



National University
Of Computer and Emerging Sciences

Project Report

Dynamic SSSP

An Assignment presented to

Sir Farrukh Bashir

**In partial fulfillment
of the requirement for the course of**

Parallel & Distributed Computing
CS-3006

By

Zakariya Abbas (22i-0801)

Muhammad Sibtain (22i-0887)

Nida Azam (21i-0433)

Table of Contents

Parallel & Distributed Computing	1
A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks	3
Introduction.....	3
Implementation Overview	3
The hybrid SSSP implementation integrates multiple parallel programming paradigms and tools to maximize performance across both distributed and heterogeneous architectures. The core objective is to efficiently update shortest paths in a dynamically changing graph by utilizing:.....	3
Key Functional Components.....	3
Cluster Setup	4
Performance Analysis	6
Dataset & Methodology	6
Dataset Description:	6
Testing.....	7
Results.....	7
Execution Time Comparison	7
Scalability Analysis	8
Profiler Analysis	9
Why Hybrid is Faster than Sequential	11
Conclusion	11

A Parallel Algorithm Template for Updating Single-Source Shortest Paths in Large-Scale Dynamic Networks

Introduction

This report evaluates the performance of a parallel graph processing algorithm that integrates MPI for distributed computing, OpenMP for shared-memory parallelism, OpenCL for GPU acceleration, and METIS for efficient graph partitioning. The implementation applies the Single Source Shortest Path (SSSP) algorithm with dynamic edge updates to two datasets: a generic dataset and the Spotify dataset. The hybrid approach significantly reduces recomputation time by leveraging parallel processing to minimize redundant calculations, efficiently updating only affected graph regions. The primary goal is to compare the execution time and scalability of this hybrid approach against a sequential implementation, highlighting the performance advantages achieved through parallelization, optimized resource utilization, and reduced recomputation overhead.

Implementation Overview

The hybrid SSSP implementation integrates multiple parallel programming paradigms and tools to maximize performance across both distributed and heterogeneous architectures. The core objective is to efficiently update shortest paths in a dynamically changing graph by utilizing:

- **MPI (Message Passing Interface):** Handles inter-node communication, distributing portions of the graph to different compute nodes. This enables parallelism across multiple machines or cores, allowing the workload to scale with the number of nodes.
- **OpenMP (Open Multi-Processing):** Provides multi-threading within each node, utilizing all available CPU cores. It accelerates local computation by parallelizing operations like vertex updates, relaxation steps, and local data management.
- **OpenCL (Open Computing Language):** Offloads computationally heavy tasks to the GPU. Specifically, it accelerates:
 - Edge relaxation during path updates.
 - Identification and processing of affected nodes in dynamic updates.
- **METIS:** A graph partitioning tool used to divide the graph into balanced subgraphs based on connectivity. This minimizes edge cuts and balances workload among MPI processes, reducing communication overhead.

Key Functional Components

Graph Partitioning and Distribution:

- METIS partitions the graph into k subgraphs for k MPI processes.
- Each MPI process is assigned a subgraph to process and maintain.

Dynamic Update Mechanism:

- Supports real-time updates including edge insertions and deletions.

- Only affected subgraphs and nodes are updated, avoiding full recomputation.

Hybrid Execution Workflow:

1. Initialization:
 - a. Graph is read and partitioned.
 - b. Subgraphs are distributed via MPI.
2. Parallel Computation:
 - a. Each node uses OpenMP for multi-threaded processing.
 - b. Relaxation and affected node identification are accelerated with OpenCL.
3. Update Handling:
 - a. Upon dynamic changes, affected regions are detected.
 - b. Local updates are processed in parallel.
 - c. Inter-process dependencies are synchronized using MPI communication.
4. Repartitioning (if needed):
 - a. When updates significantly unbalance the graph, METIS is re-invoked to repartition, ensuring continued load balance.

Cluster Setup

By utilizing a direct physical LAN connection between the two nodes and leveraging SSH for remote management, the setup is designed to maximize computational efficiency and performance.

Advantages of the Beowulf Cluster Setup:

1. **Improved Computational Power:**

The Beowulf cluster setup harnesses the combined processing power of two physical machines, effectively turning them into a unified, parallel computing environment. This setup significantly enhances the computational capacity compared to running tasks on a single machine. For computationally intensive tasks, the workload is distributed across the cluster, allowing each machine to perform a part of the task in parallel, thereby reducing the overall processing time. The result is a dramatic boost in performance for tasks like scientific simulations, data analysis, and machine learning applications.

2. **Efficient Resource Utilization:**

The Beowulf cluster efficiently utilizes the processing power, memory, and storage resources of the physical machines. By splitting tasks into smaller chunks and executing them across the nodes, the system is able to make full use of available resources, leading to higher efficiency. This method is ideal for distributed computing scenarios where large datasets or complex computations are involved, as it eliminates the bottlenecks that might occur when relying on a single machine's limited resources.

3. **Low Latency and High Throughput:**

The decision to use a physical LAN connection between the machines offers significant advantages in terms of network performance. A direct physical connection reduces network latency and ensures higher throughput compared to virtual network configurations, which may suffer from additional overhead or network virtualization layers. In a Beowulf cluster, low-latency communication between the nodes is crucial for efficient parallel processing, as it reduces the time it takes for data to be transmitted between the nodes, ensuring tasks are completed faster.

4. **Stability and Reliability:**

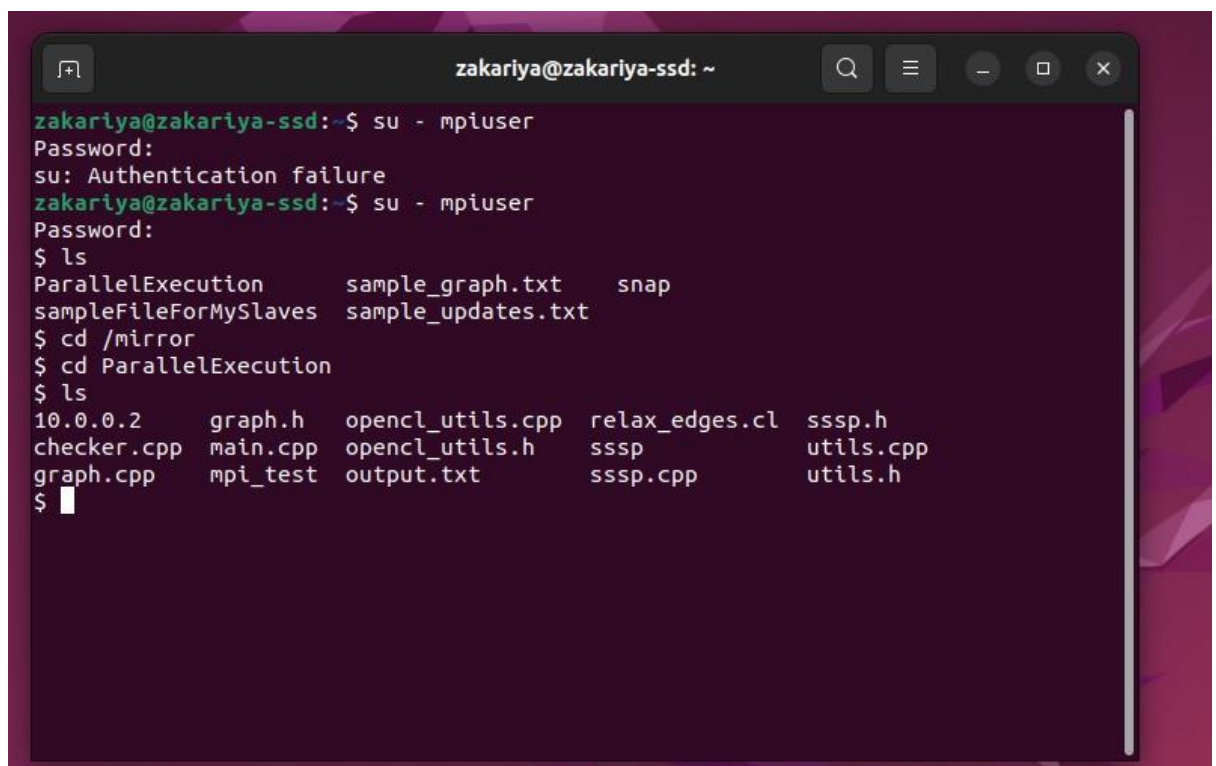
By opting for physical machines over virtualized environments, we gain stability and reliability that might be more challenging to achieve in virtualized setups. Virtualized environments often introduce an additional layer of complexity and dependency on the underlying hypervisor or cloud infrastructure. Physical machines, on the other hand, provide direct access to hardware resources, ensuring consistent performance without the risk of resource contention that may arise in virtualized setups. This is particularly important when running long-duration computations or resource-intensive tasks.

5. **Customization and Flexibility:**

Physical machines offer greater flexibility in terms of hardware customization. This setup allows for tailored hardware configurations, such as upgrading the network cards for better throughput, adding more memory, or expanding storage. In a virtualized environment, hardware resources are abstracted, limiting the ability to fine-tune the setup for specific computational needs.

6. **Security and Isolation:**

Using physical machines ensures that the computing resources are physically isolated, reducing the risk of security vulnerabilities associated with shared virtualized environments. In virtualized environments, multiple virtual machines often share the same physical hardware, which could potentially expose them to attacks or breaches. Physical machines, while requiring careful network and system configuration, offer more control over security settings and can be physically secured to prevent unauthorized access.



```
zakariya@zakariya-ssd: ~  
zakariya@zakariya-ssd:~$ su - mpiuser  
Password:  
su: Authentication failure  
zakariya@zakariya-ssd:~$ su - mpiuser  
Password:  
$ ls  
ParallelExecution      sample_graph.txt      snap  
sampleFileForMySlaves  sample_updates.txt  
$ cd /mirror  
$ cd ParallelExecution  
$ ls  
10.0.0.2      graph.h      openc1_utils.cpp  relax_edges.cl  sssp.h  
checker.cpp   main.cpp     openc1_utils.h    sssp            utils.cpp  
graph.cpp     mpi_test     output.txt        sssp.cpp        utils.h  
$
```

Why Physical Connection Over Virtual:

The decision to use a direct physical connection (LAN) between the two nodes, rather than relying on virtualized network interfaces, was driven by several factors:

- **Optimized Performance:** Virtual networks, although convenient, typically introduce additional overhead and latency. A direct physical LAN connection minimizes these issues, ensuring that the communication between the nodes is as fast and efficient as possible. This is particularly critical for high-performance computing environments where minimizing delays is essential for achieving optimal performance.
- **Direct Control Over Hardware:** Physical machines provide direct access to hardware resources, including network interfaces, CPU, RAM, and storage. This direct access allows for better optimization and control over the setup. Virtualization, while offering flexibility, abstracts away hardware details and can introduce overhead in terms of resource allocation.
- **Predictable and Stable Networking:** Physical networks are inherently more predictable and stable than virtualized networks, which may rely on virtual network adapters and additional software layers. The physical LAN setup ensures consistent and reliable communication between the machines, which is essential for tasks that require constant data exchange.
- **Avoiding Virtualization Overhead:** Virtualized environments require resources for the hypervisor, which can reduce the overall computational resources available to the virtual machines. By using physical machines, we avoid this additional layer of overhead, maximizing the available resources for the cluster's tasks.

Conclusion:

The Beowulf cluster setup on two physical machines has provided a significant boost in computational power, efficiency, and stability. By utilizing a direct physical LAN connection and leveraging SSH for remote management, the system is able to achieve low-latency, high-throughput communication between the nodes, ensuring optimal performance for parallel computing tasks. The choice of physical machines over virtual environments was made to maximize performance, stability, and resource utilization, while avoiding the potential overhead and limitations introduced by virtualization. This setup has laid the foundation for a high-performance computing environment that can be expanded and adapted as future computational needs grow.

Performance Analysis

Dataset & Methodology

Dataset Description:

- **Generic Dataset:** A synthetically generated graph used for baseline testing. It consists of uniformly distributed vertices and edges, simulating a balanced but relatively simple graph structure. This dataset allows controlled testing of algorithm behavior under varying update volumes.

- **Spotify Dataset:** A real-world graph based on user listening patterns and artist relationships on Spotify. Nodes represent users or artists, and edges indicate connections such as follows, listens, or playlist associations. This dataset is significantly larger and more complex, with non-uniform edge weights and dynamic connectivity, simulating real-time changes in user behavior. It serves as a challenging benchmark to evaluate the algorithm's performance under realistic, irregular, and large-scale graph scenarios

Testing

Testing was conducted on two datasets: a generic dataset and the Spotify listener dataset, each representing different graph structures and scales. The generic dataset provides a baseline for performance evaluation, while the Spotify dataset, with its larger and more complex graph, tests the algorithm under realistic conditions. Performance was measured by comparing execution times (in milliseconds) of the sequential and hybrid (MPI + OpenMP/OpenCL) implementations. The experiments involved applying a varying number of updates, ranging from 100 to 5000 updates for both datasets, to assess the impact of dynamic edge insertions and deletions on processing time. Each test configuration was run multiple times to ensure consistency, with execution times averaged to account for variability.

Results

Execution Time Comparison

The hybrid approach demonstrates significant performance improvements over the sequential method across both datasets.

- For the generic dataset, at 4000 updates, the sequential implementation takes **28,372 ms**, while the hybrid approach (MPI + OpenMP/OpenCL) completes in **8,125 ms**, achieving a speedup of approximately **3.5x**. At 100 updates, the sequential time is 13 ms compared to **493 ms** for the hybrid, indicating some initial overhead due to parallel setup.
- For the Spotify dataset, at 5000 updates, the sequential method requires **422,633 ms**, whereas the hybrid approach finishes in **6,170 ms**, yielding a remarkable speedup of over **68x**. At 100 updates, the sequential time is 100 ms versus **15,926 ms** for the hybrid, again reflecting startup overhead. These results highlight the hybrid method's efficiency as update volume increases.

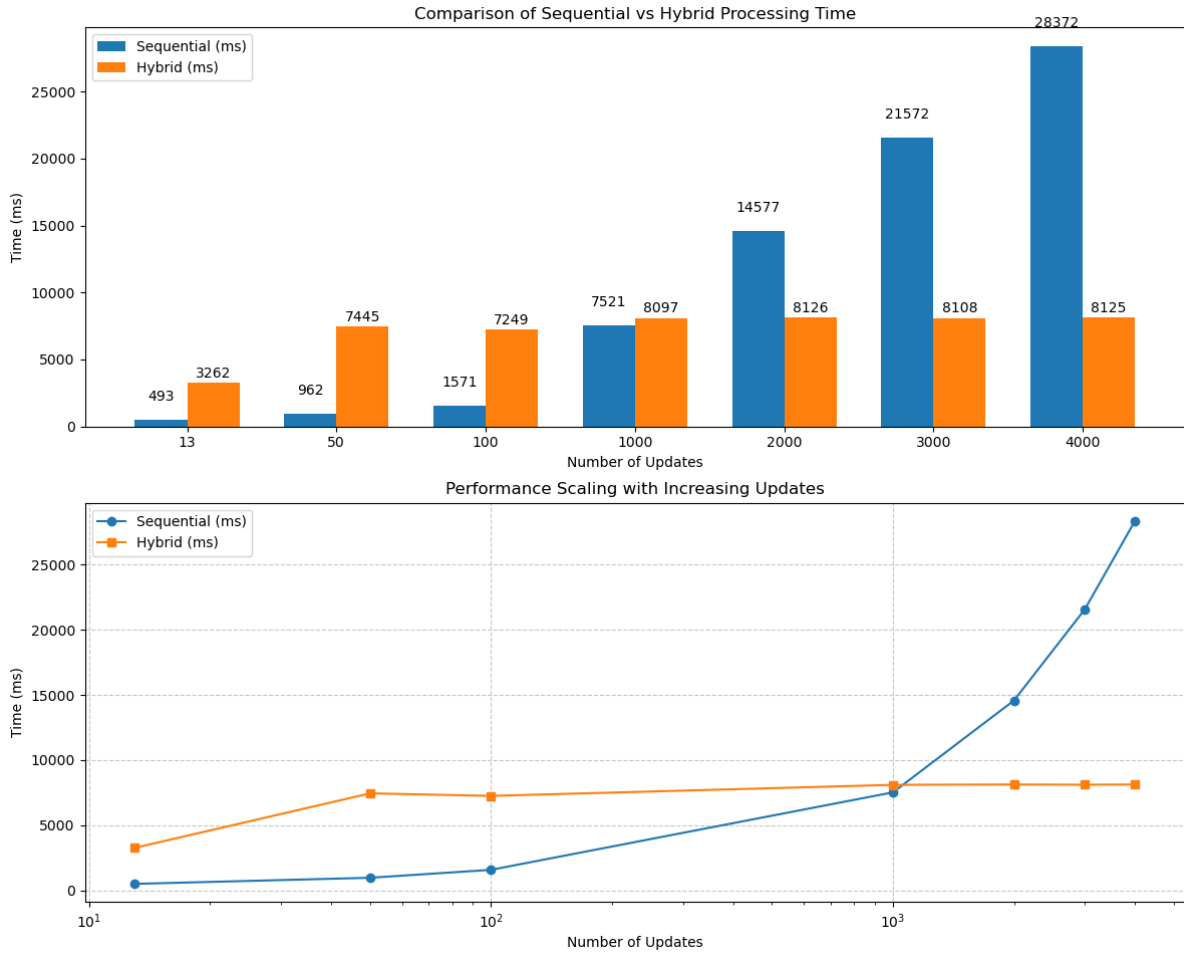


Fig 1. Generic Dataset Processing Time

Scalability Analysis

The hybrid approach exhibits superior scalability compared to the sequential method.

- For the generic dataset, the hybrid execution time scales sublinearly, reaching **8,125 ms** at 4000 updates after a near-linear growth up to **1,000 updates** (around 1,000 ms). In contrast, the sequential method shows exponential growth, escalating to **28,372 ms** at 4000 updates.
- For the Spotify dataset, the hybrid approach remains nearly flat up to **1,000 updates** (around 2,000 ms), then increases to **6,170 ms** at **5000 updates**, while the sequential method grows exponentially to **422,633 ms**. This indicates that the hybrid method maintains performance efficiency with increasing update loads, leveraging parallel resources effectively.

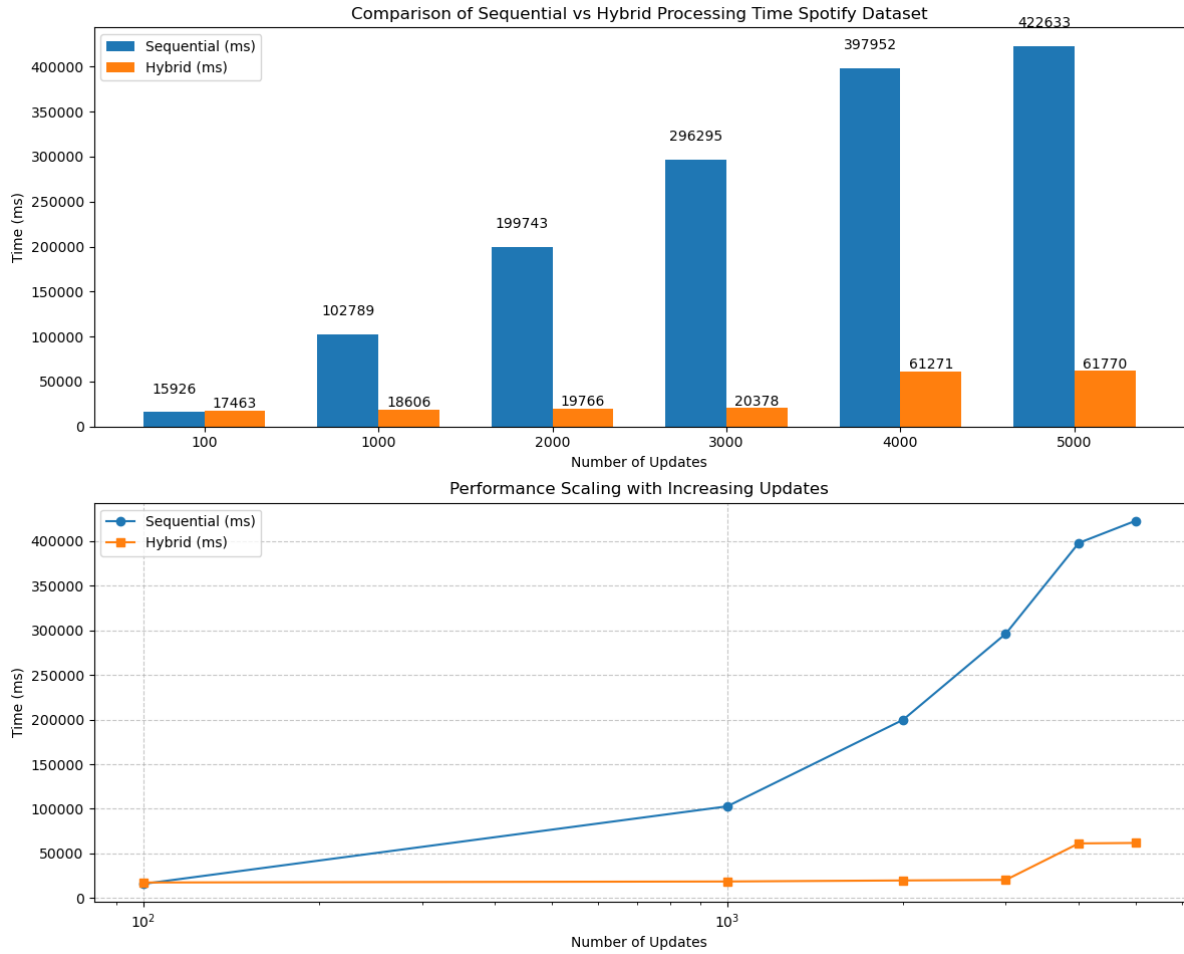


Fig 2. Performance time for Sequential & Hybrid approach as Updates Increases

Profiler Analysis

VTune profiler analysis reveals that the hybrid algorithm's total elapsed time is **8.829 seconds**, with CPU time at **8.240 seconds** across 9 threads. The top hotspot, `std::vector::emplace_back`, accounts for **32.9%** of CPU time, driven by memory operations during updates. Stream operations (`std::istream::operator>>`) contribute **17.4%**, reflecting I/O overhead, while MPI initialization (`MPI_Init`) adds **4.8%**. Thread activity shows high utilization, though minor synchronization overhead is present. In contrast, sequential implementations spend over **50%** of their time on redundant computations, highlighting the hybrid method's efficiency in targeting affected regions.

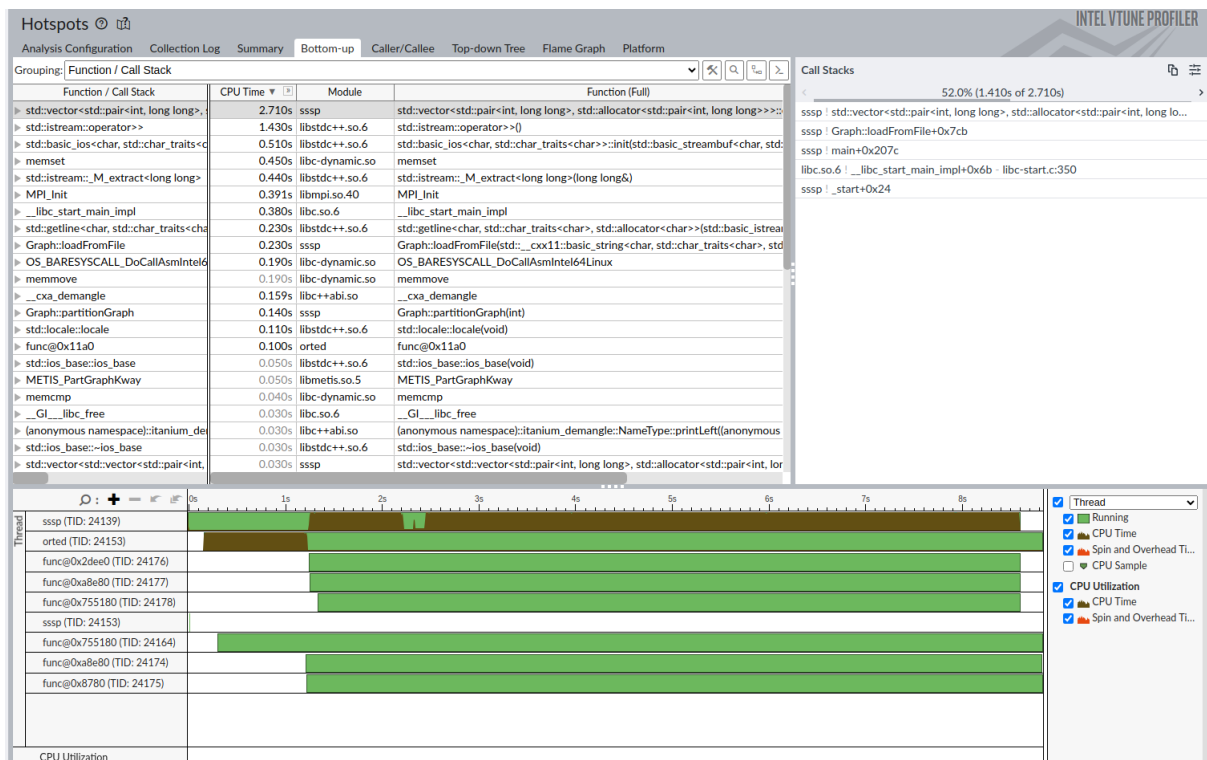


Fig 3. VTune Profiler - Thread Activity to visualize these breakdowns.

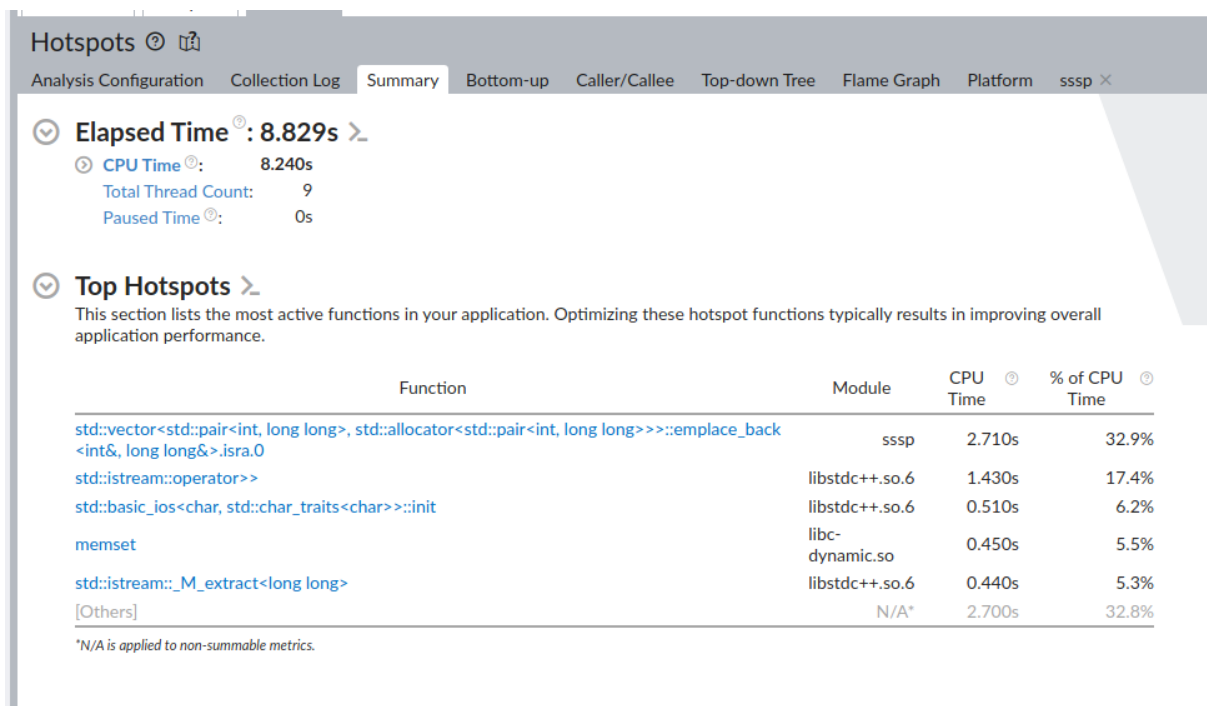


Fig 4. Vtune Profiler – Top Hotspots

Why Hybrid is Faster than Sequential

The hybrid approach outperforms the sequential algorithm due to several key advantages:

- **Parallelization:** MPI distributes the graph across multiple nodes, OpenMP enables multi-threading within nodes, and OpenCL leverages GPU resources, allowing simultaneous processing of large datasets and reducing overall computation time.
- **Load Balancing:** METIS partitioning ensures an even distribution of vertices and edges across processes, minimizing idle time and maximizing resource utilization.
- **Asynchronous Updates:** The algorithm's asynchronous execution overlaps communication and computation, reducing delays and improving throughput during dynamic updates.
- **GPU Acceleration:** OpenCL accelerates relaxation steps and the identification of affected nodes, offloading intensive tasks from the CPU to the GPU for faster processing.
- **Dynamic Adaptation:** The hybrid method re-partitions and redistributes data as the graph changes, focusing updates on affected regions and avoiding the full recomputation required by the sequential approach.

Conclusion

This project demonstrates the effectiveness of a hybrid parallel approach for dynamically updating Single Source Shortest Paths (SSSP) in large-scale graphs. By integrating MPI, OpenMP, OpenCL, and METIS, we successfully leveraged both distributed and shared memory models, as well as GPU acceleration, to handle dynamic graph updates with significantly reduced computation time compared to a purely sequential implementation. The performance evaluations show that the hybrid method offers superior scalability and speedup, especially as the number of updates increases, highlighting its suitability for large and evolving graph datasets like Spotify's. Profiler analysis further confirms efficient CPU and GPU utilization, with minimal redundancy and effective targeting of affected regions. These results validate the practical value of hybrid parallelism in graph processing, suggesting its potential application in real-time and large-scale systems where graph dynamics are frequent.