Distributed-Memory Algorithms for Cardinality Matching using Matrix Algebra

Ariful Azad, Lawrence Berkeley National Laboratory

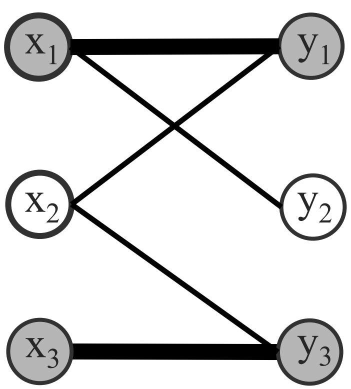
Joint work with Aydın Buluç (LBNL)

Support: DOE Office of Science

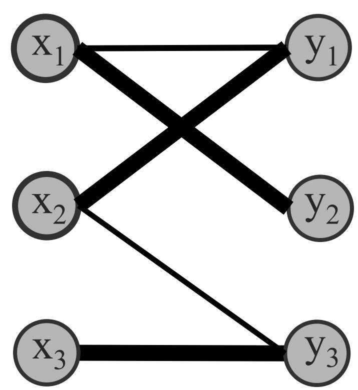
SIAM PP 2016, Paris

A matching in a graph

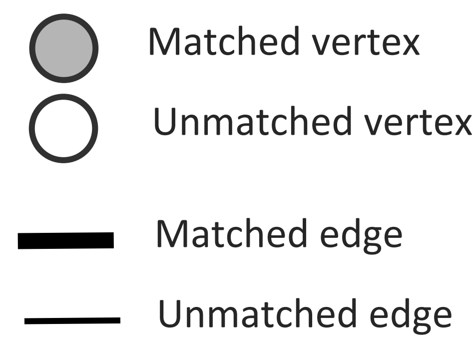
- **Matching:** A subset of *independent* edges, i.e., at most one edge in the matching is incident on each vertex.
- **Maximal cardinality matching:** A matching where if another edge is added it is not a matching anymore.
- **Maximum cardinality matching (MCM)** has the maximum possible cardinality



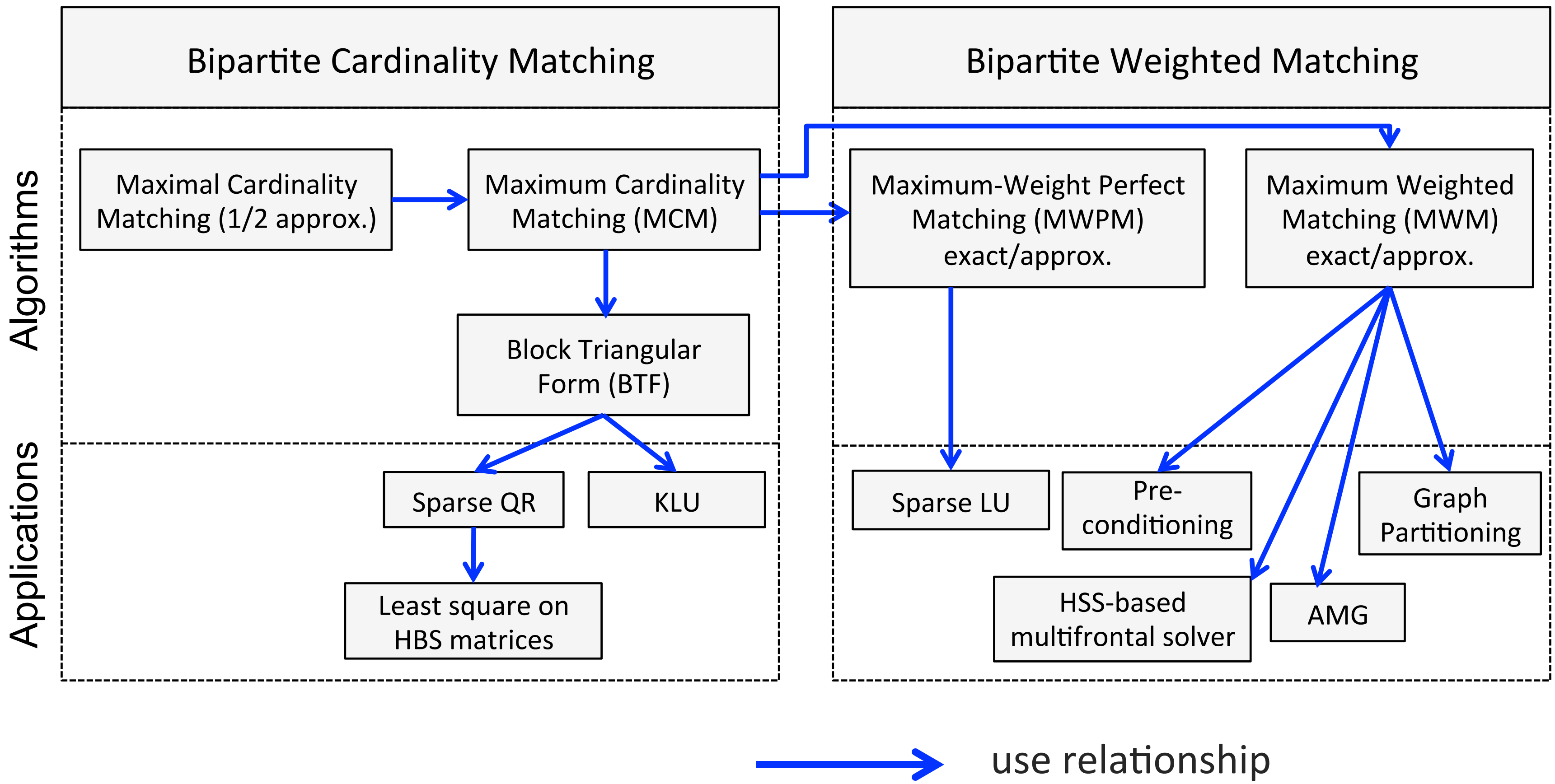
Maximal Cardinality Matching
Cardinality = 2



Maximum Cardinality Matching
Cardinality = 3



Application of matching in scientific computing

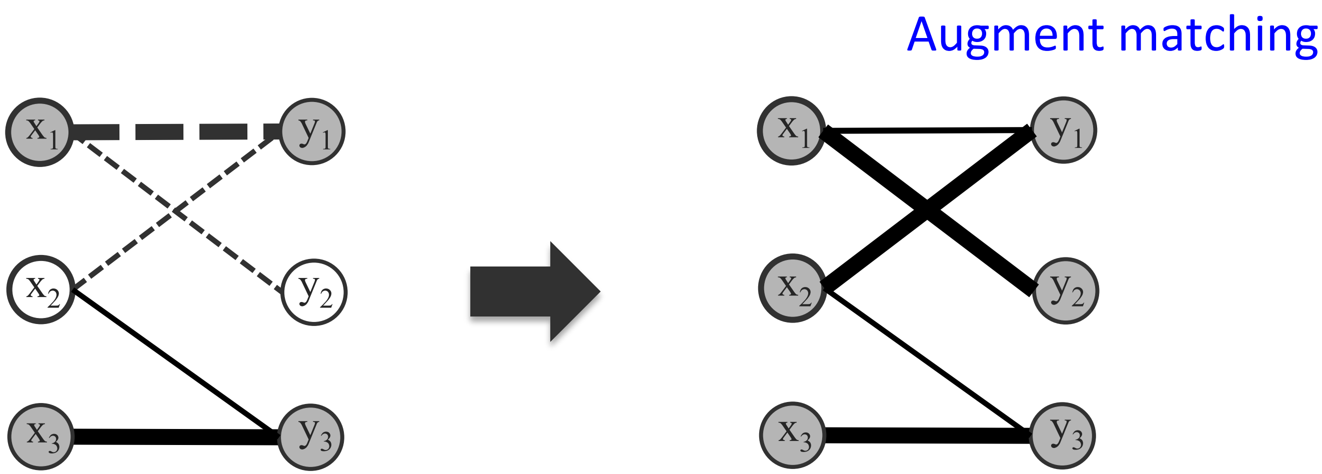


Scope of this talk

- **Problem**: Cardinality matching in a bipartite graph
  - Maximum cardinality matching (**MCM**)
  - Maximal cardinality matching (used to initialize MCM)
- **Algorithm**: Distributed-memory parallel algorithms
- **Approach**: Matrix-algebraic formulations of graph primitives. Inspired by Graph BLAS (http://graphblas.org/).
  - More discussion on Friday (MS68): The GraphBLAS Effort: Kernels, API, and Parallel Implementations by Aydin Buluc.
- Covers two recent papers:
  - Maximal matching: Azad and Buluç, IEEE CLUSTER 2015
  - Maximum matching: Azad and Buluç, IPDPS 2016

MCM algorithm based on augmenting-path searches

- **Augmenting path**: A path that alternates between matched and unmatched edges with unmatched end points.



- **Algorithm: Search** for augmenting paths and flip edges across the paths to increase cardinality of the matching.
  - **Algorithmic options:** single source or multi-source, breadth-first search (BFS) or depth-first search (DFS)

Algorithmic landscape of cardinality matching

Duff, Kaya and Ucar (ACM TOMS 2011), Azad, Buluç, Pothen (TPDS 2016)

	Class	Search strategy	Serial Complexity
Maximum cardinality matching	Single-source augmenting path search	DFS or BFS	O(nm)
	Multi-source augmenting path search	DFS w lookahead (Pothen-Fan)	O(nm)
		BFS (MS-BFS)	O(nm)
		DFS & BFS (Hopcroft-Karp)	O(√nm)
	Push relabel	Label guided FIFO search	O(nm)
Maximal cardinality matching	Greedy Karp-Sipser Dynamic mindegree	Local	O(m)

Hopcroft-Karp: best asymptotic complexity

MS-BFS: exposes more parallelism

Initializes MCM

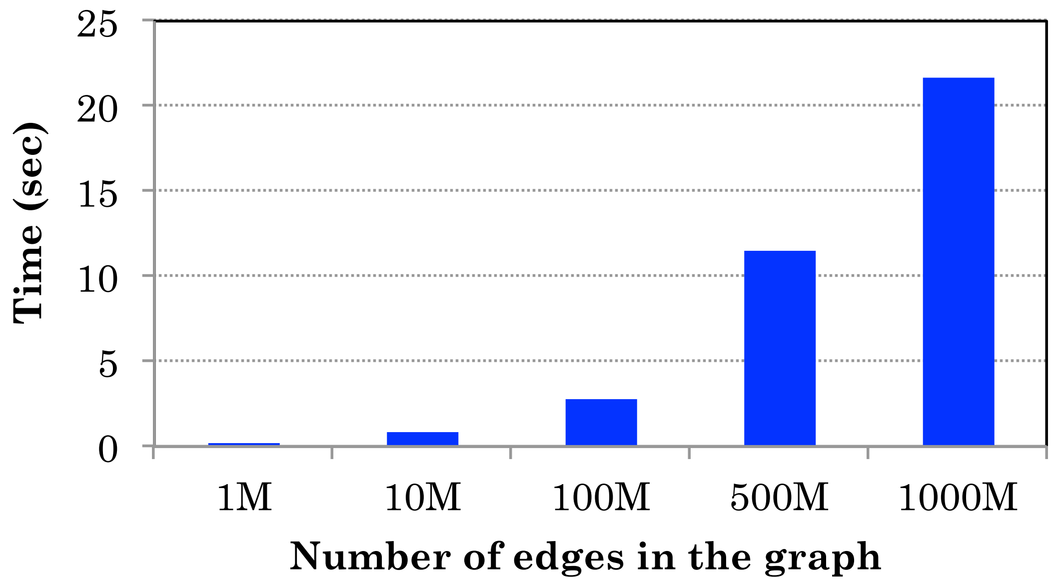
Our focus

The need for distributed-memory algorithms

- When a graph does not fit in the memory of a node
- The graph is already distributed
  - Example: static pivoting in SuperLU_DIST (Li and Demmel, 2003)
  - The graph is gathered on a single node and MC64 is used to compute the matching, which is **unscalable and expensive**

Time to gather a graph and scatter the matching on 2048 cores of NERSC/Edison (Cray XC30)

Distributed algorithms are cheaper and scalable



Distributed-memory cardinality matching

- Prior work: Push-relabel by Langguth *et al.* (2011) and Karp-Sipser on general graph by Patwary *et al.* (2010).
  - does not scale beyond 64 processors

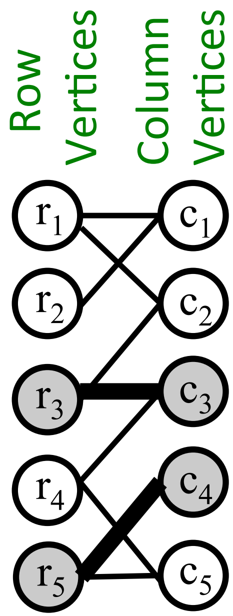
- Challenge
  - long paths passing through multiple processors
  - lots of fine-grained asynchronous communication

- Here we use graph-matrix duality and design matching algorithms using scalable matrix and vector operations.
  - A handful of standard operations
  - Offers bulk-synchronous parallelism
  - Jumping among algorithms is easier

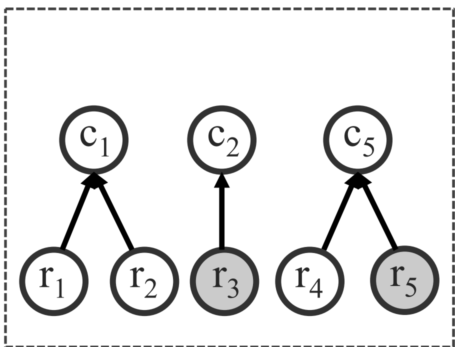
Two required primitives

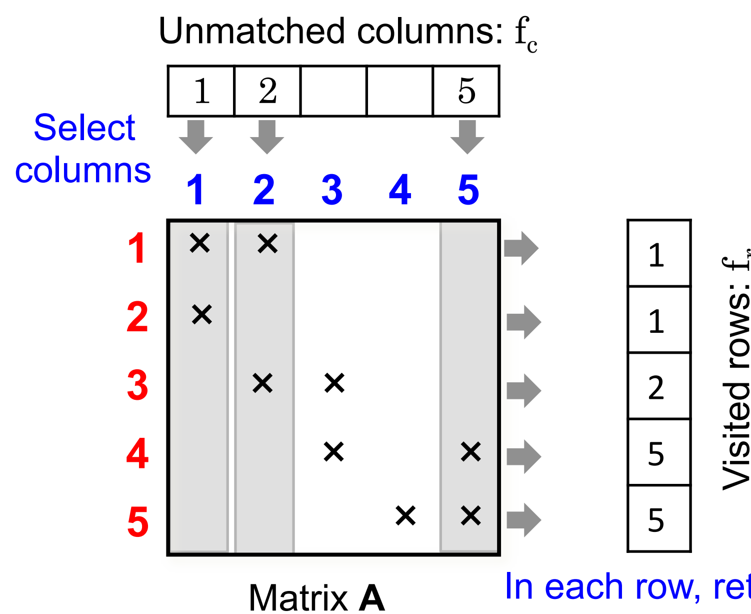
1. Sparse matrix-sparse vector multiply (SpMSpV)

Semiring Option: (multiply,add)
(select2nd, min)



Bipartite graph
G(R,C,E)





In each row, retain
the minimum product
from the selected columns

A matching

Graph Operation
Traverse
unvisited neighbors

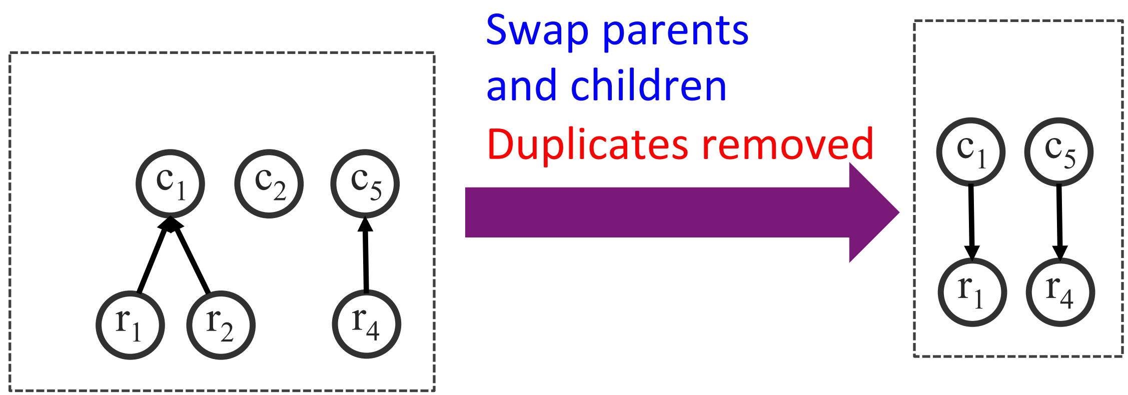
Matrix Operation
SpMSpV

Two required primitives

2. Inverted index in a sparse vector

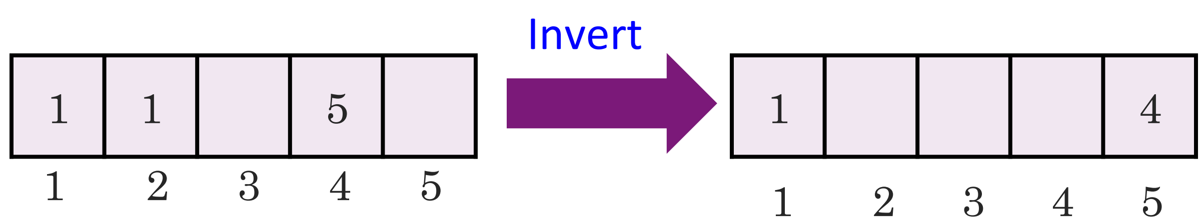
Graph Operation

1. Keep unique child
2. Swap matched and unmatched edges



Vector Operation

Inverted index in a sparse vector

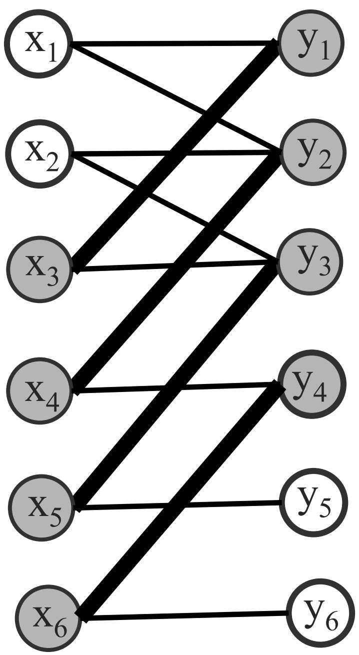


Index: child
Value: parent

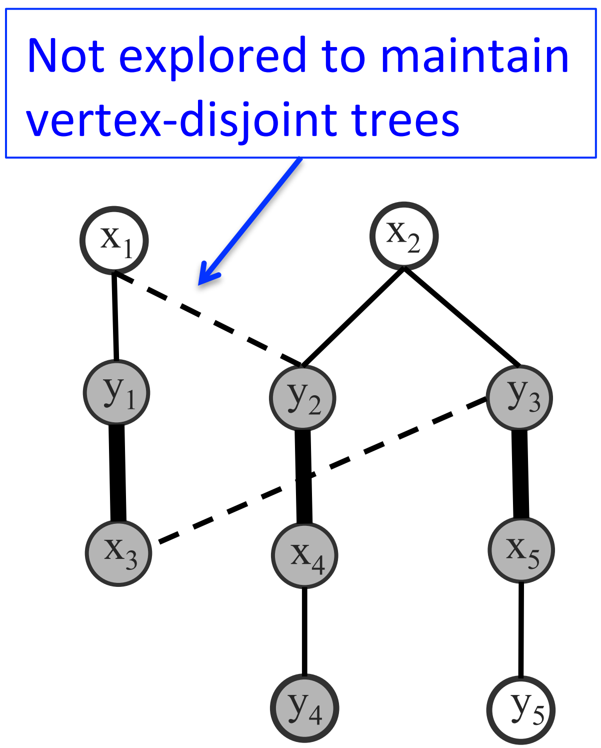
Index: parent
Value: child

Multi-source BFS (MS-BFS) algorithm using matrix and vector operations

Step-1: Discover vertex-disjoint augmenting paths



(a) A maximal matching in a Bipartite Graph



(b) Alternating BFS Forest

Roots of BFS trees

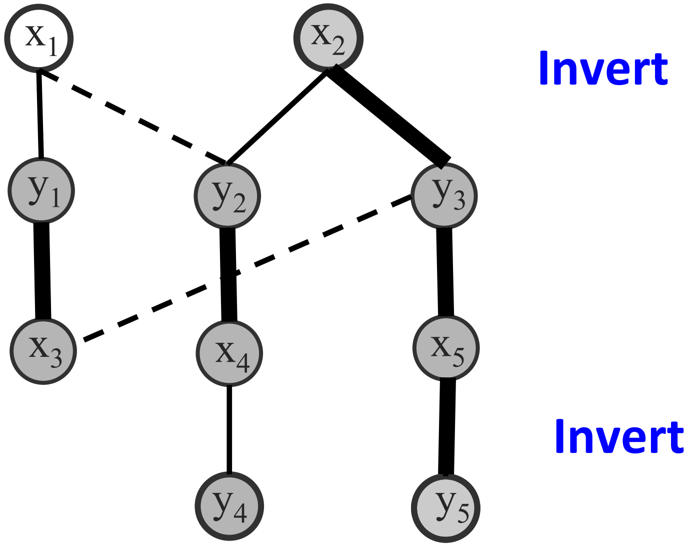
Sparse matrix-sparse vector multiply (SpMSpV)

Inverted index using matching vector

Sparse matrix-sparse vector multiply (SpMSpV)

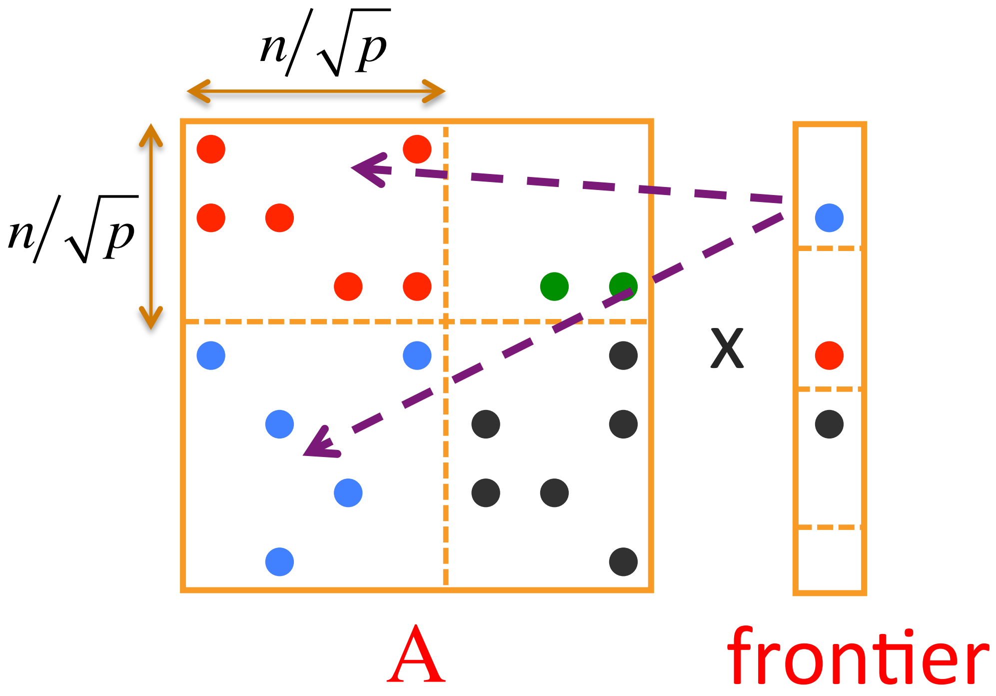
MS-BFS algorithm
using matrix and vector operations

Step-2: Augment matching by flipping matched and unmatched edges along the augmenting paths



Augment matching

Distributed memory parallelization (SpMSpV)

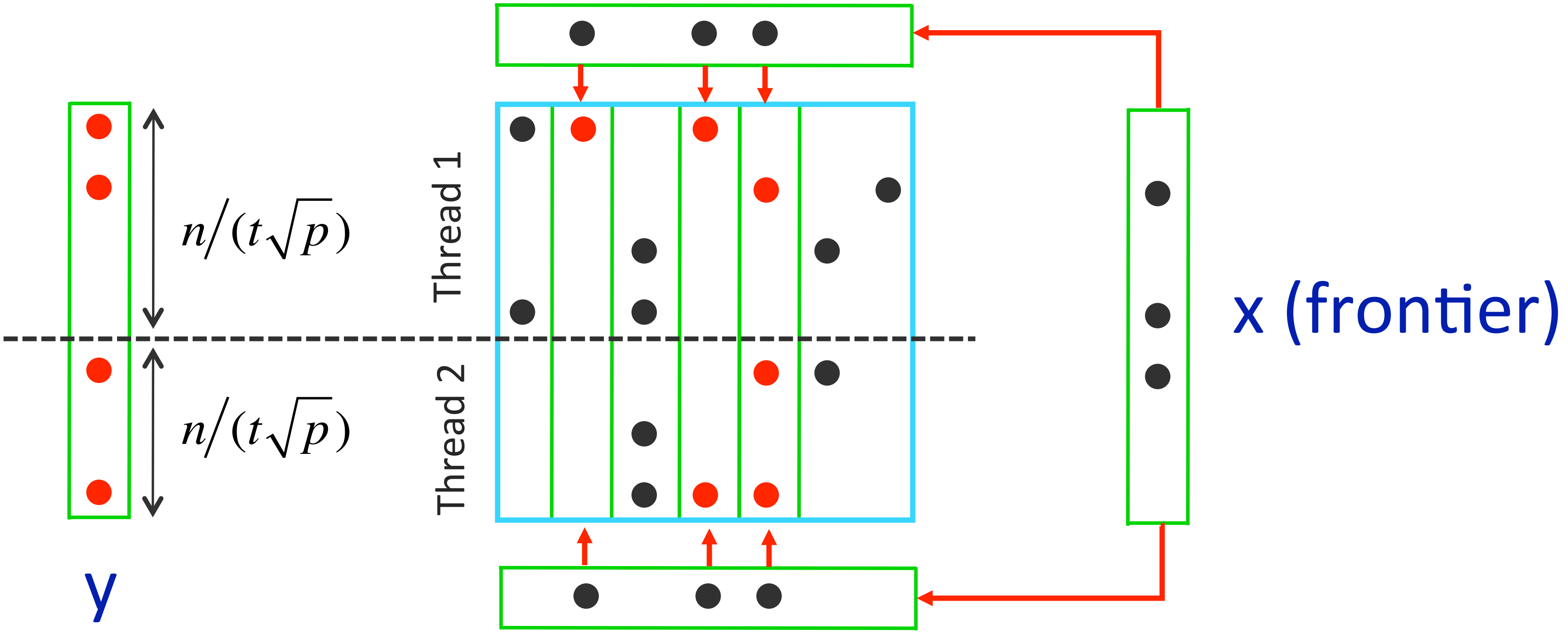


P processors are arranged in $\sqrt{p} \times \sqrt{p}$ Processor grid

ALGORITHM:

1. Gather vertices in *processor column* **[communication]**
2. Local multiplication [computation]
3. Find owners of the current frontier's adjacency and exchange adjacencies in *processor row* **[communication]**

Shared-memory parallelization (SpMSpV)



- Explicitly split local submatrices to t (#threads) pieces along the rows.

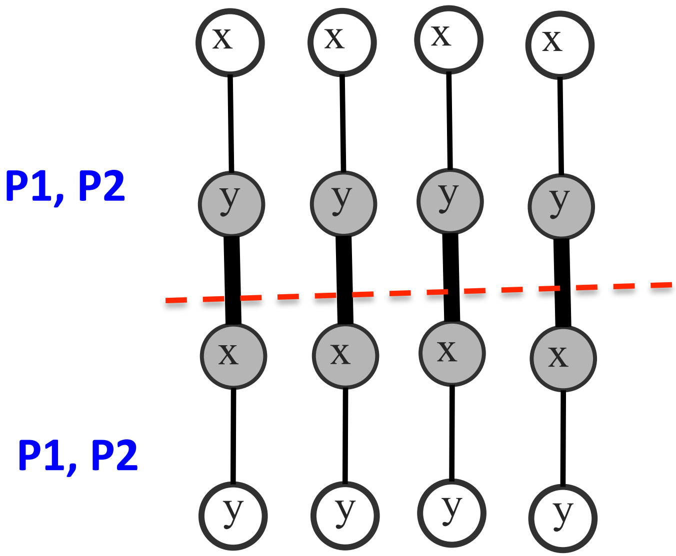
Computation and communication time of discovering vertex-disjoint augmenting paths (a phase)

Operation	Per processor **Computation** (lower bound)	Per processor **Comm (latency)**	Per processor **Comm (bandwidth)**
SpMSpV	$\frac{m}{p}$	$height * \alpha\sqrt{p}$	$\beta\left(\frac{m}{p} + \frac{n}{\sqrt{p}}\right)$
Invert	$\frac{n}{p}$	$height * \alpha p$	$\beta\frac{n}{p}$

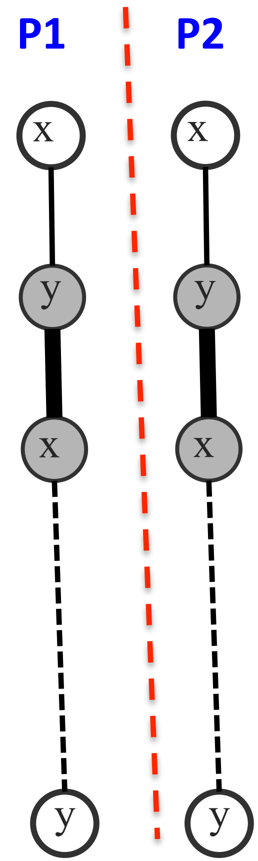
n: number of vertices, m: number of edges
height: maximum height of the BFS forest

α : latency (0.25 μs to 3.7 μs MPI latency on Edison)
β : inverse bandwidth (~8GB/sec MPI bandwidth on Edison)
p : number of processors

Special treatments for long augmenting paths



Level synchronous:
BFS Style



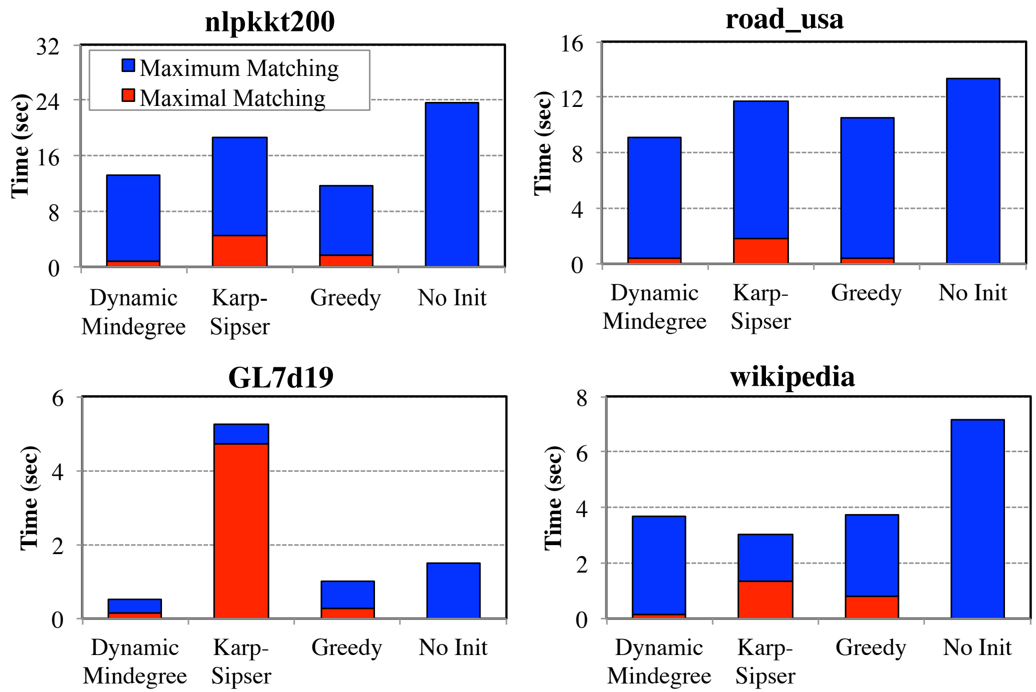
One path per process
Using one-sided communication
via MPI Remote Memory Access (RMA)

Results: experimental Setup

- Platform: Edison (NERSC)
  - 2.4 GHz Intel Ivy Bridge processor, 24 cores (2 sockets) and 64 GB RAM per node
  - Cray Aries network using a Dragonfly topology (0.25 µs to 3.7 µs MPI latency, ~8GB/sec MPI bandwidth)
  - Programming environment: C++ and Cray MPI, Combinatorial BLAS library (Buluc and Gilbert, 2011)

- Input graphs
  - Real matrices from Florida sparse matrix collection and randomly generated matrices.
  - Matrix- bipartite graph conversion
    - rows: vertices in one part, columns: vertices in another part, nonzeros: edges.

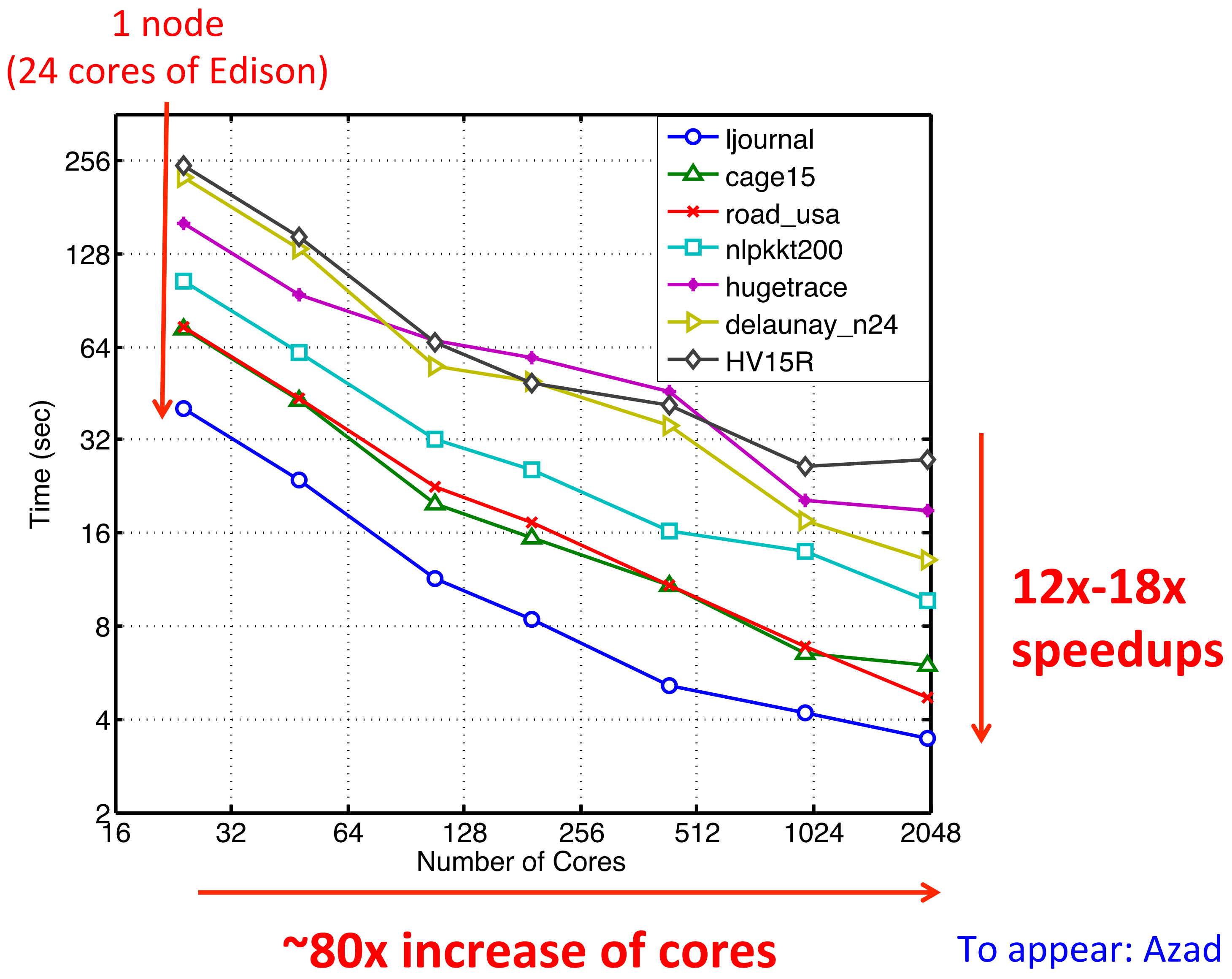
Impact of initialization on MCM



On 1024 cores of Edison

- Karp-Sipser obtains the highest cardinality for many practical problems, but it runs the slowest on high concurrency
- We found that dynamic mindegree + MCM often runs the fastest on high concurrency.

MCM strong scaling (real matrices)

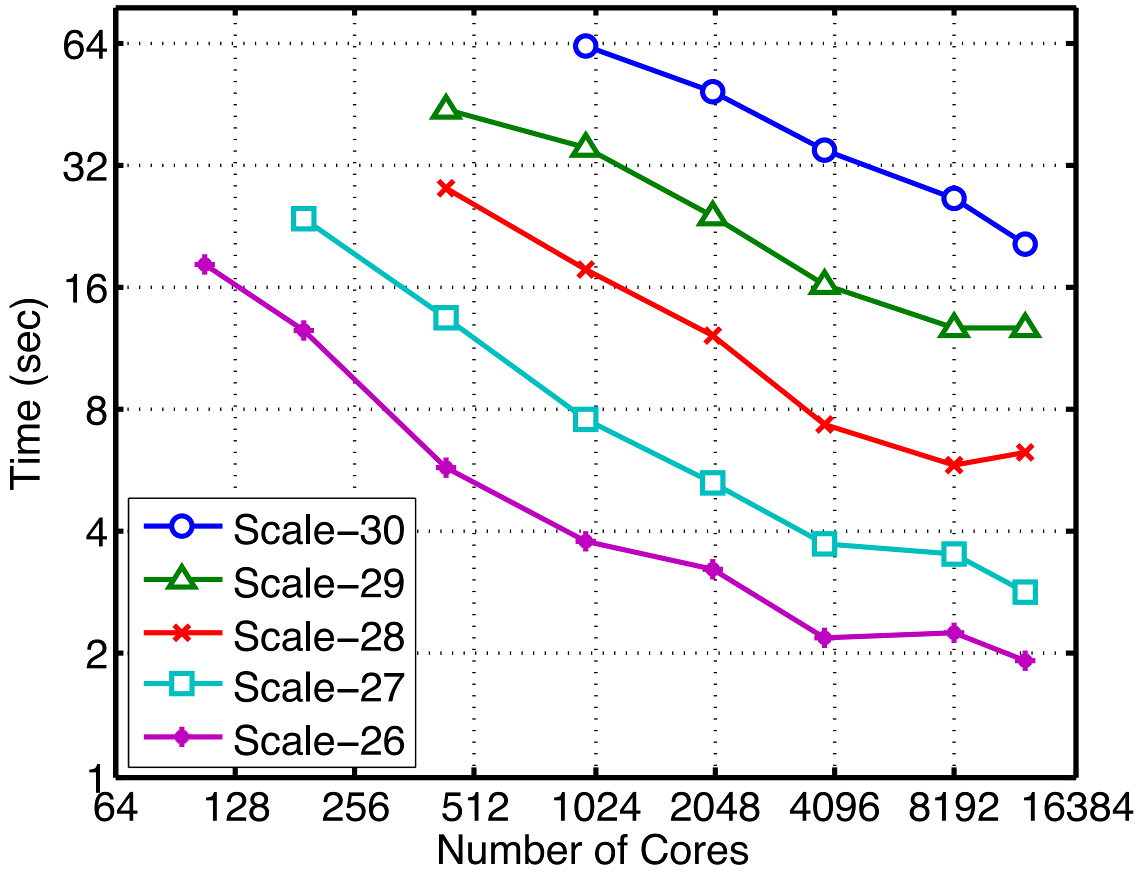


To appear: Azad and Buluç, IPDPS 2016

MCM strong scaling (G500 RMAT matrices)

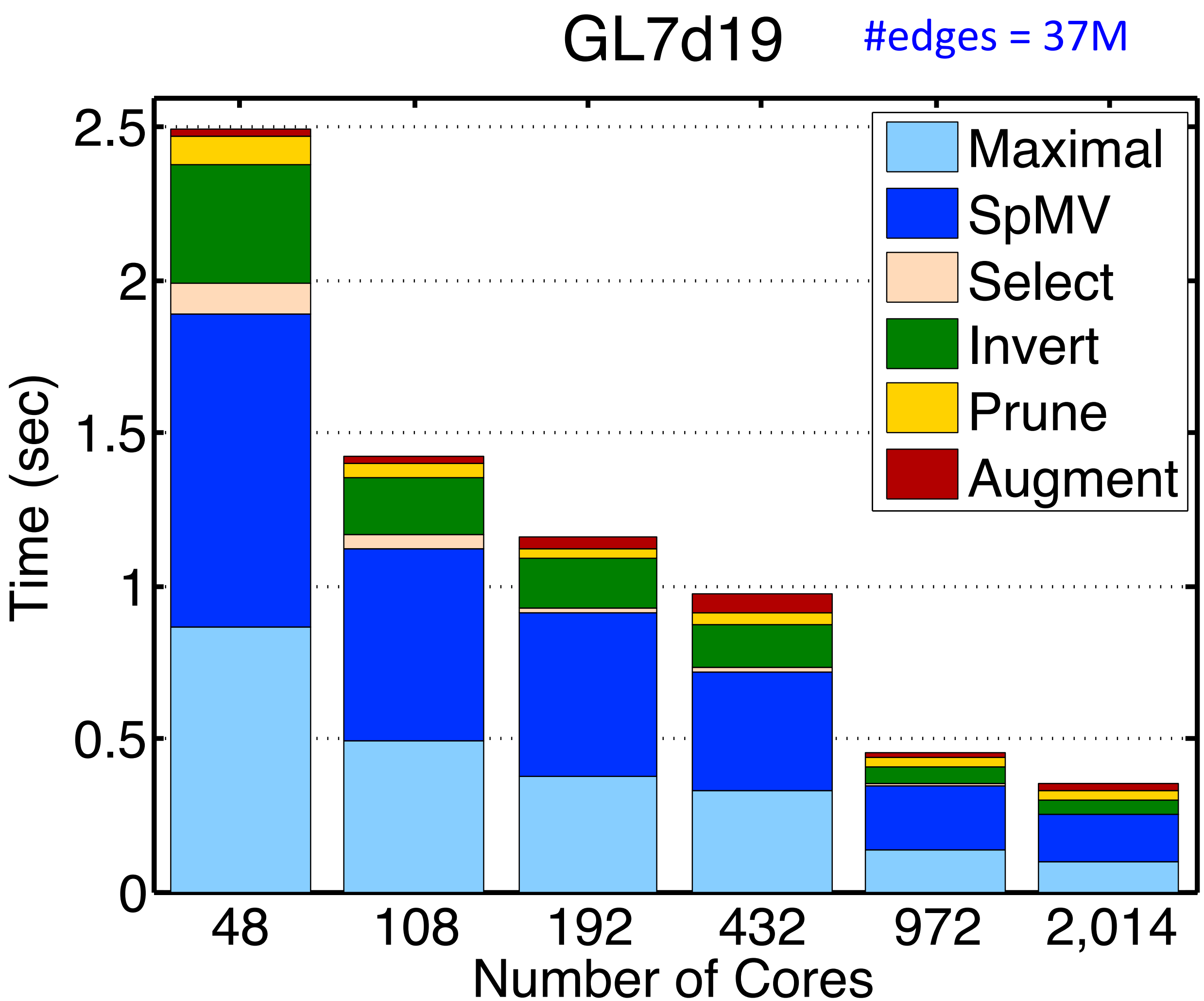
Scale-30 RMAT: 2 billion vertices, 32 billion edges

Scaling continues beyond 10K core on Large matrices



MCM: Breakdown of runtime

V1= 1.91M,
V2 = 1.96M,
#edges = 37M



Ideas for weighted matching

- Similar graph-matrix transformation applies to weighted matching algorithms.

- Auction algorithm ideas [Ongoing work]
  - Bidders bid for most profitable objects: SpMSpV with (select2nd, max) semiring
  - An object selects the best bidder from which it received bid: Inverted index
  - Dual updates can be done using vector operations

Summary

- Summary of contributions
  - Methods: distributed memory matching algorithms based on matrix algebra
  - Performance: scales up to 10K cores on large graphs.
  - Easy to implement an algorithm using matrix-algebraic primitives.
  - Source code publicly available at:

    http://gauss.cs.ucsb.edu/~aydin/CombBLAS/html/

- Future work
  - Distributed weighted matching using matrix algebra

Relevant references

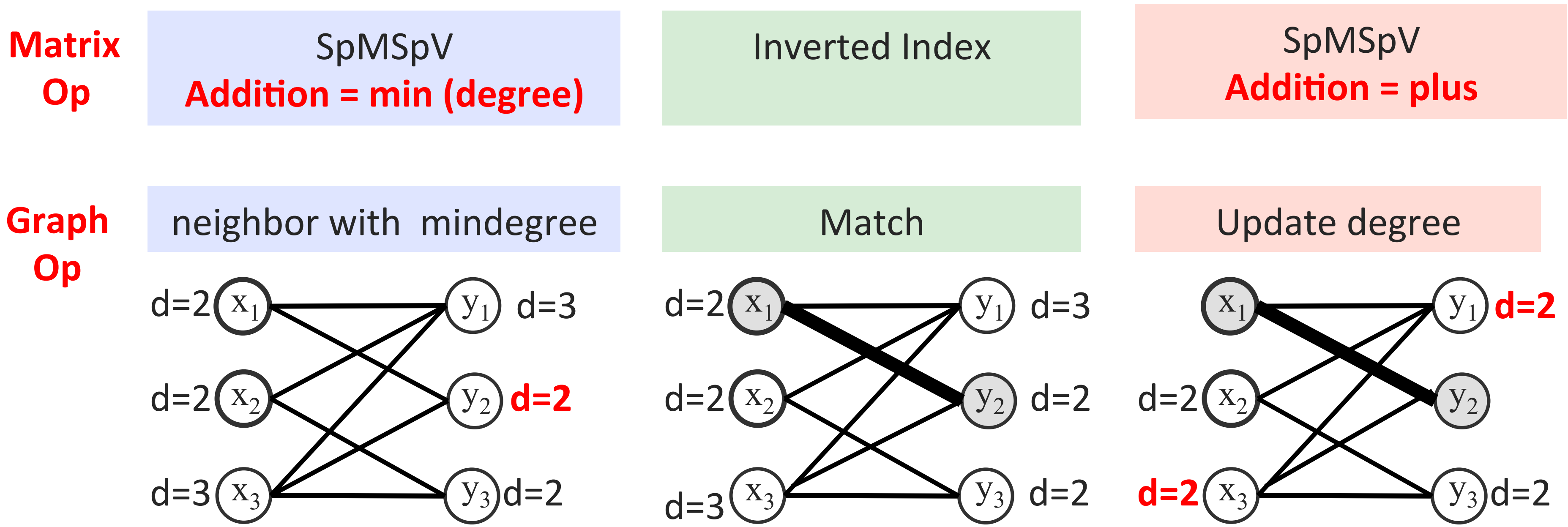
- A. Azad and A. Buluç, to appear IPDPS 2016, Distributed-Memory Algorithms for Maximum Cardinality Matching in Bipartite Graphs.
- A. Azad and A. Buluç, CLUSTER 2015, Distributed-memory algorithms for maximal cardinality matching using matrix algebra.
- Langguth *et al.*, Parallel Computing 2011, Parallel algorithms for bipartite matching problems on distributed memory computers.
- M. Patwary, R. Bisseling, F. Manne, HPPA 2010, Parallel greedy graph matching using an edge partitioning approach.
- M. Sathe, O. Schenk, H. Burkhart, Parallel Computing 2012, An auction-based weighted matching implementation on massively parallel architectures.

Thanks for your attention

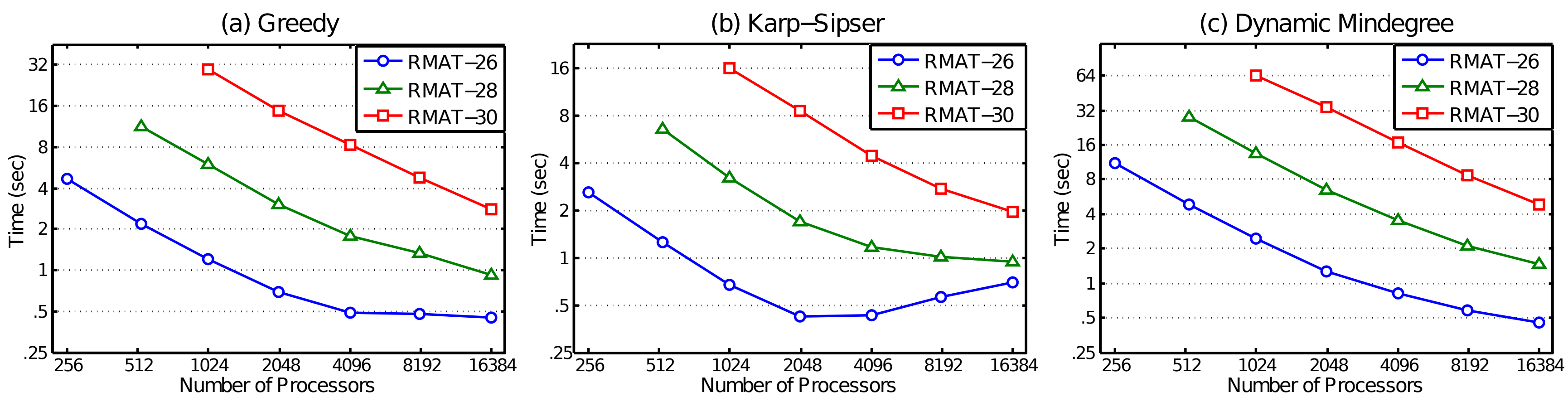
Supporting slides

Maximal matching algorithms using matrix and vector operations

- Used to initialize MCM
- Example: dynamic mindegree algorithm
  - Greedy and Karp-Sipser are similar (Azad and Buluc, 2015)



Maximal matching strong Scaling
Randomly generated RMAT graphs



For 16x increase of cores: 1,024 – 16,384

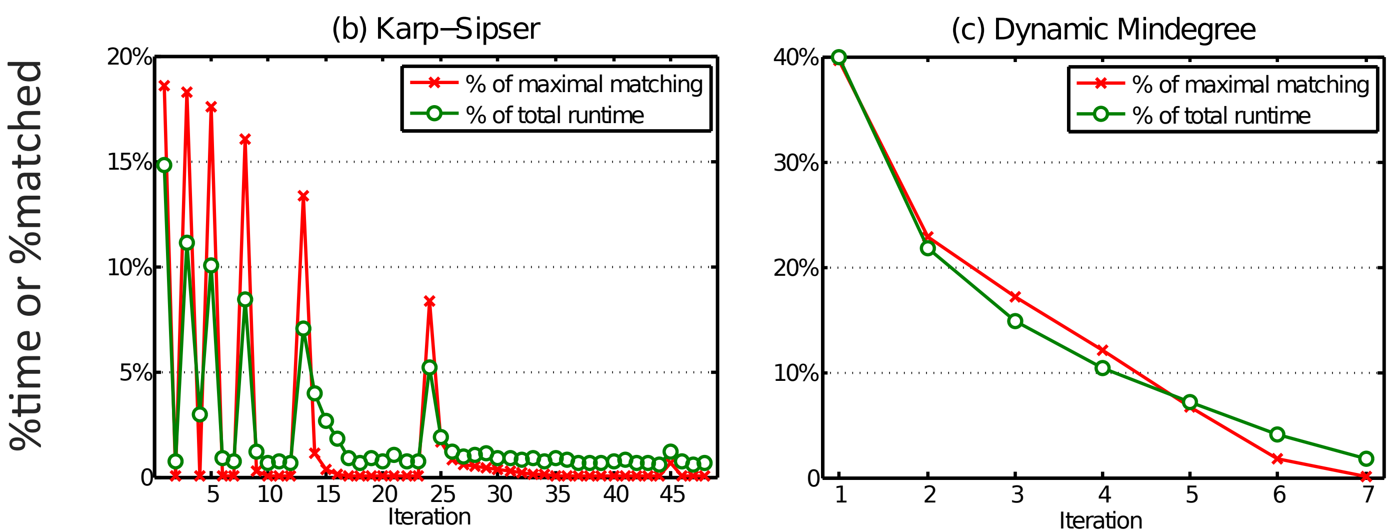
Graph	#vertices	#edges	Greedy	Karp-Sipser	Dynamic Mindegree
RMAT-26	128 million	2 billion	3x	no	**6x**
RMAT-28	512 million	8 billion	7x	3x	**10x**
RMAT-30	2 billion	32 billion	12x	8x	**15x**

Larger graphs
Higher speedups

Strong Scaling
Why does dynamic mindegree scale better?

For 16x increase of cores: 1,024 – 16,384

Graph	#vertices	#edges	Greedy	Karp-Sipser	Dynamic Mindegree
RMAT-26	128 million	2 billion	3x	0x	**6x**
RMAT-28	512 million	8 billion	7x	3x	**10x**
RMAT-30	2 billion	32 billion	12x	8x	**15x**



Graph-based vs. Matrix-based parallel algorithms

- For graph-based algorithms, matching quality decreases with increased concurrency