HPC Program Work Report

Name: Ge Huang, GTID: 903071530

02/26/2015

1 Introduction

The problem needs students to find out all the number that is within d hamming distance of all given numbers. This problem is different from transitional problem since it puts high demands on high performance computation. For example, for a sequence of binary number of 20 digit, it has 1,048,576 possibilities. For more digits, the number of possibilities would become extreme large. So we need to take advantage of parallel computing, combine a bunch of processors to finish this task. Another interesting part is that this problem is a problem of "large number".

2 Sequential Algorithm

I uses binary tree DFS to fulfill my goal. At first, I form a virtual base node and it hold the value of S1 at level 0. I start by creating left tree node. When creating left tree node, I inverse the digit at tree level, while do nothing for the right tree node. To control node creating, I define path sum and node height (it is the same as tree level). Left node has the value of 1 while right node has the value of 0, so path sum stands for the current account for how many digits has been inverted.

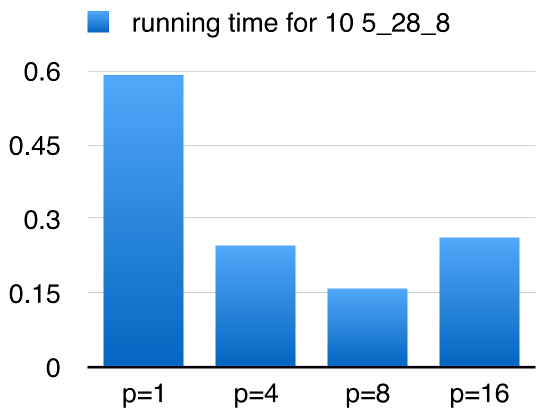
Using the same spirit of sequential algorithm, master makes a similar tree. Its height is 'masterdepth', rather then l. The number that left node holds is a qualified number, so master wants to send it to workers. Before sending the number, master makes a message with length of 2. The first part of the message contains a mark to determine when to terminate the worker. Normal message contains mark of the value of 1, while terminate message contains 0 in its first digit. After master send out all qualified number, it send another p-1 terminate messages to p - 1 workers. Master's another task is to collect workers' solutions. This will happens after all workers finish their job.

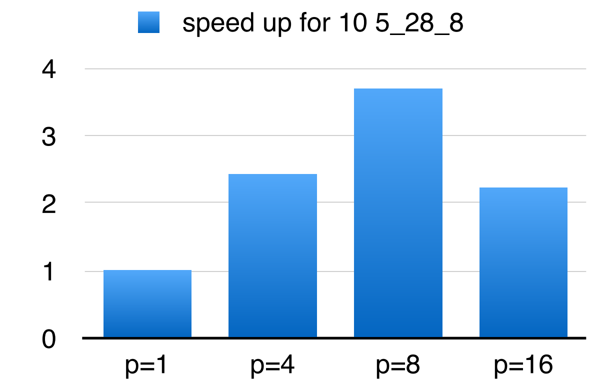
3 Communication Protocols

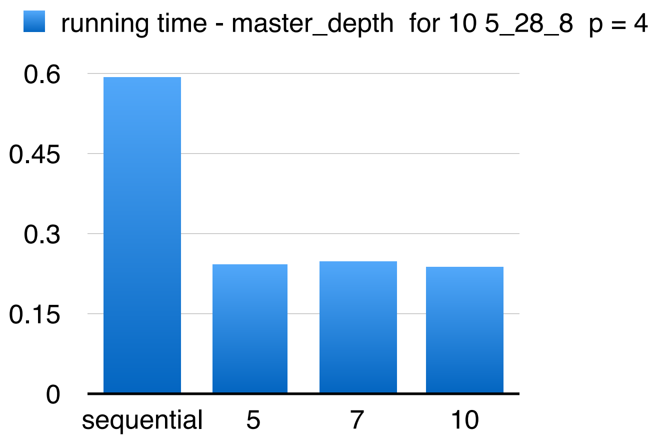
In my communication protocols, I guarantee that master produce messages and workers receive that message at the same time. When worker finishes its task, it stores the results in a result vector. After every worker finish its job, it resends the results back to master. On master side, it keeps producing messages. After that, the master waits to receive results from worker.

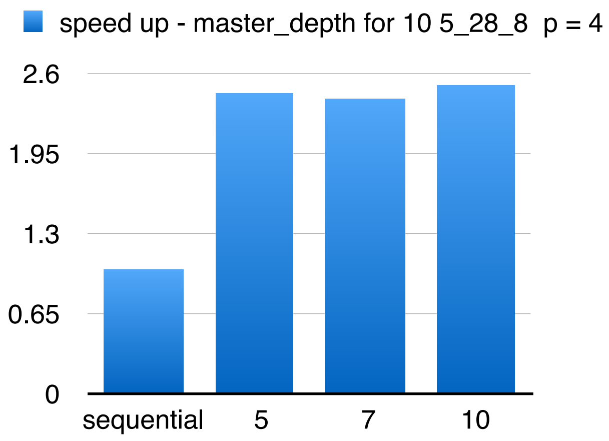
The computation time for master includes the time for producing messages. The communication time is sending out message to worker and receiving results. Master keeps the same style: making message and giving out. Meanwhile, worker performs the following: getting message, calculating, and storing. In this way, computation time and communication time are overlapped. My algorithm achieves that master can keep sending message out, rather than waiting for workers' response. On the other hand, worker can simultaneously keep receiving message and produce results. However, my algorithm has its disadvantage. Master sends out its receiving request after it finishing producing messages, while worker will only send back results after it completes all computations. The timing is hard to achieve, so the back coming results may be lost.

4 Runtime and speedup plots by varying problem size, parameter k and number of processors





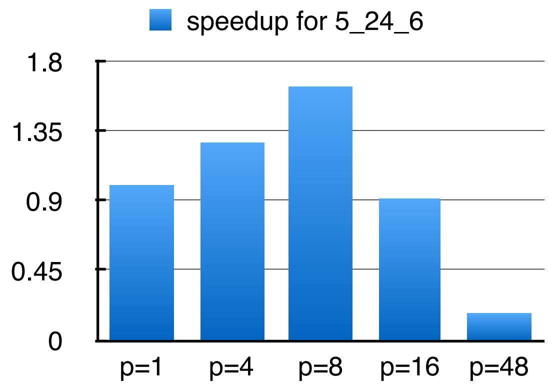


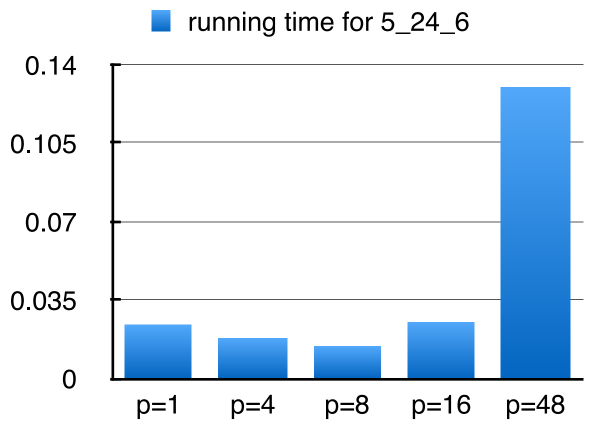


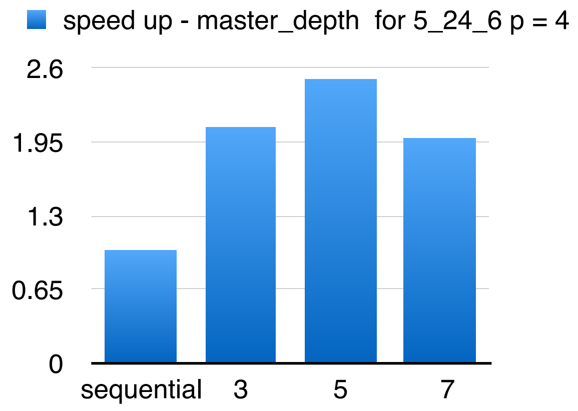
For problem of 5_28_8, the maximum speedup happens when using 8 processors. When using 16 processors, the speedup is almost half of maximum speedup. As for master_depth, there is no huge difference in this input.

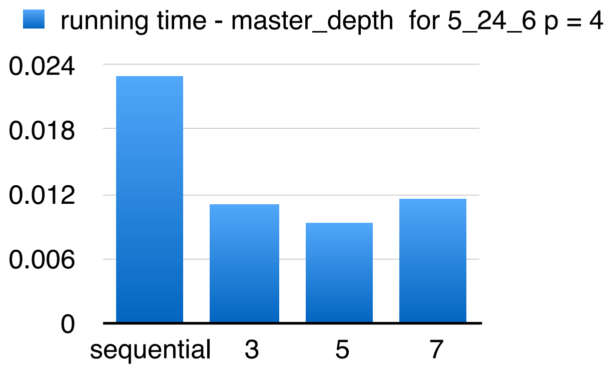
Input	Output

5 24 6
7044851
2179191
7098723
4294775
6768819
	Output matched the given solution.





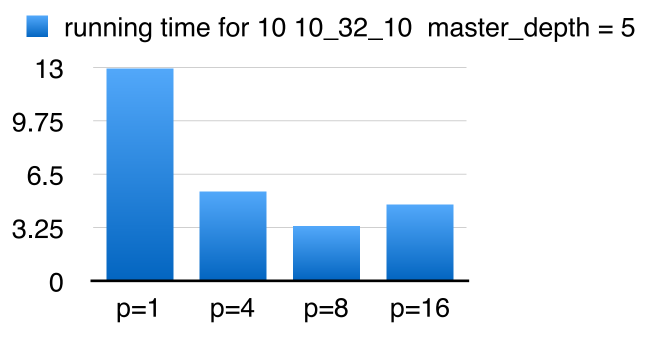


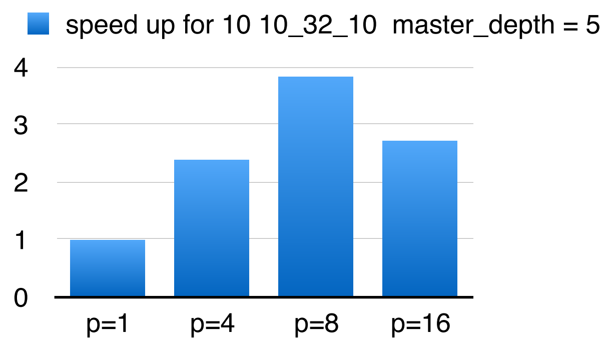


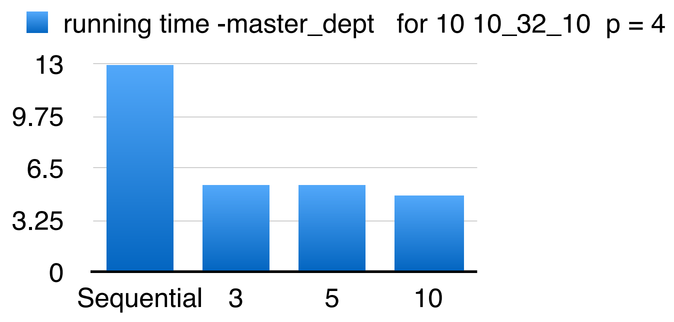
Input	Output

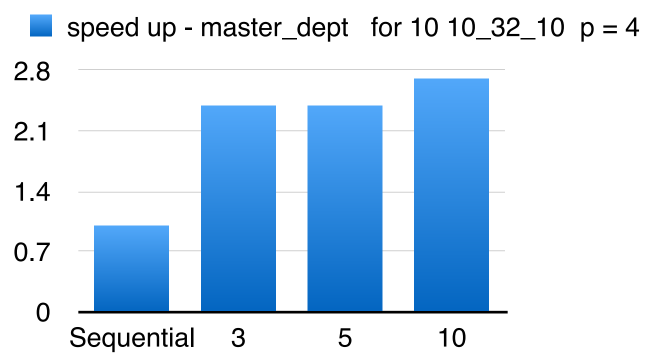
5 28 8
140659837
19398707
73484926
216239187
183060692
	Output matched the given solution.

For problem of 5_24_6, the maximum speedup happens when using 5 processors. As for master_depth, k=5 achieves the best speedup for 4 processors.









Input	Output

10 32 10
342849139
2172217791
472189499
3186321499
3514655283
2826635509
3362278129
3383346291
2172442739
3295235199
	141125683
	141125747
	141126259
	141387891
	157895283
	158156915
	226052723
	226314355
	275335283
	292096115
	409552947
	409561203
	410599547
	410601587
	426330227
	426330743
	426592375
	494487667