symPACK

symPACK: a solver for sparse Symmetric Matrices

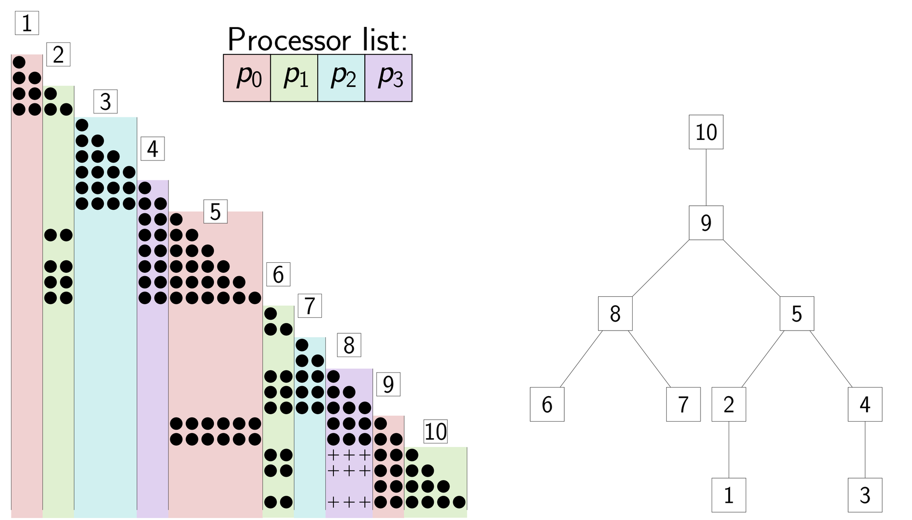
Mathias Jacquelin, Esmond Ng, and Katherine Yelick
Scalable Solvers Group, Lawrence Berkeley National Laboratory, Berkeley, California, USA
Yili Zheng
Google, California, USA

Abstract

We present a parallel distributed-memory sparse symmetric solver, symPACK, based on an asynchronous task paradigm using one-sided communication and dynamic scheduling for computing the Cholesky factorization.
The performance and resource usage of sparse matrix factorizations are critical to time-to-solution and maximum problem size solvable on a given platform. Exploiting symmetry in sparse matrices reduces both the amount of work and storage cost required for factorization. On large-scale distributed memory platforms, communication cost is critical to performance.
symPACK relies on efficient and flexible communication primitives provided by the UPC++ library. Experiments shows good scalability and that symPACK often outperforms state-of-the-art parallel distributed memory factorization packages.

Right-looking Sparse Cholesky factorization

- Fill-in-reducing ordering
  - Multiple Minimum Degree, Approximate Minimum Degree
  - Nested-dissection ((PAR)Metis and (PT)Scotch graph partitioners)
  - Symmetric permutation
- Supernodal elimination tree
  - A Supernode is a set of columns with a dense diagonal block and same off-diagonal non-zero row structure
  - The Elimination tree describes dependencies between supernodes.
- At every node in the elim. tree
  - Supernode factored locally
  - Updates some ancestors in the tree
- Linear solve phase
  - Forward and backward solve: bottom-up and top-down tree traversals



(a) Structure of factor **L** (b) Supernodal elim. tree

Figure 1: Sparse matrix **A** supernode partition, $\boxed{i}$ denotes the i-th supernode. • represents original nonzero elements in **A**, while + denotes fill-in entries. Colors correspond to the 4 distributed memory nodes on which supernodes are mapped in a 1D-cyclic way.

Fan-in, Fan-out and Fan-both factorization algorithms

- Three families of parallel Cholesky algorithms [1]:

  fan-in: Updates from a column k to other columns i computed on the processor owning column k.
  Processor owning i will have to "fan-in" (or collect) updates from previous columns.

  fan-out: Updates from k computed on processors owning columns i.
  Processor owning k has to "fan-out" (or broadcast) column k of the factor.

  fan-both: Updates are allowed to be performed on any processors. Relies on *computation maps*.

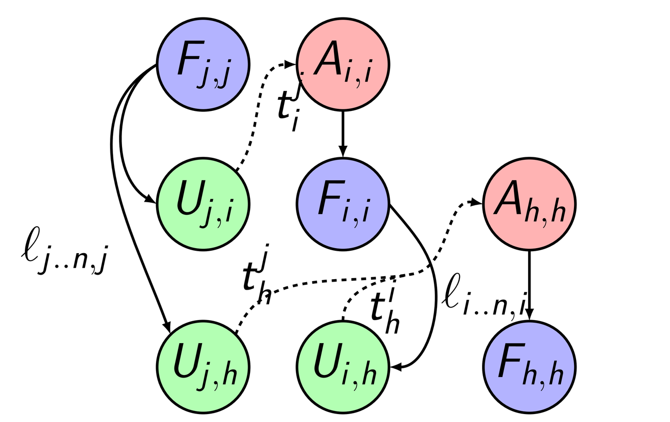


Figure 2: *fan-both* task dependencies for three columns j, i and h

- Three types of tasks:
  - *Factorization* $F_{i,i}$: compute column i of the Cholesky factor.
  - *Update* $U_{i,j}$: compute the update from $\ell_{j:n,i}$ to column j, with $i < j$ such that $\ell_{j,i} \neq 0$, and put it to an *aggregate vector* t_j^i.
  - *Aggregation* $A_{j,j}$: apply all aggregate vectors t_j^i from columns $i < j$, with $\ell_{j,i} \neq 0$, to column j.

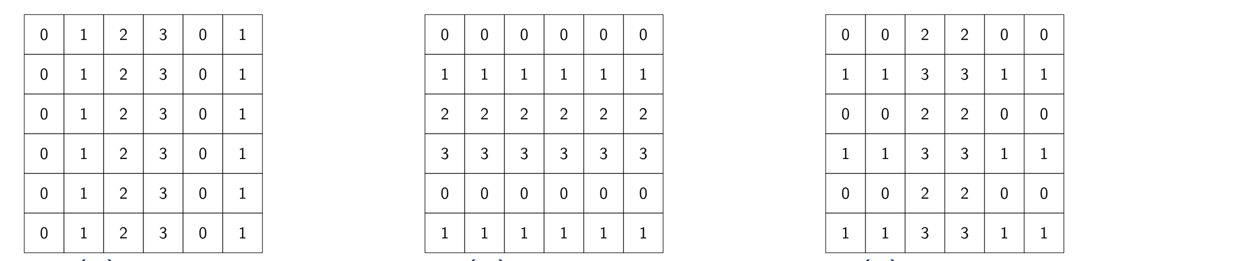
symPACK – symmetric matrix PACKage

Solver for Sparse Symmetric Linear Systems – http://www.sympack.org/

- Sparse direct linear solver for symmetric positive definite matrices
- Aimed at matrices from PDE discretizations, shift-invert Lanczos, ...
- Sparse symmetric data storage: lower storage cost
- Real, complex, single & double precision and 32/64-bit indexing (C++ templates)
- Scalable distributed memory UPC++ code, no multi-threading curently available

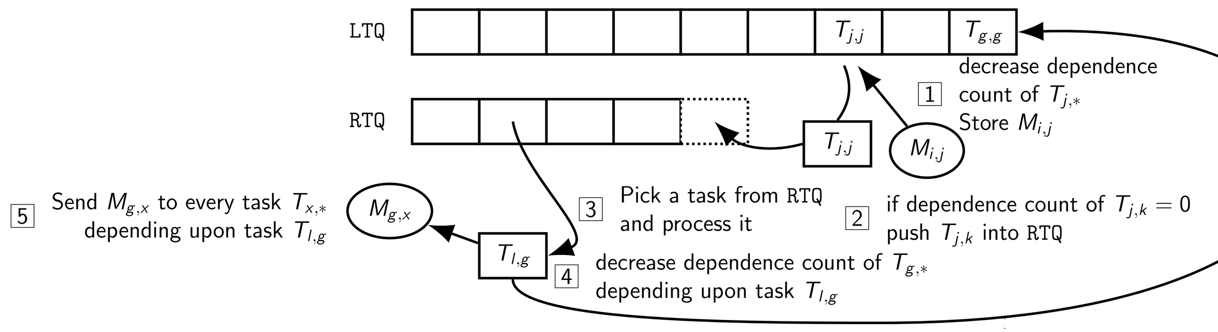
Data layout and computation mapping

- 1D cyclic Supernodal layout, sequential BLAS is called within each supernode
- Supernodes assigned to nodes based on estimated work (proportional mapping)
- Computation map determines node ranks where tasks are mapped/executed:



(a) *Fan-in* $\mathcal{M}_{i,j} = mod(i, P)$ (b) *Fan-out* $\mathcal{M}_{i,j} = mod(j, P)$ (c) *Fan-both* $\mathcal{M}_{i,j} = mod(\min(i,j), \sqrt{P}) + \sqrt{P}\lfloor mod(\max(i,j), P)/\sqrt{P}\rfloor$

Asynchronous Task Execution



- Dynamic task scheduling within distributed memory node:
  - *local task queue* (LTQ), contains all local tasks awaiting execution,
  - *ready task queue* (RTQ), contains all local tasks ready for execution.
- One sided "pull" communication protocol using UPC++:
  - 1 Sender notifies available data (sends global pointer)
  - 2, 3 & 4 Recipients periodically gets incoming data
  - 5 One-sided communications without interrupting sender/recipient
  - Remote temporary data deallocation

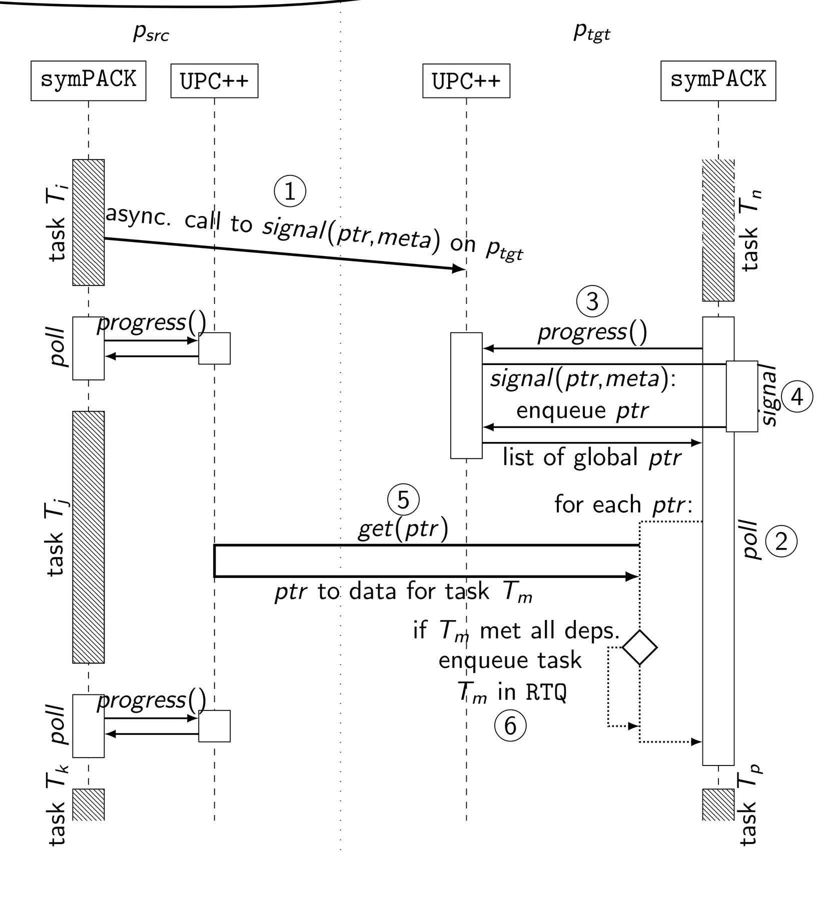


Figure 3: Data exchange protocol in symPACK.

Numerical Results on NERSC Edison, 2x 12-core processors per node

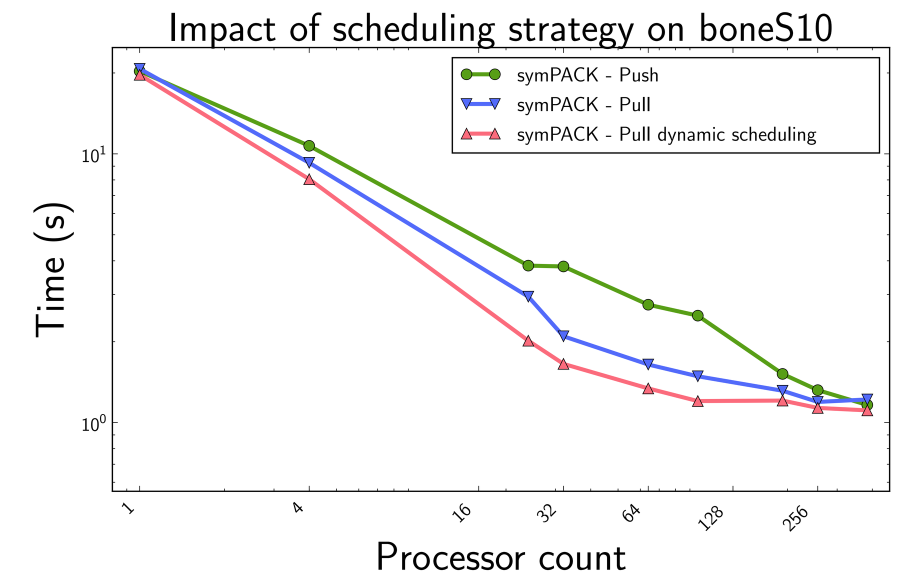


Figure 4: Impact of comm. strategy and scheduling

- *Push*: two-sided MPI communications + deadlock avoiding strategy
- *Pull*: one-sided UPC++ communications
- *Pull + dynamic scheduling*: UPC++ and dynamic task scheduling within a node
- *Pull* protocol has low overhead
- Ordering computation and comm. to *avoid deadlocks* constrains the schedule
- **Dynamic scheduling improves performance**

- Optimizations and scheduling strategy for distributed memory nodes
- Every solver is run in "Flat-MPI" mode
- SuperLU_DIST timings given for scalability trend
- PASTIX and MUMPS are two symmetric solvers
- symPACK **outperforms state-of-the-art**

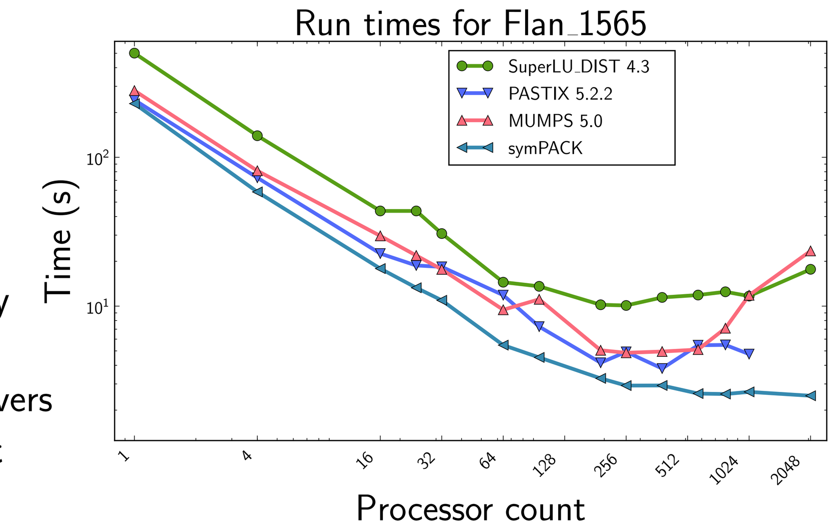


Figure 5: Strong scaling on Flan_1565

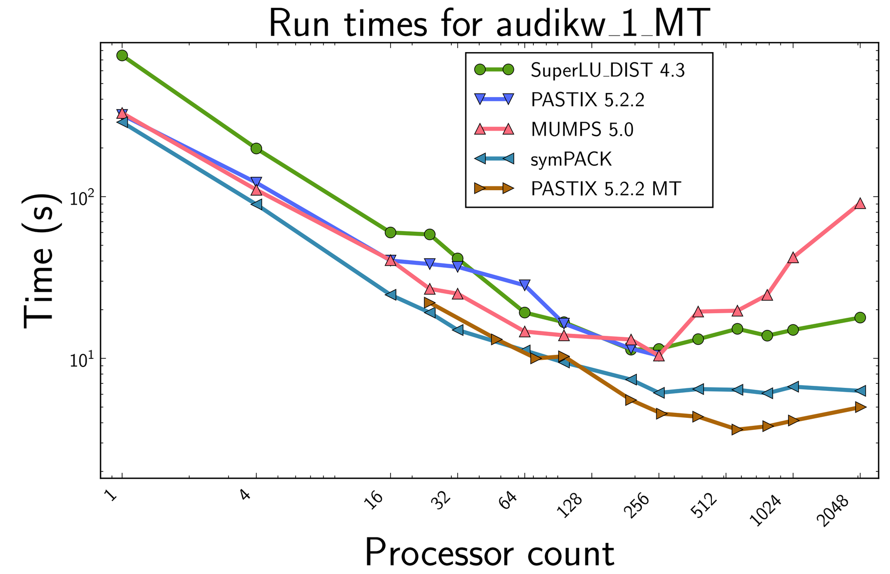


Figure 6: Strong scaling on audikw_1

- Hybrid approach useful to reduce communication costs:
  - MPI / OpenMP
  - UPC++ / OpenMP
  - UPC++ / UPC++
- PASTIX uses 24 threads per node
- Multi-threading helps PASTIX a lot
- symPACK has no hybrid implementation yet but coming soon

Partner frameworks and applications

- Now integrated in the PEXSI library, available in the ELectronic Structure Infrastructure (ELSI) and used in several applications: DGDFT, SIESTA, CP2K.

UPC++ PEXSI ELSI

References

[1] C. C. Ashcraft, "A taxonomy of column-based cholesky factorizations," 1996.

[2] Y. Zheng, A. Kamil, M. B. Driscoll, H. Shan, and K. Yelick, "UPC++: A PGAS extension for C++," in *28th IEEE International Parallel and Distributed Processing Symposium (IPDPS'14)*.

[3] "UPC++ website," https://bitbucket.org/upcxx/upcxx.

[4] "GASNet home page," http://gasnet.lbl.gov.

[5] X. S. Li, "An overview of SuperLU: Algorithms, implementation, and user interface," *ACM Trans. Math. Software*, vol. 31, no. 3, pp. 302–325, Sep. 2005.

[6] P. Hénon, P. Ramet, and J. Roman, "Pastix: a high-performance parallel direct solver for sparse symmetric positive definite systems," *Parallel Computing*, vol. 28, no. 2, pp. 301–321, 2002.

[7] P. Amestoy, I. Duff, J.-Y. L'Excellent, and J. Koster, "A fully asynchronous multifrontal solver using distributed dynamic scheduling," *SIAM J. Matrix Anal. and Appl.*, vol. 23, 2001.

yilizheng@google.com, {mjacquelin,egng,kayelick}@lbl.gov