Parallel Algorithms across the GraphBLAS Stack

Ariful Azad and **Aydın Buluç**
Computational Research Division
Lawrence Berkeley National Laboratory (LBNL)

June 28, 2017
ACS HPC and Data Analytics Workshop
Baltimore, MD

Outline of the talk

Part 1: GraphBLAS and Talk Overview

Part 2: Reverse Cuthill-McKee (RCM) Graph Ordering in Distributed-Memory using GraphBLAS

Part 3: Work-Efficient Parallel Sparse Matrix-Sparse Vector Multiplication (SpMSpV) in Shared-Memory

The GraphBLAS Effort

Standards for Graph Algorithm Primitives

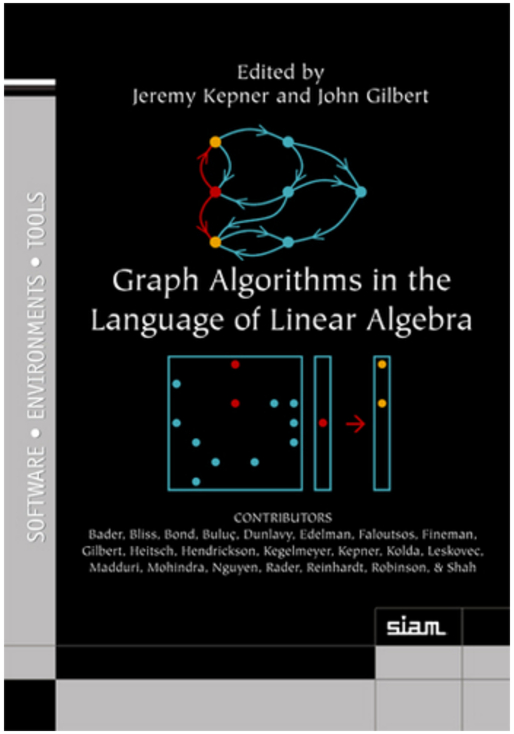
Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

Abstract-- It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

- The GraphBLAS Forum: http://graphblas.org
- IEEE Workshop on Graph Algorithms Building Blocks (at IPDPS): http://www.graphanalysis.org/workshop2017.html

Fast-forward in history

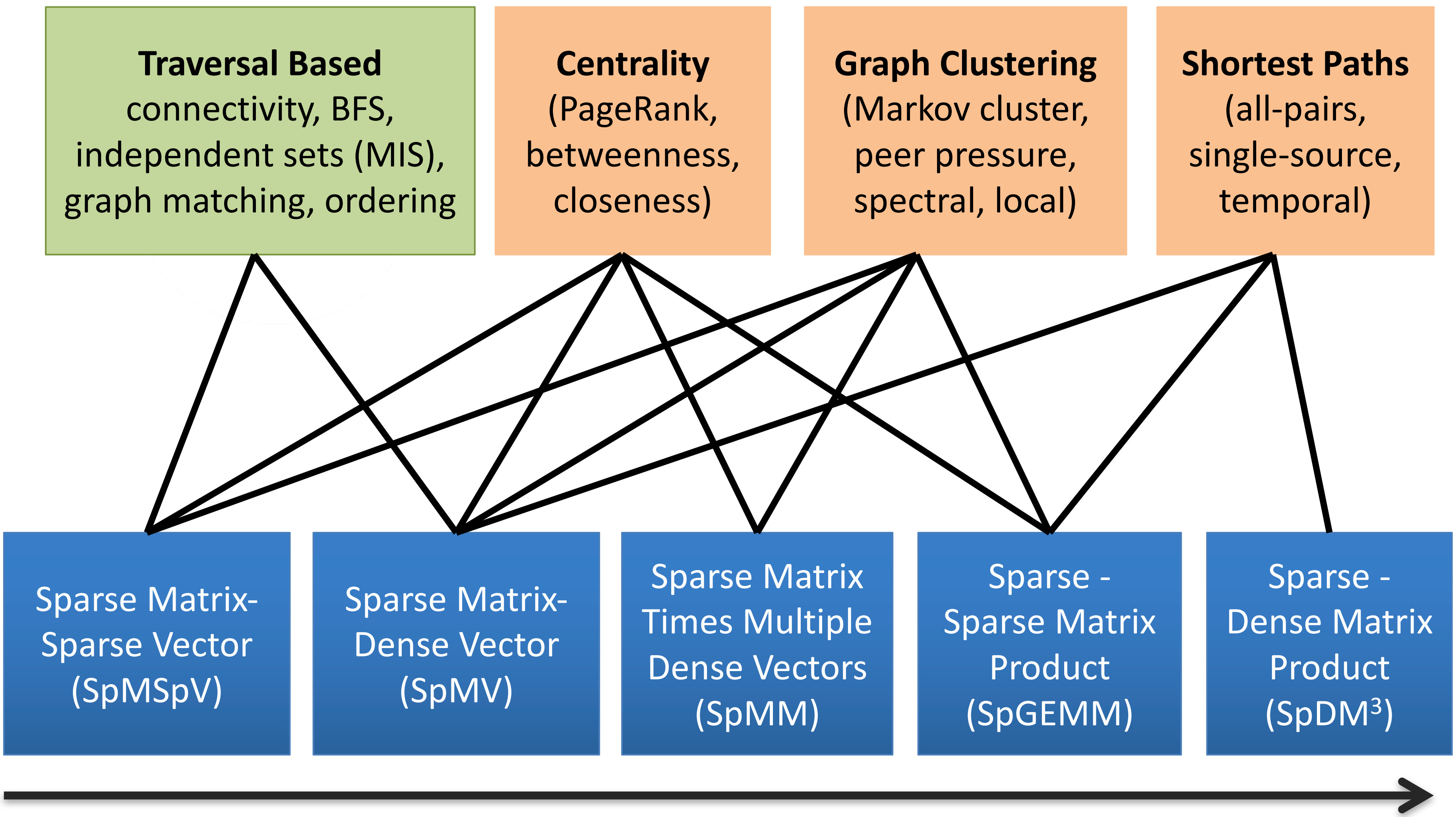
- The idea is older than this SIAM book:
- Several platforms implemented the ideas in the past, such as Star*P
- Current list of active implementations (and version 1.0 of the draft proposal) is available at http://graphblas.org



A. Buluc, T. Mattson, S. McMillan, J. Moreira, C. Yang. "Design of the GraphBLAS API for C", GABB'17

Today, I will talk about a **graph ordering algorithm (RCM) in GraphBLAS** and a **work-efficient shared-memory algorithm for sparse matrix-sparse vector (SpMSpV)** operation in GraphBLAS

The GraphBLAS Stack



GraphBLAS primitives in increasing arithmetic intensity

GraphBLAS C API Spec (http://graphblas.org)

- **Goal:** A crucial piece of the GraphBLAS effort is to translate the mathematical specification to an actual Application Programming Interface (API) that
  i. is faithful to the mathematics as much as possible, and
  ii. enables efficient implementations on modern hardware.
- **Impact:** All graph and machine learning algorithms that can be expressed in the language of linear algebra
- **Innovation:** Function signatures (e.g. mxm, vxm, assign, extract), parallelism constructs (blocking v. non-blocking), fundamental objects (masks, matrices, vectors, descriptors), a hierarchy of algebras (functions, monoids, and semiring)

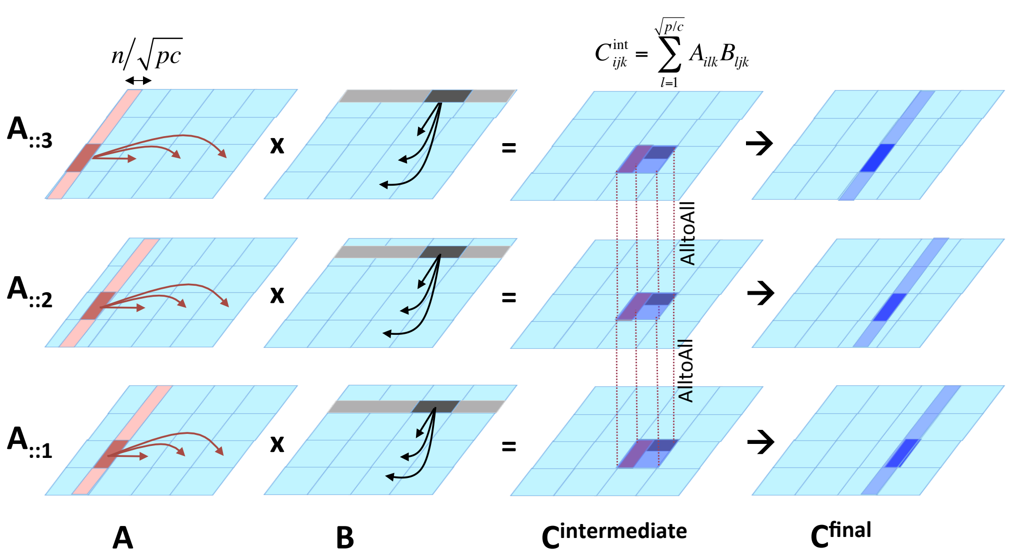
```
GrB_info GrB_mxm(GrB_Matrix                    *C,      // destination
                 const GrB_Matrix               Mask,
                 const GrB_BinaryOp             accum,
                 const GrB_Semiring             op,
                 const GrB_Matrix               A,
                 const GrB_Matrix               B
               [, const Descriptor              desc]);
```

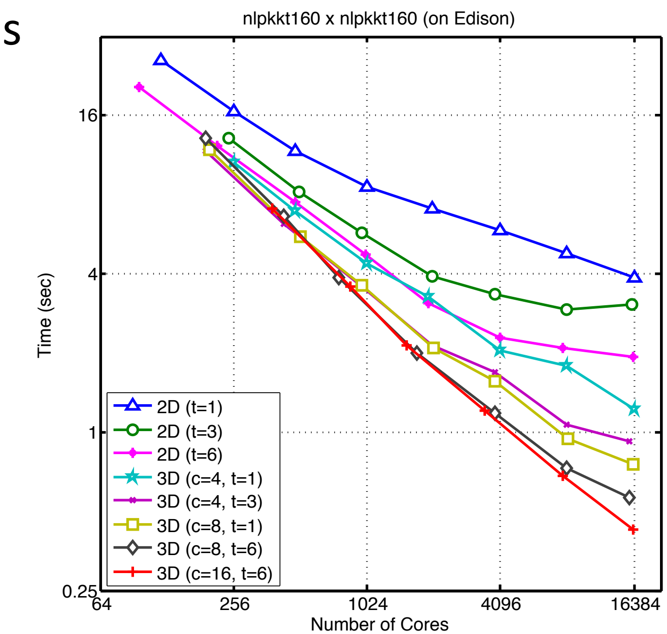
$$C(\neg M) \oplus= A^{T} \oplus.\otimes B^{T}$$

A. Buluç, T. Mattson, S. McMillan, J. Moreira, **C. Yang**. "Proposal for a GraphBLAS C API" (Working document from the GraphBLAS Signatures Subgroup)

Parallel algorithms for sparse-matrix- sparse matrix multiplication (SpGEMM)

- **Goal:** More scalable SpGEMM algorithms in shared and distributed-memory
- **Applications:** Algebraic multigrid (AMG) restriction, graph computations, quantum chemistry, data mining, interior-point optimization
- **Algorithmic innovations:** (1) Novel shared-memory kernel for in-node parallelism, (2) Split-3D-SpGEMM: an efficient implementation of communication-avoiding SpGEMM
- **Performance:** Split-3D-SpGEMM with new shared-memory kernel (red) beats old state-of-the-art (blue) by 8X at large concurrencies

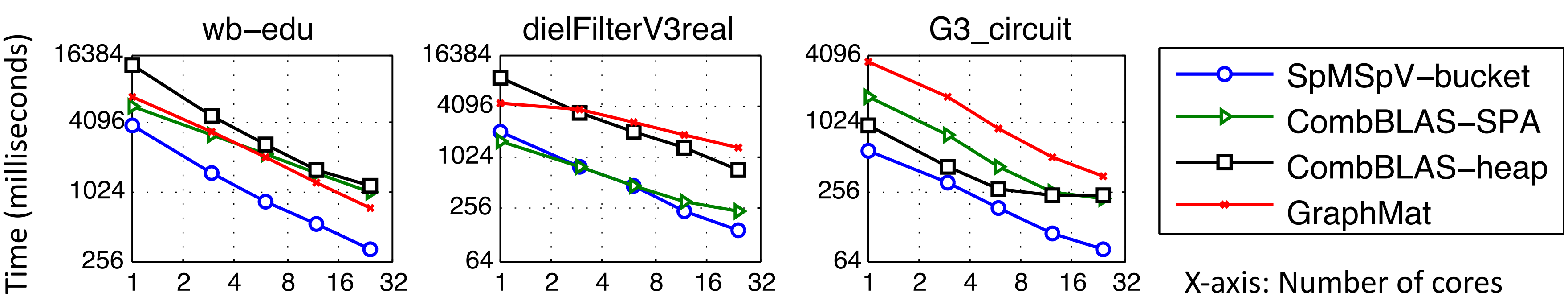




A. Azad, G. Ballard, **A. Buluç**, J. Demmel, L. Grigori, O. Schwartz, S. Toledo, S. Williams. Exploiting multiple levels of parallelism in sparse matrix-matrix multiplication. SIAM Journal of Scientific Computing (SISC), 2016.

An work-efficient parallel algorithm for sparse matrix-sparse vector multiplication (SpMSpV)

- **Goal:** A scalable SpMSpV algorithm without doing more work on higher concurrency
- **Application:** Breadth-first search, graph matching, support vector machines, etc.
- **Algorithmic innovation:**
  - Attains work-efficiency by arranging necessary columns of the matrix into buckets where each bucket is processed by a single thread
  - Avoids synchronization by row-wise partitioning of the matrix on the fly
- **Performance:**
  - First ever work-efficient algorithm for SpMSpV that attains up to 15x speedup on a 24-core Intel Ivy Bridge processor and up to 49x speedup on a 64-core KNL processor
  - Up to an order of magnitude faster than its competitors, especially for sparser vector



X-axis: Number of cores (Intel Ivy Bridge)

A.Azad, **A. Buluç.** A work-efficient parallel sparse matrix-sparse vector multiplication algorithm. IPDPS'17

The Reverse Cuthill-McKee Algorithm in Distributed-Memory

- **Goal:** Find a permutation **P** of a sparse matrix **A** so that the bandwidth of **PAP**^{T} is small.
- **Application:** Faster iterative solvers, e.g., preconditioned conjugate gradients (PCG).
- **Innovation in Parallel RCM Algorithm:**
    1. Step1: level-by-level vertex exploration and ordering. Approach: specialized breadth-first search using sparse matrix-sparse vector multiplication (SpMSpV) over a semiring
    2. Step2: Ordering of vertices in each level by (parents' order, degree) pairs. Approach: parallel partial sorting.
- **Performance:**
    - First ever distributed-memory RCM algorithm that scales up to 4096 cores on NERSC/Edison.
    - Attains up to 38x speedup on 1028 cores.

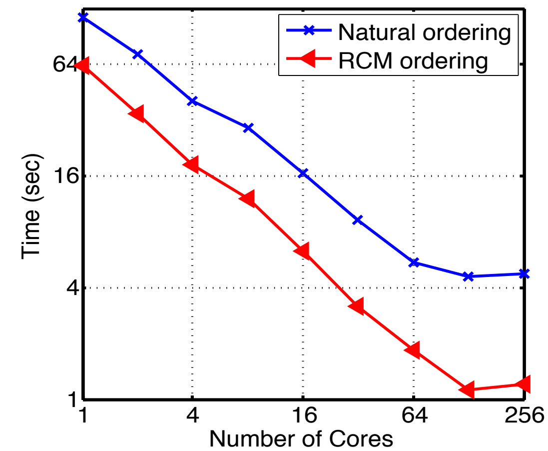


Fig2: Solving PCG in PETSc with/without RCM ordering (on thermal2 matrix)

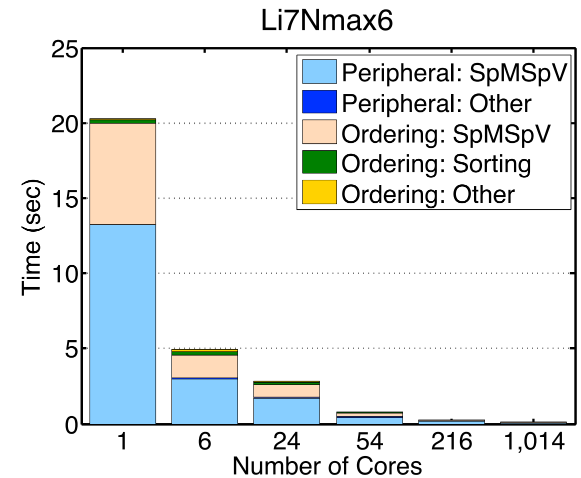


Fig2: Strong scaling of the RCM algorithm on NERSC/Edison

A.Azad, M. Jacquelin, **A. Buluç,** E.Ng. The Reverse Cuthill-McKee Algorithm in Distributed-Memory. IPDPS'17

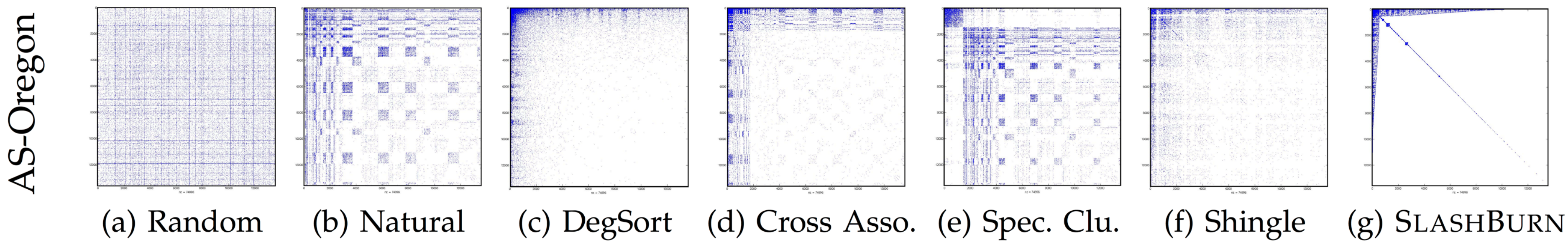
Outline of the talk

Part 1: GraphBLAS and Talk Overview

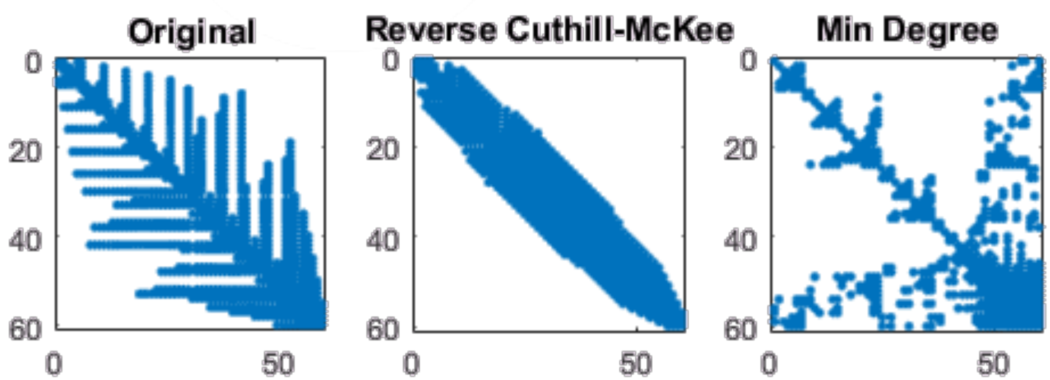
Part 2: Reverse Cuthill-McKee (RCM) Graph Ordering in Distributed-Memory using GraphBLAS

Part 3: Work-Efficient Parallel Sparse Matrix-Sparse Vector Multiplication (SpMSpV) in Shared-Memory

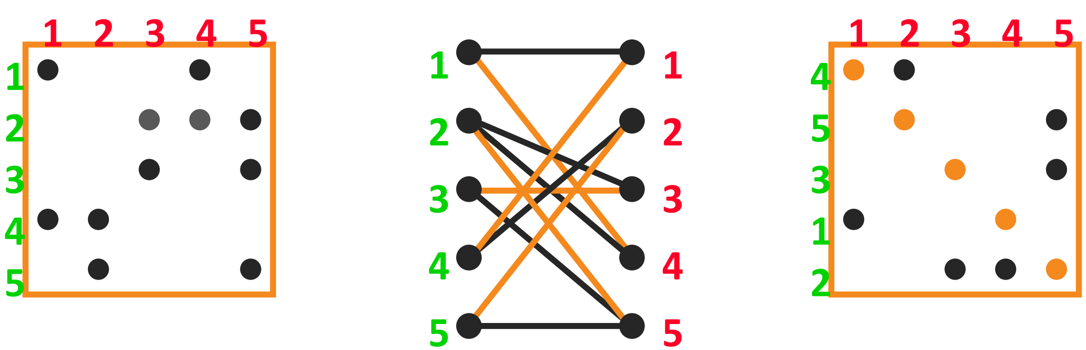
Many ways of ordering a matrix



Lim, Kang, and Faloutsos, TKDE'14



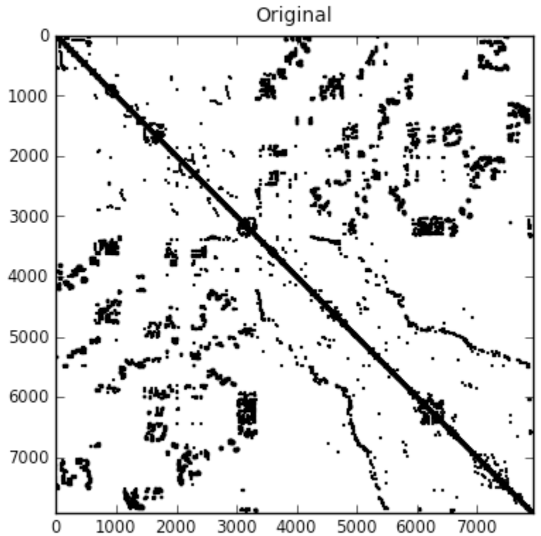
Traditional sparse matrix orderings



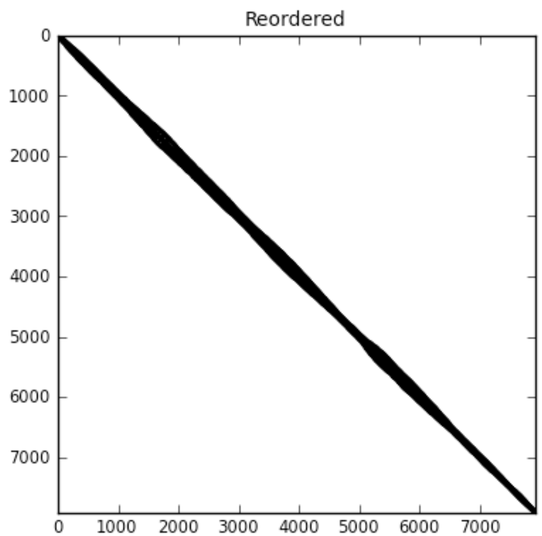
Permuting for heavy diagonal
(bipartite graph matching)

Reordering for reducing bandwidth & profile

- In this talk, we consider parallel algorithms for **reordering** sparse matrices
- **Goal:** Find a permutation P so that the bandwidth/profile of PAP^T is small.



Before permutation

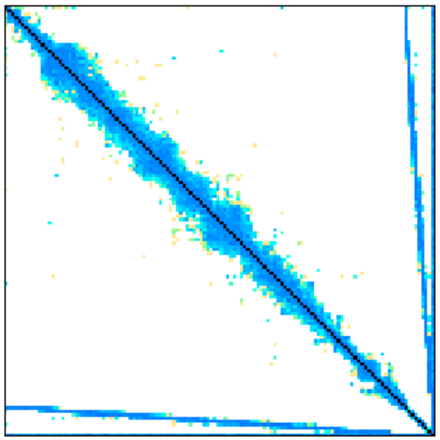


After permutation

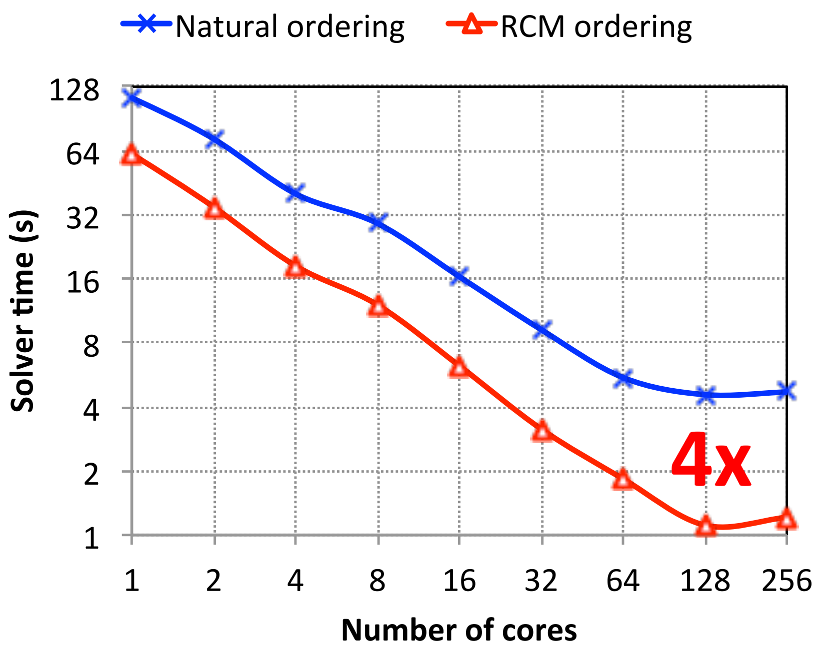
Why reordering a matrix

- ❑ Better cache reuse in SpMV [Karantasis et al. SC '14]
- ❑ Faster iterative solvers such as preconditioned conjugate gradients (PCG).

Example: PCG implementation in PETSc

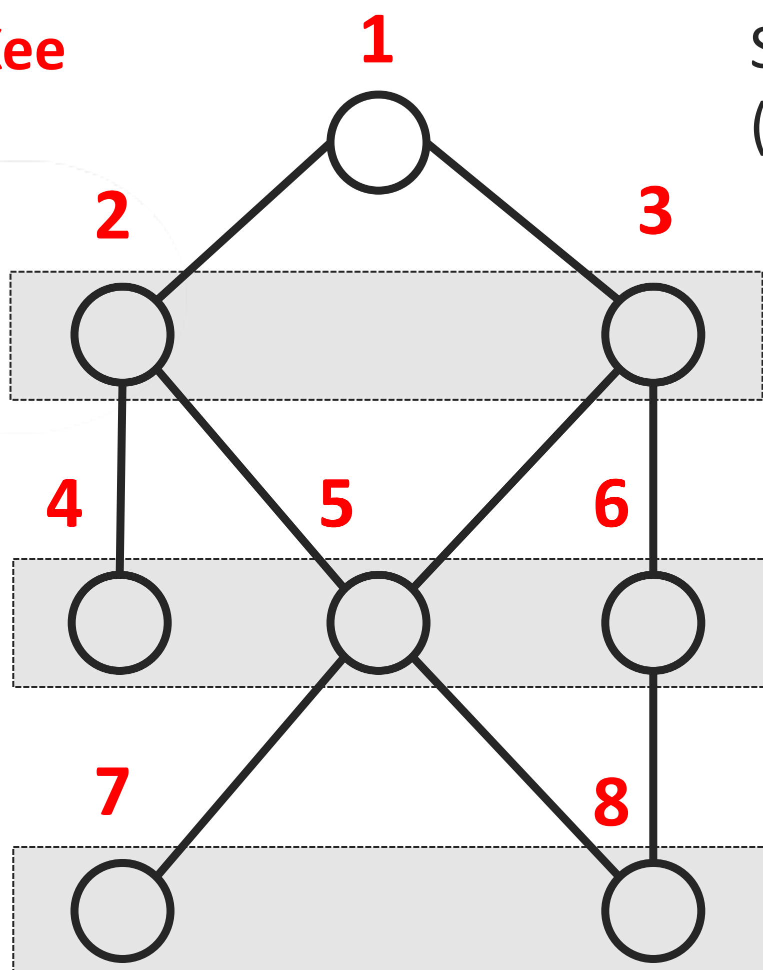


Thermal2 (n=1.2M, nnz=4.9M)



The RCM algorithm

Cuthill-McKee order



Start vertex
(a pseudo-peripheral vertex)

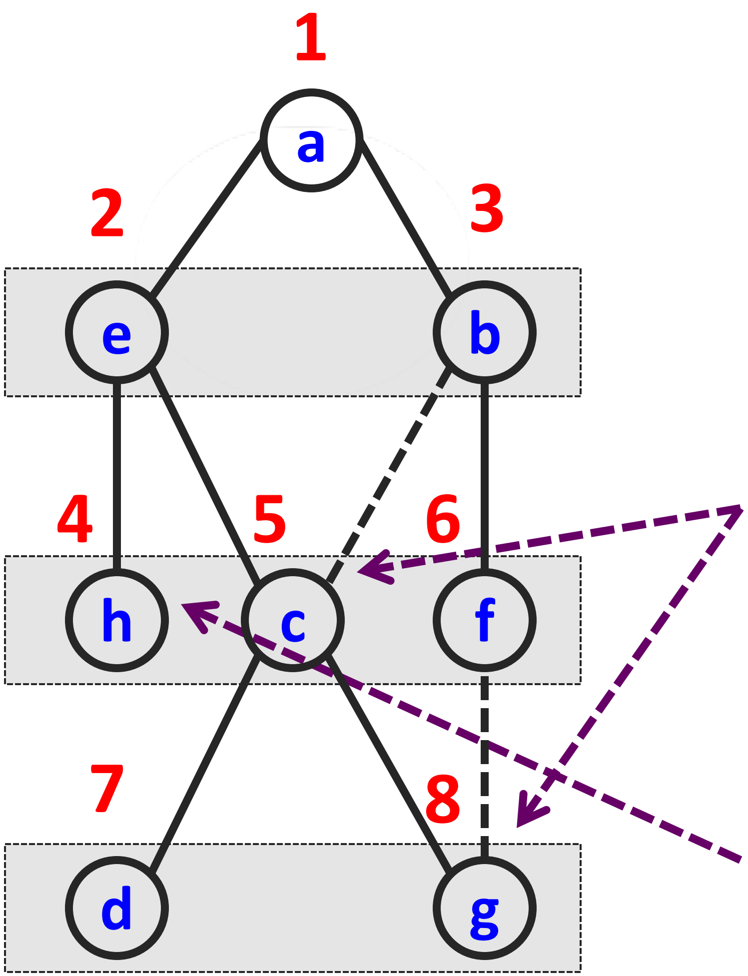
Order vertices by increasing degree

Order vertices by (parents' order, degree)

Order vertices by parents' order

Reverse the order of vertices to obtain the RCM ordering

RCM: Challenges in parallelization (in addition to parallelizing BFS)



- Given a start vertex, the algorithm gives a fixed ordering except for tie breaks. **Not parallelization friendly**.
- **Unlike traditional BFS**, the parent of a vertex is set to a vertex with the minimum label. (i.e., bottom-up BFS is not beneficial)
- Within a level, vertices are labeled by **lexicographical order of (parents' order, degree) pairs,** needs sorting

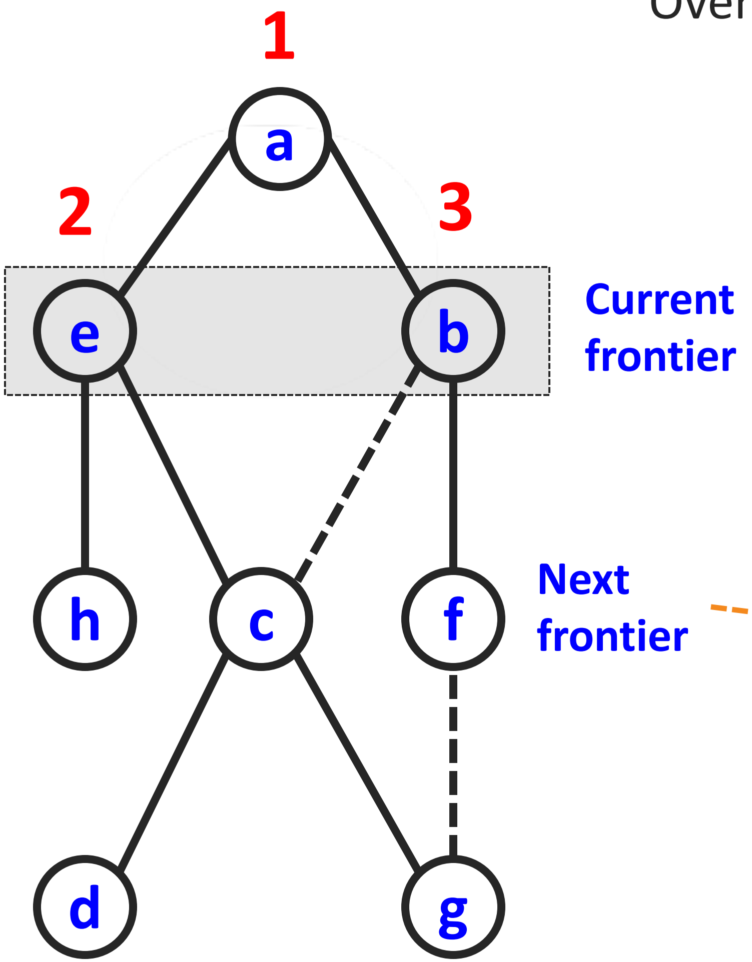
Our approach to address parallelization challenges

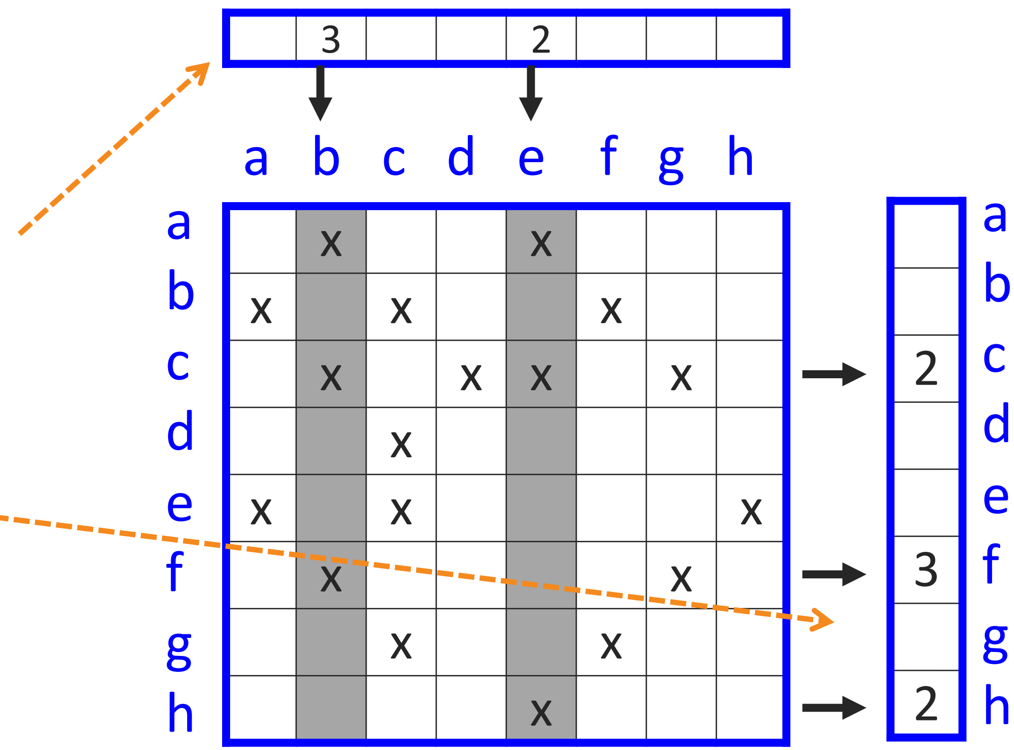
- We use **specialized** level-synchronous BFS
- Key differences from traditional BFS (Buluç and Madduri, SC '11)
  1. A parent with smaller label is preferred over another vertex with larger label
  2. The labels of parents are passed to their children
  3. Lexicographical sorting of vertices in BFS levels

- The first two of them are addressed by sparse matrix-sparse vector multiplication (SpMSpV) over a semiring
- The third challenge is addressed by a lightweight sorting function

Exploring the next-level vertices via SpMSpV

Overload (multiply,add) with (select2nd, min)

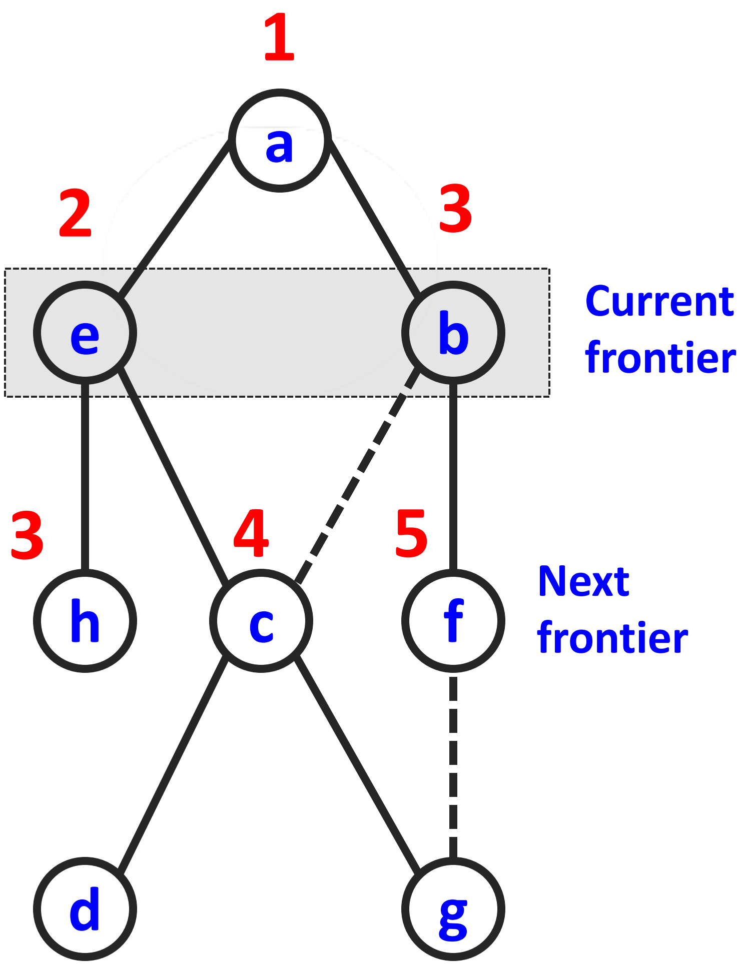




Adjacency matrix

We propagate the parent label information to children during BFS (via semiring multiply)

Ordering vertices via partial sorting



Sort degrees of the siblings
many instances of small sortings
(avoids expensive parallel sorting)

	a	b	c	d	e	f	g	h
Parent's label			2			3		2
My degree			4			2		1

Rules for ordering vertices

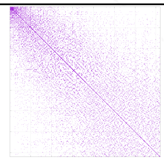
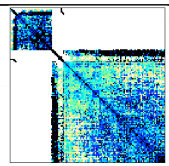
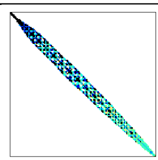
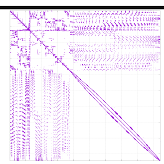
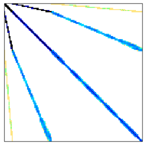
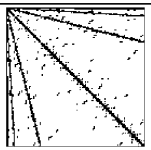
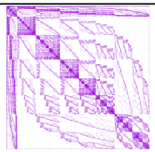
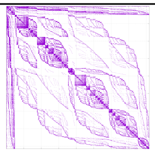
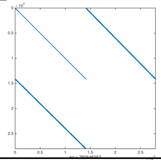
1. c and h are ordered before f
2. h is ordered before c

Other aspects of the algorithm

- Finding a **pseudo peripheral vertex**: Repeated application of the usual BFS (no ordering of vertices within a level).
- This is actually quite expensive
  - Choose a vertex u.
  - Among all the vertices that are as far from u as possible, let v be one with minimal degree.
  - If v is more eccentric than u, then set u=v and repeat previous step, else v is a pseudo-peripheral vertex.

- Our SpMSpV is hybrid OpenMP-MPI implementation
  - Multithreaded SpMSpV is also fairly complicated and subject to the second half of this talk upcoming slides

What did we test on?

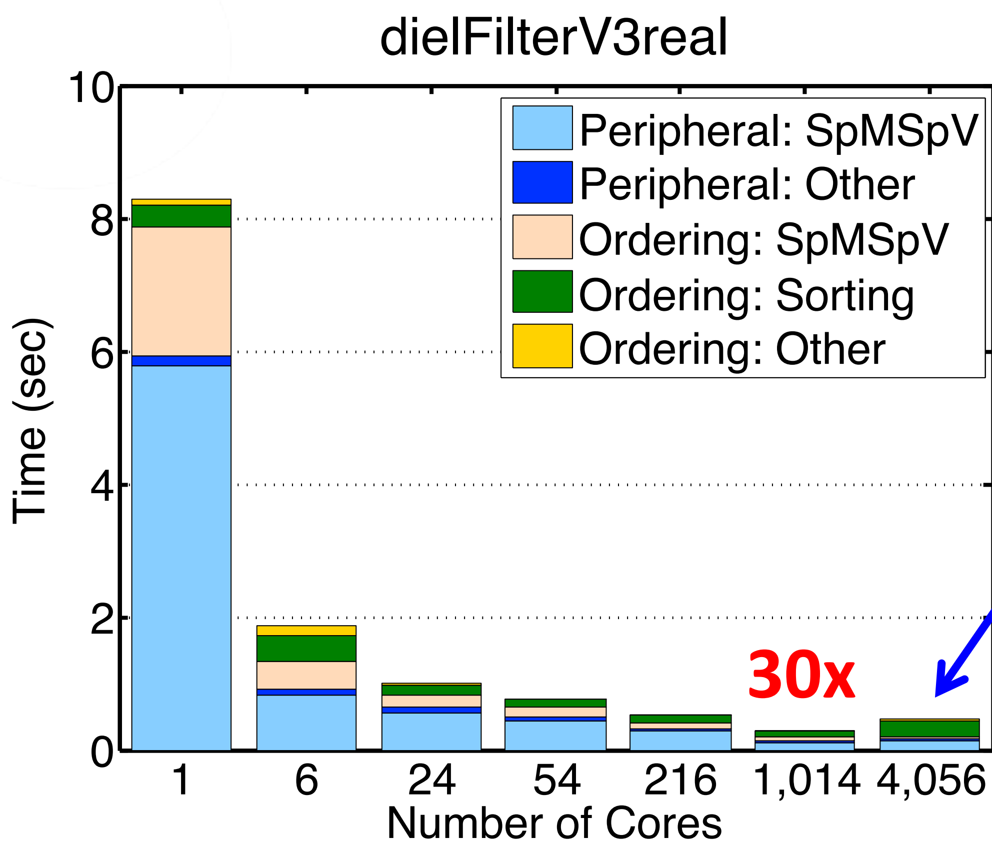
Name Description	Spy Plot	Dimensions Nonzeros	BW (pre-RCM) BW (post-RCM) Pseudo-diameter
nd24k 3D mesh problem		72K×72K 29M	68,114 10,294 14
Ldoor structural prob.		952K×952K 42.49M	686,979 9,259 178
Serena gas reservoir simulation		1.39M×1.39M 64.1M	81,578 81,218 58
audikw_1 structural prob		943K×943K 78M	925,946 35,170 82
dielFilterV3real higher-order finite element		1.1M×1.1M 89.3M	1,036,475 23,813 84
Flan_1565 3D model of a steel flange		1.6M×1.6M 114M	20,702 20,600 199
Li7Nmax6 nuclear configuration interaction calculations		664K×664K 212M	663,498 490,000 7
Nm7 nuclear configuration interaction calculations		4M×4M 437M	4,073,382 3,692,599 5
nlpkkt240 Sym. indefinite KKT matrix		78M×78M 760M	14,169,841 361,755 243

Structural information on the sparse matrices used in our experiments. All matrices, except two, are from the University of Florida sparse matrix collection. Li7Nmax6 and Nm7 [22] are from nuclear configuration interaction calculations.

Results: Scalability on NERSC/Edison (6 threads per MPI process)

#vertices: 1.1M #edges: 89M

Bandwidth before: 1,036,475 after: 23,813

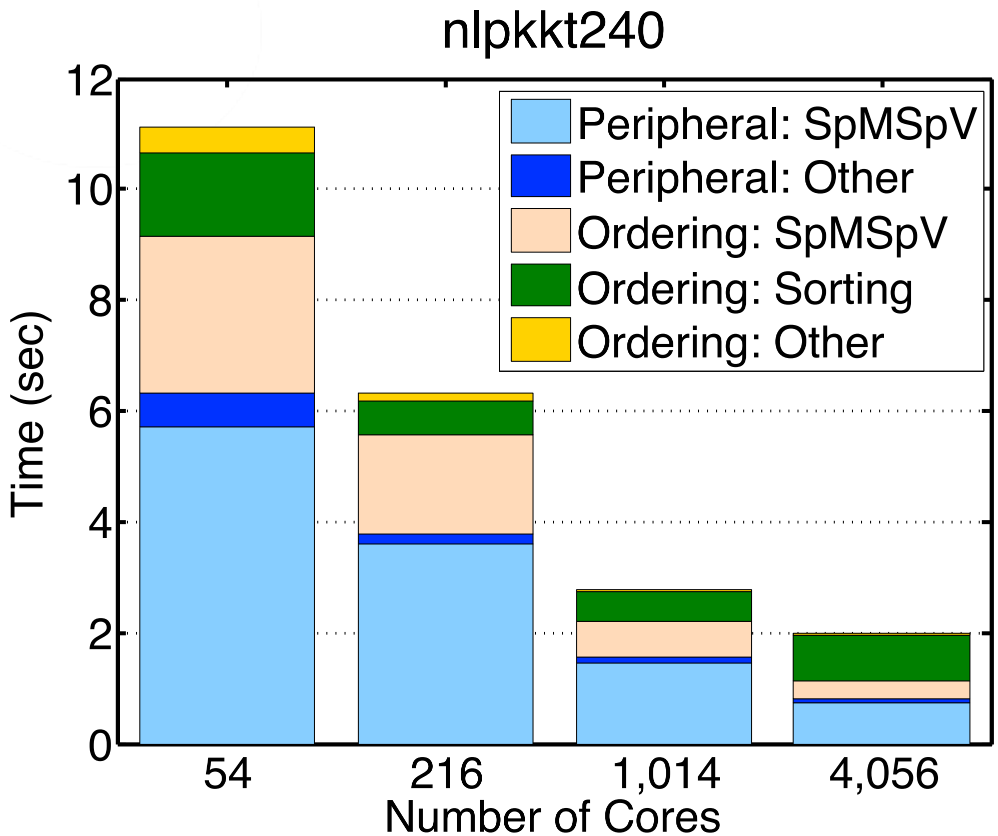


Communication dominates

Scalability on NERSC/Edison (6 threads per MPI process)

#vertices: 78M #edges: 760M

Bandwidth before: 14,169,841 after: 361,755



Larger graphs continue scaling

Why is this significant?

- We managed to scaling RCM in distributed memory relatively easily.
- The community should pad themselves on the back
- Research in BFS (e.g. Graph500) and graph primitives (GraphBLAS) made this possible.
- Flashback to 1995 [Barnard, Pothen, Simon]:

"The spectral envelope-reduction algorithm has several features which set it apart from the **earlier reordering algorithms such as the GPS, GK, or RCM algorithms**. These algorithms employ local-search in the adjacency graph of the matrix. All of them try to find a pseudo-diameter in the graph by generating a long level-structure by breadth first-search beginning from a suitable vertex. **These types of algorithms generally do not vectorize, and there is no obvious way to implement them in parallel**."

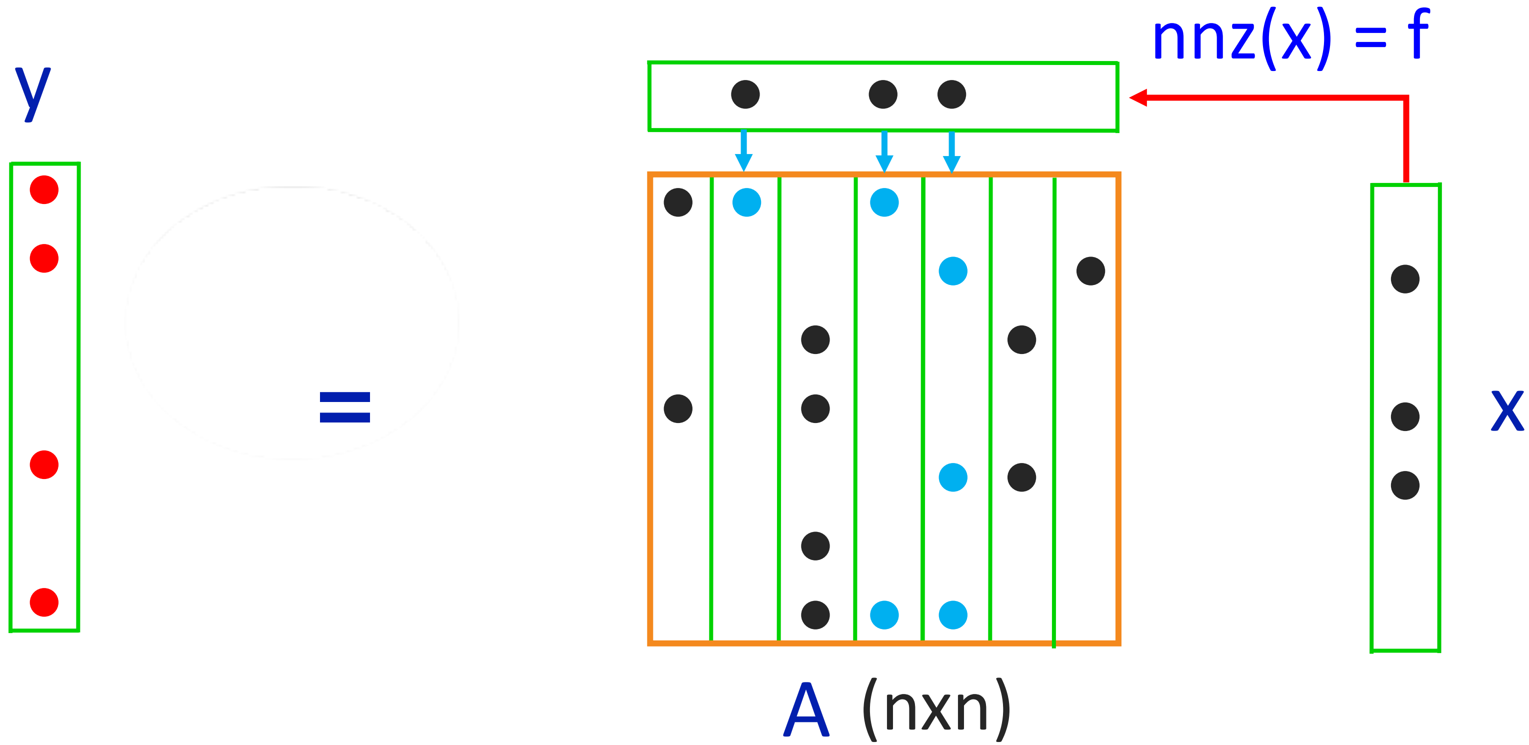
Outline of the talk

Part 1: GraphBLAS and Talk Overview

Part 2: Reverse Cuthill-McKee (RCM) Graph Ordering in Distributed-Memory using GraphBLAS

Part 3: Work-Efficient Parallel Sparse Matrix-Sparse Vector Multiplication (SpMSpV) in Shared-Memory

A lower bound for SpMSpV

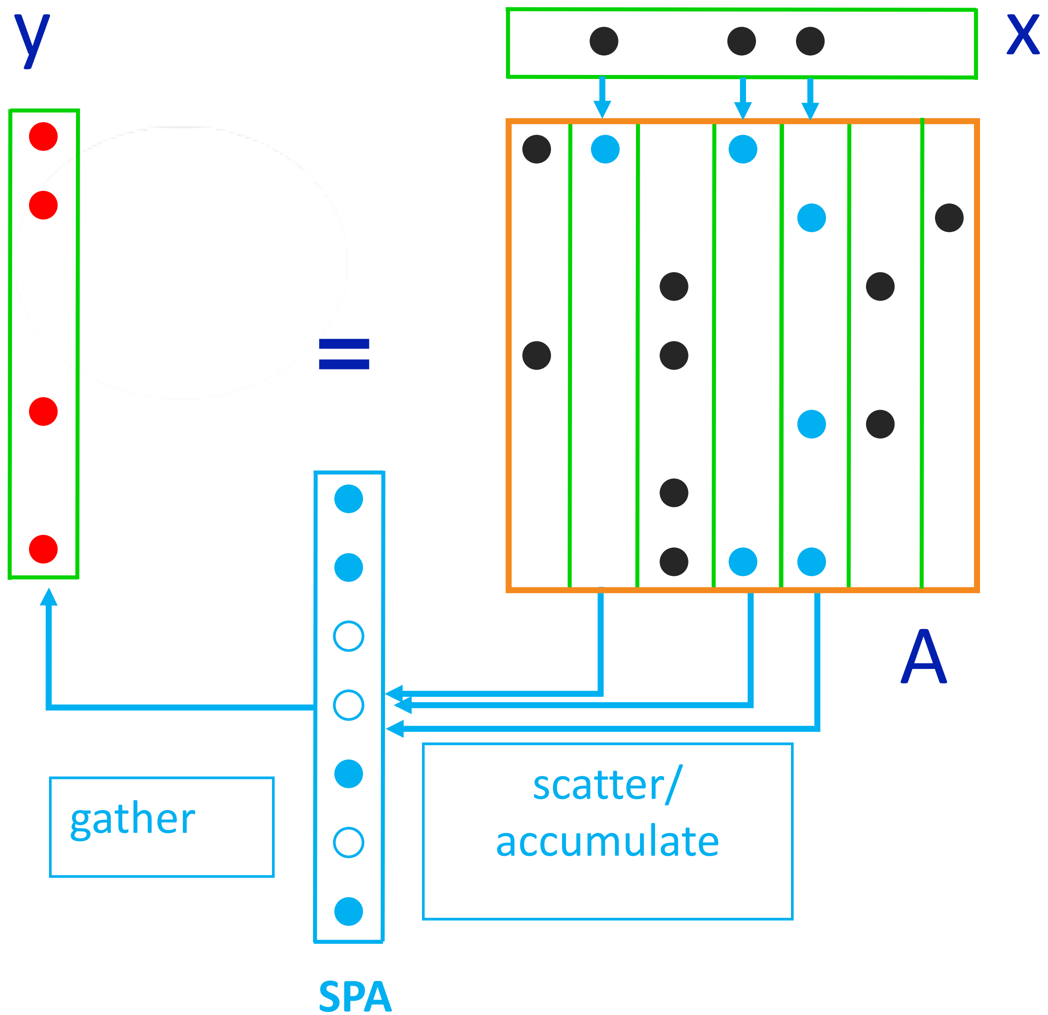


Average nnz in each column of A = d

Considering Erdos-Renyi graph G(n, d/n)

Lower bound of SpMSpV: df (no matrix/vector dimension)

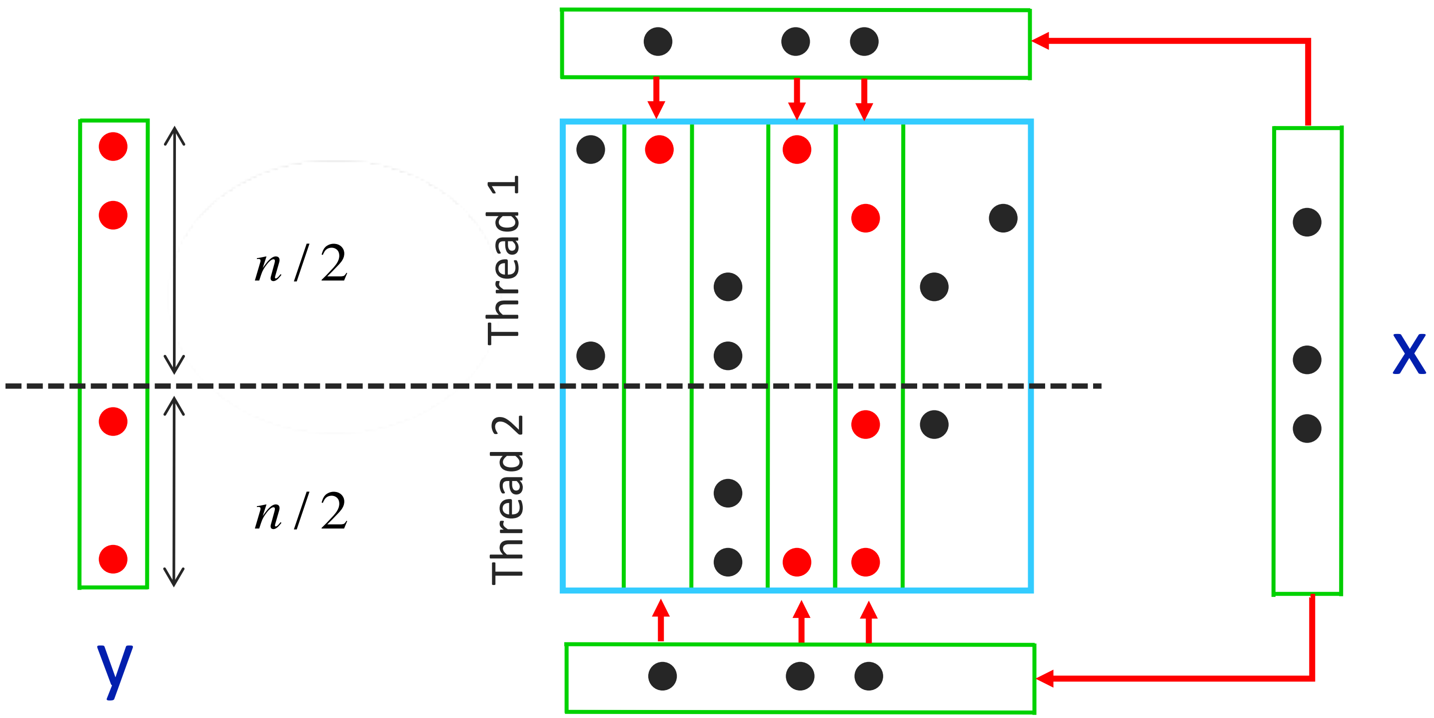
SpMSpV via sparse accumulator (SPA)



Can be done in O(df) time; attains lower bound

We parallelize this algorithm

Shared-memory parallelization of SpMSpV (row split)

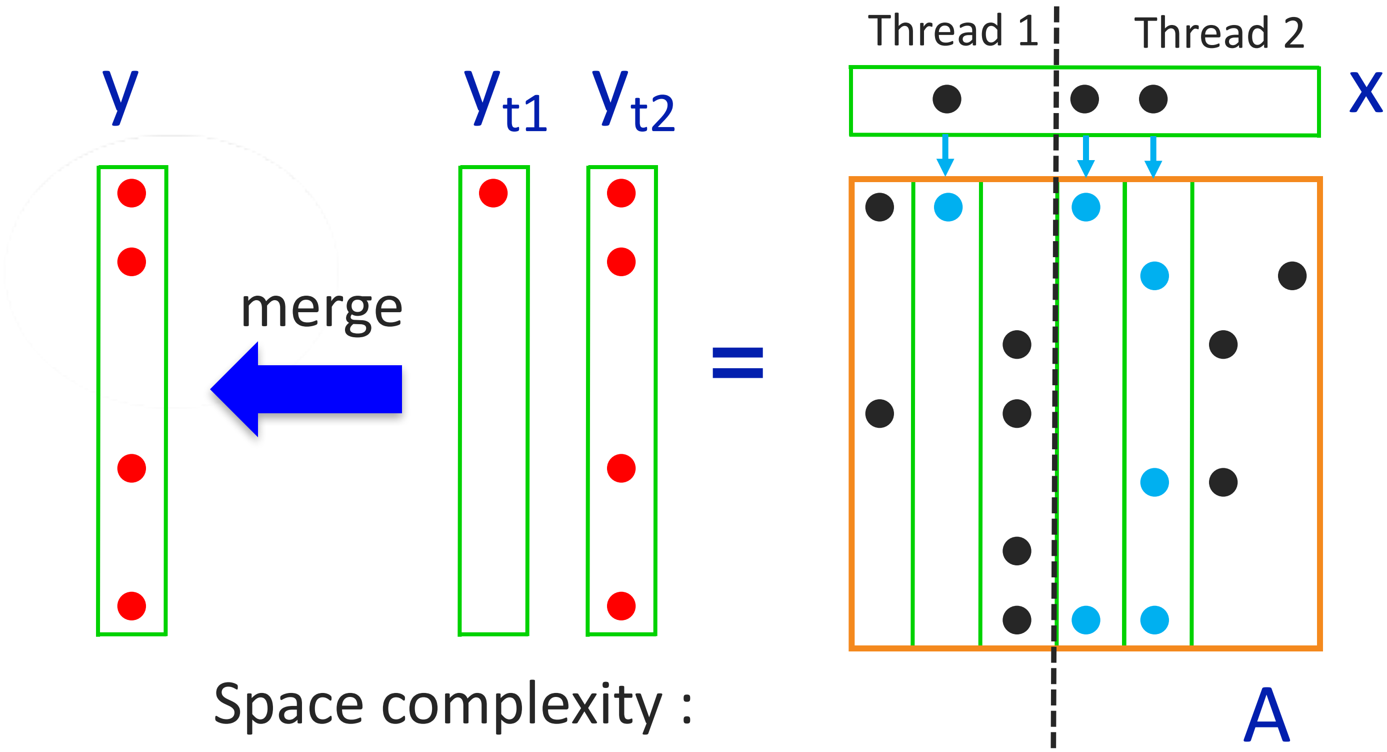


Used in CombBLAS (Buluc and Gilbert, IJHPCA 2011)

Explicitly split local submatrices into t (#threads) pieces

Work efficient?	Synchronization needed?
No : O(tf + df) total work	**No**

Shared-memory parallelization of SpMSpV (**column split**)



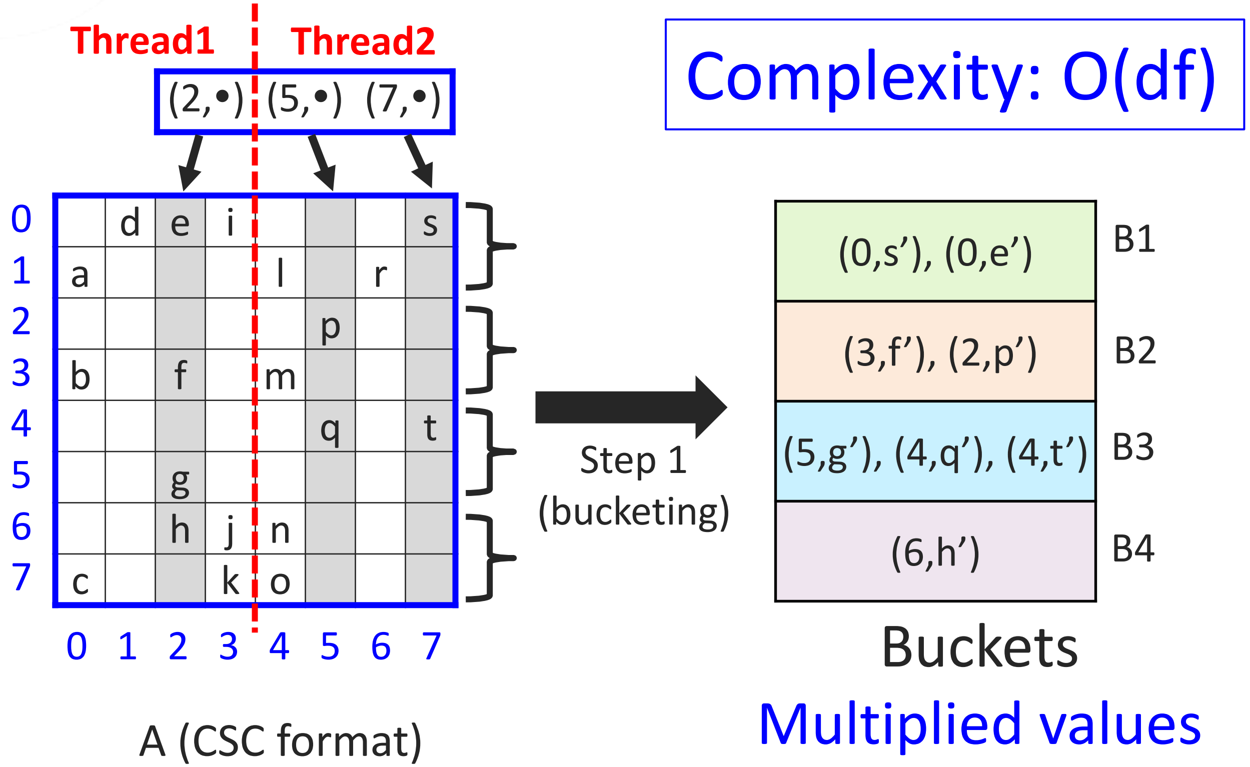
Space complexity :

O(n) storage per thread

Work efficient?	Synchronization needed?
Yes : O(df) total work	**Yes (in merging)**

A work-efficient and synchronization-avoiding SpMSpV algorithm using buckets

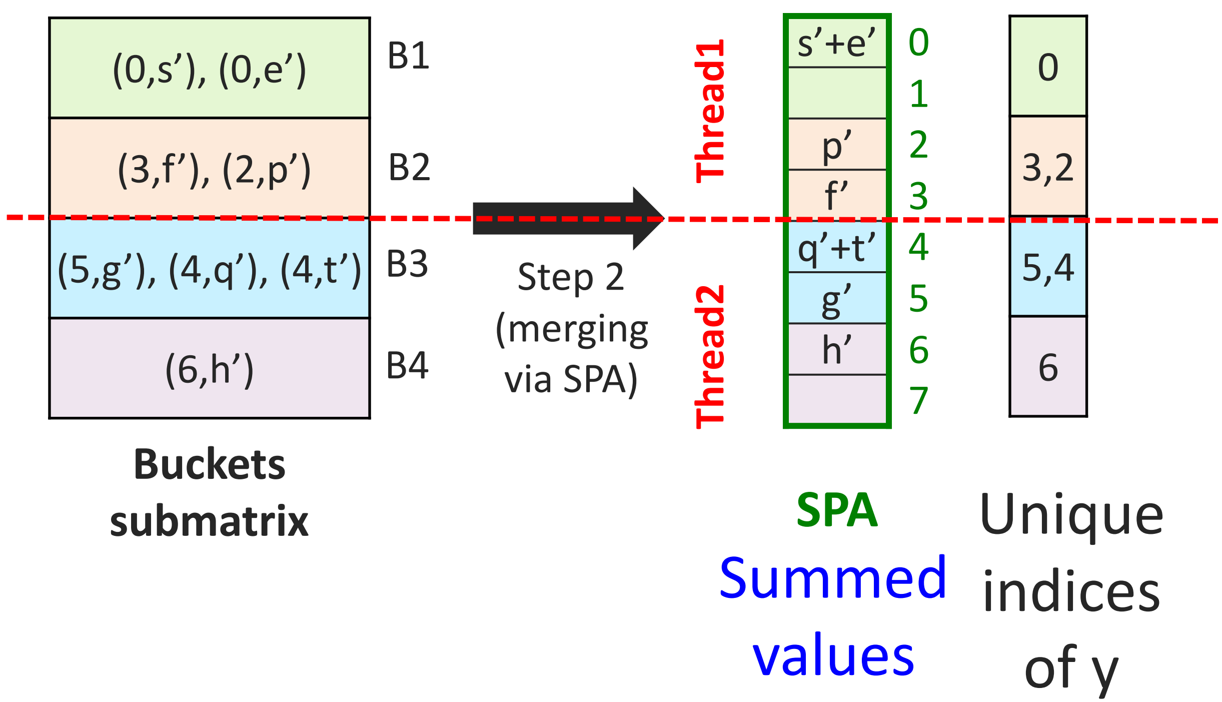
- **Multi-step algorithm**: keep good features of both row-split and column-split algorithms (**SpMSpV-bucket**)
- **Step1:** Arrange columns in buckets. Each bucket stores consecutive row indices. [similar to the column-split algorithm]



A work-efficient and synchronization-avoiding SpMSpV algorithm

- **Step2:** Merge each bucket independently by a thread. [similar to the row-split algorithm]

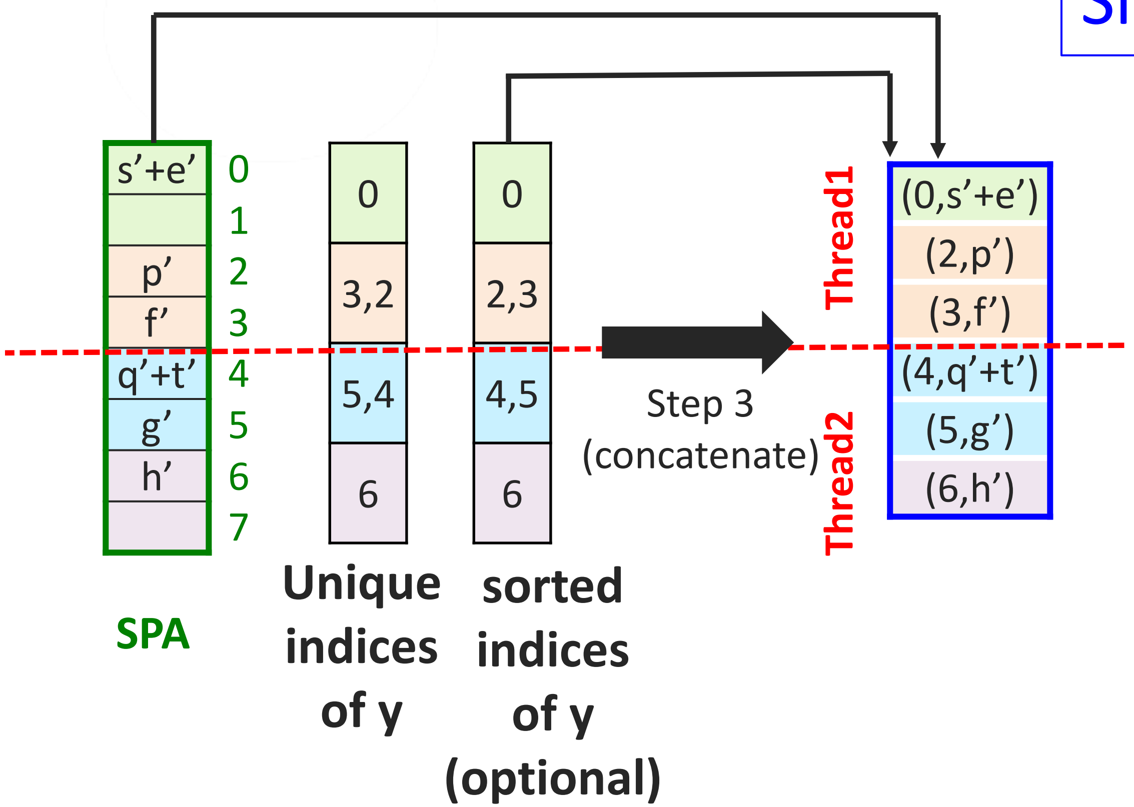
Complexity: O(df)



A work-efficient and synchronization-avoiding SpMSpV algorithm

- Step3: concatenate entries to the result vector

Complexity: O(nnz(y))
Smaller than O(df)



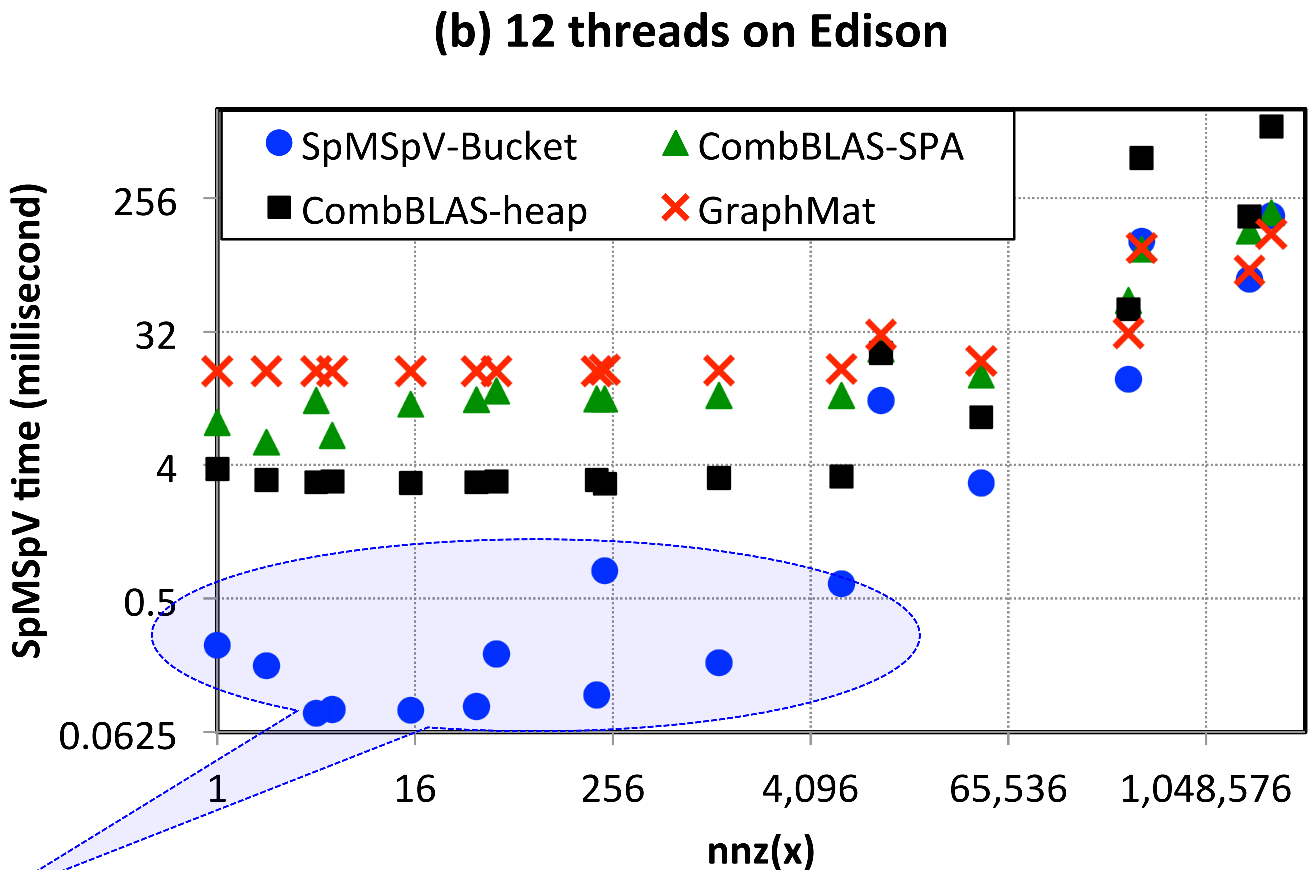
SpMSpV-bucket algorithm

Work efficient?	Synchronization needed?
Yes : O(df) total work At most **3df** work	**No (in each step)**

- Other tricks for practical performance
  - Load balancing: multiple buckets per thread
  - Cache efficiency: small thread-private buffers filled up first before writing to buckets

Relative performance of SpMSpV algorithms

Graph: ljournal-2008   Vertices: 5M, Edges: 78M

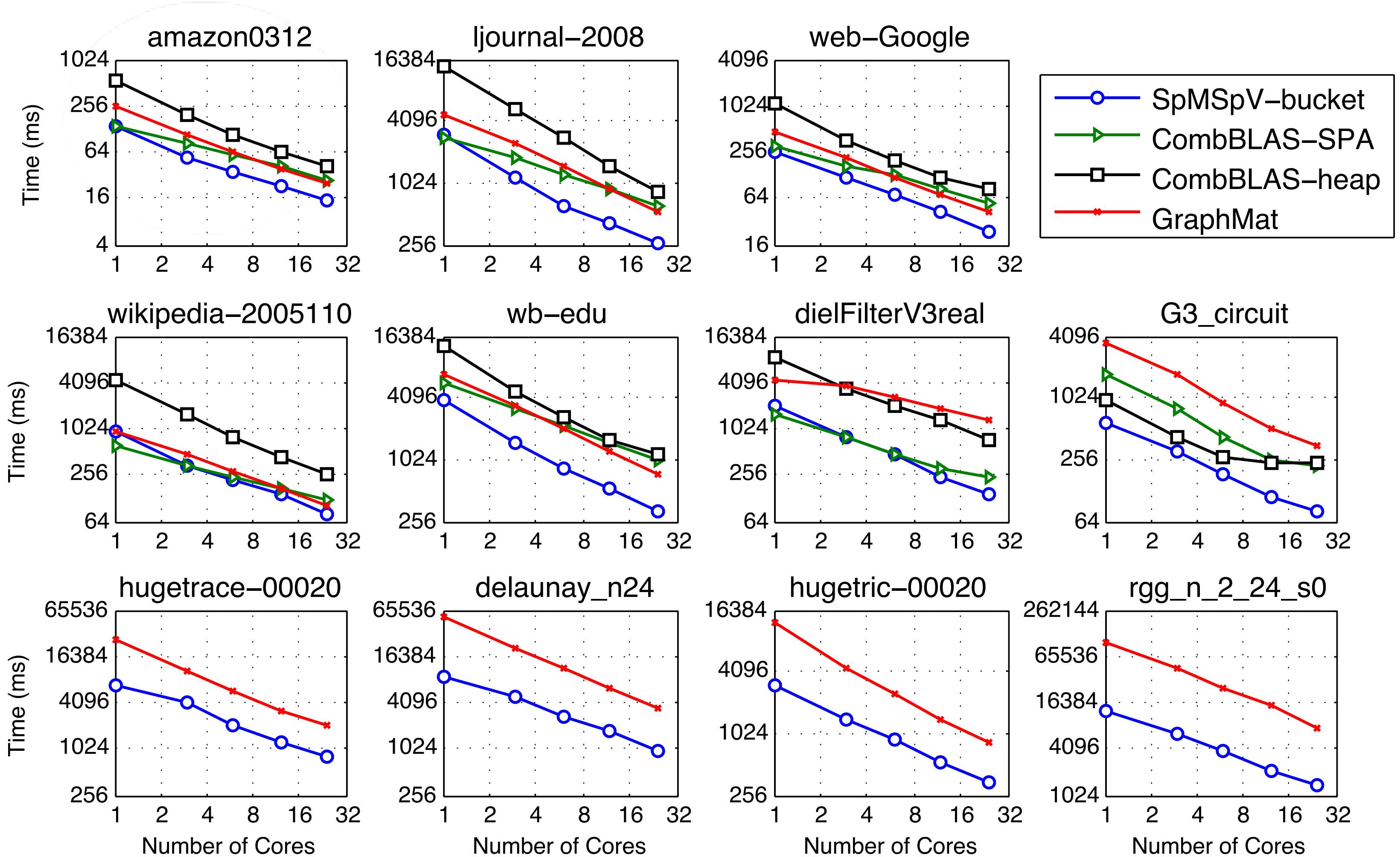


An order of magnitude faster for very sparse vectors

Strong scaling of SpMSpV algorithms on Edison

SpMSpV when used in BFS: varying sparsity of the input vector

Up to 4x faster than the second best algorithm



Conclusions

- **Algorithmic innovation**:
  - First-ever work-efficient SpMSpV algorithm
  - Attains work-efficiency by arranging necessary columns of the matrix into buckets
  - Avoids synchronization by processing buckets independently

- **Impact:**
  - Up to an order of magnitude faster than state-of-the-art algorithms when the input vector is very sparse
  - Will expedite a large class of graph and machine learning algorithms

Thanks for your attention