linear or super-linear depends on particular key. Actually, it's a matter of luck. I will analyze these three scenarios from fundamental origin and convince you with specific data.

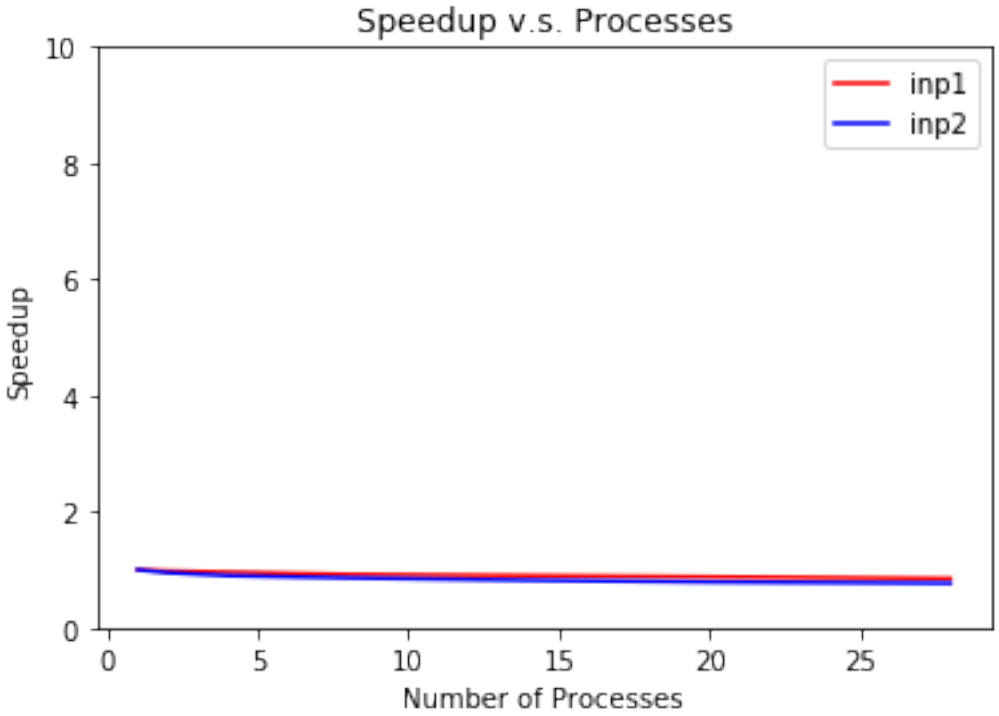
There are totally 2^{32} possible keys, we distribute them to p equal parts and give each processor one part.

Then processor i (i from 0 to p-1) will take care of the scope from $2^{32} \times \frac{i}{p}$ to $2^{32} \times \frac{i+1}{p} - 1$.

Moreover, each processor's workload is a factor of 1/p as before.

a. Sub-linear

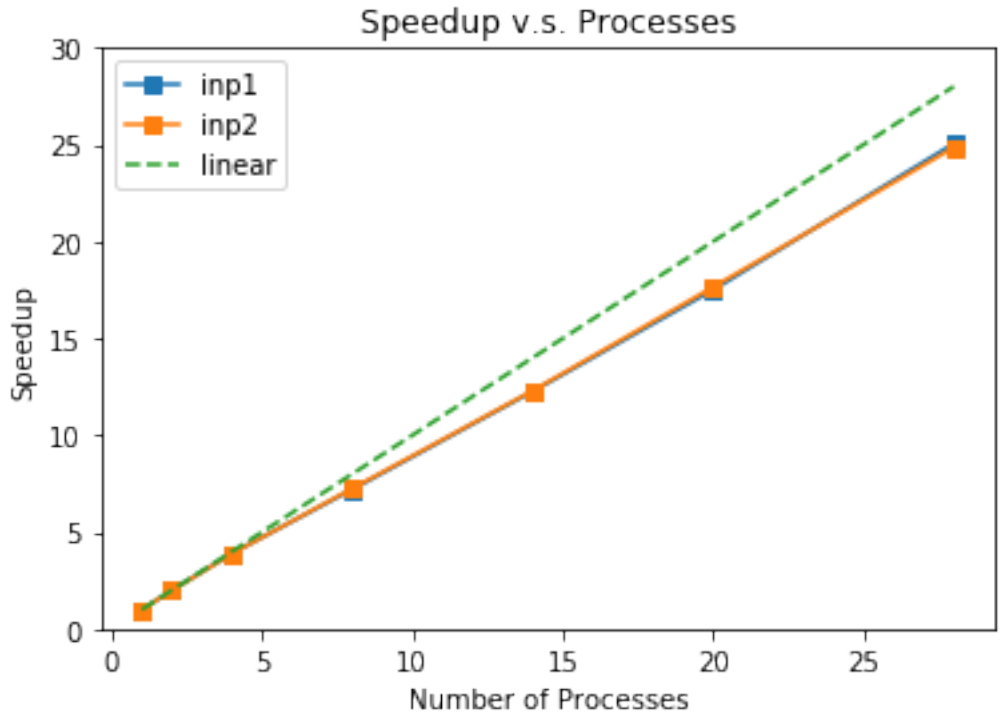
If my key locates in interval $[0, \frac{2^{32}}{p}]$, it will be found in processor 0 by trying as many times as before, which means other processors are wasting of time. Thus, no matter how many processors we have, they cannot shorten our decryption time. I choose the key 101010 in both inp1.txt and inp2.txt. The results are shown in below: speedup decreases a little due to overhead created by communication between processors as number of thread increases.



b. Linear

When key $= 2^{32} - 1$, the speedup can be linear for either number of processors. It can be understood as the number of attempts to reach the last one will be a factor of $\frac{1}{p}$ compared with initial case. Note that in tests of linear and super-linear scenarios, I truncate the scope of key to $[0, 2^{24}]$. It saves computation resources and has not much influence on our observation of these scenarios.

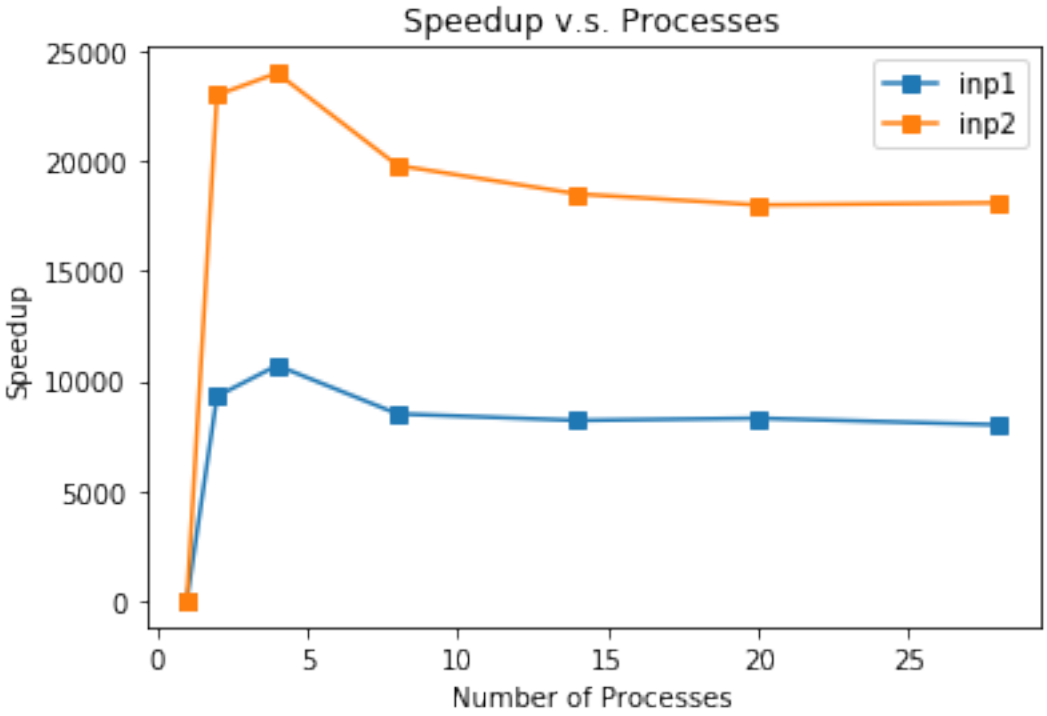
In the figure below, key = $2^{24} - 1 = 16777215$. We can see linear relations for both files.



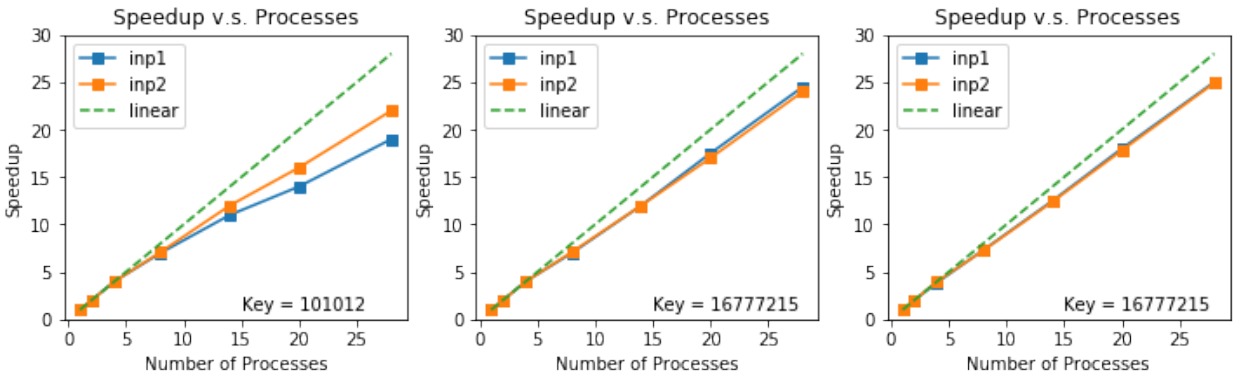
c. Super-Linear

If key locates near the starting points of one processor other than 0, it takes less time to be found soon by this processor. Then we achieve super-liner speedup.

I choose key = $2^{23} + 3 = 8388611$ and the results are shown in the figure below. This key has same distance from a certain processor when p is even, so we can expect similar execution time for any even p. We can see that the figure has great agreement on above analysis.



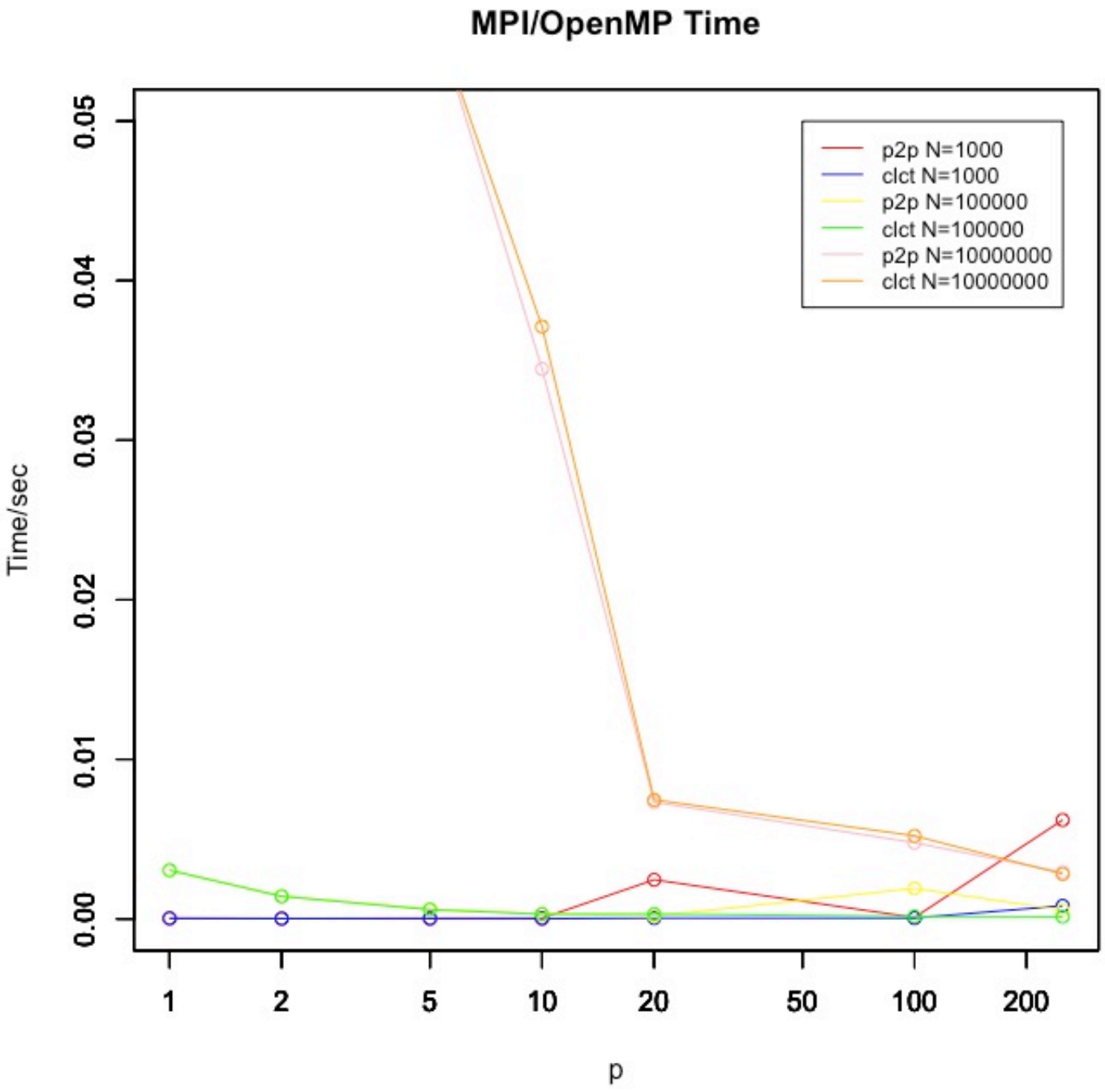
(c) Such dramatic fluctuation of speedup can be attribute the our partition strategy. In order to improve current situation, I suggest to use cyclic partition. The advantage is as stated below: As $n >> p$, unless key locates at beginning few numbers, we can achieve nearly linear speedup. The reason is that the number of attempts to reach either number will be reduced by a factor of $\frac{1}{p}$. It's also a smooth process.



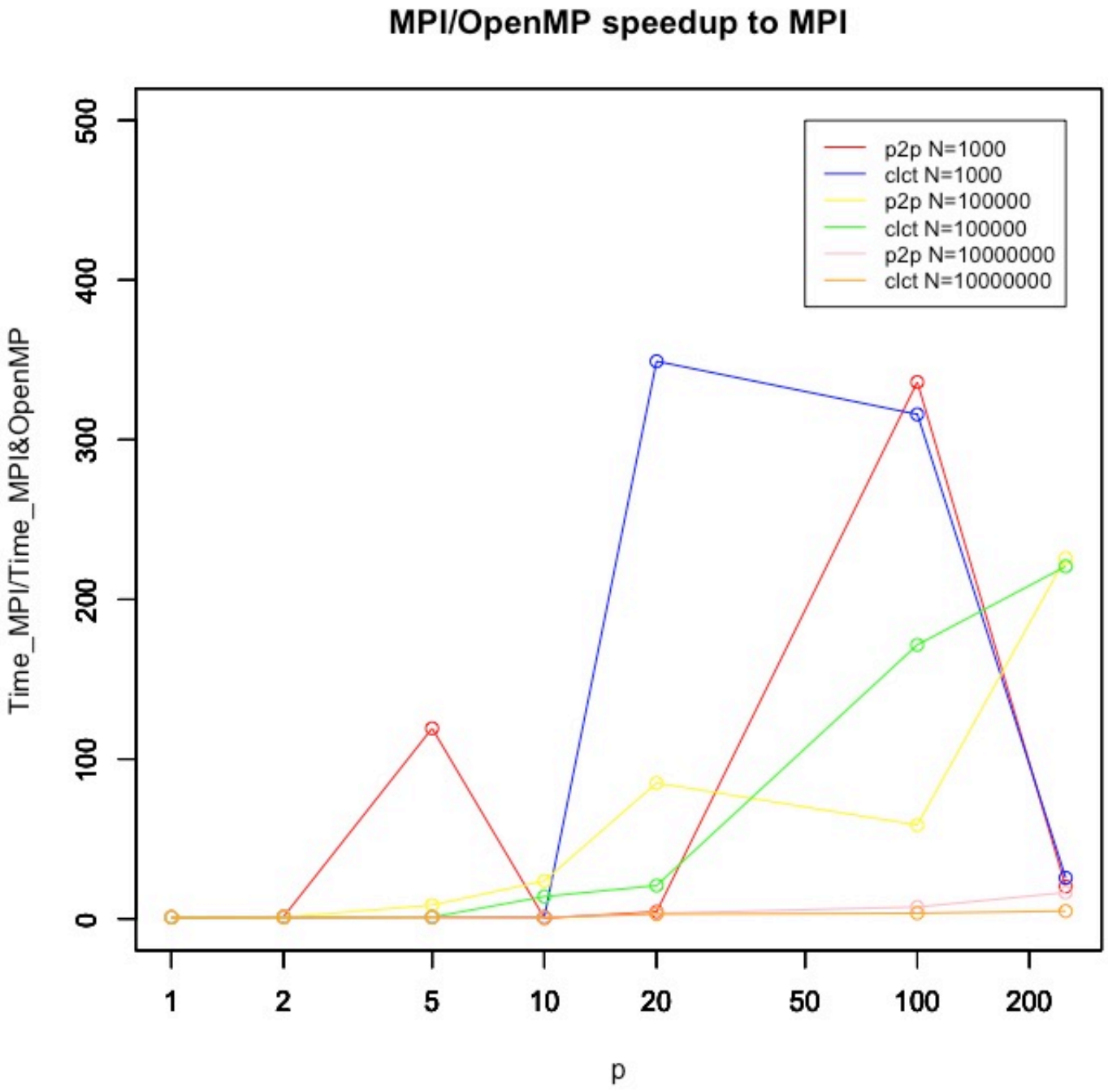
Bonus

set OMP_NUM_THREADS=28

- time of MPI/OpenMP version



- Time of just MPI / Time of MPI/OpenMP



Based on the plot, we observe that the performance of OpenMP/MPI is better than MPI. In each processor, OpenMP changes the sequential computing to parallel version using 28 threads, which makes the code faster. However, there are some points that show extremely high speed up and sharp change in OpenMP/MPI, which may due to the small N. When N goes up, the speed up increases stably and slowly .