Project Proposal: Parallel Union-find

Group members: Zhaowei Zhang, Eric Zhu

Project Website: https://pentene.github.io/parallel-union-find/

Summary

In this project, we will explore different lock techniques of the Union-find Data Structure, specifically coarse-grained lock, fine-grained lock, and lock-free along with Union-Find specific optimizations (path compression, union by rank). We aim to benchmark these implementations rigorously, analyzing their performance characteristics and scalability compared to the serialized version. Finally, we plan to test our implementation in different graph algorithms to further analyze its performance in real-world applications.

Background

The Union-Find data structure is essential in applications that require efficient detection of connectivity, such as clustering, graph connectivity, and image segmentation. The two fundamental operations are **find** and **union**. Parallelizing these operations requires careful handling of concurrent access to shared memory structures, along with optimizations such as path compression and union by rank. Below is a high-level pseudocode with optimizations:

Algorithm 1 Union-Find (Disjoint Set) Implementation

```
function FIND($x$)
    if $parent[x] \neq x$ then
        $parent[x] \leftarrow$ Find($parent[x]$)        ▷ Path compression
    end if
    return $parent[x]$
end function
function UNION($a, b$)
    $rootA \leftarrow$ Find($a$)
    $rootB \leftarrow$ Find($b$)
    if $rootA \neq rootB$ then
        if $rank[rootA] < rank[rootB]$ then
            $parent[rootA] \leftarrow rootB$
        else
            $parent[rootB] \leftarrow rootA$
            if $rank[rootA] = rank[rootB]$ then
                $rank[rootA] \leftarrow rank[rootA] + 1$
            end if
        end if
    end if
end function
```

Parallelism combined with these optimizations significantly enhances performance for large-scale sets. Efficient synchronization must be paired with effective optimizations to achieve peak performance.

Goals and Deliverables
Here are our planned goals:

- Implement Multiple parallel implementations on CPU with OpenMP using:
  - Coarse-grained locking
  - Fine-grained locking
  - Lock-free synchronization
- Implementation of Union-Find optimizations (path compression, union by rank) along with synchronization methods
- Measure and compare throughput, latency, and scalability of each synchronization and optimization method on CPU under diverse workloads, including:
  - Varying graph sizes and densities (sparse, dense)
  - Different access patterns (find-heavy, union-heavy workloads)
  - Real-world datasets

Here are our aspirational goals:

- Investigate dynamic selection of locking vs. lock-free approaches depending on workload size or distribution.
- Demonstrate parallel Union-Find performance improvements in practical applications such as image segmentation or network analysis.

Here are our fallback goals:

- Focus on fewer synchronization methods and optimizations with a thorough performance analysis.
- Identify the potential bottlenecks in the parallel algorithms.

Challenges
Parallelizing and optimizing Union-Find using OpenMP on CPUs presents several significant challenges, driven primarily by workload characteristics, synchronization complexity, and architectural constraints.

- Workload Characteristics: While many find and union operations can run concurrently, conflicts can arise when multiple threads simultaneously update shared structures (parent and rank arrays). Proper synchronization methods must mitigate these conflicts without severely limiting concurrency. And this is one important field we will be testing on.
- Memory access characteristics: Union-Find operations, especially with path compression, lead to irregular memory access patterns. These irregular accesses result in scattered memory reads and writes, limiting cache performance and reducing locality.

- Communication-to-Computation ratio: Despite the simplicity of union-find computations, synchronization overhead can dominate performance, particularly with frequent concurrent updates.
- Divergent execution: The pointer-chasing nature of Union-Find, particularly path compression, inherently generates unpredictable branching and memory access, complicating optimization and efficient parallel execution on CPUs.

By addressing these challenges specifically within the context of OpenMP, we aim to deepen our understanding of parallel programming principles, synchronization mechanisms, and efficient optimization strategies tailored for shared-memory multicore systems.

Resources

Implementation and benchmarking will be performed primarily on GHC lab machines and Pittsburgh Supercomputing Center (PSC) resources. Initial sequential implementations will be written from scratch, with xxx function used for correctness verification. PSC machines, which offer more robust CPU and GPU capabilities, will be particularly useful for large-scale tests and performance analysis.

Here is a list of references:

- Fedorov, A., Hashemi, D., Nadiradze, G., and Alistarh, D. (2023). *Provably-efficient and internally-deterministic parallel union-find.* In *Proceedings of the 35th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA '23)*, Orlando, FL, USA, June 17–19, 2023. Association for Computing Machinery.

  `https://doi.org/10.1145/3558481.3591082`

- Anderson, R. J., and Woll, H. (1991). *Wait-free parallel algorithms for the union–find problem.* In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC '91)*, pp. 370–380. Association for Computing Machinery.

  `https://doi.org/10.1145/103418.103458`

Platform Choice

The GHC lab machines and PSC machines provide multi-core CPU resources suitable for our parallel implementations. GHC machines will facilitate development and initial testing, while the powerful GPU systems at PSC will enable detailed performance profiling and large-scale experimentation.

Schedule (Tentative)

Week	**Planned Activities**
3.26 - 4.2	Update Proposal
4.2 - 4.9	Implement serial Union-Find and coarse-grained locks. Design and generate different workloads.
4.9 - 4.15	Implement fine-grained locks and lock-free Union-Find. Evaluate CPU scalability under diverse workloads.
4.15 - 4.23	Finish the implementations. Test on real-world workloads and analyze performance.
4.23 - 4.28	Finalize benchmarking across all implementations. Analyze trade-offs of locking strategies across workloads. Generate figures for performance comparisons.
4.29	Poster session.