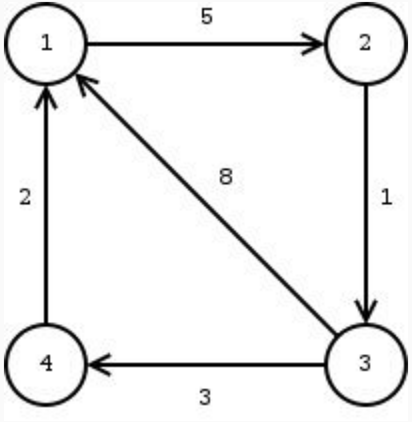
Scaling graph algorithm

(All Pair Shortest Path)

Project Members:-
Jatin Dev-18111027
Swapnil Raykar-18111078

Introduction

- All Pair shortest path find shortest path between every pair of vertices in a graph.

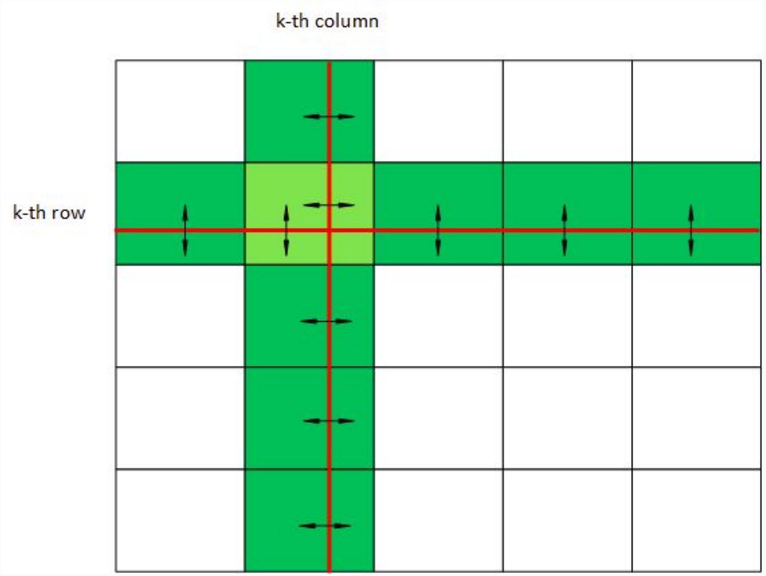


- Applications:
  - Shortest paths in directed graphs
  - Optimal routing problem
  - Transitive Closure
  - Kleene's algorithm

- Parallelizing APSP problem can save lots of time.

Related Work

- **Speeding up APSP**
    - **Checkerboard:** The cost matrix P is divided into equal parts of size (n/ p)X(n/ p) , and each is allocated to a different processor.



Related Work

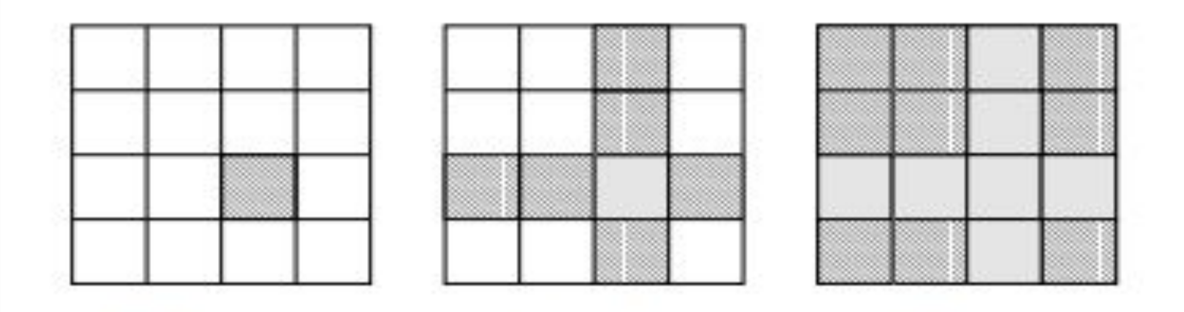
- **Speeding up APSP**
  - **Checkerboard:** The cost matrix P is divided into equal parts of size (n/ p)X(n/ p) , and each is allocated to a different processor.
  - **Pipelined Checkerboard**: Remove the synchronisation step and each process starts the computation as soon as it has the data necessary to compute.
  - **Blocked version APSP**: makes better utilization of the cache.
  - Slight optimization on original APSP problem which gives $O(n^{2.4})$, $O((n^3)/\log n)$ algorithm.

Implementation

- We studied and implemented following algorithms:
    - Naive Checkerboard Version
    - Naive Parallel APSP
    - Blocked Parallel Version
    - Pipelined Blocked Parallel version

Implementation

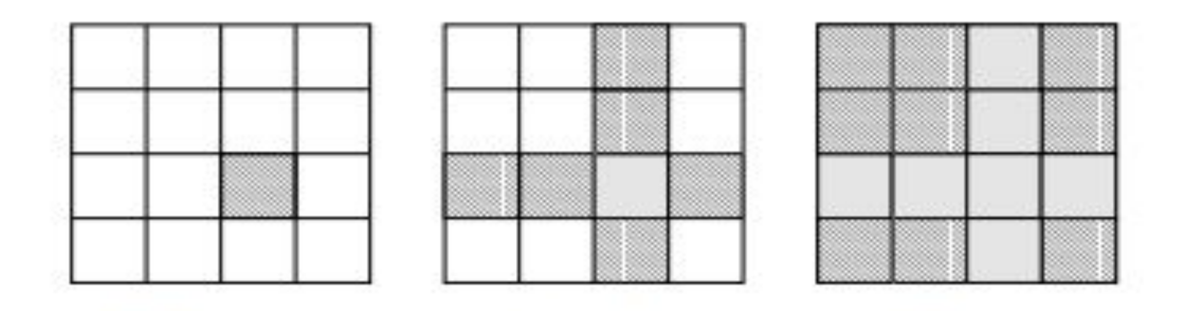
- **Blocked Parallel Version:**



First pivot element**(2,2)** compute its own block then it send it to corresponding column(**2**) and row**(2)** blocks. After their computation rest of the blocks are computed

Implementation

- **Blocked Parallel Version:**



First pivot element**(2,2)** compute its own block then it send it to corresponding column and row blocks. After their computation rest of the blocks are computed

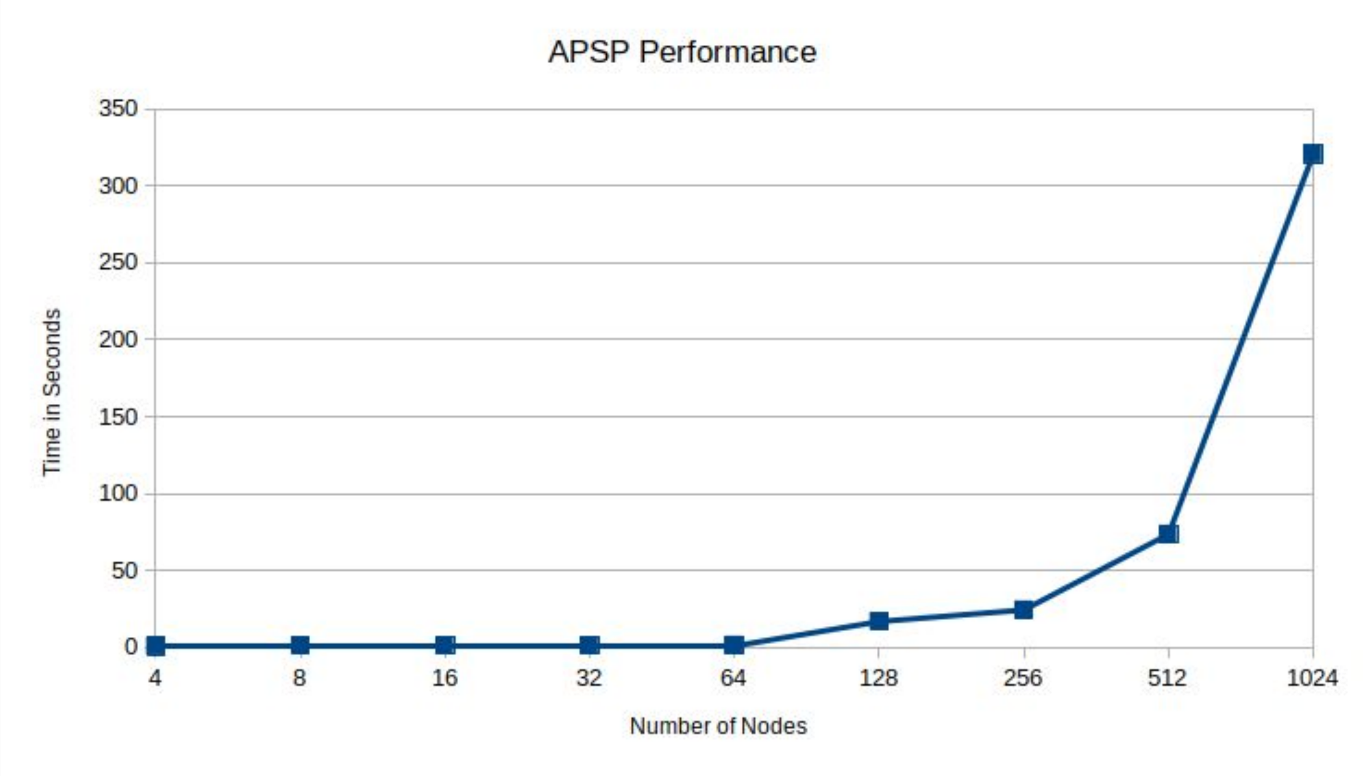
- **Pipelined Blocked Version:** Removed barrier. Pivot block for the next iteration can't be blocked due to other processes.

Implementation

- **Input Graphs**
  - **Random Graph Generator**
    python program
  - **SNAP(Stanford Network Analysis Project) Graphs**
    C program to generate the graph matrix from the benchmark

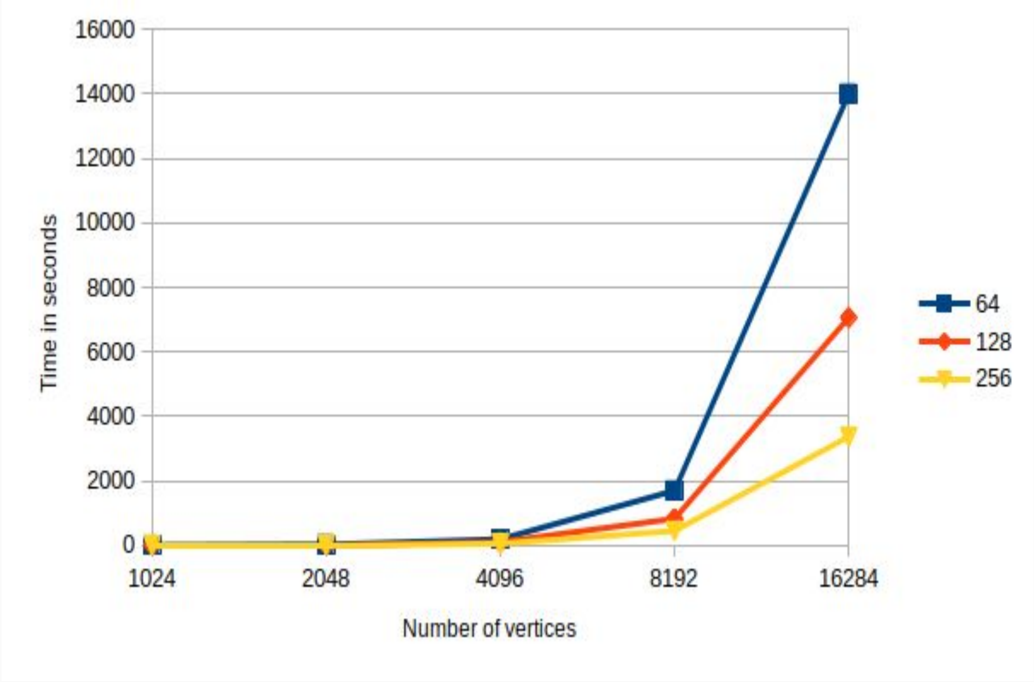
Experiments

Naive APSP



Experiments

Blocked Parallel Version - CSE Cluster

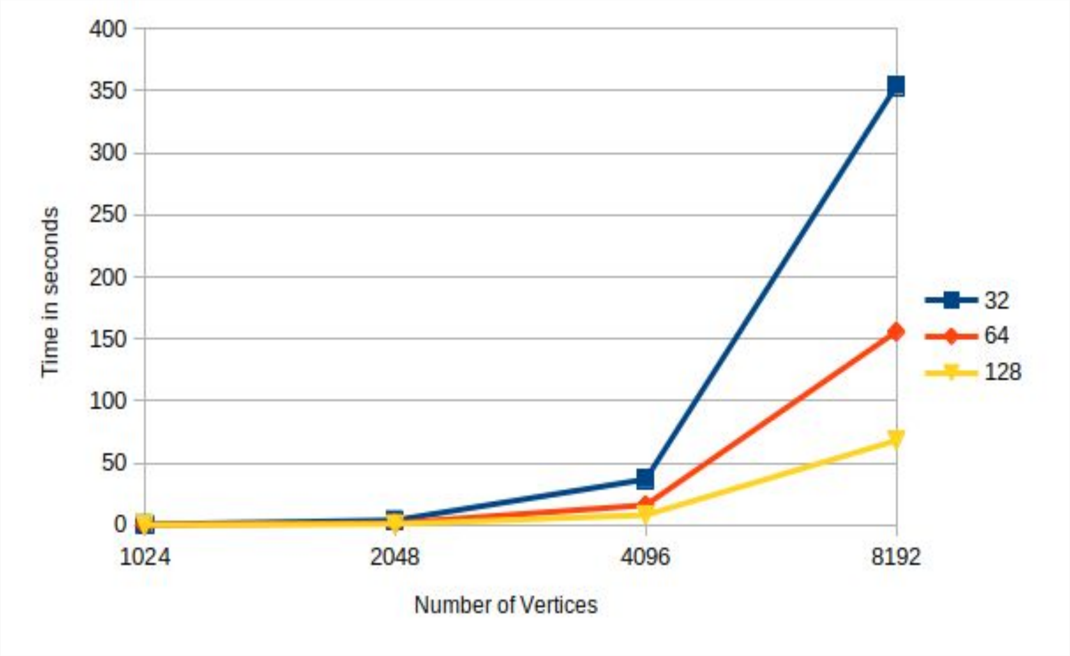


Observations:

- Time decreases ~50% with increase number of processes
- Decrease becomes more significant as graph size increases

Experiments

Blocked Parallel Version - HPC 2010

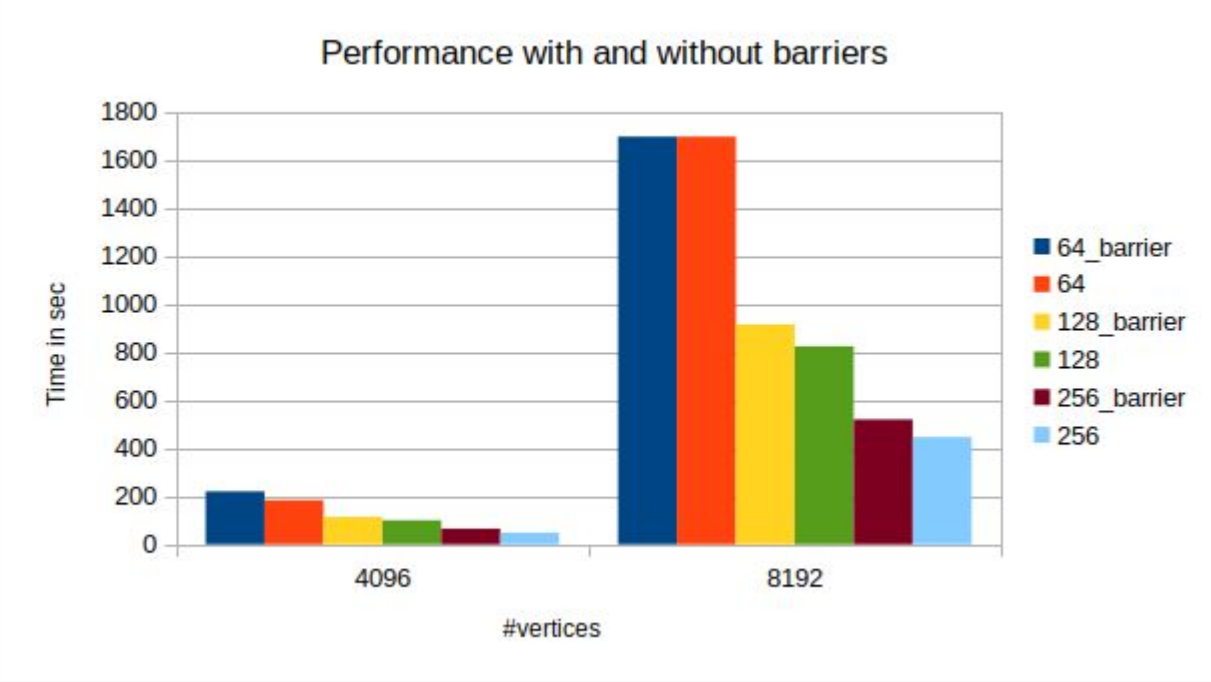


Observations:

- Time decreases with increase number of processes, becomes more significant with big graphs
- Less time required compared to CSE cluster

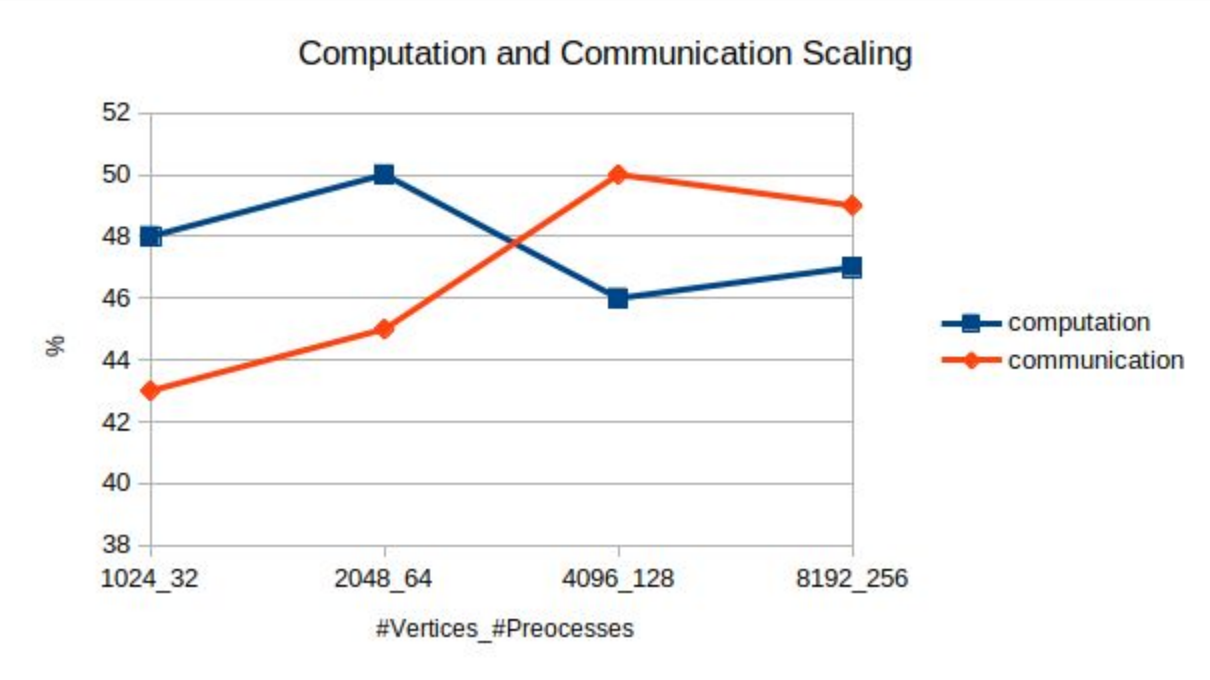
Performance Analysis

Pipelined Blocked Parallel Version



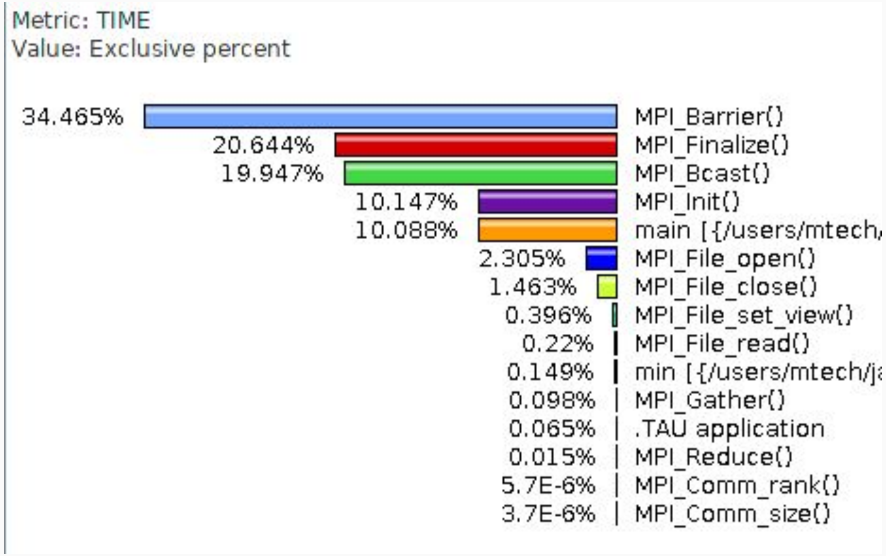
Performance Analysis

- TAU

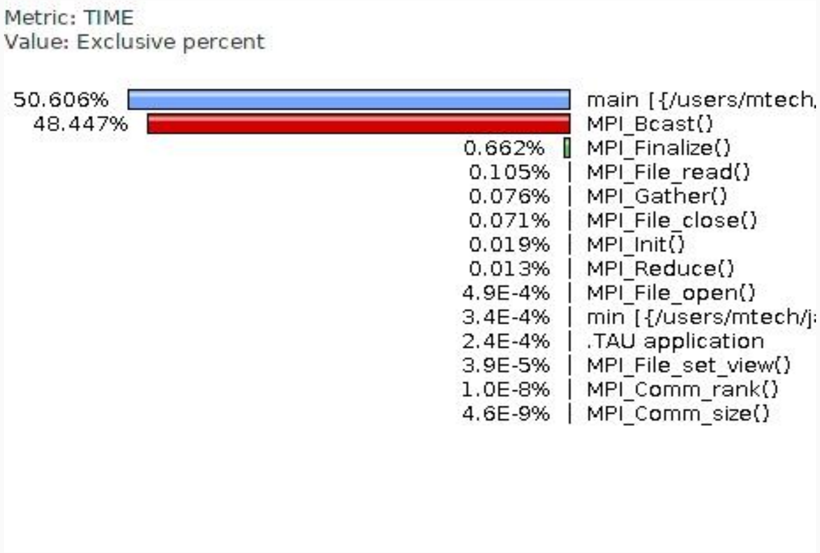


Performance Analysis

Blocked Parallel Version



Pipelined Blocked Parallel Version



Profiling of 16384 vertices with 128 processes

Future work

- Assignment and placement of nodes
- Distribute load evenly amongst the nodes during a given iteration ,for example pivot node remains idle for most part of the iteration.
- Overlapping computation and communication,such that data of the next iteration becomes ready at the start of the previous iteration.

THANK YOU