MPIAssignment

1.Overview

This report is mainly focused on solving the problem mentioned by the assignment, which is to accelerate the Floyd algorithm with three different methods using MPI framework. The efficiency of those methods are quantitatively measured.

2. Tools and theoretical estimation to quantitatively analysis

We use the efficiency to quantitatively evaluate the multi-process accelerated methods. Normally, the efficiency is define as

$$efficiency = \frac{time\ on\ 1cpu}{p \times time\ on\ p\ cpus}$$

Specifically, in this assignment, the efficiency is

$$seqT = min_r T_r$$
$$efficiency = \frac{seqT}{np \times min_r max_w T_{rw}}$$

where r represent the run, w the worker, np the number of processor and seqT the execution time of the sequential code.

Theoretically, in this assignment the efficiency should be below 100%, because there are always some parts which are not parallel, as well as communication overhead caused by distributive computing.

3.Implementation for each method

All these three method are based on the original sequential Floyd algorithm, but:

- In the first method we use `MPI.Send` and `MPI.Recv` to communicate between different processes, and use `MPI.Barrier` to coordinate between these processes.
- In the second method we use `MPI.Bcast` to broadcast messages between different processes and coordinate them.
- In the third method we only use `MPI.Send` and `MPI.Recv` to send messages to communicate between different processes, and in the message, we set the tag to be the row index of the row contained in this message. Each worker maintains a cache and store the newly received row if the row is not the currently wanted row. In this way we no longer need any sort of blocking method.

4. Experiment Set-Up

For the size of graph in Floyd algorithm N, we set N=3200.

Before testing these three methods, we shall run test on the sequential method first as the baseline and record the time usage. It will be repeated 5 times, and the minimum value will be used.

We will run tests for different number of processors/workers(NP). Possible values for NP is 2,4,8,16.

For each NP, we run tests on different methods(1 to 3). In the test each method is repeated for 5 times, record the time used on each worker each time, and calculate the efficiency mentioned in Section 2.

5. Experiment Result and Analysis

5.1 How does your implementation perform in terms of runtime, and efficiency?

When the number of processor is 4, method 1, 2 and 3 can achieve an efficiency of 95.2%, 95.4%, 96.1%

When the number of processor is 2, method 1, 2 and 3 can achieve an efficiency of 96.2%, 95.9%, 96.6%

When the number of processor is 8, method 1, 2 and 3 can achieve an efficiency of 59.4%, 60.0%, 59.5% When the number of processor is 16, method 1, 2 and 3 can achieve an efficiency of 30.7%, 30.8%, 31.1%

5.2 What happens to the performance if you scale to more workers?

No matter which method, the efficiency(performance) drops significantly when the number of workers increases. This is because the proportions of the communication costs among all costs on each worker increases.

When the number of workers increases, for each worker, the data amount transferred through communications is still the same, while the amount of calculations decreases. Thats's why the proportions of the communication costs increases and the efficiency drops.

5.3 What effect does the different implementations have on your code?

When the number of workers is fixed, it is always that efficiency of method 1 is smallest, then efficiency of method 2, and efficiency of method 3 is highest. The reasons are as follow:

1. Method1 uses blocking methods to send messages(Send!), and method1 has to send messages to other works one by one. Besides method1 also use barrier to coordinate different workers every round(for each k). That is why method1 is the least efficient.

2. Method2 uses blocking methods (Bcast!), which also coordinate different workers every round(for each k) . However, method2 calls less blocking methods because it doesn't sends data to other workers one by one.

3. Method3 uses blocking methods to send messages(Send!), but it no longer requires any sort of synchronization and coordination between workers. Instead, tag in MPI message is used to mark the row index of transferred row, and rows supposed to be used later will be cached. By doing so, we can deal without disordered messages correctly without blocking any workers. That is why method3 is the most efficient.

6. Appendix 1: Efficiency Results Table (N=3200)

Functions	Number of Processors = 2	Number of Processors = 4	Number of Processors = 8	Number of Processors = 16
method 1	0.962914	0.952904	0.594619	0.307001
method 2	0.959932	0.954545	0.600062	0.308778
method 3	0.966486	0.961104	0.595842	0.311578

7. Appendix 2: Original Test results for np=4

7.1 Baseline

r	Tr
1	32.302392802
2	32.630568008
3	32.261376537
4	32.263299236
5	32.253622576

seqT=32.253622576

7.2 Method 1(NP=4)

r	w1	w2	w3	w4	max
1	8.559353881	8.559353191	8.559353182	8.559353234	8.559353881
2	8.716228715	8.716230015	8.716227632	8.71623005	8.71623005
3	8.56136055	8.561364865	8.561361961	8.561364777	8.561364865
4	8.461913956	8.461925055	8.461920958	8.461925042	8.461925055
5	8.53417861	8.534180714	8.534177737	8.534180736	8.534180736

$min_r max_w T_{rw} = 8.461925055$ efficiency=0.952904

7.3 Method 2(NP=4)

r	w1	w2	w3	w4	max
1	8.586442386	8.586388021	8.58644173	8.586360023	8.586442386
2	8.568839447	8.568825858	8.568862961	8.568796127	8.568862961
3	8.588283984	8.58824637	8.588309558	8.588222052	8.588309558
4	8.665619672	8.665560708	8.665618896	8.665538174	8.665619672
5	8.447359869	8.447381558	8.447338707	8.447358918	8.447381558

$min_r max_w T_{rw} = 8.447381558$ efficiency=0.954545

7.4 Method 3(NP=4)

r	w1	w2	w3	w4	max
1	8.480138924	8.480139154	8.480127013	8.480188394	8.480188394
2	8.691141677	8.691045032	8.691124222	8.69104559	8.691141677
3	8.517798033	8.51781129	8.517809677	8.517830471	8.517830471
4	8.389717679	8.38967942	8.389729057	8.38968674	8.389729057
5	8.513858496	8.513849661	8.513889121	8.513849854	8.513889121

$min_r max_w T_{rw} = 8.389729057$ efficiency=0.961104