ECP 2018 Annual Meeting Tutorials

Accelerating Linear Algebra with MAGMA

Stan Tomov
Mark Gates
Azzam Haidar
{tomov, mgates3, haidar}@icl.utk.edu

Innovative Computing Laboratory
University of Tennessee, Knoxville

THE UNIVERSITY OF TENNESSEE KNOXVILLE

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

ECP EXASCALE COMPUTING PROJECT

Outline

- **Part I**

  Overview of dense linear algebra libraries

  Design principles and fundamentals

- **Part II**

  MAGMA Overview

  Availability, routines, code, testers, methodology

- **Part III**

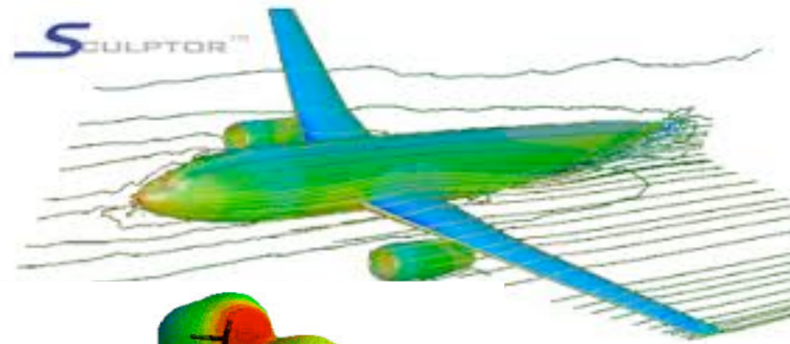
  MAGMA Batched

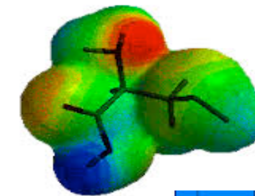
  MAGMA Sparse

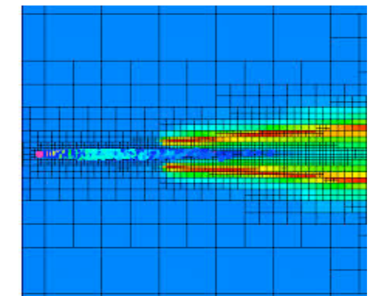
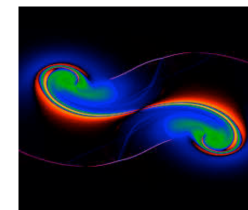
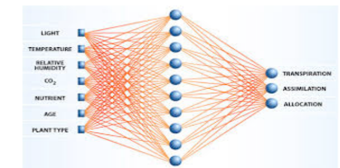
Dense Linear Algebra in Applications

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

icl.utk.edu/research

BLAS — Kernels for dense linear algebra

LAPACK — Sequential dense linear algebra

ScaLAPACK — Parallel distributed dense linear algebra

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

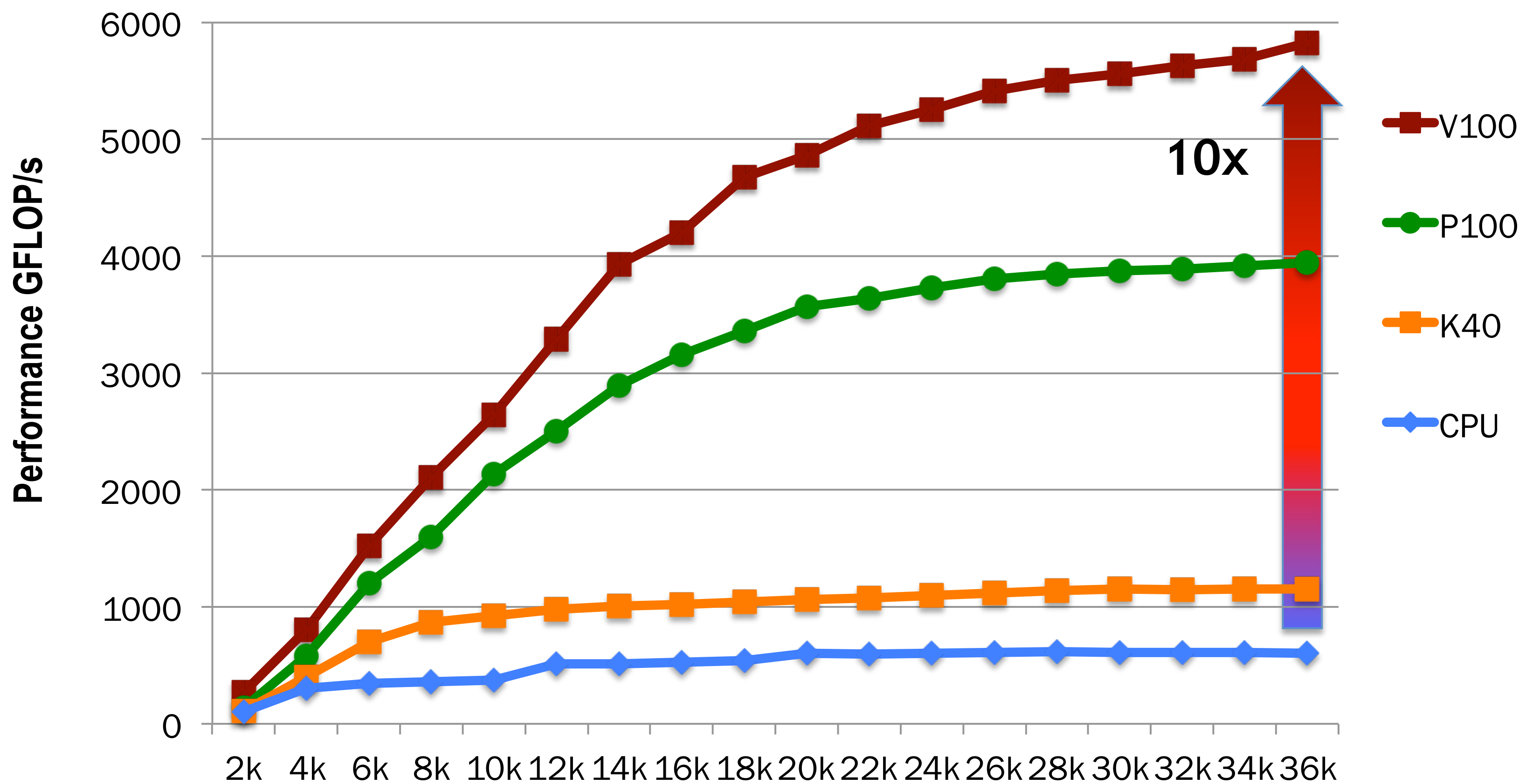
ICL
ECP EXASCALE COMPUTING PROJECT

Why use GPUs in HPC?

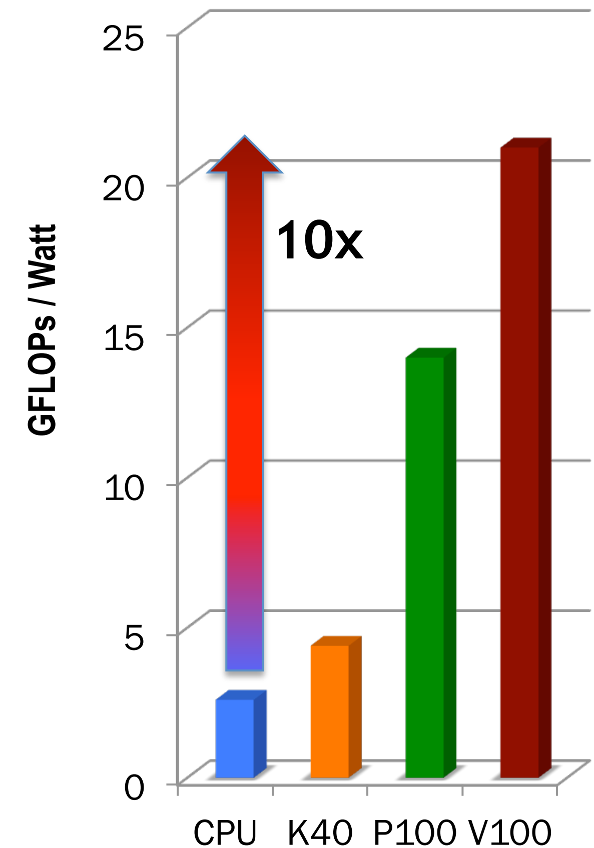
PERFORMANCE & ENERGY EFFICIENCY

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



Energy efficiency (under ~ the same power draw)



Matrix size N x N

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

$$y = \alpha x + \beta y$$

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

$$y = \alpha x + \beta y$$

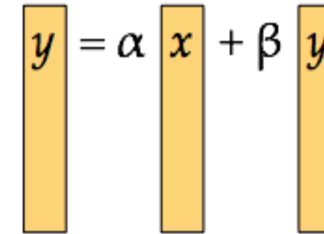
- Level 2 BLAS — matrix-vector operations
  - $O(n^2)$ data and flops
  - Memory bound:
    $O(1)$ flops per memory access

$$y = \alpha A x + \beta y$$

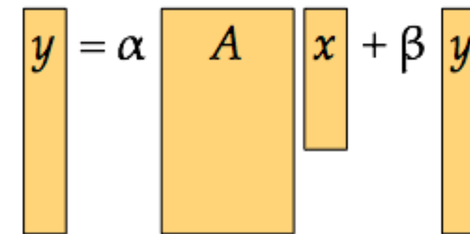
ICL
ECP EXASCALE COMPUTING PROJECT

BLAS: Basic Linear Algebra Subroutines

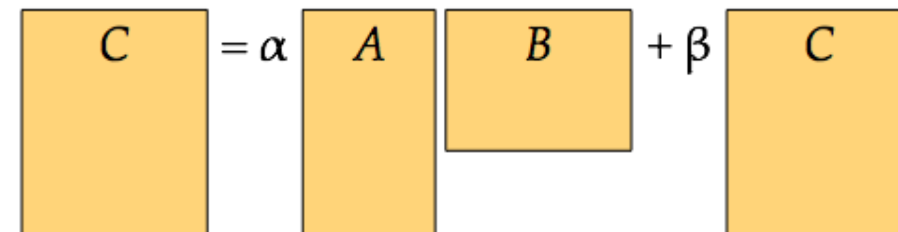
- Level 1 BLAS — vector operations
  - $O(n)$ data and flops (floating point operations)
  - Memory bound:
    $O(1)$ flops per memory access



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



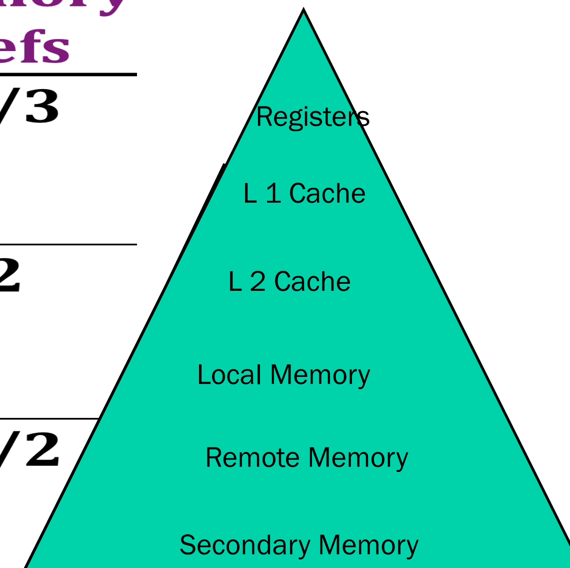
- Level 3 BLAS — matrix-matrix operations
  - $O(n^2)$ data, $O(n^3)$ flops
  - Surface-to-volume effect
  - Compute bound:
    $O(n)$ flops per memory access



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



ICL
ECP EXASCALE COMPUTING PROJECT

Level 1, 2 and 3 BLAS

Nvidia **P100**, 1.19 GHz, Peak DP = 4700 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): $(A^T A)x=\sigma^2 x$
    - 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

ICL
ECP EXASCALE COMPUTING PROJECT

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

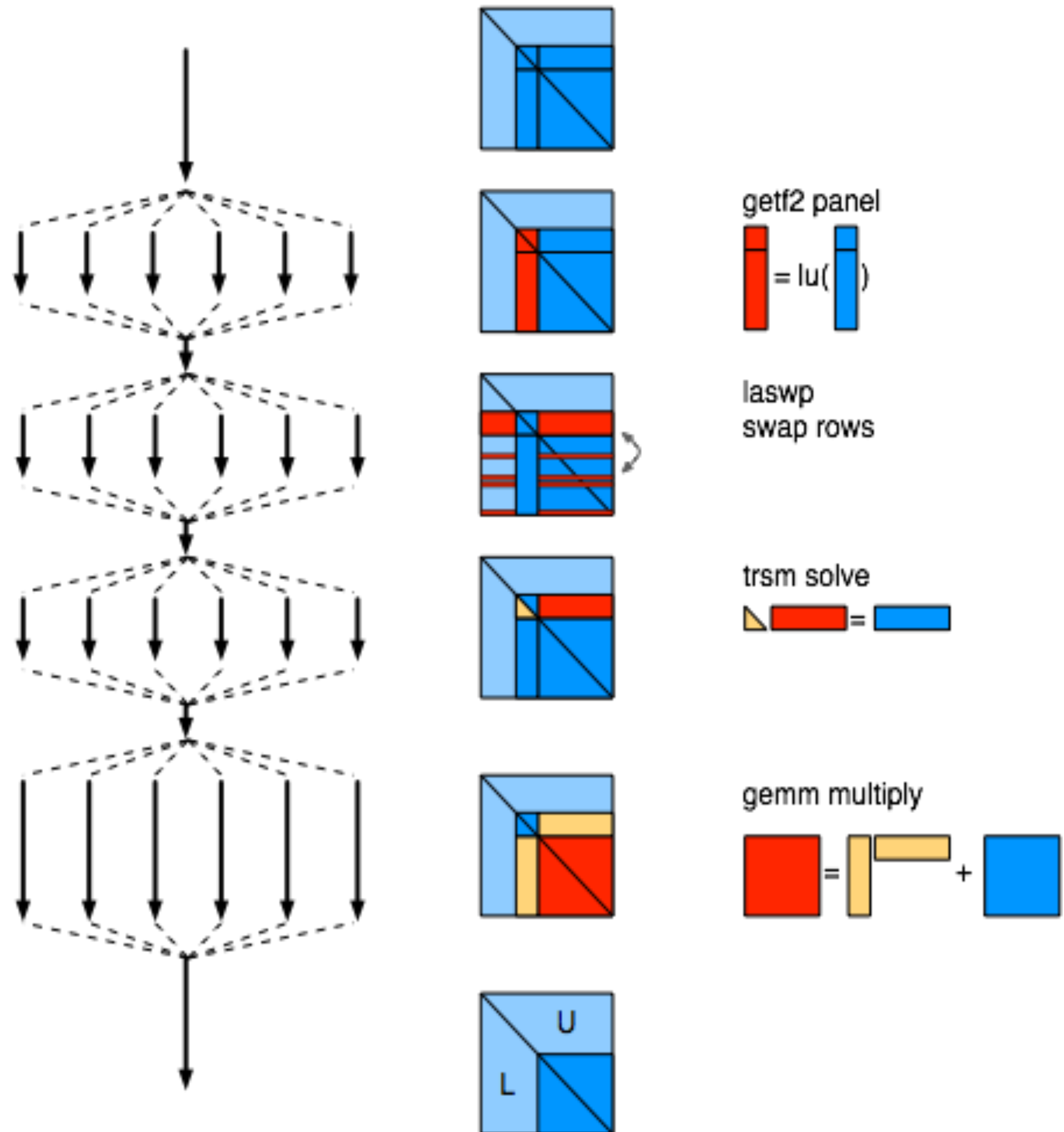
LAPACK is SEQUENTIAL

LAPACK is a REFERENCE implementation

ICL
ECP EXASCALE COMPUTING PROJECT

Parallelism in LAPACK

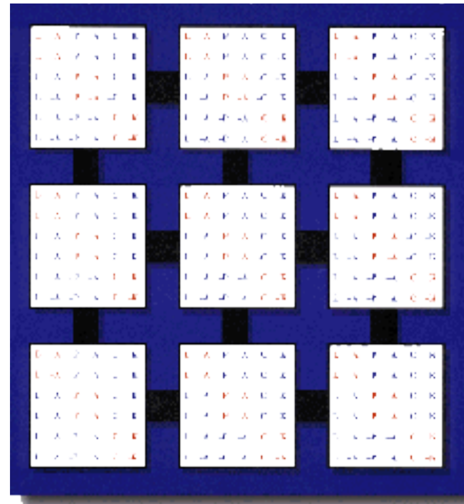
- Most flops in gemm update
  - 2/3 n^3 term
  - Easily parallelized using multi-threaded BLAS
  - Done in any reasonable software
- Other operations lower order
  - Potentially expensive if not parallelized



ICL
ECP
EXASCALE COMPUTING PROJECT

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

ICL
ECP EXASCALE COMPUTING PROJECT

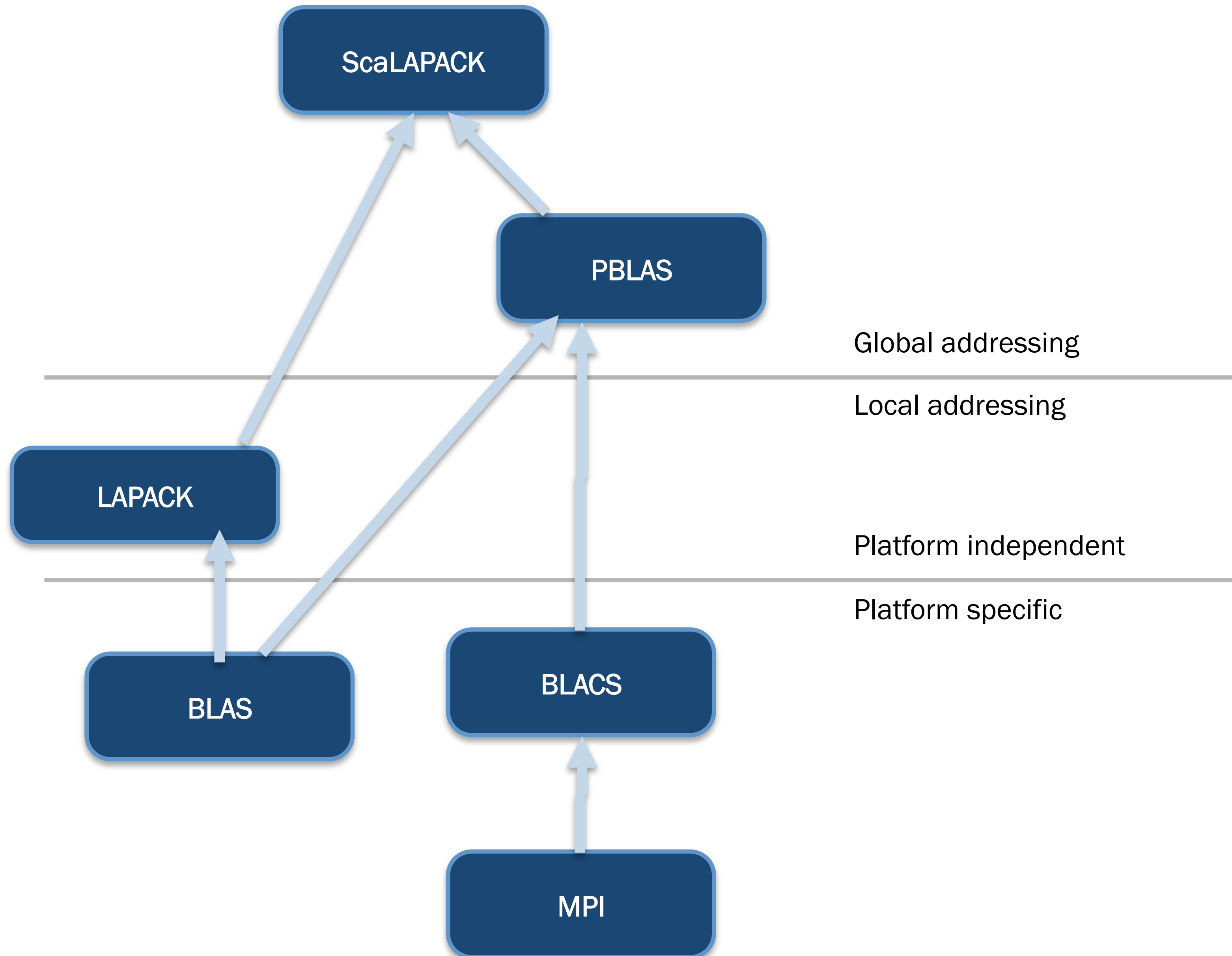
PBLAS

- 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

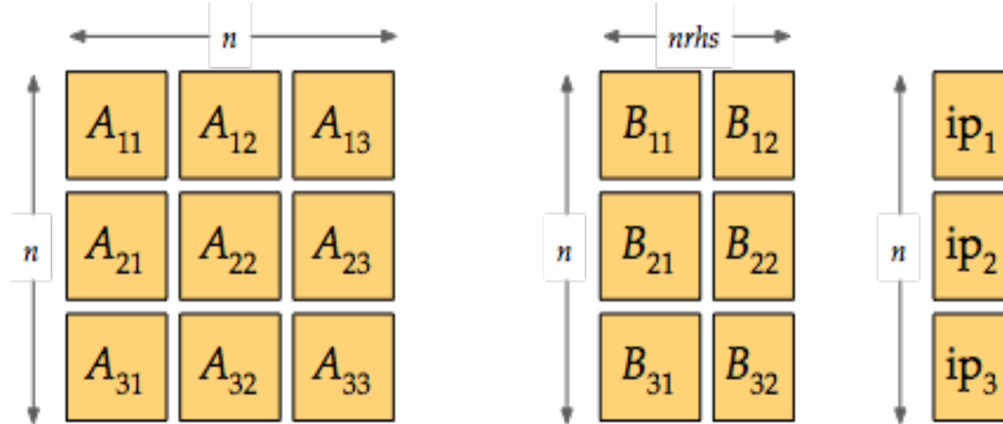


ICL
ECP EXASCALE COMPUTING PROJECT

ScaLAPACK routine, solve $AX = B$

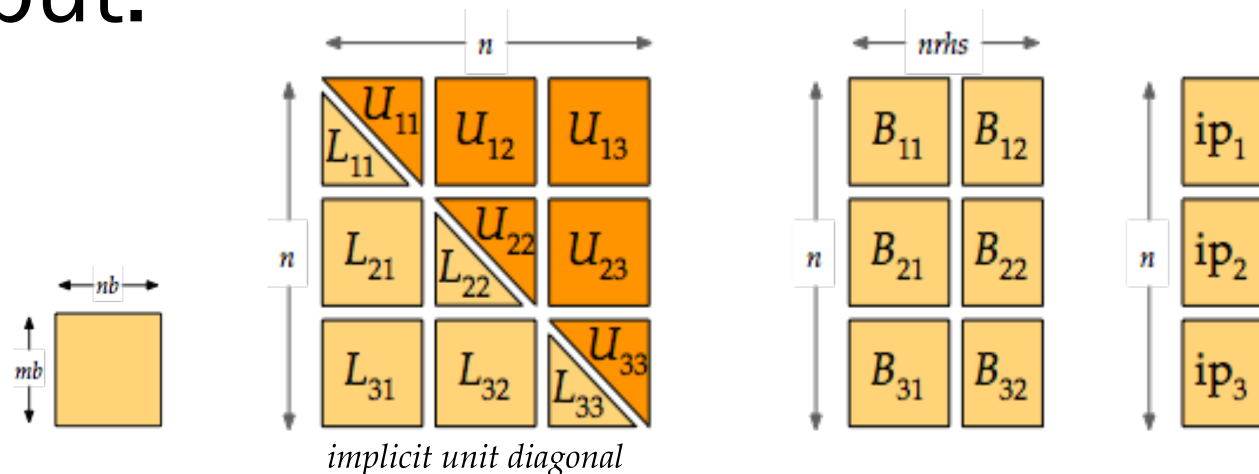
- LAPACK: `dgesv(n, nrhs, A, lda, ipiv, B, ldb, info)`
- ScaLAPACK: `pdgesv(n, nrhs, A, ia, ja, descA, ipiv, B, ib, jb, descB, info)`

- input:



Global matrix point of view

- output:



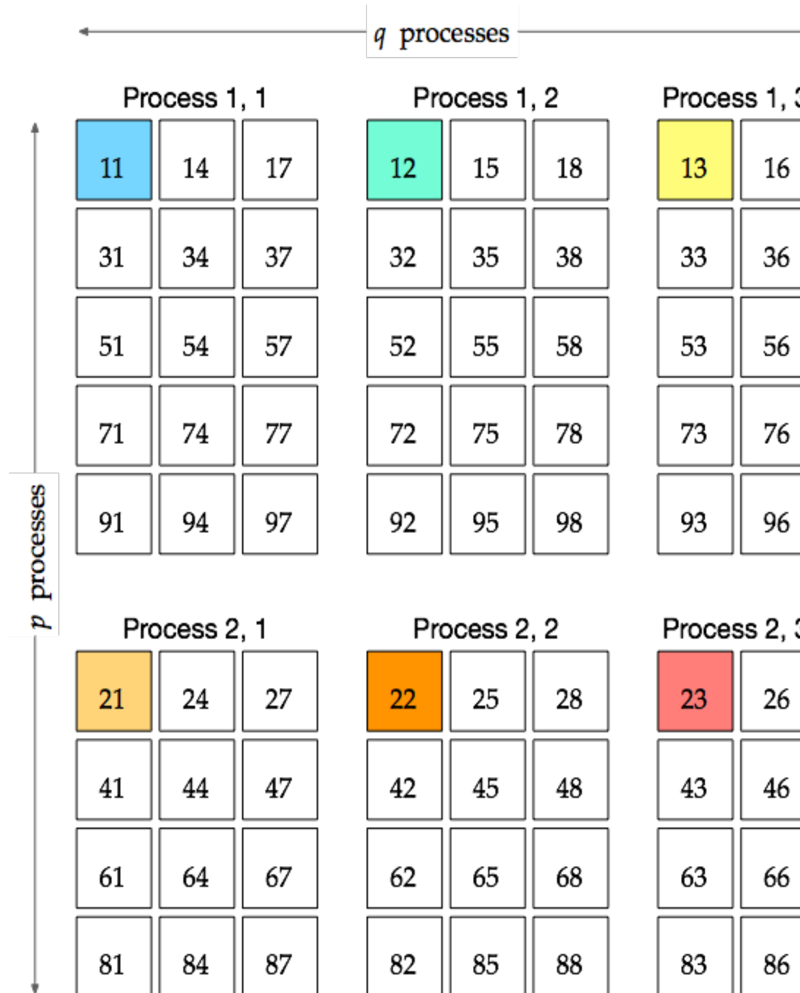
info (error code)
- = 0: no error
- < 0: invalid argument
- > 0: numerical error (e.g., singular)

L, *U* overwrite *A*
X overwrites *B*

ICL
ECP EXASCALE COMPUTING PROJECT

2D block-cyclic layout

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



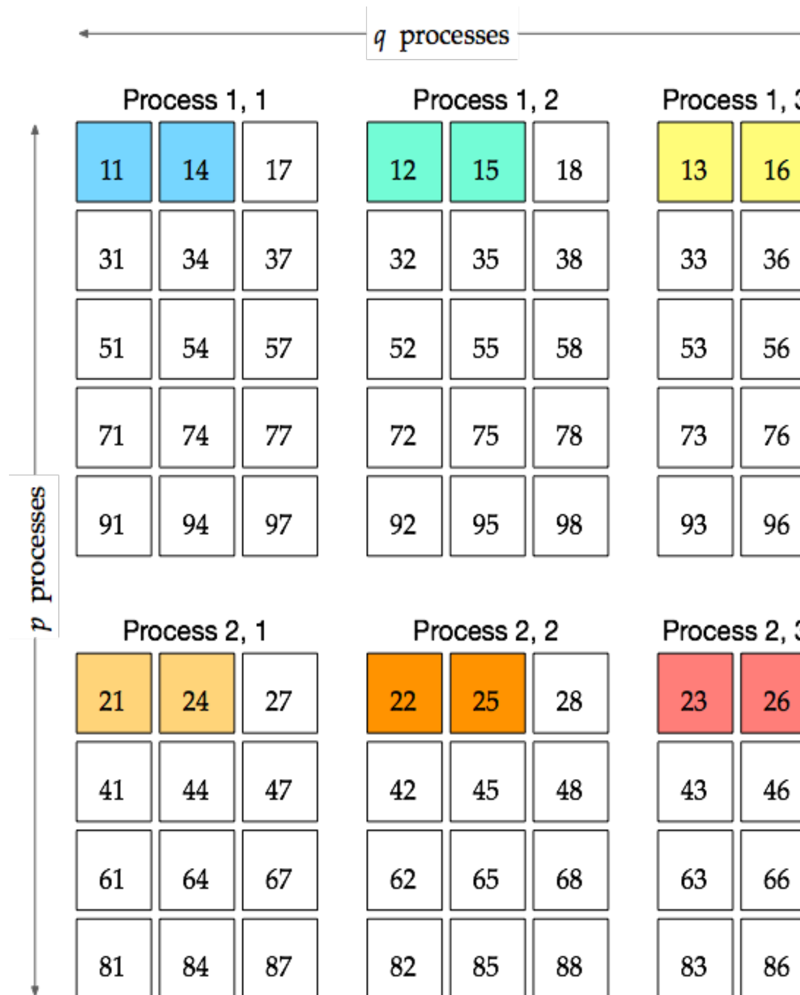
ICL
ECP EXASCALE COMPUTING PROJECT

2D block-cyclic layout

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

Global matrix view

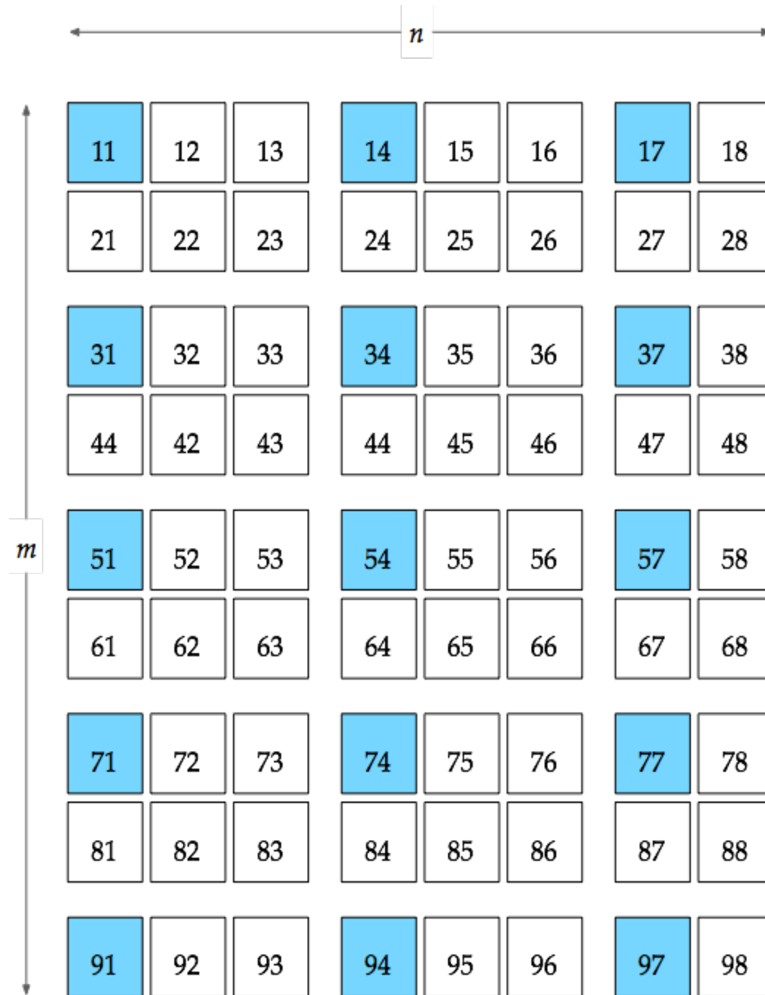
Local process point of view



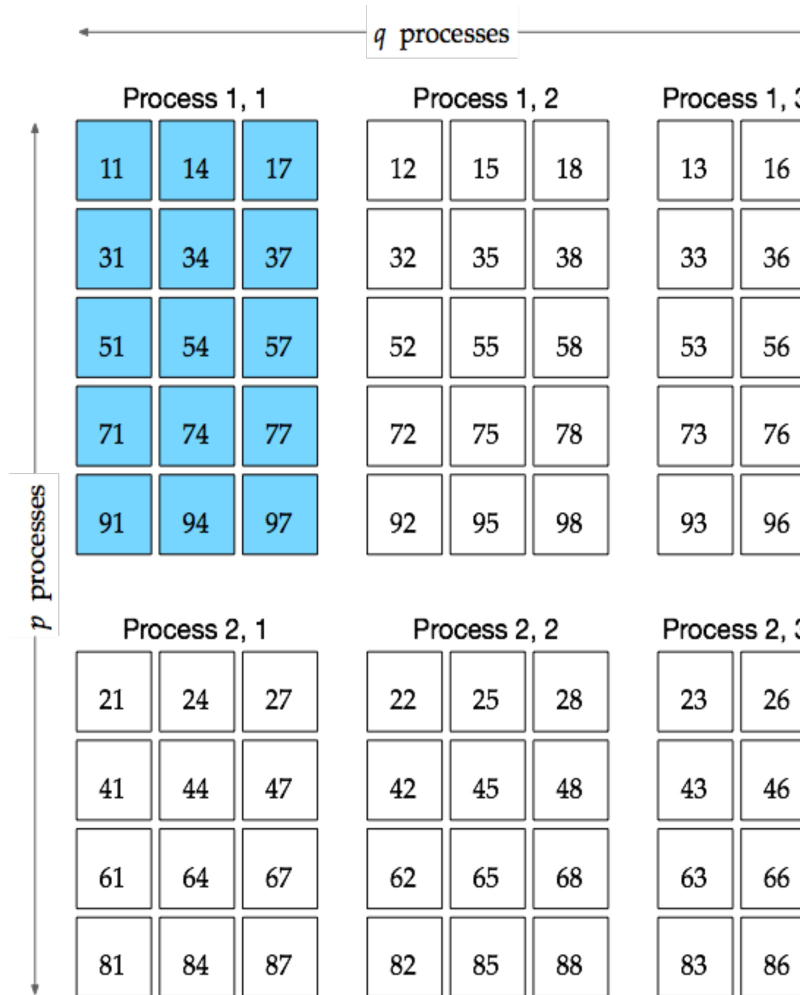
2D block-cyclic layout

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

Global matrix view



Local process point of view



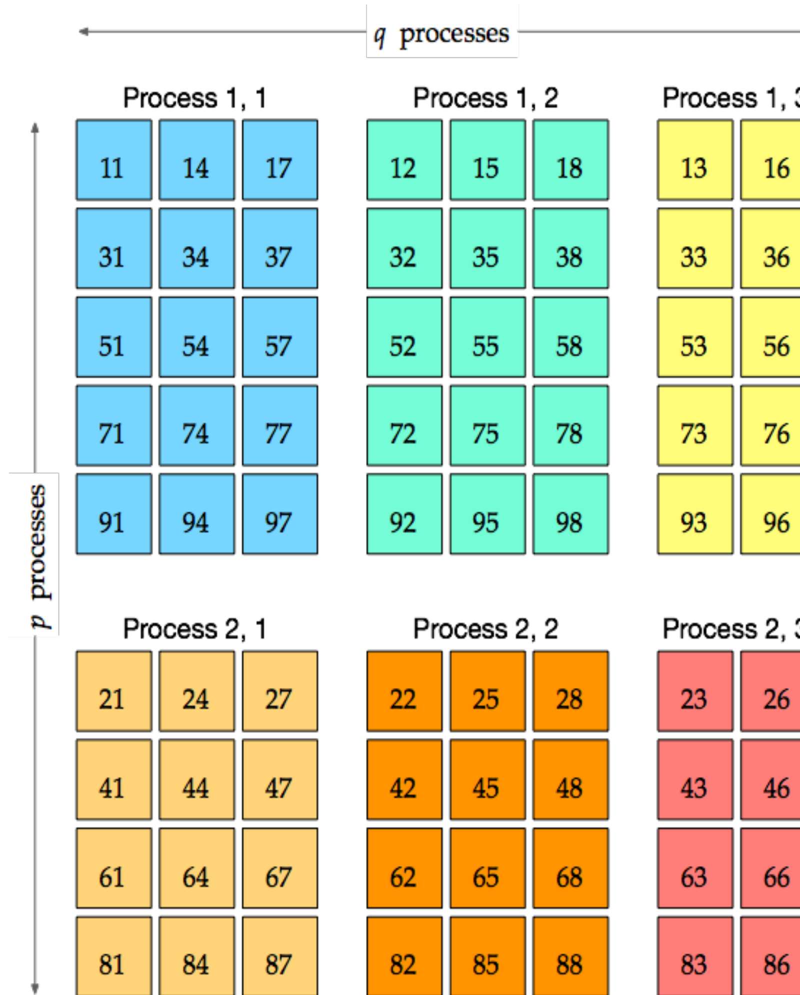
ICL
ECP EXASCALE COMPUTING PROJECT

2D block-cyclic layout

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

Global matrix view

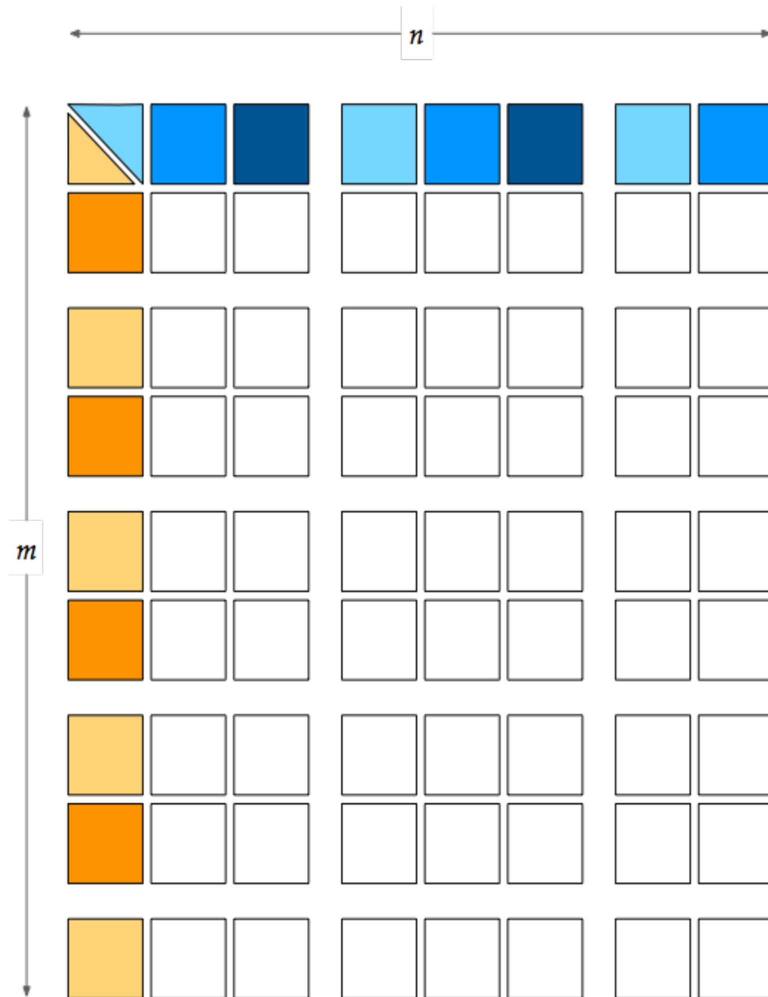
Local process point of view

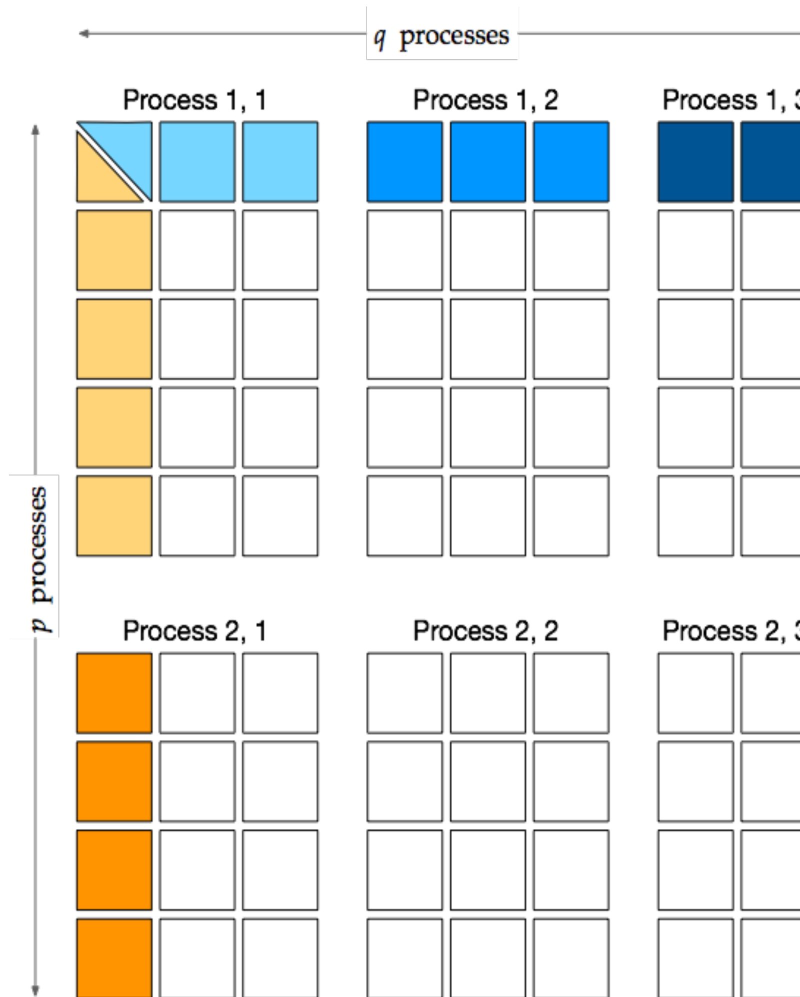


ICL
ECP EXASCALE COMPUTING PROJECT

2D block-cyclic layout

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

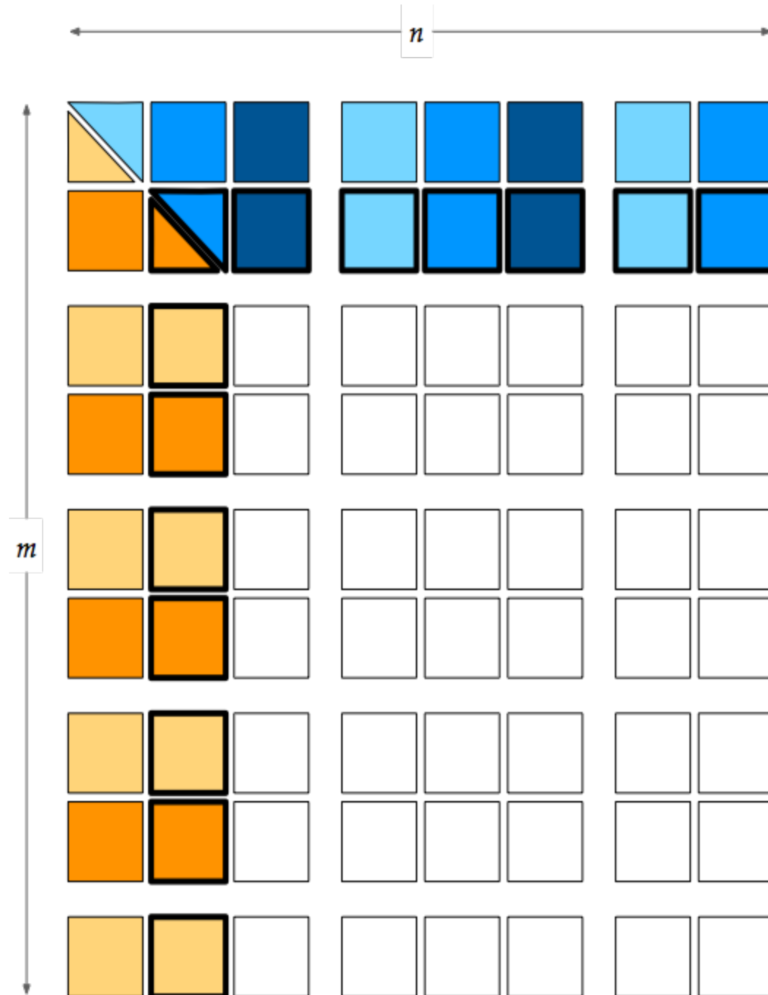


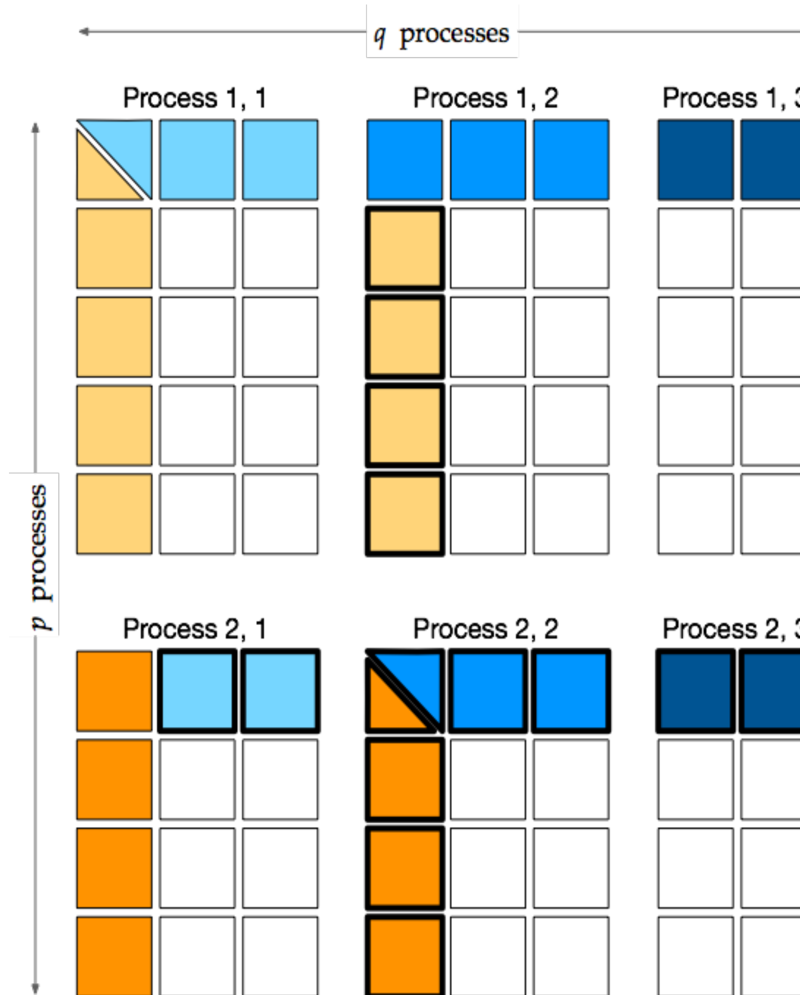


ICL
ECP EXASCALE COMPUTING PROJECT

2D block-cyclic layout

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

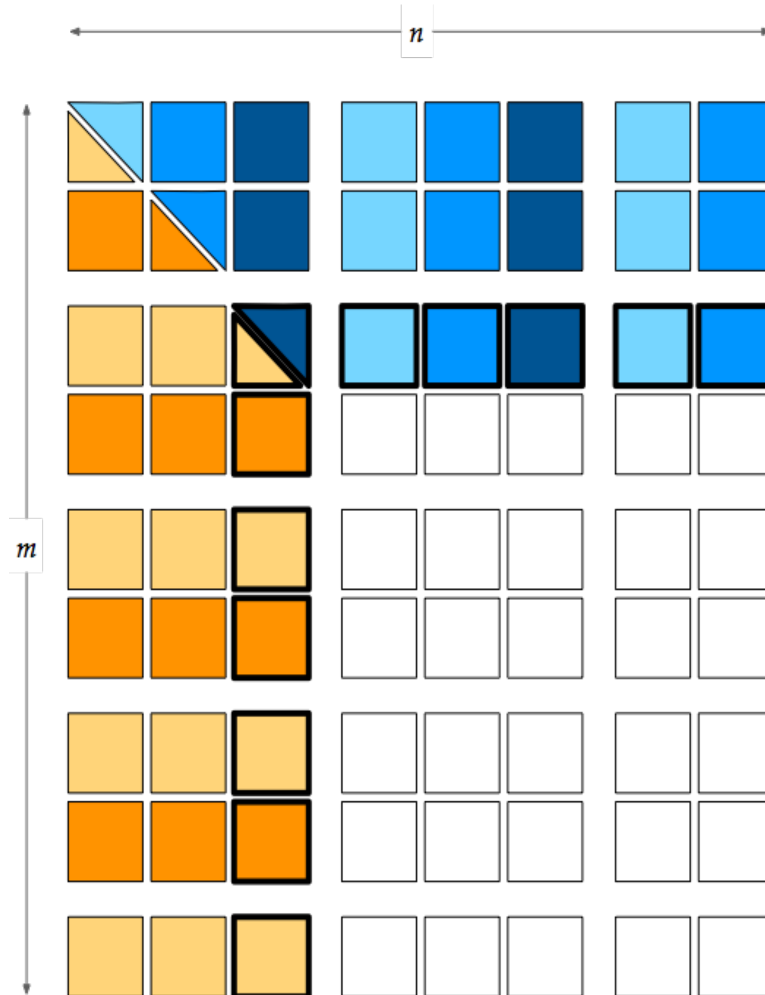


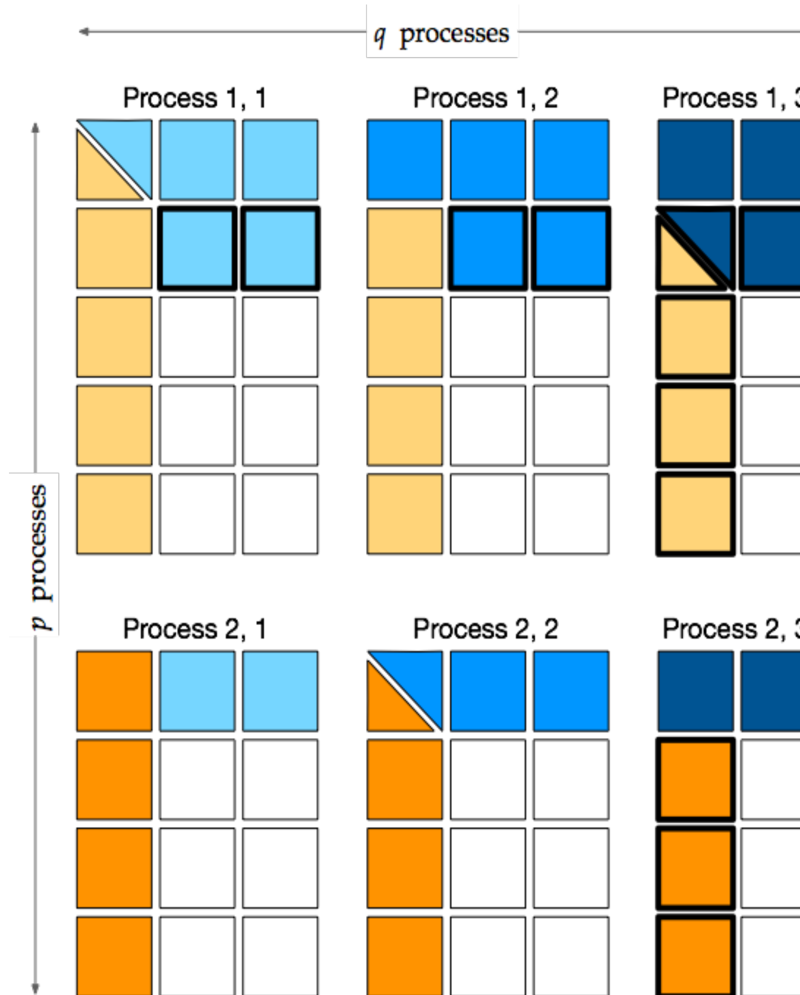


ICL
ECP EXASCALE COMPUTING PROJECT

2D block-cyclic layout

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

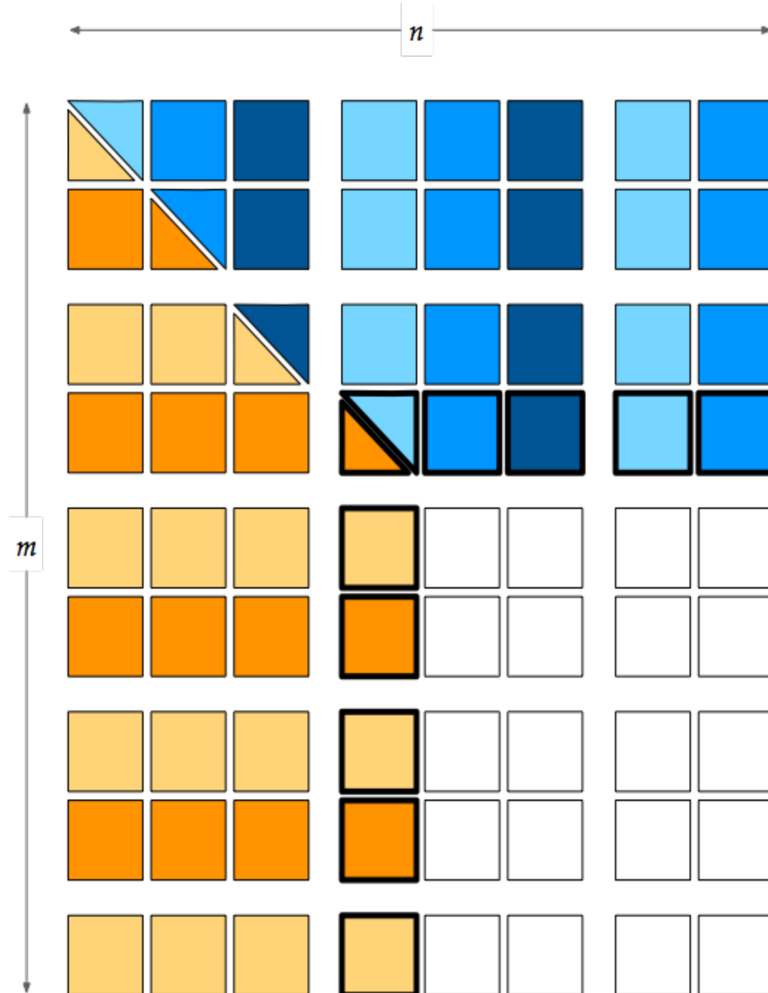


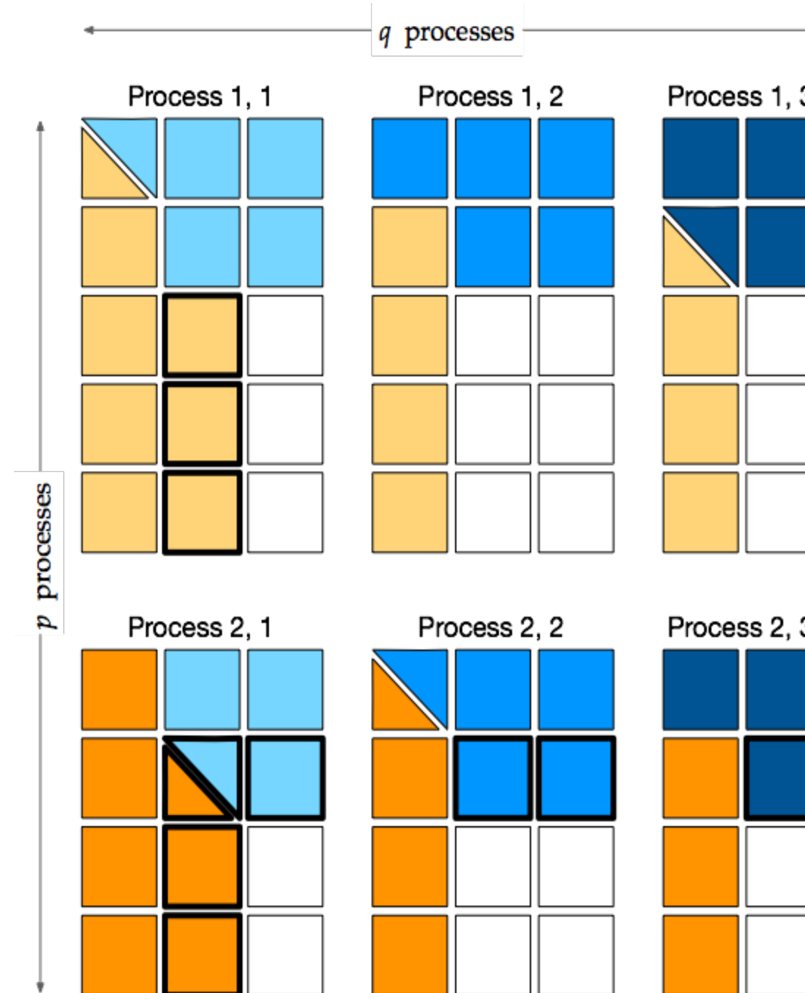


ICL
ECP EXASCALE COMPUTING PROJECT

2D block-cyclic layout

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



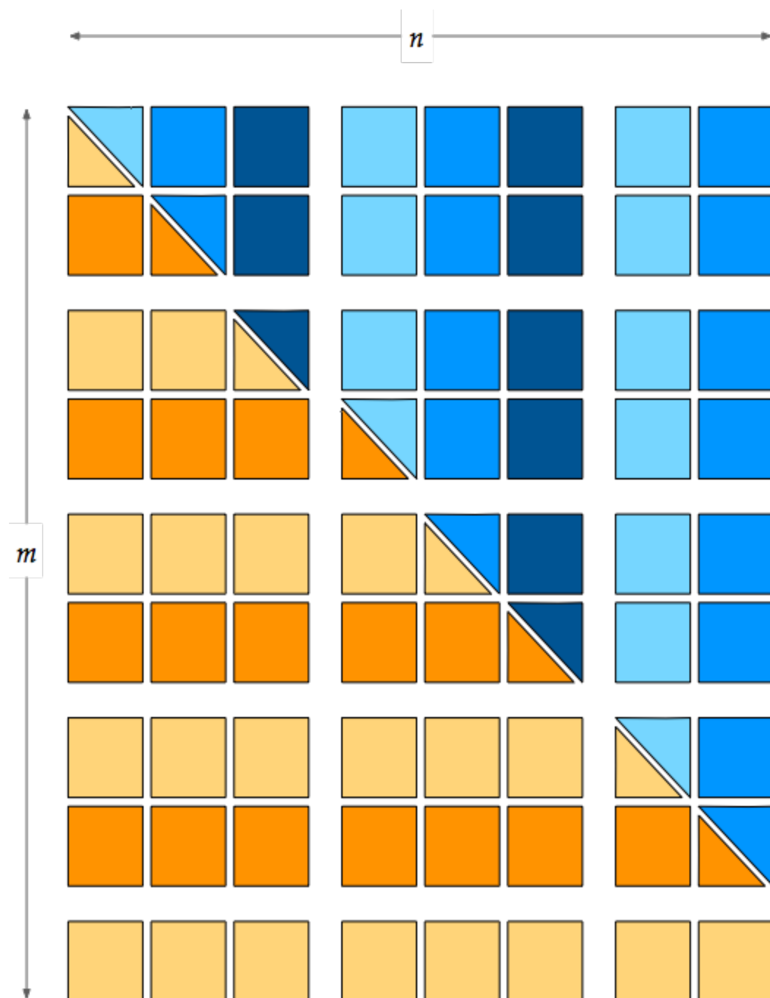


ICL
ECP EXASCALE COMPUTING PROJECT

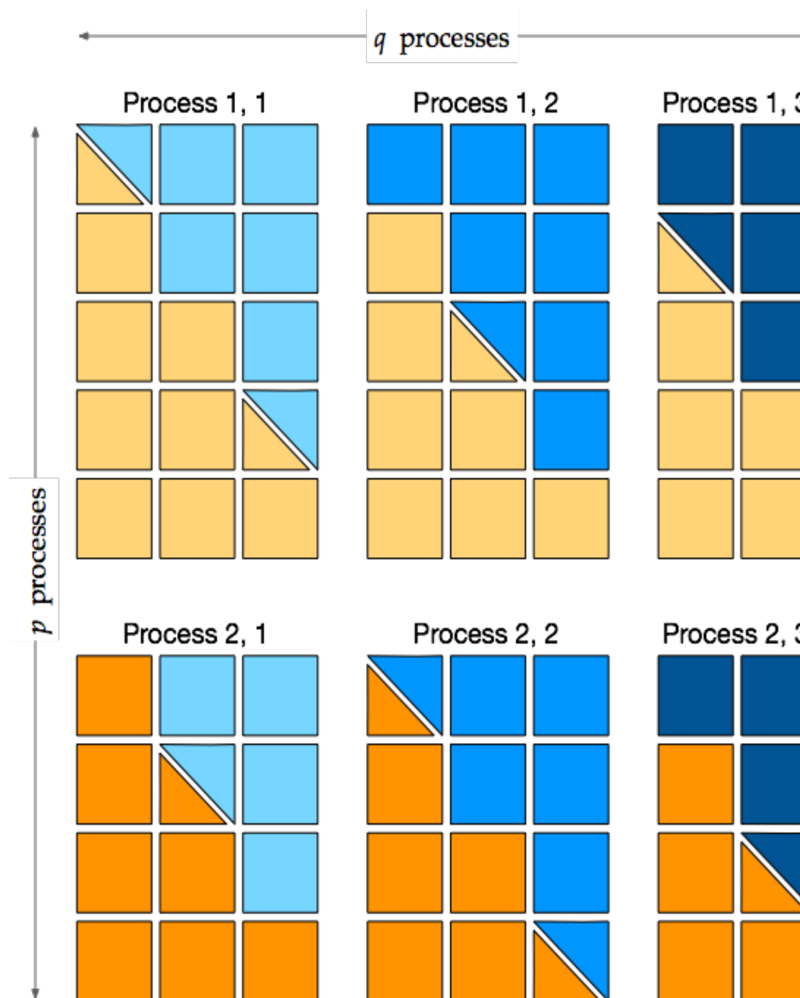
2D block-cyclic layout

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

Global matrix view



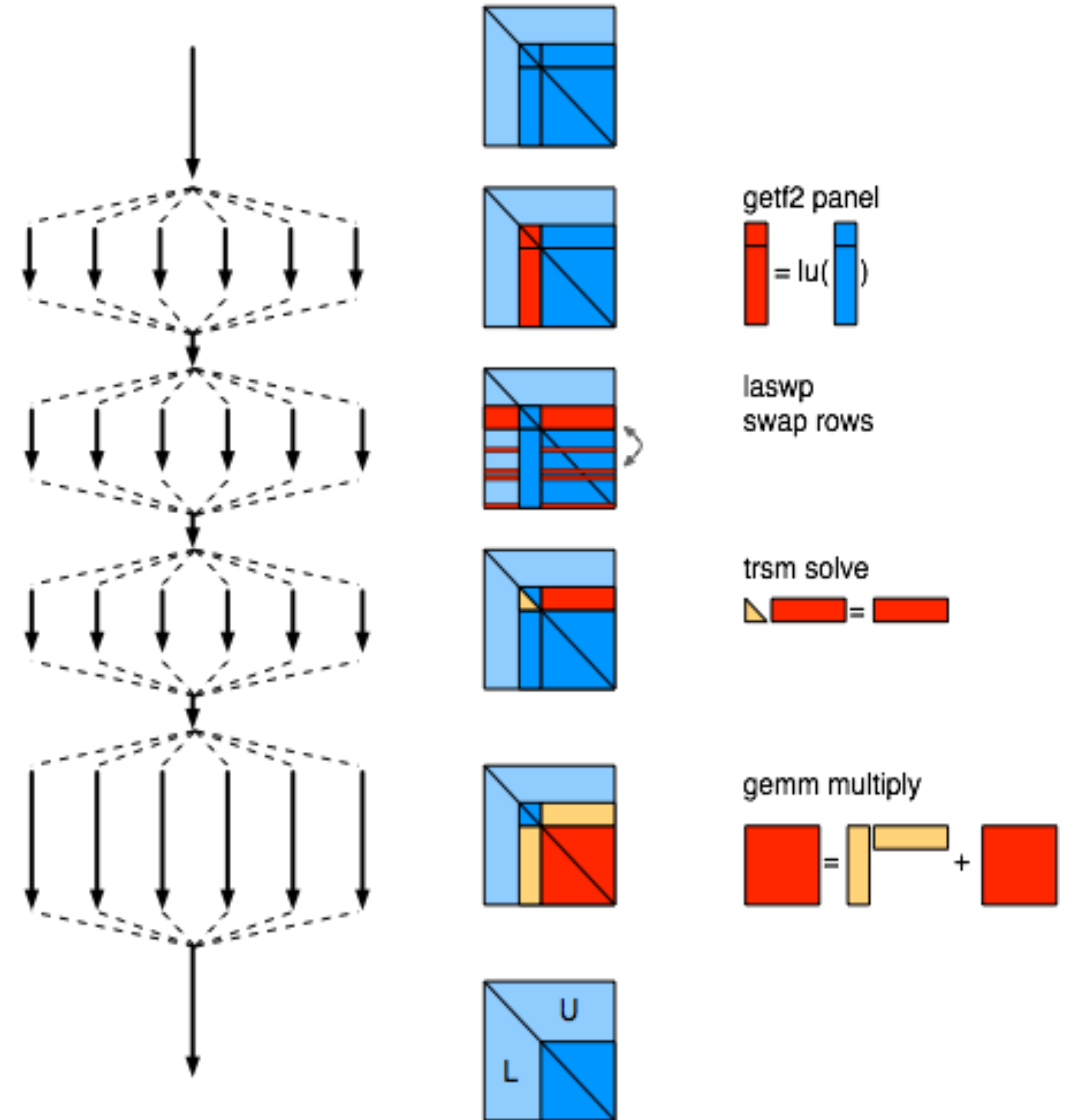
Local process point of view



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



ICL

ECP EXASCALE COMPUTING PROJECT

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

ICL ECP EXASCALE COMPUTING PROJECT

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

ICL
ECP EXASCALE COMPUTING PROJECT

Software Projects

netlib.org

icl.utk.edu/research

LAPACK

ScaLAPACK

BLAS

CBLAS

LAPACKE

PLASMA

MAGMA

SLATE

QUARK

PaRSEC

scheduling
(multicore)

scheduling
(distributed memory)

dynamic
runtime
schedulers

ICL
ECP EXASCALE COMPUTING PROJECT

Software Projects

netlib.org

icl.utk.edu/research

LAPACK

ScaLAPACK

BLAS

CBLAS

LAPACKE

PLASMA

OpenMP

scheduling
(multicore)

MAGMA

SLATE

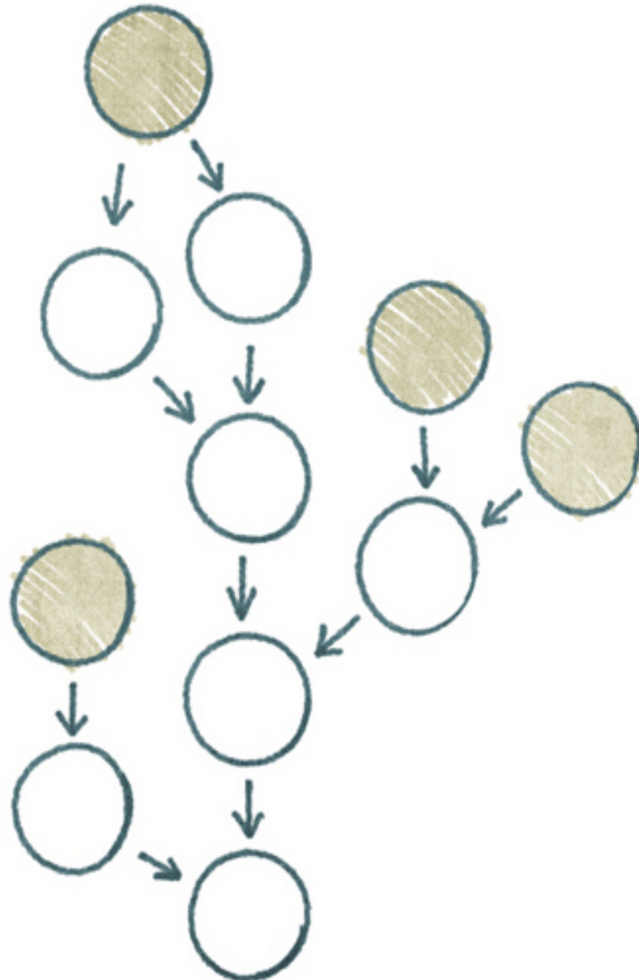
PaRSEC

scheduling
(distributed memory)

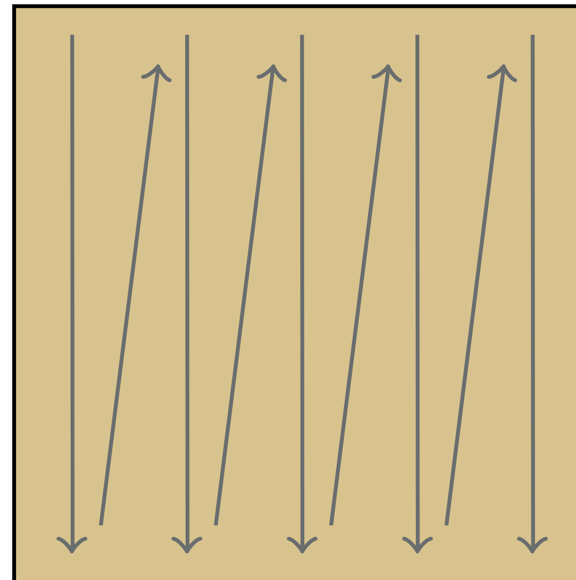
dynamic
runtime
schedulers

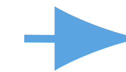
ICL ECP EXASCALE COMPUTING PROJECT

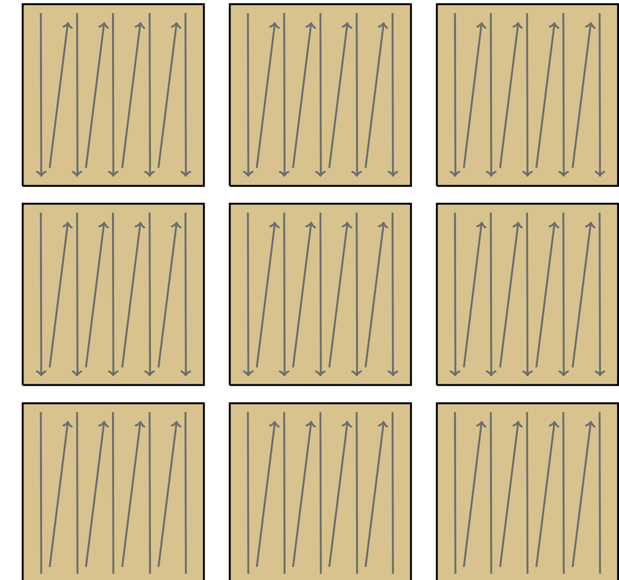
PLASMA



- 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 );
}
```


ECP EXASCALE COMPUTING PROJECT

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 );
}
```

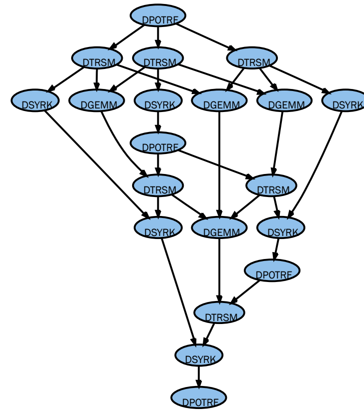
```
void dgemm_tile_quark(Quark* quark, Quark_Task_Flags * task_flags,
    enum CBLAS_ORDER order, enum CBLAS_TRANSPOSE transa,
    enum CBLAS_TRANSPOSE transb, enum CBLAS_TRANSPOSE transc,
    int m, int n, int k, double alpha, double *A, int lda, double *B, int ldb,
    double beta, double *C, int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

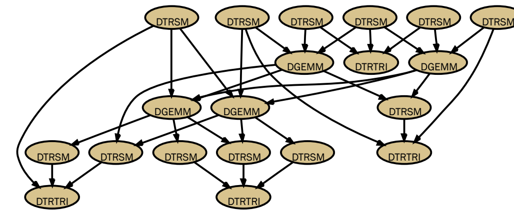
```
#include <omp.h>

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);
            }
        ...
    }
}
```

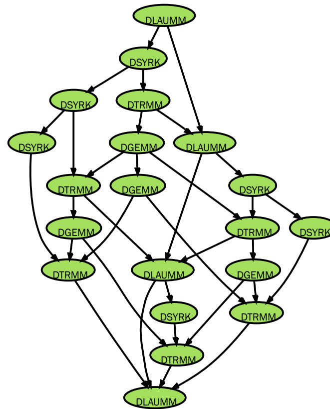
```
void dgemm_tile_task( Quark* quark ) {

    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 );
}
```

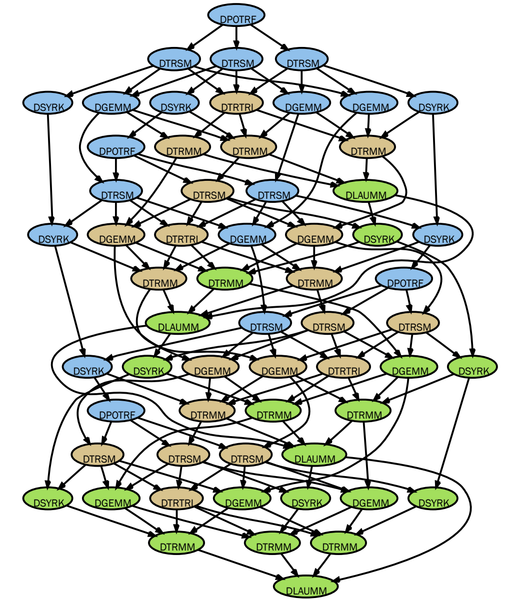

EXASCALE COMPUTING PROJECT

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



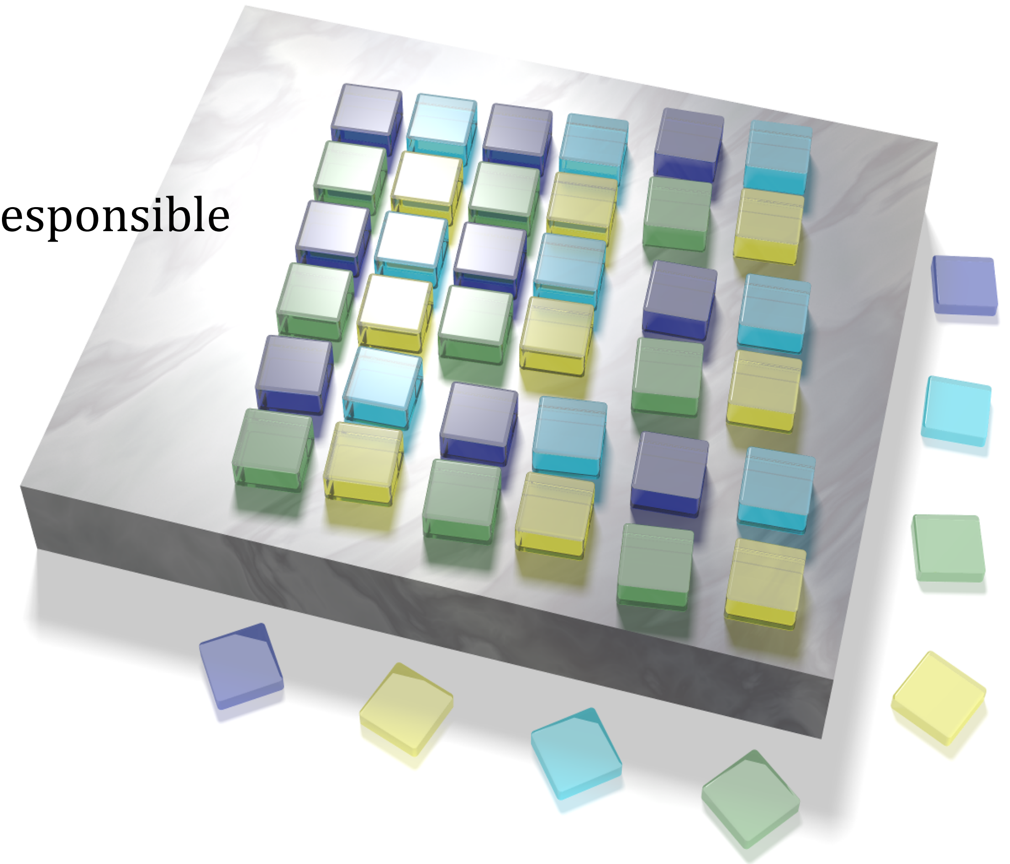






SLATE
Software for Linear Algebra Targeting Exascale

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

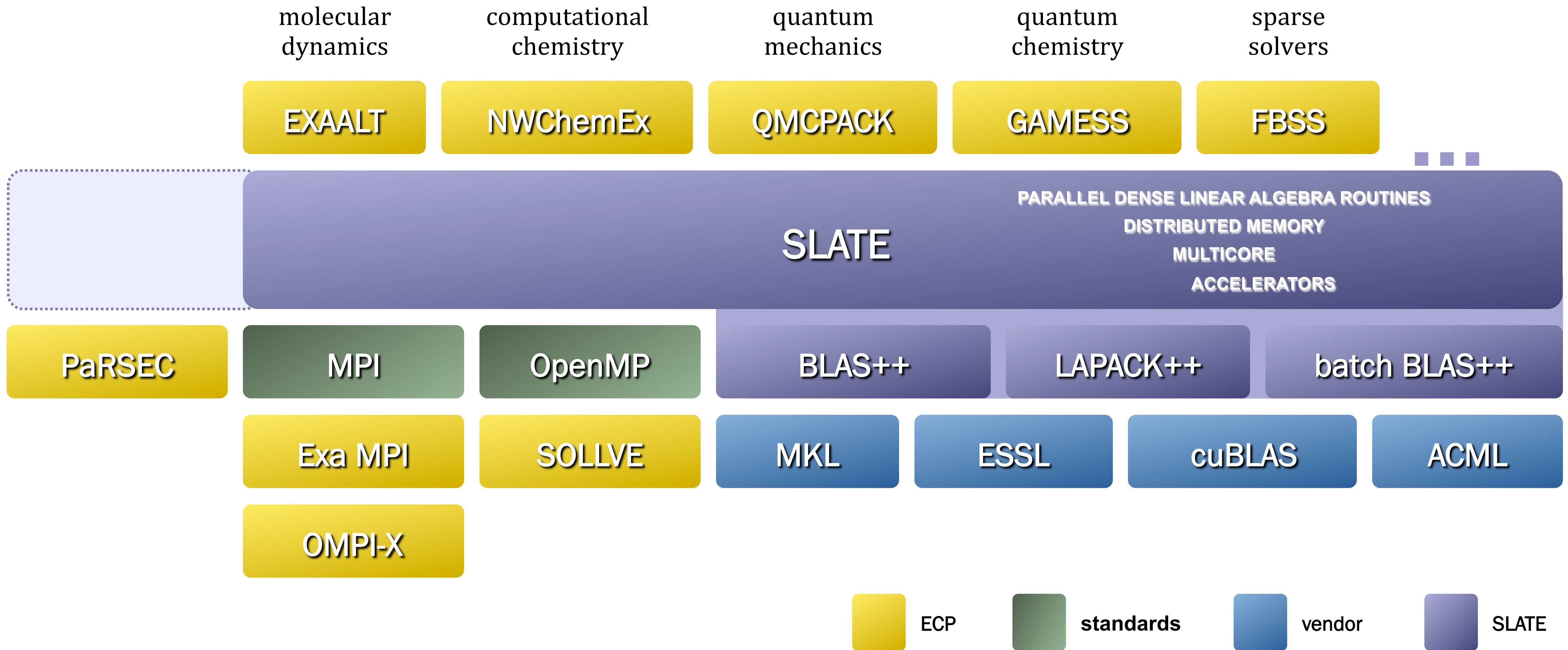
Coverage	ScaLAPACK and beyond
Modern Hardware	DOE CORAL (pre Exascale) → DOE Exascale
Portability	Intel Xeon (&Phi), IBM POWER, ARM, NVIDIA, AMD, ...
Modern Language	C++11/14/17 (templates, STL, overloading, polymorphism, ...)
Modern Standards	MPI 3, OpenMP 4/5 (&omp target)
Performance	80-90% of peak (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

ICL
ECP EXASCALE COMPUTING PROJECT

SLATE Stack



ICL ECP EXASCALE COMPUTING PROJECT

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

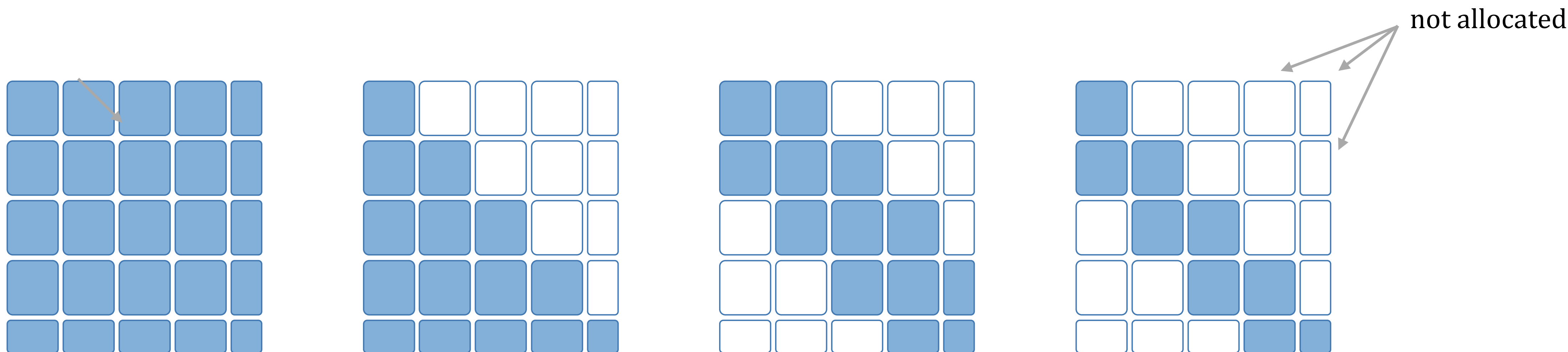
https://bitbucket.org/icl/lapackpp

- **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_;
```

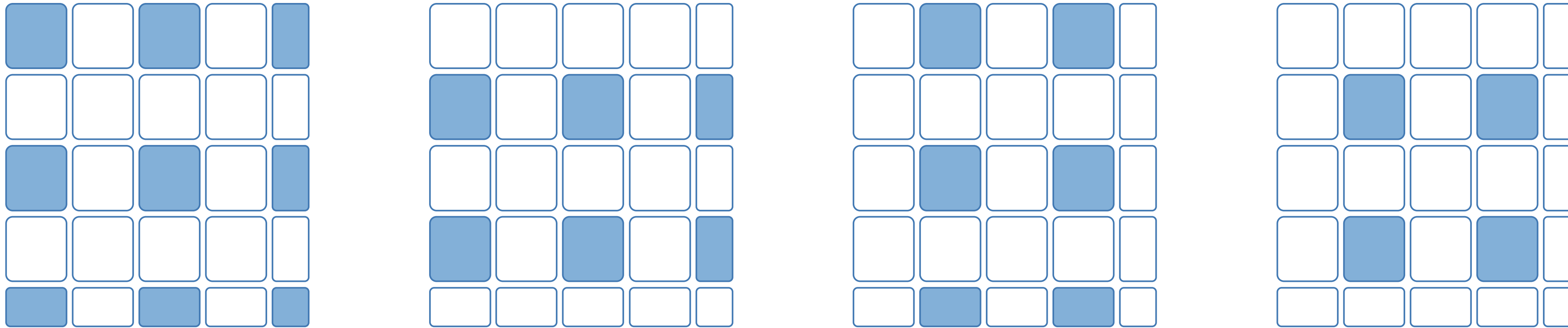
- collection of tiles
- **individually allocated**
- only allocate what is needed
- accommodates: symmetric, triangular, band, ...

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.

ICL
ECP EXASCALE COMPUTING PROJECT

SLATE Distributed Matrix



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

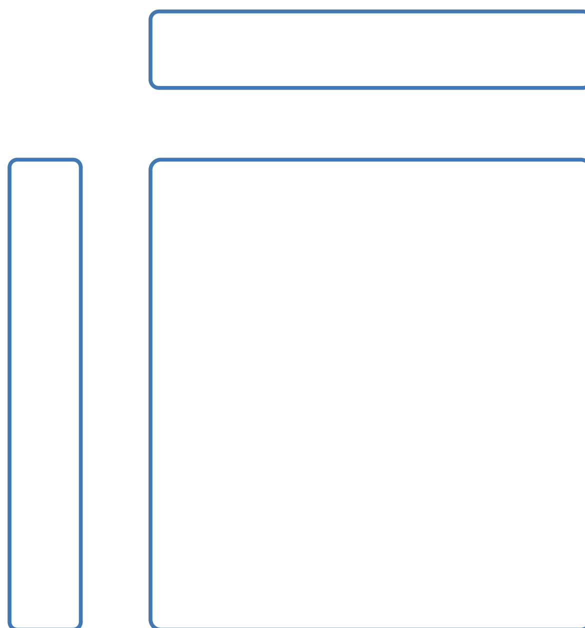
- distributed matrix
- global indexing of tiles
- only allocate the local part
- any distribution is possible (2D block cyclic by default)

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

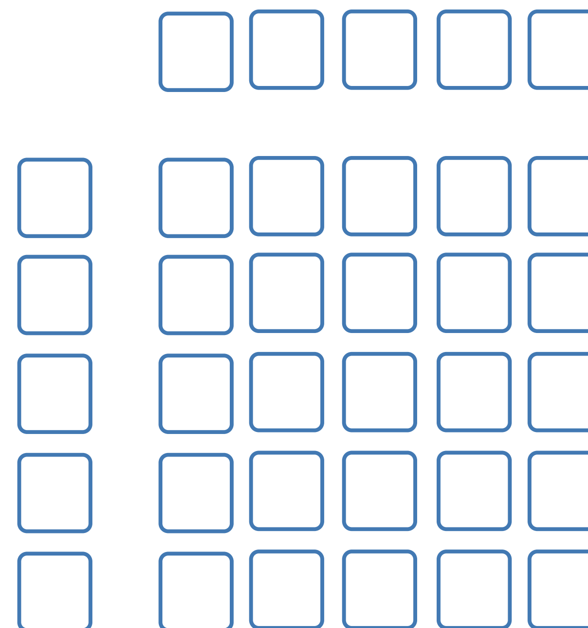
ICL
ECP EXASCALE COMPUTING PROJECT

GEMM Efficiency

LAPACK
MAGMA

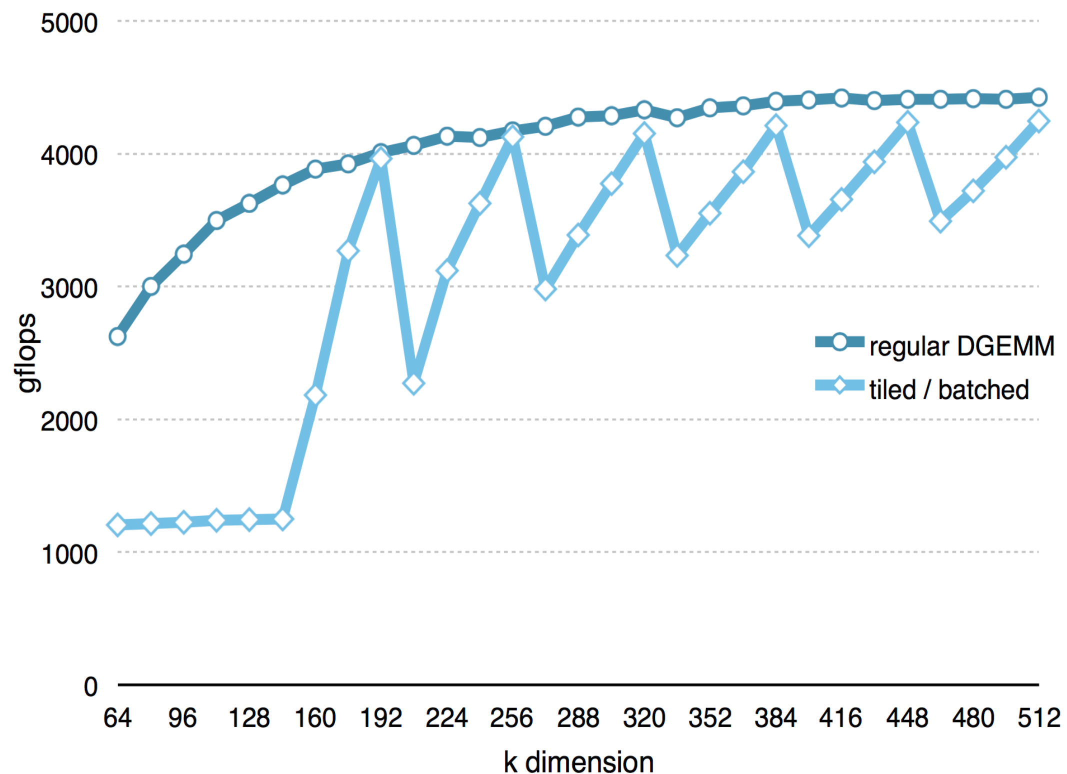


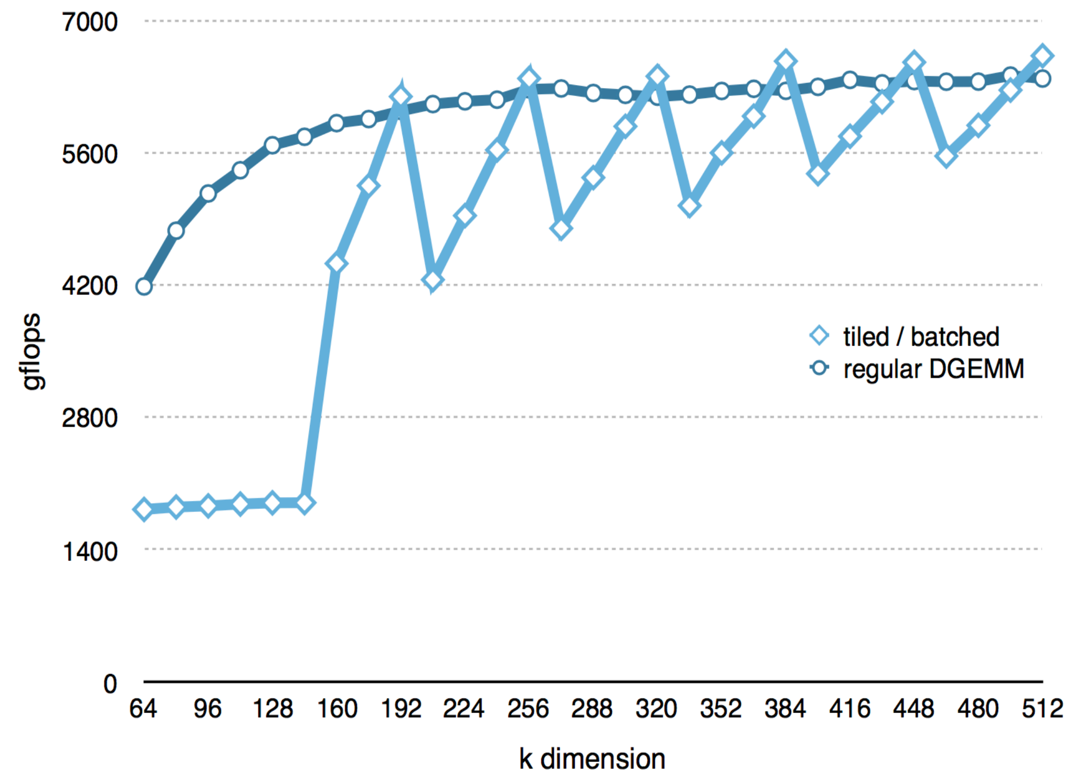
SLATE



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

GEMM Efficiency

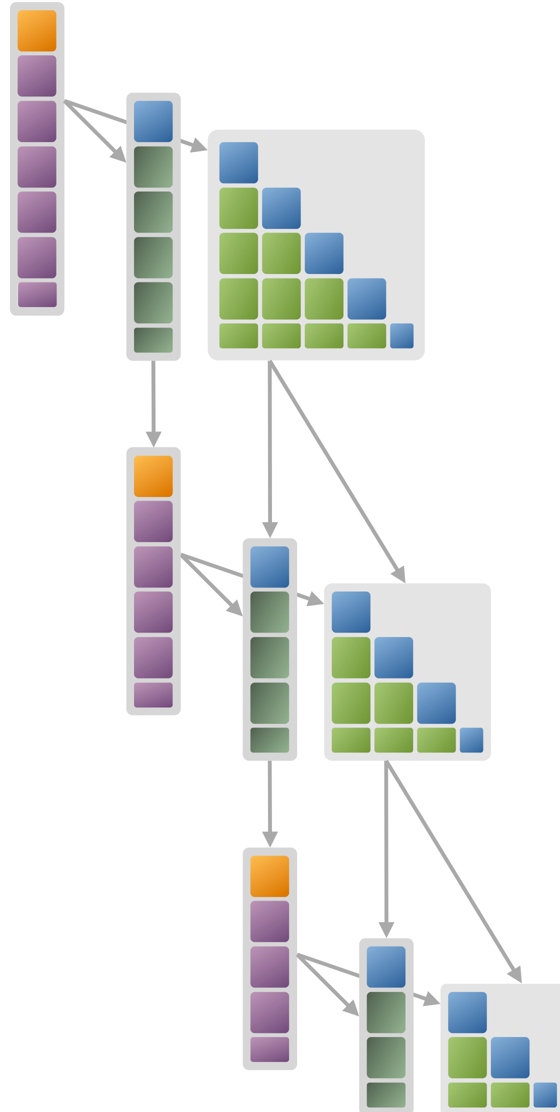




C = C – A × B with small k, i.e., the DGEMM called in LU factorization

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

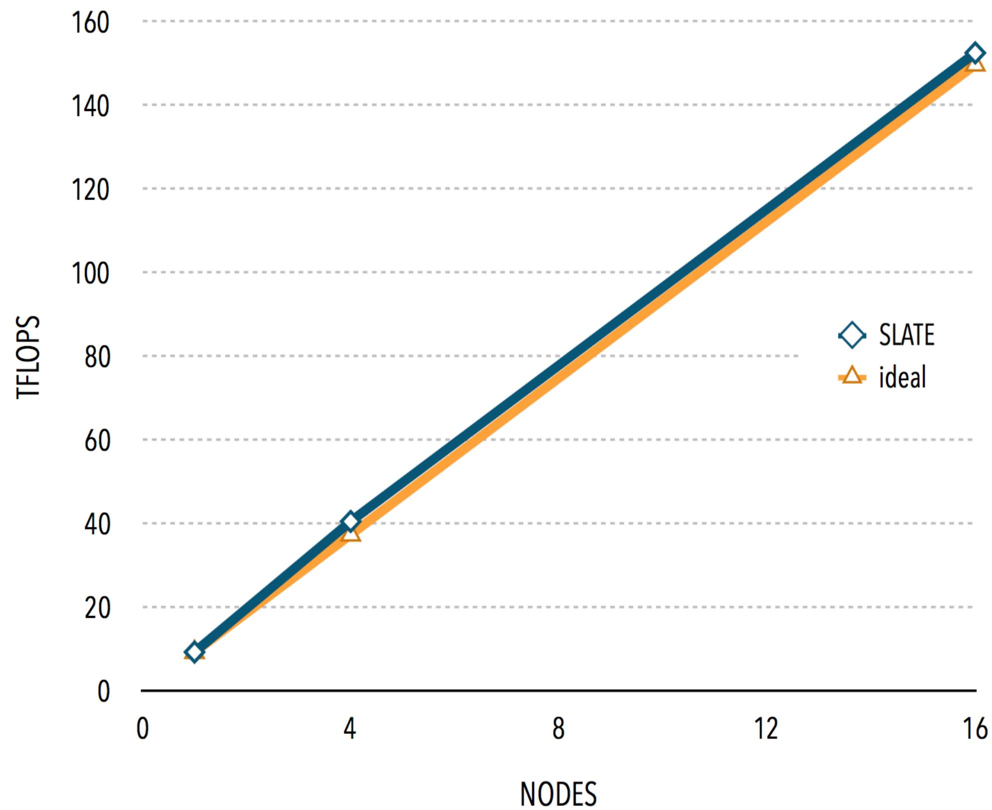
GEMM Scheduling



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

ICL
ECP EXASCALE COMPUTING PROJECT

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

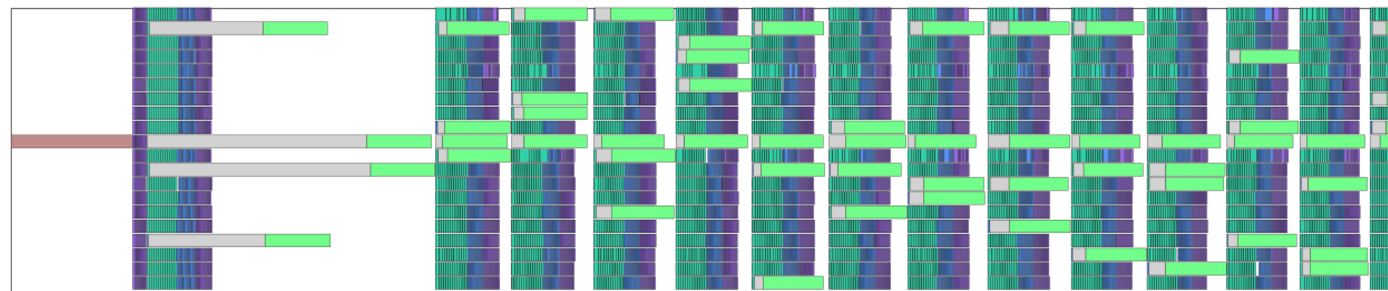
SummitDev @ OLCF

- 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.

ICL
ECP EXASCALE COMPUTING PROJECT

SLATE GPU Trace

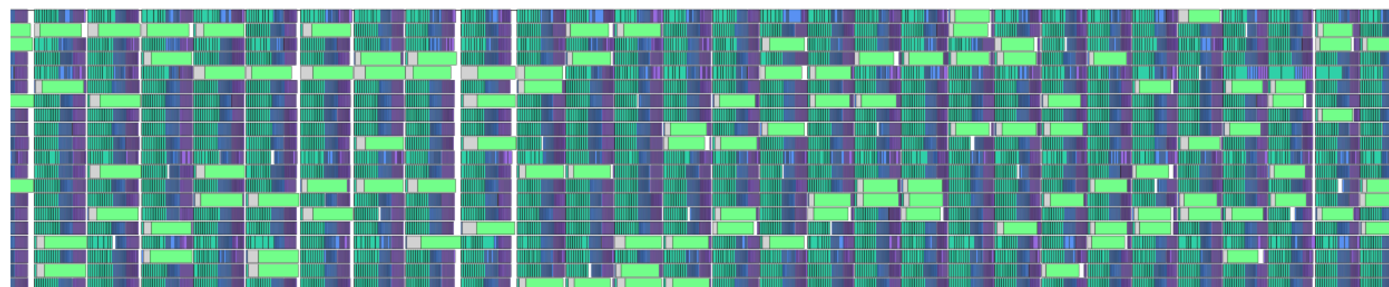


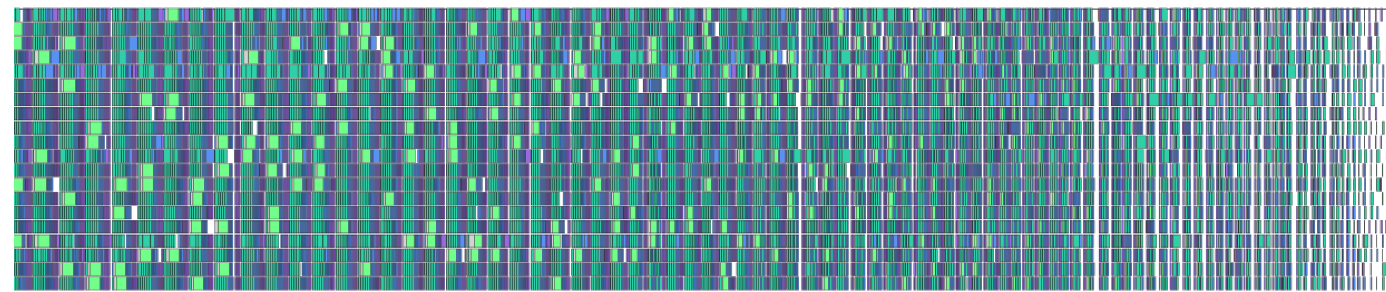
Cholesky factorization

20 cores + 4 GPUs

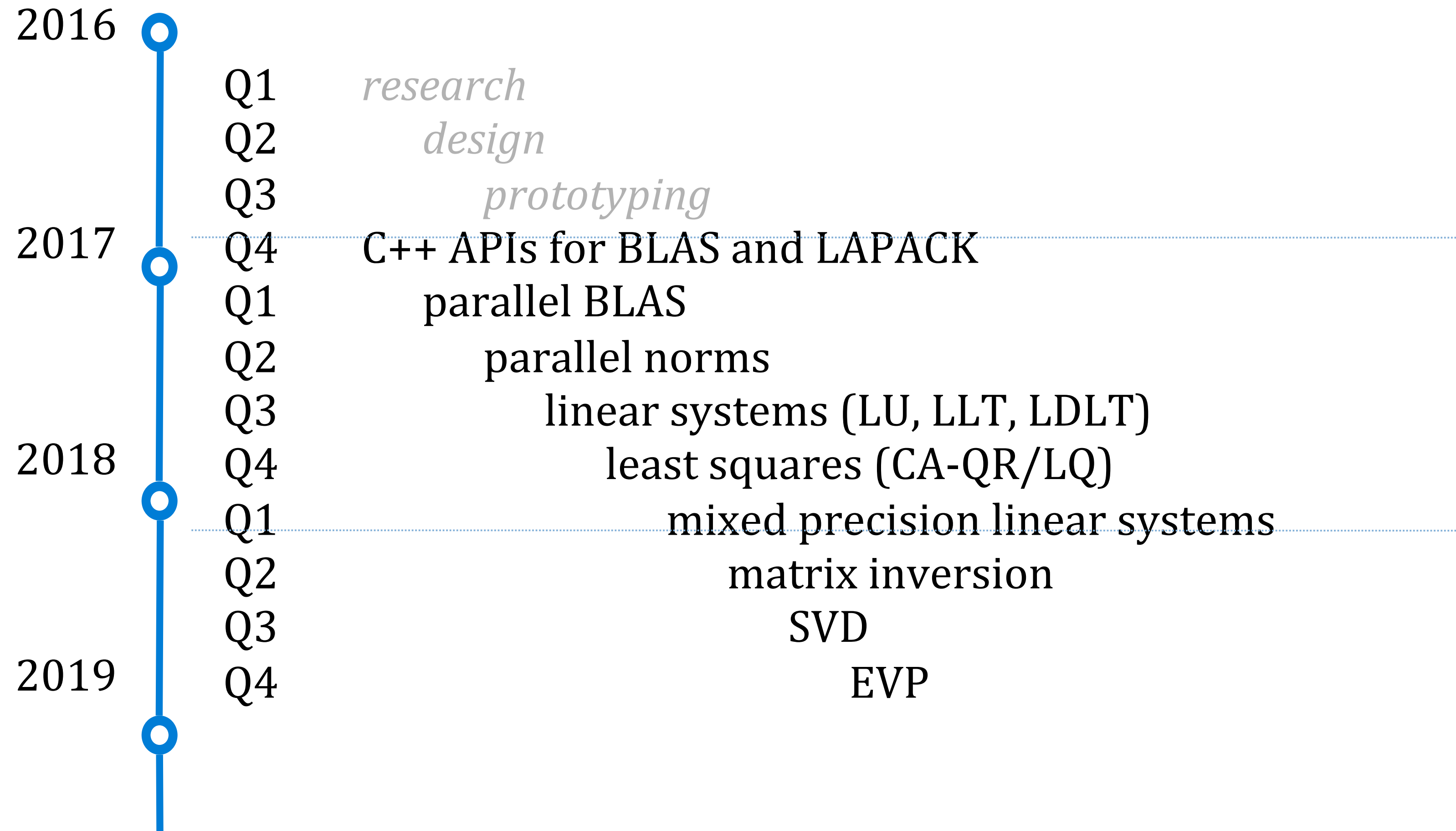
112 K × 112 K matrix

tile size of 512





SLATE Timeline



ICL
ECP EXASCALE COMPUTING PROJECT

Outline

Availability

Routines

Code

Testers

Methodology

ICL

ECP EXASCALE COMPUTING PROJECT

MAGMA 2.3

Availability

- http://icl.utk.edu/magma/ download, documentation, forum
- https://bitbucket.org/icl/magma Mercurial repo

Support

- Linux, macOS, Windows
- CUDA >= 5.0; recommend latest CUDA
- CUDA architecture >= 2.0 (Fermi, Kepler, Maxwell, Pascal, Volta)
- BLAS & LAPACK: Intel MKL, OpenBLAS, macOS Accelerate, ...

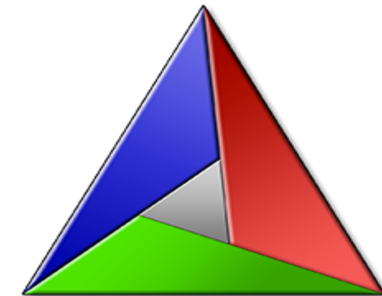
May be pre-installed on supercomputers

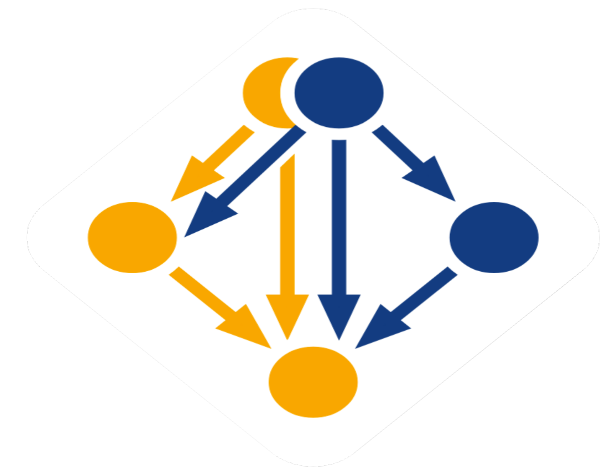
```
titan-ext1> module avail magma
----------- /sw/xk6/modulefiles -----------
magma/1.3                    magma/1.6.2(default)
```


ICL ECP EXASCALE COMPUTING PROJECT

Installation options

1. **Makefile**
   - Edit make.inc for compilers and flags (see make.inc examples)
   - `magma>` **`make && make install`**
2. **CMake**
   - `magma>` **`mkdir build && cd build`**
   - `magma/build>` **`cmake ..`** or **`ccmake ..`**
   - Adjust settings, esp. LAPACK_LIBRARIES and GPU_TARGET
   - `magma/build>` **`make && make install`**
3. **Spack**
   - Part of xSDK
   - **`spack install magma`**
   - Caveat: nvcc is picky about its host compiler (gcc, ...)





ICL
ECP EXASCALE COMPUTING PROJECT

Outline

Availability

Routines

Code

Testers

Methodology

MAGMA Overview

Hybrid LAPACK-style functions

- Matrix factorizations: LU, Cholesky, QR, eigenvalue, SVD, ...
- Solve linear systems and linear least squares, ...
- Nearly all are synchronous:
  return on CPU when computation is finished

GPU BLAS and auxiliary functions

- Matrix-vector multiply, matrix norms, transpose (in-place and out-of-place), ...
- Most are asynchronous:
  return immediately on CPU; computation proceeds on GPU

Wrappers around CUDA and cuBLAS

- BLAS routines (gemm, symm, symv, ...)
- Copy host ⇔ device, queue (stream) support, GPU malloc & free, ...

Naming example

magma_zgesv_gpu

magma_ or **magmablas_** prefix

Precision (1–2 characters)

- **S**ingle, **D**ouble, single **C**omplex, “**Z**” double complex, **I**nteger
  Mixed precision (**DS** and **ZC**)

Matrix type (2 characters)

- **GE**neral, **SY**mmetric, **HE**rmetian, **PO**sitive definite
  ORthogonal, **UN**itary, **TR**iangular

Operation (2–3+ characters)

- **SV** solve
  TRF triangular factorization
  EV eigenvalue problem
  GV generalized eigenvalue problem
  etc.

_gpu suffix for interface

Linear solvers

Solve linear system: $AX = B$

Solve linear least squares: **minimize** $\| AX - B \|_2$

Type	Routine	Mixed precision routine	Interface CPU	Interface GPU
General	dgesv	dsgesv	✓	✓
Positive definite	dposv	dsposv	✓	✓
Symmetric	dsysv		✓	
Hermitian	zhesv		✓	
Least squares	dgels		✓	✓

Selected routines; complete documentation at http://icl.utk.edu/magma/

ICL ECP EXASCALE COMPUTING PROJECT

Eigenvalue / singular value problems

Eigenvalue problem: $Ax = \lambda x$

Generalized eigenvalue problem: $Ax = \lambda Bx$ (and variants)

Singular value decomposition: $A = U \Sigma V^H$

Matrix type	Operation	Routine		Interface CPU	GPU
General	SVD	dgesvd,	dgesdd	✓	
General non-symmetric	Eigenvalue	dgeev		✓	
Symmetric	Eigenvalue	dsyevd /	zheevd	✓	✓
Symmetric	Generalized	dsygvd /	zhegvd	✓	

Additional variants; complete documentation at http://icl.utk.edu/magma/
Fastest are divide-and-conquer (gesdd, syevd) and 2-stage versions.

ICL ECP EXASCALE COMPUTING PROJECT

Computational routines

Computational routines solve one part of problem

Matrix type	Operation	Routine		Interface CPU	Interface GPU
General	LU	dgetrf		✓	✓
	Solve (given LU)	dgetrs			✓
	Inverse	dgetri			✓
SPD	Cholesky	dpotrf		✓	✓
	Solve (given LL^T)	dpotrs			✓
	Inverse	dpotri		✓	✓
General	QR	dgeqrf		✓	✓
	Generate Q	dorgqr /	zungqr	✓	✓
	Multiply by Q	dormqr /	zunmqr	✓	✓

Selected routines; complete documentation at http://icl.utk.edu/magma/

ICL ECP EXASCALE COMPUTING PROJECT

BLAS and auxiliary routines

Category	Operation	Routine (all GPU interface)
Level 1 BLAS	$y = \alpha x + y$	daxpy
	$r = x^{\mathrm{T}} y$	ddot
Level 2 BLAS	$y = \alpha A x + \beta y$, general A	dgemv
	$y = \alpha A x + \beta y$, symmetric A	dsymv
Level 3 BLAS	$C = \alpha AB + \beta C$	dgemm
	$C = \alpha AB + \beta C$, symmetric A	dsymm
	$C = \alpha AA^{T} + \beta C$, symmetric C	dsyrk
Auxiliary	$\|A\|_1$, $\|A\|_{\mathrm{inf}}$, $\|A\|_{\mathrm{fro}}$, $\|A\|_{\max}$	dlange (norm, general A)
		dlansy (norm, symmetric A)
	$B = A^{\mathrm{T}}$ (out-of-place)	dtranspose
	$A = A^{\mathrm{T}}$ (in-place, square)	dtranspose_inplace

Selected routines; complete documentation at http://icl.utk.edu/magma/

ICL
ECP EXASCALE COMPUTING PROJECT

Outline

Availability

Routines

Code

Testers

Methodology

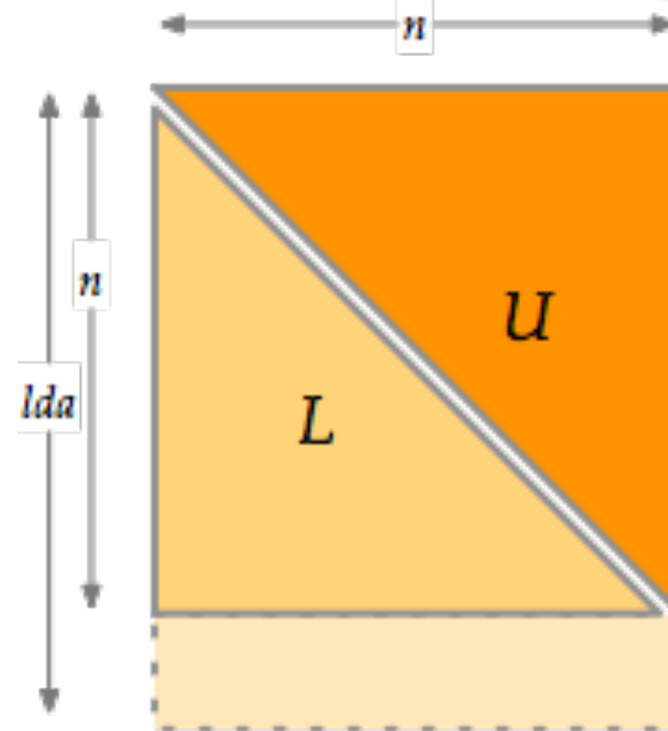
ICL
ECP EXASCALE COMPUTING PROJECT

Matrix layout

General m-by-n matrix, LAPACK column-major layout

Symmetric / Hermitian / Triangular n-by-n matrix

- uplo = Lower or Upper
- Entries in opposite triangle ignored



Simple example

Solve $AX = B$

- **D**ouble precision, **GE**neral matrix, **S**ol**V**e (**DGESV**)

Traditional LAPACK call

Complete codes available at
bit.ly/magma-tutorial

```
// tutorial0_lapack.cc
#include "magma_lapack.h"

int main( int argc, char** argv )
{
    int n = 100, nrhs = 10;
    int lda = n, ldx = n;
    double *A = new double[ lda*n ];
    double *X = new double[ ldx*nrhs ];
    int* ipiv = new int[ n ];

    // ... fill in A and X with your data
    // A[ i + j*lda ] = A_ij
    // X[ i + j*ldx ] = X_ij

    // solve AX = B where B is in X
    int info;
    lapackf77_dgesv( &n, &nrhs,
                     A, &lda, ipiv,
                     X, &ldx, &info );
    if (info != 0) {
        throw std::exception();
    }

    // ... use result in X

    delete[] A;
    delete[] X;
    delete[] ipiv;
}
```

Simple example

MAGMA CPU interface

- Input & output matrices in CPU host memory

Add MAGMA init & finalize

MAGMA call direct replacement for LAPACK call

```
// tutorial1_cpu_interface.cc
#include "magma_v2.h"

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

    int n = 100, nrhs = 10;
    int lda = n, ldx = n;
    double *A = new double[ lda*n ];
    double *X = new double[ ldx*nrhs ];
    int* ipiv = new int[ n ];

    // ... fill in A and X with your data
    // A[ i + j*lda ] = A_ij
    // X[ i + j*ldx ] = X_ij

    // solve AX = B where B is in X
    int info;
    magma_dgesv( n, nrhs,
                 A, lda, ipiv,
                 X, ldx, &info );
    if (info != 0) {
        throw std::exception();
    }

    // ... use result in X

    delete[] A;
    delete[] X;
    delete[] ipiv;

    magma_finalize();
}
```

Simple example

MAGMA GPU interface

- Add _**gpu** suffix
- Input & output matrices in GPU device memory (“d” prefix on variables)
- ipiv still in CPU memory
- Set GPU stride (ldda) to multiple of 32 for better performance
- roundup returns ceil(n / 32) * 32

MAGMA malloc & free

- Type-safe wrappers around cudaMalloc & cudaFree

```
// tutorial2_gpu_interface.cc
int main( int argc, char** argv )
{
    magma_init();

    int n = 100, nrhs = 10;
    int ldda = magma_roundup( n, 32 );
    int lddx = magma_roundup( n, 32 );
    int* ipiv = new int[ n ];

    double *dA, *dX;
    magma_dmalloc( &dA, ldda*n );
    magma_dmalloc( &dX, lddx*nrhs );
    assert( dA != nullptr );
    assert( dX != nullptr );

    // ... fill in dA and dX (on GPU)

    // solve AX = B where B is in X
    int info;
    magma_dgesv_gpu( n, nrhs,
                     dA, ldda, ipiv,
                     dX, lddx, &info );
    if (info != 0) {
        throw std::exception();
    }

    // ... use result in dX

    magma_free( dA );
    magma_free( dX );
    delete[] ipiv;

    magma_finalize();
}
```

BLAS example

Matrix multiply $C = -AB + C$

- **D**ouble-precision, **GE**neral **M**atrix **M**ultiply (**DGEMM**)

Asynchronous

- BLAS take queue and are async
- Return to CPU immediately
- Queue wraps CUDA stream and cuBLAS handle
- Can create queue from existing CUDA stream and cuBLAS handle, if required

```
// tutorial3_blas.cc
int main( int argc, char** argv )
{
    // ... setup matrices on GPU:
    // m-by-k matrix dA,
    // k-by-n matrix dB,
    // m-by-n matrix dC.

    int device;
    magma_queue_t queue;
    magma_getdevice( &device );
    magma_queue_create( device, &queue );

    // C = -A B + C
    magma_dgemm( MagmaNoTrans,
                 MagmaNoTrans, m, n, k,
                 -1.0, dA, ldda,
                       dB, lddb,
                  1.0, dC, lddc, queue );

    // ... do concurrent work on CPU

    // wait for gemm to finish
    magma_queue_sync( queue );

    // ... use result in dC

    magma_queue_destroy( queue );

    // ... cleanup
}
```

Copy example

Copy data host ⇔ device

- setmatrix (host to device)
- getmatrix (device to host)
- copymatrix (device to device)
- setvector (host to device)
- getvector (device to host)
- copyvector (device to device)

Default is synchronous

- Return when transfer is done

Strides (lda, ldda) can differ on CPU and GPU

- Set GPU stride (ldda) to multiple of 32 for better performance

```
// tutorial4_copy.cc
int main( int argc, char** argv )
{
    // ... setup A, X in CPU memory;
    // dA, dX in GPU device memory

    int device;
    magma_queue_t queue;
    magma_getdevice( &device );
    magma_queue_create( device, &queue );

    // copy A, X to dA, dX
    magma_dsetmatrix( n, n,
                       A, lda,
                      dA, ldda, queue );
    magma_dsetmatrix( n, nrhs,
                       X, ldx,
                      dX, lddx, queue );

    // ... solve AX = B

    // copy result dX to X
    magma_dgetmatrix( n, nrhs,
                      dX, lddx,
                       X, ldx, queue );

    // ... use result in X

    magma_queue_destroy( queue );

    // ... cleanup
}
```

Async copy

Add _async suffix

Use pinned CPU memory

- Page locked, so DMA can access it
- Better performance
- Required by CUDA for async behavior
- But pinned memory is limited resource, and expensive to allocate

Overlap:

- Sending data (host to device)
- Getting data (device to host)
- Host computation
- Device computation

```
// tutorial5_copy_async.cc
int main( int argc, char** argv )
{
    // ... setup dA, dX, queue

    // allocate A, X in pinned CPU memory
    double *A, *X;
    magma_dmalloc_pinned( &A, lda*n );
    magma_dmalloc_pinned( &X, ldx*nrhs );

    // ... fill in A and X

    // copy A, X to dA, dX, then wait
    magma_dsetmatrix_async( n, n,
                    A, lda, dA, ldda, queue );
    magma_dsetmatrix_async( n, nrhs,
                    X, ldx, dX, lddx, queue );
    magma_queue_sync( queue );

    // ... solve AX = B

    // copy result dX to X, then wait
    magma_dgetmatrix_async( n, nrhs,
                    dX, ldx, X, lddx, queue );
    magma_queue_sync( queue );

    // ... use result in X

    magma_free_pinned( A );
    magma_free_pinned( X );

    // ... cleanup
}
```

Outline

Availability

Routines

Code

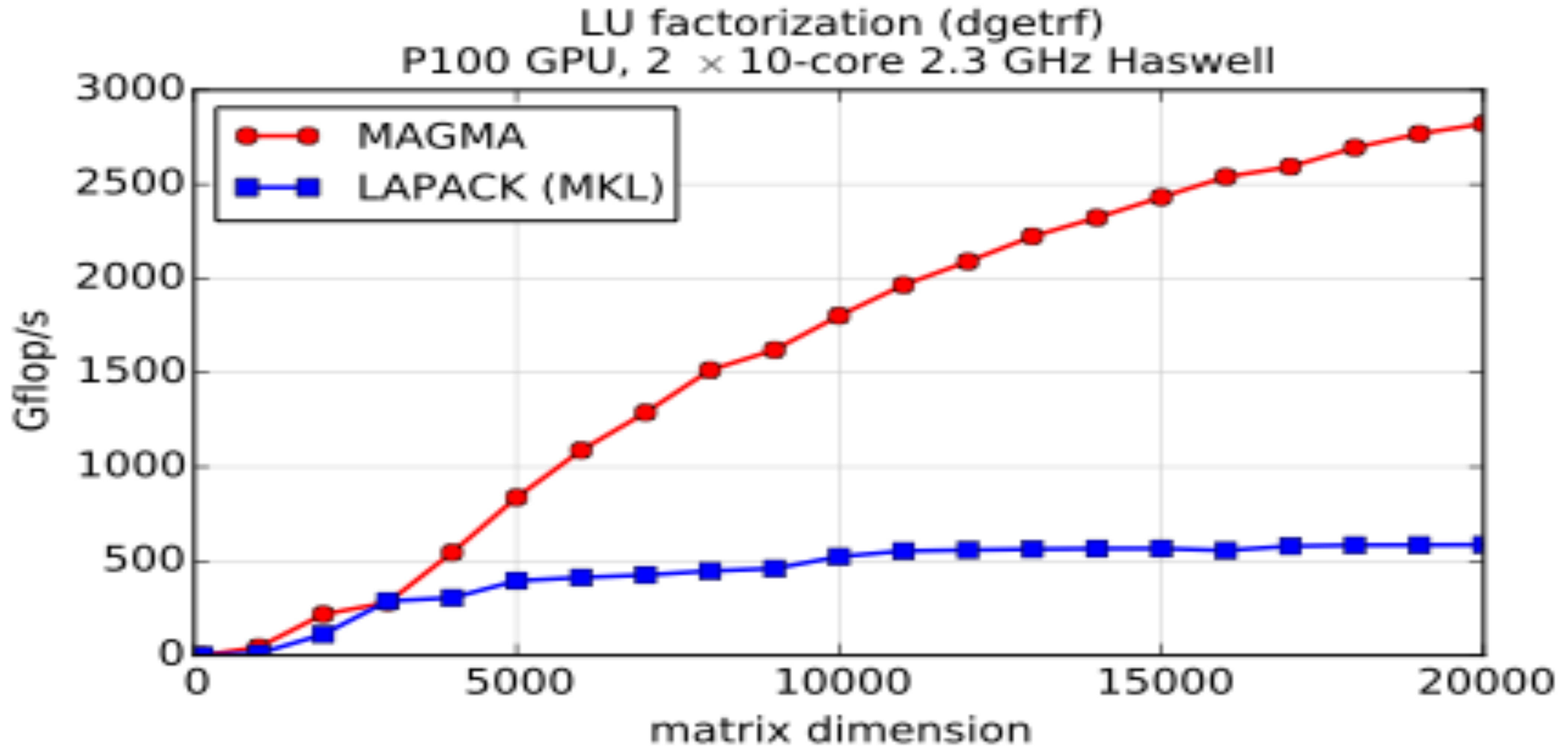
Testers

Methodology

ICL
ECP EXASCALE COMPUTING PROJECT

```
magma> cd testing
magma/testing> ./testing_dgetrf -n 123 -n 1000:20000:1000 --lapack --check
% MAGMA 2.2.0  compiled for CUDA capability >= 6.0, 32-bit magma_int_t, 64-bit pointer.
% CUDA runtime 8000, driver 9000. OpenMP threads 20. MKL 2017.0.1, MKL threads 20.
% device 0: Tesla P100-PCIE-16GB, 1328.5 MHz clock, 16276.2 MiB memory, capability 6.0

%   M     N    CPU Gflop/s (sec)   GPU Gflop/s (sec)   |PA-LU|/(N*|A|)
%========================================================================
  123   123      0.20 (   0.01)      0.40 (   0.00)   3.59e-18   ok       # warmup run
 1000  1000     10.40 (   0.06)     43.50 (   0.02)   2.76e-18   ok
 2000  2000    111.64 (   0.05)    218.26 (   0.02)   2.68e-18   ok
 3000  3000    288.38 (   0.06)    280.28 (   0.06)   2.65e-18   ok
 4000  4000    305.58 (   0.14)    545.90 (   0.08)   2.81e-18   ok
 5000  5000    396.16 (   0.21)    838.09 (   0.10)   2.71e-18   ok
 6000  6000    413.37 (   0.35)   1088.14 (   0.13)   2.71e-18   ok
 7000  7000    426.71 (   0.54)   1288.60 (   0.18)   2.67e-18   ok
 8000  8000    447.85 (   0.76)   1514.43 (   0.23)   2.66e-18   ok
 9000  9000    461.05 (   1.05)   1621.29 (   0.30)   2.87e-18   ok
10000 10000    524.06 (   1.27)   1802.39 (   0.37)   2.84e-18   ok
11000 11000    554.16 (   1.60)   1965.85 (   0.45)   2.84e-18   ok
12000 12000    559.33 (   2.06)   2090.42 (   0.55)   2.82e-18   ok
13000 13000    563.56 (   2.60)   2223.62 (   0.66)   2.80e-18   ok
14000 14000    566.58 (   3.23)   2323.04 (   0.79)   2.78e-18   ok
15000 15000    567.17 (   3.97)   2431.59 (   0.93)   2.77e-18   ok
16000 16000    556.86 (   4.90)   2539.66 (   1.08)   2.79e-18   ok
17000 17000    579.82 (   5.65)   2593.40 (   1.26)   2.75e-18   ok
18000 18000    584.93 (   6.65)   2694.57 (   1.44)   2.76e-18   ok
19000 19000    585.78 (   7.81)   2768.67 (   1.65)   2.75e-18   ok
20000 20000    587.08 (   9.08)   2821.48 (   1.89)   2.74e-18   ok
```

Testers: LU factorization (dgetrf)

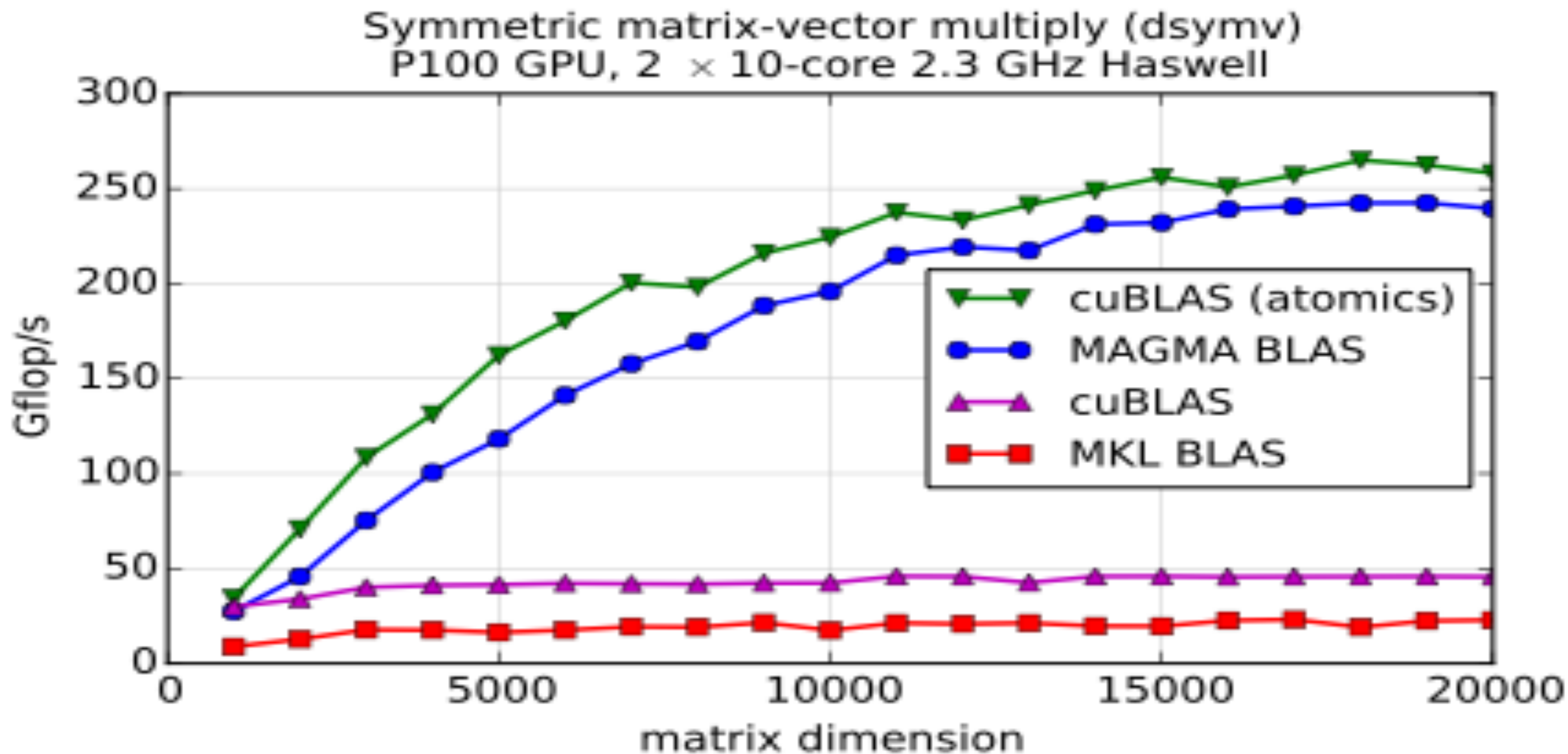


ICL

ECP EXASCALE COMPUTING PROJECT

```
# (abbreviated output)
magma> cd testing
magma/testing> ./testing_dsymv -n 123 -n 1000:20000:1000 --lapack --check
% MAGMA 2.2.0  compiled for CUDA capability >= 6.0, 32-bit magma_int_t, 64-bit pointer.
% CUDA runtime 8000, driver 9000. OpenMP threads 20. MKL 2017.0.1, MKL threads 20.
% device 0: Tesla P100-PCIE-16GB, 1328.5 MHz clock, 16276.2 MiB memory, capability 6.0

% uplo = Lower
%   N      MAGMA    Atomics     cuBLAS       CPU    error
%         Gflop/s   Gflop/s    Gflop/s    Gflop/s
%=====================================================
  123       0.76       0.76       0.51       0.58    ok     # warmup run
 1000      27.44      34.41      29.88       8.86    ok
 2000      45.74      70.83      33.91      12.78    ok
 3000      75.30     108.51      40.09      17.76    ok
 4000     100.64     131.23      41.13      17.57    ok
 5000     118.17     162.35      41.46      16.33    ok
 6000     141.21     180.43      42.16      17.55    ok
 7000     157.81     200.44      41.94      19.32    ok
 8000     169.54     198.21      41.78      19.12    ok
 9000     188.40     216.07      42.28      21.50    ok
10000     195.92     224.50      42.36      17.44    ok
11000     214.93     237.51      45.91      21.30    ok
12000     219.33     233.44      45.76      20.81    ok
13000     217.52     241.45      42.49      21.29    ok
14000     231.26     249.06      45.84      19.71    ok
15000     232.12     255.98      45.87      19.60    ok
16000     239.26     250.89      45.58      22.61    ok
17000     240.74     257.13      45.69      23.15    ok
18000     242.45     265.05      45.75      19.09    ok
19000     242.53     262.48      45.81      22.42    ok
20000     239.53     258.24      45.63      22.83    ok
```

Testers: symmetric matrix-vector multiply (dsymv)



ICL

ECP EXASCALE COMPUTING PROJECT

Test everything: run_tests.py

Python script to run:

- All testers
- All possible options (left/right, lower/upper, ...)
- Various size ranges (small, medium, large; square, tall, wide)

Occasionally, tests fail innocuously

- E.g., error = 1.1e-15 > tol = 1e-15

Some experimental routines are known to fail

- E.g., gegqr_gpu, geqr2x_gpu
- See magma/BUGS.txt

Running ALL tests can take > 24 hours

ICL ECP EXASCALE COMPUTING PROJECT

Test everything: run_tests.py

```
magma/testing> python ./run_tests.py *trsm --xsmall --small > trsm.txt
testing_strsm -SL -L -DN -c       ok  # left, lower, non-unit, [no-trans]
testing_dtrsm -SL -L -DN -c       ok
testing_ctrsm -SL -L -DN -c       ok
testing_ztrsm -SL -L -DN -c       ok
testing_strsm -SL -L -DU -c       ok  # left, lower, unit, [no-trans]
testing_dtrsm -SL -L -DU -c       ok
testing_ctrsm -SL -L -DU -c       ok
testing_ztrsm -SL -L -DU -c       ok
testing_strsm -SL -L -C -DN -c    ok  # left, lower, non-unit, conj-trans
testing_dtrsm -SL -L -C -DN -c    ok
testing_ctrsm -SL -L -C -DN -c    ok
testing_ztrsm -SL -L -C -DN -c    ok
...
testing_strsm -SR -U -T -DU -c    ok  # right, upper, unit, trans
testing_dtrsm -SR -U -T -DU -c    ok
testing_ctrsm -SR -U -T -DU -c    ok
testing_ztrsm -SR -U -T -DU -c    ok

****************************************************************************************
summary
****************************************************************************************
 6240 tests in 192 commands passed
   96 tests failed accuracy test
    0 errors detected (crashes, CUDA errors, etc.)
routines with failures:
    testing_ctrsm --ngpu 2 -SL -L -C -DN -c
    testing_ctrsm --ngpu 2 -SL -L -C -DU -c
...
```


ICL
ECP EXASCALE COMPUTING PROJECT

Outline

Availability

Routines

Code

Testers

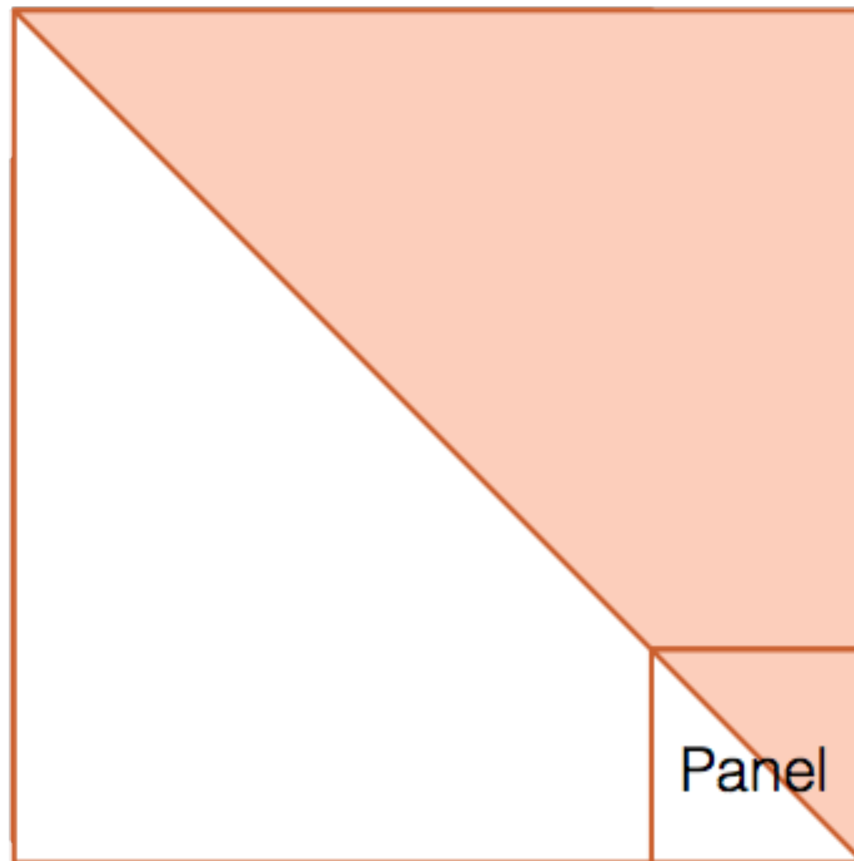
Methodology

One-sided factorizations

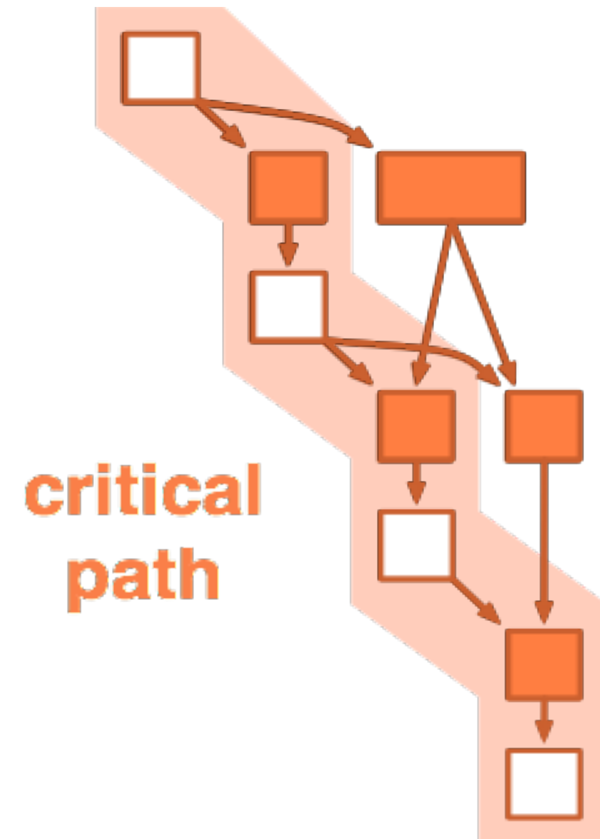
LU, Cholesky, QR factorizations for solving linear systems

Level 2 BLAS on CPU

Level 3 BLAS on GPU



DAG

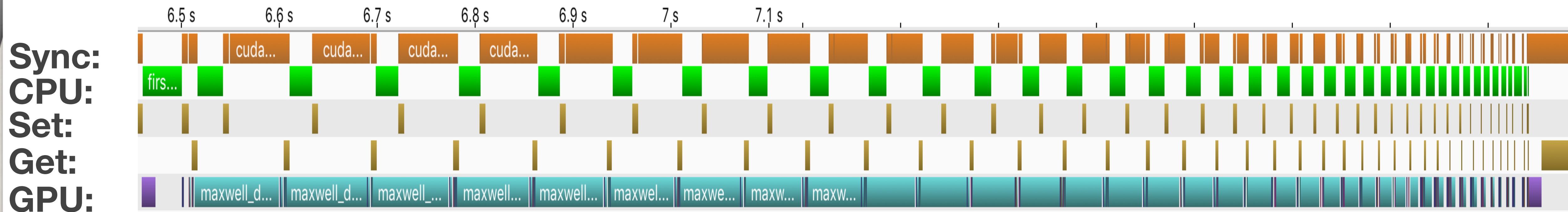


ICL
ECP EXASCALE COMPUTING PROJECT

Execution trace

Panels on CPU **(green)** and **set/get communication (brown)** overlapped with **trailing matrix updates (teal)** on GPU

Goal to keep GPU busy all the time; CPU may idle



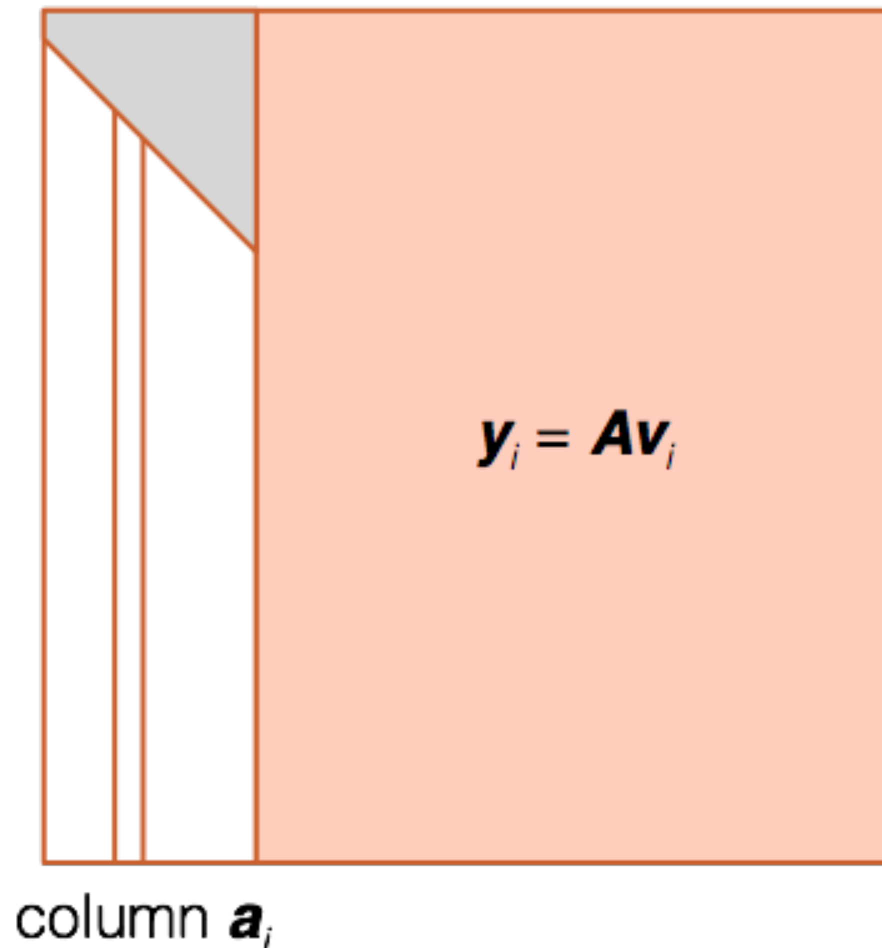
LU factorization (dgetrf), n = 20000
P100 GPU, 2 × 10-core 2.3 GHz Haswell

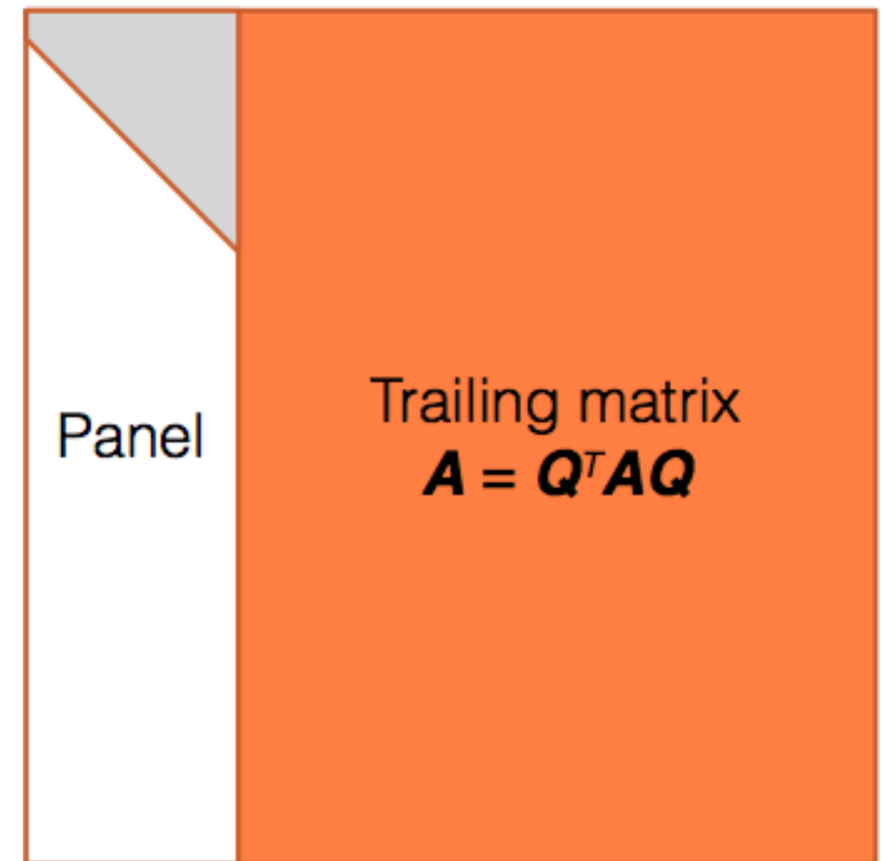
Optimization: for LU, we transpose matrix on GPU so row-swaps are fast

ICL
ECP EXASCALE COMPUTING PROJECT

Two-sided factorizations

Hessenberg, tridiagonal, bidiagonal factorizations for eigenvalue and singular value problems



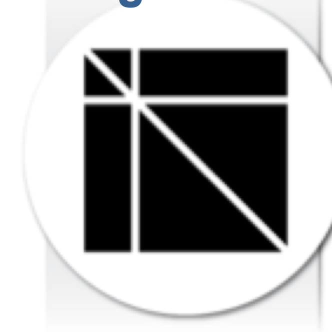


ICL
ECP EXASCALE COMPUTING PROJECT

Numerical Linear Algebra (NLA) in Applications

- **For big NLA problems**
  (BLAS, convolutions, SVD, linear system solvers, etc.)

Large matrices



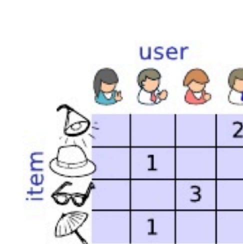
In contemporary libraries:
BLAS
LAPACK
ScaLAPACK

MAGMA (for GPUs)

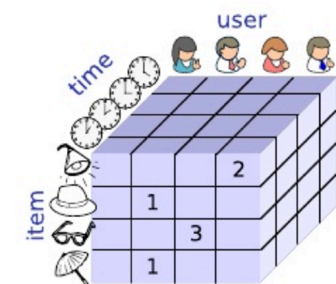
- **Numerous important applications need NLA for small problems**

  - **Machine learning / DNNs**
  - **Data mining / analytics**
  - High-order FEM,
  - Graph analysis,
  - Neuroscience,
  - Astrophysics,
  - Quantum chemistry,
  - Signal processing, and more

Where data can be multidimensional / relational



matrix



3 order tensor

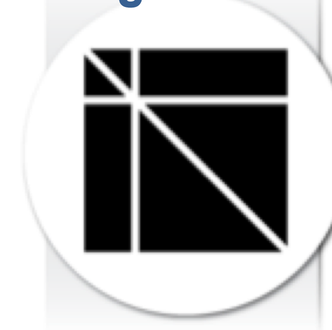
Numerical Linear Algebra (NLA) in Applications

- **For big NLA problems**
  (BLAS, convolutions, SVD, linear system solvers, etc.)

- **Adding in MAGMA application backends for small problems**

  - Machine learning / DNNs
  - Data mining / analytics
  - High-order FEM,
  - Graph analysis,
  - Neuroscience,
  - Astrophysics,
  - Quantum chemistry,
  - Signal processing, and more

Large matrices



In contemporary libraries:

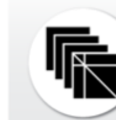
BLAS

LAPACK

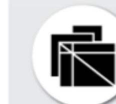
ScaLAPACK

MAGMA (for GPUs)

Small matrices / tensors



Fixed-size batches



Variable-size batches

Dynamic batches

Tensors

What about DLA on many small matrices?

Batched routines

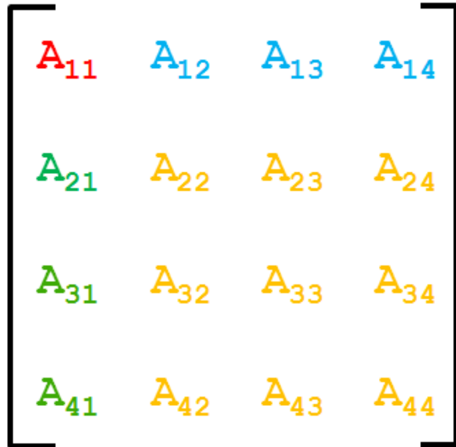
- MAGMA provide a set of GPU only routines for batched computation

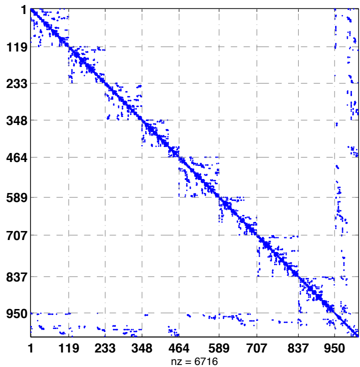
MAGMA Batched Computations

Need of **Batched** routines for **Numerical LA**

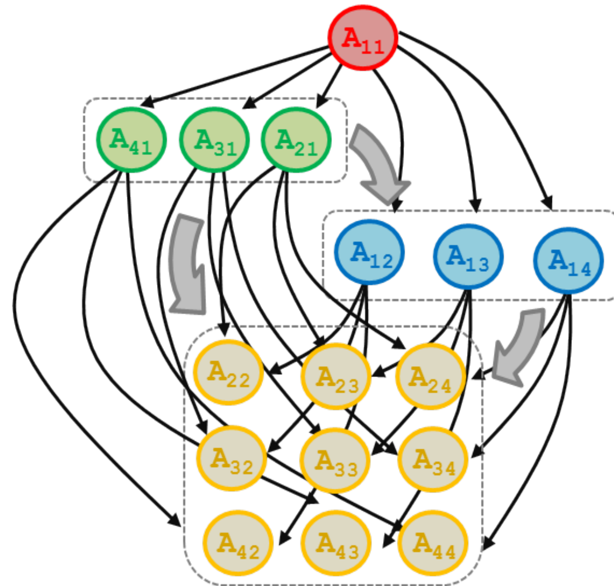
[e.g., sparse direct multifrontal methods, preconditioners for sparse iterative methods, tiled algorithms in dense linear algebra, etc.;]
[collaboration with Tim Davis at al., Texas A&M University]

Sparse / Dense Matrix System





DAG-based factorization



To capture main LA patterns needed in a numerical library for Batched LA

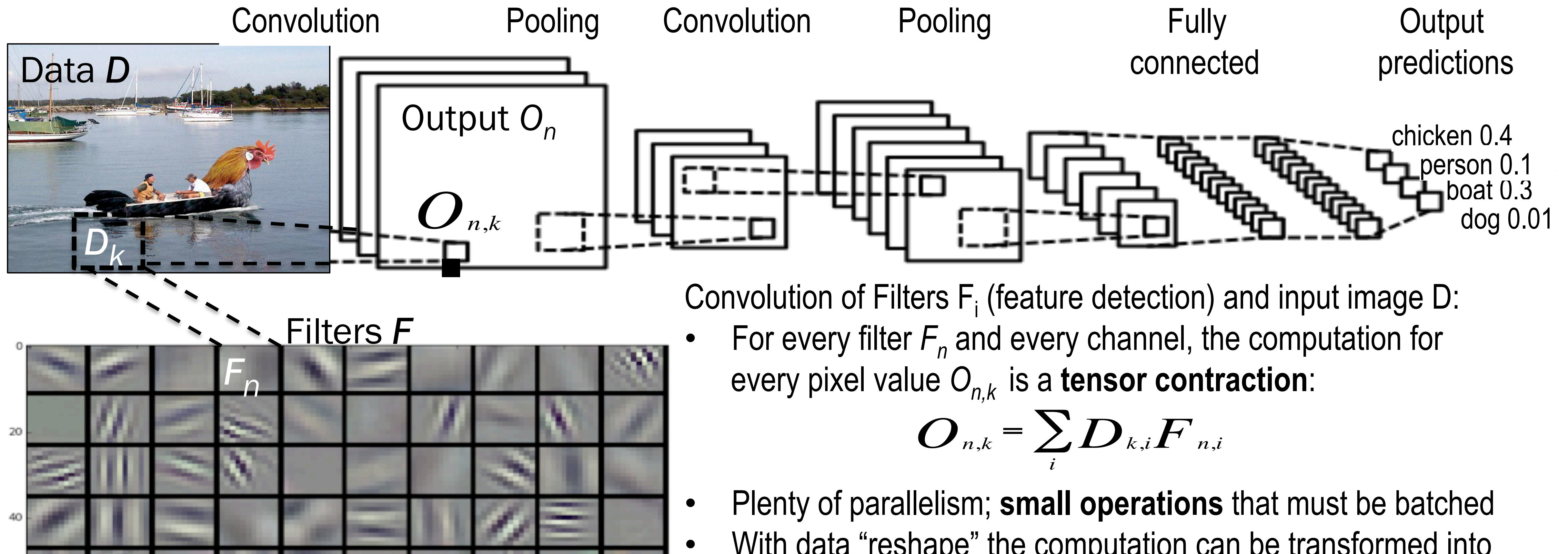
- **LU, QR,** or **Cholesky** on small diagonal matrices
- **TRSMs**, **QRs**, or **LUs**
- **TRSMs**, **TRMMs**
- **Updates (Schur complement) GEMMs, SYRKs, TRMMs**

- Example matrix from Quantum chromodynamics
- Reordered and ready for sparse direct multifrontal solver
- Diagonal blocks can be handled in parallel through batched LU, QR, or Cholesky factorizations

MAGMA Batched: Backend for DNN and Data Analytics

Support for various **Batched and/or Tensor contraction** routines

e.g., Convolutional Neural Networks (CNNs) used in computer vision
Key computation is convolution of Filter Fi (feature detector) and input image D (data):



Convolution of Filters F_i (feature detection) and input image D:

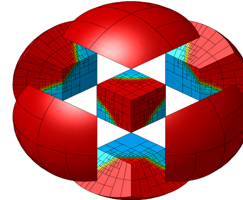
- For every filter F_n and every channel, the computation for every pixel value $O_{n,k}$ is a **tensor contraction**:

$$O_{n,k} = \sum_i D_{k,i} F_{n,i}$$

- Plenty of parallelism; **small operations** that must be batched
- With data “reshape” the computation can be transformed into a **batched GEMM** (for efficiency; among other approaches)

MAGMA Batched: Tensor contractions for high-order FEM

Lagrangian Hydrodynamics in the BLAST code[1]



On semi-discrete level our method can be written as

$$\textbf{Momentum Conservation:} \quad \frac{d\mathbf{v}}{dt} = -\mathbf{M}_\mathbf{v}^{-1}\mathbf{F}\cdot\mathbf{1}$$

$$\textbf{Energy Conservation:} \quad \frac{de}{dt} = \mathbf{M}_\mathbf{e}^{-1}\mathbf{F}^\mathbf{T}\cdot\mathbf{v}$$

$$\textbf{Equation of Motion:} \quad \frac{d\mathbf{x}}{dt} = \mathbf{v}$$

where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M_v}$ and $\mathbf{M_e}$ are independent of time velocity and energy mass matrices; and $\mathbf{F}$ is the generalized corner force matrix depending on $(\mathbf{v}, \mathbf{e}, \mathbf{x})$ that needs to be evaluated at every time step.

[1] V. Dobrev, T.Kolev, R.Rieben. *High order curvilinear finite element methods for Lagrangian hydrodynamics*. SIAM J.Sci.Comp.34(5), B606–B641. (36 pages)

- Contractions can often be implemented as index reordering plus **batched GEMM** (and hence, be highly efficient)

stored components	FLOPs for assembly	amount of storage	FLOPs for matvec	numerical kernels
full assembly				
M	$O(p^{3d})$	$O(p^{2d})$	$O(p^{2d})$	$B, D \mapsto B^TDB,\ x \mapsto Mx$
decomposed evaluation				
B, D	$O(p^{2d})$	$O(p^{2d})$	$O(p^{2d})$	$x \mapsto Bx,\ x \mapsto B^Tx,\ x \mapsto Dx$
near-optimal assembly – equations (1) and (2)				
$M_{i_1,\cdots,j_d}$	$O(p^{2d+1})$	$O(p^{2d})$	$O(p^{2d})$	$A_{i_1,k_2,j_1} = \sum_{k_1} B^{1d}_{k_1,i_1} B^{1d}_{k_1,j_1} D_{k_1,k_2}$ (1a) $A_{i_1,i_2,,j_1,j_2} = \sum_{k_2} B^{1d}_{k_2,i_2} B^{1d}_{k_2,j_2} C_{i_1,k_2,j_1}$ (1b) $A_{i_1,k_2,k_3,j_1} = \sum_{k_1} B^{1d}_{k_1,i_1} B^{1d}_{k_1,j_1} D_{k_1,k_2,k_3}$ (2a) $A_{i_1,i_2,k_3,j_1,j_2} = \sum_{k_2} B^{1d}_{k_2,i_2} B^{1d}_{k_2,j_2} C_{i_1,k_2,k_3,j_1}$ (2b) $A_{i_1,i_2,i_3,j_1,j_2,j_3} = \sum_{k_3} B^{1d}_{k_3,i_3} B^{1d}_{k_3,j_3} C_{i_1,i_2,k_3,j_1,j_2}$ (2c)
near-optimal evaluation (partial assembly) – equations (3) and (4)				
B^{1d}, D	$O(p^d)$	$O(p^d)$	$O(p^{d+1})$	$A_{j_1,k_2} = \sum_{j_2} B^{1d}_{k_2,j_2} V_{j_1,j_2}$ (3a) $A_{k_1,k_2} = \sum_{j_1} B^{1d}_{k_1,j_1} C_{j_1,k_2}$ (3b) $A_{k_1,i_2} = \sum_{k_2} B^{1d}_{k_2,i_2} C_{k_1,k_2}$ (3c) $A_{i_1,i_2} = \sum_{k_1} B^{1d}_{k_1,i_1} C_{k_1,i_2}$ (3d) $A_{j_1,j_2,k_3} = \sum_{j_3} B^{1d}_{k_3,j_3} V_{j_1,j_2,j_3}$ (4a) $A_{j_1,k_2,k_3} = \sum_{j_2} B^{1d}_{k_2,j_2} C_{j_1,j_2,k_3}$ (4b) $A_{k_1,k_2,k_3} = \sum_{j_1} B^{1d}_{k_1,j_1} C_{j_1,k_2,k_3}$ (4c) $A_{k_1,k_2,i_3} = \sum_{k_3} B^{1d}_{k_3,i_3} C_{k_1,k_2,k_3}$ (4d) $A_{k_1,i_2,i_3} = \sum_{k_2} B^{1d}_{k_2,i_2} C_{k_1,k_2,i_3}$ (4e) $A_{i_1,i_2,i_3} = \sum_{k_1} B^{1d}_{k_1,i_1} C_{k_1,i_2,i_3}$ (4f)
matrix-free evaluation				
none	none	none	$O(p^{d+1})$	evaluating entries of B^{1d}, D, (3a)–(4f) sums

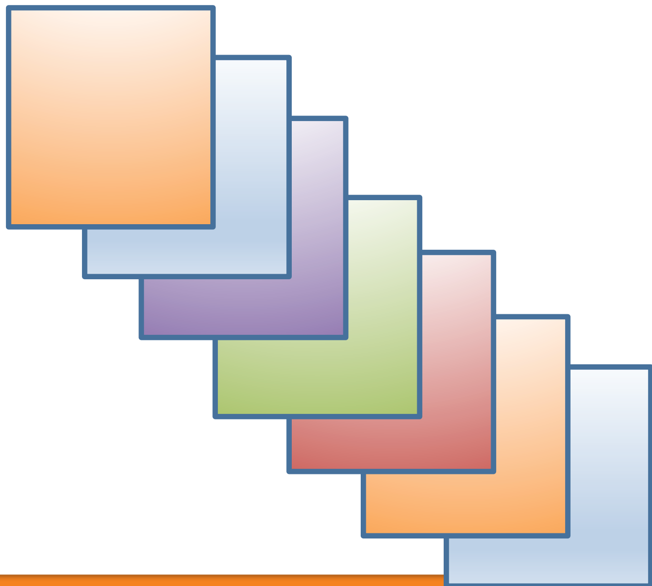
A. Abdelfattah, M. Baboulin, V. Dobrev, J. Dongarra, C. Earl, J. Falcou, A. Haidar, I. Karlin, Tz. Kolev, I. Masliah, S. Tomov,
High-Performance Tensor Contractions for GPUs, ICCS 2016, San Diego, CA, June 6—8, 2016.

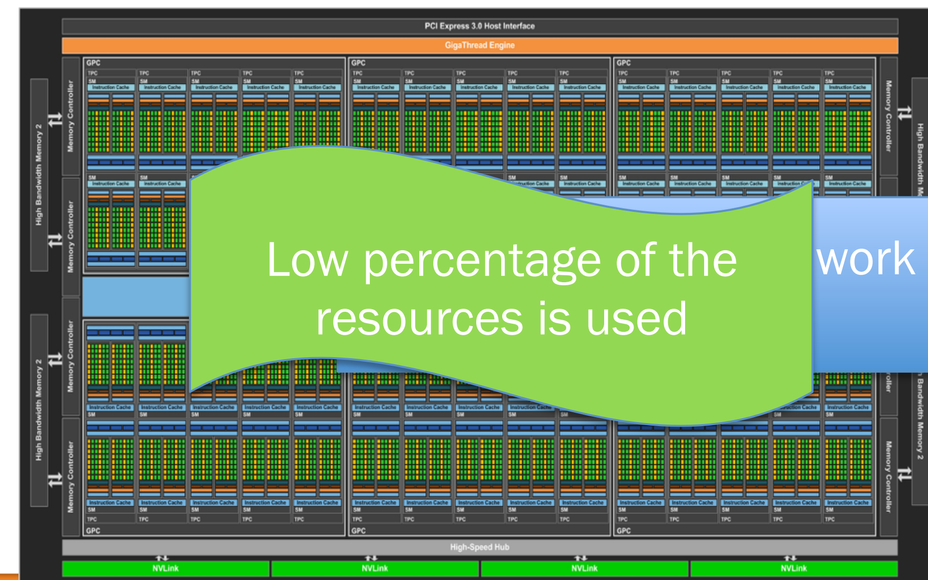
MAGMA Batched Computations

1. Non-batched computation

- **loop over the matrices one by one** and compute using multithread (note that, since matrices are of small sizes there is not enough work for all the cores). So we expect low performance as well as threads contention might also affect the performance

```
for (i=0; i<batchcount; i++)
   dgemm(…)
```





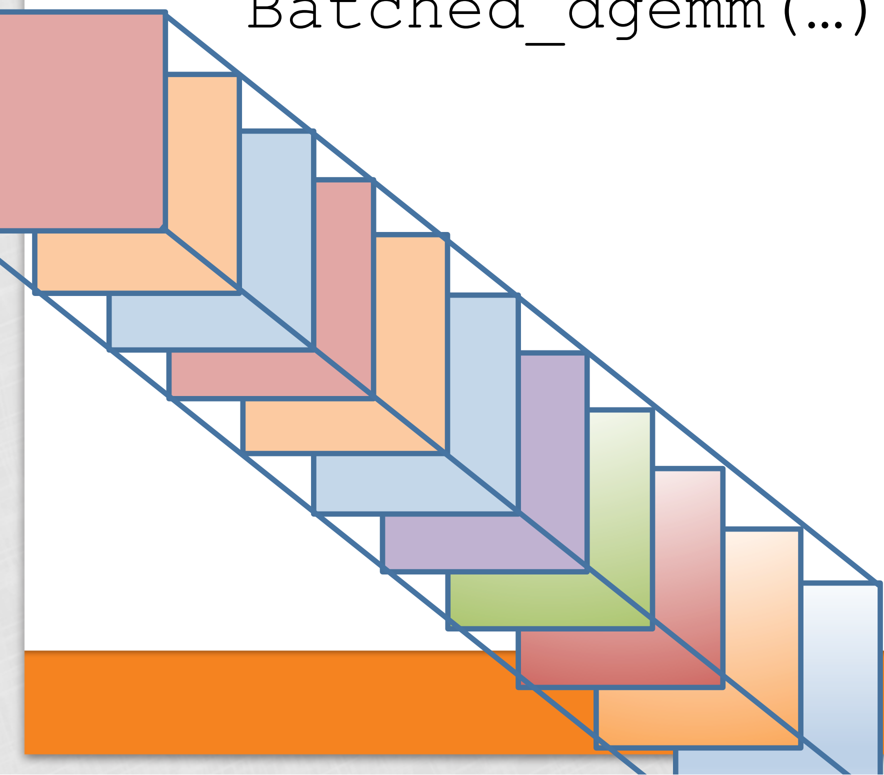
MAGMA Batched Computations

1. Batched computation

Distribute all the matrices over the available resources by assigning a matrix to each group of core/TB to operate on it independently

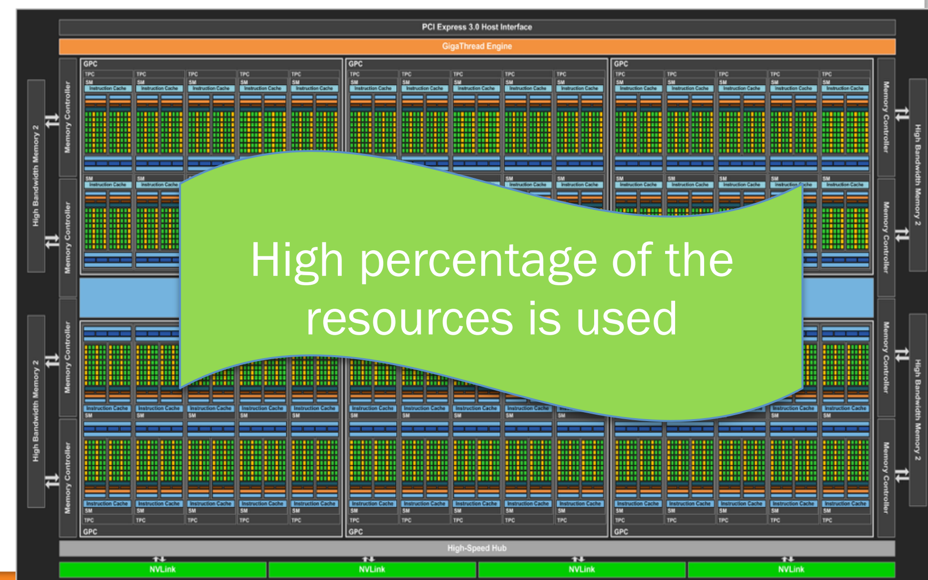
- For very small matrices, assign a matrix/core (CPU) or per TB for GPU
- For medium size a matrix go to a team of cores (CPU) or many TB's (GPU)
- For large size switch to multithreads classical 1 matrix per round.

```
Batched_dgemm(…)
```



Tasks manager dispatcher

Based on the kernel design that decide the number of TB or threads (GPU/CPU) and through the Nvidia/ OpenMP scheduler



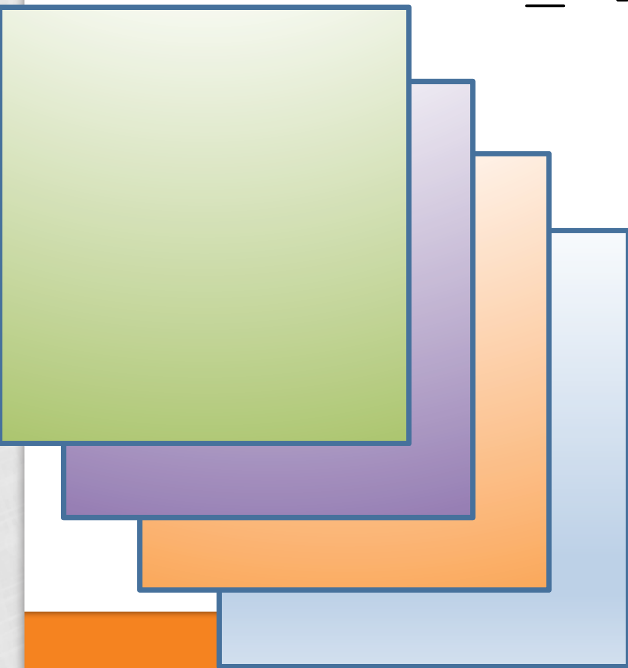
MAGMA Batched Computations

1. Batched computation

Distribute all the matrices over the available resources by assigning a matrix to each group of core/TB to operate on it independently

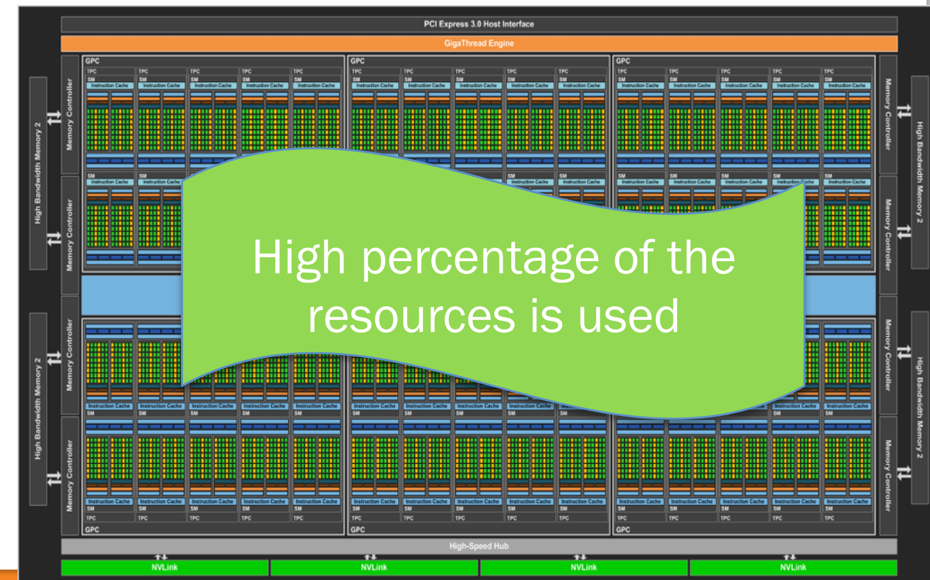
- For very small matrices, assign a matrix/core (CPU) or per TB for GPU
- For medium size a matrix go to a team of cores (CPU) or many TB's (GPU)
- For large size switch to multithreads classical 1 matrix per round.

```
Batched_dgemm(…)
```



Tasks manager dispatcher

Based on the kernel design that decide the number of TB or threads (GPU/CPU) and through the Nvidia/OpenMP scheduler



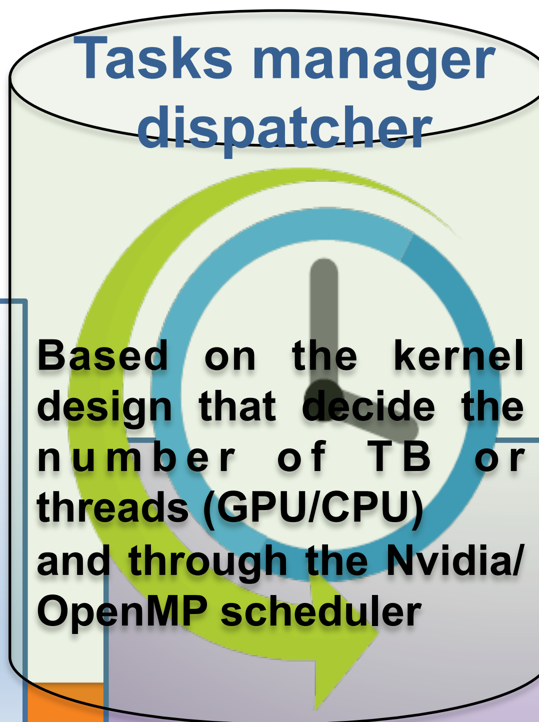
MAGMA Batched Computations

1. Batched computation

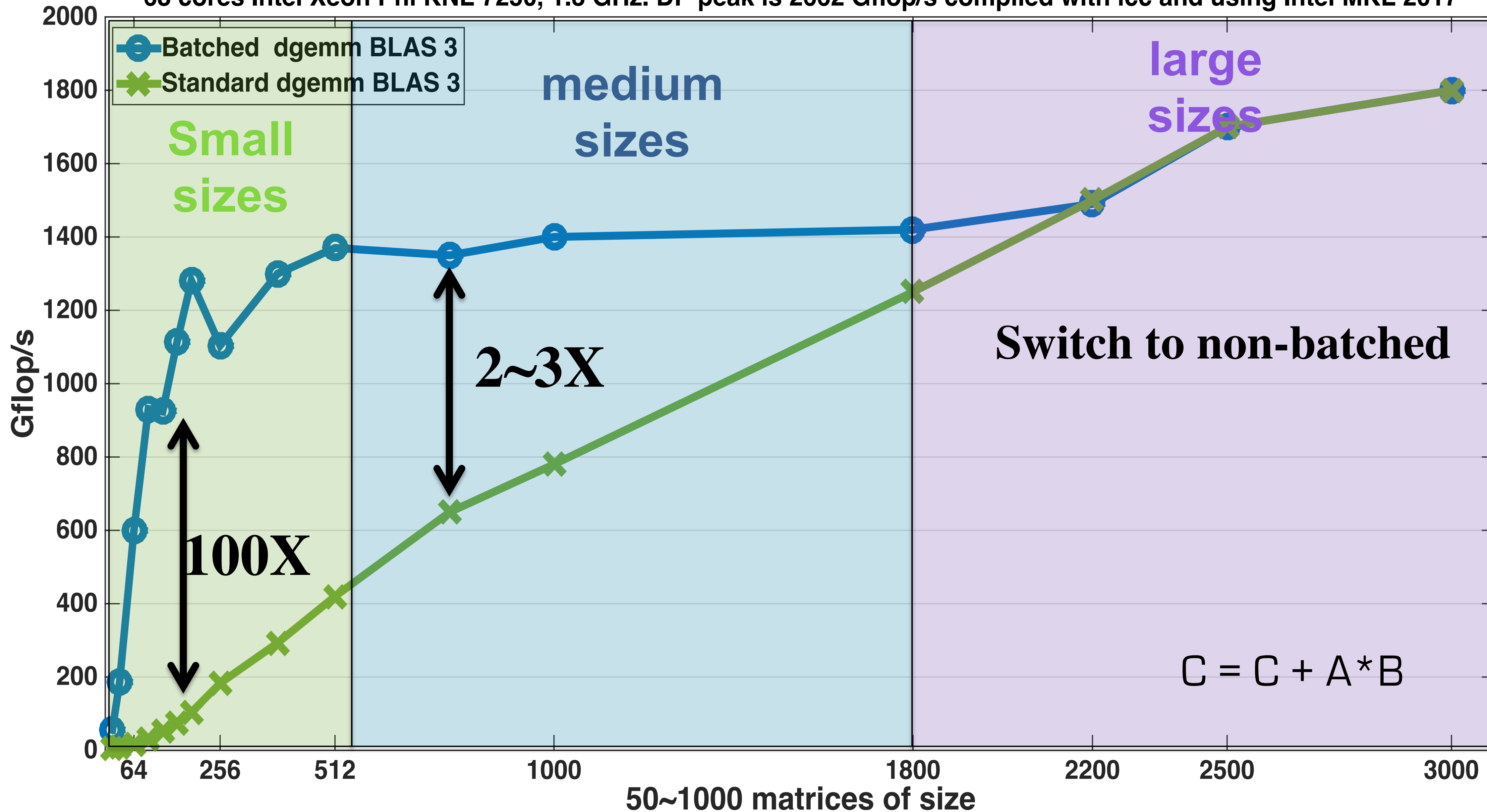
Distribute all the matrices over the available resources by assigning a matrix to each group of core/TB to operate on it independently

- For very small matrices, assign a matrix/core (CPU) or per TB for GPU
- For medium size a matrix go to a team of cores (CPU) or many TB's (GPU)
- For large size switch to multithreads classical 1 matrix per round.

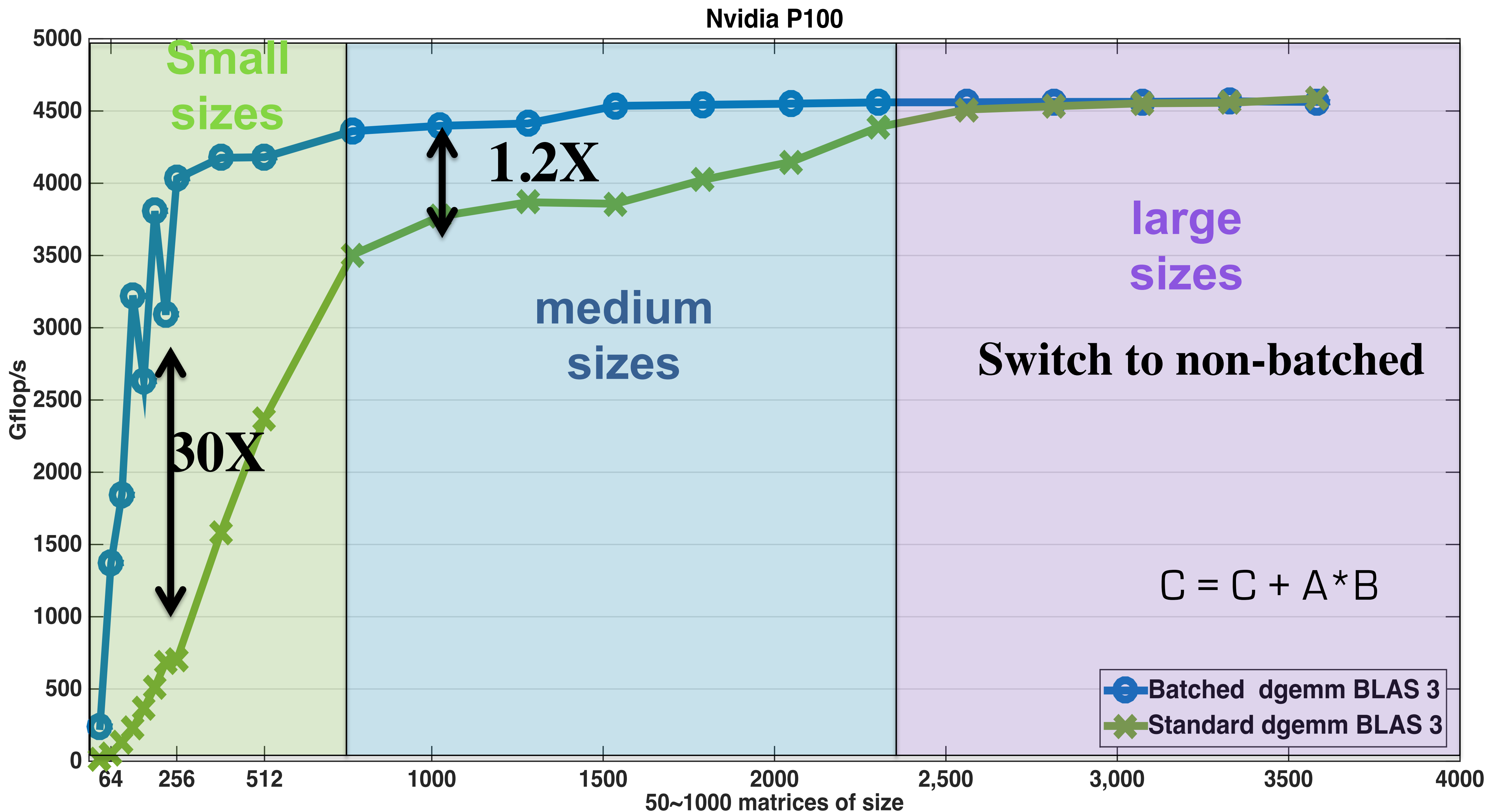
```
Batched_dgemm(…)
```



MAGMA Batched Computations

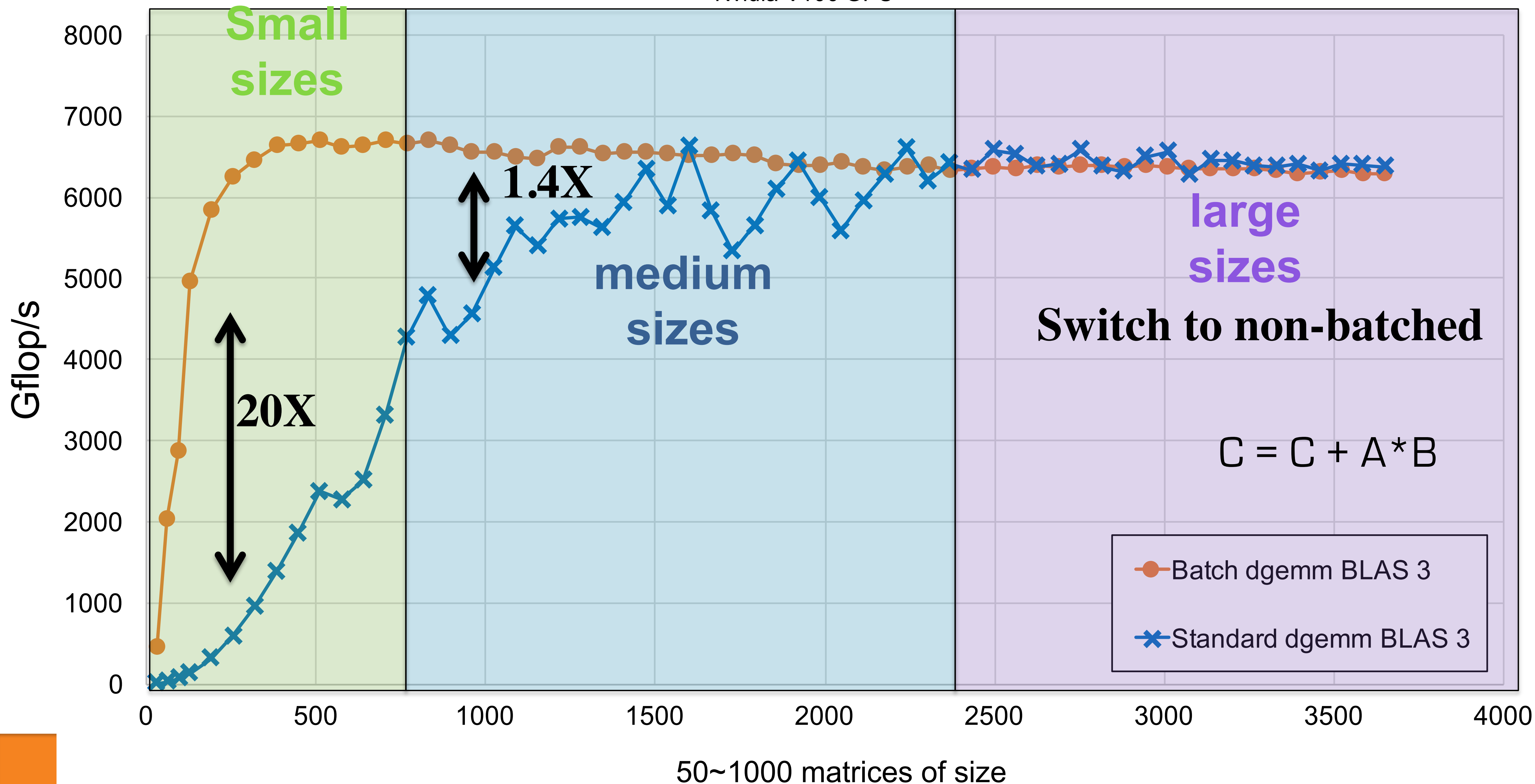


MAGMA Batched Computations



XASCALE OMPUTING ROJECT

MAGMA Batched Computations



MAGMA Batched Computations

MAGMA BATCHED

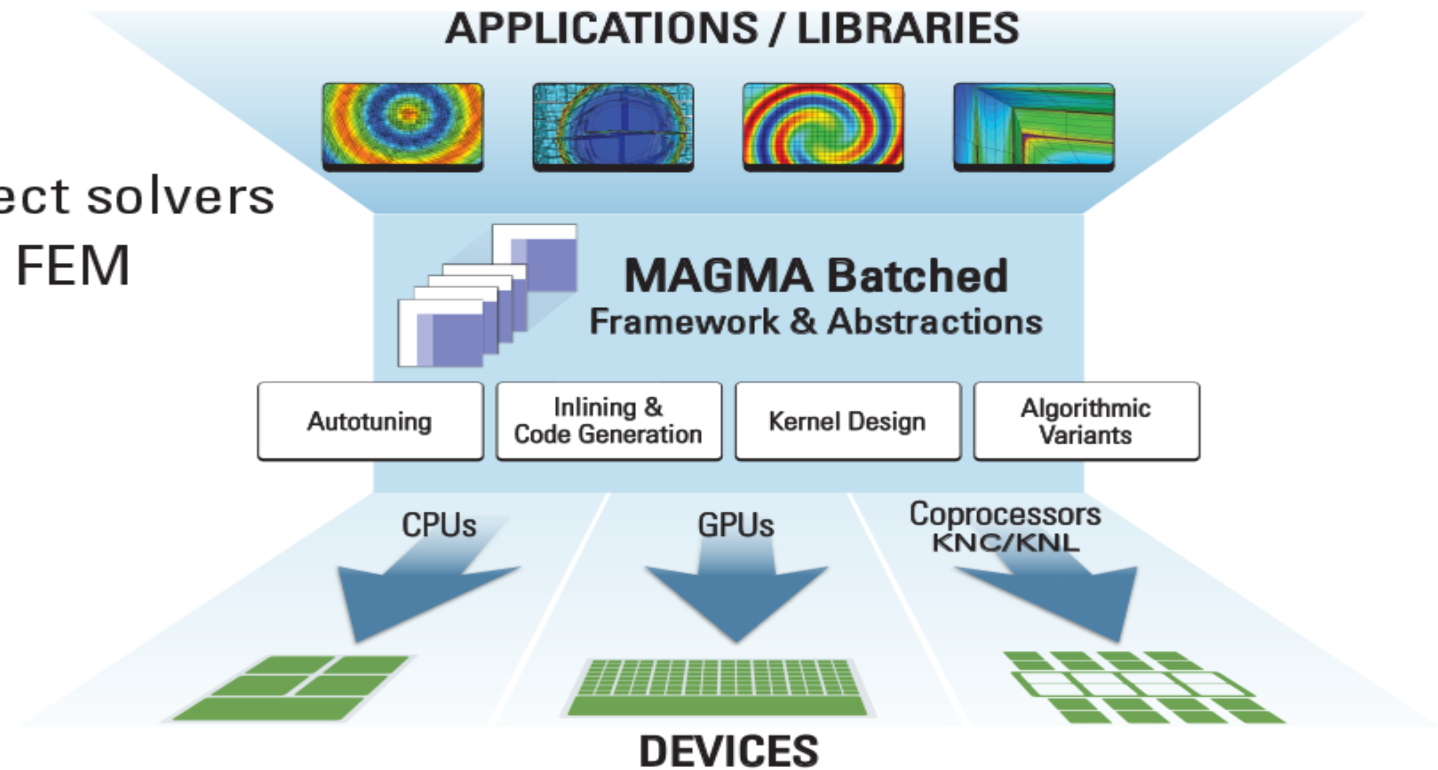
BATCHED FACTORIZATION OF A SET OF SMALL MATRICES IN PARALLEL

Numerous applications require factorization of many small matrices

- Deep learning
- Structural mechanics
- Astrophysics
- Sparse direct solvers
- High-order FEM simulations

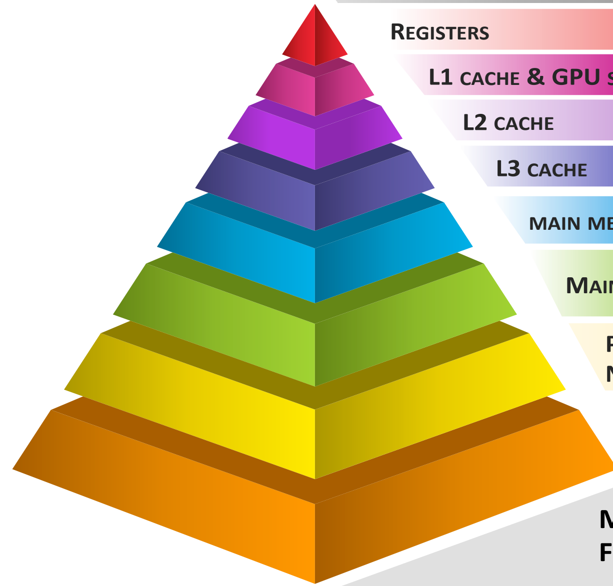
ROUTINES

LU, QR, and Cholesky	✓
Solvers and matrix inversion	✓
All BLAS 3 (fixed + variable)	✓
SYMV, GEMV (fixed + variable)	✓



ICL ECP EXASCALE COMPUTING PROJECT

Implementation on current hardware is becoming challenging



Memory hierarchies	Intel Haswell E5-2650 v3	Intel KNL 7250 DDR5|MCDRAM	ARM Cortex A57	Nvidia P100	Nvidia V100
	10 cores 368 Gflop/s 105 Watts	68 cores 2662 Gflop/s 215 Watts	4 cores 32 Gflop/s 7 Watts	56 SM 64 cores 4700 Gflop/s 250 Watts	80 SM 64 cores 7500 Gflop/s 300 Watts
Registers	16/core AVX2	32/core AVX-512	32/core	256 KB/SM	256 KB/SM
L1 cache & GPU shared memory	32 KB/core	32 KB/core	32 KB/core	64 KB/SM	96 KB/SM
L2 cache	256 KB/core	1024 KB/2cores	2 MB	4 MB	6 MB
L3 cache	25 MB	0...16 GB	N/A	N/A	N/A
Main memory	64 GB	384 |16 GB	4 GB	16 GB	16 GB
Main memory BW	68 GB/s 5.4 flops/byte	115 | 421 GB/s 23 | 6 Flops/byte	26 GB/s 1.2 flops/byte	720 GB/s 6.5 flops/byte	900 GB/s 8.3 flops/byte
PCI express gen3x16 NVLink	16 GB/s 23 flops/byte	16 GB/s 166 flops/byte	16 GB/s 2 flops/byte	16 GB/s 294 flops/byte	300 GB/s (NVL) 25 flops/byte
Interconnect Infiniband EDR	12 GB/s 30 flops/byte	12 GB/s 221 flops/byte	12 GB/s 2.6 flops/byte	12 GB/s 392 flops/byte	12 GB/s 625 flops/byte

Memory hierarchies for different type of architectures
Flops per byte transfer (all flop rates for 64 bit operands)

Workshop on Batched, Reproducible, and Reduced Precision BLAS

Georgia Tech
Computational Science and Engineering
Atlanta, GA
February 23—25, 2017

http://bit.ly/Batch-BLAS-2017

Draft Reports
Batched BLAS Draft Reports:
https://www.dropbox.com/s/olocmipyxfvcaui/batched_api_03_30_2016.pdf?dl=0

Batched BLAS Poster:
https://www.dropbox.com/s/ddkym76fapddf5c/Batched%20BLAS%20Poster%2012.pdf?dl=0

Batched BLAS Slides:
https://www.dropbox.com/s/kz4fhcipz3e56ju/BatchedBLAS-1.pptx?dl=0

Webpage on ReproBLAS:
http://bebop.cs.berkeley.edu/reproblas/

Efficient Reproducible Floating Point Summation and BLAS:
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-229.pdf

Toward a standard Batched BLAS API

Status and goal

- Batched BLAS functionalities becomes a major factor in our community
  - Batched routines gradually make their steps into vendor libraries (Intel, Nvidia, etc) as well as into research software (MAGMA, Kokkos, etc)
- Today's API differ significantly which can lead to poor portability
- **Thus the community needs to make an effort to standardize the Batched BLAS API**

Toward a standard Batched BLAS API

Status and goal

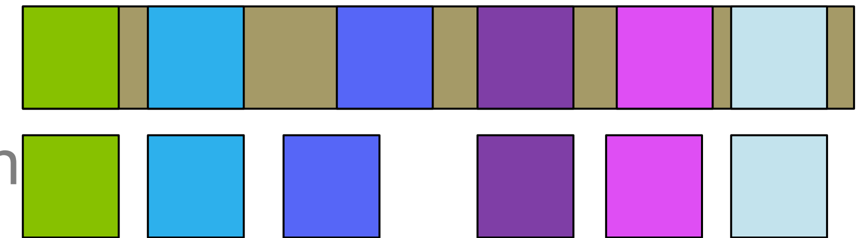
- Heterogeneity in the hardware (GPU, Phi, CPU) deeply complicates efforts to provide a standard interface
  - The calling interface may affect the implementation (performance) which depend on the architecture
- **Our objective today, is to try to define a cross-architecture standard without a severe performance penalty**
- Other API's could be considered as auxiliary API's or API with extra features

Toward a standard Batched BLAS API

Matrices are stored BLAS-like *"the usual storage that we know"*

- array of pointers: that consists of a pointer to each matrix
  - Data could belong to one memory allocation
  - Data could be anywhere, different allocations
  - Matrices could be equidistant or not from each oth
  - Is suitable for CPU, GPU, Phi
  - Accommodate most of the cases

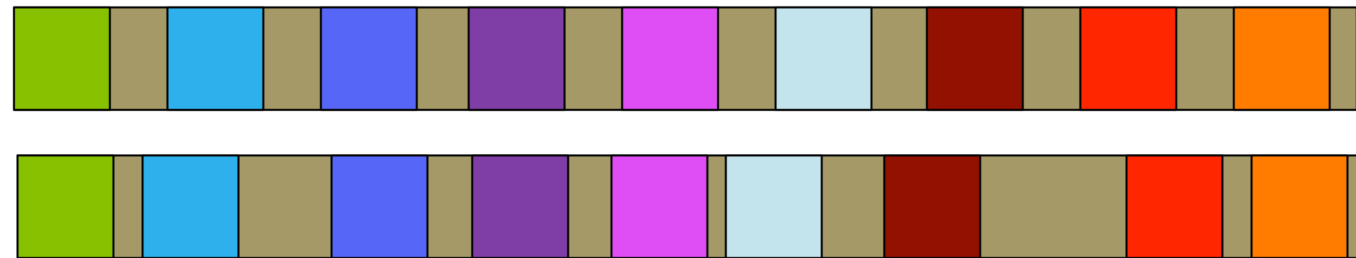
  - User has to fill-up the array of pointers



Toward a standard Batched BLAS API

Matrices are stored BLAS like *"the usual storage that we know"*

- array of pointers: that consists of a pointer to each matrix
- strided: as one pointer to a memory and matrices are strided inside
  - Fixed stride
  - Variable stride
  - Suitable for CPU, GPU, Phi



  - For variable stride, user has to fill-up the array
  - Cannot accommodate data that was not been allocated within the same chunk of memory. Think about adding matrices to the batch.

Toward a standard Batched BLAS API

Matrices are stored BLAS like *"the usual storage that we know"*

- array of pointers: that consists of a pointer to each matrix
- strided: as one pointer to a memory and matrices are strided inside

Matrices are stored in interleaved fashion or *compact*

- data can be interleaved by batchcount or by chunk (SIMD, AVX, Warp)
- Is only good for sizes less than 20 and only for some routines such as GEMM, TRSM, while it has performance and implementation issues for routines like LU or QR factorization
- Requires user or implementation to convert/reshuffle the memory storage since most of the storage are BLAS-like

Toward a standard Batched BLAS API

API discussion

- Same or separate API for fixed and variable size batches?
  - Have two separate API's?
  - Have a flag that switch between fixed and variable?
  - To simplify user life and avoid a combinatorial combination of parameter, we propose to **distinguish between fixed and variable size APIs**

```
void batchedblas_dgemm_vbatched (
       batched_trans_t transA , batched_trans_t transB ,
       batched_int_t *m, batched_int_t *n, batched_int_t *k,
       double alpha ,
       double const * const *dA_array , batched_int_t *ldda ,
       double const * const *dB_array , batched_int_t *lddb ,
       double beta ,
       double **dC_array , batched_int_t *lddc ,
       batched_int_t batchCount , batched_queue_t queue );
```

Toward a standard Batched BLAS API

API discussion

- Same or separate API for fixed and variable size batches?
  - Have two separate API's?
  - Have a flag that switch between fixed and variable?
  - To simplify user life and avoid a combinatorial combination of parameter, we propose to **distinguish between fixed and variable size APIs**

Toward a standard Batched BLAS API

- Consider GEMM as an example
- 'F' = Fixed, unified across a batch/group of problem
- 'V' = Variable, each problem has its assigned value

Type	Name		option args (transA, transB)	scaling args (alpha, beta)	sizes, ld's, inc's (m, n, k) (lda, ldb,ldc)	data pointers (A, B, C)
Flag-based Flat	Standard	BATCH_FIXED	F	F	F	V
		BATCH_VAR	V	V	V	V
Group	MKL	per group	F	F	F	V
		across groups	V	V	V	V
Flat	MAGMA	Fixed API	F	F	F	V
		Var. API	F	F	V	V
	cuBLAS		F	F	F	V

MAGMA Batched moving forward: C++ API

- The goal is to have one API that allows any argument to be unified or varied across the batch/group
- We can support different scenarios through overloading and/or advanced standard containers
  - Overloading works, but will results in several APIs per each routine

- The main idea is to use the std::vector container
  - Every argument is std::vector<…>
  - The size of the vector determines if it is unified or varied

MAGMA Batched moving forward: C++ API

```
template<typename T>
void gemm_batch(
  std::vector<Op> const &transA, std::vector<Op> const &transB,
  std::vector<blas_int> const &m, std::vector<blas_int> const &n, std::vector<blas_int> const &k,
  std::vector<T> const &alpha,
  std::vector<T*> const &A, std::vector<blas_int> const &lda,
  std::vector<T*> const &B, std::vector<blas_int> const &ldb,
  std::vector<T> const &beta,
  std::vector<T*> const &C, std::vector<blas_int> const &ldc,
  const blas_int batchCount);
```

- This API supports over 1000 possibilities for GEMM
- The size of C must be equal to batchCount
- The size of other arguments can be either 1 or batchCount
- Error checking through std::vector<blas_int> info
- The group API can be supported by promoting batchCount to be a std::vector

MAGMA Batched moving forward: C++ API

```
template<typename T>
void gemm_batch(
  std::vector<Op> const &transA, std::vector<Op> const &transB,
  std::vector<blas_int> const &m, std::vector<blas_int> const &n, std::vector<blas_int> const &k,
  std::vector<T> const &alpha,
  std::vector<T*> const &A, std::vector<blas_int> const &lda,
  std::vector<T*> const &B, std::vector<blas_int> const &ldb,
  std::vector<T> const &beta,
  std::vector<T*> const &C, std::vector<blas_int> const &ldc,
  const blas_int batchCount);
```

- Pros:
  1) Many possibilities through a single interface (opposed to overloading)
  2) Duplicate argument values are entirely eliminated
  3) Using a widely-used standard container in C++
     - Function to get the size, max., min., … etc are already available
     - Also available for GPUs (e.g. Thrust Library for CUDA)
  4) A reference implementation for all possibilities is easily feasible
     - BLAS + OpenMP on CPUs
     - BLAS + Streams on GPUs

MAGMA Batched moving forward: C++ API

```
template<typename T>
void gemm_batch(
  std::vector<Op> const &transA, std::vector<Op> const &transB,
  std::vector<blas_int> const &m, std::vector<blas_int> const &n, std::vector<blas_int> const &k,
  std::vector<T> const &alpha,
  std::vector<T*> const &A, std::vector<blas_int> const &lda,
  std::vector<T*> const &B, std::vector<blas_int> const &ldb,
  std::vector<T> const &beta,
  std::vector<T*> const &C, std::vector<blas_int> const &ldc,
  const blas_int batchCount);
```

- Cons:
  1) Error checking can be an overhead
  2) Should also check size consistencies
     - E.g. If the size of A is 1, then m, k, and lda should be of size 1 as well
  3) The user has to be careful about the vector sizes
     - This is the price of a flexible API

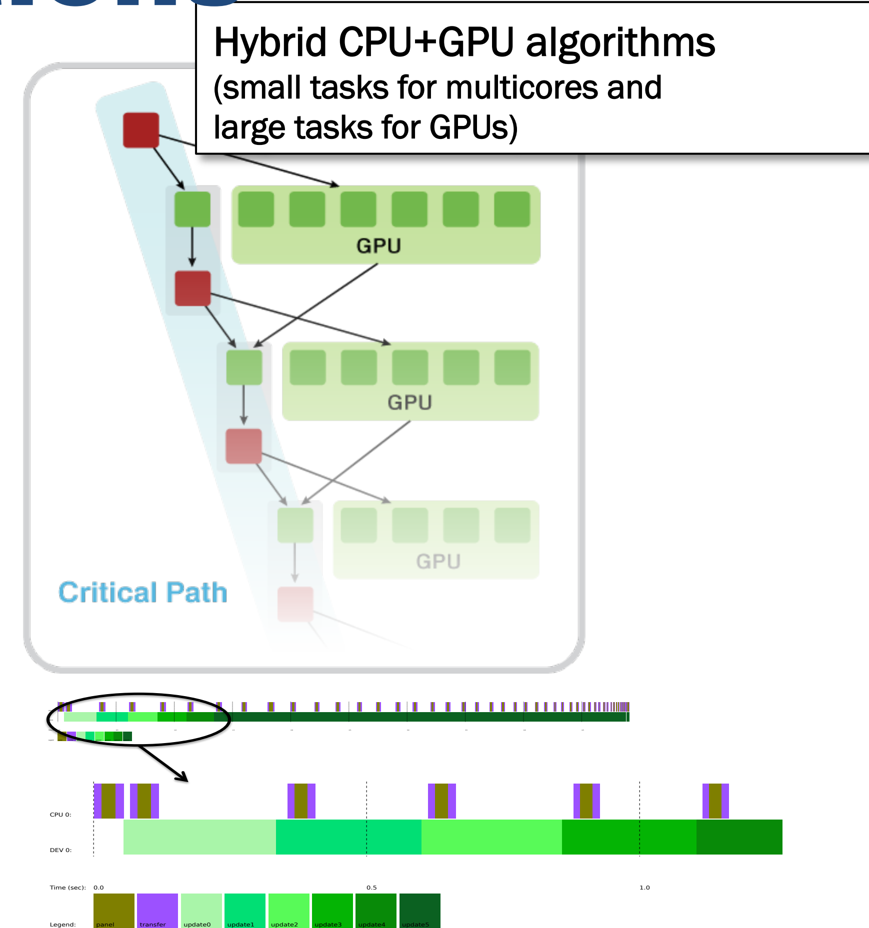
MAGMA Batched API

- API unification
  - Current batch APIs are divergent
    - MKL, MAGMA, cuBLAS, and others have their own distinct APIs
    - Flat vs group API?
    - Fixed vs. variable size?
    - Lots of possibilities -> explosion of interfaces
  - Standardization effort is ongoing
    - Community effort
    - Two batch BLAS workshops so far
    - First standard C API (http://eprints.maths.manchester.ac.uk/2464/)
    - More details (http://icl.utk.edu/bblas/)

MAGMA Batched API

- C++ API for batch routines
  - An effort to have one unified (standard) API
  - Overcomes C/Fortran shortcomings, by using standard containers and overloading
  - One API that supports fixed/variable/group interfaces
  - Heavily relies on `std::vector` container in C++
  - First draft is published (http://www.icl.utk.edu/publications/swan-004)

MAGMA Batched Computations



Classical strategies design

- For large problems the strategy is to prioritize the data-intensive operations to be executed by the accelerator and keep the memory-bound ones for the CPUs since the hierarchical caches are more appropriate to handle it

Challenges

- **Cannot be used** here since matrices are very small and communication becomes expensive

Proposition

- **Develop a GPU-only implementation**

MAGMA Batched Computations

Classical strategies design

- For large problems performance is driven by the Level 3 BLAS (GEMM)

Challenges

- For batched small matrices it is more complicated

Proposition

- **Rethink and Redesign both phases** in a tuned efficient way

MAGMA Batched Computations

Key observations and current situation:

Classical strategies design

- A recommended way of writing efficient GPU kernels is to **use the whole GPU's shared memory,registers/TB** – load it with data and reuse that data in computations as much as possible.

Challenges

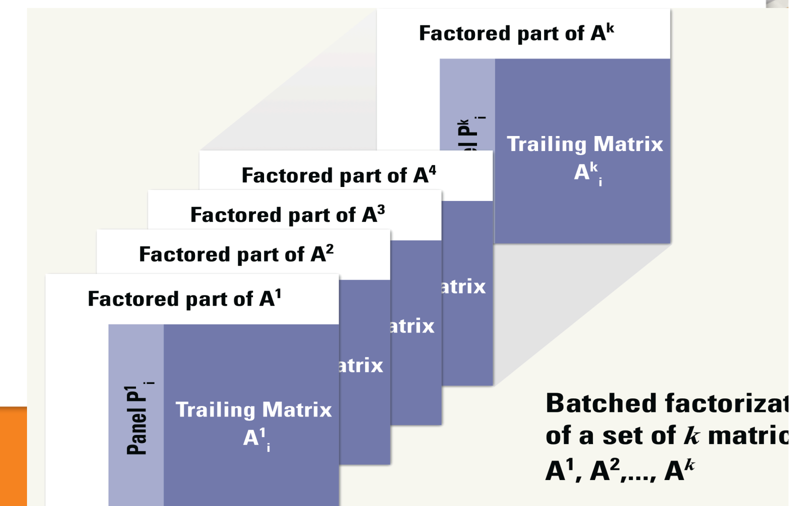
- Our study and experience shows that this procedure provides very good performance for classical GPU kernels but is **not that appealing for batched algorithm** for different reasons.

MAGMA Batched Computations

Challenges

- Completely **saturating the shared memory** per SMX can decrease the performance of memory bound operations, since only one thread-block will be mapped to that SMX at a time (**low occupancy**)
- Due to the **limited parallelism** in the small matrices, the number of threads used in the thread block will be limited, resulting in low occupancy, and subsequently poor core utilization
- **Shared memory is small** (48KB/SMX) to fit the whole panel
- The panel involves Non-GPU friendly operations:
  - Vectors column (find the max, scale, norm, reduction)
  - Row interchanges (swap)
  - Small number of vectors (apply)

Proposition: custom design per operations type

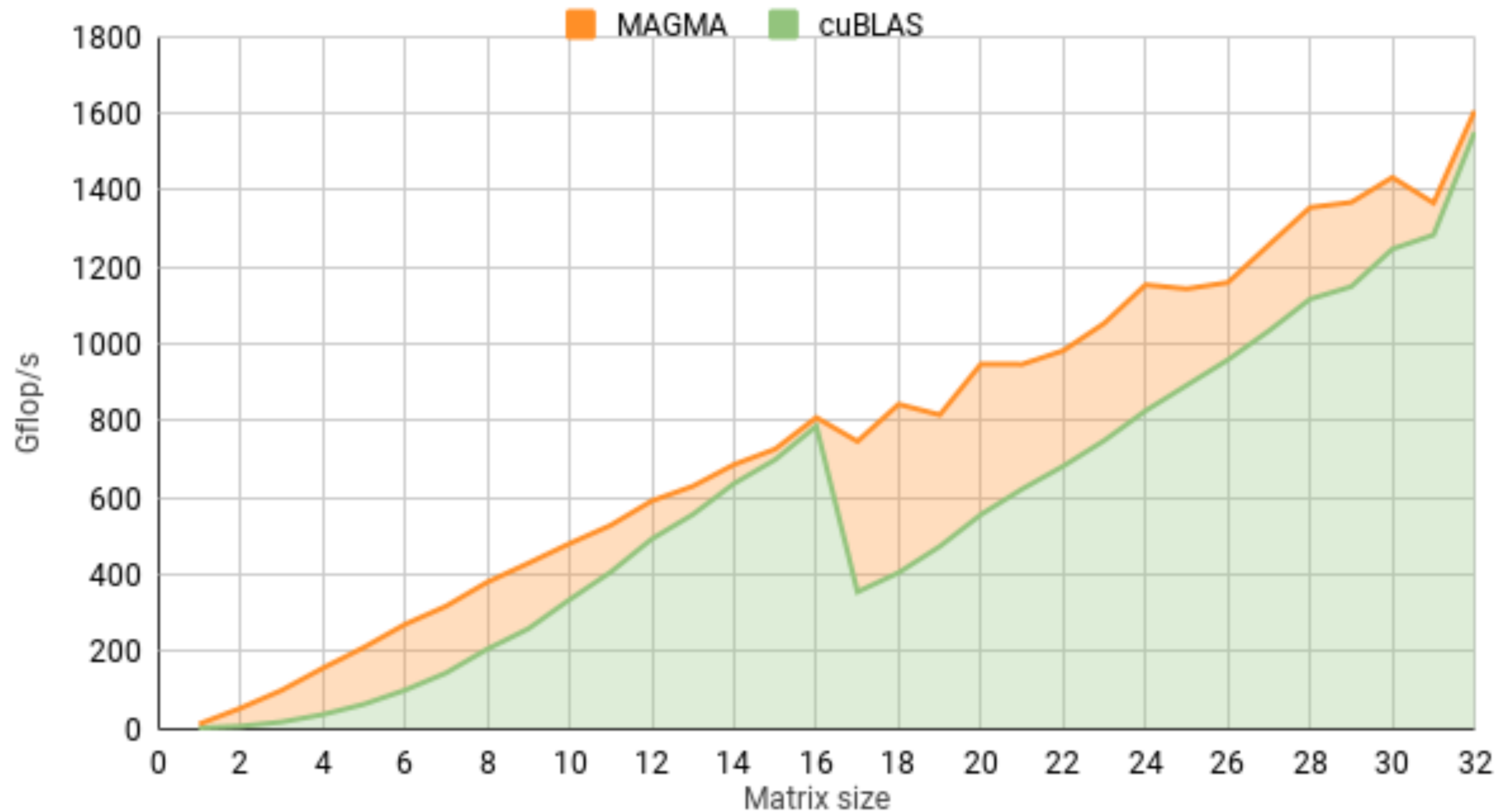


MAGMA Batched Computations

GPU Optimization Summary

- **Hardware concepts**
  - CUDA core
  - Warp
  - Half-warp
  - Register file
  - Shared memory
  - Atomics
  - Shuffles
  - SMX
- **Software concepts**
  - Stream
  - Thread block
  - Kernel
  - Inlining
  - Intrinsics
- **Algorithmic concepts**
  - Blocking
  - Recursive blocking
  - Kernel replacement
  - Out-of-place operations

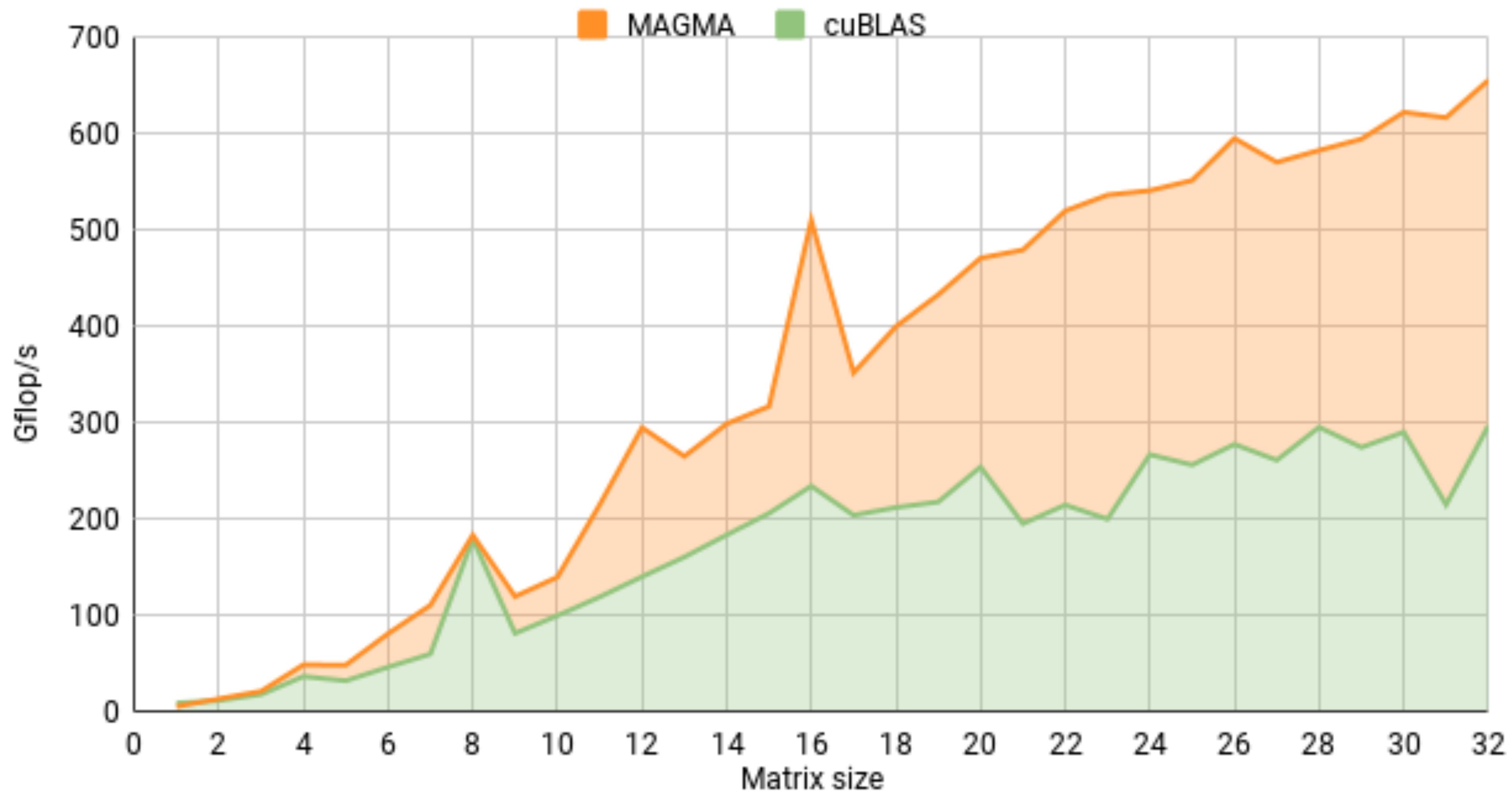
MAGMA Batched Computations



ICL

ECP EXASCALE COMPUTING PROJECT

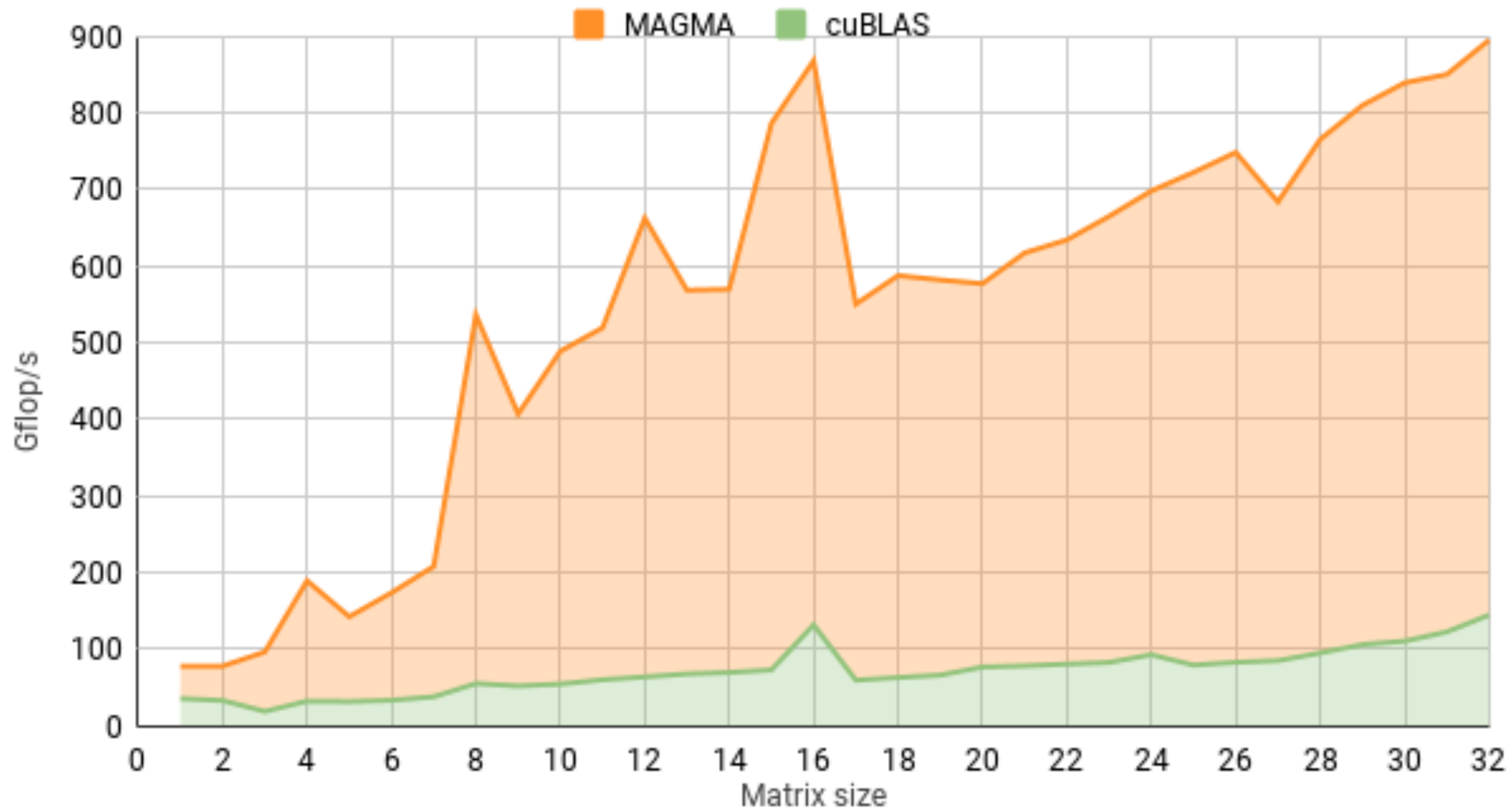
MAGMA Batched Computations



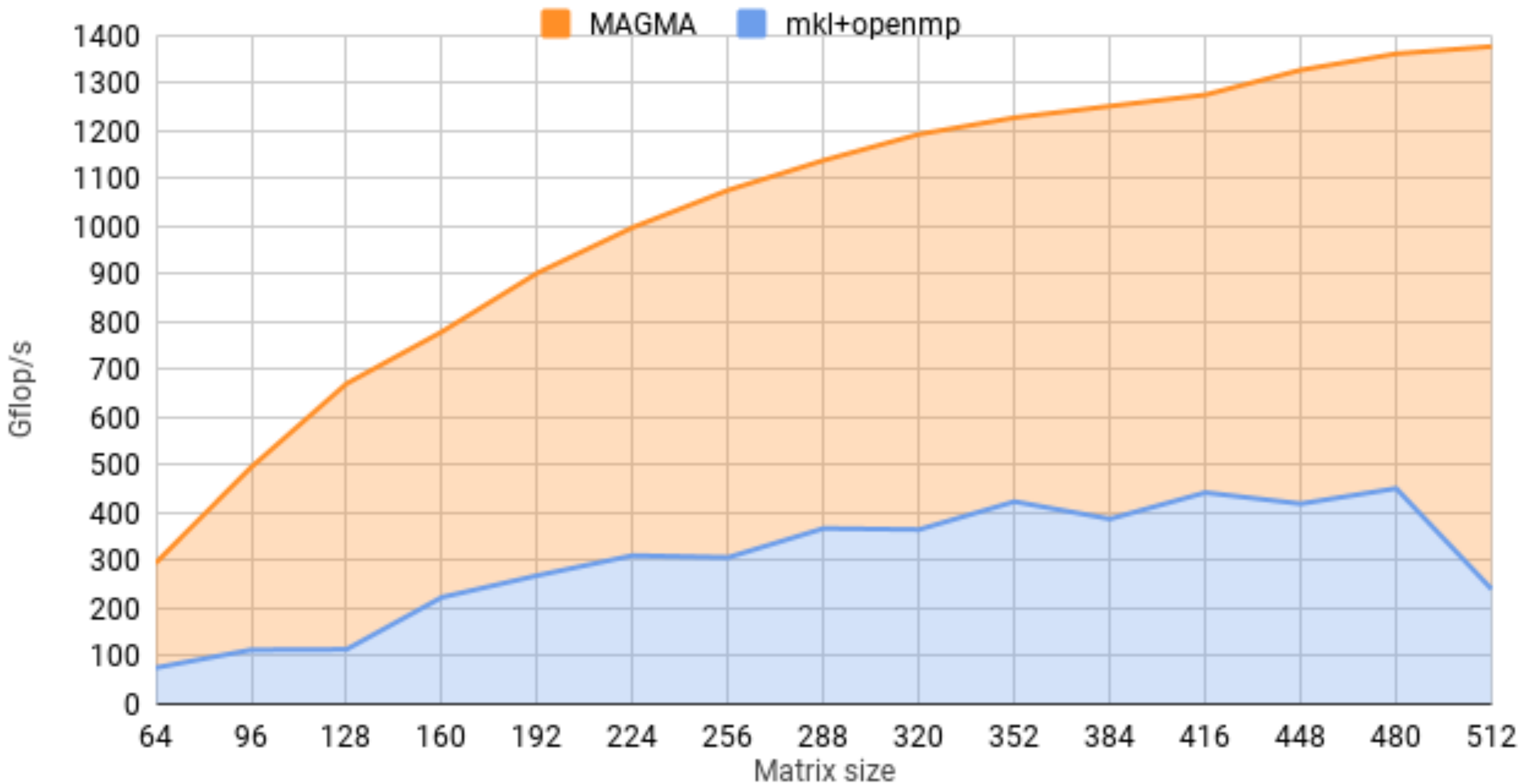
ICL

ECP EXASCALE COMPUTING PROJECT

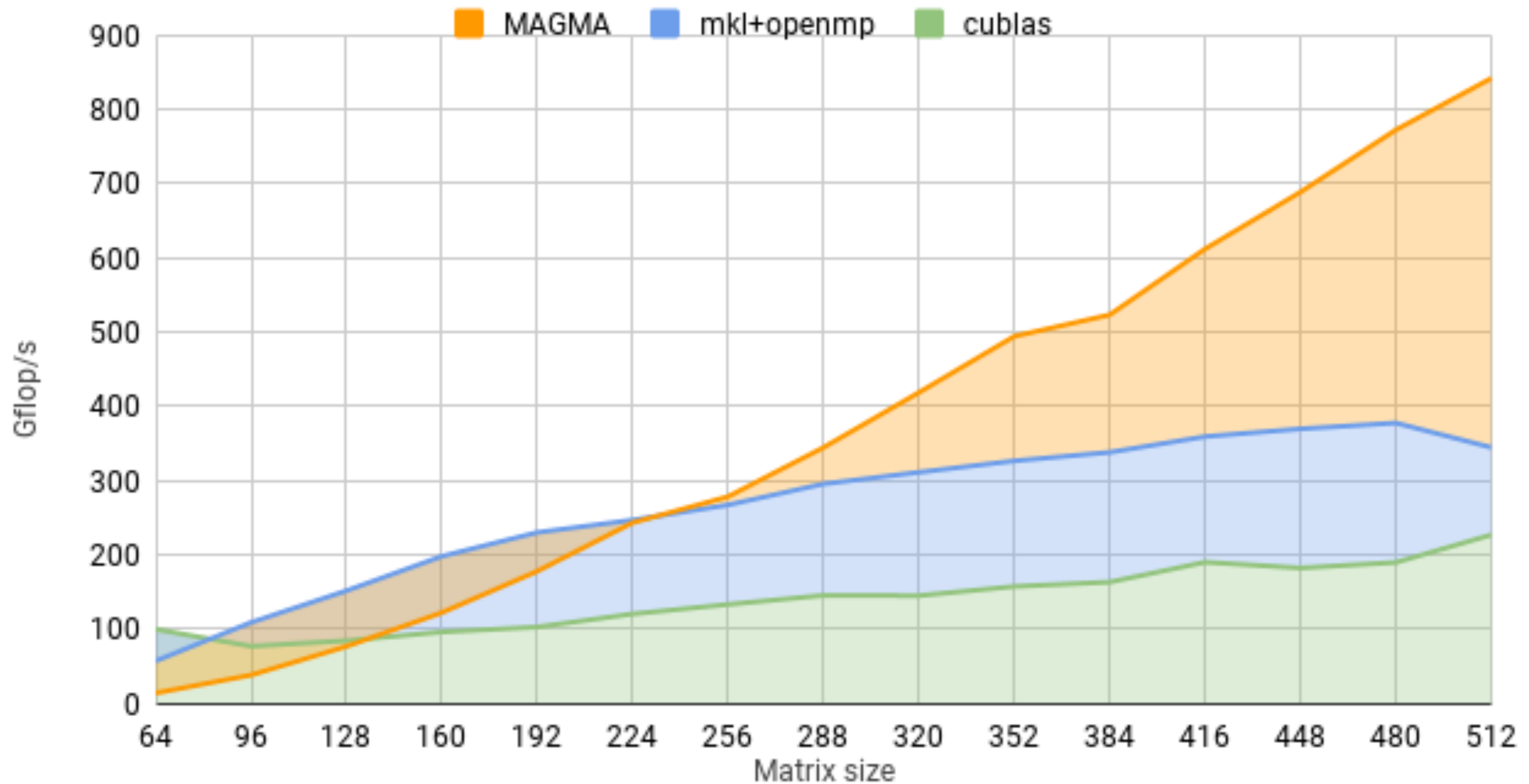
MAGMA Batched Computations



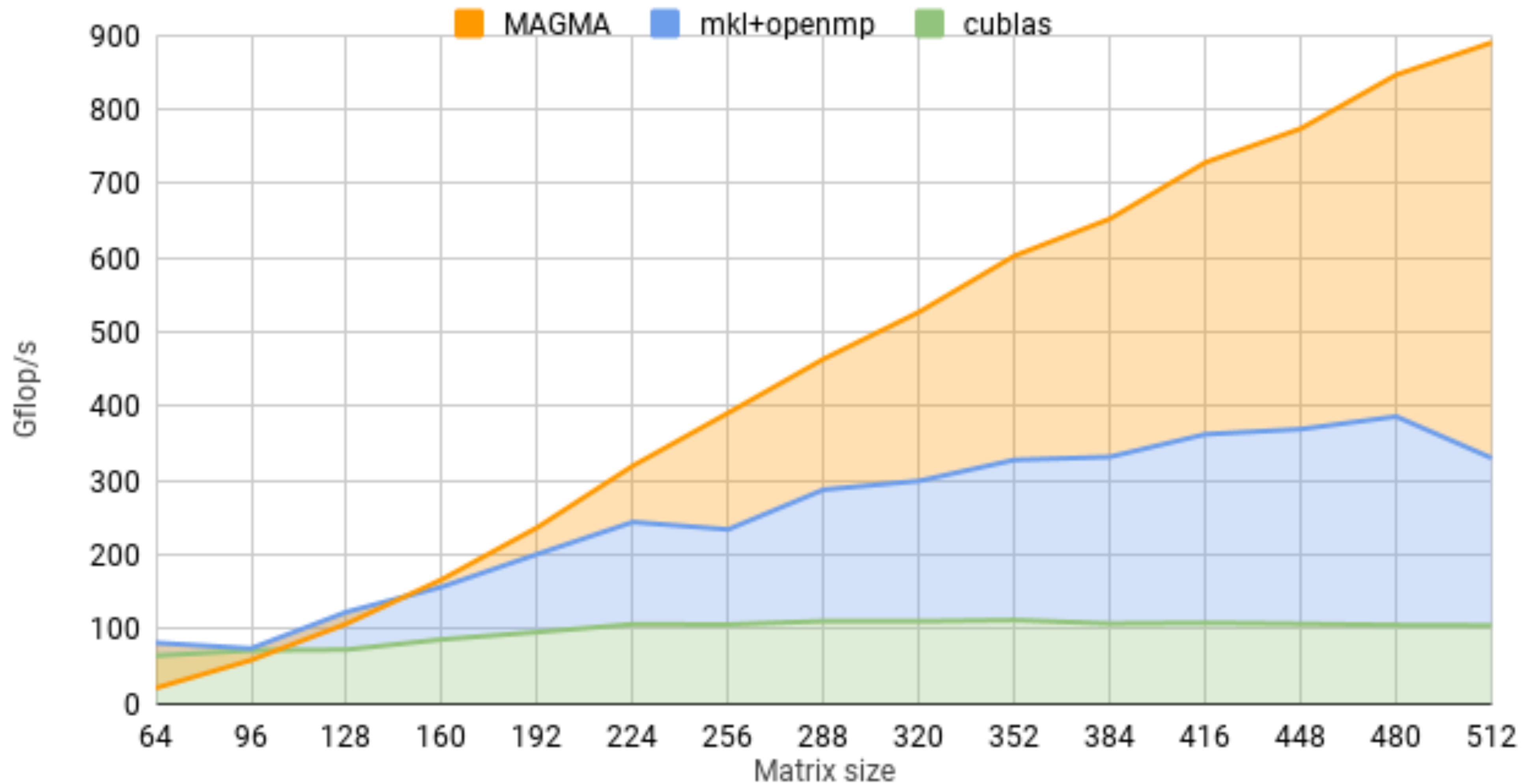
MAGMA Batched Computations



MAGMA Batched Computations



MAGMA Batched Computations

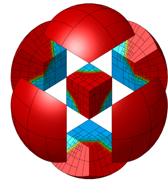


MAGMA Batched Computations

Problem sizes influence algorithms & optimization techniques

Used in High-order (HO) Finite Element Methods (FEM)

Lagrangian Hydrodynamics in the BLAST code[1]



On semi-discrete level our method can be written as

$$\textbf{Momentum Conservation:} \quad \frac{d\mathbf{v}}{dt} = -\mathbf{M}_\mathbf{v}^{-1}\mathbf{F}\cdot\mathbf{1}$$

$$\textbf{Energy Conservation:} \quad \frac{de}{dt} = \mathbf{M}_\mathbf{e}^{-1}\mathbf{F}^\mathbf{T}\cdot\mathbf{v}$$

$$\textbf{Equation of Motion:} \quad \frac{d\mathbf{x}}{dt} = \mathbf{v}$$

where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M}_\mathbf{v}$ and $\mathbf{M}_\mathbf{e}$ are independent of time velocity and energy mass matrices; and $\mathbf{F}$ is the generalized corner force matrix depending on $(\mathbf{v}, \mathbf{e}, \mathbf{x})$ that needs to be evaluated at every time step.

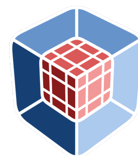
[1] V. Dobrev, T.Kolev, R.Rieben. *High order curvilinear finite element methods for Lagrangian hydrodynamics*. SIAM J.Sci.Comp.34(5), B606–B641. (36 pages)

Matrix-free basis evaluation needs efficient tensor contractions,

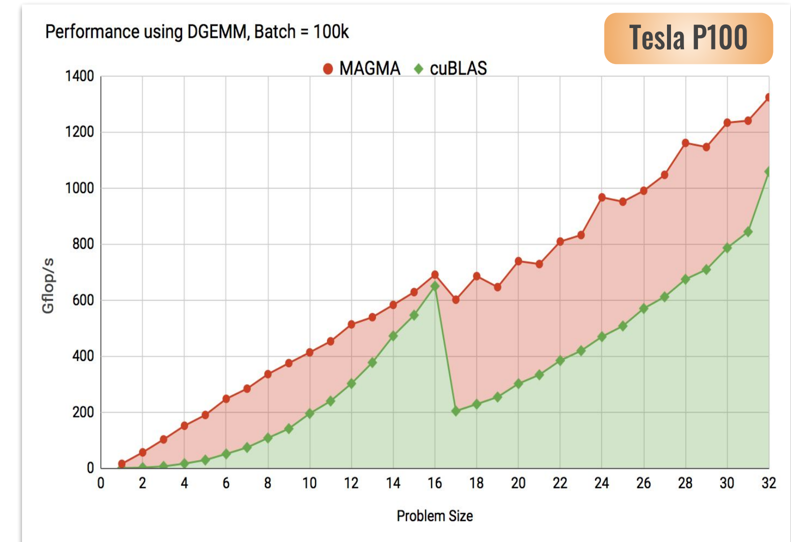
$$C_{i1,i2,i3} = \sum_k A_{k,i1} B_{k,i2,i3}$$

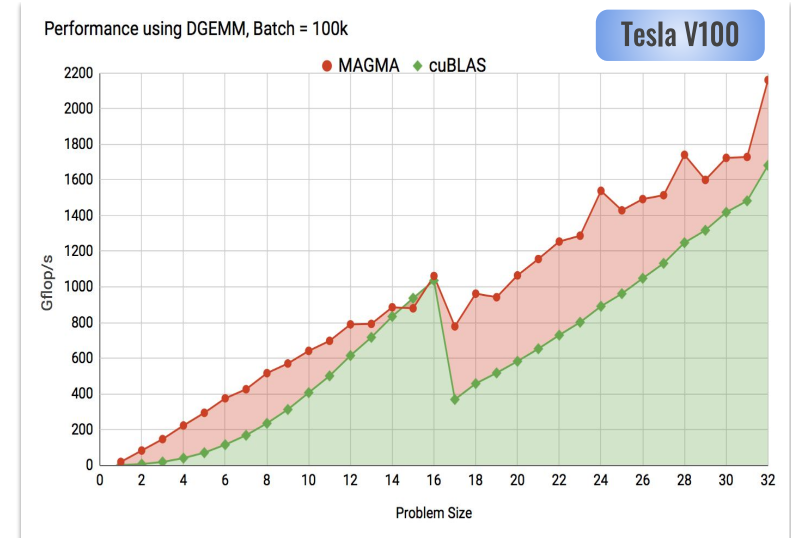
Within ECP CEED Project, designed MAGMA batched methods to split the computation in many small high-intensity GEMMs, grouped together (batched) for efficient execution:

Batch_{ $C_{i3} = A^T B_{i3}$, for range of i3 }



http://ceed.exascaleproject.org/





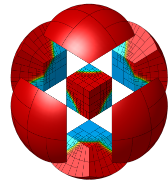
ICL
ECP EXASCALE COMPUTING PROJECT

MAGMA Batched : fused kernels

Problem sizes influence algorithms & optimization techniques

Used in High-order (HO) Finite Element Methods (FEM)

Lagrangian Hydrodynamics in the BLAST code[1]



On semi-discrete level our method can be written as

Momentum Conservation: $\frac{d\mathbf{v}}{dt} = -\mathbf{M}_\mathbf{v}^{-1}\mathbf{F}\cdot\mathbf{1}$

Energy Conservation: $\frac{de}{dt} = \mathbf{M}_e^{-1}\mathbf{F}^\mathbf{T}\cdot\mathbf{v}$

Equation of Motion: $\frac{d\mathbf{x}}{dt} = \mathbf{v}$

where $\mathbf{v}$, $\mathbf{e}$, and $\mathbf{x}$ are the unknown velocity, specific internal energy, and grid position, respectively; $\mathbf{M}_\mathbf{v}$ and $\mathbf{M}_\mathbf{e}$ are independent of time velocity and energy mass matrices; and $\mathbf{F}$ is the generalized corner force matrix depending on $(\mathbf{v}, \mathbf{e}, \mathbf{x})$ that needs to be evaluated at every time step.

[1] V. Dobrev, T.Kolev, R.Rieben. *High order curvilinear finite element methods for Lagrangian hydrodynamics*. SIAM J.Sci.Comp.34(5), B606–B641. (36 pages)

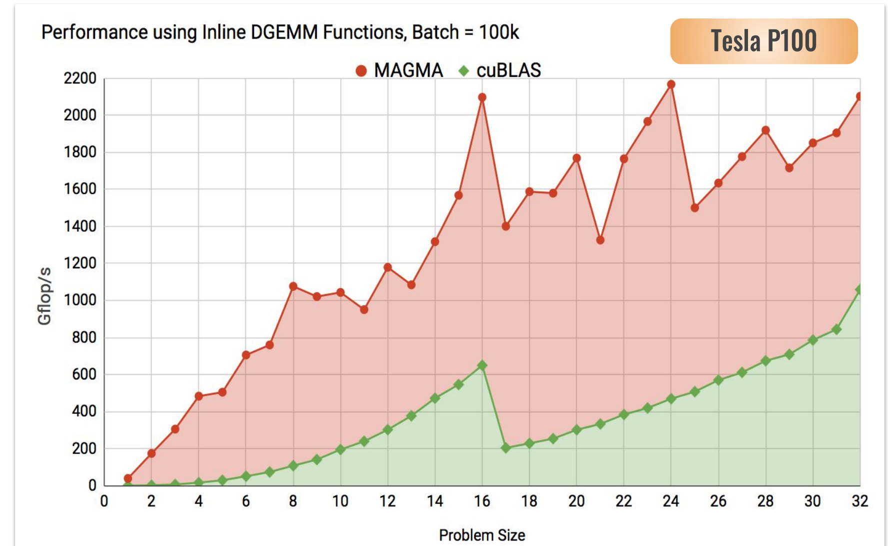
Matrix-free basis evaluation needs efficient tensor contractions,

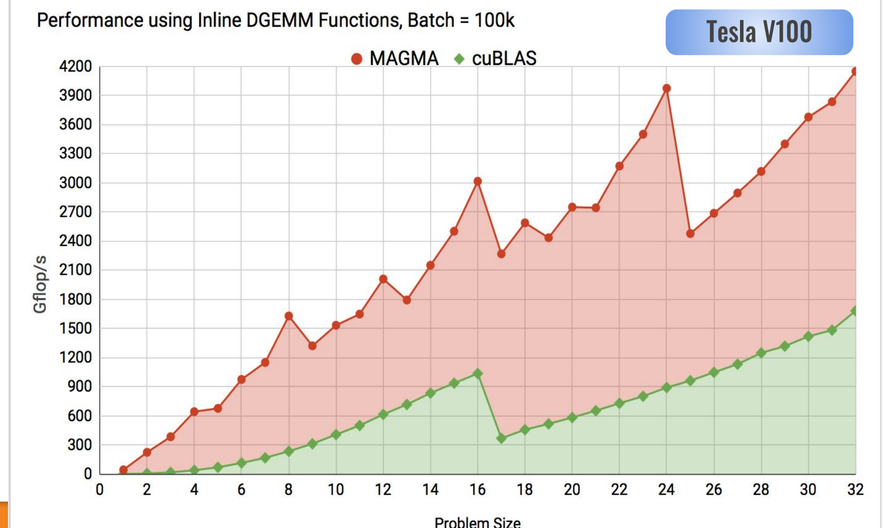
$$C_{i1,i2,i3} = \sum A_{k,i1} B_{k,i2,i3}$$

Within ECP CEED Project, designed MAGMA batched methods to split the computation in many small high-intensity GEMMs, grouped together (batched) for efficient execution:

Batch_{ $C_{i3} = A^T B_{i3}$, for range of i3 }

http://ceed.exascaleproject.org/





EXASCALE COMPUTING PROJECT

MAGMA now has GPU-only routines

GPU only routines

➢ MAGMA provide a set of GPU only routines such as Cholesky, LU and QR

Reference:

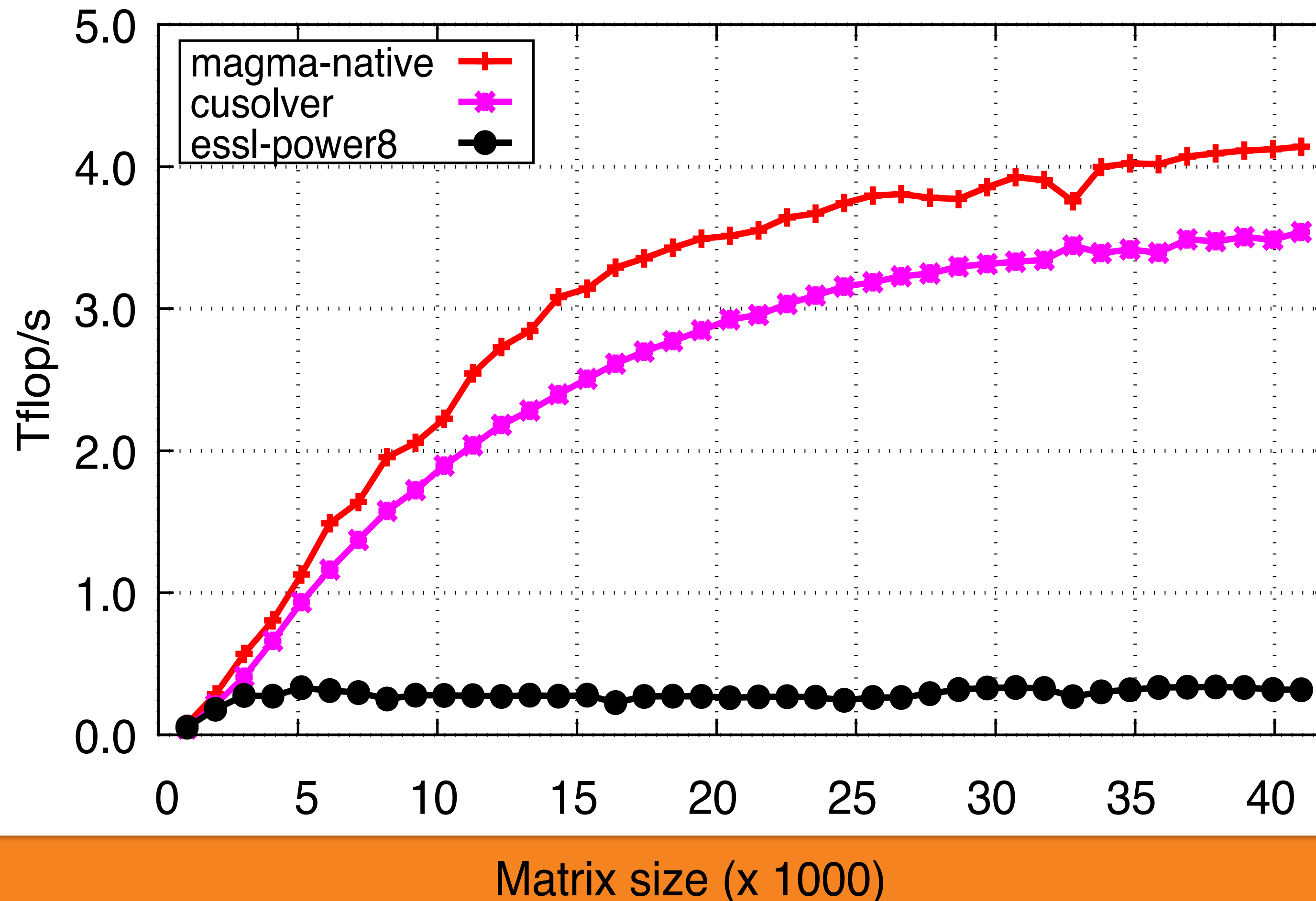
1. A. Haidar, S. Tomov, P. Luszczek, and J. Dongarra,
   MAGMA Embedded: Towards a Dense Linear Algebra Library for Energy Efficient Extreme Computing,
   2015 IEEE High Performance Extreme Computing Conference (HPEC '15), **Best paper**, Waltham, MA, September 15-17, 2015.

2. Haidar, A. Abdelfatah, S. Tomov, and J. Dongarra.
   High-performance Cholesky factorization for GPU-only execution.
   In Proceedings of the General Purpose GPUs (GPGPU-10). ACM, New York, NY, USA, 42-52. DOI: https://doi.org/10.1145/3038228.3038237

3. A. Haidar, A. Abdelfattah, M. Zounon, S. Tomov, J. Dongarra,
   A Guide For Achieving High Performance With Very Small Matrices on GPU: A case Study of Batched LU and Cholesky Factorizations,
   IEEE Transactions on Parallel and Distributed Systems (TPDS in press 2018)

4. A. Abdelfattah, A. Haidar, S. Tomov, J. Dongarra,
   Analysis and Design Techniques for High-Performance and Energy-Efficient Dense Linear Solvers on GPUs
   Submitted to IEEE Transactions on Parallel and Distributed Systems (TPDS 2017)

MAGMA now has GPU-only routines



MAGMA outperforms vendor library

Reference:
A. Haidar, A. Abdelfatah, S. Tomov, and J. Dongarra. **High-performance Cholesky factorization for GPU-only execution.** *In Proceedings of the General Purpose GPUs (GPGPU-10)*. ACM, New York, NY, USA, 42-52.
DOI: https://doi.org/10.1145/3038228.3038237

Leveraging Half Precision in HPC on V100

MAGMA Mixed Precision algorithms

Idea: use lower precision to compute the expensive flops (LU $O(n^3)$) and then iteratively refine the solution in order to achieve the FP64 arithmetic

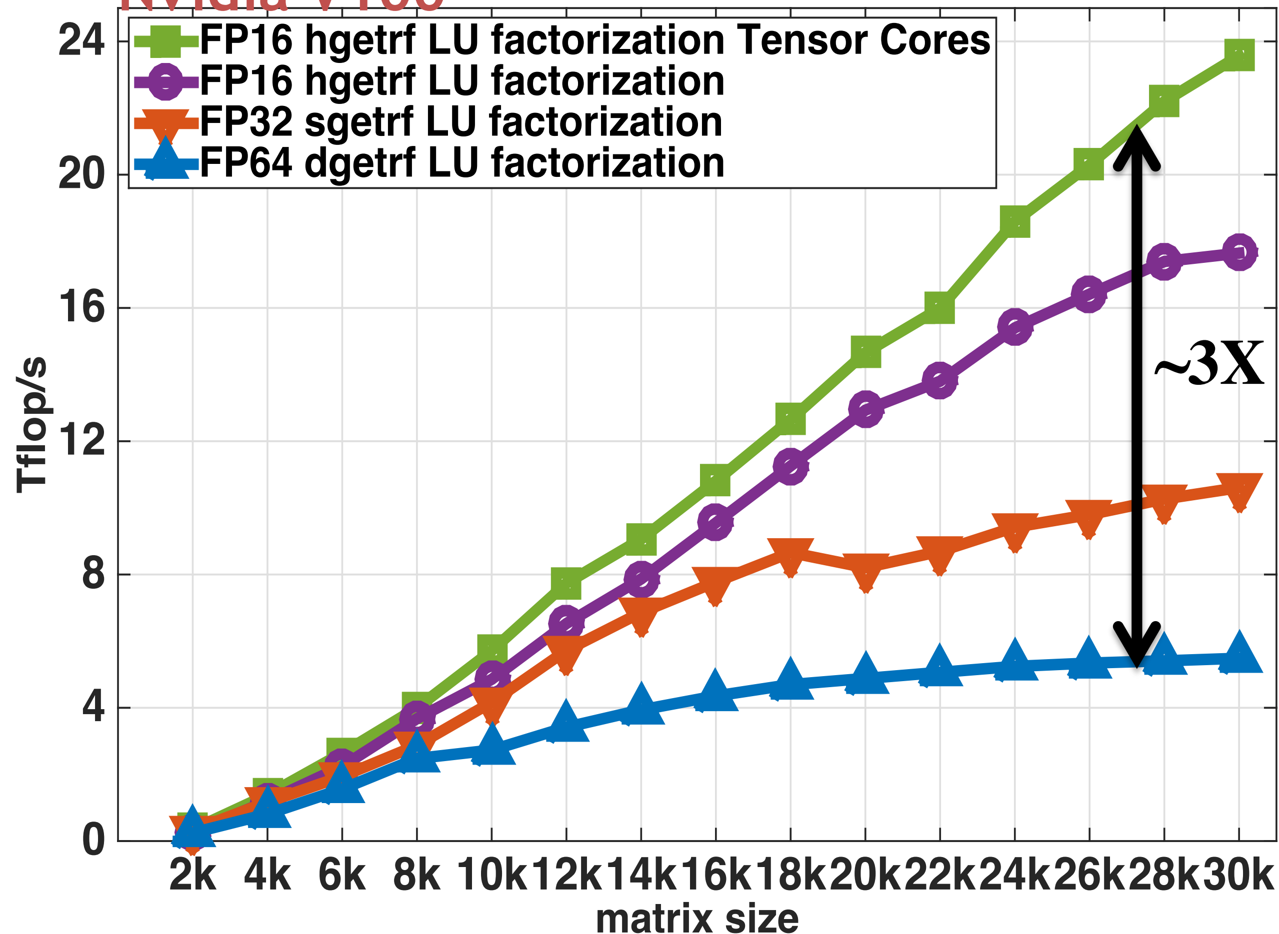
- Achieve higher performance → faster time to solution
- Reduce power consumption by decreasing the execution time → Energy Savings !!!

Reference:

1. Haidar, A., Wu, P., Tomov, S., Dongarra, J.
   Investigating Half Precision Arithmetic to Accelerate Dense Linear System Solvers,
   ScalA17: 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, ACM, Denver, Colorado, November 12-17, 2017.
2. Haidar, A., Tomov, S., Dongarra, J.
   Leveraging Half Precision in HPC,
   In preparation for ACM TOMS

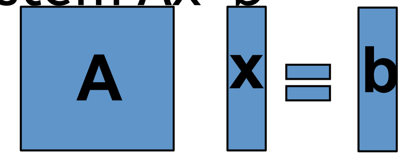
Leveraging Half Precision in HPC on V100 Motivation

Study of the LU factorization algorithm on Nvidia V100

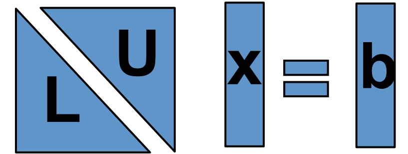


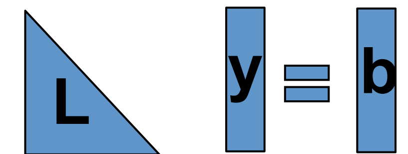
- LU factorization is used to solve a linear system Ax=b

A x = b

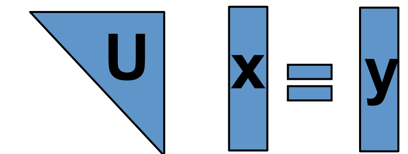


LUx = b





Ly = b



then

Ux = y

ICL ECP EXASCALE COMPUTING PROJECT

Leveraging Half Precision in HPC on V100
Iterative Refinement

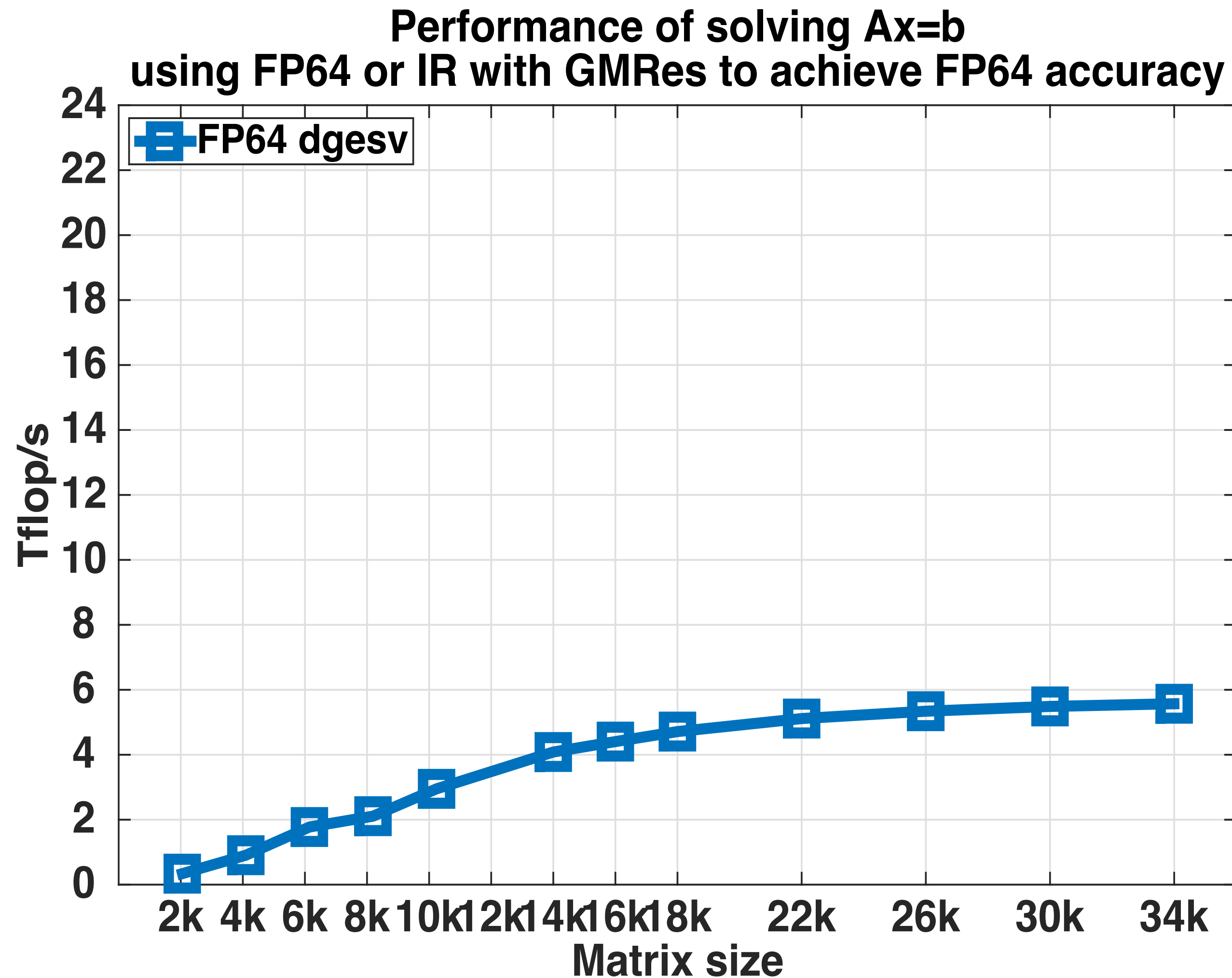
Idea: use lower precision to compute the expensive flops (LU $O(n^3)$) and then iteratively refine the solution in order to achieve the FP64 arithmetic

Iterative refinement for dense systems, $Ax = b$, can work this way.

L U = lu(A)		lower precision	$O(n^3)$
x = U\(L\b)		lower precision	$O(n^2)$
r = b – Ax		FP64 precision	$O(n^2)$
WHILE || r || not small enough			
1. find a correction "z" to adjust x that satisfy Az=r solving Az=r could be done by either:			
➢ z = U\(L\r)	Classical Iterative Refinement	lower precision	$O(n^2)$
➢ GMRes preconditioned by the LU to solve Az=r	Iterative Refinement using GMRes	lower precision	$O(n^2)$
2. x = x + z		FP64 precision	$O(n^1)$
3. r = b – Ax		FP64 precision	$O(n^2)$
END			

- ➢ Wilkinson, Moler, Stewart, & Higham provide error bound for SP fl pt results when using DP fl pt.
- ➢ It can be shown that using this approach we can compute the solution to 64-bit floating point precision.

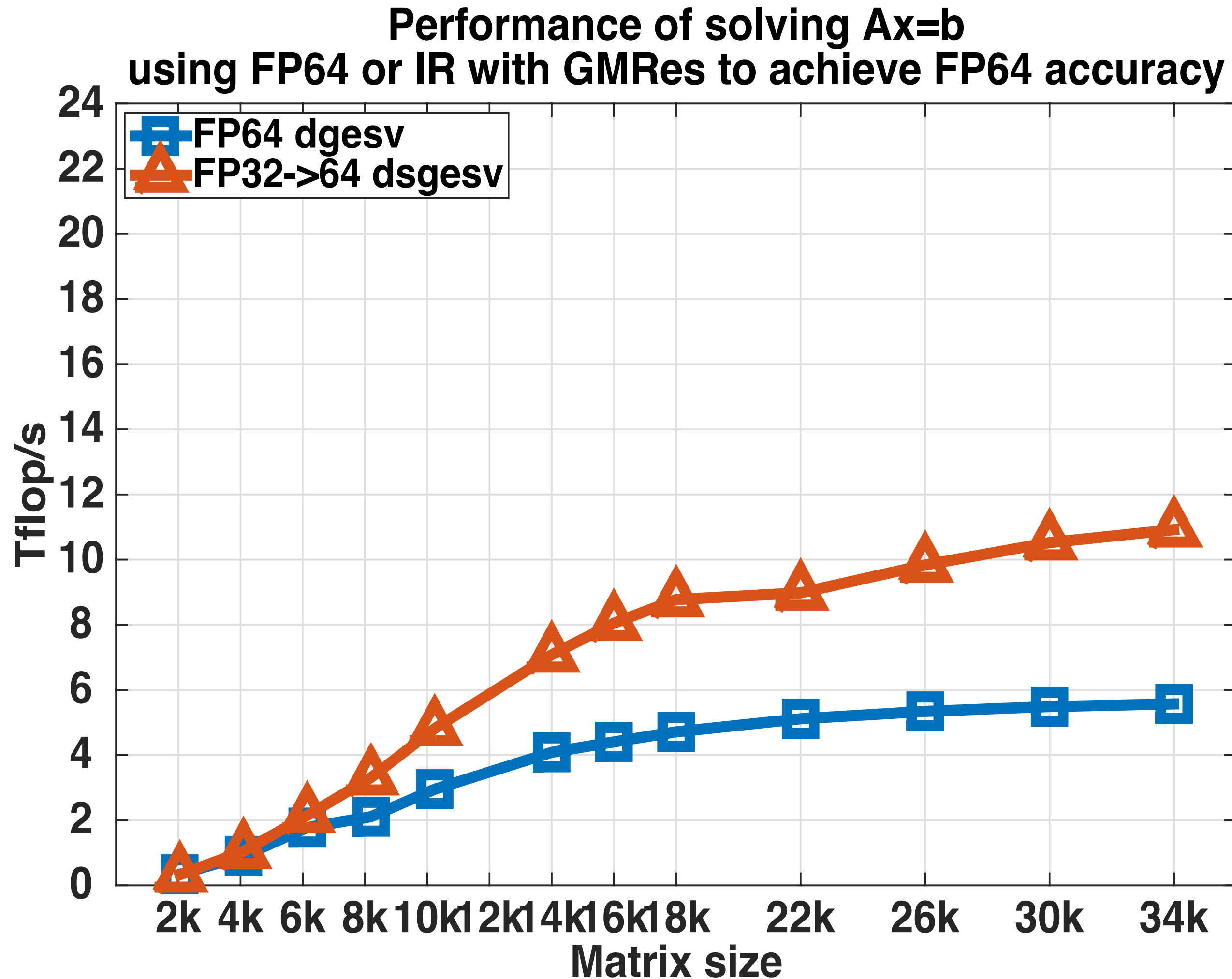
Leveraging Half Precision in HPC on V100
Performance Behavior



Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- solving Ax = b using **FP64 LU**

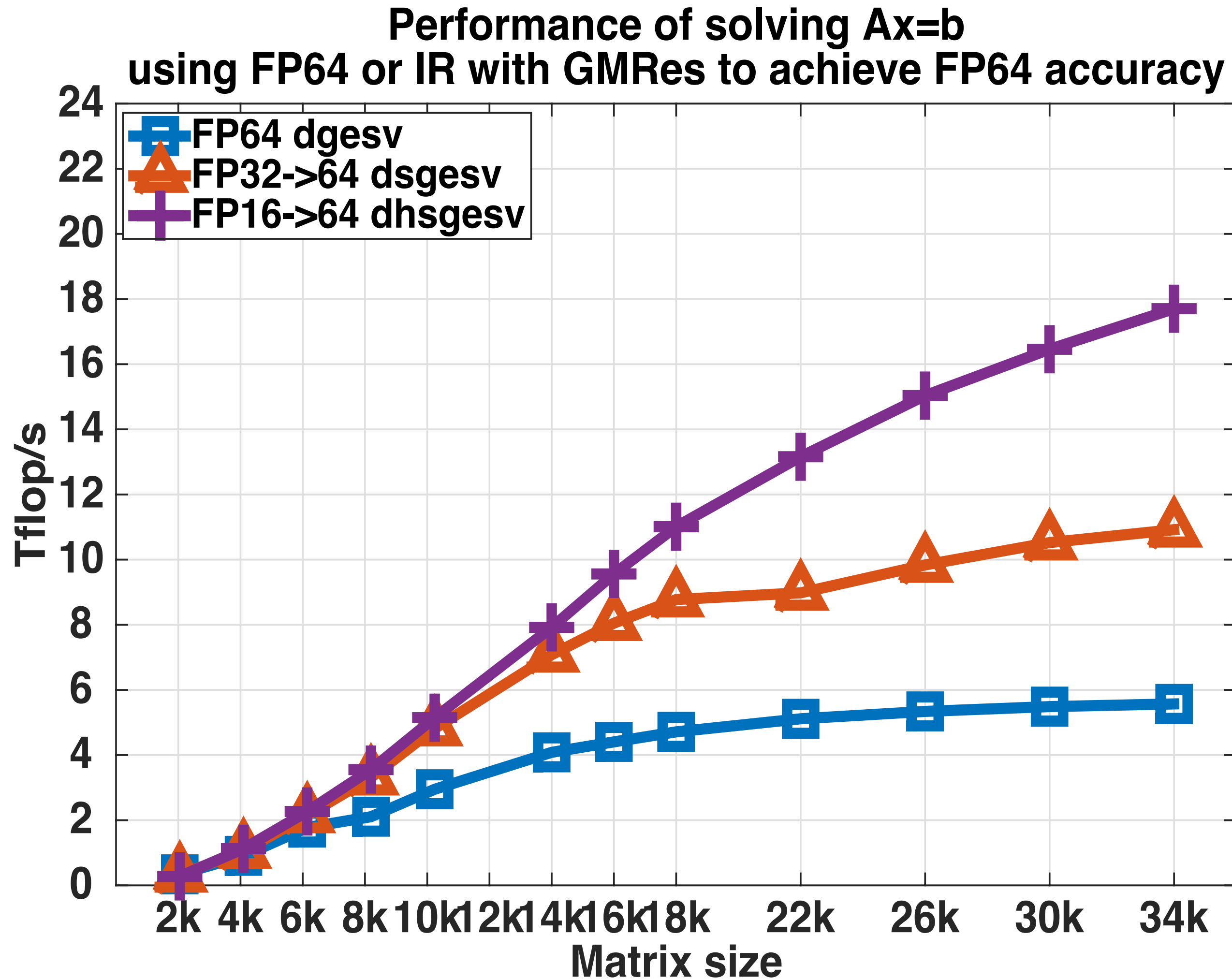
Leveraging Half Precision in HPC on V100
Performance Behavior



Flops = $2n^3/(3\ time)$
meaning twice higher is twice faster

- solving Ax = b using **FP64 LU**
- solving Ax = b using **FP32 LU** and iterative refinement to achieve FP64 accuracy

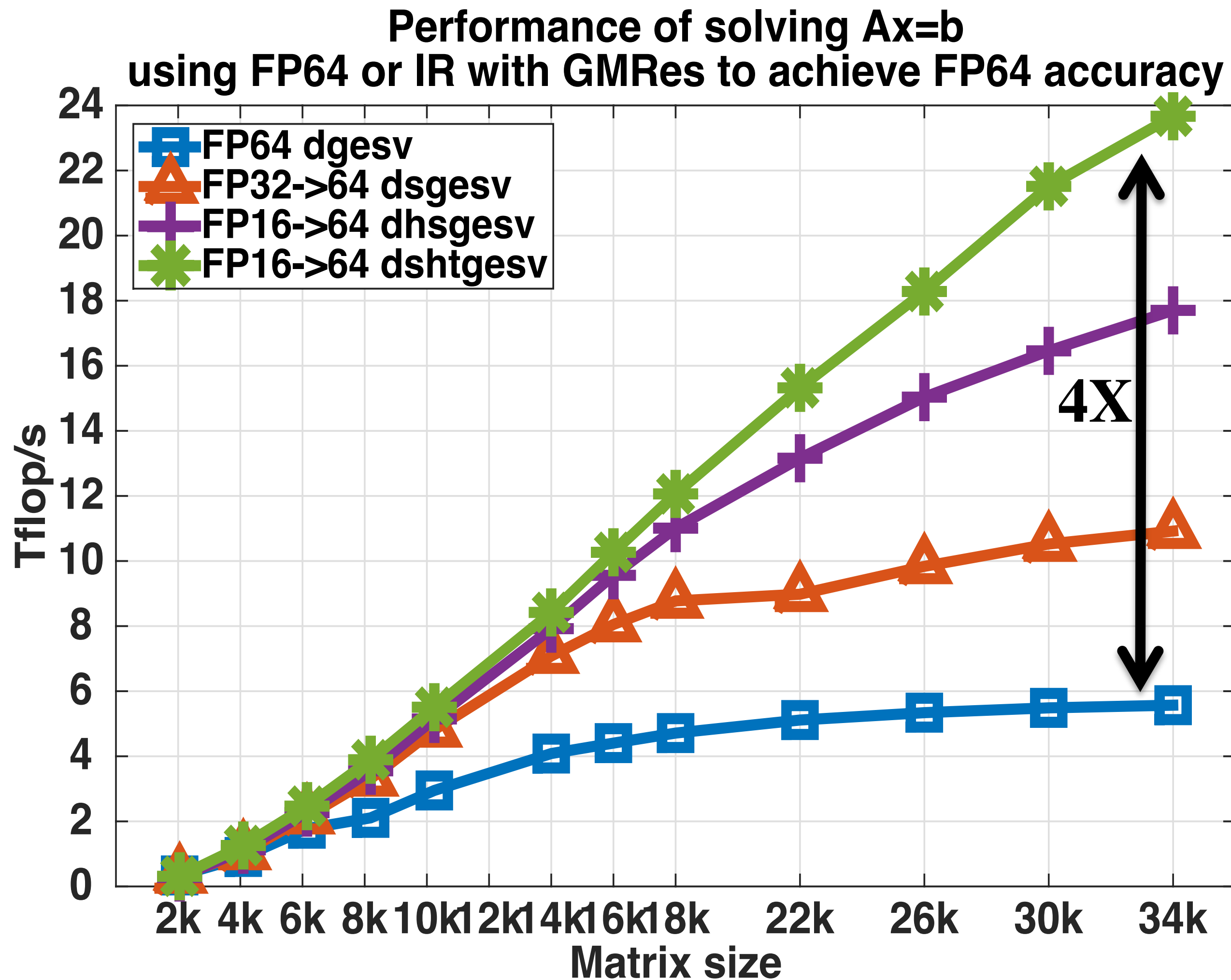
Leveraging Half Precision in HPC on V100
Performance Behavior



Flops = $2n^3/(3 \text{ time})$
meaning twice higher is twice faster

- solving Ax = b using **FP64 LU**
- solving Ax = b using **FP32 LU** and iterative refinement to achieve FP64 accuracy
- solving Ax = b using **FP16 LU** and iterative refinement to achieve FP64 accuracy

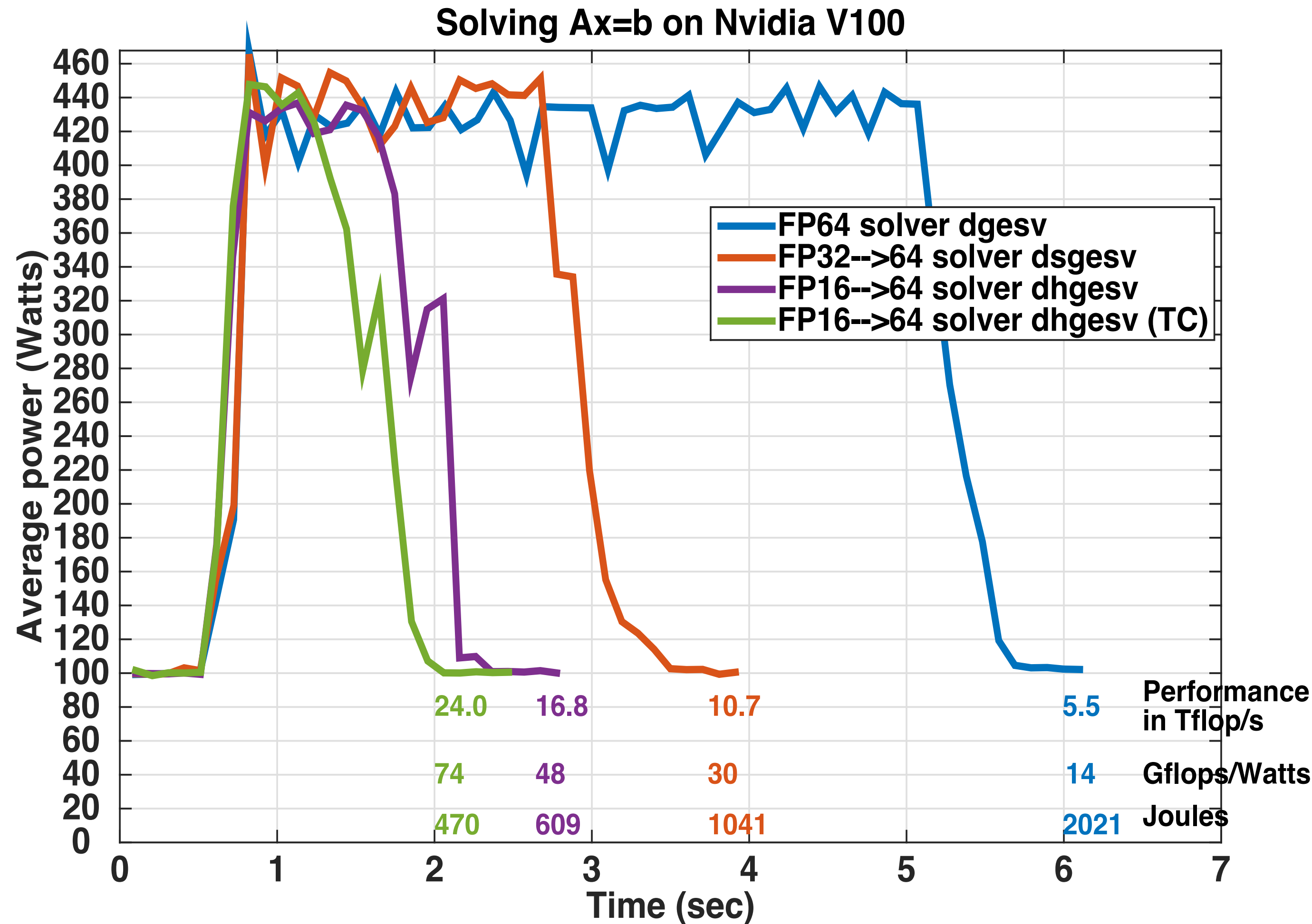
Leveraging Half Precision in HPC on V100
Performance Behavior



Flops = $2n^3/(3\ \text{time})$
meaning twice higher is twice faster

- solving Ax = b using **FP64 LU**
- solving Ax = b using **FP32 LU** and iterative refinement to achieve FP64 accuracy
- solving Ax = b using **FP16 LU** and iterative refinement to achieve FP64 accuracy
- solving Ax = b using **FP16 Tensor Cores LU** and iterative refinement to achieve FP64 accuracy

Leveraging Half Precision in HPC
Power awareness



Mixed precision techniques can provide a large gain in energy efficiency

- Power consumption of the FP64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 5.5 Tflop/s and requires about 2021 joules providing about 14 Gflops/Watts.
- Power consumption of the mixed precision FP32→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 10.7 Tflop/s and requires about 1041 joules providing about 30 Gflops/Watts.
- Power consumption of the mixed precision FP16→64 algorithm to solve Ax=b for a matrix of size 34K, it achieve 16.8 Tflop/s and requires about 609 joules providing about 48 Gflops/Watts.
- Power consumption of the mixed precision FP16→64 TC algorithm using Tensor Cores to solve Ax=b for a matrix of size 34K, it achieve 24 Tflop/s and requires about 470 joules providing about 74 Gflops/Watts.

MAGMA Sparse

- C-based, uses structures to pack complex objects
- Comprehensive set of SpMV formats/ routines
- Vector operations and building blocks
- Iterative solvers (Krylov, relaxation)
- Preconditioners
- Focus on GPU hardware

MAGMA SPARSE

ROUTINES	BiCG, BiCGSTAB, Block-Asynchronous Jacobi, CG, CGS, GMRES, IDR, Iterative refinement, LOBPCG, LSQR, QMR, TFQMR
PRECONDITIONERS	ILU / IC, Jacobi, ParILU, ParILUT, Block Jacobi, ISAI
KERNELS	SpMV, SpMM
DATA FORMATS	CSR, ELL, SELL-P, CSR5, HYB

SpMV performance (DP)
Matrices from SuiteSparse Matrix Collection

NVIDIA P100, 5.3 GFLOPs peak (DP), 720 GB/s

MAGMA Sparse

Sparse linear algebra objects are handled with **magma_z_matrix** structure containing info about:

- memory location
- storage format
- data, size, nnz, ...
  - magma_zmconvert(**A**, **&B**, Magma_CSR, Magma_ELL, queue);
  - magma_zmtransfer(**A**, **&B**, Magma_CPU, Magma_DEV, queue);
  - magma_z_spmv(alpha, **A**, x, beta, y, queue);

    ...

```
typedef struct magma_z_solver_par
{
    magma_solver_type  solver;                    // solver type
    magma_int_t        version;                   // sometimes there are different versions
    double             atol;                       // absolute residual stopping criterion
    double             rtol;                      // relative residual stopping criterion
    magma_int_t        maxiter;                   // upper iteration limit
    magma_int_t        restart;                   // for GMRES
    magma_ortho_t      ortho;                     // for GMRES
    magma_int_t        numiter;                   // feedback: number of needed iterations
    magma_int_t        spmv_count;                // feedback: number of needed SpMV
    double             init_res;                  // feedback: initial residual
    double             final_res;                 // feedback: final residual
    double             iter_res;                  // feedback: iteratively computed residual
    real_Double_t      runtime;                   // feedback: runtime needed
    real_Double_t      *res_vec;                  // feedback: array containing residuals
    real_Double_t      *timing;                   // feedback: detailed timing
    magma_int_t        verbose;                   // print residual ever 'verbose' iterations
    magma_int_t        num_eigenvalues;           // number of EV for eigensolvers
    magma_int_t        ev_length;                 // needed for framework
    double             *eigenvalues;              // feedback: array containing eigenvalues
    magmaDoubleComplex_ptr      eigenvectors;     // feedback: array containing eigenvectors on DEV
    magma_int_t        info;                      // feedback: did the solver converge etc.
} magma_z_solver_par;
```

MAGMA Sparse

Sparse linear algebra objects are handled with **magma_z_matrix** structure containing info about:

- memory location
- storage format
- data, size, nnz, …
  - magma_zmconvert(**A**, **&B**, Magma_CSR, Magma_ELL, queue);
  - magma_zmtransfer(**A**, **&B**, Magma_CPU, Magma_DEV, queue);
  - magma_z_spmv(alpha, **A**, x, beta, y, queue);

    …

```
typedef struct magma_z_solver_par
{
    magma_solver_type  solver;              // solver type
    magma_int_t        version;             // sometimes there are different versions
    double             atol;                 // absolute residual stopping criterion
    double             rtol;                // relative residual stopping criterion
    magma_int_t        maxiter;             // upper iteration limit
    magma_int_t        restart;             // for GMRES
    magma_ortho_t      ortho;               // for GMRES
    magma_int_t        numiter;             // feedback: number of needed iterations
    magma_int_t        spmv_count;          // feedback: number of needed SpMV
    double             init_res;            // feedback: initial residual
    double             final_res;           // feedback: final residual
    double             iter_res;            // feedback: iteratively computed residual
    real_Double_t      runtime;             // feedback: runtime needed
    real_Double_t      *res_vec;            // feedback: array containing residuals
    real_Double_t      *timing;             // feedback: detailed timing
    magma_int_t        verbose;             // print residual ever 'verbose' iterations
    magma_int_t        num_eigenvalues;     // number of EV for eigensolvers
    magma_int_t        ev_length;           // needed for framework
    double             *eigenvalues;        // feedback: array containing eigenvalues
    magmaDoubleComplex_ptr      eigenvectors;   // feedback: array containing eigenvectors on DEV
    magma_int_t        info;                // feedback: did the solver converge etc.
} magma_z_solver_par;
```

For calling a solver from MAGMA-sparse, an additional structure is used, containing information about the solver and the preconditioner:

- **opts**.solver_par.solver = Magma_CG;
- **opts**.solver_par.rtol = 1e-10;
- **opts**.solver_par.maxiter = 1000;
- **opts**.precond_par.solver = Magma_JACOBI;
- magma_z_solver(A, b, &x, **&opts**, queue);

All structures are defined and documented in **magmasparse_types.h**

MAGMA Sparse

```
// Initialize MAGMA and create some LA structures.
magma_init();
magma_queue_t queue; magma_queue_create( 0, &queue );

// Pass the system to MAGMA.
magma_d_matrix A={Magma_CSR}, dA={Magma_CSR}, b={Magma_CSR}, db={Magma_CSR}, x={Magma_CSR}, dx={Magma_CSR};
magma_dcsrset( m, m, row, col, val, &A, queue );
magma_dvset( m, 1, rhs, &b, queue );
magma_dvset( m, 1, sol, &x, queue );

// Copy the system to the device (optional, only necessary if using the GPU)
magma_dmtransfer( A, &dA, Magma_CPU, Magma_DEV, queue );
magma_dmtransfer( b, &db, Magma_CPU, Magma_DEV, queue );
magma_dmtransfer( x, &dx, Magma_CPU, Magma_DEV, queue );

// Choose a solver, preconditioner, etc. - see documentation for options.
magma_dopts opts; opts.solver_par.solver = Magma_PCG; opts.precond_par.solver = Magma_JACOBI;
magma_dsolverinfo_init( &opts.solver_par, &opts.precond_par, queue );
magma_d_precondsetup( dA, db, &opts.solver_par, &opts.precond_par, queue );

// to solve the system, run:
magma_d_solver( dA, db, &dx, &opts, queue );

// Then copy the solution back to the host and extract it to the application code
magma_dmfree( &x, queue );
magma_dmtransfer( dx, &x, Magma_CPU, Magma_DEV, queue );
magma_dvget( x, &m, &n, &sol, queue );

// Free the allocated memory and finalize MAGMA
magma_dmfree( &dx, queue ); magma_dmfree( &db, queue ); magma_dmfree( &dA, queue );
magma_queue_destroy( queue );
magma_finalize();

// From here on, the application code may continue with the solution in sol...
for (i = 0; i < 20; ++i) {
    printf("%.4f\n", sol[i]);
}
```

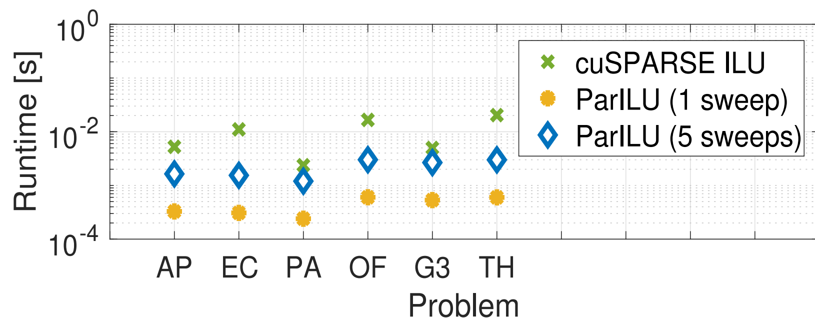
Interactive example...

Parallel Preconditioning in MAGMA-sparse

ECP PEEKS ICL

Parallel Incomplete Factorization Preconditioners

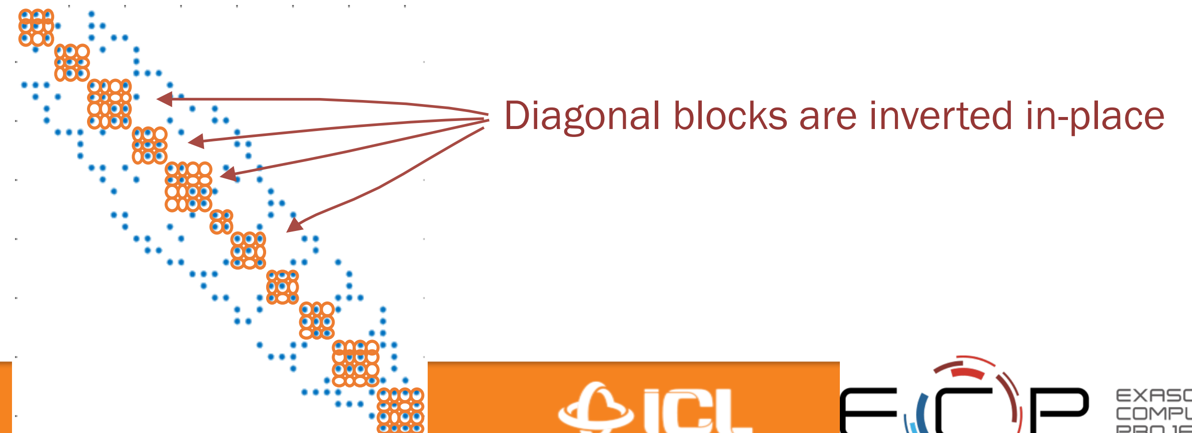
- Kernel for computing **ILU preconditioners** in parallel.
- Implementations for hardware supporting **OpenMP** or **CUDA** available in **MAGMA-sparse.**
- Reported speedups up to 10x when comparing against NVIDIA's cuSPARSE library (version 9.0).



Performance comparison on NVIDIA's Volta architecture.

Parallel Triangular solves

1. Based on approximating inverse of triangular factors (ISAI).
   - Replaces trsv with SPAI
   - Sparsity pattern & Accuracy of SPAI user-controlled
2. Based on Relaxation steps of parallel iterative method.
   - Jacobi / block-Jacobi
   - Block-Jacobi based on batched matrix inversion
   - Flexible block size, supervariable amalgation



Diagonal blocks are inverted in-place

Sparse direct multifrontal solvers

DOD CREATE

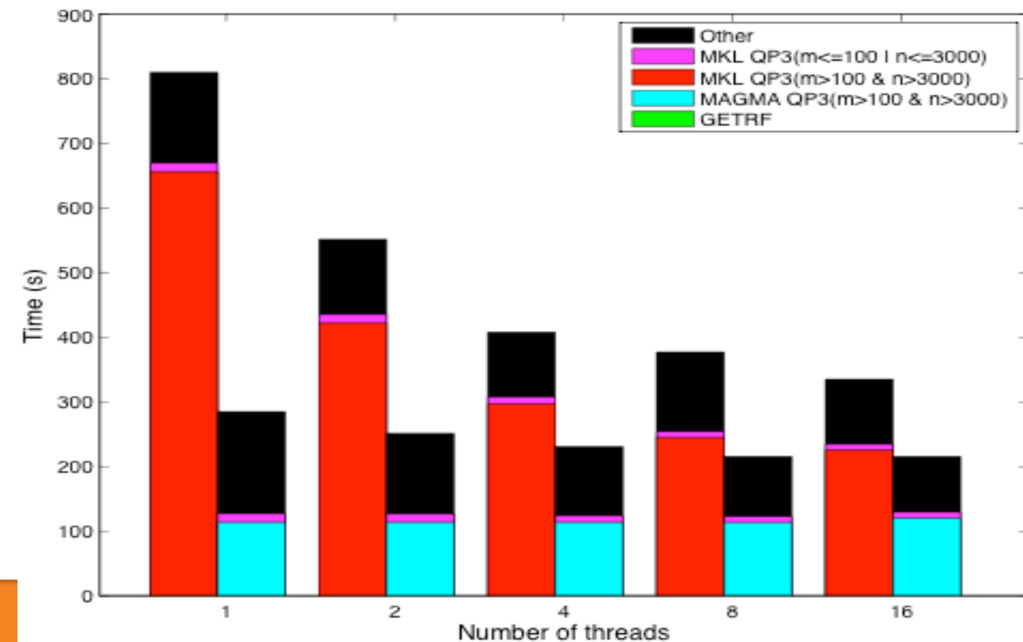
“High-Performance Numerical Libraries with Support for Low-Rank Matrix Computations”

Accelerating StruMF HSS solver

- **StruMF computes preconditioner for Hierarchical Semiseparable (HSS) matrices using low-rank approximations in a direct multifrontal solver**
  - **We developed and use a GPU-accelerated QP3 algorithm for the low-rank approximations**
  - **2x performance improvement using K40c GPU vs. 16 Intel Sandy Bridge cores@2.6 GHz**

    [I. Yamazaki, A. Napov, S. Tomov, and J. Dongarra, *“Computing Low--Rank Approximation of Dense Submatrices in a Hierarchically Semiseparable Matrix and its GPU Acceleration”*, UTK Technical Report, August, 2013.]

Factorization times in a double precision HSS solver. The pairs of bars show times from using one to 16 cores. The bar on the left is the time on the CPU cores vs. the time on the right using the same number of CPU cores plus a Kepler GPU.

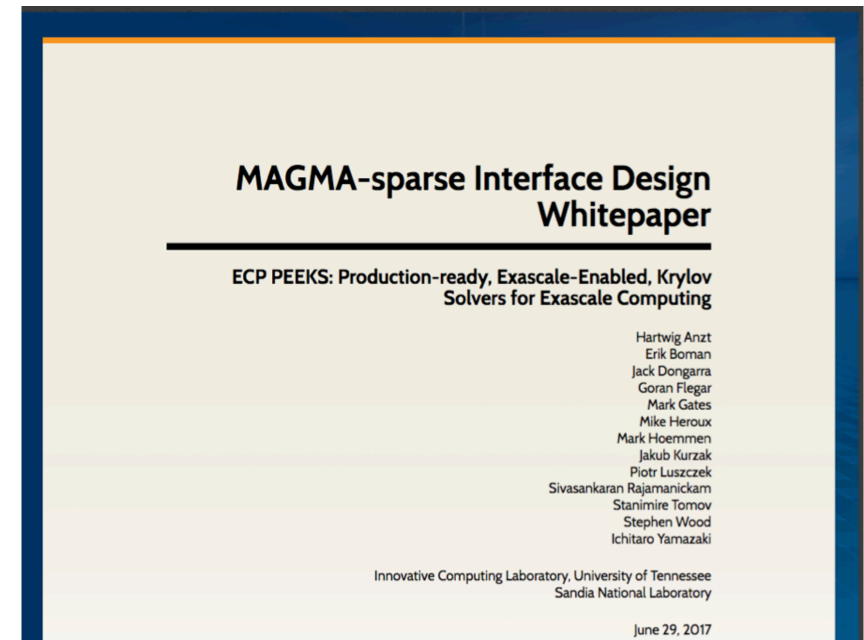


MAGMA-sparse interface to Trilinos

- **ECP PEEKS** effort develops interface between Trilinos and MAGMA-sparse.

- Higher goal: Design a general standard interface for sparse LA.
- Initial **interface design document** published in June 2017. (available on PEEKS project webpage http://icl.utk.edu/peeks/)
- **Living document** - may be updated over time…

  Your feedback is welcome!



MAGMA-sparse Interface Design Whitepaper

ECP PEEKS: Production-ready, Exascale-Enabled, Krylov Solvers for Exascale Computing

Hartwig Anzt
Erik Boman
Jack Dongarra
Goran Flegar
Mark Gates
Mike Heroux
Mark Hoemmen
Jakub Kurzak
Piotr Luszczek
Sivasankaran Rajamanickam
Stanimire Tomov
Stephen Wood
Ichitaro Yamazaki

Innovative Computing Laboratory, University of Tennessee
Sandia National Laboratory

June 29, 2017

- Populate all functionality via the xSDK ecosystem.

Collaborators and Support

MAGMA team

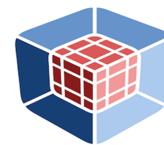
http://icl.cs.utk.edu/magma

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

MANCHESTER 1824
The University of Manchester

University of Manchester