Cyndy Ishida
High Performance Computing
Homework #4

Problem 1:

A.)

Block distribution means each processor will have floor(14/4 )or ( n/ comm_sz) where the first chunk will be allocated sequentially through the rank number for each processor.

| block = floor(14/4) =3 | | | | |
|---|---|---|---|---|
| **Time** | **p0** | **p1** | **p2** | **p3** |
| **0** | 4 | 8 | 12 | 14 |

B.)

Cyclic distribution uses a similar to the round-robin partition so every processor will always receive a single component and wrap around to the first rank number until all components have been allocated

i.e. in this example

| **Time** | **p0** | **p1** | **p2** | **p3** |
|---|---|---|---|---|
| **0** | 0 | 2 | 3 | 4 |
| **1** | 5 | 6 | 7 | 8 |
| **2** | 9 | 10 | 11 | 12 |
| **3** | 13 | 14 | | |

C.)

Block – Cyclic uses a hybrid of both, by assigning blocks of components by a programmer defined block size than applying a round robin assignment to processors.

i.e. in this example

| block size = 2 | | | | |
|---|---|---|---|---|
| **Time** | **p0** | **p1** | **p2** | **p3** |
| **0** | 2 | 4 | 6 | 8 |
| **1** | 10 | 12 | 14 | |

B.)

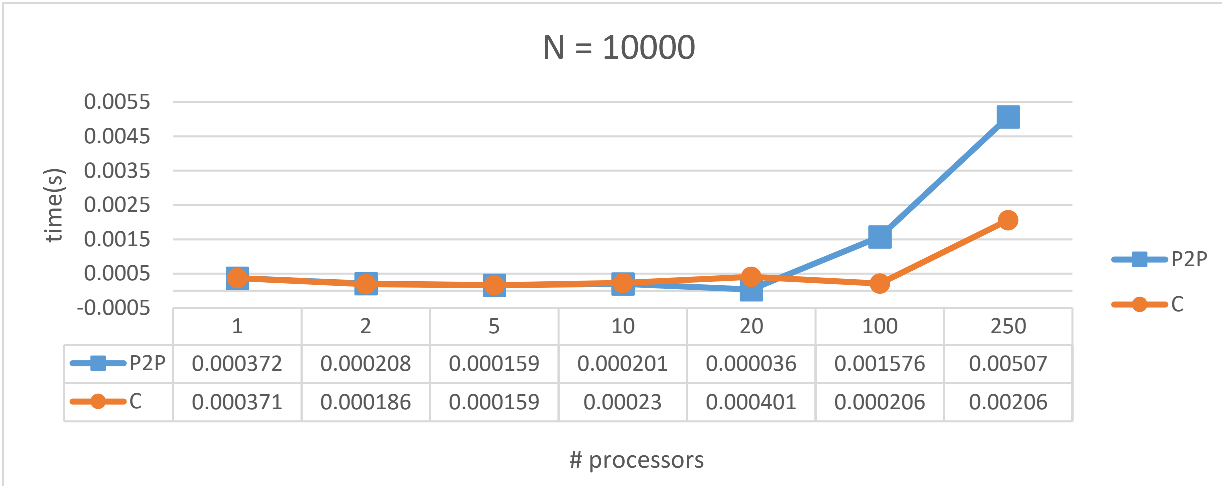


Figure 2 A.

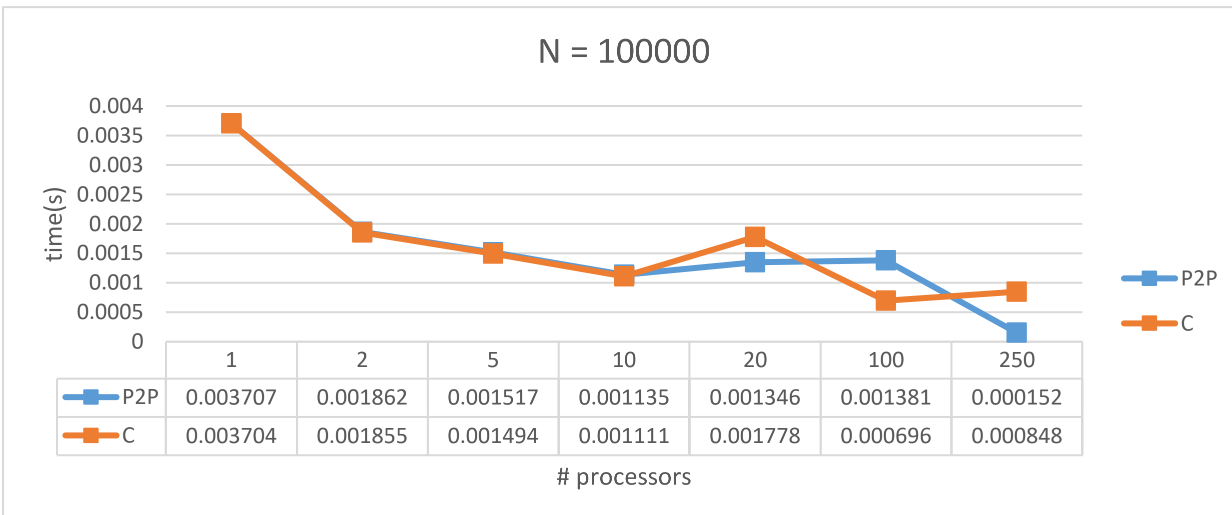


Figure 2 B.

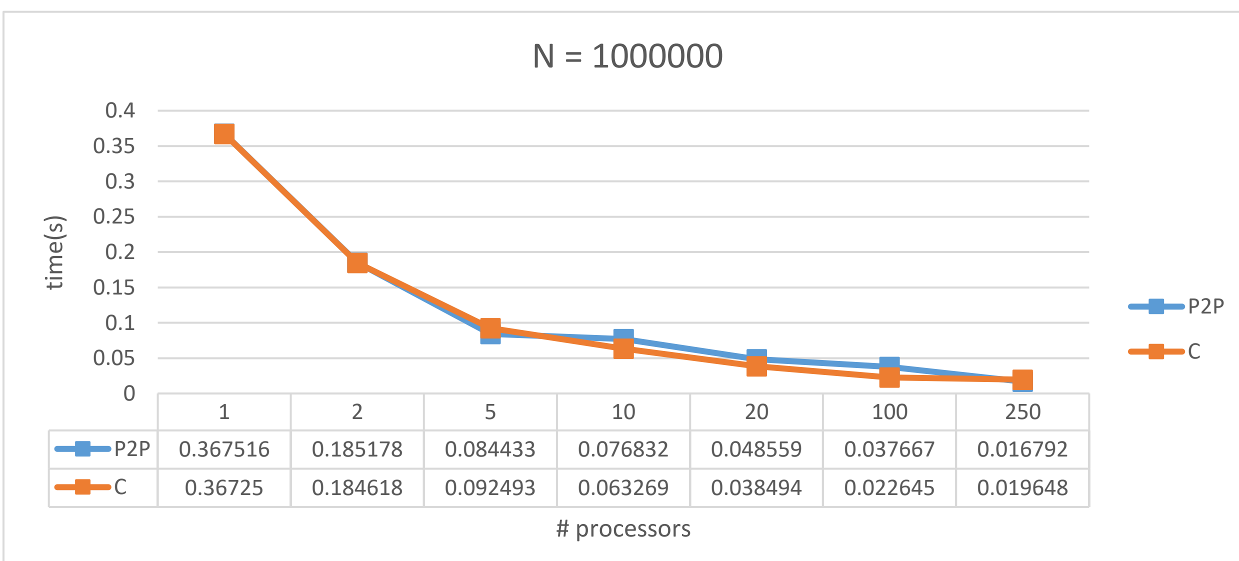


Figure 2 C.

For the most part the collective program ran quicker in all instances of N, but only in an almost indistinguishable amount, as shown by graphs.

I would say that the trapezoidal rule is not a scalable program. As N increases the time also increases by a strong factor so this is isn't a scalable program.

C.)

For the least-squares estimate I parsed my output in Python to create a 21x 2 matrix of all of my (n/p) and (log(p)) values for each processor. Where here p = 21. Put that in Matlab to find the x vector according y hat formula and plugged that in object an 2x 1 matrix of the a and b values.

A = 0.00000000235

B = 0.001923

Here I believe those are expected results since this is a measure of accuracy of the actual result, so the measure of inaccuracy is small meaning the parallelized code is correct and effective.

Problem 3:

B.) Here the program is not very scalable the as the K grows to a larger value the longer it takes to find it.

Here is a graph showing the average of 3 trials with 28 processors and increasing keys.

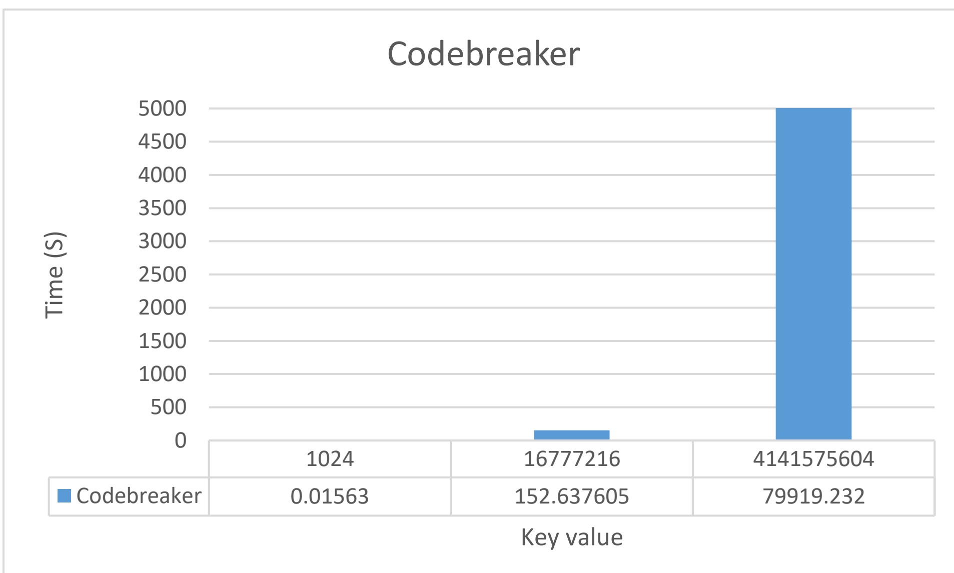


Figure 1. C

C.) Unfortunately I was unable to get the program to compile and execute correctly. I believe that the program should overcome the speedup pitfall by approaching a different partition. Here I believe using a purely cyclic implementation where comm_sz processors work on single keys simultaneously, then check results after trying out another key.

Bonus 1:

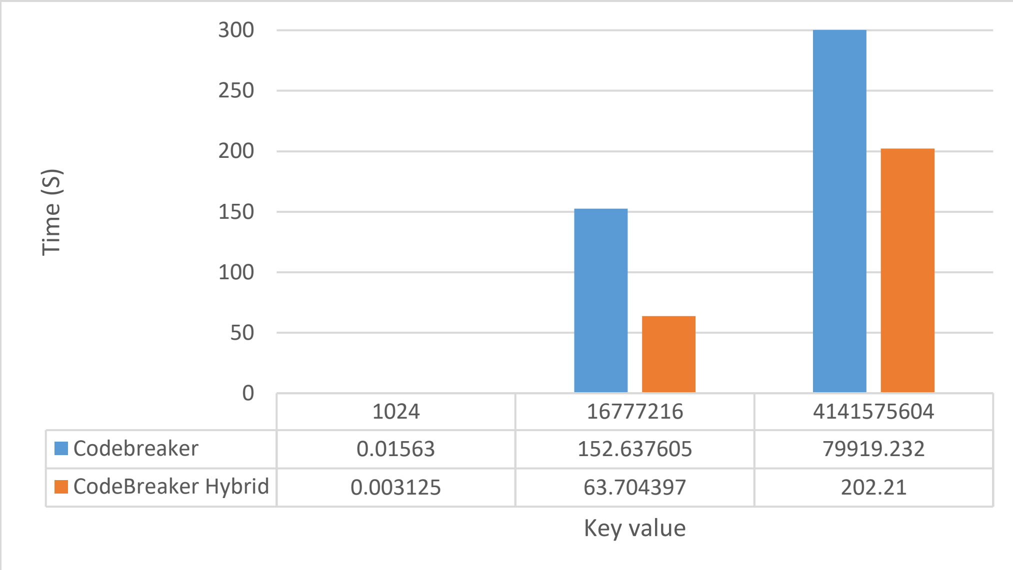


Figure 1. D

I observed very good results.

I created a pragma for to parallelize the outer for loop and create a private key for every thread. I set my thread number to 28 and tried with 28 processors and the speed down was significant.