SAND2017-2516C

ShyLU: A Collection of Node-Scalable Sparse Linear Solvers

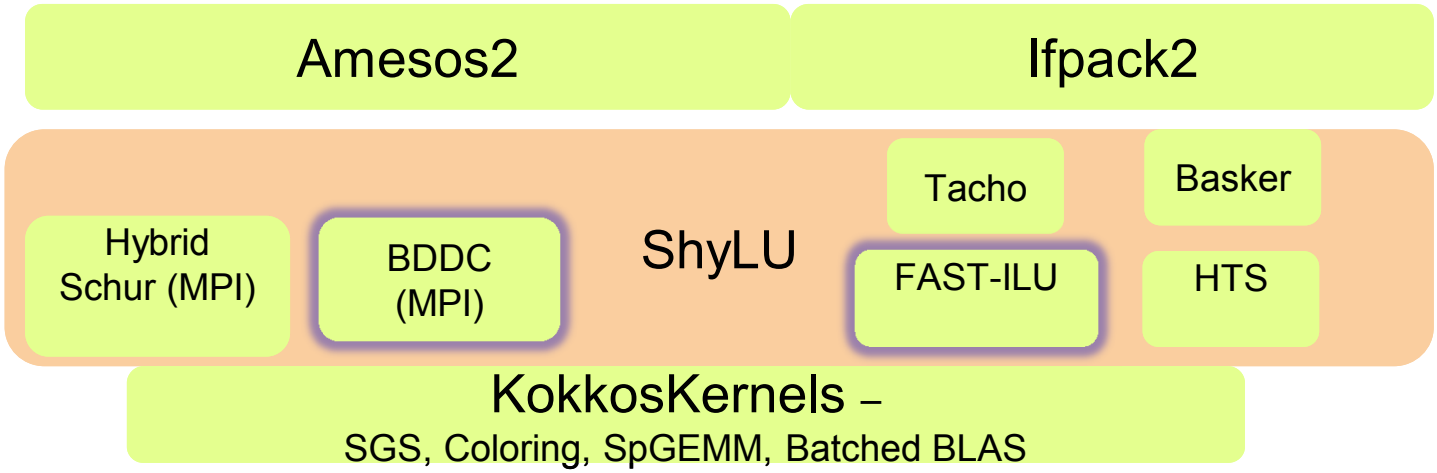
Siva Rajamanickam

Joint Work: Kyungjoo Kim, Mehmet Deveci, Andrew Bradley, Erik Boman

SIAM CSE 2017, Atlanta

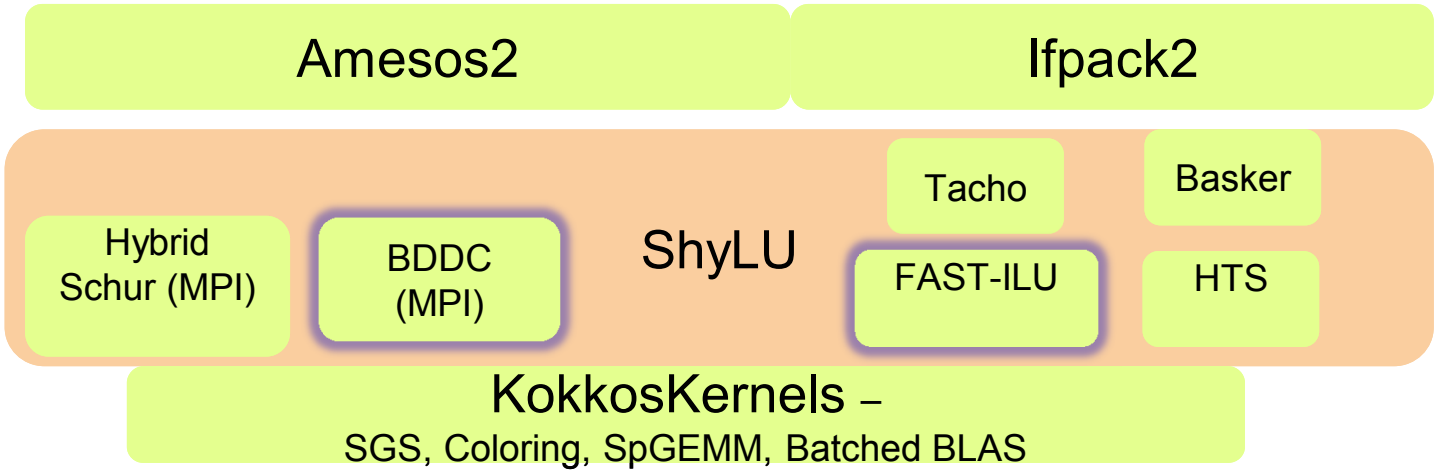
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Trilinos Subdomain solvers: Overview



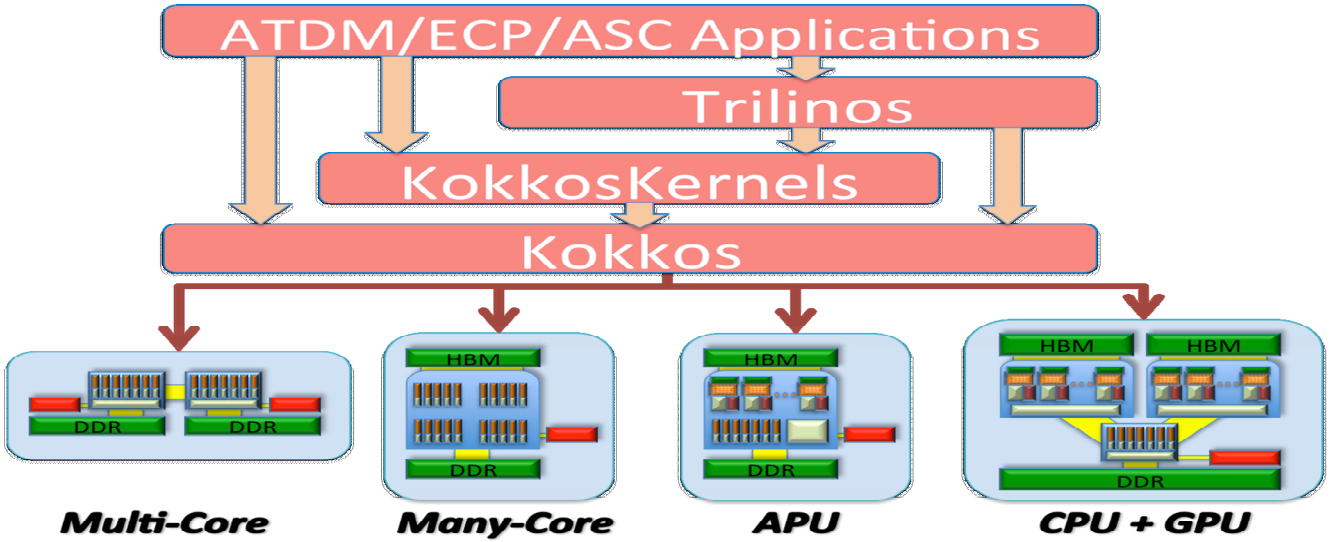
- **ShyLU Domain decomposition Solvers**
  - **Hybrid-Schur**:
    - MPI+X hybrid programming model
    - Direct+Iterative Hybrid Schur Complement Solver
  - **BDDC**: Balancing Domain Decomposition
    - Experimental mode now
    - 2-level domain-decomposition method
    - Can be extended to multilevel
  - **GDSW**: Generalized Drija Smith Widlund (planned)

Trilinos Subdomain solvers: Overview



- **ShyLU on Node Solvers**
- Multiple Kokkos-based options for on-node parallelism
  - **Basker :** LU or ILU (t) factorization
  - **Tacho: (**In)complete Cholesky - IC (k)
  - **Fast-ILU:** Fast-ILU factorization for GPUs
  - **HTS:** Multithreaded Triangular solves
- Under active development. Jointly funded by ASC, ATDM, FASTMath, LDRD.

KokkosKernels : Overview



- Layer of **performance portable** kernels on top of Kokkos
  - Sparse linear algebra kernels
  - Dense linear algebra kernels (Batched BLAS as well as traditional BLAS)
  - Graph kernels
  - Tensor Contraction kernels (upcoming)

Themes for Architecture Aware Solvers and Kernels : Data layouts

Sandia National Laboratories

- Specialized memory layouts
  - Architecture aware data layouts
    - Coalesced memory access
    - Padding
    - Array of Structures vs Structure of Arrays
    - Kokkos based abstractions (H. C. Edwards and C. Trott)
- Two dimensional layouts for matrices
  - Allows using 2D algorithms for solvers and kernels
  - Bonus: Fewer synchronizations with 2D algorithms
  - Cons : Much more harder to design correctly
  - Better utilization of hierarchical memory like High Bandwidth Memory (HBM) in Intel Xeon Phi or NVRAM
- Hybrid layouts
  - Better for very heterogeneous problems

Themes for Architecture Aware Solvers and Kernels : Fine-grained Synchronization

Sandia National Laboratories

- Synchronizations are expensive
  - 1D algorithms for factorizations and solvers, such as ND based solvers have a huge synchronization bottleneck for the final separator
  - Impossible to do efficiently in certain architectures designed for massive data parallelism (GPUs)
  - This is true only for global synchronizations, fork/join style model.
- Fine grained synchronizations
  - Between handful of threads (teams of threads)
  - Point to Point Synchronizations instead of global synchronizations
    - Park et al (ISC14) showed this for triangular solve
  - Thread parallel reductions wherever possible
    - Atomics are cheap
      - Only when used judiciously

Themes for Architecture Aware Solvers and Kernels : Task Parallelism

Sandia National Laboratories

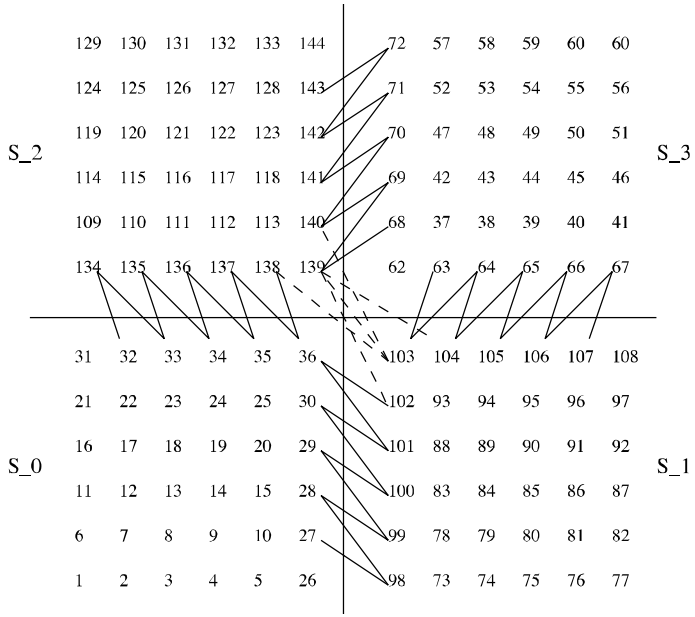
- Statically Scheduled Tasks
  - Determine the static scheduling of tasks based on a task graph
  - Eliminate unnecessary synchronizations
    - Tasks scheduled in the same thread do not need to synchronize
    - Find transitive relationships to reduce synchronization even further
      - Jongsoo Park et al
- Dynamically scheduled tasks
  - Use a tasking model that allows fine grained synchronizations
    - Requires support for futures
    - Not the fork-join model where the parent forks a set of tasks and blocks till they finish
    - Kokkos Tasking API
      - Joint work with Carter Edwards, Stephen Olivier, Kyungjoo Kim, Jon Berry, George Stelle

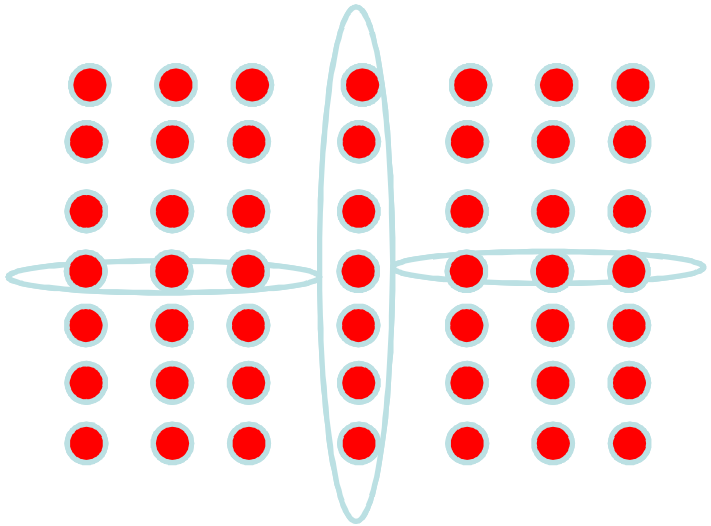
Parallel ILU(k) factorization

Sandia National Laboratories

- Parallel ILU(k) factorization
  - Can we focus on ILU(k) algorithm to reduce synchronizations ?
  - Starting Point : *Hysom and Pothen 01* based on **three assumptions**
    - Good edge separators exist for the adjacency graph of the coefficient matrix
    - Size of the problem sufficiently large relative to number of processors
    - Subdomain intersection graph should have a small chromatic number

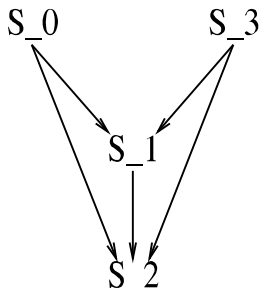
Parallel ILU(k) factorization : Assumption 1 in H&P

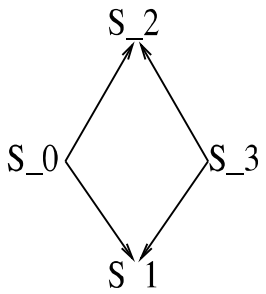


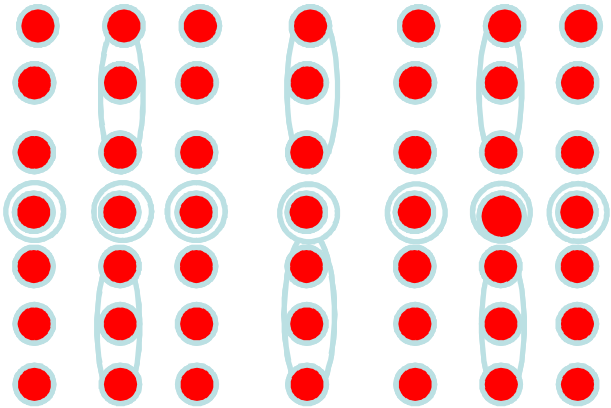


- Hysom and Pothen numbers all internal and boundary vertices of a subdomain before it numbers any other subdomain – requires good edge separators
- This can be relaxed to vertex separators if all “leaf” vertices are numbered before internal or “non-leaf” vertices in the ND tree in a levelwise fashion
  - The incomplete fill path theorem can be extended to support this

Parallel ILU(k) factorization : Assumption 3 in H&P







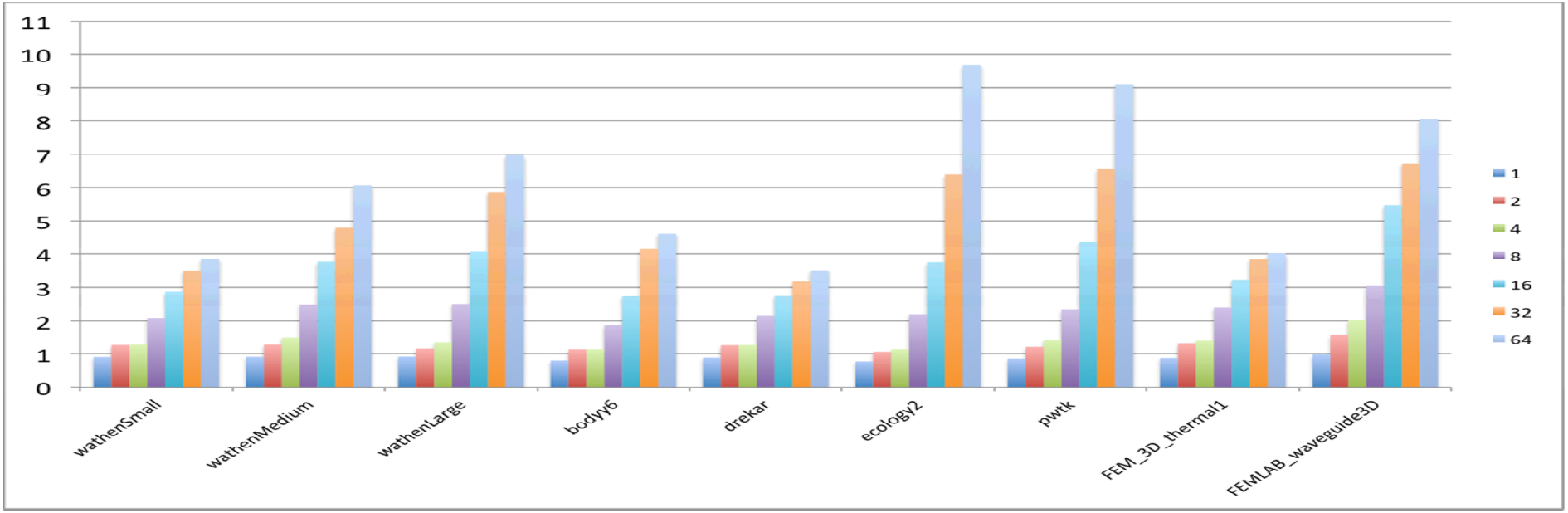
- Hysom and Pothen colors the “subdomain” graph with the directed edges representing the ordering – reduce the path length with coloring the subdomain graph. Fill is also limited by this graph.
- This can be relaxed to an “interface graph” where each “corner”, “edge” and “face” in 3D for a subdomain is represented by a vertex
  - Color the interface graph to reduce the path length. Fill is also limited by this graph.

Parallel ILU(k) factorization

Sandia National Laboratories

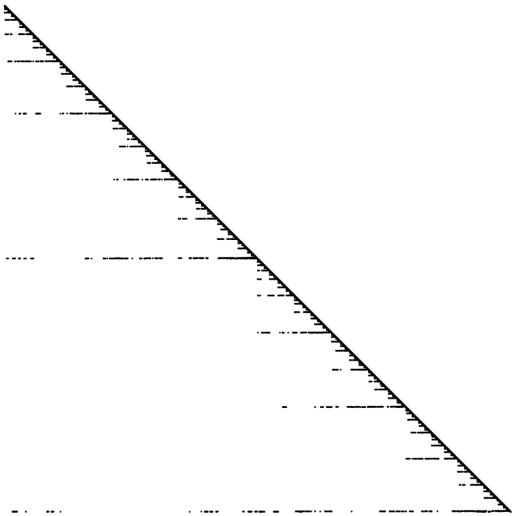
- Currently using Scotch for the ND
  - Special options provided by Scotch developers (Thank you !)
    - Graph partitioning tools can provide the finer granularity ND tree but they don't
    - Most expensive portion when graph structure changes
- Uses Coarse Nested Dissection intersection graph instead of the fine graph for the level sets.
  - Can be improved by adapting the fine ND
- Currently uses barriers between coarse level sets
  - Can be improved by adapting the same techniques used by HTS

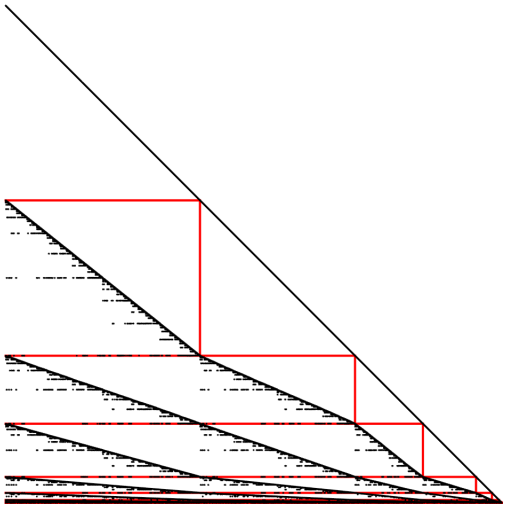
Parallel ILU(k) factorization speedup

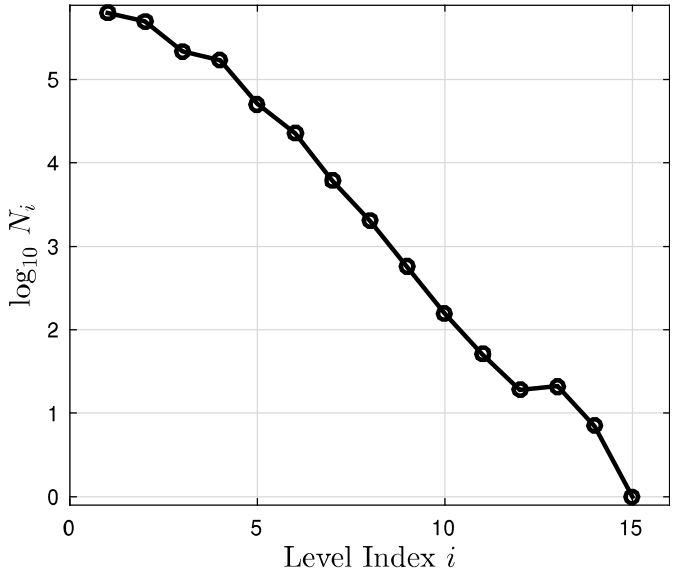


- Slightly more expensive in one thread than a sequential algorithm
- The assumption on work per subdomain still holds
- Reducing the synchronization even further will definitely help. Last separator (root) is a problem in number of cases.

HTS: Hybrid Triangular Solve

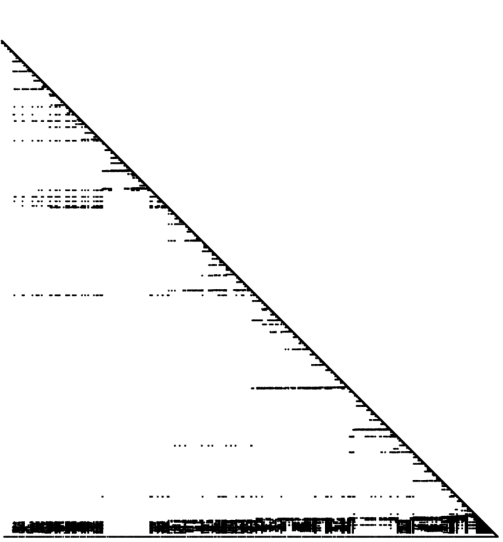


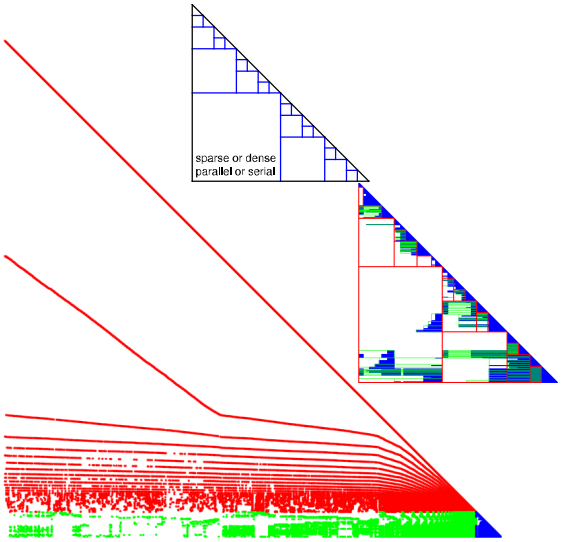


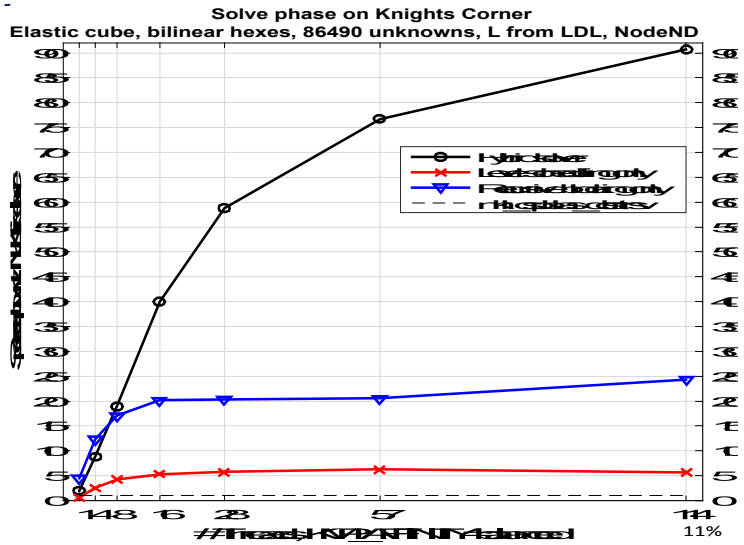


- Solve R * P * T * Q * x = b
  - Row Scaling (R), Row and Column Permutations (P, Q)
- Solve multiple triangular solves with single right hand side with same T or pattern(T)
- Symbolic (Find parallelism), Numeric (Refresh data structures), (Actual) Solve Phases
- Number of rows in level vary widely
- ***A. Bradley (Developer)***

HTS: Hybrid Triangular Solve

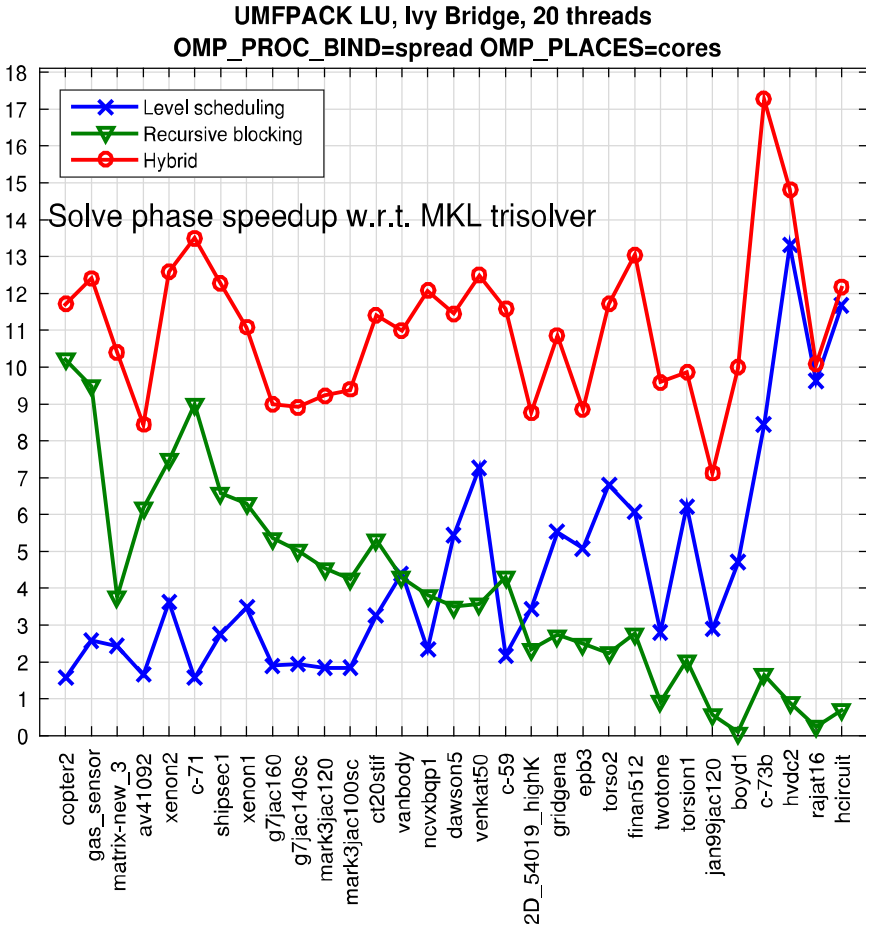


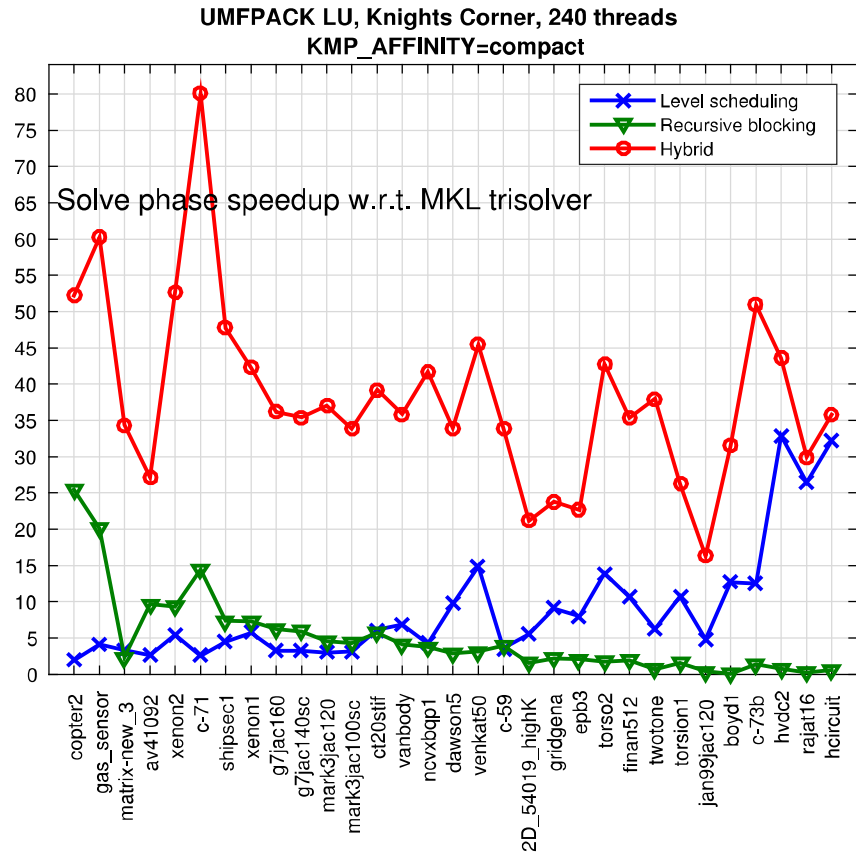




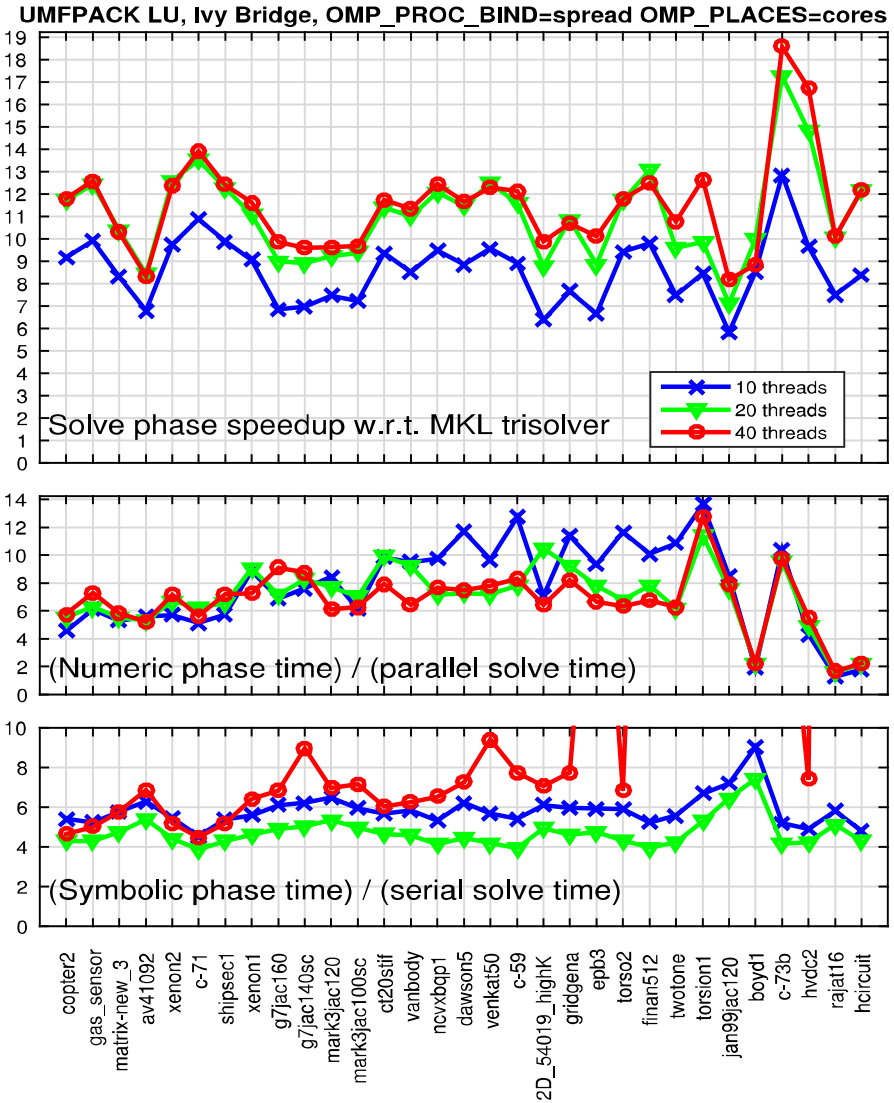
- Denser (sub) structures in the higher levels that can be exploited.
- Recursive Block Decomposition of the dense triangular matrix.
- Sequential or parallel spmv and triangular solves within the dense triangular factors
- A hybrid triangular solve does better than either level-sets or recursive block format by themselves
- Using the P2P communication between threads and other tricks related to it (Park et. Al)

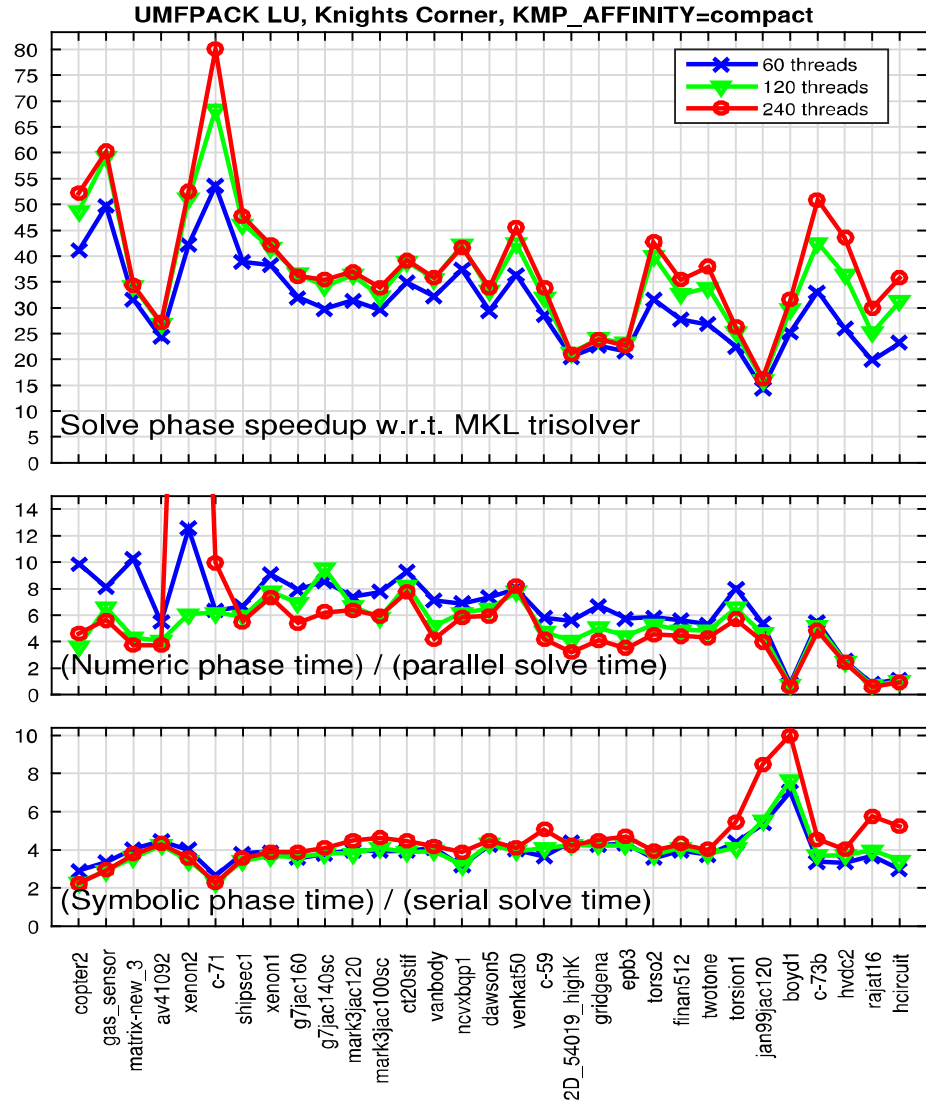
HTS Triangular Solve: Comparison of Different Methods



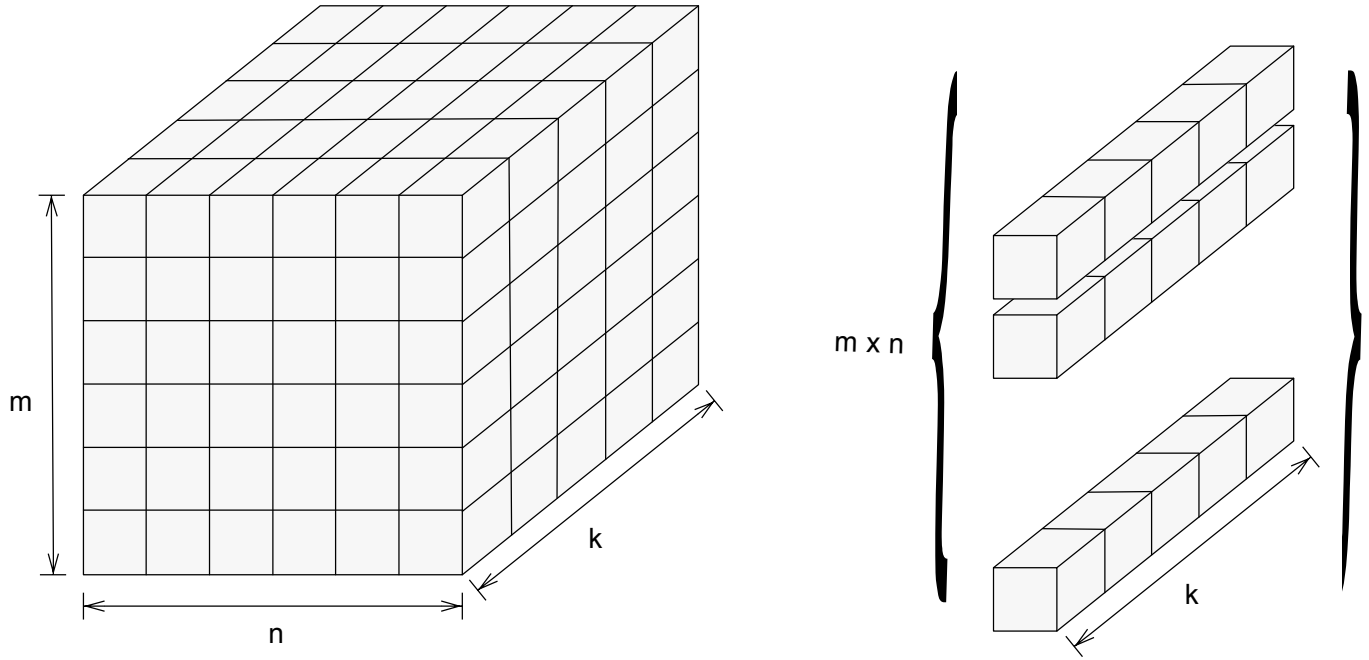


HTS Triangular Solve – Results on IvyBridge and KNC



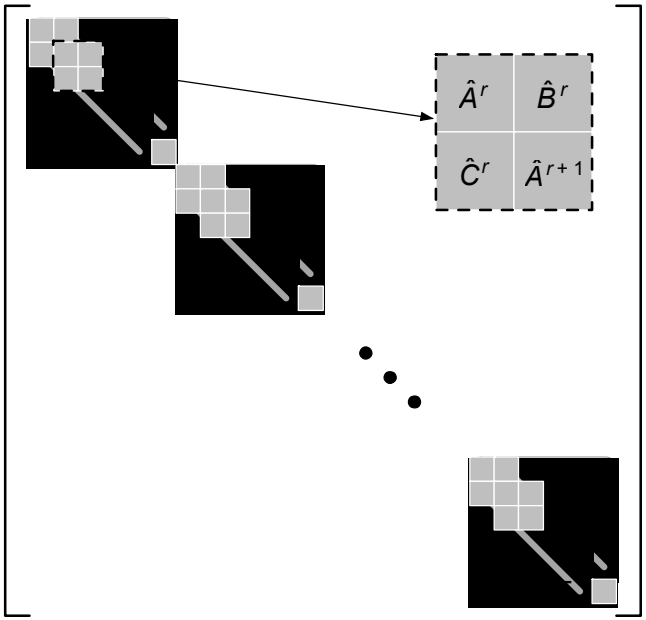


Motivation for Batched BLAS with Compact Layouts



- Sandia application characteristics
  - One dimension of the mesh more important than the others when preconditioning
  - Multiple degrees of freedom per element gives rise to tiny blocks

Motivation for Batched BLAS/LAPACK

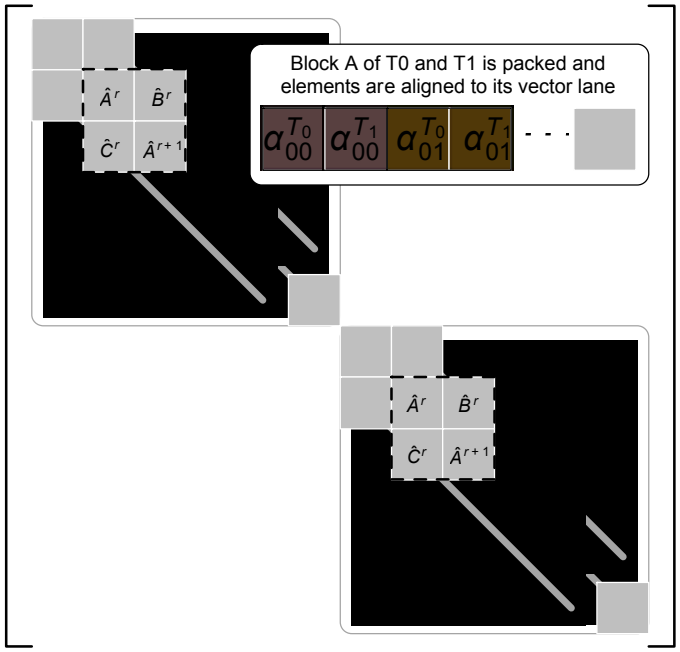


Algorithm 1: Reference impl. TriLU

```
1 for T in {T_0, T_1, ..., T_{m-1}} do in parallel
2     for r = 0 to k-2 do
3         Â^r := LU(Â^r);
4         B̂^r := L^{-1} B̂^r;
5         Ĉ^r := Ĉ^r U^{-1};
6         Â^{r+1} := Ĉ^{r+1} - Ĉ^r B̂^r;
7     end
8     Â^{k-1} := {L · U};
9 end
```

- Block Jacobi preconditioner where each block is a Tridiagonal matrix
- Every scalar in the tridiagonal matrix is a small block matrix
  - Block sizes 5x5, 9x9, 15x15 etc
- Typical number of diagonal blocks 512-1024
- **Key kernels needed DGEMM, LU, TRSM**

KokkosKernels Compact Layouts for Batched BLAS



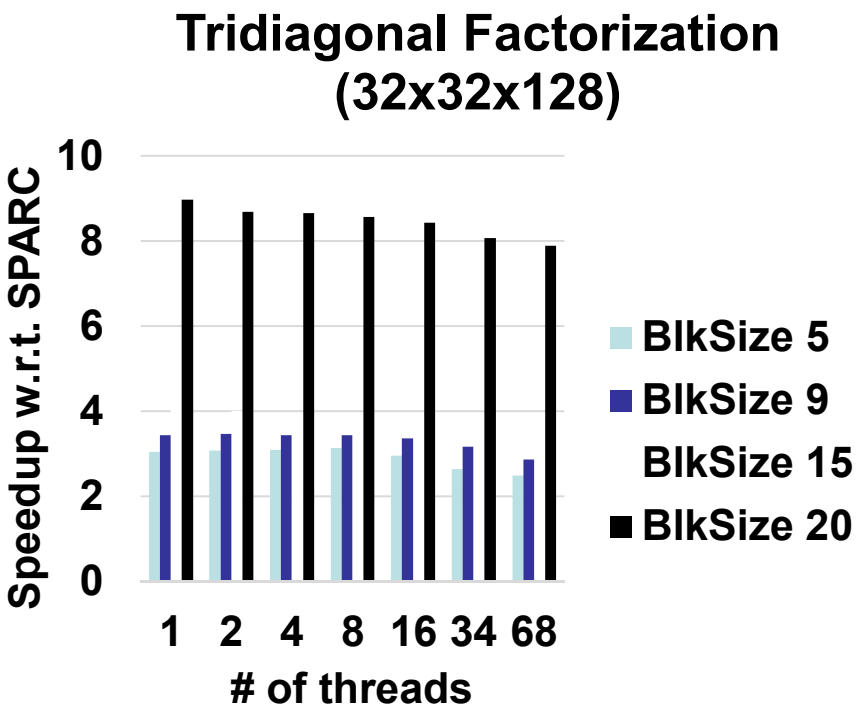
Algorithm 2: Batched impl. TriLU

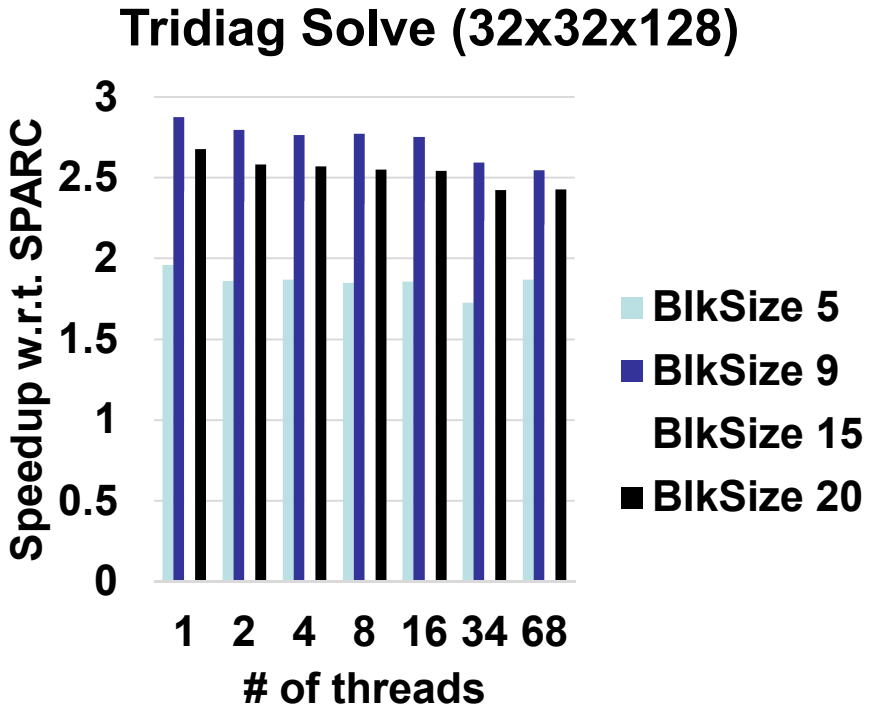
1 for a pair $T(0,1)$ in $\{\{T_0, T_1\}, \{T_2, T_3\}, \cdots, \{T_{m-2}, T_{m-1}\}\}$ do in parallel
2 for r 0 to $k-2$ do
3 $\hat{A}^{r(0,1)} := LU(\hat{A}^{r(0,1)})$;
4 $\hat{B}^{r(0,1)} := L^{-1}\hat{B}^{r(0,1)}$;
5 $\hat{C}^{r(0,1)} := \hat{C}^{r(0,1)}U^{-1}$;
6 $\hat{A}^{r+1(0,1)} := \hat{C}^{r+1(0,1)} - \hat{C}^{r(0,1)}\hat{B}^{r(0,1)}$;
7 end
8 $\hat{A}^{k-1(0,1)} := \{L \cdot U\}$;
9 end

- Data Layout for better vector intrinsics
  - Pack entries from up to vlen block diagonal matrices, vlen is the vector length (vector length = 2 shown)
  - Use vector intrinsics on the new data vector data with operator overloading
- Scalar Performance is due to explicit loop unrolling

KokkosKernels Batched BLAS : Usage within Preconditioner, KNL, 1x68x4, 1.4 Ghz, Intel 17.1.132

Sandia National Laboratories





- Performance comparisons for Large-Block Jacobi Small-Block Tridiagonal factorization and Triangular Solve
- One right hand side per solve
- Speedups against a hand-tuned version of the code within the application

Conclusions

- Themes around Thread Scalable Subdomain solvers
  - Data Layouts
  - Fine-grained Synchronizations
  - Task Parallelism
  - Asynchronous Algorithms
- Presented three upcoming methods
  - A Traditional multithreaded ILU(k)
  - A Hybrid triangular solve
  - A compact layout based LU factorization

Sandia National Laboratories

Questions ?

Sandia National Laboratories

Backup Slides

Iterative Inexact Triangular Solves

- Just Computing the LU factors is only part of the cost. We need to apply the factors in fine-grained fashion.
- Solve the triangular factors with fixed number of sweeps of Jacobi iteration
- Use the matrix splitting

$$A = (A - D) + D;$$

- Given an initial guess use the update:

$$x_{k+1} = (I - D^{-1}A)x_k + D^{-1}b.$$

- Will converge if :

$$\rho(I - D^{-1}A)) < 1$$

- This is always true for triangular matrices (spectral radius of zero)
  - Note: *This is asymptotic convergence*

Themes for Architecture Aware Solvers and Kernels : Asynchronous Algorithms

Sandia National Laboratories

- System Level Algorithms
  - Communication Avoiding Methods (s-step methods)
    - Not truly asynchronous but can be done asynchronously as well.
    - Multiple authors from early 1980s
  - Pipelined Krylov Methods
    - Recently Ghysels, W. Vanroose et al. and others
- Node Level Algorithms
  - Finegrained Asynchronous iterative ILU factorizations
  - An iterative algorithm to compute ILU factorization (Chow et al)
    - Asynchronous in the updates
  - Finegrained Asynchronous iterative Triangular solves
    - Jacobi iterations for the triangular solve.
  - Kacmarcz, Cimmino type methods and their block variants (Boman's Talk, Monday)

Asynchronous ILU factorization + Tri Solves vs Exact ILU factorization

Sandia National Laboratories

FastILU
10 sweeps,
RCM ordering
GPUs, damping
Factor = 0.5

	0	1	2	3	4	5
thermal2	1343	924	840	815	819	811
af_shell3	901	653	565	589	554	599
ecology2	1704	1103	925	910	893	922
apache2	1043	629	432	484	427	497
offshore	350	211	184	175	172	172
G3_circuit	904	607	512	471	431	410
Parabolic_fem	356	328	295	288	285	286

Exact ILU

	0	1	2	3	4	5
thermal2	1934	1225	856	637	507	440
af_shell3	1248	788	583	462	369	309
ecology2	1625	988	696	576	467	414
apache2	1294	619	394	289	235	188
offshore	485	*	*	*	*	*
G3_circuit	1414	757	546	421	341	303
Parabolic_fem	313	238	164	129	106	91

Asynchronous ILU factorization vs Exact ILU factorization

Sandia National Laboratories

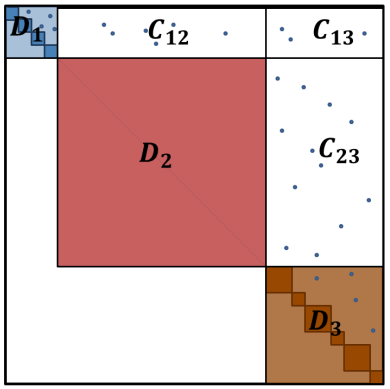
FastILU
5 sweeps,
RCM ordering
GPUs

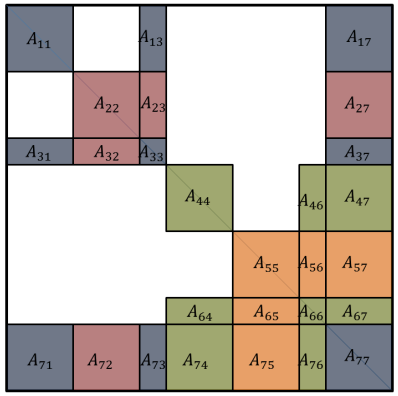
	0	1	2	3	4	5
thermal2	1421	1110	1086	1145	1172	1178
af_shell3	*	*	*	*	*	*
ecology2	1807	1311	1271	1300	1344	1308
apache2	1001	768	815	818	827	847
offshore	*	*	*	*	*	*
G3_circuit	868	612	586	574	568	562
Parabolic_fem	425	467	421	474	480	527

Exact ILU

	0	1	2	3	4	5
thermal2	1934	1225	856	637	507	440
af_shell3	1248	788	583	462	369	309
ecology2	1625	988	696	576	467	414
apache2	1294	619	394	289	235	188
offshore	485	*	*	*	*	*
G3_circuit	1414	757	546	421	341	303
Parabolic_fem	313	238	164	129	106	91

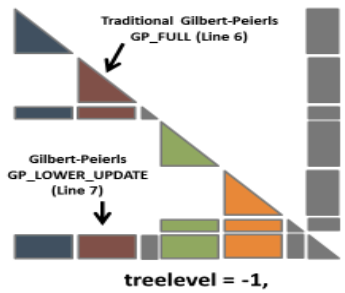
ShyLU/Basker : (I)LU factorization



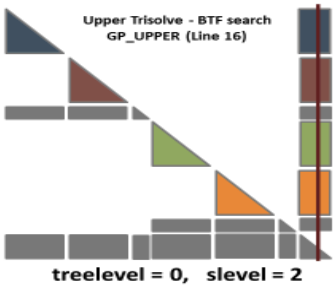


- Basker: Sparse (I)LU factorization
  - Block Triangular form (BTF) based LU factorization, Nested-Dissection on large BTF components
  - 2D layout of coarse and fine grained blocks
    - Previous work by Sherry Li, Rothberg & Gupta
  - Data-Parallel, Kokkos based implementation
  - Fine-grained parallel algorithm with P2P synchronizations
  - Parallel version of Gilbert-Peirels' algorithm (or KLU)
  - Left-looking 2D algorithm requires careful synchronization between the threads
    - All reduce operations between threads to avoid atomic updates
- See "*Basker: A Threaded Sparse LU Factorization Utilizing Hierarchical Parallelism and Data Layouts*" (J. Booth, S. Rajamanickam and H. Thornquist, IPDPSW)

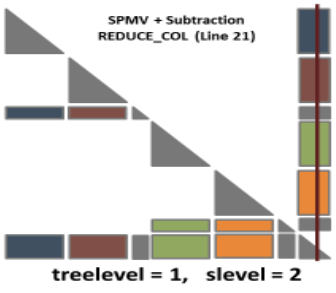
ShyLU/Basker : Steps in a Left looking factorization



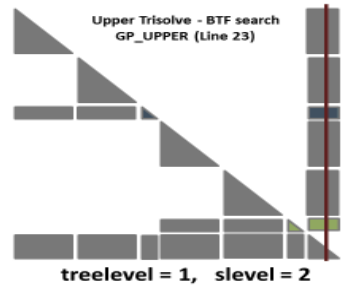
Bottom level of Dependency tree



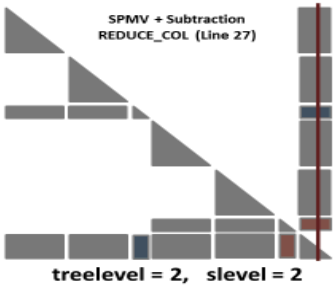
Walking from level 0, slvel is separator level



Fine grain reduction needed for level 1

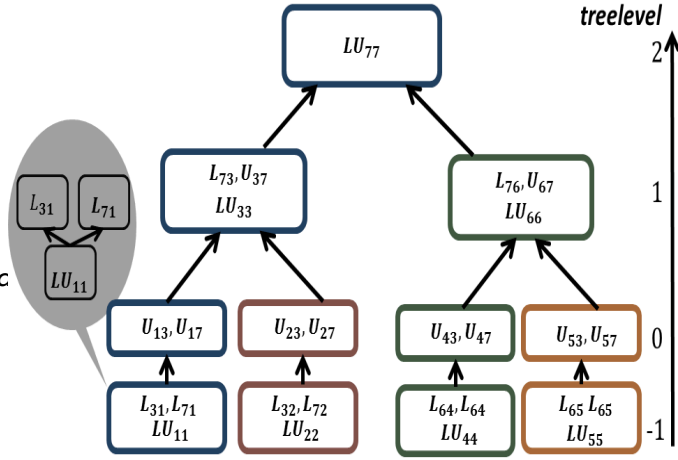


Level 1 factorization



Fine grain reduction needed for level 2

Level 2



- Different Colors show different threads
- Grey means not active at any particular step
- Every left looking factorization for the final separator shown here involves four independent triangular solve, a mat-vec and updates (P2P communication), two independent triangular solves, a mat-vec and updates, and triangular solve. (Walking up the nested-dissection tree)

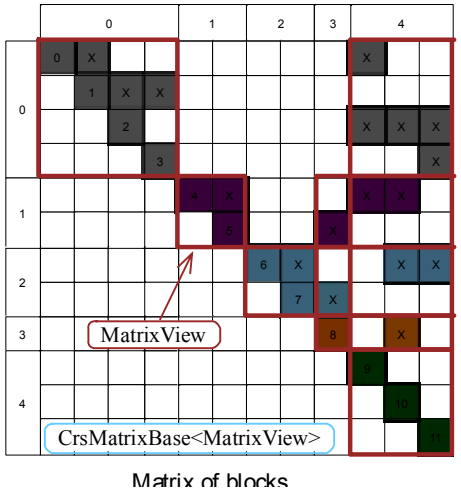
Kokkos Tasking API

- Kokkos Tasking API
  - **H. C. Edwards, S. Olivier, J. Berry, S. Rajamanickam et al**
  - Supports Pthreads, Qthreads, *Cuda (really experimental)* backends
- Kokkos Tasking API

```
1  void SimpleTask() {
2    typedef Kokkos::Threads exec_space; //Serial, Threads, Qthread
3
4    Kokkos::TaskPolicy<exec_space> policy;
5    Kokkos::Future<int> f = policy.create( Functor<exec_space>() );
6
7    policy.spawn( f );
8
9    Kokkos::wait( f );
10 }
```

```
11 class Functor<exec_space> {
12 public:
13   Kokkos::View<exec_space> data;
14
15   void apply( int &r_val ) {
16     r_val = doSomething( data );
17   }
18 };
```

ShyLU/Tacho : Task Based Cholesky factorization



Matrix of blocks

Algorithm: $A := \text{CHOL\_BLK}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

while $length(A_{TL}) < length(A)$ **do**

Determine block size b

Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

$A_{11} := \text{CHOL\_UNB}(A_{11})$

$A_{12} := \text{TRIU}(A_{11})^{-1} A_{12}$

$A_{22} := A_{22} - A_{12}^{T} A_{12}$

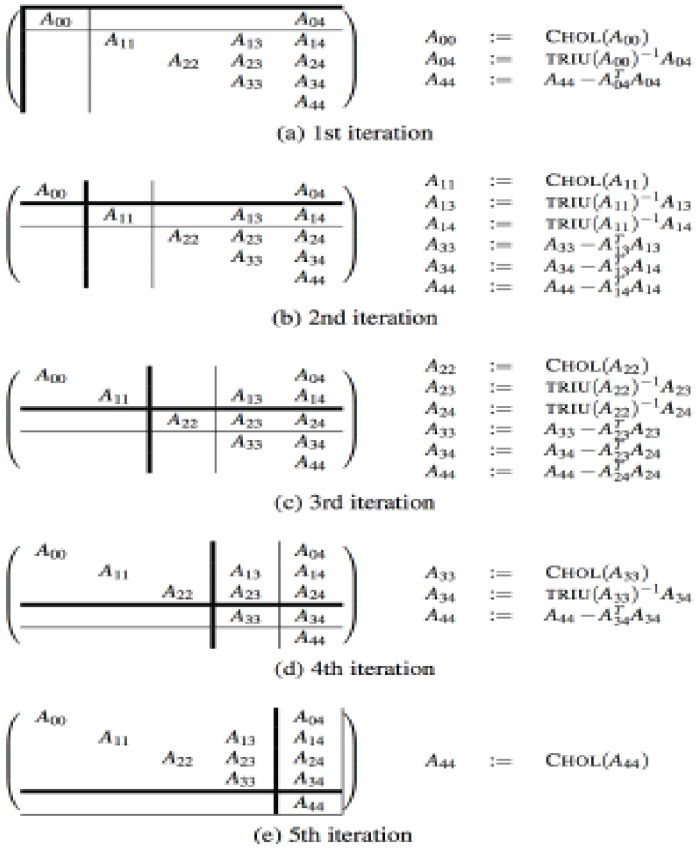
Continue with

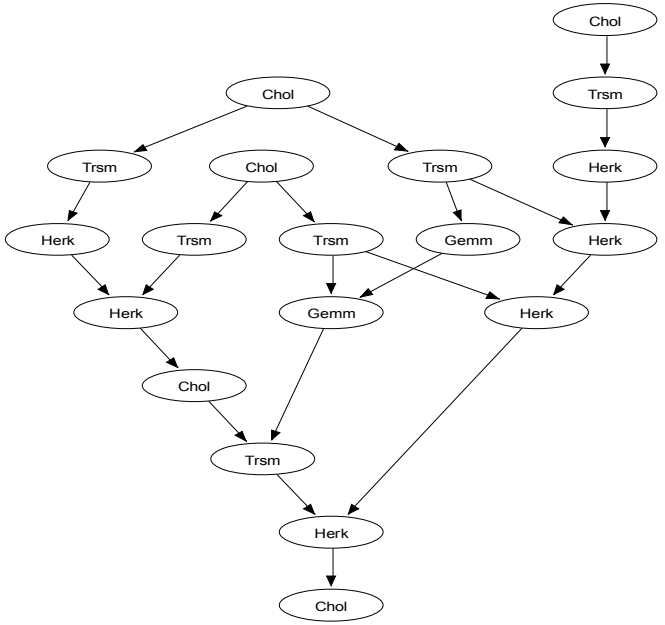
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

- Fine-grained Task-basked, Right Looking Cholesky Factorization
  - 2D layout of blocks based on nested dissection and fill pattern
  - Task-Parallel, Kokkos based implementation
  - Fine-grained parallel algorithm with synchronizations represented as a task DAG
  - Algorithm-by-blocks style algorithm
    - Originally used for parallel out-of-core factorizations (Quintana et al, Buttari et al)
    - Block based algorithm rather than scalar based algorithm
- See "*Task Parallel Incomplete Cholesky Factorization using 2D Partitioned-Block Layout*" (K. Kim, S. Rajamanickam, G. Stelle, H. C. Edwards, S. Olivier) arXiv.

ShyLU/Tacho : Steps in the factorization

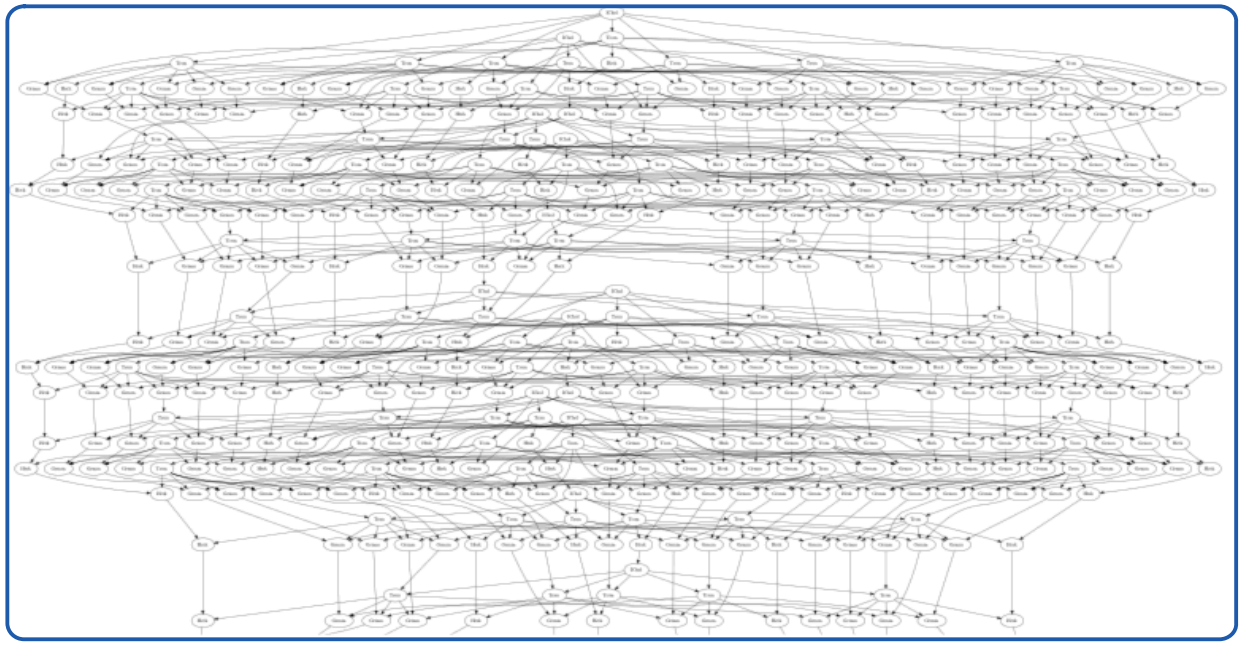




Task DAG

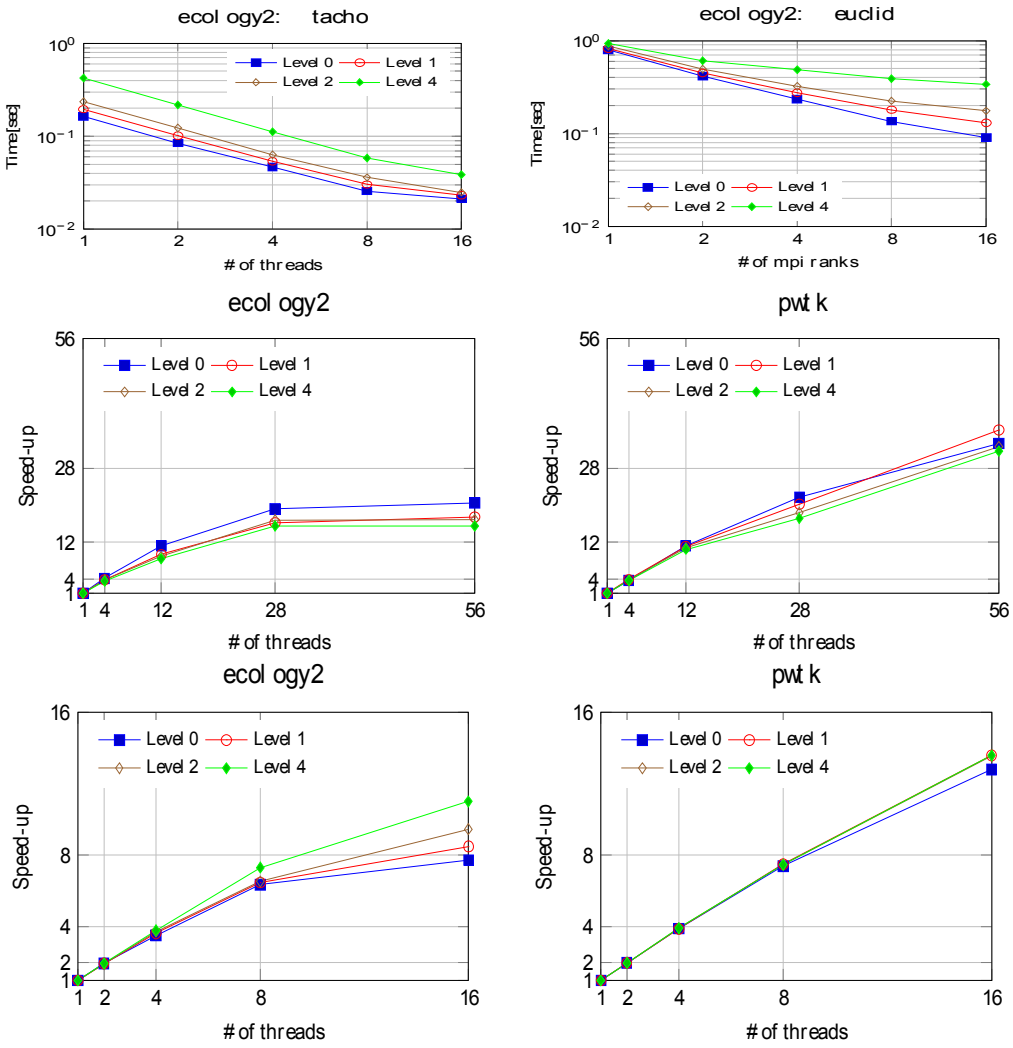
- Degree of Concurrency still depends on nested dissection ordering
- Parallelism is not tied to nested dissection ordering

ShyLU/Tacho : More realistic task DAG



- Complete Task DAG never formed. Shown here for demonstration of the degree of concurrency.
- The concurrency is from fine-grained tasking and a 2D right looking algorithm

ShyLU/Tacho : Experimental Results



- Results shown for two matrices with different levels of fill
- Euclid results shown for reference
  - It is an MPI code, using RCM ordering (best for Euclid)
  - Not many parallel IC(k) codes
- Speedup numbers are in comparison with single threaded Cholesky
  - Small overhead for single threaded Cholesky over serial Cholesky
- Results are shown for both CPU and Xeon Phi architectures
- The two matrices are chosen for very different nnz/n.

ShyLU/Tacho and ShyLU/Basker: Experimental Results