# Sparse and Batched Numerical Linear Algebra

# with

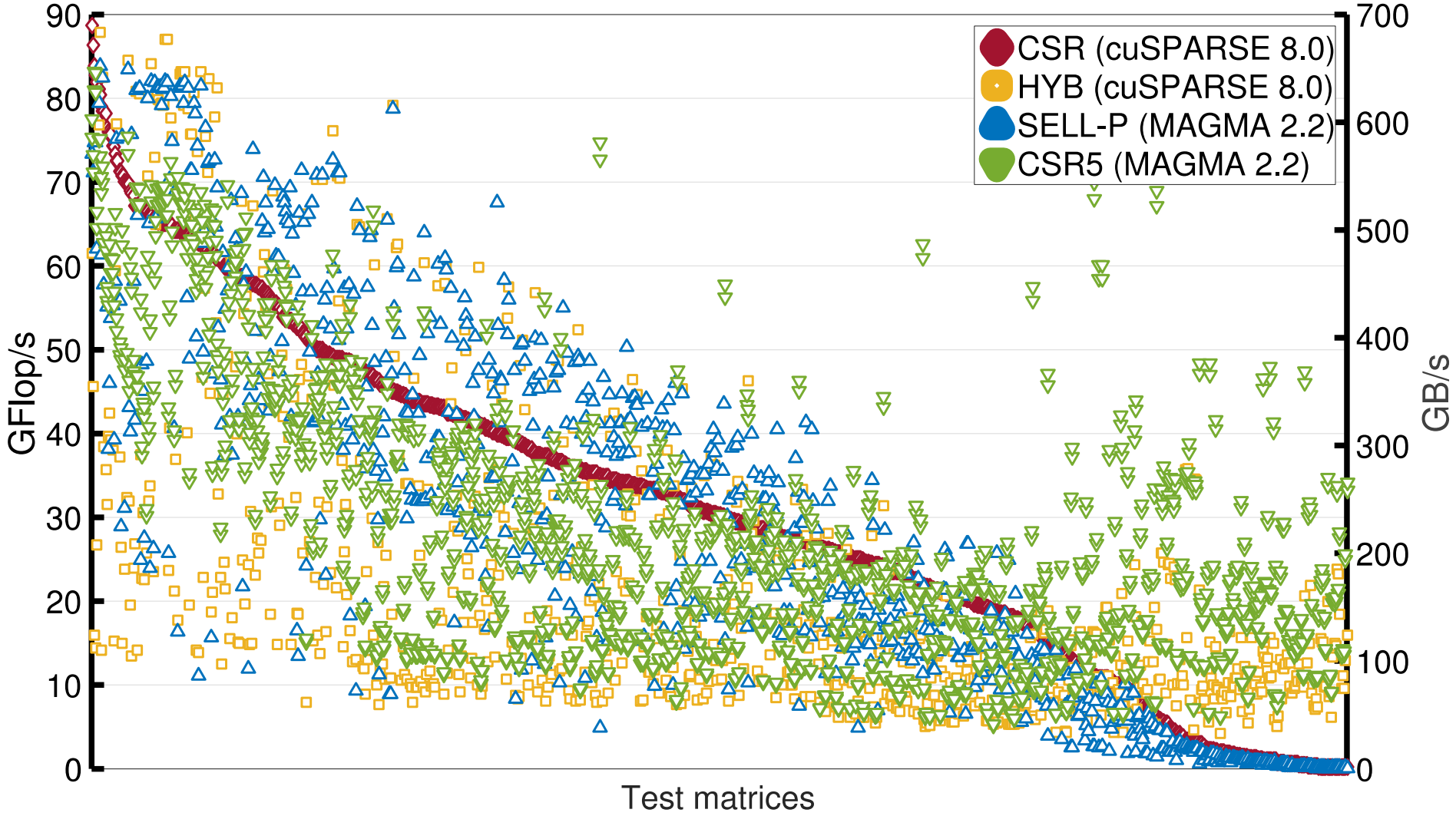
# MAGMA

*Piotr Luszczek*

# MAGMA Sparse: Scope

| Scope | Content |
|---|---|
| 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 |

# MAGMA Sparse Performance



# MAGMA Sparse Overview

- Sparse linear algebra objects are in magma_z_matrix with:
  - 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 )
- For calling a MAGMA-sparse solver, a 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

# sparse/include/magmasparse_types.h

- Solver type and version
- Absolute and relative residual
- Upper bound: iteration count
- Restart and ortho. (GMRES)
- Verbosity level
- Number of eig-vals

- Number of needed iterations, SpMV's
- Initial, final, and computed residual and an array thereof
- Runtime needed and detailed timing
- Eigenvalues and eigenvectors
- Convergence achieved flag

```
// 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}, r={Magma_CSR}, dx={Magma_CSR};
magma_dcrset( 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 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_dmtrasfer( 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();
```

# MAGMA Sparse Bibliography


- E. Chow, H. Anzt, and J. Dongarra. Asynchronous Iterative Algorithm for Computing Incomplete Factorizations on GPUs. In Lecture Notes in Comp. Sci., vol. 9137, pp.1–16, Jul12 –16 2015.
- H. Anzt, E. Chow, J. Saak, and J. Dongarra. Updating Incomplete Factorization Preconditioners for Model Order Reduction. Numerical Algorithms, 2016.
- H. Anzt, E. Chow, D.B. Szyld, and J. Dongarra. Domain Overlap for Iterative Sparse Triangular Solves on GPUs. In Lecture Notes in Computer Science and Engineering, 2016.
- H. Anzt, J. Dongarra, M. Kreutzer, and M. Koehler. Efficiency of general Krylov methods on GPUs – An experimental study. In The Sixth International Workshop on Accelerators and Hybrid Exascale Systems (AsHES), 2016.
- H. Anzt, E. Chow, T. Huckle, J. Dongarra. Batched Generation of Incomplete Sparse Approximate Inverses on GPUs. In ScaLA'16

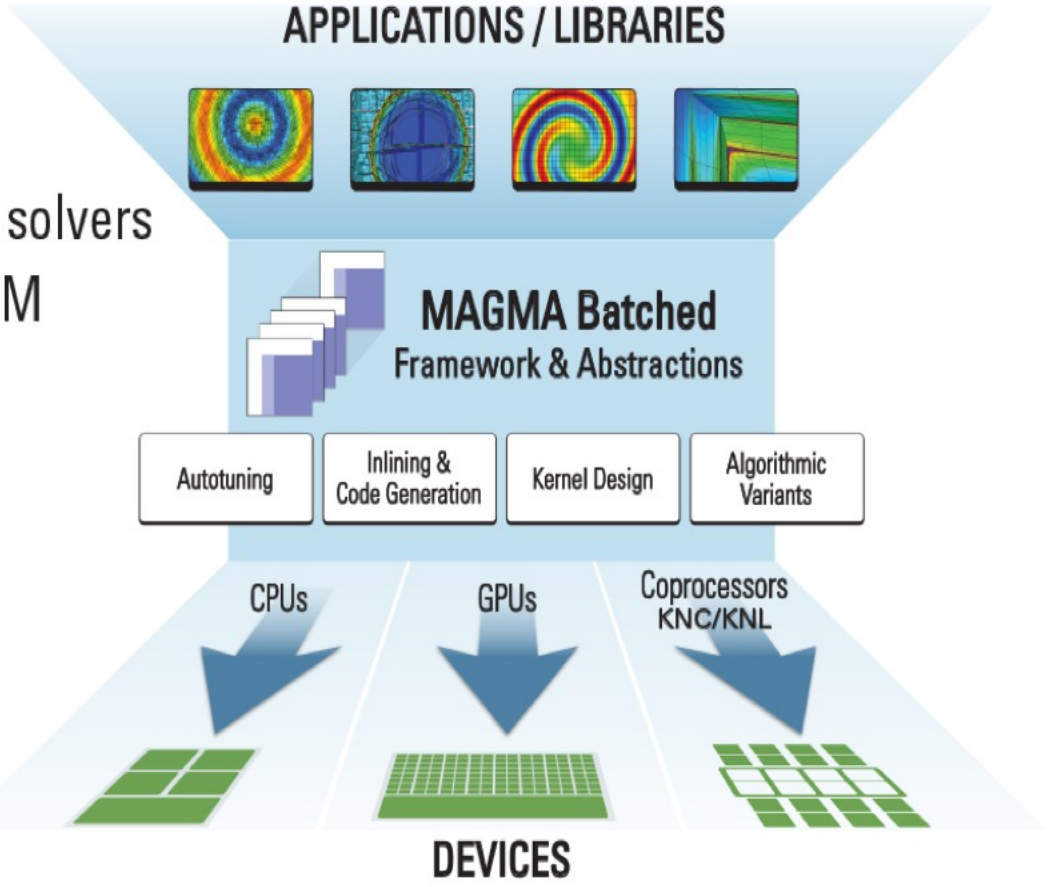
# Batched Routines

*Piotr Luszczek*

# 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

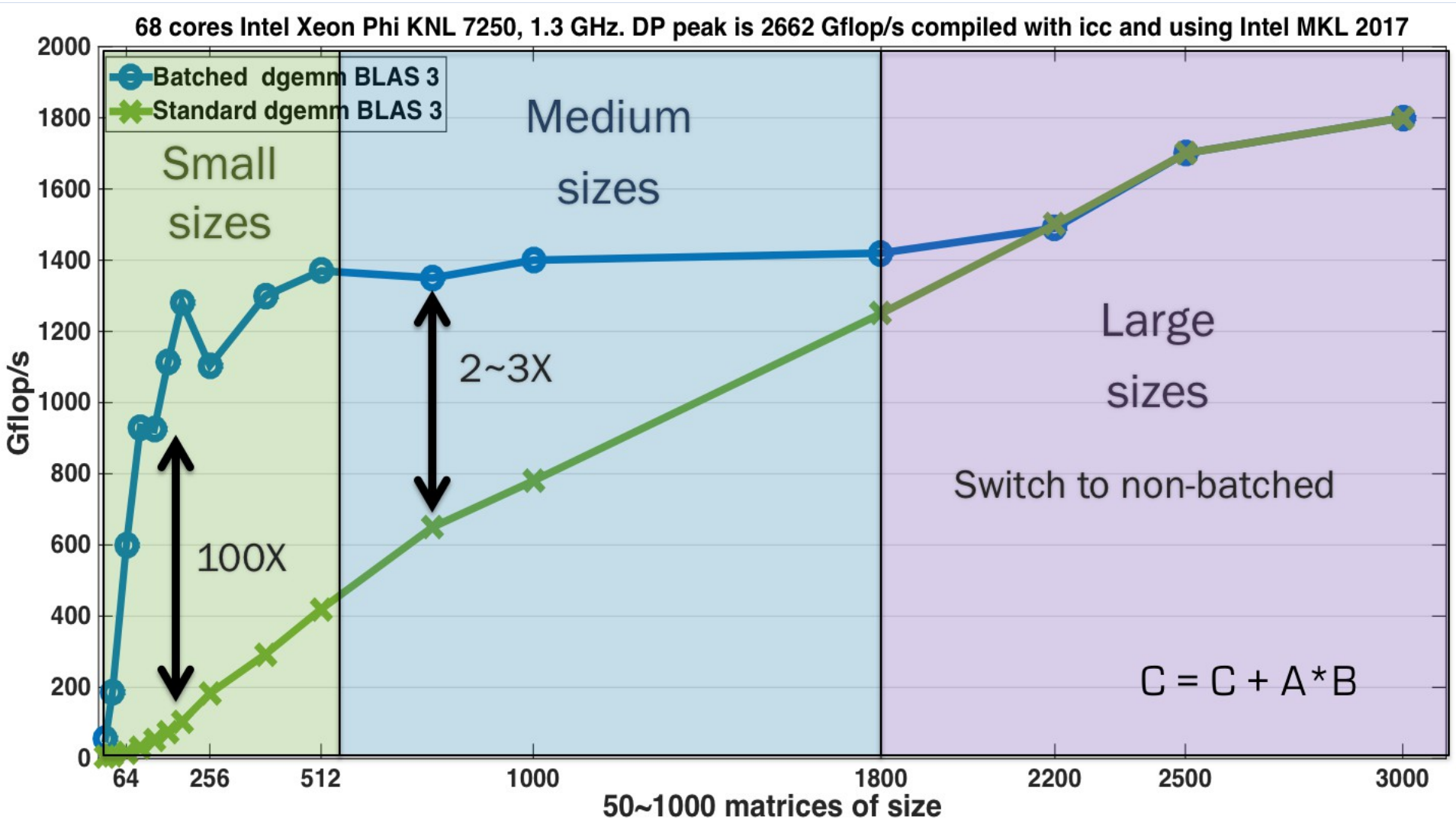
| | |
|---|---|
| LU, QR, and Cholesky | ✓ |
| Solvers and matrix inversion | ✓ |
| All BLAS 3 (fixed + variable) | ✓ |
| SYMV, GEMV (fixed + variable) | ✓ |

# Small Problem Sizes for Linear Systems Occur Often in Science

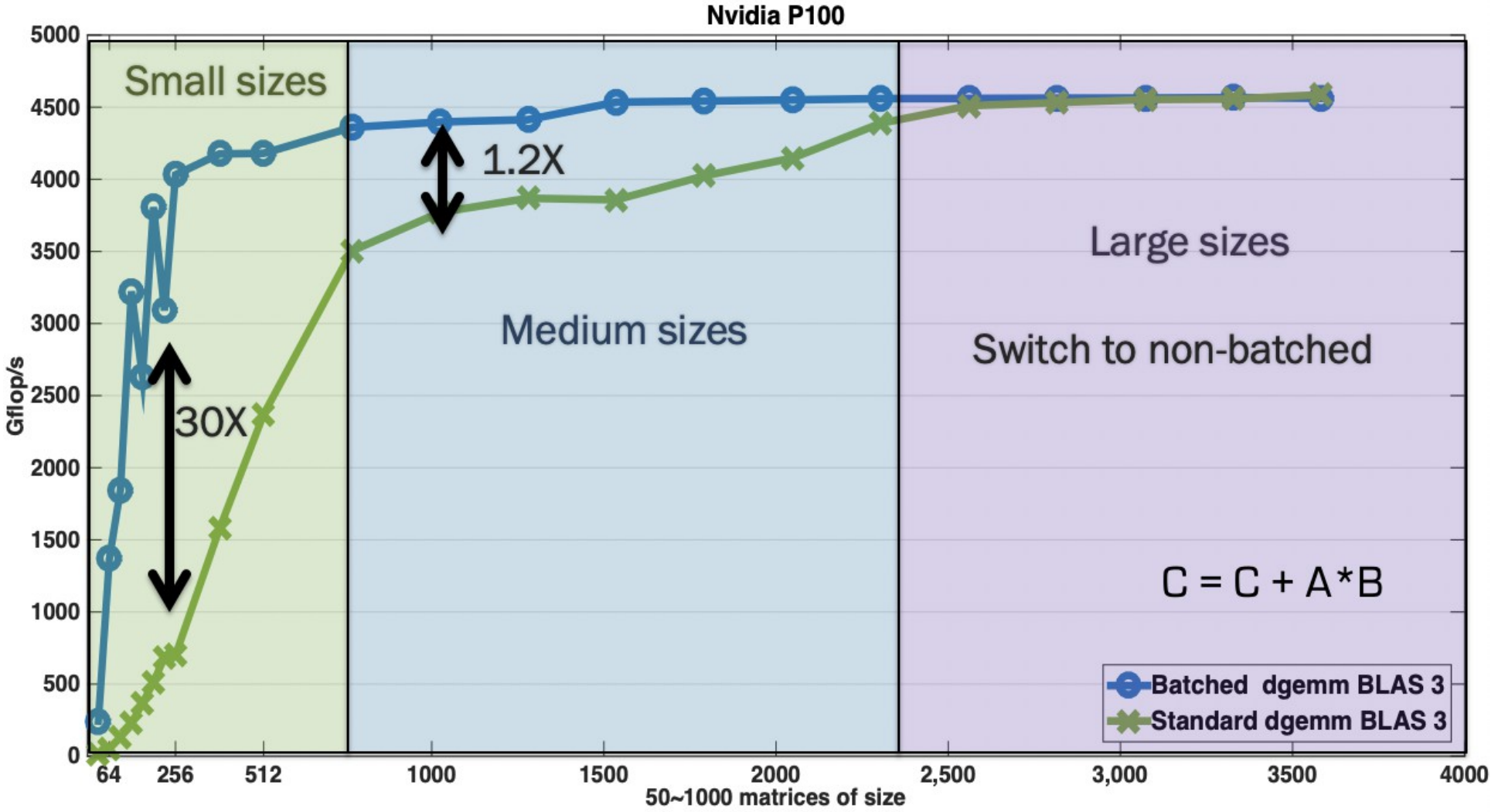
- Machine learning / DNNs
- Data mining / analytics
- High-order FEM,
- Graph analysis,
- Neuroscience,
- Astrophysics,
- Quantum chemistry,
- Signal processing
- and more...

- Fixed-size batches
- Variable-size batches
- Dynamic batches
- Tensors

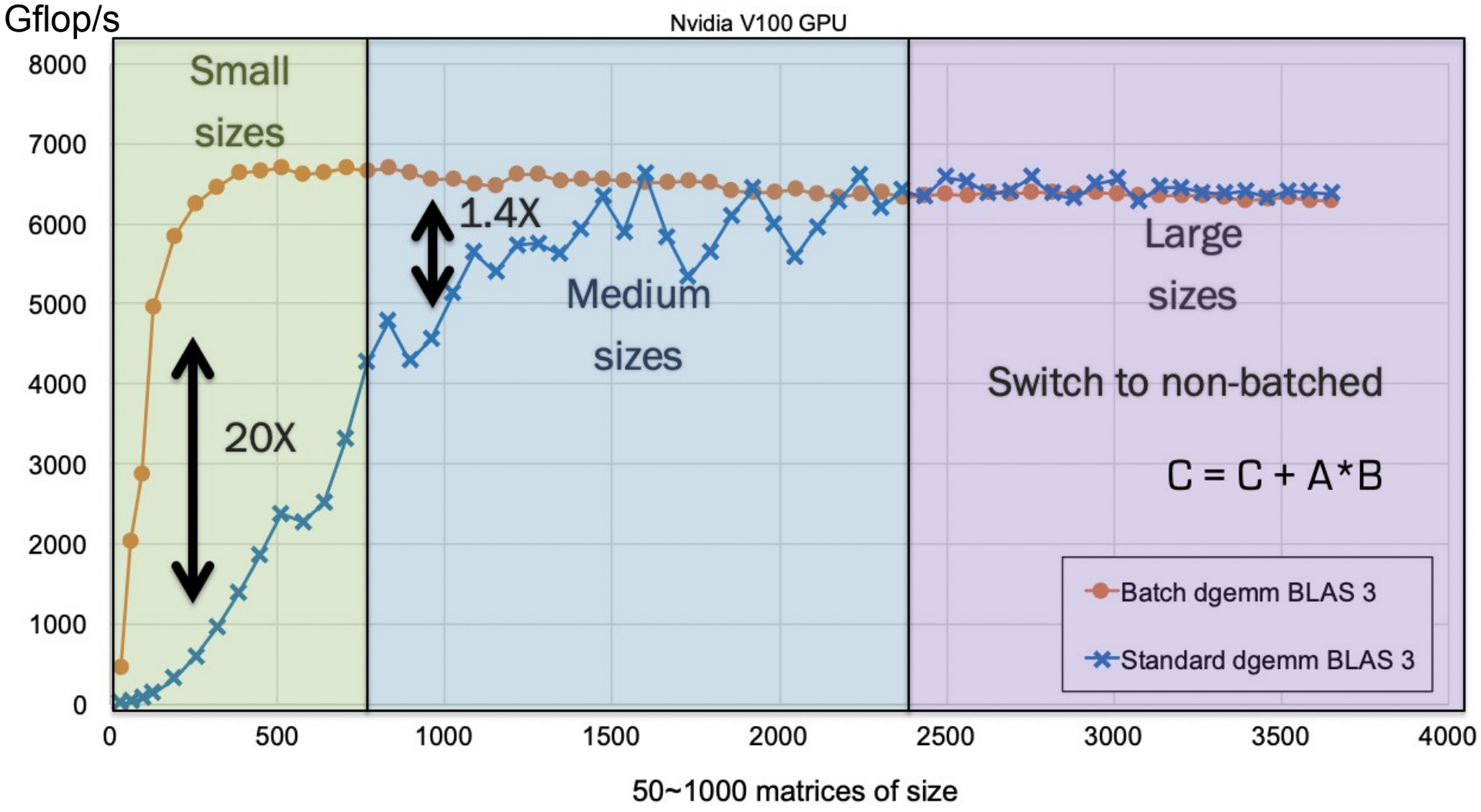
# Batched BLAS Origin: Performance on Xeon Phi



# Batched BLAS Origin: Performance on Pascal



# Batched BLAS Origin: Performance on Volta



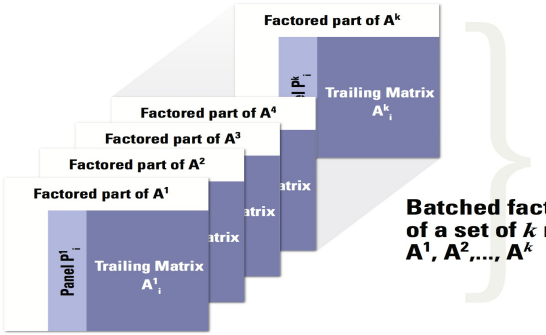
# MAGMA Batched Optimization Techniques Overview

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

# MAGMA Batched Performance

## ROUTINES

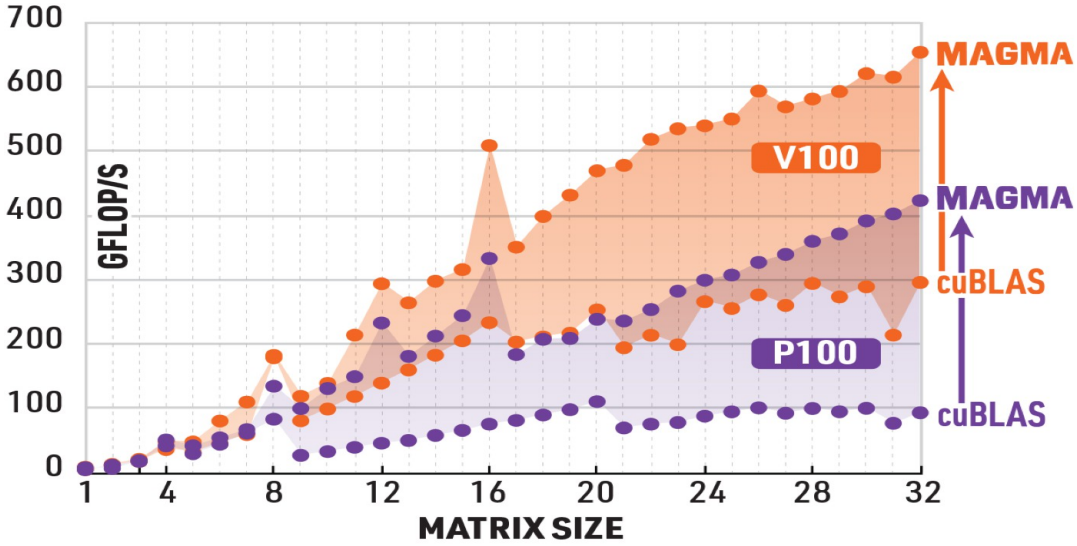
- LU, QR, and Cholesky ✓
- Solvers and matrix inversion ✓
- All BLAS 3 (fixed + var) ✓
- SYMV, GEMV (fixed + var) ✓




**Batched factorization of a set of $k$ matrices $A^1, A^2, \ldots, A^k$**

## PERFORMANCE OF BATCHED LU

in double precision arithmetic on 1 million matrices



# Batched BLAS Storage

- Array of pointers
  - float * A[ ] = { ptr_1, ptr_2, ptr_3 };
- Strided
  - Fixed stride
    - float * A; size_t stride = STRIDE;
  - Variable stride
    - float *A; size_t strides[ ] = { stride_1, stride_2, stride_3 };
- Interleaved (AKA compact) storage
  - float A[ ] = {
    a_1_1, b_1_1, ..., z_1_1,
    a_2_1, b_2_1, ..., z_2_1,
    a_1_2, b_1_2, ..., z_1_2,
    a_2_2, b_2_2, ..., z_2_2 };

# Batched BLAS Standardization Efforts

| Type | Name | | option args (trans) | scaling args (alpha) | sizes (m, n, k, ld*) | 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 |

# Towards Standard Batched BLAS API

- Workshops on Batched, Reproducible, and Reduced Precision BLAS
  - University of Tennessee, Knoxville, May 18-19, 2016, http://bit.ly/Batch-BLAS-2016
  - Georgia Tech, Computational Science and Engineering, Atlanta, GA, February 23-25, 2017, http://bit.ly/Batch-BLAS-2017
- ISC and SC BoFs
  - SC17, ISC18, SC18

- Draft Reports
  - https://www.dropbox.com/s/olocmipyxfvcaui/batched_api_03_30_2016.pdf
- Batched BLAS Poster
  - https://www.dropbox.com/s/ddkym76fapddf5c/Batched%20BLAS%20Poster%2012.pdf
- Batched BLAS Slides
  - https://www.dropbox.com/s/kz4fhcipz3e56ju/BatchedBLAS-1.pptx
- 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