On the scheduling of sparse matrix factorizations: is it time for a change?

Mathias Jacquelin, Esmond Ng
Computational Research Division
Lawrence Berkeley National Laboratory

Sparse Matrix Factorization

- Factorization of a sparse symmetric positive definite matrix incurs fill.
  - i.e., some zero entries in the original matrix will become nonzero in the factors.
- Elimination trees
  - The elimination tree is a compact structure that encapsulates a lot of information related to the sparsity of the Cholesky factor and the dependency among the columns.
    - Schreiber 1982
    - Liu 1986
    - Liu 1990
  - Have been generalized to nonsymmetric matrices
- Elimination trees have played an important role in designing efficient matrix algorithms.

Parallel Sparse Matrix Factorization

- Work on *practical* parallel sparse matrix factorization started in 1980's.
- New challenges (at the time) included
  - Identifying tasks
  - Mapping the matrix to the processing units and scheduling computational tasks among the processing units.
- Because the elimination tree provides information on the column dependency, it was considered to be an appropriate tool for studying data mapping and task scheduling.
  - Much of the work was initially focused on level-by-level approaches.
    - *e.g.*, the tasks associated with the leaves of an elimination tree are independent.

Communication Needs in Parallel Sparse Matrix Factorization

- Another challenge in parallel sparse matrix factorization is getting a handle on the communication cost.
  - The level-by-level scheduling is convenient but it may not balance the work load, particularly on distributed-memory platforms.
- The subtree-to-subcube mapping [George, Liu, Ng (1989)]
  - Working on the Intel hypercube at the time
  - Interested in analyzing the communication requirements in parallel sparse Cholesky on the hypercube.
  - Took a model problem – k x k grid, with a 5-point (or 9-point) operator, ordered by nested dissection (which is optimal in terms of fill and operations).
  - The elimination tree was perfectly balanced.

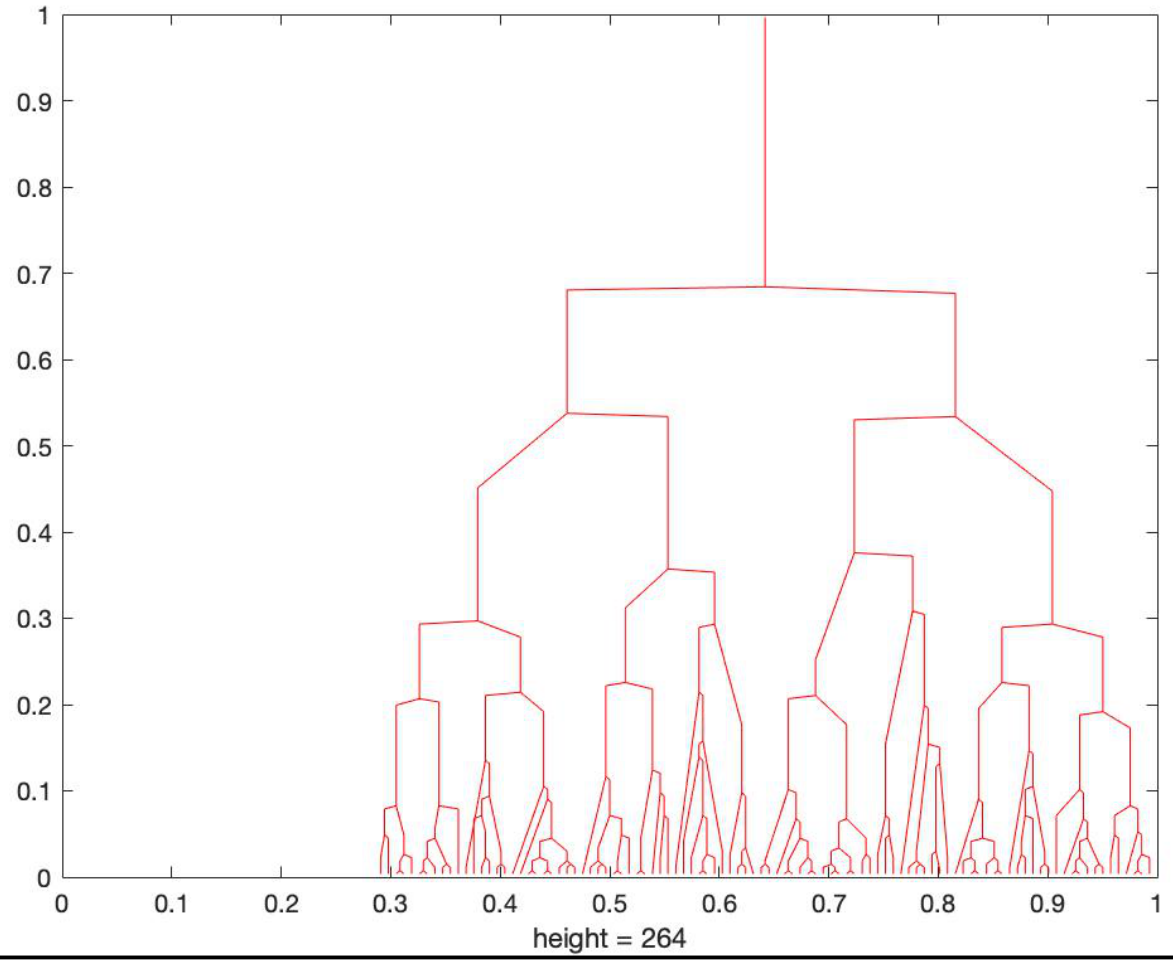
Communication Needs in Parallel Sparse Matrix Factorization

- Proposed scheduling:
  - Assign all columns in the top separator to the dimension d hypercube.
  - The separator divides the grid into 2 halves. Each half forms a subtree. Assign a dimension d-1 hypercube to each subtree.
  - Recurse on the subtrees and subcubes.

- [George, Liu, Ng (1989)] proved that the communication volume was optimal for the model problem.
- [Gao, Parlett (1990)] further proved that the number of messages was optimal and that the communications are balanced.

Communication Needs in Parallel Sparse Matrix Factorization

- The results in [Geo … k grids, ordered using nested dissec…
- How about general…
- The elimination tre… nced.
  - [Liu (1988)] propos… ination tree while preserving fill.
    - Effects tended to b…
  - The papers cited al… uld work well for general sparse sym…
  - But it didn't prever…



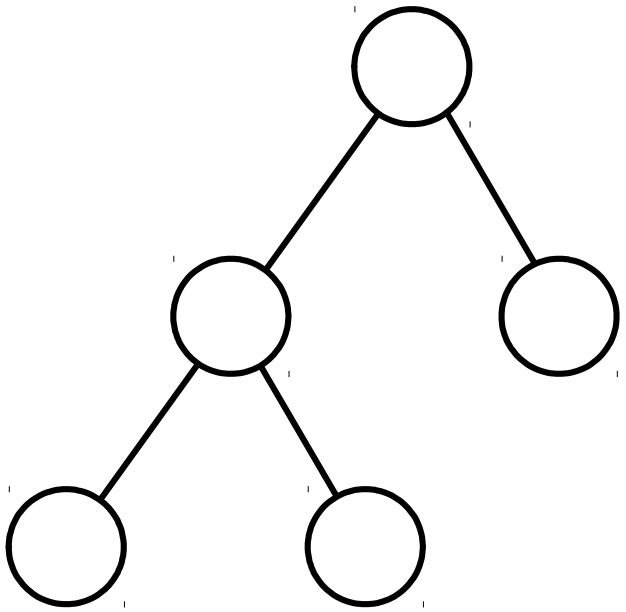
bcsstk14 (stiffness matrix associated with the roof of the Omni Coliseum in Atlanta)

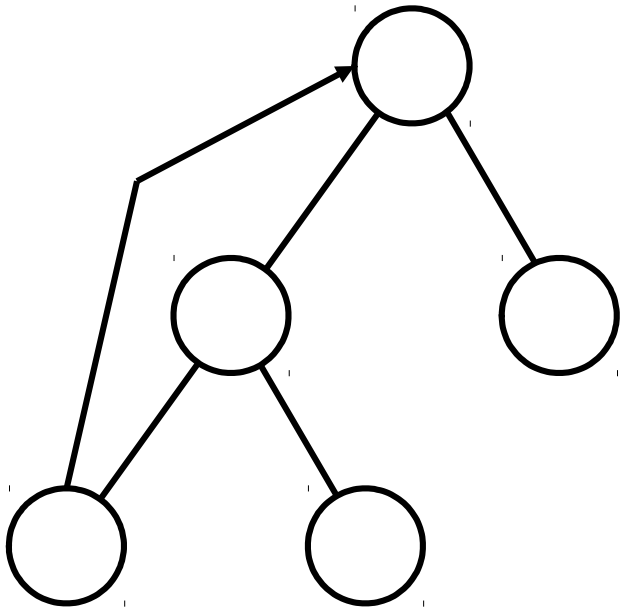
Extending subtree-to-subcube to general sparse matrices

- [Geist, Ng (1990)] probably was the first to attempt generalizing subtree-to-subcube to general sparse symmetric matrices.
  - Assign weights to nodes in the elimination tree.
  - The weights are functions of the numbers of operations required by the subtrees.
  - Use a bin-packing strategy to map tasks.
- [Pothen, Sun (1993)] proposed proportional subtree-to-subcube mapping.
  - It was essentially a generalization of the one proposed by [George, Liu, Ng (1989)], but the processing units are partitioned according to ratios of the weights associated with the subtrees.
- The subtree-to-subcube (and its variants) worked reasonably well until the number of processing units has become large, as on today's platforms.
  - Dynamic scheduling [Amestoy,Duff, L'Excellent, Koster (2000)], [Faverge, Ramet (2008)]
  - Prasanna and Musicus [Beaumont, Guermouche (2007)]
- Subtree-to-subcube mappings don't work well in all situations.
  - Fan-both mappings conflicts with it (updates may be computed on processor not owning the data)

What to do then???

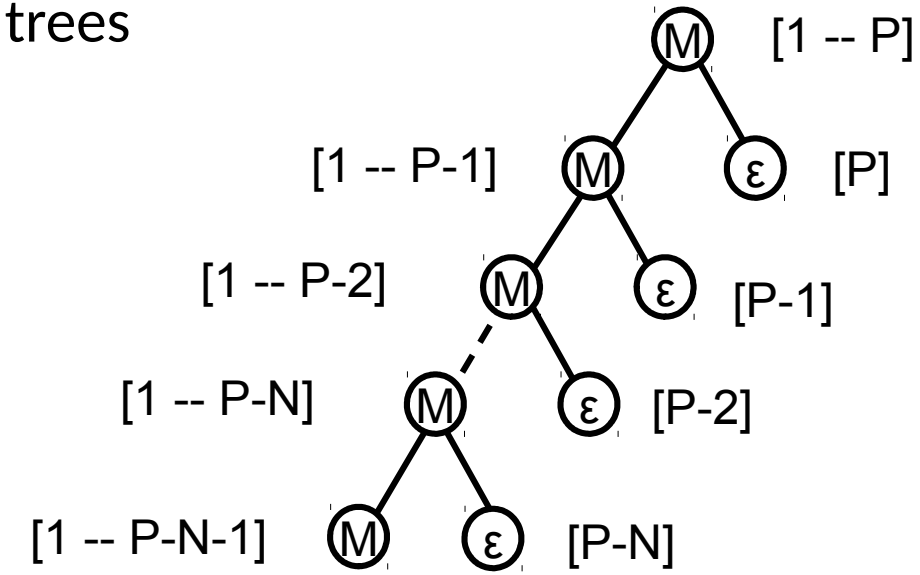
- Elimination tree is spanning tree of task graph
  - Not all dependencies are represented





What to do then???

- Elimination tree is spanning tree of task graph
  - Not all dependencies are represented
- Subtree-to-subcube / proportional mapping sacrifice processors on small outlying branches, bad for unbalanced trees



What to do then???

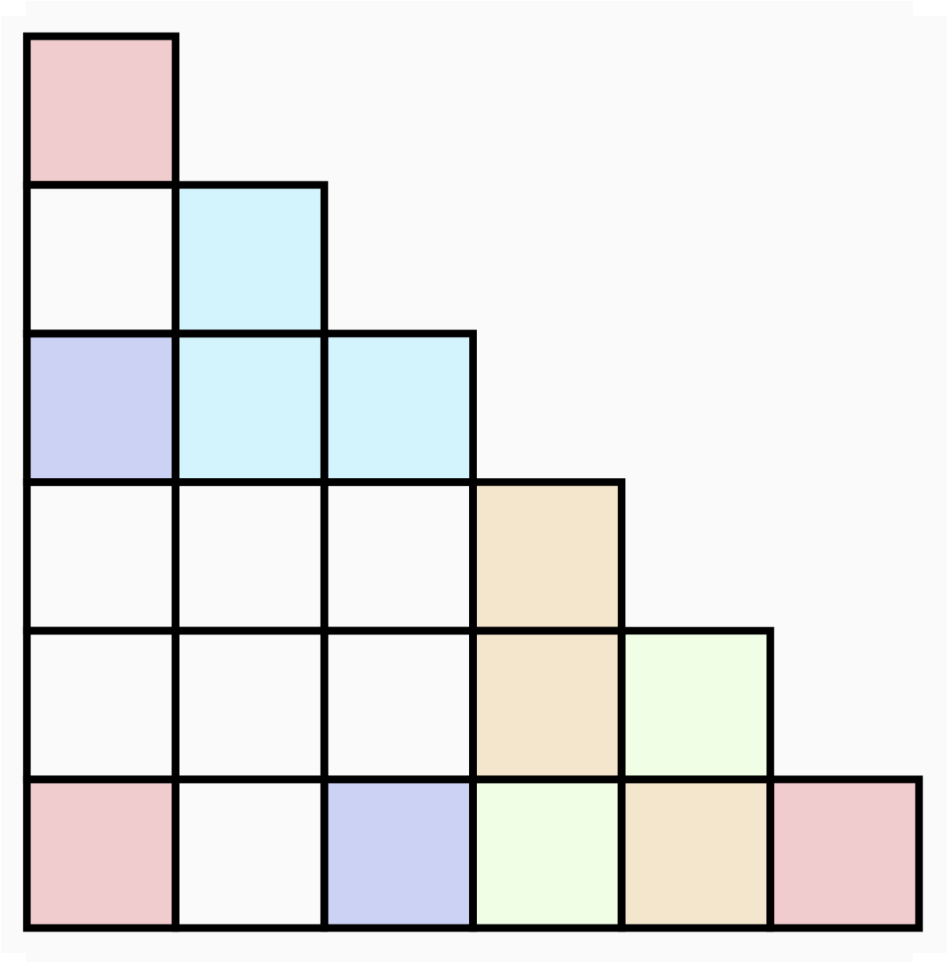
- Elimination tree is spanning tree of task graph
  - Not all dependencies are represented
- Subtree-to-subcube / proportional mapping sacrifice processors on small outlying branches, bad for unbalanced trees
- Current & future platforms (manycore, GPUs) have a significantly larger number of processing elements

→ Finer granularity more likely to keep the hardware busy

→ Finer granularity more likely to overlap comm. with comp.

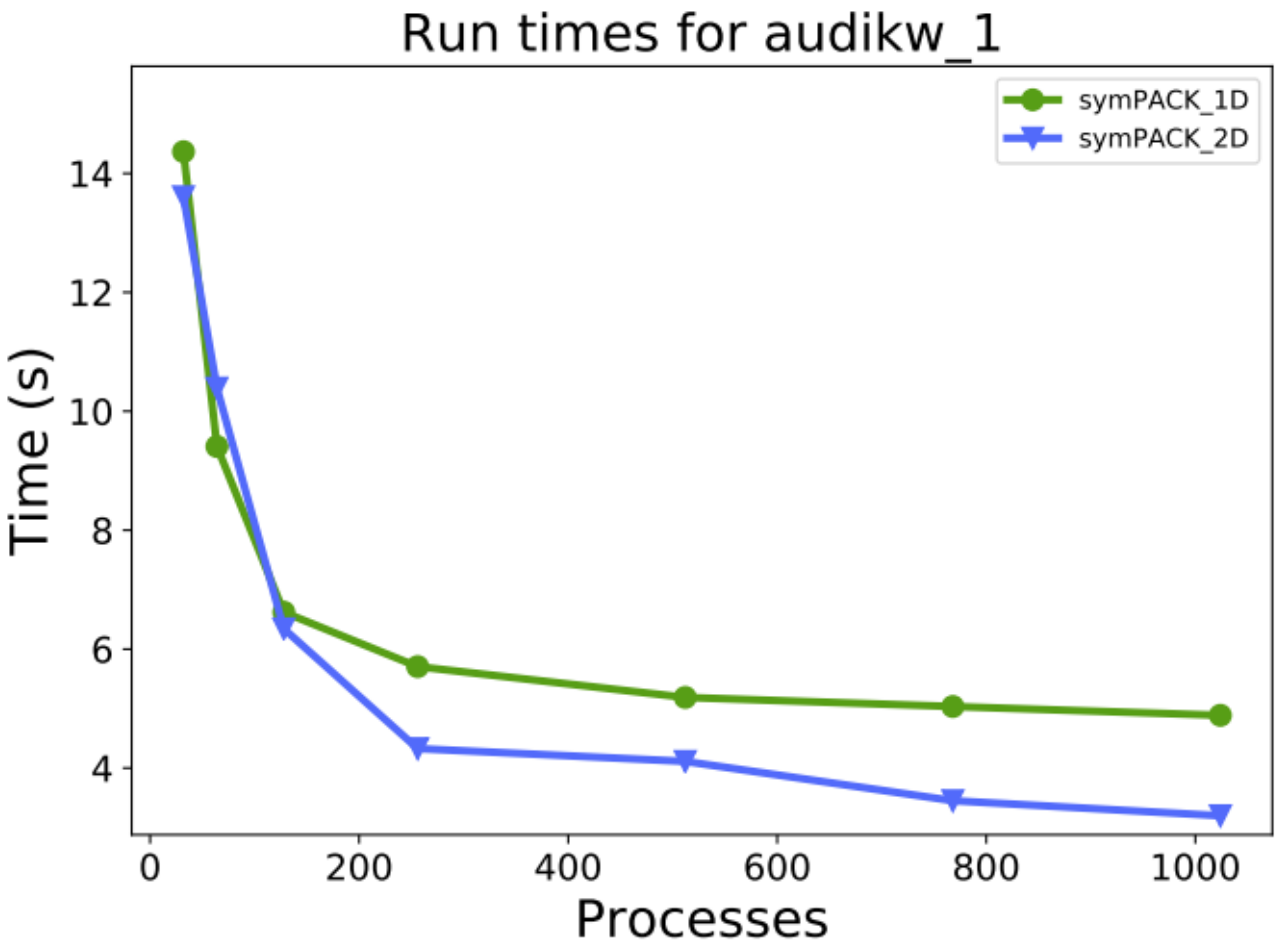
Task formulations do work

- SymPACK, solver for sparse symmetric matrices
  - Task based execution model
  - UPC++ one-sided communications
  - Coarse granularity distribution & task scheduling
    - Unit is supernode
  - Finer granularity & task scheduling
    - Unit is block within a supernode



Task formulations do work

- SymPACK, solver for sparse symmetric matrice
  - Task based execution model
  - UPC++ one-sided communications
  - Coarse granularity distribution & task scheduling
    - Unit is supernode
  - Finer granularity & task scheduling
    - Unit is block within a supernode
  - Pastix, similar approach



Conclusion & future work

- Scheduling is **VERY important**
- Mapping based on the task graph instead of elimination tree?
- Aggregate updates using a tree pattern (reduction)
- Hybrid strategies:
  - 1D data distribution at leaves?
  - 3D layout at higher levels: multiple tasks on the same cell
- Accelerator / GPU support via asynchronous tasks
  - Upcoming UPC++ with seamless local/remote host/device memory accesses
- Acknowledgments:
  - DOE SciDAC FASTMath, CompCat, ComPASS4, ECP Pagoda