HYBRID PARALLEL BFS USING MPI AND OPEN MP

Author – Aditya Nongmeikapam

UB Number - 50418825

Course – SEM708

Instructor – Russ Miller

CONTENTS

- Introduction to BFS (Breadth-First Search)
- Applications of BFS
- Parallel BFS Algorithm using vertex partitioning
- Results

Breath-First Search

- It is a graph traversing algorithm
- It traverses a tree/graph by exploring all of the nodes at the current level before moving on to the next level.
- A queue is used to keep track of unexplored child nodes.
- Runtime is O(N^2) where N is no. of vertices. (For adjacency matrix based approach).

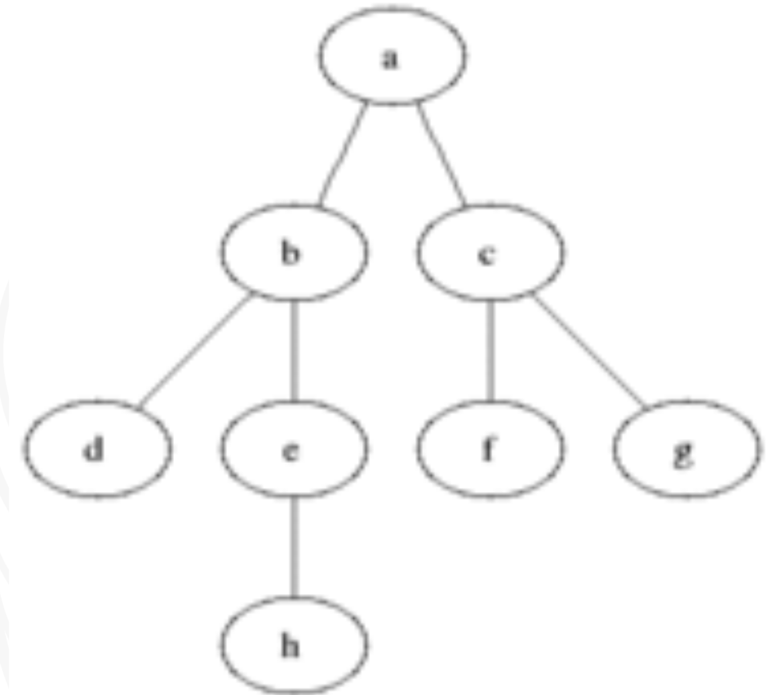


Image Source - Wikipedia

Applications of BFS

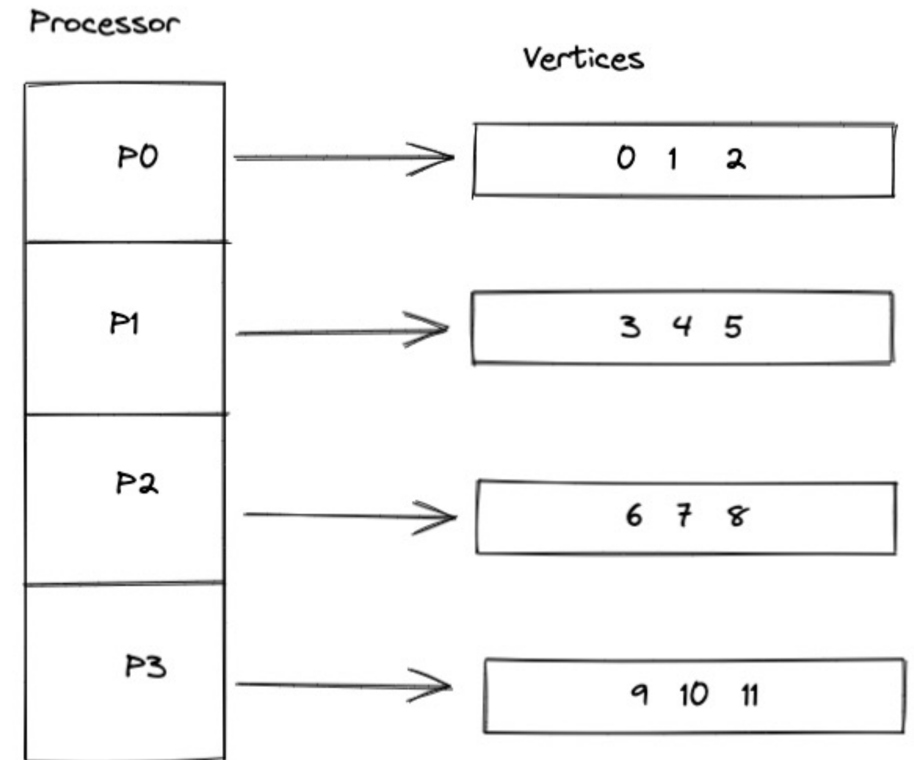
- Used to solve many graph theory problems like shortest path between two nodes for an unweighted graph.
- For computing the maximum flow in a flow network.
- In Social networking websites(e.g Linkedin), we can find the ith connection of a source person.
- Detect cycles in an undirected graph

Parallel BFS Algorithm Using Vertex Partition (1-D Partitioning)

- We do a level synchronous traversal.
- A set of vertices are assigned to a single processor. This processor handles all the computation involved for exploring the neighbors.
- All the processors calculate the neighbors using OMP Threads.
- All to All communication between all the processor to know the next set of frontier vertices.
- MPI is executed in one processor in a node.
- OpenMP threads are executed in cores inside that processor.

1-D(Vertex) Partition Parallel BFS Steps

- Chunk Size is calculated as vertices/processors. Each chunk of vertices are assigned to one processor.
- All the calculations for next neighbors are done by the same processor.
- The calculations done by a processor is multithreaded using Open MP.
- All to All communications between all the processor to sync the next frontier vertices using MPI.



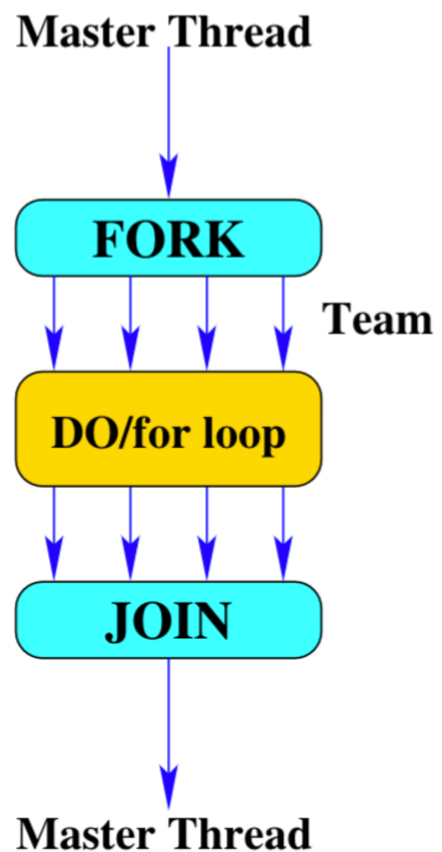
1-D Partition of Adjacency matrix

	0	1	2	3	4	5	6	7	8	9	10	11		
0														
1														
2														
3	0	1	0	1	1	1	0	1	0	0	1	1		
4	0	1	1	1	1	0	1	1	0	0	1	0	**MPI Processor - P1**	
5	0	0	0	1	0	1	0	1	1	0	0	0		
6														
7														
8														
9														
10														
11														

Shaded region of the adjacency matrix is the chunk which is available to MPI Processor P1.

The calculation of next frontier elements are done parallelly using OMP threads

OMP Loop Level Parallelization



Source – UB CCR Handouts

Hybrid BFS Pseudocode

Algorithm 2 Hybrid parallel BFS with vertex partitioning.

Input: $G(V, E)$, source vertex s.
Output: $d[1..n]$, where $d[v]$ gives the length of the shortest path from s to $v \in V$.

1: **for all** $v \in V$ **do**
2: $\quad d[v] \leftarrow \infty$
3: $level \leftarrow 1$, $FS \leftarrow \phi$, $NS \leftarrow \phi$
4: $op_s \leftarrow find_owner(s)$
5: **if** $op_s = rank$ **then**
6: $\quad$ push $s \rightarrow FS$
7: $\quad d[s] \leftarrow 0$
8: **for** $0 \leq j < p$ **do**
9: $\quad SendBuf_j \leftarrow \phi$ $\quad\triangleright$ p shared message buffers
10: $\quad RecvBuf_j \leftarrow \phi$ $\quad\triangleright$ for MPI communication
11: $\quad tBuf_{ij} \leftarrow \phi$ $\quad\triangleright$ thread-local stack for thread i
12: **while** $FS \neq \phi$ **do**
13: $\quad$ **for each** u in FS **in parallel do**
14: $\quad\quad$ **for each** neighbor v of u **do**
15: $\quad\quad\quad p_v \leftarrow find_owner(v)$
16: $\quad\quad\quad$ push $v \rightarrow tBuf_{ip_v}$
17: $\quad$ Thread Barrier
18: $\quad$ **for** $0 \leq j < p$ **do**
19: $\quad\quad$ Merge thread-local $tBuf_{ij}$'s **in parallel**, form $SendBuf_j$
20: $\quad$ Thread Barrier
21: $\quad$ **All-to-all collective step** with the master thread: Send data in $SendBuf$, aggregate newly-visited vertices into $RecvBuf$
22: $\quad$ Thread Barrier
23: $\quad$ **for each** u in $RecvBuf$ **in parallel do**
24: $\quad\quad$ **if** $d[u] = \infty$ **then**
25: $\quad\quad\quad d[u] \leftarrow level$
26: $\quad\quad\quad$ push $u \rightarrow NS_i$
27: $\quad$ Thread Barrier
28: $\quad FS \leftarrow \bigcup NS_i$ $\quad\triangleright$ thread-parallel
29: $\quad$ Thread Barrier

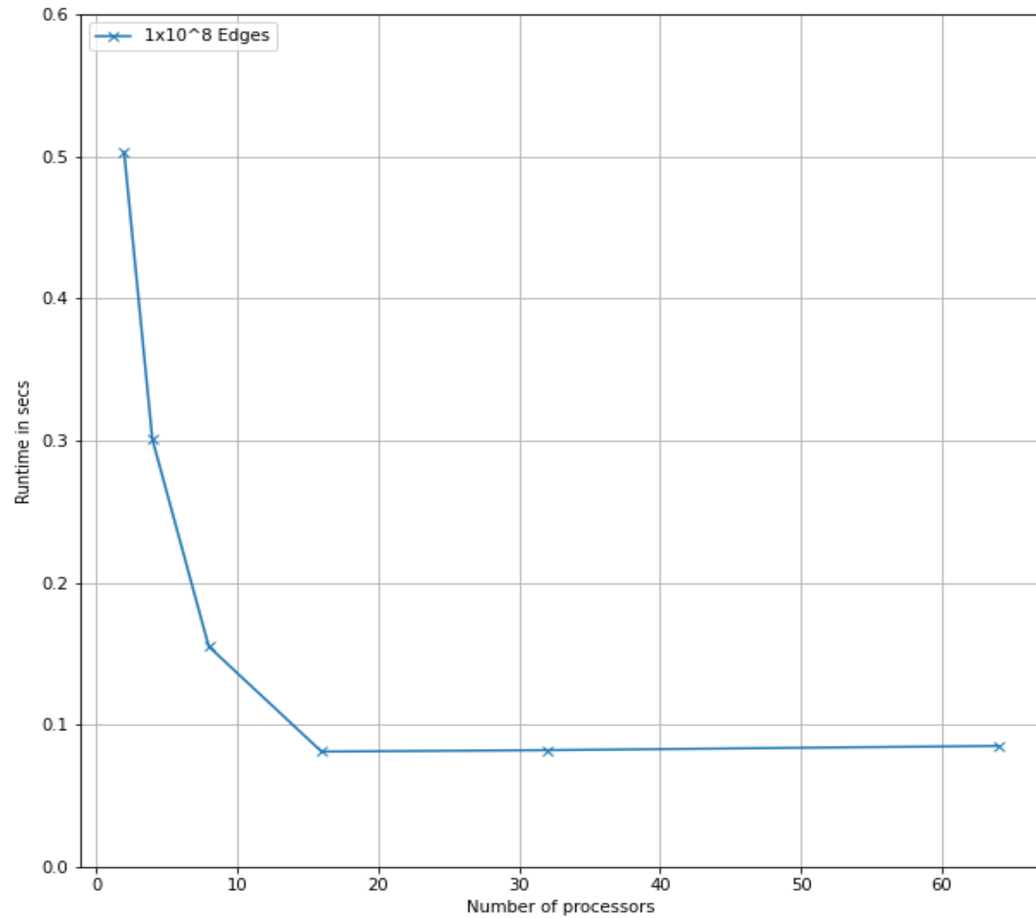
Source - https://people.eecs.berkeley.edu/~aydin/sc11_bfs.pdf

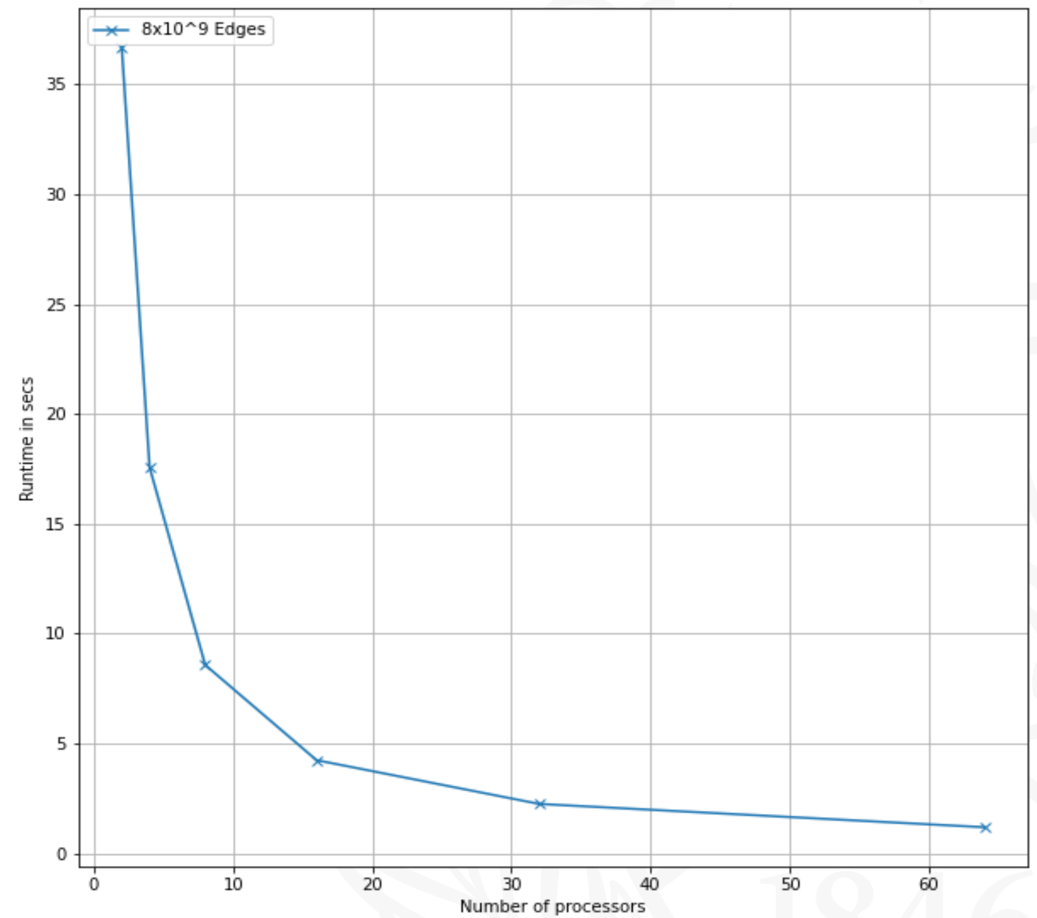
Results Pure MPI

Vertices	Approx. Edges	Seq	MPI 2	MPI 4	MPI 8	MPI 16	MPI 32	MPI 64
20000	132,000,000	0.812	0.503	0.301	0.155	0.081	0.082	0.085
40000	528,000,000	3.977	2.02	1.12	0.502	0.312	0.165	0.155
80000	2,122,000,000	16.65	8.56	4.15	2.08	1.05	0.55	0.28
160000	8,448,000,000	69.9	36.67	18.47	8.95	4.65	2.36	1.2

Runtime in secs
Density of the graph is 33%

Pure MPI Results



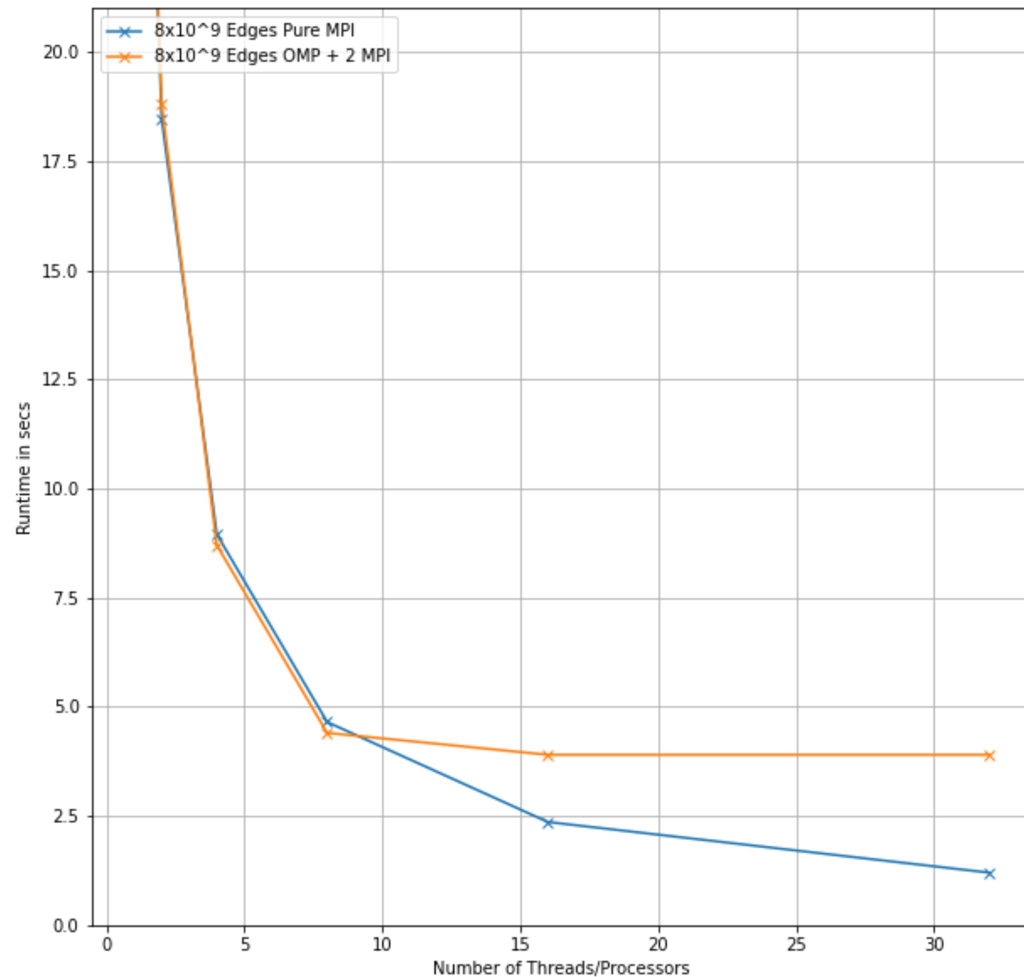


(MPI + OMP) Hybrid Results

Vertices	Approx. Edges	MPI Processor	Pure MPI	OMP 2	OMP 4	OMP 8	OMP 16	OMP 32
10000	33000000	2	0.187	0.103	0.1	0.11	0.12	1.7
160000	8,448,000,000	2	36.67	18.81	8.7	4.4	3.9	3.9
160000	8,448,000,000	4	18.57	9.31	4.73	2.28	2.1	2.1

Runtime in secs

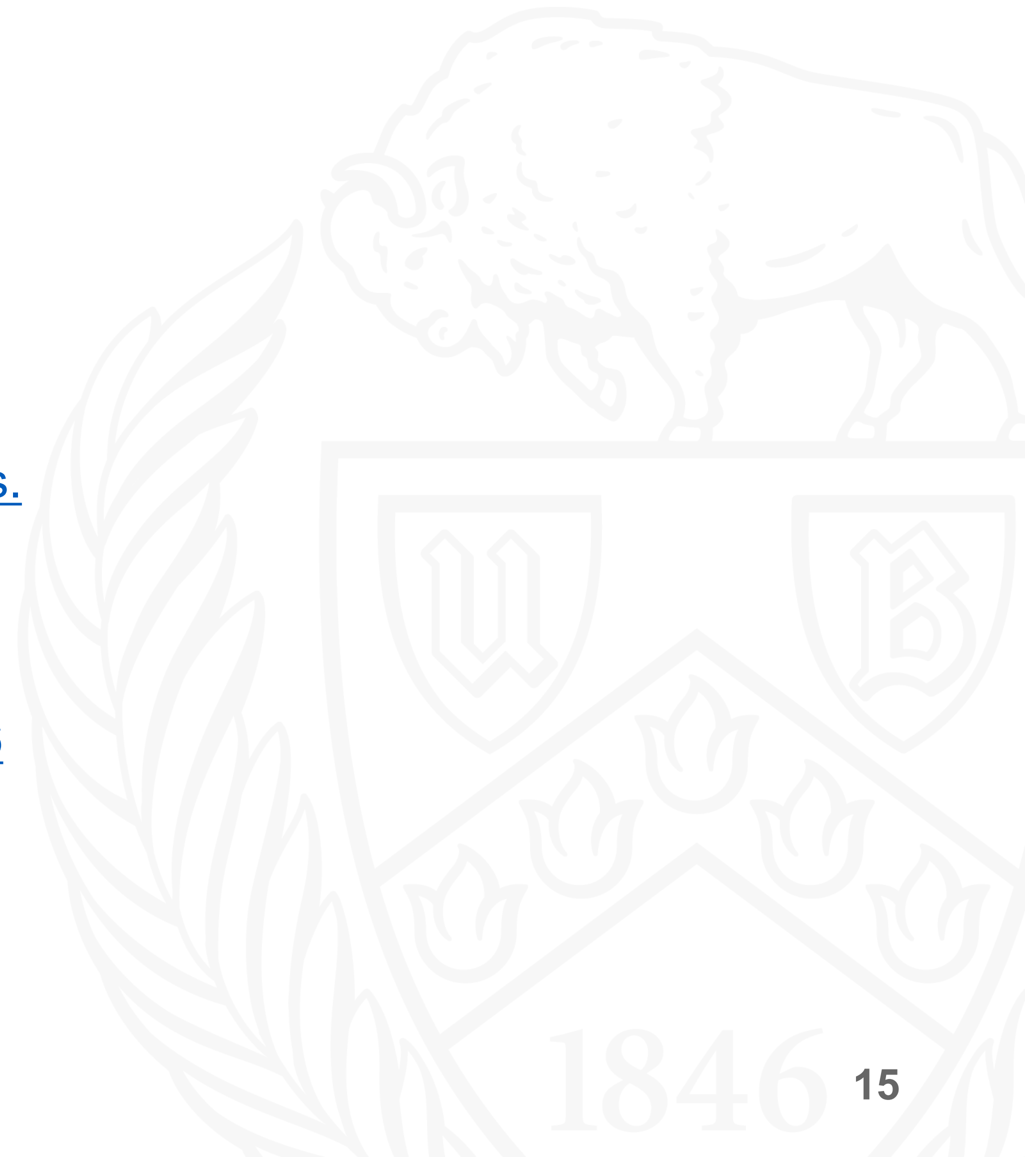
Hybrid(OMP + MPI) vs Pure MPI



Observations

- With Pure MPI we observe that for smaller input data after 16 MPI nodes we don’t see any gain in speed up. Whereas for larger input data we see that there is consistent gain in speedup even for 64 MPI processor.
- For Hybrid (OMP + MPI) solution, There is a minor improvement in runtime when compared to Pure MPI if we use less then 16 OMP cores.

References



- https://people.eecs.berkeley.edu/~aydin/sc11_bfs.pdf[Parallel Breadth-First Search on Distributed Memory Systems]
- https://en.wikipedia.org/wiki/Parallel_breadth-first_search
- https://janth.home.xs4all.nl/MPIcourse/PDF/05_MPI_Collectives.pdf
- https://slurm.schedmd.com/
- https://ubccr.freshdesk.com/support/solutions/articles/13000026245-tutorials-workshops-and-training-documents