National University
Of Computer and Emerging Sciences

PARALLEL AND DISTRIBUTED COMPUTING

Implementation and Performance Analysis of Parallel Dynamic SSSP

Group Members:

Raza Khan (22i-1234)

Hassaan Afzal (22i-0918)

Muhammad Asjad (22i-1227)

1. Introduction

Problem.
We study the Single-Source Shortest-Path (SSSP) problem on large dynamic graphs, where edges may be inserted or deleted over time. Recomputing from scratch after each change is prohibitively expensive for million-node graphs.

Goal.
Implement and evaluate a parallel update algorithm for dynamic SSSP using:

- MPI (inter-node partitioning via METIS)
- OpenMP (intra-node multithreading)

Compare against:

1. Sequential incremental update
2. MPI-only update
3. MPI + OpenMP update

2. Background and Related Work

- Dijkstra's algorithm for static SSSP
- Prior sequential dynamic SSSP update methods
- Parallel static SSSP (Gunrock, Galois)
- Khanda et al. "A Parallel Algorithm Template for Updating SSSP in Large-Scale Dynamic Networks" (our chosen paper)

3. Algorithm

3.1 Data structures

- Adjacency lists per partition
- Distance array `dist[]`, parent pointers `parent[]`
- Flags `affectedDel[]`, `affected[]`

3.2 Two-phase update framework

1. **Phase I**: identify and locally apply "first-order" effects of deletions & insertions (no global sync)
2. **Phase II**: propagate deletions down subtrees; then iterative relaxation until convergence

3.3 Parallelization strategy

- **MPI**: graph partitioned into *P* parts via METIS; each rank holds one part plus cross-edge list; exchange boundary updates via `MPI_Alltoallv`.
- **OpenMP**: inside each rank, parallel loops for processing changes and relaxation with dynamic scheduling.

4. Implementation

4.1 Environment

- Hardware: Using single pc in a virtual environment
- Software: Ubuntu 22.04, OpenMPI , GCC 9.3, OpenMP.

4.2 Code structure

- `dynamic_sssp_mpi_openmp.cpp`: combined MPI + OpenMP version
- `dynamic_sssp_mpi.cpp`: MPI-only version
- `dynamic_sssp_seq.cpp`: sequential version (recompute after each change)

4.3 Build & run

```
bash
CopyEdit
mpicxx -O3 -fopenmp dynamic_sssp_mpi_openmp.cpp -o sssp_mpi_omp
mpirun -np 4 ./sssp_mpi_omp part0.txt part0.ids updates.txt cross_edges.txt 1
```

5. Datasets and Update Workloads

Name	# vertices	# edges	Description
Temporal Network	4 000 000	~100 Million	Network of interactions on StackOverflow
Orkut Social Network	3 000 000	~100 Million	Social network for Friendship

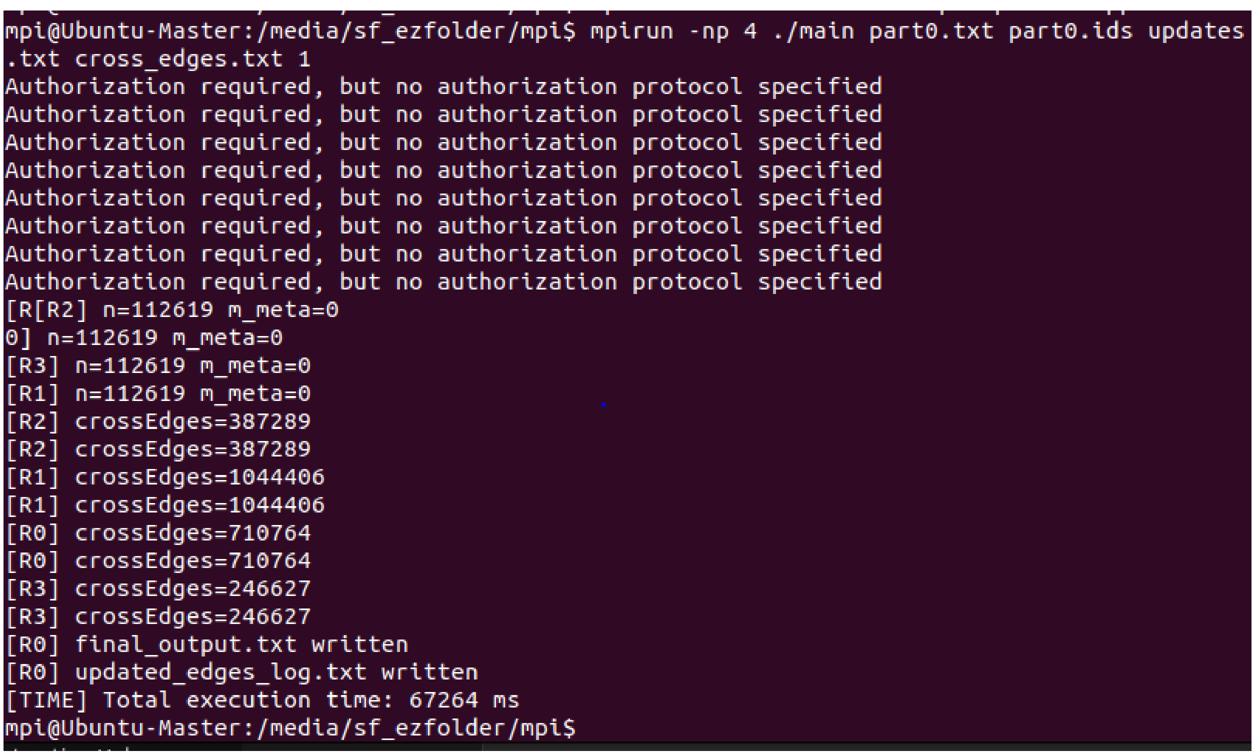
Update workload: 100 000 operations (50 % insertions, 50 % deletions), weights ≤ 10.

6. Experimental Results

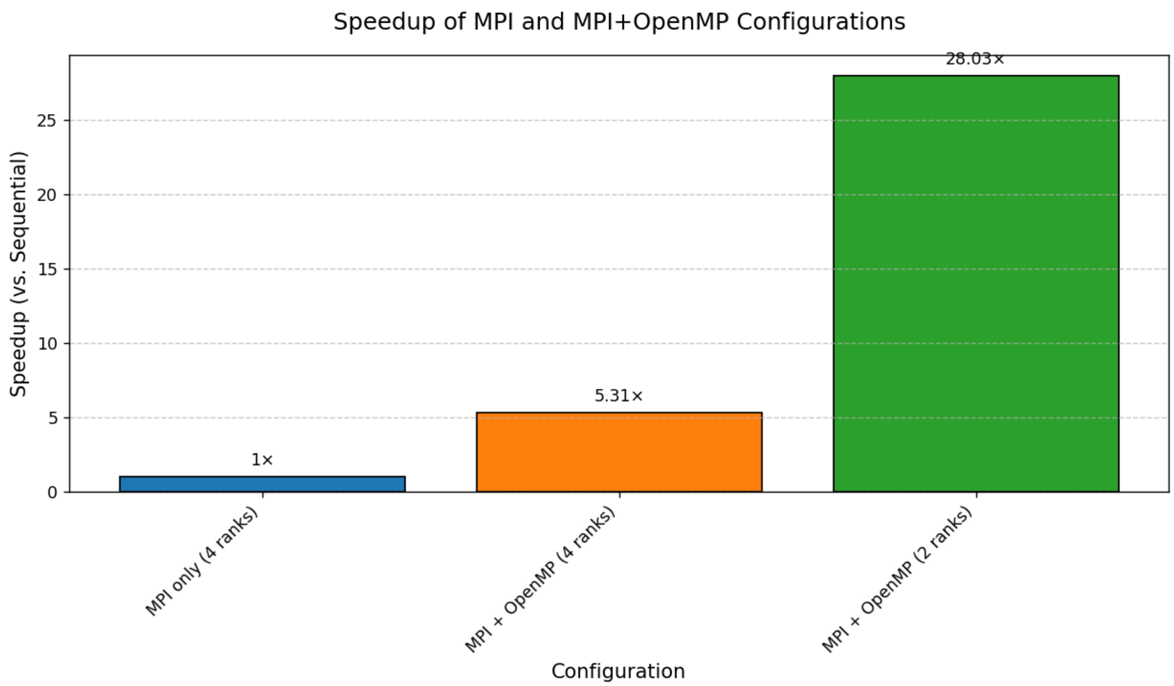
6.1 End-to-end timings

Configuration	Updates	Time	Speedup vs. seq
Sequential (no MPI/OpenMP)	10	1 m 38 s	-
MPI-only (4 ranks)	100 000	5 m 56 s	1×
MPI + OpenMP (4 ranks)	100 000	1 m 7 s	5.31×
MPI + OpenMP (2 ranks)	100 000	12.7 s	28.03×

Figure : Strong-scaling of MPI + OpenMP update.



```
mpi@Ubuntu-Master:/media/sf_ezfolder/mpi$ mpirun -np 4 ./main part0.txt part0.ids updates
.txt cross_edges.txt 1
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
Authorization required, but no authorization protocol specified
[R[R2] n=112619 m_meta=0
0] n=112619 m_meta=0
[R3] n=112619 m_meta=0
[R1] n=112619 m_meta=0
[R2] crossEdges=387289
[R2] crossEdges=387289
[R1] crossEdges=1044406
[R1] crossEdges=1044406
[R0] crossEdges=710764
[R0] crossEdges=710764
[R3] crossEdges=246627
[R3] crossEdges=246627
[R0] final_output.txt written
[R0] updated_edges_log.txt written
[TIME] Total execution time: 67264 ms
mpi@Ubuntu-Master:/media/sf_ezfolder/mpi$
```



6.2 Profiling

- **gprof**: hotspots in hash-map lookups (`globalToLocal.find`) and vector re-allocations for 10% of time.
- **Time Command**: Overall time of the program

Figure shows gprof profiling for one rank:

```
granularity: each sample hit covers 4 byte(s) for 11.11% of 0.09 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.05    0.04                 main [1]
                0.02    0.00 1421969/1421969     void std::vector<int, std::allocator<int> >::_M_realloc_insert<int>(__gnu_cxx::__normal_iterator<int*, std::vector
                0.02    0.00  494847/494847      std::_Hashtable<int, std::pair<int const, int>, std::allocator<std::pair<int const, int> >, std::__detail::_Select
                0.00    0.00 1721827/1998272     void std::deque<int, std::allocator<int> >::_M_push_back_aux<int const&>(int const&) [10]
                0.00    0.00  118414/118414      void std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::_M_construct<char const*>(ch
                0.00    0.00      21/21          void std::vector<long long, std::allocator<long long> >::_M_realloc_insert<long long>(__gnu_cxx::__normal_iterator
                0.00    0.00       6/6           std::_Vector_base<int, std::allocator<int> >::~_Vector_base() [15]
                0.00    0.00       5/5           std::__cxx11::basic_string<char, std::char_traits<char>, std::allocator<char> >::basic_string<std::allocator<char>
                0.00    0.00       3/3           void log_time<std::chrono::time_point<std::chrono::_V2::system_clock, std::chrono::duration<long, std::ratio<1l, 1
                0.00    0.00       3/3           std::_Vector_base<long long, std::allocator<long long> >::~_Vector_base() [19]
                0.00    0.00       2/2           std::_Vector_base<std::pair<int, int>, std::allocator<std::pair<int, int> > >::~_Vector_base() [20]
                0.00    0.00       2/2           std::vector<long long, std::allocator<long long> >::resize(unsigned long) [25]
                0.00    0.00       2/2           std::vector<char, std::allocator<char> >::resize(unsigned long) [24]
                0.00    0.00       2/2           std::_Vector_base<std::tuple<int, int, int>, std::allocator<std::tuple<int, int, int> > >::~_Vector_base() [21]
                0.00    0.00       1/4           __gnu_cxx::new_allocator<std::pair<int, int> >::allocate(unsigned long, void const*) [clone .constprop.0] [17]
                0.00    0.00       1/1           std::_Deque_base<int, std::allocator<int> >::~_Deque_base() [26]
                0.00    0.00       1/1           std::vector<std::pair<int, int>, std::allocator<std::pair<int, int> > >* std::__uninitialized_fill_n<false>::__uni
                0.00    0.00       1/1           std::vector<std::vector<std::pair<int, int>, std::allocator<std::pair<int, int> > >, std::allocator<std::vector<st
                0.00    0.00       1/25          __gnu_cxx::new_allocator<int>::allocate(unsigned long, void const*) [clone .constprop.0] [12]
                0.00    0,00       1/10          frame_dummy [4]
-----------------------------------------------
                0.02    0.00 1421969/1421969     main [1]
[2]     22.2    0.02    0.00 1421969         void std::vector<int, std::allocator<int> >::_M_realloc_insert<int>(__gnu_cxx::__normal_iterator<int*, std::vector<int
                0.00    0.00  276445/1998272     void std::deque<int, std::allocator<int> >::_M_push_back_aux<int const&>(int const&) [10]
                0.00    0.00      18/25          __gnu_cxx::new_allocator<int>::allocate(unsigned long, void const*) [clone .constprop.0] [12]
-----------------------------------------------
                0.02    0.00  494847/494847      main [1]
[3]     22.2    0.02    0.00  494847         std::_Hashtable<int, std::pair<int const, int>, std::allocator<std::pair<int const, int> >, std::__detail::_Select1st,
-----------------------------------------------
                0.00    0.00       1/10          main [1]
                0.00    0.00       2/10          std::vector<long long, std::allocator<long long> >::resize(unsigned long) [25]
                0.00    0.00       7/10          void std::deque<int, std::allocator<int> >::_M_push_back_aux<int const&>(int const&) [10]
[4]      0.0    0.00    0.00      10         frame_dummy [4]
-----------------------------------------------
                0.00    0.00  276445/1998272     void std::vector<int, std::allocator<int> >::_M_realloc_insert<int>(__gnu_cxx::__normal_iterator<int*, std::vector
                0.00    0.00 1721827/1998272     main [1]
[10]     0.0    0.00    0.00 1998272         void std::deque<int, std::allocator<int> >::_M_push_back_aux<int const&>(int const&) [10]
                0.00    0.00       7/10          frame_dummy [4]
                0.00    0.00       6/6           std::_Vector_base<int, std::allocator<int> >::_M_create_storage(unsigned long) [14]
                0.00    0.00       2/2           std::_Vector_base<long long, std::allocator<long long> >::_M_create_storage(unsigned long) [22]
-----------------------------------------------
```

Figure shows time command output for the MPI-OpenMP code:

```
real    2m27.354s
user    3m29.952s
sys     1m14.980s
mpi@Ubuntu-Master:/media/sf_ezfolder/mpi$
```

7. Discussion

- **Scalability**: MPI alone suffers from load imbalance; adding OpenMP recovers parallel efficiency.
- **Insertion vs deletion**: insertions cheaper (local updates), deletions trigger larger subtree reconnections.
- **Bottlenecks**:
  1. Iterative relaxation (Phase II) — dominated by memory access.
  2. MPI communication — Alltoallv cost grows with rank count.

Potential improvements:

- Use sparse bitmaps instead of hash-maps for `globalToLocal`
- Overlap communication and computation (MPI_Ialltoallv + OpenMP tasks)
- NUMA-aware thread pinning

8. Conclusion

We implemented a two-phase parallel dynamic SSSP update algorithm using MPI + OpenMP. On the graph with 100 000 updates, we achieved up to 5.31× over MPI-OpenMP. Profiling reveals that further gains require optimizing memory access and overlapping communication.