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

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

AlexNet Training Performance

ONE ARCHITECTURE BUILT FOR BOTH DATA SCIENCE & COMPUTATIONAL SCIENCE

Pascal & Volta

NVIDIA DGX-1

NVIDIA DGX SATURNV

65x in 3 Years

AlexNet Training Performance
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

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-5</th>
<th>Weights</th>
<th>FLOPs</th>
<th>S6</th>
<th>Titan X</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>80.03</td>
<td>61M</td>
<td>725M</td>
<td>117ms</td>
<td>245mJ</td>
</tr>
<tr>
<td>AlexNet* (imp.)</td>
<td>78.33</td>
<td>11M</td>
<td>272M</td>
<td>43ms</td>
<td>72mJ</td>
</tr>
<tr>
<td>VGG-S</td>
<td>84.60</td>
<td>103M</td>
<td>2640M</td>
<td>357ms</td>
<td>825mJ</td>
</tr>
<tr>
<td>VGG-S*</td>
<td>84.05</td>
<td>14M</td>
<td>549M</td>
<td>97ms</td>
<td>193mJ</td>
</tr>
<tr>
<td>GoogLeNet (imp.)</td>
<td>88.90</td>
<td>6.9M</td>
<td>1566M</td>
<td>273ms</td>
<td>473mJ</td>
</tr>
<tr>
<td>GoogLeNet* (imp.)</td>
<td>88.66</td>
<td>4.7M</td>
<td>760M</td>
<td>192ms</td>
<td>296mJ</td>
</tr>
<tr>
<td>VGG-16</td>
<td>89.90</td>
<td>138M</td>
<td>15484M</td>
<td>1926ms</td>
<td>4757mJ</td>
</tr>
<tr>
<td>VGG-16* (imp.)</td>
<td>89.40</td>
<td>127M</td>
<td>3139M</td>
<td>576ms</td>
<td>1346mJ</td>
</tr>
</tbody>
</table>

[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

CUDA 8

Pascal GPU

Unified Memory

Allocate Beyond GPU Memory Size

CPU

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

- **Libraries**: Easy to use, Most Performance
- **Compiler Directives**: Easy to use, Portable code
- **Programming Languages**: Most Performance, Most Flexibility
NVIDIA SOLUTIONS FOR ACCELERATED LINEAR ALGEBRA
Sparse Problems

- **Linear Systems**
  \[ A \mathbf{x} = \mathbf{f} \]

- **Linear Least Squares**
  \[ \min ||B \mathbf{y} - \mathbf{f}||_2 \]

- **Eigenvalue Problems**
  \[ A \mathbf{V} = \mathbf{V} \mathbf{D} \]

- **Singular Value Decomposition**
  \[ A = U \mathbf{D} V^T \]
GPU-Accelerated Libraries

- CUDA Toolkit Libraries
  - CUSPARSE, CUSOLVER, CUBLAS

- NVIDIA Proprietary libraries
  - AmgX
  - NVGRAPH

- Third Party libraries
  - Trilinos, PETSc
  - ArrayFire, CHOLMOD
  - MAGMA

2. cuBLAS and nvBLAS
BLAS on GPUs

1. NVBLAS

2. cuBLAS-XT

cuBLAS
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+lda], lda);
        cublasGetMatrix(nb, nb, sizeof(double), &A[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+lda], lda);
        dpotf2_cpu(nb, work, nb);
        cublasSetMatrix(nb, nb, sizeof(double), work, nb, &A[j], lda);
        cublasDtrsm('R', 'L', 'T', 'N', n-j-nb, nb, 1.0, &A[j], lda, &A[j+nb], 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

<table>
<thead>
<tr>
<th>Routine</th>
<th>Types</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gemm</td>
<td>S,D,C,Z</td>
<td>multiplication of 2 matrices</td>
</tr>
<tr>
<td>syrk</td>
<td>S,D,C,Z</td>
<td>symmetric rank-k update</td>
</tr>
<tr>
<td>herk</td>
<td>C,Z</td>
<td>hermitian rank-k update</td>
</tr>
<tr>
<td>syr2k</td>
<td>S,D,C,Z</td>
<td>symmetric rank-2k update</td>
</tr>
<tr>
<td>her2k</td>
<td>C,Z</td>
<td>hermitian rank-2k update</td>
</tr>
<tr>
<td>trsm</td>
<td>S,D,C,Z</td>
<td>triangular solve with multiple right-hand sides</td>
</tr>
<tr>
<td>symm</td>
<td>S,D,C,Z</td>
<td>symmetric matrix-matrix multiplication</td>
</tr>
<tr>
<td>hemm</td>
<td>C,Z</td>
<td>hermitian matrix-matrix multiplication</td>
</tr>
</tbody>
</table>
### cuBLAS-XT

#### Level 3 BLAS

<table>
<thead>
<tr>
<th>Routine</th>
<th>Types</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gemm</td>
<td>S,D,C,Z</td>
<td>multiplication of 2 matrices</td>
</tr>
<tr>
<td>syrk</td>
<td>S,D,C,Z</td>
<td>symmetric rank-k update</td>
</tr>
<tr>
<td>herk</td>
<td>C,Z</td>
<td>hermitian rank-k update</td>
</tr>
<tr>
<td>syr2k</td>
<td>S,D,C,Z</td>
<td>symmetric rank-2k update</td>
</tr>
<tr>
<td>her2k</td>
<td>C,Z</td>
<td>hermitian rank-2k update</td>
</tr>
<tr>
<td>trsm</td>
<td>S,D,C,Z</td>
<td>triangular solve with multiple right-hand sides</td>
</tr>
<tr>
<td>symm</td>
<td>S,D,C,Z</td>
<td>symmetric matrix-matrix multiplication</td>
</tr>
<tr>
<td>hemm</td>
<td>C,Z</td>
<td>hermitian matrix-matrix multiplication</td>
</tr>
</tbody>
</table>
Large dense matrix fronts factored on GPU if not too large for GPU memory

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

**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
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()`
  
  \[ y = \alpha \ast \text{op}(A)\ast x + \beta \ast y \]
  
  \( A = \text{dense matrix} \)
  
  \( x = \text{sparse vector} \)
  
  \( y = \text{dense vector} \)

Sparse vector could be frequencies of words in a text sample.
5. AmgX Library
Nested Solvers

Solver hierarchy, e.g.

- **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

Example solvers
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

Academia
- CHOLMOD
- MAGMA (Matrix Algebra on GPU and Multicore)
- FLAME Library
- PETSc
- Trilinos

National Labs
- Wolfram Mathematica

Industry
- ArrayFire Matrix Computations
- STONE RIDGE TECHNOLOGY
- CULA tools (GPU Accelerated Linear Algebra)
- Rogue Wave Software
- IMSL Library

Toolboxes
- MAILAB
- AMBER
- Wolfram Mathematica
Third Party Libraries

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

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

http://trilinos.sandia.gov/

Third Party Libraries

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

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

---

http://arrayfire.com/

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

<table>
<thead>
<tr>
<th>PAGERANK</th>
<th>SINGLE SOURCE SHORTEST PATH</th>
<th>SINGLE SOURCE WIDEST PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>Robotic path planning</td>
<td>IP routing</td>
</tr>
<tr>
<td>Recommendation engines</td>
<td>Power network planning</td>
<td>Chip design / EDA</td>
</tr>
<tr>
<td>Social Ad placement</td>
<td>Logistics &amp; supply chain planning</td>
<td>Traffic sensitive routing</td>
</tr>
</tbody>
</table>

developer.nvidia.com/nvgraph

nvGRAPH: 3x Speedup

PageRank on Twitter 1.5B edge dataset

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

<table>
<thead>
<tr>
<th>PageRank</th>
<th>Single Source Shortest Path</th>
<th>Single Source Widest Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search</td>
<td>Robotic Path Planning</td>
<td>IP Routing</td>
</tr>
<tr>
<td>Recommendation Engines</td>
<td>Power Network Planning</td>
<td>Chip Design / EDA</td>
</tr>
<tr>
<td>Social Ad Placement</td>
<td>Logistics &amp; Supply Chain Planning</td>
<td>Traffic sensitive routing</td>
</tr>
</tbody>
</table>
> 200X SPEEDUP ON PAGERANK VS GALOIS

![Chart showing speedup comparison between Galois, GraphMat, M40, and P100 for PageRank and PageRank soc-LiveJournal/Twitter datasets.]

- 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

- 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 2B-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

https://developer.nvidia.com/cublas
cuBLAS: > 8 TFLOPS SINGLE PRECISION

16 TFLOPS FP16 GEMM Performance

32 TFLOPS INT8 GEMM Performance

- cuBLAS 8 on P100 (r361) and P40 (r367); Base clocks
- cuBLAS 7.5 on K40m; Base clocks, ECC OFF (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

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

Used in language modeling and auto-encoders for recommender systems

Improved GEMVI Performance 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
- 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

1.75x
cuSPARSE: > 4X FASTER WITH P100

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

**cuSolver DN: Cholesky Analysis, Factorization and Solve**

- SPOTRF: 1.23
- DPOTRF: 1.38
- CPOTRF: 3.66
- ZPOTRF: 2.04

**cuSolver SP: Sparse QR Analysis, Factorization and Solve**

- 1138_bus.mtx: 1.98
- Chem97/Z1.mtx: 11.26
- Muu.mtx: 1.92
- ex9.mtx: 1.42
- nasa1824.mtx: 1.2

Specifications:

- 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

> 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

https://developer.nvidia.com/amgx
> 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
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

```r
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

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

Matrix-Matrix Multiplication in R

NO CODE CHANGE REQUIRED
WATSON SPARSE MATRIX PACKAGE


WSMP - BLAS INTERCEPT

available

![Graph showing performance comparison between CPU and Drop-in for various benchmarks](image-url)
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
$ cd $HOME/octave/octave
$ vi sgemm.m

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 = %.9f, GFlops = %.9f\n", N, elapsedTime, gFlops ) );
endfor
$ cd $HOME/octave/octave-4.2.1/scripts

$ vi dgemm.m
$ cd $HOME/octave/octave-

$ vi dgemm.m

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_THREADS=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_THREADS=20 LD_PRELOAD=libopenblas.octave octave scripts/sgemm.m

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 = 82.151518
N = 4096, elapsed Time =  1.695442, GFlops = 81.037800
N = 8192, elapsed Time = 13.557732, GFlops = 81.098495
NVBLAS
Drop-in GPU Acceleration

<table>
<thead>
<tr>
<th>Routines</th>
<th>Types</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gemm</td>
<td>S,D,C,Z</td>
<td>Multiplication of 2 matrices</td>
</tr>
<tr>
<td>syrk</td>
<td>S,D,C,Z</td>
<td>Symmetric rank-k update</td>
</tr>
<tr>
<td>herk</td>
<td>C,Z</td>
<td>Hermitian rank-k update</td>
</tr>
<tr>
<td>syr2k</td>
<td>S,D,C,Z</td>
<td>Symmetric rank-2k update</td>
</tr>
<tr>
<td>her2k</td>
<td>C,Z</td>
<td>Hermitian rank-2k update</td>
</tr>
<tr>
<td>trsm</td>
<td>S,D,C,Z</td>
<td>Triangular solve, mult right-hand</td>
</tr>
<tr>
<td>trmm</td>
<td>S,D,C,Z</td>
<td>Triangular matrix-matrix mult</td>
</tr>
<tr>
<td>symm</td>
<td>S,D,C,Z</td>
<td>Symmetric matrix-matrix mult</td>
</tr>
<tr>
<td>hemm</td>
<td>C,Z</td>
<td>Hermitian matrix-matrix mult</td>
</tr>
</tbody>
</table>

1. Introduction
The NVBLAS Library is a GPU-accelerated library that implements BLAS (Basic Linear Algebra Subprograms). It can accelerate most BLAS Level-3 routines by dynamically routing BLAS calls to one or more NVIDIA GPUs present in the system, when the characteristics of the call make it to speedup on a GPU.

2. Overview
The NVBLAS Library is built on top of the cuBLAS Library using only the CUBLAS API. (See the CUBLAS API section of the cuBLAS Documentation for more details). NVBLAS also requires the presence of a GPU BLAS library on the system. Currently, NVBLAS intercepts only compute intensive BLAS Level-3 calls (see table below). Depending on the characteristics of those BLAS calls, NVBLAS will redirect the calls to the GPUs present in the system or to CPU. That decision is based on a simple heuristic that estimates if the BLAS call will execute for long enough to amortize the PCI transfers of the input and output data to the GPU. Because NVBLAS does not support all standard BLAS routines, it might be necessary to associate it with an existing full BLAS Library. Please refer to the Usage section for more details.

3. GPU accelerated routines
NVBLAS offloads only the compute-intensive BLAS3 routines which have the best potential for acceleration on GPUs.

The current supported routines are in the table below:

<table>
<thead>
<tr>
<th>Routine</th>
<th>Types</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gemm</td>
<td>S,D,C,Z</td>
<td>Multiplication of 2 matrices.</td>
</tr>
<tr>
<td>syrk</td>
<td>S,D,C,Z</td>
<td>Symmetric rank-k update.</td>
</tr>
</tbody>
</table>
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/octave-4.2.1/nvblas.conf

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

Using NVBLAS

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_PRELOAD=$/path/to/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

Using nvprof

<table>
<thead>
<tr>
<th>Time(%)</th>
<th>Time</th>
<th>Calls</th>
<th>Avg</th>
<th>Min</th>
<th>Max</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>96.97%</td>
<td>98.7731s</td>
<td>16385</td>
<td>6.0283ms</td>
<td>1.5040us</td>
<td>8.5727ms</td>
<td>[CUDA memcpy HtoD]</td>
</tr>
<tr>
<td>2.57%</td>
<td>2.61325s</td>
<td>8192</td>
<td>319.00us</td>
<td>275.20us</td>
<td>419.08us</td>
<td>maxwell_sgemm_128x64_raggedMn_nn_splitK</td>
</tr>
<tr>
<td>0.45%</td>
<td>462.77ms</td>
<td>256</td>
<td>1.8077ms</td>
<td>529.86us</td>
<td>2.1076ms</td>
<td>[CUDA memcpyDtoH]</td>
</tr>
<tr>
<td>0.01%</td>
<td>6.5752ms</td>
<td>8192</td>
<td>802ns</td>
<td>541ns</td>
<td>20.035us</td>
<td>[CUDA memset]</td>
</tr>
</tbody>
</table>
```

```
==154108== Device "Tesla P100-SXM2-16GB (0)"

Time(%)  Time     Calls | Avg       | Min       | Max       | Name                               |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>96.98%</td>
<td>98.7318s</td>
<td>16385</td>
<td>6.0257ms</td>
<td>1.5040us</td>
<td>8.5792ms</td>
<td>[CUDA memcpy HtoD]</td>
</tr>
<tr>
<td>2.56%</td>
<td>2.60935s</td>
<td>8192</td>
<td>318.52us</td>
<td>273.47us</td>
<td>419.97us</td>
<td>maxwell_sgemm_128x64_raggedMn_nn_splitK</td>
</tr>
<tr>
<td>0.45%</td>
<td>462.49ms</td>
<td>256</td>
<td>1.8066ms</td>
<td>536.29us</td>
<td>2.1133ms</td>
<td>[CUDA memcpyDtoH]</td>
</tr>
<tr>
<td>0.01%</td>
<td>7.1308ms</td>
<td>8192</td>
<td>870ns</td>
<td>500ns</td>
<td>671.54us</td>
<td>[CUDA memset]</td>
</tr>
</tbody>
</table>
```

```
==154108== Device "Tesla P100-SXM2-16GB (1)"

Time(%)  Time     Calls | Avg       | Min       | Max       | Name                               |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>96.97%</td>
<td>98.7734s</td>
<td>16385</td>
<td>6.0283ms</td>
<td>1.4720us</td>
<td>8.5063ms</td>
<td>[CUDA memcpy HtoD]</td>
</tr>
<tr>
<td>2.57%</td>
<td>2.61821s</td>
<td>8192</td>
<td>321.29us</td>
<td>274.66us</td>
<td>449.12us</td>
<td>maxwell_sgemm_128x64_raggedMn_nn_splitK</td>
</tr>
<tr>
<td>0.45%</td>
<td>462.22ms</td>
<td>256</td>
<td>1.8055ms</td>
<td>529.16us</td>
<td>2.1152ms</td>
<td>[CUDA memcpyDtoH]</td>
</tr>
<tr>
<td>0.01%</td>
<td>6.6188ms</td>
<td>8192</td>
<td>807ns</td>
<td>520ns</td>
<td>27.498us</td>
<td>[CUDA memset]</td>
</tr>
</tbody>
</table>
```

```
==154108== Device "Tesla P100-SXM2-16GB (2)"

Time(%)  Time     Calls | Avg       | Min       | Max       | Name                               |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>96.92%</td>
<td>97.3329s</td>
<td>16385</td>
<td>5.9404ms</td>
<td>1.4720us</td>
<td>8.5063ms</td>
<td>[CUDA memcpy HtoD]</td>
</tr>
<tr>
<td>2.62%</td>
<td>2.63205s</td>
<td>8192</td>
<td>321.29us</td>
<td>274.66us</td>
<td>449.12us</td>
<td>maxwell_sgemm_128x64_raggedMn_nn_splitK</td>
</tr>
<tr>
<td>0.45%</td>
<td>454.87ms</td>
<td>256</td>
<td>1.7769ms</td>
<td>642.56us</td>
<td>1.9169ms</td>
<td>[CUDA memcpyDtoH]</td>
</tr>
<tr>
<td>0.01%</td>
<td>7.1363ms</td>
<td>8192</td>
<td>871ns</td>
<td>502ns</td>
<td>571.01us</td>
<td>[CUDA memset]</td>
</tr>
</tbody>
</table>
```
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

<table>
<thead>
<tr>
<th>Routine</th>
<th>Types</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>gemm</td>
<td>S,D,C,Z</td>
<td>multiplication of 2 matrices</td>
</tr>
<tr>
<td>syrk</td>
<td>S,D,C,Z</td>
<td>symmetric rank-k update</td>
</tr>
<tr>
<td>herk</td>
<td>C,Z</td>
<td>hermitian rank-k update</td>
</tr>
<tr>
<td>syr2k</td>
<td>S,D,C,Z</td>
<td>symmetric rank-2k update</td>
</tr>
<tr>
<td>her2k</td>
<td>C,Z</td>
<td>hermitian rank-2k update</td>
</tr>
<tr>
<td>trsm</td>
<td>S,D,C,Z</td>
<td>triangular solve with multiple right-hand sides</td>
</tr>
<tr>
<td>symm</td>
<td>S,D,C,Z</td>
<td>symmetric matrix-matrix multiplication</td>
</tr>
<tr>
<td>hemm</td>
<td>C,Z</td>
<td>hermitian matrix-matrix multiplication</td>
</tr>
</tbody>
</table>
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

```c
#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
// 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
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);
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?