ECP 2018 Annual Meeting Tutorials

Accelerating Linear Algebra with MAGMA

Stan Tomov, Mark Gates, and Azzam Haidar
Innovative Computing Laboratory
University of Tennessee, Knoxville

Exascale Computing Project
2nd Annual Meeting
Knoxville, TN
February 9, 2018

Outline

- **Part I**

  Overview of dense linear algebra libraries
  Design principles and fundamentals

- **Part II**

  MAGMA Overview
  Availability, routines, code, testers, methodology

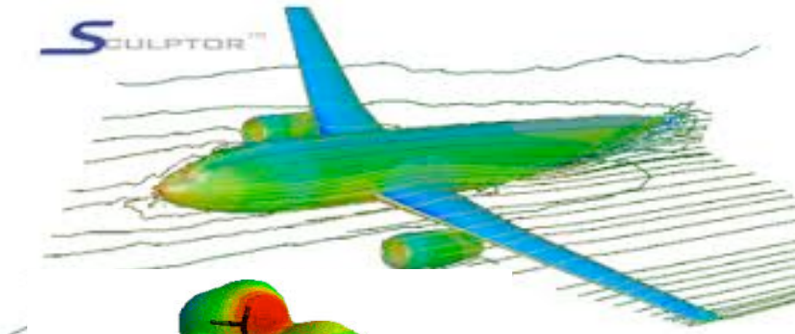
- **Part III**

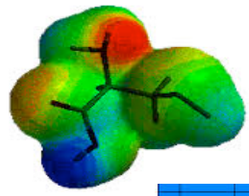
  MAGMA Batched
  MAGMA Sparse

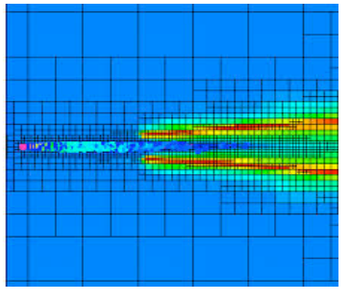
Dense Linear Algebra in Applications

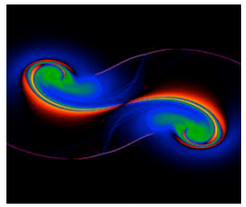
Dense Linear Algebra (DLA) is needed in a wide variety of science and engineering applications:

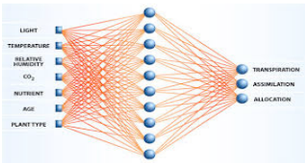
- Linear systems: Solve $Ax = b$
  - Computational electromagnetics, material science, applications using boundary integral equations, airflow past wings, fluid flow around ship and other offshore constructions, and many more
- Least squares: Find x to minimize $\| Ax - b \|$
  - Computational statistics (e.g., linear least squares or ordinary least squares), econometrics, control theory, signal processing, curve fitting, and many more
- Eigenproblems: Solve $Ax = \lambda x$
  - Computational chemistry, quantum mechanics, material science, face recognition, PCA, data-mining, marketing, Google Page Rank, spectral clustering, vibrational analysis, compression, and many more
- SVD: $A = U \Sigma V^*$ ($Au = \sigma v$ and $A^*v = \sigma u$)
  - Information retrieval, web search, signal processing, big data analytics, low rank matrix approximation, total least squares minimization, pseudo-inverse, and many more
- Many variations depending on structure of A
  - A can be symmetric, positive definite, tridiagonal, Hessenberg, banded, sparse with dense blocks, etc.
- DLA is crucial to the development of sparse solvers











ICL

ECP EXASCALE COMPUTING PROJECT

Overview of Dense Numerical Linear Algebra Libraries

netlib.org

BLAS	Kernels for dense linear algebra
LAPACK	Sequential dense linear algebra
ScaLAPACK	Parallel distributed dense linear algebra

icl.utk.edu/research

PLASMA	dense linear algebra (multicore)
MAGMA	Dense/batched/sparse linear algebra (accelerators)
SLATE	dense linear algebra (distributed memory / muticore / accelerators)

new software for multicore and accelerators

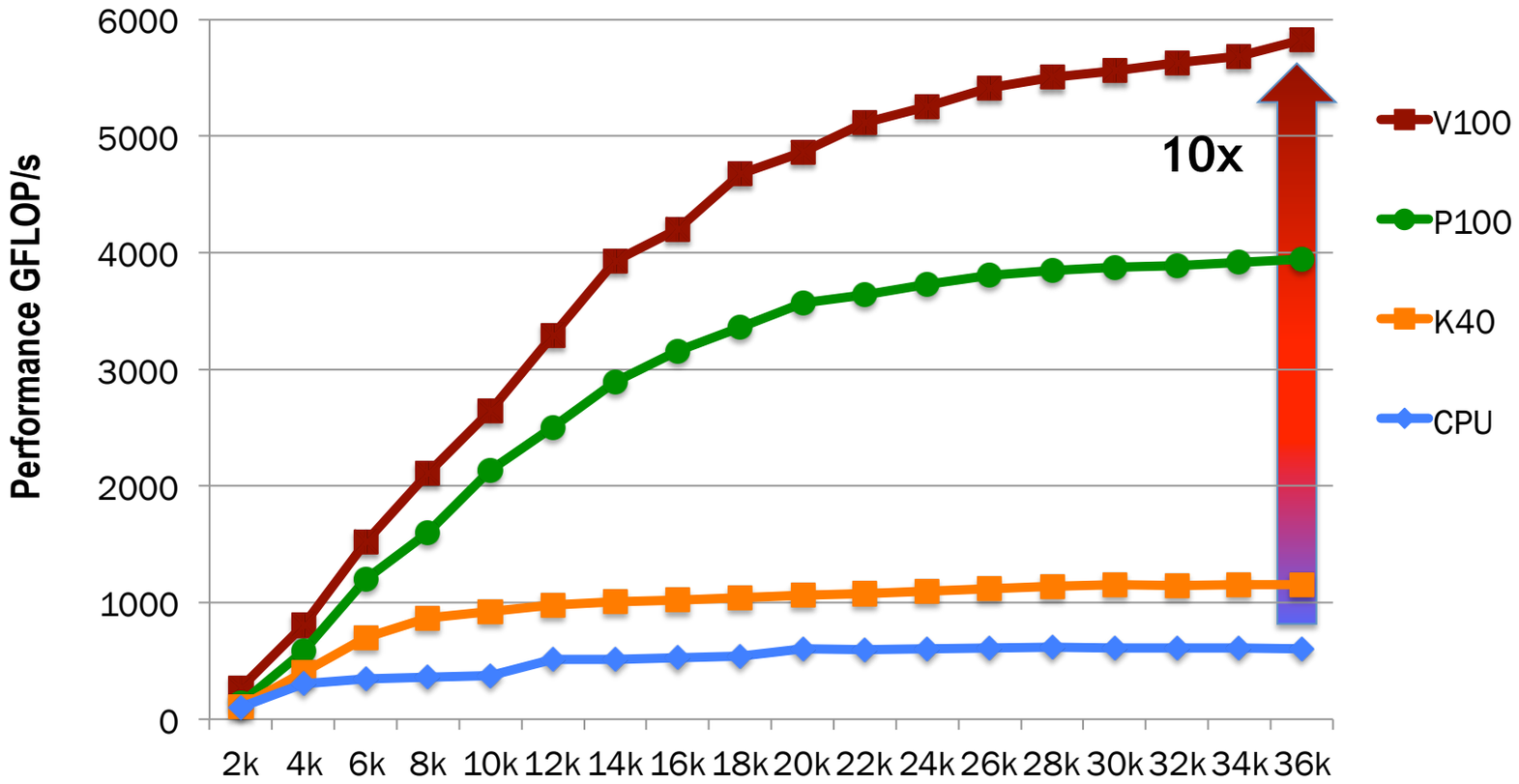
Support from
ECP SLATE, CEED, PEEKS, xSDK

Why use GPUs in HPC?

PERFORMANCE & ENERGY EFFICIENCY

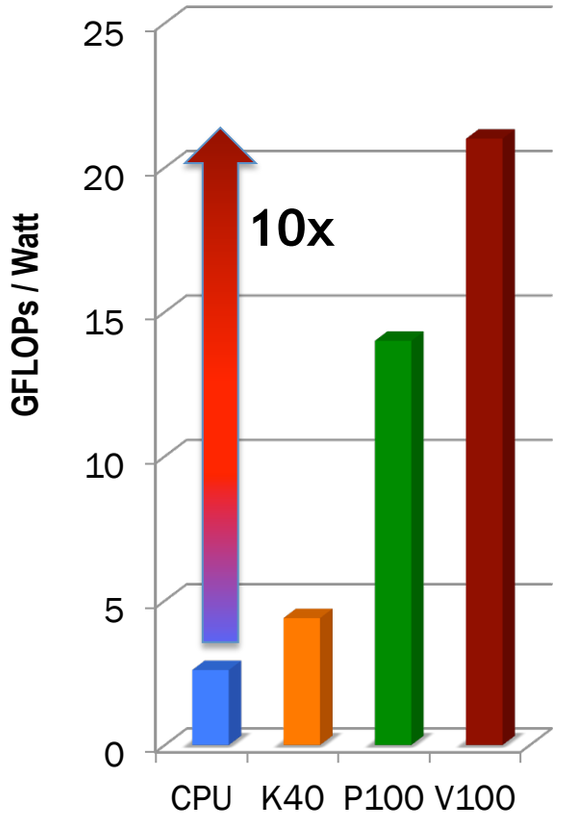
MAGMA 2.3 LU factorization in double precision arithmetic

CPU Intel Xeon E5-2650 v3 (Haswell) 2x10 cores @ 2.30 GHz
K40 NVIDIA Kepler GPU 15 MP x 192 @ 0.88 GHz
P100 NVIDIA Pascal GPU 56 MP x 64 @ 1.19 GHz
V100 NVIDIA Volta GPU 80 MP x 64 @ 1.38 GHz



Matrix size N x N

Energy efficiency (under ~ the same power draw)

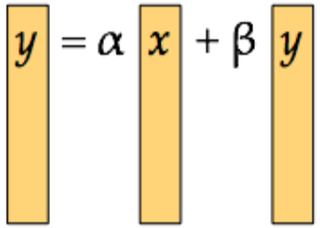


ICL

ECP EXASCALE COMPUTING PROJECT

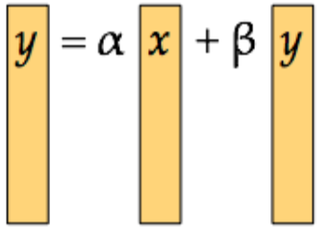
BLAS: Basic Linear Algebra Subroutines

- Level 1 BLAS — vector operations
  - $O(n)$ data and flops (floating point operations)
  - Memory bound:
    $O(1)$ flops per memory access

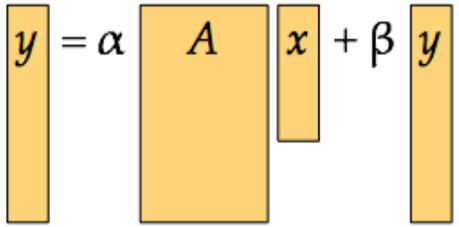


BLAS: Basic Linear Algebra Subroutines

- Level 1 BLAS — vector operations
  - $O(n)$ data and flops (floating point operations)
  - Memory bound:
    $O(1)$ flops per memory access

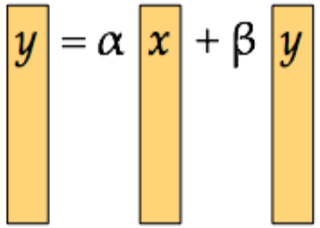


- Level 2 BLAS — matrix-vector operations
  - $O(n^2)$ data and flops
  - Memory bound:
    $O(1)$ flops per memory access

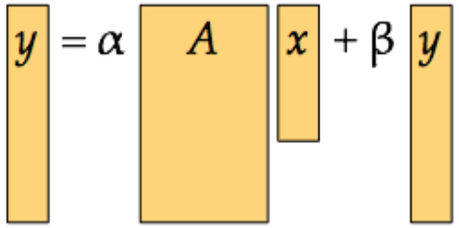


BLAS: Basic Linear Algebra Subroutines

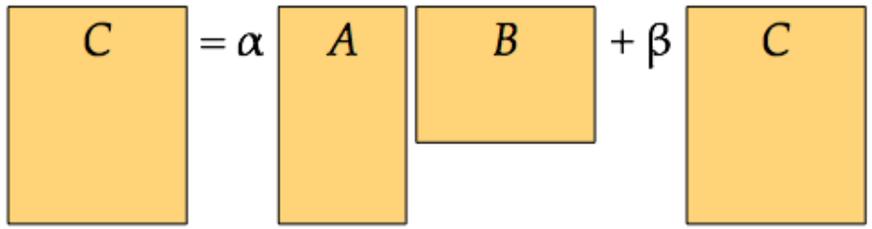
- Level 1 BLAS — vector operations
  - $O(n)$ data and flops (floating point operations)
  - Memory bound:
    $O(1)$ flops per memory access



- Level 2 BLAS — matrix-vector operations
  - $O(n^2)$ data and flops
  - Memory bound:
    $O(1)$ flops per memory access



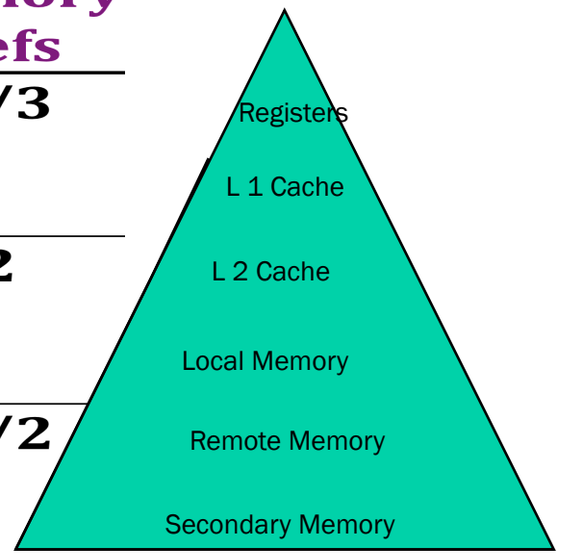
- Level 3 BLAS — matrix-matrix operations
  - $O(n^2)$ data, $O(n^3)$ flops
  - Surface-to-volume effect
  - Compute bound:
    $O(n)$ flops per memory access



Why Higher Level BLAS?

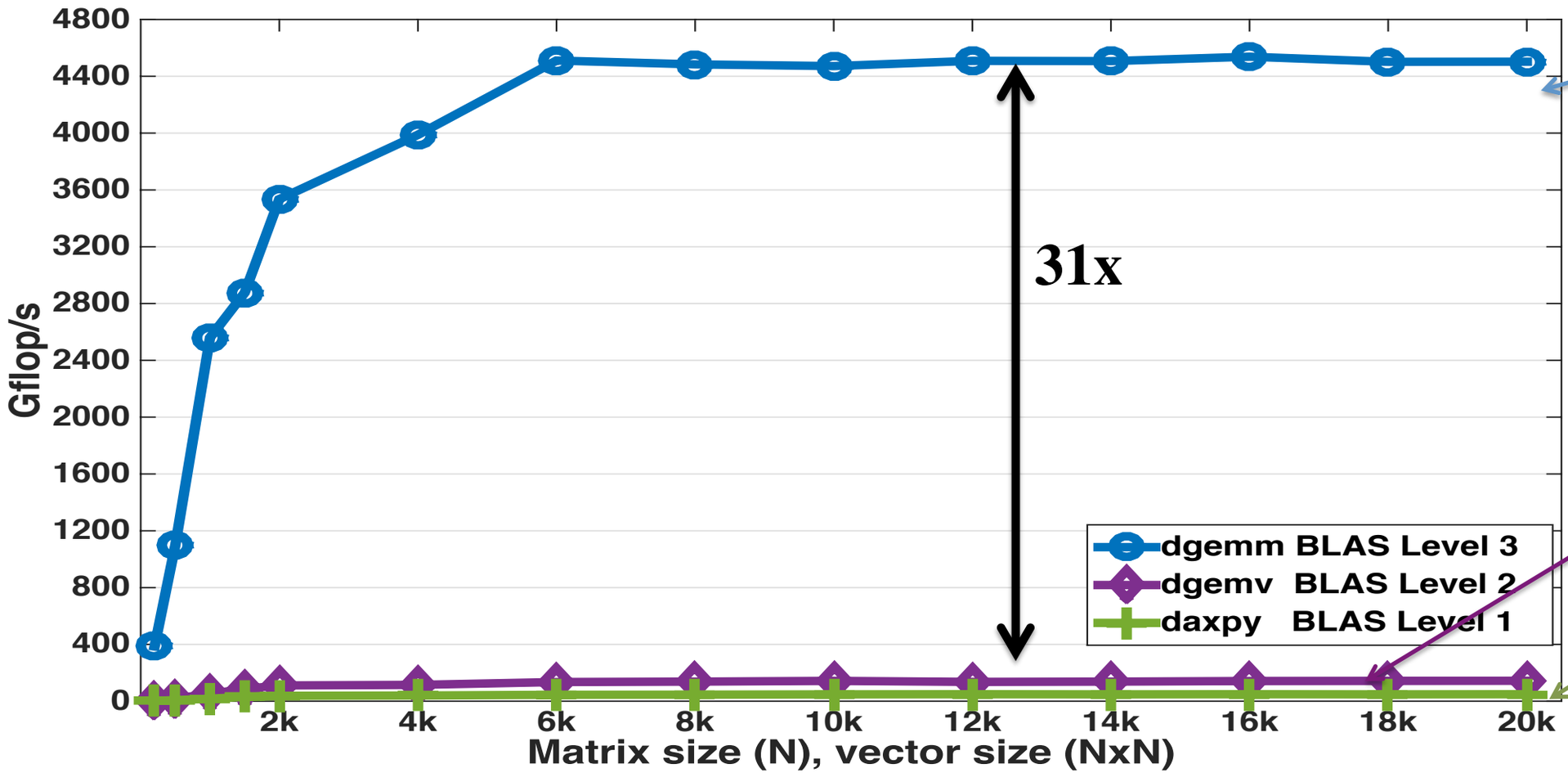
- By taking advantage of the principle of locality:
- Present the user with as much memory as is available in the cheapest technology.
- Provide access at the speed offered by the fastest technology.
- Can only do arithmetic on data at the top of the hierarchy
- Higher level BLAS lets us do this

BLAS	Memory Refs	Flops	Flops/ Memory Refs
Level 1 $y=y+\alpha x$	$3n$	$2n$	$2/3$
Level 2 $y=y+Ax$	n^2	$2n^2$	2
Level 3 $C=C+AB$	$4n^2$	$2n^3$	$n/2$



Level 1, 2 and 3 BLAS

Nvidia **P100**, 1.19 GHz, Peak DP = 4700 Gflop/s



C = C + A*B

4503 Gflop/s

y = y + A*x

145 Gflop/s

y = α *x + y

52 Gflop/s

Nvidia P100
The theoretical peak double precision is 4700 Gflop/s
CUDA version 8.0

ICL

ECP EXASCALE COMPUTING PROJECT

A brief history of (Dense) Linear Algebra software

- **LAPACK – "Linear Algebra PACKage" - uses BLAS-3 (1989 – now)**
  - **Ex: Obvious way to express Gaussian Elimination (GE) is adding multiples of one row to other rows – BLAS-1**
    - How do we reorganize GE to use BLAS-3 ?
  - **Contents of LAPACK (summary)**
    - Algorithms we can turn into (nearly) 100% BLAS 3
      - Linear Systems: solve Ax=b for x
      - Least Squares: choose x to minimize $||Ax - b||_2$
    - Algorithms that are only 50% BLAS 3 (so far)
      - "Eigenproblems": Find λ and x where Ax = λ x
      - Singular Value Decomposition (SVD): $(A^TA)x=\sigma^2x$
    - Generalized problems (e.g., Ax = λ Bx)
    - Error bounds for everything
    - Lots of variants depending on A's structure (banded, $A=A^T$, etc)
  - **How much code? (Release 3.8, Nov 2017) (www.netlib.org/lapack)**
    - Source: 1674 routines, 490K LOC, Testing: 448K LOC

A brief history of (Dense) Linear Algebra software

- **Is LAPACK parallel?**
  - Only if the BLAS are parallel (possible in shared memory)
- **ScaLAPACK – "Scalable LAPACK" (1995 – now)**
  - For distributed memory – uses MPI
  - More complex data structures, algorithms than LAPACK
    - Only (small) subset of LAPACK's functionality available
  - All at www.netlib.org/scalapack

LAPACK

- http://www.netlib.org/lapack/

- LAPACK (Linear Algebra Package) provides routines for
  - solving systems of simultaneous linear equations,
  - least-squares solutions of linear systems of equations,
  - eigenvalue problems,
  - and singular value problems.

- LAPACK relies on BLAS

- The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers.

- Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

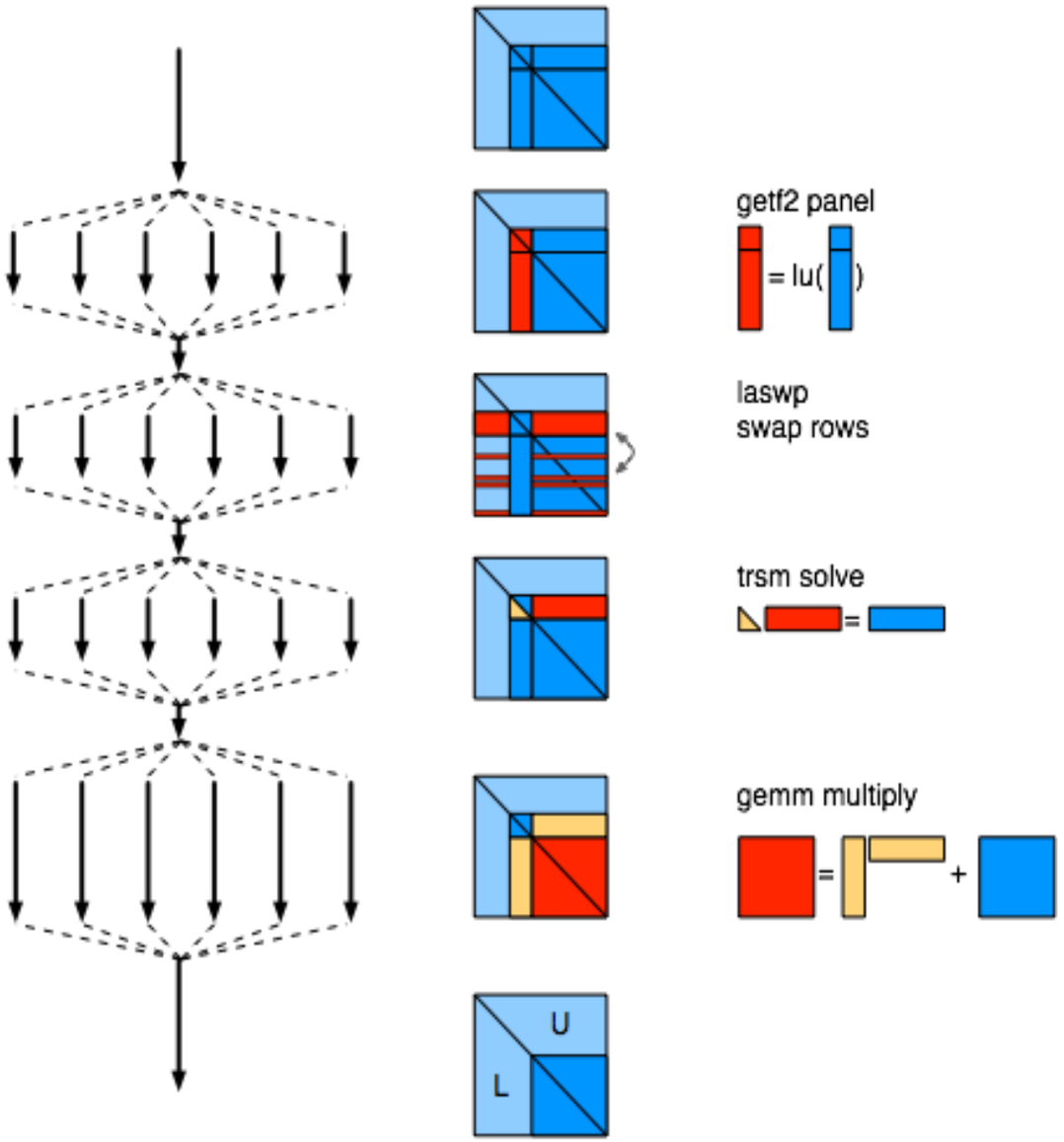
LAPACK is in FORTRAN Column Major

LAPACK is SEQUENTIAL

LAPACK is a REFERENCE implementation

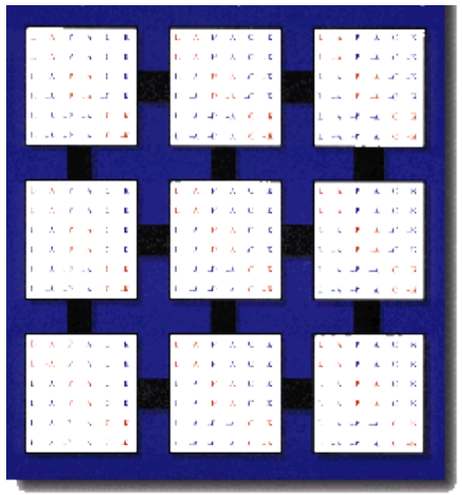
Parallelism in LAPACK

- Most flops in gemm update
  - 2/3 n^3 term
  - Easily parallelized using multi-threaded BLAS
  - Done in any reasonable software
- Other operations lower order
  - Potentially expensive if not parallelized



Overview of Dense Numerical Linear Algebra Libraries

- **BLAS**: kernel for dense linear algebra
- **LAPACK**: sequential dense linear algebra
- **ScaLAPACK**: parallel distributed dense linear algebra



Scalable **L**inear **A**lgebra **PACK**age

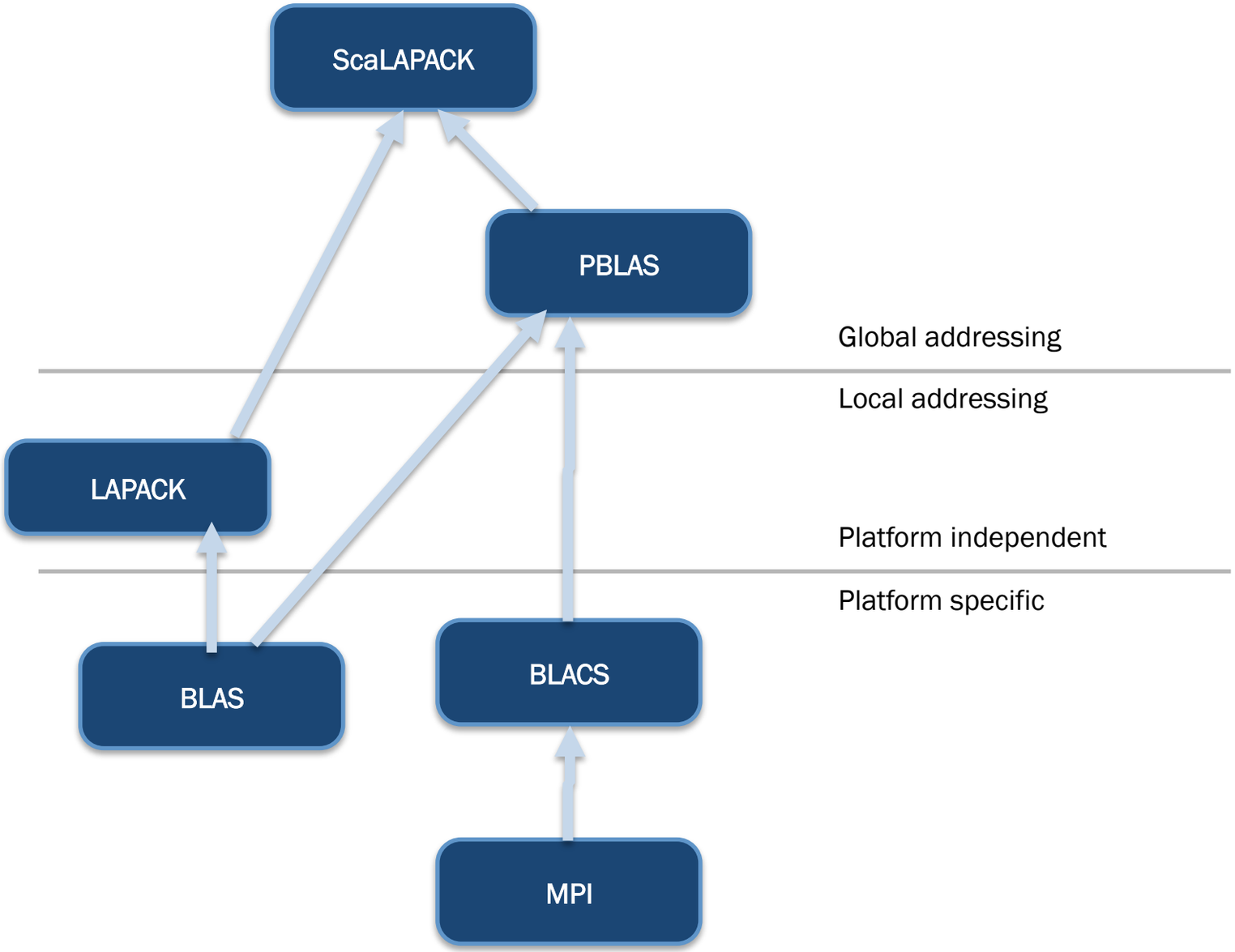
PBLAS

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

- LAPACK: dge___(m, n, A(ia, ja), lda, ...)
  - Submatrix offsets implicit in pointer

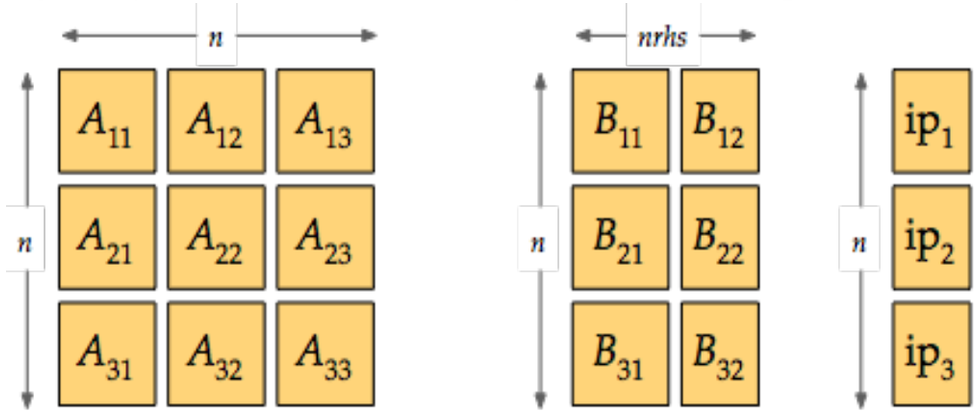
- ScaLAPACK: pdge___(m, n, A, ia, ja, descA, ...)
  - Pass submatrix offsets and matrix descriptor

ScaLAPACK structure



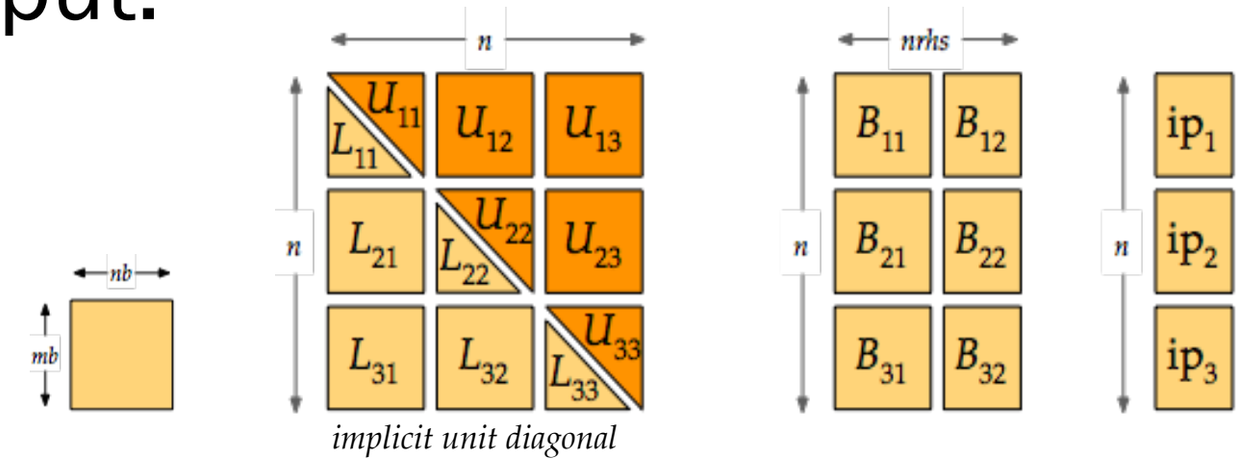
ScaLAPACK routine, solve *AX = B*

- LAPACK: `dgesv(n, nrhs, A, lda, ipiv, B, ldb, info)`
- ScaLAPACK: `pdgesv(n, nrhs, A, ia, ja, descA, ipiv, B, ib, jb, descB, info)`
- input:



Global matrix point of view

- output:



implicit unit diagonal

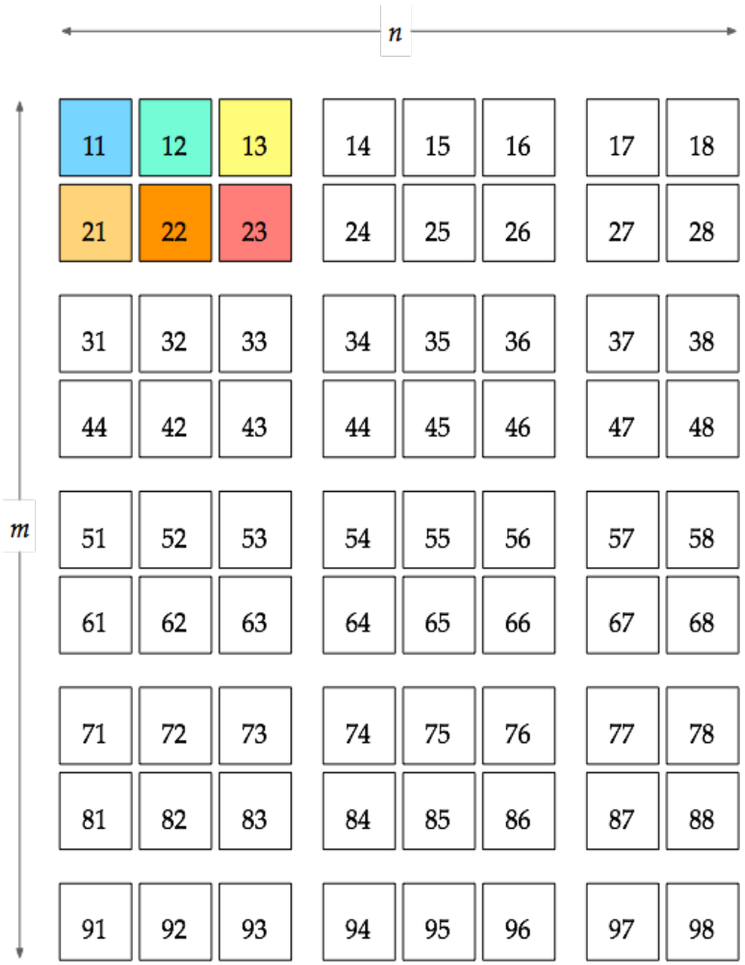
info (error code)
- = 0: no error
- < 0: invalid argument
- > 0: numerical error (e.g., singular)

L, U overwrite A
X overwrites B

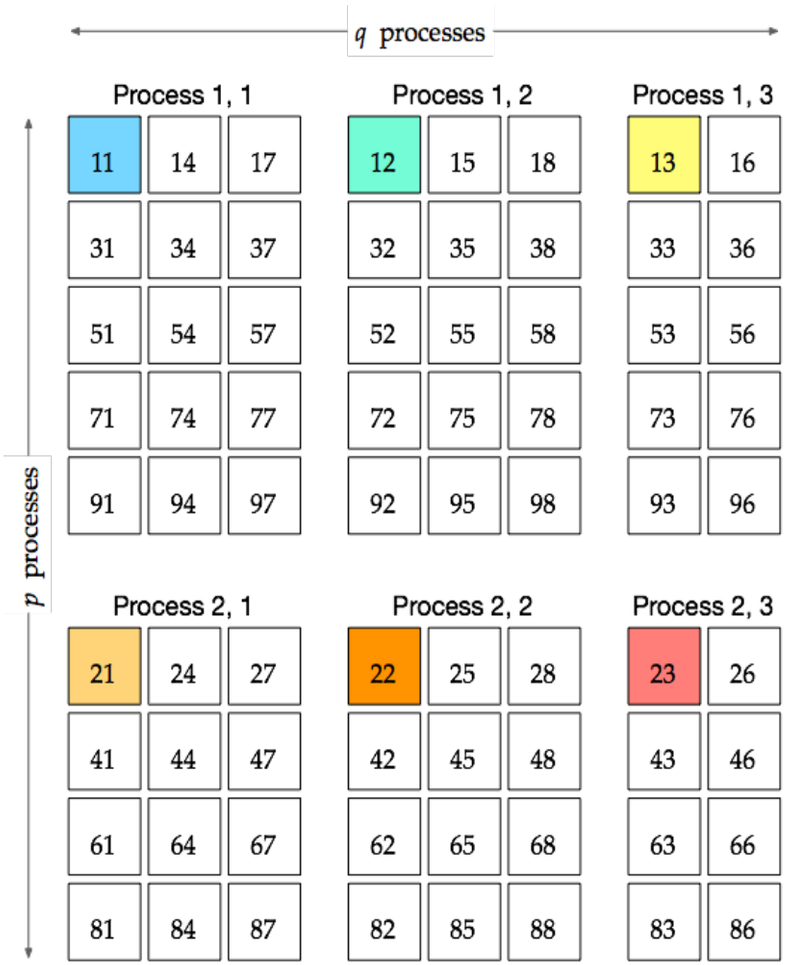
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



Local process point of view



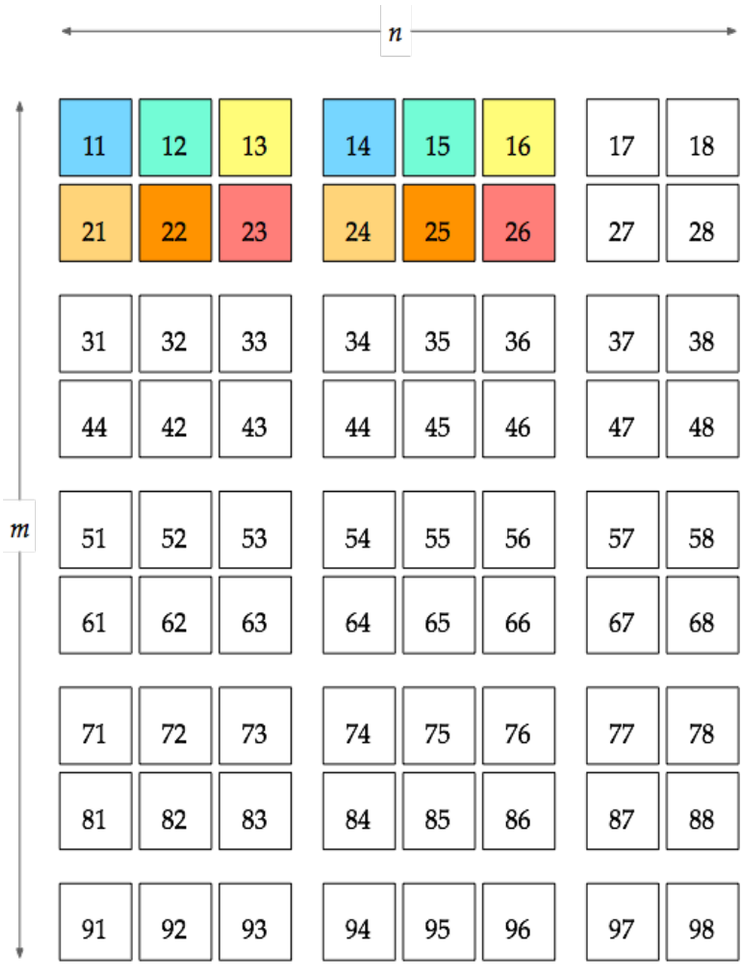
ICL

ECP EXASCALE COMPUTING PROJECT

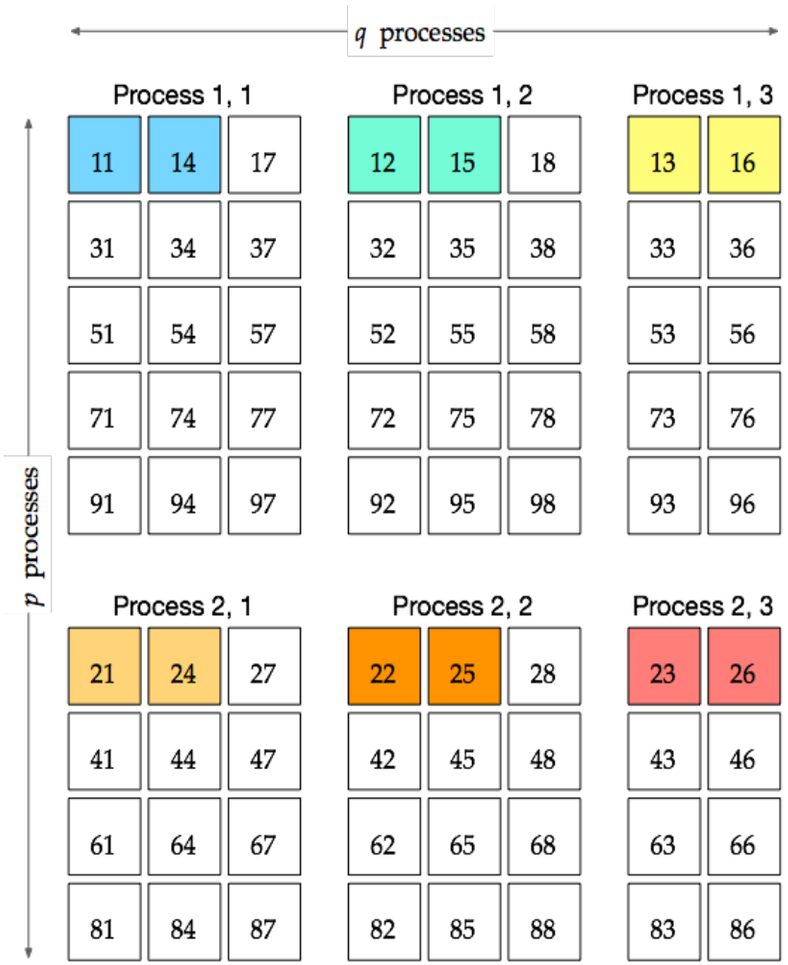
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



Local process point of view



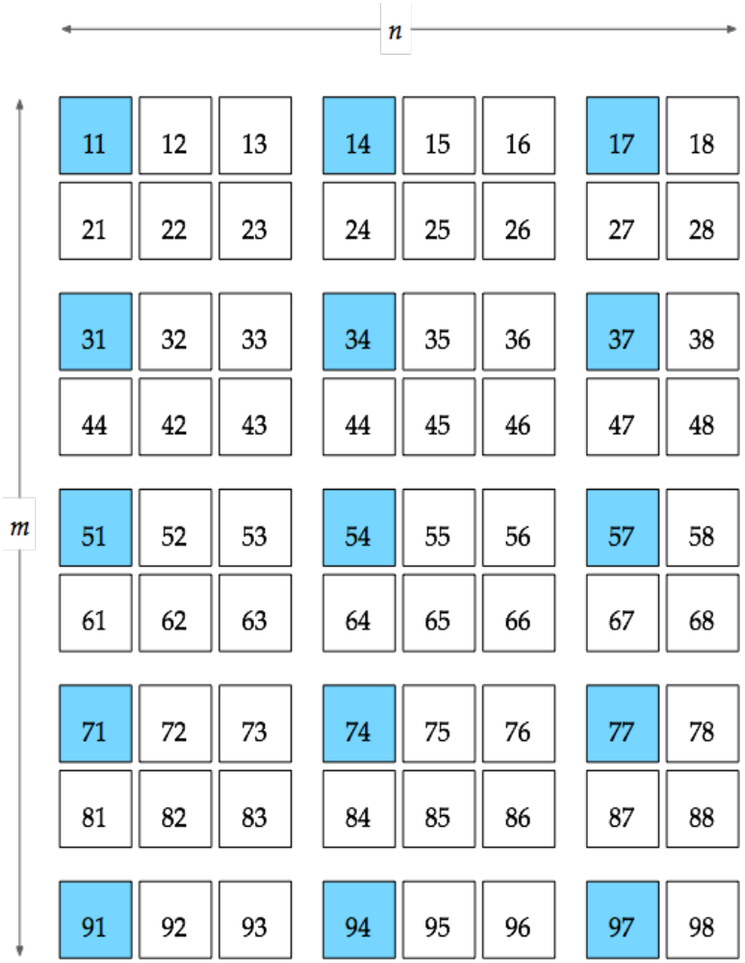
ICL

ECP EXASCALE COMPUTING PROJECT

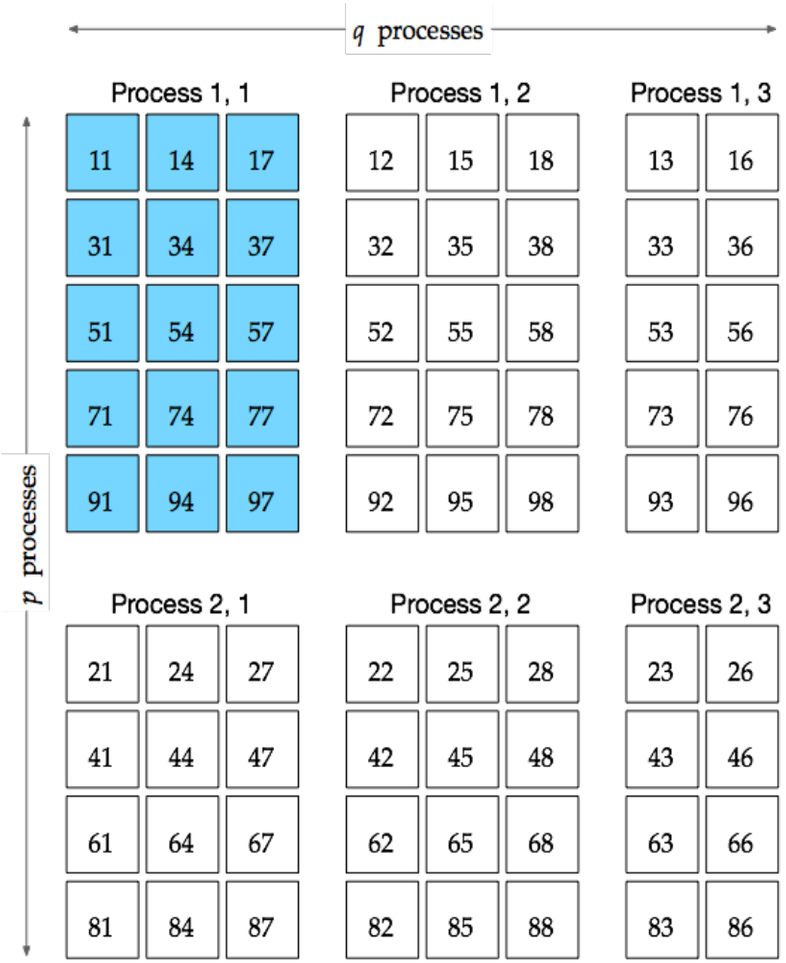
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



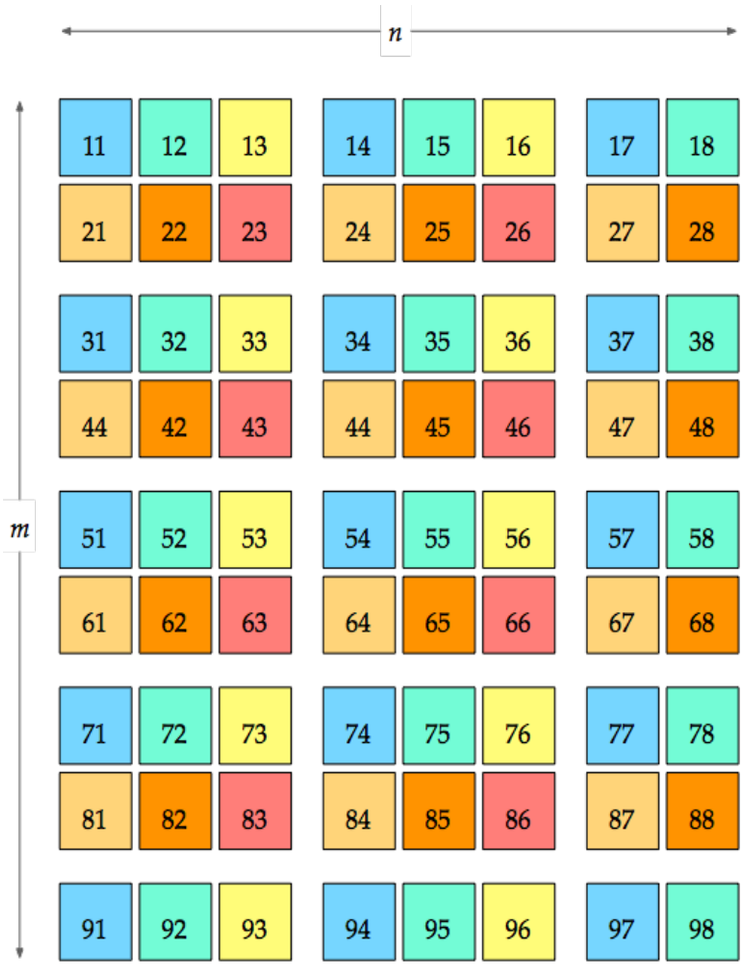
Local process point of view



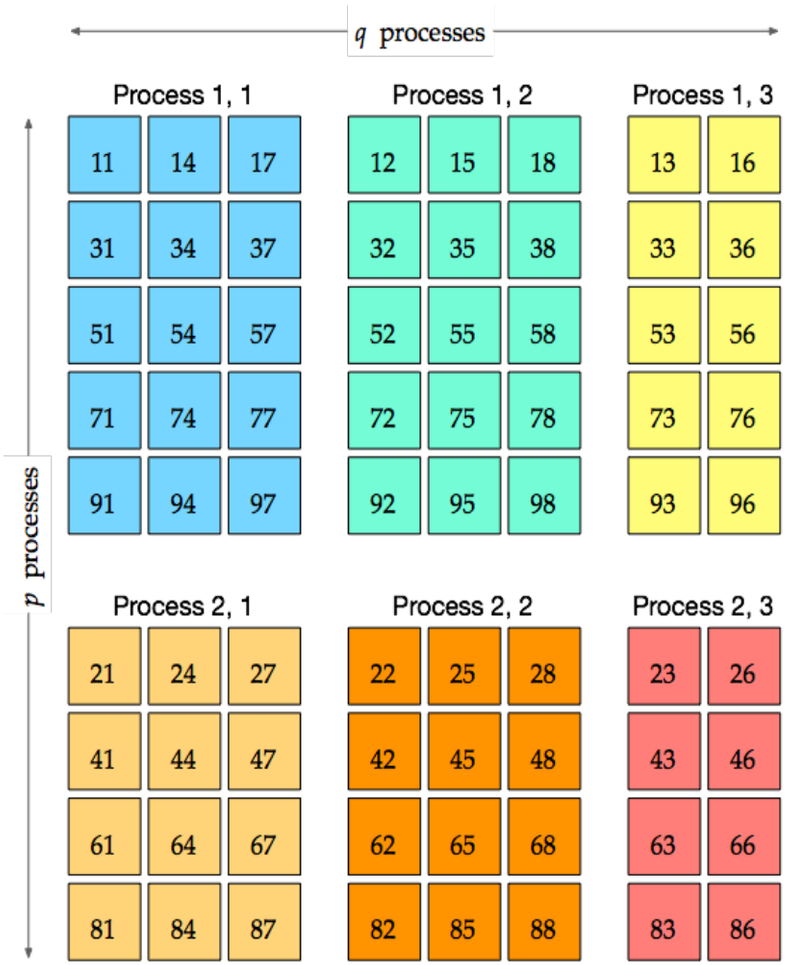
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



Local process point of view



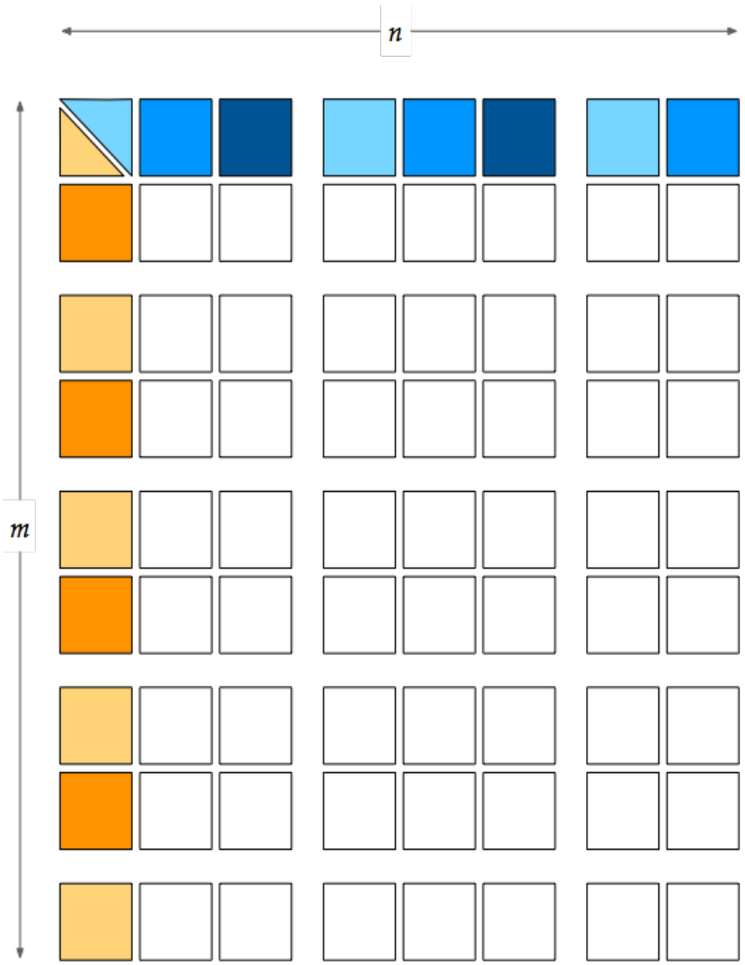
ICL

ECP EXASCALE COMPUTING PROJECT

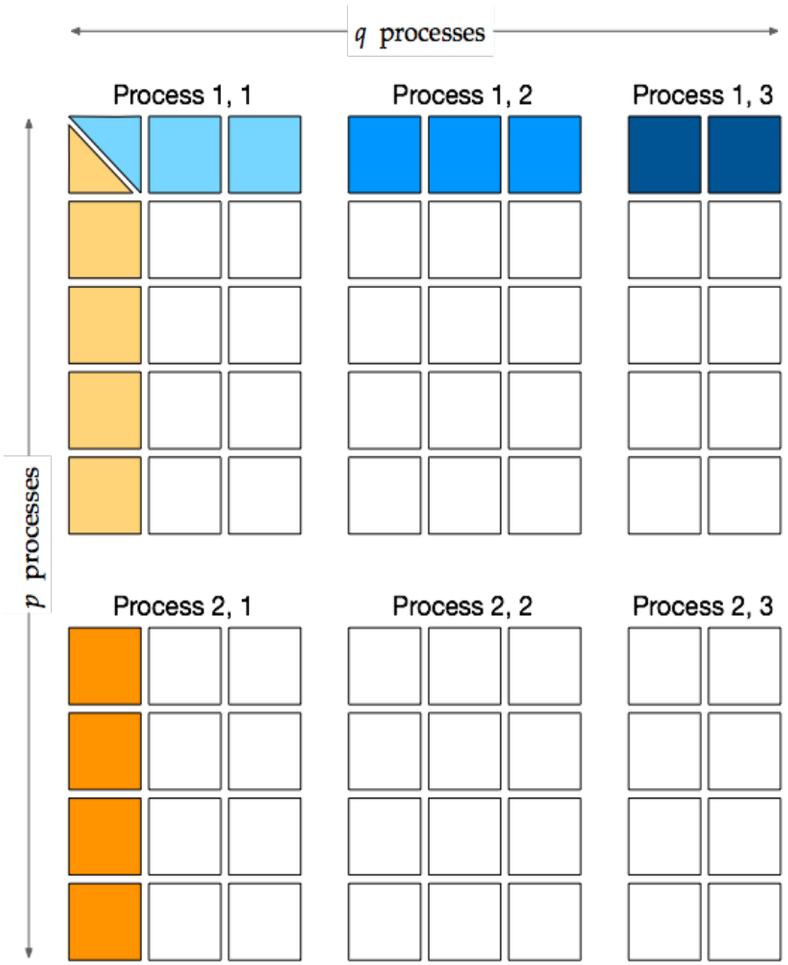
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



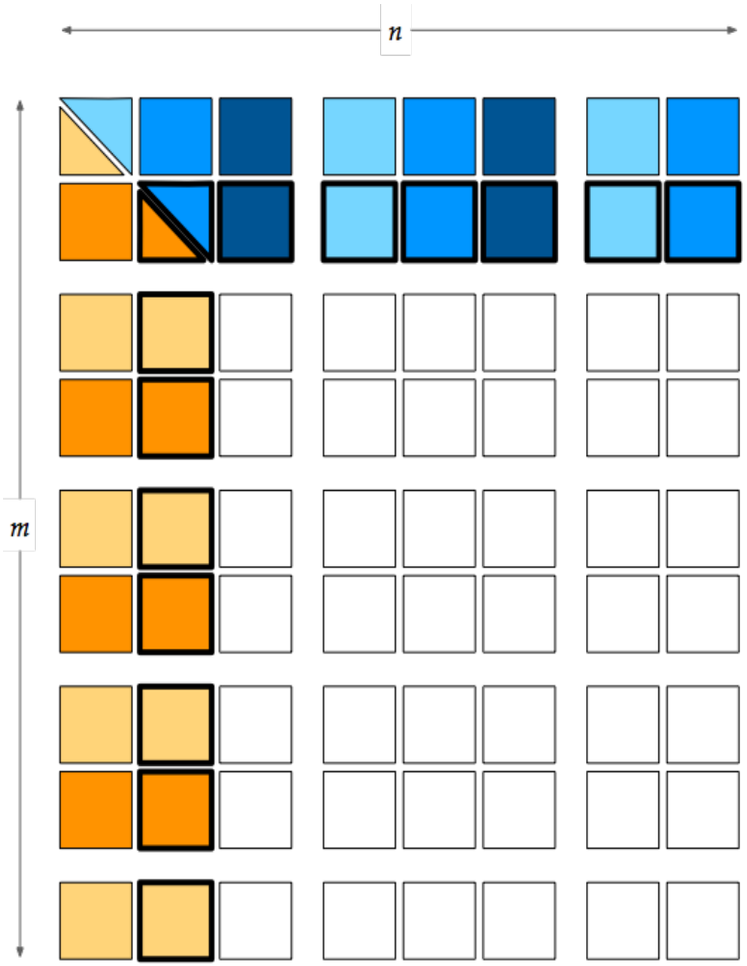
Local process point of view



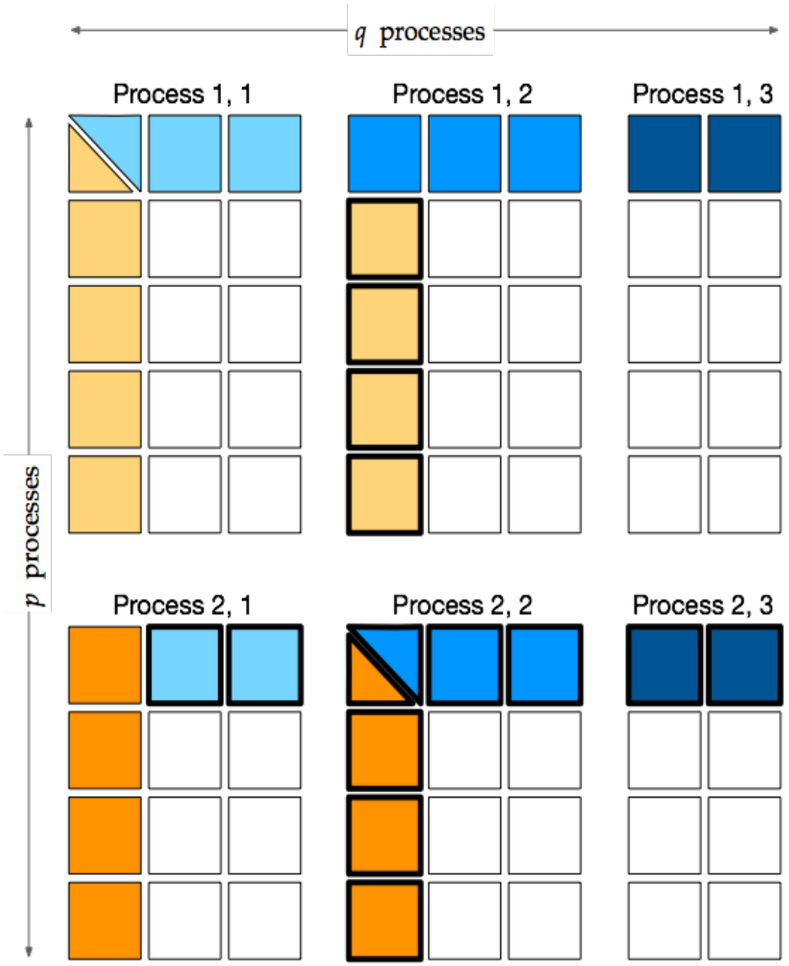
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



Local process point of view



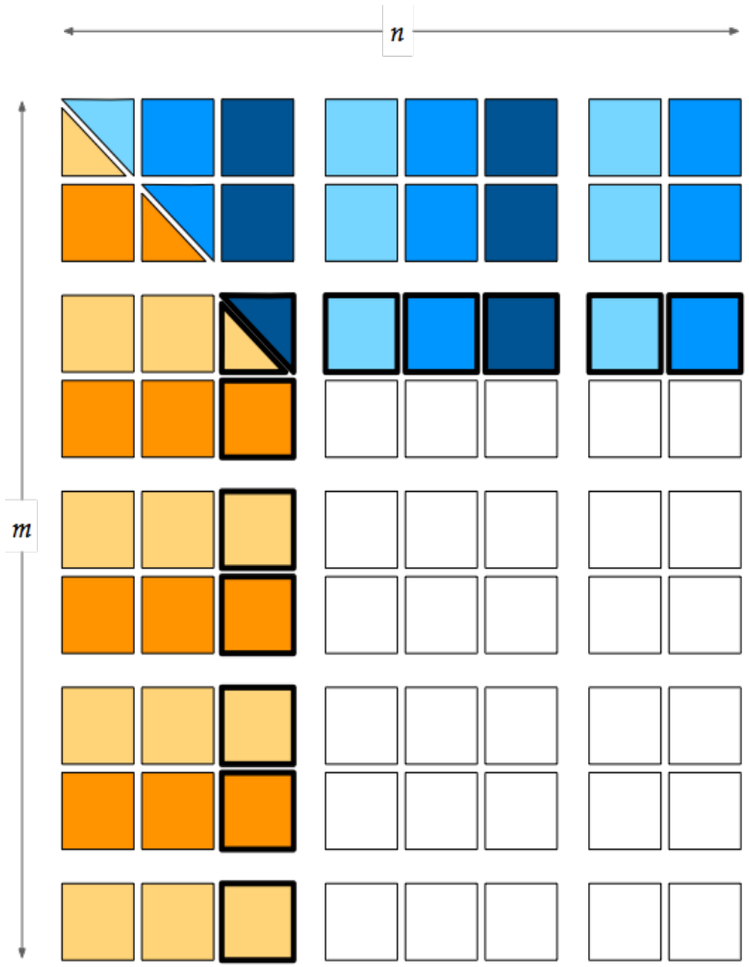
ICL

ECP EXASCALE COMPUTING PROJECT

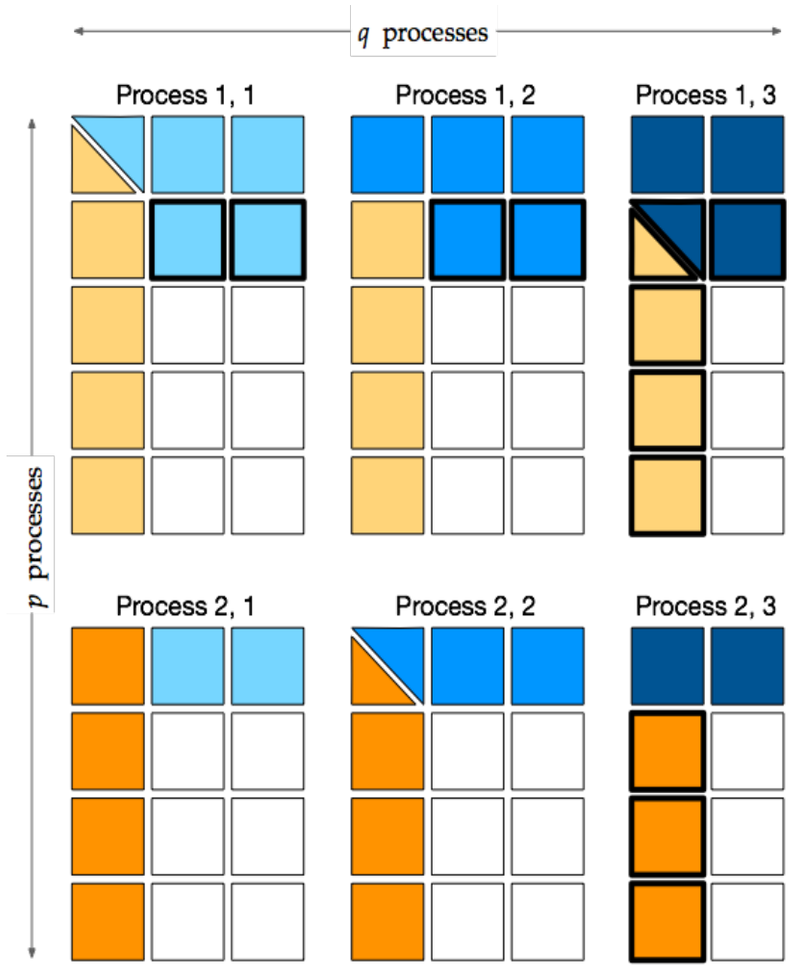
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



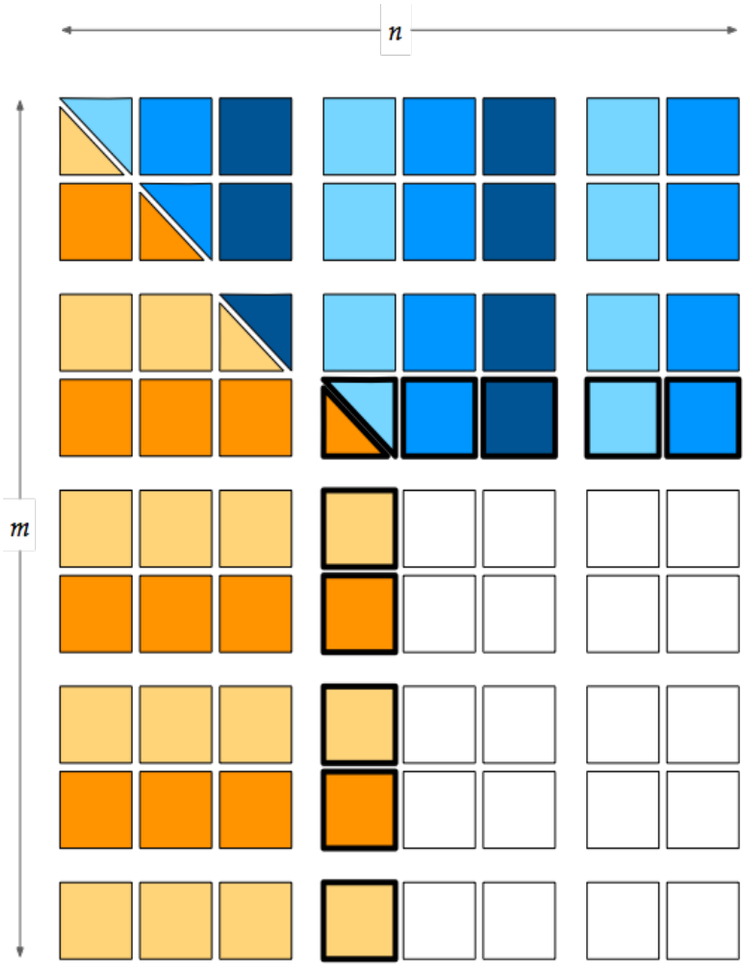
Local process point of view



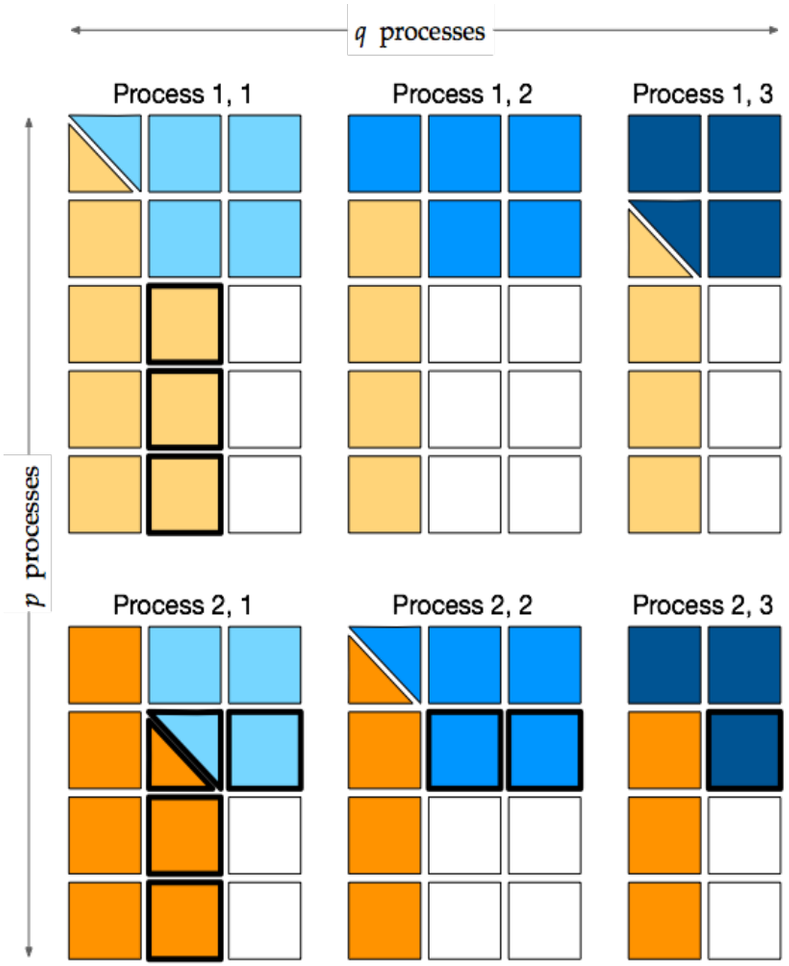
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



Local process point of view



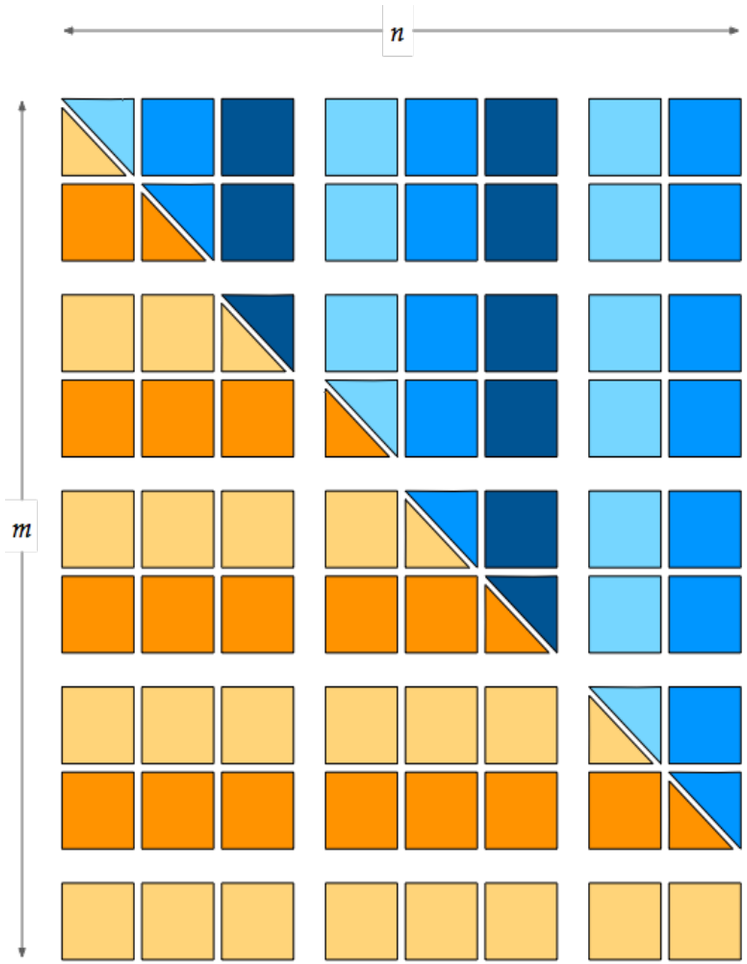
ICL

ECP EXASCALE COMPUTING PROJECT

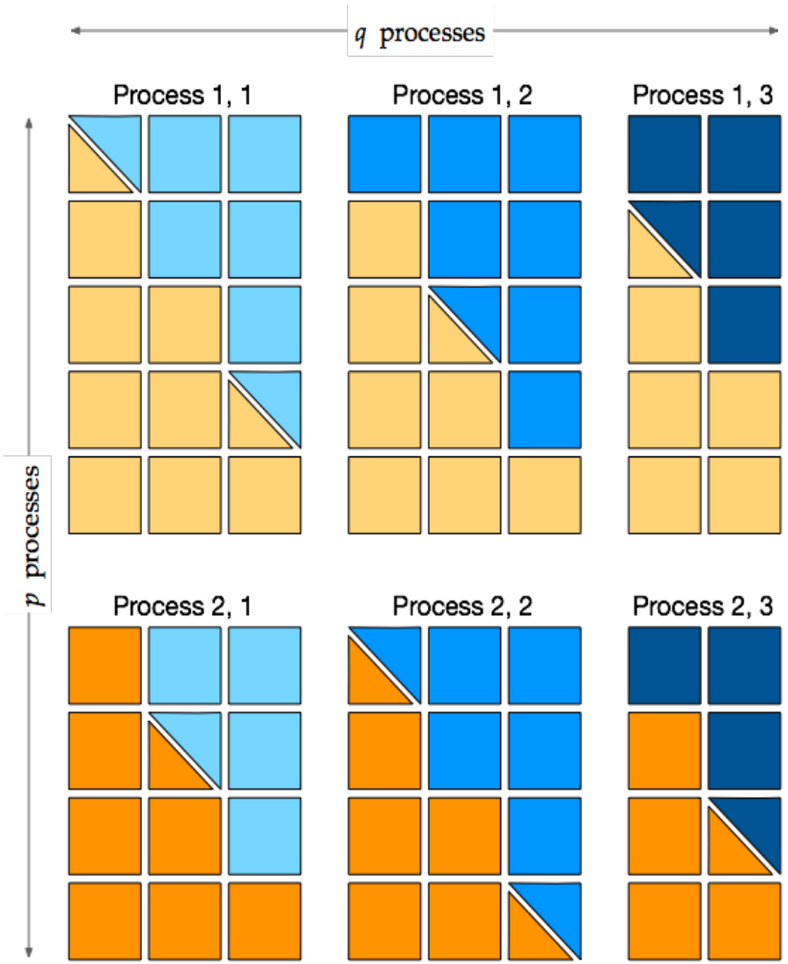
2D block-cyclic layout

$m \times n$ matrix
$p \times q$ process grid

Global matrix view



Local process point of view

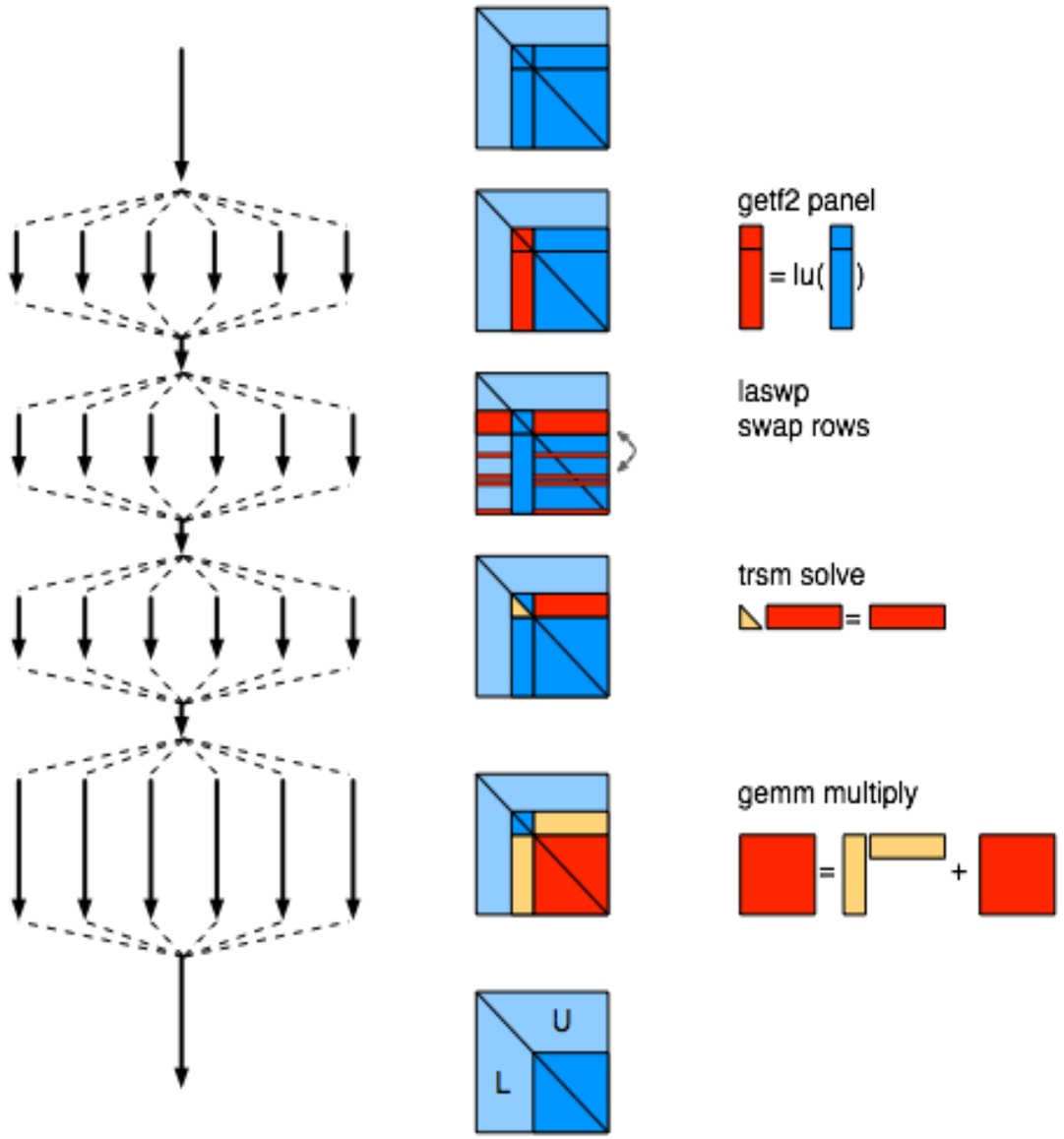


ICL

ECP EXASCALE COMPUTING PROJECT

Parallelism in ScaLAPACK

- Similar to LAPACK
- Bulk-synchronous
- Most flops in gemm update
  - 2/3 n^3 term
  - Can use **sequential BLAS**,
    p x q = # cores
    = # MPI processes,
    num_threads = 1
  - Or **multi-threaded BLAS**,
    p x q = # nodes
    = # MPI processes,
    num_threads = # cores/node



Major Changes to Software

- **Must rethink the design of our software**
  - Another disruptive technology
    - Similar to what happened with cluster computing and message passing
  - Rethink and rewrite the applications, algorithms, and software
- **Numerical libraries for example are changing**
  - For example, both LAPACK and ScaLAPACK undergo major changes to accommodate this

Software Projects

netlib.org

- LAPACK
- ScaLAPACK
- BLAS
- CBLAS
- LAPACKE

icl.utk.edu/research

- PLASMA — dense linear algebra (multicore)
- MAGMA — dense linear algebra (accelerators)
- SLATE — dense linear algebra (distributed memory / muticore / accelerators)

new software for multicore and accelerators

Software Projects

netlib.org

LAPACK

ScaLAPACK

BLAS

CBLAS

LAPACKE

icl.utk.edu/research

PLASMA

MAGMA

SLATE

QUARK

scheduling
(multicore)

PaRSEC

scheduling
(distributed memory)

**dynamic
runtime
schedulers**

Software Projects

netlib.org

LAPACK

ScaLAPACK

BLAS

CBLAS

LAPACKE

icl.utk.edu/research

PLASMA

MAGMA

SLATE

OpenMP

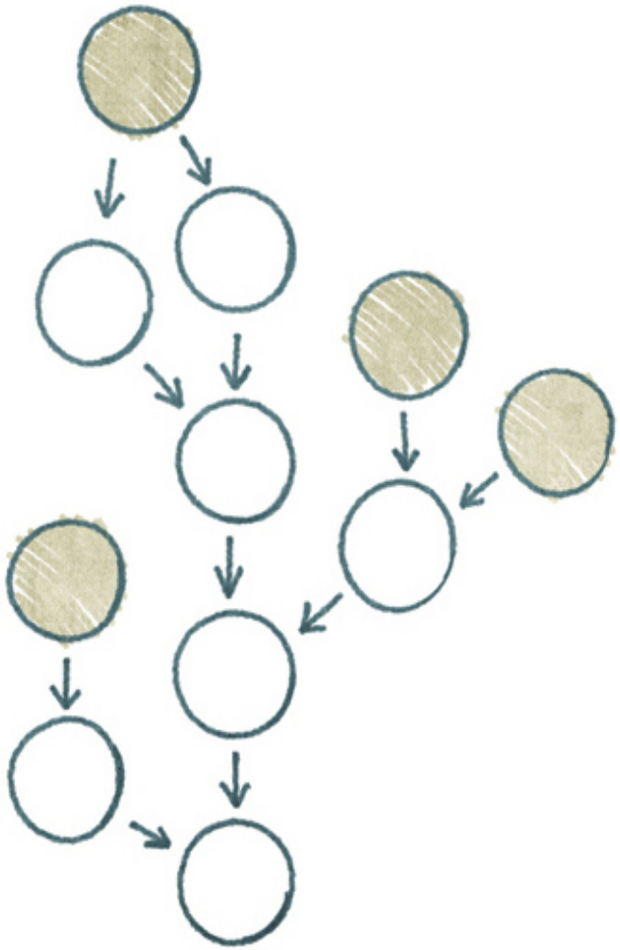
scheduling
(multicore)

PaRSEC

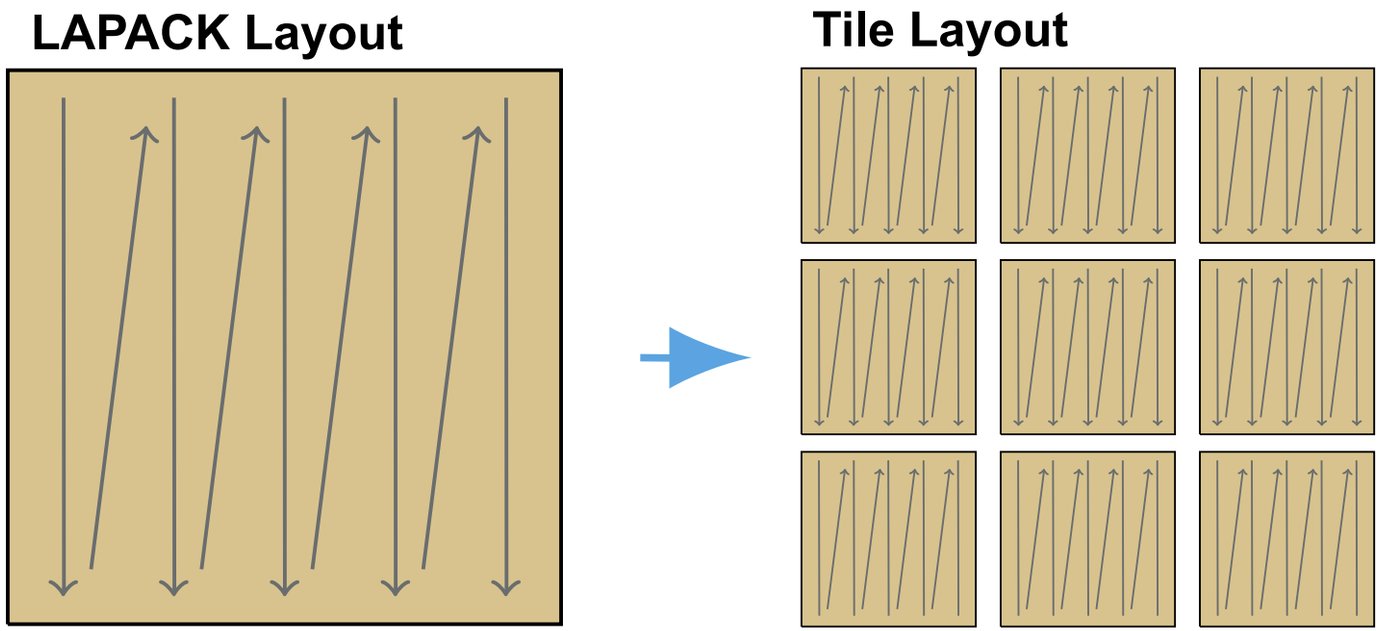
scheduling
(distributed memory)

**dynamic
runtime
schedulers**

PLASMA



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



Programming with Quark tasking

```
#include <quark.h>

int main(int argc , char** argv) {
    Quark * quark = QUARK_New( nthreads );
    ...
    for (int m = 1; m <= 8; m++) {
        for (int n = 1; n <= 7; n++) {
            dgemm_tile_quark( quark, NULL,
                              CblasColMajor, CblasNoTrans, CblasNoTrans,
                              nb, nb, nb, −1.0,
                              A(m, 0), nb, A(0, n), nb, 1.0, A(m, n), nb);
        }
    }
    ...
    QUARK_Delete( quark );
}
```

```
void dgemm_tile_quark(Quark* quark, Quark_Task_Flags * task_flags,
    enum CBLAS_ORDER order, enum CBLAS_TRANSPOSE transa,
    enum CBLAS_TRANSPOSE transb, enum CBLAS_TRANSPOSE transc,
    int m, int n, int k, double alpha, double *A, int lda, double *B, int ldb,
    double beta, double *C, int ldc) {

    QUARK_Insert_Task( quark, dgemm_tile_task, task_flags,
          sizeof(enum CBLAS_ORDER),          &order ,   VALUE,
          sizeof(enum CBLAS_TRANSPOSE),      &transa,   VALUE,
          sizeof(enum CBLAS_TRANSPOSE),      &transb,   VALUE,
          sizeof(enum CBLAS_TRANSPOSE),      &transc,   VALUE,
          sizeof(int),                       &m     ,   VALUE,
          sizeof(int),                       &n     ,   VALUE,
          sizeof(int),                       &k     ,   VALUE,
          sizeof(double),                    &alpha ,   VALUE,
          sizeof(double *),                  &A     ,   INPUT,
          sizeof(int),                       &lda   ,   VALUE,
          sizeof(double *),                  &B     ,   INPUT,
          sizeof(int),                       &ldb   ,   VALUE,
          sizeof(double),                    &beta  ,   VALUE,
          sizeof(double *),                  &C     ,   INOUT,
          sizeof(int),                       &ldc   ,   VALUE,   0 );
}
```

```
void dgemm_tile_task( Quark* quark ) {

    enum CBLAS_ORDER order;
    enum CBLAS_TRANSPOSE transa, transb, transc;
    int m, n, k;
    double alpha, beta, *A, *B, *C;

    quark_unpack_args_15(quark, order, transa, transb, transc,
                         m, n, k,
                         alpha, A, lda,
                                B, ldb,
                         beta  , C, ldc );

    cblas_dgemm(order, transa, transb, transc,
                m, n, k,
                alpha, A, lda,
                       B, ldb,
                beta,  C, ldc );
}
```

Programming with OpenMP4 tasking

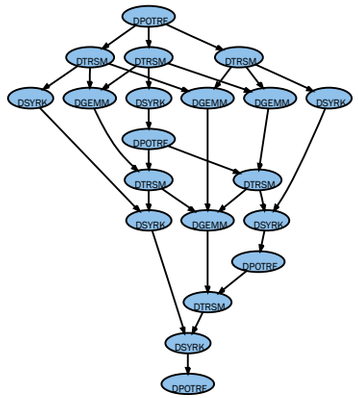
```
#include <quark.h>

int main(int argc , char** argv) {
    Quark * quark = QUARK_New( nthreads );
    ...
    for (int m = 1; m <= 8; m++) {
        for (int n = 1; n <= 7; n++) {
            dgemm_tile_quark( quark, NULL,
                              CblasColMajor, CblasNoTrans, CblasNoTrans,
                              nb, nb, nb, −1.0,
                              A(m, 0), nb, A(0, n), nb, 1.0, A(m, n), nb);
        }
    }
    ...
    QUARK_Delete( quark );
}
```

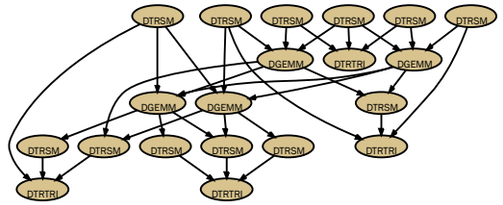
```
void dgemm_tile_quark(Quark* quark, Quark_Task_Flags * task_flags,
    enum CBLAS_ORDER order, enum CBLAS_TRANSPOSE transa,
    enum CBLAS_TRANSPOSE transb, enum CBLAS_TRANSPOSE transc,
    int m, int n, int k, double alpha, double *A, int lda, double *B, int ldb,
    double beta, double *C, int ldc) {

    QUARK_Insert_Task( quark, dgemm_tile_task, task_flags,
          sizeof(enum CBLAS_ORDER),          &order ,   VALUE,
          sizeof(enum CBLAS_TRANSPOSE),      &transa,   VALUE,
          sizeof(enum CBLAS_TRANSPOSE),      &transb,   VALUE,
          sizeof(enum CBLAS_TRANSPOSE),      &transc,   VALUE,
          sizeof(int),                       &m     ,   VALUE,
          sizeof(int),                       &n     ,   VALUE,
          sizeof(int),                       &k     ,   VALUE,
          sizeof(double),                    &alpha ,   VALUE,
          sizeof(double *),                  &A     ,   INPUT,
          sizeof(int),                       &lda   ,   VALUE,
          sizeof(double *),                  &B     ,   INPUT,
          sizeof(int),                       &ldb   ,   VALUE,
          sizeof(double),                    &beta  ,   VALUE,
          sizeof(double *),                  &C     ,   INOUT,
          sizeof(int),                       &ldc   ,   VALUE,   0 );
}
```

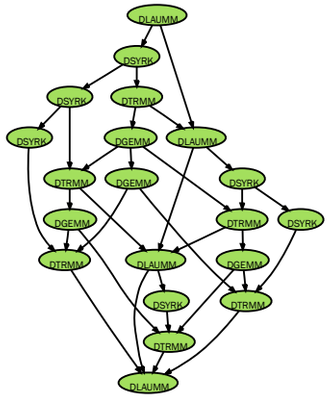
```
#include <omp.h>

int main(int argc , char** argv) {

    #pragma omp parallel
    #pragma omp master
    {
        ...
        for (int m = 1; m <= 8; m++)
            for (int n = 1; n <= 7; n++) {
                #pragma omp task depend(     in:A(m,0)[0:nb*nb] )  \
                                 depend(     in:A(0, n)[0:nb*nb])  \
                                 depend( inout:A(m,n)[0:nb*nb])
                cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans,
                            nb, nb, nb, −1.0,
                            A(m, 0), nb, A(0, n), nb, 1.0, A(m, n), nb);
            }
        ...
    }
}
```

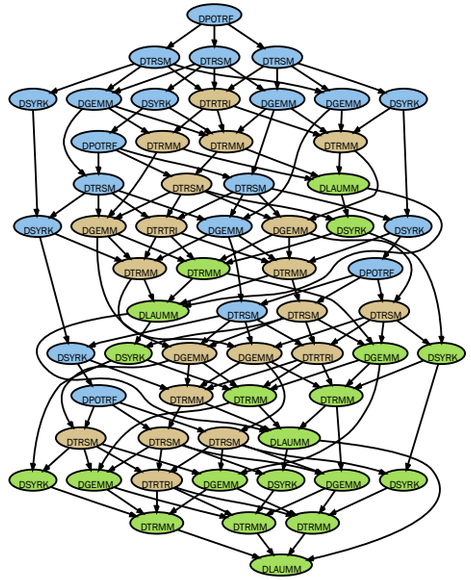
```
void dgemm_tile_task( Quark* quark ) {

    enum CBLAS_ORDER order;
    enum CBLAS_TRANSPOSE transa, transb, transc;
    int m, n, k;
    double alpha, beta, *A, *B, *C;

   quark_unpack_args_15(quark, order, transa, transb, transc,
                        m, n, k,
                        alpha, A, lda,
                               B, ldb,
                        beta  , C, ldc );

    cblas_dgemm(order, transa, transb, transc,
                m, n, k,
                alpha, A, lda,
                       B, ldb,
                beta,  C, ldc );
}
```

```
#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);
            }
        }
    }
}
```



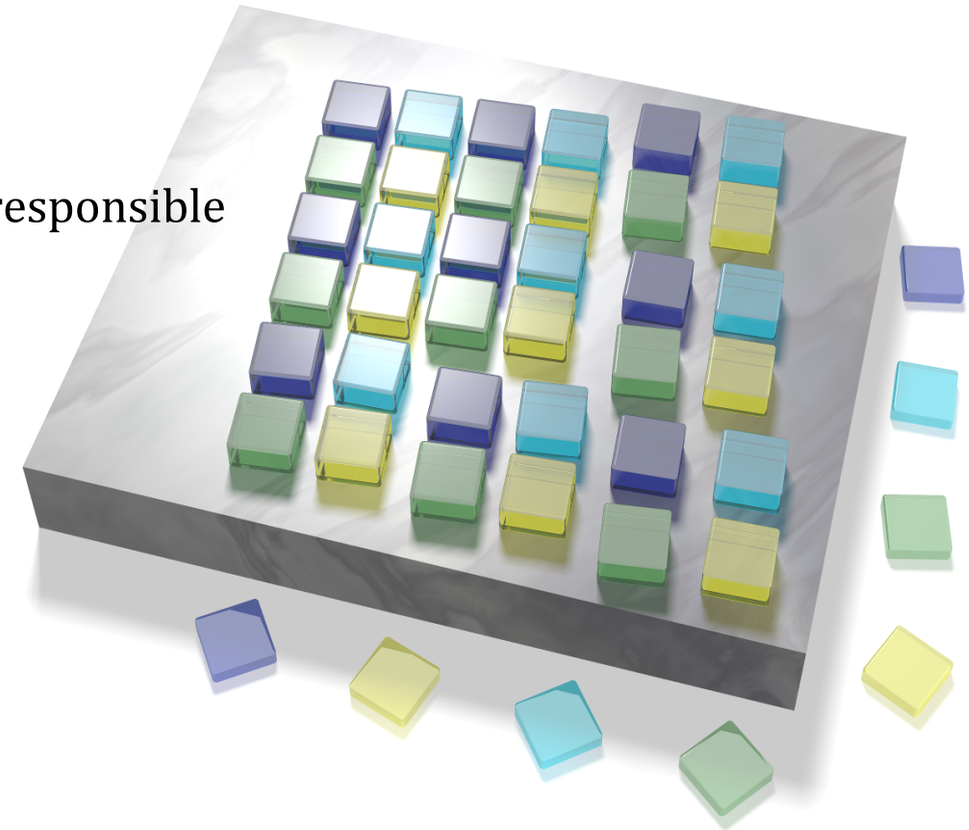






SLATE
Software for Linear Algebra Targeting Exascale

This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy's Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation's exascale computing imperative.



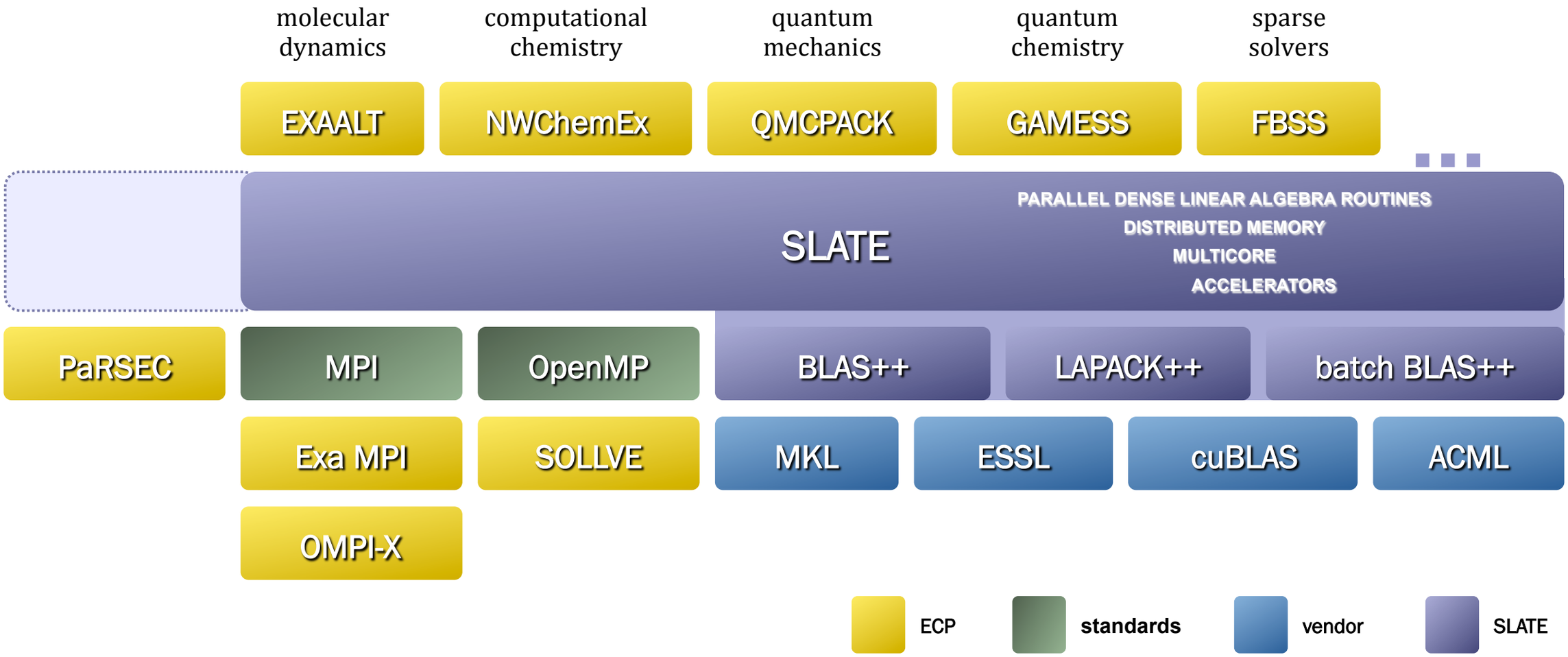
SLATE Objectives

- **Coverage** — ScaLAPACK and beyond
- **Modern Hardware** — DOE CORAL (pre Exascale) → DOE Exascale
- **Portability** — Intel Xeon (&Phi), IBM POWER, ARM, NVIDIA, AMD, …
- **Modern Language** — C++11/14/17 (templates, STL, overloading, polymorphism, …)
- **Modern Standards** — MPI 3, OpenMP 4/5 (&omp target)
- **Performance** — 80-90% of peak (asymptotic)
- **Scalability** — full machine (tens of thousands of nodes)
- **Productivity** — ca. 4 full time developers
- **Maintainability** — part time developers + community

can be built:

- serial
- OpenMP multithreading
- MPI message passing
- GPU acceleration

SLATE Stack


molecular dynamics
computational chemistry
quantum mechanics
quantum chemistry
sparse solvers
EXAALT
NWChemEx
QMCPACK
GAMESS
FBSS
SLATE
PARALLEL DENSE LINEAR ALGEBRA ROUTINES
DISTRIBUTED MEMORY
MULTICORE
ACCELERATORS
PaRSEC
MPI
OpenMP
BLAS++
LAPACK++
batch BLAS++
Exa MPI
SOLLVE
MKL
ESSL
cuBLAS
ACML
OMPI-X
ECP
standards
vendor
SLATE

SLATE Resources

- main ECP website: https://exascaleproject.org
- main SLATE website: http://icl.utk.edu/slate/
- main SLATE repository: https://bitbucket.org/icl/slate
- BLAS++ repository: https://bitbucket.org/icl/blaspp
- LAPACK++ repository: https://bitbucket.org/icl/lapackpp
- SLATE Working Notes: http://www.icl.utk.edu/publications/series/swans
- Research Gate project: https://www.researchgate.net/project/ECP-SLATE
- SLATE User https://groups.google.com/a/icl.utk.edu/forum/#!forum/slate-user

SLATE Working Notes

http://www.icl.utk.edu/publications/series/swans

- **Designing SLATE: Software for Linear Algebra Targeting Exascale**

http://www.icl.utk.edu/publications/swan-003

- **C++ API for BLAS and LAPACK**

http://www.icl.utk.edu/publications/swan-002

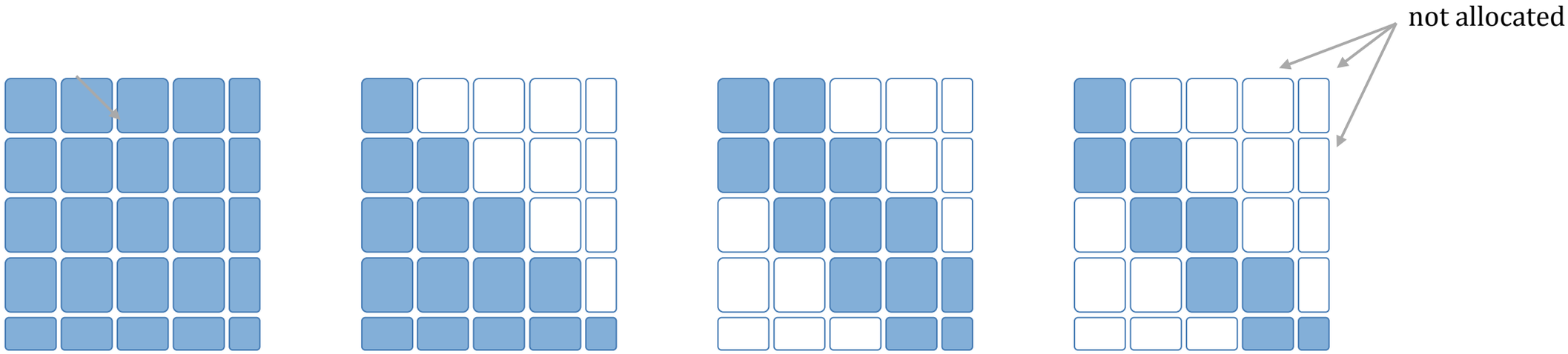
https://bitbucket.org/icl/blaspp

https://bitbucket.org/icl/lapackpp

- **Roadmap for the Development of a Linear Algebra Library for Exascale Computing: SLATE: Software for Linear Algebra Targeting Exascale**

http://www.icl.utk.edu/publications/swan-001

SLATE Matrix



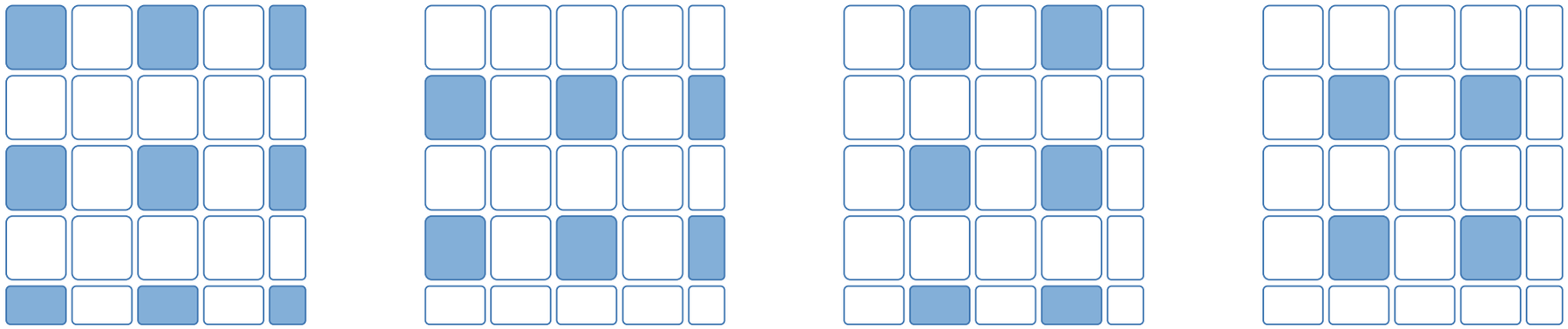

```
std::map<std::tuple<int64_t, int64_t, int>, Tile<FloatType>*> *tiles_;
```

- collection of tiles
- **individually allocated**
- only allocate what is needed
- accommodates: symmetric, triangular, band, …

While in the PLASMA library the matrix is also stored in tiles, the tiles are laid out contiguously in memory.

In contrast, in SLATE, the tiles are individually allocated, with no correlation of their locations in the matrix to their addresses in memory.

SLATE Distributed Matrix



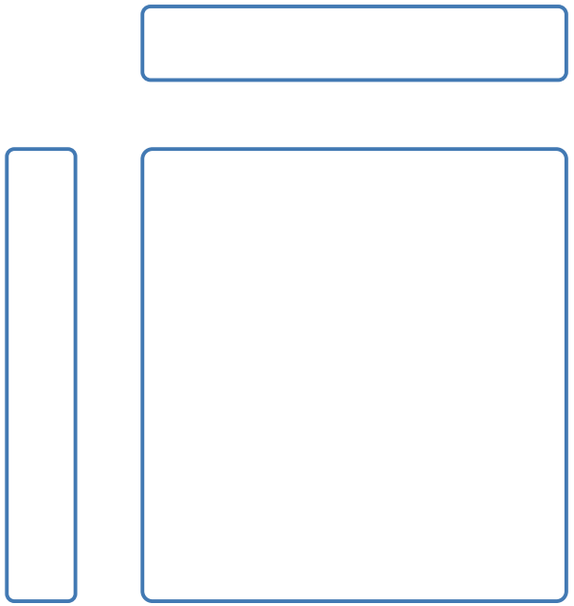
```
std::map<std::tuple<int64_t, int64_t, int>, Tile<FloatType>*> *tiles_;
```

- distributed matrix
- global indexing of tiles
- only allocate the local part
- any distribution is possible (2D block cyclic by default)

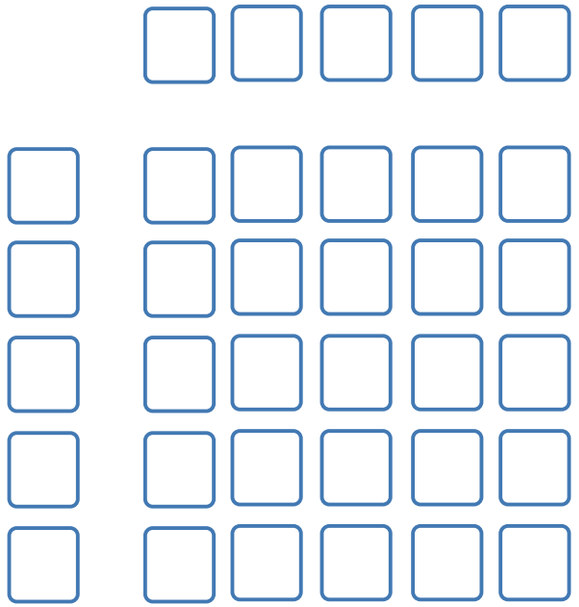
The same structure, used for single node representation, naturally supports distributed memory representation.

GEMM Efficiency

LAPACK
MAGMA

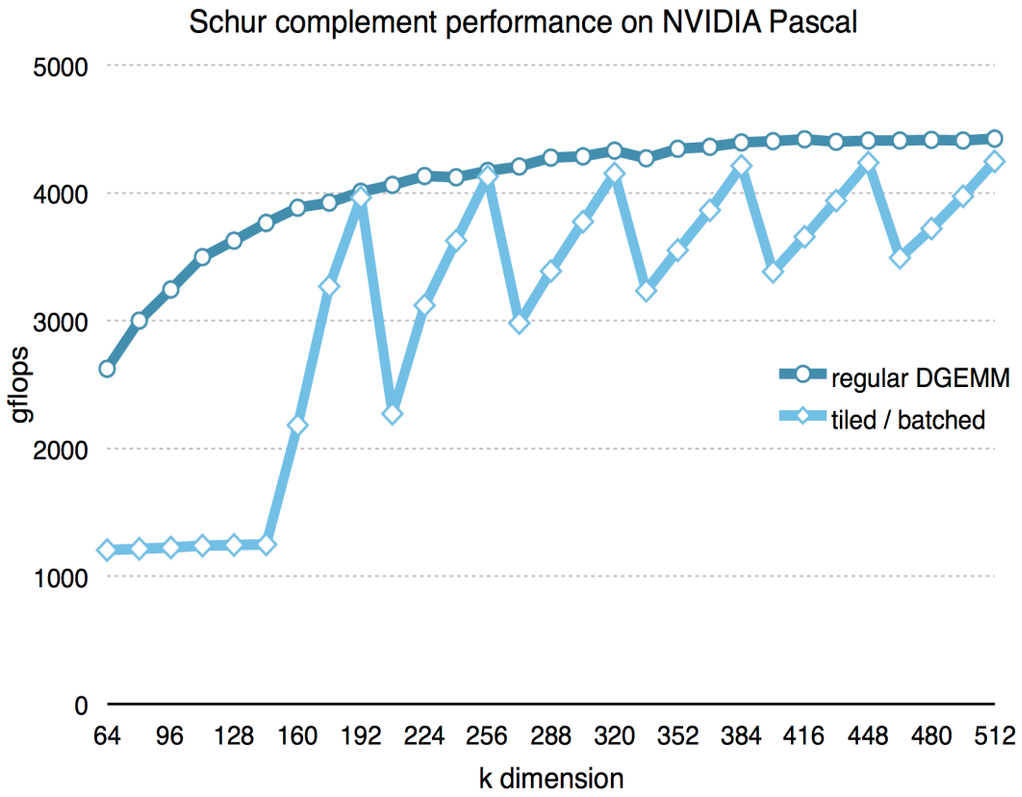


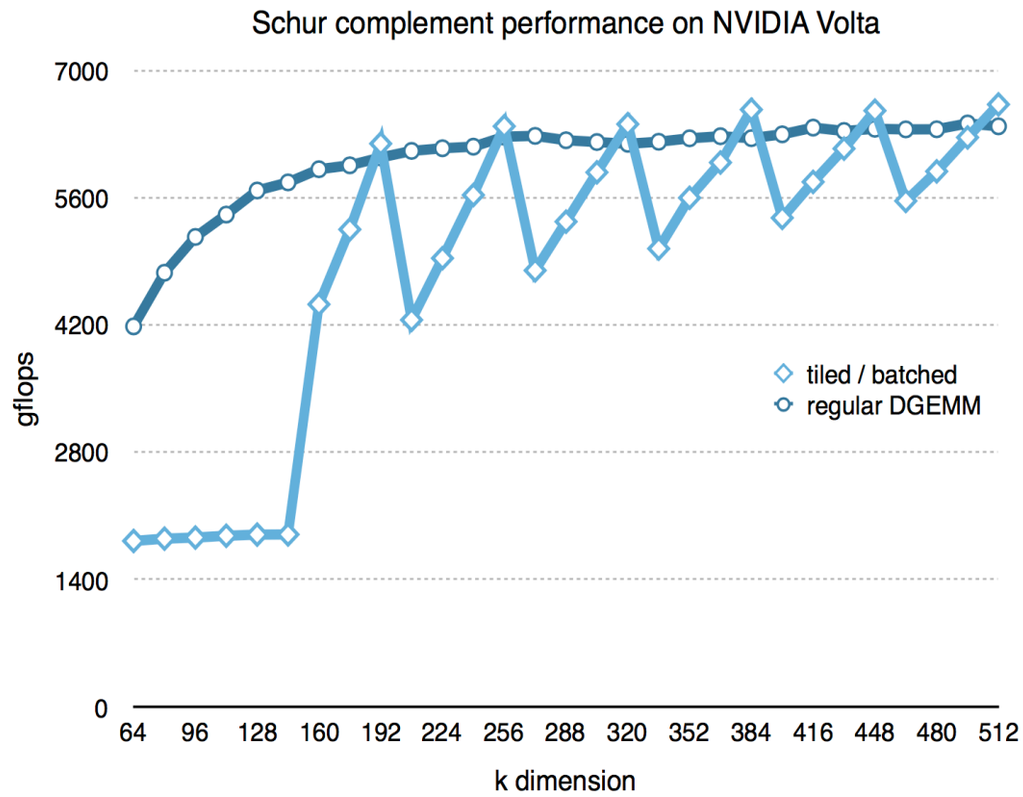
SLATE



$$C = C - A \times B$$

GEMM Efficiency

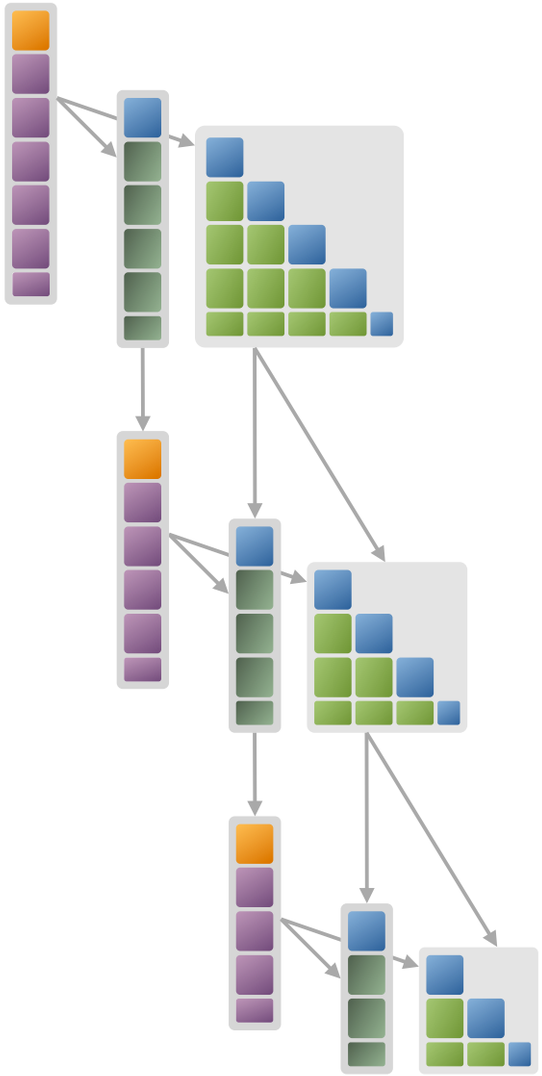




C = C – A × B with small k, i.e., the DGEMM called in LU factorization

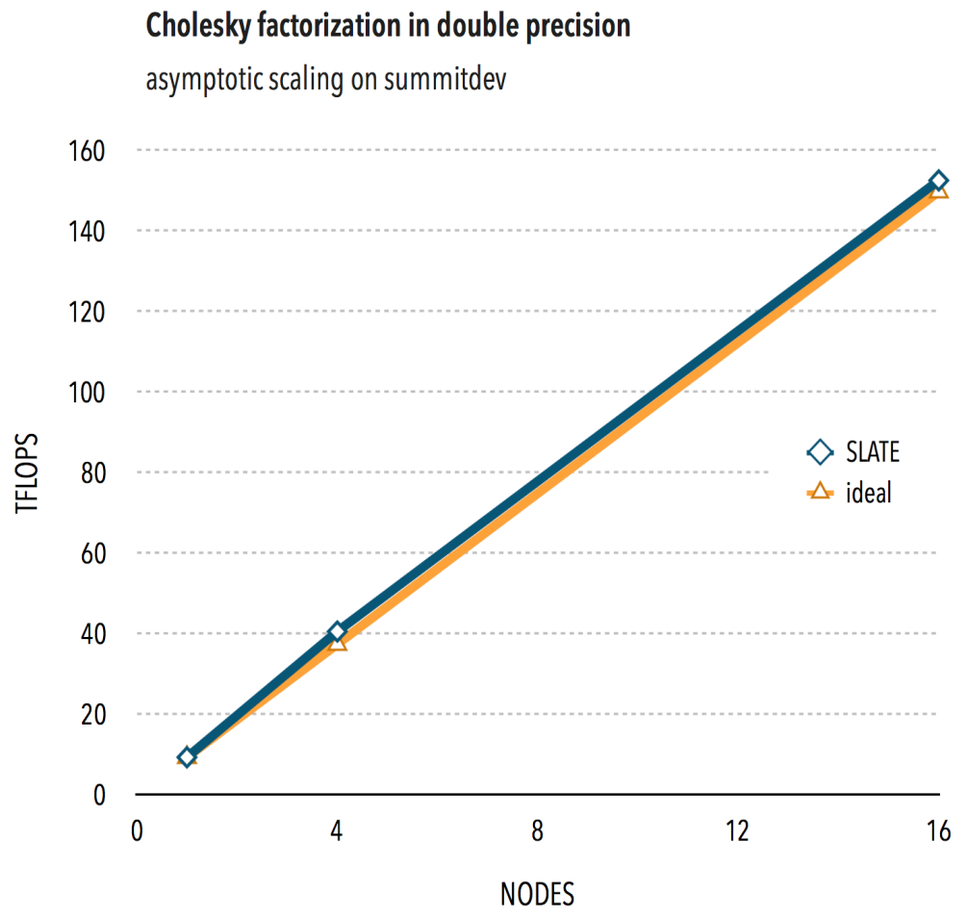
The matrix fills out the GPU memory. The X axis shows the k dimension.

GEMM Scheduling



- nested parallelism
- top level: #pragma omp task depend
- bottom level:
  - #pragma omp task
  - batch GEMM

SLATE GPU Performance



asymptotic scaling

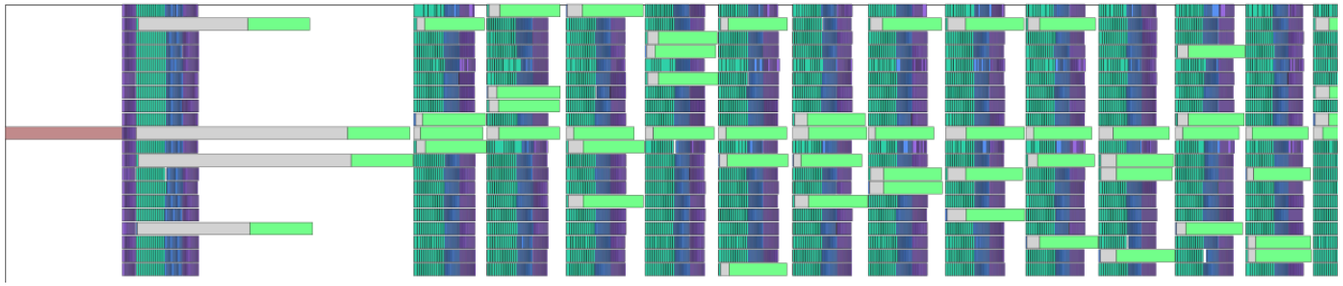
- 112 K × 112 K — 1 node 4 GPUs
- 225 K × 225 K — 4 nodes 16 GPUs
- 450 K × 450 K — 16 nodes 64 GPUs

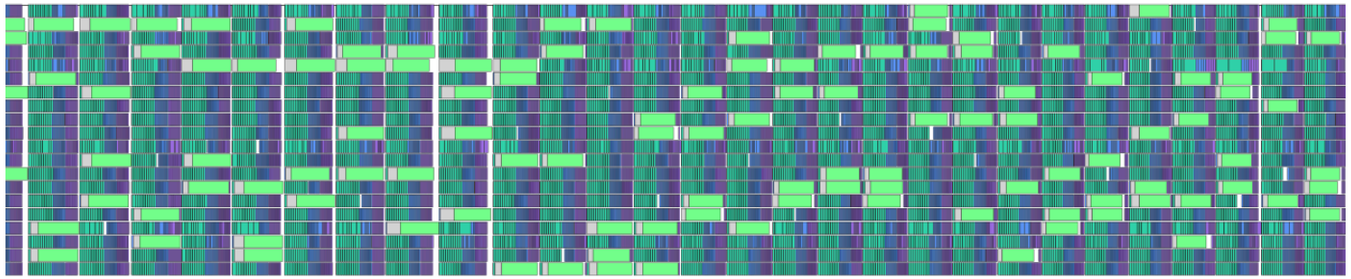
SummitDev @ OLCF

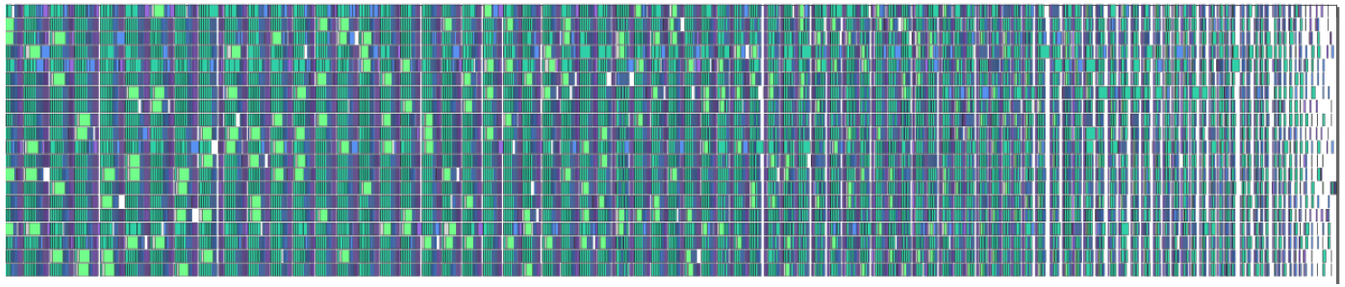
- 3×18 = **54 nodes** (IBM S822LC)
- 2×10 = **20 cores** (IBM POWER8) — ca. 0.5 TFLOPS (2.5%)
- **4 GPUs** (NVIDIA P100) — ca. 20 TFLOPS (97.5%)
- **256 GB DDR4**
- 4×16 = **64 GB HBM2**
- NVLink 1.0 — 80 GBPS (advertised)

- GCC 7.1.0
- ESSL 5.5.0
- CUDA 8.0.54
- Spectrum MPI 10.1.0.3.

SLATE GPU Trace







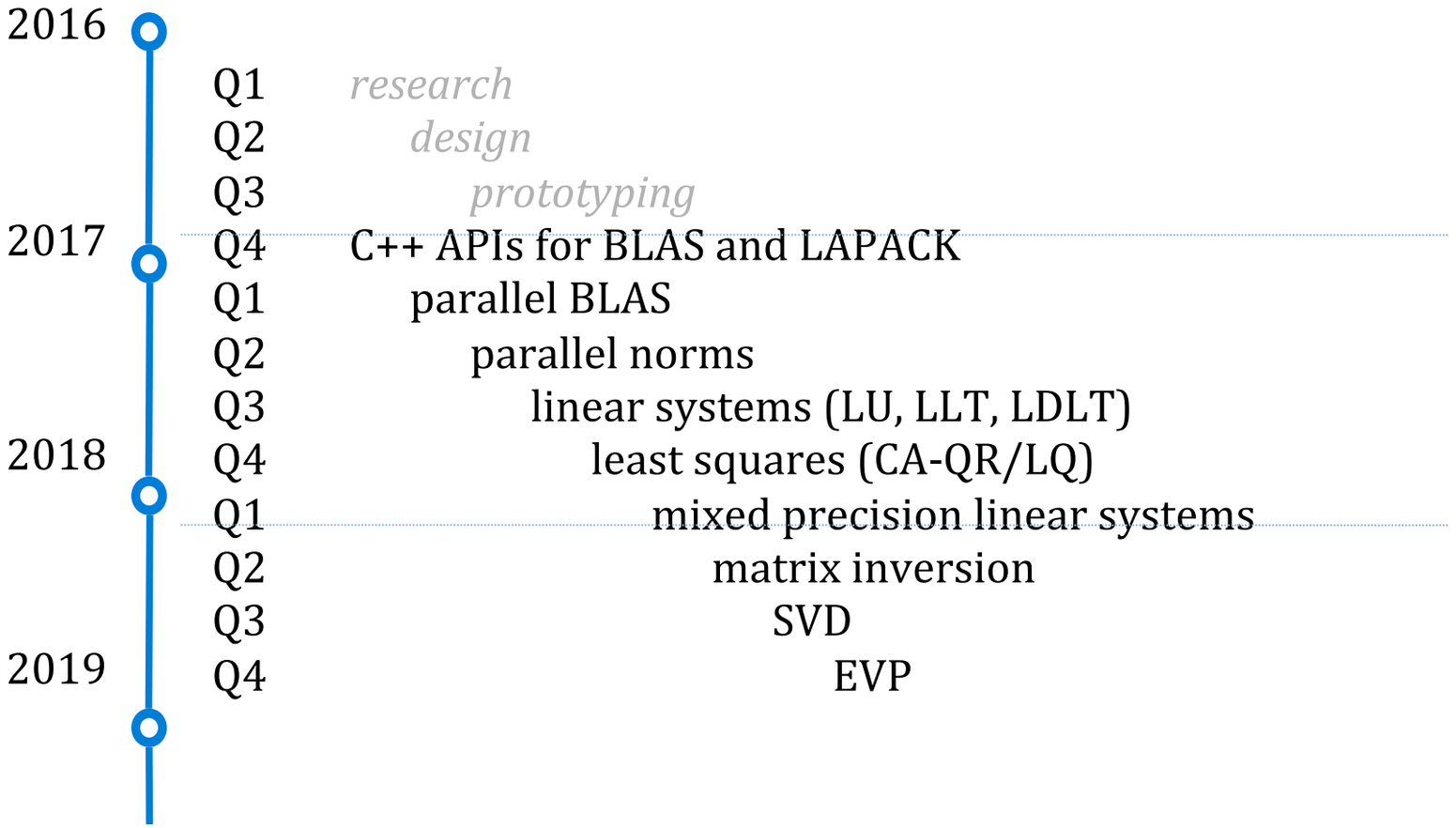
Cholesky factorization

20 cores + 4 GPUs

112 K × 112 K matrix

tile size of 512

SLATE Timeline



Collaborators and Support

MAGMA team
http://icl.cs.utk.edu/magma

PLASMA team
http://icl.cs.utk.edu/plasma

Collaborating partners
University of Tennessee, Knoxville
Lawrence Livermore National Laboratory,
Livermore, CA
LLNL led ECP CEED:
Center for Efficient Exascale Discretizations
University of Manchester, Manchester, UK
University of Paris-Sud, France
INRIA, France

Umeå University
INRIA
Rutherford Appleton Laboratory
University of Manchester