Factorization Based Sparse Solvers and Preconditioners for Exascale

X. Sherry Li
xsli@lbl.gov
Lawrence Berkeley National Laboratory

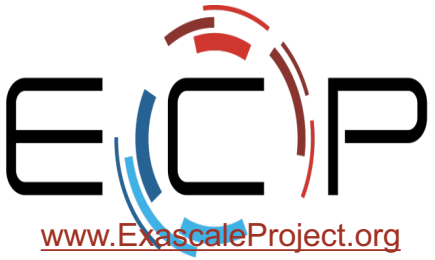
Sparse Days, CERFACS, September 6-8, 2017

Happy Birthday Iain !

Acknowledgments

This research was supported by the Exascale Computing Project (http://www.exascaleproject.org), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.

Project Number: 17-SC-20-SC



ECP system and funded projects

- **Capable exascale ecosystem** in 2021
  - Delivers 50 × the performance of today's 20 PF systems, supporting applications that deliver high- fidelity solutions in less time and address problems of greater complexity
  - Operates in a power envelope of 20–30 MW
  - Is sufficiently resilient (perceived fault rate: ≤ 1/week)
  - Includes a software stack that supports a broad spectrum of applications and workloads

- Funded projects
  - Applications: 25
  - Co-design centers: 5
  - Software technology: 66
  - Hardware technology: working with vendors

“Factorization based sparse solvers and preconditioners”

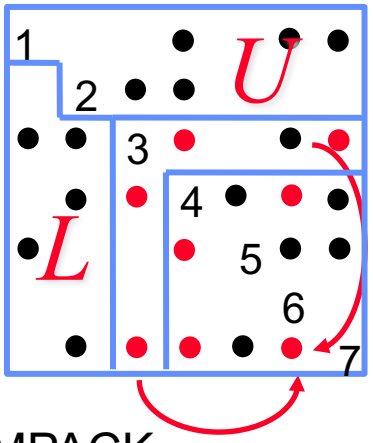
- “Software Technology” → “Mathematical and Scientific Libraries and Framework”
  - STRUMPACK : http://portal.nersc.gov/project/sparse/strumpack/
  - SuperLU : http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

- Collaborating with other ECP ST projects:
  - xSDK4ECP (Mike Heroux, Lois McInnes, et al.)
    - Interoperability among numerical libraries, easy plug-in in applications.
  - Pagoda Project (Scott Baden, et al.)
    - UPC++ programming to speed up fine grained, asynchronous communication and computation.
  - Autotuning Compiler Technology (Mary Hall, et al.)
    - Search for best parameters setting to achieve optimal performance, dependent on applications and machines.

Goals of math libraries

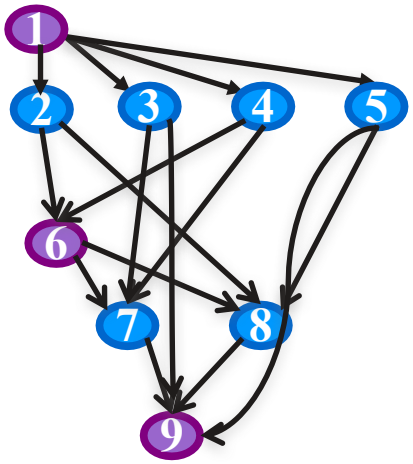
- Explore new algorithms that require lower arithmetic complexity, communication and synchronization, faster convergence rate
  - STRUMPACK: “inexact” direct solver, preconditioner, based on hierarchical low rank structures: HSS, HODLR, etc.
  - SuperLU: new 3D algorithm to reduce communication
- Refactor existing codes and implement new codes for current and next-generation machines (exascale in 2021)
  - Fully exploit manycore node architectures
    - Vectorization, multithreading, …
    - GPU accelerator
  - Reduce communication and synchronization

Sparse factorization for linear systems

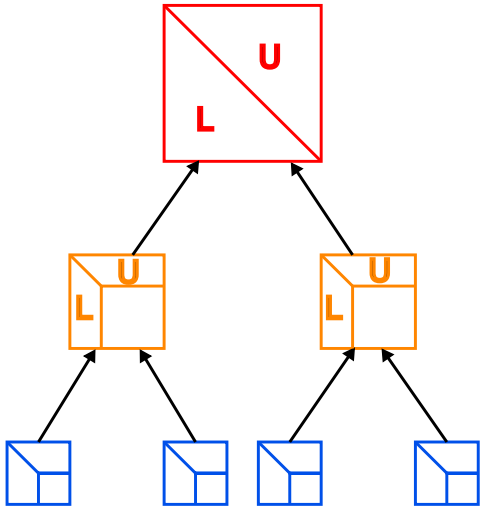


Two algorithm variants

DAG based
Supernodal: SuperLU



Tree based
Multifrontal: STRUMPACK



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

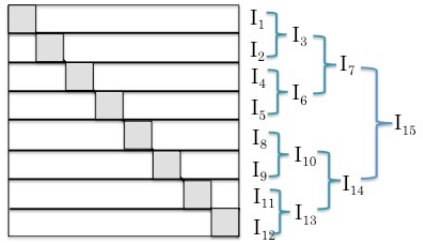
$S^{(j)} \leftarrow S^{(j)} - ((D^{(k1)} + D^{(k2)}) + \ldots)$

STRUMPACK "inexact" direct solver

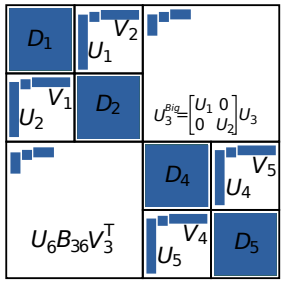
P. Ghysels, G. Chavez, C. Gorman, F.-H. Rouet, XL

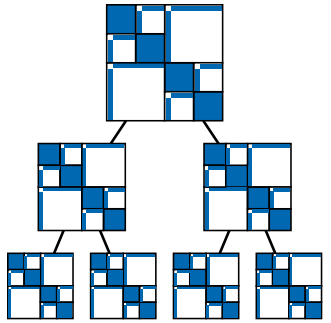
- Baseline is a sparse multifrontal direct solver.
- In addition to structural sparsity, further apply data-sparsity with low-rank compression:
  - O(N logN) flops, O(N) memory for 3D elliptic PDEs.
- Hierarchical matrix algebra generalizes Fast Multipole
  - Diagonal block **("near field")** exact; 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



$$A \approx \begin{bmatrix} \begin{bmatrix} D_1 & U_1 B_1 V_2^T \\ U_2 B_2 V_1^T & D_2 \end{bmatrix} & U_3 B_3 V_6^T \\ U_6 B_6 V_3^T & \begin{bmatrix} D_4 & U_4 B_4 V_5^T \\ U_5 B_5 V_4^T & D_5 \end{bmatrix} \end{bmatrix}$$





Randomized sampling to reveal rank

Approximate range of A:

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

Accuracy: [Halko, Martinsson, Tropp, '11]

- On average: $\mathbb{E}(\|A - QQ^*A\|) = \left(1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{\min\{m,n\}}\right)\sigma_{k+1}$
- Probabilistic bound: with probability $\geq 1 - 3 \cdot 10^{-p}$,

$$\|A - QQ^*A\| \leq (1 + 9\sqrt{k+p}\sqrt{\min\{m,n\}})\,\sigma_{k+1}$$

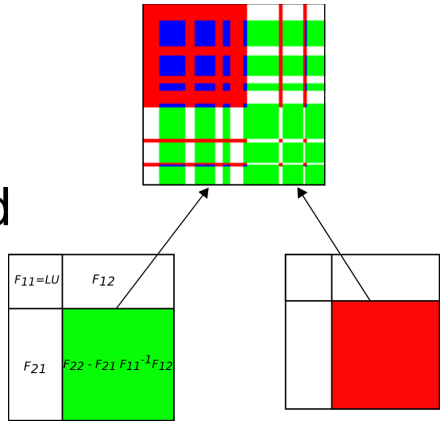
(in 2-norm)

Benefits:

- Matrix-free, only need AΩ, can be faster: FMM, FFT, HSS, …
- When embedded in frontal solver, simplies 'extend-add'

RS Simplifies extend-add in MF+HSS

Traditional extend-add

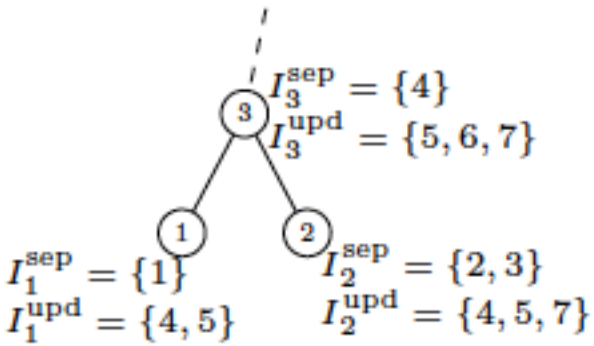


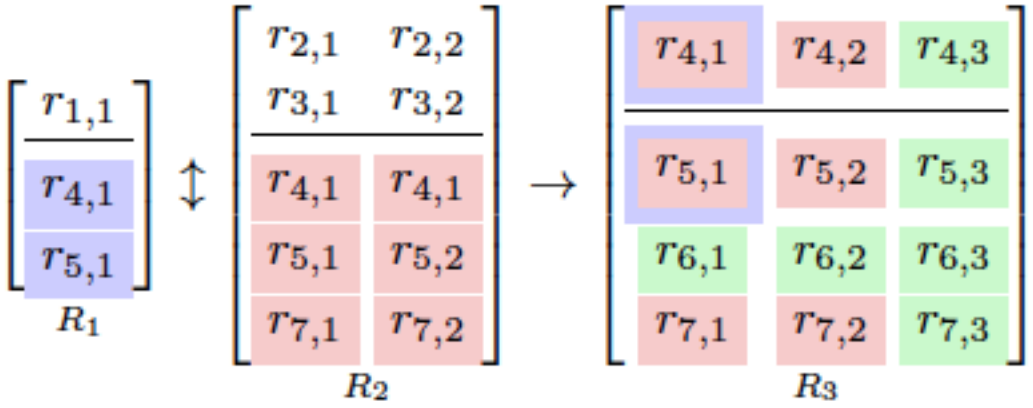
New: need to sample frontal matrix, no need to form dense F

$F\,\Omega$ = (A[.] + HSS + low-rank) Ω

= A[.]$\cdot\,\Omega$ + HSS$\cdot\,\Omega$ + low-rank$\cdot\,\Omega$

"extend-merge" of random/sample matrices: extend is done only for rows





HSS compression via RS

[Martinsson '11, Xia '13]

- Ω random matrix, with d = r + p columns
  - r is estimated maximum rank, p is oversampling parameter
- Sampling of matrix A
  - $S^r = A\Omega$, columns of S^r span the column space of A
  - $S^c = A^*\Omega$, columns of S^r span the row space of A
- Only sample off-diagonal blocks (Hankel blocks):

  Block diagonal matrix at level ℓ: $D^{(\ell)} = diag(D_{\tau_1}, D_{\tau_2}, \ldots D_{\tau_q})$

  $$S^{(\ell)} = (A - D^{(\ell)})\Omega = S - D^{(\ell)}\Omega$$

- Practical issues:
  - Need ε-rank: $\|A - QQ^*A\| \leq \varepsilon$
  - Flat singular spectrum

Adaptive sampling is essential for robustness

Increase sample size d, build Q incrementally

$[\ S_1\ \ S_2\ \ S_3\ \ldots\ ]$

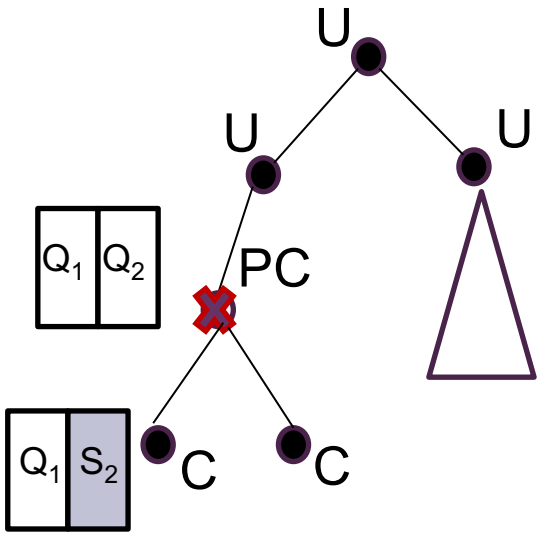
$Q \leftarrow [];\ S_1 \leftarrow A\,\Omega_1;\ i \leftarrow 1;$
WHILE (error large) {
  $Q_i \leftarrow QR(S_i)$ // Orthogonalize within current block
  $Q \leftarrow [\ Q\ \ Q_i\ ]$
  $S_{i+1} \leftarrow A\,\Omega_{i+1}$ // New samples
  $S_{i+1} \leftarrow (I - QQ^*)\,S_{i+1}$ // Orthogonalize against previous Q
  Compute error;
  $i \leftarrow i+1$
}

Adaptive sampling in HSS tree

Recall …

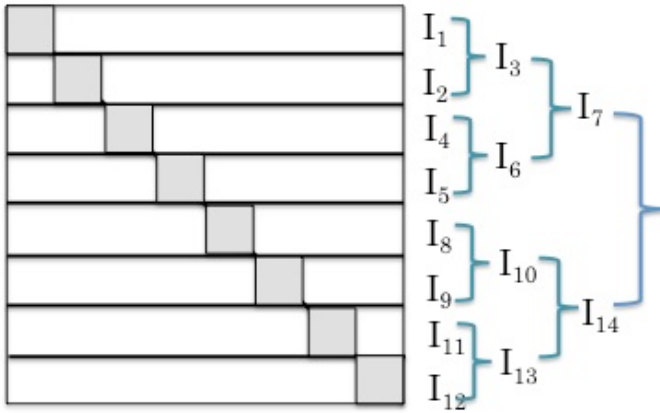
- Only have $S = A\Omega$
- At level ℓ:

$$S^{(\ell)} = (A - D^{(\ell)})\Omega = S - D^{(\ell)}\Omega$$

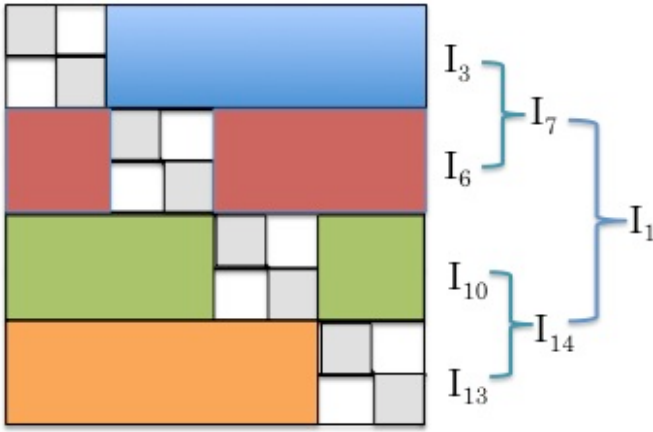


Node state:
U – untouched
C – compressed
PC – partially compressed

level L



level L-1



Adaptive sampling – cont.

- Stochastic error estimation: for random $x, S = Ax$

$$\mathbb{E}(\|S\|_2^2) = \frac{1}{n}\|A\|_F^2$$

$$\frac{\sqrt{\mathbb{E}(\|(I - QQ^*)S\|_2^2)}}{\sqrt{\mathbb{E}(\|S\|_2^2)}} = \frac{\|(I - QQ^*)A\|_F}{\|A\|_F}$$

- Stopping criteria: [Q_1 S_2]

$$\frac{\sqrt{mean_k(\|(I - Q_1{Q_1}^*)S_2(:,k)\|_2^2)}}{\sqrt{mean_k\|S_2(:,k)\|_2^2}} = \frac{\|(I - Q_1{Q_1}^*)S_2\|_F}{\|S_2\|_F} \leq \varepsilon$$

Cost: one reduction to compute norms of the sample vectors.

Adaptivity example – decay singular value

$A = \alpha I + UDV^*,\ U,V\ rank = 120,\ D_{k,k} = 2^{-24(k-1)/r}$

$\varepsilon_r \backslash \varepsilon_a$	1e-2	1e-4	1e-6	1e-8	1e-10	1e-12	1e-14
1e-1	24; 64	24; 64	24; 64	24; 64	24; 64	24; 64	24; 64
1e-2	42; 80	42; 80	42; 80	42; 80	42; 80	42; 80	42; 80
1e-3	59; 96	59; 96	59; 96	59; 96	59; 96	59; 96	59; 96
1e-4	77; 112	77; 112	77; 112	77; 112	77; 112	77; 112	77; 112
1e-5	94; 128	94; 128	94; 128	94; 128	94; 128	94; 128	94; 128
1e-6	111; 128	111; 128	111; 128	111; 128	111; 128	111; 128	111; 128
1e-7	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-8	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128

Table 10: Size: 100000; Alpha: 100000; Rank: 120; Decay-value: 24; d-start: 16; d-add: 16. These numbers are "(Computed HSS Rank); (Random Samples Used)". Here, we are using $\mathrm{fl}\left[\left(I - Q_1 Q_1^*\right) S_2\right] = \left(I - Q_1 Q_1^*\right)^2 S_2$, with the products computed intelligently.

Adaptivity example – constant singular value

$A = \alpha I + UDV^*,\ U,V\ rank = 120,\ D_{k,k}$=1

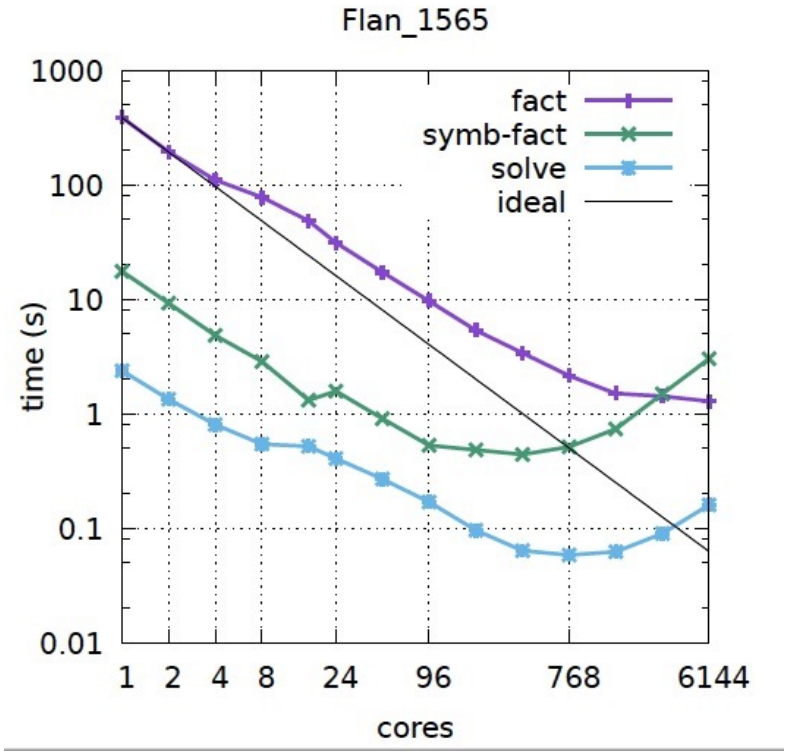
$\varepsilon_r \backslash \varepsilon_a$	1e-2	1e-4	1e-6	1e-8	1e-10	1e-12	1e-14
1e-1	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-2	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-3	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-4	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-5	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128	120; 128
1e-6	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768
1e-7	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768
1e-8	120; 128	120; 128	120; 128	120; 128	120; 768	120; 768	120; 768

Table 8: Size: 100000; Alpha: 100000; Rank: 120; Decay-value: 0; d-start: 16; d-add: 16. These numbers are "(Computed HSS Rank); (Random Samples Used)". Here, we are using $\mathrm{fl}\left[\left(I - Q_1 Q_1^*\right) S_2\right] = \left(I - Q_1 Q_1^*\right)^2 S_2$, with the products computed intelligently.

Performance scaling

Matrix from SuiteSparse Matrix Collection:

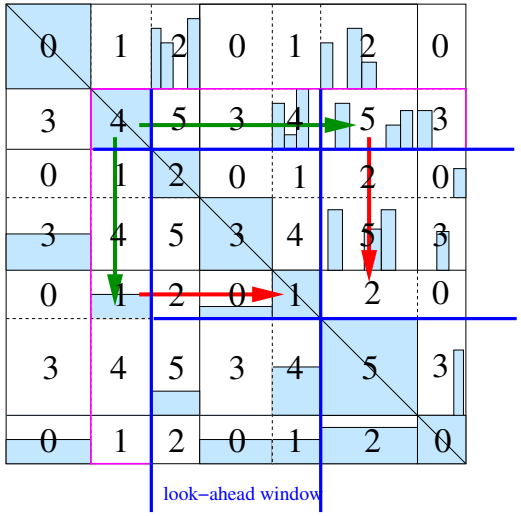
Flan_1565 (N= 1,564,794, NNZ = 114,165,372)

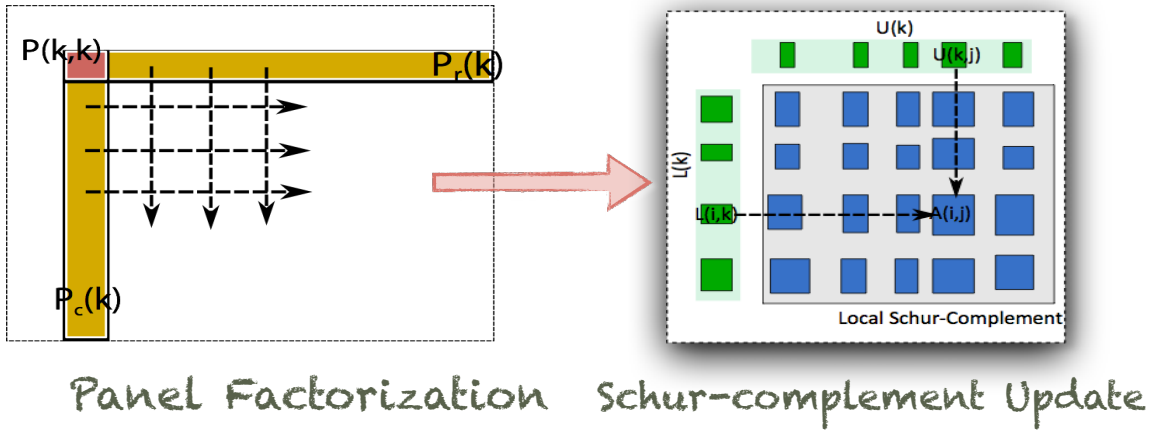


- Flat MPI on nodes with 2 12-core Intel Ivy Bridge, 64GB (NERSC Edison)
- Fill-reducing reordering (ParMetis) has poor scalability, quality decreases

SuperLU direct solver – communication pattern

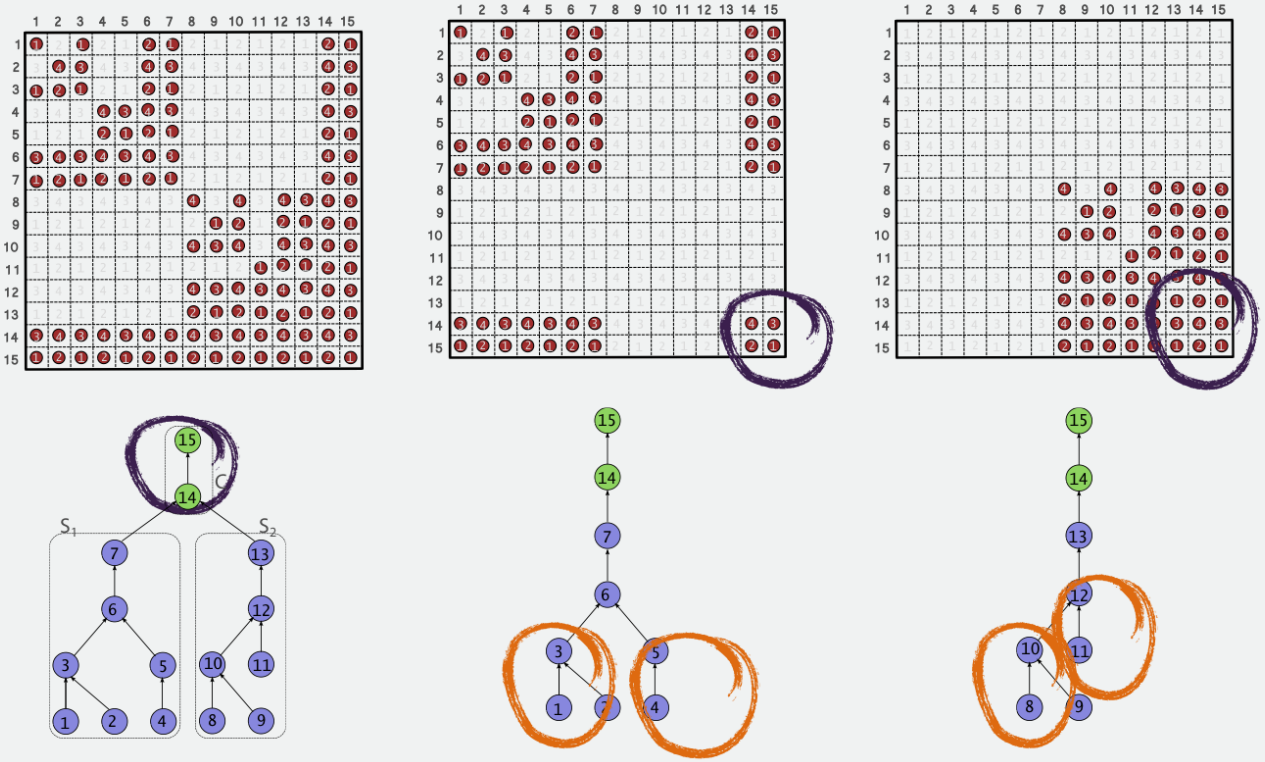
XL, J. Demmel, J. Gilbert, L. Grigori, Y. Liu, P. Sao, Meiyue Shao, I. Yamazaki





- Graph at step k+1 differs from step k
- Panel factorization on critical path

3D Sparse LU: Data Distribution

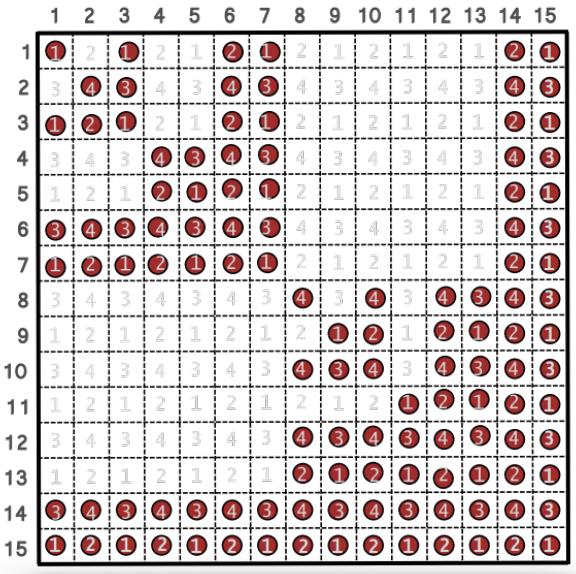


Final reduction: Schur complement of common ancestor

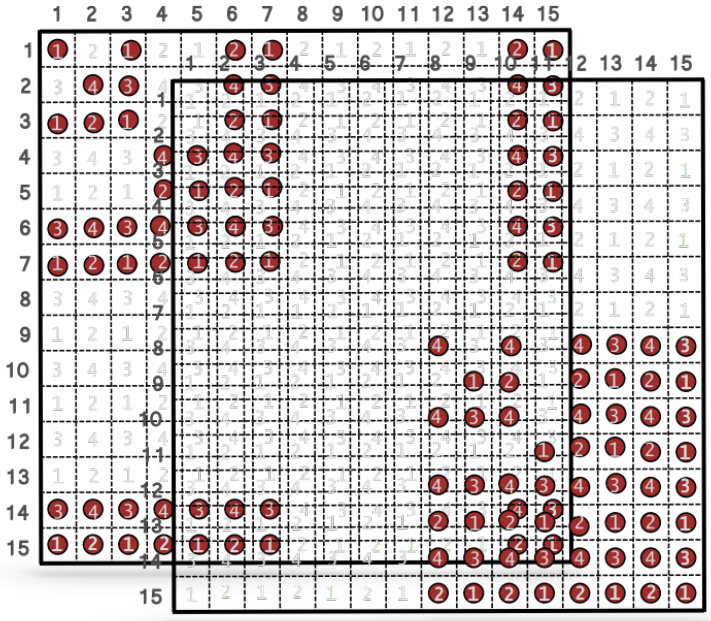
C = C1 + C2 + …

[Piyush Sao's thesis, Georgia Tech., 2017]

3D Sparse LU: cost of communication



Grid size	Communication volume
$\sqrt{P} \times \sqrt{P}$	$V_{2D}(P) = v_p$
$\sqrt{2P} \times \sqrt{2P}$	$V_{2D}(2P) = \frac{v_p}{\sqrt{2}}$



Grid size	Communication volume
$2 \times \sqrt{P} \times \sqrt{P}$	$V_{3D}(2P) = \frac{v_p}{2}$

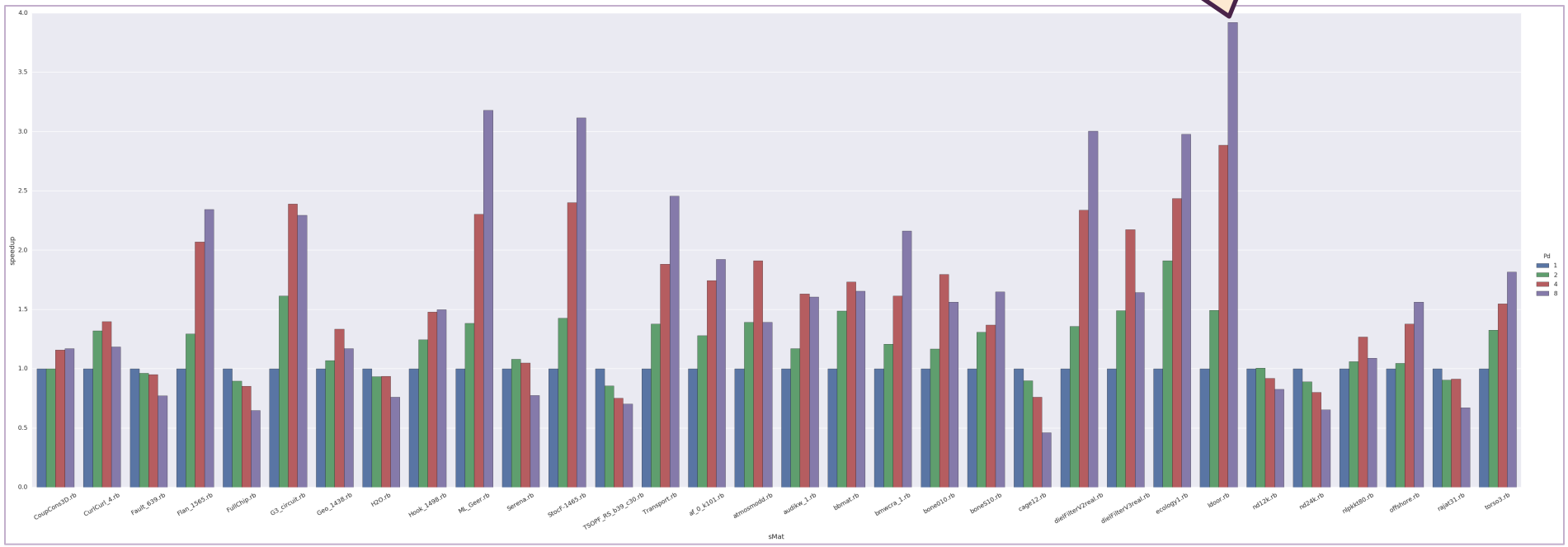
3D performance

32 matrices
Cori at NERSC:

- 16 Haswell nodes = 512 cores
- 128 MPI ranks x 4 OMP threads

Replication factor c = 1, 2, 4, 8

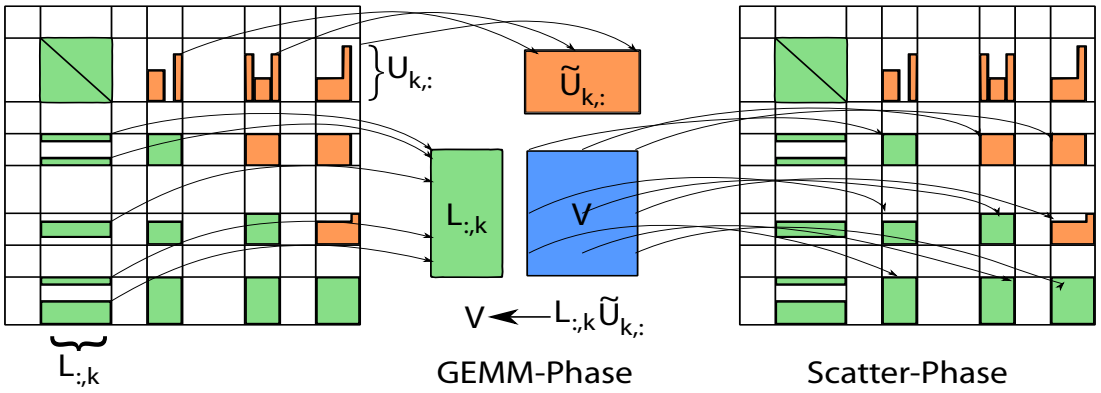
ldoor: nearly 4x faster, c = 8 (4x4, 8)



Local computation

Schur complement update on each MPI rank

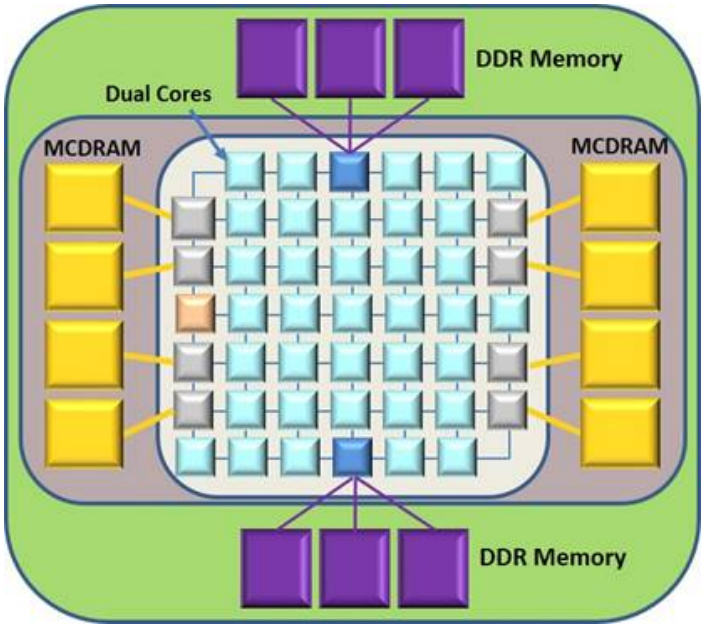
Loop through N steps: (Gaussian Elimination)
FOR (k = 1, N) {
1) **Gather** sparse blocks A(:, k) and A(k,:) into dense work[]
2) Call dense **GEMM** on work[]
3) **Scatter** work[] into remaining sparse blocks
}



Intel Xeon Phi: Knights Landing

Cray XC40 supercomputer at NERSC:

- 9688 KNL nodes: single socket
- 2388 Haswell nodes: 2 sockets X 16 cores



KNL node

- 72 cores @ 1.3 GHz, self hosted
- 2 cores form a tile
- 4 hardware threads per core (272 threads)
- 2 512-bit (8 doubles) vector units (SIMD)

Memory hierarchy

- L1 cache per core, 64 KB
- L2 cache per tile (2 cores share), 1MB
- 16 GB MCDRAM, >400 GB/s peak bandwidth
- 96 GB DDR4, 102 GB/s peak bandwidth

SuperLU optimization on Cori KNL node (1/2)

Work with Sam Williams, Jack Deslippe, Steve Leak, Thanh Phung

- Replace small independent single-threaded MKL GEMMs by large multithreaded MKL GEMMs: **15-20% faster.**
- 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];
    }
```

SuperLU optimization on Cori KNL node (2/2)

Reduce cache misses

- TLB (Translation Look-aside Buffer): a small cache for mapping virtual address to physical address
  - Large page requires smaller number of TLB entries

- Alignment during malloc
  - Page-aligned for large arrays → reduce TLB read frequency
  - CacheLine-aligned malloc for threads-shared data structures → reduce L1 read frequency, avoid false sharing

Roofline model (S. Williams)

basic concept

- Is the code computation-bound or memory-bound?
- Synthesize communication, computation, and locality into a single visually-intuitive performance figure using bound analysis
  - Assume perfect overlap computation and communication w/ DRAM
  - Arithmetic Intensity (AI) is computed based on DRAM traffic

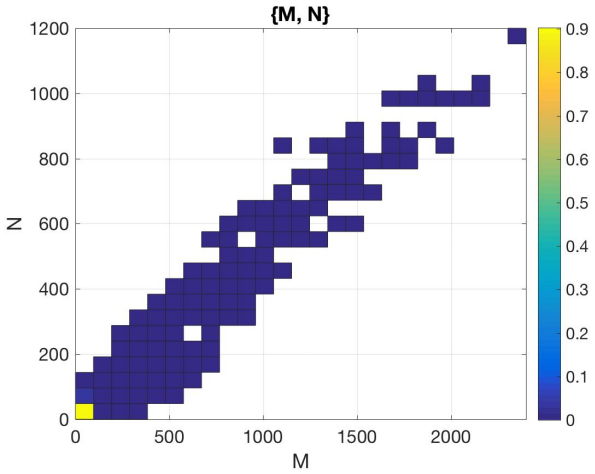
  E.g.: DGEMM AI = 2*M*N*K / (M*K + K*N + 2*M*N) / 8

- Time is the maximum of the time required to transfer the data and the time required to perform the floating point operations.

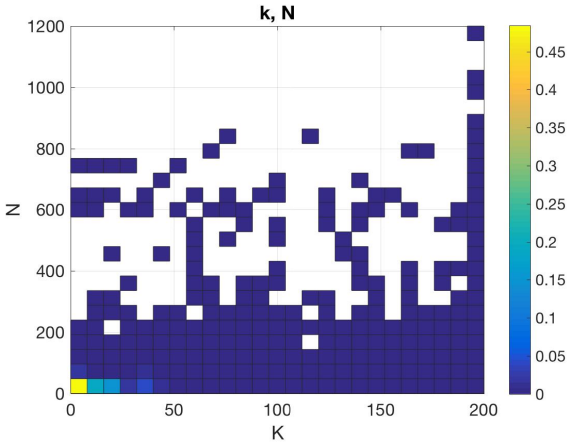
$$\text{Attainable GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI} * \text{Peak GB/s} \end{cases}$$

GEMM: non-uniform block size, non-square, many small

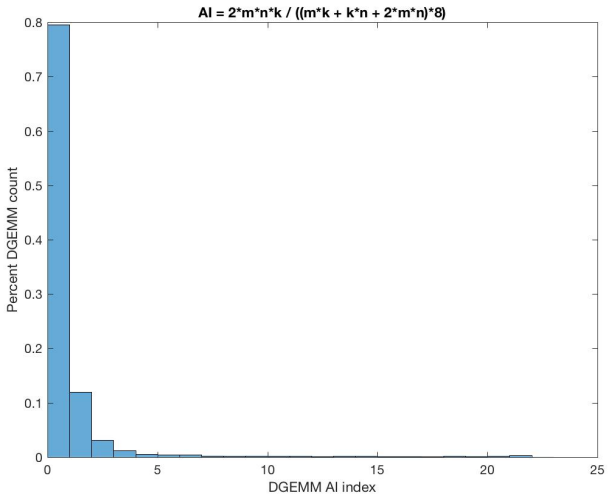
GEMM {m, n} dimensions



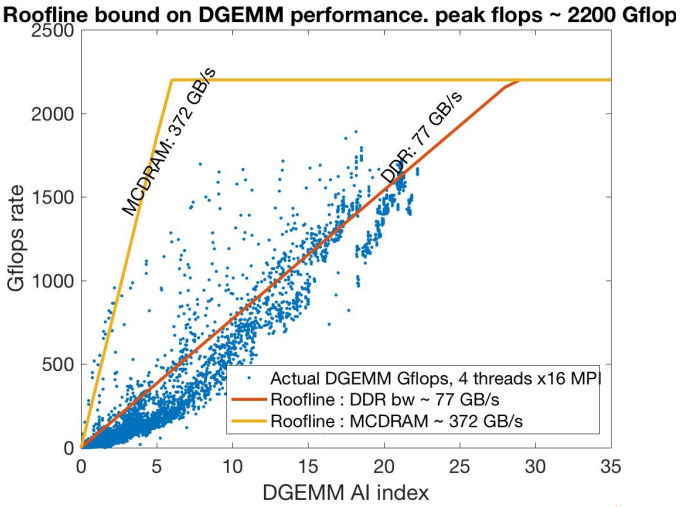
GEMM {k, n} dimensions



Arithmetic Intensity

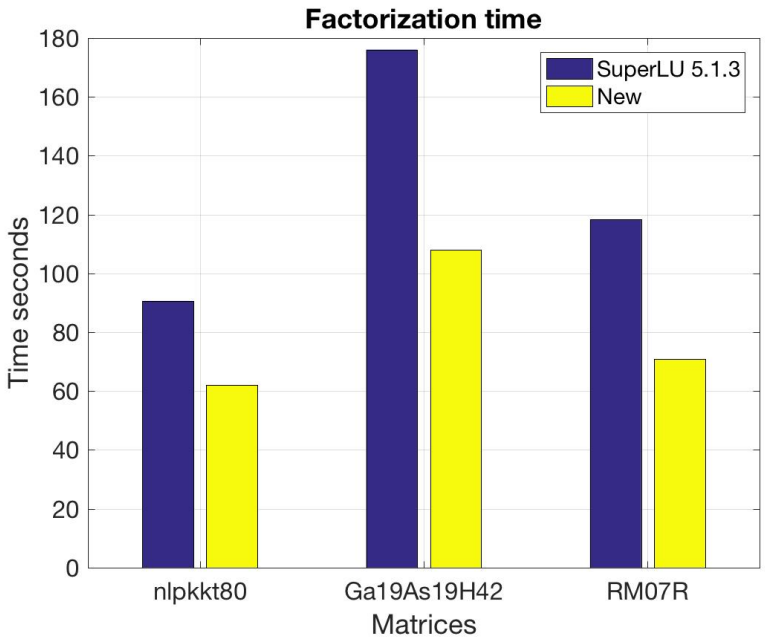


DGEMMs performance profile



Single node improvement

- up to 68% faster



nlpttk80, n = 1.1M, nnz = 28M
Ga19As19H42, n = 1.3M, nnz = 8.8M
RM07R, n = 0.3M, nnz = 37.5M

Summary

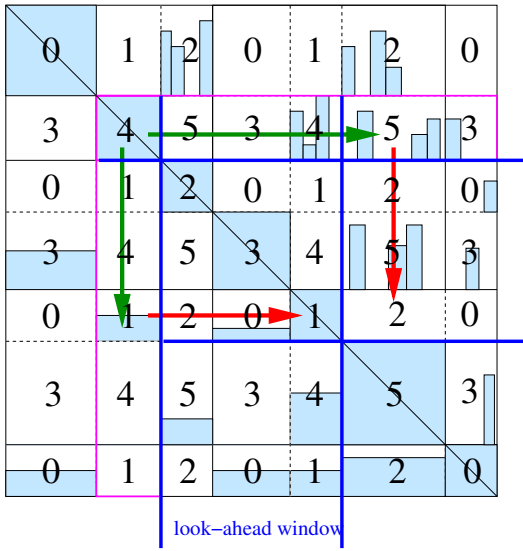
- Explore new algorithms that require lower arithmetic complexity, communication, synchronization
  - STRUMPACK: “inexact” direct solver, preconditioner, based on hierarchical low rank structures: HSS, HODLR, etc.
  - SuperLU: new 3D algorithm to reduce communication.

- Refactor existing codes and implement new codes for current and next-generation machines (exascale in a few years)
  - Fully exploit manycore node architectures
    - Vectorization, multithreading, …
    - GPU accelerator
  - Reduce communication and synchronization

- Quaterly release of software

THANK YOU

SuperLU tunable parameters



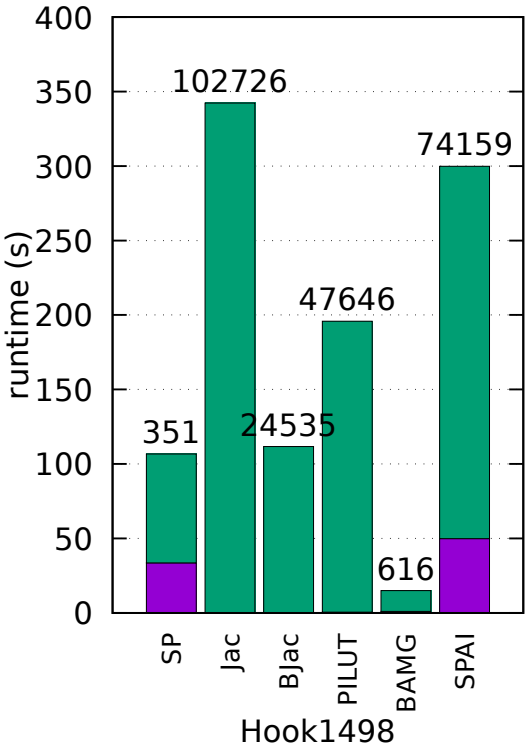
See the inquiry function **SRC/sp_ienv.c**

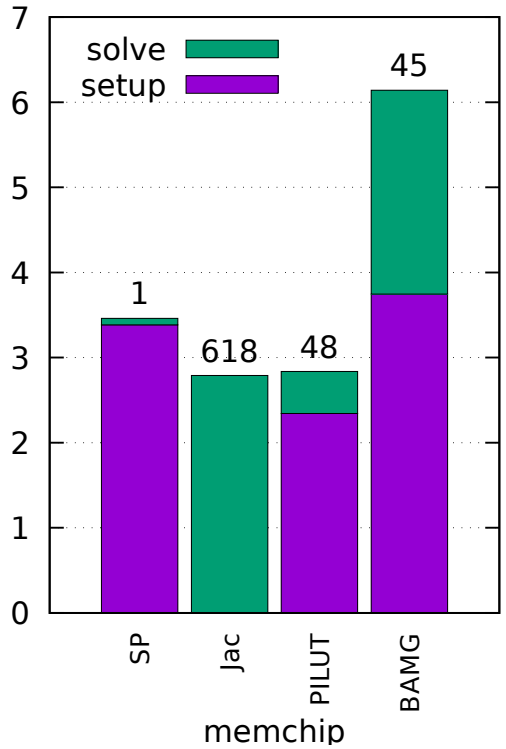
- Maximum supernode size: (env NSUP, range: 50 - 300)
  - Larger → bigger k-dimension in GEMM, but less inter-task parallelism, longer critical path.
- Relaxed supernode size: (env NREL, range: 10-40)
  - Larger → better GEMM flop rate, but more zeros stored, more flops.
- Pipeline depth. (options->num_lookaheads, range: 5-20)
- Reordering options: ParMetis, Min-degree, (options->ColPerm, {0, 1, 2, 3, 4, 5}, see colperm_t in superlu_enum_consts.h)
  - Affects fill-ins, block size, flop count.
- # threads/process count. (env OMP_NUM_THREADS)

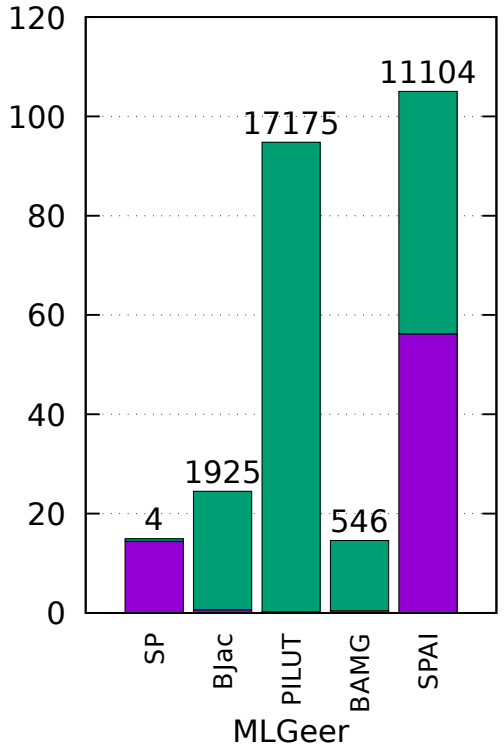
Cori/KNL timing: Matrix nlpttk80, n = 1.1M, nnz = 28M

	1 node = 64 cores		8 nodes = 512 cores		
MPI, Threads	32 p, 2 t	16 p, 4 t	256 p, 2 t	128 p, 4 t	64 p, 8 t
Panel factorization	4.87 1.61, 3.26	6.48 3.44, 3.04	1.60 1.21, .39	1.31 .82, .49	2.59 2.01, .58
Schur Gather GEMM Scatter	43.61 2.64 22.15 (1.3 TF) 18.01	45.24 2.78 22.41 18.79	7.80 1.17 3.49 2.60	8.43 1.31 3.60 2.77	9.75 1.63 3.95 3.34
Total factorization time (% un-timed comm.)	64.44 (25%)	71.34 (27%)	31.94 (71%)	38.03 (74%)	28.47 (57%)

STRUMPACK as preconditioner: Setup & Solve







- For memchip, solver acts as direct method (small frontal matrices)
- AMG very efficient for many PDE based systems

Strong scaling

- Communication bottleneck
- Lacking parallelism in 2D process grid