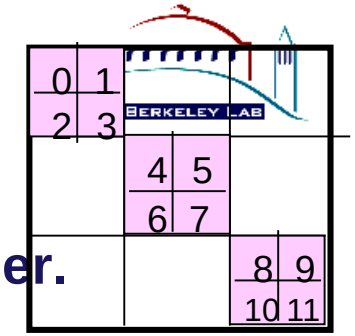
Sparse solvers & preconditioners

Sherry Li, LBNL



- **SuperLU direct solver used in M3D-C1 and NIMROD for many years. In 3D, used as Block-Jacobi preconditioner.**

- **STRUMPACK also based on factorization, but more flexible**
  1. **Direct solver without approximation**
  2. **"Inexact" direct solver with small low-rank compression (e.g. set drop tolerance = 1e-8)**
     - **One factorization, one triangular solve.**
     - **May ask for a few steps of iterative refinement to get higher accuracy.**
  3. **Preconditioner with large compression (e.g. set drop tolerance = 1e-2)**
     - **Can use PETSc's GMRES (or any iterative solver), specify STRUMPACK as preconditioner.**
- **May be worthwhile applying STRUMPACK preconditioner for the entire 3D problem, not just block diagonals.**

SuperLU factorization optimization on Cori KNL

Work with Sam Williams, Jack Deslippe, Steve Leak, Thanh Phung (Intel)
(New release SuperLU_DIST version 5.2.0)

- **Overall, factorization is up to 80% speedup on single node.**
  - **Jin Chen's experiments at Intel Dungeon: M3D-C1 10% speedup.**
  - **Total time in SuperLU 50-80% : LU 43%, Triangular solve 57%**

- Replace small independent single-threaded MKL GEMMs by large multithreaded MKL GEMMs: **15-20% faster.**
  - **Important to link with a good multithreaded BLAS.**
- Use new OpenMP features: **10-15% faster.**
  - "task parallel" to reduce load imbalance
  - "nested parallel for" to increase parallelism
- Vectorizing Gather/Scatter: **10-20% faster.**
  - **Hardware support: Load Vector Indexed / Store Vector Indexed**

```
#pragma omp simd     // vectorized Scatter
    for (i = 0; i < b; ++i) {
            nzval[ indirect2[i] ] = nzval[ indirect[i] ] - tempv[i];
    }
```

- Several techniques to reduce cache misses.

Dungeon Sept 2017: M3D-C1

NERSC: Woo-Sun Yang, Nate Ferraro, Jin Chen, Pieter Ghysels, Yang Liu
Intel: Ruchira Sasanka

- **Yang Liu's report:**

1. **What was your approach and/or technique?**
   - **Improve SuperLU_DIST in Petscs called from M3D-C1**
2. **What obstacles did you encounter?**
   - **The bottleneck in SuperLU_DIST (in M3D-C1) are the numerical factorization and solve phases**
3. **What are the before and after results (performance, etc.)?**
   - **Factorization phase of SuperLU_DIST improves significantly**
   - **Build SuperLU_DIST with craype-mic-knl leads to 10% improvement in Petsc**
   - **The OMP scaling of PCSetUp in Petsc shows no improvement**
4. **What key learnings did you experience?**
   - **Learn and understand Vtune functionalities**
5. **What additional work is still needed?**
   - **Understand how Petsc integrates SuperLU_DIST**
   - **Develop threaded version of the solve phase of SuperLU_DIST**

- Factorization phase of SuperLU_DIST for A22.mtx
- 8 KNL ndoes, 16 MPI ranks

	Before		After	
#thread	factor time	speedup	factor time	speedup
1	207.96	1	139.21	1
2	119.15	1.745363	85.29	1.632196
4	72.05	2.886329	47.55	2.927655
8	50.52	4.11639	29.85	4.663652
16	43.58	4.771914	22.63	6.151569
32	40.54	5.129748	17.56	7.927677

- Integrating SuperLU_DIST into Petsc
- 1 KNL node, 32 MPI ranks

	Before		After	
#thread	PCSetUp	Speedup	PCSetUp	Speedup
1	22.50	1.00	19.99	1.00
2	18.39	1.22	16.06	1.24
4	19.30	1.17	17.10	1.17
8	31.52	0.71	28.66	0.70

Future work – next 0.5 year

SuperLU:

1. **Improve triangular solve.**

   As preconditioner, factorization needs only once, but each iterative step needs a triangular solve. Several techniques will be explored: asynchronous tree broadcast, selective inversion.
2. **Improve factorization with 3D algorithm: replicate data structure, while reducing communication.**
3. **Suggestion for M3D-C1 configuration: use more than one 2D plane as diagonal block.**

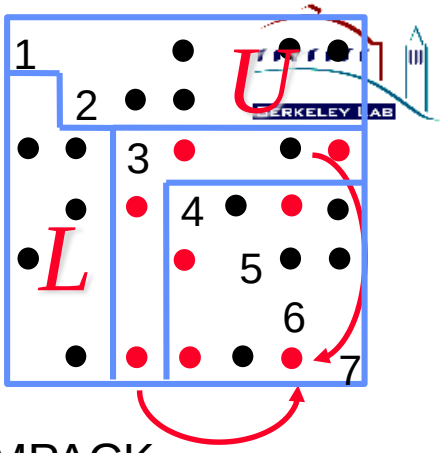
STRUMPACK:

- **More detailed performance profiling to identify bottlenecks, understand various solver/preconditioner configurations.**
  - **PETSc multithreading is lacking??**

Optimization: with Sam and postdoc

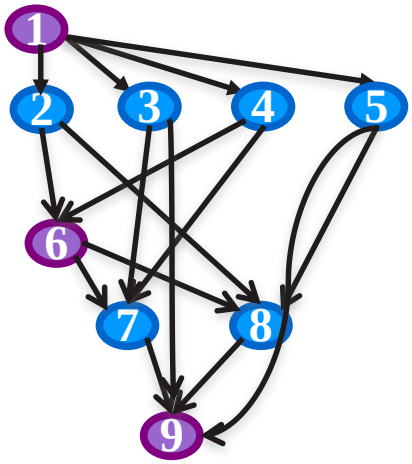
- **Autotuning of the solvers parameter space.**

Sparse factorization for linear systems

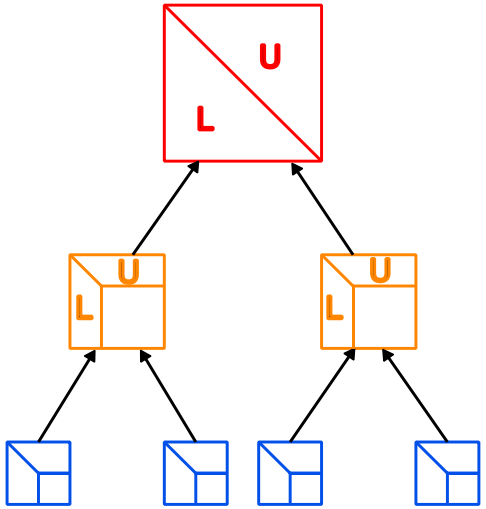


Two algorithm variants

DAG based
Supernodal: SuperLU



Tree based
Multifrontal: STRUMPACK



$S^{(j)} \leftarrow ((S^{(j)} - D^{(k1)}) - D^{(k2)}) - \ldots$

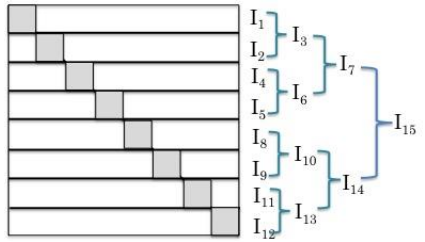
$S^{(j)} \leftarrow S^{(j)} - ((D^{(k1)}) + D^{(k2)}) + \ldots)$

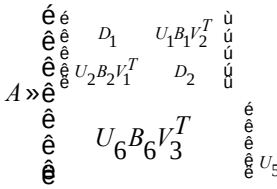
STRUMPACK "inexact" direct solver

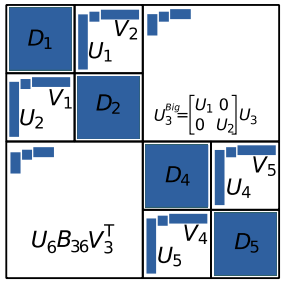
Developers: P. Ghysels, C. Gorman, F.-H. Rouet, XL
Web site contains code and papers:
http://portal.nersc.gov/project/sparse/strumpack/

- Baseline is a sparse multifrontal direct solver.
- In addition to structural sparsity, further apply data-sparsity with low-rank compression:
  - O(N logN) flops, O(N) memory for 3D elliptic PDEs.
- Hierarchical matrix algebra generalizes Fast Multipole
  - Diagonal block **("near field")** exact; off-diagonal block **("far field")** approximated via low-rank compression.
  - Hierarchically semi-separable (HSS), HODLR, etc.
  - **Nested bases + randomized sampling** to achieve linear scaling.
- Applications: PDEs, BEM methods, integral equations, machine learning, and structured matrices such as Toeplitz, Cauchy matrices.

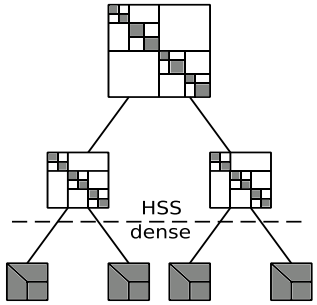
Cluster tree







Sparse embedding in MF solver



Overview of STRUMPACK status and plan

- Provides flexibility with inexact factorization via tolerance-controlled compression (save FLOPS and memory)

Status:

- 5x faster than dense LU (ACM TOMS 2016)
- 7x faster than traditional sparse MF solver (SIAM SISC 2016)

Future tasks:

1. STRUMPACK already in PETSc, will incorporate in Trillinos soon.
2. Tested Jin's recent matrices, found some inefficiency in the ordering phase, we fixed it.
3. Code is very new, less mature than SuperLU, will do lots of algorithm optimization, architecture-oriended code optimization …

HSS approximation error vs. drop tolerance

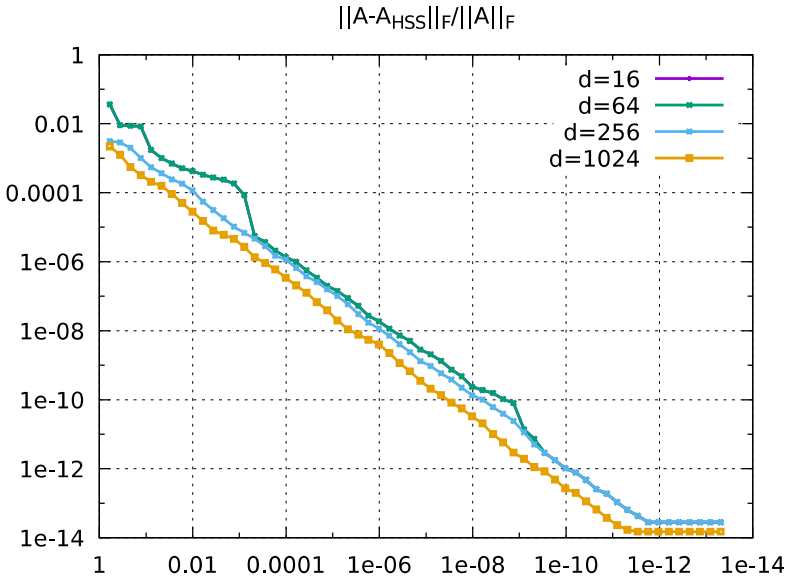
Randomized sampling to reveal rank

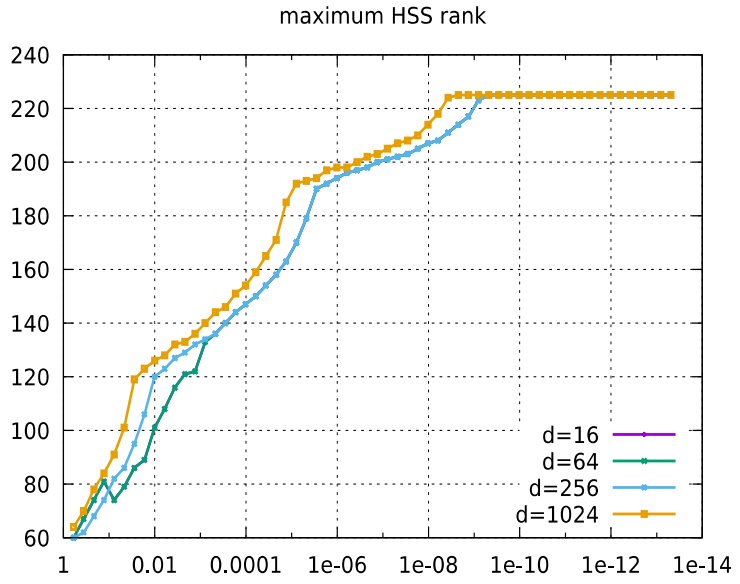
1. Pick random matrix $\Omega_{nx(k+p)}$, k target rank, p small, e.g. 10
2. Sample matrix S = A Ω, with slight oversampling p
3. Compute Q =ON-basis(S) via rank-revealing QR

Accurate with high probability. [N. Halko,P.G. Martinsson, J.A. Tropp, 2011]

➔ Adaptive sampling is essential for robustness:

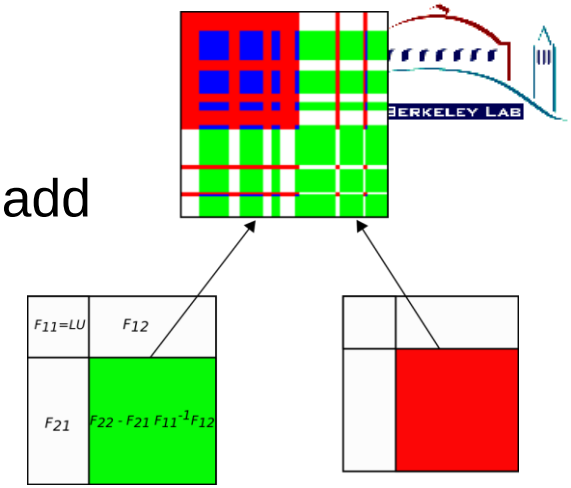
Increase d, number of columns in Ω, until error small



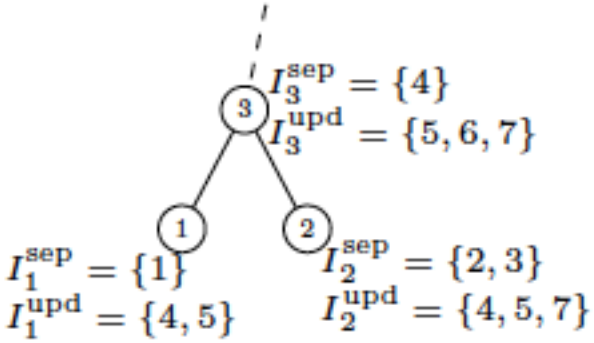


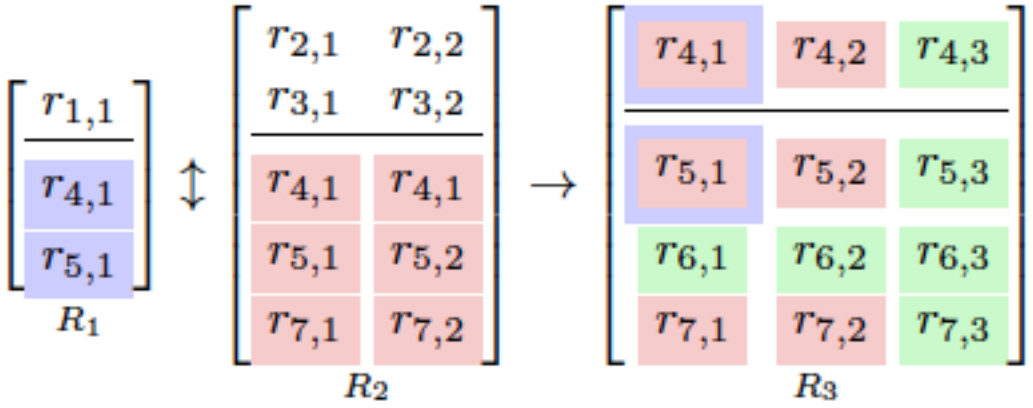
RS Simplifies “extend-add” in MF+HSS

Traditional extend-add



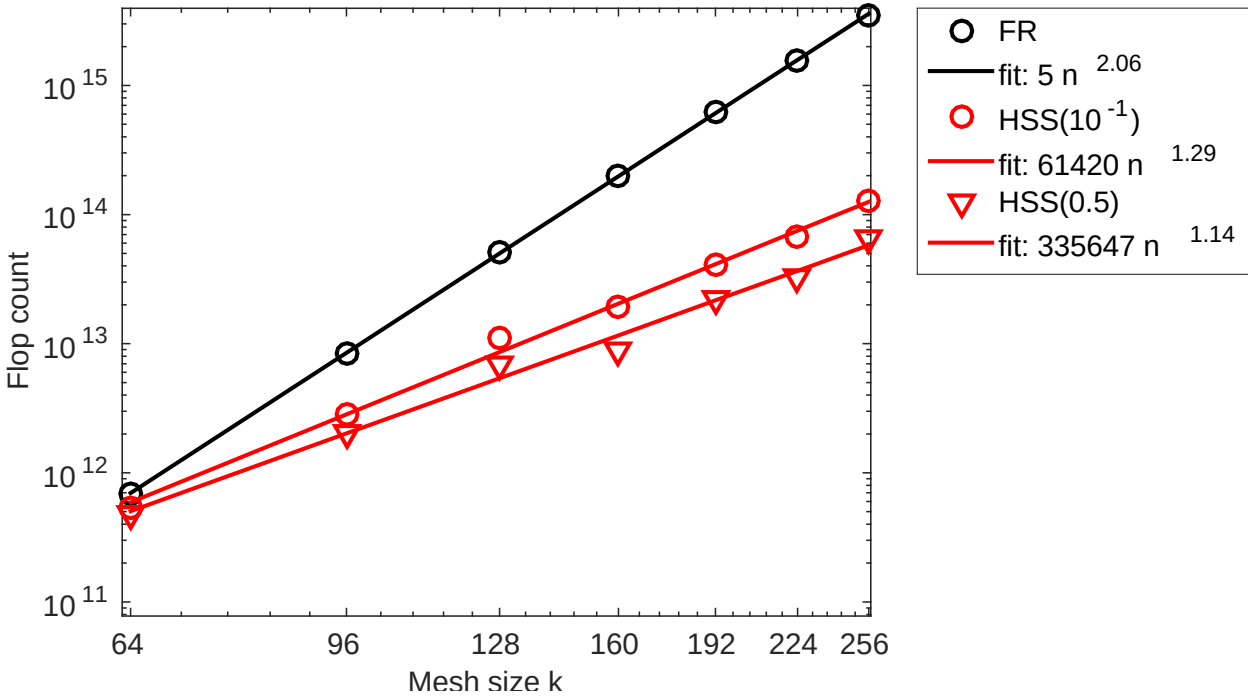
New “extend-merge” of sample matrices



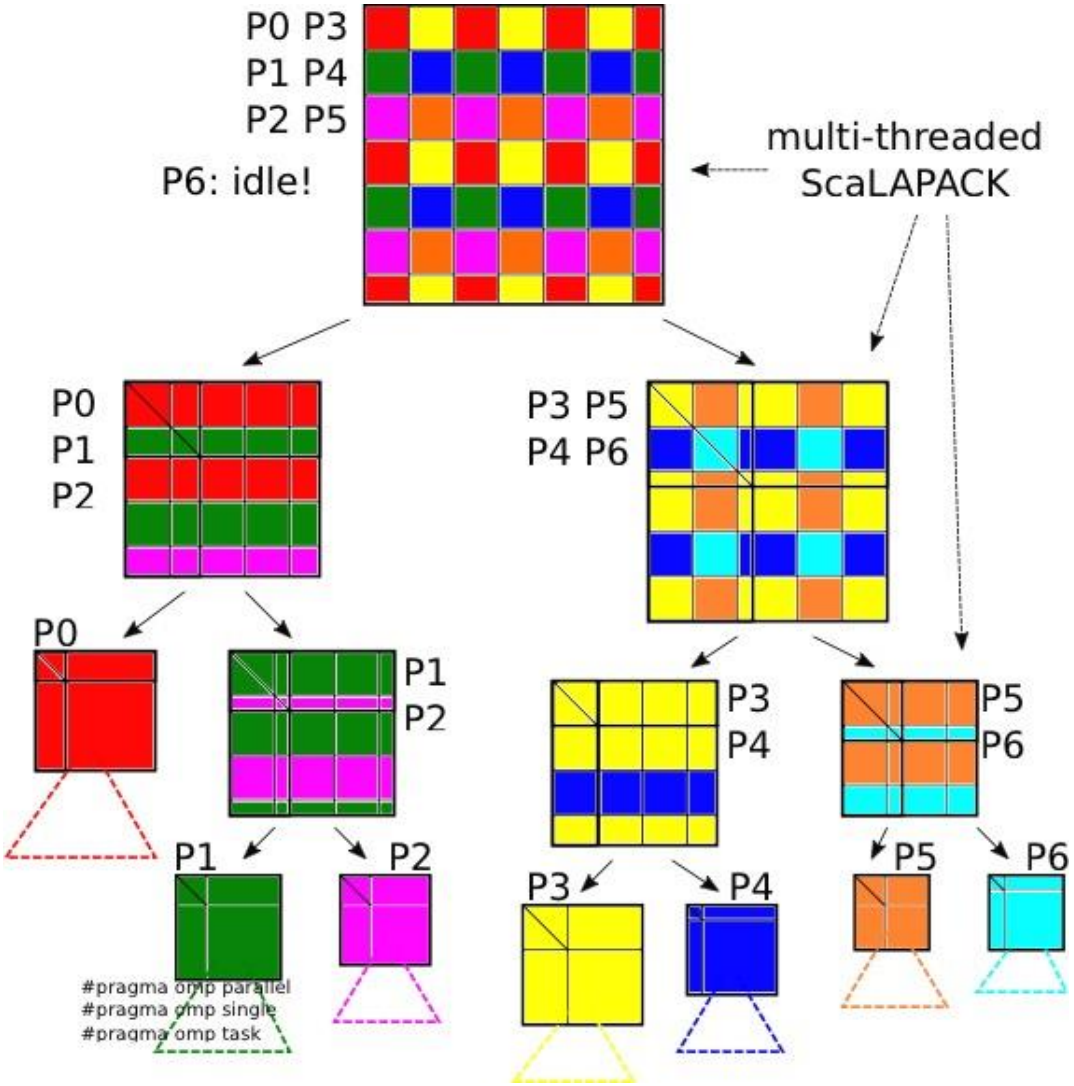


Algorithm scaling for 3D Poisson

- Theory predicts $O(n^{4/3} \log n)$ flops for compression.
- HSS ranks grow with mesh size ~ $n^{1/3} = k$
- Use as a preconditioner with aggressive compression.



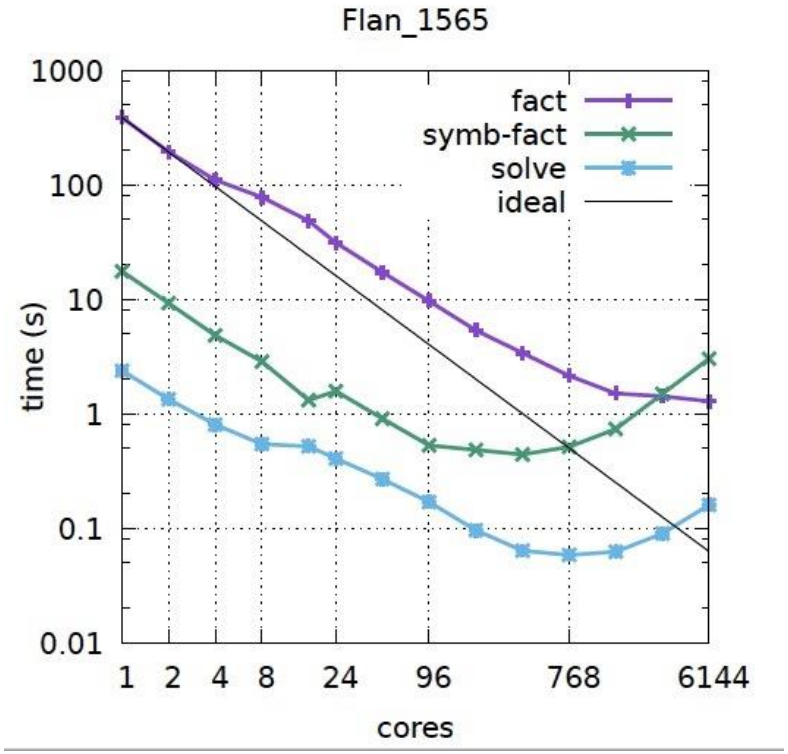
Tree-based parallelization



Performance scaling

Matrix from SuiteSparse Matrix Collection:

Flan_1565 (N= 1,564,794, NNZ = 114,165,372)



- Flat MPI on nodes with 2 12-core Intel Ivy Bridge, 64GB (NERSC Edison)
- Fill-reducing reordering (ParMetis) has poor scalability, quality decreases