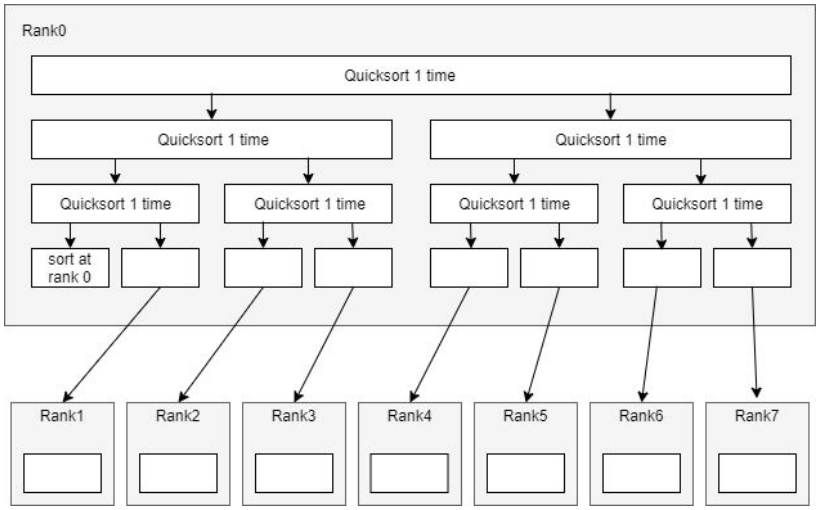
Assignment 2: Quick Sort

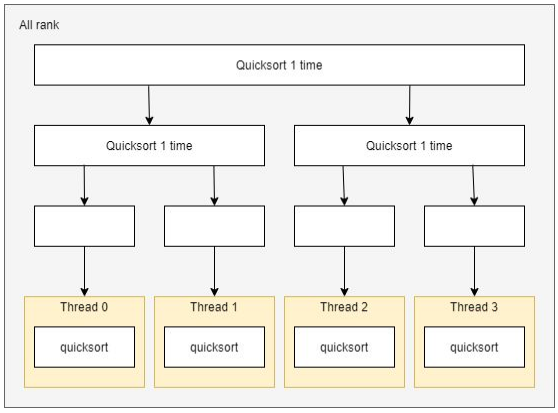
	Member
Nathaphop Sundarabhogin	60070503420
Supakit Artsamart	60070503461

Code Explanation



After reading from the file in rank 0, rank 0 will use quicksort and separate data in partition until the number of partitions equals the number of processes (In this procedure we split work to sort in each thread too).

After we get the number of partitions equal to the number of processes, we'll send data in each partition to each process for sorting.



Then each process will do the above procedure again but separate data into 4 partitions (Follow number of threads). After separating into 4 partitions, this time will send data in each partition to each thread for sorting.

We have tried to use recursive quick sort and iterative quicksort. And found that recursive quicksort is faster than iterative quicksort. Maybe because of access data cost makes iterative quicksort slower. But after that we found optimize iterative quicksort, and we use it in my code right now.

But our code could occur an error if the data is not big enough. If data is not big enough, the program will cannot split data in partitions enough to each process/thread.

Or on the other hand if pivot points of 2 points are close to each other, it could be an error too. Because it cannot split data in that partition enough to send to each process/thread. We've tried to fix this by selecting a pivot point from the middle of the partition instead of last position of partition

Performance Analysis

Amdahl's Law

We've written the code that counts the number of lines in code that can run in parallel and the number of lines in code that needn't run in parallel. We find the number of lines in each code to find the portion between sequence and parallel.

```
cmd: timeout 150 mpirun -f mpi_host -n 1 ./60070503461 sortInput_8M_4.txt 60070503461_out 4
time: 11.809544465038925
stdout: >>Total line seq = 85546433
>>Total line para = 471317330
>>Total seq time = 10.499820
>>Total para time = 1.293669

Output is correct
```

Sequential line of code = 85546433 lines
Parallel line of code = 471317330 lines

Sequential portion = $\frac{85546433}{85546433+471317330}$ **= 0.1536218348**

Parallel portion = $\frac{471317330}{85546433 + 471317330}$ **= 0.8463781652**

Amdahl's Law speedup = $\frac{1}{Sequential\ ratio + (Parallel\ ratio\ /\ number\ of\ process)}$

No. of Processor	Amdahl Speedup
16	4.842135017
8	3.854766203
4	2.738102849
2	1.733670376
1	1

Analytical Model

We've written code to find time communication in different numbers of data size and different numbers of cores. In our experiment there are 1 million data, 2 million data, 4 million data, 6 million data, and 8 million data. All of the data starts sending from 1 core to 15 cores. And this is result

- 1 million

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 1.4883763110265136
stdout: Time communication for 1 core, 1000000 number of data: 0.009689
Time communication for 2 core, 1000000 number of data: 0.016554
Time communication for 3 core, 1000000 number of data: 0.021615
Time communication for 4 core, 1000000 number of data: 0.019668
Time communication for 5 core, 1000000 number of data: 0.026633
Time communication for 6 core, 1000000 number of data: 0.029644
Time communication for 7 core, 1000000 number of data: 0.035834
Time communication for 8 core, 1000000 number of data: 0.038243
Time communication for 9 core, 1000000 number of data: 0.046885
Time communication for 10 core, 1000000 number of data: 0.048448
Time communication for 11 core, 1000000 number of data: 0.054469
Time communication for 12 core, 1000000 number of data: 0.058042
Time communication for 13 core, 1000000 number of data: 0.063566
Time communication for 14 core, 1000000 number of data: 0.065991
Time communication for 15 core, 1000000 number of data: 0.075882
End

Result is not generated
```

- 2 million

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 2.9895587428472936
stdout: Time communication for 1 core, 2000000 number of data: 0.017400
Time communication for 2 core, 2000000 number of data: 0.019608
Time communication for 3 core, 2000000 number of data: 0.035179
Time communication for 4 core, 2000000 number of data: 0.037966
Time communication for 5 core, 2000000 number of data: 0.052357
Time communication for 6 core, 2000000 number of data: 0.056922
Time communication for 7 core, 2000000 number of data: 0.071342
Time communication for 8 core, 2000000 number of data: 0.142303
Time communication for 9 core, 2000000 number of data: 0.218059
Time communication for 10 core, 2000000 number of data: 0.202565
Time communication for 11 core, 2000000 number of data: 0.218254
Time communication for 12 core, 2000000 number of data: 0.146785
Time communication for 13 core, 2000000 number of data: 0.196626
Time communication for 14 core, 2000000 number of data: 0.182793
Time communication for 15 core, 2000000 number of data: 0.243836
End

Result is not generated
```

- 4 millon

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 5.978632339742035
stdout: Time communication for 1 core, 4000000 number of data: 0.033928
Time communication for 2 core, 4000000 number of data: 0.037288
Time communication for 3 core, 4000000 number of data: 0.069761
Time communication for 4 core, 4000000 number of data: 0.082146
Time communication for 5 core, 4000000 number of data: 0.112759
Time communication for 6 core, 4000000 number of data: 0.119302
Time communication for 7 core, 4000000 number of data: 0.148847
Time communication for 8 core, 4000000 number of data: 0.145499
Time communication for 9 core, 4000000 number of data: 0.681962
Time communication for 10 core, 4000000 number of data: 0.534006
Time communication for 11 core, 4000000 number of data: 0.222045
Time communication for 12 core, 4000000 number of data: 0.229803
Time communication for 13 core, 4000000 number of data: 0.259772
Time communication for 14 core, 4000000 number of data: 0.873229
Time communication for 15 core, 4000000 number of data: 0.535206
End

Result is not generated
```

- 6 million

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 8.987113439012319
stdout: Time communication for 1 core, 6000000 number of data: 0.050813
Time communication for 2 core, 6000000 number of data: 0.059342
Time communication for 3 core, 6000000 number of data: 0.104683
Time communication for 4 core, 6000000 number of data: 0.112540
Time communication for 5 core, 6000000 number of data: 0.157702
Time communication for 6 core, 6000000 number of data: 0.166498
Time communication for 7 core, 6000000 number of data: 0.806860
Time communication for 8 core, 6000000 number of data: 0.564106
Time communication for 9 core, 6000000 number of data: 0.270725
Time communication for 10 core, 6000000 number of data: 0.281353
Time communication for 11 core, 6000000 number of data: 0.505806
Time communication for 12 core, 6000000 number of data: 1.032107
Time communication for 13 core, 6000000 number of data: 0.382421
Time communication for 14 core, 6000000 number of data: 0.615695
Time communication for 15 core, 6000000 number of data: 1.066850
End

Result is not generated
```

- 8 million

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 11.551729473285377
stdout: Time communication for 1 core, 8000000 number of data: 0.066587
Time communication for 2 core, 8000000 number of data: 0.079350
Time communication for 3 core, 8000000 number of data: 0.149779
Time communication for 4 core, 8000000 number of data: 0.149205
Time communication for 5 core, 8000000 number of data: 0.231901
Time communication for 6 core, 8000000 number of data: 1.097197
Time communication for 7 core, 8000000 number of data: 0.280215
Time communication for 8 core, 8000000 number of data: 0.282887
Time communication for 9 core, 8000000 number of data: 1.305473
Time communication for 10 core, 8000000 number of data: 0.364368
Time communication for 11 core, 8000000 number of data: 0.418636
Time communication for 12 core, 8000000 number of data: 1.331714
Time communication for 13 core, 8000000 number of data: 0.509329
Time communication for 14 core, 8000000 number of data: 1.401532
Time communication for 15 core, 8000000 number of data: 0.570046
End

Result is not generated
```

From this experiment we found that communication time depends on size of data and number of cores. So, next thing that we do is find communication time sending per core and then find the average.

1M data	Summation of time sending to all core	Communciation Time per core (Time sending/no. of core)	Average of communication time per core
1	0.009689	0.009689	0.005650648316
2	0.016554	0.008277	
3	0.021615	0.007205	
4	0.019668	0.004917	
5	0.026633	0.0053266	
6	0.029644	0.004940666667	
7	0.035834	0.005119142857	
8	0.038243	0.004780375	
9	0.046885	0.005209444444	
10	0.048448	0.0048448	
11	0.054469	0.004951727273	
12	0.058042	0.004836833333	
13	0.063566	0.004889692308	
14	0.065991	0.004713642857	
15	0.075882	0.0050588	

2M data	Summation of time sending to all core	Communciation Time per core (Time sending/no. of core)	Average of communication time per core
1	0.0174	0.0174	0.1227996667
2	0.019608	0.009804	
3	0.035179	0.01172633333	
4	0.037966	0.0094915	
5	0.052357	0.0104714	
6	0.056922	0.009487	
7	0.071342	0.01019171429	
8	0.142303	0.017787875	
9	0.218059	0.02422877778	
10	0.202565	0.0202565	
11	0.218254	0.01984127273	
12	0.146785	0.01223208333	
13	0.196626	0.01512507692	
14	0.182793	0.01305664286	
15	0.243836	0.01625573333	

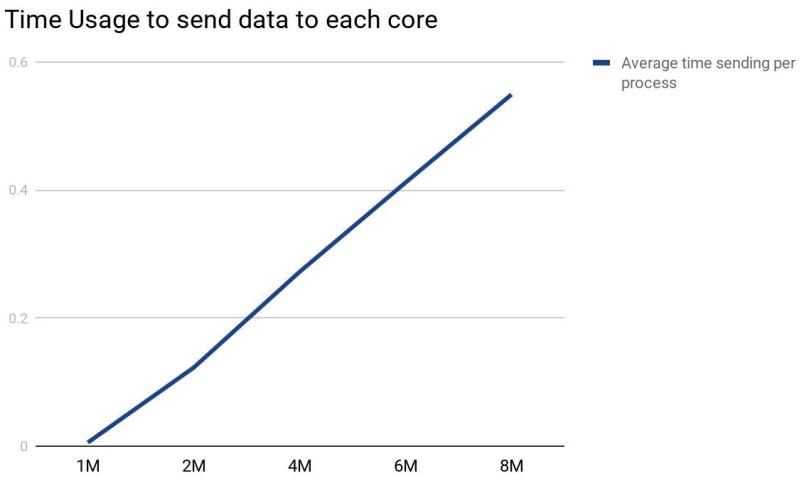
4M data	Summation of time sending to all core	Communciation Time per core (Time sending/no. of core)	Average of communication time per core
1	0.033928	0.033928	0.2723702
2	0.037288	0.018644	
3	0.069761	0.02325366667	
4	0.082146	0.0205365	
5	0.112759	0.0225518	
6	0.119302	0.01988366667	
7	0.148847	0.02126385714	
8	0.145499	0.018187375	
9	0.681962	0.07577355556	
10	0.534006	0.0534006	
11	0.222045	0.02018590909	
12	0.229803	0.01915025	
13	0.259772	0.01998246154	
14	0.873229	0.0623735	
15	0.535206	0.0356804	

6M data	Summation of time sending to all core	Communciation Time per core (Time sending/no. of core)	Average of communication time per core
1	0.050813	0.050813	0.4118334
2	0.059342	0.029671	
3	0.104683	0.03489433333	
4	0.11254	0.028135	
5	0.157702	0.0315404	
6	0.166498	0.02774966667	
7	0.80686	0.1152657143	
8	0.564106	0.07051325	
9	0.270725	0.03008055556	
10	0.281353	0.0281353	
11	0.505806	0.04598236364	
12	1.032107	0.08600891667	
13	0.382421	0.029417	
14	0.615695	0.04397821429	
15	1.06685	0.07112333333	

8M data	Summation of time sending to all core	Communciation Time per core (Time sending/no. of core)	Average of communication time per core
1	0.066587	0.066587	0.5492146
2	0.07935	0.039675	
3	0.149779	0.04992633333	
4	0.149205	0.03730125	
5	0.231901	0.0463802	
6	1.097197	0.1828661667	
7	0.280215	0.04003071429	
8	0.282887	0.035360875	
9	1.305473	0.1450525556	
10	0.364368	0.0364368	
11	0.418636	0.03805781818	
12	1.331714	0.1109761667	
13	0.509329	0.03917915385	
14	1.401532	0.1001094286	
15	0.570046	0.03800306667	

And this is a summary and line graph.

Average of communication time per core	
no. of data	communciation Time
1 M	0.005650648316
2 M	0.1227996667
4 M	0.2723702
8 M	0.4118334
16 M	0.5492146



From the graph, it is almost straight line. So, we think that it could have some constant value to multiply with the number of data to get communication time. (Actually it should be an equation that gets a number of data and finds communication time but we do not know how to find it.)

We find this constant value by finding the average from the table above. And the constant value is 0.00000005448675133. The equation of our communication time is $2 \times p \times n \times k$. Which p for number of processes, n for number of data, and k for constant value (0.00000005448675133), 2 for sending and receiving data.

Next we find read/write time per data and quicksort per data. We find the count time from running the system. And this is our result.

- 1 million data

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 1.0052519207820296
stdout: Read time: 0.395599
Qsort time: 0.127710

Result is generated but it is wrong
```

- 2 million data

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 1.373899437021464
stdout: Read time: 0.757050
Qsort time: 0.269173

Result is generated but it is wrong
```

- 4 million data

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 2.376489704940468
stdout: Read time: 1.469339
Qsort time: 0.566202

Result is generated but it is wrong
```

- 6 million data

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 3.4667849699035287
stdout: Read time: 2.190607
Qsort time: 0.873631

Result is generated but it is wrong
```

- 8 million data

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host -n 16 ./60070503420 sortInput_8M_4.txt 60070503420_out 4
time: 4.522243659943342
stdout: Read time: 2.954980
Qsort time: 1.175682

Result is generated but it is wrong
```

Note: we assume that read time and write time are equal.

From the execution, we bring the read time and quicksort time divided by the number of data to find each data and find average time. Table below is how we calculate it.

No. of Data	Time Read	Qsort Time	TimeRead/Data	Qsort/Data	Avg. TimeRead/Data	Avg. TimeSort/Data
1000000	0.395599	0.12771	0.000000395599	0.00000012771	0.0000003751864833	0.0000001392824833
2000000	0.75705	0.269173	0.000000378525	0.0000001345865		
4000000	1.469339	0.566202	0.00000036733475	0.0000001415505		
6000000	2.190607	0.873631	0.0000003651011667	0.0000001456051667		
8000000	2.95498	1.175682	0.0000003693725	0.00000014696025		

From the table, Avg. time read/data is 0.0000003751864833 sec and Avg. time quicksort /data is 0.0000001392824833. And we'll use this value to multiply with big(O) to find the usage time.

Next let's find the Analytical model. In our code we can separate in 3 parts.

First part is reading files. This part need to be sequential and we think that BigO of this part is n

Second part is computation or quicksort. This part can be parallel and the average BigO of quicksort is nlog(n).

And the last part is writing files. This part needs to be sequential too. As same as reading files, BigO of this part is n.

So, the Analytical Model equation is

Analytical Model = Unsorted function code + (sort function / number of process) + communication time

Tp = 2n*Tread/write + $\frac{nlog(n)*Tqsort}{number\ of\ process}$ **+** $2 \times p \times n \times k$

No. of Processor	Analytical Model Speedup
16	0.6702524203
8	0.9824986277
4	1.199921673
2	1.181353982
1	0.9401516106

Experiment

Sequential code

We execute code that runs only 1 thread and 1 core to find out time usage in sequential execution (Execute without any thread or process help). We did this to find the speedup in our execution.

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_
time: 11.324412330053747
stdout:
Output is correct
---------------------------------

cmd: timeout 150 mpirun -f mpi_
time: 11.376447345130146
stdout:
Output is correct
---------------------------------

cmd: timeout 150 mpirun -f mpi_
time: 11.02336102258414
stdout:
Output is correct
---------------------------------

cmd: timeout 150 mpirun -f mpi_
time: 10.968391825910658
stdout:
Output is correct
---------------------------------

cmd: timeout 150 mpirun -f mpi_
time: 10.661688321270049
stdout:
Output is correct
```

Find Avg Ts from sequential code
Avg Ts = (11.32 + 11.37 + 11.02 + 10.96 +10.66) = 11.066

Parallel code

This is the time usage in our main assignment submission by sharing computation to process and thread.

```
Compile Log:
cmd: timeout 150 mpirun -f mpi_host ·
time: 10.804978574626148
stdout:
Output is correct
---------------------------------

cmd: timeout 150 mpirun -f mpi_host ·
time: 10.448804936837405
stdout:
Output is correct
---------------------------------

cmd: timeout 150 mpirun -f mpi_host ·
time: 12.05533798923716
stdout:
Output is correct
---------------------------------

cmd: timeout 150 mpirun -f mpi_host ·
time: 12.569184801075608
stdout:
Output is correct
---------------------------------

cmd: timeout 150 mpirun -f mpi_host ·
time: 11.8221016661264
stdout:
Output is correct
```

Parallel code runtime

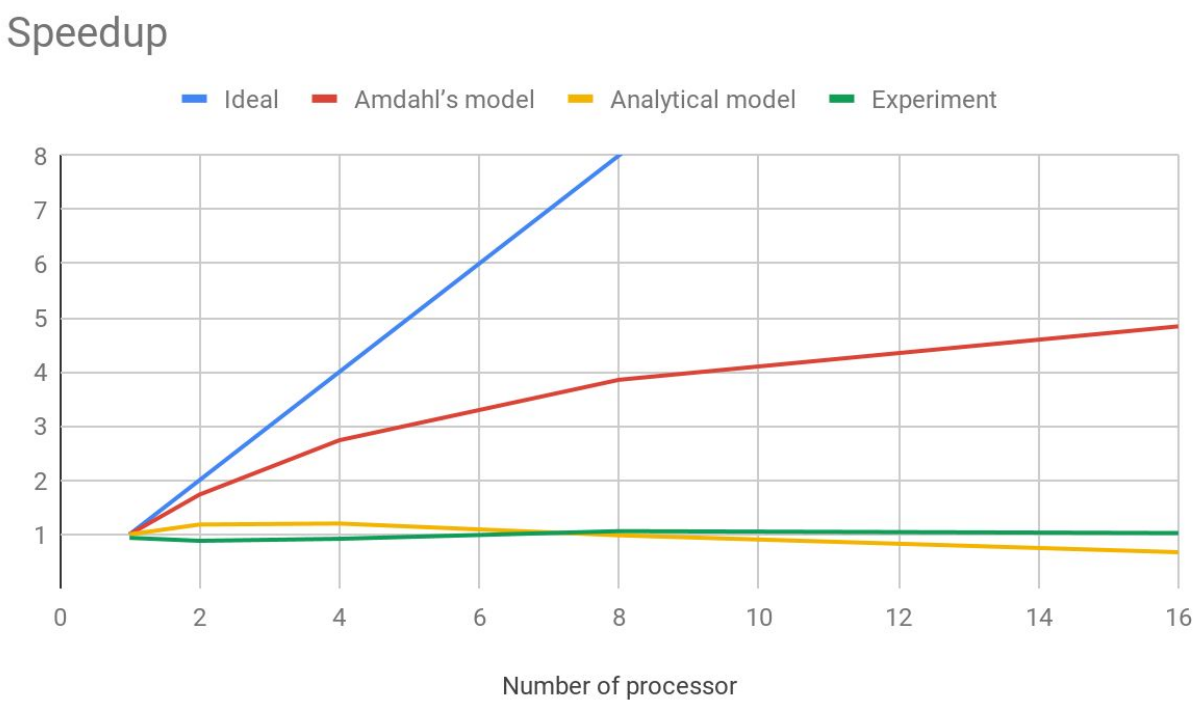
No. of Processor	Parallel Runtime
16	10.8050
8	10.4488
4	12.0553
2	12.5691
1	11.8221

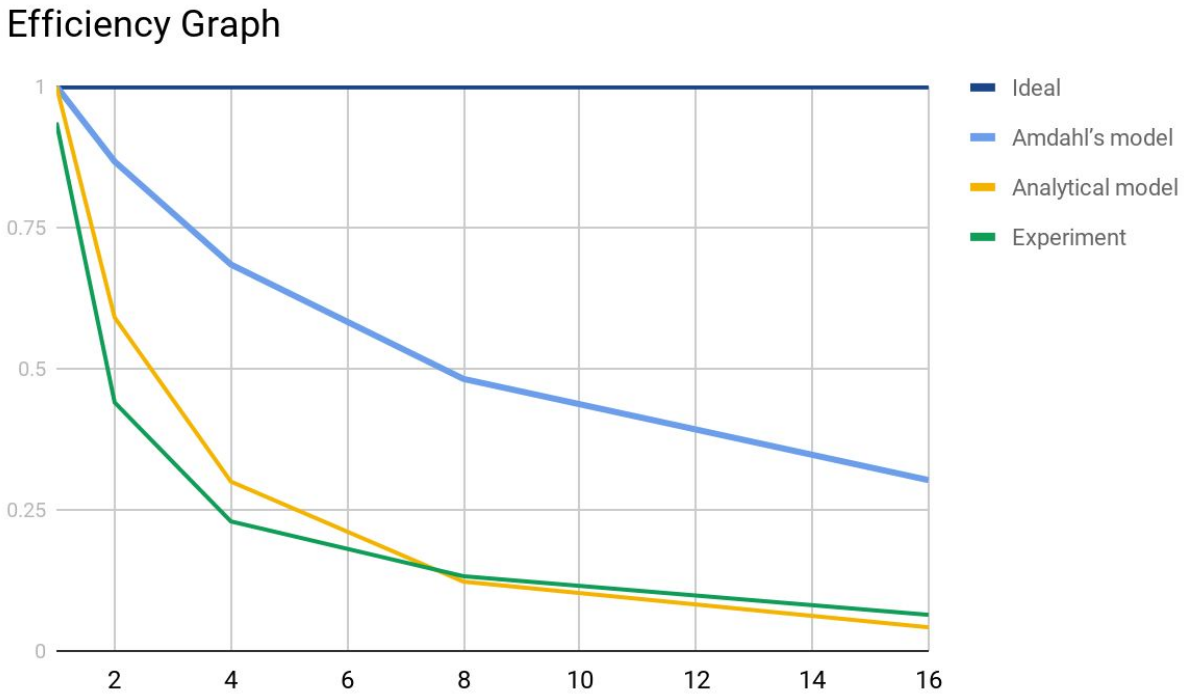
Speedup

From execution time of sequential code and execution time of parallel code, we can find speed up by sequential execution time divided by parallel execution time.

No. of Processor	Speedup
16	11.066 / 10.8050 = 1.024155484
8	11.066 / 10.4488 = 1.059068984
4	11.066 / 12.0553 = 0.9179365093
2	11.066 / 12.5691 = 0.8804130765
1	11.066 / 11.8221 = 0.9360435117

Graph


Speedup
Ideal
Amdahl's model
Analytical model
Experiment
8
7
6
5
4
3
2
1
0
2
4
6
8
10
12
14
16
Number of processor


Efficiency Graph
1
0.75
0.5
0.25
0
2
4
6
8
10
12
14
16
Ideal
Amdahl's model
Analytical model
Experiment

Discussion

From the speedup graph, as you can see Amdahl models are too overestimated from the experiment. While Analytical Model is nearly our experiment but the speedup of the Analytical model drops when the number of processes increases.

Starting to focus on Amdahl's Model. Following the limitation of Amdahl's Law, the Amdahl model is often overestimated. Also, the Amdahl model does not calculate with Time overhead. But it is too overestimated from the experiment. Maybe we should optimize more in data access in quicksort or find the new fastest way to give tasks to each process or thread. So, it will increase the performance of the experiment, and don't drop too much from the model.

Next model, Analytical Model. This model is hard to find execution time and communication time because we don't know time usage for each command. We have to find time usage from experiments on what we focus on and find the average. But this model can improve more if we can find an equation for execution time and communication time to get the time that depends on the number of data and number of processes.

From our experiment code, even we use the optimize quicksort function. But it is not fast enough. There are some points that we can improve more such as finding a new way to split an equal partition. If the partition that sends to each process/thread isn't equal, it'll have some process that gets many data to sort while some process gets only a few data to sort and finish earlier. Also, we think that there are a lot of page faults when executing the program. If we can improve the data access, our program will extremely improve.