Factorization Based Sparse Solvers and Preconditioners for Exascale

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

SIAM Annual Meeting, July 10-14, 2017

Exascale Computing Project (ECP)

https://exascaleproject.org

This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.

Exascale refers to a class of high-performance computing systems capable of performance on mission critical applications that is 50 times more powerful than the nation's most powerful systems in use today, and "capable" exascale refers to systems not measured by benchmarks alone but rather ones that can solve problems at this performance level in a power envelope of 20-30 MW, are sufficiently resilient that user intervention due to hardware or system faults is on the order of a week on average, and have a software stack that meets the needs of a broad spectrum of applications and workloads.

“Factorization based sparse solvers and preconditioners”

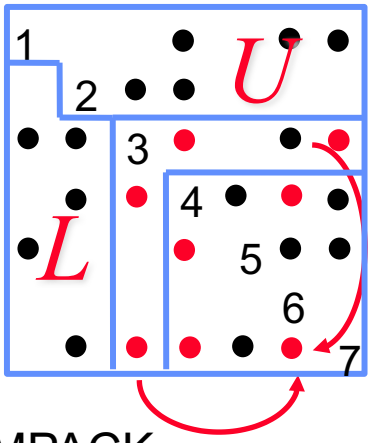
- “Software Technology” → “Mathematical and Scientific Libraries and Framework”
  - STRUMPACK : http://portal.nersc.gov/project/sparse/strumpack/
  - SuperLU : http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

- Collaborating with other ECP ST projects:
  - xSDK4ECP (Mike Heroux, Lois McInnes, et al.)
    - Interoperability among numerical libraries, easy plug-in in applications.
  - Pagota 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

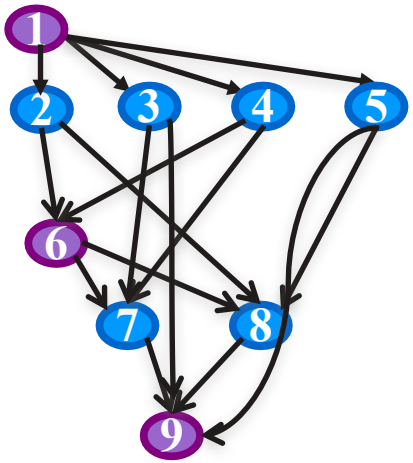
- Explore new algorithms that require lower arithmetic complexity, communication and synchronization, faster convergence rate
  - STRUMPACK: “inexact” direct solver, preconditioner, based on hierarchical low rank structures: HSS, HODLR, etc.
  - SuperLU: new 3D algorithm to reduce communication
- Refactor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

Sparse factorization for linear systems

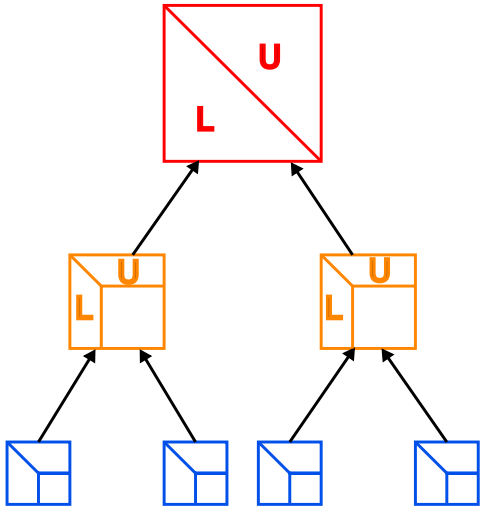


Two algorithm variants

DAG based
Supernodal: SuperLU



Tree based
Multifrontal: STRUMPACK



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

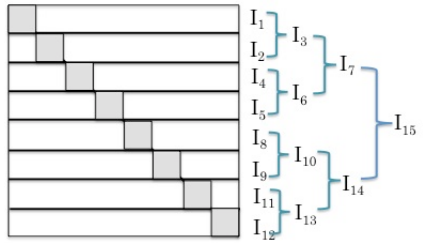
$S^{(j)} \leftarrow S^{(j)} - ((D^{(k1)}) + D^{(k2)}) + \ldots)$

STRUMPACK "inexact" direct solver

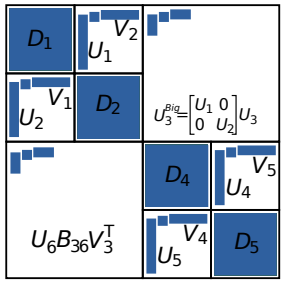
Developers: P. Ghysels, C. Gorman, F.-H. Rouet, XL

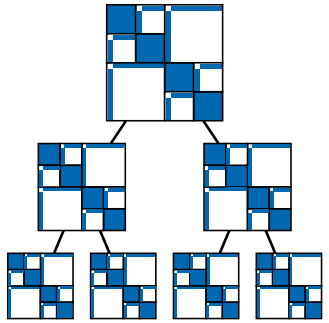
- Baseline is a sparse multifrontal direct solver.
- In addition to structural sparsity, further apply data-sparsity with low-rank compression:
  - O(N logN) flops, O(N) memory for 3D elliptic PDEs.
- Hierarchical matrix algebra generalizes Fast Multipole
  - Diagonal block **("near field")** exact;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



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





HSS approximation error vs. drop tolerance

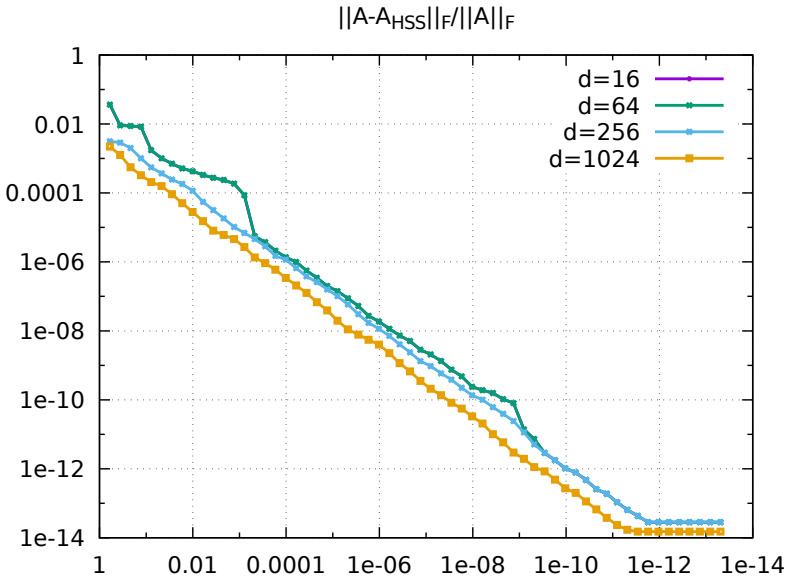
Randomized sampling to reveal rank

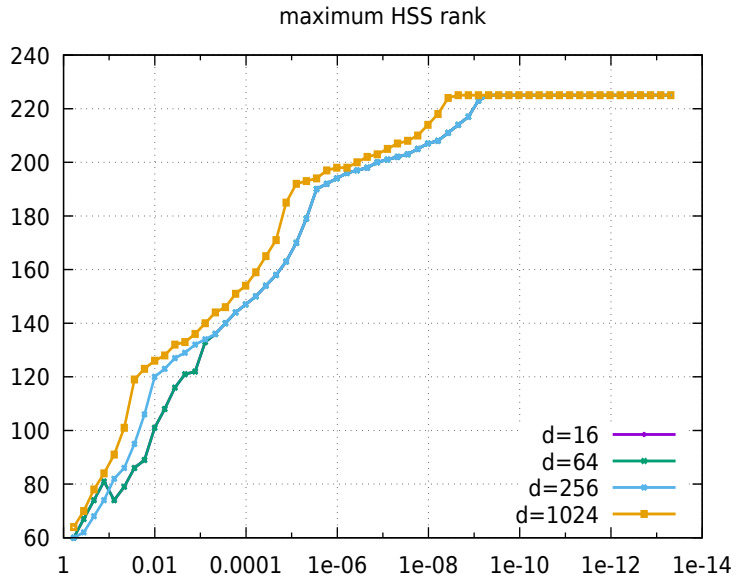
1. Pick random matrix $\Omega_{nx(k+p)}$, k target rank, p small, e.g. 10
2. Sample matrix S = A Ω, with slight oversampling p
3. Compute Q =ON-basis(S) via rank-revealing QR

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

➔ Adaptive sampling is essential for robustness:

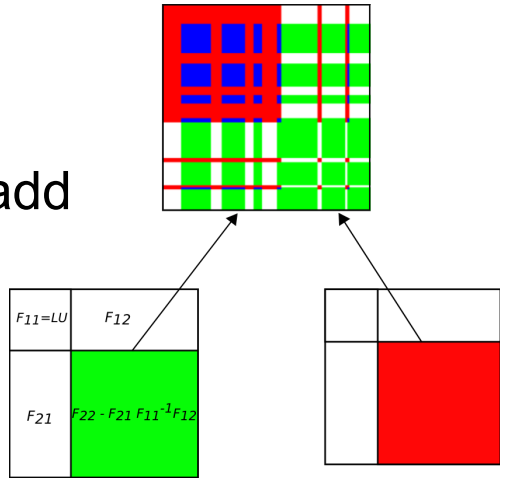
Increase d, number of columns in Ω, until error small



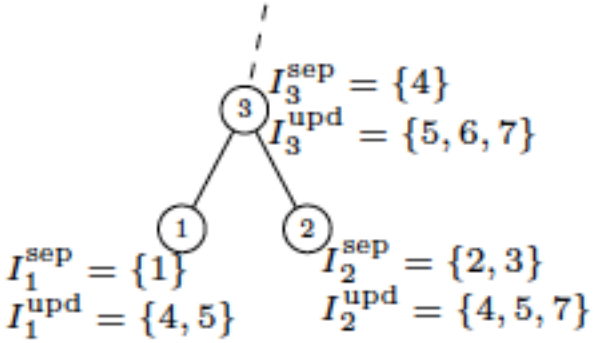


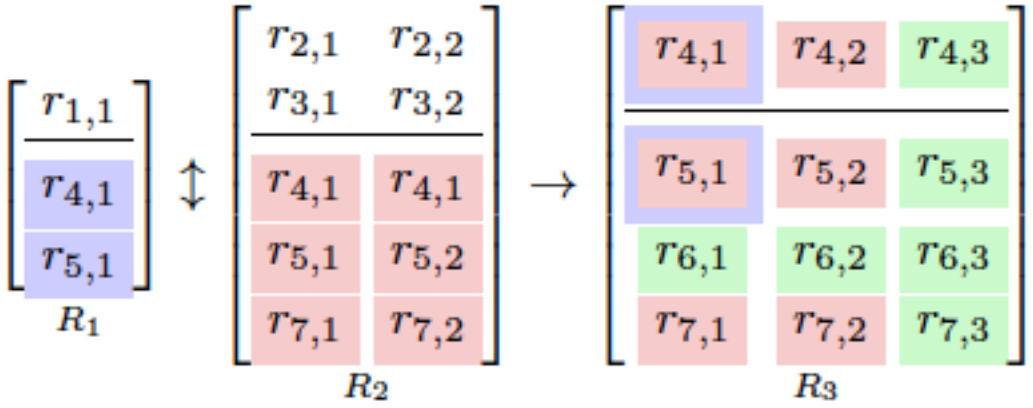
RS Simplifies “extend-add” in MF+HSS

Traditional extend-add



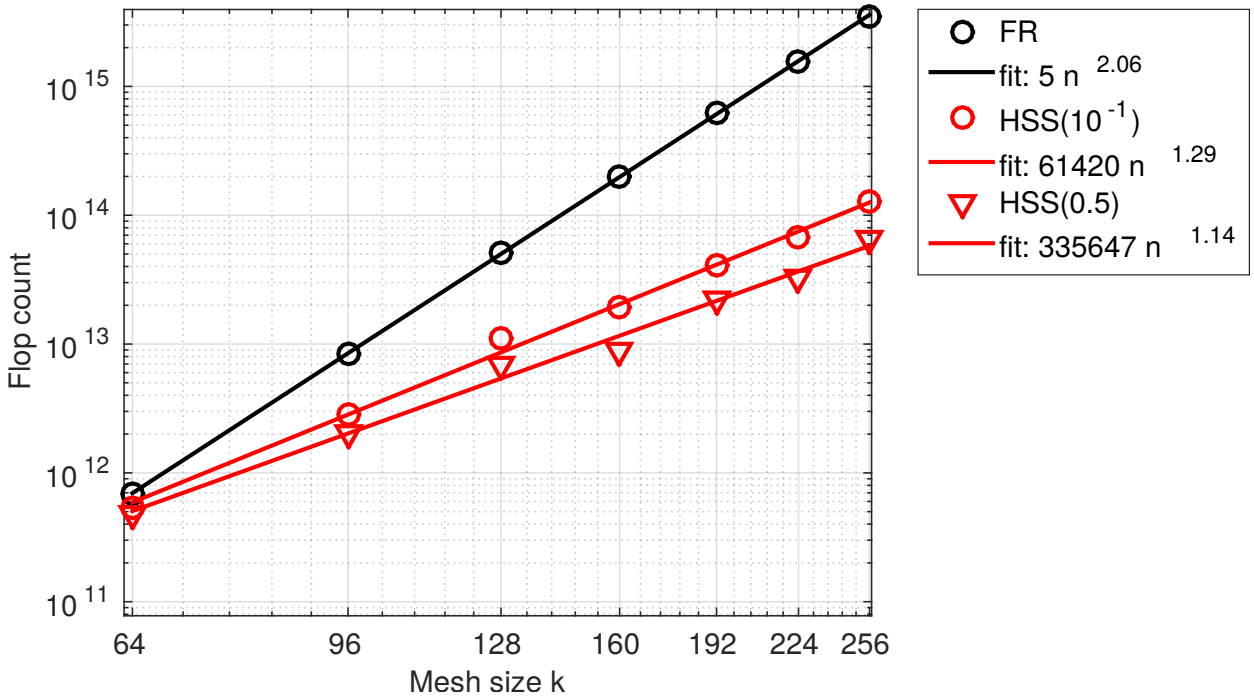
New “extend-merge” of sample matrices



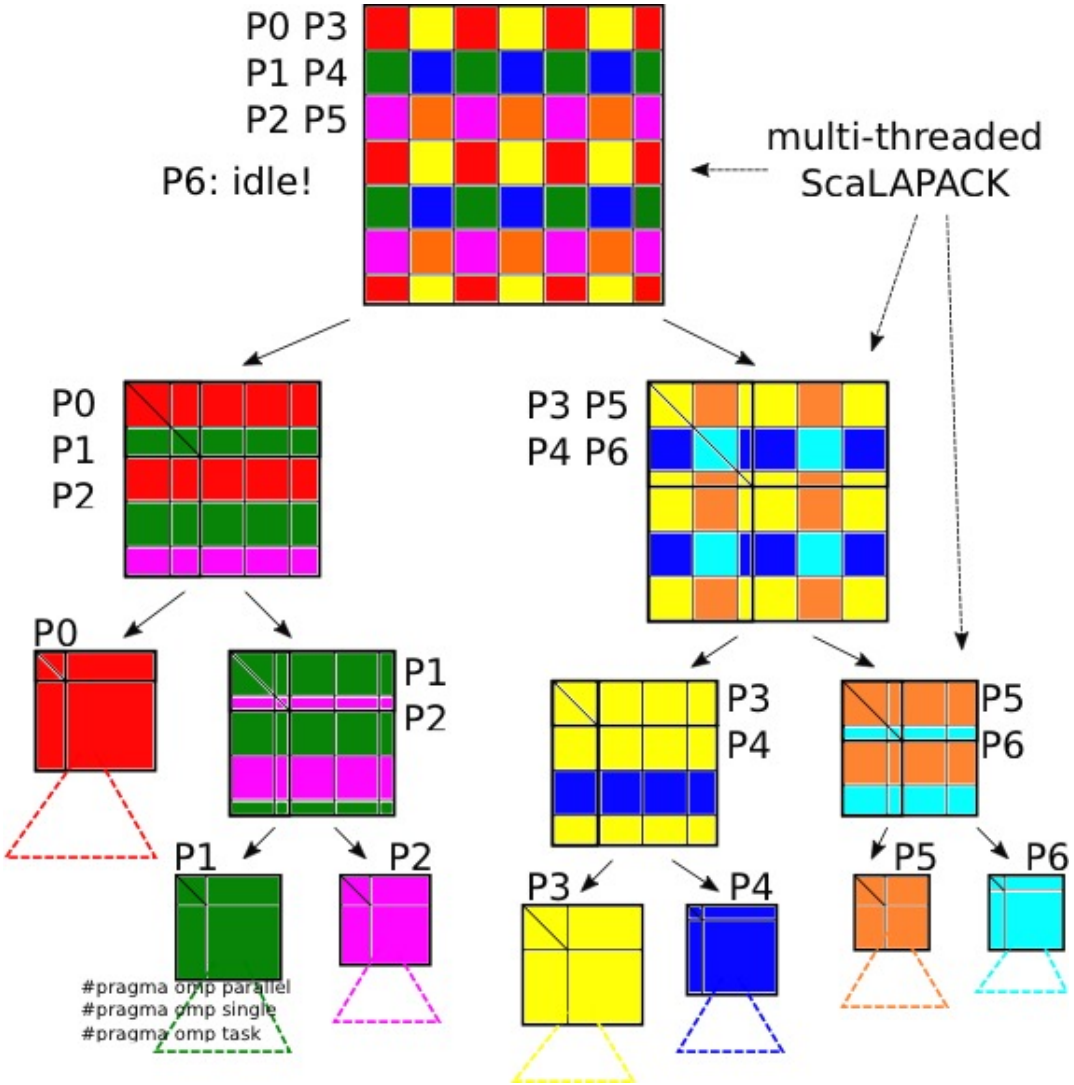


Algorithm scaling for 3D Poisson

- 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.



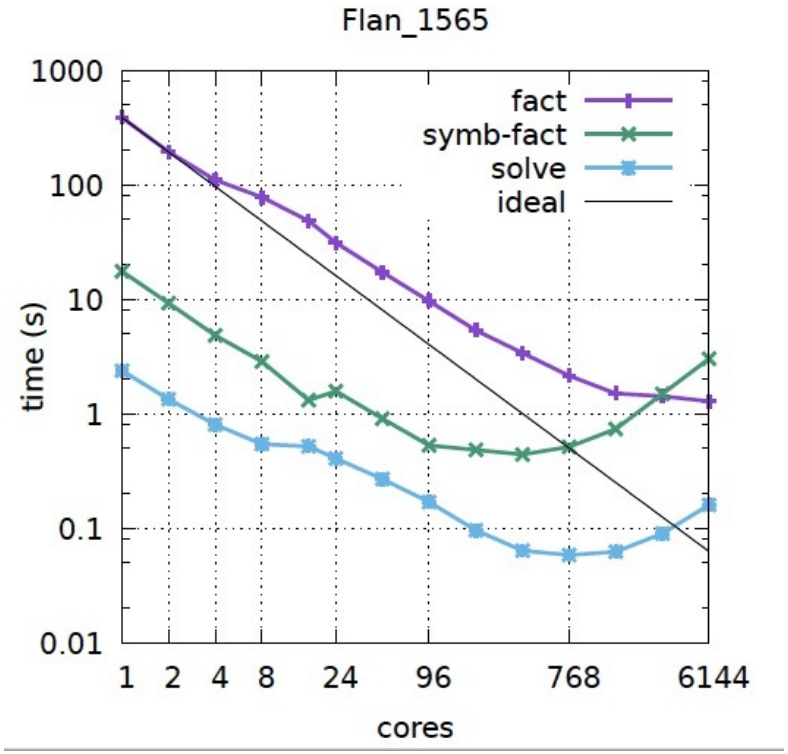
Tree-based parallelization



Performance scaling

Matrix from SuiteSparse Matrix Collection:

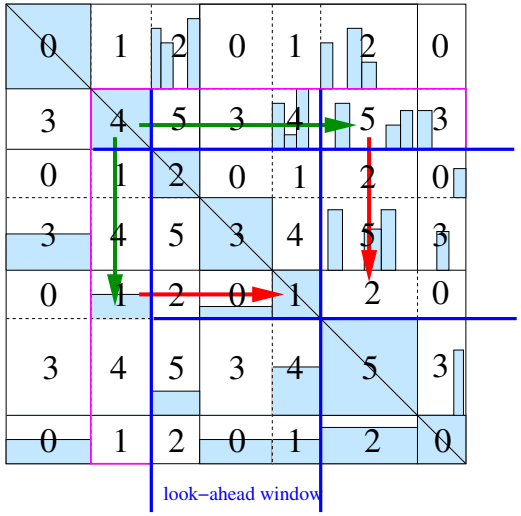
Flan_1565 (N= 1,564,794, NNZ = 114,165,372)

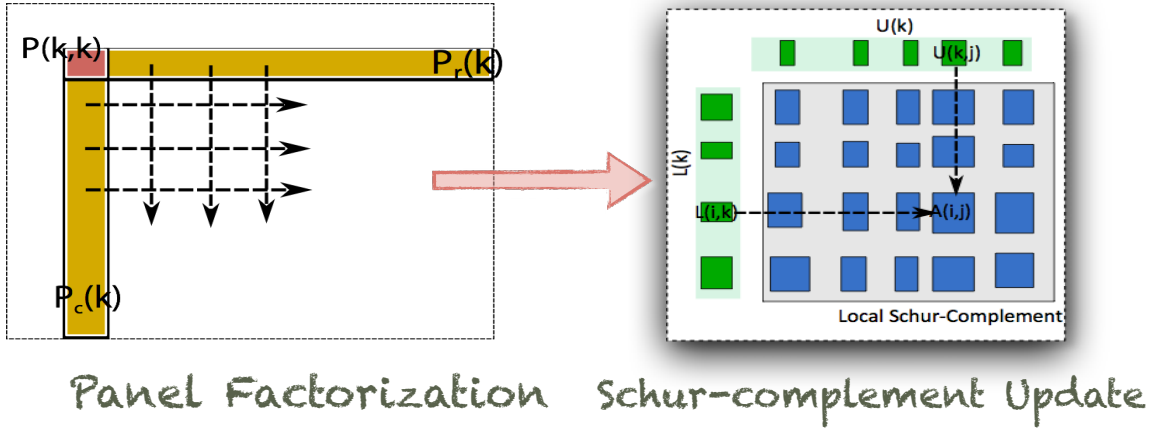


- Flat MPI on nodes with 2 12-core Intel Ivy Bridge, 64GB (NERSC Edison)
- Fill-reducing reordering (ParMetis) has poor scalability, quality decreases

SuperLU direct solver – communication pattern

Developers: XL, J. Demmel, J. Gilbert, L. Grigori, P. Sao, Meiyue Shao, I. Yamazaki

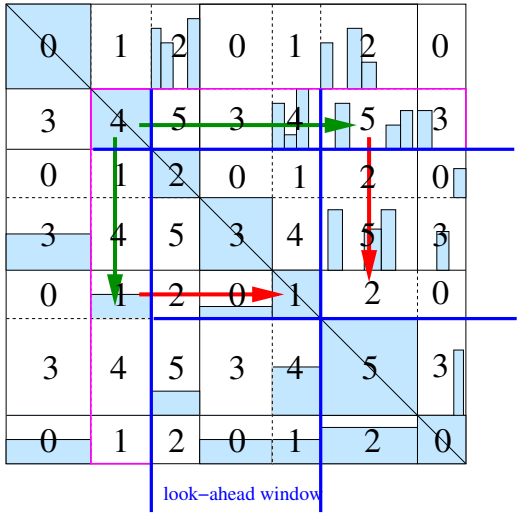


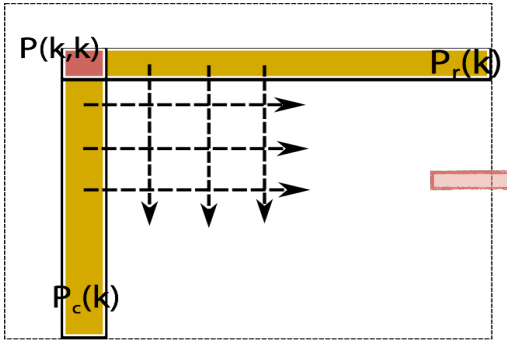


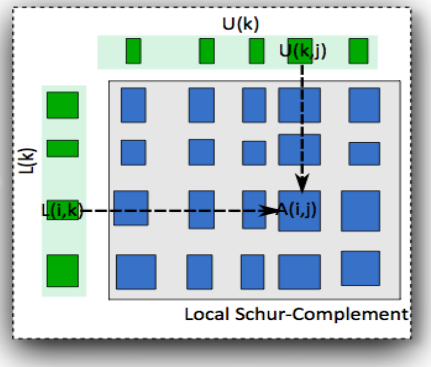
- Graph at step k+1 differs from step k
- Panel factorization on critical path

SuperLU direct solver – communication pattern

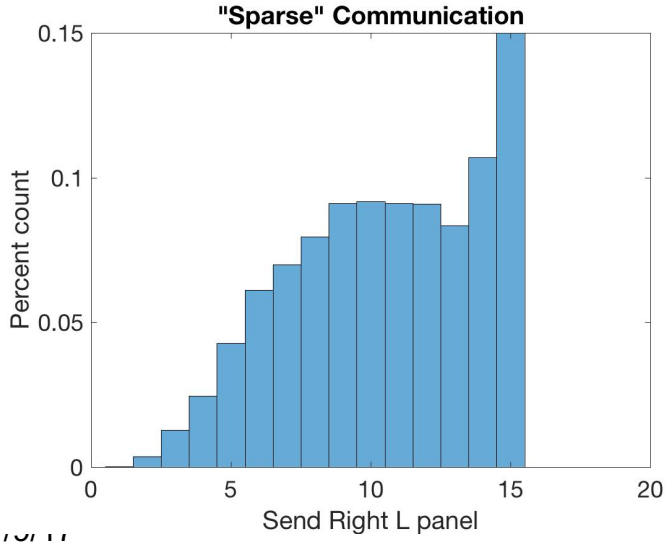
Developers: XL, J. Demmel, J. Gilbert, L. Grigori, P. Sao, Meiyue Shao, I. Yamazaki







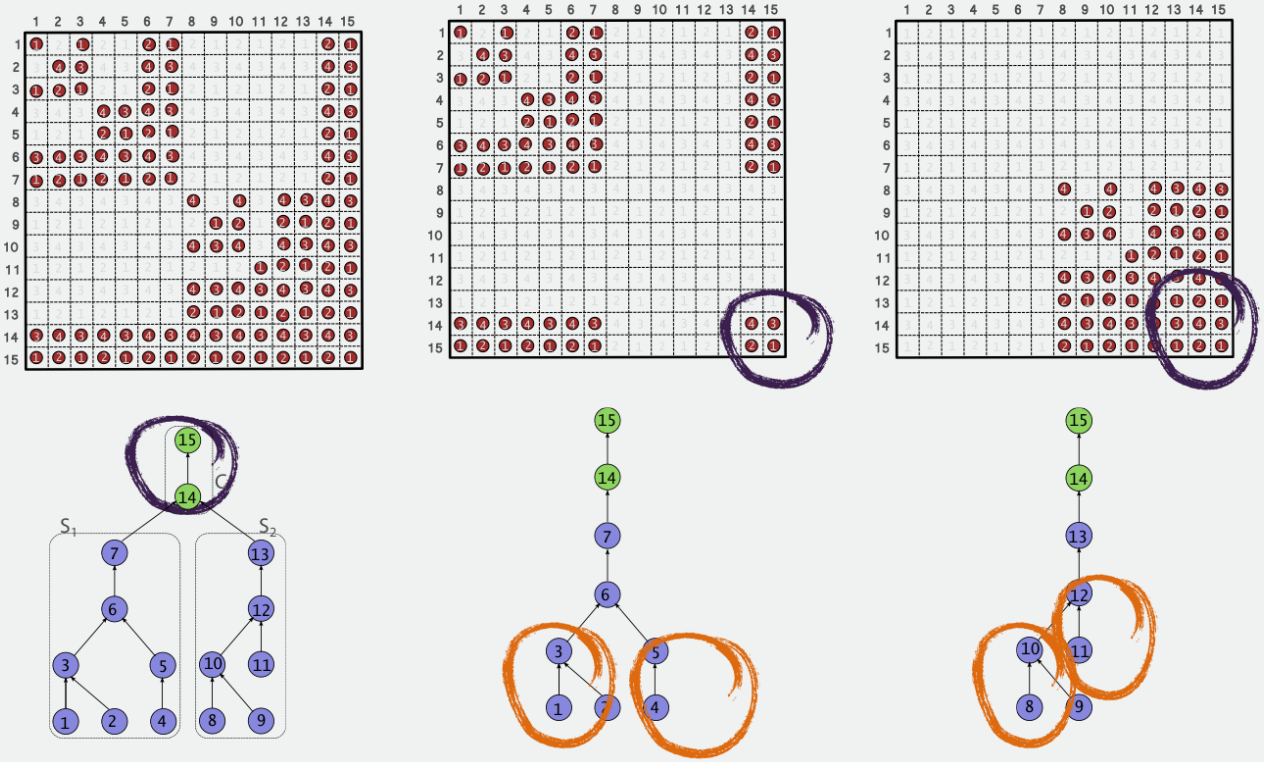
Panel Factorization Schur-complement Update



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

8 x 16 MPI grid:
Profile "Send-Right"

3D Sparse LU: Data Distribution

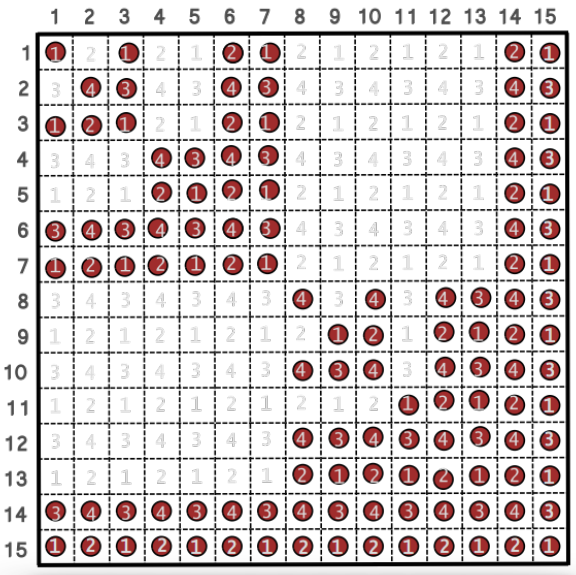


Final reduction: Schur complement of common ancestor

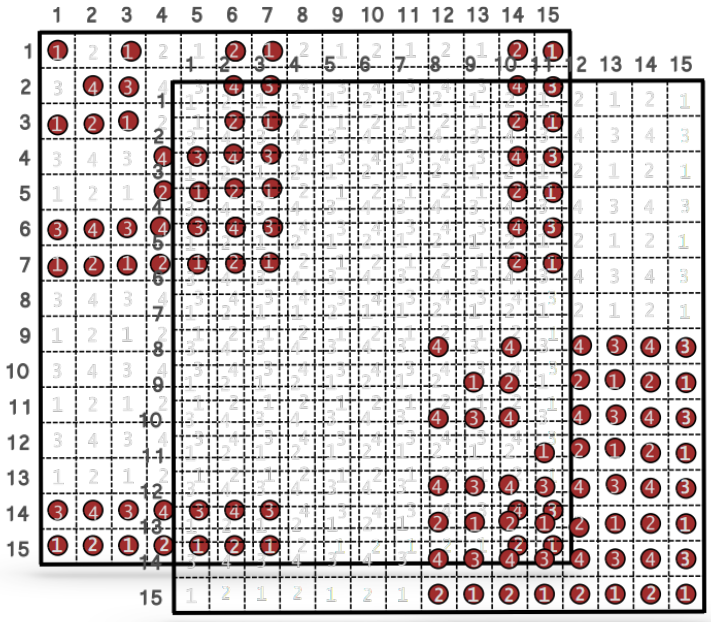
C = C1 + C2 + …

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

3D Sparse LU: cost of communication

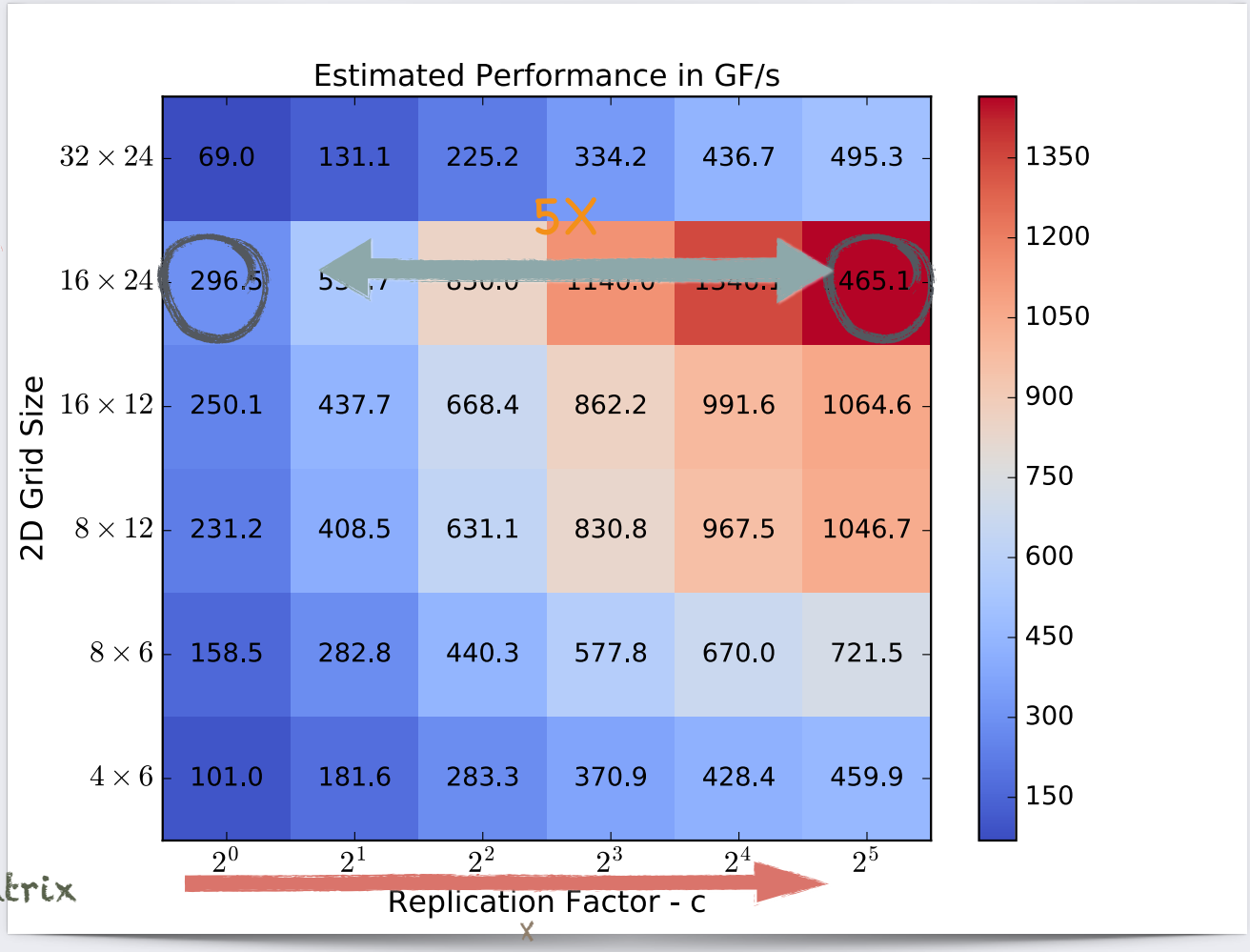


Grid size	Communication volume
$\sqrt{P} \times \sqrt{P}$	$V_{2D}(P) = v_p$
$\sqrt{2P} \times \sqrt{2P}$	$V_{2D}(2P) = \frac{v_p}{\sqrt{2}}$



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

3D Sparse LU: Performance Projection



torso3: 'sparser' matrix

Actual performance

p=768

r X c X d

	Nd24k	torso3	H2O
32X24X1	10.92	4.19	9.18
16X24X2	3.55	1.62	3.35
16x12x4	3.31	1.22	3.28
8x12x8	3.00	0.70	2.65
8x6x16	4.43	0.62	4.31
4x6x32	7.52	0.66	1 seg-fault

P=1536

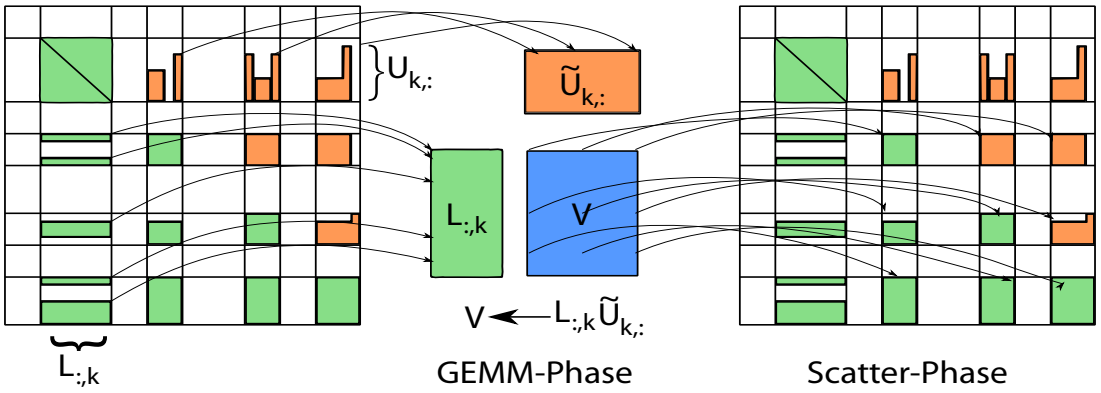
r X c X d

	nd24k	torso3	H2O
16 x24 x4	2.13	1.14	2.26
16x12x8	2.34	0.84	2.11
8x12x16	2.83	0.50	2.64
8x6x32	4.04	0.47	2 seg-fault

Local computation

Schur complement update on each MPI rank

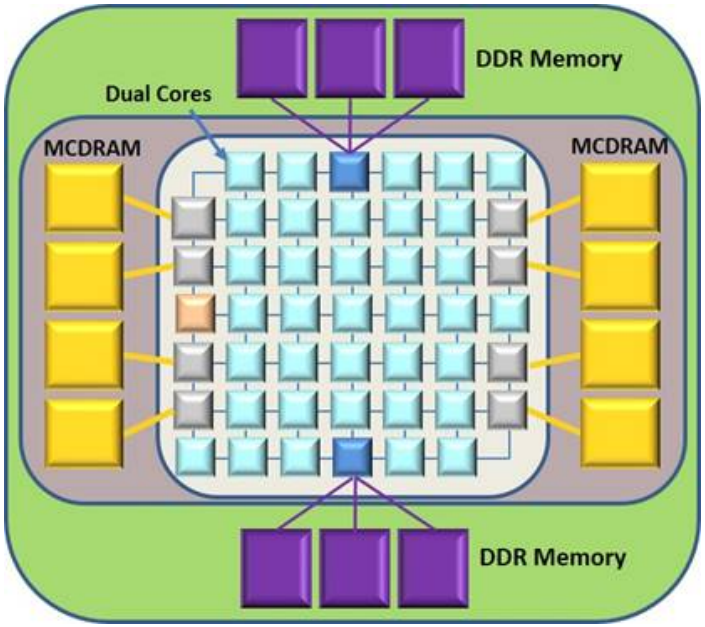
Loop through N steps: (Gaussian Elimination)
FOR (k = 1, N) {
1) **Gather** sparse blocks A(:, k) and A(k,:) into dense work[]
2) Call dense **GEMM** on work[]
3) **Scatter** work[] into remaining sparse blocks
}



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

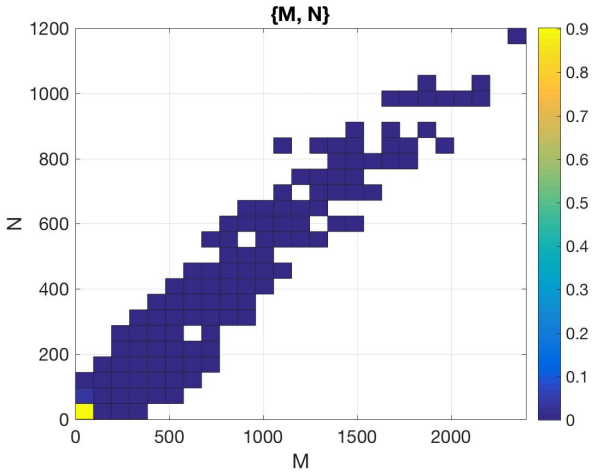
  E.g.: DGEMM AI = 2*M*N*K / (M*K + K*N + 2*M*N) / 8

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

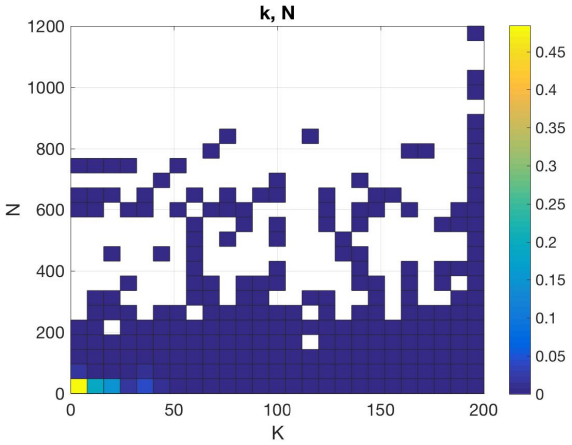
$$\text{Attainable GFLOP/s} = \min \begin{cases} \text{Peak GFLOP/s} \\ \text{AI} * \text{Peak GB/s} \end{cases}$$

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

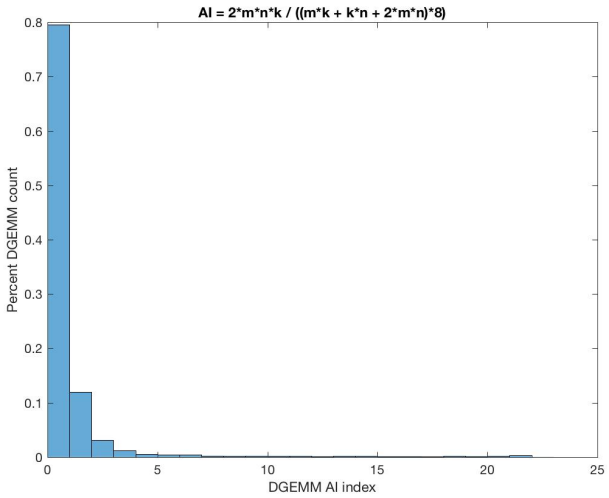
GEMM {m, n} dimensions



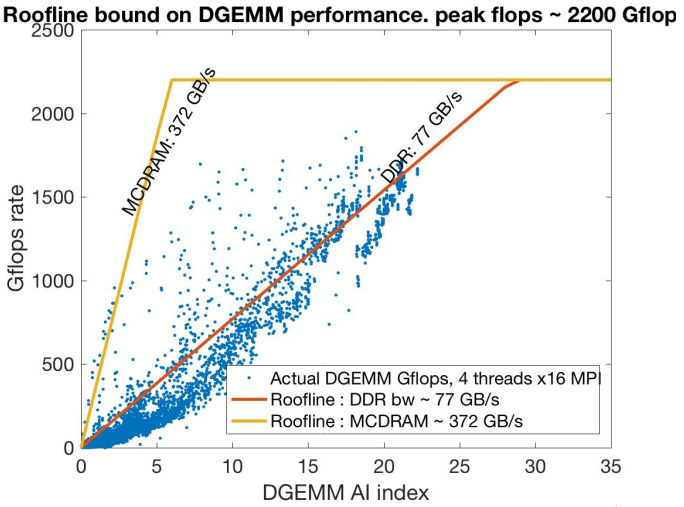
GEMM {k, n} dimensions



Arithmetic Intensity



DGEMMs performance profile



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

- Software is available

THANK YOU