Data Sciences
Project 3

Mugdha Khade

UFID: 98907680

Task 1: PageRank Using Open MP

Write a parallel PageRank program using OpenMP.
:

Compilation Steps

- `tar –zxvf Project3_Khade_Mugdha.tar ./`
- `cd Project3_Khade_Mugdha`
- `cd Task1/src/`
- `g++ -fopenmp myPageRank.cpp –o myPageRank.o`
- `./PageRank.o`

Implementation Steps:

<u>Algorithm</u>

1. First read the file and store it in a buffer.
2. Make a matrix of size n by n and store all the values in it. We consider undirected graph hence a to b if has a link then b to a also has a link.
3. Use the formula r = Mr. Do this for a number of iterations.
4. Set a threshold value. Determine the number of iterations to converge.

<u>Important :</u>
Threshold : defined as **(**new rank vector – previous rank vector) <= 10^{-7}
Dampening Factor = 0.85;

Result:
Successfully implemented the open MP to run parallel threads

Serial Results:
0.00001
user 14.85
sys 1.26
CPU usg: 178%

0.0000001 :
user 63.41
sys 0.39
CPU usg 167%

Observation Table:

Number of Threads	Convergence Threshold	Number of Iterations	Convergence Time
5	10^{-5}	12	user 2.4 sys 3.4 CPU usg 131%
10	10^{-5}	12	user 2.3 sys 4.7 CPU usg 126%
15	10^{-5}	12	user 2.42 sys 0.38 CPU usg 124%
20	10^{-5}	12	user 2.65 sys 0.11 CPU usg 126%
5	10^{-7}	12	user 2.39 sys 0.36 CPU usg 156%
10	10^{-7}	12	user 2.38 sys 0.35 CPU usg 158%
15	10^{-7}	12	user 2.50 sys 0.2 CPU usg 159%
20	10^{-7}	12	user 2.7 sys 0.90 CPU usg 160%

Observations

- Parallel code works faster than the serial code for reasonably optimal values of number of threads,
- as the number of threads increase, the task of distributing and gathering back information from threads adds extra overhead.

Task 2

Compile the File

- `tar –zxvf Project3_Khade_Mugdha.tar ./`
- `cd Project3_Khade_Mugdha`
- `cd Task2/src/`
- `mpiCC myReducer.cpp -o myReducer`
- `mpirun -n 5 Reducer`

Implementation Steps:

Master's Job

1. Read the file to get the value of N;
2. Read the file make chunks of it and send it across to the slaves using a blocked call.
3. Wait for the processes to get back with the reduced data;
4. Write it to a file

Slaves Jobs:

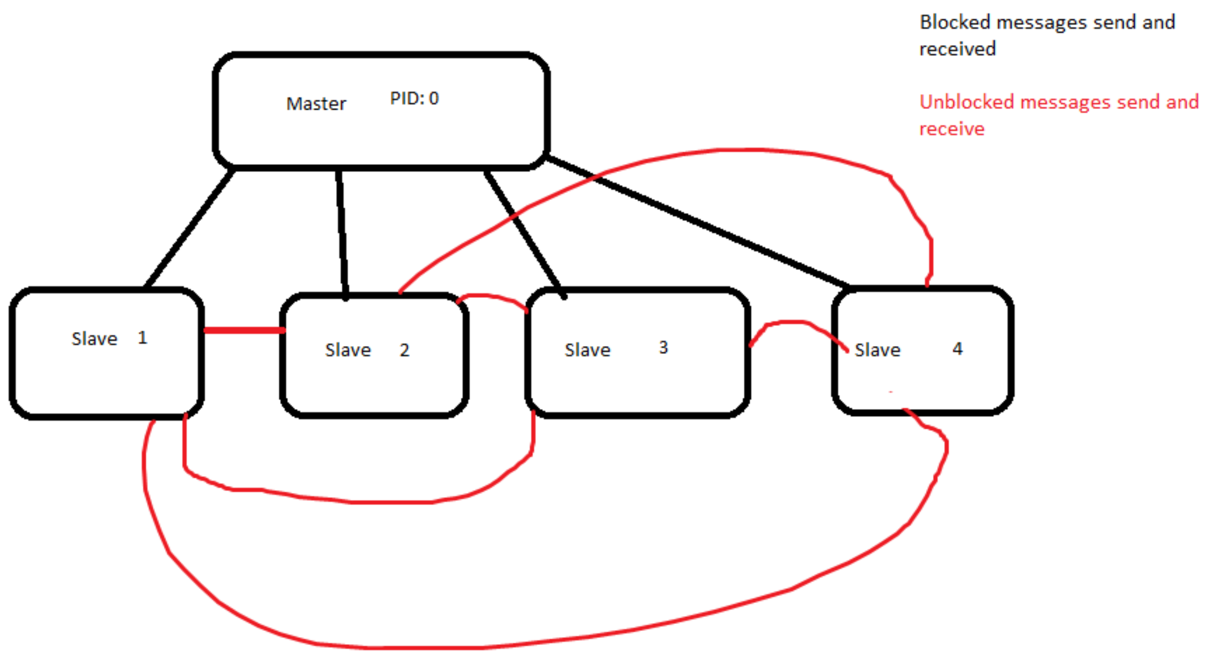
1. Get the chunk of that and perform the modulo operation on the key.
2. Send the keys to the respective process
3. Perform local reduction and send this data back to the Master

   Blocked Calls:
   Master to Processes and back to the master

   Unblocked calls:
   Processes to eachother

Please refer to the Flowchart for flow details:

FLOWCHART


Blocked messages send and received
Unblocked messages send and receive
Master PID: 0
Slave 1
Slave 2
Slave 3
Slave 4

We observer that by using the unblocked send and receive we can make parallel processes and hence increase the efficiency. But also the complexity of the system increases and we have the overhead of keep checking on the request and flag buffers.