THE DESIGN AND IMPLEMENTATION OF A DENSE LINEAR ALGEBRA LIBRARY FOR EXTREME PARALLEL COMPUTERS

Jack Dongarra, U of Tennessee, ORNL, & U of Manchester
Nicholas Higham, Zounon Mawussi, Samuel Relton, University of Manchester

MANCHESTER 1824
The University of Manchester

University of Manchester


Negin Bagherpour


Jack Dongarra


Sven Hammarling


Nick Higham


Sam Relton


Mawussi Zounon

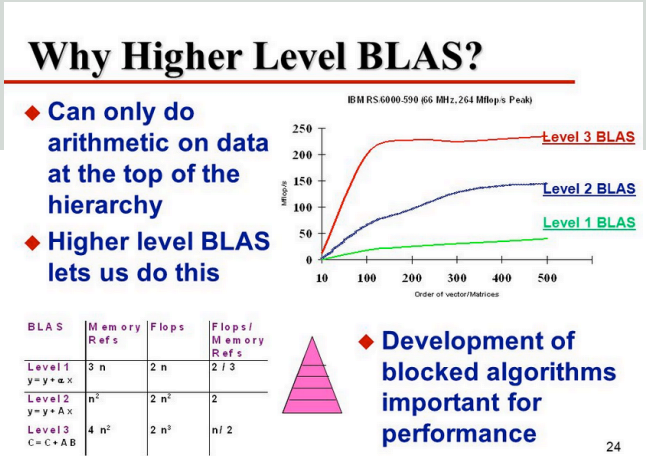
Focused on Dense Linear Algebra Problems

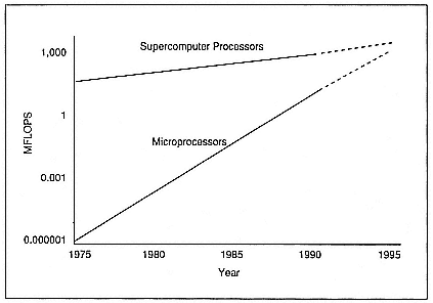
- Linear systems of equations $Ax = b$
- Linear least squares $\min \| b - Ax \|_2$
- Singular value decomposition (SVD) $A = U\Sigma V^T$
- Eigenvalue value problems (EVP) $Ax = \lambda x$

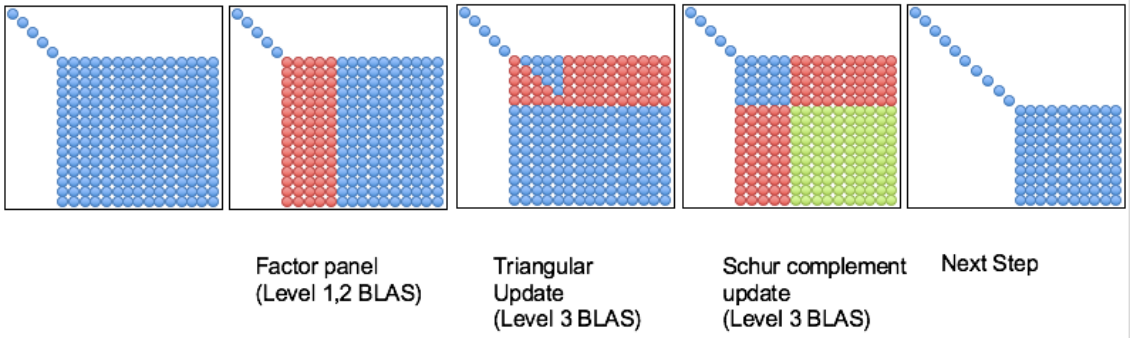
- Dense (square, rectangular)
- Band

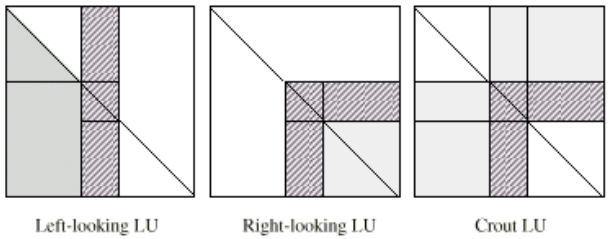
1984 - 1990

- Level 3 BLAS standardization started
- Level 2 BLAS standard published
- “Attack of the Killer Micros”, Brooks @ SC90
- Cache based & SMP machines
- Blocked partitioned algorithms was the way to go
  - Reduce data movement; today’s buzzword “Reduce Communication]”









LAPACK Functionality

Type of Problem	Acronyms
Linear system of equations	SV
Linear least squares problems	LLS
Linear equality-constrained least squares problem	LSE
General linear model problem	GLM
Symmetric eigenproblems	SEP
Nonsymmetric eigenproblems	NEP
Singular value decomposition	SVD
Generalized symmetric definite eigenproblems	GSEP
Generalized nonsymmetric eigenproblems	GNEP
Generalized (or quotient) singular value decomposition	GSVD (QSVD)

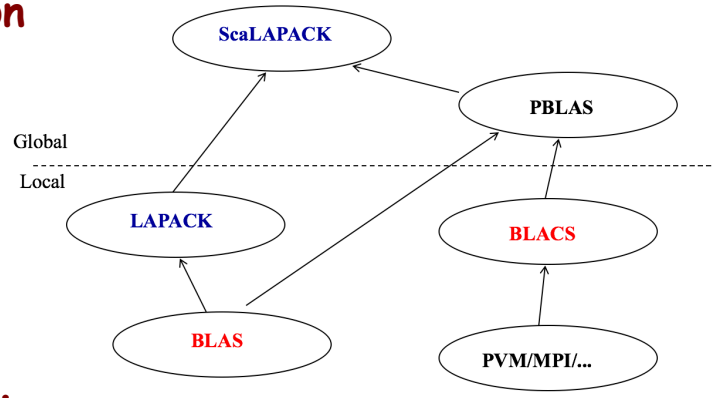
LAPACK Software

Jointly with UTK and UCB and Many Other Contributors

- First release in February **1992 (Silver Anniversary)**
- Current: LAPACK Version 3.7.0 (Dec, 2017) ~2M LoC
- **LICENSE**: Mod-BSD, freely-available software package
- Public **GITHub** repository
- **4 Precisions:** single, double, complex, double complex
  - *Considering 16-bit flpt version*
- **Multi-OS** *nix, Mac OS/X, Windows
- **Multi-build** support (Make and Cmake)
- **Reference BLAS and CBLAS**
- **LAPACKE: Standard C language APIs for LAPACK**
- Prebuilt Libraries for **Windows**
- Extensive test suite
- **Forum and User support**: http://icl.cs.utk.edu/lapack-forum/
- Goal: bug free library – Since 2009, 165 bugs reported, only 11 pending correction

ScaLAPACK Programming Style

- SPMD Fortran 77 using an object based design
- Built on various modules
  - PBLAS Interprocessor communication & computation
    - BLAS
    - BLACS
      - MPI, PVM, IBM SP, CRI T3, Intel, TMC
      - Provides right level of abstraction.



- Object based - Array descriptor
  - Contains information required to establish mapping between a global array entry and its corresponding process and memory location.
  - Provides a flexible framework to easily specify additional data distributions or matrix types.
  - Currently dense, banded, & out-of-core
- Using the concept of context

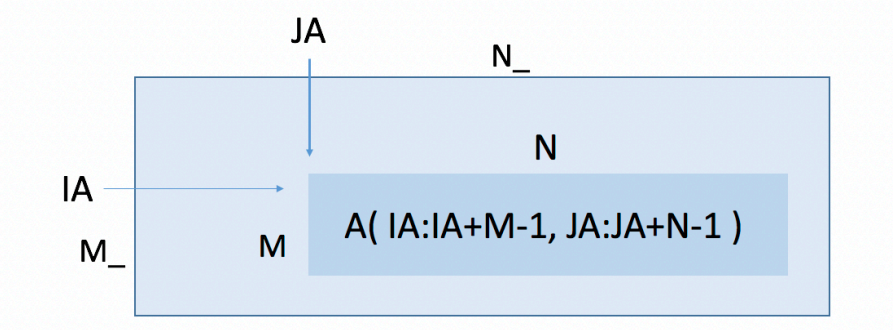
PBLAS

- Similar to the BLAS in functionality and naming.
- Built on the BLAS and BLACS
- Provide global view of matrix

```
CALL DGEXXX   ( M, N, A( IA, JA ), LDA,... )
```

```
CALL PDGEXXX( M, N, A, IA, JA, DESCA,... )
```

- Hides complex local indexing and message passing



Performance Issues with ScaLAPACK

- The major problem with ScaLAPACK is the lack of overlap of computation and communication .
- Each phase done separately, bulk synchronous.
  - Computation phase then a communication phase.
  - All (most) processes compute then a communication phase (broadcast)
  - This is how the PBLAS operate.
- No overlap, resulting in performance issues

- Need an “new” interface which allows computation and communication to take place simultaneously, in an asynchronous fashion.

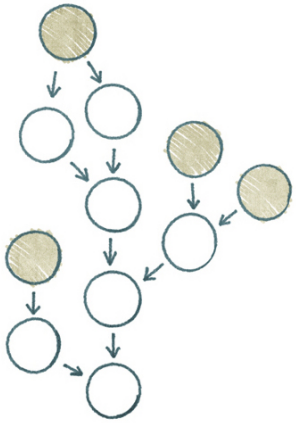
Problems with Sca/LAPACK

software engineering

- Obsolete language (F77)
  - Poor man's object orientation (array descriptors)
  - Manual generation of 4 precisions
    - Hard to accommodate lower (e.g. half) or higher (e.g. double-double)
  - No convenience of memory allocation (e.g. workspaces)
  - Hard to maintain with C/C++ educated personnel

Since LAPACK and ScaLAPACK

- A lot has changed
  - Manycore and accelerators
  - Use a different set of ideas to provide efficient use of underlying hardware
    - PLASMA (Many core)
    - MAGMA (Accelerators)



- dense linear algebra for multicore
  - dataflow scheduling
  - tile matrix layout
  - tile algorithms

NVIDIA GPU CENTER OF EXCELLENCE

NVIDIA's GPU Center of Excellence Program recognizes universities expanding the frontier of massively parallel computing using CUDA.

intel Intel Parallel Computing Center

The objective of the Innovative Computing Laboratory's IPCC is the development and optimization of numerical linear algebra libraries and technologies for applications, while tackling current challenges in heterogeneous Intel® Xeon Phi™ coprocessor-based High Performance Computing.

AMD

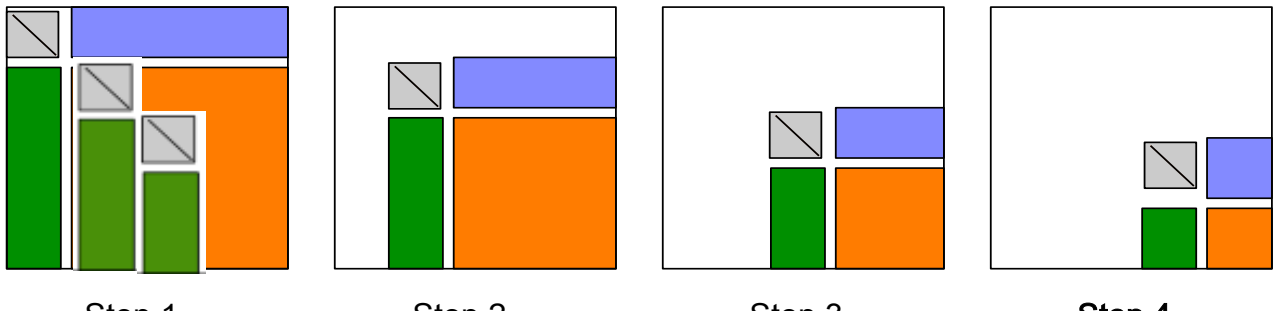
Long-term collaboration and support on the development of clMAGMA, the OpenCL™ port of MAGMA.

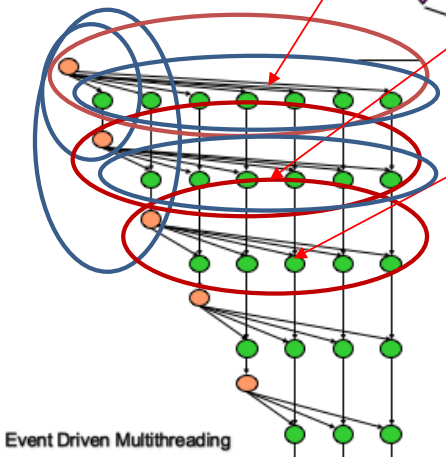
FEATURES AND SUPPORT

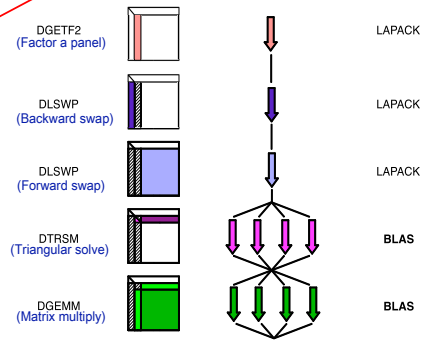
- **MAGMA 2.2** FOR **CUDA**
- **clMAGMA 1.4** FOR **OpenCL**
- **MAGMA MIC 1.4** FOR **Intel Xeon Phi**

CUDA	OpenCL	Intel Xeon Phi	
●	●	●	Linear system solvers
●	●	●	Eigenvalue problem solvers
●	●		Auxiliary BLAS
●			Batched LA
●		●	Sparse LA
●	●	●	CPU Interface
●	●	●	GPU Interface
●	●	●	Multiple precision support
●			Non-GPU-resident factorizations
●	●	●	Multicore and multi-GPU support
●	●	●	LAPACK testing
●	●	●	Linux
●	●		Windows
●	●		Mac OS

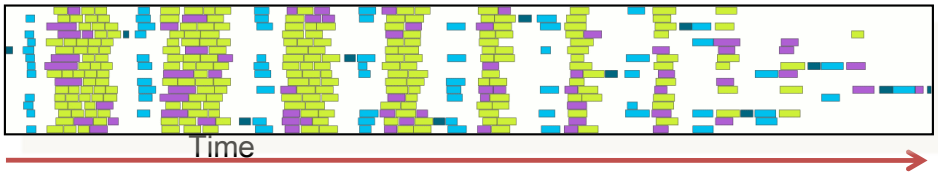
Synchronization (in LAPACK)





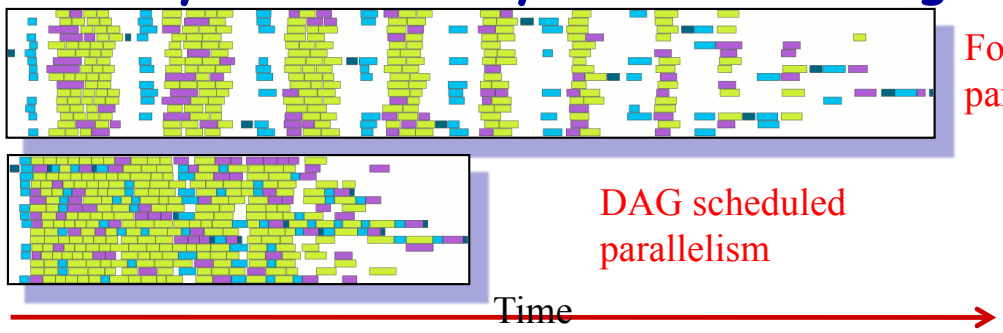


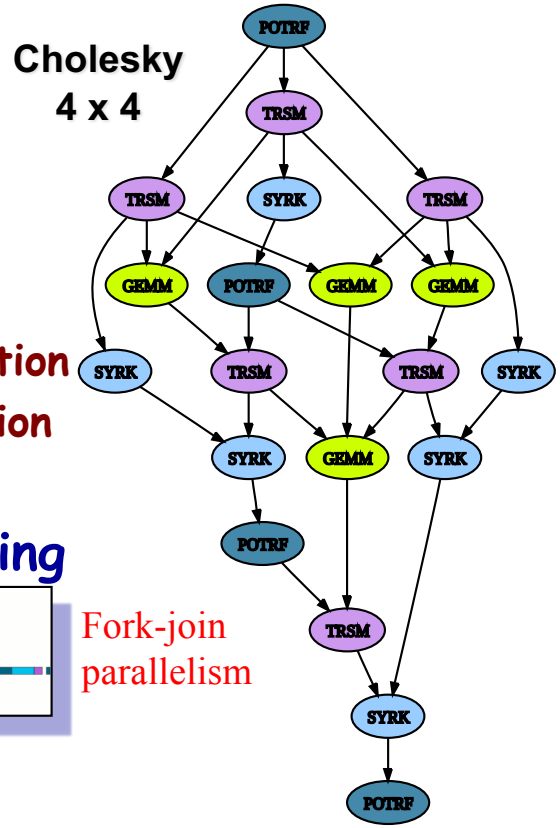
- fork join
- bulk synchronous processing



Parallel Linear Algebra s/w for Multicore Architectures

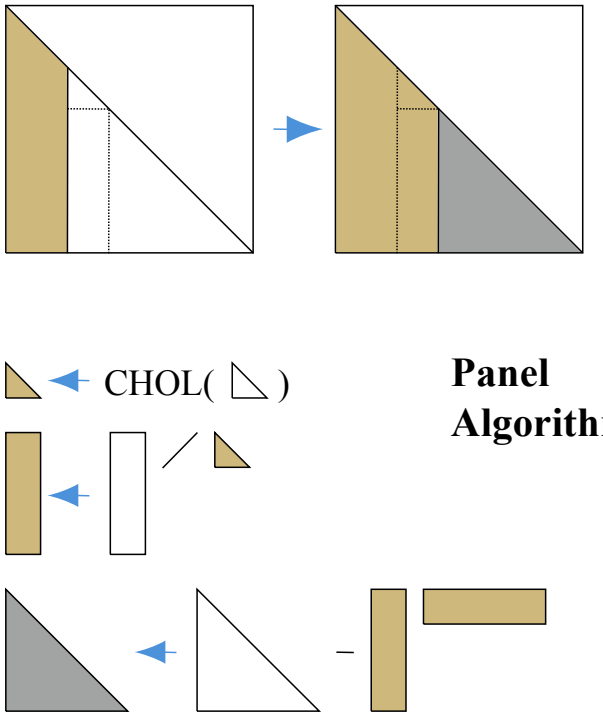
- **Objectives**
  - High utilization of each core
  - Scaling to large number of cores
  - Shared or distributed memory
- **Methodology**
  - Dynamic DAG scheduling
  - Split phases task generation and execution
  - Explicit parallelism/Implicit communication
  - Fine granularity / block data layout
- **Arbitrary DAG with dynamic scheduling**





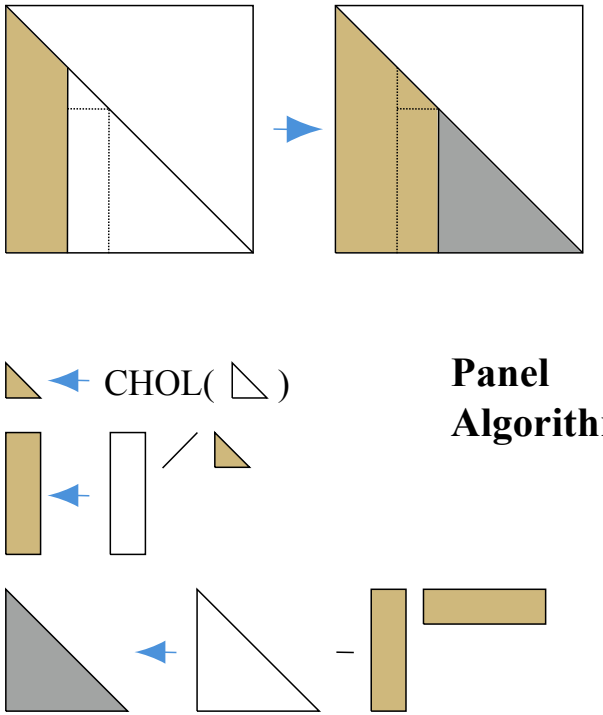
Fork-join
parallelism

LAPACK – Panel Based Algorithms



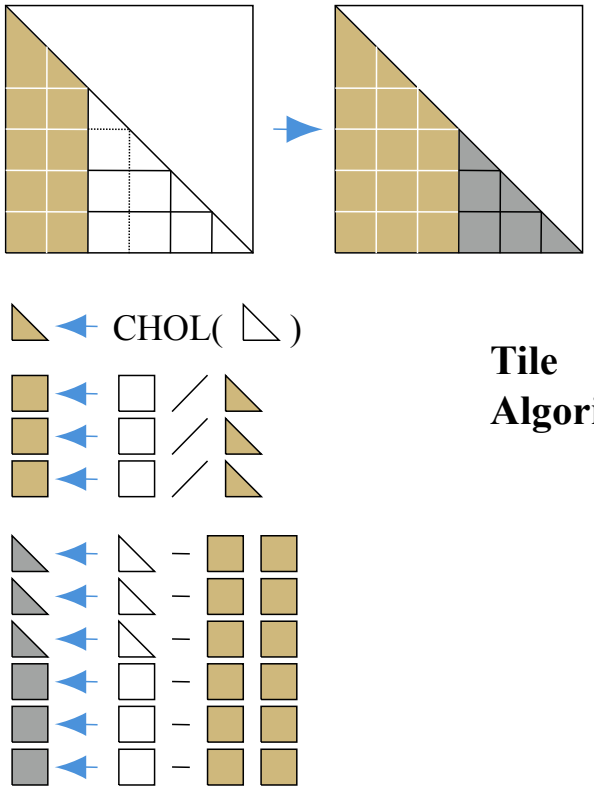
Panel Algorithm

LAPACK – Panel Based Algorithms



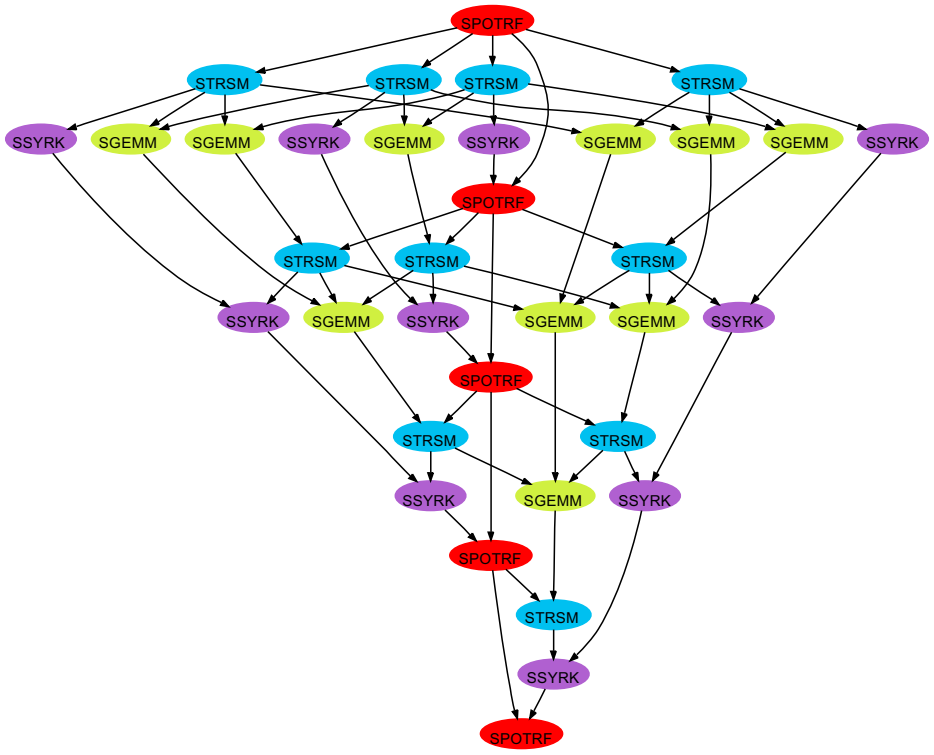
Panel Algorithm

PLASMA – Tile Based Algorithms



Tile Algorithm

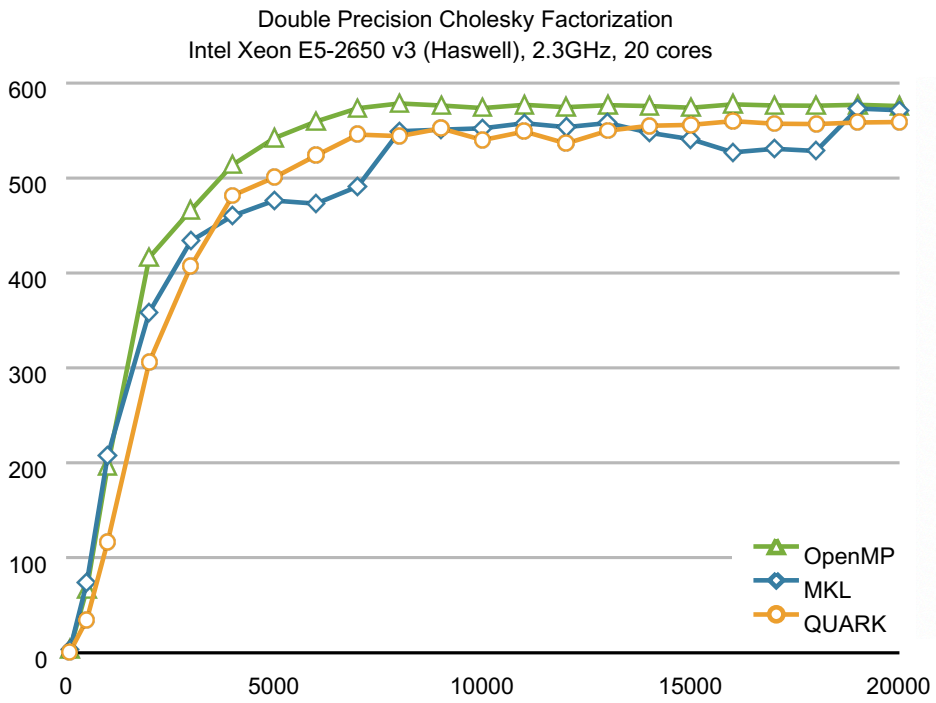
PLASMA – Dataflow Scheduling

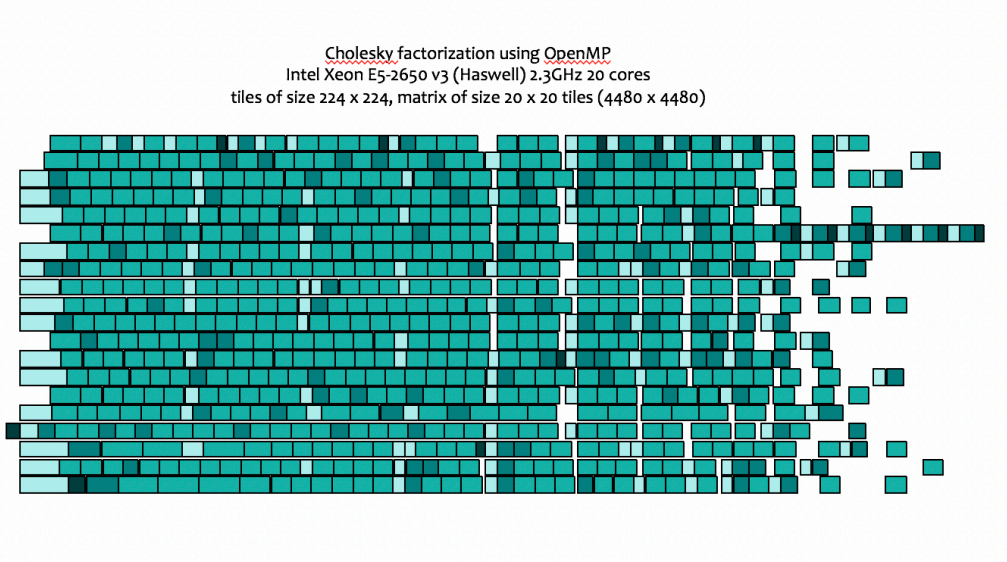


- Exploit parallelism
- Balance load
- Maximize data locality

Reasonable Performance, Good Utilization

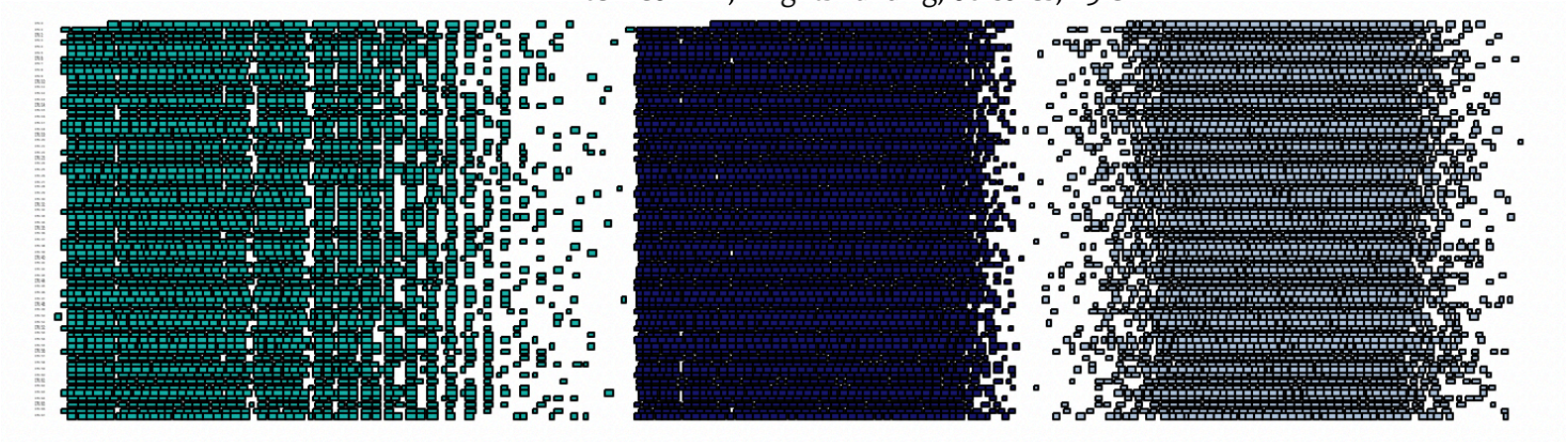
high GFLOPS



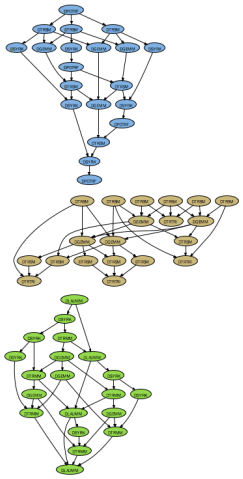


OpenMP – Inverse of the Variance-Covariance Matrix

Cholesky inversion using OpenMP
tiles of size 288 x 288, (7200 x 7200)
Intel Xeon Phi, Knights Landing, 68 cores, 1.3 GHz



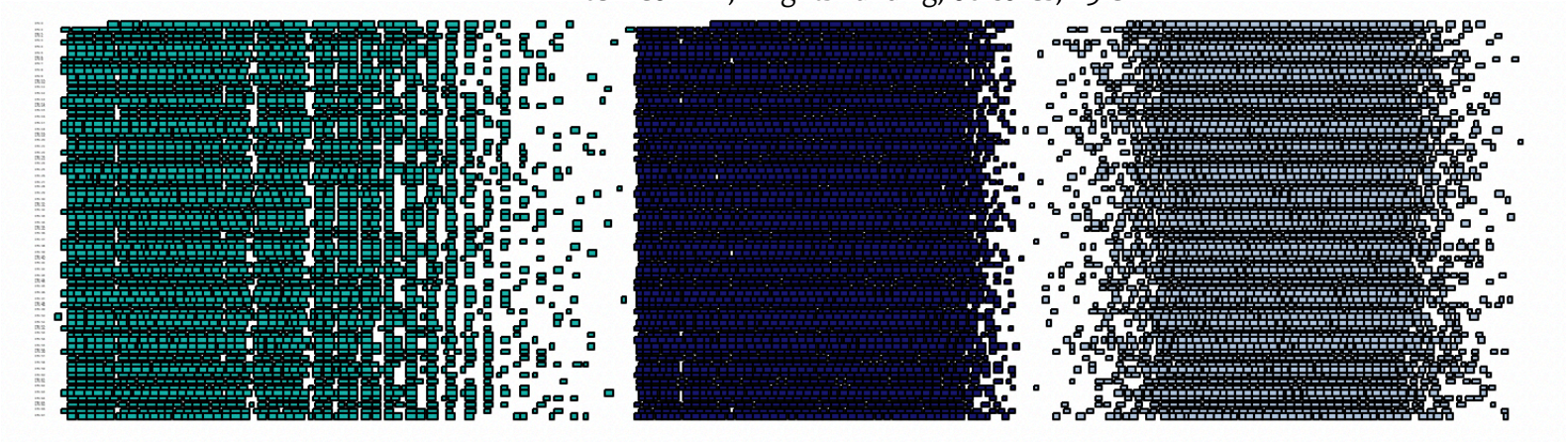
Factor matrix $A = LL^T$ Compute inverse of factor L Computer $A^{-1} = L^{-T}L^{-1}$



sync:
770 Gflop/s

OpenMP – Inverse of the Variance-Covariance Matrix

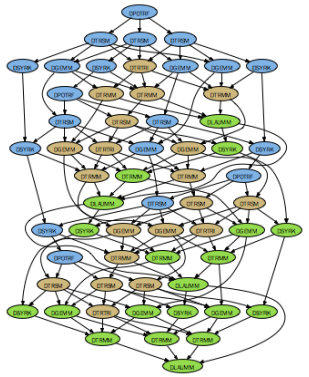
Cholesky inversion using OpenMP
tiles of size 288 x 288, (7200 x 7200)
Intel Xeon Phi, Knights Landing, 68 cores, 1.3 GHz



Factor matrix $A = LL^T$ Compute inverse of factor L Computer $A^{-1} = L^{-T}L^{-1}$

sync:
770 Gflop/s

async:
1001 Gflop/s



```
#pragma omp parallel
#pragma omp master
{
    for (k = 0; k < nt; k++) {
        #pragma omp task depend(inout:A(k, k)[0:nb*nb])
        LAPACKE_dpotrf_work(
            LAPACK_COL_MAJOR,
            'L', nb, A(k, k), nb);

        for (m = k+1; m < nt; m++) {
            #pragma omp task depend(in:A(k, k)[0:nb*nb]) \
                             depend(inout:A(m, k)[0:nb*nb])
            cblas_dtrsm(
                CblasColMajor,
                CblasRight, CblasLower,
                CblasTrans, CblasNonUnit,
                nb, nb,
                1.0, A(k, k), nb,
                     A(m, k), nb);
        }
        for (m = k+1; m < nt; m++) {
            #pragma omp task depend(in:A(m, k)[0:nb*nb]) \
                             depend(inout:A(m, m)[0:nb*nb])
            cblas_dsyrk(
                CblasColMajor,
                CblasLower, CblasNoTrans,
                nb, nb,
                -1.0, A(m, k), nb,
                 1.0, A(m, m), nb);

            for (n = k+1; n < m; n++) {
                #pragma omp task depend(in:A(m, k)[0:nb*nb]) \
                                 depend(in:A(n, k)[0:nb*nb]) \
                                 depend(inout:A(m, n)[0:nb*nb])
                cblas_dgemm(
                    CblasColMajor,
                    CblasNoTrans, CblasTrans,
                    nb, nb, nb,
                    -1.0, A(m, k), nb,
                          A(n, k), nb,
                     1.0, A(m, n), nb);
            }
        }
    }
}
```

```
#pragma omp parallel
#pragma omp master
{
    for (k = 0; k < nt; k++) {

        for (m = k+1; m < nt; m++) {
            #pragma omp task depend(in:A(k, k)[0:nb*nb]) \
                             depend(inout:A(m, k)[0:nb*nb])
            cblas_dtrsm(
                CblasColMajor,
                CblasRight, CblasLower,
                CblasNoTrans, CblasNonUnit,
                nb, nb,
                -1.0, A(k, k), nb,
                      A(m, k), nb);
        }
        for (m = k+1; m < nt; m++) {

            for (n = 0; n < k; n++) {
                #pragma omp task depend(in:A(m, k)[0:nb*nb]) \
                                 depend(in:A(k, n)[0:nb*nb]) \
                                 depend(inout:A(m, n)[0:nb*nb])
                cblas_dgemm(
                    CblasColMajor,
                    CblasNoTrans, CblasNoTrans,
                    nb, nb, nb,
                    1.0, A(m, k), nb,
                         A(k, n), nb,
                    1.0, A(m, n), nb);
            }
        }
        for (n = 0; n < k; n++) {
            #pragma omp task depend(in:A(k, k)[0:nb*nb]) \
                             depend(inout:A(k, n)[0:nb*nb])
            cblas_dtrsm(
                CblasColMajor,
                CblasLeft, CblasLower,
                CblasNoTrans, CblasNonUnit,
                nb, nb,
                1.0, A(k, k), nb,
                     A(k, n), nb);
        }
        #pragma omp task depend(inout:A(k, k)[0:nb*nb])
        LAPACKE_dtrtri_work(
            LAPACK_COL_MAJOR,
            'L', 'N',
            nb, A(k, k), nb);
    }
}
```

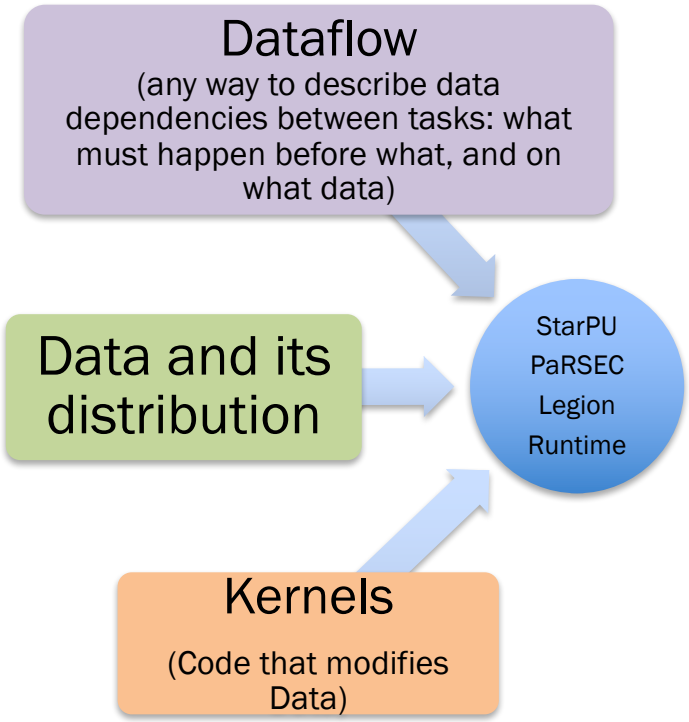
```
#pragma omp parallel
#pragma omp master
{
    for (k = 0; k < nt; k++) {

        for(n = 0; n < k; n++) {
            #pragma omp task depend(in:A(k, n)[0:nb*nb]) \
                             depend(inout:A(n, n)[0:nb*nb])
            cblas_dsyrk(
                CblasColMajor,
                CblasLower, CblasTrans,
                nb, nb,
                1.0, A(k, n), nb,
                1.0, A(n, n), nb);

            for(m = n+1; m < k; m++) {
                #pragma omp task depend(in:A(k, m)[0:nb*nb]) \
                                 depend(in:A(k, n)[0:nb*nb]) \
                                 depend(inout:A(m, n)[0:nb*nb])
                cblas_dgemm(
                    CblasColMajor,
                    CblasTrans, CblasNoTrans,
                    nb, nb, nb,
                    1.0, A(k, m), nb,
                         A(k, n), nb,
                    1.0, A(m, n), nb);
            }
        }
        for (n = 0; n < k; n++) {
            #pragma omp task depend(in:A(k, k)[0:nb*nb]) \
                             depend(inout:A(n, n)[0:nb*nb])
            cblas_dtrmm(
                CblasColMajor,
                CblasLeft, CblasLower,
                CblasTrans, CblasNonUnit,
                nb, nb,
                1.0, A(k, k), nb,
                     A(k, n), nb);
        }
        #pragma omp task depend(inout:A(k, k)[0:nb*nb])
        LAPACKE_dlauum_work(
            LAPACK_COL_MAJOR,
            'L', nb, A(k, k), nb);
    }
}
```

```
#pragma omp parallel
#pragma omp master
{
    for (k = 0; k < nt; k++) {
        #pragma omp task depend(inout:A(k, k)[0:nb*nb])
        LAPACKE_dpotrf_work(
            LAPACK_COL_MAJOR,
            'L', nb, A(k, k), nb);

        for (m = k+1; m < nt; m++) {
            #pragma omp task depend(in:A(k, k)[0:nb*nb]) \
                             depend(inout:A(m, k)[0:nb*nb])
            cblas_dtrsm(
                CblasColMajor,
                CblasRight, CblasLower,
                CblasTrans, CblasNonUnit,
                nb, nb,
                1.0, A(k, k), nb,
                     A(m, k), nb);
        }
        for (m = k+1; m < nt; m++) {
            #pragma omp task depend(in:A(m, k)[0:nb*nb]) \
                             depend(inout:A(m, m)[0:nb*nb])
            cblas_dsyrk(
                CblasColMajor,
                CblasLower, CblasNoTrans,
                nb, nb,
                -1.0, A(m, k), nb,
                 1.0, A(m, m), nb);

            for (n = k+1; n < m; n++) {
                #pragma omp task depend(in:A(m, k)[0:nb*nb]) \
                                 depend(in:A(n, k)[0:nb*nb]) \
                                 depend(inout:A(m, n)[0:nb*nb])
                cblas_dgemm(
                    CblasColMajor,
                    CblasNoTrans, CblasTrans,
                    nb, nb, nb,
                    -1.0, A(m, k), nb,
                          A(n, k), nb,
                     1.0, A(m, n), nb);
            }
        }
    }
```

```
    for (k = 0; k < nt; k++) {

        for (m = k+1; m < nt; m++) {
            #pragma omp task depend(in:A(k, k)[0:nb*nb]) \
                             depend(inout:A(m, k)[0:nb*nb])
            cblas_dtrsm(
                CblasColMajor,
                CblasRight, CblasLower,
                CblasNoTrans, CblasNonUnit,
                nb, nb,
                -1.0, A(k, k), nb,
                      A(m, k), nb);
        }
        for (m = k+1; m < nt; m++) {

            for (n = 0; n < k; n++) {
                #pragma omp task depend(in:A(m, k)[0:nb*nb]) \
                                 depend(in:A(k, n)[0:nb*nb]) \
                                 depend(inout:A(m, n)[0:nb*nb])
                cblas_dgemm(
                    CblasColMajor,
                    CblasNoTrans, CblasNoTrans,
                    nb, nb, nb,
                    1.0, A(m, k), nb,
                         A(k, n), nb,
                    1.0, A(m, n), nb);
            }
        }
        for (n = 0; n < k; n++) {
            #pragma omp task depend(in:A(k, k)[0:nb*nb]) \
                             depend(inout:A(k, n)[0:nb*nb])
            cblas_dtrsm(
                CblasColMajor,
                CblasLeft, CblasLower,
                CblasNoTrans, CblasNonUnit,
                nb, nb,
                1.0, A(k, k), nb,
                     A(k, n), nb);
        }
        #pragma omp task depend(inout:A(k, k)[0:nb*nb])
        LAPACKE_dtrtri_work(
            LAPACK_COL_MAJOR,
            'L', 'N',
            nb, A(k, k), nb);
    }
```

```
    for (k = 0; k < nt; k++) {

        for(n = 0; n < k; n++) {
            #pragma omp task depend(in:A(k, n)[0:nb*nb]) \
                             depend(inout:A(n, n)[0:nb*nb])
            cblas_dsyrk(
                CblasColMajor,
                CblasLower, CblasTrans,
                nb, nb,
                1.0, A(k, n), nb,
                1.0, A(n, n), nb);

            for(m = n+1; m < k; m++) {
                #pragma omp task depend(in:A(k, m)[0:nb*nb]) \
                                 depend(in:A(k, n)[0:nb*nb]) \
                                 depend(inout:A(m, n)[0:nb*nb])
                cblas_dgemm(
                    CblasColMajor,
                    CblasTrans, CblasNoTrans,
                    nb, nb, nb,
                    1.0, A(k, m), nb,
                         A(k, n), nb,
                    1.0, A(m, n), nb);
            }
        }
        for (n = 0; n < k; n++) {
            #pragma omp task depend(in:A(k, k)[0:nb*nb]) \
                             depend(inout:A(n, n)[0:nb*nb])
            cblas_dtrmm(
                CblasColMajor,
                CblasLeft, CblasLower,
                CblasTrans, CblasNonUnit,
                nb, nb,
                1.0, A(k, k), nb,
                     A(k, n), nb);
        }
        #pragma omp task depend(inout:A(k, k)[0:nb*nb])
        LAPACKE_dlauum_work(
            LAPACK_COL_MAJOR,
            'L', nb, A(k, k), nb);
    }
}
```

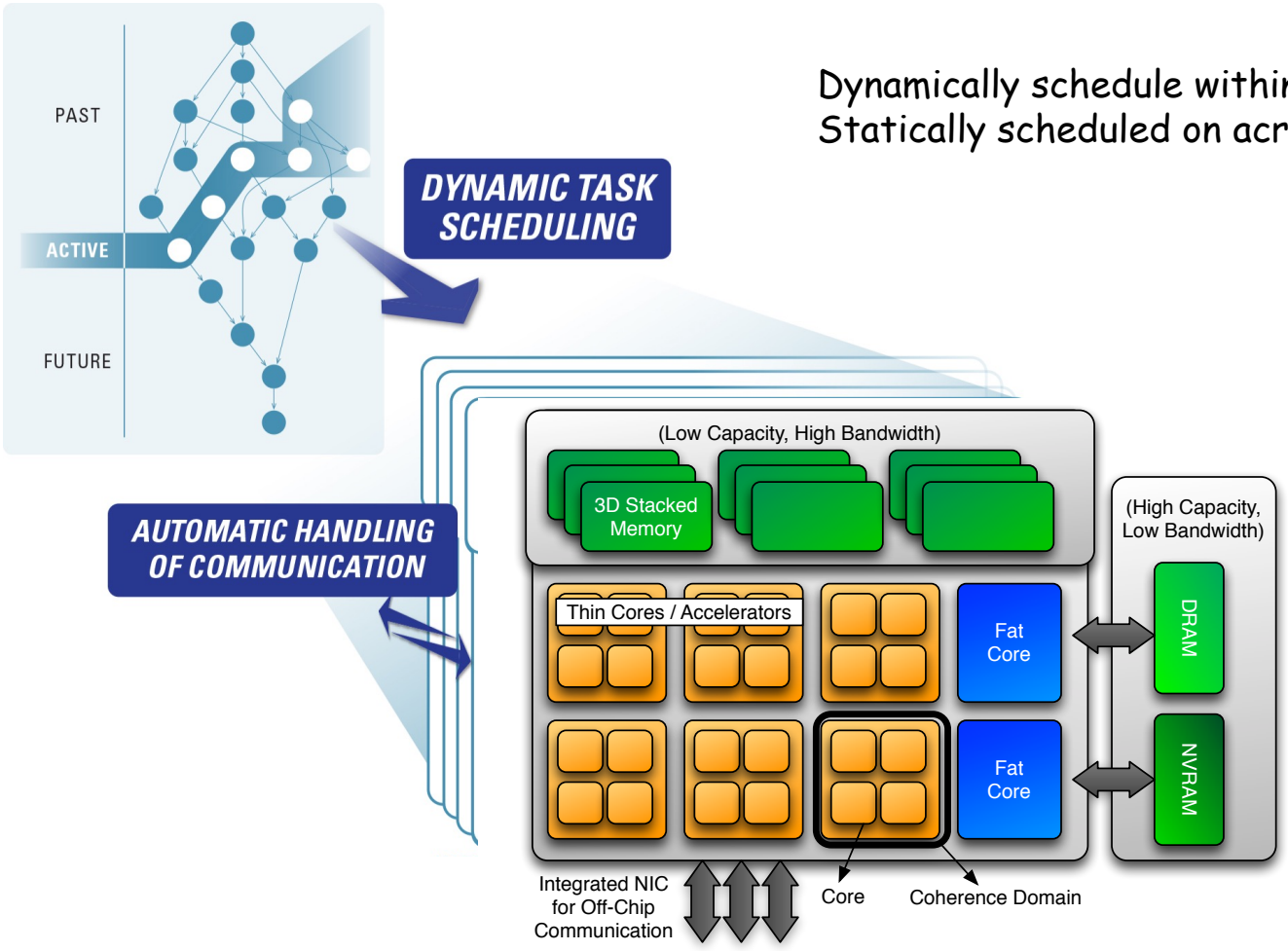
Runtime System Inputs



- Runtime System:
  - Manages local parallelism
  - Schedules tasks on cores and on accelerators
  - Manages memory
  - Adapts the execution to the local hardware (NUMA)
  - Moves data between nodes transparently and asynchronously

At the Node Level

Dynamically schedule within node;
Statically scheduled on across nodes;

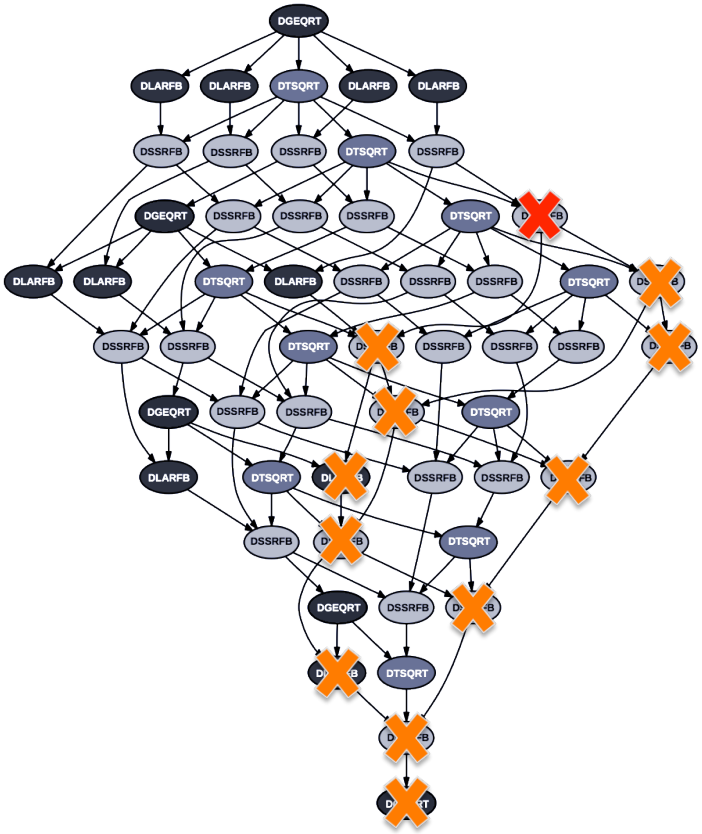


DLA NLAFET Features

- Runtime interface
  - Use Open-MP
  - Be able to plug into other systems
    - Darma, PaRSEC, Legion, StarPU, …
    - Statically scheduled on across nodes; dynamically schedule within node
- Tiled Algorithms
  - Runtime scheduling based on dataflow
  - Runtime dependency tracking
    - Plug into the different runtime systems
- Data distribution as in ScaLAPACK
  - Given the layout and arrangement of processes communication is understood
- Task based parallelism as in PLASMA
  - DAG based to allow overlap of computation and communication
- Ability to use accelerators as in MAGMA
  - Hybrid computing using the runtime system

Resilience

automatic error recovery



- A fault propagates in the system according to data dependencies.
- If the original data can be recovered, possibly using burst buffers, "automatic fault recovery" maybe possible.

NLAFET DLA: Summary

In all 4 precisions, the current NLAFET Dense Software contains:

- 115,000 lines of code, plus another 37,000 lines of testing code
- 400 API functions

Besides porting PLASMA from QUARK to OpenMP, the teams at Manchester and UTK also developed:

- New LU factorization with internally blocked and multithreaded panel factorization
- New QR/LQ factorization routines offering a couple of different tree reduction patterns (CA)
- New Aasen implementation for symmetric indefinite factorization (CA)
- New mixed precision iterative refinement routines – now following LAPACK's procedure
- New matrix norms routines - reimplemented to allow for increased multithreading
- Band linear systems routines for LU and Cholesky factorizations and solves
- 2 stage bi-diagonaliztion in the works

NLAFET DLA: Summary

In all 4 precisions, the current NLAFET Dense Software contains:

- 115,000 lines of code, plus another 37,000 lines of testing code
- 400 API functions

Besides porting PLASMA from QUARK to OpenMP, the teams at Manchester and UTK also developed:

- New LU factorization with internally blocked and multithreaded panel factorization
- New QR/LQ factorization routines offering a couple of different tree reduction patterns (CA)
- New Aasen implementation for symmetric indefinite factorization (CA)
- New mixed precision iterative refinement routines – now following LAPACK's procedure
- New matrix norms routines - reimplemented to allow for increased multithreading
- Band linear systems routines for LU and Cholesky factorizations and solves
- 2 stage bi-diagonaliztion in the works

- New testing/timing harness - to allow for easy inner/outer iteration over the parameter space
- New build system - heavily borrowing from MAGMA
- New precision generation system - heavily borrowing from MAGMA
- Integration with Lua to facilitate autotuning capabilities
- Embedded tracing capabilities
- Fortran interface
- **Software available: https://bitbucket.org/icl/plasma**

The Team

Manchester & Tennessee

University of Manchester

Negin Bagherpour

Jack Dongarra

Sven Hammarling

Nick Higham

Sam Relton

Mawussi Zounon

University or Tennessee

Ahmad Abdelfattah

Jakub Kurzak

Mark Gates

Ichi Yamazaki

Azzam Haidar

Stan Tomov

Piotr Luszczek

Some Related Talks

- Wednesday 9:10-9:30 MS150, 9:10-10:50, Room 212
- Toward Distributed Eigenvalue and Singular Value Solver Using Data Flow System
- Innovative Algorithms for Eigenvalue and Singular Value Decomposition - Part I of I

- Wednesday, March 1 MS177, 1:30 PM - 3:10 PM Room: 212
- Innovative Algorithms for Eigenvalue and Singular Value Decomposition - Part II of II
- 1:55-2:15 Accelerating the Singular Value Decomposition with a Hybrid Two-Stage Algorithm

- Thursday, March 2 MS150; 9:10 AM - 10:50 AM Room: 212
- Innovative Algorithms for Eigenvalue and Singular Value Decomposition - Part I of II
- 9:10-9:30 Toward Distributed Eigenvalue and Singular Value Solver Using Data Flow System

- Thursday, March 2 MS203; 0:00 AM - 11:40 AM Room: 210
- Programming Challenges for Portable Performance on Heterogeneous Architectures
- 10:00-10:20 Programming Model, Performance Analysis and Optimization Techniques for the Intel Knights Landing Xeon Phi