PARALLEL GRAPH COLORING ALGORITHMS

Margomenos Spyridon
Denzler Alain
Moesch Philippe

Graph coloring

- Given G = (V,E) → for all (u,v) in E :

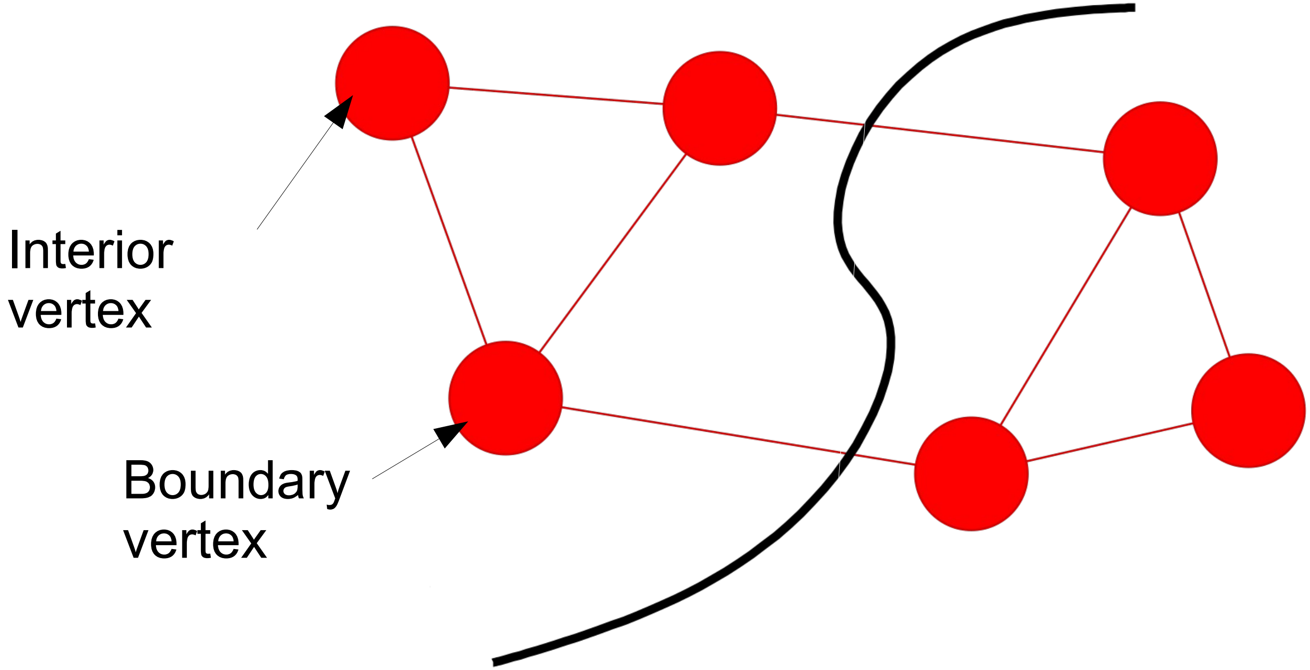
  color (u) != color (v)

- Goal is to use as few colors as possible

- Problem is NP-Complete → Faster algorithms approximate the optimal solution

Boman’s graph coloring algorithm

- Iterative distributed-memory graph coloring algorithm
- Vertices are split among p processors



- Tentative coloring phase :

- Each processors color its vertices, should be valid according to the previous global color allocation

- ! neighbors on different processors might be assigned the same color → invalid allocation

- communication regarding boundary vertices required (batched in supersteps)

- Conflict resolution phase :

- For each conflict, one of the two vertices gets marked as to be recolored in the next tentative coloring phase

- Eventually there will be no more conflicts

- No communication to be made in this phase

Tunable Parameters

- Superstep size

  Tradeoff between large superstep inducing less communication overhead but more conflicts and smaller superstep with more frequent communication

- Communication style

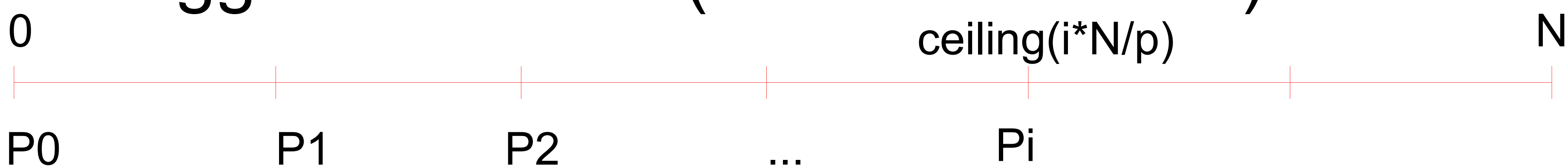
  Communication can either be synchronous or asynchronous (obvious latency advantage, but more conflicts with asynchronous)

Color assignment algorithms

- First Fit

  Each processors chooses the smallest admissible color in [1,N], where N
  Is the largest color used. If no color admissible choose N+1

- Staggered First Fit (N initial estimate)



  If no admissible look from 1 to floor(i*N/p)
  If still no admissible choose smallest permissible color > N

  Search first from ceiling(i*N/p) to ceiling((i+1)*N/p) and then
  From ceiling(i*N/p) to 0

- Modified Staggered First Fit

Matrix Storage Approach

Slightly modified CSR format with no values array

	0	1	2	3	4
0	1		1		1
1		1		1	
2		1	1		
3	1		1	1	
4		1		1	

	0	1	2	3	4	5			
rowptr	0	3	5	7	10	12			

	0	1	2	3	4	5	6	7	8	9	10	11
colind	0	2	4	1	3	1	2	0	2	3	1	3

Implementation Details

- C/C++ & MPI

- Focus on regular and predictable array accesses

- Compact data structures

- Addition of modified staggered first fit ("conflict-avoiding color fitting")

Previous work

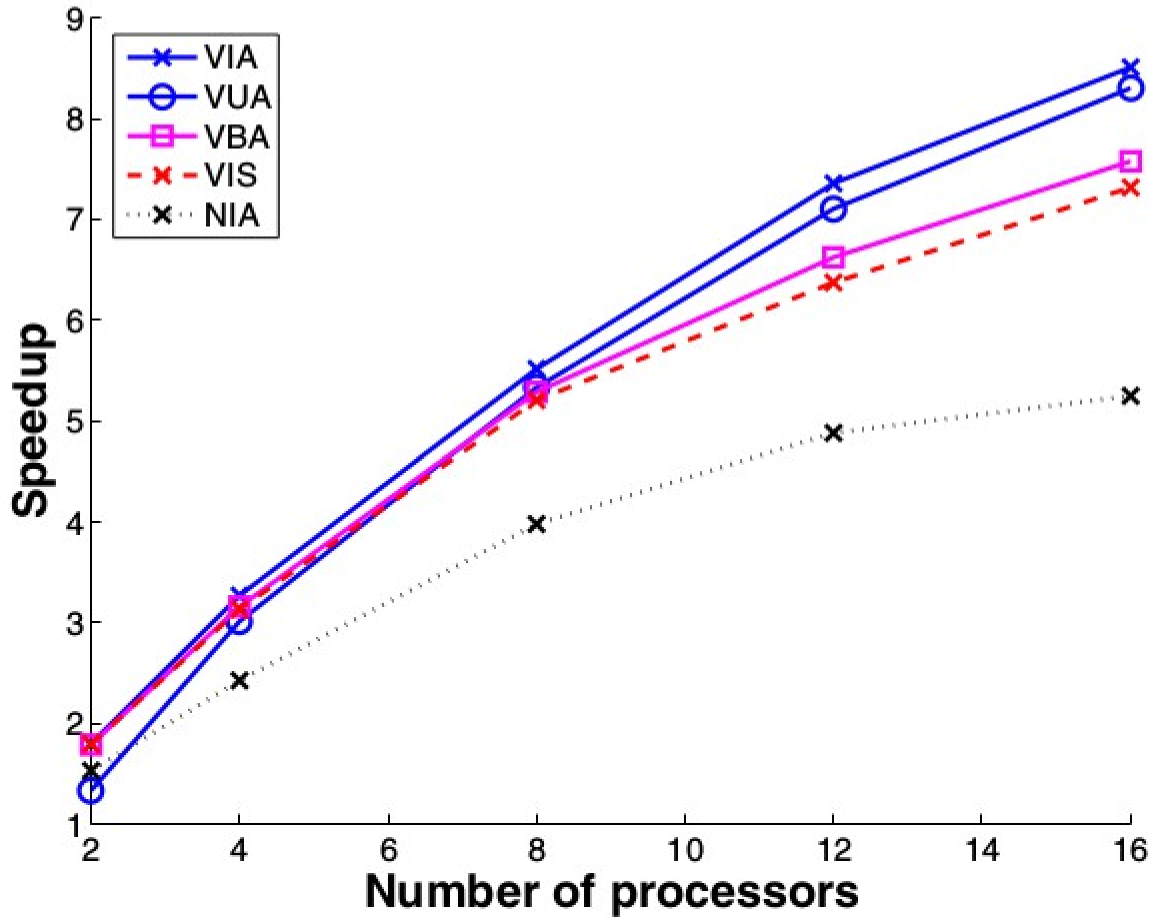
- The original paper (2005) proposed the following :

  - First-fit coloring assignment

  - 100 superstep size

  - assumes graphs are well-partitioned

Result

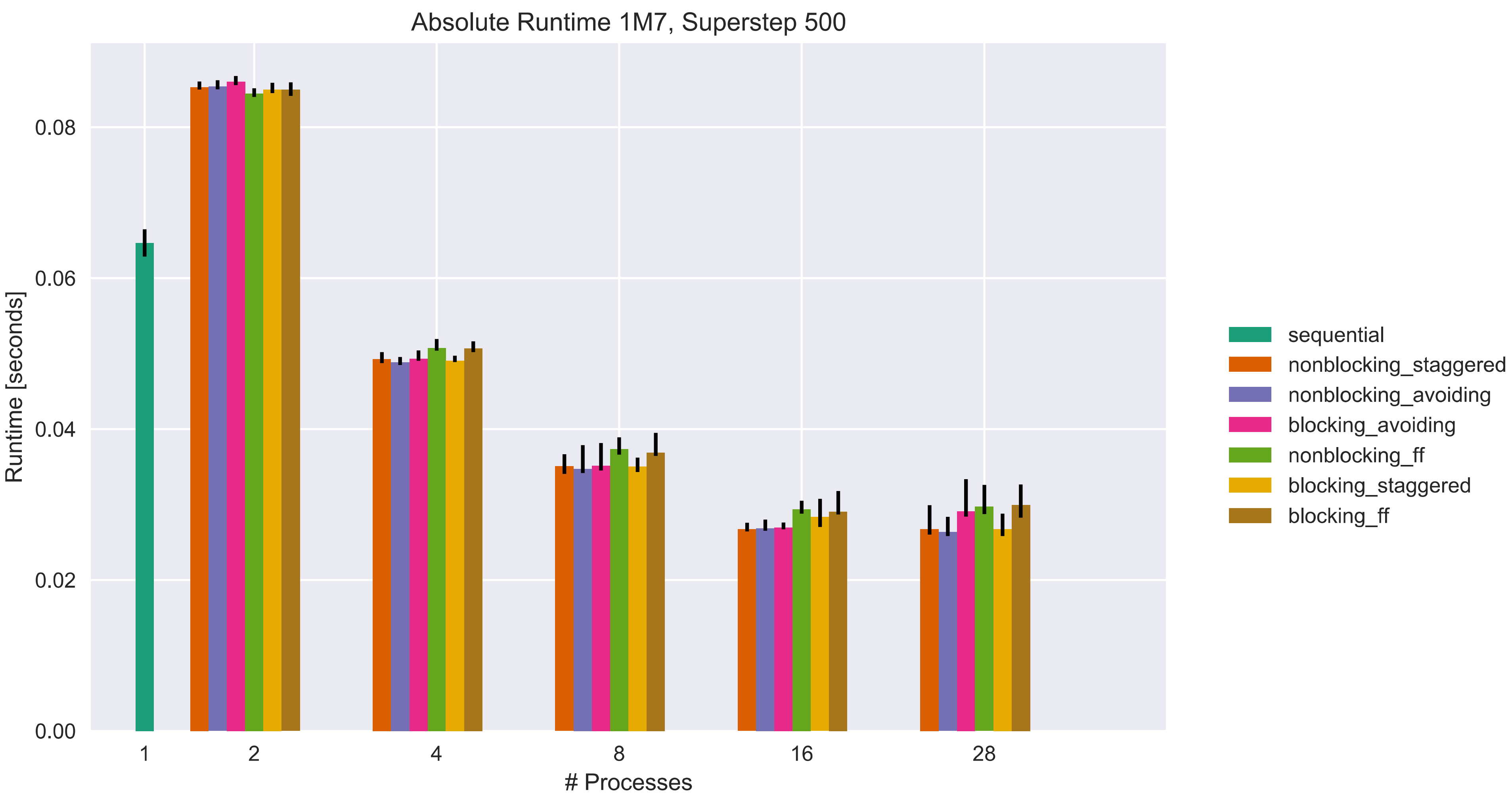


16 nodes PC-cluster with dual 900 MHz Intel Itanium 2 CPUs & 4 GB memory

Test Graphs

- RMAT random graphs
- 3 size levels: 1M/10M/30M vertices
- 3 density levels: 1:3/7/15 Vertices:Edges
- 10 graphs for each level-density combination

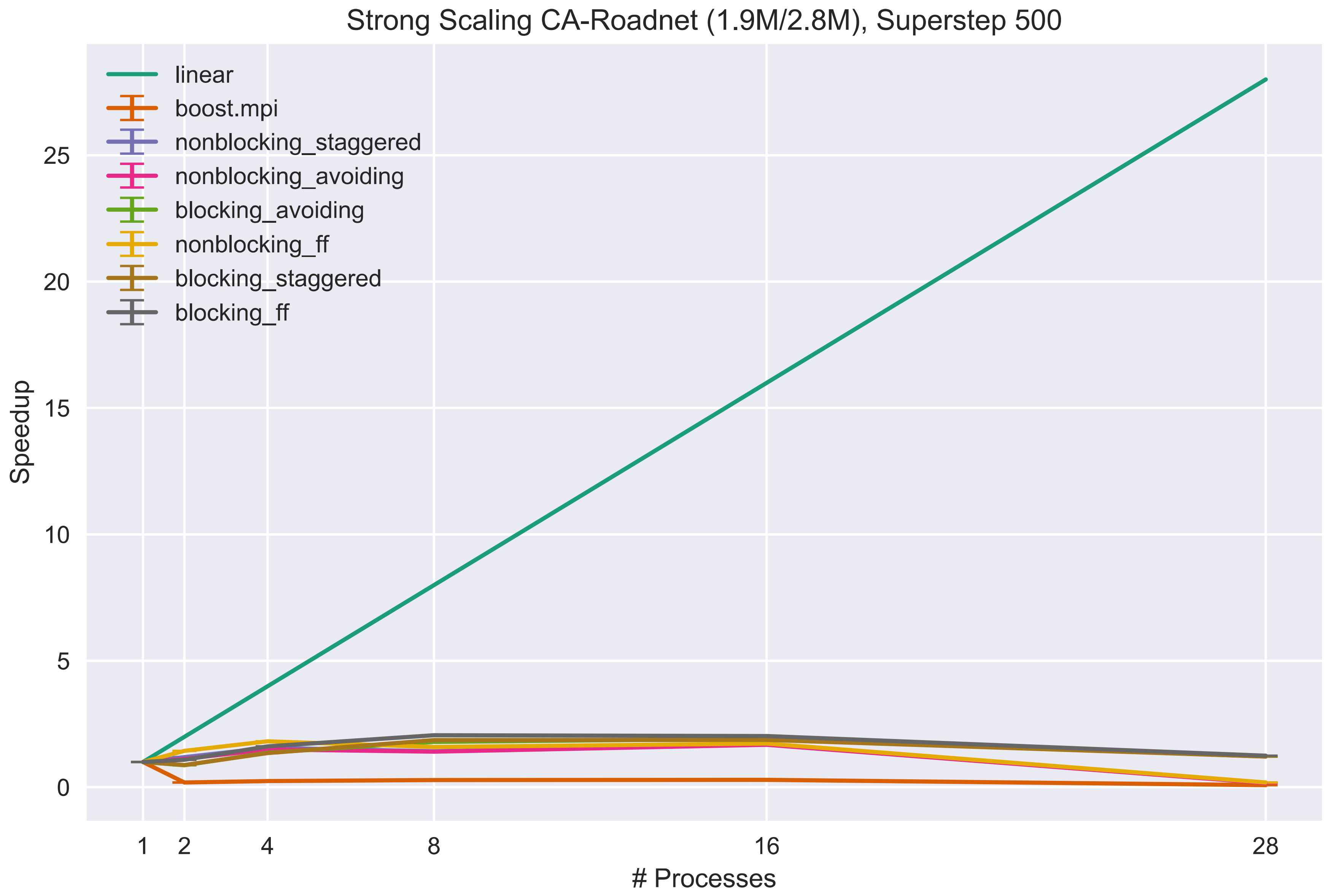
- Assumption: graph is distributed and knows about topology
- 90 graphs vs 7 configurations vs superstep sizes – unfortunately did not finish in time

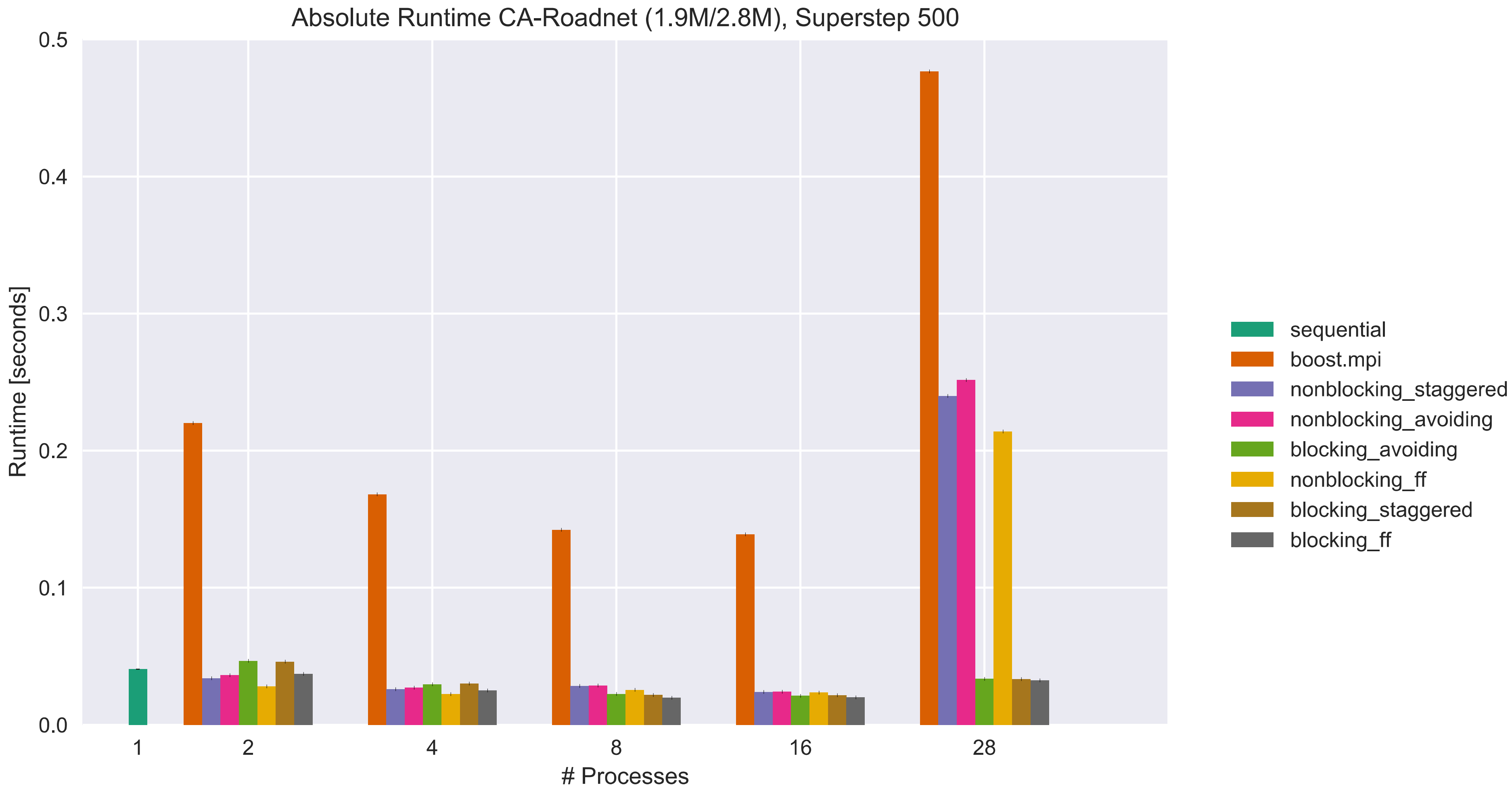

Absolute Runtime 1M7, Superstep 500
Runtime [seconds]
0.00
0.02
0.04
0.06
0.08
1
2
4
8
16
28
Processes
sequential
nonblocking_staggered
nonblocking_avoiding
blocking_avoiding
nonblocking_ff
blocking_staggered
blocking_ff

Test Graphs

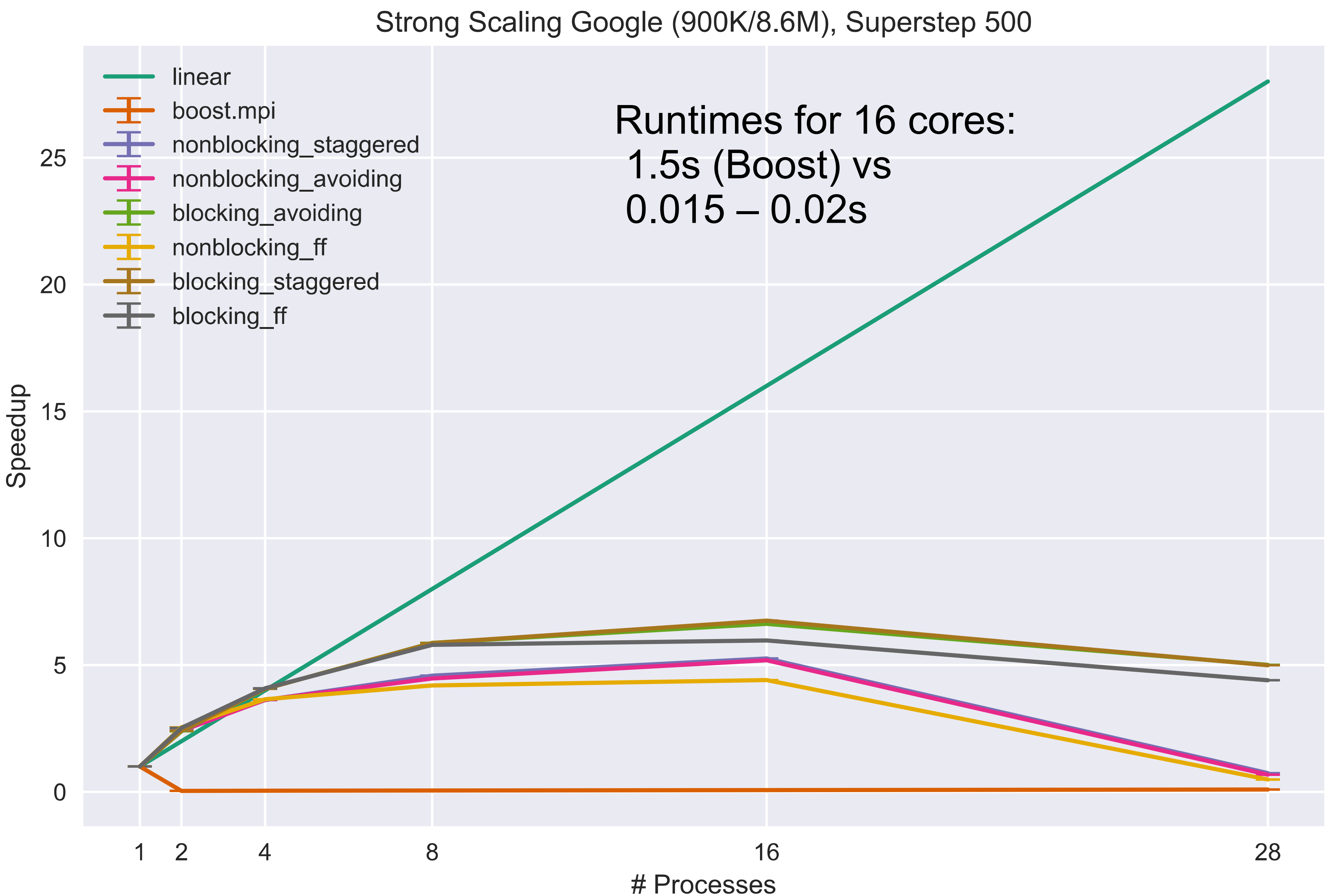
- Instead: real world graphs run on Euler

- Californian Road Network: 1.9M/2.8M
- Google Web Graph: 900K/8.6M
- Livejournal Social Network: 4M/69M
- Orkut Social Network: 3M/234M

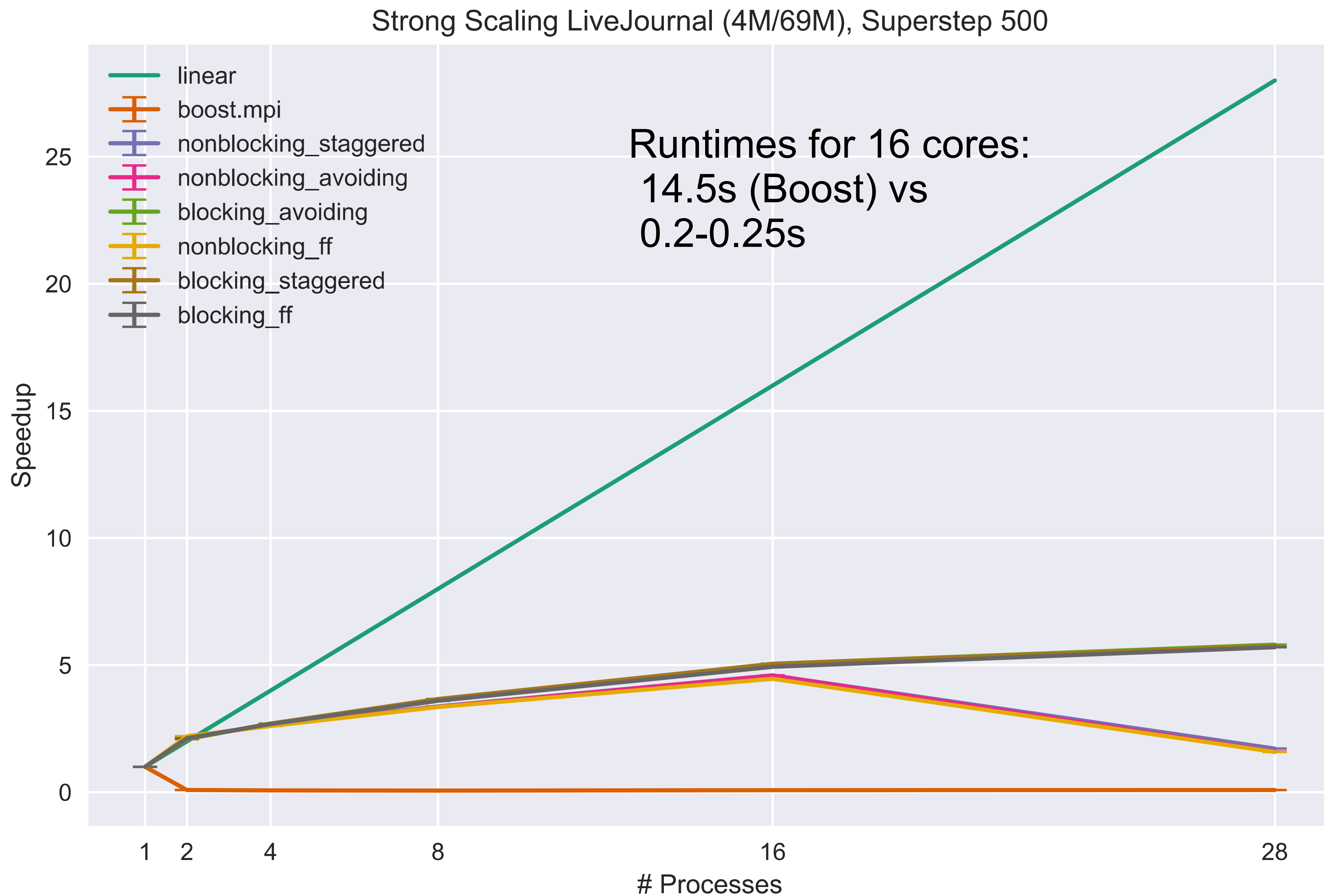

Strong Scaling CA-Roadnet (1.9M/2.8M), Superstep 500
linear
boost.mpi
nonblocking_staggered
nonblocking_avoiding
blocking_avoiding
nonblocking_ff
blocking_staggered
blocking_ff
Speedup
0
5
10
15
20
25
1
2
4
8
16
28
Processes



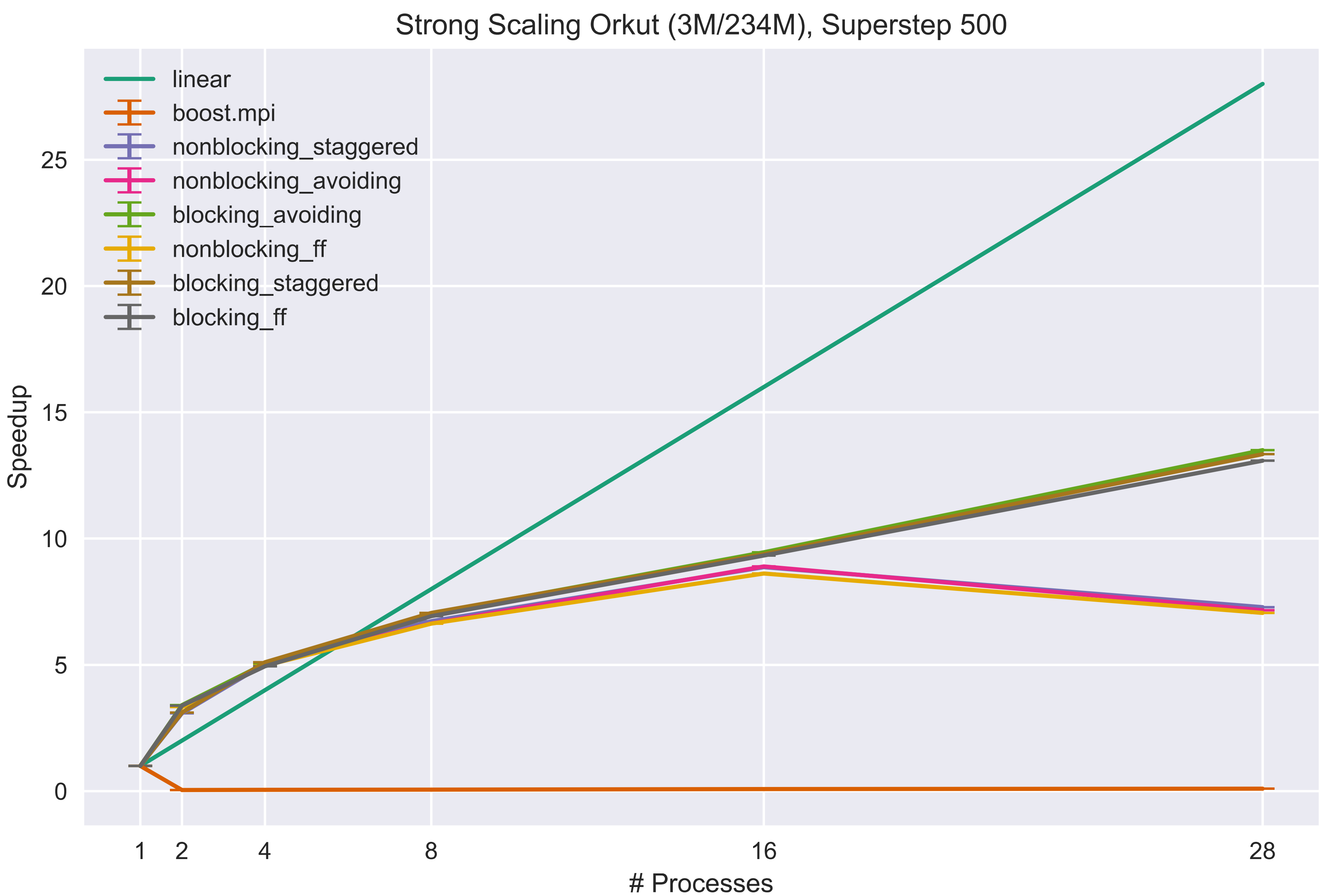
Absolute Runtime CA-Roadnet (1.9M/2.8M), Superstep 500
Runtime [seconds]
0.0
0.1
0.2
0.3
0.4
0.5
1
2
4
8
16
28
Processes
sequential
boost.mpi
nonblocking_staggered
nonblocking_avoiding
blocking_avoiding
nonblocking_ff
blocking_staggered
blocking_ff



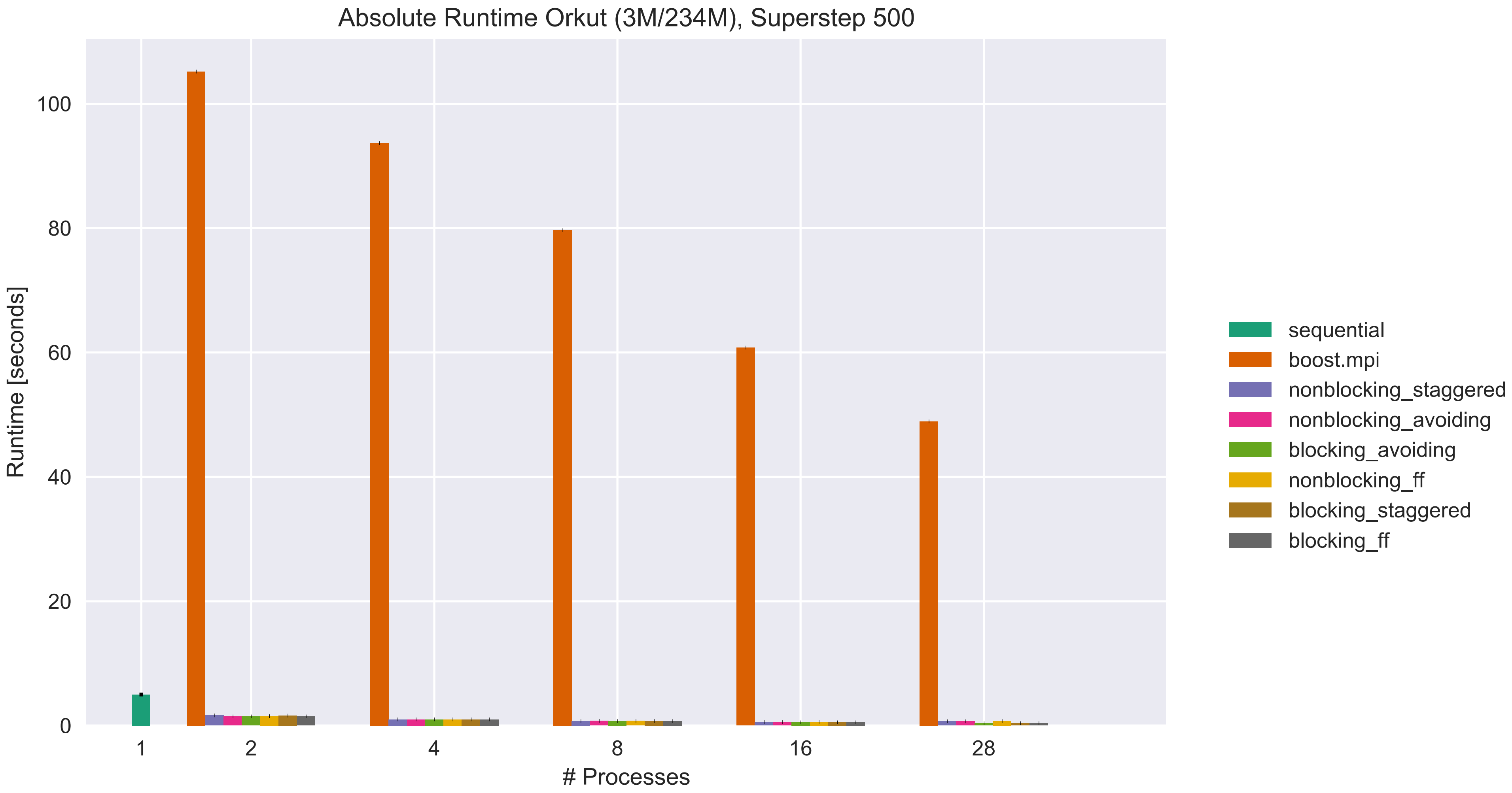
Strong Scaling Google (900K/8.6M), Superstep 500
linear
boost.mpi
nonblocking_staggered
nonblocking_avoiding
blocking_avoiding
nonblocking_ff
blocking_staggered
blocking_ff
Runtimes for 16 cores:
1.5s (Boost) vs
0.015 – 0.02s
Speedup
0
5
10
15
20
25
1
2
4
8
16
28
Processes

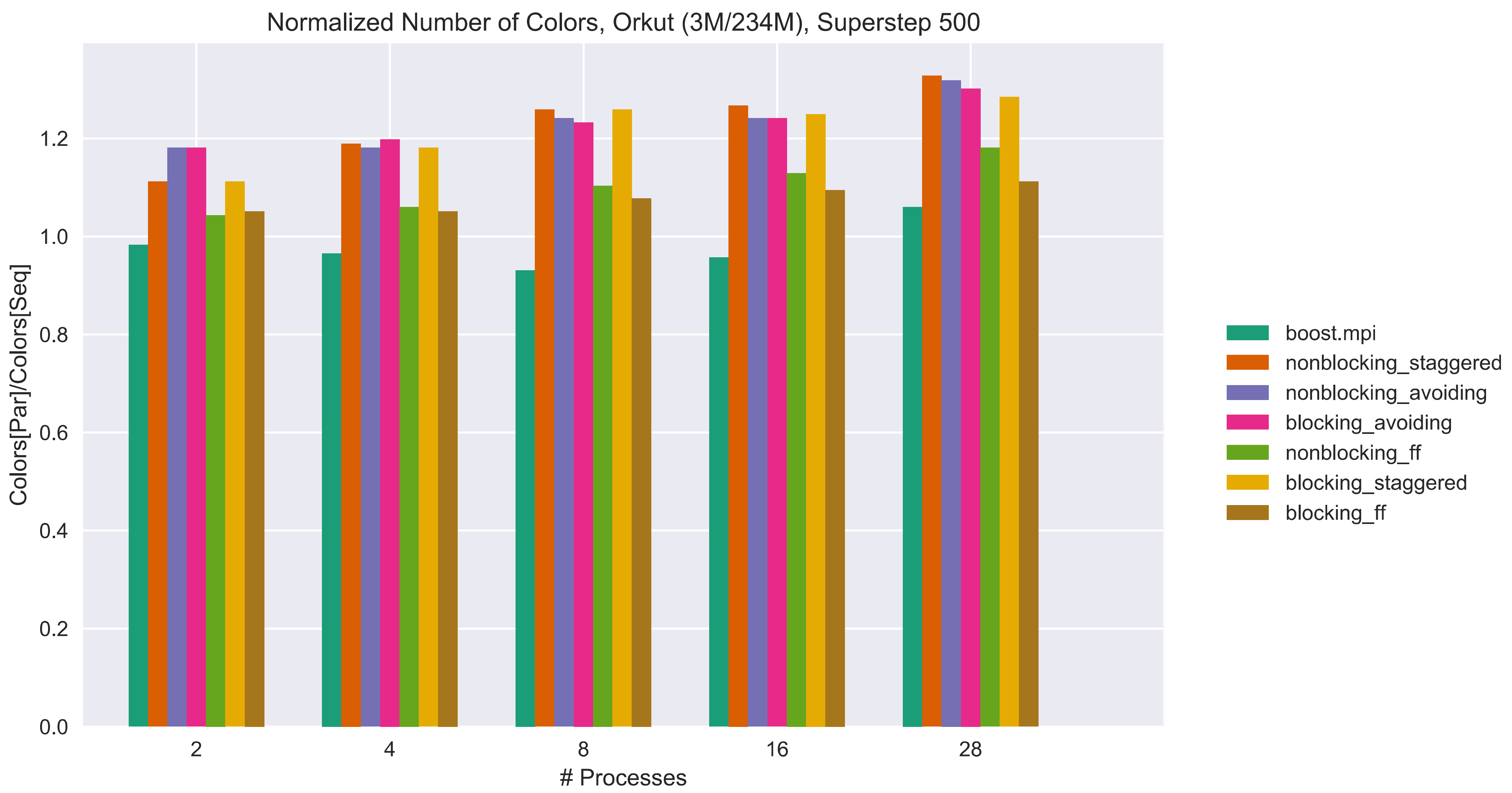


Strong Scaling LiveJournal (4M/69M), Superstep 500
linear
boost.mpi
nonblocking_staggered
nonblocking_avoiding
blocking_avoiding
nonblocking_ff
blocking_staggered
blocking_ff
Runtimes for 16 cores:
14.5s (Boost) vs
0.2-0.25s
Speedup
0
5
10
15
20
25
1
2
4
8
16
28
Processes



Strong Scaling Orkut (3M/234M), Superstep 500
linear
boost.mpi
nonblocking_staggered
nonblocking_avoiding
blocking_avoiding
nonblocking_ff
blocking_staggered
blocking_ff
Speedup
0
5
10
15
20
25
1
2
4
8
16
28
Processes


Absolute Runtime Orkut (3M/234M), Superstep 500
Runtime [seconds]
0
20
40
60
80
100
1
2
4
8
16
28
Processes
sequential
boost.mpi
nonblocking_staggered
nonblocking_avoiding
blocking_avoiding
nonblocking_ff
blocking_staggered
blocking_ff


Normalized Number of Colors, Orkut (3M/234M), Superstep 500
Colors[Par]/Colors[Seq]
0.0
0.2
0.4
0.6
0.8
1.0
1.2
2
4
8
16
28
Processes
boost.mpi
nonblocking_staggered
nonblocking_avoiding
blocking_avoiding
nonblocking_ff
blocking_staggered
blocking_ff

Conclusion

- Our implementation is really fast
- Better scaling on denser/larger graphs
- Blocking scales better than nonblocking
- Tradeoff complexity – speed – number of colors in color assignment