Distributed Systems Homework 2

Kunal Jain

2019111037

1 Question 1

1.1 Problem Statement

Given a non singular matrix, find it's inverse using the row reduction method. Assume rows are distributed across processors.

1.2 Solution Approach

We distribute the rows across the processors in a way such that row i is given to processor with id (i mod P) where P is the total number of processors. Our solution will have the following steps:

1. **Sequential** Gauss Jordan elimination: for i from 1 to n
   (a) **Distributed** Find the maximum element in column i. Say it is in column x
   (b) **Distributed** Swap row x and row i
   (c) **Distributed** Use row i as pivot to make column i as 0 for all the rows below
2. **Distributed** Divide each row by it's diagonal element to make the diagonal element 1.
3. **Sequential** Back substitution: for i from 1 to n
   (a) **Distributed** Get the i^th row as pivot
   (b) **Distributed** Use the pivot to make the i^{th} column 0.

1.3 Time Complexity

Gauss Jordan elimination now takes $O(\frac{N^3}{k})$ time where N is the size of the matrix and k is the number of processor. Back substitution also takes $O(\frac{N^3}{k})$ time while there are also k messages of size n sent.

Therefore, the computational complexity is $O(\frac{N^3}{k})$ and communication complexity is $O(N * k)$

1.4 Results

First row shows time taken by the sequential code

k	$N = 10^2$ (s)	$N = 10^3$ (s)	$N = 10^4$ (s)
1	0.030407	30.9654	>3600
2	0.00597257	2.74718	3340.02
3	0.093328	3.73689	2681.49
4	0.153057	2.65895	2271.79
5	0.0217126	2.40121	1981.23
6	0.240462	2.0233	1821.46
7	0.28813	1.8945	1562.84
8	0.43122	1.74449	1336.42
16	1.0193	3.29222	688.04
24	1.60134	5.76322	475.86
32	1.68142	10.5919	323.721

2 Question 3

2.1 Problem Statement

Use the Taylor series of sin:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots$$

to estimate it's value for a given sin and accuracy (number of terms).

2.2 Challenges

We can not calculate term with coefficient r simply by calculating x^r and $r!$ and dividing them. Doing this will greatly hinder our precision and computational capacity as both of these terms will quickly overflow when calculated independently.

Looping from 1 to r and multiplying $\frac{x}{i}$ for each i will greatly improve our results.

However, this adds another layer of challenge to the naive solution. Now, whichever processor has the n^{th} term will be required to loop through to n for calculating the result, thus rendering the distribution of compute pointless. We overcome this challenge in the next section.

2.3 Solution Approach

Assume we have to calculate for n terms across k processors. We divide the n terms of series across the processors in a sequential manner, ie, first $\frac{n}{k}$ go to processor 0, the next $\frac{n}{k}$ go to processor 1 and so on.

Therefore, on the i^{th} process will be terms from $\frac{n}{k} * (i-1) + 1$ to $\frac{n}{k} * i$. Let $P = \frac{n}{k} * i$, then the last term in processor i will be $\frac{x^{2*P-1}}{(2*P-1)!}$ and the first term of processor $(i+1)$ will be $\frac{x^{2*P+1}}{(2*P+1)!}$.

We see that we can easily take out the last term of the previous process common out of all the terms of the current processor and simplify our calculations. We then collect all the values calculated by each processor along with the last term they calculated and add them iteratively.

2.4 Time complexity

On each processor, we run a loop of $\frac{n}{k}$ in parallel. After all the processes are done, we collect their values and multiply and add them sequentially.

Therefore, our net computational complexity becomes $O(\frac{n}{k} + k)$ while our communication complexity is $O(1)$.

2.5 Results

First row shows time taken by the sequential code.

k	$N = 10^3$ (ms)	$N = 10^5$ (ms)	$N = 10^9$ (ms)
1	0.071	1.63	15906.22
2	0.5878640	0.7884490	6988.747263
3	0.0977380	0.5698540	4661.214732
4	0.1121000	0.4629980	3807.141649
5	0.1162540	0.4123320	2812.059931
6	0.1411040	0.3850420	2541.054345
7	0.1071420	0.3840430	2188.163206
8	0.0981070	0.2867760	1969.121063
16	0.1463860	0.2415500	970.629555
24	0.1938040	0.227898	651.880254
32	0.2493970	0.2288570	494.572494

3 Question 5

3.1 Problem Statement

Given a graph, find the number of cycles of length 3 and 4 in it

Note It is assumed that we are receiving the graph in the form of an adjacency matrix of size N whose transpose is equal to the matrix itself (since the graph is undirected).

3.2 Solution Approach

Let A be the adjacency matrix of the graph. Then, the sum of diagonal elements of A^3 divided by 2 will give us the number of C3 in the graph.

Similarly, let S_i be the i^{th} diagonal entry in A^4 and M_i be the sum of the i^{th} row in A^2. That is, M_i are all the paths of length 2 that start from i and S_i are all the paths of length 4 that start and end at i. Then, number of cycles of length 4 that start and end at i will be $S_i - M_i$. The summation of $S_i - M_i$ over all i will give us four times the number of cycles of length 4.

Let $S = \Sigma_{i=1} n S_i$ and $M = \Sigma_{i=1} n M_i$, then our final answer will be $\frac{S-M}{4}$. Therefore, the problem boils down to computing A^2, A^3 and A^4 efficiently.

3.3 Limitations

The algorithm assuumes that we have a square number of processors.

3.4 Results

First row shows time taken by the sequential code.

k	$N = 10^2$ (s)	$N = 10^3$ (s)	$N = 10^4$ (s)
1	0.011675	13.9106	>3600
4	0.006131	8.05663	>3600
9	0.00342	3.54469	>3600
16	0.002497	1.84915	3011.21312
25	0.002222	1.05982	1643.23115