Considerations on Parallel Graph Coloring Algorithms

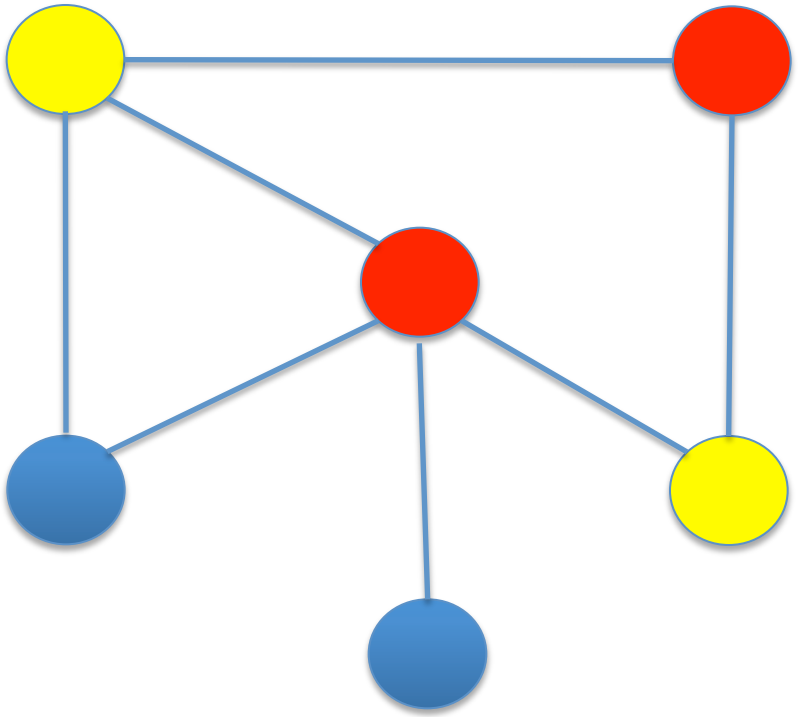
Ahmet Erdem Sarıyüce[1,2], Erik Saule[2] and Ümit V. Çatalyürek[2,3]

[1]Department of Computer Science and Engineering
[2]Department of Biomedical Informatics
[3]Department of Electric and Computer Engineering
The Ohio State University

Introduction

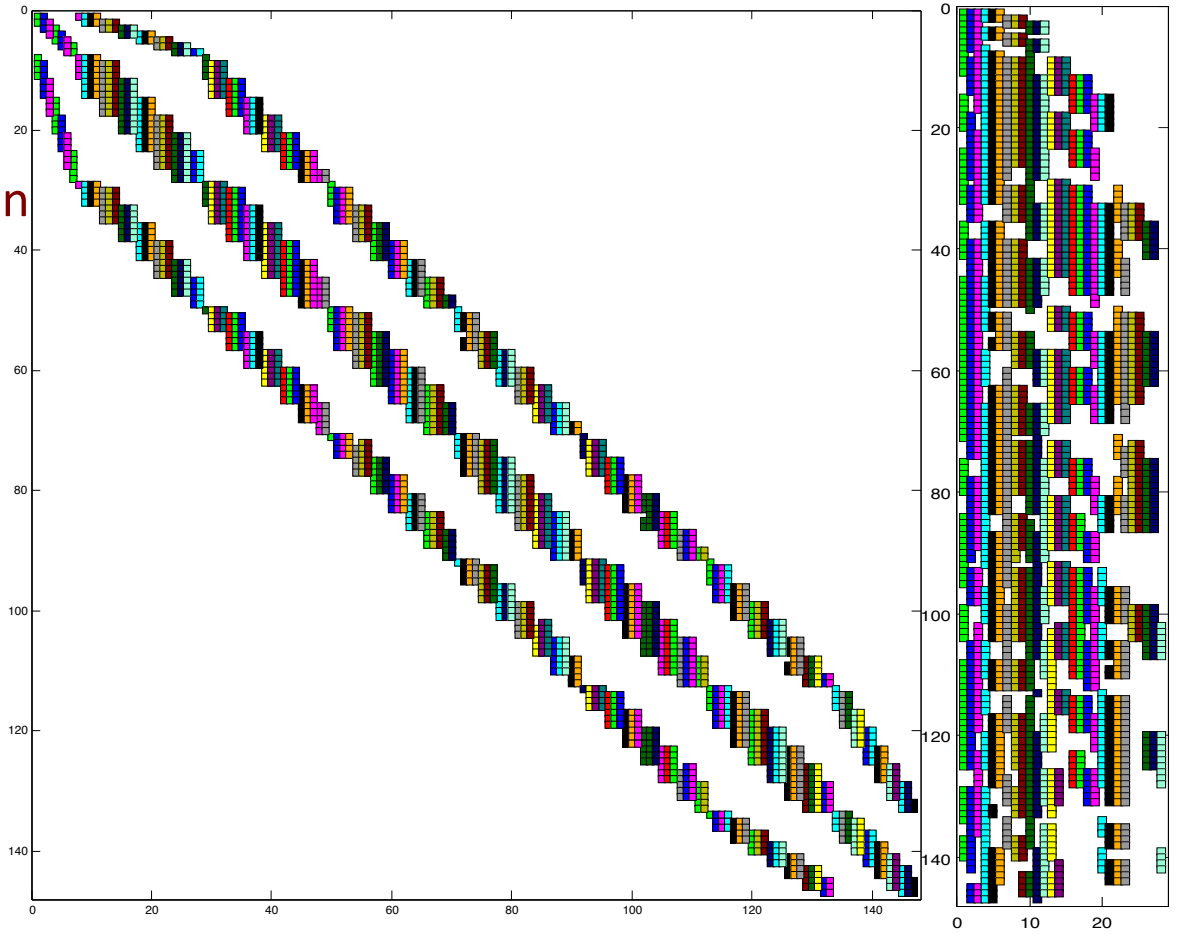
- Graph coloring is a combinatorial problem which consists of partitioning a graph in a minimum number of independent sets
  - The most classical variant of the problem is the distance-1 coloring problem where two adjacent vertices must have different colors

- The problem of finding the minimal number of colors a graph can be colored with is NP-Hard



Application Areas

- Major application areas
  - Parallel computation
  - Automatic differentiation
  - Jacobian computation
  - Printed circuit testing
  - Frequency assignment
  - Register allocation
  - Optimization

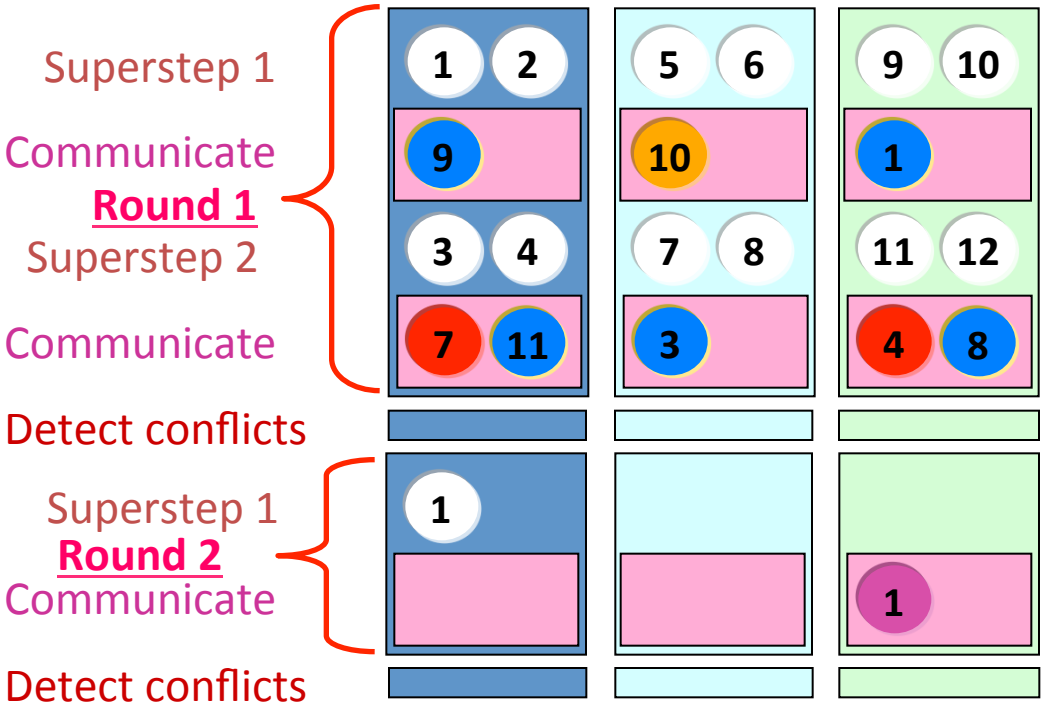


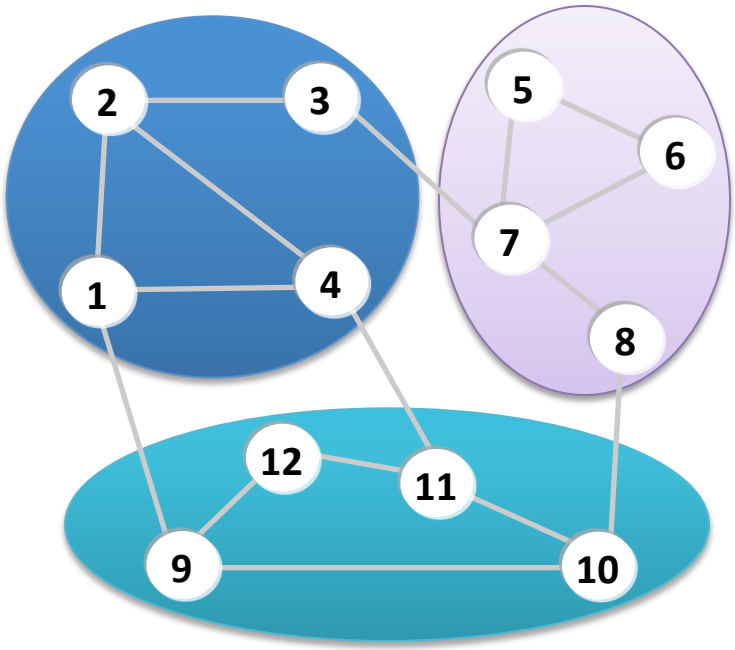
Distributed Memory Coloring Framework

- Bozdag et al. introduced a distributed-memory graph coloring framework (2008)

Distributed Memory Coloring Framework

- Bozdag et al. introduced a distributed-memory graph coloring framework (2008)





Distributed Memory Coloring Framework

- There are many parameters affecting the quality and runtime of the coloring
  - **Color Selection Strategies:** First Fit, Staggered First Fit
  - **Coloring Order:** Interior First, Boundary First, Unordered
  - **Superstep Size**
  - **Superstep Synchronization:** Synchronous, Asynchronous
  - **Inter-processor communication:** Customized, Broadcasted

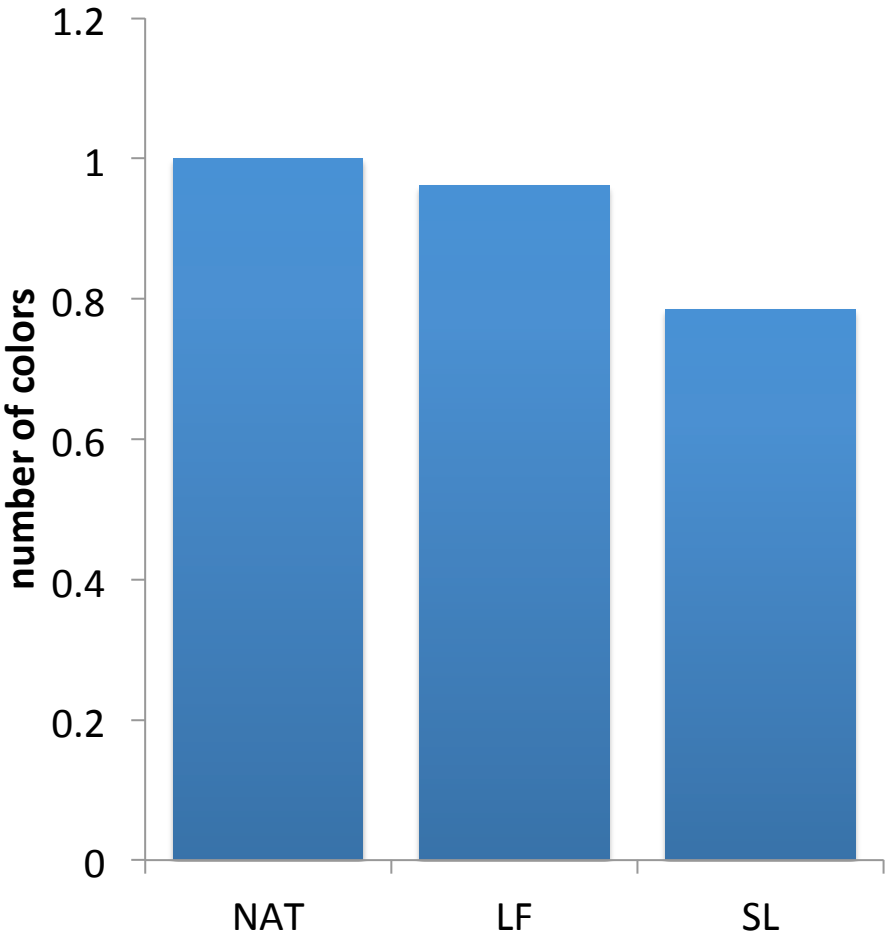
- FIA is the fastest combination whereas FBS gives the best number of colors

Contributions

- Two orthogonal improvement techniques
  - Two distributed memory vertex visit orderings
    - Largest First Ordering
    - Smallest Last Ordering
  - Post-processing procedure, called Recoloring
- Hybrid (distributed + shared memory) parallelization of graph coloring

Two vertex visit orderings

- Largest First Ordering
  - Vertices are sorted based on their degrees in non-increasing order and colored in that order (Welsh, 1967)
- Smallest Last Ordering
  - Approaches the ordering from backwards by selecting a vertex with the minimum degree to be ordered last and removing it from the graph for the rest of the ordering phase. This procedure is repeated until all vertices are ordered (Matula, 1968)



Parallel vertex visit orderings

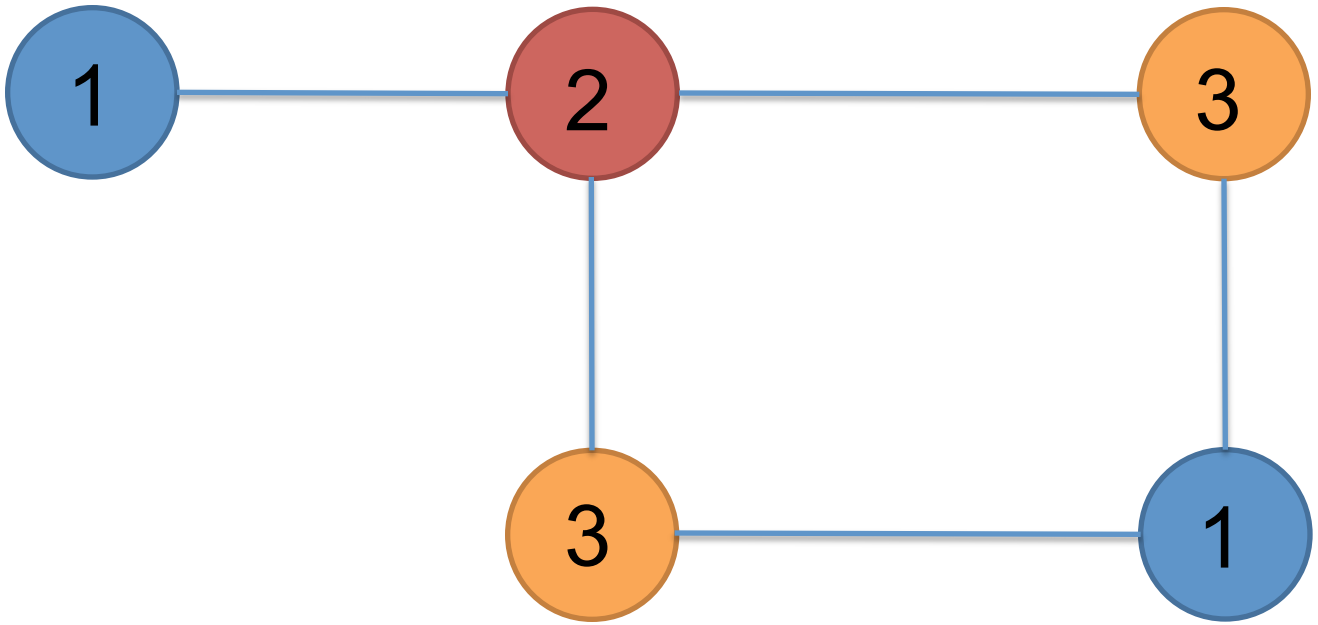
- The graph is distributed in the memory of the processors
  - The ordering is computed by each processor independently for the vertices it owns
  - No global ordering is done since it would be too expensive

Recoloring

- Post-processing procedure, called Recoloring
  - First introduced by (Culberson, 1992) as iterated greedy coloring for sequential settings
  - Recoloring the whole graph based on the previous coloring results
  - Number of colors either decrease or stay same
- Different color class permutations:
  - Reverse order
  - Non-Increasing cardinality
  - Non-Decreasing cardinality

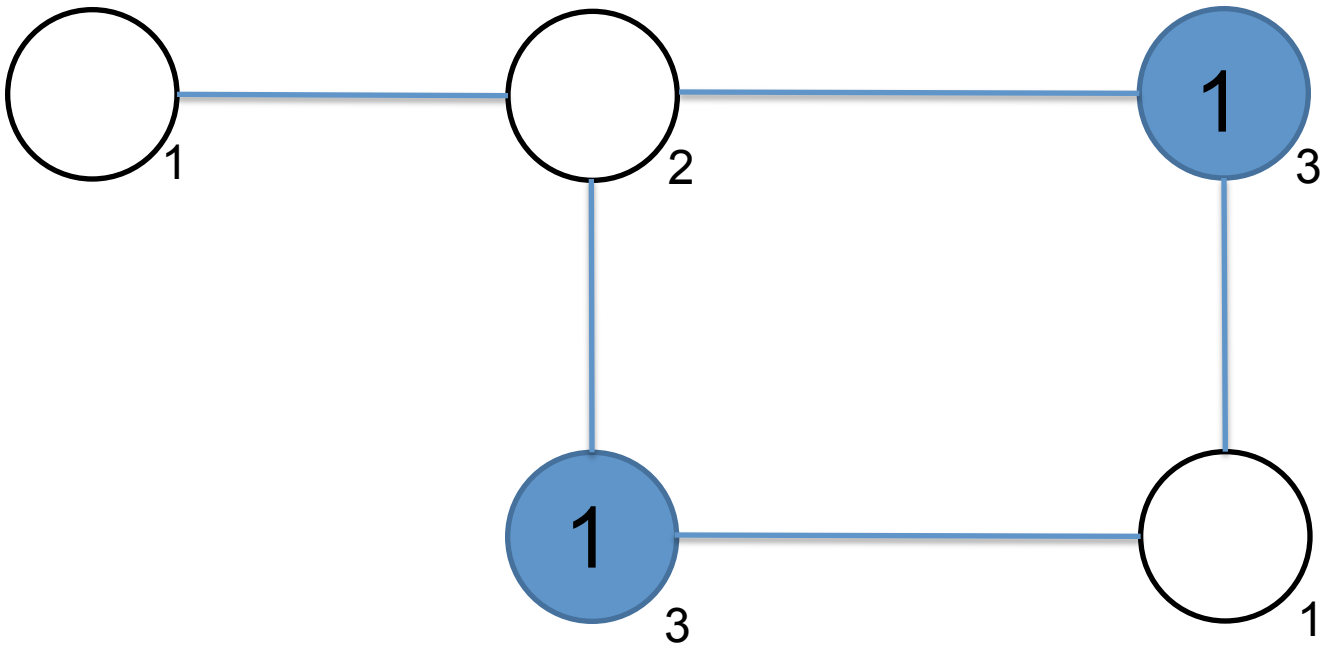
Recoloring with Reverse permutation

- Assume that the following graph is colored with 3 colors as a result of the initial coloring:



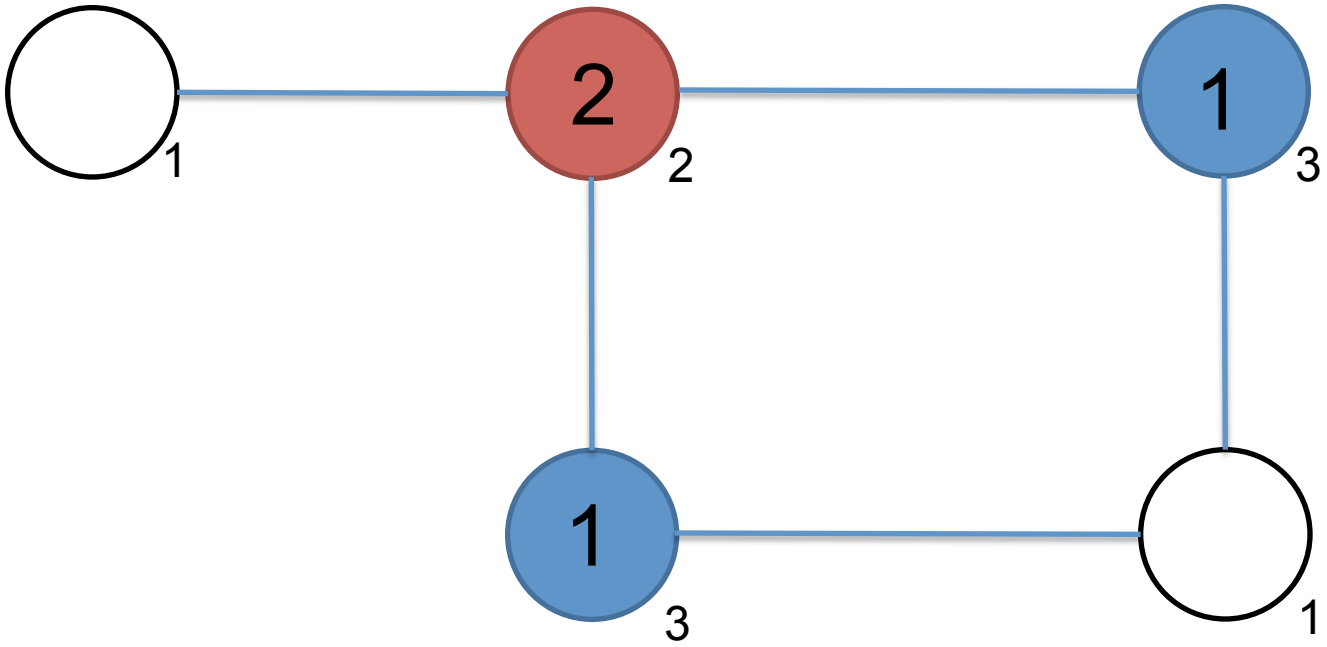
Recoloring with Reverse permutation

- Then, we recolor the graph in reverse permutation of color classes, i.e. firstly we recolor vertices with color 3,



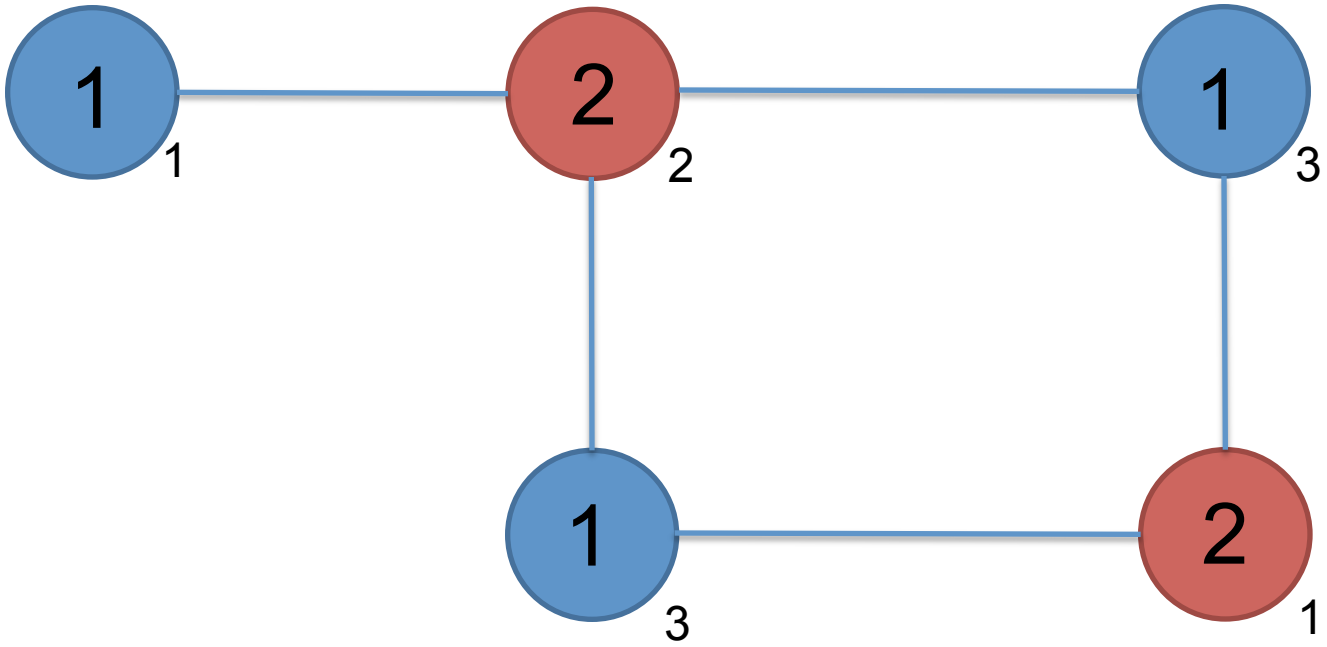
Recoloring with Reverse permutation

- Then, we recolor vertices with color 2,

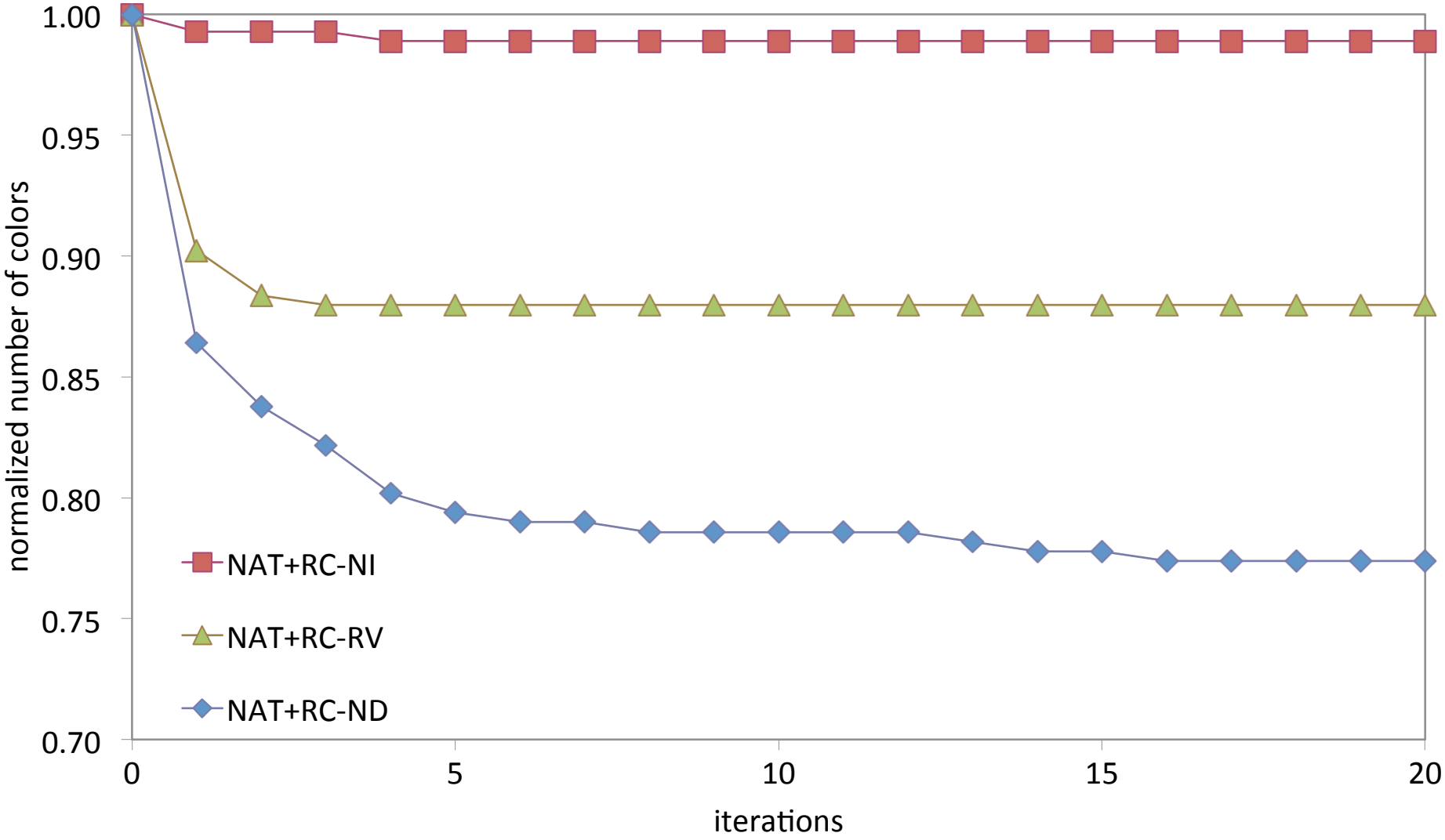


Synchronous recoloring with Reverse permutation

- Lastly, we recolor vertices with color 1,
- As a result we reduced the total number of colors to 2 !



Sequential study on real graphs



Parallel Recoloring

- Recoloring the vertices by using the information obtained at the end of the first coloring
  - Each processor independently colors all the vertices with the same color in the previous solution
  - Then, processors exchange color information

- It fits distributed memory
  - The same solution is obtained with sequential coloring
  - No conflicts occur

Experimental Setup

- All the algorithms are implemented in Zoltan
- All the algorithms are tested on an in-house cluster consisting of 64 nodes:
  - Each node has 2 Intel Xeon E5520 (quad-core clocked at 2.27 Ghz with Hyper-Threading) processors
  - Nodes are connected with 20Gbps DDR infiniband
- The experiments are run on 6 real-world application graphs and 3 randomly generated graphs (omitted)

Experiment Graphs

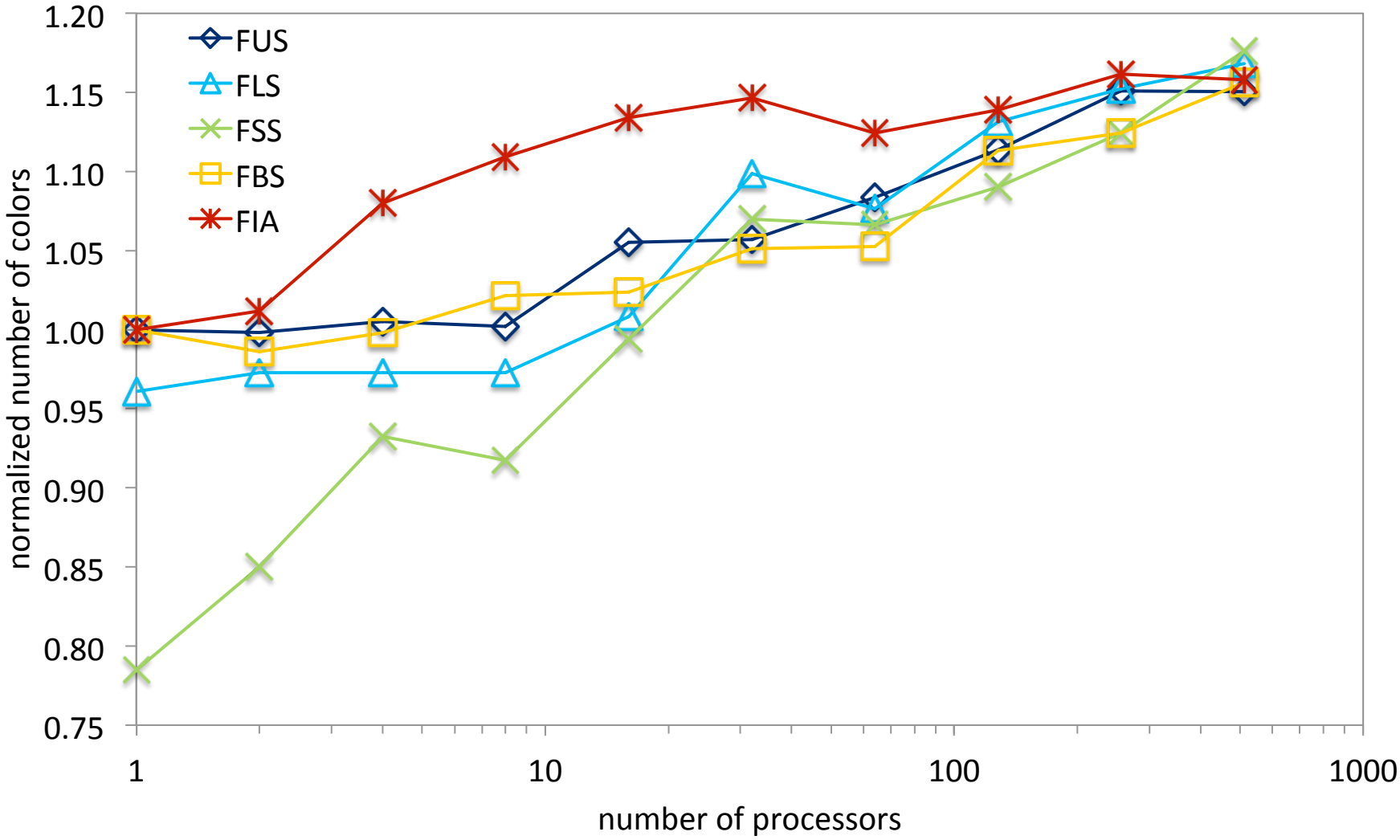
Name	$\|V\|$	$\|E\|$	Δ	NAT	LF	SL	seq time
auto	448K	3.3M	37	13	12	10	0.1103s
bmw3_2	227K	5.5M	335	48	48	37	0.0836s
hood	220K	4.8M	76	40	39	34	0.0752s
ldoor	952K	20.7M	76	42	42	34	0.3307s
msdoor	415K	9.3M	76	42	42	35	0.1458s
pwtk	217K	5.6M	179	48	42	33	0.0820s

Table 1. Properties of real-world graphs

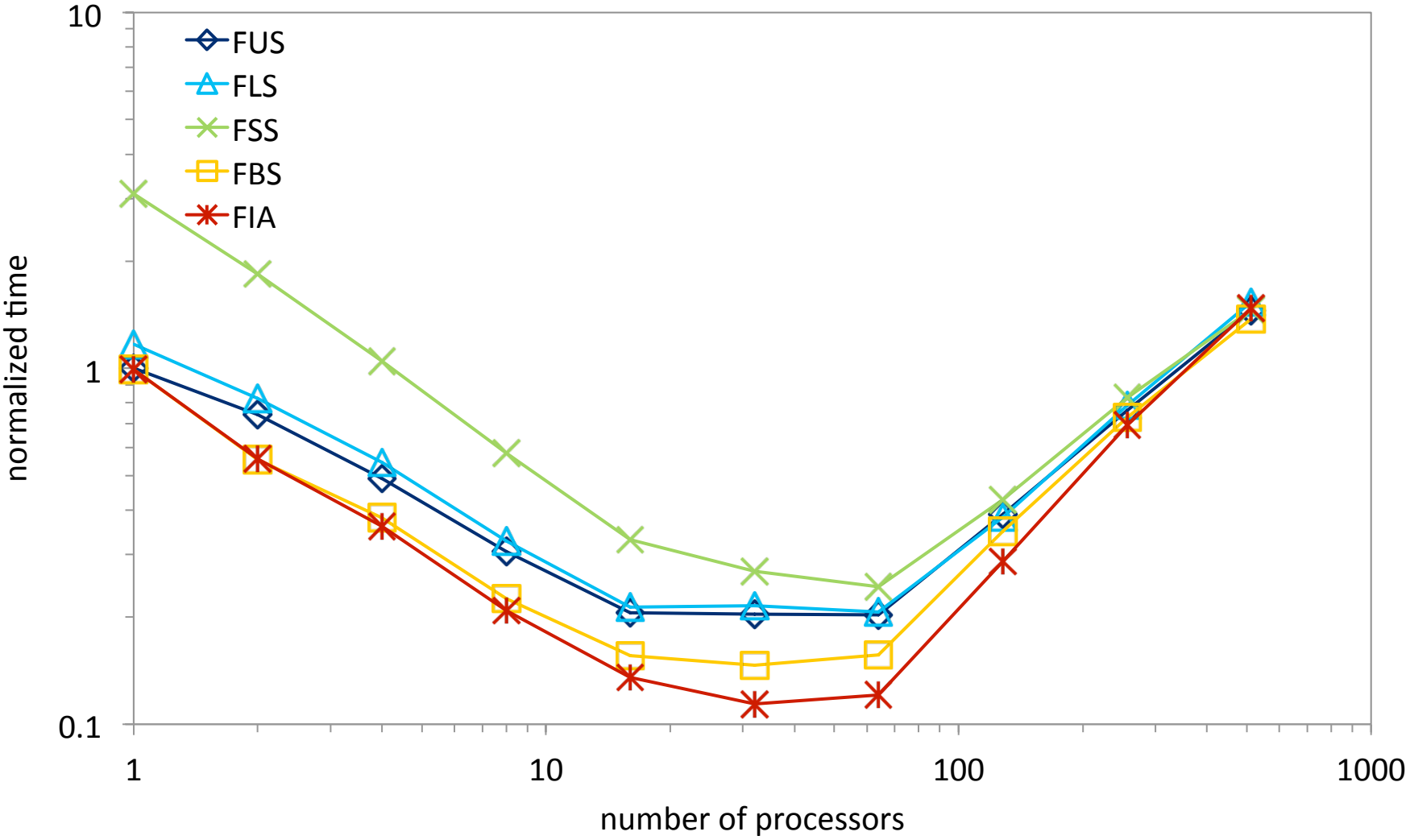
Name	$\|V\|$	$\|E\|$	Δ	NAT	LF	SL
ER	16,777,216	134,217,624	42	12	10	10
Good	16,777,216	134,181,065	1,278	28	15	14
Bad	16,777,216	133,658,199	38,143	146	89	88

Table 2. Properties of synthetic graphs

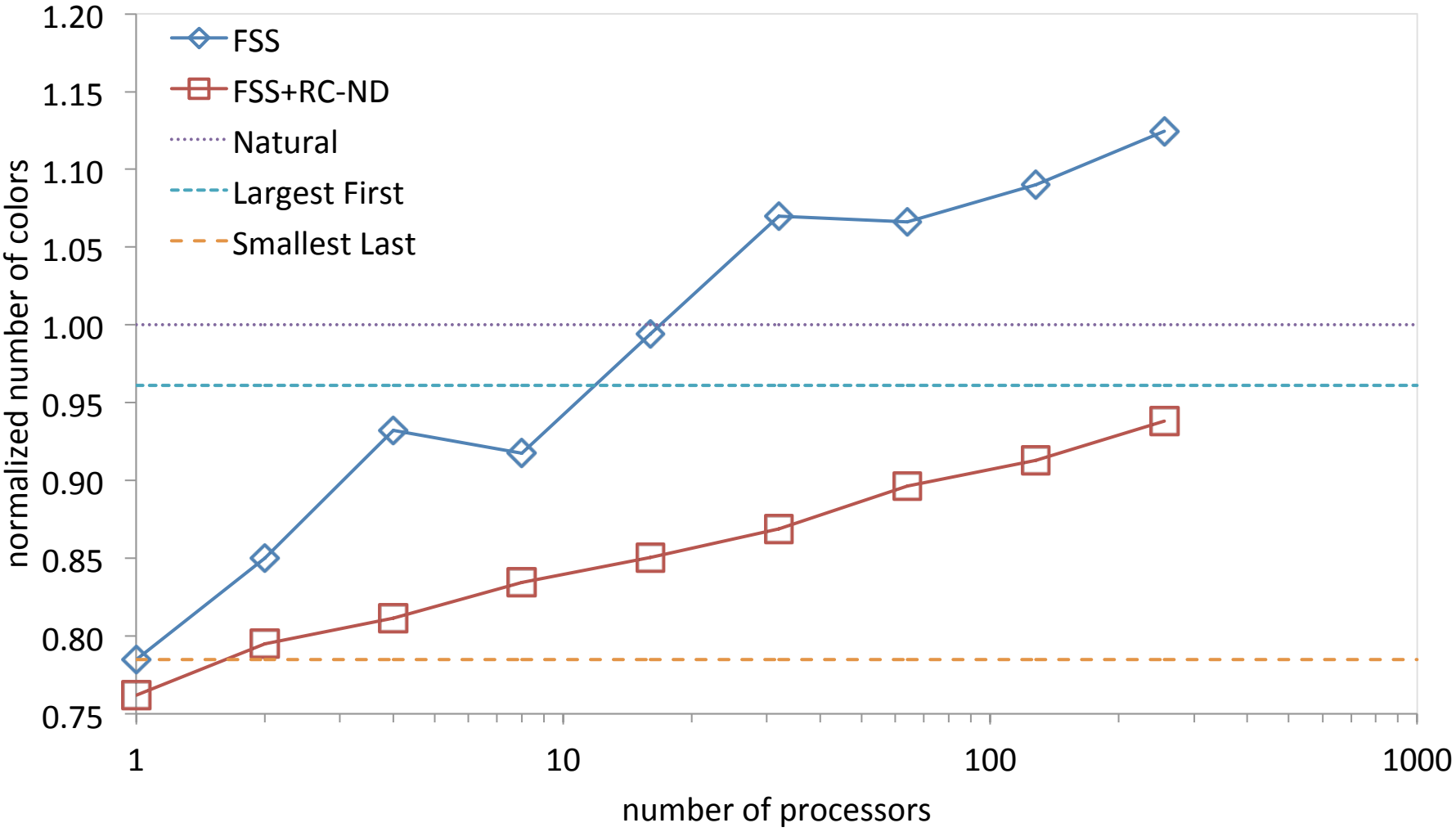
Comparison of ordering on real graphs



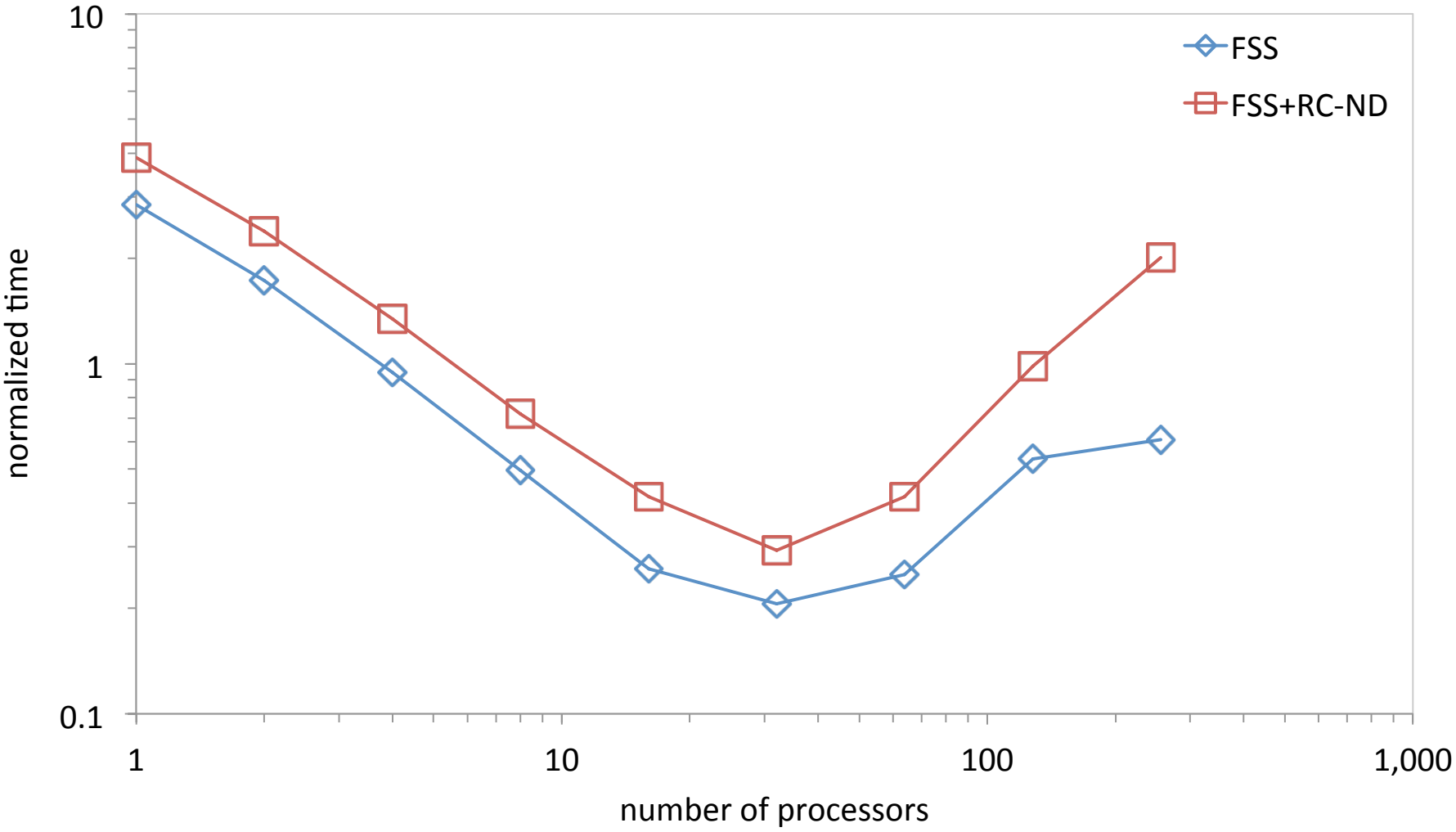
Comparison of ordering on real graphs



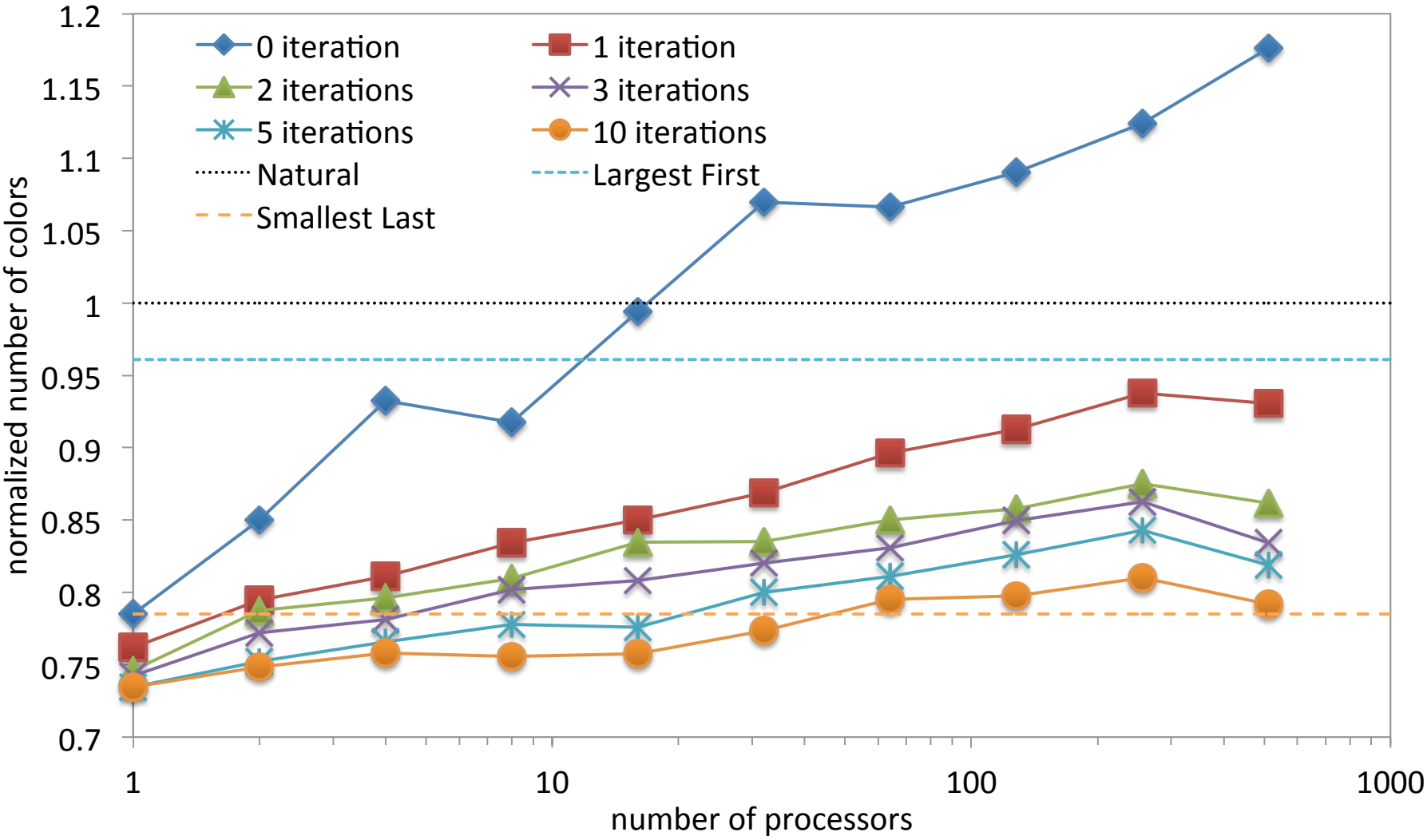
Comparison of recoloring on real graphs with SL ordering



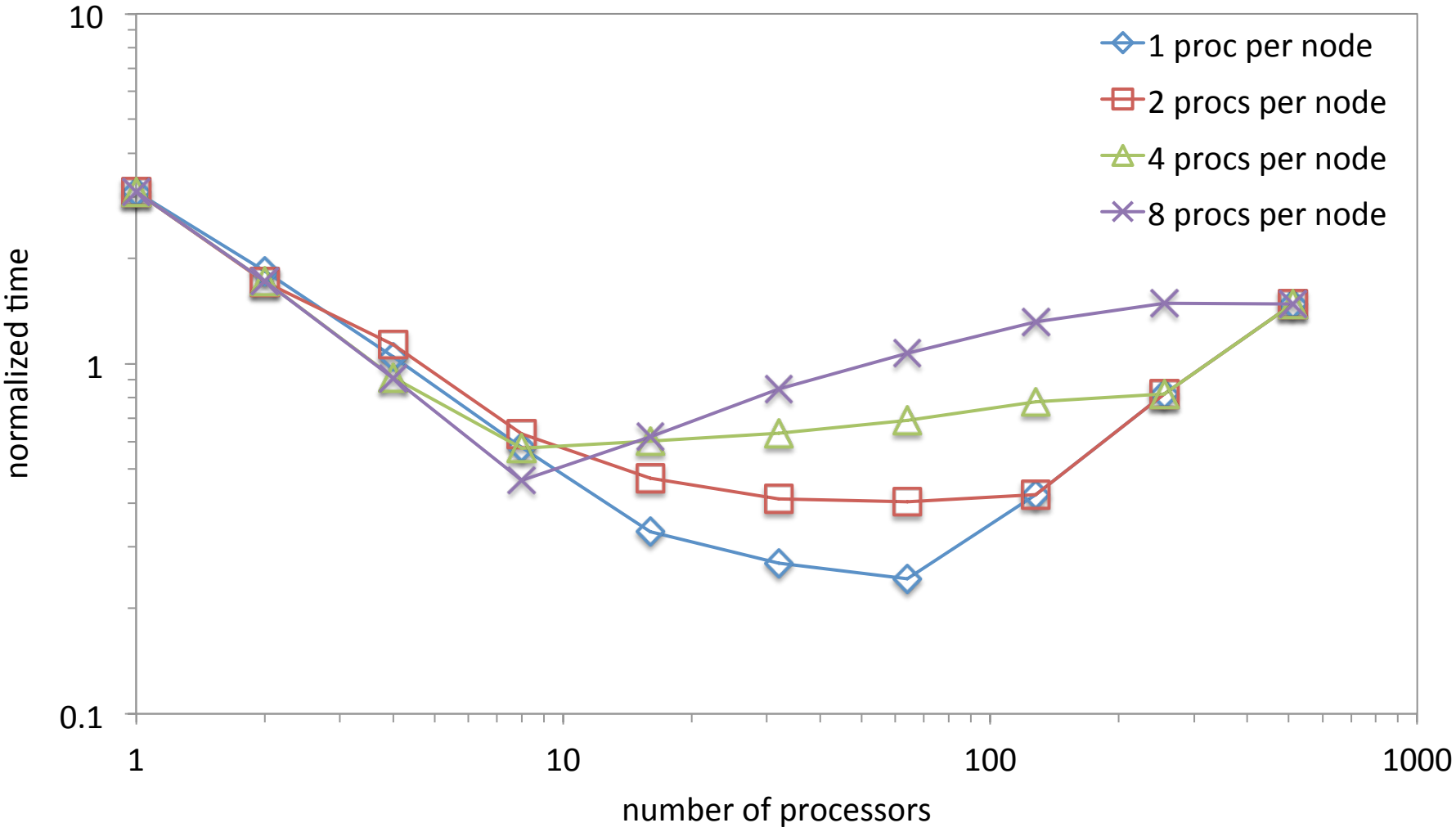
Comparison of recoloring on real graphs with SL ordering



Impact of number of iterations in distributed memory



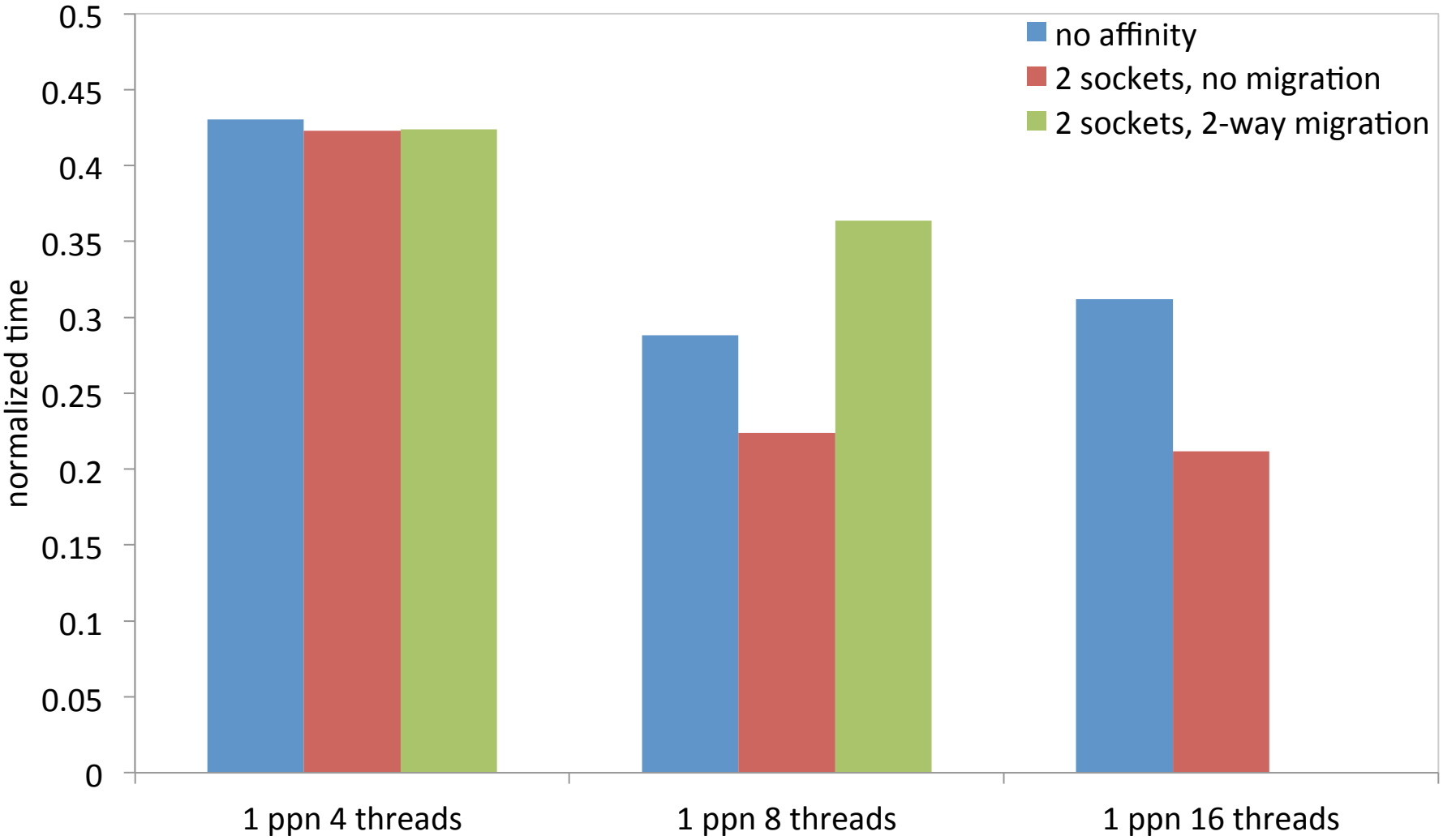
Impact of ppn allocation policies on real graphs



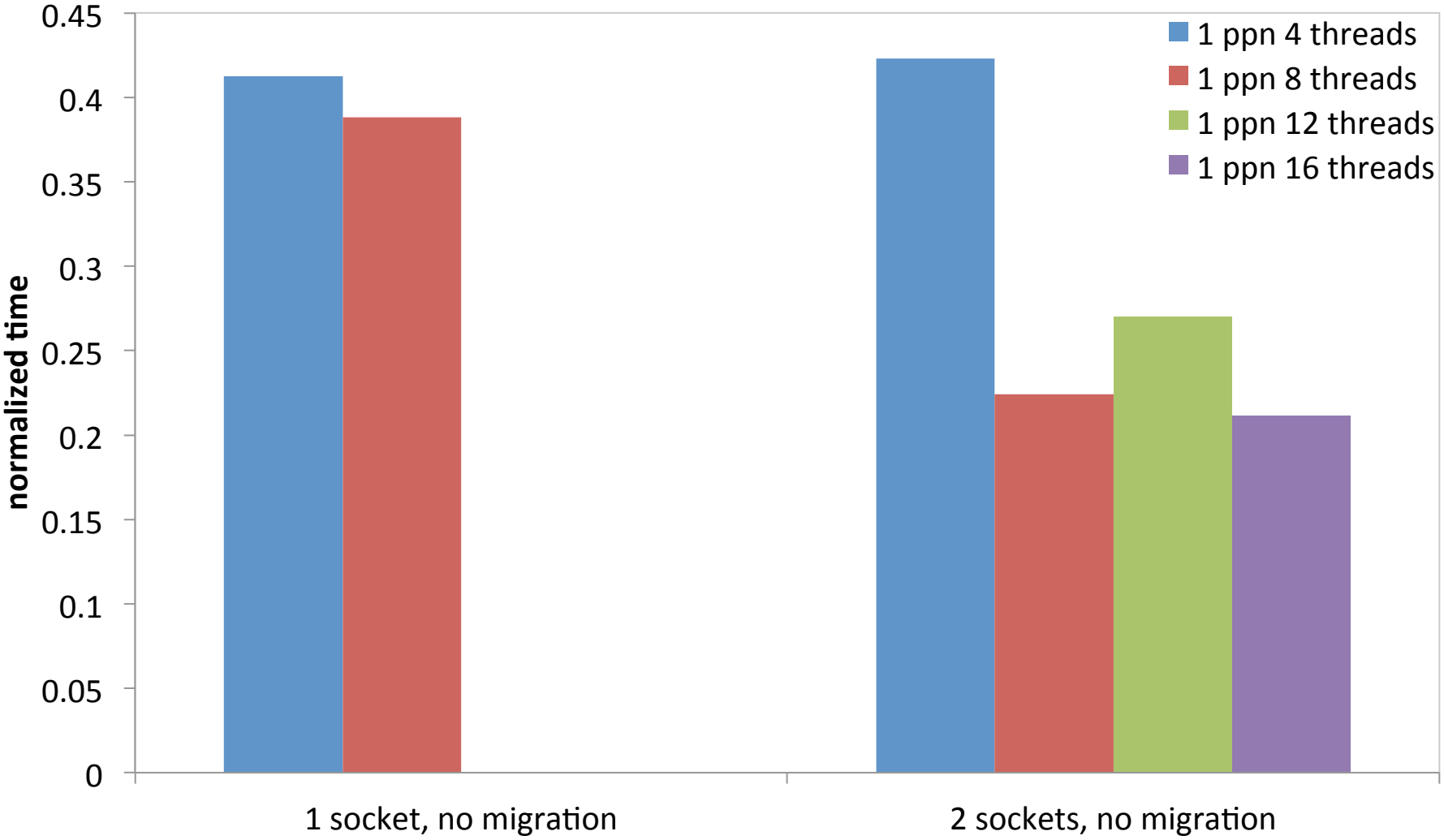
Hybrid coloring

- This problem motivates a hybrid approach where OpenMP is used for intra-node communication and MPI is used for inter-node communication.
- Many sources exist for shared memory parallelism
  - Construction of random keys
  - Reordering the vertices (boundary first, internal last)
  - Greedy coloring
    - Vertices are partitioned to multiple threads
    - Separate cache aligned mark arrays are used for each thread

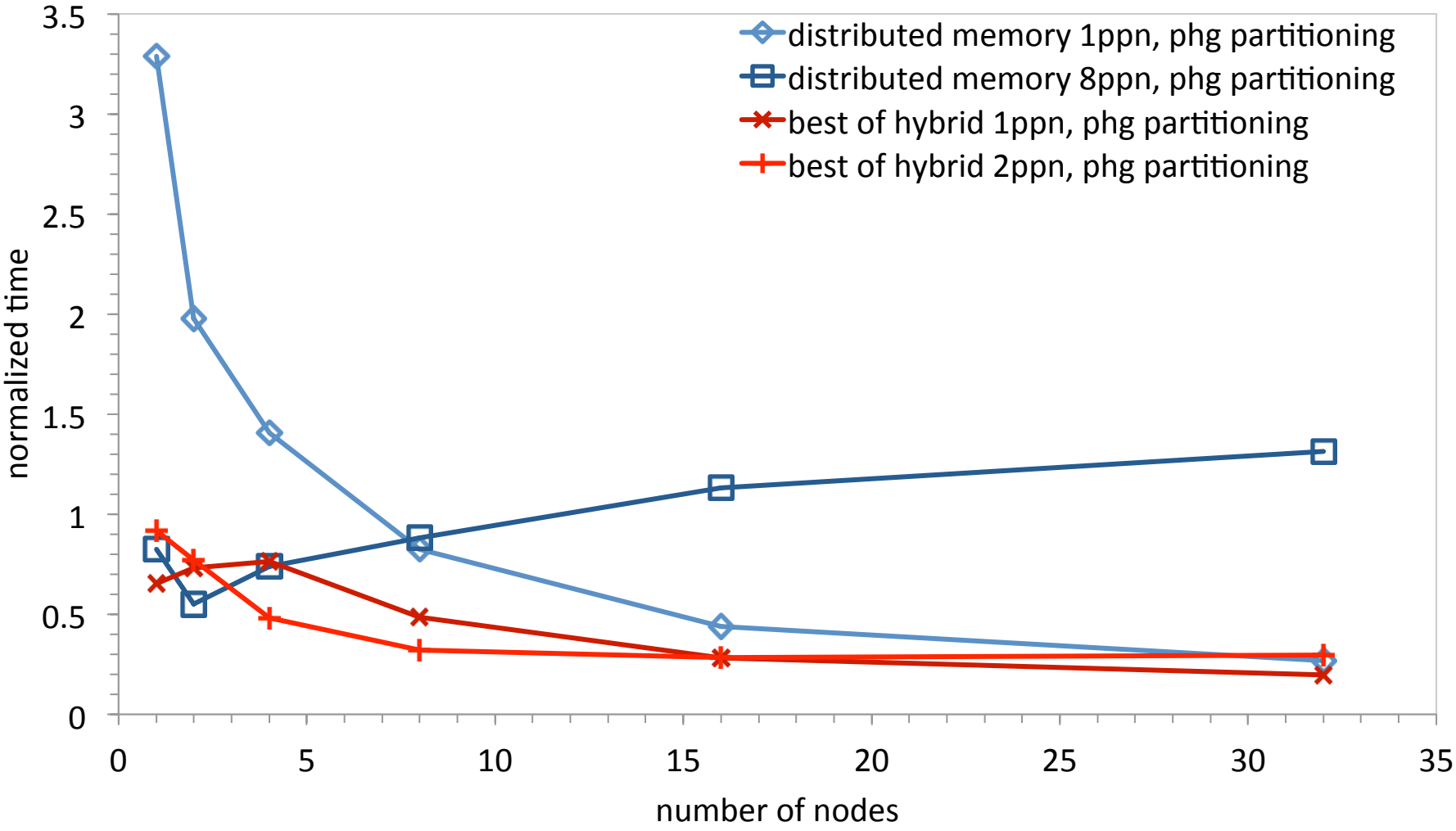
Impact of affinity policies in a single node



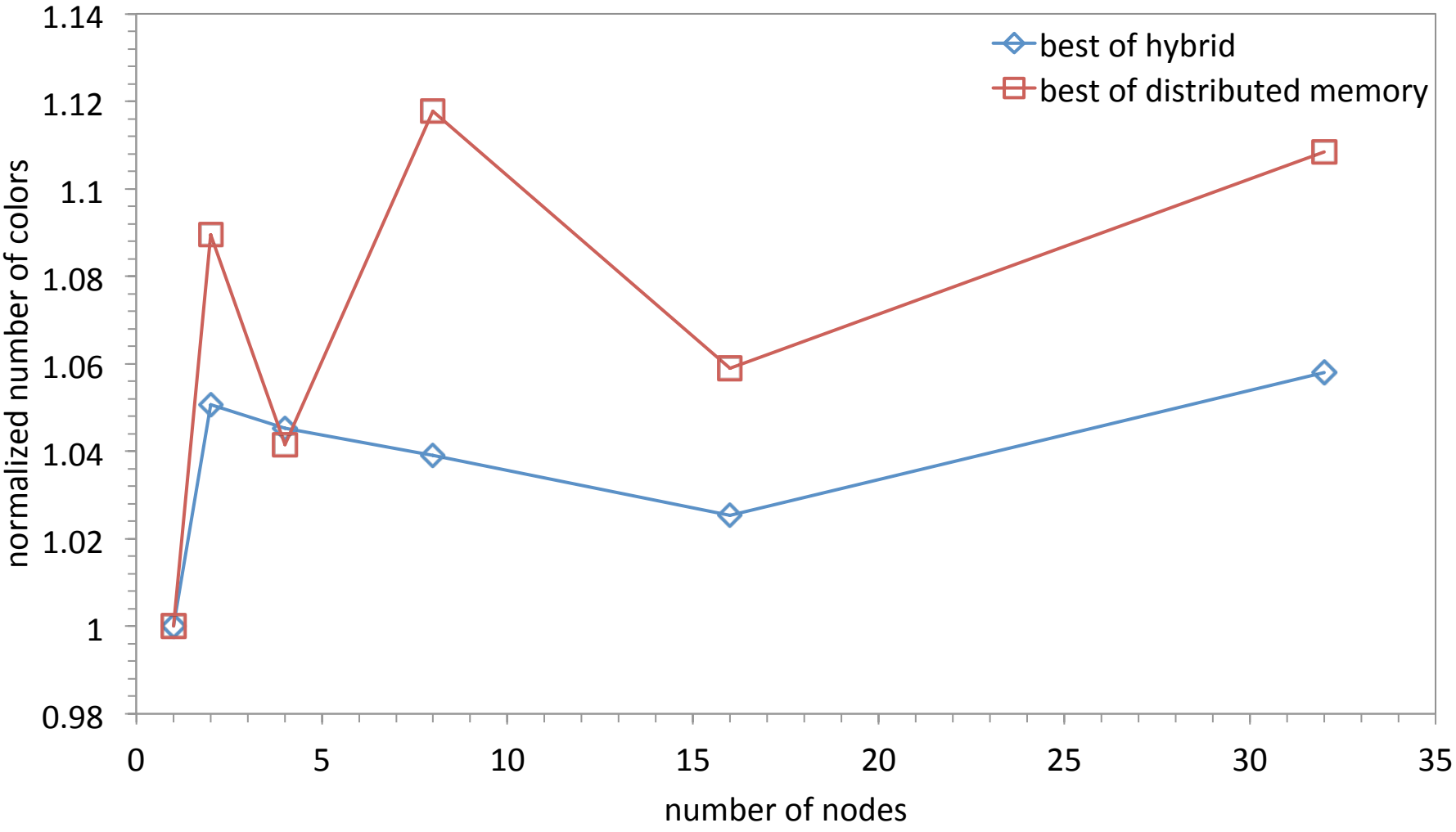
Impact of affinity policies in a single node



Comparison of hybrid and distributed-memory colorings



Number of colors comparisons



Conclusions

- Two different, but compatible, ways of improving the number of colors are introduced:
  - Using a vertex visit ordering that takes into account the properties of the graph instead of properties of the partition can yield major improvements on number of colors up to 25%
  - Recoloring to color large number of vertices independently with little synchronization and no conflicts provides 21% improvement on number of colors and brings only 30% additional runtime

Conclusions

- Hybrid parallelization for graph coloring is introduced:
  - The parameters affecting performance of hybrid execution (thread affinity, ppn policy) investigated to obtain best performance
  - Prevention of migration by pinning threads to cores and Hyper-Threading is beneficial.
  - Utilizing all the computing units of multicore clusters provides 20% to 30% improvement over distributed-memory implementation while keeping the obtained number of colors less than distributed-memory implementation

Thanks

- For more information visit
  - http://bmi.osu.edu/hpc

- Research at the HPC Lab is funded by