Hierarchical Partitioning for GPU-Based HPC Platforms

Christopher Brissette[1] George M. Slota[1] Andy Huang[2]

[1]Rensselaer Polytechnic Institute
[2]Sandia National Laboratories

SIAM Parallel Processing
Partitioning and Process Mapping for Emerging Architectures

25 February 2022

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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.

Scalable Partitioning on GPUs

Motivation for our ongoing work

Overall Goal: Scalable partitioning for Multi-GPU platforms

Why? Representative of modern HPC architectures

- We consider hierarchical CPU+multi GPU systems.
- We consider large-scale scientific computing applications.

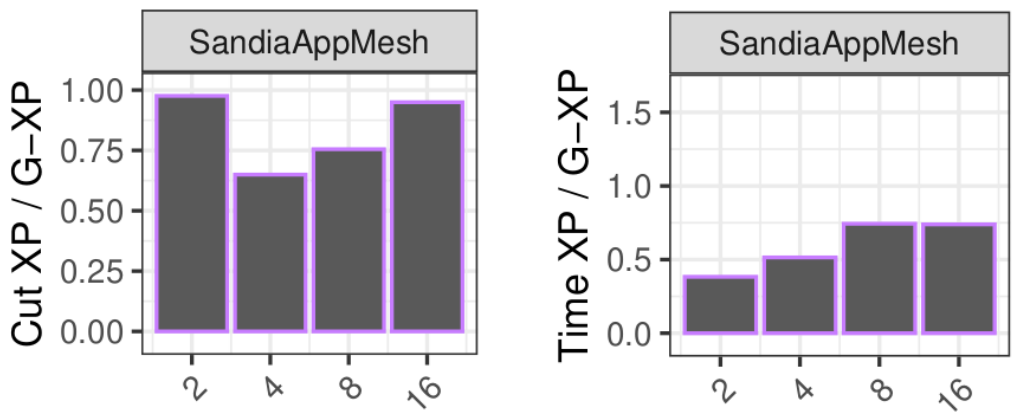
General challenges:

- Memory-constraints with efficient GPU memory usage.
- Extremely high parallel concurrency within node and across a distributed system.
- Hierarchical considerations for task placement, optimization criteria, etc. for both partitioning and target applications.

Prior Work: Porting XtraPuLP for GPUs

Challenges: Modifying CPU-only XtraPuLP for GPUs

- For distributed CPU processing, XtraPuLP demonstrated scaling to trillion-edge networks on thousands of MPI ranks. Porting to GPU at this kind of scale presents several common issues.
- Issue: Higher concurrency of GPU threading negatively impacts balancing phase. Part size and quality can wildly oscillate.
- Issue: Mitigating balance issue impacts scalability and speed.
- **However:** Refinement phase can still run quite efficiently.



Relative cut quality (left) and time (right) of GPU-XtraPuLP (G-XP) vs. XtraPuLP (XP) from 2–16 nodes/parts on a Sandia Labs application mesh. To achieve similar quality values on GPUs, we must restrict our allowable vertex→part movements over a larger number of iterations, reducing time to solution. In the plots, bigger is better.

Our Ongoing Work

And the topic of today's talk (below in bold)

Observations:

- Modern spectral partitioners can give fast and high quality cuts on multiple GPUs (Acer et al., 2021).
- Efficient graph coarsening can also be performed on GPU (Gilbert et al., 2021).

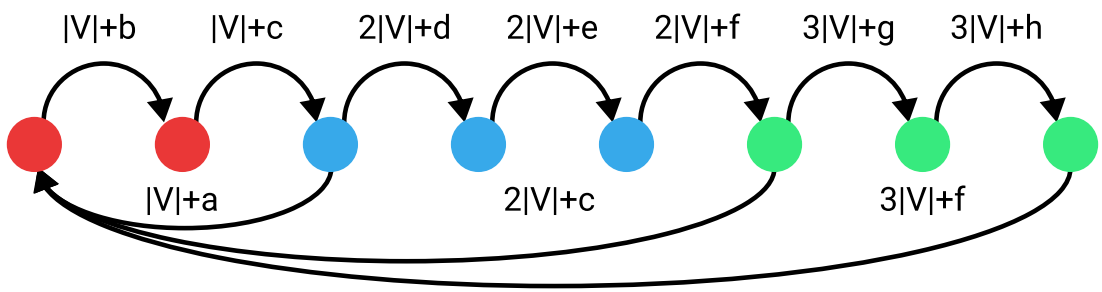
Our current approach:

- We instead consider an initial balanced partitioning via a fast k-way spectral cut using Sphynx/Multi-Jagged in Trilinos.
- To mitigate memory overheads, we developed a **constant memory representation for an arbitrary number of coarsening levels**. We can extract the graph at any level.
- For coarsening, we are developing a **parallel spectrum preserving method with nice theoretical guarantees**.
- For refinement, we can use XtraPuLP to give us a fast, scalable, and relatively high quality final partition.

Constant memory coarsening

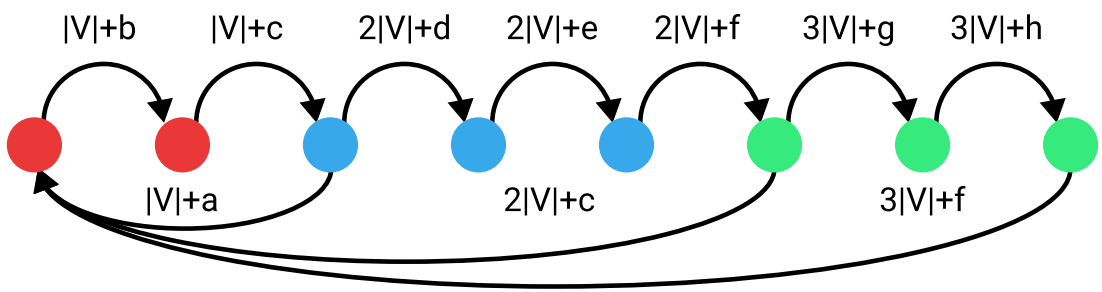
- Want to perform multilevel coarsening on a graph $G = (V, E)$ with constant memory overhead.
- We can do this using an array of size $2(n + m)$, where $n = |V|$ and $m = |E|$.

a. Start with an adjacency list with $2m$ entries.
b. For each node v add an entry $nr + u$. r is the level of coarsening that v absorbed a node and u is the node that was absorbed by v.
c. For each node v add an entry $nk + u$. k is the level of coarsening that v was absorbed and u is the node that absorbed v.
d. The first value acts as a pointer to continue the node adjacency of subsuming nodes. The other determines what node edges point to in the k^{th} level of coarsening. We see this holding true in practice.



Constant memory coarsening

- Using this data structure we perform operations on the coarsened graph by checking our pointers.
- for instance...
  a. Can find the last node in a subsumption chain in $O(k)$ operations, where k is the level of coarsening.
  b. Can determine the first node in our subsumption chain in $O(log_2(k))$ operations.
  c. Can determine the nodes at the k^{th} level of coarsening in $O(n)$ operations.
  d. Can recover the k^{th} level graph in $O(n) + O(mlog_2(k))$ operations.



Spectral partitioning

We would like to be able to partition a given graph $G = (V, E)$ into equally weighted portions. A powerful tool for this is spectral partitioning as it provides us with useful global information and guarantees on cut quality.

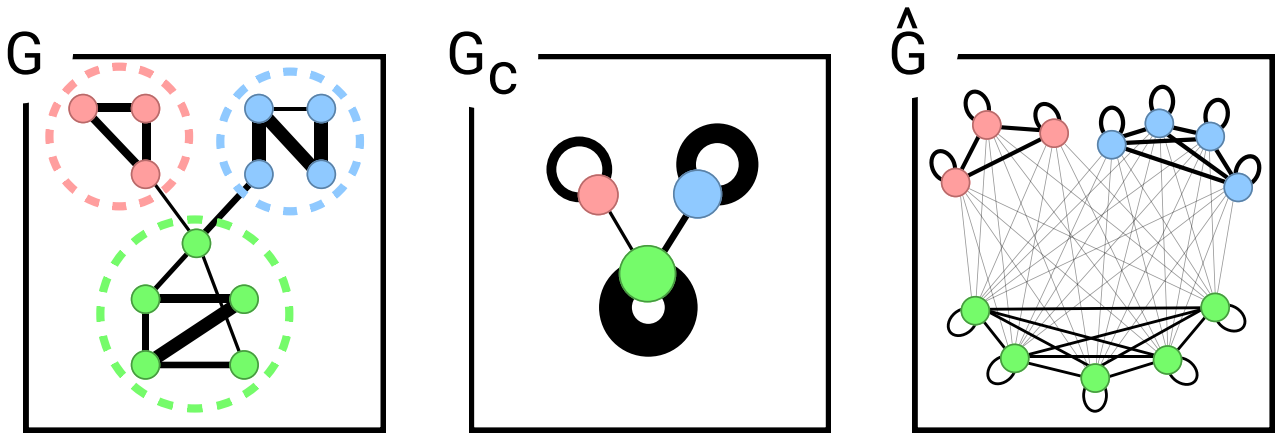
- Requires $O(n^3)$ operations in the dense case.
- Requires $O(m)$ operations in the sparse case, with limited room for parallelism, and possible convergence issues.

Is there a way to coarsen a graph while closely preserving this information?
Yes! However, they tend to be slow!
(Lescoat et al. 2020, Loukas 2019, Jin. Loukas 2020, Liu et al. 2019)

A spectral bound

A useful result (Jin. Loukas 2020) states the following...
If Λ and $\hat{\Lambda}$ sorted vectors containing the eigenvalues of G and $\hat{G}$ respectively. Then if we merge two nodes $u, v \in V$ such that $\| \frac{w_u}{d_u} - \frac{w_v}{d_v} \|_1 \leq \epsilon$, we have,

$$\|\Lambda - \hat{\Lambda}\|_\infty \leq \epsilon$$



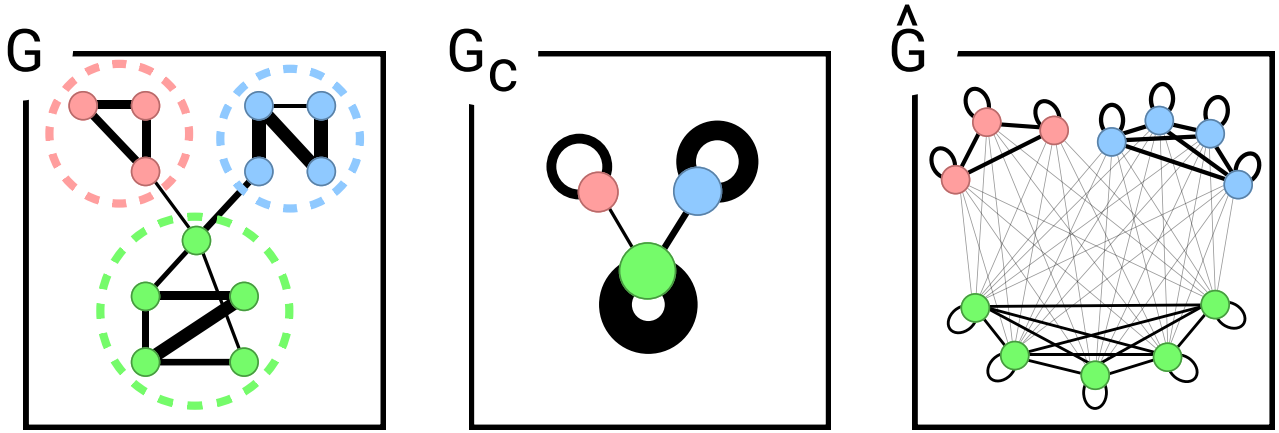
Explicit greedy method

Algorithm 1 Multilevel Graph Coarsening (**MGC**)

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \boldsymbol{W})$ and target size of the coarse graph n.
$s \leftarrow N$
while $s > n$ **do**
    for $v_i \in \mathcal{V}_s$ **do**
        for $v_j \in \mathcal{N}_i$ **do**
            $d_s(i,j) = \left\| \frac{\boldsymbol{w}(i)}{d(i)} - \frac{\boldsymbol{w}(j)}{d(j)} \right\|_1$
    $i_{\min}, j_{\min} = \arg\min_{i,j} d_s(i,j)$
    $s \leftarrow s - 1$
    Merge nodes $v_{i_{\min}}$ and $v_{j_{\min}}$ to form the coarse graph $\mathcal{G}_s$.
return $\mathcal{G}_n = (\mathcal{V}_n, \mathcal{E}_n, \boldsymbol{W}_n)$

It's too slow!

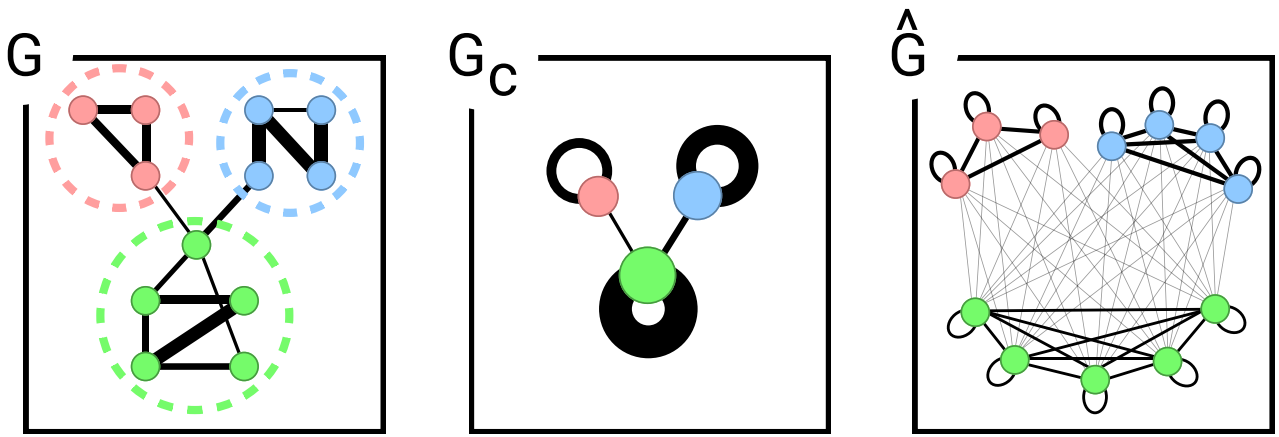
Their algorithm requires $O(n^2 m)$ work with dense vector operations, or $O(n^2 \langle d^2 \rangle)$ work with sparse vectors.
We propose a heuristic to reduce this to $O(n \langle d^2 \rangle)$ work where $\langle d^2 \rangle$ is the second moment of the degree distribution. Furthermore our method is parallelizable and can be performed in $O(\frac{n}{p} \langle d^2 \rangle)$ parallel time.



An observation

Spectral error arising from merges only compounds when multiple merges overlap.

- This means our error is bounded relative to our largest merged subgraph, instead of all merges.
- In fact, if we only perform merges between nodes which are connected, the spectral error after all merges will be bounded above by $D\epsilon$, where D is the diameter of the largest merged subgraph, and ϵ is the largest norm-difference between any two individual merged nodes.



An observation

This observation allows us to never have to recompute norm-differences!

- We can obtain a spectral approximation of our graph using only the norm-differences $\| \frac{w_u}{d_u} - \frac{w_v}{d_v} \|_1$ in the original graph.
- This avoids the outer-loop in their explicit greedy method, and reduces the complexity by a factor of n, giving us a $O(n \left\langle d^2 \right\rangle)$ work algorithm.

Our algorithm

Input: G = (V, E), r, p
Result: coarsened graph G_c
$G_c \leftarrow G$
merge_fitness $\leftarrow \emptyset$
$\{E_1, \cdots, E_p\} \leftarrow edgePartition(E, p)$
for *i=1 to p in parallel* **do**
  for *j=1 to* $|E_i|$ **do**
    u $\leftarrow E_i[j][1]$
    v $\leftarrow E_i[j][2]$
    merge_fitness $\leftarrow merge\_fitness \cup (\| \frac{w_u}{d_u} - \frac{w}{d_v} \|_1, E_i[j])$
  end
end
ascendingSort(merge_fitness)
for *i=1 to* $|V| - r$ **do**
  $G_c \leftarrow merge(G_c, merge\_fitness[i][0], merge\_fitness[i][2])$
end
return: G_c

Our algorithm

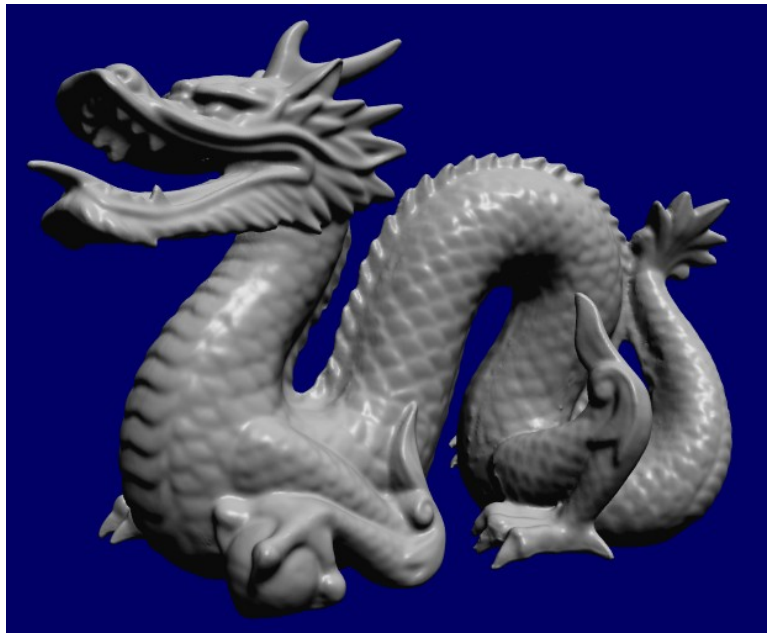
As mentioned, out algorithm without parallelization has serial time $O(n\left\langle d^2\right\rangle)$, and with parallelization the edge-weight calculations are reduced to $O(\frac{n}{p}\left\langle d^2\right\rangle)$.

- We then perform a sort that requires O(mlog(m)) work. Can be parallelized to $O(\frac{m}{p}log(m))$ time.
- The merges must be done sequentially, and require O($n - r$) work.

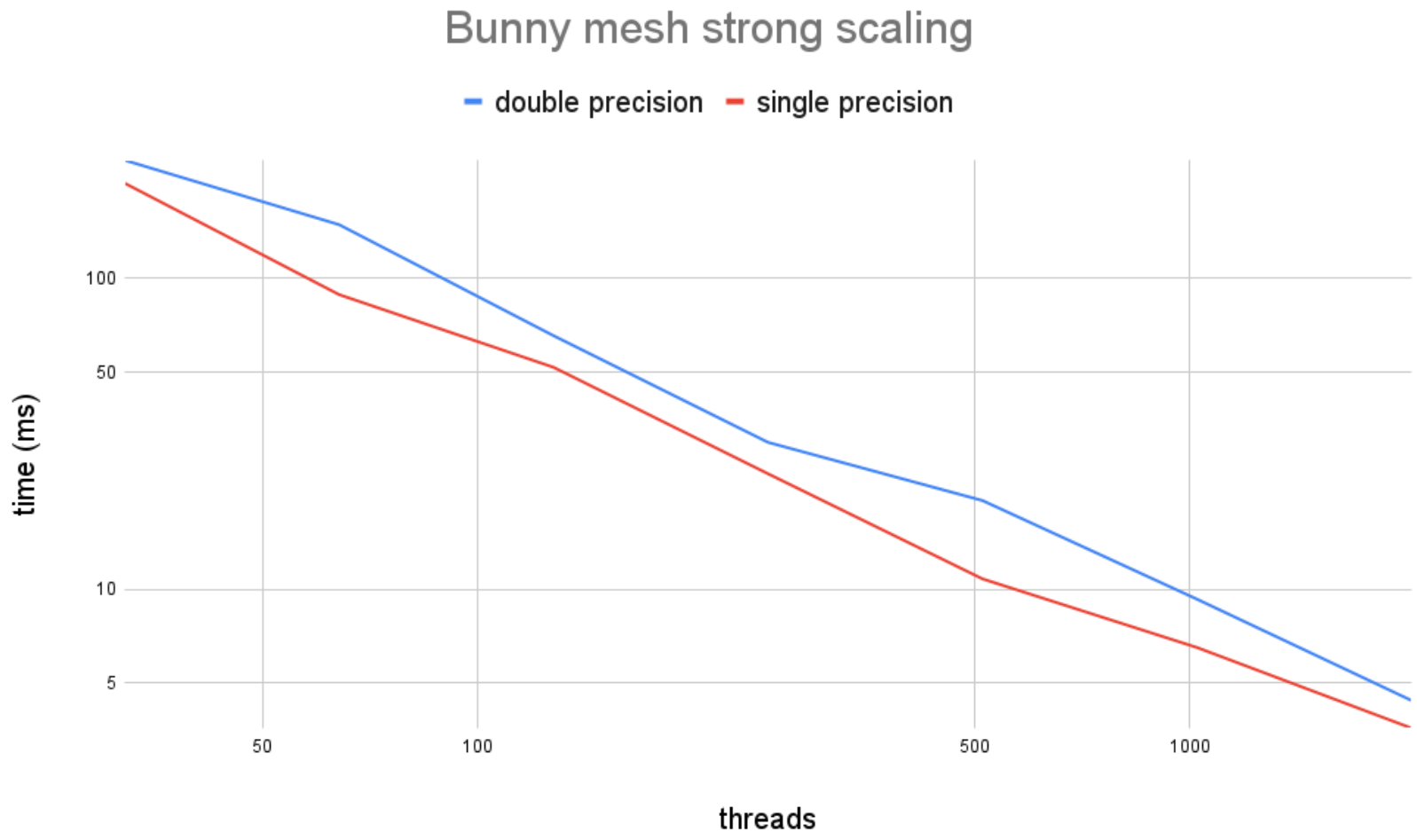
Overall for this leaves us with an $O(\frac{n}{p}\left\langle d^2\right\rangle + \frac{m}{p}log(m) + (n - r))$ parallel time algorithm.

Results

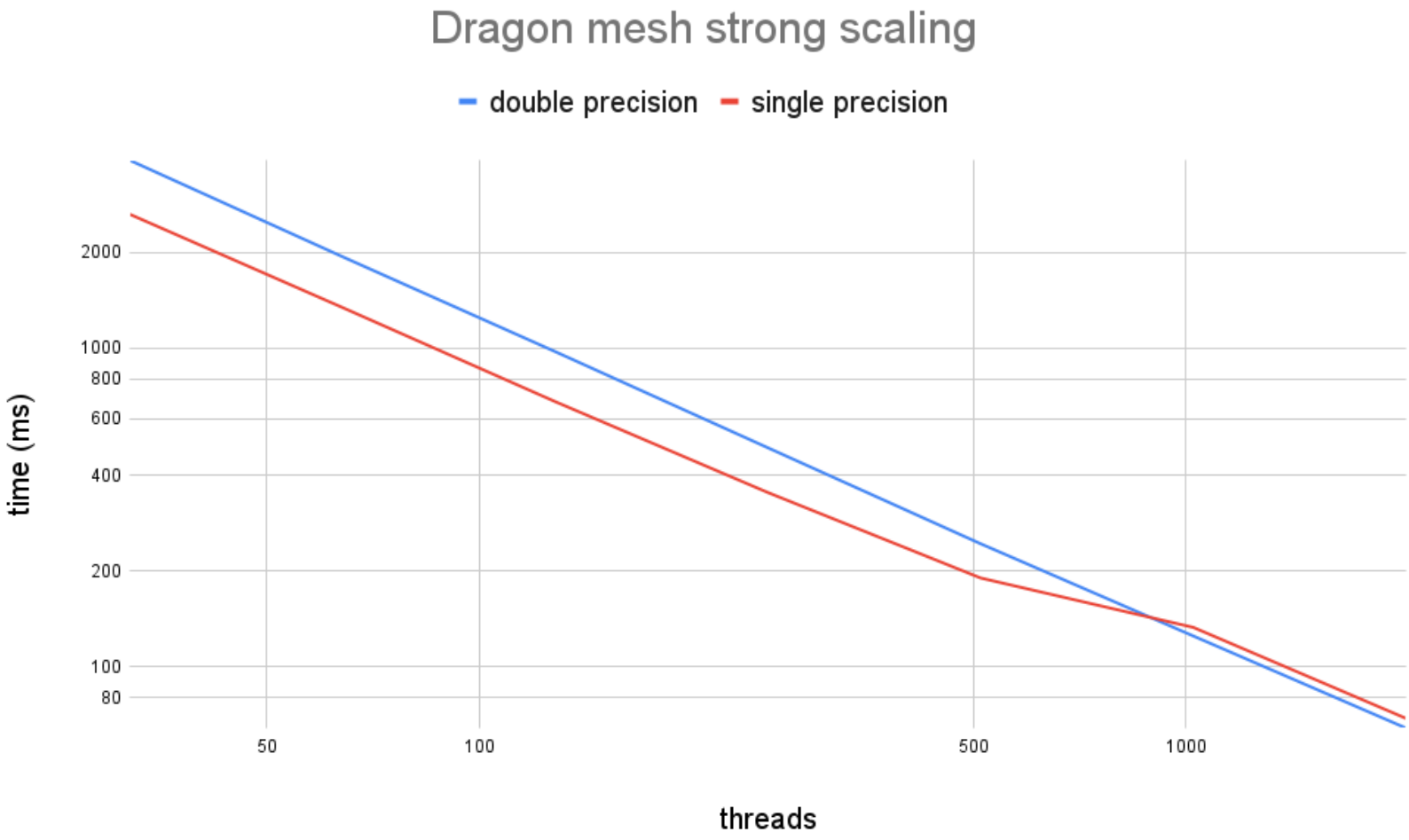
We present scaling results from three example graphs. Two meshes from the Stanford 3D scanning repository, the “Stanford Bunny” (35,947 nodes, 104,176 edges) and “Dragon” (437,645 nodes, 1,307,121 edges), as well as the small “ego-Facebook” (4,039 nodes, 88234 edges) graph from the SNAP network repository.
We also present some spectral approximation results.
*All runs were on a single nVidia Quadro M5000



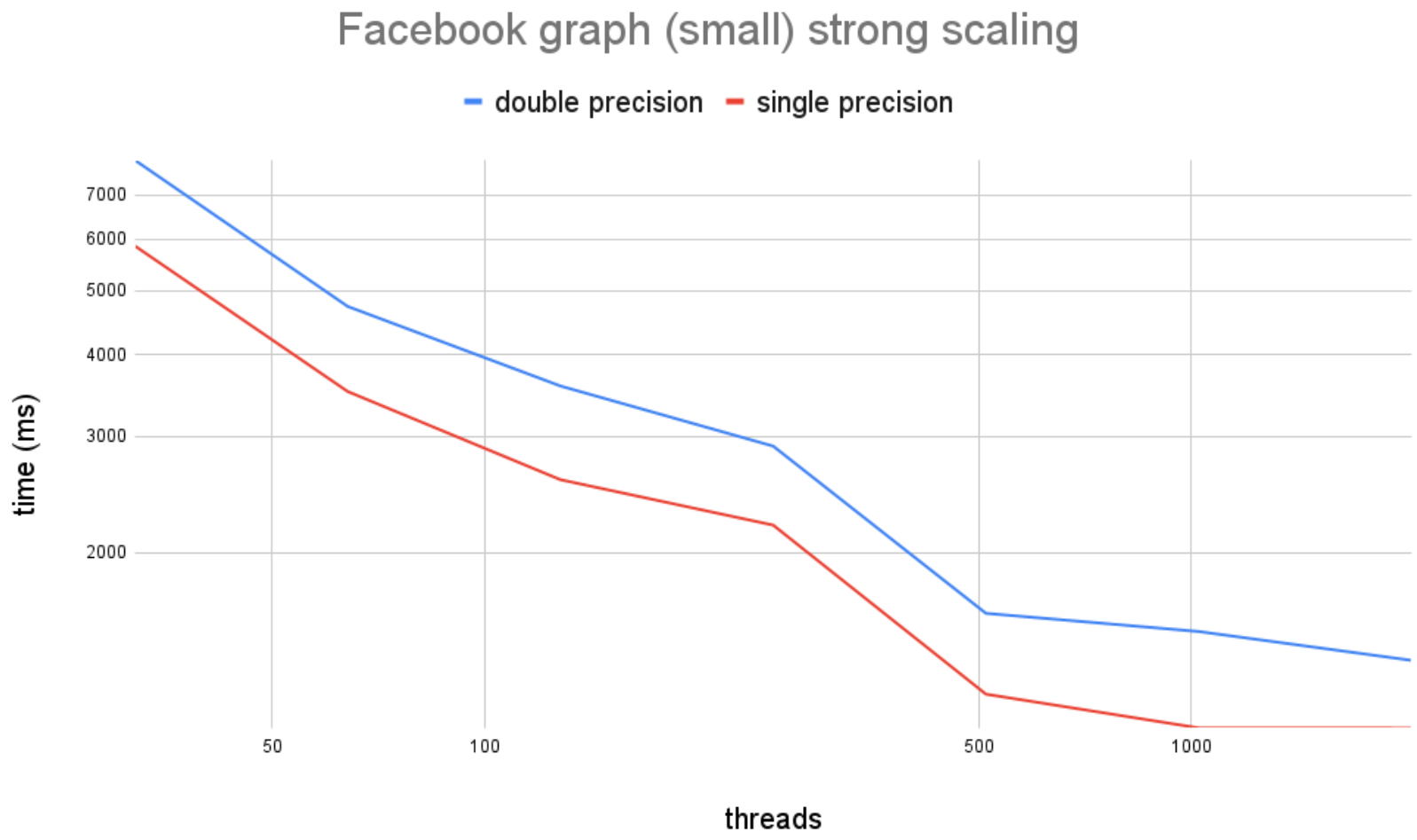
Scaling Results



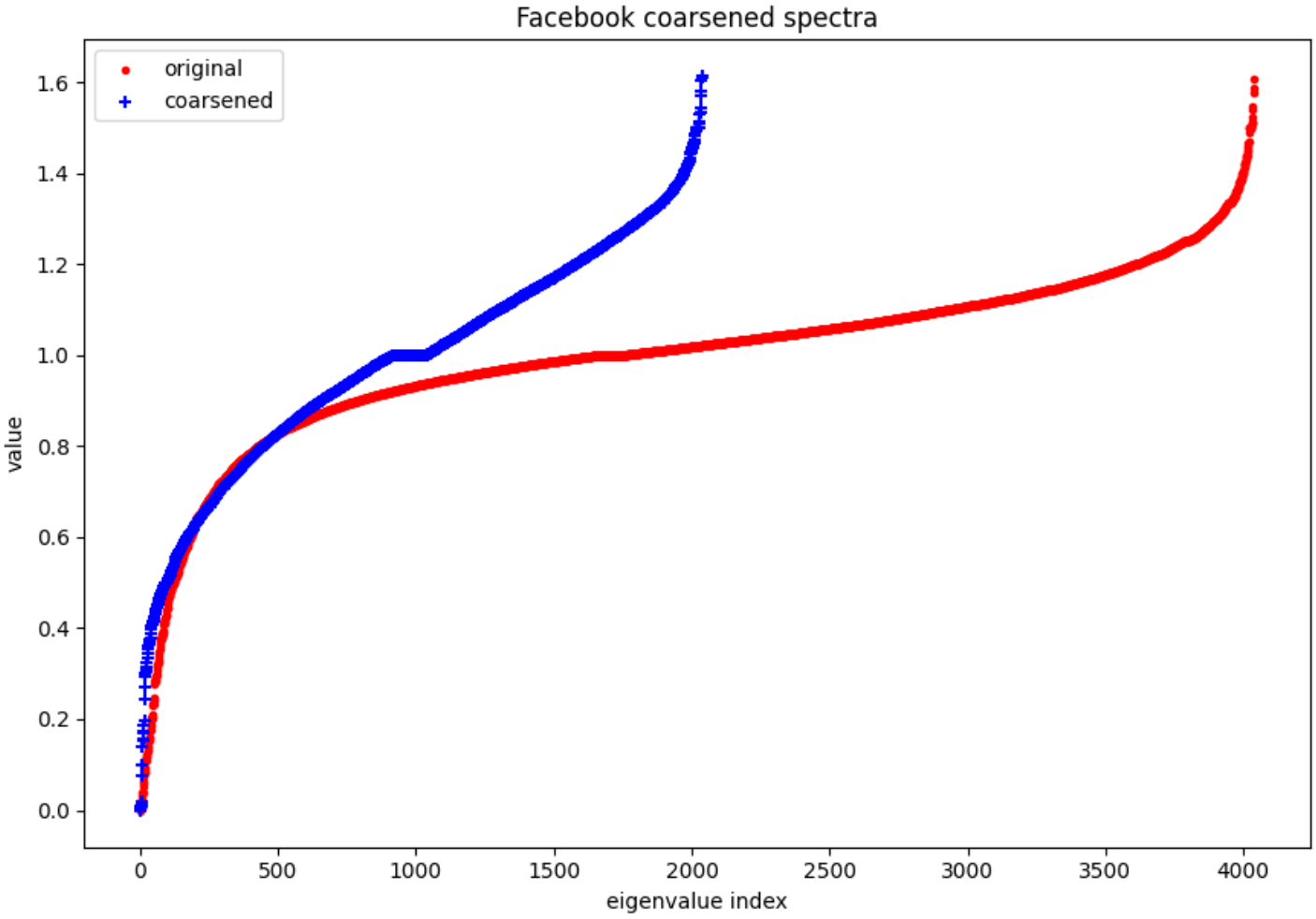
Scaling Results



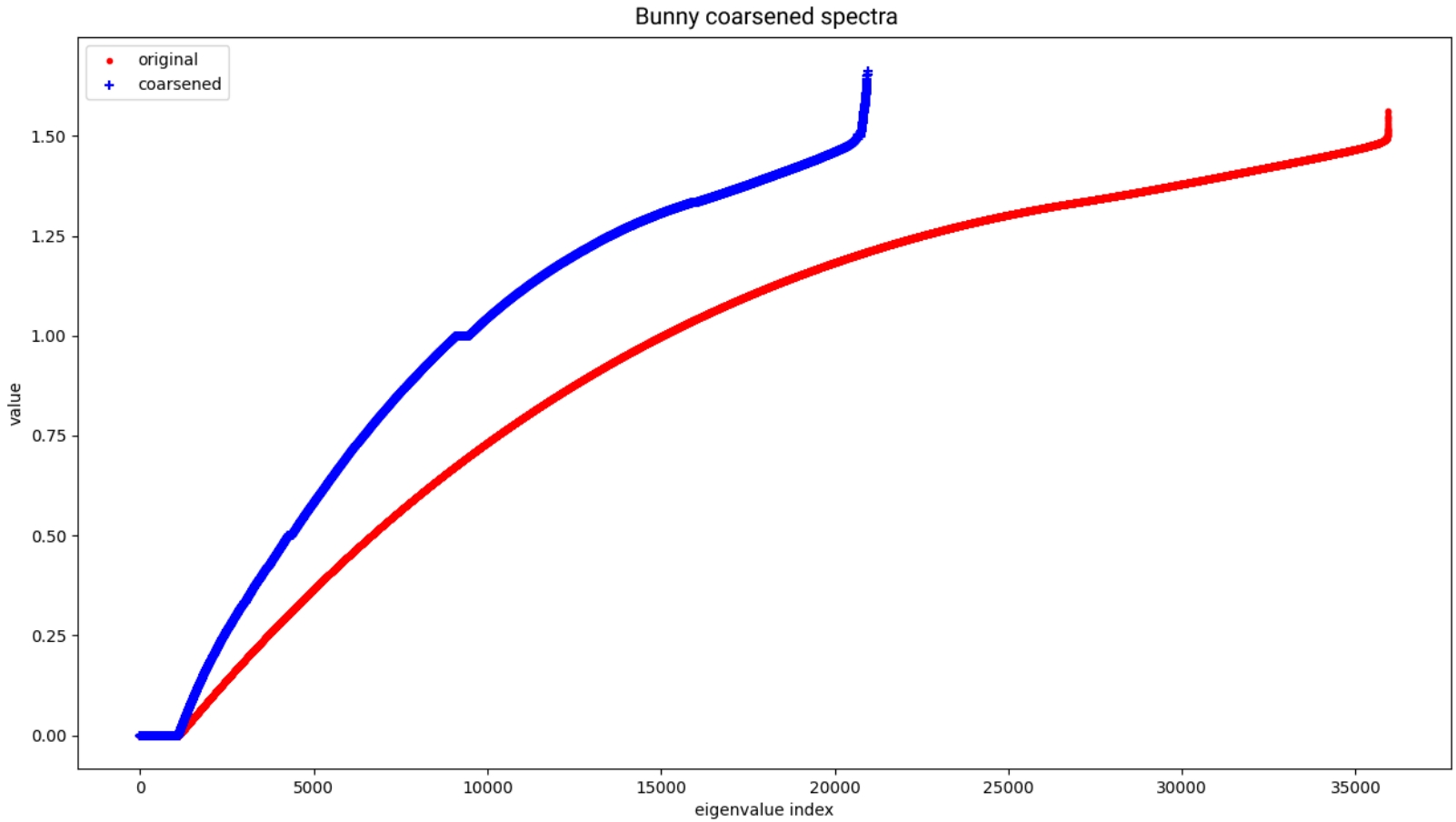
Scaling Results



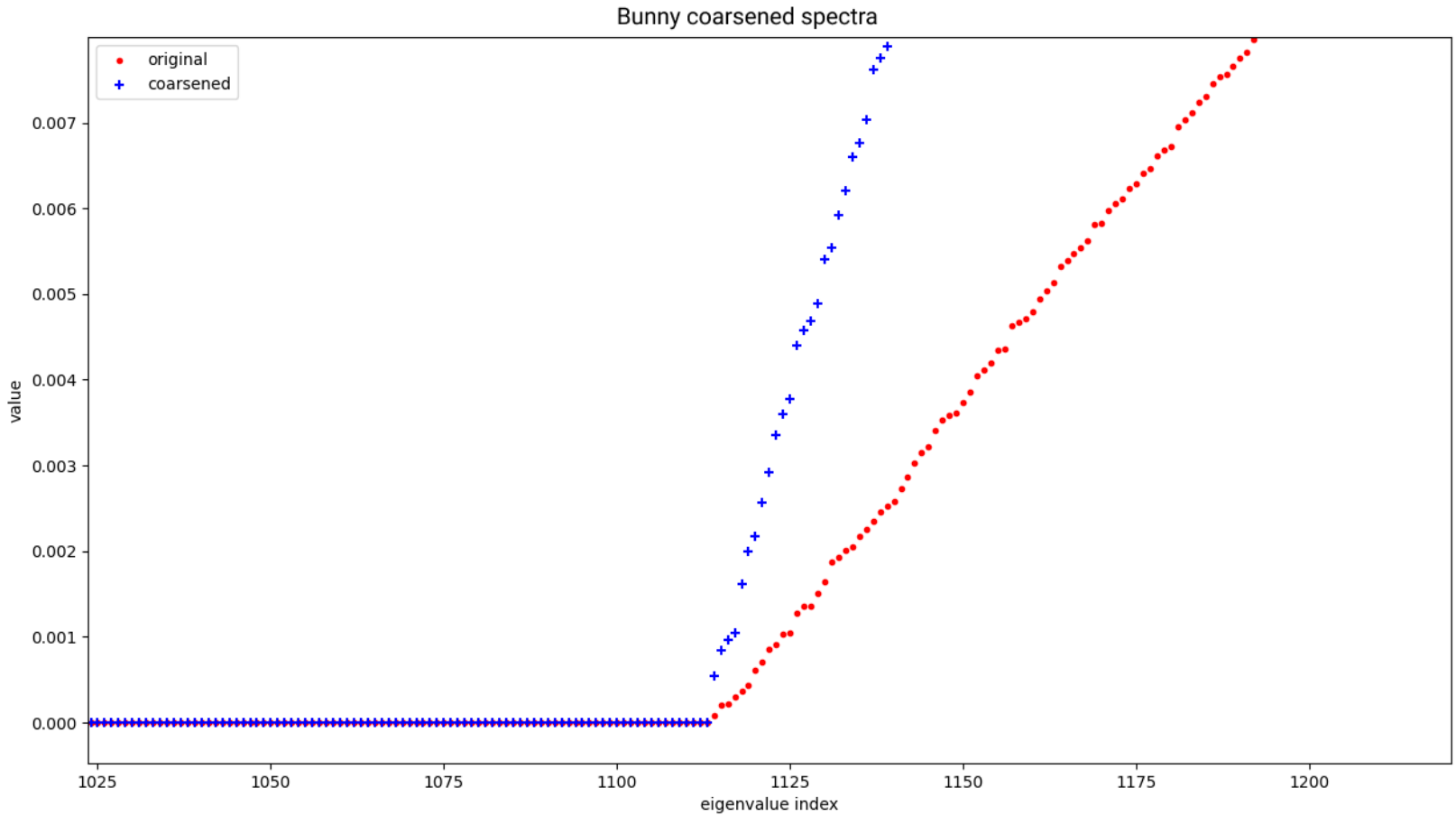
Spectrum Results



Spectrum Results



Spectrum Results



Future Work

- As mentioned, the merges currently have to be done serially. However, this isn't necesarily true. If we have disjoint merges they can be parallelized. Therefore, by weighting edges inversely to their fitness function and performing a maximal matching we can further speed up our algorithm via parallel merges.
- This method relies on storing several arrays in GPU memory. This quickly leads to spatial constraints. Large graphs will require distributing the structure over multiple GPUs.
- One of our primary avenues for ongoing and future work is full integration with Sphynx/Multi-Jagged and XtraPuLP in distributed memory.

Acknowledgements

- This work was supported by the U.S. Department of Energy, Office of Science, under award DE-AC52-07NA27344 (FASTMath SciDAC Institute).
- This work was also supported under Laboratory Directed Research and Development (LDRD) program, Project No. 218474, at Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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-NA-0003525.
- This presentation describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

Rensselaer

Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY

Summary and Thanks

- We implemented a constant-memory graph representation for graph coarsening.
- We developed a spectrum preserving coarsening algorithm.
- We observe constant memory utilization in practice and promising strong scaling results for our coarsener.
- TODO: extend our coarsening to distributed GPUs and integrate with other methods for full partitioning.

Thank you!

Contact: brissc@rpi.edu, slotag@rpi.edu