Graph Coloring and Maximum Clique: A Branch and Bound Approach

EuroHPC Krakow Student Challenge 2025 - Team 6

Arianna Amadini, Mario Capodanno, Valerio Grillo, Alberto Taddei

Politecnico di Milano

Contents

1 Introduction

Graph coloring is a fundamental problem in combinatorial optimization with the goal to color the vertices of a graph such that no two adjacent vertices share the same color. The smallest number of colors needed for such a proper coloring is the *chromatic number* $\chi(G)$. Determining this number is NP-complete, implying that finding efficient algorithms, especially for large and complex graphs, is a computationally challenging task. Still, the graph coloring problem remains crucial in a diverse range of applications, such as register allocation in compiler design, scheduling problems, frequency assignment in wireless networks, and timetable scheduling.

In the context of the EuroHPC Krakow Student Challenge 2025, our objective is to implement an efficient and parallelized solver for graph coloring, targeting the VEGA supercomputer[1].
VEGA's heterogeneous architecture, featuring both powerful CPU and GPU clusters, provides a rich environment for exploring different parallelization strategies and optimizing algorithm performance.
Our approach involves a **branch and bound** method, suitable for solving NP-hard optimization problems, combined with parallelism via **MPI** and **OpenMP**. The code is publicly available[2].

2 Approach and strategy

This section details the main ideas behind our solution strategy:

- Outline the upper and lower bound heuristics (Subsection 2.1), which play a critical role in pruning, a technique employed in the subsequent branch and bound method;
- Describe the branch and bound framework (Subsection 2.2), including how it is parallelized across multiple nodes (via MPI) and further distributed over multiple CPU cores (via OpenMP).

2.1 Upper and lower bound heuristics

2.1.1 Lower Bound: Maximum Clique (Bron–Kerbosch)

A fundamental observation in graph coloring is that the chromatic number $\chi(G)$ cannot be smaller than the size of the largest clique $\omega(G)$. To find such a clique, we use a simplified version of the **Bron–Kerbosch** algorithm, which enumerates maximal cliques by recursively building candidate cliques.

The algorithm manages three sets:

$$R \text{ (current clique)}, \quad P \text{ (potential vertices)}, \quad X \text{ (excluded vertices)},$$

and performs these steps:

1. **Check maximality**: If $P \cup X = \varnothing$, then R is a maximal clique.
2. **Branching**: For each vertex $v \in P$, move v into R, and restrict future candidates to neighbors of v. Remove v from P and add it to X.

When a maximal clique is found, $\omega(G)$ is updated if its size is larger than any previously discovered clique. Since $\omega(G) \leq \chi(G)$, this provides a powerful *lower bound* for the coloring problem.

[1] **VEGA Specifications:** `https://izum.si/`
[2] **GitHub Repository:** `https://github.com/Rudolfovoorg/EuroHPC_Student_Challenge_2025_Team_6.git`

2.1.2 Upper Bound: DSATUR Heuristic

To complement the lower bound, we need a quick method to produce a valid coloring that overestimates $\chi(G)$. We employ **DSATUR**, a greedy heuristic that assigns colors one vertex at a time by choosing the vertex with the highest "saturation degree" (i.e., the number of distinct colors among its neighbors). In practice:

1. Initialize all vertices as uncolored.
2. At each step, select an uncolored vertex whose neighbors use the greatest variety of colors; this vertex is said to have the highest saturation degree.
3. Assign the smallest valid color to that vertex (i.e., a color not used by its neighbors).
4. Update saturation degrees for the remaining uncolored vertices.

Once all vertices are colored, we have a feasible (but not necessarily minimal) coloring, giving an *upper bound* on $\chi(G)$. Throughout the main branch and bound search, any improvement on this coloring (i.e., fewer colors) will replace the DSATUR count as our best known upper bound.

Combining these bounds, we obtain

$$\omega(G) \;\le\; \chi(G) \;\le\; \mathrm{DSATUR}(G),$$

which ensures that the search space of our algorithm is pruned aggressively, stopping branches where the required number of colors exceeds the current upper bound or cannot surpass the maximum clique lower bound.

2.2 Branch and Bound Framework

Branching Scheme. Our branch and bound approach leverages the fact that merging two non-adjacent vertices forces them to share the same color, whereas enforcing them to be in different color classes is comparable to *adding an edge* between them:

- *Merge (same color)*: When two vertices v_1 and v_2 are non-adjacent, we can unify them into the same color "group" and reduce the size of the problem by effectively treating them as one "super-vertex."
- *Add Edge (different colors)*: Alternatively, we treat v_1 and v_2 as if there were an edge between them, meaning they must be distinct color classes in any valid solution.

Each such decision spawns a new subproblem in the search tree. If, during branching, a subproblem's current color count cannot possibly beat our global upper bound, that branch is pruned immediately.

Pruning Criteria. We maintain two global bounds:

- **Lower Bound (LB)**: Provided by our maximum clique size $\omega(G)$. If adding merges or edges leads to a scenario where the subproblem would require fewer than $\omega(G)$ colors, it is unfeasible and can be pruned.
- **Upper Bound (UB)**: Initially set by the DSATUR heuristic, the upper bound can be further refined during the search whenever a better coloring with fewer colors is found. If a partial solution already reaches or exceeds the UB in the number of colors used, we prune that branch.

Algorithmically, we store these bounds in shared global variables (e.g., via `globals.hpp/globals.cpp`) so that any branch can check the most up-to-date LB and UB. This requires careful synchronization, either via OpenMP atomics or MPI reduction when running in parallel.

3 Benchmark Results Across Instances

This section presents a comprehensive analysis of the benchmarks results obtained for our parallel graph coloring solver on the VEGA supercomputer. We evaluated the solver's performance across the set of provided benchmark instances, using MPI process counts ranging from 1 to 64 and consistently employing 256 OpenMP threads per process.
For completeness, Tables A and B, showcasing wall time and color counts, are moved to Appendix.
As shown, certain dense or larger graphs remain unsolved within the time limit, while many mid-sized instances are solved successfully. The color values confirm the correctness of the approach when solutions are found.

4 Performance Analysis

In this section, we present a detailed analysis of the performance characteristics of our parallel graph coloring solver based on extensive benchmarks. The evaluation integrates automated log processing, data extraction, and visualization to assess key metrics such as execution time, speedup, and scalability.

4.1 Data Extraction

Benchmarking begins with the generation of log files containing runtime and configuration details. The `convert.py` script renames raw log files (with the extension `.output`) into `.txt` files and organizes them into a dedicated output directory. This preprocessing step ensures that the data is in a consistent format for analysis.

Subsequently, the `tab_gen.py` script parses these `.txt` files to extract essential performance and configuration metrics, including:

- **Problem Instance**: The input instance name.
- **MPI Processes**: The number of MPI processes used.
- **Threads per Process**: The number of threads employed per MPI process.
- **Wall Time (sec)**: The execution time measured in seconds.
- **Time Limit Compliance**: A flag indicating if the run completed within the specified time limit.
- **Number of Colors**: The total number of colors determined by the solver.

The extracted data is compiled into a sorted configuration table (saved as `configuration_table_sorted.csv`). The table is organized first by problem instance and then by MPI process count (following a preferred order of 1, 2, 4, 8, 16, 32, 64), providing a comprehensive overview of the benchmark settings and results.

4.2 Speedup Analysis and Scalability Visualization

The next phase of the analysis focuses on the solver's scalability through detailed speedup measurements. The `visualize_speedup.py` script computes speedup by comparing the wall time of multi-process runs against a baseline single MPI process execution. For each instance, speedup is defined as the ratio of the wall time for 1 MPI process to that for multiple MPI processes.

The script generates several key visualizations:

- **Instance-Specific Speedup Plots**: These plots illustrate how speedup varies with the number of MPI processes for each benchmark instance. Each plot is annotated with the instance name and the minimum number of colors used, offering insights into individual instance performance.
- **Global Average Speedup Plot**: By aggregating the speedup data across all instances, this plot (Figure 1) reveals the overall performance improvement as more MPI processes are employed.
- **Strong and Weak Scaling Analyses**:
  - **Strong Scaling** (Figure 2): Demonstrates performance gains when the problem size is fixed and the number of MPI processes increases.
  - **Weak Scaling** (Figure 3): Assesses how the solver maintains efficiency when both the problem size and resources are proportionally scaled.

The automation of benchmark execution and analysis is described in the `INSTALL.md` file. This guide provides detailed instructions for setting up the environment, compiling the solver, and running the necessary scripts to perform a complete performance evaluation.

4.3 Summary of Key Performance Metrics

The integrated analysis framework yields several important observations:

- **Global Average Speedup**: As shown in Figure 1, the solver achieves a substantial average speedup across all instances, indicating robust parallel efficiency.

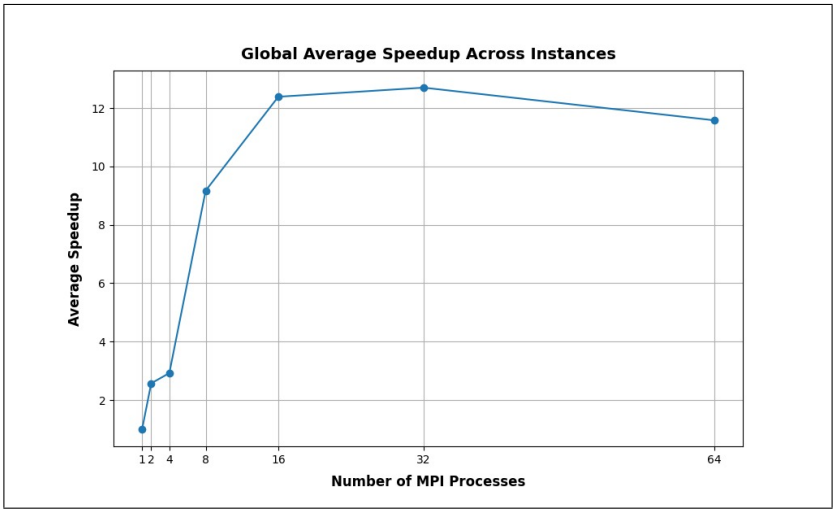

Figure 1: Global average speedup

- **Strong Scaling**: The solver effectively reduces execution time for a fixed problem size as the number of MPI processes increases, which is depicted in Figure 2.

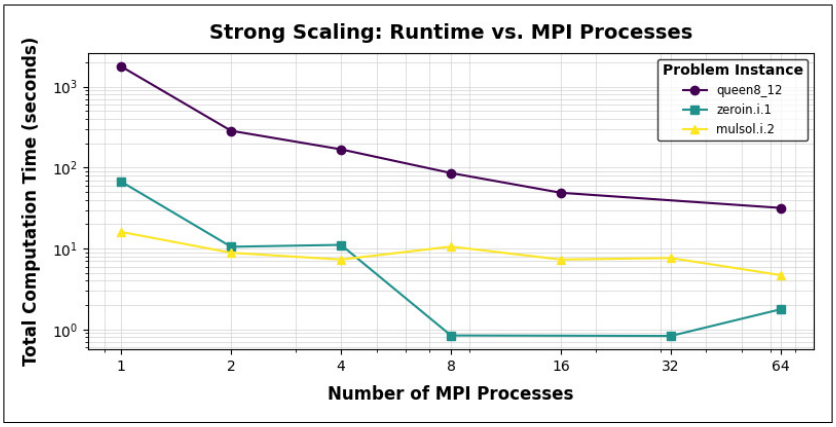

Figure 2: Strong scaling

- **Weak Scaling**: The solver maintains its performance when both the problem size and computational resources are increased, as demonstrated by the weak scaling results in Figure 3.

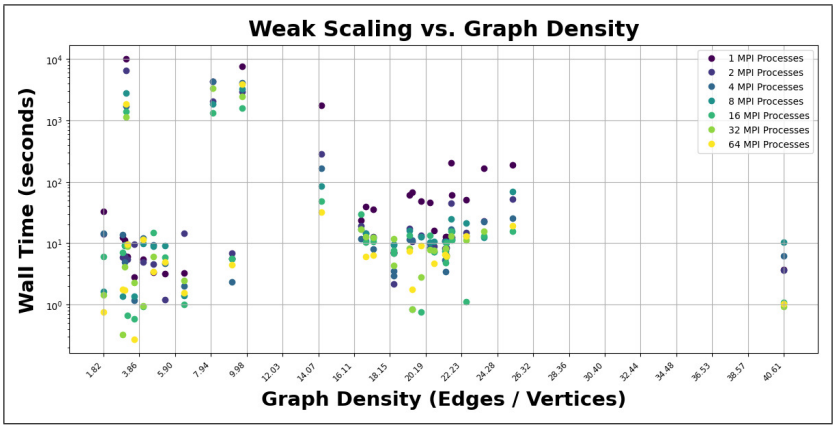

Figure 3: Weak scaling

5 Conclusion

In this work, we presented a parallel Branch and Bound solver for the graph coloring problem, combining a Bron–Kerbosch-based algorithm (for the maximum clique lower bound) with a DSATUR heuristic (for the coloring upper bound). We showed that this approach effectively prunes the search space and, when deployed on the VEGA supercomputer, scales well across multiple nodes and threads. Our benchmark results reveal promising speedups for mid-sized instances, confirming the effectiveness of our hierarchical parallelization strategy based on MPI for coarse-grained subproblem distribution and OpenMP for shared-memory concurrency.

Nevertheless, certain large or dense instances remain challenging, indicating that further improvements in load balancing and pruning heuristics are worthwhile. Overall, our results underscore the power of tight bounding procedures, when integrated with advanced parallelization techniques, to tackle computationally demanding coloring problems on leading-edge high-performance computing platforms.

6 References

[1] Xuezhen Wang, Songheng Yin. *Maximal Clique Problem with Bron–Kerbosch Algorithm.* Available at: https://www.cs.columbia.edu/ sedwards/classes/2023/4995-fall/reports/MaximalClique-report.pdf

[2] Fabio Furini, Virginie Gabrel, Ian-Christopher Ternier. *Animproved DSATUR-based Branch and Bound for the Vertex Coloring Problem.* Available at: https://www.researchgate.net/publication/309817697_An_Improved_DSATUR-Based_Branch-and-Bound_Algorithm_for_the_Vertex_Coloring_Problem

A Benchmarks results: instances

Index	Instance	MPI Processes	Threads per Process	Wall Time (sec)	Within Time Limit	Colors
0	anna	1	256	2.831120	True	11
1	anna	2	256	9.666300	True	11
2	anna	4	256	1.178040	True	11
3	anna	8	256	1.351680	True	11
4	anna	16	256	0.583549	True	11
5	anna	32	256	2.283050	True	11
6	anna	64	256	0.272818	True	11
7	david	1	256	3.354640	True	11
8	david	2	256	4.509680	True	11
9	david	4	256	9.302380	True	11
10	david	8	256	8.908540	True	11
11	david	16	256	15.016300	True	11
12	david	32	256	6.051990	True	11
13	david	64	256	3.427940	True	11
14	fpsol2.i.1	1	256	165.397000	True	65
15	fpsol2.i.1	2	256	23.228500	True	65
16	fpsol2.i.1	4	256	22.509600	True	65
17	fpsol2.i.1	8	256	12.292500	True	65
18	fpsol2.i.1	16	256	12.981200	True	65
19	fpsol2.i.1	32	256	15.608100	True	65
20	games120	1	256	3.171430	True	9
21	games120	2	256	1.212590	True	9
22	games120	4	256	4.690170	True	9
23	games120	8	256	9.225240	True	9
24	games120	16	256	5.874300	True	9
25	games120	64	256	4.972540	True	9
26	homer	1	256	12.404300	True	13
27	homer	2	256	5.849120	True	13
28	homer	4	256	13.593000	True	13
29	homer	8	256	1.364750	True	13
30	homer	16	256	7.142530	True	13
31	homer	32	256	0.322793	True	13
32	homer	64	256	1.747400	True	13
33	huck	1	256	5.437440	True	11
34	huck	2	256	4.893160	True	11
35	huck	4	256	12.129000	True	11
36	huck	8	256	9.783580	True	11
37	huck	16	256	0.919095	True	11
38	huck	32	256	0.941270	True	11
39	huck	64	256	11.496900	True	11
40	inithx.i.3	1	256	50.992900	True	31
41	inithx.i.3	2	256	14.731100	True	31
42	inithx.i.3	4	256	12.932900	True	31
43	inithx.i.3	8	256	21.085900	True	31
44	inithx.i.3	16	256	1.101950	True	31
45	inithx.i.3	32	256	11.343400	True	31

Index	Instance	MPI Processes	Threads per Process	Wall Time (sec)	Within Time Limit	Colors
46	inithx.i.3	64	256	13.115400	True	31
47	jean	1	256	6.118430	True	10
48	jean	2	256	5.409430	True	10
49	jean	4	256	9.411790	True	10
50	jean	8	256	9.523040	True	10
51	jean	16	256	0.663839	True	10
52	jean	32	256	8.887430	True	10
53	jean	64	256	9.572950	True	10
54	le450_5b	2	256	10001.000000	False	10
55	le450_5b	4	256	10000.300000	False	10
56	le450_5b	8	256	10000.500000	False	10
57	le450_5b	16	256	10000.300000	False	10
58	le450_5b	32	256	10000.400000	False	10
59	le450_5b	64	256	10000.700000	False	10
60	miles1500	1	256	3.685840	True	73
61	miles1500	2	256	3.648050	True	73
62	miles1500	4	256	6.155420	True	73
63	miles1500	8	256	10.499400	True	73
64	miles1500	16	256	1.078930	True	73
65	miles1500	32	256	0.920079	True	73
66	miles1500	64	256	1.027600	True	73
67	miles250	1	256	11.097100	True	8
68	miles250	2	256	6.261160	True	8
69	miles250	4	256	4.894980	True	8
70	miles250	8	256	9.090220	True	8
71	miles250	16	256	9.410330	True	8
72	miles250	32	256	4.086680	True	8
73	miles250	64	256	1.705000	True	8
74	mulsol.i.1	1	256	48.274400	True	49
75	mulsol.i.1	2	256	13.515400	True	49
76	mulsol.i.1	4	256	12.627700	True	49
77	mulsol.i.1	8	256	12.639600	True	49
78	mulsol.i.1	16	256	0.751244	True	49
79	mulsol.i.1	32	256	2.818180	True	49
80	mulsol.i.1	64	256	9.226510	True	49
81	myciel3	1	256	32.552300	True	4
82	myciel3	2	256	14.079100	True	4
83	myciel3	4	256	14.446400	True	4
84	myciel3	8	256	1.627790	True	4
85	myciel3	16	256	6.033320	True	4
86	myciel3	32	256	1.441370	True	4
87	myciel3	64	256	0.760713	True	4
88	myciel4	1	256	10000.100000	False	5
89	myciel4	2	256	6556.800000	True	5
90	myciel4	4	256	1715.460000	True	5
91	myciel4	8	256	2816.720000	True	5
92	myciel4	16	256	1399.080000	True	5
93	myciel4	32	256	1142.020000	True	5
94	myciel4	64	256	1871.880000	True	5
95	myciel5	1	256	10003.000000	False	6

Index	Instance	MPI Processes	Threads per Process	Wall Time (sec)	Within Time Limit	Colors
96	myciel5	2	256	10000.100000	False	6
97	myciel5	4	256	10000.100000	False	6
98	myciel5	8	256	10000.900000	False	6
99	myciel5	16	256	10000.200000	False	6
100	myciel5	32	256	10000.300000	False	6
101	myciel7	1	256	10002.200000	False	8
102	myciel7	2	256	10000.200000	False	8
103	myciel7	4	256	10000.100000	False	8
104	myciel7	8	256	10000.200000	False	8
105	myciel7	16	256	10000.900000	False	8
106	myciel7	32	256	10000.200000	False	8
107	queen10_10	1	256	10002.200000	False	13
108	queen10_10	2	256	10000.200000	False	13
109	queen10_10	4	256	10000.200000	False	12
110	queen10_10	8	256	10000.400000	False	12
111	queen10_10	16	256	10000.500000	False	12
112	queen10_10	32	256	10000.300000	False	12
113	queen10_10	64	256	10000.700000	False	12
114	queen11_11	1	256	10002.300000	False	14
115	queen11_11	2	256	10000.300000	False	14
116	queen11_11	4	256	10000.100000	False	13
117	queen11_11	8	256	10000.200000	False	14
118	queen11_11	16	256	10000.600000	False	13
119	queen11_11	32	256	10000.600000	False	14
120	queen11_11	64	256	10000.200000	False	14
121	queen12_12	1	256	10002.600000	False	16
122	queen12_12	2	256	10000.200000	False	15
123	queen12_12	4	256	10000.100000	False	15
124	queen12_12	8	256	10000.400000	False	15
125	queen12_12	32	256	10000.400000	False	15
126	queen12_12	64	256	10000.500000	False	15
127	queen13_13	1	256	10003.100000	False	17
128	queen13_13	2	256	10000.800000	False	17
129	queen13_13	4	256	10000.300000	False	17
130	queen13_13	8	256	10000.200000	False	17
131	queen13_13	16	256	10000.300000	False	17
132	queen13_13	32	256	10000.300000	False	17
133	queen14_14	1	256	10002.300000	False	18
134	queen14_14	2	256	10000.900000	False	18
135	queen14_14	4	256	10000.400000	False	18
136	queen14_14	8	256	10000.200000	False	18
137	queen14_14	16	256	10000.300000	False	18
138	queen14_14	32	256	10000.300000	False	18
139	queen14_14	64	256	10000.700000	False	18
140	queen15_15	1	256	10002.200000	False	20
141	queen15_15	2	256	10000.900000	False	19
142	queen15_15	4	256	10000.200000	False	19
143	queen15_15	8	256	10000.500000	False	19
144	queen15_15	16	256	10000.800000	False	19
145	queen15_15	32	256	10000.300000	False	19

Index	Instance	MPI Processes	Threads per Process	Wall Time (sec)	Within Time Limit	Colors
146	queen15_15	64	256	10000.400000	False	19
147	queen5_5	1	256	3.309270	True	5
148	queen5_5	2	256	14.462900	True	5
149	queen5_5	4	256	2.015740	True	5
150	queen5_5	8	256	1.394300	True	5
151	queen5_5	16	256	1.008740	True	5
152	queen5_5	32	256	2.452500	True	5
153	queen5_5	64	256	1.536890	True	5
154	queen6_6	1	256	4331.910000	True	7
155	queen6_6	2	256	2068.780000	True	7
156	queen6_6	4	256	4274.440000	True	7
157	queen6_6	8	256	1867.000000	True	7
158	queen6_6	16	256	1323.990000	True	7
159	queen6_6	32	256	3316.220000	True	7
160	queen7_7	1	256	7591.180000	True	7
161	queen7_7	2	256	2942.340000	True	7
162	queen7_7	4	256	4119.200000	True	7
163	queen7_7	8	256	3269.300000	True	7
164	queen7_7	16	256	1576.720000	True	7
165	queen7_7	32	256	2486.450000	True	7
166	queen7_7	64	256	3869.300000	True	7
167	queen8_12	1	256	1774.160000	True	12
168	queen8_12	2	256	284.779000	True	12
169	queen8_12	4	256	167.984000	True	12
170	queen8_12	8	256	85.595600	True	12
171	queen8_12	16	256	48.866200	True	12
172	queen8_12	64	256	31.742100	True	12
173	queen8_8	1	256	10002.300000	False	10
174	queen8_8	2	256	9999.820000	True	9
175	queen8_8	4	256	10000.500000	True	9
176	queen8_8	8	256	10000.400000	True	9
177	queen8_8	16	256	10000.300000	False	10
178	queen8_8	32	256	10000.200000	True	9
179	queen8_8	64	256	10000.500000	True	9
180	queen9_9	1	256	10001.900000	False	11
181	queen9_9	2	256	10000.300000	False	10
182	queen9_9	4	256	10000.200000	False	10
183	queen9_9	8	256	10000.100000	False	10
184	queen9_9	16	256	10000.100000	False	10
185	queen9_9	64	256	10000.100000	False	10
186	school1	8	256	10072.900000	False	29
187	school1	16	256	10081.600000	False	29
188	school1	32	256	10064.400000	False	29
189	zeroin.i.1	1	256	67.076900	True	49
190	zeroin.i.1	2	256	10.521200	True	49
191	zeroin.i.1	4	256	11.100300	True	49
192	zeroin.i.1	8	256	0.844417	True	49
193	zeroin.i.1	32	256	0.833230	True	49
194	zeroin.i.1	64	256	1.778100	True	49

B Benchmarks results: instances_optional

Index	Instance	MPI Processes	Threads per Process	Wall Time (sec)	Within Time Limit	Colors
0	fpsol2.i.2	1	256	61.63380	True	30
1	fpsol2.i.2	2	256	17.29340	True	30
2	fpsol2.i.2	4	256	11.55750	True	30
3	fpsol2.i.2	8	256	16.08670	True	30
4	fpsol2.i.2	16	256	13.45350	True	30
5	fpsol2.i.2	32	256	8.16692	True	30
6	fpsol2.i.2	64	256	7.36674	True	30
7	fpsol2.i.3	1	256	46.28510	True	30
8	fpsol2.i.3	2	256	9.94497	True	30
9	fpsol2.i.3	4	256	8.52328	True	30
10	fpsol2.i.3	8	256	10.47860	True	30
11	fpsol2.i.3	16	256	13.50430	True	30
12	fpsol2.i.3	32	256	7.88846	True	30
13	inithx.i.1	1	256	204.70500	True	54
14	inithx.i.1	2	256	44.42050	True	54
15	inithx.i.1	4	256	16.82530	True	54
16	inithx.i.1	8	256	25.03310	True	54
17	inithx.i.1	16	256	15.71330	True	54
18	inithx.i.1	32	256	13.04420	True	54
19	inithx.i.2	1	256	61.03930	True	31
20	inithx.i.2	2	256	12.33090	True	31
21	inithx.i.2	4	256	11.39340	True	31
22	inithx.i.2	8	256	15.57990	True	31
23	inithx.i.2	16	256	11.11510	True	31
24	le450_15a	4	256	10000.50000	False	16
25	le450_15a	8	256	10000.60000	False	16
26	le450_15a	16	256	10000.70000	False	16
27	le450_15a	32	256	10000.50000	False	16
28	le450_15a	64	256	10000.70000	False	16
29	le450_15b	4	256	10000.50000	False	16
30	le450_15b	8	256	10000.60000	False	16
31	le450_15b	16	256	10000.60000	False	16
32	le450_15b	32	256	10000.60000	False	16
33	le450_15b	64	256	10000.60000	False	16
34	le450_15c	4	256	10001.30000	False	24
35	le450_15c	8	256	10001.30000	False	24
36	le450_15c	16	256	10001.50000	False	24
37	le450_15c	32	256	10001.50000	False	24
38	le450_15d	4	256	10002.60000	False	23
39	le450_15d	8	256	10001.60000	False	23
40	le450_15d	16	256	10002.60000	False	23
41	le450_15d	32	256	10001.60000	False	23
42	le450_15d	64	256	10001.20000	False	23
43	le450_25a	1	256	7.01781	True	25
44	le450_25a	2	256	6.93015	True	25
45	le450_25a	4	256	2.92598	True	25
46	le450_25a	8	256	9.71740	True	25

Index	Instance	MPI Processes	Threads per Process	Wall Time (sec)	Within Time Limit	Colors
47	le450_25a	16	256	6.96311	True	25
48	le450_25a	32	256	4.33562	True	25
49	le450_25b	1	256	7.01981	True	25
50	le450_25b	2	256	2.14268	True	25
51	le450_25b	4	256	3.56686	True	25
52	le450_25b	8	256	9.35911	True	25
53	le450_25b	16	256	7.58611	True	25
54	le450_25b	32	256	11.80220	True	25
55	le450_5a	2	256	10001.30000	False	10
56	le450_5a	4	256	10000.30000	False	10
57	le450_5a	8	256	10000.50000	False	10
58	le450_5a	16	256	10000.40000	False	10
59	le450_5a	32	256	10000.30000	False	10
60	le450_5a	64	256	10000.30000	False	10
61	le450_5c	2	256	510.42300	True	5
62	le450_5c	4	256	10000.30000	True	5
63	le450_5c	8	256	10000.40000	True	5
64	le450_5c	16	256	10000.40000	True	5
65	le450_5c	32	256	10000.30000	True	5
66	le450_5d	2	256	547.80600	True	5
67	le450_5d	4	256	10000.30000	True	5
68	le450_5d	8	256	10000.40000	True	5
69	le450_5d	16	256	10000.40000	True	5
70	le450_5d	32	256	10000.40000	True	5
71	miles1000	1	256	189.67900	True	42
72	miles1000	2	256	51.97930	True	42
73	miles1000	4	256	25.77780	True	42
74	miles1000	8	256	68.71020	True	42
75	miles1000	16	256	15.58270	True	42
76	miles1000	64	256	19.10160	True	42
77	miles500	1	256	5.63714	True	20
78	miles500	2	256	6.90468	True	20
79	miles500	4	256	2.34719	True	20
80	miles500	8	256	5.59225	True	20
81	miles500	16	256	5.61099	True	20
82	miles500	64	256	4.43952	True	20
83	miles750	1	256	23.29990	True	31
84	miles750	2	256	19.07820	True	31
85	miles750	4	256	11.92560	True	31
86	miles750	8	256	16.97310	True	31
87	miles750	16	256	30.01960	True	31
88	miles750	32	256	16.75180	True	31
89	mulsol.i.2	1	256	16.05230	True	31
90	mulsol.i.2	2	256	8.87010	True	31
91	mulsol.i.2	4	256	7.31614	True	31
92	mulsol.i.2	8	256	10.56910	True	31
93	mulsol.i.2	16	256	7.31242	True	31
94	mulsol.i.2	32	256	7.64149	True	31
95	mulsol.i.2	64	256	4.69194	True	31
96	mulsol.i.3	1	256	10.97170	True	31

Index	Instance	MPI Processes	Threads per Process	Wall Time (sec)	Within Time Limit	Colors
97	mulsol.i.3	2	256	8.29819	True	31
98	mulsol.i.3	4	256	5.27943	True	31
99	mulsol.i.3	8	256	10.52900	True	31
100	mulsol.i.3	16	256	7.44054	True	31
101	mulsol.i.3	64	256	6.47482	True	31
102	mulsol.i.4	1	256	12.79640	True	31
103	mulsol.i.4	2	256	8.07961	True	31
104	mulsol.i.4	4	256	3.40781	True	31
105	mulsol.i.4	8	256	7.40275	True	31
106	mulsol.i.4	16	256	4.80688	True	31
107	mulsol.i.4	32	256	8.28211	True	31
108	mulsol.i.5	1	256	12.89190	True	31
109	mulsol.i.5	2	256	8.45557	True	31
110	mulsol.i.5	4	256	6.43328	True	31
111	mulsol.i.5	8	256	10.51580	True	31
112	mulsol.i.5	16	256	10.12970	True	31
113	mulsol.i.5	32	256	6.04432	True	31
114	mulsol.i.5	64	256	6.15596	True	31
115	myciel6	1	256	10000.10000	False	7
116	myciel6	2	256	10000.10000	False	7
117	myciel6	4	256	10000.10000	False	7
118	myciel6	8	256	10000.10000	False	7
119	myciel6	16	256	10000.10000	False	7
120	myciel6	64	256	10000.10000	False	7
121	queen16_16	2	256	10000.70000	False	20
122	queen16_16	4	256	10000.30000	False	20
123	queen16_16	8	256	10000.20000	False	20
124	queen16_16	16	256	10000.30000	False	20
125	queen16_16	32	256	10000.30000	False	20
126	queen16_16	64	256	10000.30000	False	20
127	zeroin.i.2	1	256	39.17890	True	30
128	zeroin.i.2	2	256	14.50310	True	30
129	zeroin.i.2	4	256	10.97050	True	30
130	zeroin.i.2	8	256	14.36730	True	30
131	zeroin.i.2	16	256	10.37460	True	30
132	zeroin.i.2	32	256	12.82700	True	30
133	zeroin.i.2	64	256	6.12948	True	30
134	zeroin.i.3	1	256	35.87730	True	30
135	zeroin.i.3	2	256	12.84560	True	30
136	zeroin.i.3	4	256	8.07518	True	30
137	zeroin.i.3	8	256	12.22500	True	30
138	zeroin.i.3	16	256	10.74100	True	30
139	zeroin.i.3	32	256	12.53780	True	30
140	zeroin.i.3	64	256	6.38595	True	30