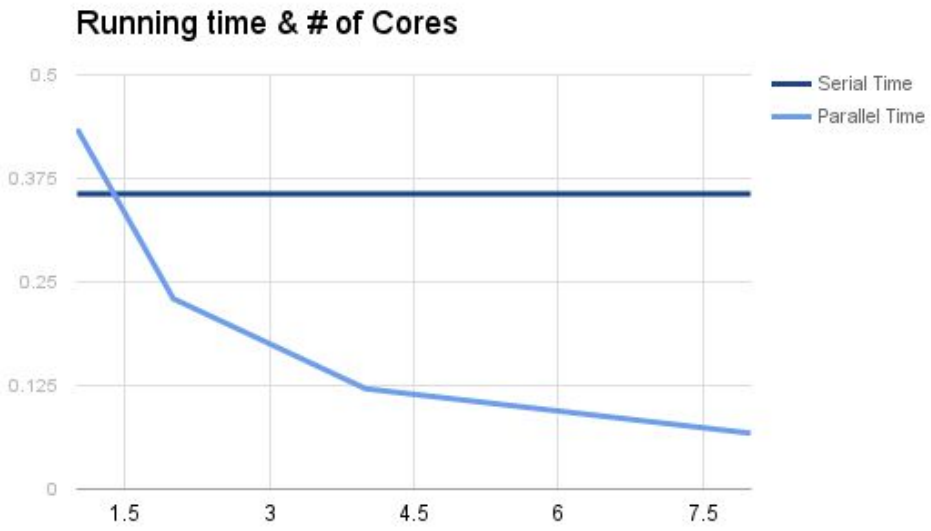
A1-Report-yulian-yih060

Team Members:
Lian, Yuhang A91018743; yulian@ucsd.edu
He, Yi A99075211 yih060@ucsd.edu

Cyclic Partition
Using input: ./mdb -i 200 -t 1 -x 1024 -y 1024 -b -2.5 1 -1 1 -c 1
Strong scaling study on the effect on time as # of core increases using cyclic Partition.

Figure 1



X-axis -- number of cores;
Y-axis -- amount of time took to finish;

Figure 2

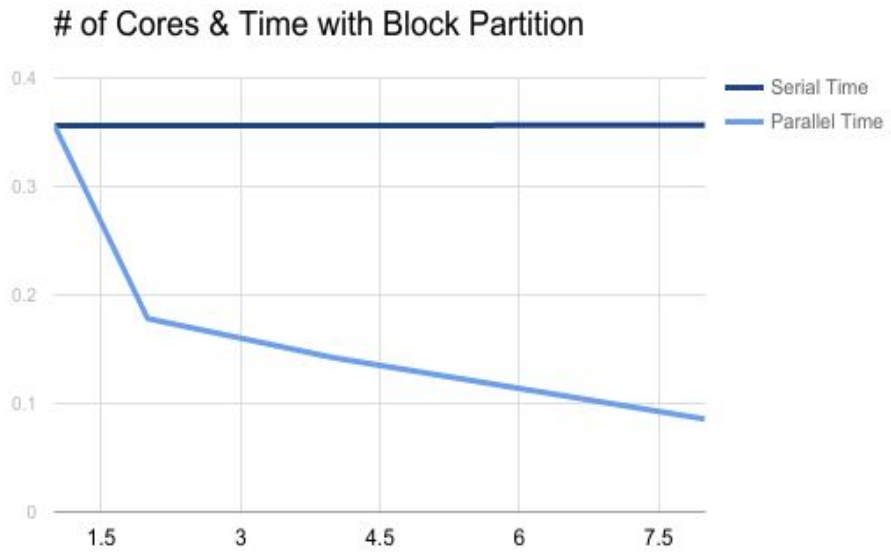
#Cores	Serial Time	Parallel Time	Parallel Speed Up
1	0.356250	0.434663	0.8196
2	0.356250	0.229791	1.55
4	0.356250	0.121092	2.94
8	0.356250	0.067435	5.283

Observation:

When running on 1 core, Parallel Speed Up is lower than 1, which indicates that the program runs in parallel algorithm is slower than in serial algorithm. This is due to the fact that parallel algorithm does not gain any advantages on single core while on the other hand, increase the total amount of workload with distributing works (especially in Block Cyclic Parallel)to different threads.
By increasing the number of cores, as we can easily find in both figure1 and figure 2, while the running time of serial program almost remains the same, the running time of the parallel one dramatically decreases from 0.4s to 0.07s when the number of core reached 8. Based on the fact we can easily conclude that the program in serial algorithm is barely influenced by the number of cores however the program in parallel algorithm benefits hugely from the increment of the number of cores.
Notice that the time of parallel algorithm program does not speed up in double when we double the number of cores. This may due to the fact that, while the number of cores increases, the amount of data the BUS can handle is remain the same. Thus there was no significant reduction in total time when the number of core increased to 8

Block Partition
Using input: ./mdb -i 200 -t 1 -x 1024 -y 1024 -b -2.5 1 -1 1 -c 0
Strong scaling study on the effect on time as # of core increases using cyclic Partition;

Figure 3



X-axis -- number of cores;
Y-axis -- amount of time took to finish;

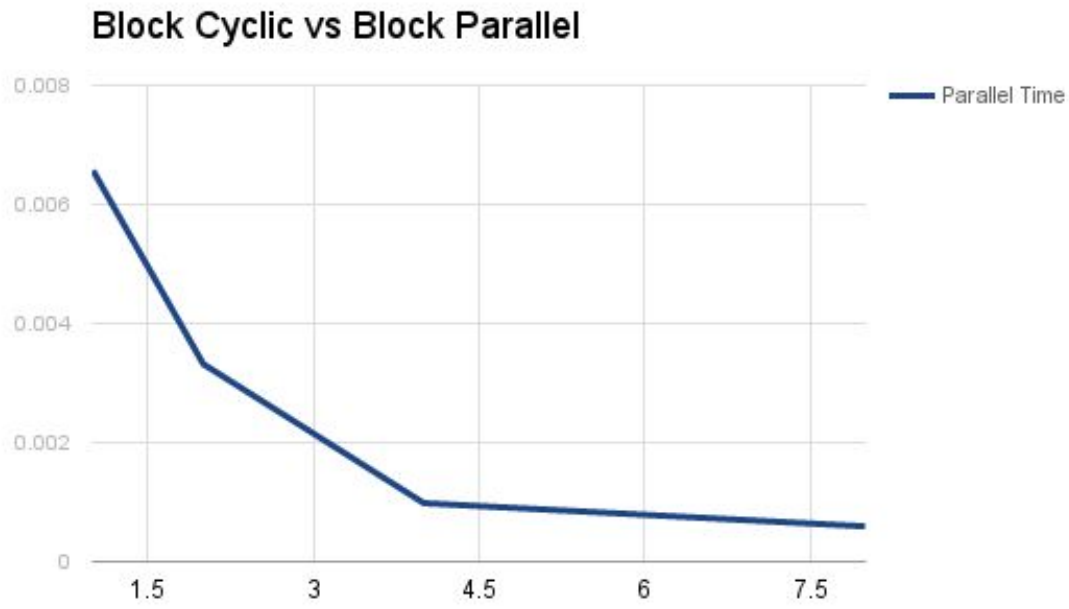
Figure 4

#Cores	Serial Time	Parallel Time	Parallel Speed Up
1	0.356250	0.355513	1.02
2	0.356250	0.17824	1.998
4	0.356250	0.142172	2.51
8	0.356250	0.085601	4.16

Observation: When running the serial part, the average time took is around 0.356250s. Very close to the runtime of using block partition on 1 core because they are essentially the same since there's only 1 thread and the amount of work are identical. However, as the number of core increases, the time take for the program to finish processing the set of data reduced dramatically. From the graph we can see that when the number of cores increased from 1 to 2, there is a huge drop in time. Theoretically it should halve the time. However, as the number of core increases, the amount of data the BUS can handle is still the same. Thus there was no significant reduction in total time when the number of core increased to 8;

Cyclic Partition (Different Chunk Size)
Using input: ./mdb -t 8 -x 128 -y 128 -i 100 -b -2.500000 -0.750000 0.000000 1.000000 -c 8

Figure 5



X-axis -- number of chunk size;
Y-axis -- amount of time took to finish;

Figure 6

ChunkSize	Parallel Time
1	0.006570100
2	0.003319000
4	0.000981808
8	0.000592000

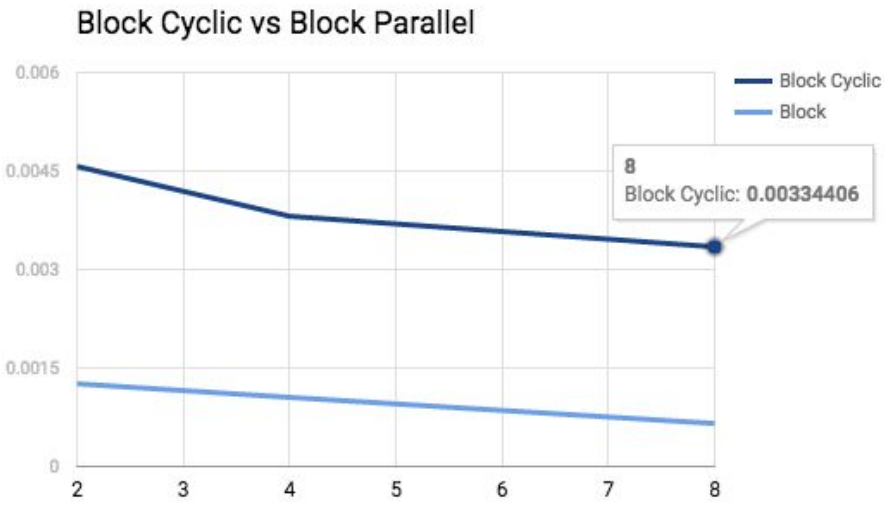
Observation: There's no huge difference between time when varying chunksize(Since the it is very fast already). However, when increase chunksize to 2 from 1 halved the total time; But further increasing chunk size did not achieve expected result -- continuously reduce total time by 2; We suspect that this is also caused by transmitting limit post by BUS.
We started to see increase in time after chunksize = 8 (chunkSize=16 time = 0.001677). So we believe the optimal chunksize is somewhere between 8 and 16. It is because when chunksize become larger, some thread will be allocated with more computation and cause unbalanced workload. And some thread will have to wait until those thread with heavy workload to finish in order to proceed. Thus slowed down total time.

Block Cyclic vs Block
Using input:
./mdb -t (core #) -x 128 -y 128 -i 100 -b -2.500000 -0.750000 0.000000 1.000000 -c 1
./mdb -t (core #) -x 128 -y 128 -i 100 -b -2.500000 -0.750000 0.000000 1.000000 -c 0

Figure 7

#Cores	Block Cyclic	Block
2	0.00457001	0.00125909
4	0.00381112	0.0010519
8	0.00334406	0.000653982

Figure 8



X-axis -- number of chunk size;
Y-axis -- amount of time took to finish;

Observation:
From figure 8 we can find that, the running times of both algorithms decrease in a similar rate while the number of cores increased.
The Cyclic algorithm runs much slower than the un-cyclic one. This may because the fact that, with the chunk size only 1, the cyclic algorithm, even though with a low load imbalance, has resulted in a high overhead of workload assignment. As being shown during the lecture, when the granularity reaches a certain point, the running time will increase while we increase the granularity