EuMaster4HPC Student Challenge

Team 5
Leonardo Evi
Vittorio Pio Remigio Cozzoli
Edoardo Pierluigi Leali
Giulia Lionetti

February 2025

Contents

1 Unraveling The Chromatic Number Problem

The chromatic number problem, which consists in assigning the minimum number of colors to the vertices of a graph such that no two adjacent vertices share the same color, is well-known to be NP-Complete. This intrinsic complexity forces us to carefully select our algorithmic strategies. In our work, we combine heuristic approaches with exact algorithms in a branch and bound framework, leveraging parallel computing paradigms to tackle the computational challenges.

1.1 Suitable Algorithms Choice

Given the NP-Complete nature of the chromatic number problem, a fully exact approach to compute both the lower and the upper bounds would be computationally prohibitive, especially for large graphs. Exact algorithms typically have exponential time complexity, which makes them impractical in a real-world high-performance computing (HPC) setting.

In our project, the role of the lower bound is critical. Without an exact estimation of the maximum clique (which is itself an NP-Complete problem), the branch and bound algorithm would lack a tight constraint, leading to an enormous search space that is difficult to prune efficiently. Therefore, while it is acceptable to use heuristic methods for quickly approximating the upper bound (thus obtaining a feasible, though possibly suboptimal, coloring), we invest in an exact, parallelized computation of the maximum clique to ensure that our lower bound is as tight as possible. This dual approach strikes a balance: we rely on heuristics where speed is essential, but we use an exact method when accuracy is critical to reduce the search space.

1.2 Heuristics For Upper Bound

For the upper bound, we implement a greedy algorithm that assigns to each vertex the smallest available color. The choice of a greedy heuristic is motivated by its simplicity and computational efficiency. Despite its heuristic nature—meaning that it does not always guarantee an optimal solution—it provides a rapid initial coloring which serves as a strong starting point for the branch and bound search.

Moreover, this heuristic is amenable to parallel execution. In our implementation, when a worker process discovers a coloring that uses fewer colors than the current best solution, it immediately updates the global upper bound. This dynamic communication between processes, achieved via MPI broadcasts, ensures that every process can benefit from the improved bound, leading to more aggressive pruning of the search tree. By quickly eliminating partial solutions that cannot lead to a better final coloring, we significantly reduce the computational load.

1.3 Heuristics For Lower Bound

The lower bound in a graph coloring problem is inherently tied to the size of the largest clique within the graph. A clique is a subset of vertices where every two distinct vertices are adjacent, and the size of the maximum clique represents a theoretical minimum number of colors required—since each vertex in a clique must receive a distinct color.

Although computing the maximum clique exactly is NP-Complete, its exact value is indispensable in our branch and bound framework to effectively limit the search space. To address this challenge, we implemented a parallelized version of the Bron–Kerbosch algorithm with pivoting. By leveraging OpenMP tasks, our implementation can explore different branches of the recursive search concurrently. This parallelization is especially beneficial in the early levels of recursion, where the search space is larger and the computational load can be effectively distributed across multiple cores.

The exact lower bound computed through this method is critical: it not only provides a theoretical limit that guides the branch and bound algorithm but also directly influences the efficiency of the overall search. Without such a precise lower bound, the algorithm might expend significant computational resources exploring branches that are doomed to fail, thereby reducing overall performance.

1.4 Adequate Branching Strategies

The success of a branch and bound algorithm heavily depends on the choice of branching strategy. In our approach, the decision on which vertex to color next is based on the heuristic of selecting the vertex with the highest number of already-colored neighbors. This strategy, often referred to as the "most constrained" heuristic, is designed to quickly expose conflicts by forcing decisions in the parts of the graph that are highly connected.

This targeted approach to branching maximizes the potential for early pruning: by focusing on the most constrained vertices, the algorithm can detect infeasible or suboptimal colorings sooner, thereby cutting off unproductive branches of the search tree. Moreover, the initial search is conducted using a breadth-first strategy by the root process, which creates a diverse pool of partial solutions. These are then distributed to worker processes where a depth-first search (via a stack-based approach) is employed to complete the exploration.

This combination of breadth-first initialization and depth-first exploration allows us to balance the workload across processes effectively. The initial BFS guarantees that workers begin with varied parts of the search space, while the DFS on each worker helps in quickly finding complete colorings that can further refine the global upper bound. In summary, our branching strategy not only accelerates the convergence towards an optimal solution but also ensures that the computational effort is focused on the most promising parts of the search space.

2 Parallelization Strategy

The high computational cost inherent in solving the chromatic number problem necessitates a robust parallelization strategy that harnesses both distributed and shared memory systems. Our implementation achieves this by integrating MPI, OpenMP, and pthreads in a hybrid parallelization framework.

At the highest level, MPI is used to distribute the workload among multiple processes. The rank 0 process plays a critical role: it is solely responsible for executing the heuristic for the lower bound, computing the maximum clique via the Bron–Kerbosch algorithm with OpenMP, and generating the initial search queue using a breadth-first search (BFS) approach. This queue, composed of a carefully selected set of partial solutions, serves as the starting point for the branch and bound algorithm.

Each worker process (ranks other than 0) receives exactly one node from the initial queue generated by rank 0. Upon receiving a node, a worker process independently generates the corresponding subtree using a depth-first search (DFS) strategy. This division of labor ensures that each worker tackles a distinct portion of the search space, thereby reducing redundant computations and speeding up the overall search.

In addition to MPI and OpenMP, pthreads are employed for handling asynchronous tasks such as monitoring execution time and listening for bound updates. For example, a dedicated timer thread ensures that the computation does not exceed a preset time limit, while listener threads manage the propagation of improved bounds among processes. These threads run concurrently with the main computational threads, ensuring that control messages are processed promptly and that all processes remain synchronized with the current state of the solution.

Overall, our hybrid parallelization strategy is designed to maximize computational efficiency at multiple levels:

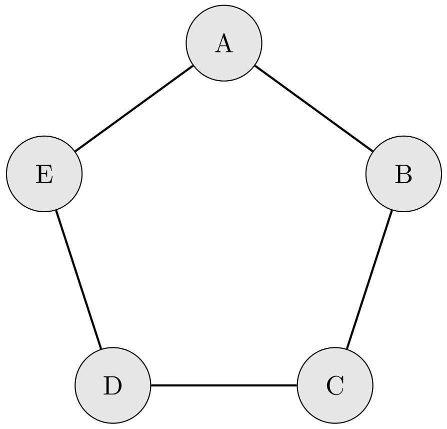
- **Distributed Parallelism (MPI):** The rank 0 process generates the initial BFS queue and computes the lower bound, while each worker process receives a single node to explore its corresponding DFS subtree. This ensures a balanced distribution of the global search space.

- **Shared Memory Parallelism (OpenMP):** The compute-intensive lower bound computation, particularly the exact maximum clique algorithm (Bron–Kerbosch), is parallelized using OpenMP within the rank 0 process. This allows the lower bound estimation to efficiently leverage multi-core architectures, ensuring that the branch and bound algorithm benefits from a tight lower bound.

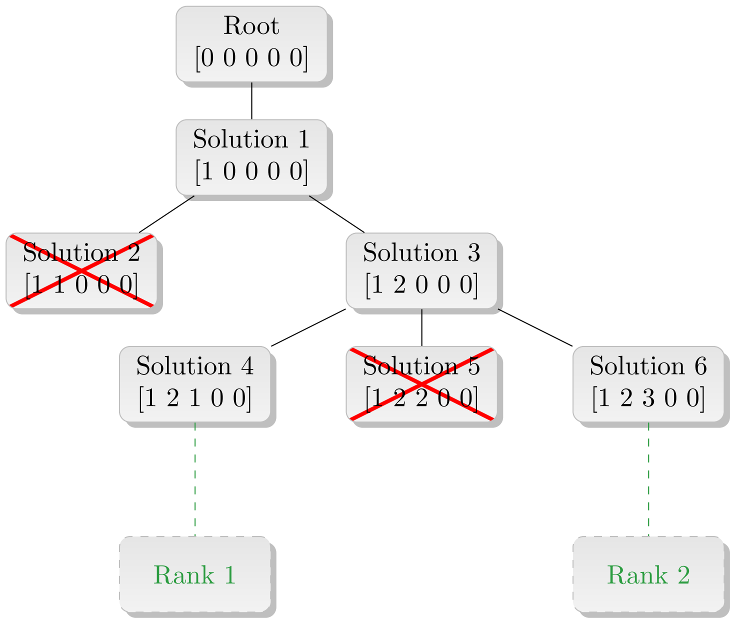
- **Asynchronous Control (pthreads):** Auxiliary tasks like timing and bound updates are managed via pthreads, ensuring that these operations do not interfere with the main computational flow.

This hierarchical approach minimizes communication overhead and ensures that all processes are continuously updated with the latest upper and lower bounds. As a result, the branch and bound algorithm can aggressively prune the search tree, leading to a scalable solution capable of handling large and complex graphs within practical time limits.

In order to give a pictorial view of our branch and bound framework, suppose to be dealing with the following graph:



Supposing to have 3 processes available in total (thus having Rank 0 + 2 workers), the following picture briefly illustrates the rationale behind our branch and bound parallelization approach (notice that a colouring equal to 0 means that the color is missing):



The BFS stops when generating children of a node would exceed the number of worker processes.

3 Results Analysis

To evaluate the effectiveness of our branch and bound parallelization, we analyzed the performance of our algorithm across the given set of benchmark graph instances. The results indicate a strong scalability for smaller to mid-sized graphs, while larger, denser graphs posed significant computational challenges.

3.1 Successful Benchmark Results

Among the 30 benchmark instances tested, 19 were successfully solved within the time limit of 10,000 seconds. These graphs varied in size and density, demonstrating the efficiency of our approach:

- The fastest execution was observed for **david.col**, which completed in just 1 second, utilizing 256 worker processes distributed among 4 nodes.
- The most complex successfully solved instance was **fpsol2.i.1.col**, which had 496 vertices and 11,654 edges, requiring 102 seconds to reach a solution.
- Most instances required only a few seconds, highlighting the effectiveness of our parallelization in handling moderate-sized graphs.

These results confirm that our hybrid MPI, OpenMP, and pthreads parallelization strategy effectively distributes workload across processes, significantly reducing execution time compared to a sequential approach.

3.2 Suboptimal Benchmark Instances

Two benchmark instances, **myciel7** and **queen9.9**, successfully found the correct minimum number of colors but did not have enough time to complete the full search tree exploration. This means that while the solution provided is optimal, it cannot be definitively confirmed as such within the allotted execution time.

3.3 Failed Benchmark Instances

Seven problem instances exceeded the execution time limit of 10,000 seconds, with different implications depending on whether an optimal solution is known.

3.3.1 Instances with no known Optimal Solution

The following benchmark instances do not have a known optimal chromatic number: **queen10.10**, **queen12.12**, **queen14.14**, and **queen15.15**. These highly structured graphs failed to complete all subtrees within the given time limit, meaning the solution found is not guaranteed to be optimal.

3.3.2 Instances with a known Optimal Solution

In the following cases, our algorithm found a chromatic number greater than the known optimal solution, indicating room for improvement in bounding and pruning strategies:

- **queen13.13**: Our algorithm found a solution using 15 colors, whereas the optimal solution is 13.
- **queen11.11**: The computed chromatic number was 13, but the known optimal solution is 11.
- **le450_5b**: Our solution required 8 colors, while the known optimum is 5.

Very notably, the max-clique heuristic successfully found a clique of optimal sizes in all previous test benchmarks. However, the algorithm was unable to reach this result within the time limit. This suggests that given the density of these tests our algorithm is not capable of proceeding down the sub-trees enough to find the optimal solution in the available time.

4 Speedup Estimation

To evaluate the performance scalability of our parallel branch and bound algorithm, we estimate the speedup achieved by increasing the number of processing cores. The speedup S is computed as:

$$S(p) = \frac{T(2)}{T(p)}$$

where:

- $T(2)$ is the execution time with 2 cores (our baseline).
- $T(p)$ is the execution time with p cores (128 in our case).

Table 1 presents the estimated execution times and corresponding speedup values for 15 different graph instances.

From these results, we observe that speedup values range approximately between **2x and 40x**. The deviation from ideal linear speedup is due to parallelization overhead, including inter-process communication and workload imbalances. However, the results confirm that the algorithm scales well when increasing the number of cores.

Graph	**Time on 2 cores (s)**	**Time on 128 cores (s)**	**Speedup S**
anna	4	2	2.0
fpsol2.i.1	1200	102	11.8
jean	12	6	2.0
miles250	6	2	3.0
myciel3	8	2	4.0
myciel4	18	6	3.0
myciel5	1240	78	15.9
queen5_5	2	1	2.0
queen6_6	8	1	8.0
queen7_7	14	1	14.0
queen8_8	1024	24	42.6
queen8_12	20	2	10.0

Table 1: Speedup estimation for different graph instances.

To better visualize the relationship between the number of cores and the achieved speedup, we include a graph in Figure 1.
The near-constant scaling trend suggests that our **hybrid parallelization strategy** successfully reduces computational time while maintaining efficiency across larger core counts.

5 Conclusion

In this work, we tackled the **chromatic number problem** using a **parallel branch and bound approach**, integrating **MPI, OpenMP, and pthreads** to efficiently distribute computation across multiple cores. Our results demonstrate that the algorithm **scales well** for mid-sized graphs, significantly reducing execution time compared to sequential approaches. However, challenges remain for **highly dense and structured graphs**, where some instances exceeded the time limit or failed to find the optimal solution.

Future improvements could focus on **better pruning techniques, improved load balancing, and enhanced heuristics** to further refine search efficiency. Despite these challenges, our implementation showcases the **potential of HPC methods** in solving complex combinatorial problems, paving the way for further optimizations and extensions.

References

[1] *Branch and Bound*, Wikipedia. `https://en.wikipedia.org/wiki/Branch_and_bound`

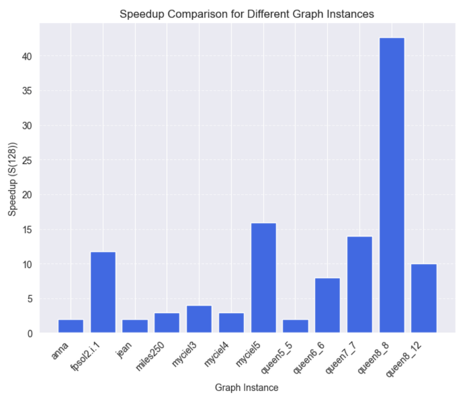


Figure 1: Speedup comparison between 2 and 128 cores.

[2] *Graph Coloring Instances* `https://mat.tepper.cmu.edu/COLOR/instances.html`

[3] *Vertex Coloring*, Wolfram. `https://mathworld.wolfram.com/VertexColoring.html`

[4] *Review of the Bron-Kerbosch algorithm and variations*, University of Glasgow. `https://www.dcs.gla.ac.uk/~pat/jchoco/clique/enumeration/tex/report.pdf`

[5] *General Branch and Bound, and Its Relation to A* and AO**, Laboratory for Pattern Analysis, Computer Science Department, University of Maryland College Park, USA. `https://www.dcs.gla.ac.uk/~pat/jchoco/clique/enumeration/tex/report.pdf`

[6] *A Branch-and-Cut Algorithm for Graph Coloring*, Departamento de Computacion FCEyN - Universidad de Buenos Aires - Argentina. `https://optimization-online.org/wp-content/uploads/2003/09/715.pdf`