EUMaster4HPC Challenge:
Chasing the Perfect Hue: A High-Performance Dive into Graph Coloring

Alessandro Ruzza, Tommaso Crippa
Elizabeth Koleva, Dimitar Penkov

Objective

Graph coloring is a fundamental problem in combinatorial optimization with applications in scheduling, register allocation, and parallel computing [4]. The goal of this project is to determine the chromatic number of a given graph using a Branch-and-Bound framework, enhanced with heuristic-based bounding strategies.

The algorithm:

- Use heuristics to find the maximum clique in the graph, which provides a lower bound on the chromatic number.
- Use heuristics to find a valid graph coloring, which provides an upper bound on the chromatic number.
- Implement a branch-and-bound strategy to refine these bounds and find the exact chromatic number.
- Parallelized branch-and-bound algorithm using MPI to improve computational efficiency.

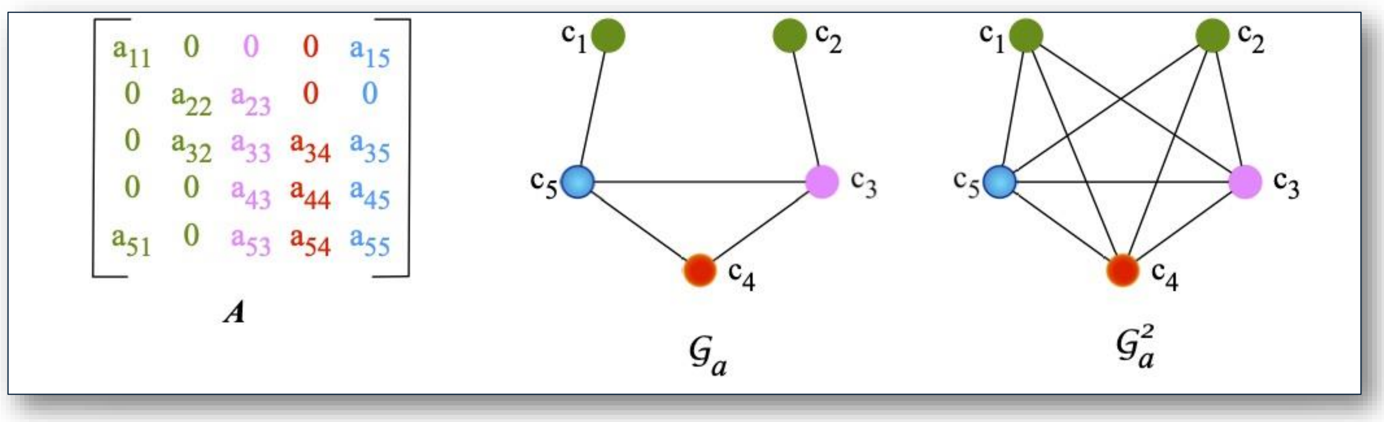
VEGA Features:

- 960 CPU nodes (overall 1920 CPUs AMD Epyc 7H12 – 122000 cores)
- 60 GPU nodes (overall 240 GPUs NVidia A100)

Theoretical Background

Graph coloring is the process of assigning distinct colors to each vertex of a graph $G = (V, E)$ such that no two adjacent vertices share the same color. A graph G is k-colorable if its vertices can be colored using at most k different colors while satisfying this condition. The smallest k for which G is k-colorable is called the chromatic number of G, denoted by $\chi(G)$.

A clique in a graph G is a subset of vertices where every pair of vertices is connected by an edge. The clique number $\omega(G)$ is the size of the largest clique in G. Since each vertex in a clique must be assigned a unique color in any valid graph coloring, the relationship $\omega(G) \leq \chi(G)$ always holds.



Picture 1. Adjacency Matrix and Its Graph Representations

Milestones (Evolution of Heuristics and Strategies)

Max Clique Heuristic Development

- **Greedy**: Initial approach prioritizing vertices with high connectivity, building maximal cliques iteratively. Efficient but prone to suboptimal solutions.
- **DLS (Delayed Local Search)**: Improved approach using penalty mechanisms to iteratively refine candidate cliques, allowing escape from local optima.
- **AdaptiveDLS**: Enhanced version that dynamically shifts between standard DLS and color-aware strategies as coloring constraints evolve during search.
- **ParallelDLS**: Final implementation running multiple DLS instances simultaneously with varied hyperparameters, significantly increasing likelihood of discovering larger cliques.

Coloring Heuristic Progression

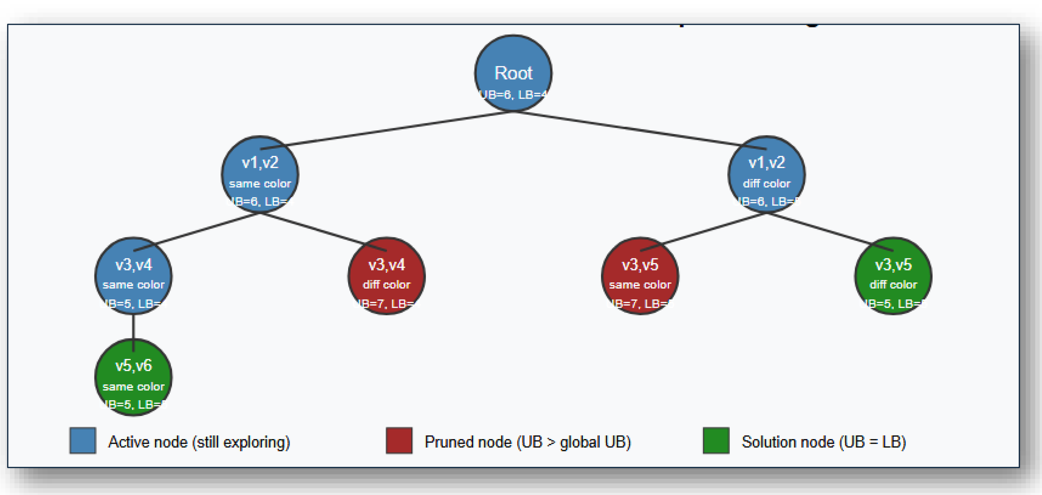
- **Greedy**: Basic coloring algorithm assigning colors in sequential order.
- **DSatur**: More sophisticated approach prioritizing vertices with highest saturation degree (number of distinct colors in neighborhood).
- **Backtracking DSatur**: Extended DSatur with backtracking capabilities to reconsider previous choices when conflicts arise, closer-to-optimal colorings.
- **Parallel Backtracking DSatur**: Multiple concurrent instances with randomization elements, exploiting multi-core architectures to explore a wider solution space.

Branching Strategy Refinement

- **Random Selection**: Initial approach selecting non-adjacent vertex pairs without structural considerations.
- **Degree-based Selection**: Improved strategy prioritizing highly connected vertices to maximize early constraints.
- **Saturation-based Selection**: Most effective approach selecting vertices with highest saturation degree, focusing on most constrained choices first.

Branch and Bound Framework Evolution

- **Sequential Version**: Initial implementation systematically exploring partial colorings using core data structures.
- **MPI First Version**: Early parallelization with ranks working independently but lacking coordination.
- **MPI Manager-Worker Version**: Advanced implementation with dynamic load balancing and task allocation, significantly improving parallel efficiency.



Picture 2. Branch-and-Bound Search Tree for Coloring
The core concept of how BB framework explores the solution space by branching on vertex pairs and using bounds to prune paths.

Key Achievements

- **Algorithm Development:** Successfully implemented a Branch-and-Bound framework enhanced with multiple heuristics for upper and lower bounds.
- **Benchmark Testing:** Successfully solved 16/30 instances to proven optimality, with tight bounds for remaining instances.
- **Parallelization Effectiveness:** Achieved significant speed improvements with the Manager/Worker model, especially for larger graph instances.
- **Computational Insights:** Identified specific graph types (Mycielski, Queen) that pose unique challenges for the approach, highlighting areas for future improvement.

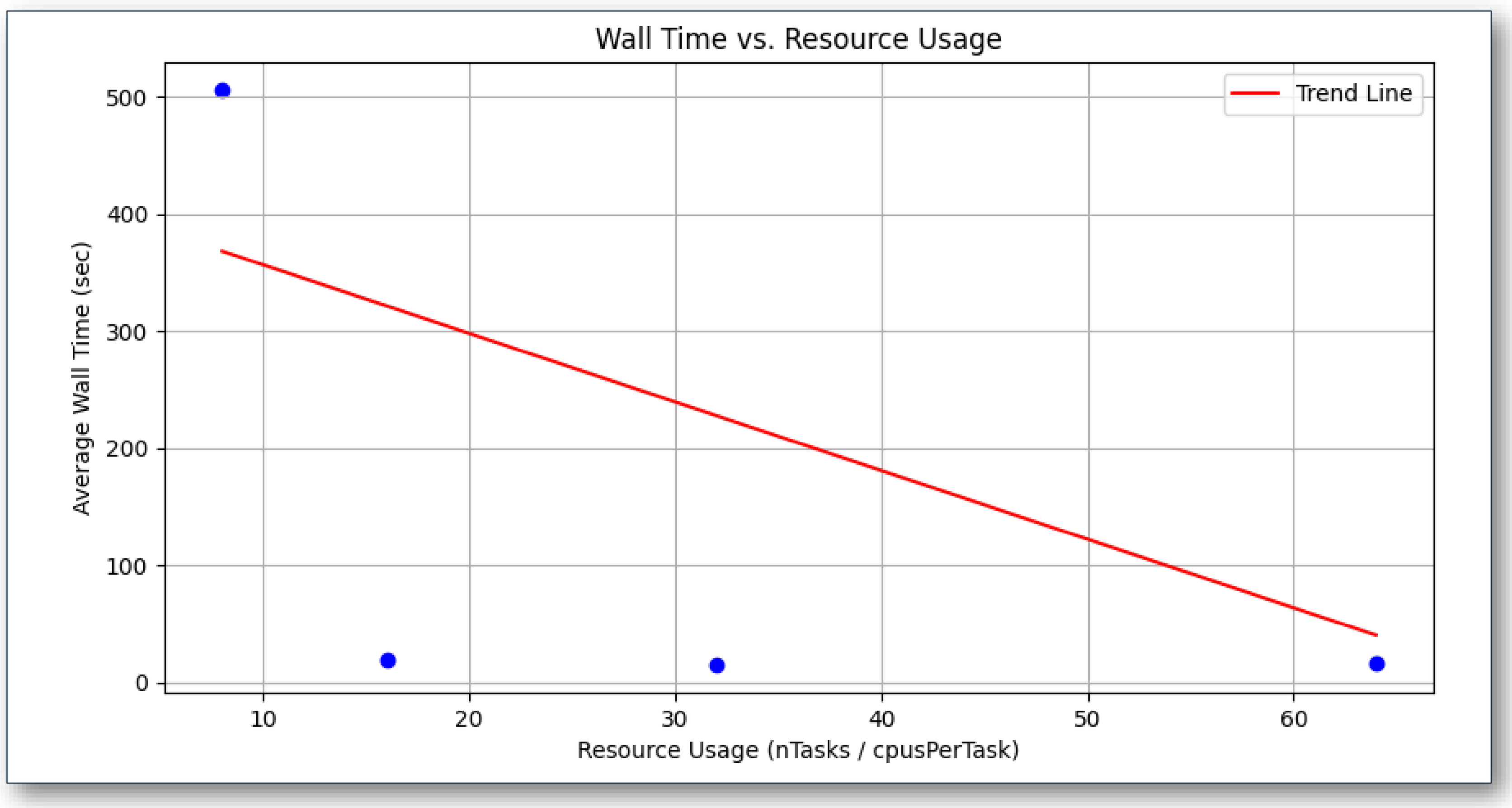
Benchmark Results

We tested our approach on 30 benchmark instances, successfully finding the optimal solution (UB = LB) for 16 instances (53.3%). For the remaining instances, we obtained tight bounds within the 10,000-second time limit.

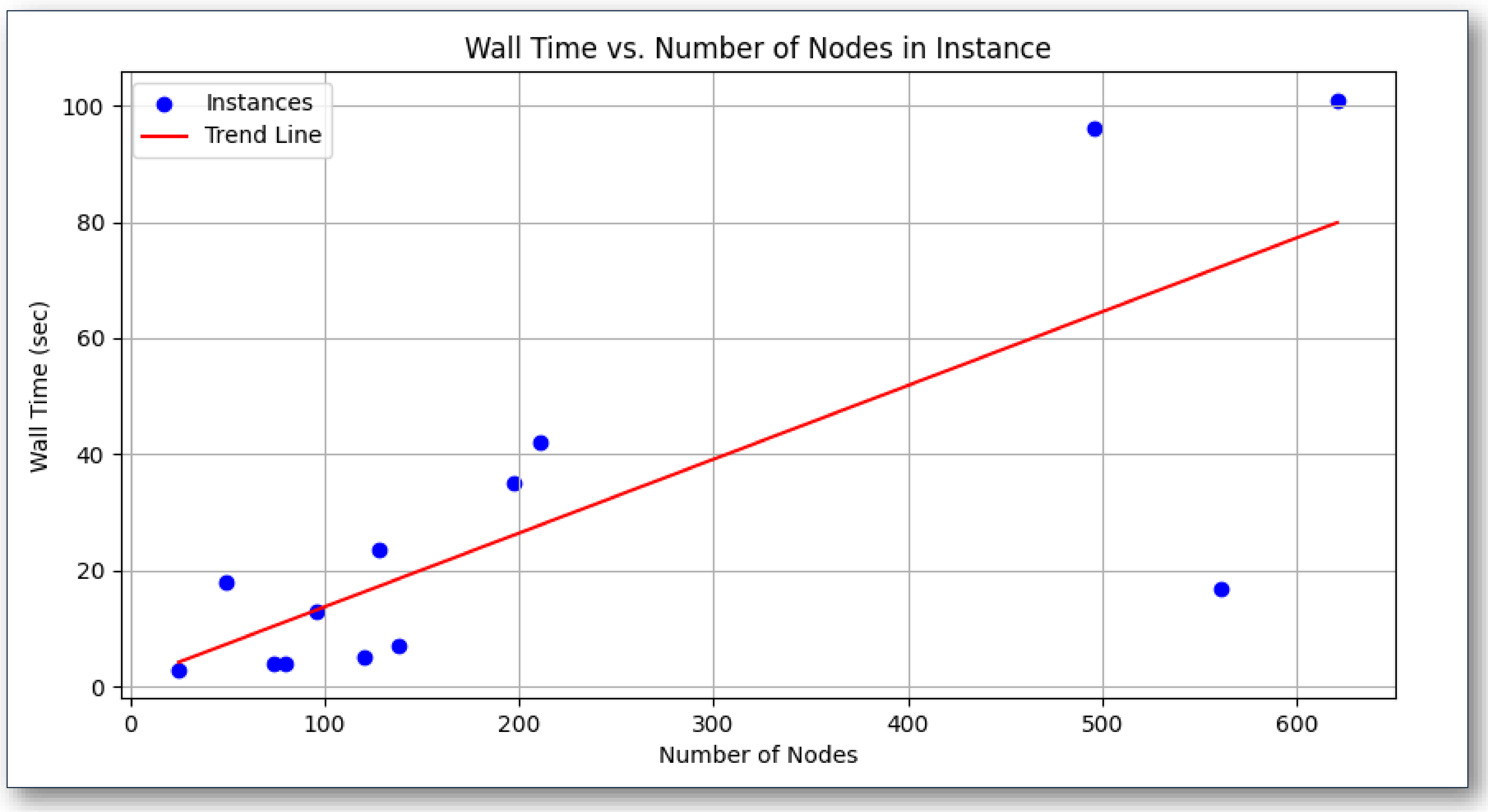
Key Findings by Graph Type:

- **Standard Benchmarks:** Near-perfect performance with most instances solved within seconds
- **Large Graphs:** Successfully solved several instances with 500+ vertices
- **Mycielski Graphs:** Triangle-free structure created a consistent gap between LB (2) and UB
- **Queen Graphs:** Smaller instances solved optimally, while larger ones proved challenging

The chromatic number χ(G) for solved instances corresponds to the matching upper and lower bound values, demonstrating that our Branch-and-Bound approach effectively navigates the solution space for various graph classes.



Graphics 1. Performance Scaling Analysis: Wall Time vs. Resource Allocation. How it changes with computational resources.



Graphics 2. Performance Scaling Analysis: Wall Time vs. Problem Size. How wall time relates to the problem size (number of nodes).

Acknowledgements

The authors would like to thank their mentor, Mirko Rahn, for his invaluable support and guidance. We also extend our gratitude to the EuroHPC Summit Student Challenge organizers for providing this opportunity.

References

[1] Brélaz, D. (1979). New methods to color the vertices of a graph. Communications of the ACM, 22(4), 251-256.
[2] Lewis, R. M. R. (2021). Guide to Graph Colouring: Algorithms, Applications and Implementations (2nd ed.). Springer.
[3] Pullan, W., & Hoos, H. H. (2006). Dynamic local search for the maximum clique problem. Journal of Artificial Intelligence Research, 25, 159-185.
[4] Ufuktepe, Ü., & Turan, G. B. (2005). Applications of graph coloring. In Proceedings of the International Conference on Computational Science and Its Applications (pp. 522-528).
[5] Wu, Q., & Hao, J.-K. (2015). A review on algorithms for maximum clique problems. European Journal of Operational Research, 242(3), 693-709.

This project has received funding from the European High Performance Computing Joint Undertaking under grant agreement No 101051997. EUROHPC SUMMIT 2025, Krakow, Poland, EUMaster4HPC
See more: https://github.com/Rudolfovoorg/EuroHPC_Student_Challenge_2025_Team_3
alessandro.ruzza@mail.polimi.it crippa.tommaso@gmail.com,
elizabeetvk@gmail.com, penkov.dimit@gmail.com