2D Sparse Communication Methods for Maximum Weight Matching Applications on GPUs

Michael Mandulak
George M. Slota

Rensselaer Polytechnic Institute

SIAM CSE25: FASTMATH Advances in High-Performance Computing for Sparse Systems, Multilevel Methods, and Scientific Simulations

March 5, 2025

Application Focus

Target: Improve graph partitioning applications by extending previous works in the field

Previous Works

- Parallel improvements to coarsening → GPU acceleration + comprehensive study
  - *Performance-Portable Graph Coarsening for Efficient Multilevel Graph Analysis* (Gilbert et al., 2021)
- Constant-memory data structure for coarsening + GPU accelerated access methods
  - *A Constant-memory Framework for Graph Coarsening* (Slota & Brissette, 2024)

Baseline: GPU accelerated + memory efficient coarsening framework for multilevel partitioning

Focus: How much more performance can we gain in the coarsening/matching component?

A Step Back: Maximum Weight Matching Problem

Matching: A matching *M* is a subset of edges such that no two edges in *M* are incident on the same vertex

- Common use cases:
  - Load balancing
  - Sparse Matrix Computations
- Applications:
  - Sparse linear solvers
  - Network switching
  - **Graph partitioners**
  - **Coarsening**
- **Problem:** Optimal matching runtimes

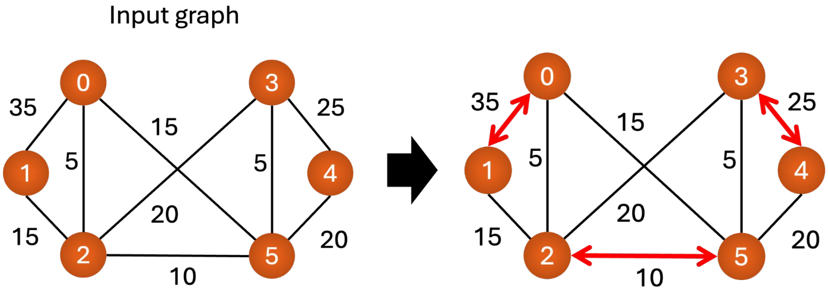
Our focus:

Approximation methods for maximum weighted matching!

- Modern implementations are very fast!
- How much performance gain from using approximate methods?

Target: Apply approx. algorithms with provable bounds within coarsening

Maximum Weight Matching Problem (MWM)



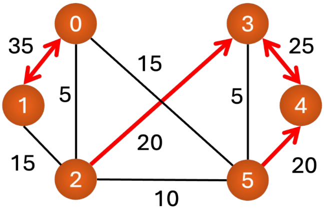
Find a matching set of edges to maximize the total weight

Traditional Half-approx Methods

Suitor: proposal + consideration-based
Locally-Dominant (LD): choose best neighbor → commit mutuals

Locally-Dominant (LD) MWM Method

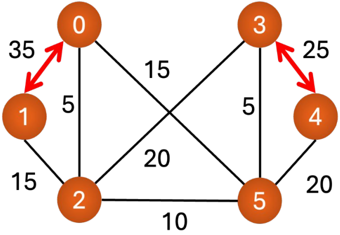
Phase 1: pointing



Vertex-independent choice of highest weight available neighbor

Tentative selection of matching partner

Phase 2: matching



Commit mutually pointing matching pairings

Reset pointers for singleton vertices and repeat

Matching Results in Practice (METIS)

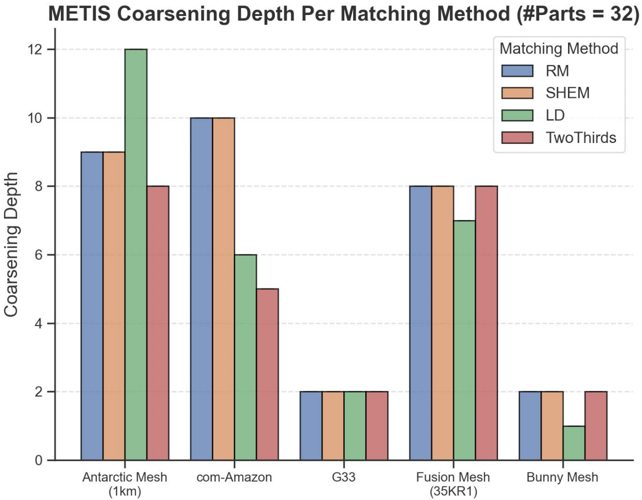
Focus: Test alternative matching strategies within partitioning

- Matching Methods
  - Locally-Dominant (LD)
  - $\frac{2}{3} - \varepsilon$ – approximate (Two Thirds) – augmenting path-based

- Sequential METIS implementation
  - Sequential matching implementations – quality-based comparison
  - Metrics:
    - Edgecut (unit weights)
    - Coarsening Depth (initial #levels)

- Datasets
  - Small-medium meshes (2K – 13M vertices, 4K – 53M edges)
  - Small real-world graphs (<500K vertices, <1M edges)

Matching Results – Coarsening Depth

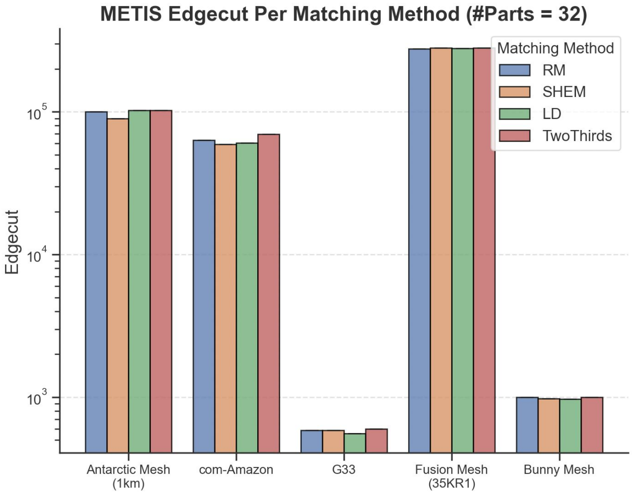


Most cases – reduction in coarsening depth

Primarily mesh datasets (+amazon)

Large impact on performance (when scaled)

Matching Results – Partition Quality



Edgecuts remain consistent

Focus: improve performance w/ consistent quality

Variability is slight among diff. #Parts

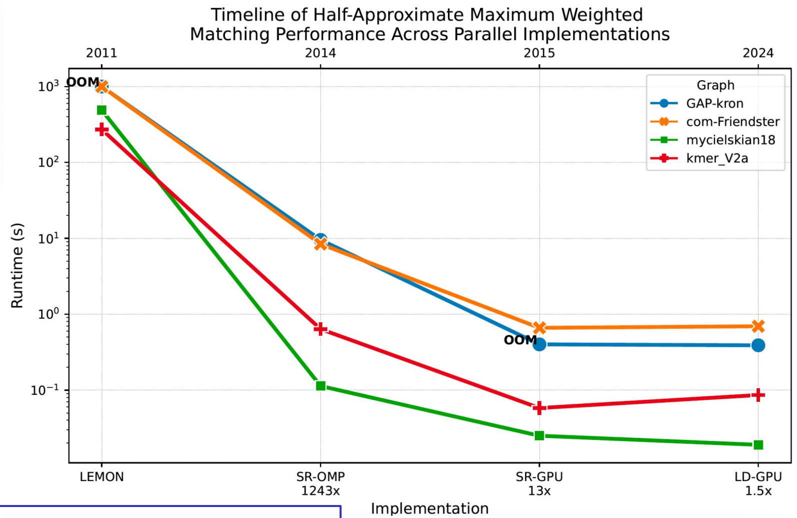
Application Focus - Goals

Idea: Can generally reduce the number of coarsening levels using approximate matching methods

Baseline: GPU accelerated + memory efficient coarsening framework for multilevel partitioning

Focus: GPU accelerated + scalable matching methods for usage in coarsening

Parallel Approximate Matching Methods



LEMON: Sequential optimal
SR-OMP: OpenMP Suitor
SR-GPU: GPU Suitor
LD-GPU: Locally-Dominant Multi-GPU

Manne et al., *New effective multi-threaded matching algorithms.* IPDPS 2014
Naim et al., *Optimizing approximate weighted matching on Nvidia Kepler K40.* HiPC 2015
Mandulak et al., *Efficient Weighted Graph Matching on GPUs.* SC'24

Alternative Improvements: Communication

Idea: Target new areas of improvement to traditional half-approx methods

LD-GPU bottleneck: Communication costs!

- Synchronized pointers and matching each round

Improve Communication Setup: 2D Approach

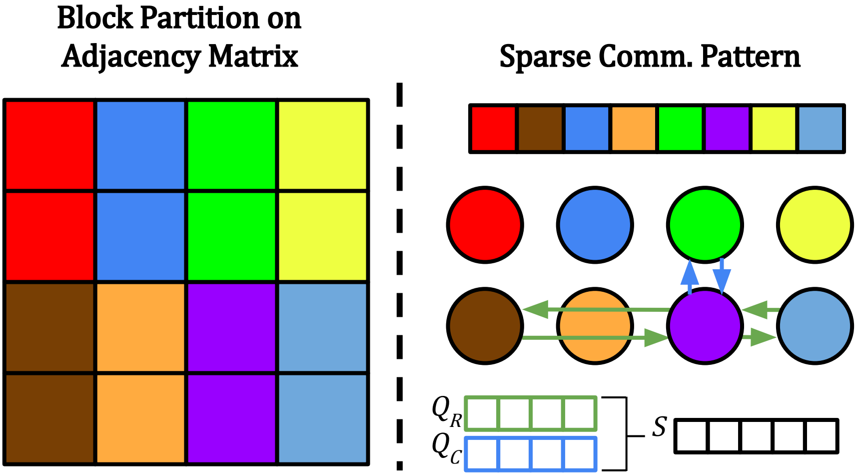
Vertex vs. Edge Split

Better balance for nonzeros

→

Graphics: Erik G. Boman, Karen D. Devine, and Sivasankaran Rajamanickam. Scalable matrix computations on large scale-free graphs using 2d graph partitioning. In Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis, SC '13, New York, NY, USA, 2013. Association for Computing Machinery.

2D Partitioning



LD Sparse - Algorithm

Algorithm 1 2D LD Matching

Input: Graph: $G(V, E)$

Output: Matching M

$Q \leftarrow$ `InitQueue`(V)

while $\exists$ matching edges **do**

  $ptrs \leftarrow$ `LocalMC`(Q, q_{in})

  $sbuf \leftarrow$ `BuildQueue`$(G, q_{in}, ptrs)$

  $rbuf \leftarrow$`Allgatherv`$(sbuf,$`COL_COMM`$)$

  $Q \leftarrow$ `ReduceQueue`$(G, rbuf)$

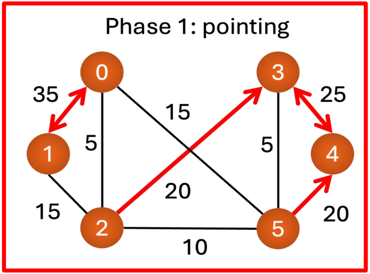
  $ptrs \leftarrow$ `UpdatePtrs`$(G, rbuf)$

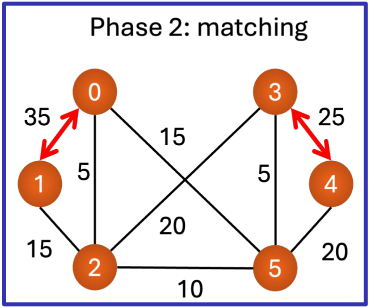
  $sbuf \leftarrow$ `BuildQueue`$(G, q_{in}, ptrs)$

  $rbuf \leftarrow$`Broadcast`$(sbuf,$`ROW_COMM`$)$

  $M \leftarrow$ `MutualCheck`$(ptrs)$

  $Q \leftarrow$ `ReduceQueue`$(G, rbuf)$





LD Sparse - 2D Communication Framework

Algorithm 1 2D LD Matching

Input: Graph: $G(V, E)$
Output: Matching M

```
Q ← InitQueue(V)
while ∃ matching edges do
    ptrs ← LocalMC(Q, q_in)
    sbuf ← BuildQueue(G, q_in, ptrs)
    rbuf ← Allgatherv(sbuf, COL_COMM)
    Q ← ReduceQueue(G, rbuf)
    ptrs ← UpdatePtrs(G, rbuf)
    sbuf ← BuildQueue(G, q_in, ptrs)
    rbuf ← Broadcast(sbuf, ROW_COMM)
    M ← MutualCheck(ptrs)
    Q ← ReduceQueue(G, rbuf)
```

Communication Pattern:

1. Computation
2. Compile relevant data (active vertices among group - <u>sparse</u>)
3. Communicate among row/column group
4. Reduce relevant data to global data queue

<u>Can repeat this block per row/column or as needed</u>

LD Sparse - 2D Communication Framework

Algorithm 1 2D LD Matching

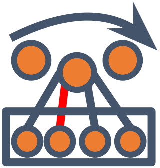
Input: Graph: $G(V, E)$
Output: Matching M

$Q \leftarrow \texttt{InitQueue}(V)$
while $\exists$ matching edges **do**
    $\boxed{ptrs \leftarrow \texttt{LocalMC}(Q, q_{in})}$
    $sbuf \leftarrow \texttt{BuildQueue}(G, q_{in}, ptrs)$
    $rbuf \leftarrow \texttt{Allgatherv}(sbuf, \texttt{COL\_COMM})$
    $Q \leftarrow \texttt{ReduceQueue}(G, rbuf)$
    $ptrs \leftarrow \texttt{UpdatePtrs}(G, rbuf)$
    $sbuf \leftarrow \texttt{BuildQueue}(G, q_{in}, ptrs)$
    $rbuf \leftarrow \texttt{Broadcast}(sbuf, \texttt{ROW\_COMM})$
    $M \leftarrow \texttt{MutualCheck}(ptrs)$
    $Q \leftarrow \texttt{ReduceQueue}(G, rbuf)$

Communication Pattern:

1. Computation
2. Compile relevant data (active vertices among group - sparse)
3. Communicate among row/column group
4. Reduce relevant data to global data queue

LD Phase 1: Neighborhood scan + set pointers



LD Sparse - 2D Communication Framework

Algorithm 1 2D LD Matching

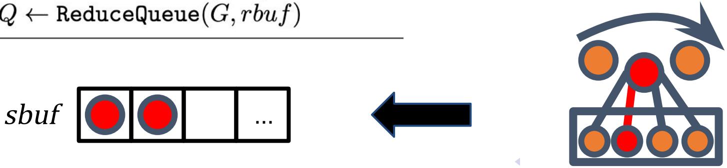
Input: Graph: $G(V, E)$
Output: Matching M

```
Q ← InitQueue(V)
while ∃ matching edges do
    ptrs ← LocalMC(Q, q_in)
    sbuf ← BuildQueue(G, q_in, ptrs)
    rbuf ←Allgatherv(sbuf,COL_COMM)
    Q ← ReduceQueue(G, rbuf)
    ptrs ← UpdatePtrs(G, rbuf)
    sbuf ← BuildQueue(G, q_in, ptrs)
    rbuf ←Broadcast(sbuf,ROW_COMM)
    M ← MutualCheck(ptrs)
    Q ← ReduceQueue(G, rbuf)
```

Communication Pattern:

1. Computation
2. Compile relevant data (active vertices among group - <u>sparse</u>)
3. Communicate among row/column group
4. Reduce relevant data to global data queue

Active Vert → if pointer/pointee was updated



LD Sparse - 2D Communication Framework

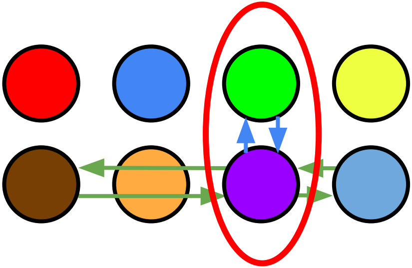
Algorithm 1 2D LD Matching

Input: Graph: $G(V, E)$
Output: Matching M

$Q \leftarrow$ `InitQueue`(V)
while $\exists$ matching edges **do**
  $ptrs \leftarrow$ `LocalMC`(Q, q_{in})
  $sbuf \leftarrow$ `BuildQueue`$(G, q_{in}, ptrs)$
  $rbuf \leftarrow$ `Allgatherv`$(sbuf$, `COL_COMM`$)$
  $Q \leftarrow$ `ReduceQueue`$(G, rbuf)$
  $ptrs \leftarrow$ `UpdatePtrs`$(G, rbuf)$
  $sbuf \leftarrow$ `BuildQueue`$(G, q_{in}, ptrs)$
  $rbuf \leftarrow$ `Broadcast`$(sbuf$, `ROW_COMM`$)$
  $M \leftarrow$ `MutualCheck`$(ptrs)$
  $Q \leftarrow$ `ReduceQueue`$(G, rbuf)$

Communication Pattern:

1. Computation
2. Compile relevant data (active vertices among group - sparse)
3. Communicate among row/column group
4. Reduce relevant data to global data queue



LD Sparse - 2D Communication Framework

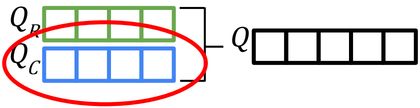
Algorithm 1 2D LD Matching

Input: Graph: $G(V, E)$
Output: Matching M

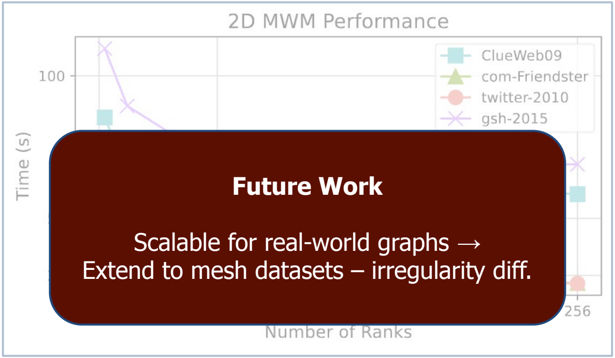
$Q \leftarrow \texttt{InitQueue}(V)$
while $\exists$ matching edges **do**
  $ptrs \leftarrow \texttt{LocalMC}(Q, q_{in})$
  $sbuf \leftarrow \texttt{BuildQueue}(G, q_{in}, ptrs)$
  $rbuf \leftarrow \texttt{Allgatherv}(sbuf, \texttt{COL\_COMM})$
  $\boxed{Q \leftarrow \texttt{ReduceQueue}(G, rbuf)}$
  $ptrs \leftarrow \texttt{UpdatePtrs}(G, rbuf)$
  $sbuf \leftarrow \texttt{BuildQueue}(G, q_{in}, ptrs)$
  $rbuf \leftarrow \texttt{Broadcast}(sbuf, \texttt{ROW\_COMM})$
  $M \leftarrow \texttt{MutualCheck}(ptrs)$
  $Q \leftarrow \texttt{ReduceQueue}(G, rbuf)$

Communication Pattern:

1. Computation
2. Compile relevant data (active vertices among group - sparse)
3. Communicate among row/column group
4. Reduce relevant data to global data queue



LD Results - Scaling



Future Work

Scalable for real-world graphs →
Extend to mesh datasets – irregularity diff.

Higher initial runtimes: computation

Strong scaling in most cases of general graphs

Plateaus at 64 ranks: problem complexity + synch.

Conclusions

- An ongoing work – experimental study to improve matching-based coarsening for partitioning
  - Prelim Results (instances of):
    - Reduced coarsening levels
    - Consistent edgecuts
- Half-approx performance wall for MWM
  - Advancement consideration through other means
    - Communication/Synchronization
    - Quality
- 2D Communication framework
  - LD implementation scalable on real-world graphs
  - Pattern is applicable across graph algorithms

Future Works - Avenues

- Comprehensive results testing (Mesh + Real-World)
  - Partitioning (Edgecut/Coarsening Depth)
  - Matching Weight/Quality correlation

- Matching Performance Wall
  - Alternative approx. algorithms (more $\frac{2}{3}$-ε)
  - LD/Suitor improvements (parallel)

- 2D Communication framework
  - Extend to alternative graph algorithms
    - Coarsening/partitioning framework
    - Include memory-efficient coarsening
  - Optimize LD Sparse

General Goal: GPU accelerated + memory efficient coarsening framework for multilevel partitioning

Acknowledgements & Contact

Contact

- Michael Mandulak: mandum@rpi.edu
- George Slota: slotag@rpi.edu

Acknowledgement

- This work is supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) Program through the FASTMath Institute under Contract No. DE-SC0021285 at Rensselaer Polytechnic Institute, Troy NY and by the National Science Foundation under Grant No. 2047821

Prior Matching Work Acks:

- Sayan Ghosh (PNNL)
- SM Ferdous (PNNL)
- Mahantesh Halappanavar (PNNL)