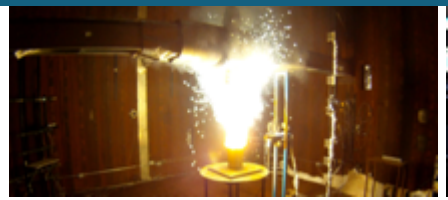
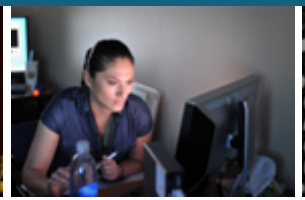
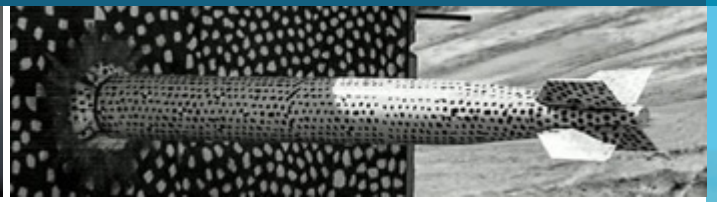
SAND2022-6268C

Sandia National Laboratories

# Parallel, Portable Algorithms for Distance-2 Maximal Independent Set and Graph Coarsening







*IPDPS22*

Brian Kelley and Sivasankaran Rajamanickam

EXASCALE COMPUTING PROJECT

kokkos

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

SAND2022....



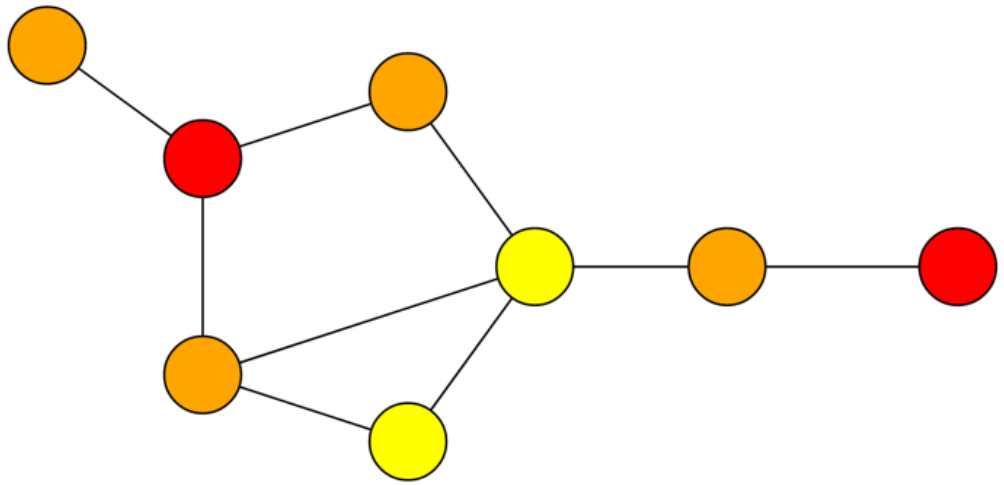
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

# Introduction

- Distance-2 Maximal Independent Set (MIS-2)
  - Enables coarsening suitable for algebraic multigrid (AMG) aggregation
- Efficient implementation in Kokkos, including 4 new algorithmic improvements
- Portable, performant and scalable across all major HPC architectures
  - Intel, ARM CPUs
  - NVIDIA, AMD GPUs
  - 5x speedup vs. ViennaCL, 6x vs. CUSP

# Distance-2 Maximal Independent Set (MIS-2)

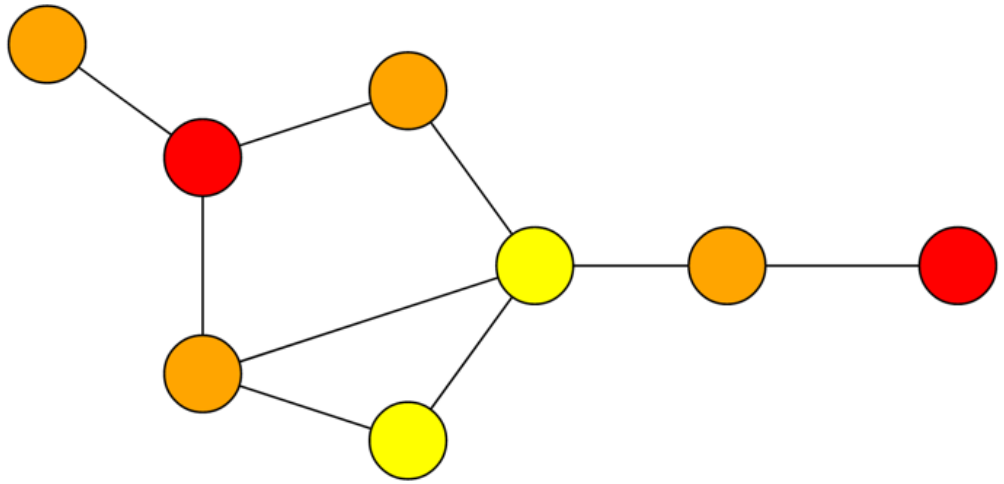
- Select a subset of vertices from undirected graph
- Satisfy two properties:
  1. Distance-2: no two vertices in set can be within 2 edges of each other
  2. Maximal: not possible to add more vertices to set without violating property 1.
     - Maximum independent set: largest maximal set, is NP-hard



Red: in MIS-2
Orange: distance-1 conflict
Yellow: distance-2 conflict

# Distance-2 Maximal Independent Set (MIS-2)

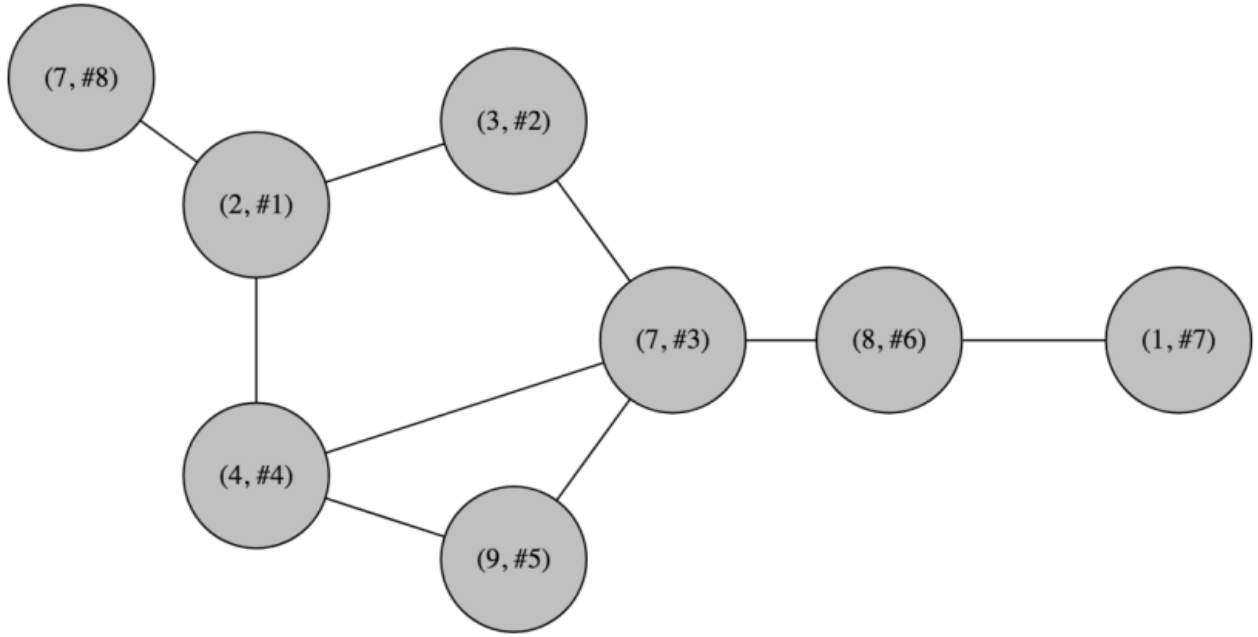
- Yields a straightforward graph coarsening suitable for aggregation-based Algebraic Multigrid (AMG) preconditioner [1]
  - Red vertices become cluster roots
  - Orange vertices join with the adjacent root
  - Yellow vertices join with an adjacent orange arbitrarily
- For all graphs, this assigns every vertex to a cluster

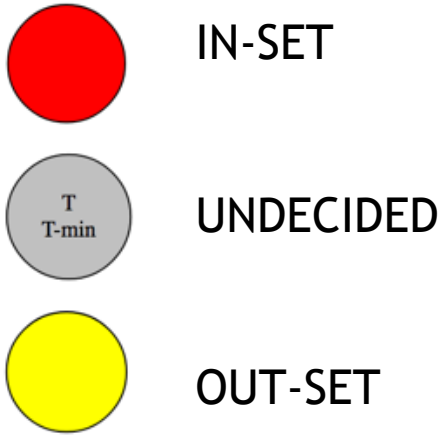


Red: in MIS-2
Orange: distance-1 conflict
Yellow: distance-2 conflict

# MIS-2 Parallel Greedy Algorithm (Bell et al.) [1]

- Computes distance-k MIS (we fix k=2)
- All vertices initially have status UNDECIDED (grey).
- Assign each vertex a tuple T: (status, random priority, #ID)
- Also set $T_{min}$ = T: the minimum tuple in a radius-k neighborhood (initially k = 0)

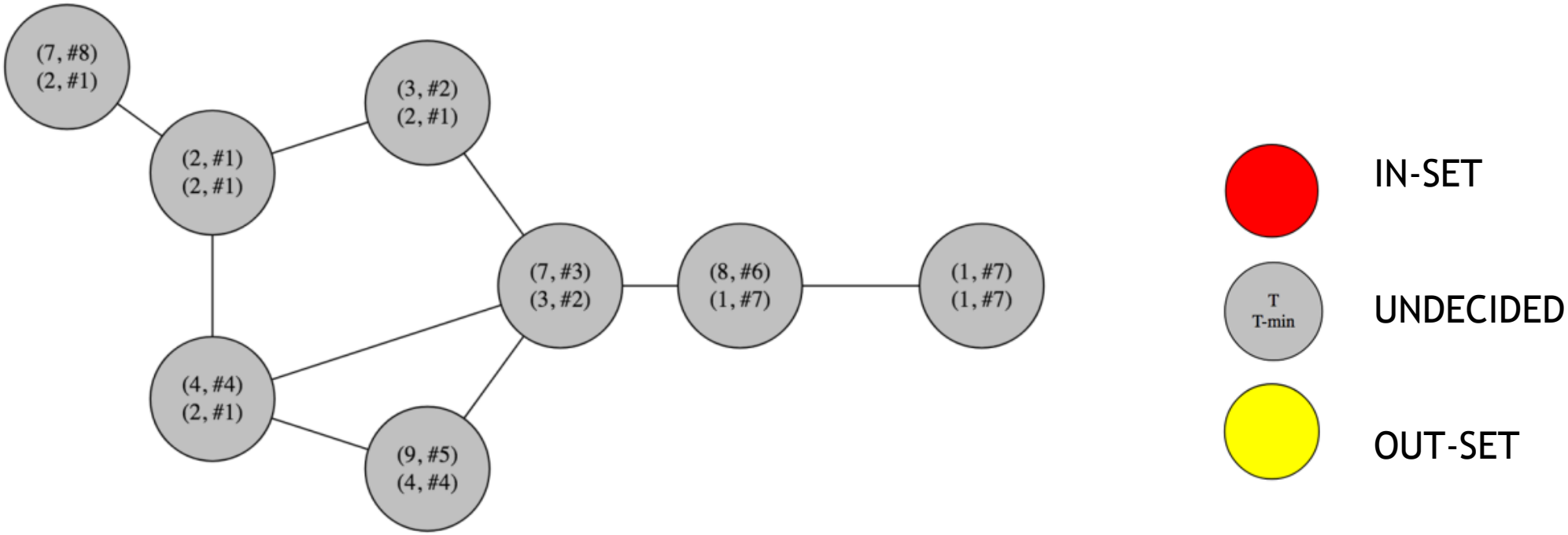



[1] Bell, S. Dalton, and L. Olson, “Exposing fine-grained parallelism in algebraic multigrid methods”, SISC 2012.

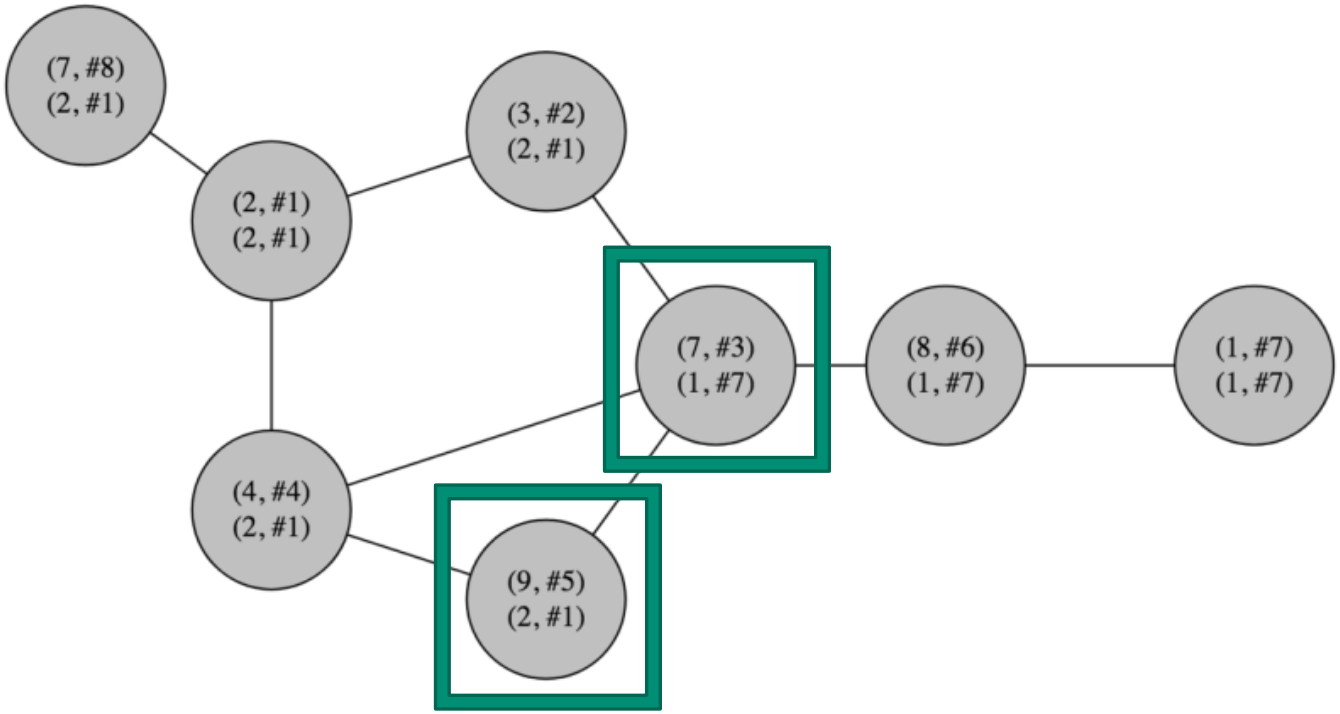
# MIS-2 Parallel Greedy Algorithm (Bell et al.)

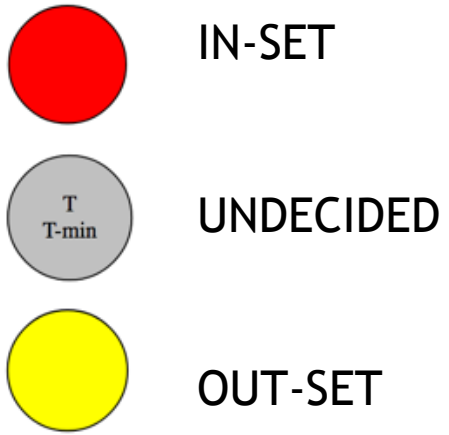
- For i = 1…k, $T_{min}$ := min($T_{min}$ among neighbors)
- Compared lexicographically: status, priority, ID
- IN-SET < UNDECIDED < OUT-SET
- i = 1:



# MIS-2 Parallel Greedy Algorithm (Bell et al.)

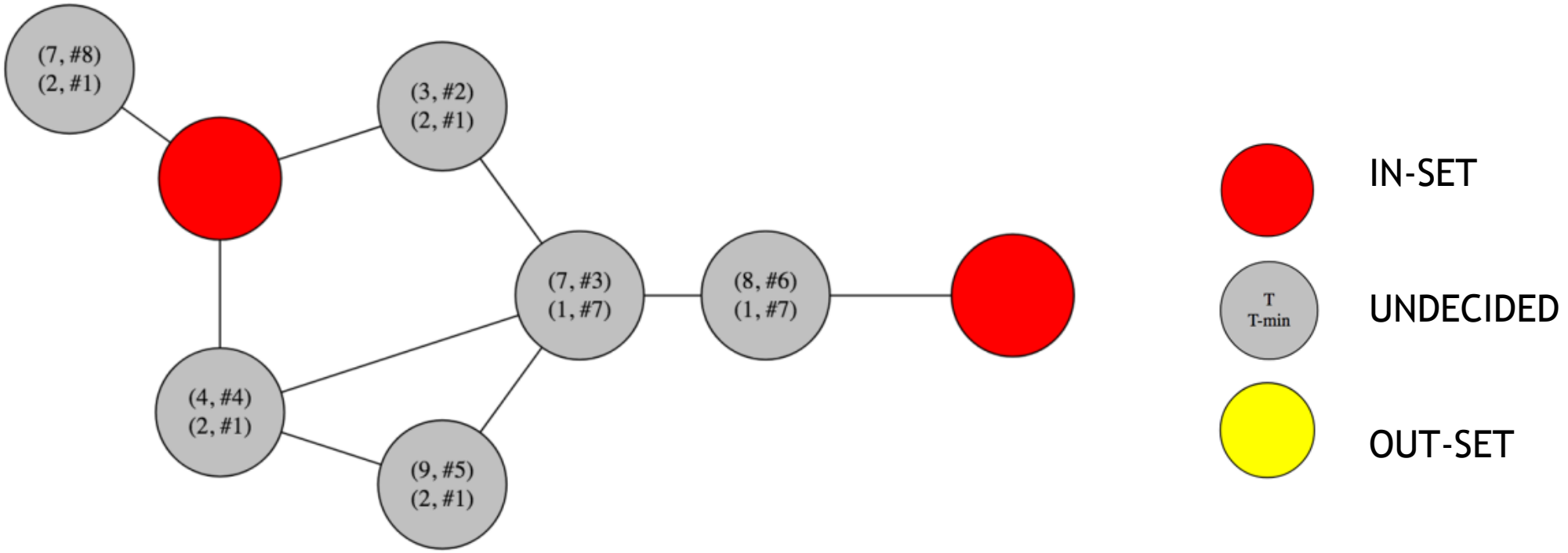
- For i = 1…k, $T_{min}$ := min($T_{min}$ among neighbors)
- Compared lexicographically: status, priority, ID
- IN-SET < UNDECIDED < OUT-SET
- i = 2:



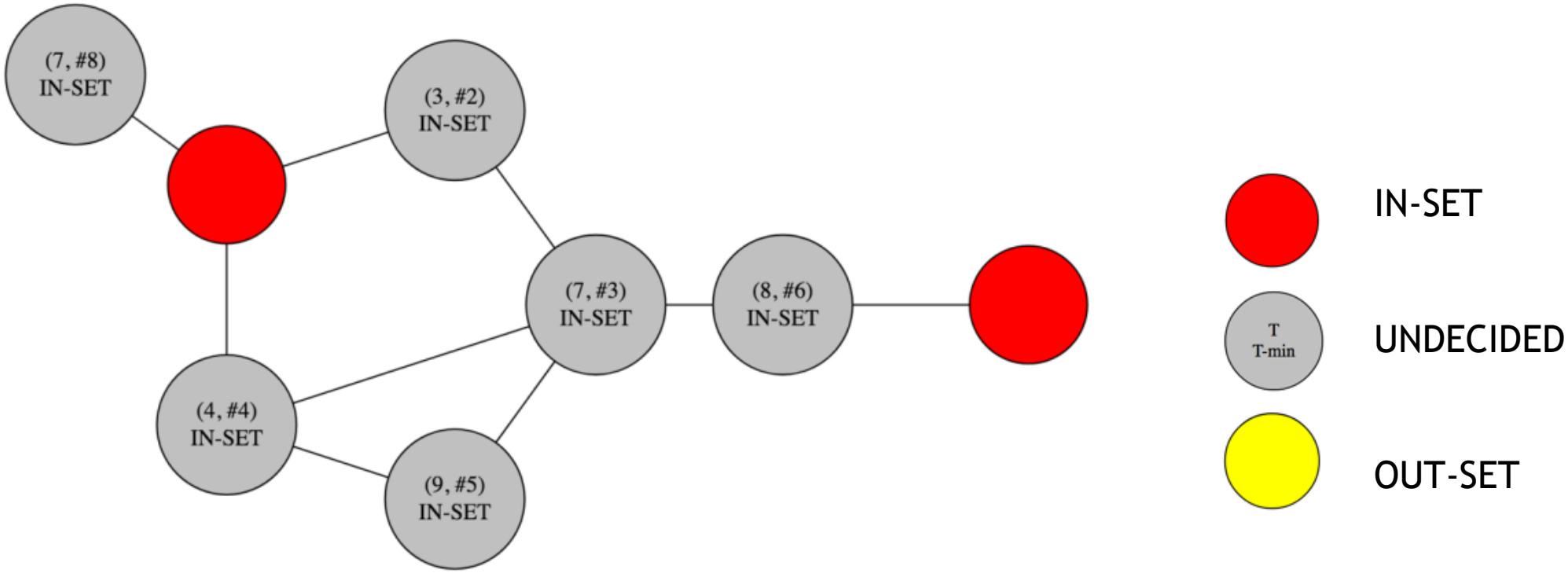
# MIS-2 Parallel Greedy Algorithm (Bell et al.)

- Vertices with T = $T_{min}$ must be in the set:
- Tuples are unique (ID guarantees this)
- In any radius-k neighborhood, only one vertex can have the minimum
- Otherwise, if $T_{min}$ = IN-SET, then definitely excluded from set



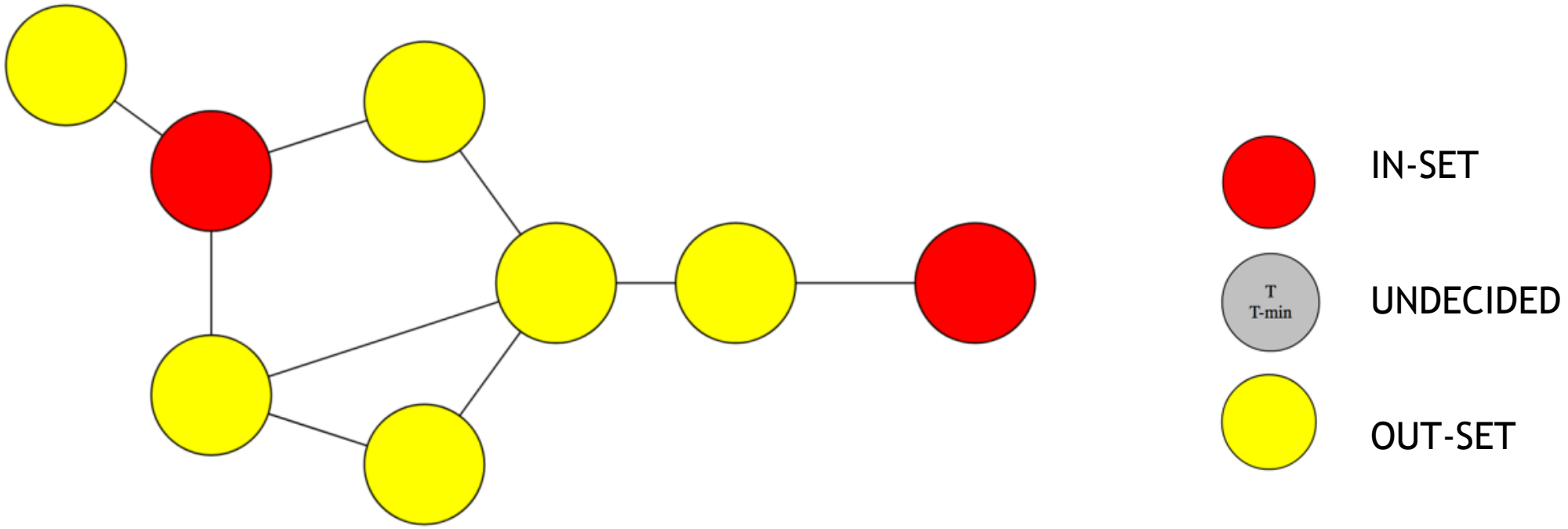
# MIS-2 Parallel Greedy Algorithm (Bell et al.)

- Next iteration's $T_{min}$ (i = 1…2 loop not shown):



# MIS-2 Parallel Greedy Algorithm (Bell et al.)

- Vertices with T = $T_{min}$ must be in the set:
  - Tuples are unique
  - In any radius-k neighborhood, only one vertex can have the minimum
- Otherwise, if $T_{min}$ = IN-SET, then definitely excluded from set
- Done in 2 iterations – no UNDECIDED vertices remain



# MIS-2: Implementation in Kokkos

# MIS-2: Efficient Kokkos Implementation

- Kokkos [2]: “[parallel] programming model for high performance on all major HPC platforms”
- Build application for any major CPU and GPU architecture (Nvidia, AMD, Intel, Power, ARM, …)
- Implement MIS-2 for graph portion of Kokkos Kernels [2a]
  - Parallel, portable linear algebra and graph library

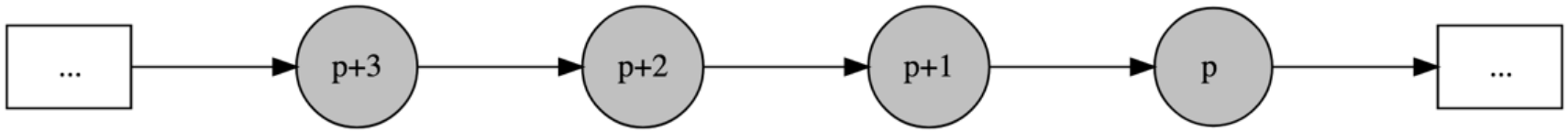
- Implementation Issues:
  - How and when to choose random priorities?
  - How to represent the (state, priority, ID) tuples?
  - How to improve memory access patterns for GPUs?

kokkos

[2]: https://github.com/kokkos/kokkos
[2a]: https://github.com/kokkos/kokkos-kernels

# MIS-2: (Pseudo)random Priorities

- Want to keep determinism for reproducibility, debuggability
  - AMG aggregates affect entire solver stack
- Use hash function of vertex ID and iteration
  - 64-bit xorshift [3] followed by linear congruential step (aka xorshift*)
  - priority = hash(hash(v_id) xor hash(iteration))




- Unlucky chain of descending priorities: only one vertex can be added to set per iteration
  - New priorities each iteration -> break dependency chains
  - Iteration count also higher with plain xorshift

[3] Marsaglia, G. “Xorshift RNGs”, JSS 2003

# MIS-2: Packed Tuples

- Represent (state, priority, ID) tuple with a single integer
- IN-SET = 0
- OUT-SET = ~0 (0xFFFF…)
  - [4] applied similar strategy to MIS-1
- Let $b = \lceil \log_2(|V| + 1) \rceil$
  - then any UNDECIDED tuple is: `(priority << b) | (v_id + 1)`
  - Priority bits more significant than ID
  - ID bits are unique, so no ties are possible

[4] M. Burtscher, S. Devale, S. Azimi, J. Jaiganesh, and E. Powers. "A High-Quality and Fast Maximal Independent Set Implementation for GPUs." ACM TOPC 2018

# MIS-2: Prefix-sum Worklist

- Avoid unnecessary work in inner loop (i = 1…2)
  - In later iterations, most vertices have already been labeled
- For each i, vertices within a radius-i neighborhood of an IN-SET vertex don't need to be processed
- Maintain two separate worklists: one for i = 1, and one for i = 2
  - This means the implementation only computes MIS-2, not MIS-k
- Use a parallel prefix sum ("scan") to filter out the newly labeled vertices
  - Kokkos provides high performance scan
  - Scan-based worklists were used for Kokkos Kernels greedy coloring in [5]

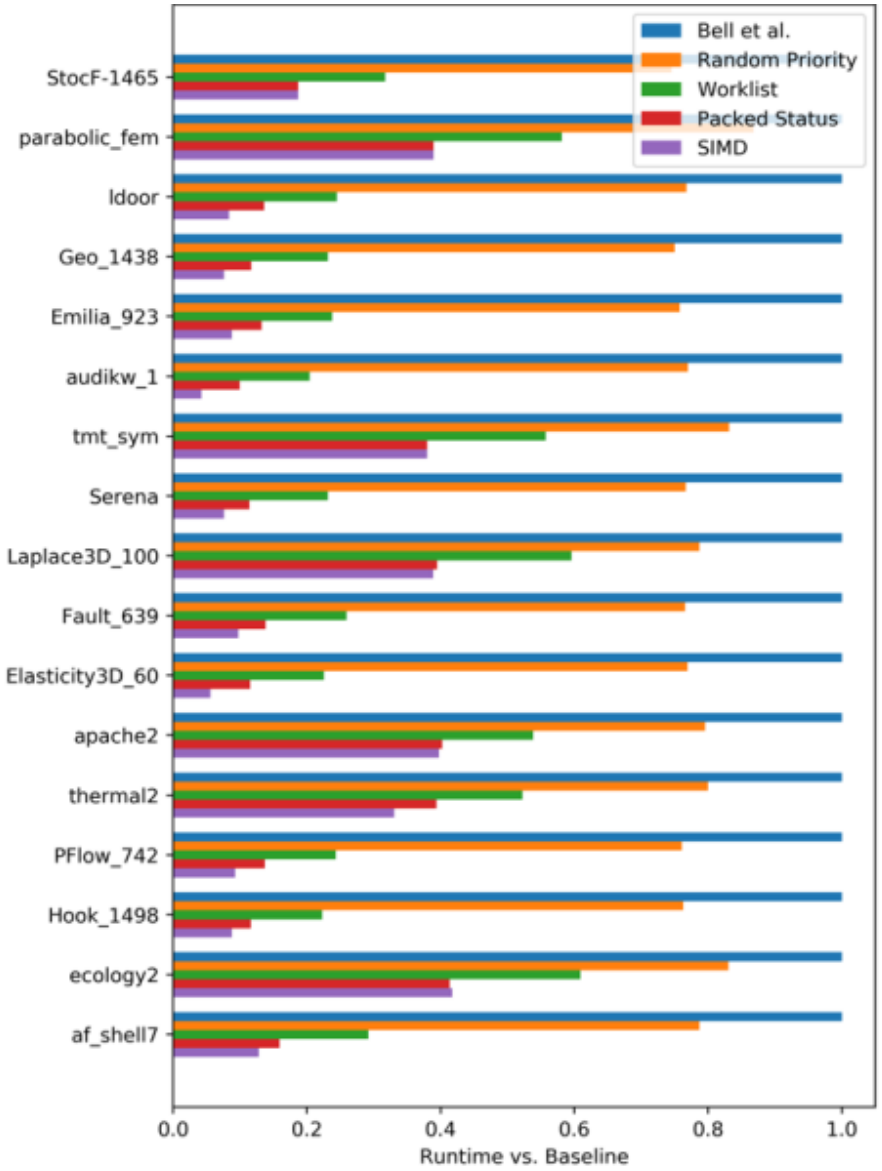
[5] M. Deveci, E. G. Boman, K. D. Devine and S. Rajamanickam, "Parallel Graph Coloring for Manycore Architectures," IPDPS 2016.

# MIS-2: SIMD Neighbor Processing

- Uses Kokkos 2-level parallelism
- Only for GPUs; not practical to vectorize on CPUs
- Express each loop over neighbors as a parallel loop
  - A warp (NVIDIA) or wavefront (AMD) can process 32 or 64 (resp.) loop iterations simultaneously
  - Coalesces memory accesses of CRS adjacency list
- Generally not profitable if avg. degree < 16

# MIS-2 in Kokkos: Performance and Applications

# MIS-2: Cumulative Effects of Optimizations



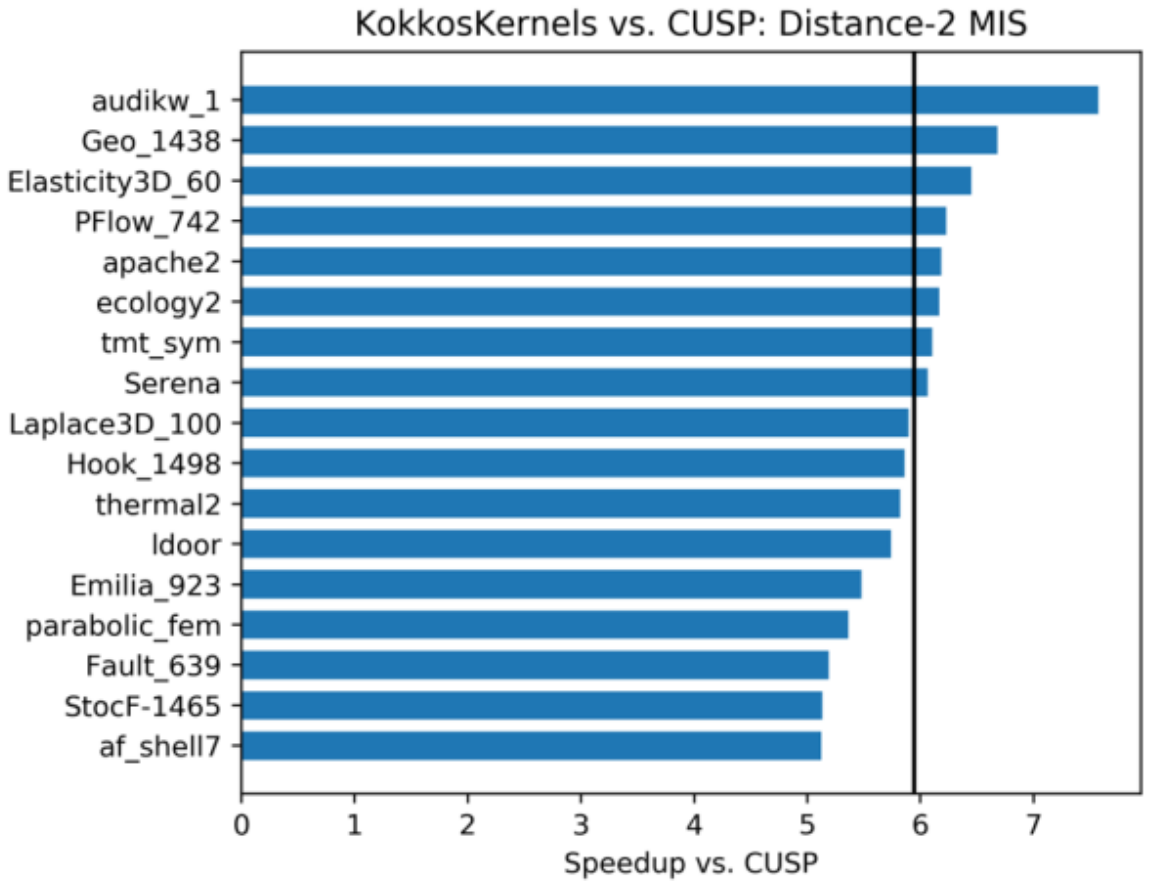

- 15 graphs from SuiteSparse, and 2 generated by Trilinos (Laplace3D, Elasticity3D)
- Bell et al. is a reference implementation of Bell 2012 in Kokkos.
- Each implementation includes all previous optimizations too
- “Avg. degree < 16” heuristic disables SIMD for some graphs

Mean speedup from each optimization:

- RandomPriority: 1.28x
- Worklist: 2.55x
- PackedStatus: 1.72x
- SIMD: 1.37x

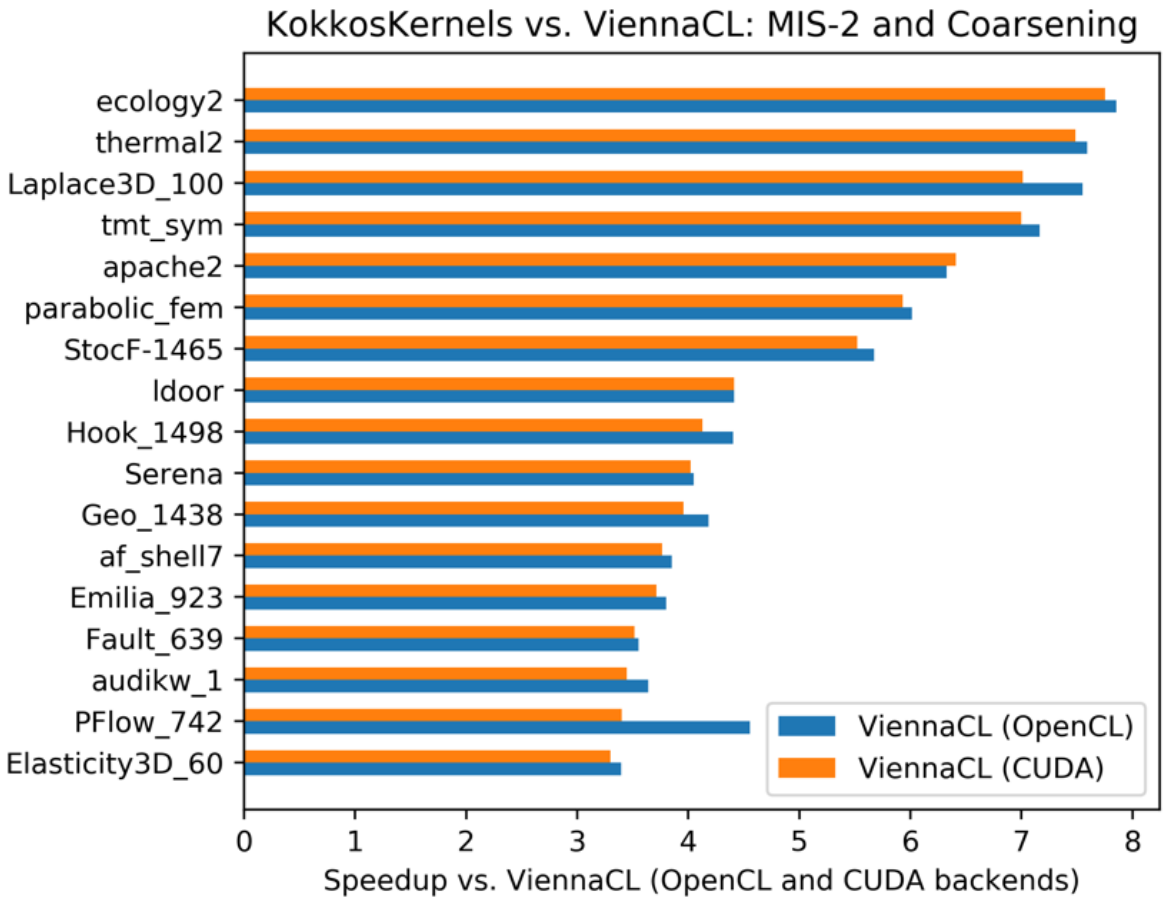
**Total: 8.97x**

# MIS-2: Performance vs. CUSP (NVIDIA V100)



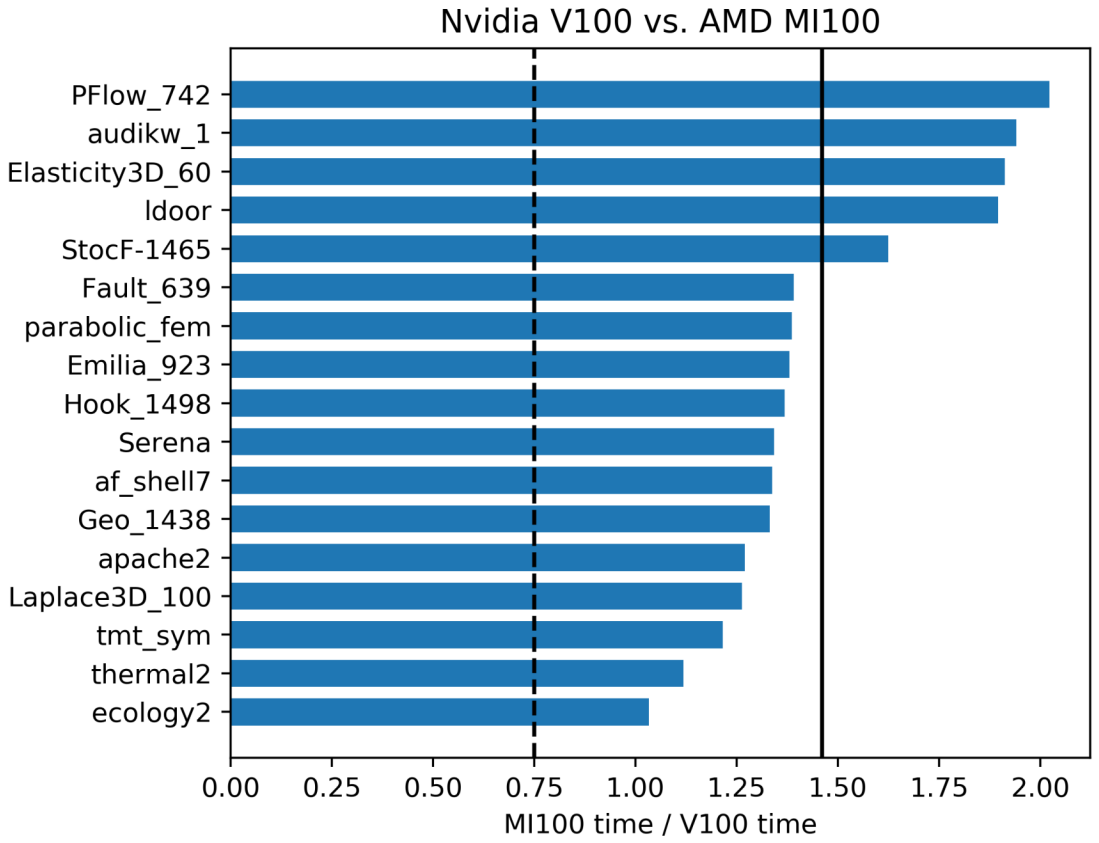

- Mean 6x speedup over CUSP library
- CUSP Library [6] (Dalton/Bell are primary contributors)

# MIS-2: Performance vs. ViennaCL (NVIDIA V100)



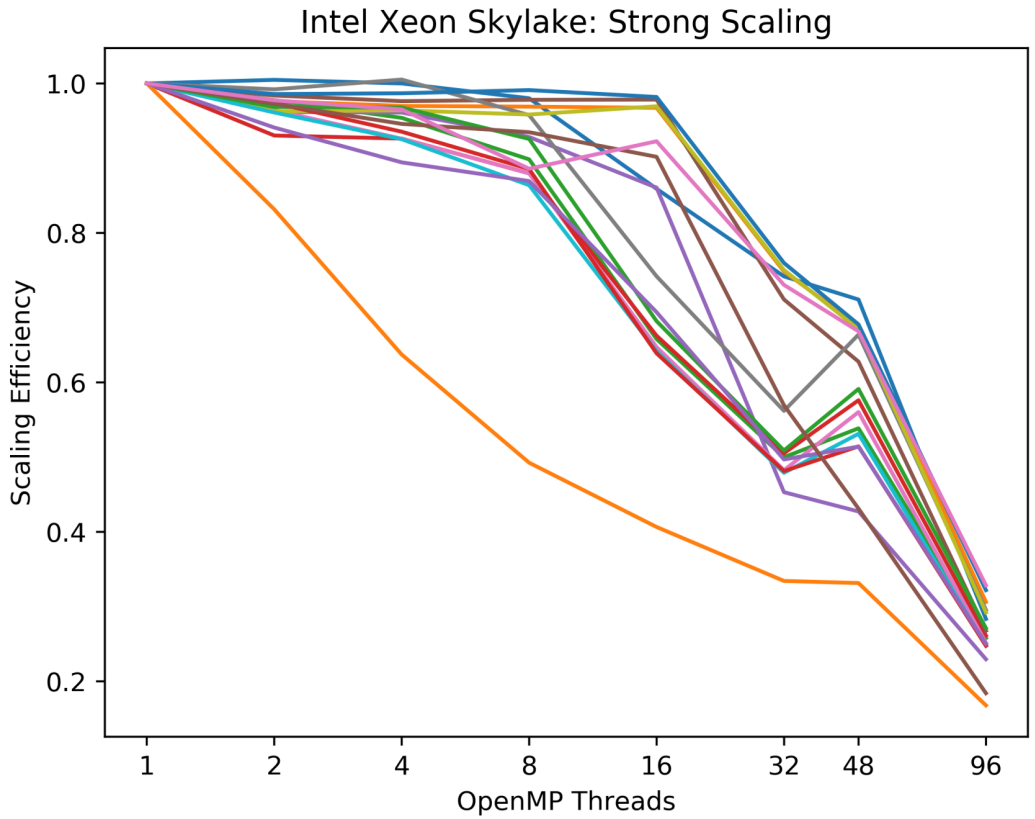

- Measures **MIS-2 plus coarsening**
- Mean 5x speedup over ViennaCL (both OpenCL and CUDA backends) [7]

# MIS-2: GPU Portability to AMD



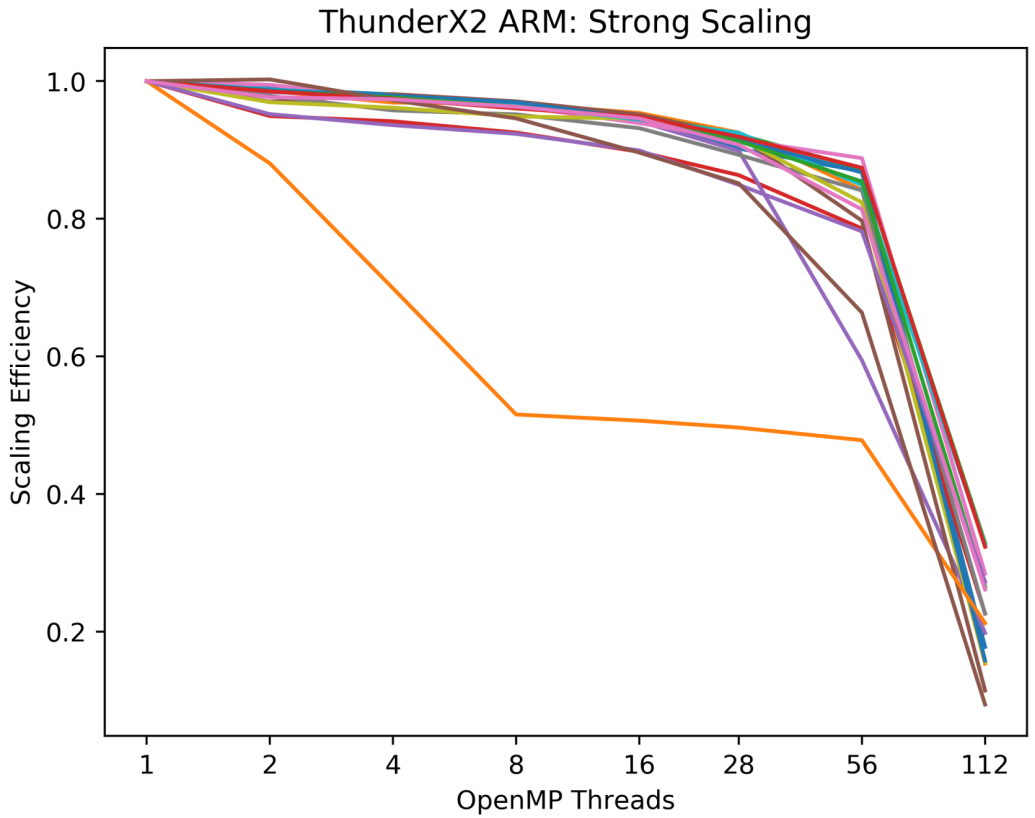

- No architecture-specific code needed
- Dashed line is theoretical speedup given memory bandwidths
- Solid line is actual mean speedup (V100 is more efficient with bandwidth)

# MIS-2: Intel CPU Strong Scaling




- Xeon Platinum 8160 (Skylake)
  - 2 sockets x 24 cores x 2 hyperthreads
- Each line is one of 17 graphs from before
- Scales to all 48 cores, but not hyperthreads

# MIS-2: ARM CPU Strong Scaling




- 2 sockets x 28 cores x 2 threads
- Very good scaling to all 56 cores but not threads

# Cluster Multicolor Gauss-Seidel

- Gauss-Seidel uses sequential forward/backward substitution
- Approximate version can be parallelized by coloring vertices (red-black, multicolor)
  - Same color -> not adjacent, can update in parallel
  - As a preconditioner, this gives slower convergence than sequential
- Cluster strategy: coarsen graph (using MIS-2) before coloring
  - Get small neighborhoods of vertices, and the neighborhoods of same color are independent
  - Go over neighborhoods of a color in parallel, but do the updates sequentially within the neighborhoods.
- Setup faster: coarsening is much cheaper than coloring, and coarse graph is smaller
- Solve faster: faster convergence, better locality (vertices in neighborhood reused)

# Cluster Multicolor Gauss-Seidel

- Comparing regular multicolor vs cluster multicolor preconditioners
  - (on the 5 problems out of 17 that converge to $|Ax-b|<10^{-8}$ in fewer than 800 iterations)
  - GMRES + preconditioner
- Setup time, total solve time and iterations all improved by cluster

Table VI
POINT VS. CLUSTER MULTICOLOR GAUSS-SEIDEL AS PRECONDITIONERS FOR GMRES. WE COMPARE SETUP AND TOTAL APPLY TIME, AS WELL AS GMRES ITERATIONS.

| | P. Setup | C. Setup | P. Apply (P. Iters) | C. Apply (C. Iters) |
|---|---|---|---|---|
| bodyy5 | 0.0154 | **0.00849** | 0.124 (187.0) | **0.0616** (**172.6**) |
| Elasticity3D_60 | 0.174 | **0.0438** | 7.41 (**328.2**) | **4.56** (337.4) |
| Geo_1438 | 0.209 | **0.0662** | 11.1 (408.5) | **4.73** (**388.4**) |
| Laplace3D_100 | 0.0553 | **0.0409** | 0.664 (158.4) | **0.567** (**144.6**) |
| Serena | 0.215 | **0.0664** | 6.55 (227.0) | **2.93** (**219.2**) |

# Conclusions

- MIS-2 based coarsening
  - Suitable for algebraic multigrid aggregation
  - New application in Gauss-Seidel parallel preconditioning
- Efficient implementation in Kokkos
  - Regenerate random priorities every iteration
  - Represent 3-tuples in single integer
  - Use prefix-sum based worklists to avoid redundant work
  - Use SIMD parallelism to improve memory access pattern
- Portable, performant and scalable across all major HPC architectures
  - Intel, ARM CPUs
  - NVIDIA, AMD GPUs
  - 5x speedup vs. ViennaCL, 6x vs. CUSP

# References


1. Bell, S. Dalton, and L. Olson, “Exposing fine-grained parallelism in algebraic multigrid methods”, SISC 2012
2. Kokkos parallel programming model: https://github.com/kokkos/kokkos
3. Marsaglia, G. “Xorshift RNGs”, JSS 2003
4. M. Burtscher, S. Devale, S. Azimi, J. Jaiganesh, and E. Powers. "A High-Quality and Fast Maximal Independent Set Implementation for GPUs.” ACM TOPC 2018
5. M. Deveci, E. G. Boman, K. D. Devine and S. Rajamanickam, "Parallel Graph Coloring for Manycore Architectures,” IPDPS 2016.
6. CUSP Library: https://github.com/cusplibrary/cusplibrary
7. ViennaCL Library: http://viennacl.sourceforge.net


- Graph images generated using GraphViz and Edotor