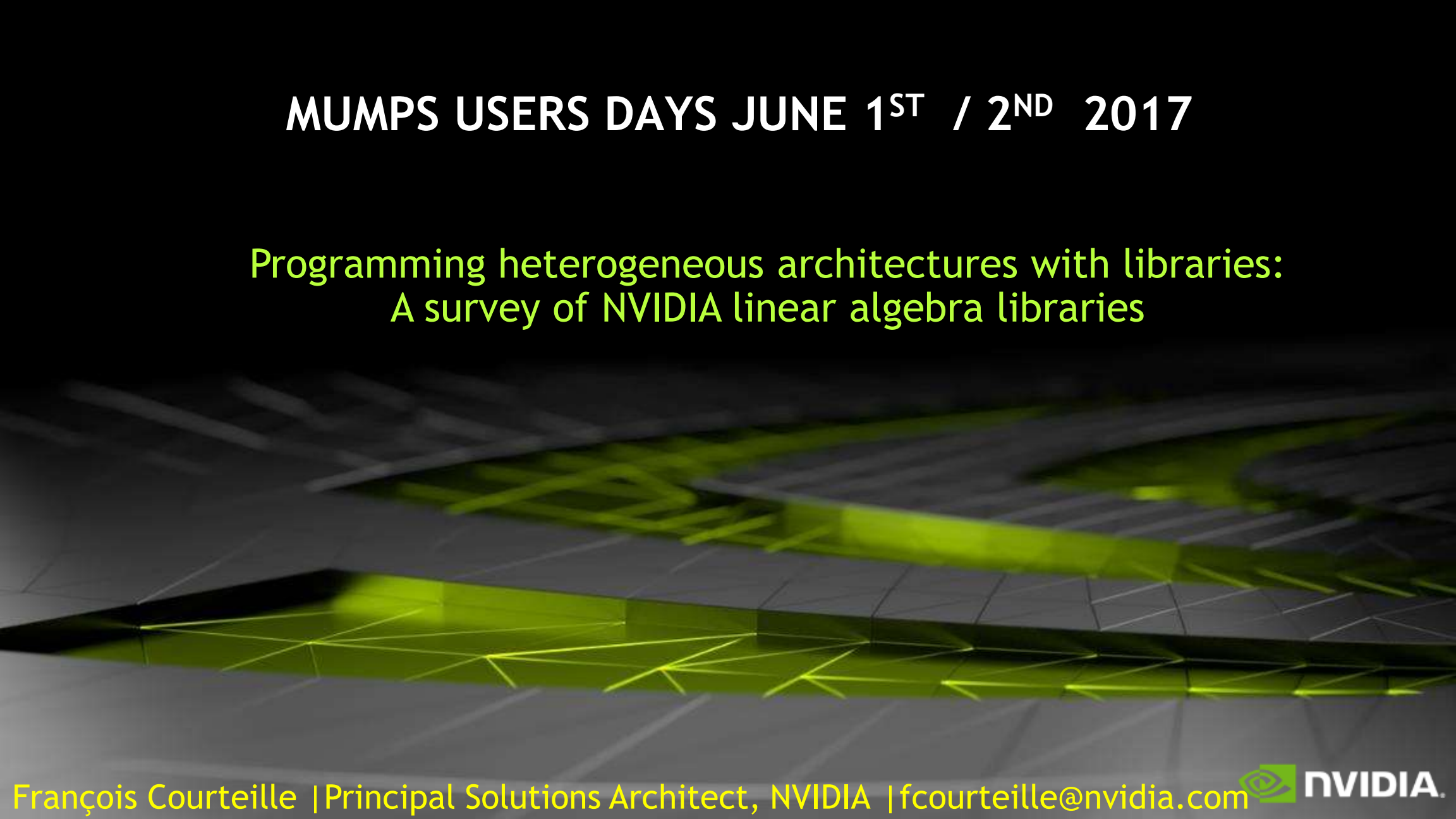
MUMPS USERS DAYS JUNE 1ST / 2ND 2017

Programming heterogeneous architectures with libraries: A survey of NVIDIA linear algebra libraries



François Courteille | Principal Solutions Architect, NVIDIA | fcourteille@nvidia.com

NVIDIA

ACKNOWLEDGEMENTS

Joe Eaton , NVIDIA

Ty McKercher , NVIDIA

Lung Sheng Chien , NVIDIA

Nikolay Markovskiy , NVIDIA

Stan Posey , NVIDIA

Steve Rennich , NVIDIA

Dave Miles , NVIDIA

Peng Wang, NVIDIA

Questions: fcourteille@nvidia.com

AGENDA

Prolegomena

NVIDIA Solutions for Accelerated Linear Algebra

Libraries performance on Pascal

Rapid software development for heterogeneous architecture

PROLEGOMENA

NVIDIA

Gaming

VR

AI & HPC

Self-Driving Cars

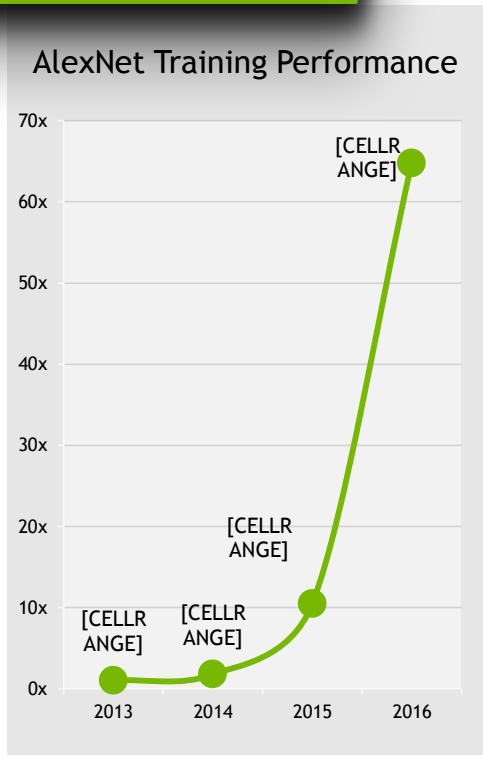
GPU Computing

ONE ARCHITECTURE BUILT FOR BOTH DATA SCIENCE & COMPUTATIONAL SCIENCE

Pascal & Volta

NVIDIA DGX-1

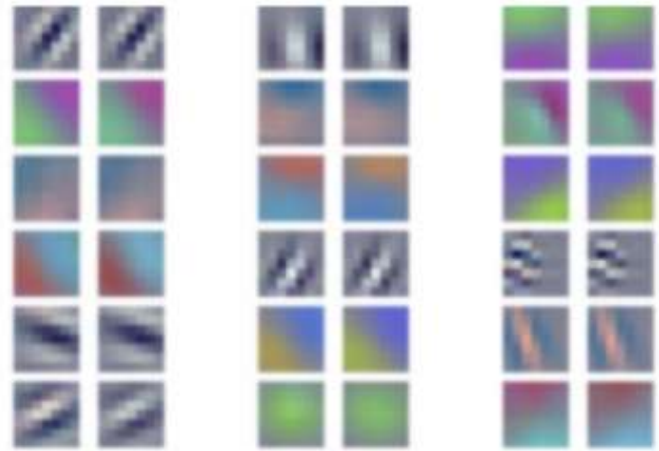
NVIDIA DGX SATURNV



65x in 3 Years

Low Rank Approximation

- **Low Rank approximation**
  - **Tensor decomposition based on singular value decomposition (SVD)**
  - **Filter Clustering with modified K-means**
  - **Fine Tuning**



- **Speed up by 1.6 – 2.7x on CPU/GPU for CONV1, CONV2 layers**
- **Reduce size by 5 - 13x for FC layer**
- **< 1% drop in accuracy**

[Denton et al., NIPS 2014]

Low Rank Approximation on Phone

- **Rank selection per Layer**
- **Tucker Decomposition (extension of SVD)**
- **Fine tuning**

Model	Top-5	Weights	FLOPs	S6		Titan X
AlexNet	80.03	61M	725M	117ms	245mJ	0.54ms
*AlexNet**	78.33	11M	272M	43ms	72mJ	0.30ms
(imp.)	(-1.70)	(×5.46)	(×2.67)	(×2.72)	(×3.41)	(×1.81)
VGG-S	84.60	103M	2640M	357ms	825mJ	1.86ms
*VGG-S**	84.05	14M	549M	97ms	193mJ	0.92ms
(imp.)	(-0.55)	(×7.40)	(×4.80)	(×3.68)	(×4.26)	(×2.01)
GoogLeNet	88.90	6.9M	1566M	273ms	473mJ	1.83ms
*GoogLeNet**	88.66	4.7M	760M	192ms	296mJ	1.48ms
(imp.)	(-0.24)	(×1.28)	(×2.06)	(×1.42)	(×1.60)	(×1.23)
VGG-16	89.90	138M	15484M	1926ms	4757mJ	10.67ms
*VGG-16**	89.40	127M	3139M	576ms	1346mJ	4.58ms
(imp.)	(-0.50)	(×1.09)	(×4.93)	(×3.34)	(×3.53)	(×2.33)

[Kim et al., ICLR 2016]

TESLA V100

THE MOST ADVANCED DATA CENTER GPU EVER BUILT

5,120 CUDA cores
640 NEW Tensor cores
7.5 FP64 TFLOPS | 15 FP32 TFLOPS
120 Tensor TFLOPS
20MB SM RF | 16MB Cache | 16GB HBM2 @ 900 GB/s
300 GB/s NVLink

NVLINK TO CPU

IBM Power Systems Server S822LC (codename “Minsky”)

2x IBM Power8+ CPUs and 4x P100 GPUs

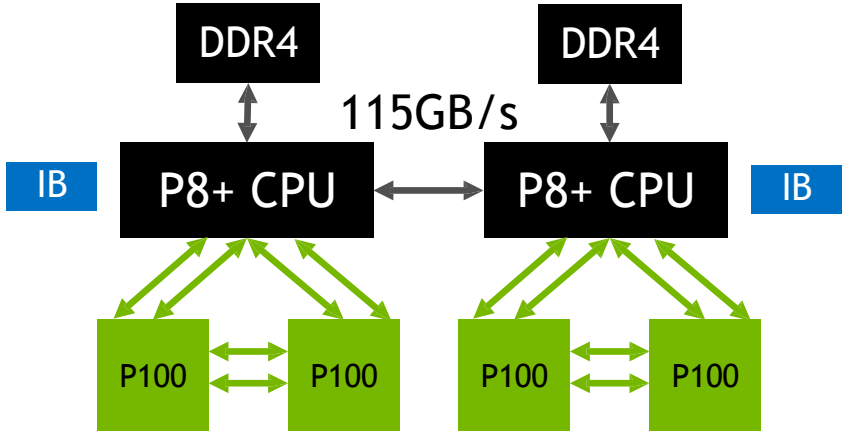
80 GB/s per GPU bidirectional for peer traffic

80 GB/s per GPU bidirectional to CPU

115 GB/s CPU Memory Bandwidth

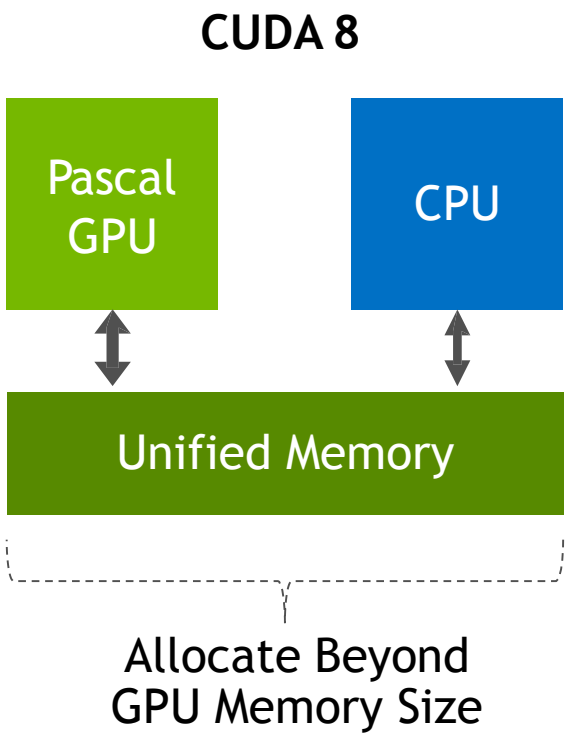
Direct Load/store access to CPU Memory

High Speed Copy Engines for bulk data movement



UNIFIED MEMORY ON PASCAL

Large datasets, Simple programming, High performance



Enable Large Data Models	Oversubscribe GPU memory Allocate up to system memory size
Higher Application Performance	Demand paging &Page Migration HW User APIs for prefetching & migration hints
Simpler Data Access	CPU/GPU Data coherence Unified memory atomic operations

INTRODUCING THE DGX FAMILY

AI WORKSTATION

DGX Station

The Personal
AI Supercomputer

AI DATA CENTER

DGX-1

with

Tesla P100

The World's First
AI Supercomputer
in a Box

with

Tesla V100

The Essential
Instrument for AI
Research

CLOUD-SCALE AI

NVIDIA GPU Cloud

Cloud service with the highest
deep learning efficiency

3 WAYS TO ACCELERATE APPLICATIONS

Easy to use
Most Performance

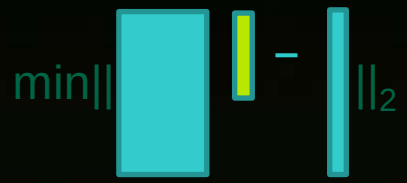
Easy to use
Portable code

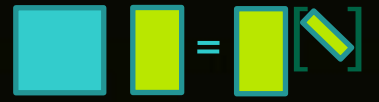
Most Performance
Most Flexibility

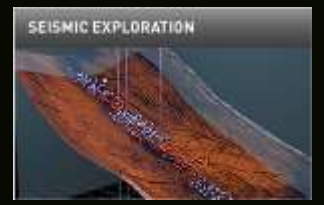
NVIDIA SOLUTIONS FOR ACCELERATED LINEAR ALGEBRA

Sparse Problems

- Linear Systems
  $A x = f$
- Linear Least Squares
  $\min \|B y - f\|_2$
- Eigenvalue Problems
  $A V = V D$
- Singular Value Decomposition
  $A = U D V^T$



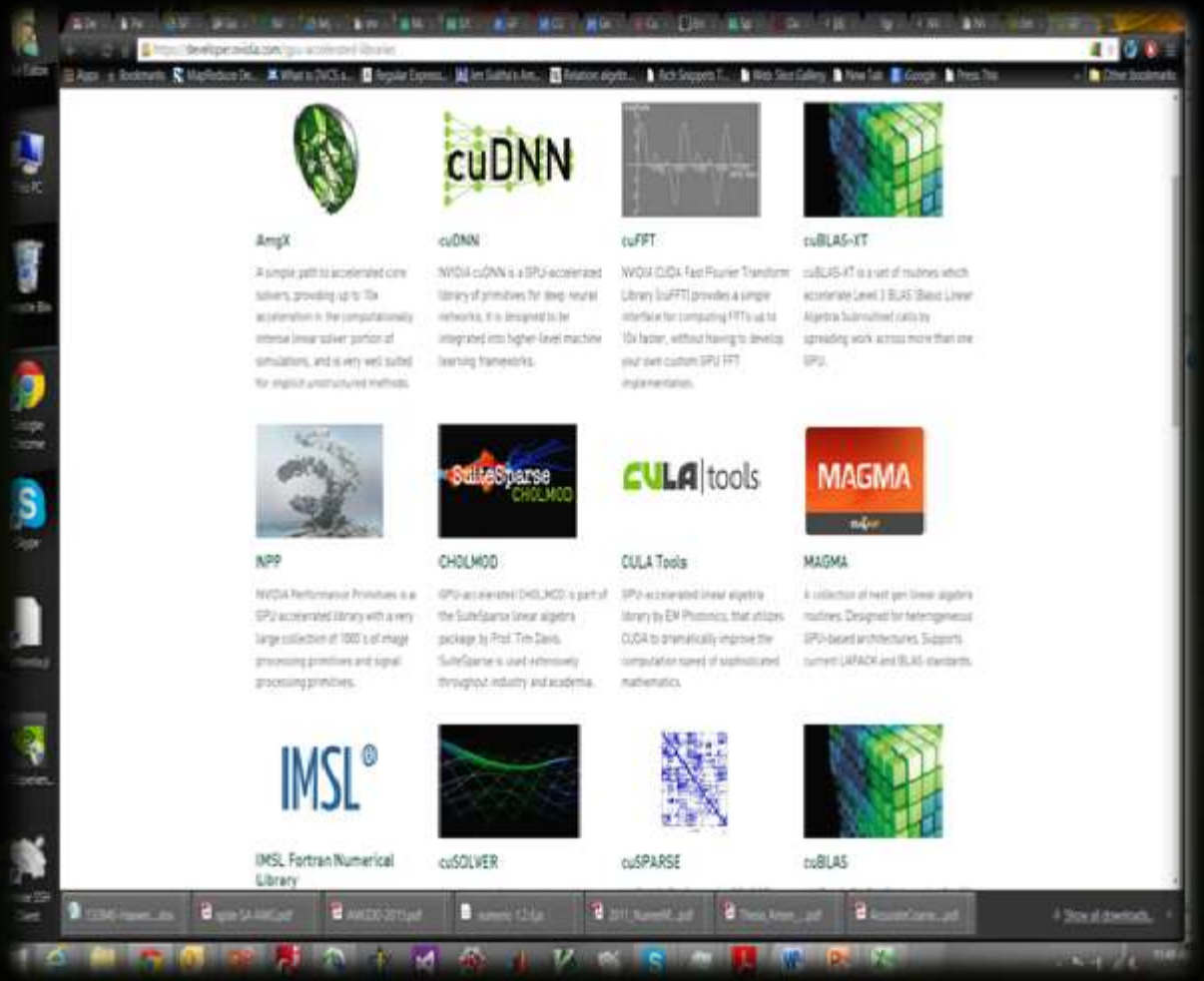




COMPUTATIONAL
CHEMISTRY
BIOLOGY
Machine Learning

GPU-Accelerated Libraries

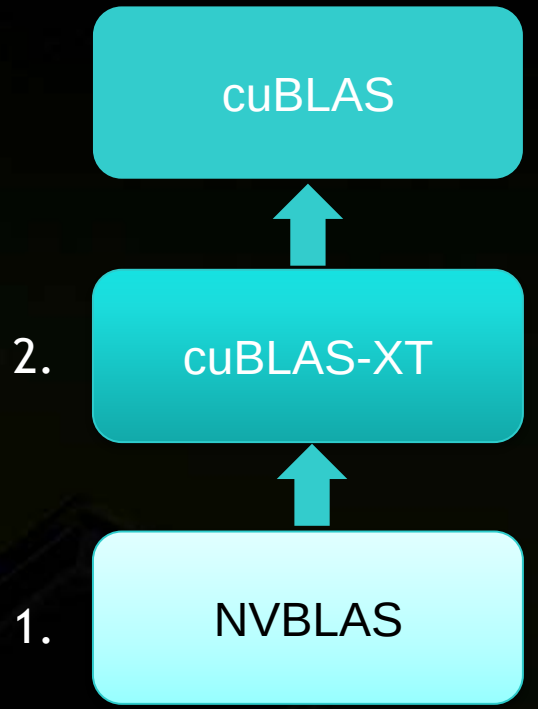
- CUDA Toolkit Libraries
  - CUSPARSE, CUSOLVER, CUBLAS
- NVIDIA Proprietary libraries
  - AmgX
  - NVGRAPH
- Third Party libraries
  - Trilinos, PETSc
  - ArrayFire, CHOLMOD
  - MAGMA



https://developer.nvidia.com/gpu-accelerated-libraries

2. cuBLAS and nvBLAS

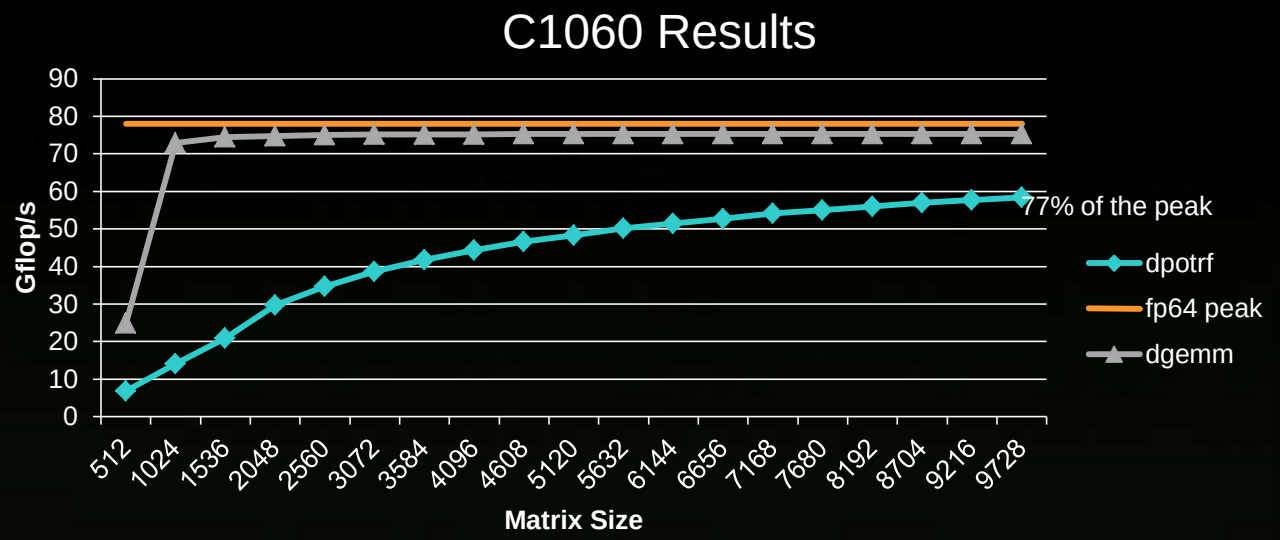
BLAS on GPUS



Cholesky Factorization on GPU

$$\left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

$$\begin{aligned} A_{11} &:= A_{11} - A_{10}A_{10}^{T} && \text{(SYRK)} \\ A_{11} &:= \text{CHOL\_UNB}(A_{11}) \\ A_{21} &:= A_{21} - A_{20}A_{10}^{T} && \text{(GEMM)} \\ A_{21} &:= A_{21}\,\text{TRIL}(A_{11})^{-T} && \text{(TRSM)} \end{aligned}$$




```
void dpotrf(char uplo, int n, double *A, int lda, double *work, int *info)
{
    for (int j = 0; j < n; j += nb) {
        cublasDsyrk('L', 'N', nb, j, -1.0, &A[j], lda, 1.0, &A[j+j*lda], lda);
        cublasGetMatrix(nb, nb, sizeof(double), &A[j+j*lda], lda, work, nb);
        cublasDgemm('N', 'T', n-j-nb, nb, j, -1.0, &A[j+nb], lda, &A[j], lda, 1.0, &A[j+nb+j*lda], lda);
        dpotf2_cpu(nb, work, nb);
        cublasSetMatrix(nb, nb, sizeof(double), work, nb, &A[j+j*lda], lda);
        cublasDtrsm('R', 'L', 'T', 'N', n-j-nb, nb, 1.0, &A[j+j*lda], lda, &A[j+nb+j*lda], lda);
    }
}
```

What is NVBLAS?

- Drop-in replacement of BLAS
  - Built on top of cuBLAS-XT
  - BLAS Level 3
- Zero coding effort
  - R, Octave, Scilab , etc
- Limited only by amount of host memory
  - Large data sets on single GPU
  - PCI transfers can become the bottleneck in complicated scenarios

NVBLAS supported api

Routine	Types	Operation
gemm	S,D,C,Z	multiplication of 2 matrices
syrk	S,D,C,Z	symmetric rank-k update
herk	C,Z	hermitian rank-k update
syr2k	S,D,C,Z	symmetric rank-2k update
her2k	C,Z	hermitian rank-2k update
trsm	S,D,C,Z	triangular solve with multiple right-hand sides
symm	S,D,C,Z	symmetric matrix-matrix multiplication
hemm	C,Z	hermitian matrix-matrix multiplication

cuBLAS-XT

- Level 3 BLAS

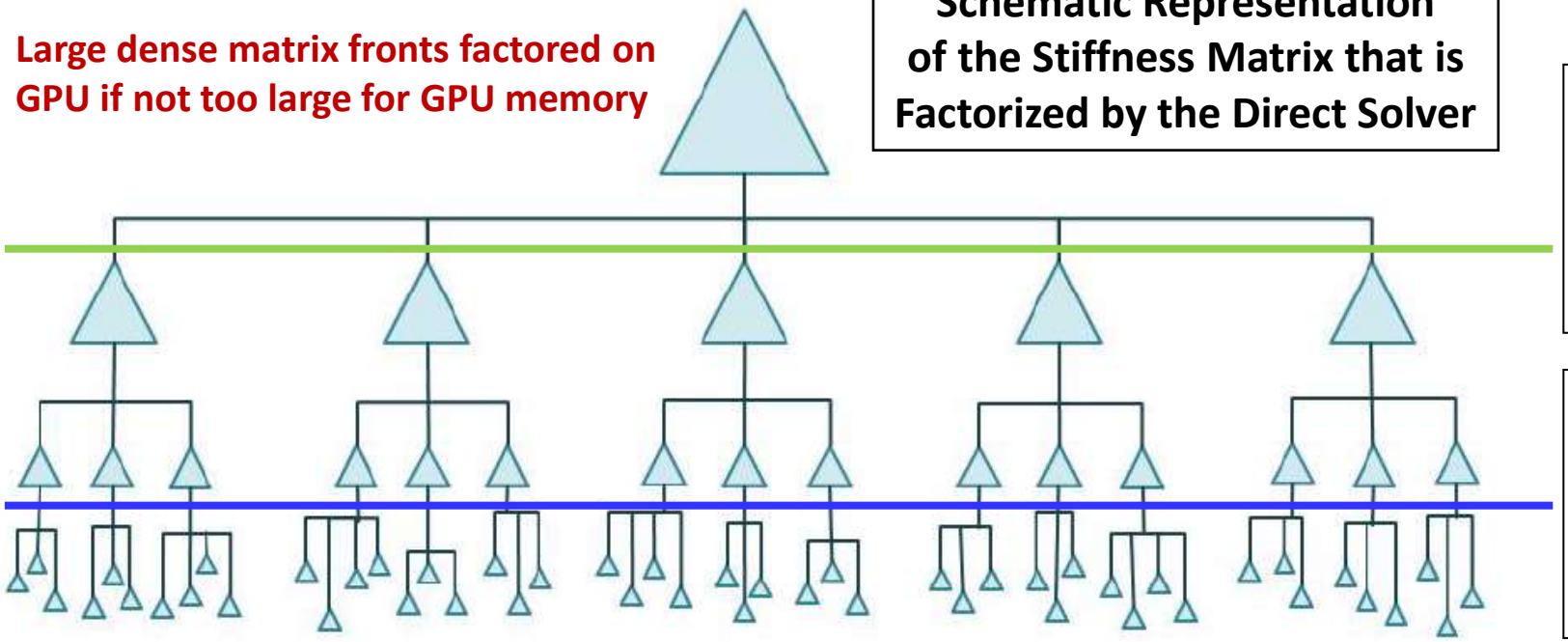
Routine	Types	Operation
gemm	S,D,C,Z	multiplication of 2 matrices
syrk	S,D,C,Z	symmetric rank-k update
herk	C,Z	hermitian rank-k update
syr2k	S,D,C,Z	symmetric rank-2k update
her2k	C,Z	hermitian rank-2k update
trsm	S,D,C,Z	triangular solve with multiple right-hand sides
symm	S,D,C,Z	symmetric matrix-matrix multiplication
hemm	C,Z	hermitian matrix-matrix multiplication

Basics of Direct Solvers for Mostly Implicit CSM

Typical implicit CSM deployment of multi-frontal direct sparse solvers

Large dense matrix fronts factored on GPU if not too large for GPU memory

Schematic Representation of the Stiffness Matrix that is Factorized by the Direct Solver

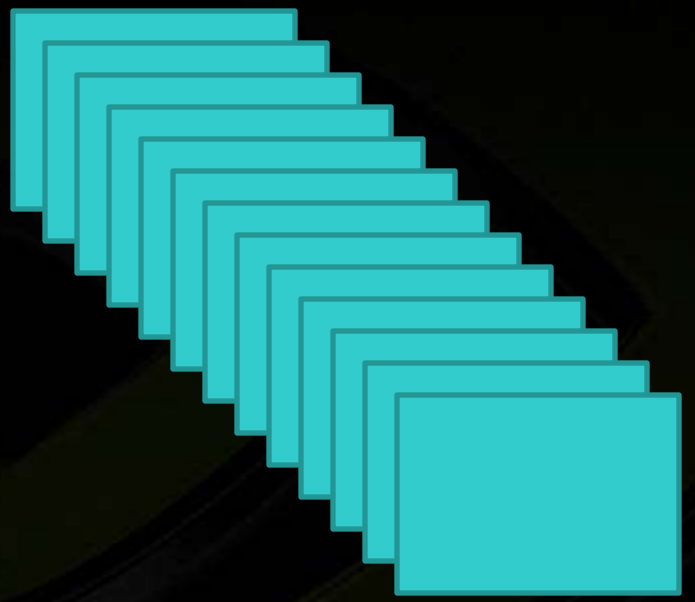


Upper threshold: Fronts too large for single GPU memory need multiple GPUs

Lower threshold: Fronts too small to overcome PCIe data transfer costs stay on CPU cores

Small dense matrix fronts factored in parallel on CPU – more cores means higher performance

Batched cuBLAS routines



Process many similar matrices at once

- Factorize (LU)
- Multiply (GEMM)
- Good for FEM codes, multi-scale methods
- Chemistry kinetics, up to 72 species, combined with CVODE

3. cuSolver

cuSOLVER

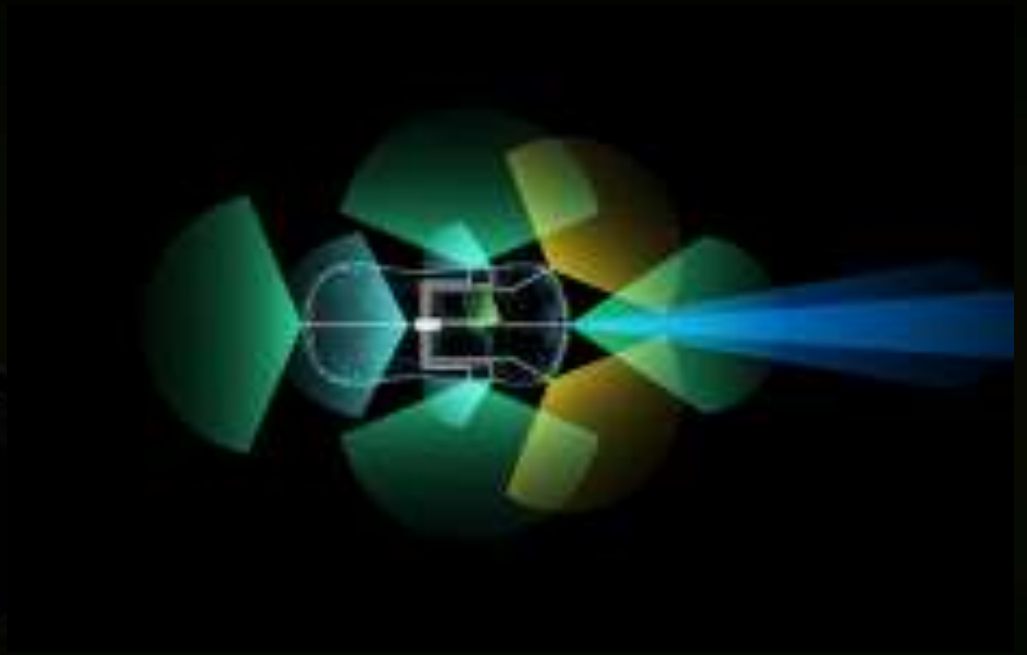
Routines for solving sparse or dense linear systems and Eigen problems.

Divided into 3 APIs for 3 different use cases:

- **cuSolverDN – subset of LAPACK for small dense systems**
  - LU, Cholesky, QR, LDLT, SVD
- **cuSolverSP– sparse direct solvers and Eigensolvers,**
  - sparse QR, sparse batch QR, least squares
- **cuSolverRF– fast refactorization solver for sparse matrices**
  - Multiscale methods, Chemistry

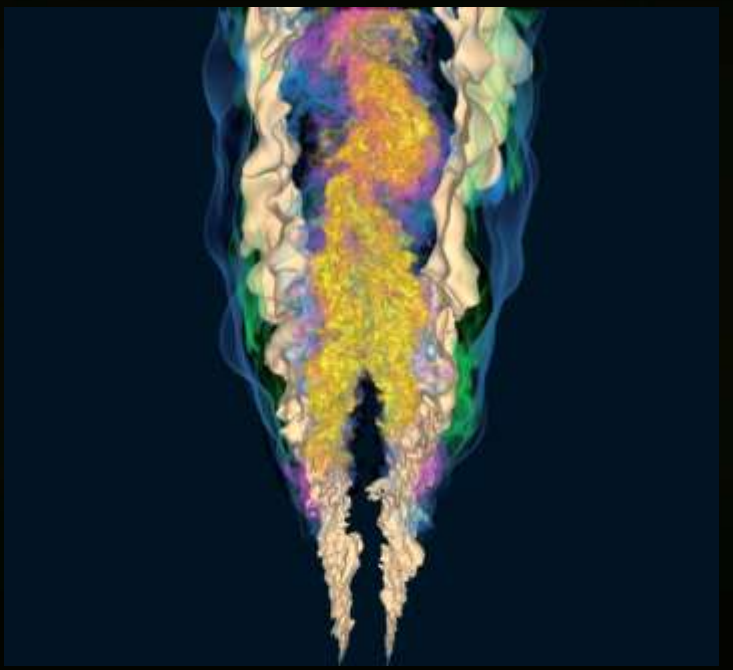
cuSolverDN API

- Subset of LAPACK (direct solvers for dense matrices) – only few most popular methods
  - Cholesky / LU
  - QR,SVD
  - Bunch-Kaufman LDLT
  - Batched QR
- Useful for:
  - Computer vision
  - Optimization
  - CFD by FEM



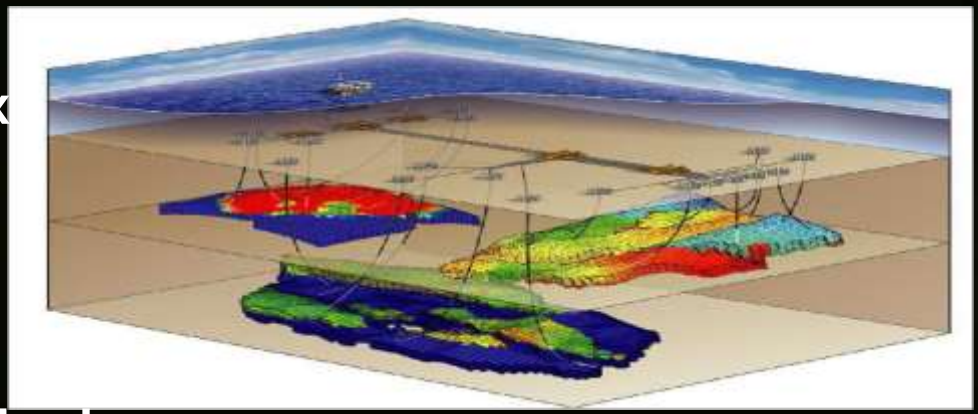
cuSolverRF API

- LU-based sparse direct solver, requires factorization to already be computed (e.g. using KLU)
- + batched version – many small matrices to be solved in parallel
- Useful for:
  - SPICE
  - Combustion simulation
  - Chemically reacting flow calculation
  - Other types of ODEs, mechanics
  - Multiscale methods, FEM



cuSolverSP API

- Sparse direct solver based on QR factorization
  - Linear solver A*x = b (QR or Cholesky-based)
  - Least-squares solver min|A*x – b|
  - Eigenvalue solver based on shift-inverse
  - A*x = \lambda*x
  - Find number of Eigenvalues in a box
- Useful for:
  - Well models in Oil & Gas
  - Non-linear solvers via Newton's method
  - Anywhere a sparse-direct solver is required



4. cuSPARSE

cuSPARSE: (Dense matrix) X (sparse vector)

- Speeds up Natural Language Processing
  - cusparse<T>gemvi()
    - y = α ∗ op(A)∗x + β∗y
    - A = dense matrix
    - x = sparse vector
    - y = dense vector

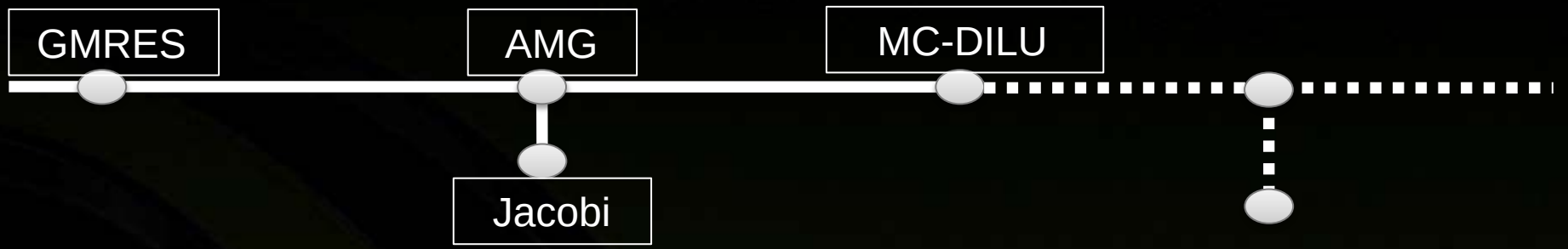
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \alpha \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} \end{bmatrix} \begin{bmatrix} - \\ 2 \\ - \\ - \\ 1 \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Sparse vector could be frequencies of words in a text sample

5. AmgX Library

Nested Solvers

- Solver hierarchy, e.g.



- Example solvers
  - **GMRES:** local and global operations, no setup
  - **AMG:** setup - graph coarsening and matrix-matrix products
    solve - smoothing and matrix-vector products
  - **Jacobi:** simple local (neighbor) operations smoothing, no setup
  - **MC-DILU:** setup - graph coloring and factorization
    solve – local (sub)matrix-vector multiplication

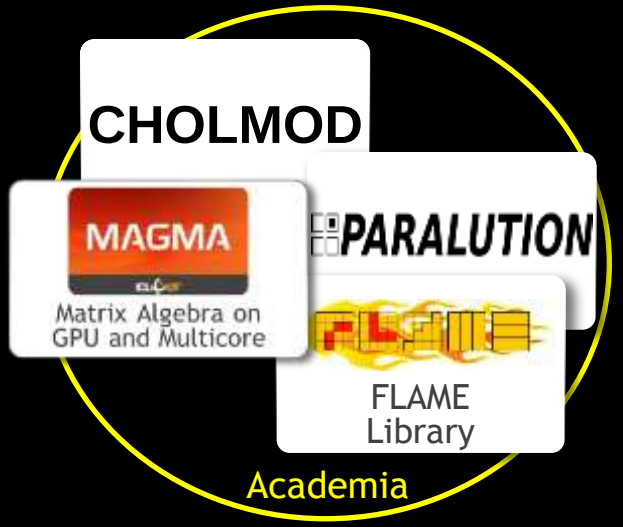
AmgX Key Features

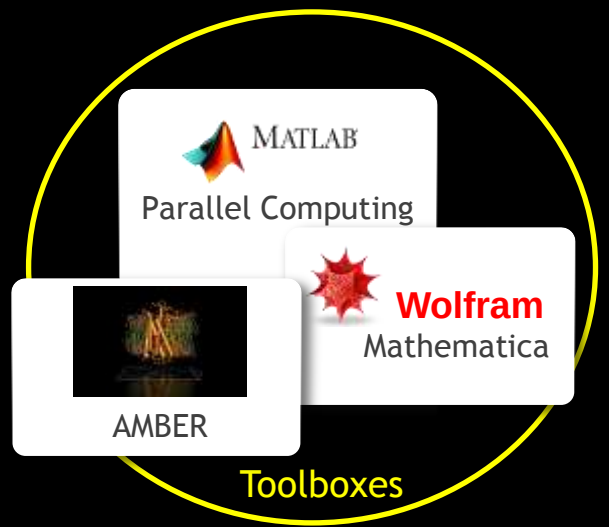
- **Open Source soon (legal issues under final review)**
- **Multi-GPU support**
  - workstation and clusters up to hundreds of nodes
  - Sweet spot seems to be 8 GPUs/Node
- **More solvers, smoothers and preconditioners**
  - Krylov methods, basic iterative solvers, AMG
- **Eigenvalue Solvers**
  - Subspace Iteration, Restarted Arnoldi (ARPACK) and Jacobi-Davidson

ANSYS® Fluent 15.0

4. Third Party Libraries

Third Party Libraries





Third Party Libraries

- Trilinos (Sandia National Labs)
  - Parallel Primitives
  - Discretizations, Linear System & Eigenvalue Solvers
  - Supports NVIDIA GPUs
    (parallel primitives)

http://trilinos.sandia.gov/

- PETSc (Argonne National Labs)
  - Parallel Primitives
  - Iterative Methods, Nonlinear Solvers, ...
  - Supports NVIDIA GPUs
    (iterative methods + some preconditioners)

PETSc

http://www.mcs.anl.gov/petsc/index.html

Third Party Libraries

- ArrayFire (AccelerEyes)
  - Factorizations, Eigenvalue solvers, ...
  - Supports NVIDIA GPUs
    (parallel primitives)

http://arrayfire.com/

- CHOLMOD (Tim Davis)
  - Sparse Direct Solver
  - Cholesky factorization (s.p.d. $A=LL^T$)
  - Supports NVIDIA GPUs
    (offloads dense linear algebra calls)

CHOLMOD

http://www.cise.ufl.edu/research/sparse/cholmod/

4. NVGRAPH

INTRODUCING NVGRAPH

Accelerate graph analytics applications

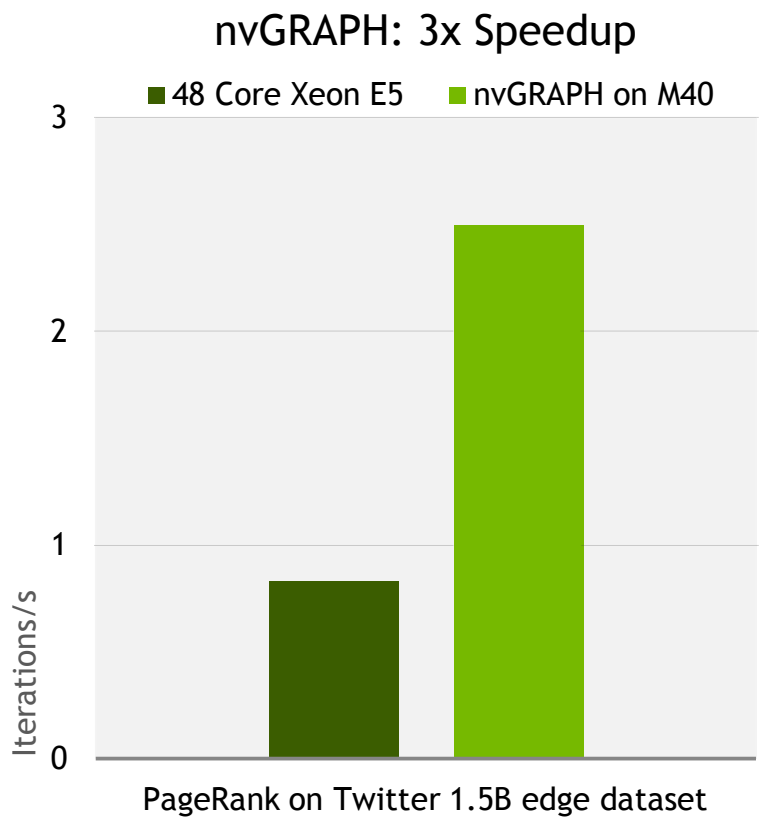
Deliver results up to 3x faster than CPU-only

Solve graphs with up to 2.5 Billion edges on 1x M40

Accelerates a wide range of graph analytics apps

PAGERANK	SINGLE SOURCE SHORTEST PATH	SINGLE SOURCE WIDEST PATH
Search	Robotic path planning	IP routing
Recommendation engines	Power network planning	Chip design / EDA
Social Ad placement	Logistics & supply chain planning	Traffic sensitive routing

developer.nvidia.com/nvgraph



CPU System: 4U server w/ 4x12-core Xeon E5-2697 CPU, 30M Cache, 2.70 GHz, 512 GB RAM

LIBRARIES PERFORMANCE ON PASCAL

GPU ACCELERATED LIBRARIES:
GRAPH ANALYTICS

nvGRAPH

GPU Accelerated Graph Analytics

Parallel Library for Interactive and High Throughput Graph Analytics

Solve graphs with up to 2.5 Billion edges on a single GPU (Tesla M40)

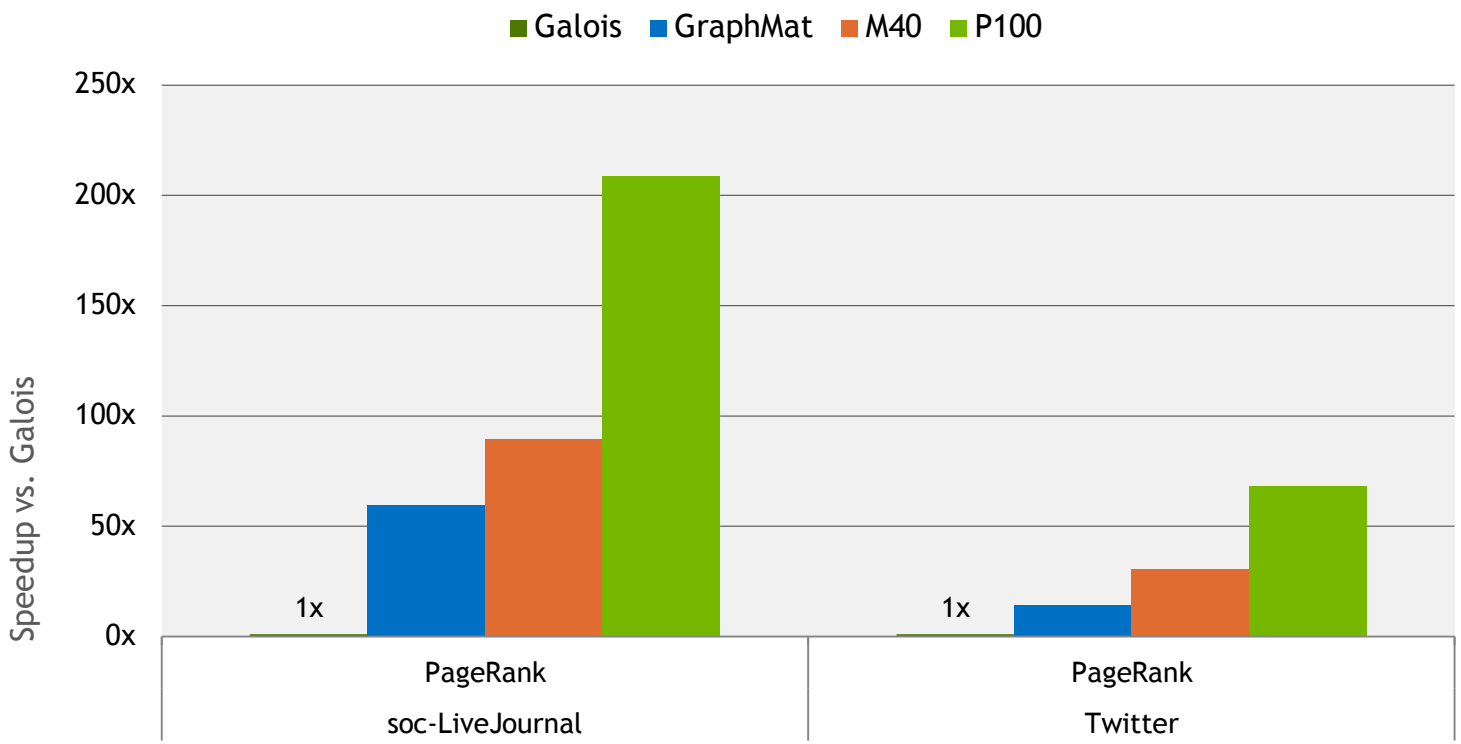
Includes — PageRank, Single Source Shortest Path and Single Source Widest Path algorithms

Semi-ring SPMV operations provides building blocks for graph traversal algorithms

PageRank	Single Source Shortest Path	Single Source Widest Path
Search	Robotic Path Planning	IP Routing
Recommendation Engines	Power Network Planning	Chip Design / EDA
Social Ad Placement	Logistics & Supply Chain Planning	Traffic sensitive routing

https://developer.nvidia.com/nvgraph

> 200X SPEEDUP ON PAGERANK VS GALOIS

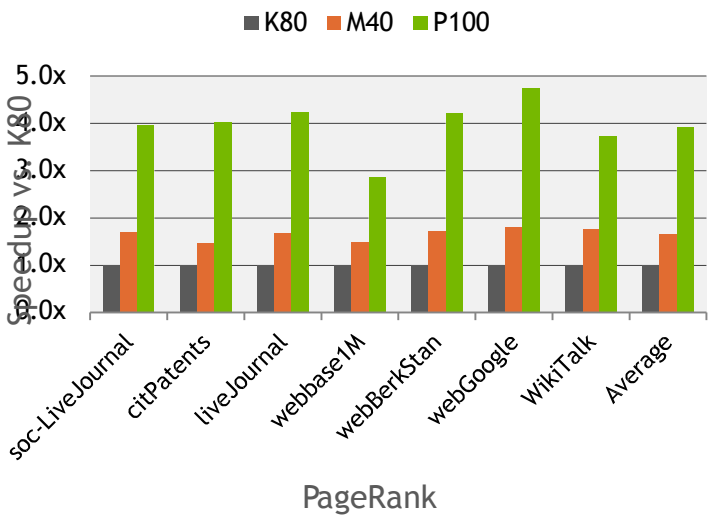


- nvGRAPH on M40 (ECC ON, r352), P100 (r361), Base clocks, input and output data on device
- GraphMat, Galois (v2.3) on Intel Xeon Broadwell dual-socket 22-core/socket E5-2699 v4 @ 2.22GHz, 3.6GHz Turbo
- Comparing Average Time per Iteration (ms) for PageRank
- Host System: Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

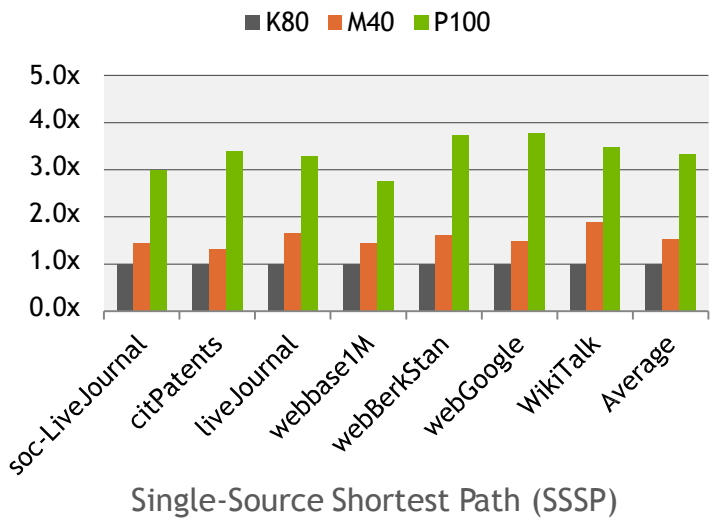
Performance may vary based on OS and software versions, and motherboard configuration

> 4X SPEEDUPS WITH P100

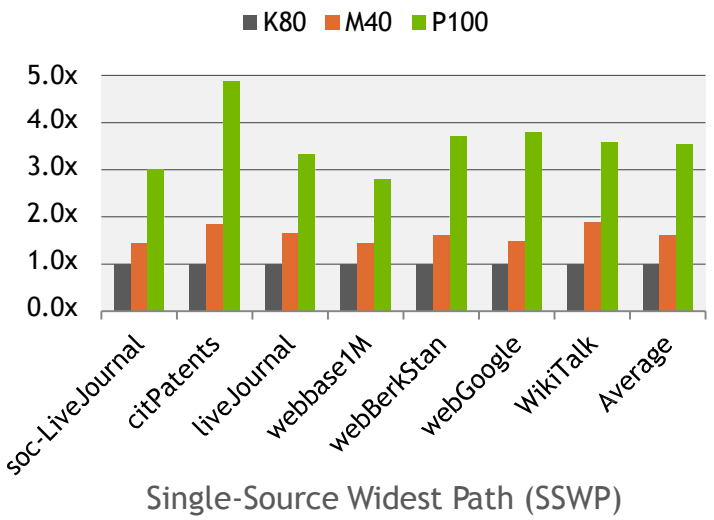
Using Different Algorithms in nvGRAPH



PageRank



Single-Source Shortest Path (SSSP)



Single-Source Widest Path (SSWP)

- nvGRAPH on K80, M40, P100, ECC ON, Base clocks, input and output data on device
- GraphMat, Galois (v2.3) on Intel Xeon Broadwell dual-socket 44-core E5-2699 v4 @ 2.22GHz, 3.6GHz Turbo
- Comparing Average Time per Iteration (ms) for PageRank and Total Solver Time (ms) for SSSP and SSWP
- Host System: Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

Performance may vary based on OS and software versions, and motherboard configuration

GPU ACCELERATED LIBRARIES:
FAST FOURIER TRANSFORMS

cuFFT

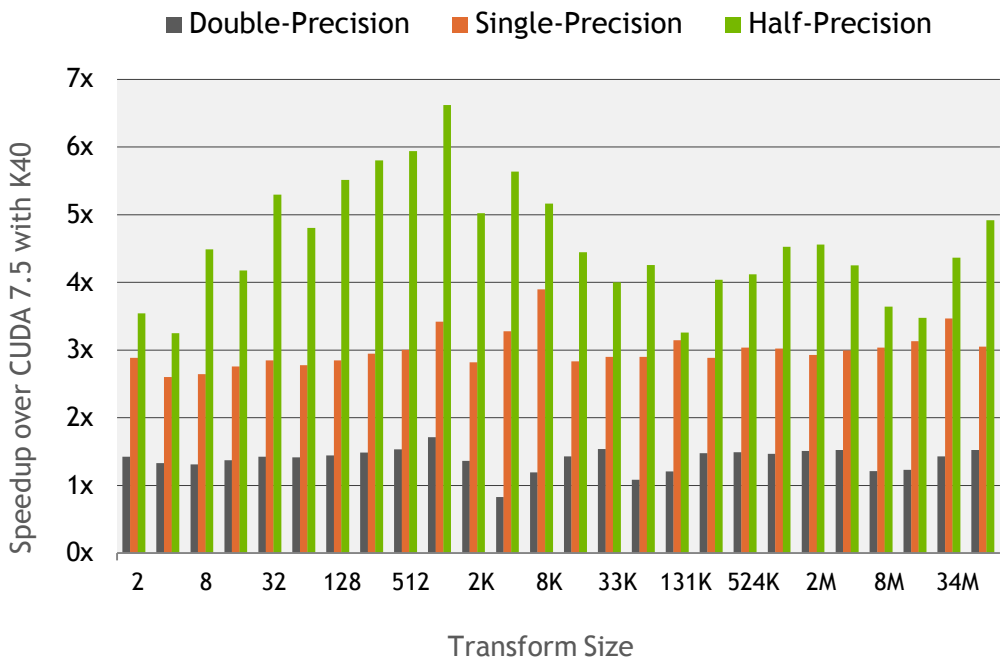
Complete Fast Fourier Transforms Library

Complete Multi-Dimensional FFT Library

- Simple “drop-in” replacement of a CPU FFTW library
- Real and complex, single- and double-precision data types
- Includes 1D, 2D and 3D batched transforms
- Support for half-precision (FP16) data types
- Supports flexible input and output data layouts
- XT interface now supports up to 8 GPUs

https://developer.nvidia.com/cufft

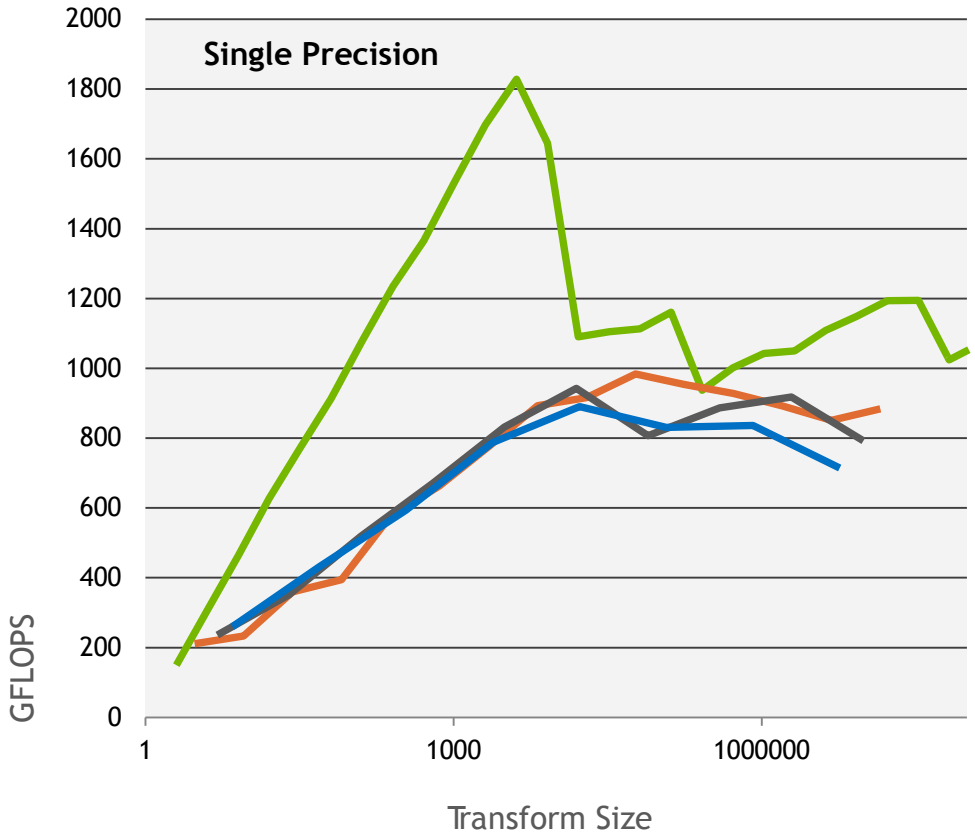
> 6x Speedup with Half-Precision on P100

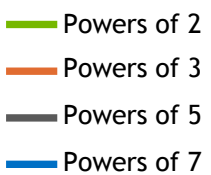


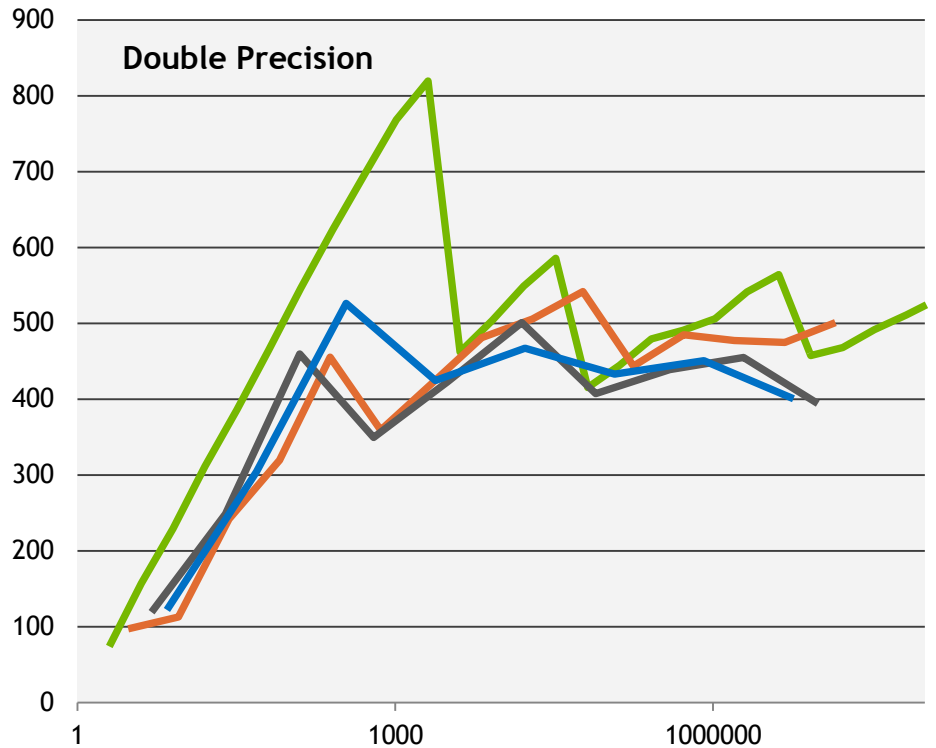
- Speedup of P100 with CUDA 8 vs. K40m with CUDA 7.5
- cuFFT 7.5 on K40m, Base clocks, ECC on (r352)
- cuFFT 8.0 on P100, Base clocks, ECC on (r361)
- 1D Complex, Batched transforms on 28-33M elements
- Input and output data on device
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

cuFFT: > 1800 GFLOPS SINGLE PRECISION

1D Complex, Batched FFTs





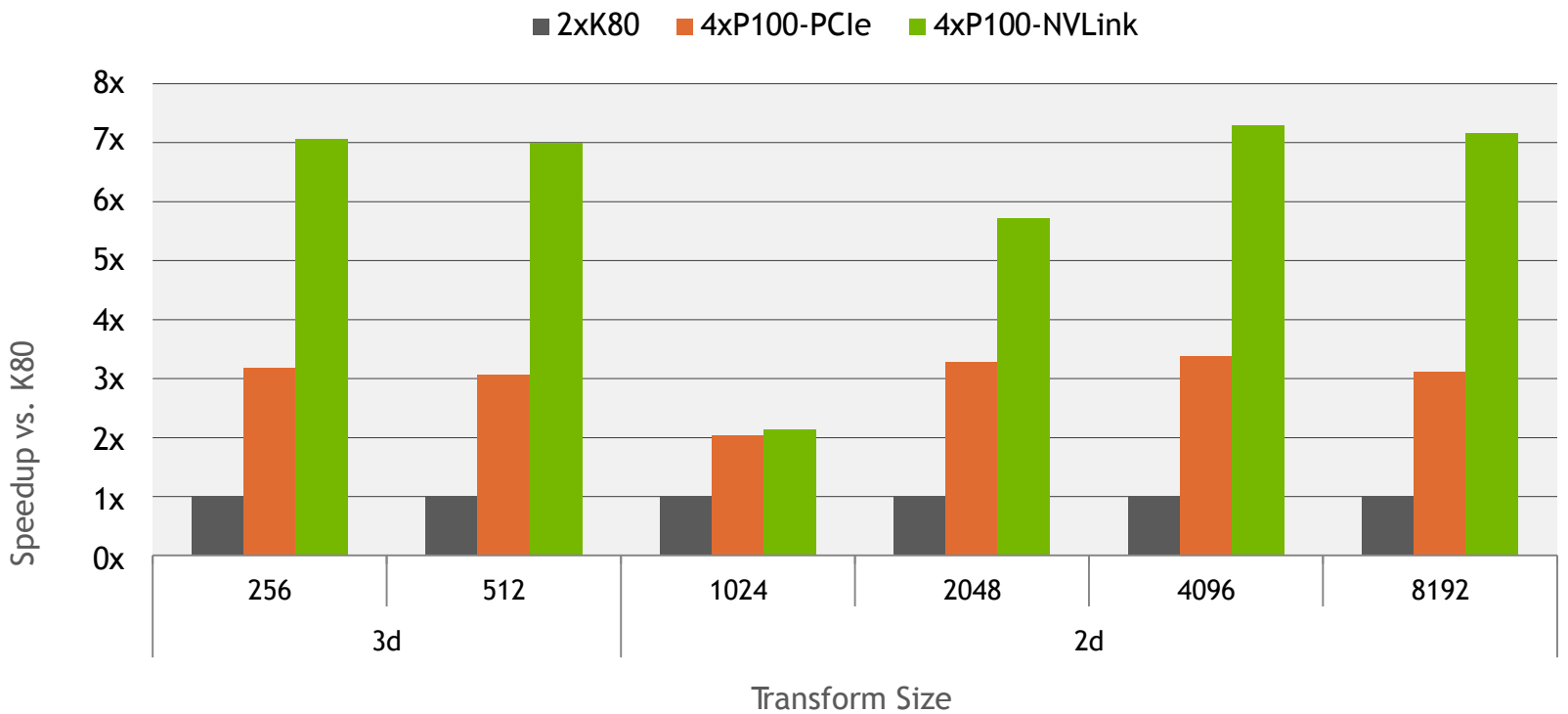


- cuFFT 8 on P100, Base clocks (r361)
- Batched transforms on 28-33M elements
- Input and output data on device
- Excludes time to create cuFFT “plans”
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3@ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

Performance may vary based on OS and software versions, and motherboard configuration

cuFFT-XT: > 7X IMPROVEMENTS WITH NVLINK

2D and 3D Complex FFTs



- cuFFT 7.5 on 2xK80m, ECC ON, Base clocks (r352)
- cuFFT 8 on 4xP100 with PCIe and NVLink (DGX-1), Base clocks (r361)
- Input and output data on device
- Excludes time to create cuFFT "plans"
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3@ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

Performance may vary based on OS and software versions, and motherboard configuration

GPU ACCELERATED LIBRARIES:
SPARSE and DENSE LINEAR ALGEBRA

cuBLAS

Dense Linear Algebra on GPUs

Complete BLAS Library Plus Extensions

Supports all 152 standard routines for single, double, complex, and double complex

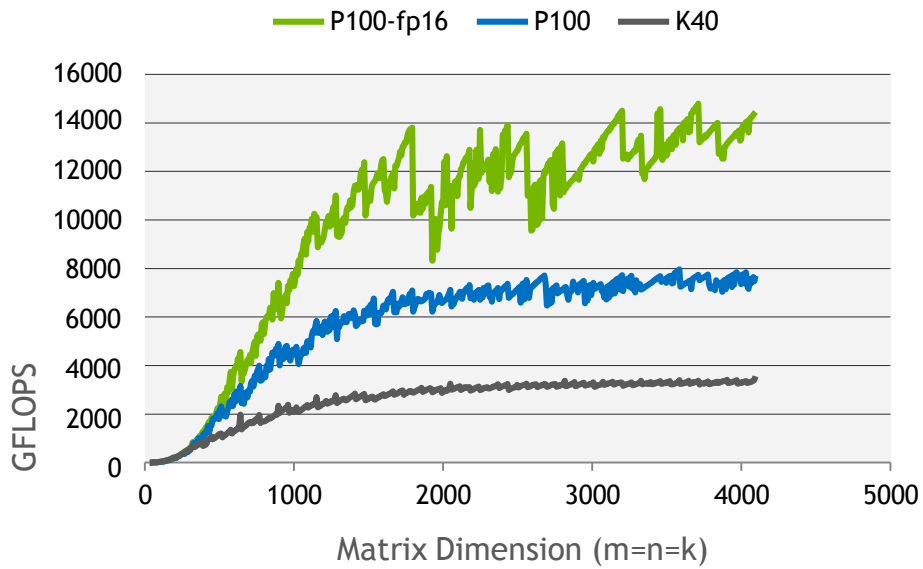
Supports half-precision (FP16) and integer (INT8) matrix multiplication operations

Batched routines for higher performance on small problem sizes

Host and device-callable interface

XT interface supports distributed computations across multiple GPUs

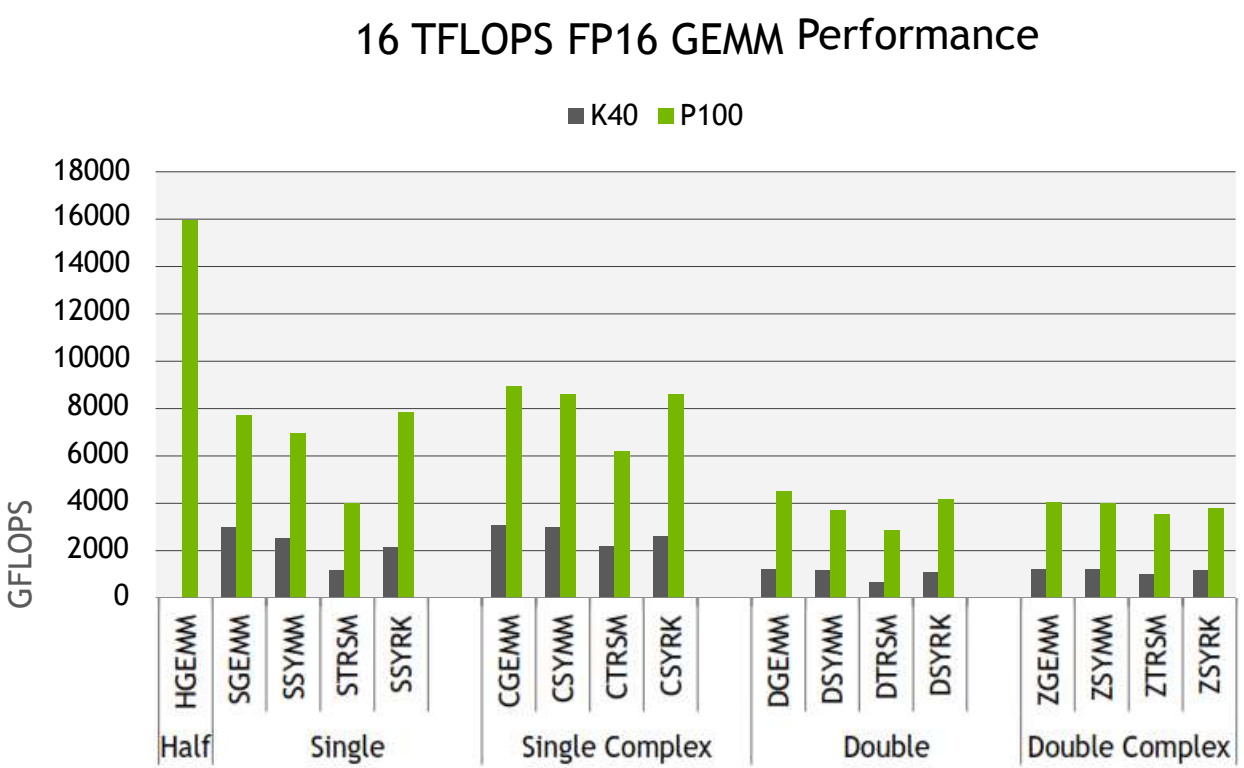
> 4x Faster GEMM Performance with FP16 on P100

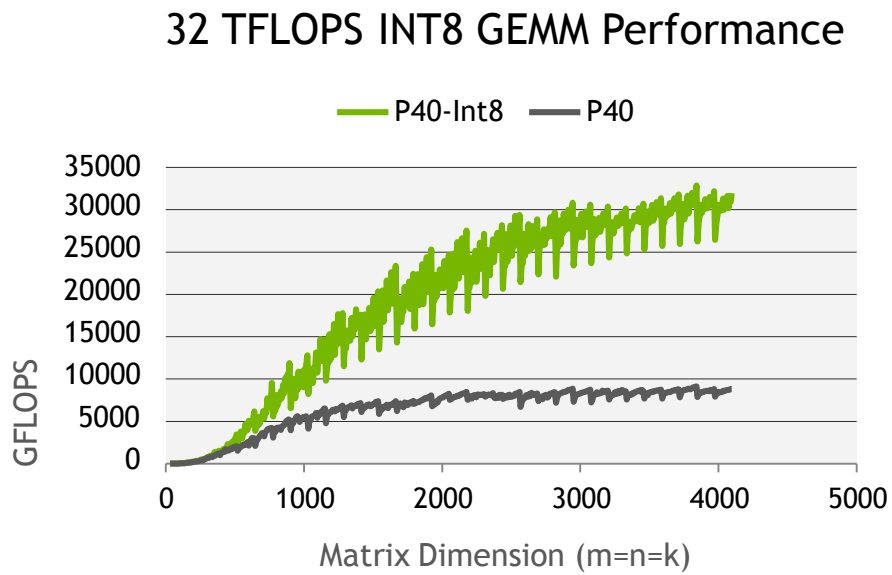


- Comparing GEMM performance on K40m (FP32) and P100 (FP32 and FP16)
- cuBLAS 8 on P100, Base clocks (r361)
- cuBLAS 8 on P40, Base clocks (r367)
- cuBLAS 7.5 on K40m, Base clocks, ECC ON (r352)
- Input and output data on device
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3@ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory
- m=n=k=4096

https://developer.nvidia.com/cublas

cuBLAS: > 8 TFLOPS SINGLE PRECISION

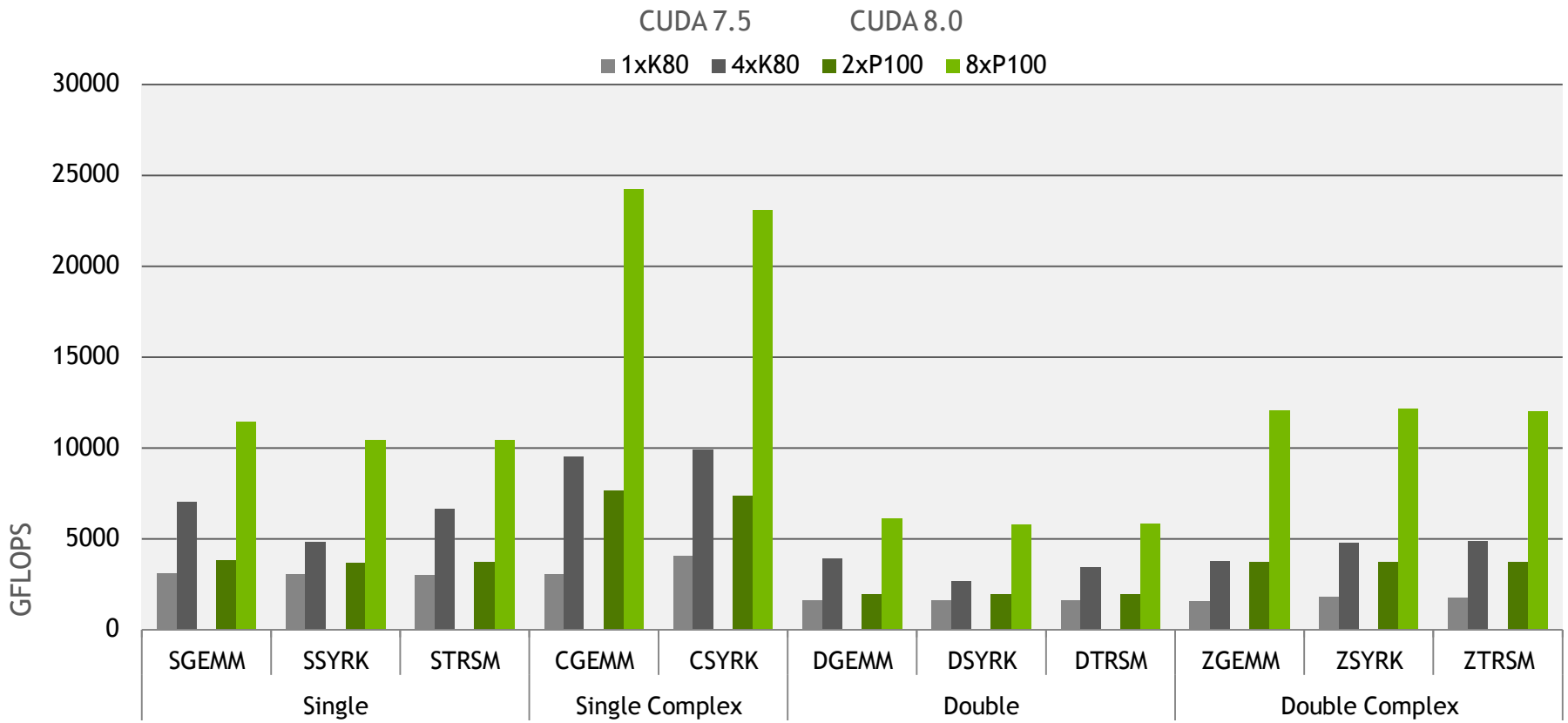




- cuBLAS 8 on P100 (r361) and P40 (r367) ; Base clocks
- cuBLAS 7.5 on K40m ; Base clocks, ECC ON (r352)
- Input and output data on device
- m=n=k=4096, transpose=no, side=right, fill=lower
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3@ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

Performance may vary based on OS and software versions, and motherboard configuration

cuBLAS-XT: > 24 TFLOPS ON A SINGLE NODE



- cuBLAS 8 on P100 (r361); Base clocks
- cuBLAS 7.5 on K80 ; Base clocks, ECC ON (r352)
- 1xK80 indicates 2-GPUs (or one K80 board)
- Input and output data on device
- m=n=k=4096, transpose=no, side=right, fill=lower
- Host system: Intel Xeon Haswell dual-socket 22-core E5-2699 v4@ 2.2GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 256GB System Memory

Performance may vary based on OS and software versions, and motherboard configuration

cuSPARSE

Sparse Linear Algebra on GPUs

Optimized Sparse Matrix Library

Optimized sparse linear algebra BLAS routines for matrix-vector, matrix-matrix, triangular solve

Support for variety of formats (CSR, COO, block variants)

Incomplete-LU and Cholesky preconditioners

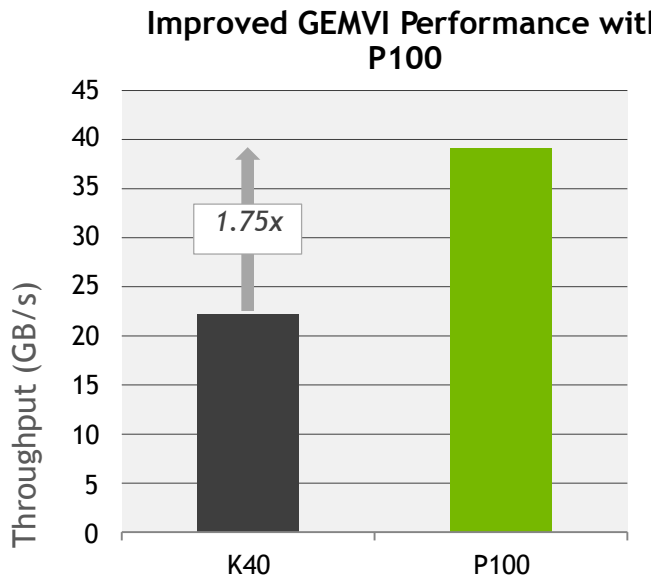
Support for half-precision (fp16) sparse matrix-vector operations

https://developer.nvidia.com/cusparse

GEMVI – Dense Matrix X Sparse Vector

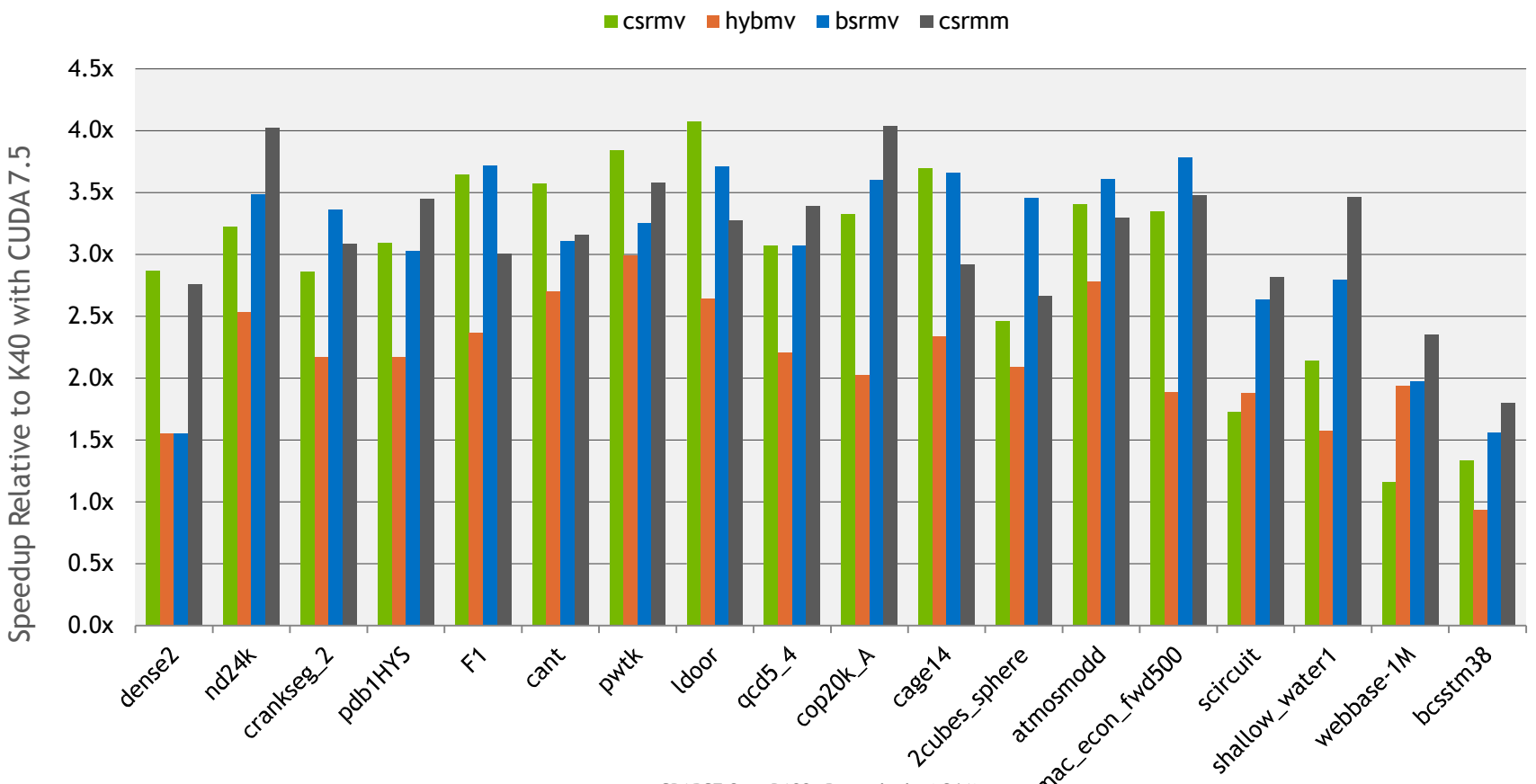
$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \alpha \begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} \\ A_{21} & A_{22} & A_{23} & A_{24} \\ A_{31} & A_{32} & A_{33} & A_{34} \\ A_{41} & A_{42} & A_{43} & A_{44} \end{bmatrix} \begin{bmatrix} - \\ 2.0 \\ - \\ 4.0 \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Used in language modeling and auto-encoders for recommender systems



- cuSPARSE 8 on P100, Base clocks (r361)
- cuSPARSE 7.5 on K40m, Base clocks, ECC ON (r352)
- Input and output data on device
- Dense matrices with 1e6 columns and 1e3 rows; Sparse vectors with less than 100 non-zeros out of 1e6 locations
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

cuSPARSE: > 4X FASTER WITH P100



- cuSPARSE 8 on P100, Base clocks (r361)
- cuSPARSE 7.5 on K40m, Base clocks, ECC ON (r352)
- Input and output data on device; Average speedup across different data types
- Matrices obtained from: http://www.cise.ufl.edu/research/sparse/matrices/
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

Performance may vary based on OS and software versions, and motherboard configuration

cuSOLVER

Linear Solver Library

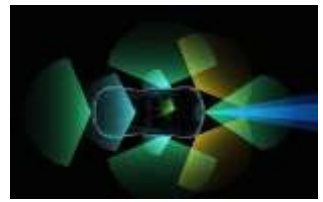
Library for Dense and Sparse Direct Solvers

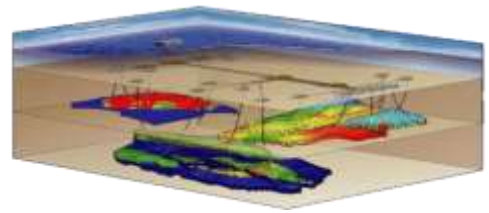
Supports Dense Cholesky, LU, (batched) QR, SVD and Eigenvalue solvers (new in CUDA 8)

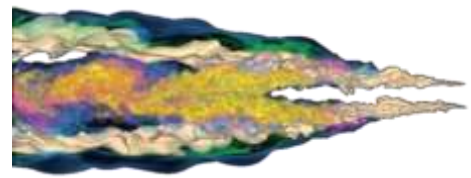
Sparse direct solvers & Eigensolvers

Includes a sparse refactorization solver for solving sequences of matrices with a shared sparsity pattern

Used in a variety of applications such as circuit simulation and computational fluid dynamics





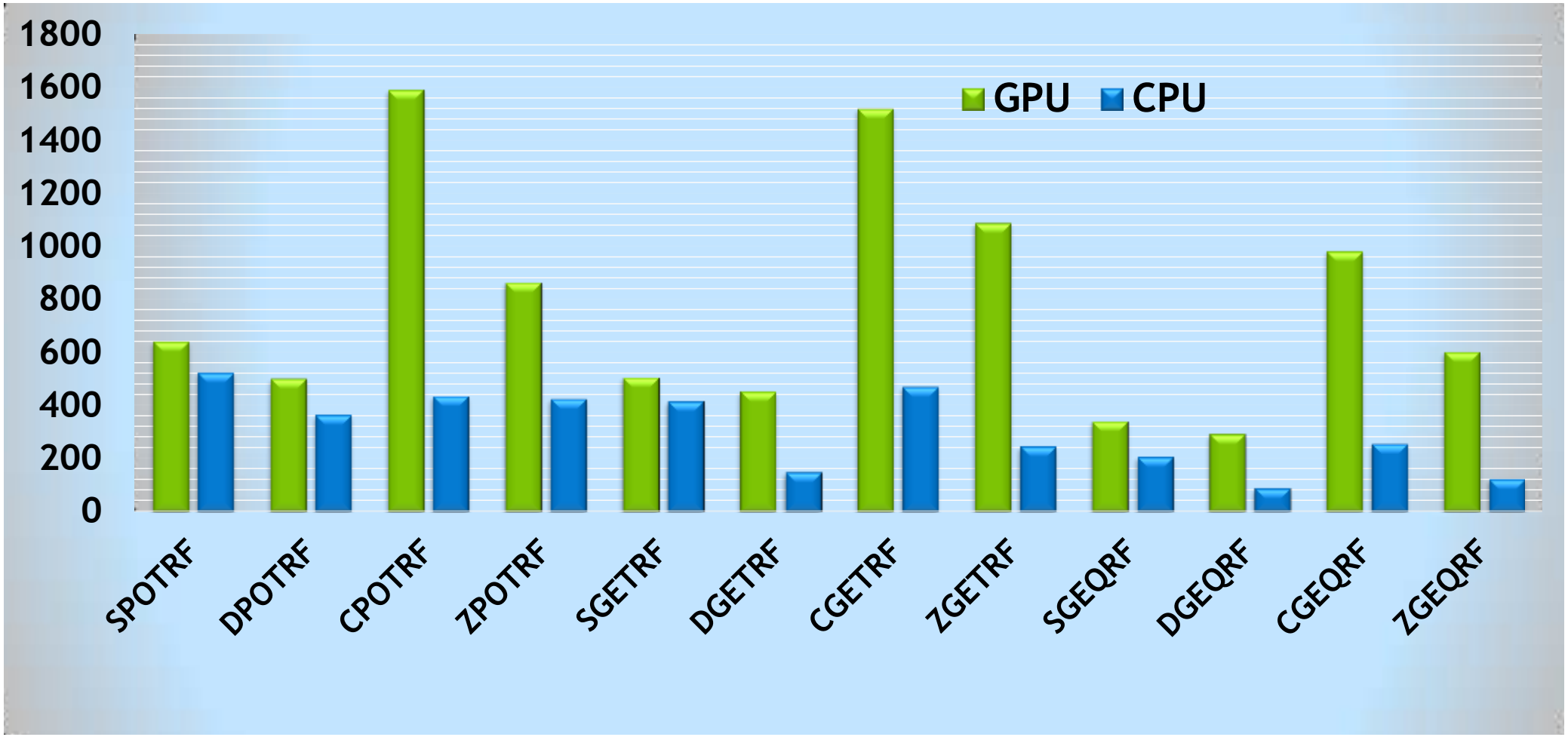


Sample Applications

- Computer Vision
- CFD
- Newton's method
- Chemical Kinetics
- Chemistry
- ODEs
- Circuit Simulation

https://developer.nvidia.com/cusolver

CUSOLVER DENSE GFLOPS VS MKL

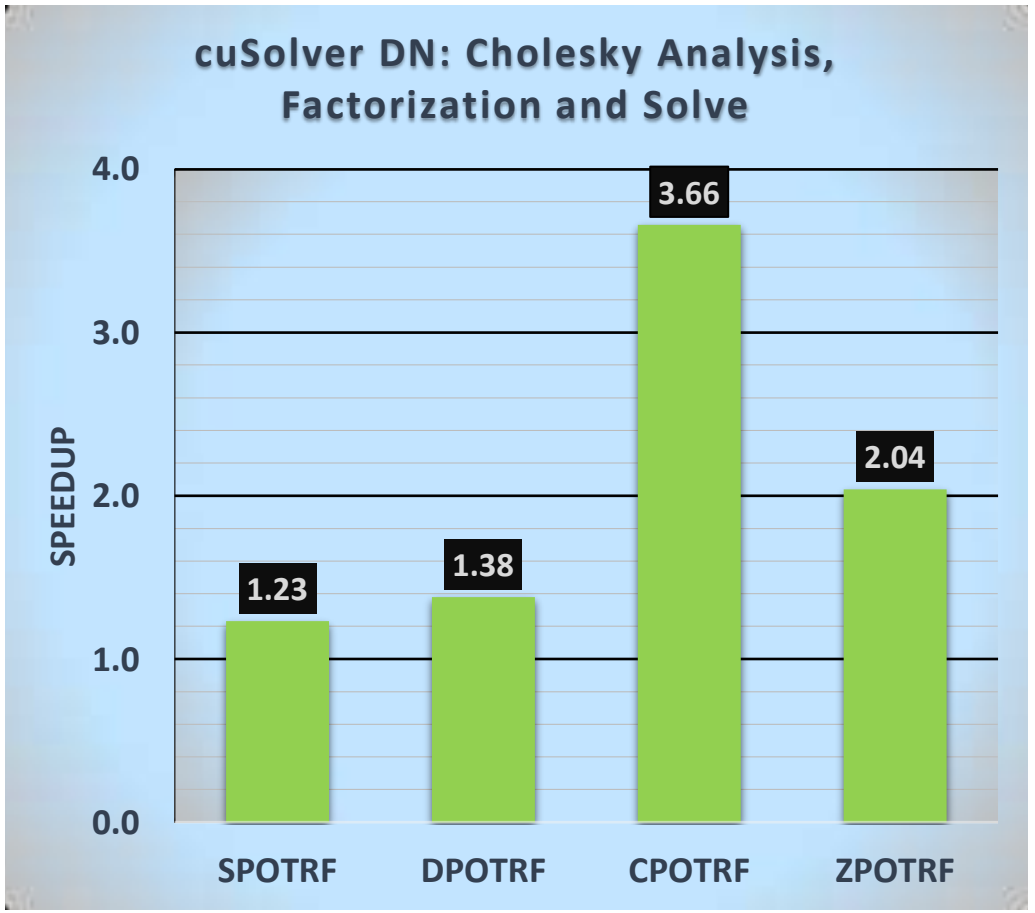


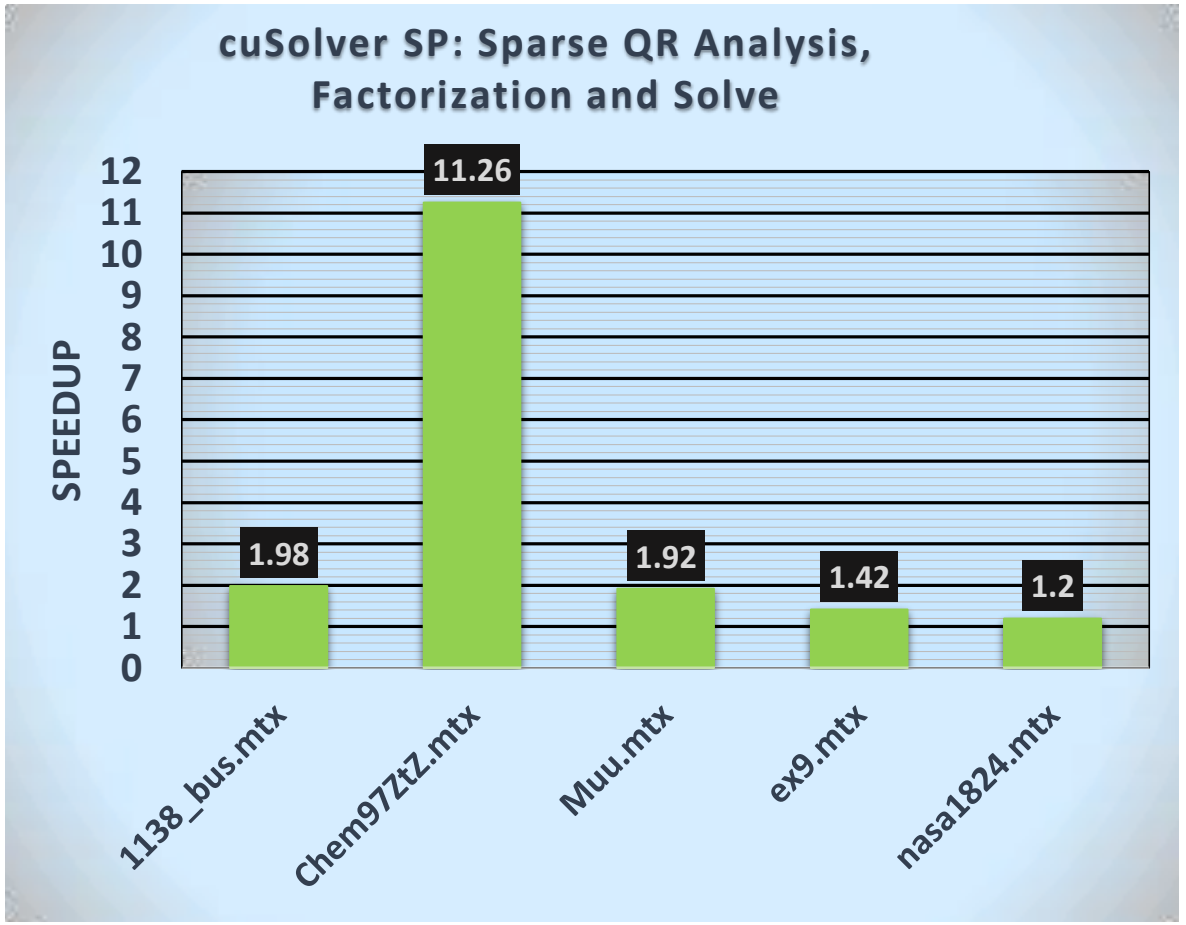
GPU:K40c M=N=4096

CPU: Intel(R) Xeon(TM) E5-2697v3 CPU @ 3.60GHz, 14 cores

MKL v11.04

CUSOLVER SPEEDUP



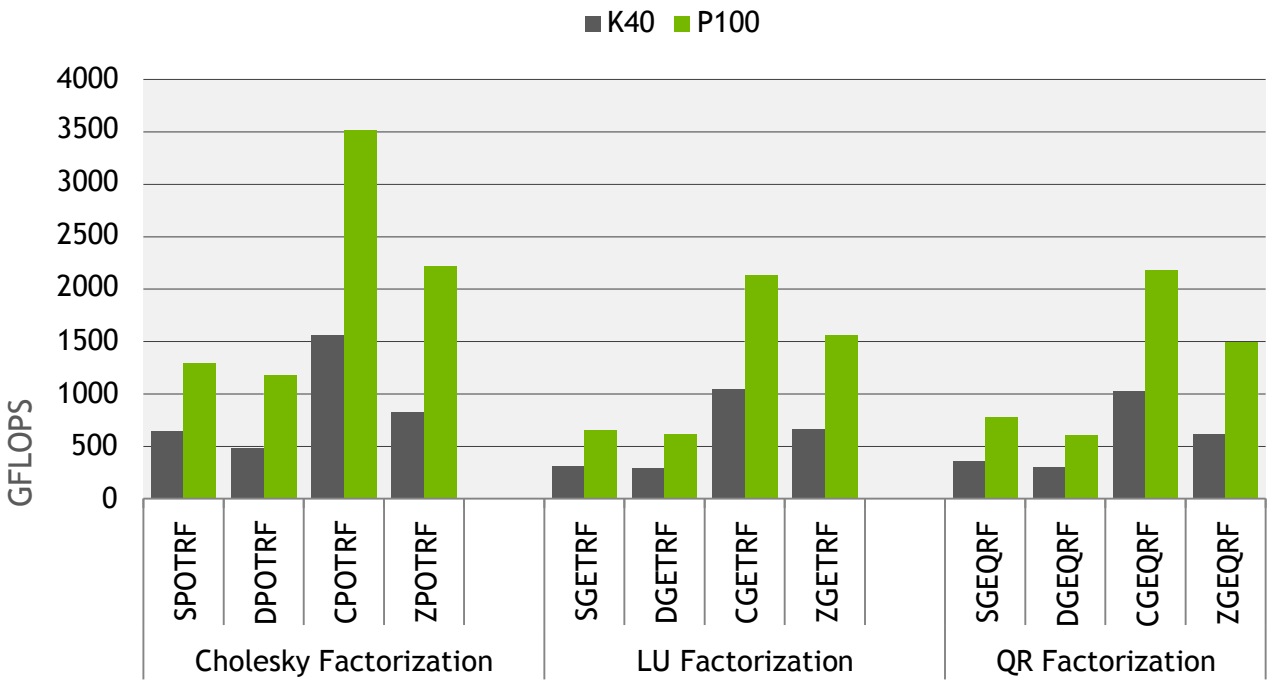


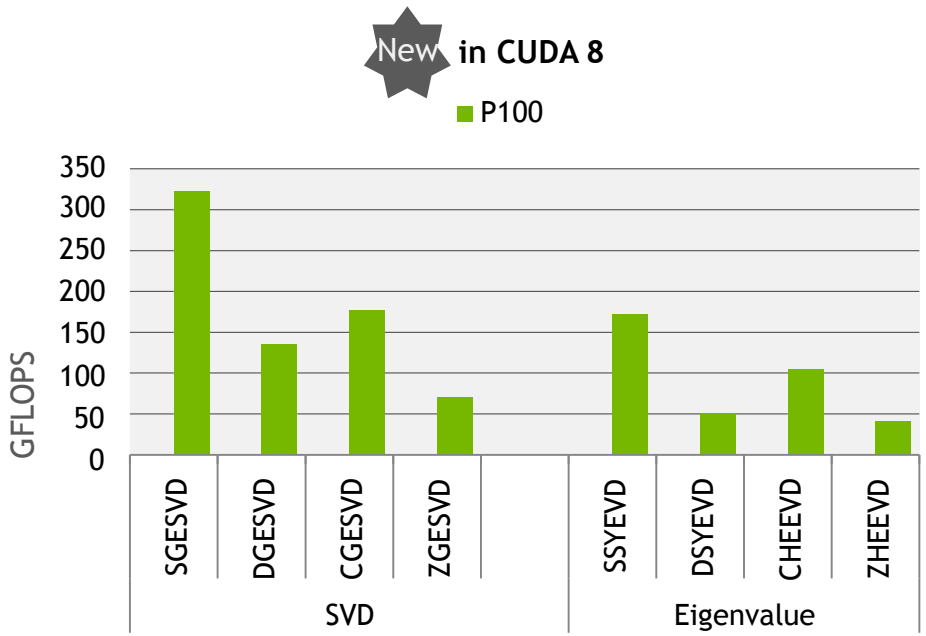
GPU:K40c M=N=4096

CPU: Intel(R) Xeon(TM) E5-2697v3 CPU @ 3.60GHz, 14 cores

MKL v11.04 for Dense Cholesky, Nvidia csr-QR implementation for CPU and GPU

DENSE PERFORMANCE: > 2X FASTER





- cuSOLVER 8 on P100, Base clocks (r361)
- cuSOLVER 7.5 on K40m, Base clocks, ECC ON (r352)
- M=N=4096
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

Performance may vary based on OS and software versions, and motherboard configuration

AmgX

Algebraic Multi-Grid Solvers

Flexible Solver Composition System

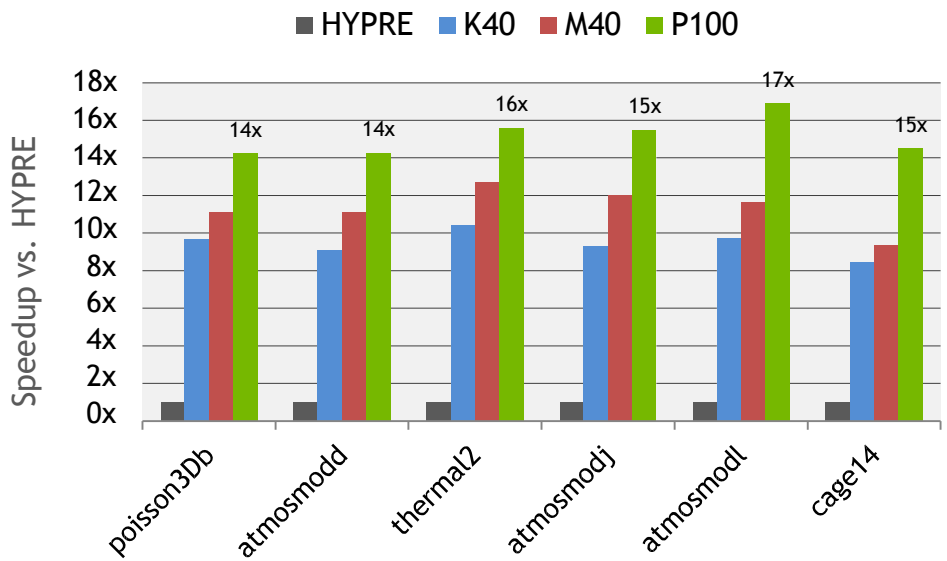
Easy construction of complex nested solvers and pre-conditioners

Flexible and simple high level C API that abstracts parallelism and GPU implementation

Includes Ruge-Steuben, un-smoothed aggregation, Krylov methods and different smoother algorithms

https://developer.nvidia.com/amgx

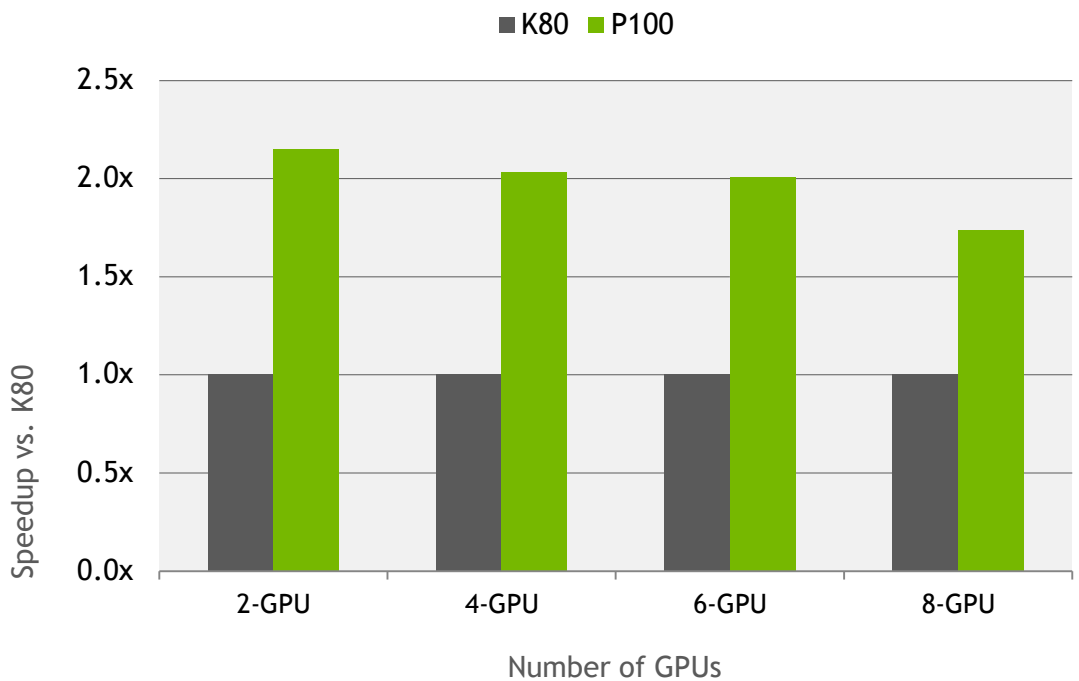
> 15x Speedup vs HYPRE



- Florida Matrix Collection; Total Time to Solution
- HYPRE AMG Package (http://acts.nersc.gov/hypre) on Intel Xeon E5-2697 v4@2.3GHz, 3.6GHz Turbo, Hyperthreading off
- AmgX on K40, M40, P100 (SXM2); Base clocks
- Host system: Intel Xeon Haswell single-socket 16-core E5-2698 v3 @ 2.3GHz, 3.6GHz Turbo
- CentOS 7.2 x86-64 with 128GB System Memory

> 2X FASTER SOLUTION TIMES WITH P100

SPE10 Using Multi-GPUs



- AmgX on K80 and M40 with CUDA 7.5, and P100 (PCIe) with CUDA 8; Base clocks, ECC ON
- Society of Petroleum Engineers 10th comparative simulation model (SPE10)
- Matrix size varied with 100 x Number of GPUs x 200 x 50 for SPE10
- Time to solution includes both setup and solve times

Performance may vary based on OS and software versions, and motherboard configuration

RAPID SOFTWARE DEVELOPMENT ON HETEROGENEOUS SYSTEMS

DROP-IN ACCELERATION WITH GPU LIBRARIES

BLAS | LAPACK | SPARSE | FFT

Math | Deep Learning | Image Processing

5x-10x speedups out of the box

Automatically scale with multi-GPU libraries (cuBLAS-XT, cuFFT-XT, AmgX,…)

75% of developers use GPU libraries to accelerate their application

“DROP-IN” ACCELERATION: CUDA GPU LIBRARY ADVISOR

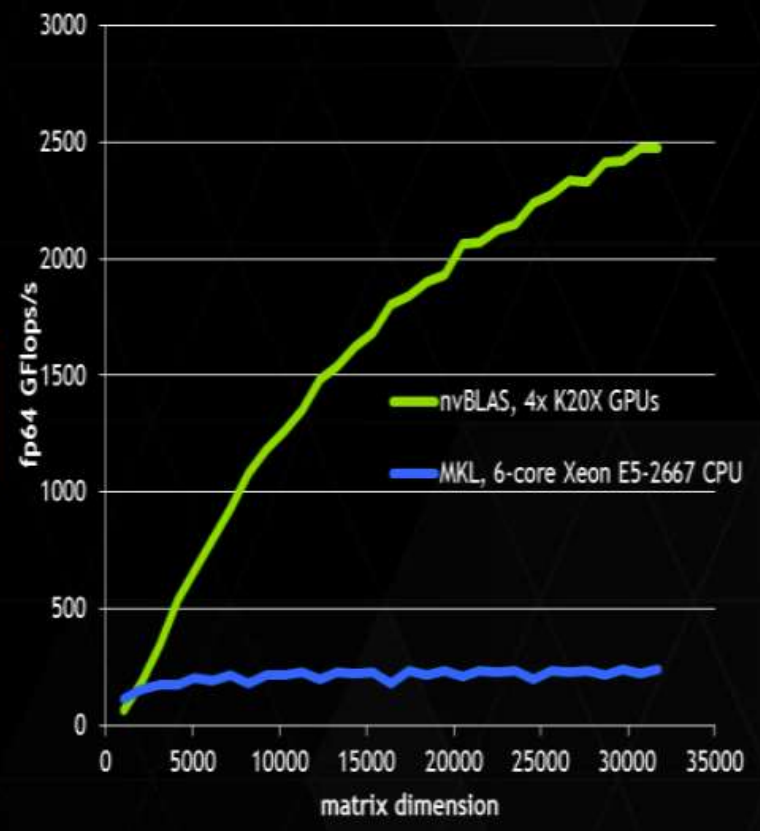
```
./gpu-library-advisor ./cpu_blas 1024 1024
CPU Matrix Multiply. Done.

GPU Library Advisor detected that your application uses functions from BLAS level 3
 that can be accelerated using NVIDIA CUDA.
The NVIDIA NVBLAS library is a GPU-accelerated version of the complete standard BLAS library
 that often delivers faster performance than optimized implementations on high-end CPUs.
The NVBLAS library is freely available and can be used without recompilation of your application.
For documentation on the NVBLAS library, please see http://docs.nvidia.com/cuda/nvblas.

For more information on the performance and the various BLAS library options available from NVIDIA for
GPU acceleration, please see https://developer.nvidia.com/cublas.
```

“DROP-IN” ACCELERATION: NVBLAS

- Automatic Speedup for “R” application

```
  >
LD_PRELOAD=/usr/local/cuda/lib64/libnvblas.so R

  > A <- matrix(rnorm(4096*4096), nrow=4096,
ncol=4096)
  > B <- matrix(rnorm(4096*4096), nrow=4096,
ncol=4096)
  > system.time(C <- A %*% B)

    user  system elapsed

    0.348   0.142   0.289
```

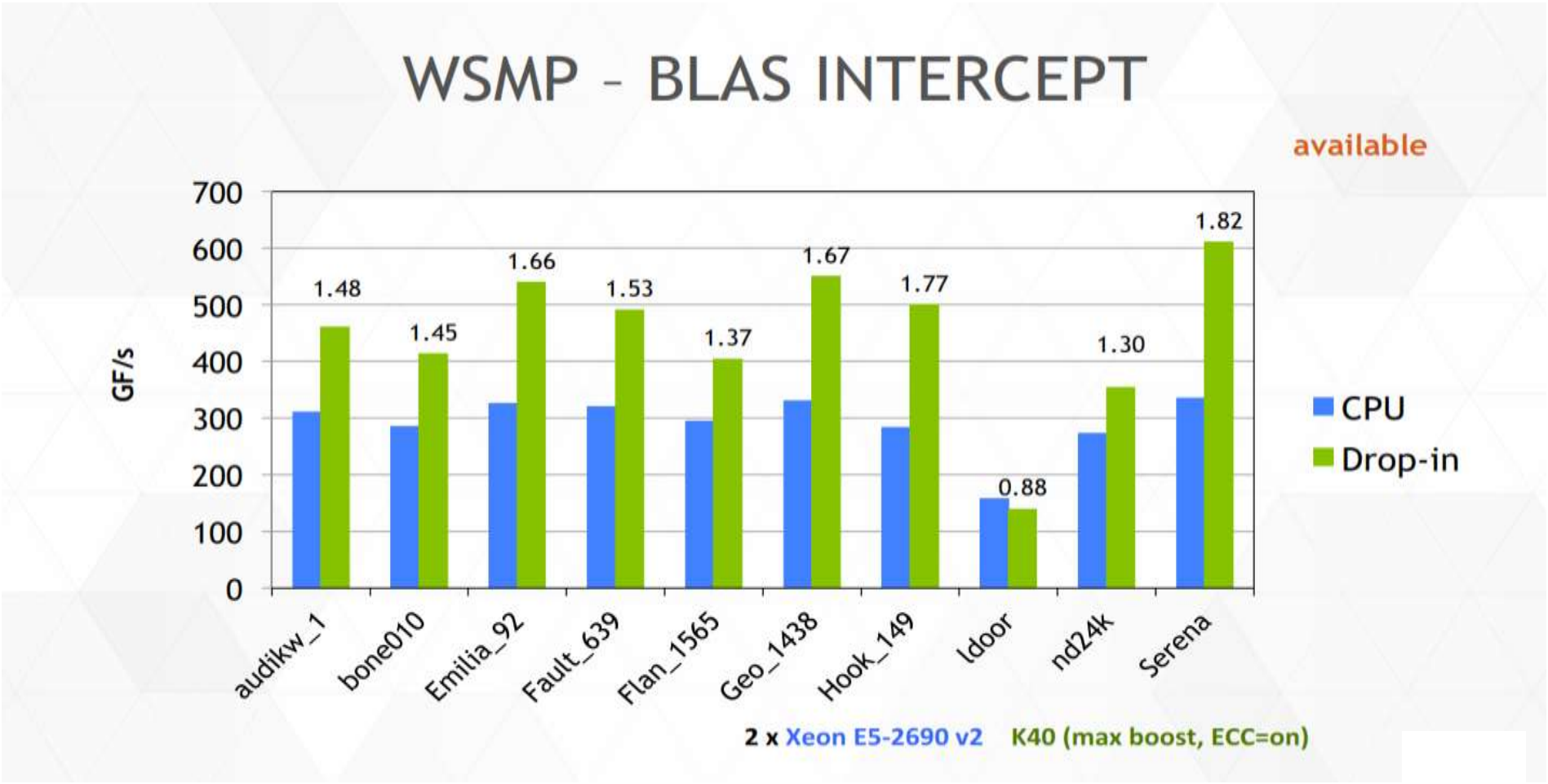
NO CODE CHANGE REQUIRED



- Use in any app that uses standard BLAS3
  - R, Octave, Scilab, etc.

WATSON SPARSE MATRIX PACKAGE

http://on-demand.gputechconf.com/gtc/2014/presentations/S4201-gpu-acceleration-sparse-matrix-factorization-cholmod.pdf
http://on-demand.gputechconf.com/gtc/2015/presentation/S5355-Steve-Rennich.pdf
http://on-demand.gputechconf.com/gtc/2015/presentation/S5232-Natalia-Gimelshein.pdf



ACCELERATING OCTAVE

Scientific Programming Language

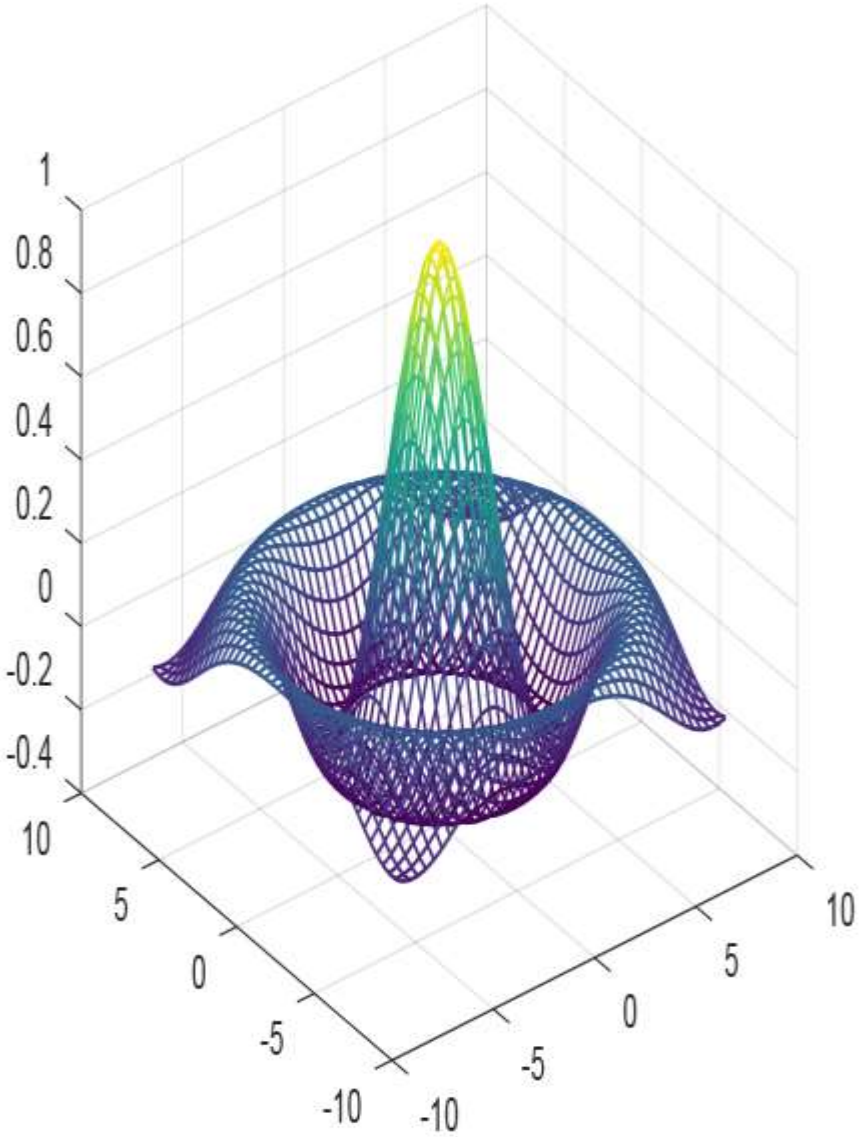
Mathematics-oriented syntax

Drop-in compatible with many MATLAB scripts

Built-in plotting and visualization tools

Runs on GNU/Linux, macOS, BSD, and Windows

Free Software



Source: http://www.gnu.org/software/octave/

INSTALLING OCTAVE (SOURCE CODE)

Download v4.2.1: https://ftp.gnu.org/gnu/octave/

RedHat v7.x | CentOS v7.x

```
mkdir $HOME/octave; cd $HOME/octave
tar xvzf /tmp/octave-4.2.1.tar.gz
cd octave-4.2.1/
sudo yum install libopenblas-dev
sudo ./configure
sudo make -j6 all
sudo make install
export PATH=/usr/local/bin:$PATH
```

INSTALLING OCTAVE (SOURCE CODE)

Download v4.2.1: https://ftp.gnu.org/gnu/octave/

RedHat v7.x | CentOS v7.x

```
mkdir $HOME/octave; cd $HOME/octave
tar xvzf /tmp/octave-4.2.1.tar.gz
cd octave-4.2.1/
sudo yum install libopenblas-dev
sudo ./configure
sudo make -j6 all
sudo make install
export PATH=/usr/local/bin:$PATH
```

Ubuntu 14.04

```
mkdir $HOME/octave; cd $HOME/octave
tar xvzf /tmp/octave-4.2.1.tar.gz
cd octave-4.2.1/
sudo apt-get build-dep octave
sudo apt-get update
sudo ./configure
sudo make -j6 all
sudo make install
export PATH=/usr/local/bin:$PATH
```

CONFIGURING SGEMM TEST

Single Precision Matrix Multiplication

```
$ cd $HOME/octave/octave-4.2.1/scripts

$ vi sgemm.m
```

CONFIGURING SGEMM TEST

Single Precision Matrix Multiplication

```
$ cd $HOME/octave/

$ vi sgemm.m
```

```
#
# Example SGEMM

for N = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]

   A = single( rand(N, N) );
   B = single( rand(N, N) );

   start = clock();
   C = A * B;
   elapsedTime = etime( clock(), start );

   gFlops = 2*N*N*N / (elapsedTime * 1e+9);

   disp( sprintf( "N = %4d, elapsed Time = %9.6f, GFlops = %9.6f\n", \
                  N, elapsedTime, gFlops ) );

endfor
```

CONFIGURING DGEMM TEST

Double Precision Matrix Multiplication

```
$ cd $HOME/octave/octave-4.2.1/scripts

$ vi dgemm.m
```

CONFIGURING DGEMM TEST

Double Precision Matrix Multiplication

```
$ cd $HOME/octave/

$ vi dgemm.m
```

```
#
# Example DGEMM

for N = [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096]

   A = double( rand(N, N) );
   B = double( rand(N, N) );

   start = clock();
   C = A * B;
   elapsedTime = etime( clock(), start );

   gFlops = 2*N*N*N / (elapsedTime * 1e+9);

   disp( sprintf( "N = %4d, elapsed Time = %9.6f, GFlops = %9.6f\n", \
                  N, elapsedTime, gFlops ) );

endfor
```

RUNNING SGEMM TEST

Baseline

```
$ cd $HOME/octave/octave-4.2.1

$ octave-cli scripts/sgemm.m
```

RUNNING SGEMM TEST

Baseline

```
$ cd $HOME/octave/octave-4.2.1

$ octave-cli scripts/sgemm.m
```

```
N =    2, elapsed Time =  0.000206, GFlops =  0.000078
N =    4, elapsed Time =  0.000153, GFlops =  0.000839
N =    8, elapsed Time =  0.000137, GFlops =  0.007457
N =   16, elapsed Time =  0.000137, GFlops =  0.059652
N =   32, elapsed Time =  0.000175, GFlops =  0.373475
N =   64, elapsed Time =  0.000473, GFlops =  1.108379
N =  128, elapsed Time =  0.002541, GFlops =  1.650918
N =  256, elapsed Time =  0.016747, GFlops =  2.003666
N =  512, elapsed Time =  0.077370, GFlops =  3.469517
N = 1024, elapsed Time =  0.546814, GFlops =  3.927266
N = 2048, elapsed Time =  4.635429, GFlops =  3.706209
N = 4096, elapsed Time = 40.742439, GFlops =  3.373361
N = 8192, elapsed Time = 335.189804, GFlops =  3.280266
```

RUNNING SGEMM TEST

Using Multiple CPU cores, and CPU-optimized libopenblas

```
$ cd $HOME/octave/octave-4.2.1

$ OMP_NUM_THREAS=20 LD_PRELOAD=libopenblas.so octave-cli scripts/sgemm.m
```

RUNNING SGEMM TEST

Using Multiple CPU cores, and CPU-optimized libopenblas

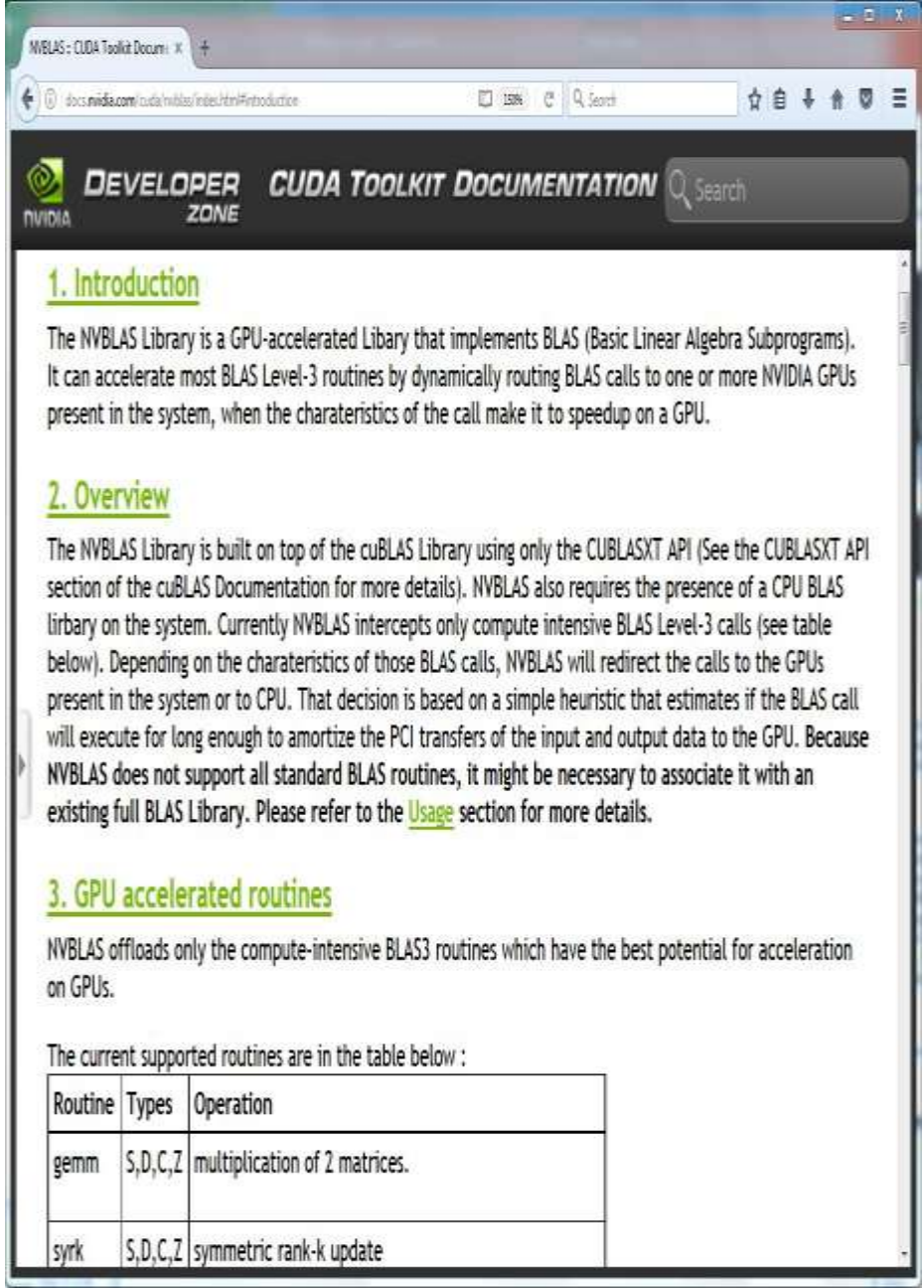
```
$ cd $HOME/octave/octave-4.2.1

$ OMP_NUM_THREAS=20 LD_PRELOAD=li
```

```
N =    2, elapsed Time =  0.000237, GFlops =  0.000068
N =    4, elapsed Time =  0.000137, GFlops =  0.000932
N =    8, elapsed Time =  0.000122, GFlops =  0.008389
N =   16, elapsed Time =  0.000137, GFlops =  0.059652
N =   32, elapsed Time =  0.000130, GFlops =  0.505290
N =   64, elapsed Time =  0.000160, GFlops =  3.272356
N =  128, elapsed Time =  0.000298, GFlops = 14.096303
N =  256, elapsed Time =  0.001022, GFlops = 32.821243
N =  512, elapsed Time =  0.005600, GFlops = 47.935112
N = 1024, elapsed Time =  0.026619, GFlops = 80.674972
N = 2048, elapsed Time =  0.208870, GFlops = 82.251518
N = 4096, elapsed Time =  1.695442, GFlops = 81.063780
N = 8192, elapsed Time = 13.557732, GFlops = 81.098495
```

NVBLAS

Drop-in GPU Acceleration

Routines	Types	Operation
gemm	S,D,C,Z	Multiplication of 2 matrices
syrk	S,D,C,Z	Symmetric rank-k update
herk	C,Z	Hermitian rank-k update
syr2k	S,D,C,Z	Symmetric rank-2k pdate
her2k	C,Z	Hemitian rank-2k update
trsm	S,D,C,Z	Triangular solve, mult right-hand
trmm	S,D,C,Z	Triangular matrix-matrix mult
symm	S,D,C,Z	Symmetric matrix-matrix mult
hemm	C,Z	Hermitian matrix-matrix mult



CONFIGURING NVBLAS

Specify which GPUs participate in intercepted BLAS calls

```
$ cd $HOME/octave/octave-4.2.1

$ vi nvblas.conf
```

CONFIGURING NVBLAS

Specify which GPUs participate in intercepted BLAS calls

```
$ cd $HOME/octave/octave-4.2.1

$ vi nvblas.conf
```

```
#Copyright 2013 NVIDIA Corporation.  All rights reserved.
# This is the configuration file to use NVBLAS Library

NVBLAS_LOGFILE  nvblas.log

NVBLAS_CPU_BLAS_LIB  libopenblas.so

NVBLAS_GPU_LIST 0 1 2 3

NVBLAS_AUTOPIN_MEM_ENABLED
```

RUNNING SGEMM TEST

Using NVBLAS

```
$ cd $HOME/octave/octave-4.2.1

$ export NVBLAS_CONFIG_FILE=$HOME/octave/octave-4.2.1/nvblas.conf

$ LD_PRELOAD=libnvblas.so octave-cli scripts/sgemm.m
```

RUNNING SGEMM TEST

Using NVBLAS

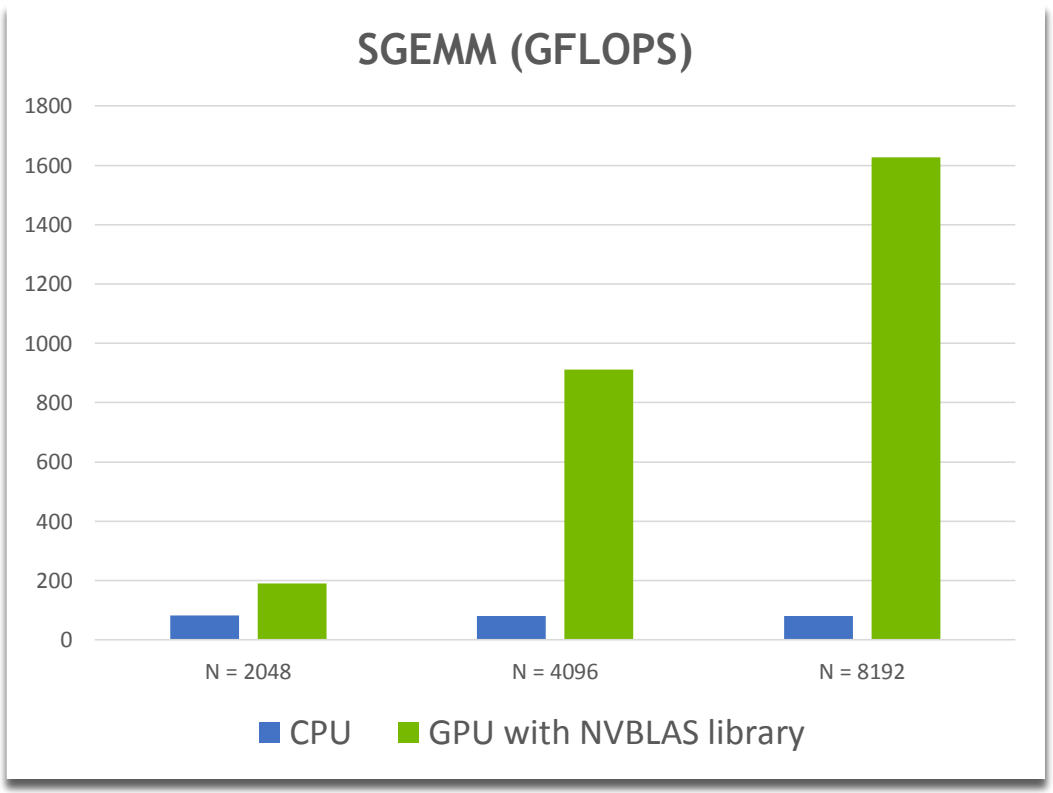
```
$ cd $HOME/octave/octave-4.2.1

$ export NVBLAS_CONFIG_FILE=$HOME/octave/octa

$ LD_PRELOAD=libnvblas.so octave-cli sc
```

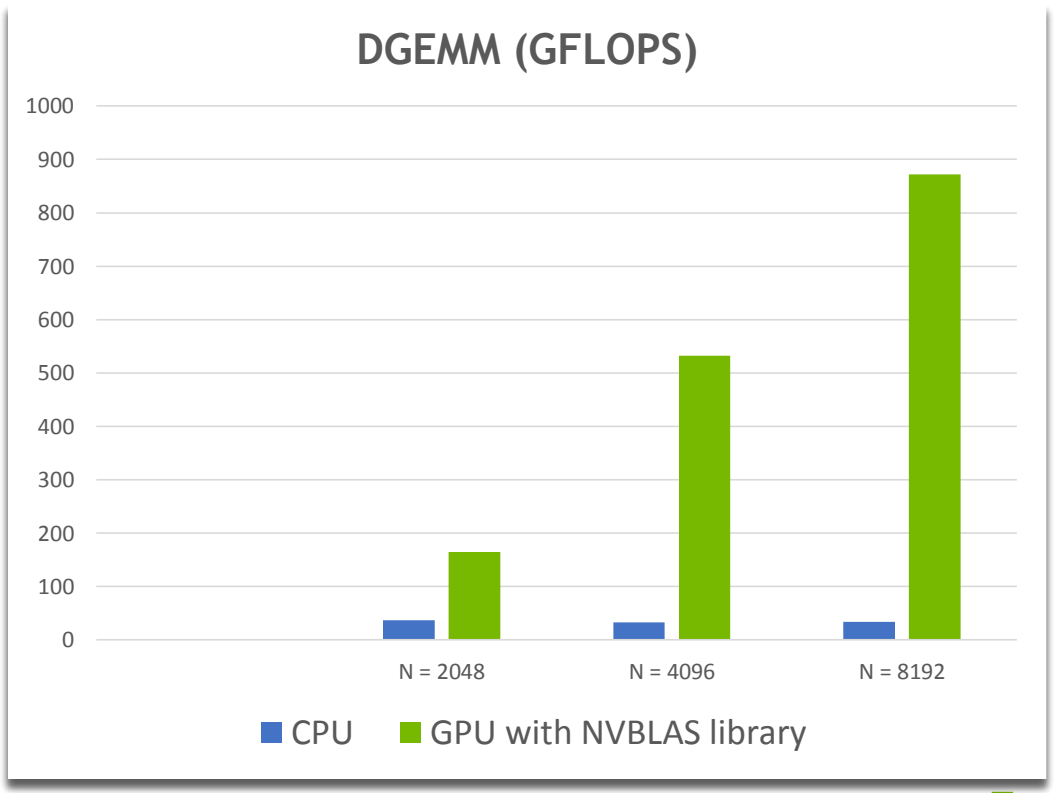
```
NVBLAS_CONFIG_FILE $HOME/octave/octave-4.2.1/nvblas.conf

N =    2, elapsed Time =  0.000755, GFlops =  0.000021

N =    4, elapsed Time =  0.000137, GFlops =  0.000932

N =    8, elapsed Time =  0.000122, GFlops =  0.008389

N =   16, elapsed Time =  0.000137, GFlops =  0.059652

N =   32, elapsed Time =  0.000130, GFlops =  0.505290

N =   64, elapsed Time =  0.000160, GFlops =  3.272356

N =  128, elapsed Time =  0.000237, GFlops = 17.734059

N =  256, elapsed Time =  0.000801, GFlops = 41.886157

N =  512, elapsed Time =  0.071548, GFlops =  3.751799

N = 1024, elapsed Time =  0.075218, GFlops = 28.550053

N = 2048, elapsed Time =  0.090515, GFlops = 189.801063

N = 4096, elapsed Time =  0.150894, GFlops = 910.830140

N = 8192, elapsed Time =  0.675545, GFlops = 1627.592615
```

PERFORMANCE COMPARISON

CPU (openblas) vs GPU (NVBLAS)

Dell C4130 | 128 GB | 36-core, E5-2697 v4 @ 2.30GHz | 4x NVIDIA Tesla P100-SXM2 + NVLINK





PROFILING OCTAVE

Using nvprof

```
$ LD_PRELOAD=libnvblas.so nvprof --print-summary-per-gpu octave-cli-4.2.1 scripts/sgemm.m

==154108== NVPROF is profiling process 154108, command: octave-cli-4.2.1 scripts/sgemm.m

N = 8192, elapsed Time =  0.675545, GFlops = 1627.592615

==154108== Profiling application: octave-cli-4.2.1 scripts/sgemm.m
```

PROFILING OCTAVE

```
$ LD_PRELOA

==154108==

N = 8192, e

==154108==
```

```
==154108== Device "Tesla P100-SXM2-16GB (0)"
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.97%  98.7731s     16385  6.0283ms  1.5040us  8.5727ms  [CUDA memcpy HtoD]
  2.57%  2.61325s      8192  319.00us  275.20us  419.08us  maxwell_sgemm_128x64_raggedMn_nn_splitK
  0.45%  462.77ms       256  1.8077ms  529.86us  2.1076ms  [CUDA memcpy DtoH]
  0.01%  6.5752ms      8192     802ns     541ns  20.035us  [CUDA memset]

==154108== Device "Tesla P100-SXM2-16GB (1)"
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.98%  98.7318s     16385  6.0257ms  1.5040us  8.5792ms  [CUDA memcpy HtoD]
  2.56%  2.60935s      8192  318.52us  273.47us  419.97us  maxwell_sgemm_128x64_raggedMn_nn_splitK
  0.45%  462.49ms       256  1.8066ms  536.29us  2.1133ms  [CUDA memcpy DtoH]
  0.01%  7.1308ms      8192     870ns     500ns  671.54us  [CUDA memset]

==154108== Device "Tesla P100-SXM2-16GB (2)"
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.97%  98.7734s     16385  6.0283ms  1.4720us  8.5664ms  [CUDA memcpy HtoD]
  2.57%  2.61821s      8192  319.61us  274.56us  433.06us  maxwell_sgemm_128x64_raggedMn_nn_splitK
  0.45%  462.22ms       256  1.8055ms  529.16us  2.1152ms  [CUDA memcpy DtoH]
  0.01%  6.6188ms      8192     807ns     520ns  27.498us  [CUDA memset]

==154108== Device "Tesla P100-SXM2-16GB (3)"
Time(%)      Time     Calls       Avg       Min       Max  Name
 96.92%  97.3329s     16385  5.9404ms  1.4720us  8.5063ms  [CUDA memcpy HtoD]
  2.62%  2.63205s      8192  321.29us  274.66us  449.12us maxwell_sgemm_128x64_raggedMn_nn_splitK
  0.45%  454.87ms       256  1.7769ms  642.56us  1.9169ms  [CUDA memcpy DtoH]
  0.01%  7.1363ms      8192     871ns     502ns  571.01us  [CUDA memset]
```

NVBLAS: MORE OPTIONS

NVBLAS_GPU_DISABLED_<BLAS_FUNC_NAME>

E.g. NVBLAS_GPU_DISABLED_SGEMM

NVBLAS_CPU_RATIO_<BLAS_FUNC_NAME>

E.g. NVBLAS_CPU_RATIO_SGEMM 0.07

See CUDA documentation for more information

cuda-8.0/doc/pdf/NVBLAS_Library.pdf

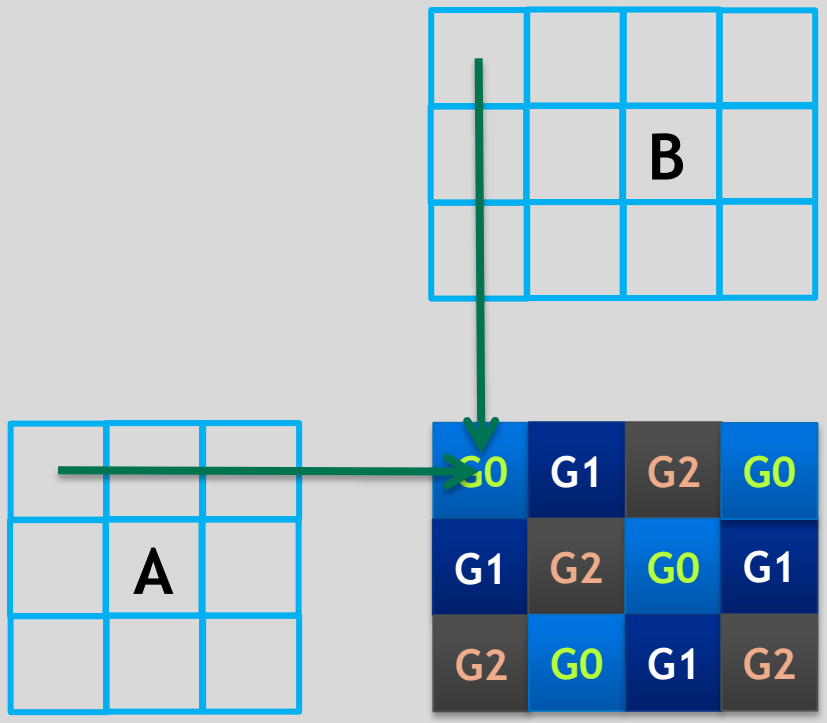
CUBLAS-XT

Level 3 BLAS

Routine	Types	Operation
gemm	S,D,C,Z	multiplication of 2 matrices
syrk	S,D,C,Z	symmetric rank-k update
herk	C,Z	hermitian rank-k update
syr2k	S,D,C,Z	symmetric rank-2k update
her2k	C,Z	hermitian rank-2k update
trsm	S,D,C,Z	triangular solve with multiple right-hand sides
symm	S,D,C,Z	symmetric matrix-matrix multiplication
hemm	C,Z	hermitian matrix-matrix multiplication

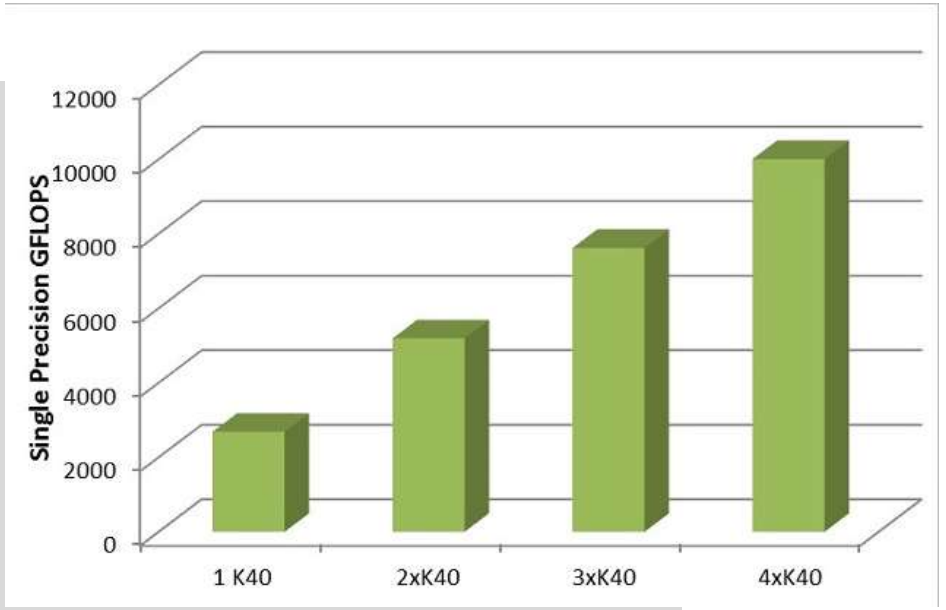
CUBLAS-XT

- Host interface
  - Tiling strategy
- Example cublasXt<t>gemm()
- Tiling for 3 GPUs:



CUBLAS-XT

- Two versions of library
- 64 bit (UVA support)
- Hybrid CPU-GPU computation
  - Currently **cublasXt<t>gemm()**



- Problem size can be larger than available GPU memory

CUBLAS WORKFLOW

```
#include <cublas_v2.h>
// Allocate and fill h_A and h_B with data:
…
cublasCreate(&handle);
cudaMalloc(&d_A, mem_size_A);
… // Allocate d_B, d_C
cudaMemcpy(d_A, h_A, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, cudaMemcpyHostToDevice);

cublasSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, M, N, K,
&alpha, d_A, M, d_B, K, &beta, d_C, M);

cudaMemcpy(h_C, d_C, cudaMemcpyDeviceToHost);
cudaFree(…);
cublasDestroy(handle);
```

CUBLAS-XT EXERCISE 1

- Start from cuBLAS example
  - ‘cd $HOME/cublas-xt’
  - ‘vim ./cublas.cpp’
- Build cublas.cpp and run
  - See instructions.txt on how to build
  - g++ -O2 -I$CUDA_HOME/include -L$CUDA_HOME/lib64 -lcublas -lcudart cublas.cpp
  - ‘srun a.out’

~ 1050 GFLOPS

CUBLAS-XT WORKFLOW

```
// Allocate and fill h_A and h_B with data:
#include <cublasXt.h>
…
cublasXtHandle_t handle;
cublasXtCreate(&handle);
const int nDevices = 2;
int deviceId[nDevices] = {0, 1};

cublasXtDeviceSelect(handle, nDevices, deviceId);

cublasXtSgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, M, N, K,
&alpha, h_A, M, h_B, K, &beta, h_C, M);

cublasXtDestroy(handle);
```

CUBLAS-XT EXERCISE 2

- cp cublas.cpp cublas-xt.cpp
- Implement the same functionality using XT features
- g++ -I$CUDA_HOME/include -L/$CUDA_HOME/lib64 -lcublas -lcudart cublas-xt.cpp
- Execute on 2 GPUs

~ **772 GFLOPS on 2 GPUs**

cp progress/cublas-xt.cpp .

CUBLAS-XT EXERCISE 3

Optimize cublas-xt.cpp

```
// Use pinned memory

cudaMallocHost((void **) &h_A, mem_size_A);
cudaFreeHost((void *) h_A);

// Optional: optimize tile size

int blockDim;

// Get current tile size

cublasXtGetBlockDim(handle, &blockDim);

// Set tile size

cublasXtSetBlockDim(handle, blockDim);
```

CUBLAS-XT EXERCISE 3

Optimize cublas-xt.cpp

~ 2000 GFLOPS using `cudaMallocHost`

Alternative route:

```
cublasXtSetPinningMemMode(handle,CUBLASXT_PINNING_ENABLED
);
```

Calls `cudaHostRegister/cudaHostUnregister` on the fly

~ 1700 GFLOPS

cp progress/cublas-opt-xt.cpp .

CUBLAS-XT: MORE OPTIONS

cublasXt(G,S)etPinningMemMode(…)

cublasXtSetCpuRatio(…)

Other cublasXt API Math Functions

See CUDA documentation for more information

cuda-8.0/doc/pdf/CUBLAS_Library.pdf

MORE INFO

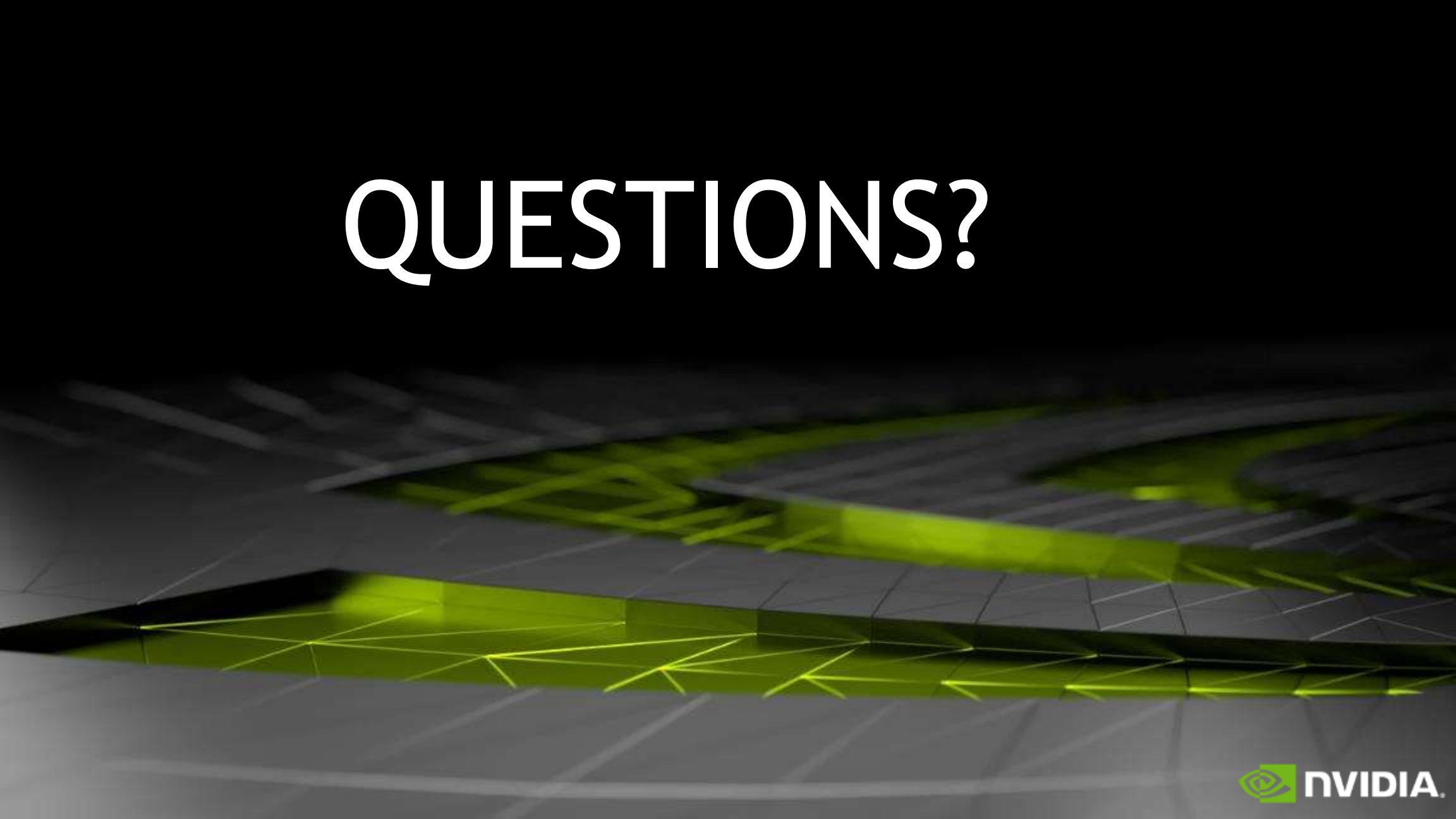
CUDA TOOLKIT : https://docs.nvidia.com/cuda/index.html

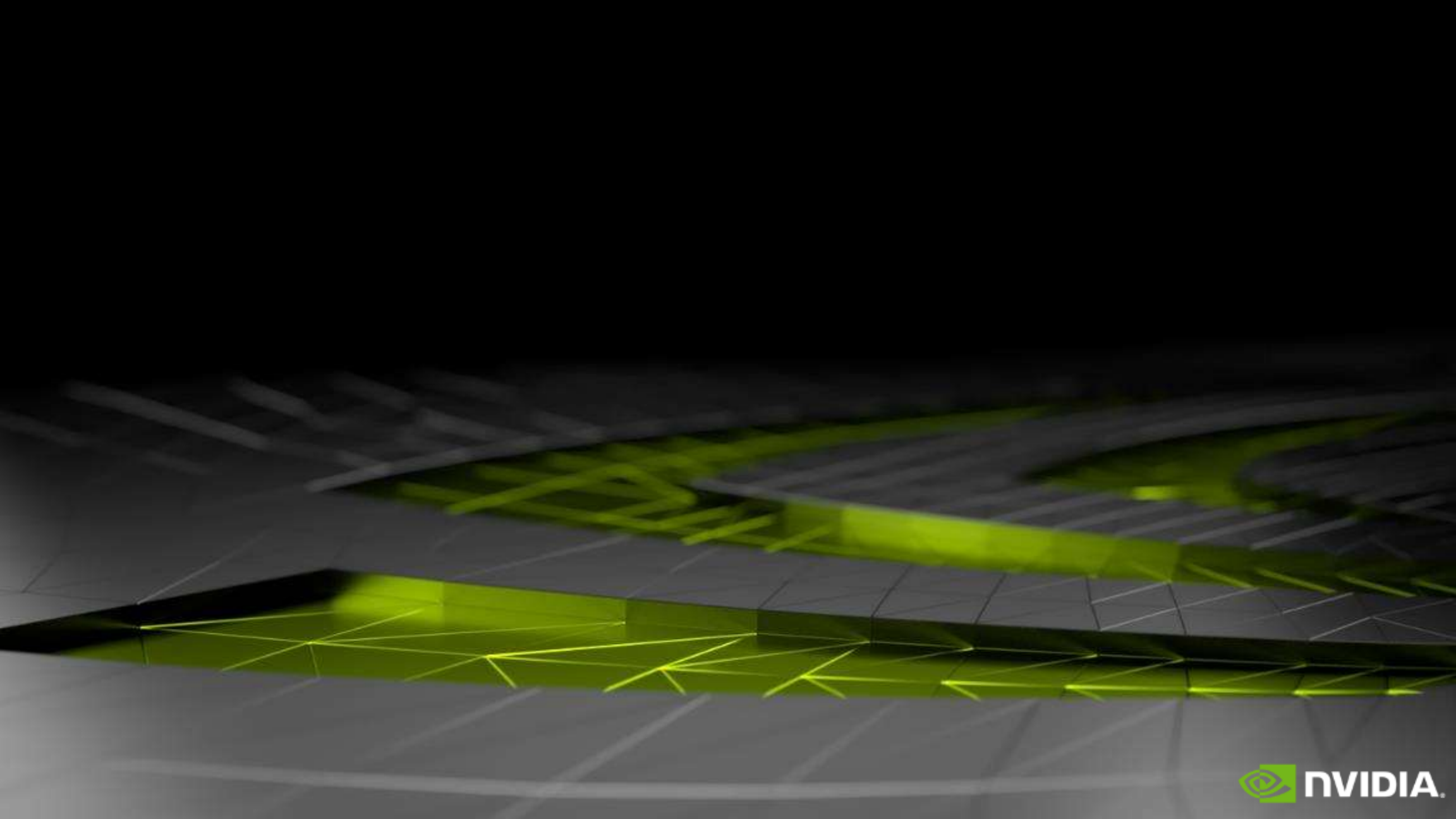
CUDA LIBRARY ADVISOR : https://docs.nvidia.com/cuda/gpu-library-advisor/index.html

- **NVBLAS : https://docs.nvidia.com/cuda/nvblas/index.html**
- **CUBLAS : https://docs.nvidia.com/cuda/cublas/index.html**
- **CUSPARSE: https://docs.nvidia.com/cuda/cusparse/index.html**
- **CUSOLVER: https://docs.nvidia.com/cuda/cusolver/index.html**

CUDA SAMPLES : https://docs.nvidia.com/cuda/cuda-samples/index.html#cudalibraries

QUESTIONS?




NVIDIA.