Combinatorial problems in sparse matrix computations

Sherry Li

Lawrence Berkeley National Laboratory

1st ACDA Workshop in Aussois, Sept. 5-9, 2022

Acknowledgement

- Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute under Contract No. DE-AC02-05CH11231.

FASTMATH

- Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration.

ECP

EXASCALE COMPUTING PROJECT

Collaborators

Collaboration of two ECP projects:

- Sparse solvers and preconditioners (SuperLU, STRUMPACK)
  - Lisa Claus, Pieter Ghysels, Yang Liu (LBNL)
  - Anil Gaihre, Hang Liu (Stevens Institute of Technology)
- ExaGraph (CombBLAS)
  - Ariful Azad, Aydin Buluc, Oguz Selvitopi (LBNL)
  - Johannes Langguth (Simula Lab)

Motivation
redesign of many combinatorial scientific computing algorithms

- Architecture driven
  - Coarse-grain massive parallelism:
    - **Communication-avoiding** for multi-node MPI
  - Fine-grain massive parallelism:
    - **Accelerators**: GPU, IPU (Johannes talk, Monday), FPGA, …
    - Future extreme heterogeneity
  - Different compromises to think
    - Larger memory capacity
    - Can afford more flops (as long as high Arithmetic Intensity)
- Application driven
  - GPU-resident solvers
  - KKT systems from optimization: preserve symmetry, compute matrix initia
  - Power grid optimization
  - …

Solving a linear system

For stability and efficiency, need to solve transformed linear system:

$Ax = b \rightarrow$

$P_c (P_r (D_r A D_c)) P_c^T P_c D_c^{-1} x = P_c P_r D_r b$

Preprocessing steps

- D_r , D_c : diagonal scaling (a.k.a. equilibration, balancing)
- P_r : row permutation vector (numerical pivoting for stability)
  - e.g. maximum weight matching
- P_c : row/column permutation vector (ordering to minimize fill-in, minimize communication)
  - e.g. graph partitioning

1. Numerical pivoting in sparse LU

- **Goal:** swap rows or columns to make diagonal elements large
- **Partial pivoting** does this dynamically
- Alternative pre-pivoting methods: quite stable in practice
  - Sequential: MC64 in Harwell Subroutine Library
    - DFS to grow a shortest alternating path tree

1. Numerical pivoting in sparse LU

- **Goal:** swap rows or columns to make diagonal elements large
- **Partial pivoting** does this dynamically
- Alternative pre-pivoting methods: quite stable in practice
  - Sequential: MC64 in Harwell Subroutine Library
    - DFS to grow a shortest alternating path tree
  - Distributed parallel heavy-weight perfect matching (**HWPM, available in CombBLAS**)
    - Approximate
    - Bipartite graph: Perfect matching + heavy weight

Ariful Azad, Aydın Buluc, Xiaoye S Li, Xinliang Wang, Johannes Langguth, A distributed-memory algorithm computing a heavy-weight perfect matching on bipartite graphs, SISC, 2020

Distributed heavy-weight perfect matching

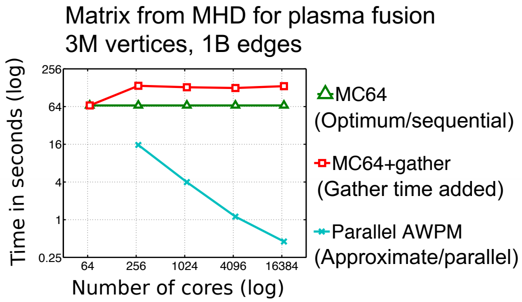
Algorithm elements: Use maximal, maximum cardinality, approx. weight matchings

- **Maximal**: A variant of the Karp-Sipser algorithm
- **Maximum cardinality**: A variant of the Hopcroft-Karp algorithm
- **Approximate weight**: A variant of Pettie-Sanders algorithm

- Primitives: Use sparse matrix (**GraphBLAS)** operations for performance

Results

- **Quality:** For most real matrices, HWPM returns perfect matchings within 99% (weight) of the optimum solution
- **Scalability**: Scales to 256 nodes (17K cores) on Cori/KNL
- **Speedup:** Can run up to 2500x faster than the sequential algorithm on 256 nodes of Cori/KNL

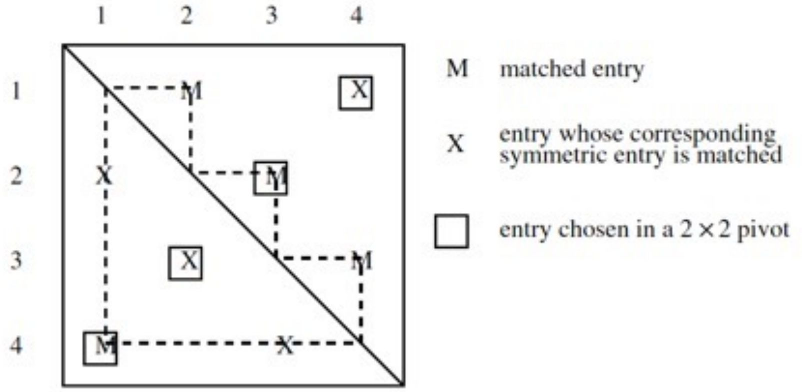


The parallel algorithm runs 300x faster than the sequential algorithm on 16K cores of NERSC/Cori

Extensions desirable

- GPU, or other accelerators?
- Symmetric pivoting for symmetric indefinite linear systems?
  - Performs symmetric factorization: LDL^T

  (Duff & Pralet): symmetric weighted matching to predefine 1x1 and 2x2 pivots in D
  - Start from a nonsymmetric matching M
  - Break into product of component cycles
    - Cycle of length 1 is on the diagonal
    - Even cycle length 2k gives k 2x2 pivots
    - Odd cycle length 2k+1 gives 2k+1 2x2 pivots

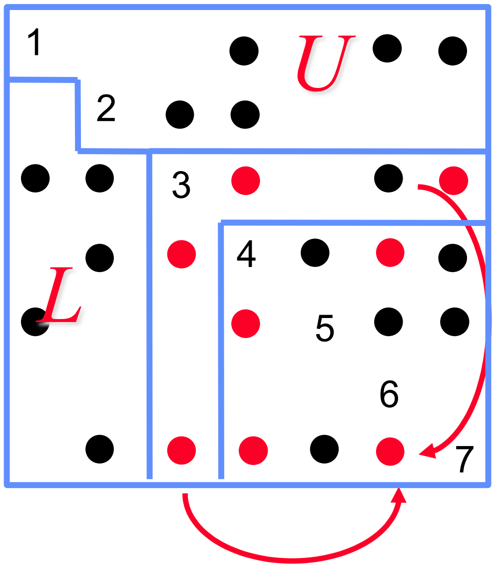


M : = (1,2), (2,3), (3,4), (4,1)
2 pairs of symmetric 2x2 pivots: (2,3)-(3,2), (4,1)-(1,4)

2. Symbolic factorization

Fill-in in sparse LU

“Transitive closure” for fill-in detection

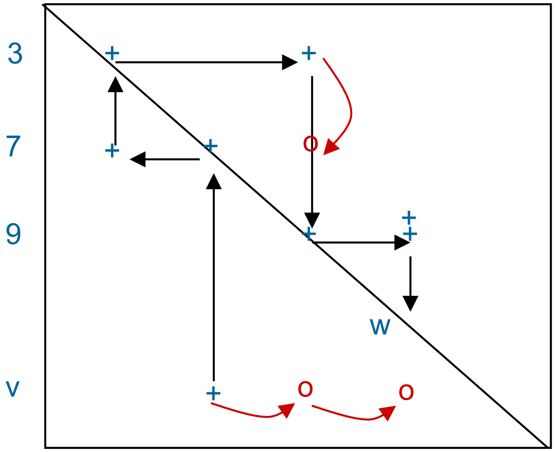


Fill-ins can be computed by reachability in original graph G(A)

"Fill-path" theorem (Rose/Tarjan 1978):

An edge (*v*,*w*) exists in the filled graph if and only if there exists a directed path from *v* to *w*, with intermediate vertices smaller than *v* and *w*

The edge (*v*,*w*) exists due to the path *v* → *7* → *3* → *9* → *w*



Traverse G(A)

Rose-Tarjan path-based algorithm

… essentially SSSP in disguise (Single Source Shortest Paths)

$\underline{Dijkstra(G, s, \cos t)}$

$\underline{\text{for each }} v \in V, \quad d(v) \leftarrow \infty, \quad pred(v) \leftarrow NIL$

$d(s) \leftarrow 0, \quad S \leftarrow \varnothing, \quad Q \leftarrow V$

$\underline{\text{While}}\ Q \neq \varnothing\ \underline{\text{do}}$

$\quad u \leftarrow \text{EXTRACT-MIN } (Q)$

$\quad S \leftarrow S \cup \{u\}$

$\quad \underline{\text{for each}}\ v \in Adj(u)\ \underline{\text{do}}$

$\quad\quad \underline{\text{if}}\ d(v) > d(u) + \cos t(u,v)\ \underline{\text{then}}$

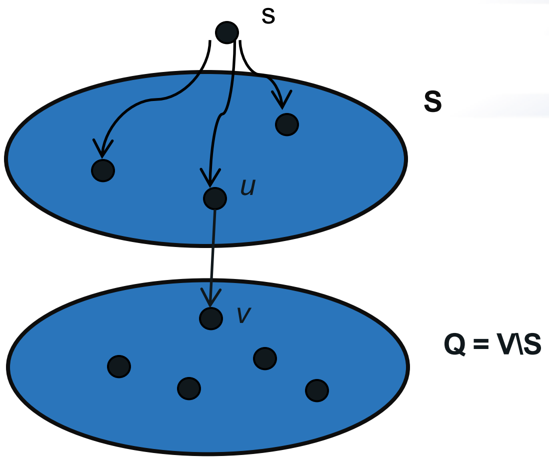
$\quad\quad\quad pred(v) \leftarrow u$

$\quad\quad\quad d(v) \leftarrow d(u) + \cos t(u,v)$

$\quad\quad \underline{\text{endif}}$

$\quad \underline{\text{endfor}}$

$\underline{\text{endwhile}}$



S := vertices with shortest paths found

Q := vertices with shortest paths upper bound given by d (d decreases through iterations)

Proposed parallel path-based algorithm

Redefine variables in Dijkstra

1. "d(v)" replaced by "max_id_array(v)"

   current maximum intermediate vertex number of all the paths leading to v

   (it decreases through iterations)

   S := { u | fill-in (s, u) already checked}

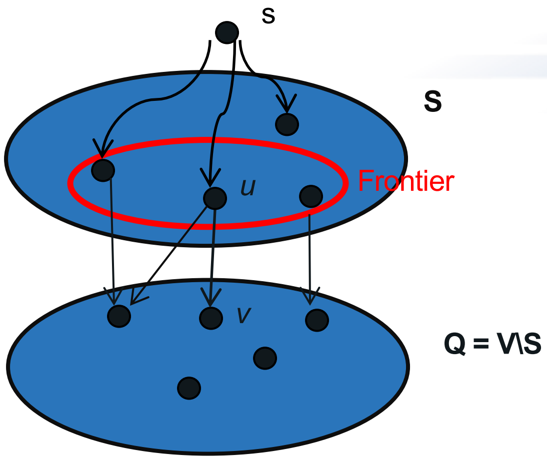
   Q := { v | fill-in (s, v) unknown yet }

2. Add constraint: Frontier := set of vertices eligible

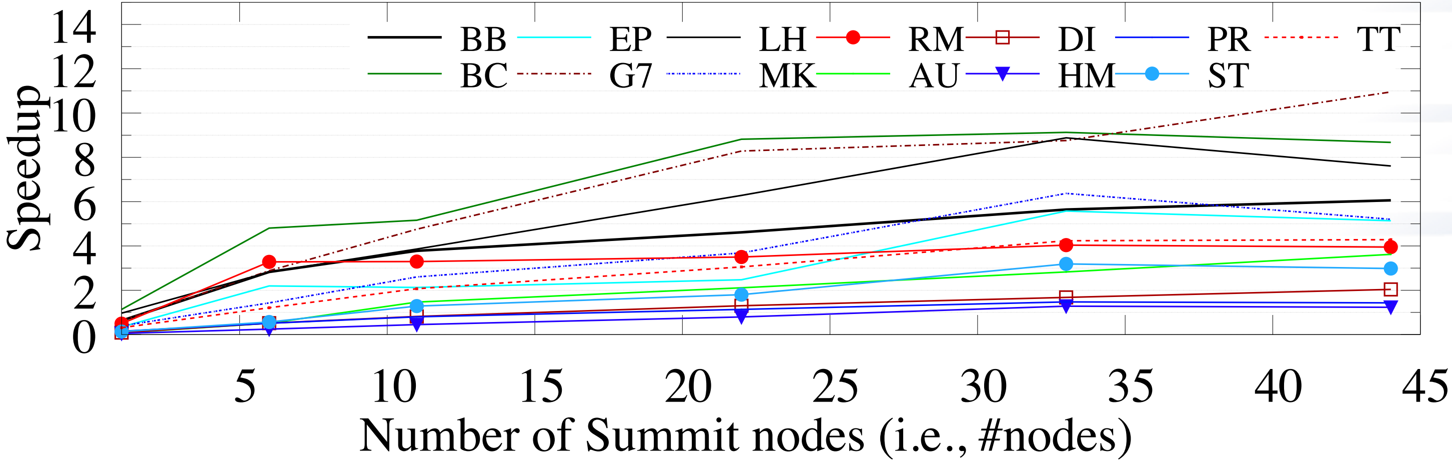
   for extending paths (their numbers are lower than source s)

3. Update rule is defined in order to find the path to v

   with minimum of max_id_array(v)



GPU parallel symbolic factorization result



- Encouraging: Summit per node: 6 GPUs, 42 IBM Power9 CPU cores
  - 1.3x - 10.9x speedups on 44 Summit nodes
  - By-product: a GPU parallel SSSP / MSSP
- No so satisfactory: only 5x faster than parallel CPU algorithm

Anil Gaihre, Yang Liu, Xiaoye Li, GSoFa: Scalable Sparse Symbolic Factorization on GPUs, IEEE TPDS 2020

An efficient left-looking algorithm (Gilbert/Peierls 1988)

-- edge-based elimination using filled graph G(L+U)

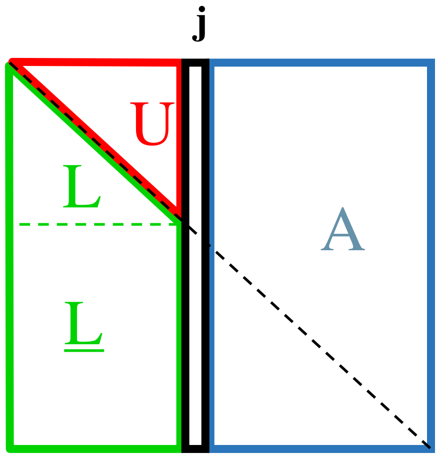
- Column j of A becomes column j of L and U
- Perform sparse triangular solve with A(:,j) as sparse RHS
- Time proportional to number of FLOPS in numerical factorization

<u>for</u> column j = 1 to n <u>do</u>

<u>Triangular solve</u>

$$\begin{pmatrix} L & \\ \underline{L} & I \end{pmatrix}\begin{pmatrix} u_j \\ l_j \end{pmatrix} = a_j \text{ for } u_j \text{ and } l_j$$

<u>scale</u>: $l_j = l_j / u_{jj}$

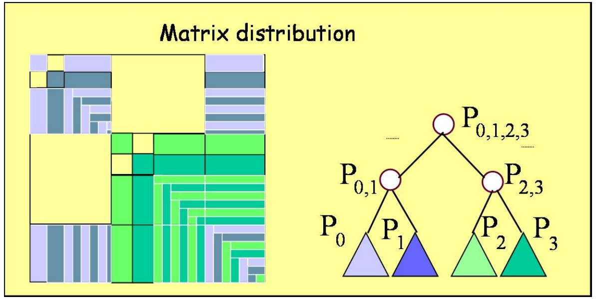


This is essentially what's implemented in SuperLU

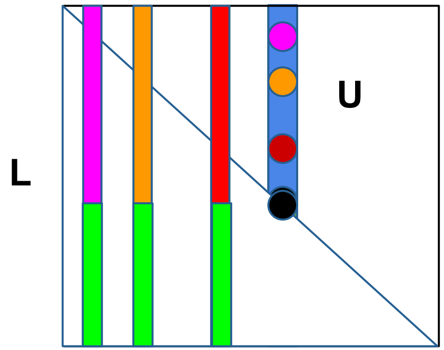
- **Two enhancements:**
  - Symmetric pruning to reduce redundant search (Eisenstat/Liu 1992)
  - Supernodes

- Shared-memory: SuperLU_MT (Demmel/Gilbert/Li 1999)
  - Partial pivoting: symbolic & numerical factorizations interleave
  - 20x speedup @ 32 processors
- Distributed-memory: SuperLU_DIST (Grigori/Demmel/Li 2007)
  - 9x speedup @ 32 processors

- How to do it on GPU?

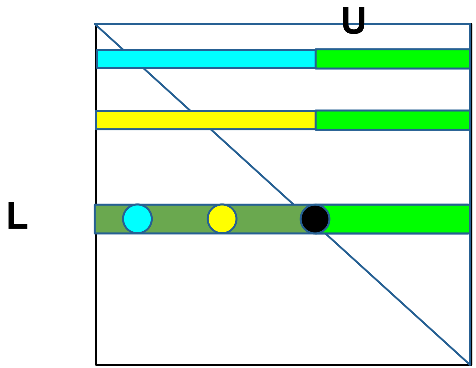


Attempt: Leverage GraphBLAS: left-up-looking (Oguz Selvitopi)



- Extract column of U
- Compute fill-ins using corr. cols of L **(SpMV)**

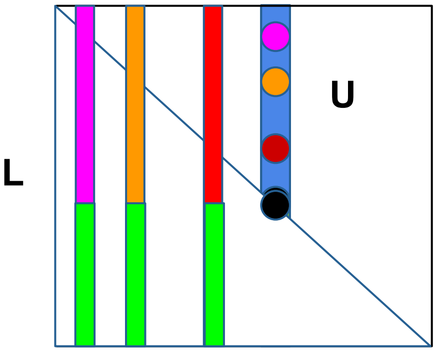
- Use mask to avoid comp. with upper part



- Extract row of L
- Compute fill-ins using corr. rows of U **(SpMV)**

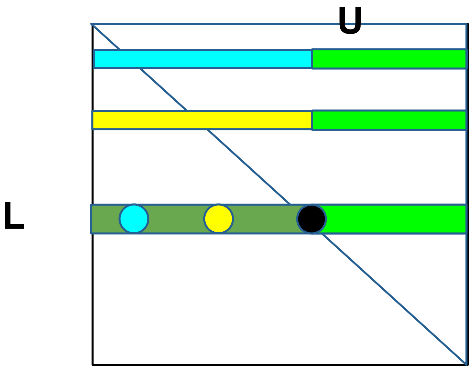
- Use mask to avoid comp. with left part

Attempt: Leverage GraphBLAS: left-up-looking (Oguz Selvitopi)



- Extract column of U
- Compute fill-ins using corr. cols of L **(SpMV)**

- Use mask to avoid comp. with upper part



- Extract row of L
- Compute fill-ins using corr. rows of U **(SpMV)**

- Use mask to avoid comp. with left part

Block version: SpGEMM