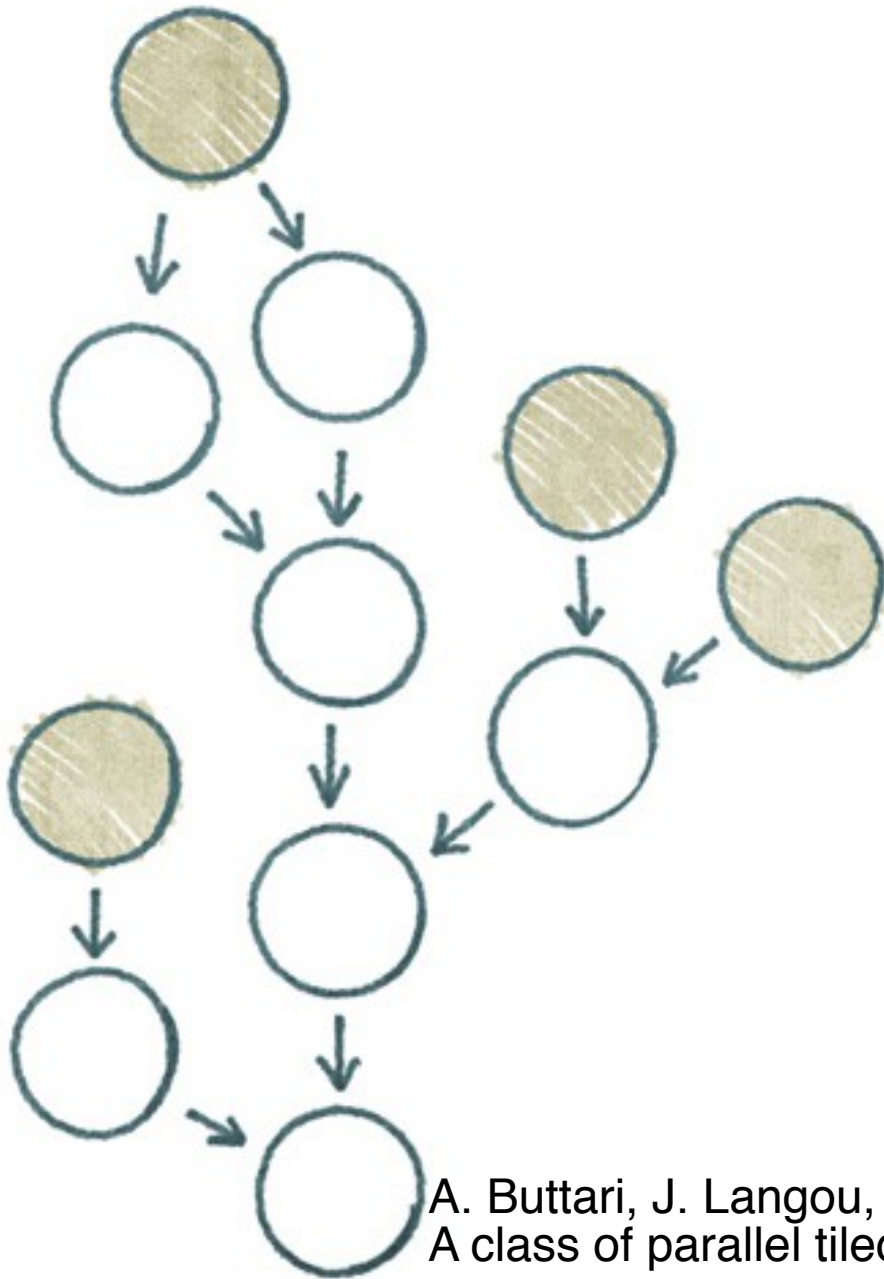


## The PLASMA Library for Tile Linear Algebra Algorithms

# PLASMA Concept



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

A. Buttari, J. Langou, J. Kurzak, J. Dongarra,  
A class of parallel tiled linear algebra algorithms for multicore architectures,  
Parallel Computing, 35(1):38-53, 2009. DOI: [10.1016/j.parco.2008.10.002](https://doi.org/10.1016/j.parco.2008.10.002)

<http://icl.cs.utk.edu/plasma/>

<https://bitbucket.org/icl/plasma>

→ Documentation (Doxygen)

→ Get Assistance (Google Group)

The screenshot shows a web browser window displaying the Bitbucket repository page for 'Parallel Linear Algebra Software for Multicore Architectures'. The browser's address bar shows 'Atlassian, Inc.'. The repository page includes a navigation sidebar on the left with icons for home, search, and other repository features. The main content area displays the repository name, affiliation with the University of Tennessee (US) and University of Manchester (UK), and a list of links: About, Documentation, Getting Assistance, Citing, Funding, People, and License. Below this is an 'About' section with a detailed description of the PLASMA software package and its transition from QUARK to OpenMP. A URL is provided for downloading the plasma\_2.8.0.tar.gz file. On the right side, a commit history sidebar shows several recent commits by Jakub Kurzak, including pushes to icl/plasma with commit hashes like ea0c4f3, efc9b69, f01908a, 763aaa1, 5fc9cb6, da9833c, d62af66, 03ae51b, 972d169, and another commit.

# Calling PLASMA – Basic Layout

```
// A is the matrix (pointer)
```

```
info = LAPACKE_dpotrf (layout, uplo, n, A, lda);
```

```
// A is the matrix (pointer)
```

```
info = plasma_dpotrf (layout, uplo, n, A, lda);
```

```
// A is a descriptor (matrix object)
```

```
plasma_omp_dpotrf(          uplo, A, sequence, request);
```

# Calling PLASMA – Full Example

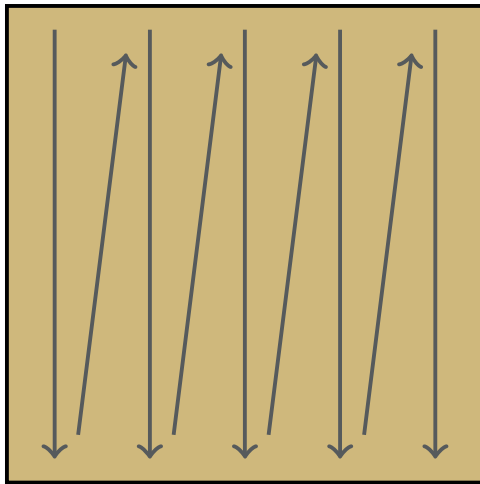
```
plasma_init();
plasma_desc_general_create(..., &A);
plasma_sequence_create(&sequence);
plasma_request_t request = PlasmaRequestInitializer;

#pragma omp parallel
#pragma omp master
{
    plasma_omp_zge2desc(pA, lda, A, sequence, &request);
    plasma_omp_dpotrf(uplo, A, sequence, &request);
    plasma_omp_zdesc2ge(A, pA, lda, sequence, &request);
}

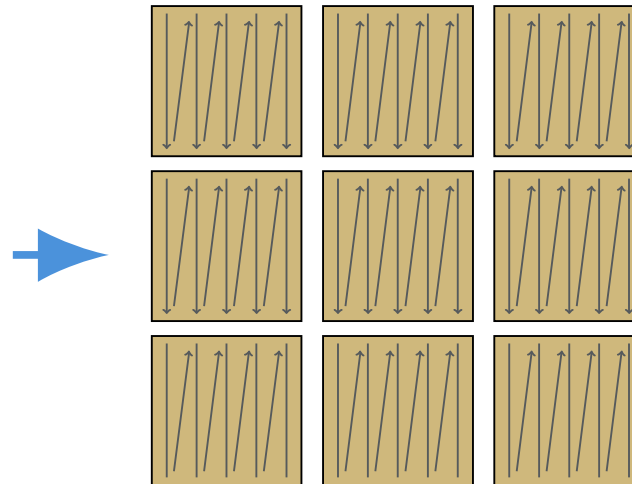
plasma_desc_destroy(&A);
plasma_sequence_destroy(sequence);
plasma_finalize();
```

# PLASMA and Tile Matrix Layout

LAPACK Layout



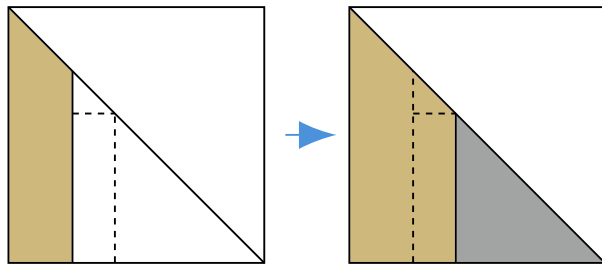
Tile Layout




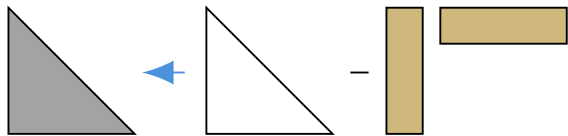
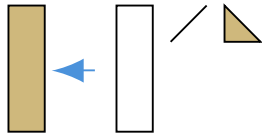
- defines the unit of parallelism
- enables dataflow scheduling
- helps memory efficiency
- simplifies communication

F. Gustavson, L. Karlsson, Bo Kågström, Parallel and Cache-Efficient In-Place Matrix Storage Format Conversion, ACM Transactions on Mathematical Software (TOMS), 38(3), Article No. 17, 2012. DOI: [10.1145/2168773.2168775](https://doi.org/10.1145/2168773.2168775)

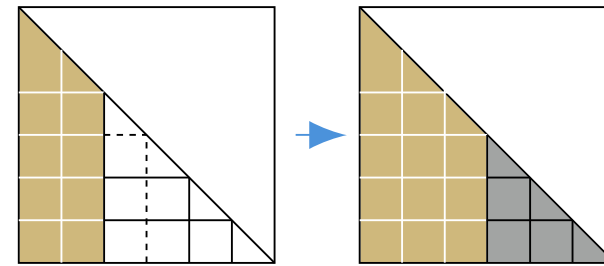
# PLASMA Tile Algorithms




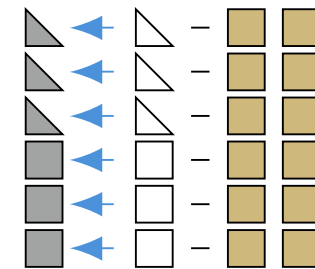
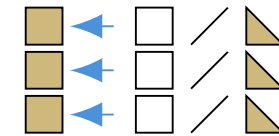
  $\leftarrow$  CHOL( $\triangle$ )



**LAPACK  
Algorithm**



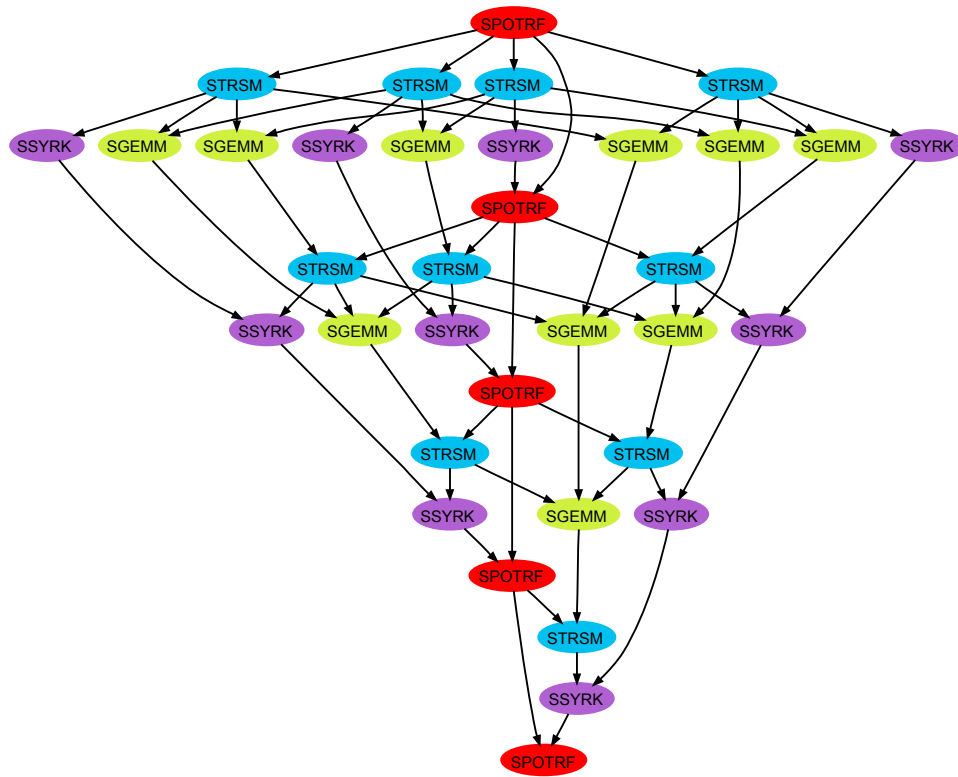
  $\leftarrow$  CHOL( $\triangle$ )



**Tile  
Algorithm**

H. Bouwmeester,  
Tiled Algorithms for Matrix Computations on Multicore Architectures,  
PhD dissertation, University of Colorado Denver, 2012. [arxiv.org/abs/1303.3182](http://arxiv.org/abs/1303.3182)

# PLASMA and Dataflow Scheduling

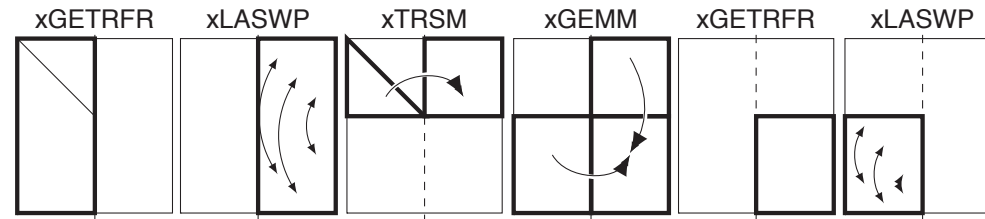
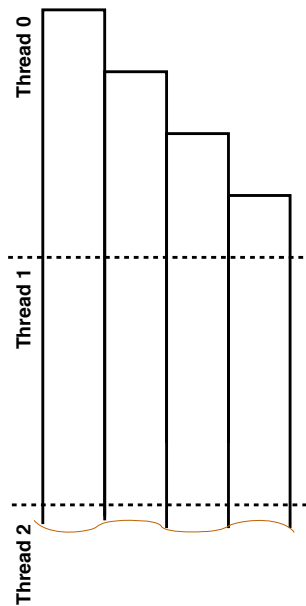


- exploit parallelism
- balance load
- maximize data locality

J. Kurzak, H. Ltaief, J. Dongarra, R.M. Badia, Scheduling dense linear algebra operations on multicore processors, *Concurrency and Computation: Practice and Experience*, 22(1):15-44, 2010. DOI: [10.1002/cpe.1467](https://doi.org/10.1002/cpe.1467)



# PLASMA Algorithms: LU Factorization



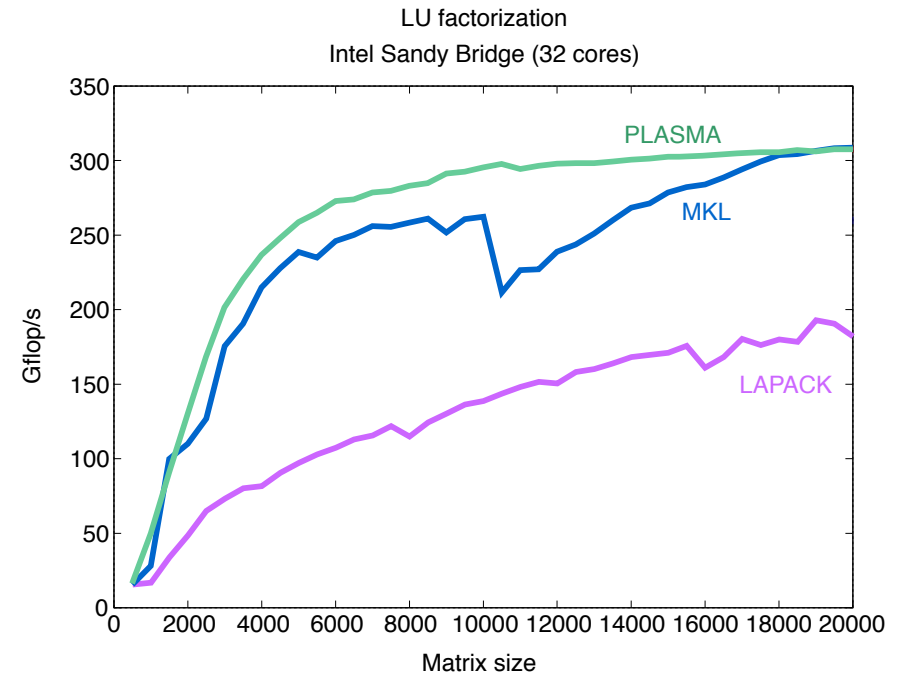
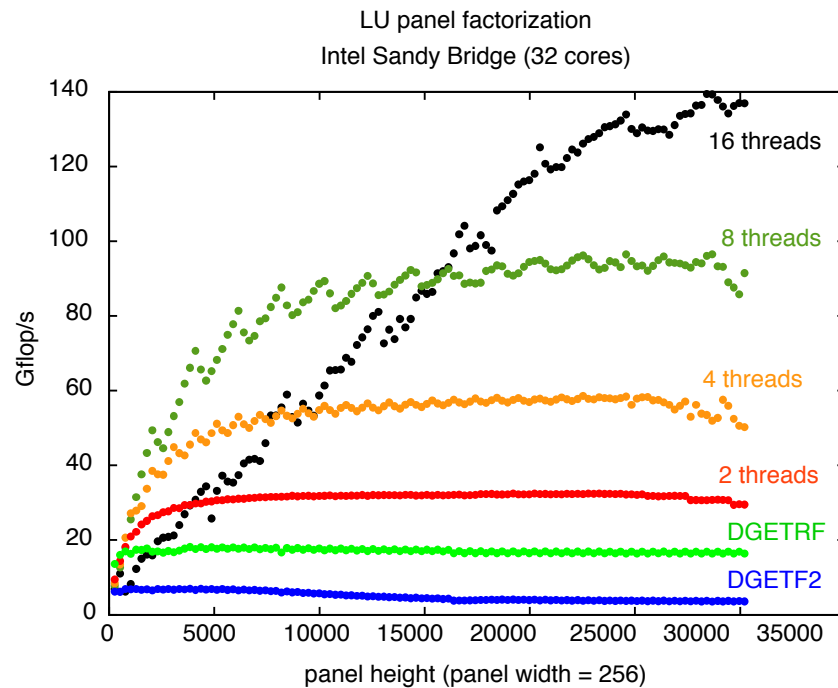
## Fast & scalable panel factorization

- multithreaded
- cache resident
- recursive

A.M. Castaldo, R.C. Whaley, S. Samuel, Scaling LAPACK panel operations using parallel cache assignment, ACM Transactions on Mathematical Software (TOMS), 39(4), Article No. 22, 2013. DOI: [10.1145/2491491.2491492](https://doi.org/10.1145/2491491.2491492)

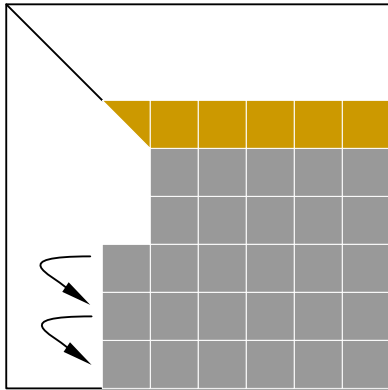
J. Dongarra, M. Faverge, H. Ltaief, P. Luszczek, Achieving numerical accuracy and high performance using recursive tile LU factorization with partial pivoting, Concurrency and Computation: Practice and Experience, 26(7):1408-1431, 2014. DOI: [10.1002/cpe.3110](https://doi.org/10.1002/cpe.3110)

# PLASMA Algorithms: LU Performance

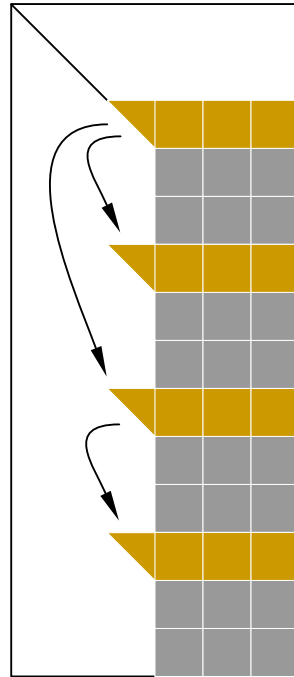


S. Donfack, J. Dongarra, M. Faverge, M. Gates, J. Kurzak, P. Luszczek, I. Yamazaki, A survey of recent developments in parallel implementations of Gaussian elimination, *Concurrency and Computation: Practice and Experience*, 27(5):1292–1309, 2015. DOI: [10.1002/cpe.3110](https://doi.org/10.1002/cpe.3110)

# PLASMA – Algorithms – QR



**Tile QR  
(incremental)**



**TSQR / CAQR  
(tree reduction)**

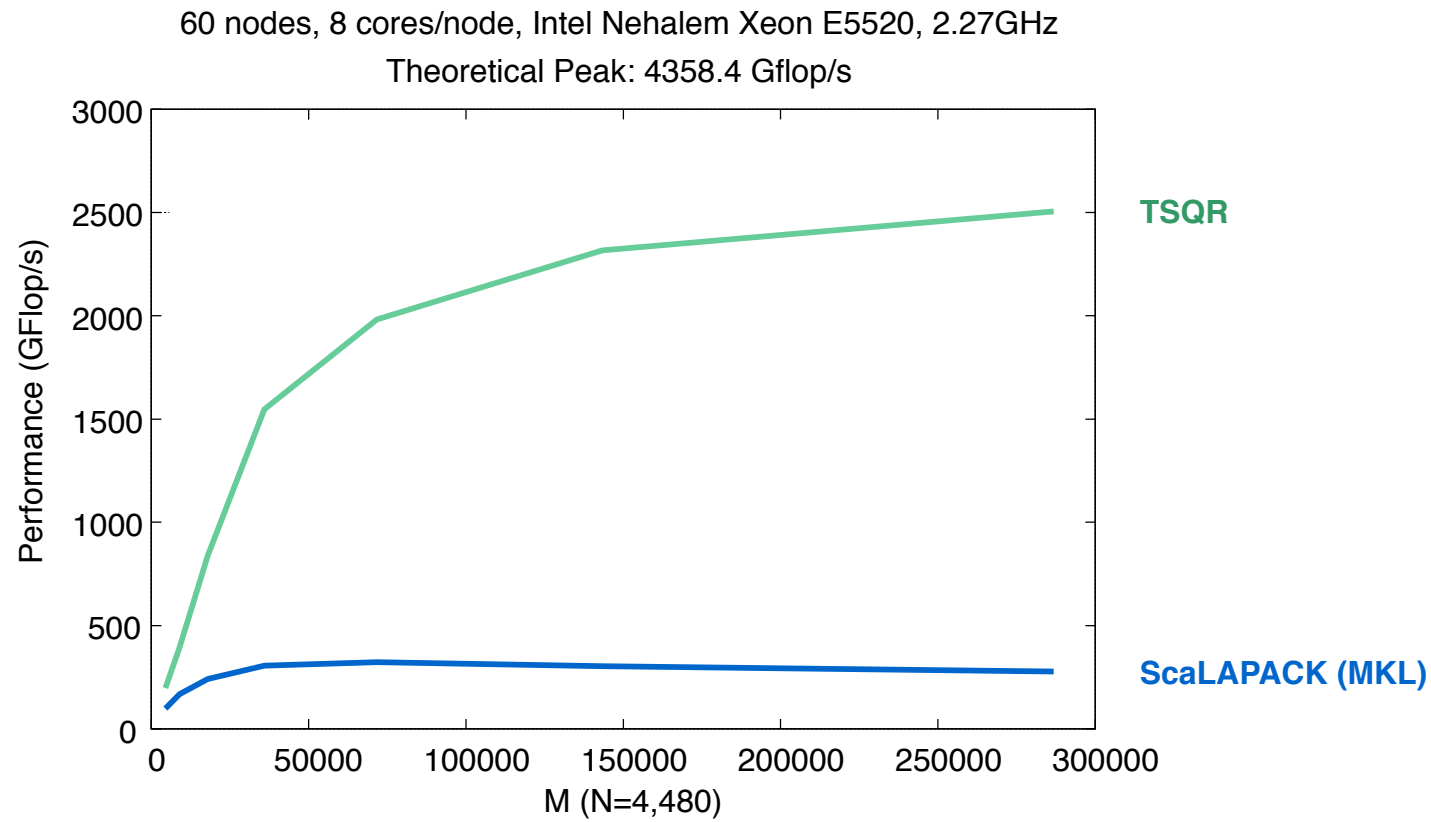
## Tile QR

- great for square matrices
- great for multicore processors

## TSQR / CAQR

- great for tall and skinny matrices
- great for distributed memory

# PLASMA – Algorithms – QR – Performance



# PLASMA – Algorithms – QR – References

J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-optimal Parallel and Sequential QR and LU Factorizations, *SIAM Journal on Scientific Computing*, 34(1):A206-A239, 2012. DOI: [10.1137/080731992](https://doi.org/10.1137/080731992)

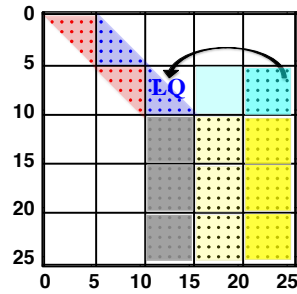
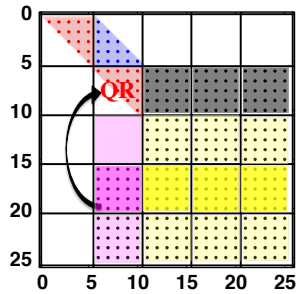
J. Dongarra, M. Faverge, T. Herault, M. Jacquelin, J. Langou, Y. Robert, Hierarchical QR factorization algorithms for multi-core clusters, 39(4-5):212–232, 2013. DOI: [10.1016/j.parco.2013.01.003](https://doi.org/10.1016/j.parco.2013.01.003)

H. Bouwmeester, M. Jacquelin, J. Langou, Y. Robert, Tiled QR factorization algorithms, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2011. [arxiv.org/abs/1104.4475](https://arxiv.org/abs/1104.4475)

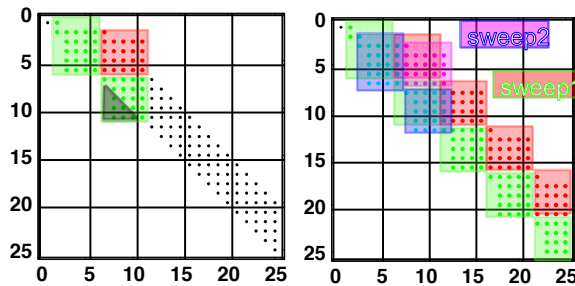
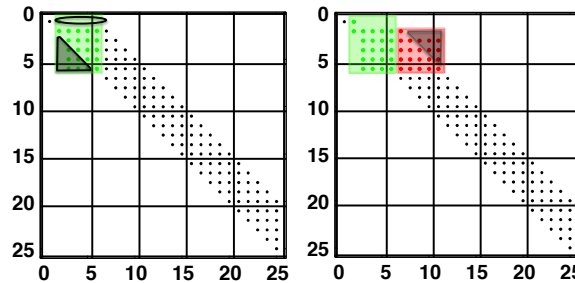
B. Hadri, H. Ltaief, E. Agullo, J. Dongarra, Tile QR factorization with parallel panel processing for multicore architectures. *Proceedings of the International Parallel & Distributed Processing Symposium (IPDPS)*, 2010. DOI: [10.1109/IPDPS.2010.5470443](https://doi.org/10.1109/IPDPS.2010.5470443)

J. Demmel, L. Grigori, M. Hoemmen, J. Langou, Communication-avoiding parallel and sequential QR factorizations, Technical Report No. UCB/EECS-2008-74, University of California Berkeley, 2008. [digitalassets.lib.berkeley.edu/techreports/ucb/text/EECS-2008-74.pdf](https://digitalassets.lib.berkeley.edu/techreports/ucb/text/EECS-2008-74.pdf)

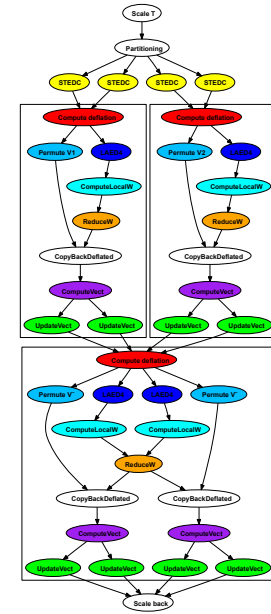
# PLASMA – Algorithms – SVD/EVP (symmetric)



reduction to band  
parallel & cache efficient  
tile algorithm

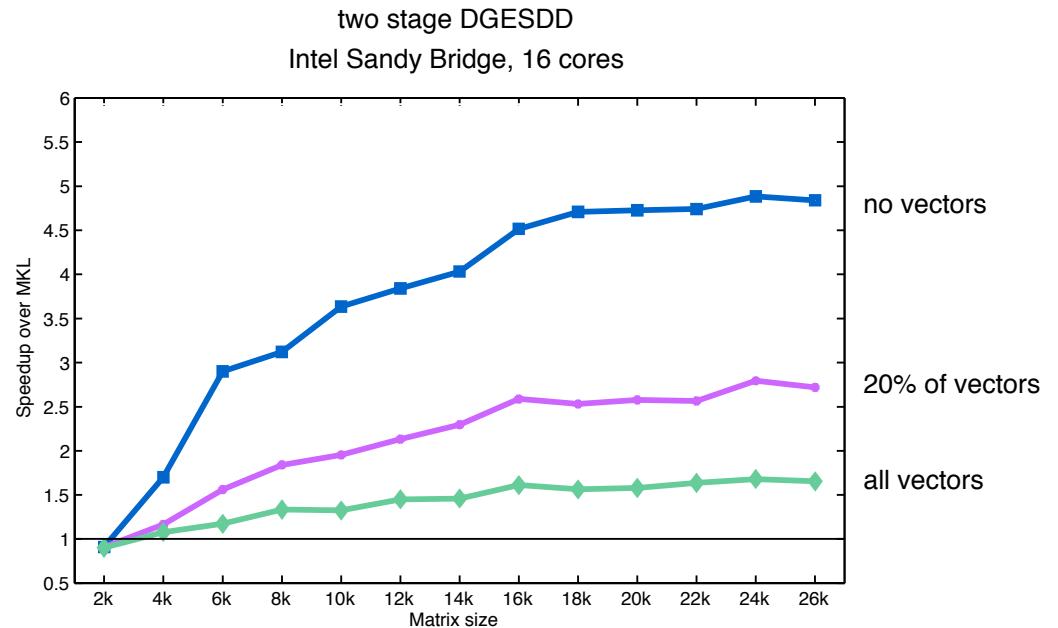
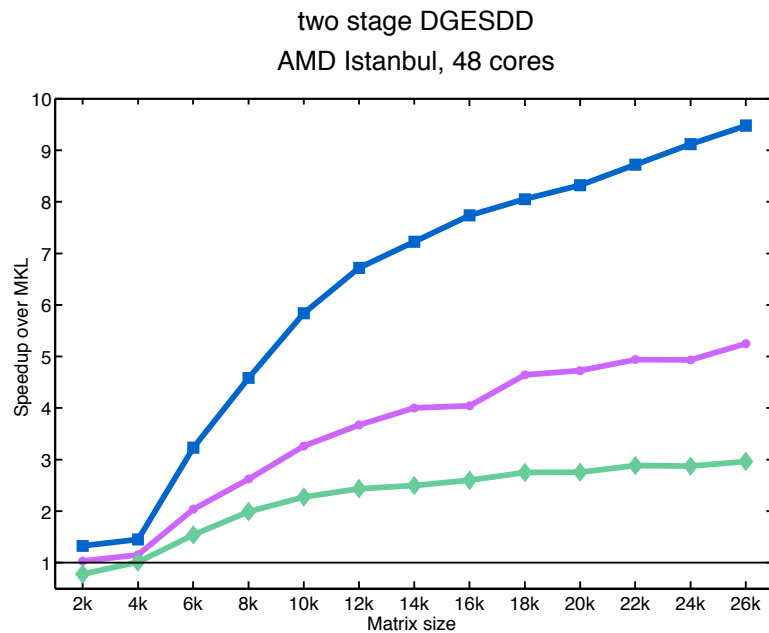


band reduction  
parallel & cache efficient  
a flavor of communication avoiding



divide and conquer  
task-based  
dataflow

# PLASMA – Algorithms –SVD/EVP – Performance



A. Haidar, J. Kurzak, P. Luszczek, An improved parallel singular value algorithm and its implementation for multicore hardware, Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), Article No. 90, 2013. DOI: [10.1145/2503210.2503292](https://doi.org/10.1145/2503210.2503292)

G. Pichon, A. Haidar, M. Faverge, J. Kurzak, Divide and Conquer Symmetric Tridiagonal Eigensolver for Multicore Architectures, Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS), pages 51-60, 2015. DOI: [10.1109/IPDPS.2015.51](https://doi.org/10.1109/IPDPS.2015.51)

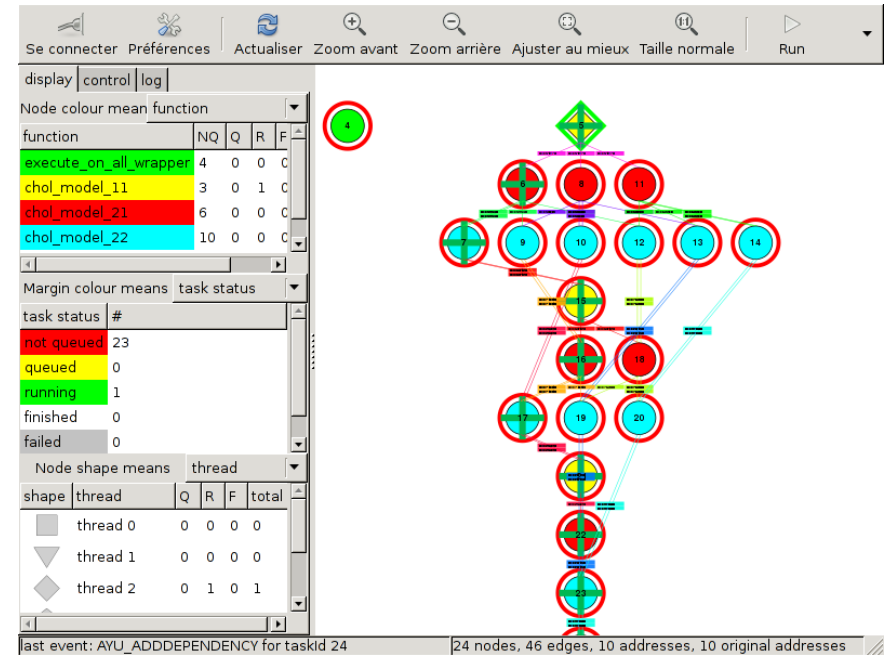
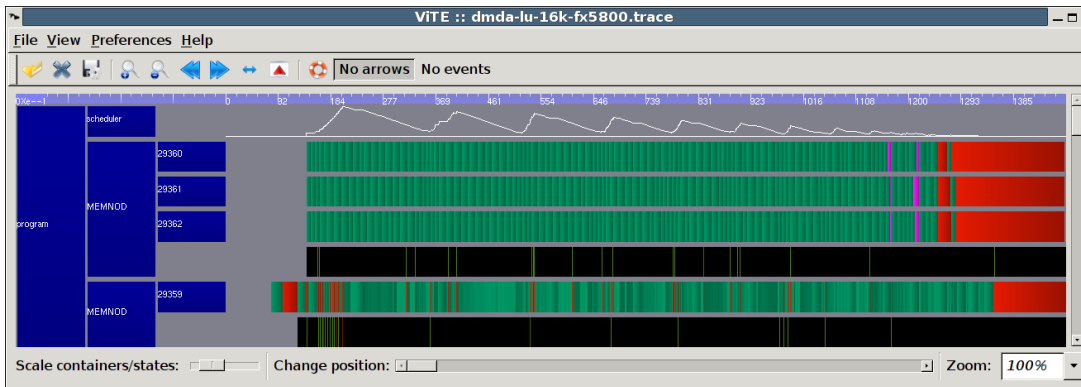
# Dynamic Scheduling

- Jade Stanford University
- Legion Stanford University
- SMPSs / OMPSs Barcelona Supercomputer Center
- StarPU INRIA Bordeaux
- PaRSEC University of Tennessee
- QUARK University of Tennessee
- SuperGlue & DuctTEiP Uppsala University



# Dynamic Scheduling – Tools

- DAG visualization
- tracing
- MPI support
- accelerator support
- simulation



# Dynamic Scheduling – References

<http://suif.stanford.edu/jade.html>

<https://pm.bsc.es/ompss>

<http://starpu.gforge.inria.fr>

<http://icl.cs.utk.edu/quark/>

<https://github.com/tillenius/superglue>

M. C. Rinard, M. S. Lam, The design, implementation, and evaluation of Jade, *ACM Transactions on Programming Languages and Systems*, 20(1):1-64, 1998. DOI: 10.1145/291889.291893

A. Duran, E. Ayguadé, R. M. Badia, J. Labarta, L. Martinell, X. Martorell, J. Planas, OmpSs: A proposal for programming heterogeneous multi-core architectures, *Parallel Processing Letters*, 21(02):173-193, 2011. DOI: 10.1142/S0129626411000151

C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, StarPU: A unified platform for task scheduling on heterogeneous multicore architectures, *Concurrency and Computation: Practice and Experience*, Special Issue: Euro-Par 2009, 23(2):187–198, 2011, DOI: 10.1002/cpe.1631

A. YarKhan, *Dynamic Task Execution on Shared and Distributed Memory Architectures*, PhD dissertation, University of Tennessee, 2012. [http://trace.tennessee.edu/utk\\_graddiss/1575](http://trace.tennessee.edu/utk_graddiss/1575)

M. Tillenius, SuperGlue: A shared memory framework using data versioning for dependency-aware task-based parallelization, *SIAM Journal on Scientific Computing*, 37(6):C617–C642, 2015. DOI:10.1137/140989716

A. Zafari, M. Tillenius, E. Larsson, Programming models based on data versioning for dependency-aware task-based parallelisation, *Computational Science and Engineering (CSE)*, 2012 IEEE 15th International Conference on. IEEE, 2012. DOI: 10.1109/ICCSE.2012.45

# Dynamic Scheduling, OpenMP, GNU GCC



May 2008

OpenMP 3.0



GCC 4.4

`#pragma omp task`

April 2009

July 2013

OpenMP 4.0



GCC 4.9

`#pragma omp task depend`

April 2014

Nov. 2015

OpenMP 4.5

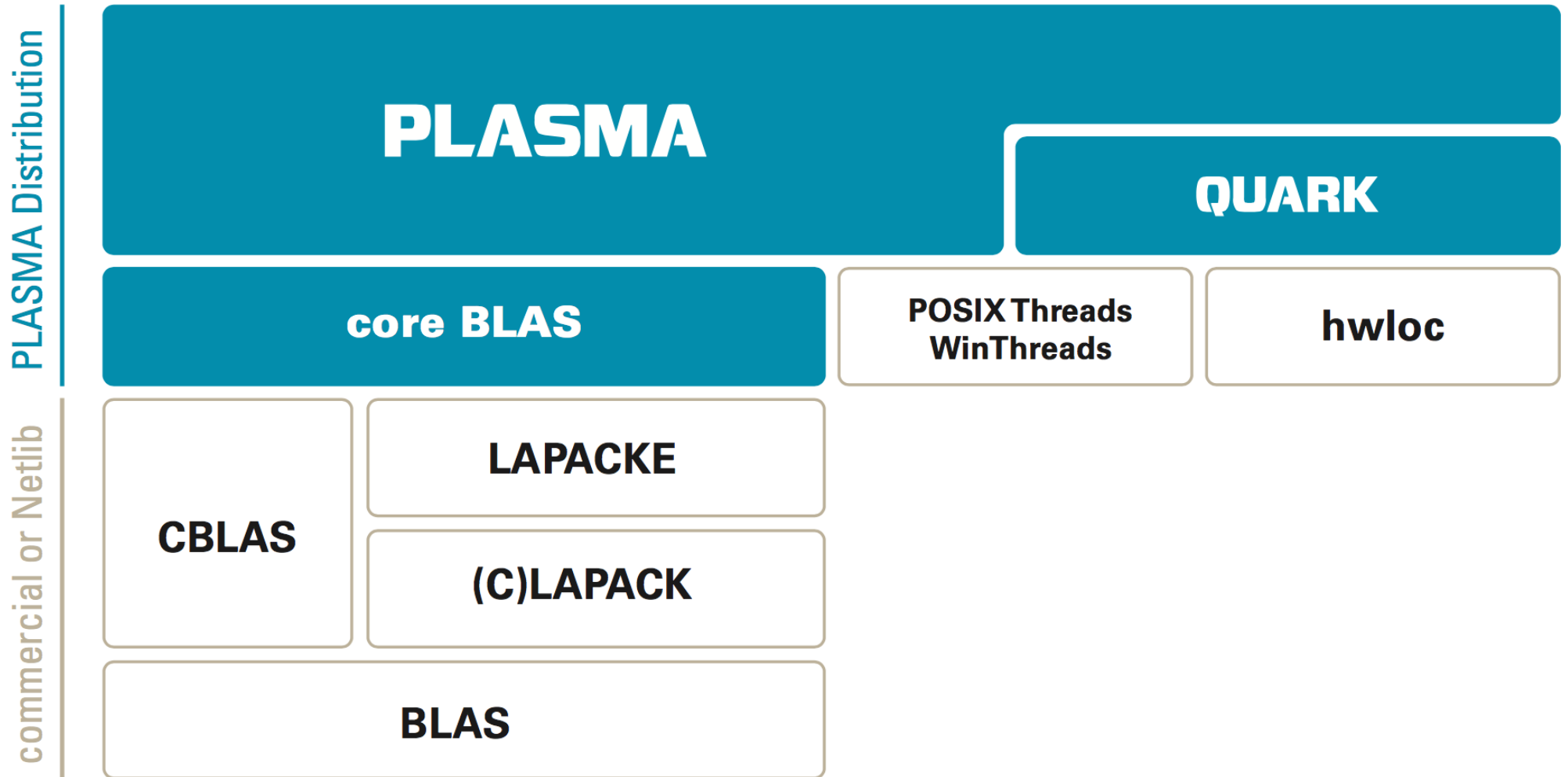


GCC 6.1

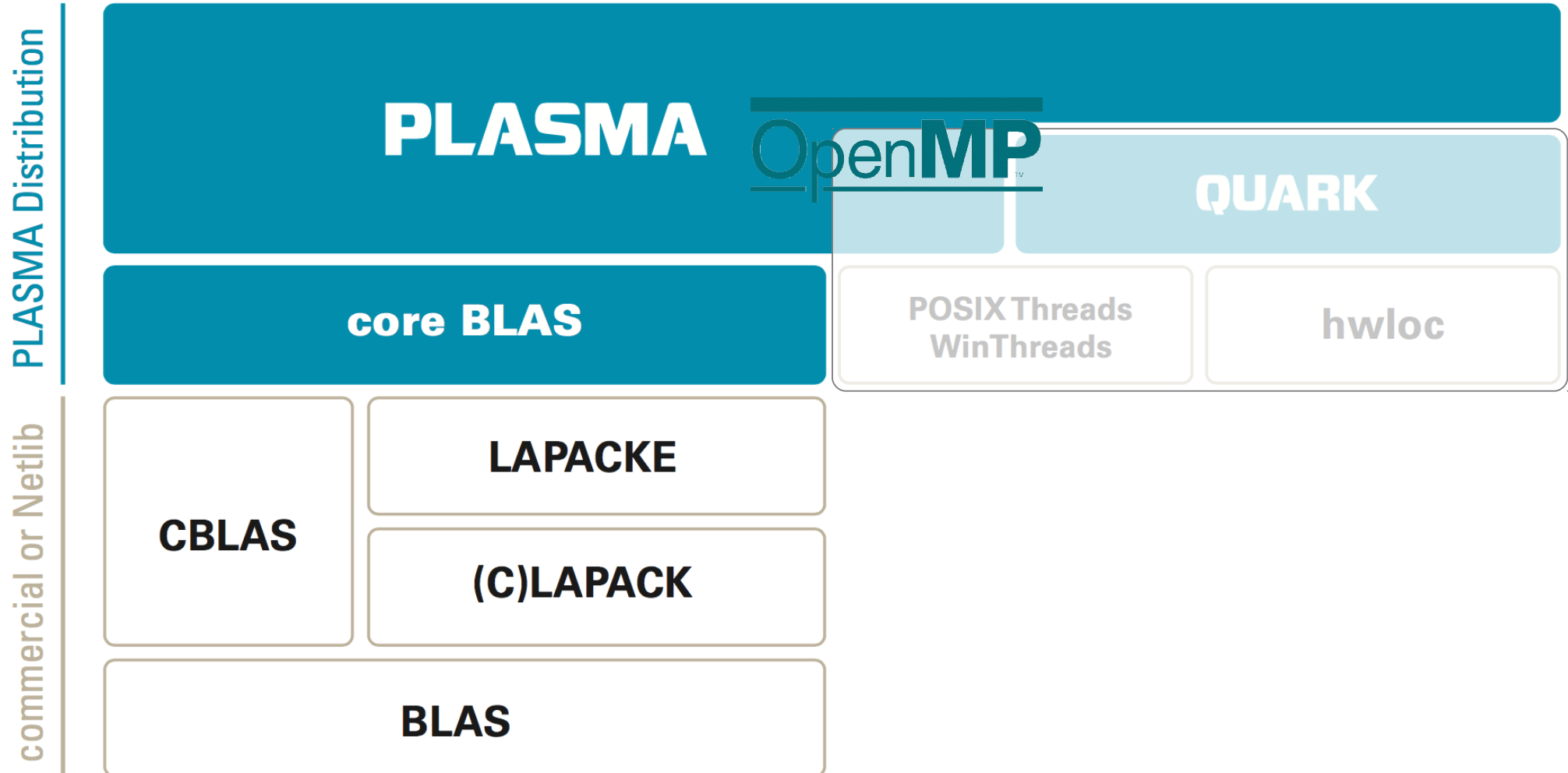
`#pragma omp task priority`

April 2016

# PLASMA Old Design



# PLASMA New Design



A. YarKhan, J. Kurzak, P. Luszczek, J. Dongarra, Porting the PLASMA Numerical Library to the OpenMP Standard, International Journal of Parallel Programming, 1-22, 2016. DOI:

[10.1007/s10766-016-0441-6](https://doi.org/10.1007/s10766-016-0441-6)

```

for (k = 0; k < nt; k++) {
    QUARK_CORE_dpotrf(
        plasma->quark, &task_flags,
        PlasmaLower,
        nb, nb,
        A(k, k), nb,
        sequence, request, nb*k);

    for (m = k+1; m < nt; m++) {
        QUARK_CORE_dtrsm(
            plasma->quark, &task_flags,
            PlasmaRight, PlasmaLower,
            PlasmaTrans, PlasmaNonUnit,
            nb, nb, nb,
            1.0, A(k, k), nb,
            A(m, k), nb);
    }
    for (m = k+1; m < nt; m++) {
        QUARK_CORE_dsyrrk(
            plasma->quark, &task_flags,
            PlasmaLower, PlasmaNoTrans,
            nb, nb, nb,
            -1.0, A(m, k), nb,
            1.0, A(m, m), nb);

        for (n = k+1; n < m; n++) {
            QUARK_CORE_dgemm(
                plasma->quark, &task_flags,
                PlasmaNoTrans, PlasmaTrans,
                nb, 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++) {
    QUARK CORE dpotrf(
        plasma->quark, &task_flags,
        PlasmaLower,
        nb, nb,
        A(k, k), nb,
        sequence, request, nb*k);
    for (m = k+1; m < nt; m++) {
        QUARK CORE dtrsm(
            plasma->quark, &task_flags,
            PlasmaRight, PlasmaLower,
            PlasmaTrans, PlasmaNonUnit,
            nb, nb, nb,
            1.0, A(k, k), nb,
            A(m, k), nb);
    }
    for (m = k+1; m < nt; m++) {
        QUARK CORE dsyrk(
            plasma->quark, &task_flags,
            PlasmaLower, PlasmaNoTrans,
            nb, nb, nb,
            -1.0, A(m, k), nb,
            1.0, A(m, m), nb);
        for (n = k+1; n < m; n++) {
            QUARK CORE dgemm(
                plasma->quark, &task_flags,
                PlasmaNoTrans, PlasmaTrans,
                nb, nb, nb, nb,
                -1.0, A(m, k), nb,
                A(n, k), nb,
                1.0, A(m, n), nb);
        }
    }
}

```

```

void QUARK CORE dpotrf(
    Quark *quark, Quark_Task_Flags *task_flags,
    PLASMA_enum uplo,
    int n, int nb,
    double *A, int lda,
    PLASMA_sequence *sequence, PLASMA_request *request,
    int iinfo)
{
    QUARK Insert Task(
        quark, CORE dpotrf_quark, task_flags,
        sizeof(PLASMA_enum), &uplo, VALUE,
        sizeof(int), &n, VALUE,
        sizeof(double)*nb*nb, A, INOUT,
        sizeof(int), &lda, VALUE,
        sizeof(PLASMA_sequence*), &sequence, VALUE,
        sizeof(PLASMA_request*), &request, VALUE,
        sizeof(int), &iinfo, VALUE,
        0);
}

```

```

void CORE_dpotrf_quark(Quark *quark)
{
    PLASMA_enum uplo;
    int n;
    double *A;
    int lda;
    PLASMA_sequence *sequence;
    PLASMA_request *request;
    int iinfo;

    int info;
    quark_unpack_args_7(quark, uplo, n, A, lda, sequence, request, iinfo);
    info = LAPACKE_dpotrf_work(
        LAPACK_COL_MAJOR,
        lapack_const(uplo),
        n, A, lda);
    if (sequence->status == PLASMA_SUCCESS && info != 0)
        plasma_sequence_flush(quark, sequence, request, iinfo+info);
}

```

```

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

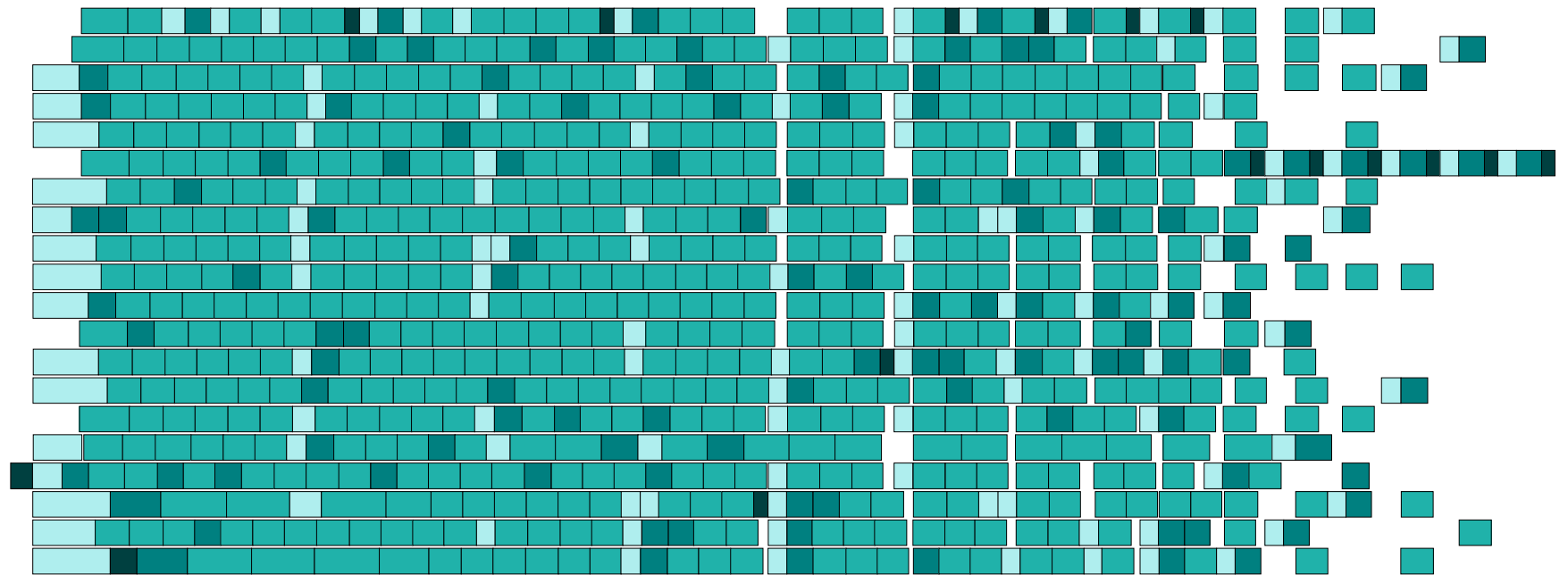
cblas_dgemm(
    CblasColMajor,
    CblasNoTrans, CblasTrans,
    nb, nb, nb,
    -1.0, A(m,k), nb,
        A(n,k), nb,
    1.0, A(m,n), nb);

```

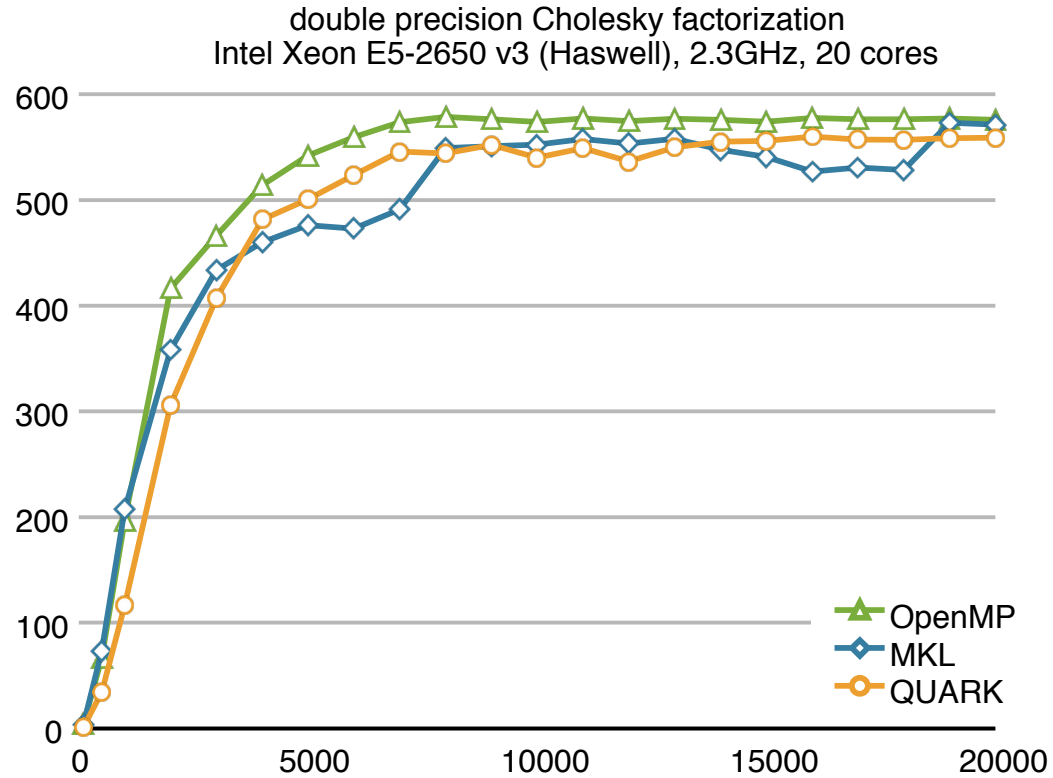


# PLASMA OpenMP Trace

PLASMA Cholesky factorization using OpenMP  
Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores  
tiles of size 224 x 224, matrix of size 20 x 20 tiles (4480 x 4480)

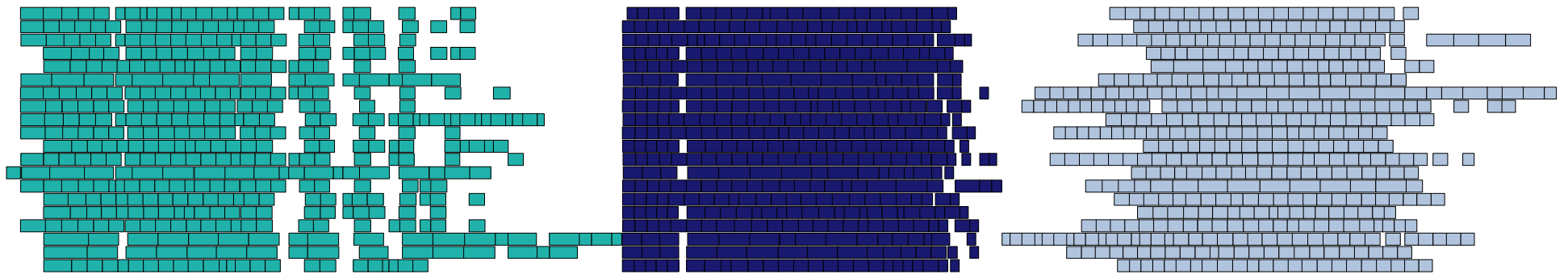


# PLASMA OpenMP Cholesky Performance



# PLASMA OpenMP Cholesky Inversion Trace

PLASMA Cholesky inversion using OpenMP  
Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores  
tiles of size 224 x 224, matrix of size 13 x 13 tiles (2912 x 2912)



**plasma\_dpotrf**(uplo, n, pA, lda);

**plasma\_dlauum**(uplo, n, pA, lda);

**plasma\_dtrtri**(uplo, diag, n, pA, lda);

# PLASMA OpenMP Cholesky Inversion Code

```
#pragma omp parallel
#pragma omp master
{
    plasma_omp_zge2desc(pA, lda, A, sequence, &request);

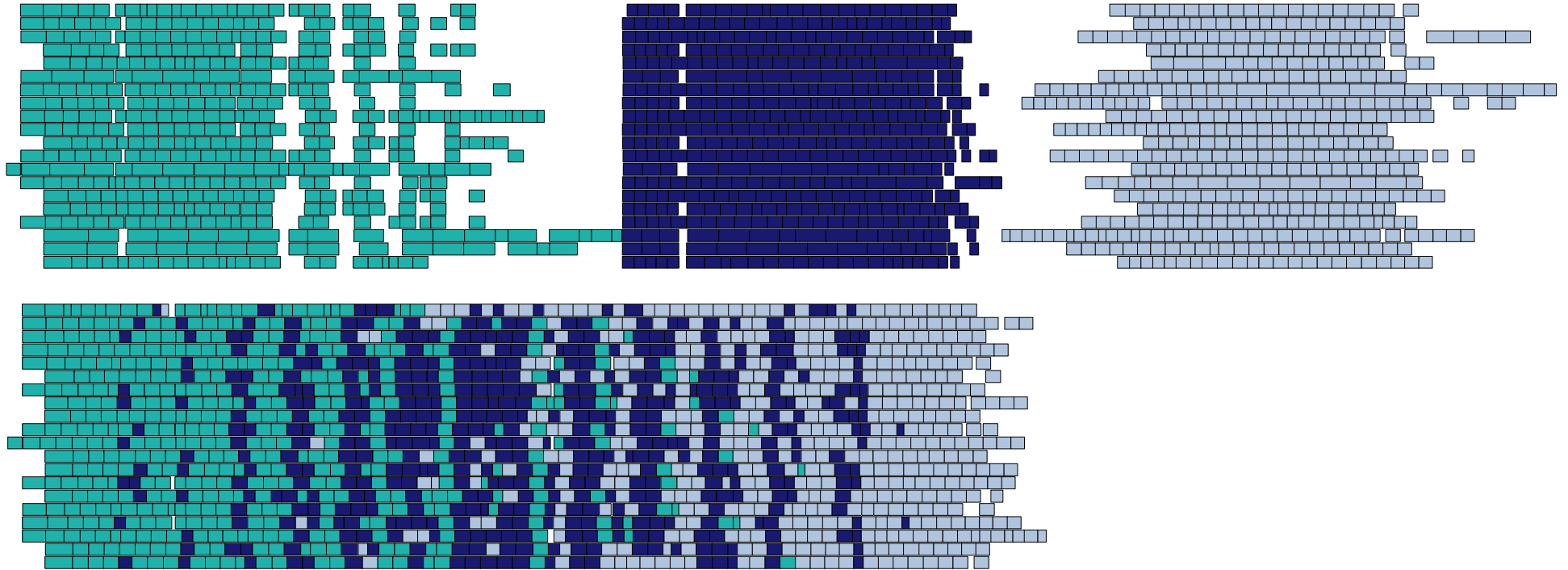
    plasma_omp_dpotrf(uplo, A, sequence, &request);
    plasma_omp_zlauum(uplo, A, sequence, &request);
    plasma_omp_ztrtri(uplo, diag, A, sequence, &request);

    plasma_omp_zdesc2ge(A, pA, lda, sequence, &request);
}
```

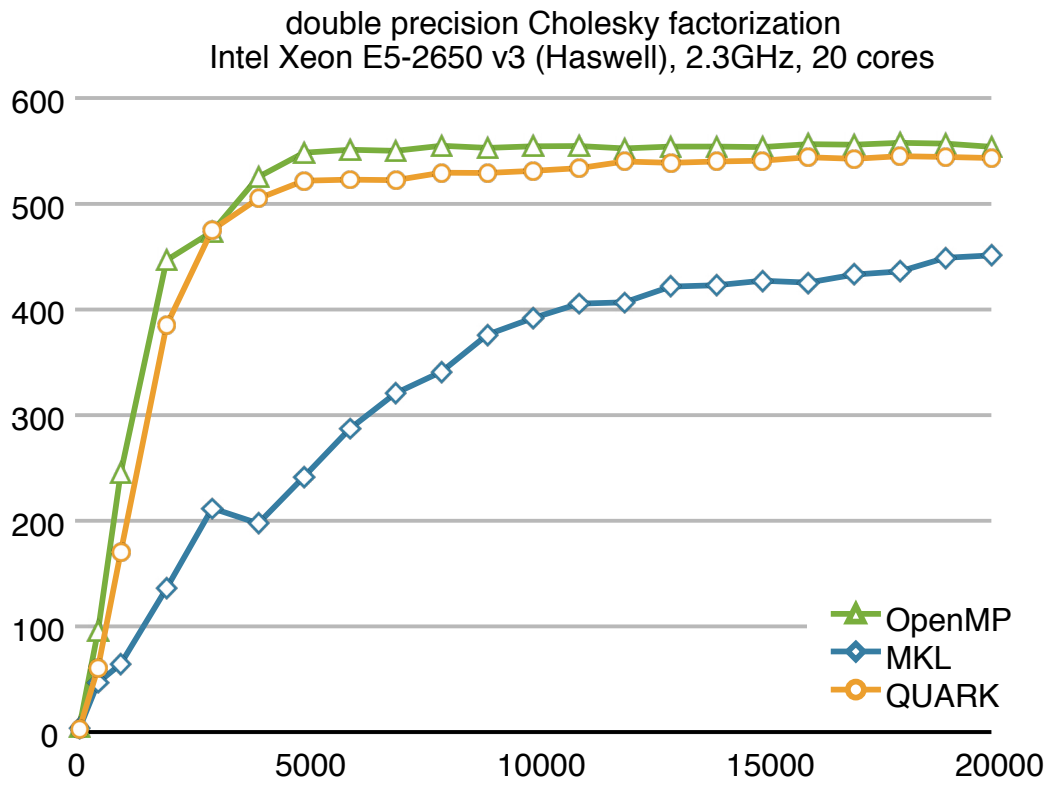


# PLASMA OpenMP Cholesky Inversion Traces

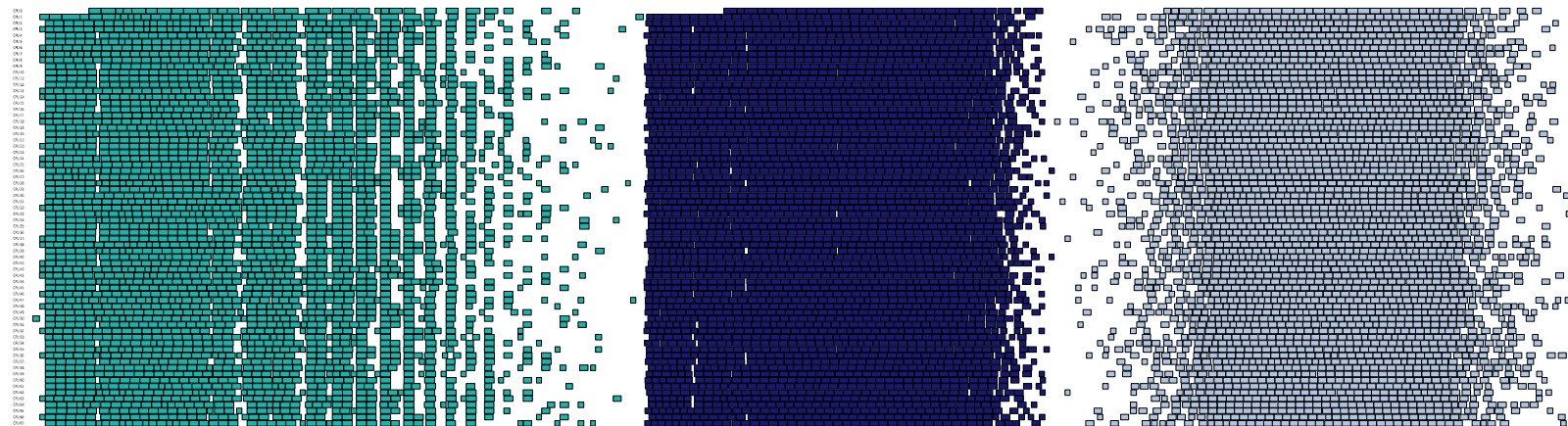
PLASMA Cholesky inversion using OpenMP  
Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores  
tiles of size 224 x 224, matrix of size 13 x 13 tiles (2912 x 2912)



# PLASMA OpenMP Cholesky Factorization Perf.



# Traces of Merging Routines on KNL



Intel Xeon Phi, Knights Landing, 68 cores, 1.4 GHz



N = 7200  
NB = 288  
(25x25 tiles)

sync:  
0.485 sec  
770 Gflop/s

async:  
0.373 sec  
1001 Gflop/s