

# PLASMA

## Parallel Linear Algebra Software for Multicore Architectures

Innovative Computing Laboratory  
Electrical Engineering and Computer Science  
University of Tennessee  
Piotr Luszczek (presenter)



# TOC: Table of Contents

- **Functionality**
- **Performance**
- **Software Stack**
- **Documentation**
- **Installation**
- **Matrix Layout**
- **Algorithms**
- **Classes of routines and interfaces**
- **Settings**

# PLASMA Institutions



Knoxville Tennessee



Berkeley California



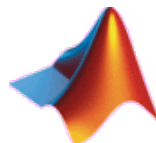
Denver Colorado



**Microsoft**



U.S. DEPARTMENT OF  
**ENERGY**



The MathWorks™

**FUJITSU**

shaping tomorrow with you

# PLASMA People

- Current Team

- Dulceneia Becker
- Henricus Bouwmeester
- Jack Dongarra
- Mathieu Faverge
- Mark Gates
- Azzam Haidar
- Blake Haugen
- Jakub Kurzak
- Julien Langou
- Hatem Ltaief
- Piotr Łuszczek
- Ichitaro Yamazaki
- Asim YarKhan
- Vijay Joshi

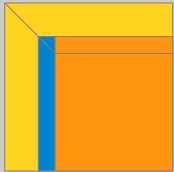
- Past Members

- Emmanuel Agullo
- Wesley Alvaro
- Alfredo Buttari
- Bilel Hadri

- Outside Contributors

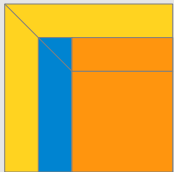
- Fred Gustavson
- Lars Karlsson
- Bo Kågström

# Dense Linear Algebra: Software Evolution



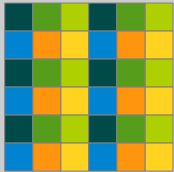
**LINPACK** (70's)  
vector operations

- Level 1 BLAS



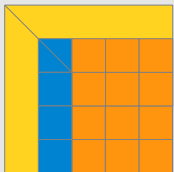
**LAPACK** (80's)  
block operations

- Level 3 BLAS



**ScaLAPACK** (90's)  
block cyclic  
data distribution

- PBLAS
- BLACS  
(message passing)

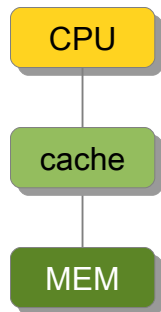


**PLASMA** (00's)  
tile operations

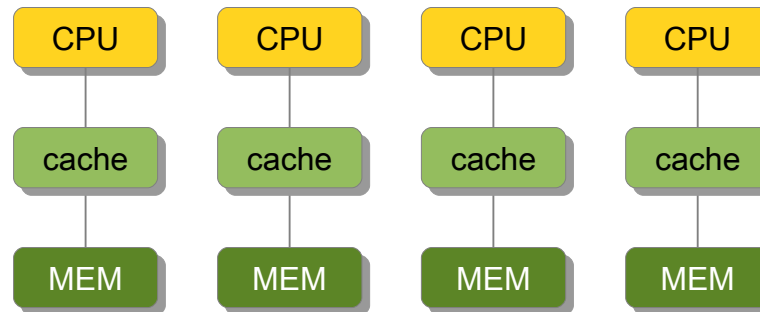
- tile layout
- dataflow scheduling

# Dense Linear Algebra: Architectural Models

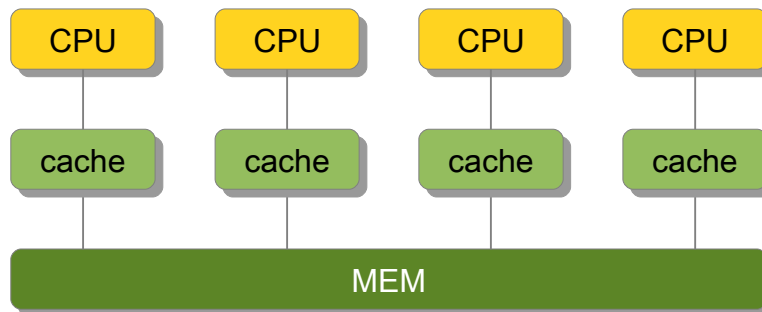
LAPACK



ScaLAPACK



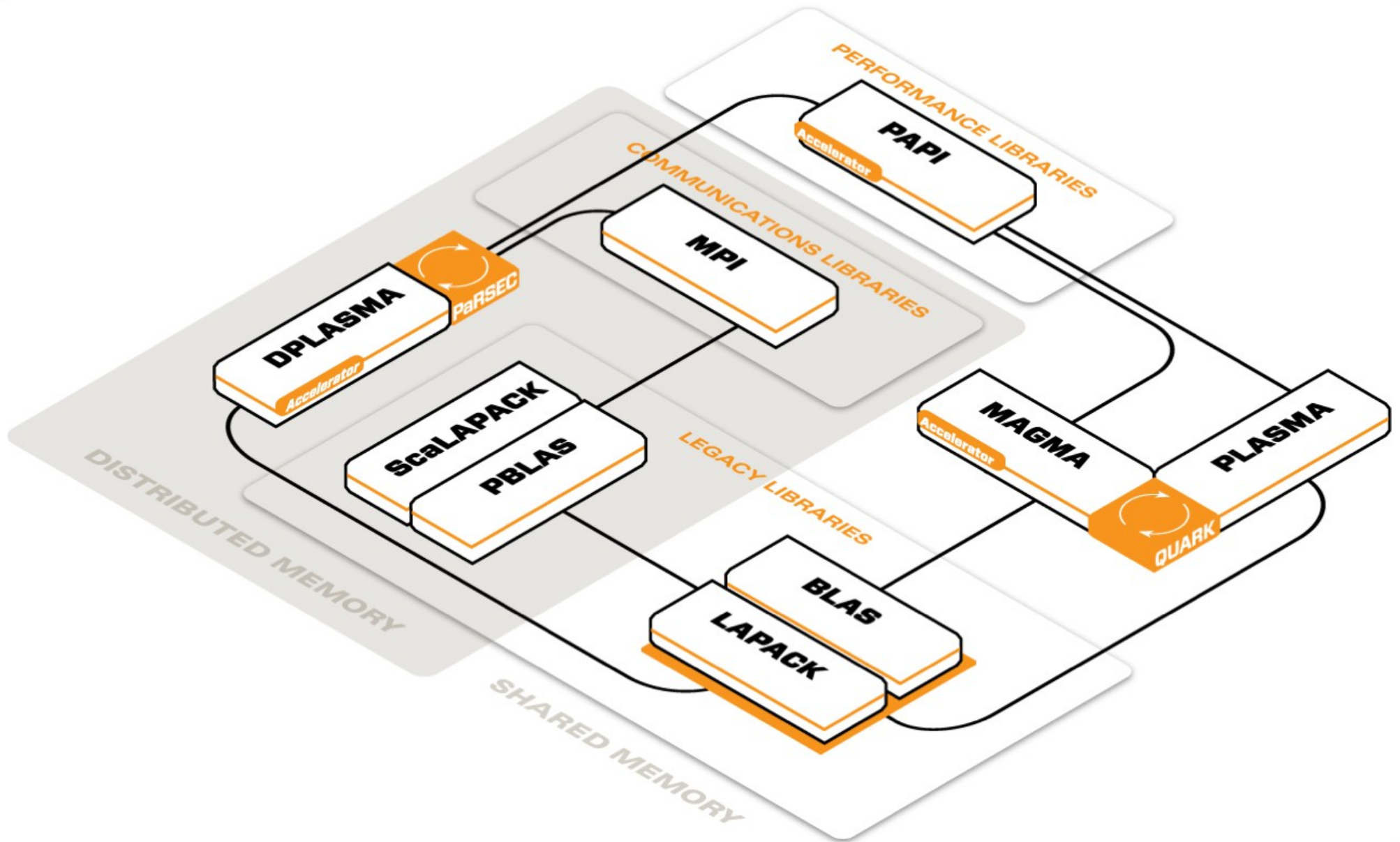
PLASMA



# Dense Linear Algebra: Related Developments

- Matrix Algebra for GPUs and Multicore Architectures - MAGMA
  - shared memory
  - GPU acceleration
- Distributed Memory PLASMA – DPLASMA
- Parallel Runtime Scheduler & Execution Control - ParSEC
  - distributed memory
  - compact DAG representation
  - dynamic DAG scheduling
- Parallel Unified Linear Algebra with Systolic Arrays – PULSAR
  - distributed memory
  - Virtual Systolic Array architecture

# ICL Project Map





# PLASMA In a Nutshell

- Software Library
- Dense Linear Algebra
  - linear systems of equations
  - least square problems
  - singular value problems
  - eigenvalue problems
- Multicore Systems
  - multicore
  - multi-socket
  - shared memory
  - possibly NUMA

- all precisions (S, D, C, Z)
- Linux, Windows, Mac OS, AIX

(e.g. 12-core AMD Magny-Cours)

(e.g. quad-socket Dell PowerEdge)

hardware:  
today – tens of cores  
tomorrow – hundreds of cores

# Linear Systems: $AX = B$

- Fully Determined  $m = n$ 
  - $PA = LU$  Gaussian elimination non-symmetric
  - $A = LL^T$  Cholesky factorization symmetric positive definite
  - $A = LDL^T$   $LDL^T$  decomposition non-symmetric positive definite
- Overdetermined  $m > n$  (least squares) prototypes
  - $A = QR$
- Underdetermined  $m < n$  (minimum norm)
  - $A = LQ$

# Linear Systems: $AX = B$

- **Explicit Matrix Inversion**
  - non-symmetric
  - symmetric positive definite
  - explicitly calculating the inverse of a matrix
  - e.g., calculating the variance-covariance matrix in statistics
- **Tall and Skinny Factorizations**       $m \ll n$        $m \gg n$ 
  - fast (highly parallel) algorithms
  - based on tree reductions
- **Mixed Precision Iterative Refinement**
  - factorization in single precision
  - iterative refinement in double precision

# Eigenvalues, SVD: $A=XX^T$ , $A = U\Sigma V^T$ , $AX = \Lambda BX$

- Symmetric Eigenvalue Problem

- reduction to “block tridiagonal” (band)
- band reduction to tridiagonal (bulge chasing)
- LAPACK eigensolver (QR/D&C/MRRR)

non-symmetric eigenvalue problem  
work in progress

- Singular Value Problem

- reduction to “block bidiagonal” (band)
- band reduction to bidiagonal (bulge chasing)
- QR algorithm (LAPACK)

extremely fast eigenvalues / singular values  
fast eigenvectors / singular vectors  
possibility of calculating a subset

- Generalized Eigenvalue Problem

- Cholesky factorization of B and application to A
- reduction to band & band reduction; LAPACK eigensolver (QR/D&C/MRRR)

symmetric positive definite matrices only

# Auxiliary Functionality: Layouts & Tile BLAS 3

- In-Place Matrix Layout Translation

- CM, RM (column-major / row-major)
- CCRB, CRRB, RCRB, RRRB (column / row – rectangular block)
- LAPACK ↔ PLASMA (CM ↔ CCRB)
- any other combination
- (fast transposition)
- cache-efficient
- parallel

Thank you Fred, Lars, Bo!

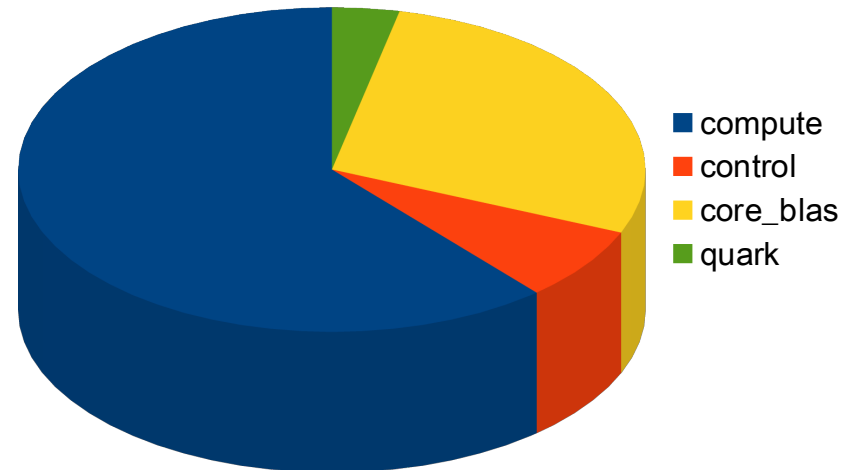
There also is a fast out-of-place  
PLASMA ↔ LAPACK translation

- Tile BLAS 3

- GEMM, HEMM, HER2K, HERK, SYMM, SYR2K, SYRK, TRMM, TRSM
  - Complete set

# PLASMA Size In Numbers

Size of PLASMA Main Components

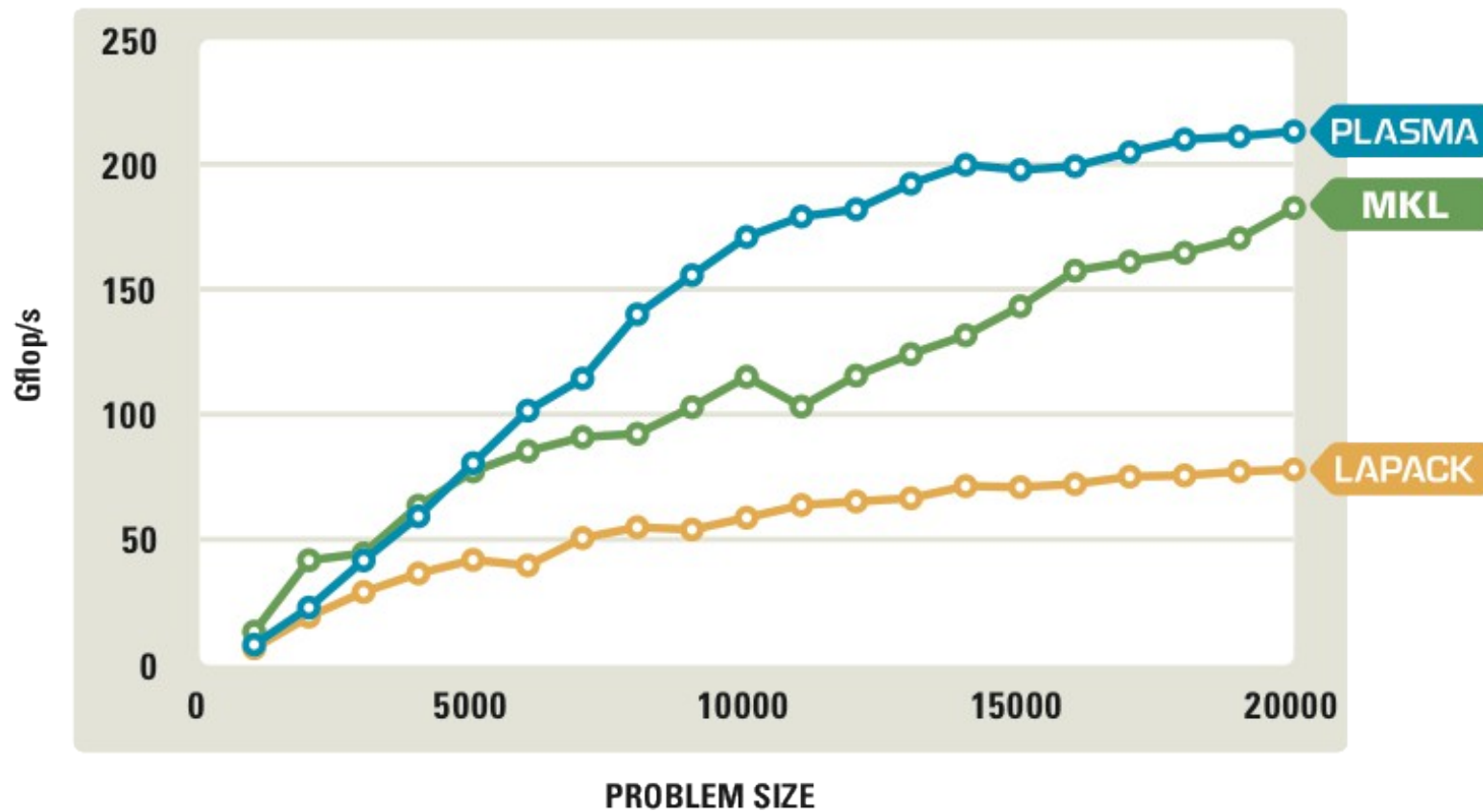


- ~200,000 lines of code
- ~1,000 source files
- ~600 API functions

# Motivation: Performance for LU Solve

partial pivoting

**Solving Linear System (DGESV)**  
48-core, 2.1 GHz AMD Magny-Cours System

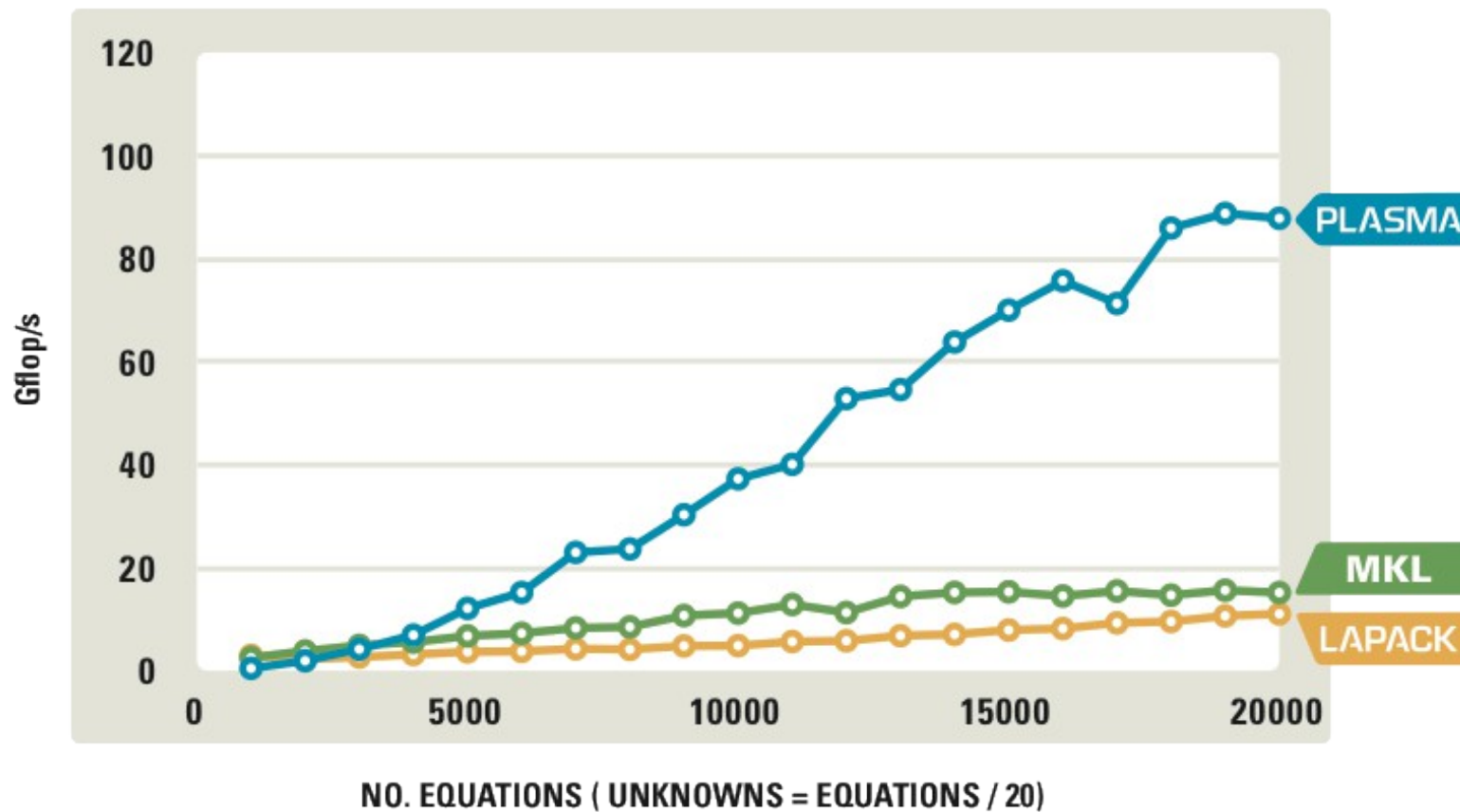


# Motivation: Performance for QR Factorization

## Solving Least Squares Problem (DGELS)

48-core, 2.1 GHz AMD Magny-Cours System

tree-based

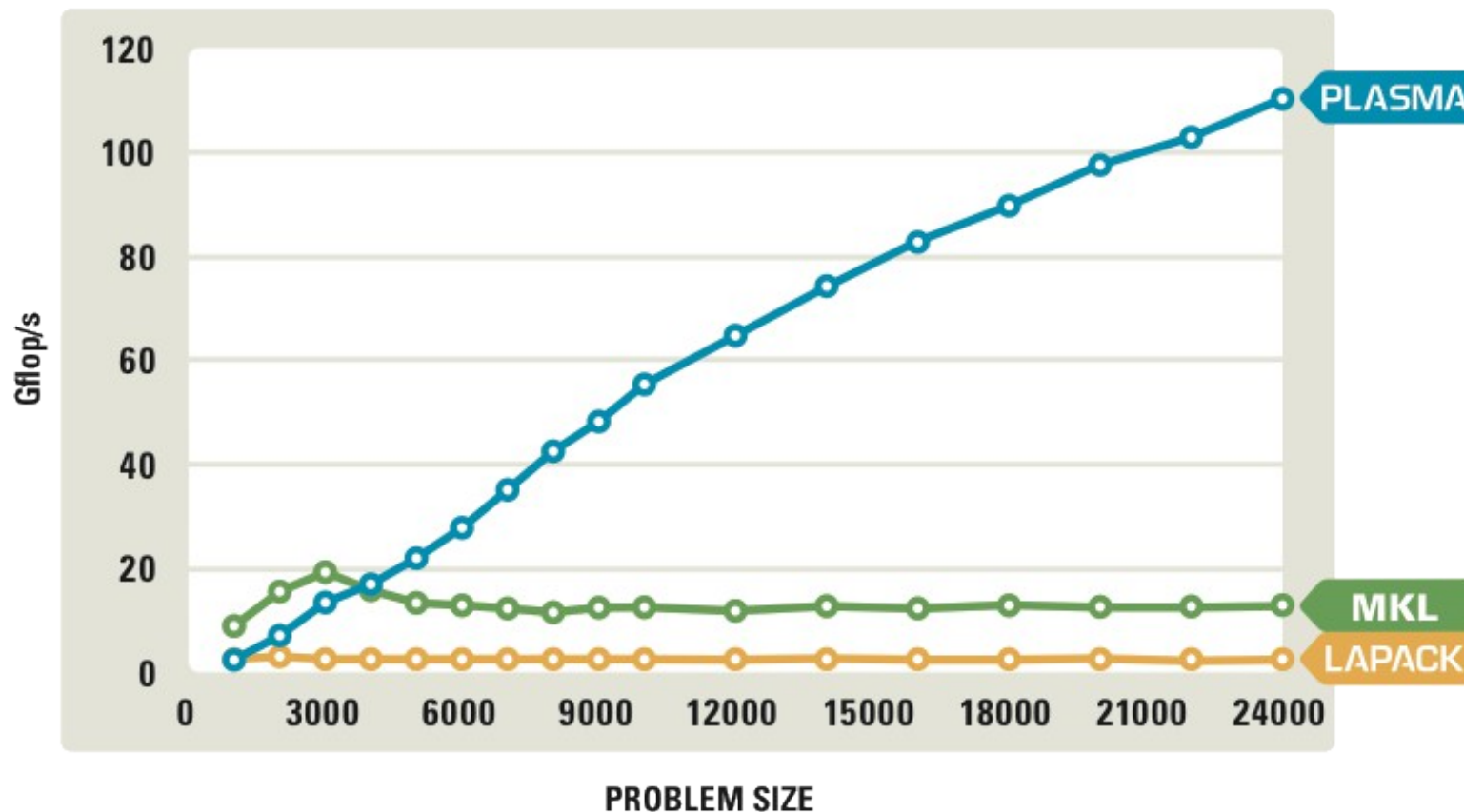




# Motivation: Performance for Eigenvalues

Solving Symmetric EVP (DSYEV)  
48-core, 2.1 GHz AMD Magny-Cours System

no vectors requested

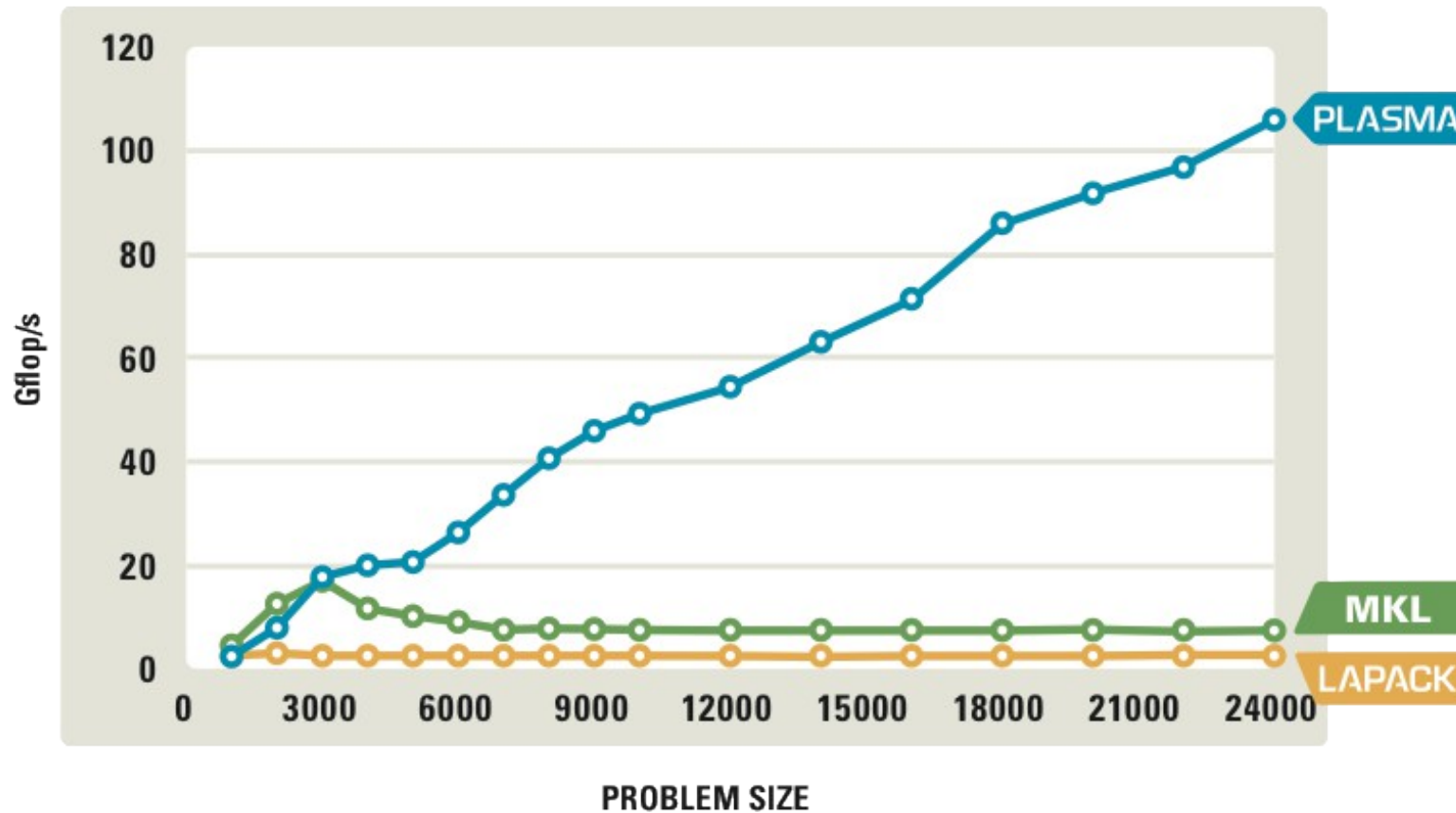


# Motivation: Performance for Singular Values

## Solving Singular Value Problem (DGESVD)

48-core, 2.1 GHz AMD Magny-Cours System

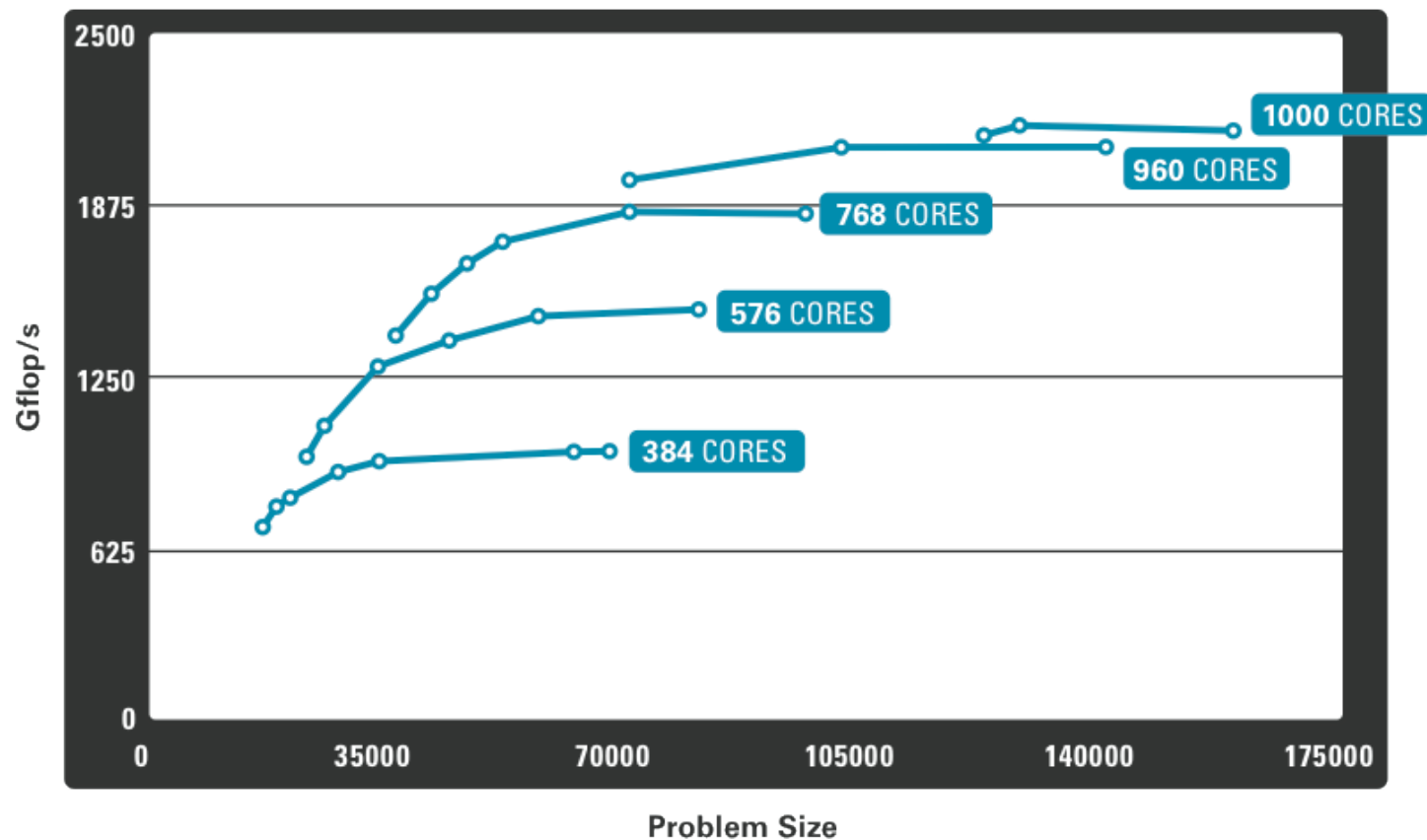
no vectors requested



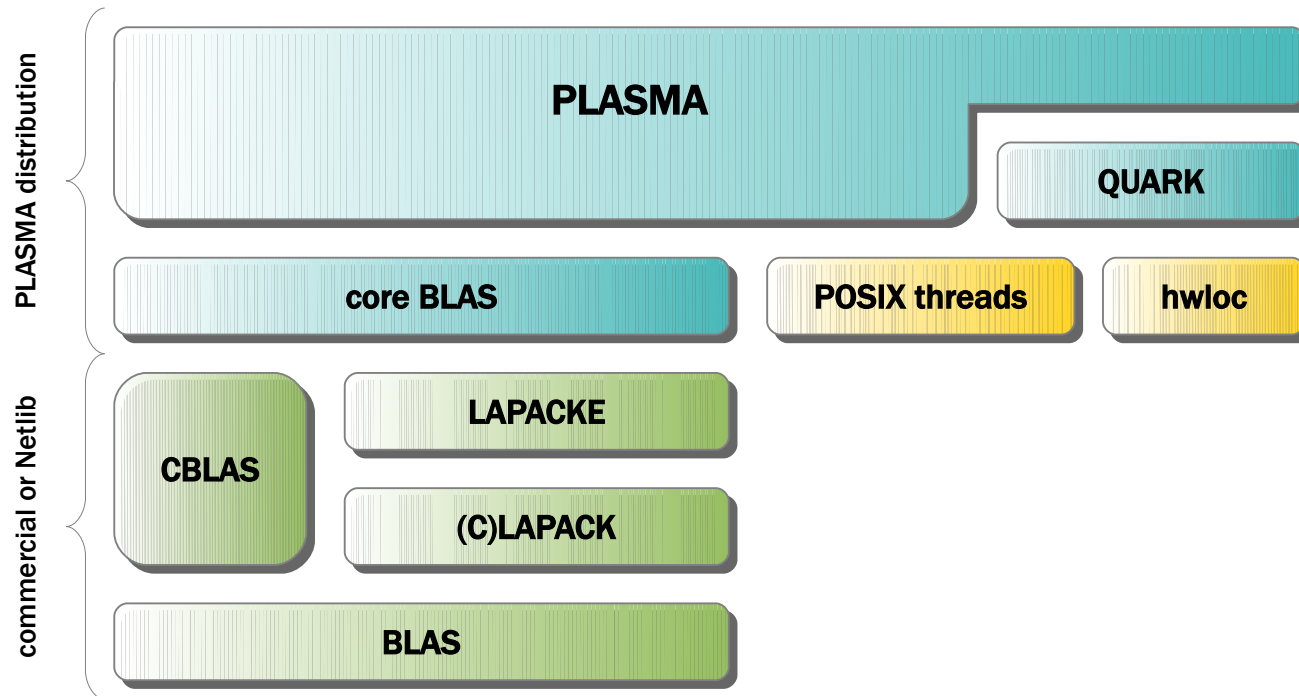
# Motivation: Performance on 1000 Cores

## Solving Symmetric Positive Definite System (DPOSV)

SGI Altix UV, 2.0 GHz Intel Nehalem EX System



# Software Stack of PLASMA



**QUARK** - **Q**Ueuing **A**nd **R**untime for **K**ernels

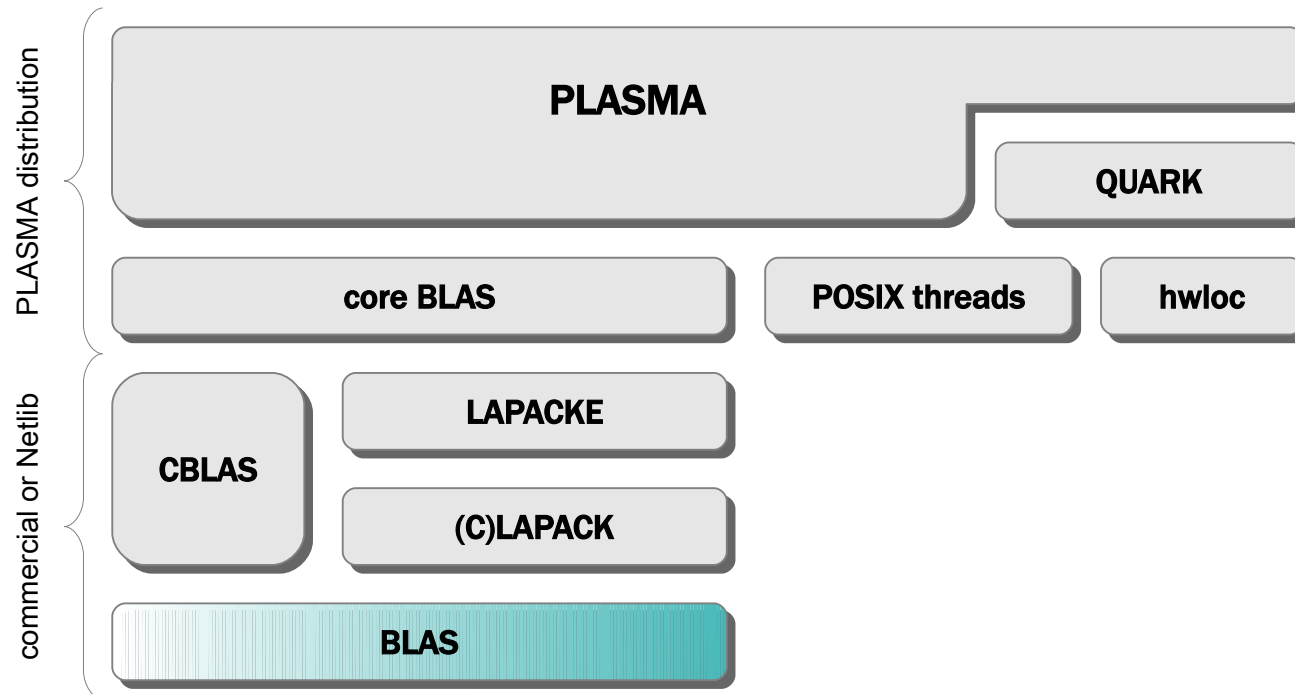
**LAPACK** - **L**inear **A**lgebra **P**ACKage

**BLAS** - **B**asic **L**inear **A**lgebra **S**ubroutines

**hwloc** - **h**ardware **l**ocality

LAPACKE is now included  
with the Netlib LAPACK

# Software Stack: BLAS

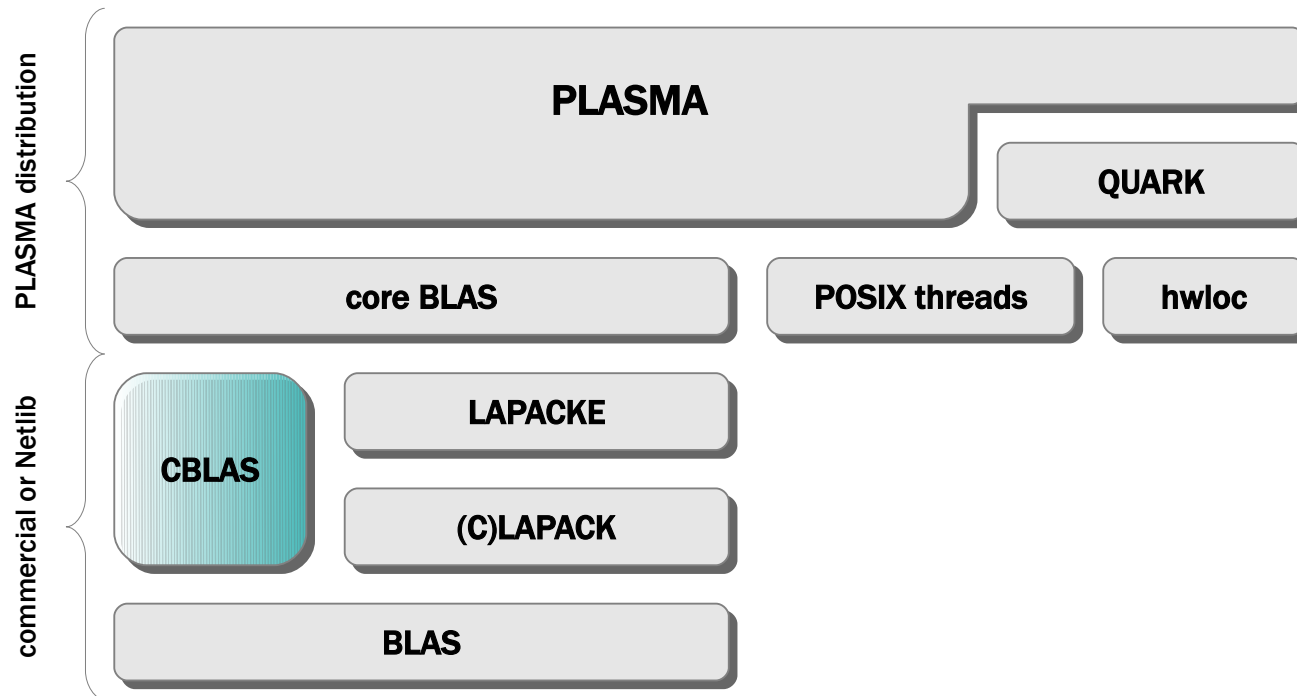


- **Basic Linea Algebra Subroutines**

- critical for performance
- vendor: Intel MKL, AMD ACML, IBM ESSL, Apple VecLib, Cray LibSci
- academic: ATLAS, Goto BLAS
- FORTRAN *definition* available from Netlib

[www.netlib.org/blas/](http://www.netlib.org/blas/)

# Software Stack: CBLAS

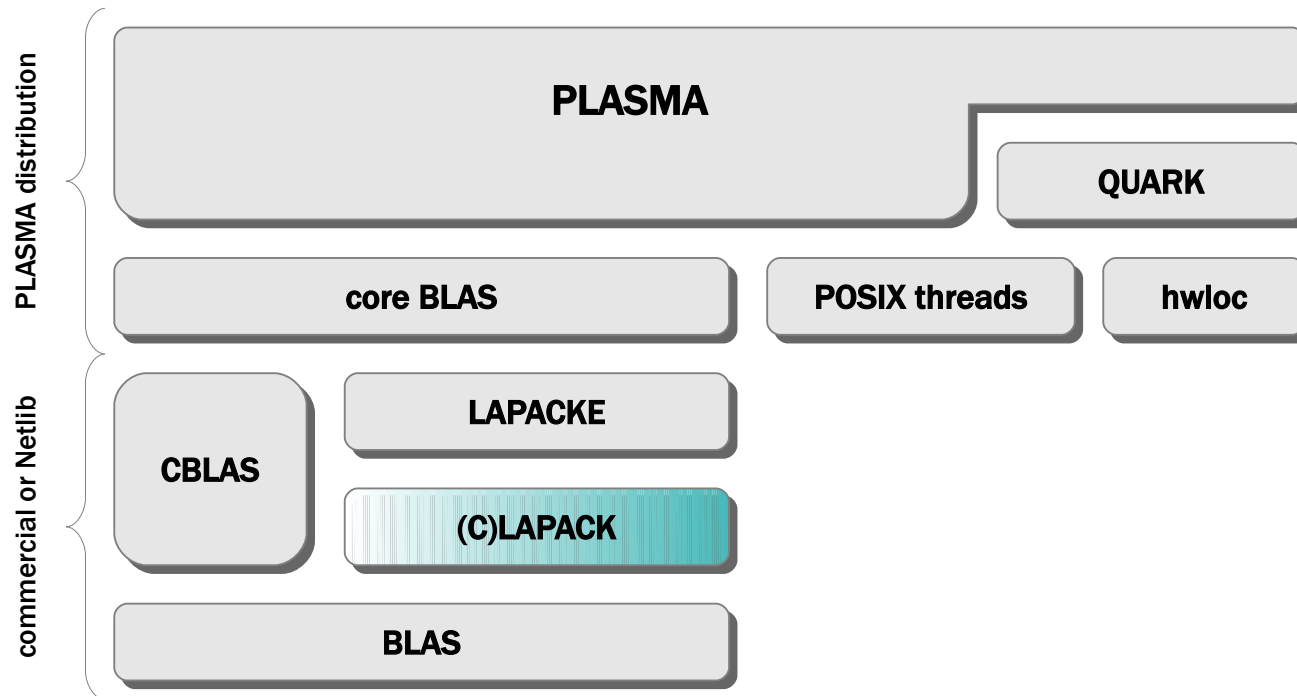


- **C API for BLAS**

- a set of wrappers
- not performance critical
- commonly included in vendor and academic packages
- available from Netlib

<http://www.netlib.org/blas/blast-forum/cblas.tgz>

# Software Stack: LAPACK

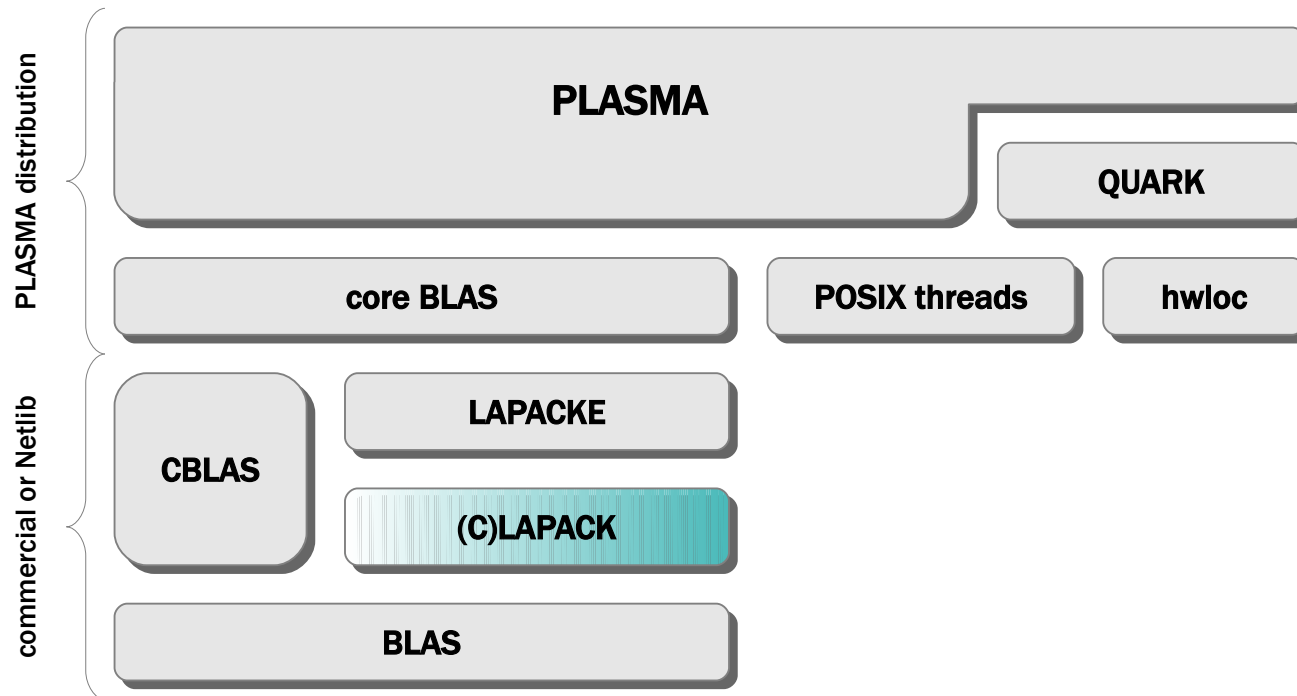


- **Linear Algebra PACKage**

- dense linear algebra software library
- FORTRAN implementation
- no parallel constructs
- available from Netlib

[www.netlib.org/lapack/](http://www.netlib.org/lapack/)

# Software Stack: CLAPACK



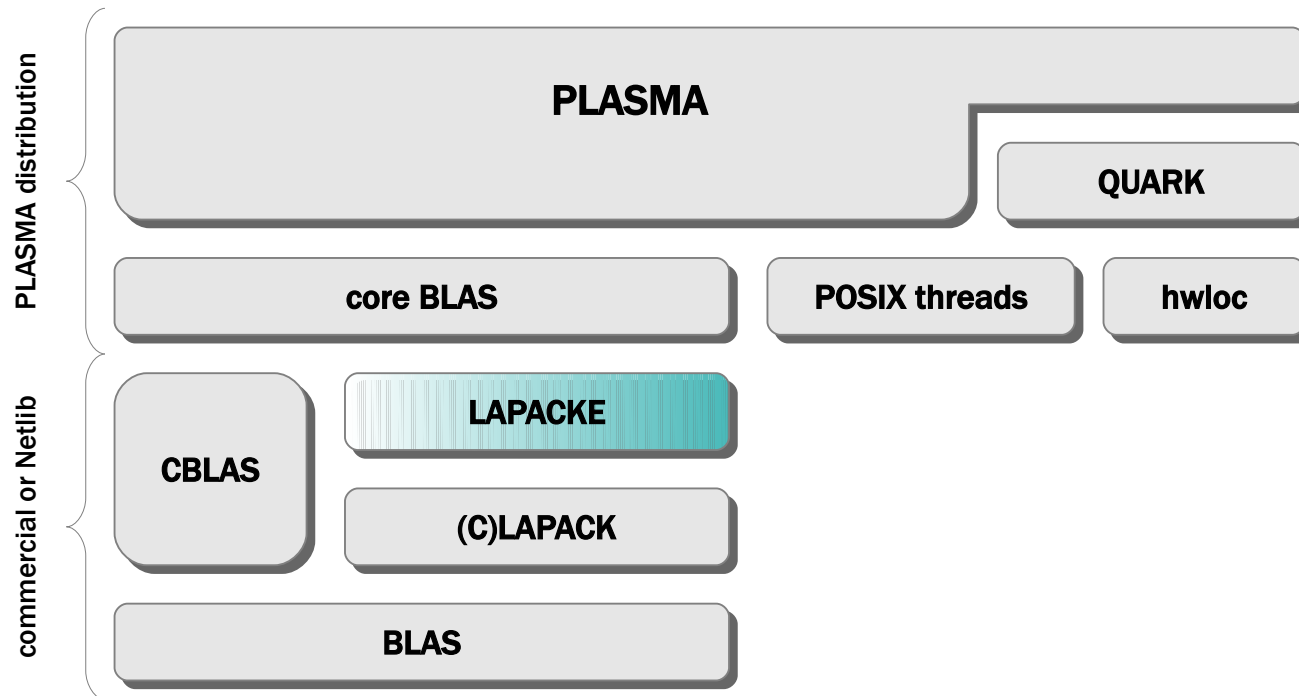
- **C Implementation of LAPACK**

- automatically generated
- C implementation
- FORTRAN API
- available from Netlib

[www.netlib.org/clapack/](http://www.netlib.org/clapack/)



# Software Stack: LAPACKE



- **C API for LAPACK & CLAPACK**

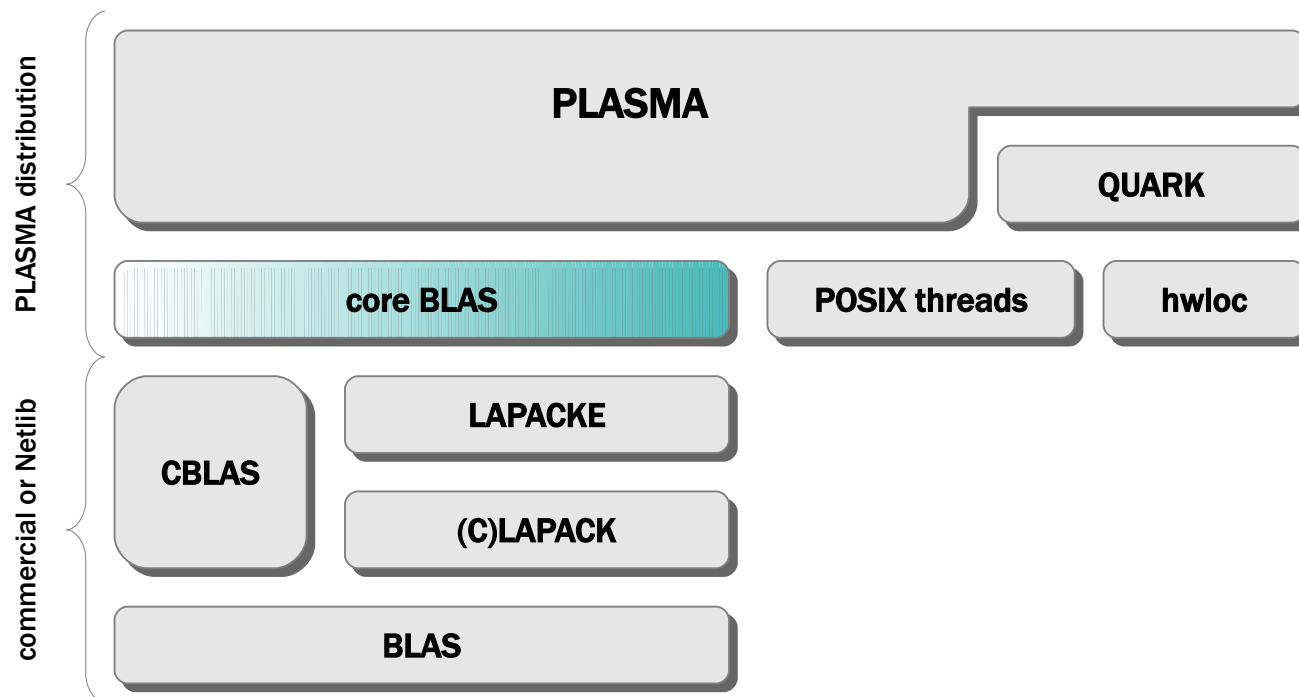
- a set of wrappers
- developed by: LAPACK team & Intel
- provided by: Netlib & MKL
- now included in Netlib LAPACK

U Tennessee, UC Berkeley, UC Denver

[www.netlib.org/lapack/lapacke.tgz](http://www.netlib.org/lapack/lapacke.tgz)

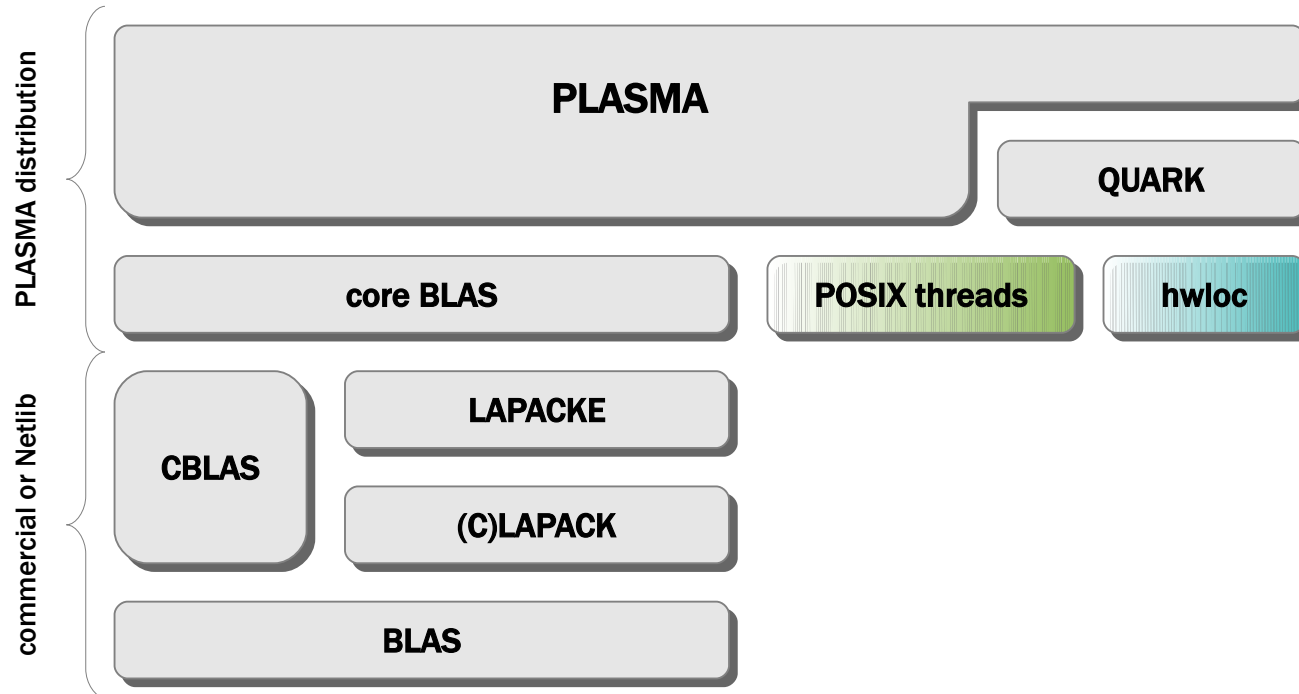
[icl.cs.utk.edu/projectsfiles/plasma/pubs/lapacke-3.3.0.tgz](http://icl.cs.utk.edu/projectsfiles/plasma/pubs/lapacke-3.3.0.tgz)

# Software Stack: core BLAS



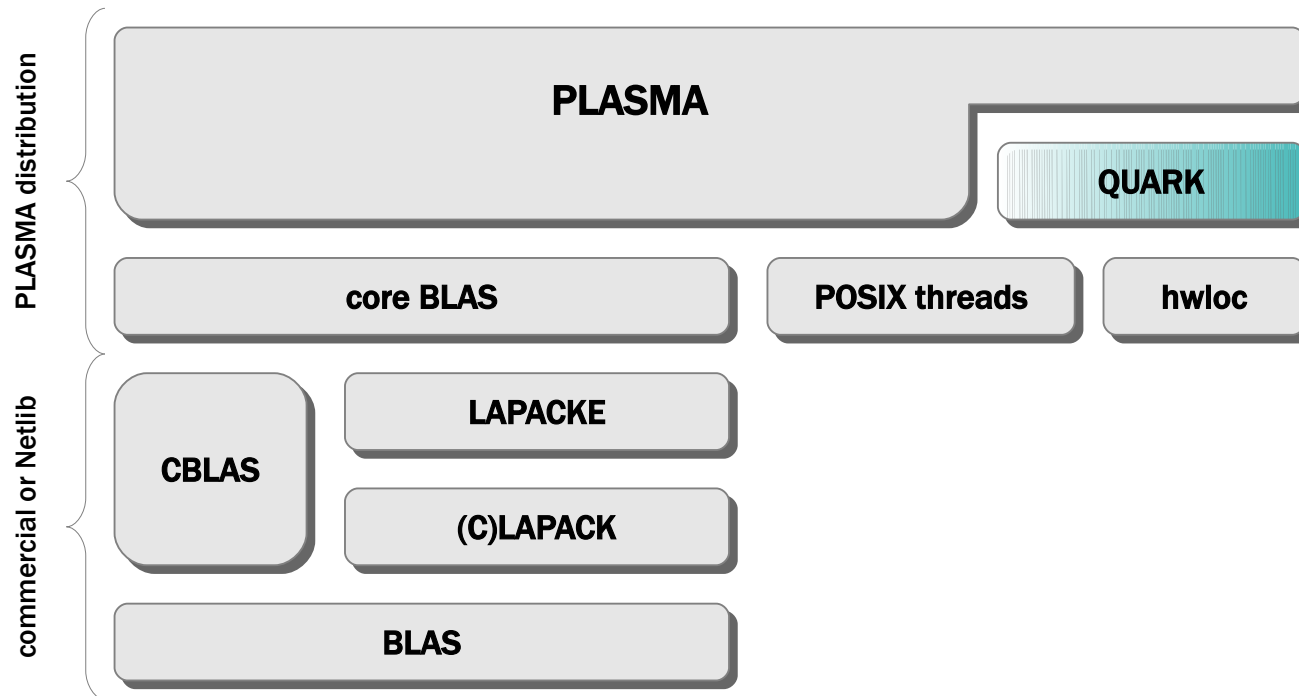
- **core BLAS**
  - serial kernels
  - PLASMA's building blocks
  - ideally monolithic & highly tuned
  - currently implemented with multiple calls to BLAS

# Software Stack: pthreads & hwloc



- **hwloc – Portable Hardware Locality**
  - topology abstraction
  - caches / NUMA nodes / sockets / cores / SMT
  - in PLASMA thread affinity control (thread placement)

# Software Stack: QUARK



- Queuing and Runtime for Kernels

- API-based
- superscalar scheduler
- similar to StarPU and SMPs
- suitable for implementing a numerical library

# PLASMA Software Engineering: Precision Generation

- Python scripts
- All precisions automatically generated
  - $Z \rightarrow D, C, S$
  - $ZC \rightarrow DS$  (mixed precision)
  - one mechanism covers all scenarios
- Easily extendable
- Intended to also cover testing, timing, autotuning

# PLASMA Software Engineering: Documentation Generation

- Doxygen comments used for all source code
- LAPACK-like Doxygen comments used for API routines
  - general description
  - parameters
  - return values
  - “see also” section
- Automatic generation of online documentation
  - grouping of routines by precision (Z, C, D, S, ZC, DS)
  - grouping of routines by interface (standard, tile, tile async)

# PLASMA Documentation Website

- Home
- Overview
- News
- Software
- Publications
- Links
- People
- Documentation
- User Forum

The image shows two overlapping browser windows. The top window displays the PLASMA homepage with a large blue header and navigation links. The bottom window shows the PLASMA User Forum, featuring a search bar, a 'Board index' link, and two tables of forum content.

**PLASMA User Forum - View forum - User discussion**

ANNOUNCEMENTS

|  | REPLIES | VIEWS | LAST POST                                |
|--|---------|-------|--|
| <b>New PLASMA v2.3.0 and the PLASMA Installer v2.3.0 released!</b><br>by fike-admin - Thu Nov 18, 2010 6:05 pm | 2       | 394   | by mateo70 - Tue Jan 25, 2011 5:47 pm    |
| <b>New PLASMA v2.2.0 and the PLASMA Installer v1.2.0 released!</b><br>by fike-admin - Sat Jul 10, 2010 4:47 pm | 0       | 5666  | by fike-admin - Sat Jul 10, 2010 4:47 pm |
| <b>PLASMA 2.0 released</b><br>by admin - Wed Jul 15, 2009 10:58 am   | 0       | 7465  | by admin - Wed Jul 15, 2009 10:58 am     |

TOPICS

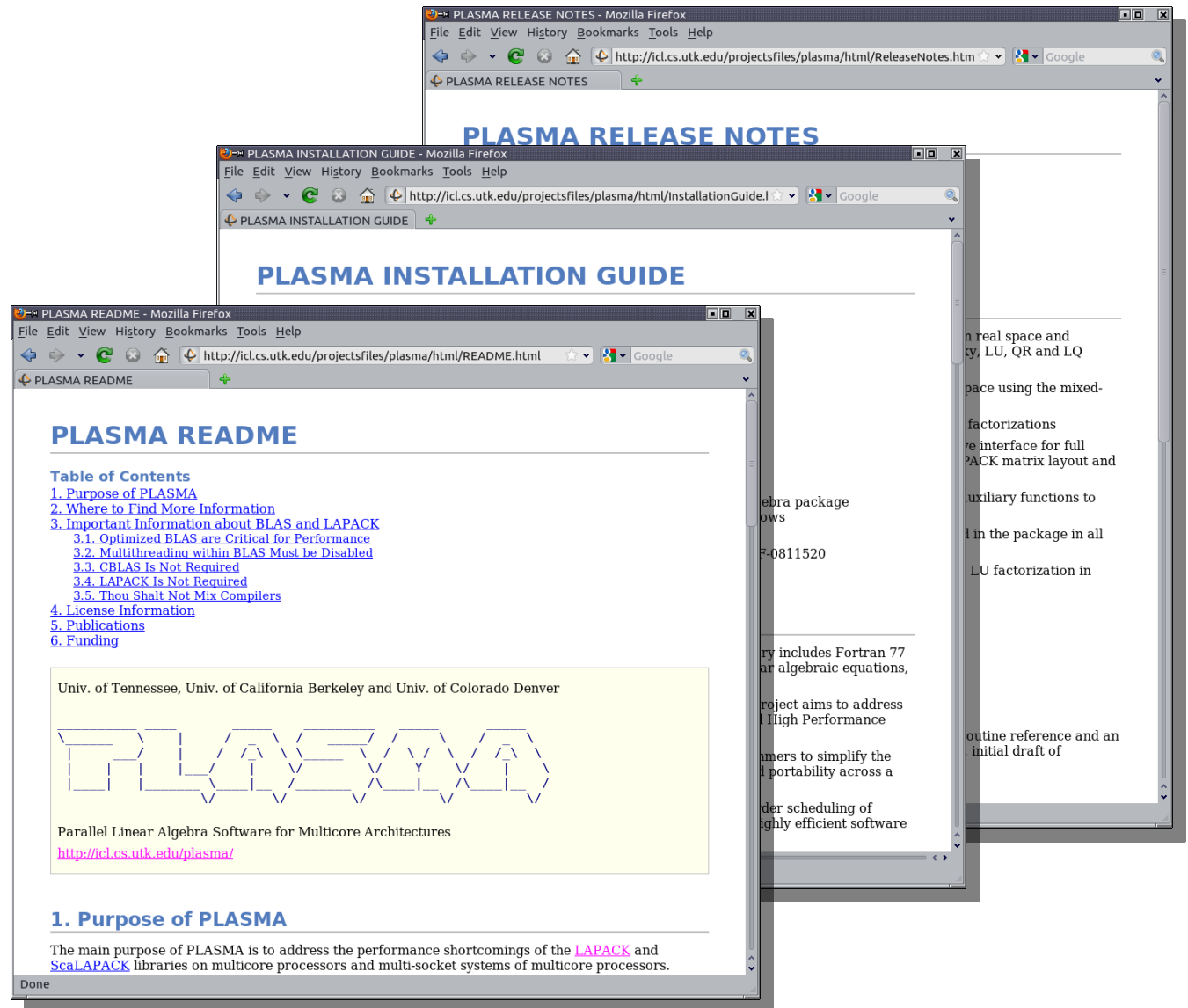
|   | REPLIES | VIEWS | LAST POST                              |
|---|---------|-------|--|
| <b>Fortran interfaces</b><br>by chrisa - Wed Apr 06, 2011 6:13 am                             | 10      | 201   | by admin - Fri May 27, 2011 1:39 pm    |
| <b>thread control</b><br>by katayama - Mon Nov 22, 2010 7:58 pm                               | 11      | 650   | by yarkhan - Thu May 12, 2011 10:23 am |
| <b>Tiled QR operation count</b><br>by gul - Mon Apr 18, 2011 5:53 pm                          | 4       | 168   | by gul - Tue Apr 19, 2011 3:40 pm      |
| <b>C++ Interface to plasma?</b><br>by kaveh.kohan - Tue Mar 29, 2011 1:42 pm                  | 1       | 105   | by admin - Tue Mar 29, 2011 3:45 pm    |
| <b>TILED LU Operation Count</b><br>by rational - Mon Mar 07, 2011 2:23 pm                     | 1       | 151   | by admin - Mon Mar 07, 2011 2:53 pm    |
| <b>PLASMA tests on CGELS , CGELQF ... falls on Mac</b><br>by paolo - Thu Feb 17, 2011 8:46 pm | 0       | 137   | by paolo - Thu Feb 17, 2011 8:46 pm    |

Latest PLASMA News

- 2010-11-30 [PLASMA 2.3.1 Release](#)
- 2010-11-16 [PLASMA 2.3.0 Release](#)
- 2010-07-10 [New versions of PLASMA v2.2.0 and the PLASMA installer v1.2.0 have been released!](#)
- 2009-11-15 [PLASMA 2.1.0 released!](#)
- 2009-07-04 [PLASMA 2.0.0 released!](#)

# PLASMA Documentation: ASCII

- README
- LICENSE
- Release Notes
- Installation Guide
- Cmake Build Notes





# PLASMA Documentatio: HTML Reference

PLASMA: PLASMA\_dgesv - Konqueror

file:///home/koobas/Koobas/PLASMA/plasma/trunk/docs/html/doxygen/group\_double\_gaa19575c91be4c4c

Main Page Modules Data Structures Files Search

Simple Interface - Double Real

```
int PLASMA_dgesv ( int      N,
                  int      NRHS,
                  double * A,
                  int      LDA,
                  double * L,
                  int *    IPIV,
                  double * B,
                  int      LDB
                )
```

PLASMA\_dgesv - Computes the solution to a system of linear equations  $A * X = B$ , where  $A$  is an  $N$ -by- $N$  matrix and  $X$  and  $B$  are  $N$ -by- $NRHS$  matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor  $A$ . The factored form of  $A$  is then used to solve the system of equations  $A * X = B$ .

**Parameters:**

|           |        |  |
|-----------|--------|--|
| [in]      | $N$    | The number of linear equations, i.e., the order of the matrix $A$ . $N \geq 0$ .   |
| [in]      | $NRHS$ | The number of right hand sides, i.e., the number of columns of the matrix $B$ . $NRHS \geq 0$ .  |
| [in, out] | $A$    | On entry, the $N$ -by- $N$ coefficient matrix $A$ . On exit, the tile $L$ and $U$ factors from the factorization (not equivalent to LAPACK). |
| [in]      | $LDA$  | The leading dimension of the array $A$ . $LDA \geq \max(1, N)$ .   |
| [out]     | $L$    | On exit, auxiliary factorization data, related to the tile $L$ factor, necessary to solve the system of equations.                           |
| [out]     | $IPIV$ | On exit, the pivot indices that define the permutations (not equivalent to LAPACK).  |
| [in, out] | $B$    | On entry, the $N$ -by- $NRHS$ matrix of right hand side matrix $B$ . On exit, if return value = 0, the $N$ -by- $NRHS$ solution matrix $X$ . |
| [in]      | $LDB$  | The leading dimension of the array $B$ . $LDB \geq \max(1, N)$ .   |

**Returns:**

**Return values:**

|                   |   |
|-------------------|---|
| $PLASMA\_SUCCESS$ | successful exit   |
| $< 0$             | if $-i$ , the $i$ -th argument had an illegal value   |
| $> 0$             | if $i$ , $U(i,i)$ is exactly zero. The factorization has been completed, but the factor $U$ is exactly singular, so the solution could not be computed. |

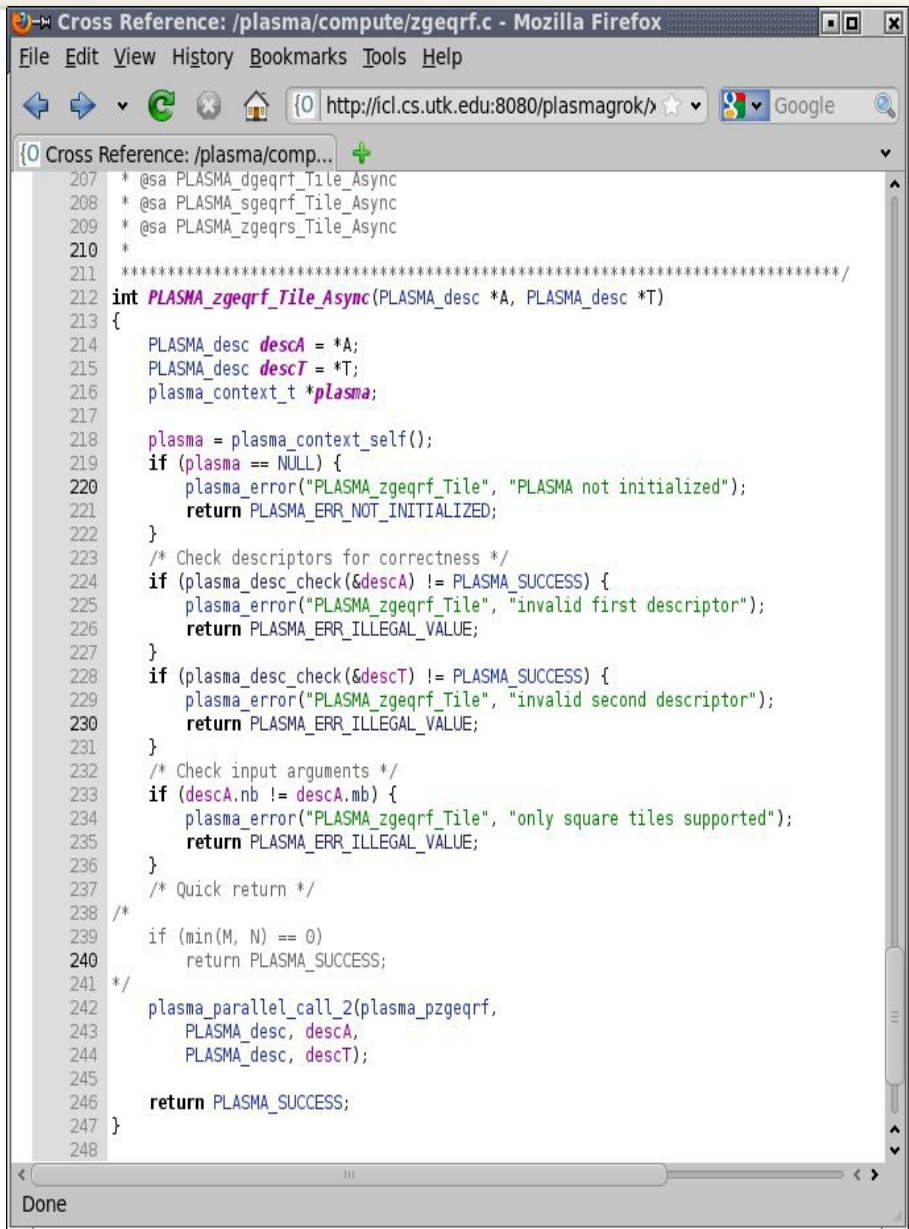
**See also:**

- [PLASMA\\_dgesv\\_Tile](#)
- [PLASMA\\_dgesv\\_Tile\\_Async](#)
- [PLASMA\\_cgesv](#)
- [PLASMA\\_dgesv](#)
- [PLASMA\\_sgesv](#)
- [PLASMA\\_dcgesv](#)

Generated on Thu Apr 22 17:42:51 2010 for PLASMA by [doxygen](#) 1.6.1

- Grouping by precision
  - Z, C, D, C
  - ZC, DS
- Grouping by interface
  - standard
  - tile
  - tile Async
- “see also” section

# PLASMA Documentation: Src Code Browser

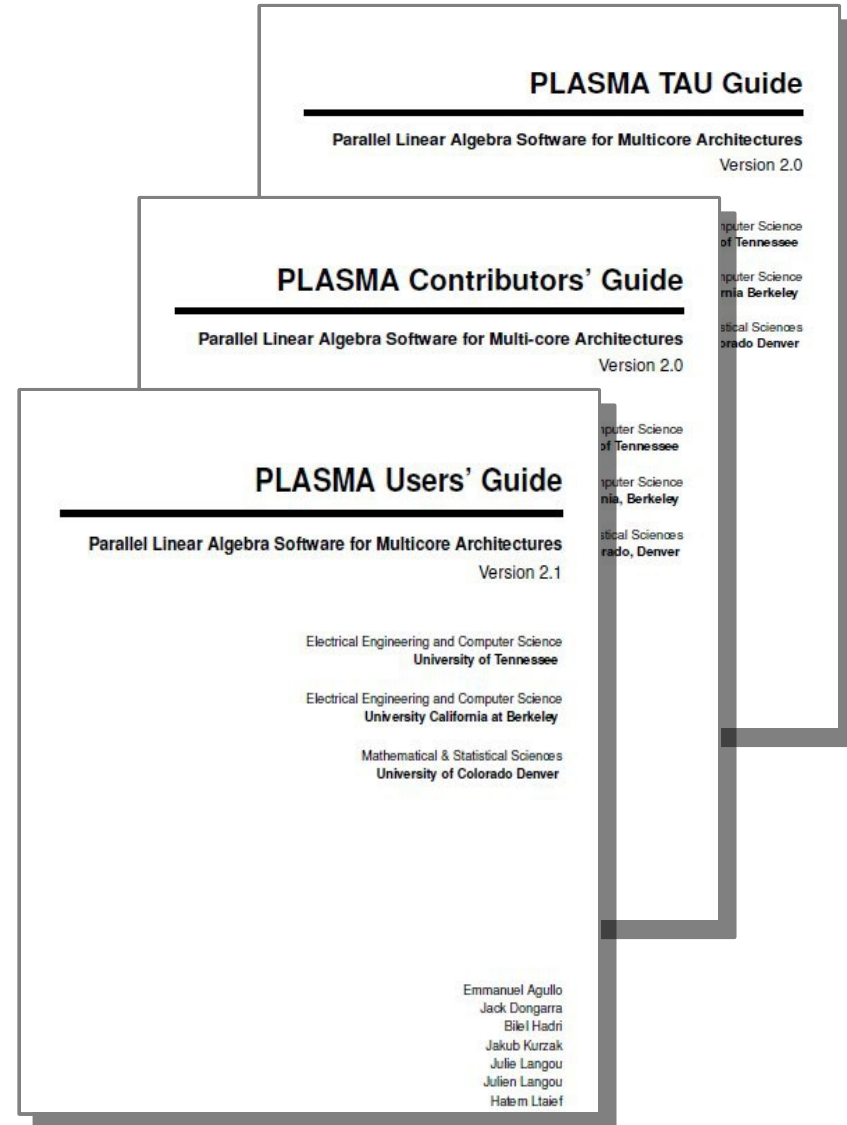


```
207 * @sa PLASMA_dgeqrf_Tile_Async
208 * @sa PLASMA_sgeqrf_Tile_Async
209 * @sa PLASMA_zgeqrs_Tile_Async
210 *
211 *****/
212 int PLASMA_zgeqrf_Tile_Async(PLASMA_desc *A, PLASMA_desc *T)
213 {
214     PLASMA_desc descA = *A;
215     PLASMA_desc descT = *T;
216     plasma_context_t *plasma;
217
218     plasma = plasma_context_self();
219     if (plasma == NULL) {
220         plasma_error("PLASMA_zgeqrf_Tile", "PLASMA not initialized");
221         return PLASMA_ERR_NOT_INITIALIZED;
222     }
223     /* Check descriptors for correctness */
224     if (plasma_desc_check(&descA) != PLASMA_SUCCESS) {
225         plasma_error("PLASMA_zgeqrf_Tile", "invalid first descriptor");
226         return PLASMA_ERR_ILLEGAL_VALUE;
227     }
228     if (plasma_desc_check(&descT) != PLASMA_SUCCESS) {
229         plasma_error("PLASMA_zgeqrf_Tile", "invalid second descriptor");
230         return PLASMA_ERR_ILLEGAL_VALUE;
231     }
232     /* Check input arguments */
233     if (descA.nb != descA.mb) {
234         plasma_error("PLASMA_zgeqrf_Tile", "only square tiles supported");
235         return PLASMA_ERR_ILLEGAL_VALUE;
236     }
237     /* Quick return */
238     /*
239     if (min(M, N) == 0)
240         return PLASMA_SUCCESS;
241     */
242     plasma_parallel_call_2(plasma_pzgeqrf,
243         PLASMA_desc, descA,
244         PLASMA_desc, descT);
245
246     return PLASMA_SUCCESS;
247 }
248
```

- Syntax highlighting
- Clickable function calls
- All software layers available from Netlib
  - PLASMA
  - QUARK
  - core BLAS
  - LAPACK C wrapper
  - LAPACK
  - Netlib CBLAS
  - Netlib BLAS

# PLASMA Documentation: PDF Manuals

- PLASMA Users' Guide
- QUARK Users' Guide
- PLASMA Contributors' Guide
- PLASMA TAU Guide



# Installation: Python Installer

- Downloads and installs PLASMA
  - No need to download the PLASMA tarball, just the installer script.
- Downloads and installs all dependencies
  - CBLAS,
  - LAPACK,
  - LAPACKE C API.
- The installer can also download and install Netlib BLAS.
- However,
  - Netlib BLAS is unoptimized (reference implementation), and
  - Will deliver very low performance.

# Installation Python Installer

-h or --help

destination directories

./setup.py --prefix="" [DIR] (./install)  
--build="" [DIR] (./build)

compilers

--cc="" [CMD] (cc)  
--fc="" [CMD] (gfortran)

compilation flags

--cflags="" [FLAGS] (-O2)  
--fflags="" [FLAGS] (-O2)

linking flags

--ldflags\_c="" [FLAGS]  
--ldflags\_fc="" [FLAGS]

build tool

--make= [CMD] (make)

locations of libraries

--blaslib="" [LIB]  
--cblaslib="" [LIB]  
--lapacklib="" [LIB]  
--lapclib="" [LIB]

force download

--downblas  
--downcblas  
--downlapack  
--downlapc  
--downall

--[no]testing (testing)  
--nbccores= [#CORES] (half)

--clean  
--src

# Installation Example

```
./setup.py
```

using default settings

```
./setup.py --prefix="/opt"  
--cc=icc  
--fc=ifort  
--blaslib="-L/intel/mkl/lib/em64t  
-lmkl_intel_lp64  
-lmkl_sequential  
-lmkl_core"  
--downlapack  
--downcblas
```

using custom settings

- install libraries in /opt
- use Intel compilers (ICC & IFORT)
- use MKL (EM64T)
- use LAPACK from Netlib
- use CBLAS from Netlib

# Installation Outcome

**/build**

/BLAS

/CBLAS

/lapack-3.4.0

/lapacke-3.4.0

/plasma-2.5.0

/download

/log

**/install**

/include

/lib

`./setup.py --downall`

blas.tgz

cblas.tgz

lapack.tgz

lapacke.tgz

plasma.tar.gz

# PLASMA Directory Structure

/compute

/control

/core\_blas

**/docs** ←

**/examples** ←

/include

/lib

/makes

/quark

/testing

/timing

/tools

- **All PLASMA documentation**

- Users' Guide
- Reference Guide
- Installation Guide
- .....

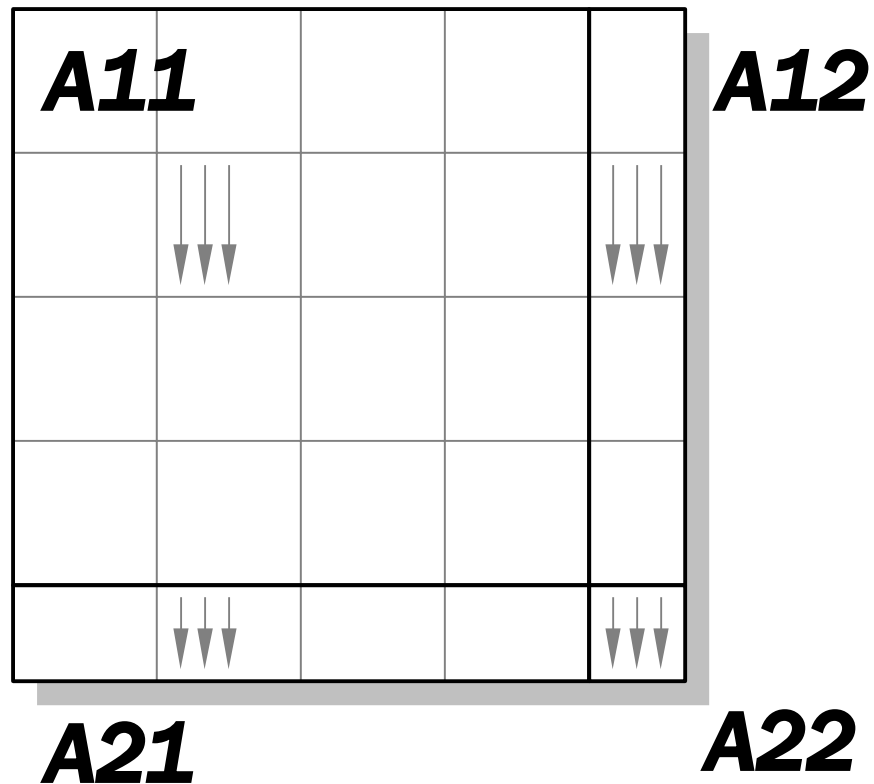
- **Simple usage examples**

- **Ideal to cut & paste**

- C
- FORTRAN
- **All precisions**



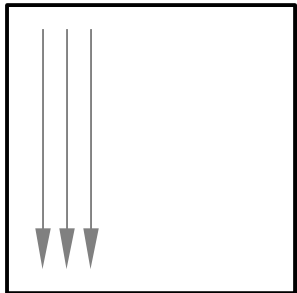
# Matrix Layout: PLASMA Native Layout



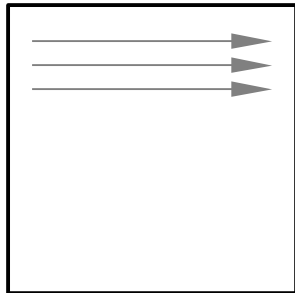
- One continuous chunk of memory containing (in order): A11, A21, A12, A22
- Each tile is continuous (column-major order)
- Tiles are stored in column-major order (A11)

# Matrix Layout: In-Place Translations

CM



RM

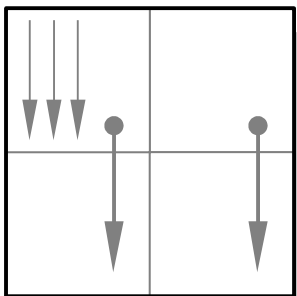


Column Major / Row Major

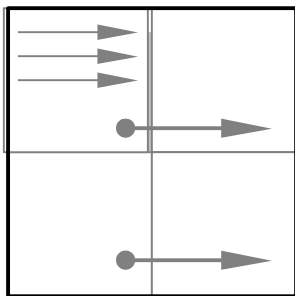
```
PLASMA_sgecfi()  
PLASMA_cgecfi()  
PLASMA_dgecfi()  
PLASMA_zgecfi()
```

C|R C|R Rectangular Block

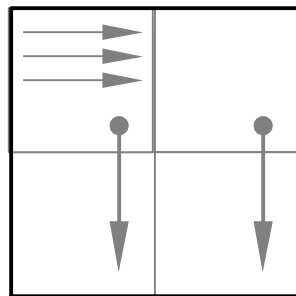
CCRB



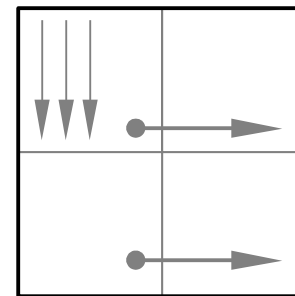
RRRB



RCRB



CRRB

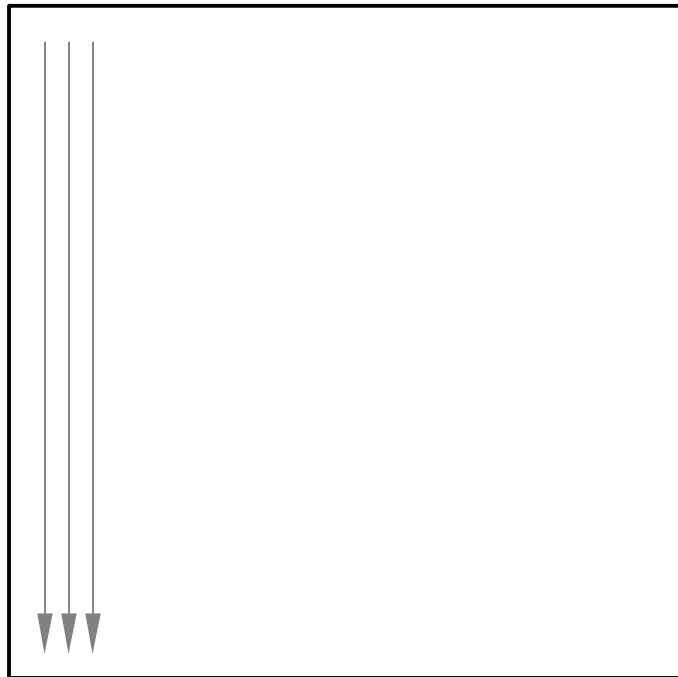


- PLASMA provides translation between each two layouts
- PLASMA only accepts input matrices in the CM and CCRB layouts

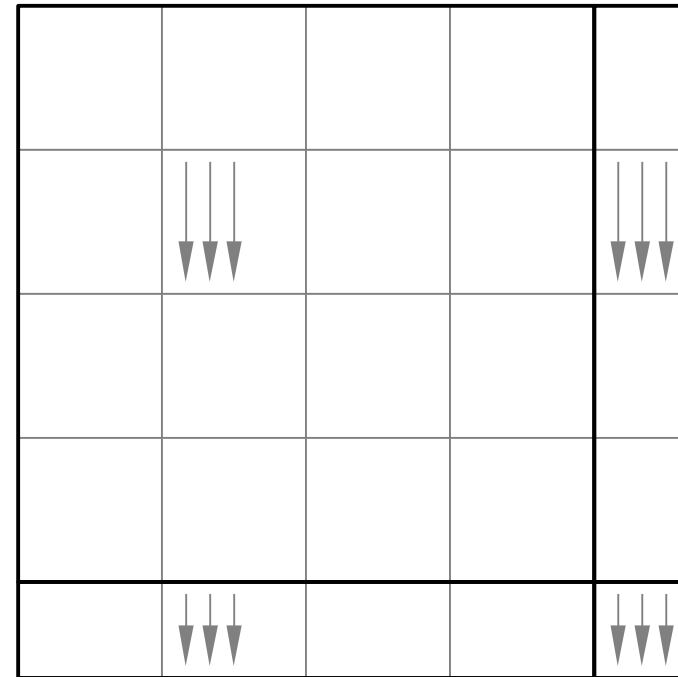
(square tiles only)

# Layout Translation: LAPACK ↔ PLASMA

LAPACK (CM)

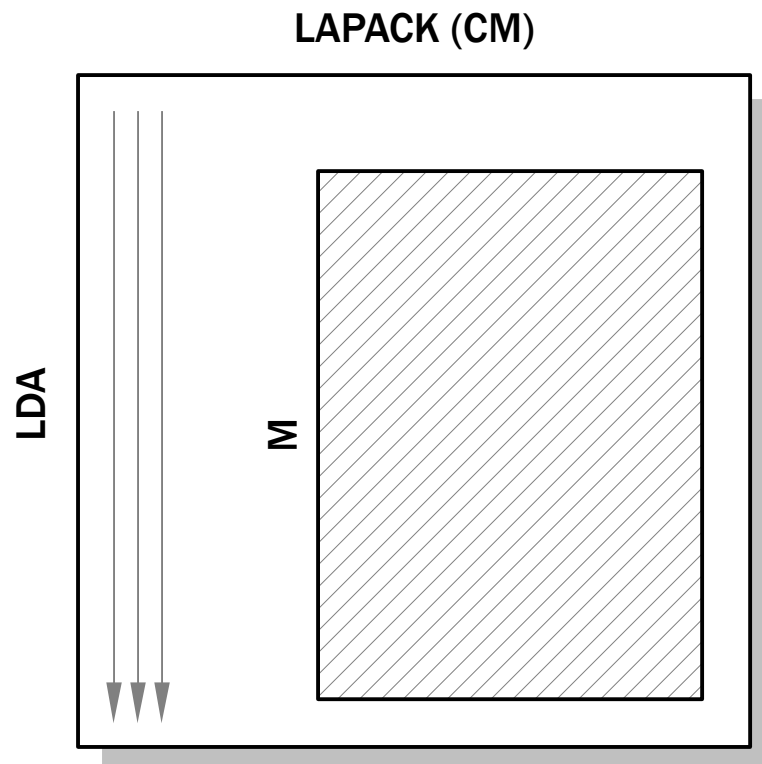


PLASMA (CCRB)



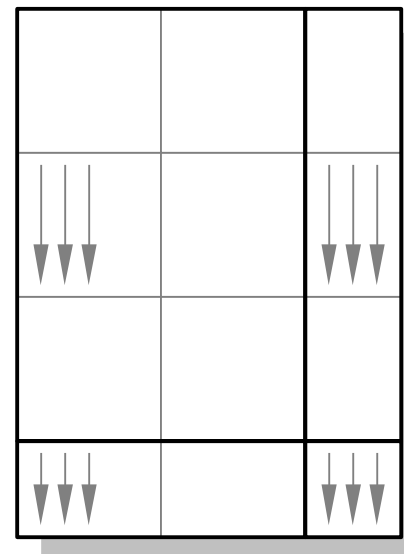
- **Tile layout & LAPACK layout accepted**
- **LAPACK layout always converted to tile layout (CCRB / square tiles)**
  - in place translation
  - out of place translation

# Layout Translation: Out-of-Place Translation



```
PLASMA_[scdz]Lapack_to_Tile()  
PLASMA_[scdz]Tile_to_Lapack()
```

PLASMA (CCRB)

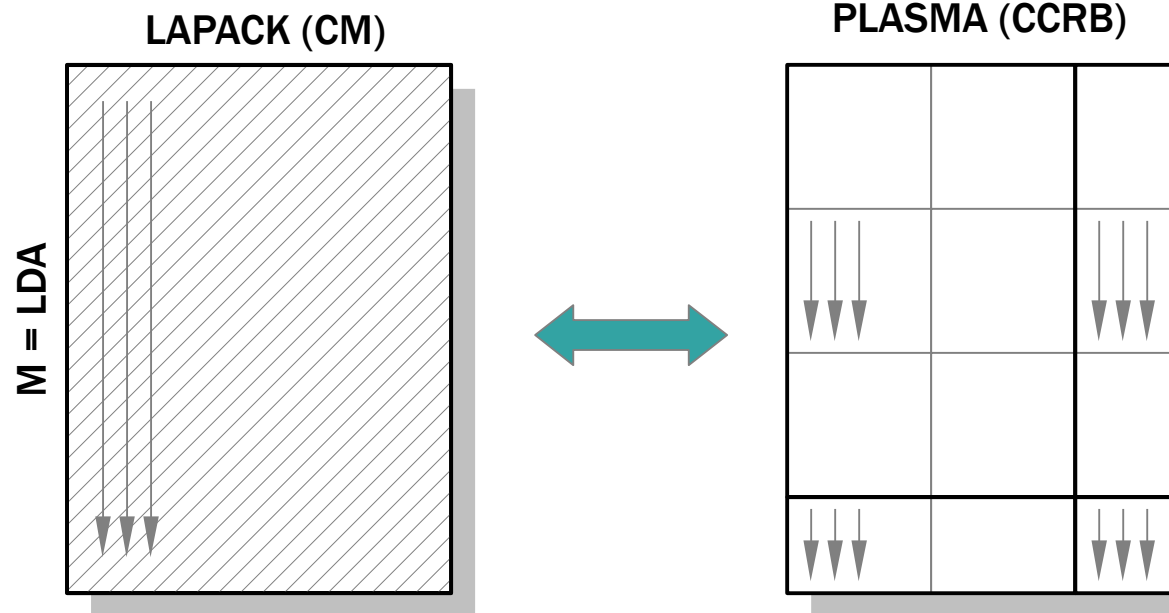


## • Out-of-Place Translation

- $M \leq LDA$
- $CM \leftrightarrow CCRB$  only
- storage overhead
- fast (parallel & cache efficient)

# Layout Translation: In-Place Translation

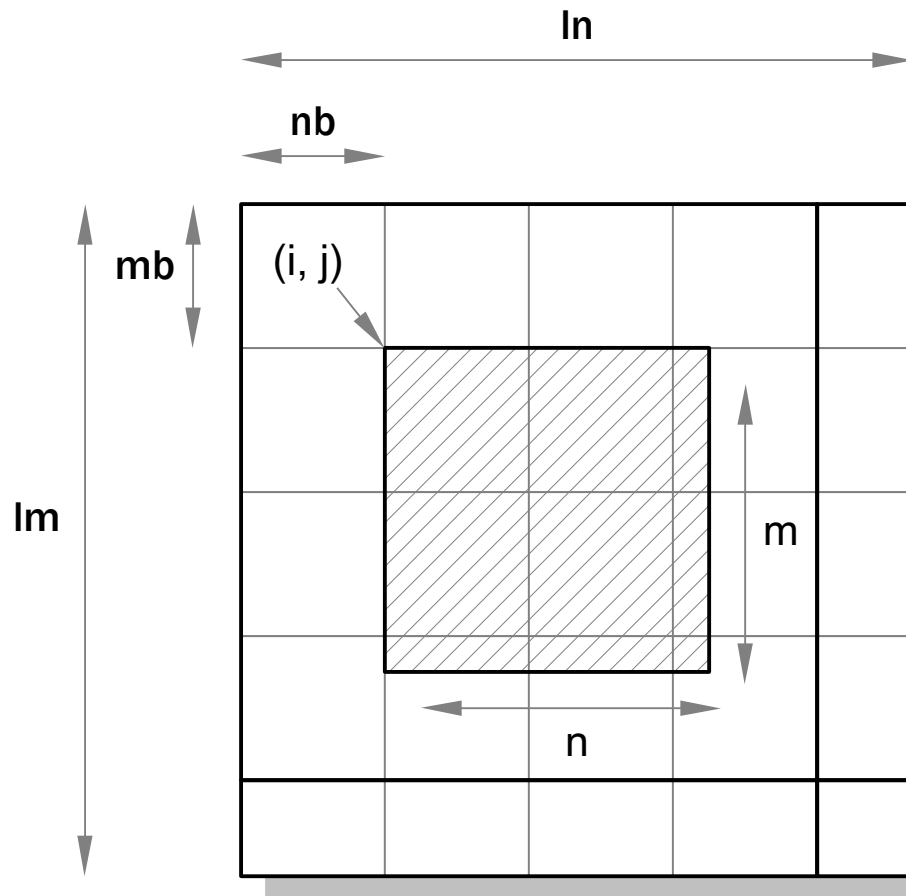
PLASMA\_<sub>[scdz]</sub>gec<sub>fi</sub>( )



- **In-Place Translation**

- M = LDA
- any layout
- no storage overhead
- fast (parallel & cache efficient)

# Tile Layout: Matrix Descriptor



```
PLASMA_Desc_Create()  
PLASMA_Desc_Destroy()
```

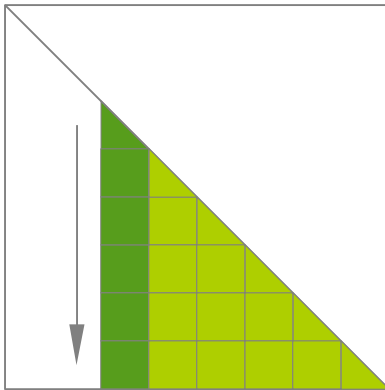
- **Tile Interface**

- tiles have to be square ( $mb = nb$ )
- submatrix has to be tile-aligned

$$(i \% mb = 0, j \% nb = 0)$$

# Algorithms: Tile Cholesky

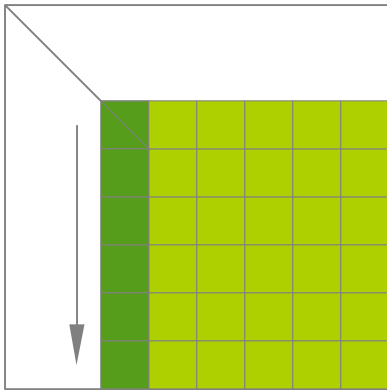
```
PLASMA_[scdz]potrf[_Tile][_Async]()
```



- **Algorithm**
  - equivalent to LAPACK
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable to vendor on few cores
  - much better than vendor with many cores

# Algorithms: LU with Partial Pivoting

```
PLASMA_[scdz]getrf[_Tile][_Async]()
```



- **Algorithm**

- equivalent to LAPACK
- same pivot vector
- same L and U factors
- same forward substitution procedure
- parallel recursive panel factorization

- **Numerics**

- same as LAPACK

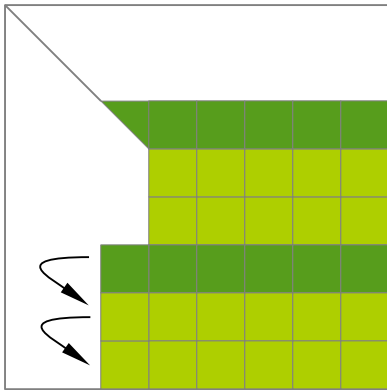
- **Performance**

- comparable to vendor on few cores
- much better than vendor with many cores



# Algorithms:QR & “Domino” Panel Reduction

```
PLASMA_[scdz]geqrt[_Tile][_Async]()
```

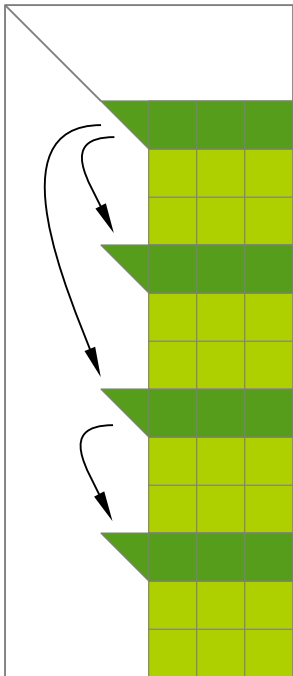


- **Algorithm**
  - the same R factor as LAPACK (absolute values)
  - different set of Householder reflectors
  - different Q matrix
  - different Q generation / application procedure
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable to vendor on few cores
  - much better than vendor with many cores

# Algorithms: QR with “Tree” Panel Reduction

```
PLASMA_[scdz]geqrt[_Tile][_Async]()
```

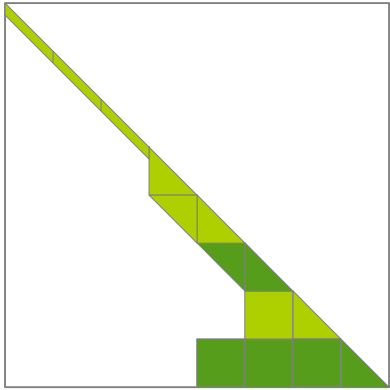
```
PLASMA_Set(  
    PLASMA_HOUSEHOLDER_MODE,  
    PLASMA_TREE_HOUSEHOLDER);
```



- **Algorithm**
  - the same R factor as LAPACK (absolute values)
  - different set of Householder reflectors
  - different Q matrix
  - different Q generation / application procedure
- **Numerics**
  - same as LAPACK
- **Performance**
  - absolutely superior for tall matrices

# Algorithms: Symmetric Eigenvalue Problem

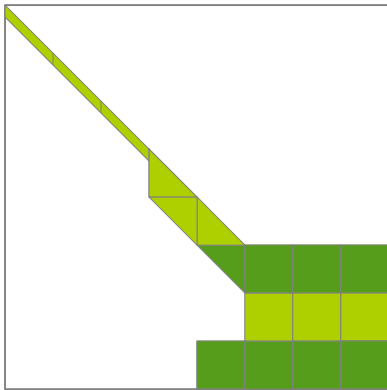
```
PLASMA_[scdz]syev[_Tile][_Async]()
```



- **Algorithm**
  - two-stage tridiagonal reduction + QR iteration
  - fast eigenvalues, slower eigenvectors  
(possibility to calculate a subset)
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable to MKL for very small problems
  - absolutely superior for larger problems

# Algorithms: Singular Value Decomposition

```
PLASMA_[scdz]gesvd[_Tile][_Async]()
```



- **Algorithm**
  - two-stage bidiagonal reduction + QR iteration
  - fast singular values, slower singular vectors  
(possibility of calculating a subset)
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable with MKL for very small problems
  - absolutely superior for larger problems

# Basic Routines: Initialization & Configuration

|                                |                                |
|--------------------------------|--------------------------------|
| <code>PLASMA_Init()</code>     | – initialize PLASMA            |
| <code>PLASMA_Finalize()</code> | – finalize PLASMA              |
| <code>PLASMA_Set()</code>      | – set a parameter              |
| <code>PLASMA_Get()</code>      | – get the value of a parameter |
| <code>PLASMA_Enable()</code>   | – enable a feature             |
| <code>PLASMA_Disable()</code>  | – disable a feature            |
| <code>PLASMA_Version()</code>  | – get PLASMA version number    |

# Computational Routines: Linear Systems

- PLASMA\_dgesv() – solve a linear system using the LU fact.
- PLASMA\_dgetrf() – LU factorization
- PLASMA\_dgetrs() – forward and backward substitution
- PLASMA\_dtrsm() – triangular solve
  
- PLASMA\_dposv() – solve a linear system using the Cholesky fact.
- PLASMA\_dpotrf() – Cholesky factorization
- PLASMA\_dpotrs() – forward and backward substitution
- PLASMA\_dtrsm() – triangular solve
  
- PLASMA\_dtrtri() – compute an inverse of an SPD matrix

# Computational Routines: Over- (Least Squares), Underdetermined (Min. Norm)

PLASMA\_dgels() – solve the problem using the QR or LQ fact.

PLASMA\_dgeqrf() – QR factorization

PLASMA\_dgelqf() – LQ factorization

PLASMA\_dormqr() – multiply by the Q matrix (QR)

PLASMA\_dormlq() – multiply by the Q matrix (LQ)

PLASMA\_dorgqr() – generate the Q matrix (QR)

PLASMA\_dorglq() – generate the Q matrix (LQ)

PLASMA\_dtrsm() – triangular solve

# Computational Routines: Symmetric Eigenvalues, Singular Values

PLASMA\_dgesvd() – find singular values

PLASMA\_dsyev() – find eigenvalues (symmetric system)

PLASMA\_dsygv() – find generalized eigenvalues  
(positive definite)



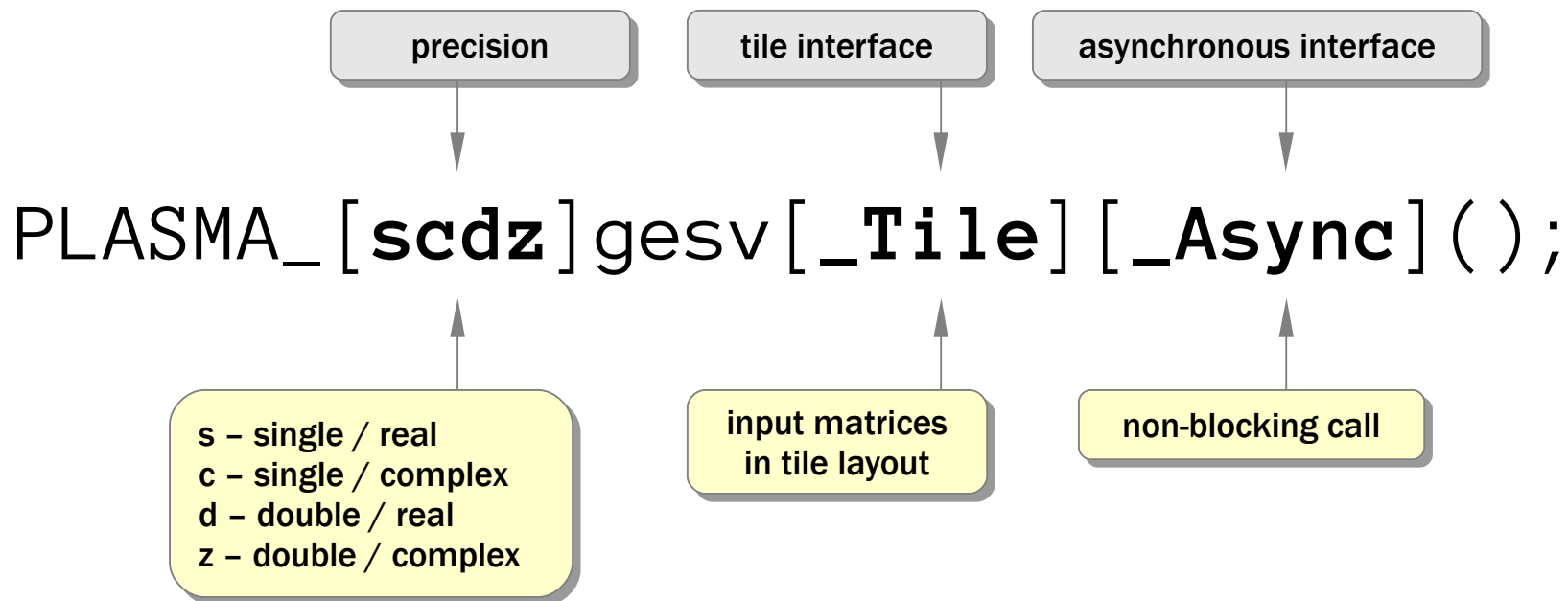
# Computational Routines: Tile BLAS 3 – Real

- PLASMA\_dgemm() – matrix multiply
- PLASMA\_dsymm() – symmetric matrix multiply
- PLASMA\_dsyr2k() – symmetric rank 2k update
- PLASMA\_dsyrk() – symmetric rank k update
- PLASMA\_dtrmm() – triangular matrix multiply
- PLASMA\_dtrsm() – triangular solve

# Computational Routines: Tile BLAS3 Complex

|                              |                              |
|------------------------------|------------------------------|
| <code>PLASMA_zgemm()</code>  | – matrix multiply            |
| <code>PLASMA_zsymm()</code>  | – symmetric matrix multiply  |
| <code>PLASMA_zhemm()</code>  | – Hermitian matrix multiply  |
| <code>PLASMA_zsyr2k()</code> | – symmetric rank 2k update   |
| <code>PLASMA_zher2k()</code> | – Hermitian rank 2k update   |
| <code>PLASMA_zsyrk()</code>  | – symmetric rank k update    |
| <code>PLASMA_zherk()</code>  | – Hermitian rank k update    |
| <code>PLASMA_ztrmm()</code>  | – triangular matrix multiply |
| <code>PLASMA_ztrsm()</code>  | – triangular solve           |

# Computational Routines: Precisions & Interfaces



# Computational Routines: Precisions / Interfaces

```
PLASMA_sgesv();  
PLASMA_cgesv();  
PLASMA_dgesv();  
PLASMA_zgesv();
```

**LAPACK Interface**

```
PLASMA_sgesv_Tile();  
PLASMA_cgesv_Tile();  
PLASMA_dgesv_Tile();  
PLASMA_zgesv_Tile();
```

**Tile Layout**

```
PLASMA_sgesv_Tile_Async();  
PLASMA_cgesv_Tile_Async();  
PLASMA_dgesv_Tile_Async();  
PLASMA_zgesv_Tile_Async();
```

**Tile, No blocking**

# Hello World; LAPACK Interface

PLASMA\_NUM\_THREADS

hwloc - HWLOC\_OBJ\_CORE

sysconf - SC\_NPROCESSORS\_ONLN

```
PLASMA_Init(0);
```

LAPACK layout

LAPACK layout

```
PLASMA_dgesv(n, nrhs, A, lda, PIV, B, ldb);
```

number of equations

number of right hand sides

system matrix

leading dimension of A

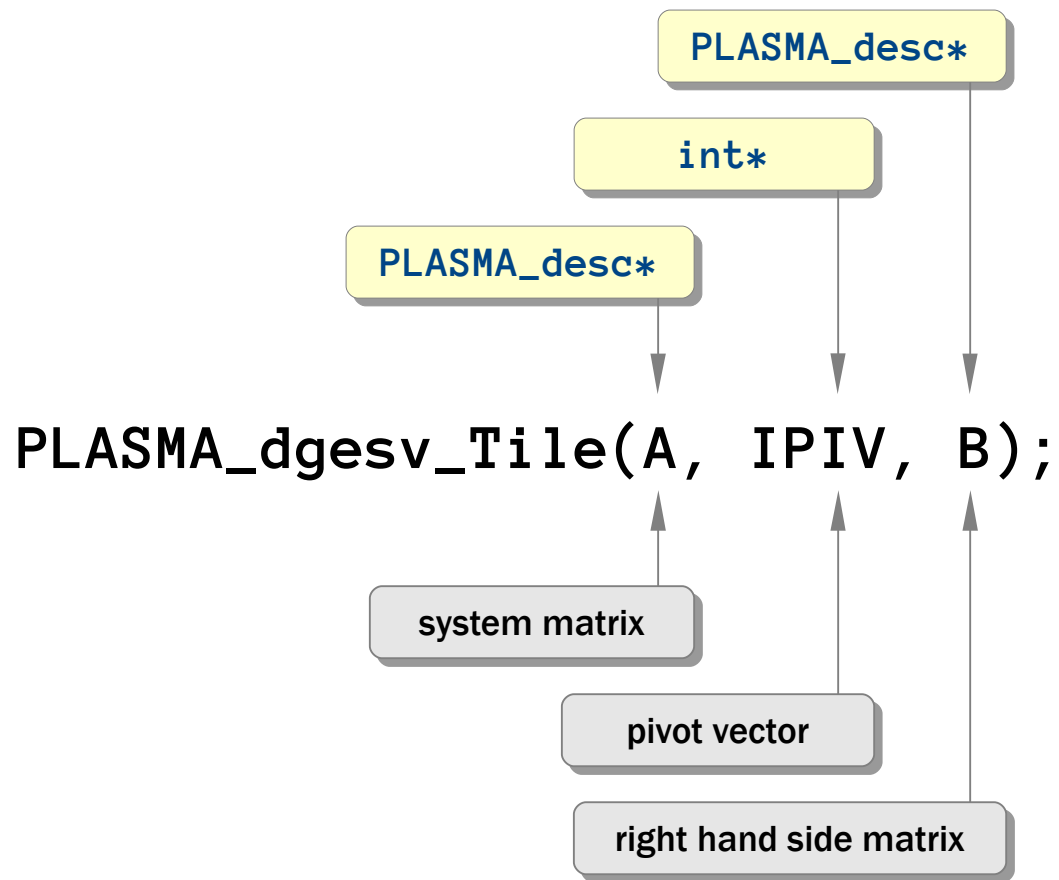
right hand side matrix

pivot vector

leading dimension of B

```
PLASMA_Finalize();
```

# Tile Interface: Linear System



# Tile Interface: Linear System

```
PLASMA_Init(0);
```

```
PLASMA_Desc_Create(&descA, ...);
```

```
PLASMA_Desc_Create(&descB, ...);
```

```
PLASMA_dLapack_to_Tile(A, lda, descA);
```

```
PLASMA_dLapack_to_Tile(B, ldb, descB);
```

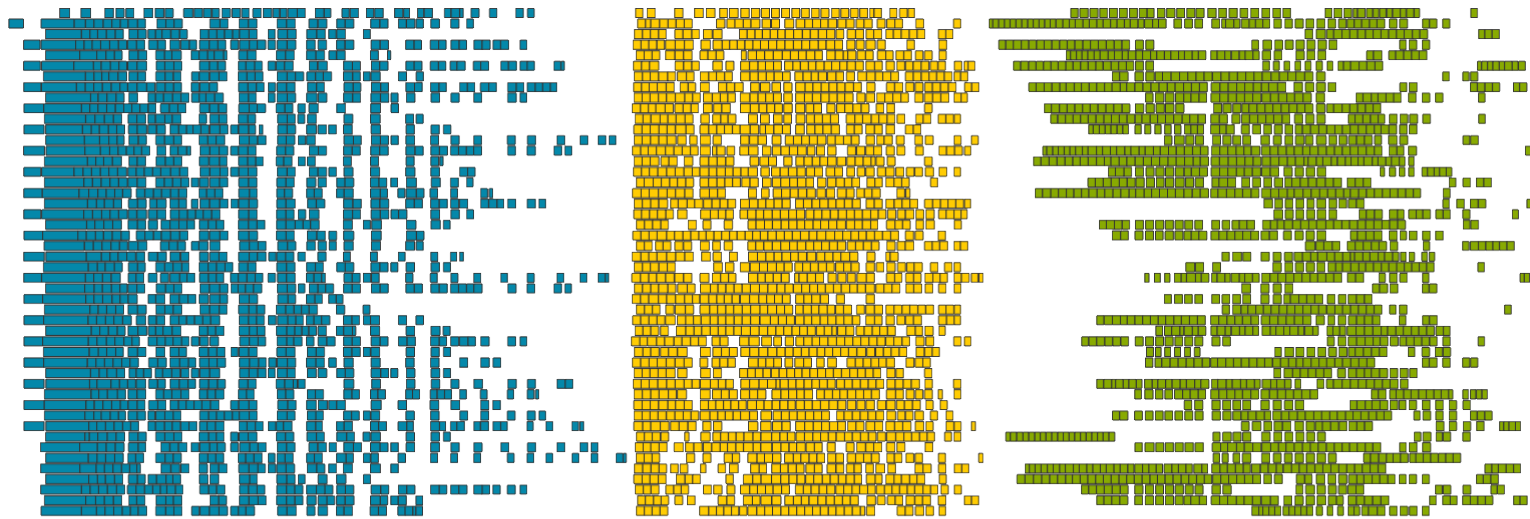
```
PLASMA_dgesv_Tile(descA, IPIV, descB);
```

```
PLASMA_Desc_Destroy(&descA);
```

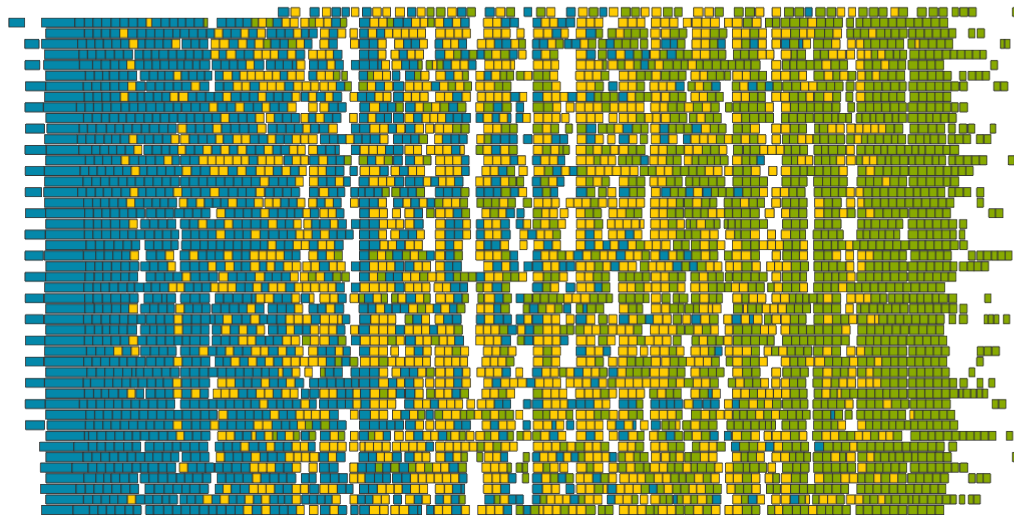
```
PLASMA_Desc_Destroy(&descB);
```

```
PLASMA_Finalize();
```

# Tile Asynch Interface: Motivation



synchronous

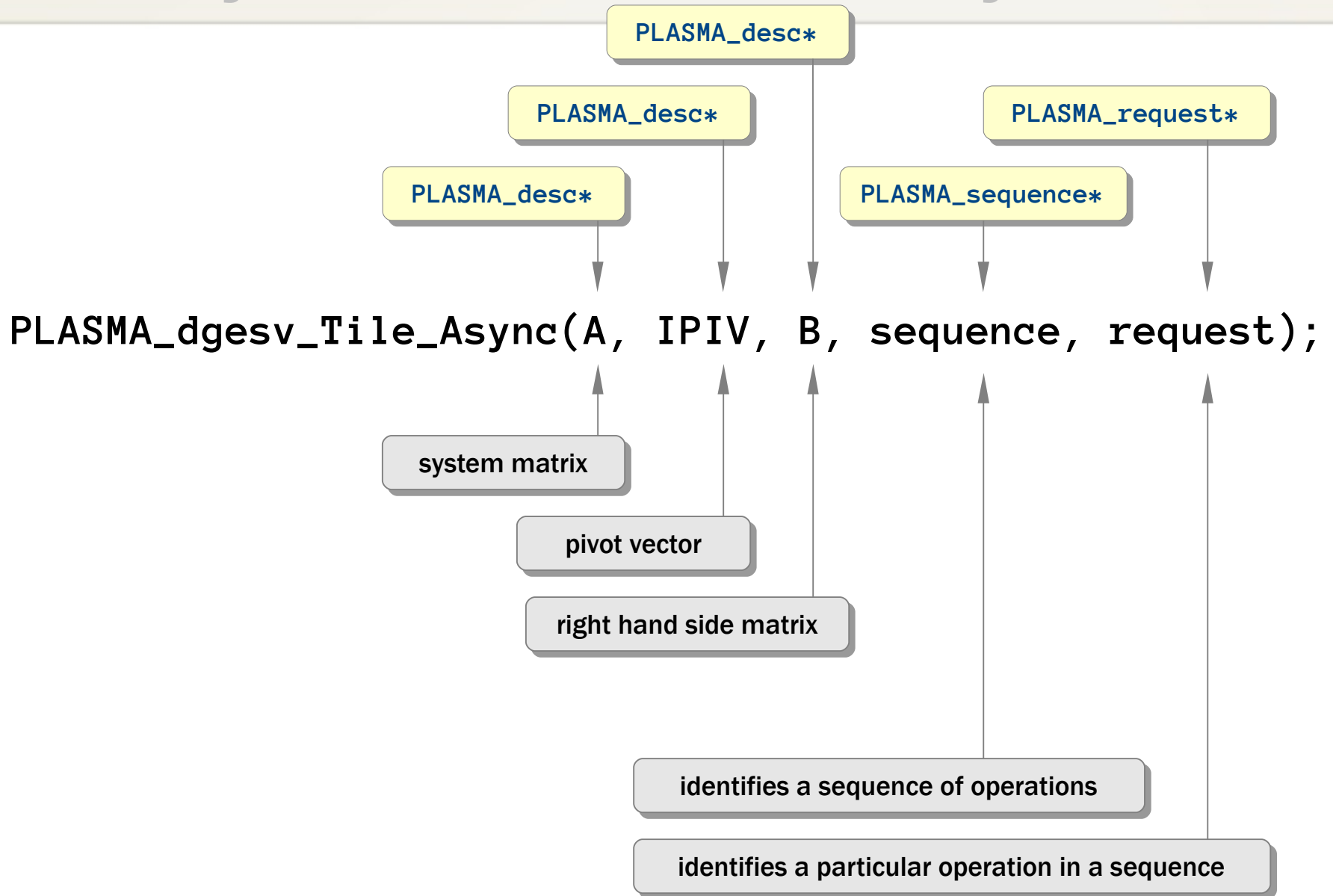


asynchronous

POTRI :  
POTRF  
TRTRI  
LAUUM



# Tile Async Interface: Linear System



# Tile Async Interface: Linear System

```
PLASMA_Init(0);
```

```
PLASMA_sequence *sequence = NULL;
```

```
PLASMA_request request1 = PLASMA_REQUEST_INITIALIZER;
```

```
PLASMA_request request2 = PLASMA_REQUEST_INITIALIZER;
```

```
PLASMA_Sequence_Create(&sequence);
```

```
PLASMA_dgetrf_Tile_Async(..., sequence, &request1);
```

```
PLASMA_dgetrs_Tile_Async(..., sequence, &request2);
```

```
PLASMA_Sequence_Wait(sequence);
```

```
PLASMA_Sequence_Destroy(sequence);
```

```
PLASMA_Finalize();
```

# Tile Async Interface: Error Handling

- IF NO Errors
  - `sequence->status == PLASMA_SUCCESS`
- IF Error
  - Entire sequence is canceled,
  - including tasks queued but not yet executed
  - as well as tasks not yet queued
  - `sequence->status` returns the error code
  - `sequence->request` returns the failed request

# Workspace: Allocation

for auxiliary data that need to be passed  
from one routine to another

```
int M;  
int N;  
PLASMA_desc *descT;
```

```
PLASMA_Alloc_Workspace_dgels (M, N, &descT);  
PLASMA_Alloc_Workspace_dgeqrf(M, N, &descT);  
PLASMA_Alloc_Workspace_dgelqf(M, N, &descT);
```

```
PLASMA_Alloc_Workspace_dgels_Tile (M, N, &descT);  
PLASMA_Alloc_Workspace_dgeqrf_Tile(M, N, &descT);  
PLASMA_Alloc_Workspace_dgelqf_Tile(M, N, &descT);
```

# Workspaces: Deallocation

```
PLASMA_Init(0);  
  
PLASMA_Alloc_Workspace_dgels(m, n, &descT);  
  
PLASMA_dgeqrf(m, n, A, lda, descT);  
PLASMA_dgeqrs(m, n, nrhs, A, lda, descT, B, ldb);  
  
PLASMA_Dealloc_Handle(&descT);  
  
PLASMA_Finalize();
```

LAPACK interface

```
PLASMA_Init(0);  
  
PLASMA_Alloc_Workspace_dgeqrf_Tile(m, n, &descT);  
  
PLASMA_dgeqrf_Tile(descA, descT);  
PLASMA_dgeqrs_Tile(descA, descT, descB);  
  
PLASMA_Dealloc_Handle_Tile(&descT);  
  
PLASMA_Finalize();
```

tile interface

# Settings: Enable / Disable

- `PLASMA_Enable(int flag)`
- `PLASMA_Disable(int flag)`
  - `PLASMA_WARNINGS`
  - `PLASMA_ERRORS`
  - `PLASMA_AUTOTUNING`

not a true autotuning  
(using hardcoded settings)

## Call

```
PLASMA_Disable(PLASMA_AUTOTUNING);  
to do manual tuning
```

# Settings: Set / Get Blocking

- PLASMA\_Set(int param, int value);
- PLASMA\_Get(int param, int \*value);
- param:

– PLASMA\_TILE\_SIZE

affects everything


– PLASMA\_INNER\_BLOCK\_SIZE

affects everything except BLAS 3 & Cholesky


# Settings: Set / Get Scheduling

- `PLASMA_Set(PLASMA_SCHEDULING_MODE, int mode);`
- `PLASMA_Get(PLASMA_SCHEDULING_MODE, int *mode);`
- `mode`:
  - `PLASMA_STATIC_SCHEDULING`
  - `PLASMA_DYNAMIC_SCHEDULING`

use static scheduling where  
static implementation exists



use dynamic scheduling where  
dynamic implementation exists



some routines only have static implementations  
some routines only have dynamic implementations  
some routines have both  
PLASMA switches between the two modes at runtime



# Settings: Set / Get Reductions

- `PLASMA_Set(PLASMA_HOUSEHOLDER_MODE, int mode)`
- `PLASMA_Get(PLASMA_HOUSEHOLDER_MODE, int *mode)`

- **mode:**

- `PLASMA_FLAT_HOUSEHOLDER`

use “domino” reduction  
(serial panel reduction)

- `PLASMA_TREE_HOUSEHOLDER`

use “tree” reduction  
(parallel panel reduction)

- `PLASMA_Set(PLASMA_HOUSEHOLDER_SIZE, int size)`
- `PLASMA_Get(PLASMA_HOUSEHOLDER_SIZE, int *size)`

size of the serial pieces of the panel in the tree reduction

# Settings: Set / Get Translation

- `PLASMA_Set(PLASMA_TRANSLATION_MODE, int mode);`
- `PLASMA_Get(PLASMA_TRANSLATION_MODE, int *mode);`

- **mode:**

- `PLASMA_INPLACE`

parallel and cache efficient  
in-place layout translation

- `PLASMA_OUTOFPLACE`

parallel and cache efficient  
out-of-place layout translation