NVIDIA 科学计算软件

TOPICS

Programming the NVIDIA Platform

NVIDIA 加速计算 SDK

Python Ecosystem

Quantum Computing

PROGRAMMING THE NVIDIA PLATFORM

PROGRAMMING THE NVIDIA PLATFORM

CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES

ISO C++, ISO Fortran

```
std::transform(par, x, x+n, y, y,
    [=](float x, float y){ return y +
a*x; }
);
```

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```

INCREMENTAL PORTABLE OPTIMIZATION

OpenACC, OpenMP

```
#pragma acc data copy(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}
```

```
#pragma omp target data map(x,y) {
...
std::transform(par, x, x+n, y, y,
    [=](float x, float y){
        return y + a*x;
});
...
}
```

PLATFORM SPECIALIZATION

CUDA

```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

int main(void) {
  ...
  cudaMemcpy(d_x, x, ...);
  cudaMemcpy(d_y, y, ...);

  saxpy<<<(N+255)/256,256>>>(...);

  cudaMemcpy(y, d_y, ...);
```

ACCELERATION LIBRARIES

Core | Math | Communication | Data Analytics | AI | Quantum

ACCELERATED STANDARD LANGUAGES

Parallel performance for wherever your code runs

ISO C++

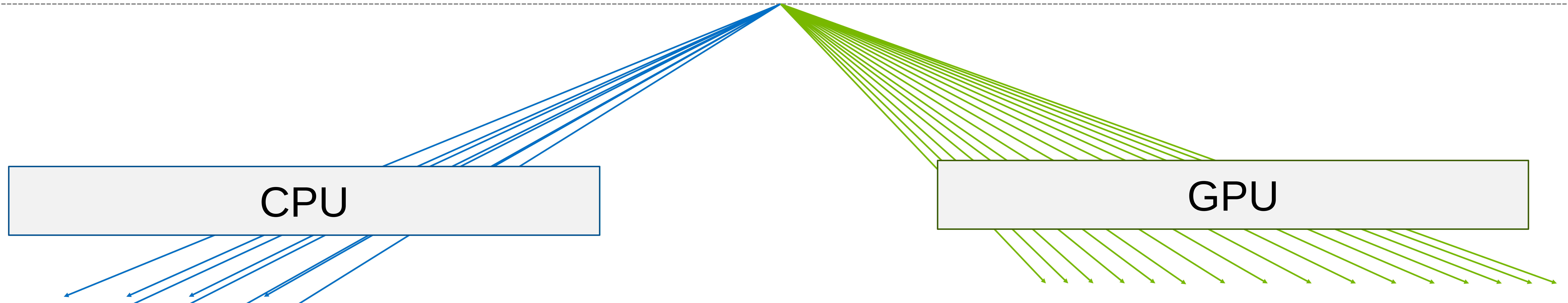
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

ISO Fortran

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

Python

```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```



CPU

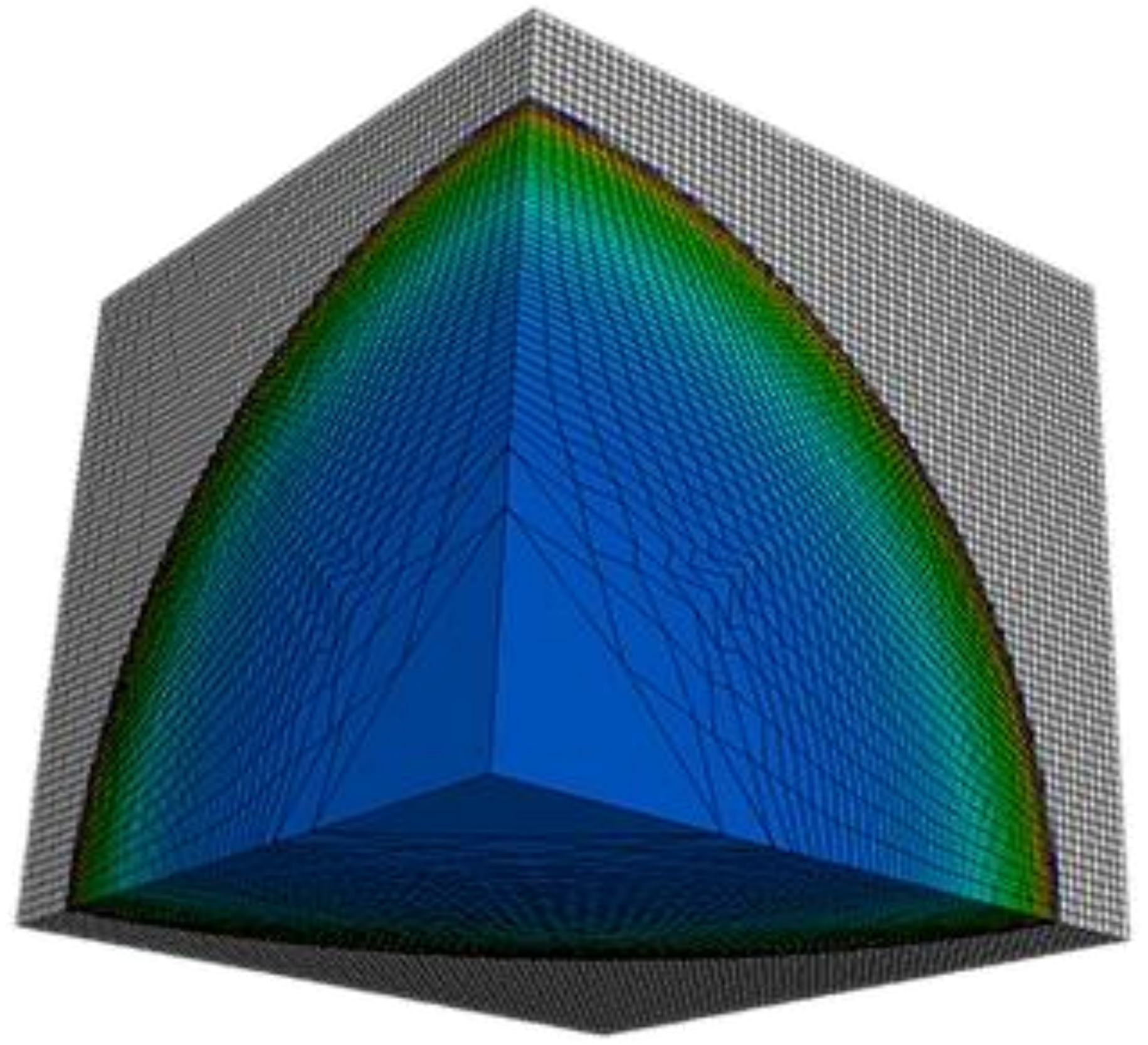
GPU

```
nvc++ -stdpar=multicore
nvfortran -stdpar=multicore
legate -cpus 16 saxpy.py
```

```
nvc++ -stdpar=gpu
nvfortran -stdpar=gpu
legate -gpus 1 saxpy.py
```

Lulesh Hydrodynamics

- Composable, compact and elegant
- Easy to read and maintain
- ISO Standard
- Portable – nvc++, g++, icpc, MSVC, …



```cpp
static inline
void CalcHydroConstraintForElems(Domain &domain, Index_t length,
    Index_t *regElemlist, Real_t dvovmax, Real_t& dthydro)
{
#if _OPENMP
  const Index_t threads = omp_get_max_threads();
  Index_t   hydro_elem_per_thread[threads];
  Real_t dthydro_per_thread[threads];
#else
  Index_t threads = 1;
  Index_t hydro_elem_per_thread[1];
  Real_t dthydro_per_thread[1];
#endif
#pragma omp parallel firstprivate(length, dvovmax)
  {
    Real_t dthydro_tmp = dthydro ;
    Index_t hydro_elem = -1 ;
#if _OPENMP
    Index_t thread_num = omp_get_thread_num();
#else
    Index_t thread_num = 0;
#endif
#pragma omp for
    for (Index_t i = 0 ; i < length ; ++i) {
      Index_t indx = regElemlist[i] ;

      if (domain.vdov(indx) != Real_t(0.)) {
        Real_t dtdvov = dvovmax / (FABS(domain.vdov(indx))+Real_t(1.e-20)) ;

        if ( dthydro_tmp > dtdvov ) {
          dthydro_tmp = dtdvov ;
          hydro_elem = indx ;
        }
      }
    }
    dthydro_per_thread[thread_num] = dthydro_tmp ;
    hydro_elem_per_thread[thread_num] = hydro_elem ;
  }
  for (Index_t i = 1; i < threads; ++i) {
    if(dthydro_per_thread[i] < dthydro_per_thread[0]) {
      dthydro_per_thread[0] = dthydro_per_thread[i];
      hydro_elem_per_thread[0] = hydro_elem_per_thread[i];
    }
  }
  if (hydro_elem_per_thread[0] != -1) {
    dthydro = dthydro_per_thread[0] ;
  }
  return ;
}
```

C++ with OpenMP

```cpp
static inline void CalcHydroConstraintForElems(Domain &domain, Index_t length,
                  Index_t *regElemlist,
                  Real_t dvovmax,
                  Real_t &dthydro)
{
  dthydro = std::transform_reduce(
    std::execution::par, counting_iterator(0), counting_iterator(length),
    dthydro, [](Real_t a, Real_t b) { return a < b ? a : b; },
    [=, &domain](Index_t i)
  {
      Index_t indx = regElemlist[i];
      if (domain.vdov(indx) == Real_t(0.0)) {
        return std::numeric_limits<Real_t>::max();
      } else {
        return dvovmax / (std::abs(domain.vdov(indx)) + Real_t(1.e-20));
      }
  });
}
```

Standard C++

https://github.com/LLNL/LULESH/tree/2.0.2-dev

C++ STANDARD PARALLELISM

Lulesh Performance

NVC++
GCC
16
14
12
10
8
6
4
2
0
1
1.03
1.53
2.08
13.57
OpenMP on 64c EPYC 7742
OpenMP on 64c EPYC 7742
Standard C++ on 64c EPYC 7742
Standard C++ on 64c EPYC 7742
Standard C++ on A100

Same ISO C++ Code

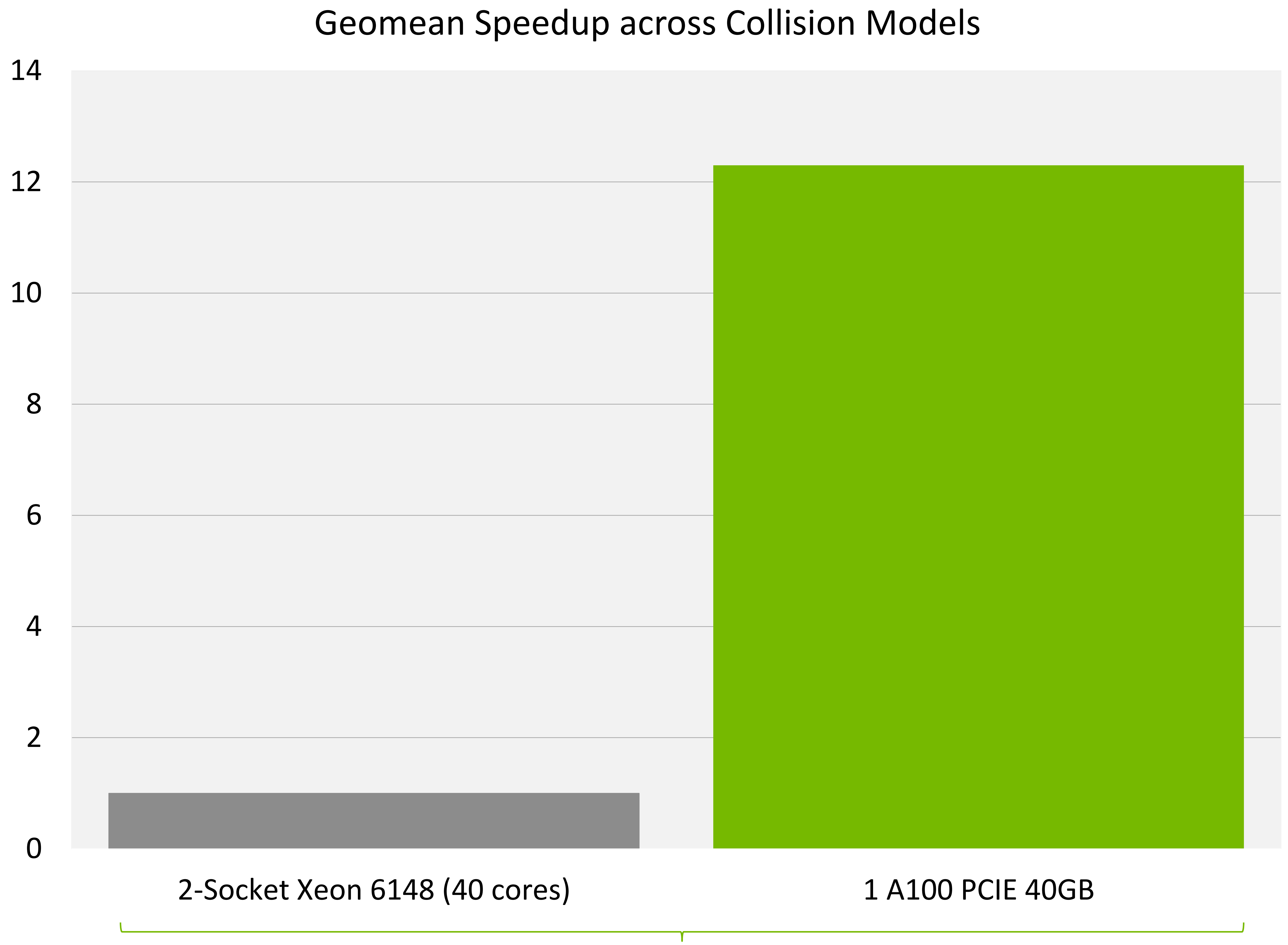
STLBM

Many-core Lattice Boltzmann with C++ Parallel Algorithms

- Framework for parallel lattice-Boltzmann simulations on multiple platforms, including many-core CPUs and GPUs
- Implemented with C++17 standard (Parallel Algorithms) to achieve parallel efficiency
- No language extensions, external libraries, vendor-specific code annotations, or pre-compilation steps

"We have with delight discovered the NVIDIA "stdpar" implementation of C++ Parallel Algorithms. … We believe that the result produces state-of-the-art performance, is highly didactical, and introduces ***a paradigm shift in cross-platform CPU/GPU programming*** *in the community.“*

-- Professor Jonas Latt, University of Geneva



Same ISO C++ Code

https://gitlab.com/unigehpfs/stlbm
https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s32076/ GTC Fall Session A31329

GAMESS

Computational Chemistry with Fortran Do Concurrent

- GAMESS is a popular Quantum Chemistry application.
- More than 40 years of development in Fortran and C
- MPI + OpenMP baseline code
- Hartree-Fock rewritten in Do Concurrent

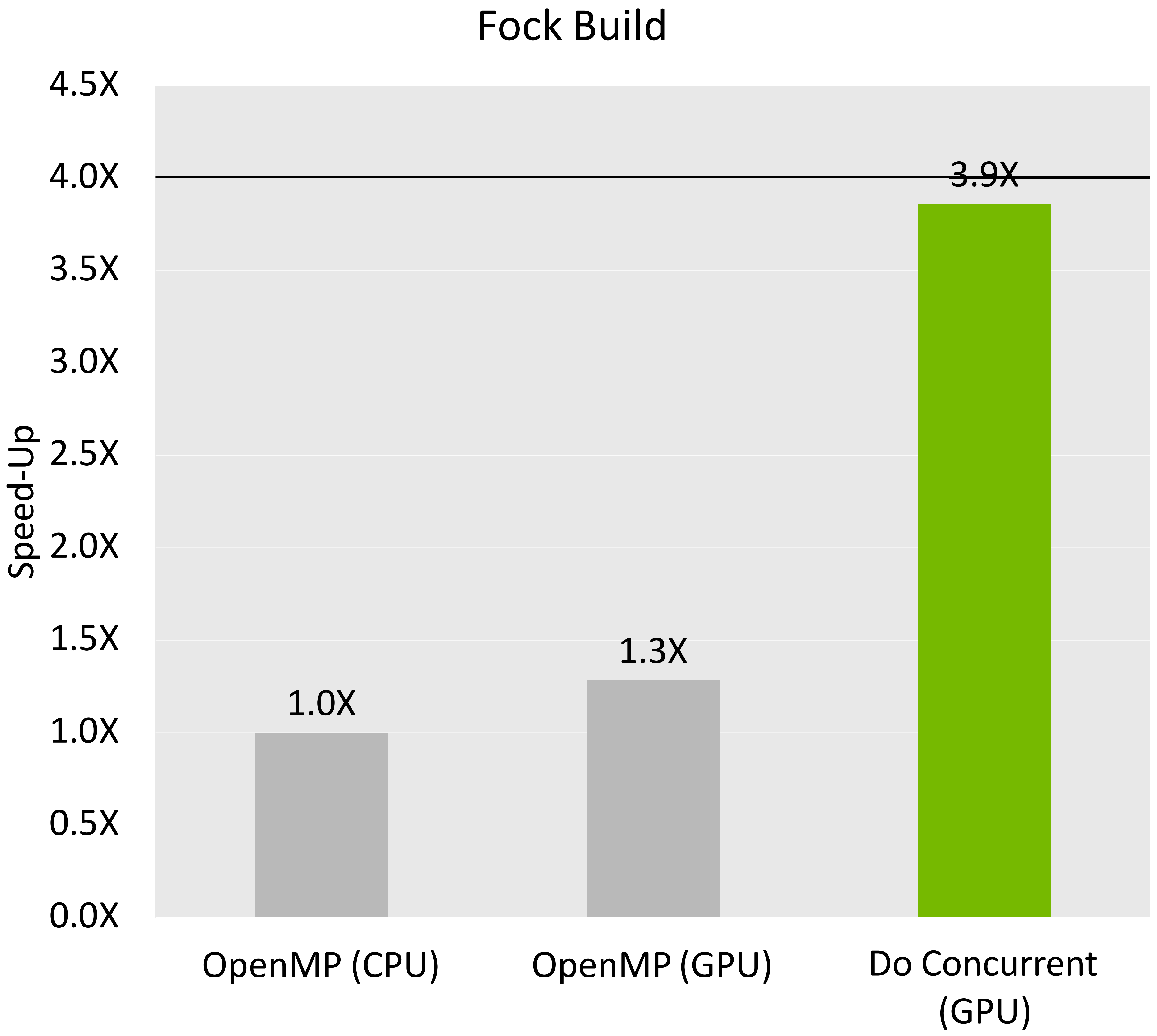
```
!pre-sorting, screening

!$omp target teams distribute &
       parallel do &
!$omp shared() private()  do
iquart = 1, ssdd_quarts
!recover shell index
ish=IDX(s_sh)  jsh=IDX(s_sh)
ksh=IDX(d_sh)  lsh=IDX(d_sh)
 !compute ints
 !digest ints  enddo
!$omp end target teams distribute &
      parallel do
```

```
!pre-sorting, screening



DO CONCURRENT (iquart=1::ssdd_quarts)&
  SHARED() LOCAL()
!recover shell index
ish=IDX(s_sh)  jsh=IDX(s_sh)
ksh=IDX(d_sh)  lsh=IDX(d_sh)
 !compute ints
 !digest ints  enddo
```



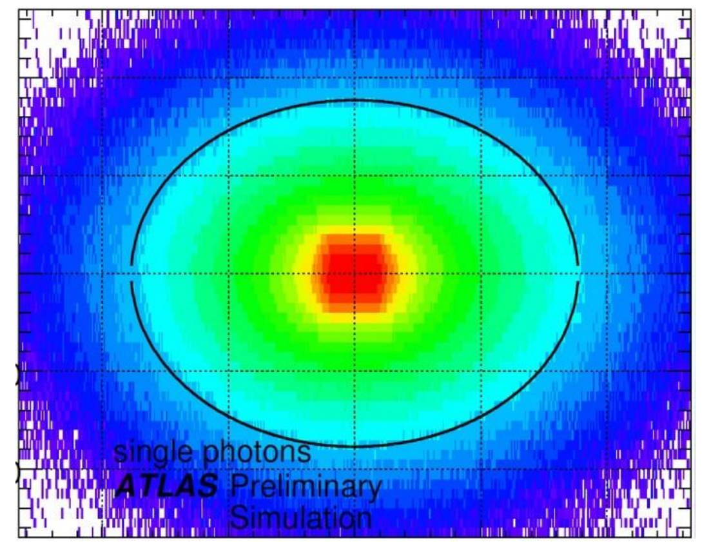
nvfortran 22.7, NVIDIA A100 GPU, AMD “Milan” CPU

* Courtesy of Melisa Alkan, Iowa State University. Not yet published.

Accelerating STDPAR Adoption

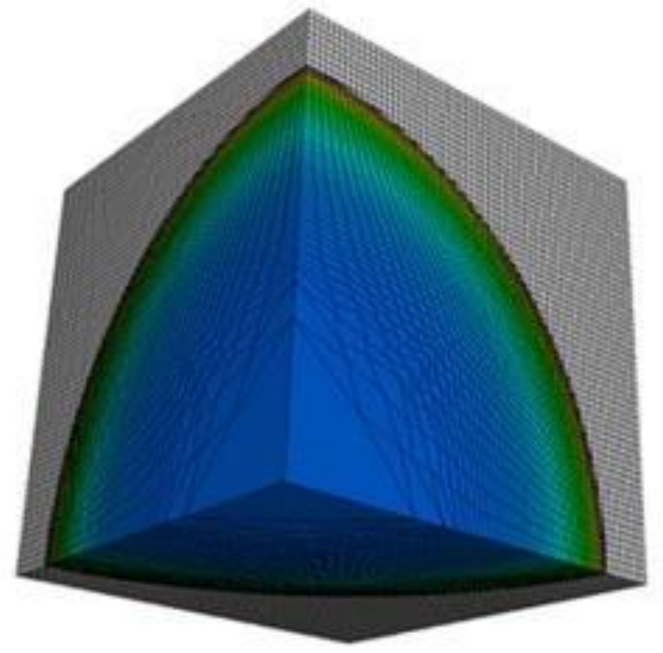
BabelStream

Cloverleaf

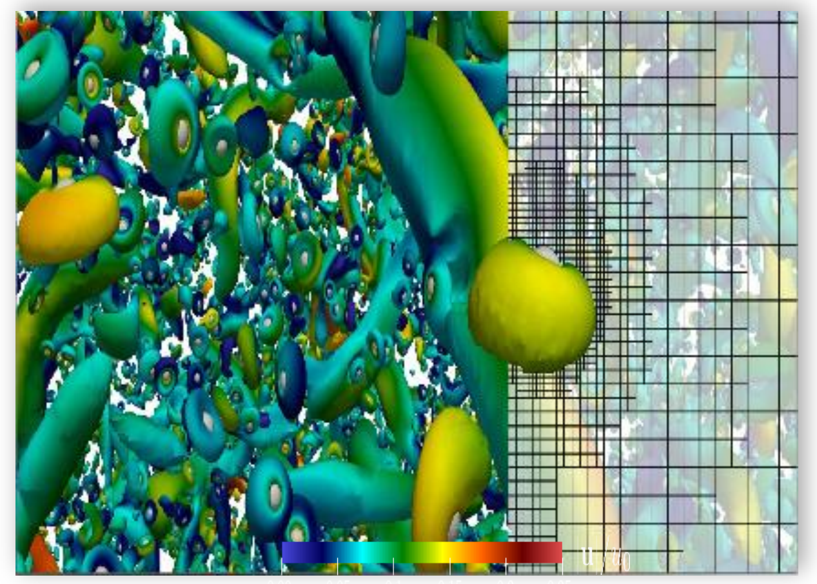


FastCaloSim

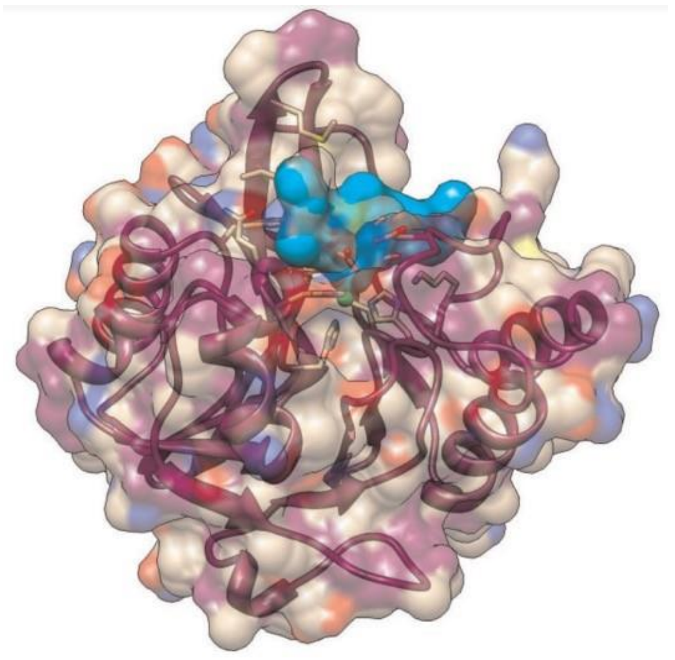
GAMESS



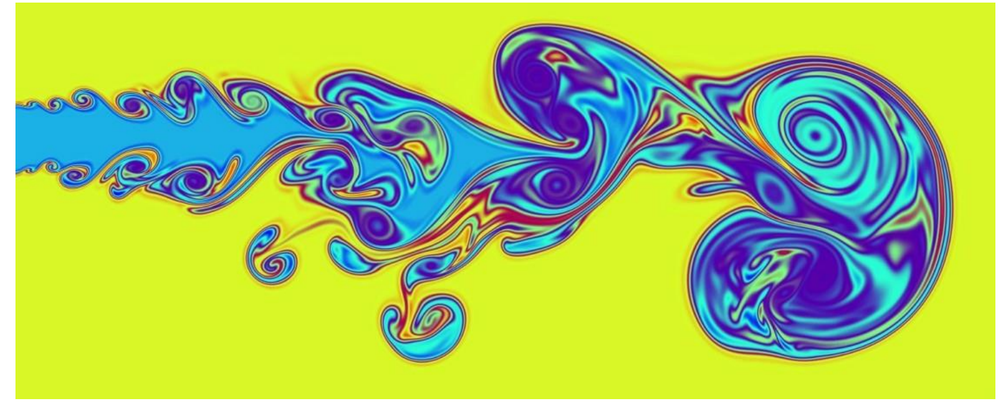
Lulesh



M-AIA



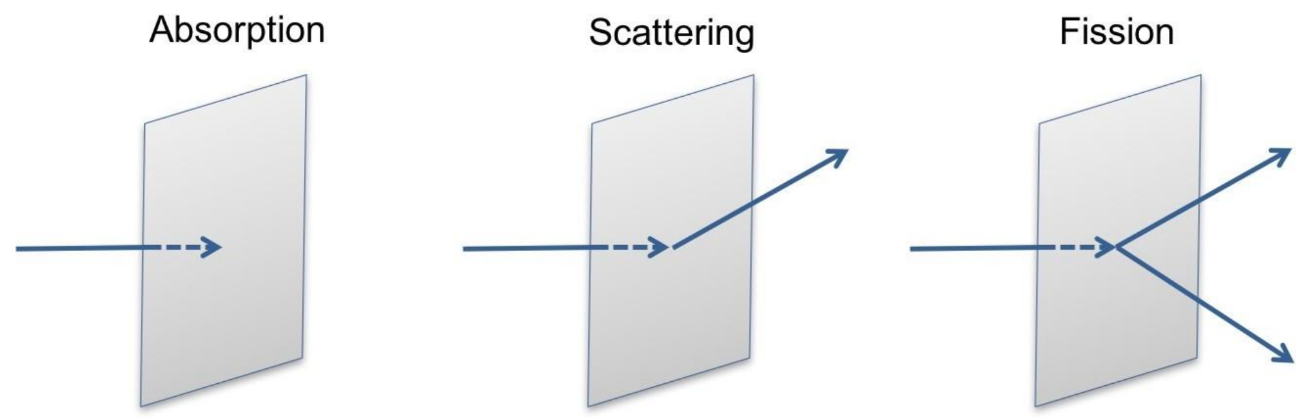
miniBUDE



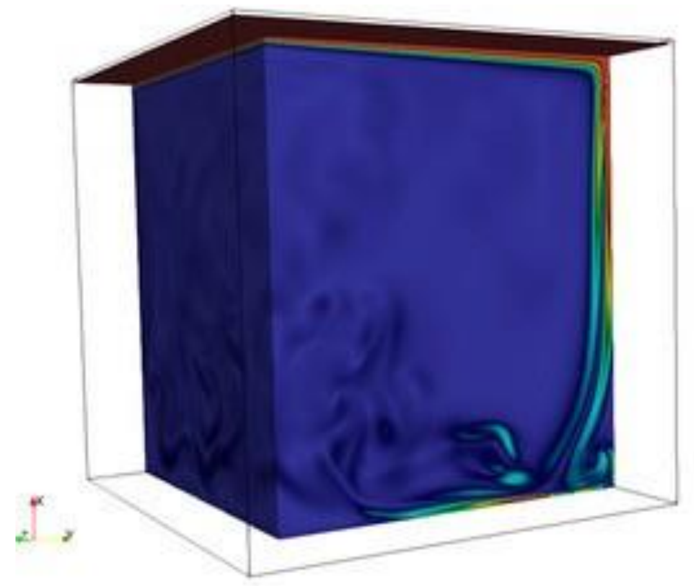
MiniWeather

Palabos

POT3D



Quicksilver



STLBM

RAJAPerf

NVIDIA 加速计算 SDK

NVIDIA HPC SDK

Available at developer.nvidia.com/hpc-sdk, on NGC, via Spack, and in the Cloud

DEVELOPMENT

Programming Models
- Standard C++ & Fortran
- OpenACC & OpenMP
- CUDA

Compilers
- nvcc
- nvc
- nvc++
- nvfortran

Core Libraries
- libcu++
- Thrust
- CUB

Math Libraries
- cuBLAS
- cuTENSOR
- cuSPARSE
- cuSOLVER
- cuFFT
- cuRAND

Communication Libraries
- HPC-X
  - MPI
  - UCX
  - SHMEM
  - SHARP
  - HCOLL
- NVSHMEM
- NCCL

ANALYSIS

Profilers
- Nsight
  - Systems
  - Compute

Debugger
- cuda-gdb
  - Host
  - Device

Develop for the NVIDIA Platform: GPU, CPU and Interconnect
Libraries | Accelerated C++ and Fortran | Directives | CUDA
7-8 Releases Per Year | Freely Available

CUBLAS

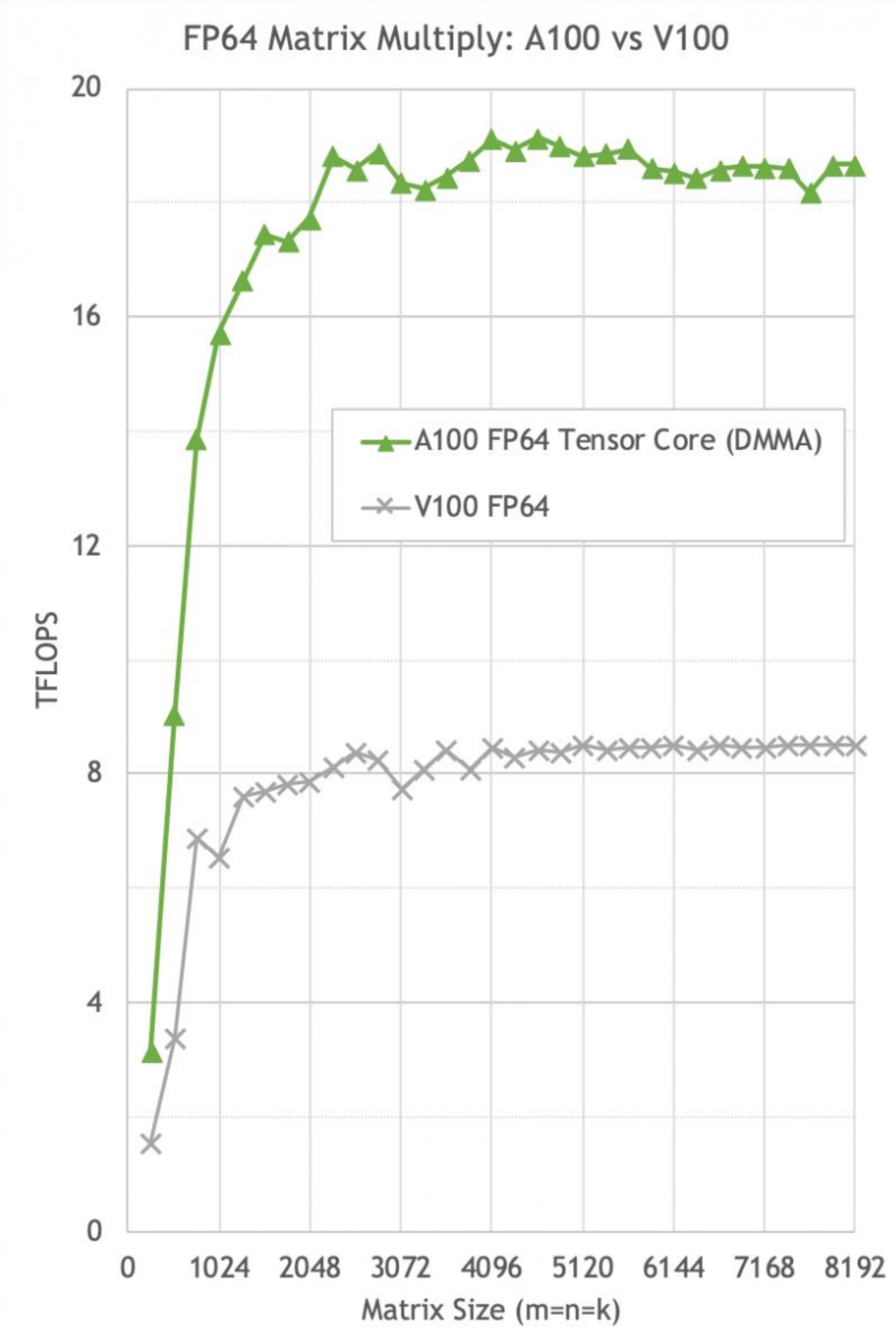
GPU Optimized BLAS Implementation

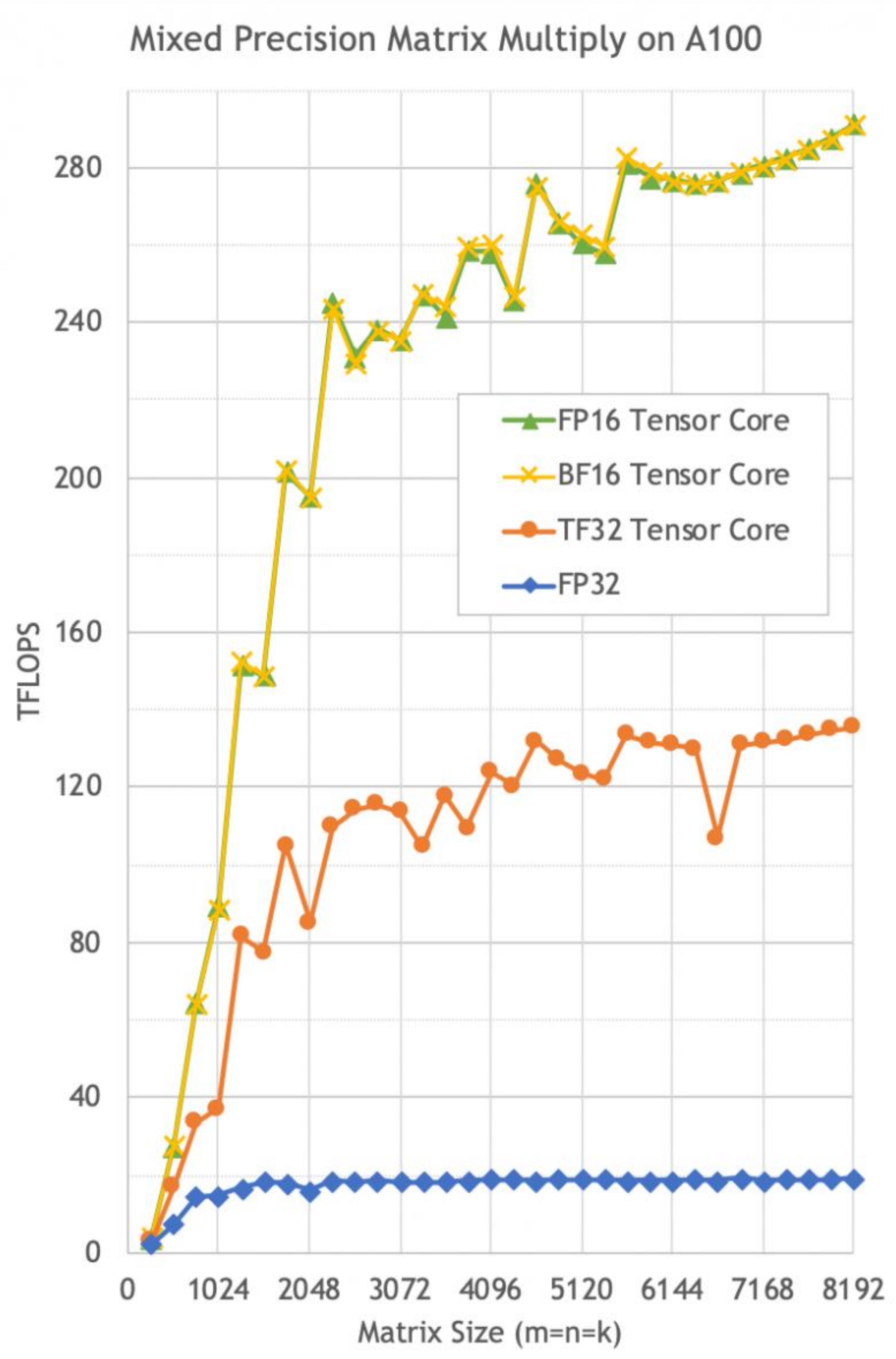
Full BLAS implementation + extensions

- Vector Vector / Matrix Vector / Matrix Matrix
- Mixed Precision / Multiple GPUs / Batched APIs

Accelerating a wide range of applications

- HPC & Scientific Computing
- Data Analytics & Deep Learning


FP64 Matrix Multiply: A100 vs V100


Mixed Precision Matrix Multiply on A100

CUSPARSE

GPU Optimized Sparse BLAS

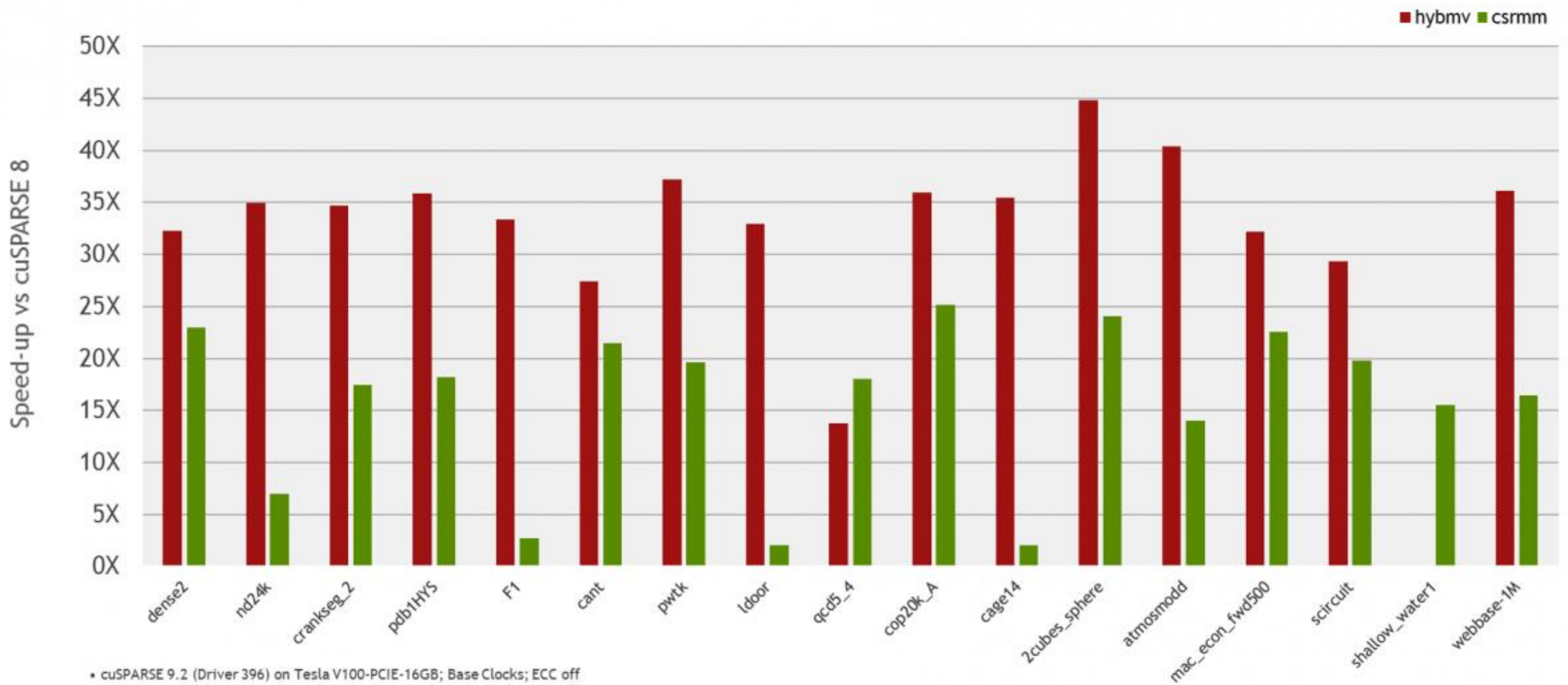
Full Sparse BLAS implementation optimized for GPU

- Sparse Vector Dense Vector
- Sparse Matrix Dense Vector
- Sparse Matrix Dense Matrix
- Sparse Matrix Sparse Matrix

Accelerating a wide range of applications

- HPC & Scientific Computing
- Data Analytics & Deep Learning

cuSPARSE 9.2 UP TO 45x FASTER THAN cuSPARSE 8.0



- cuSPARSE 9.2 (Driver 396) on Tesla V100-PCIE-16GB; Base Clocks; ECC off
- cuSPARSE 8.0 (Driver 375) on Tesla P100-PCIE-16GB; Base Clocks; ECC off
- Host System: Intel Broadwell E5-2690 v4@2.60GHz 3.5GHz Turbo (Broadwell) HT On; 88 threads;
- Ubuntu 16.04.6 and 256GB DDR4 memory; System memory speed: 2133 MHz;
- Input and output data on device; Average speedup across different data types
- Matrices obtained from http://www.cise.ufl.edu/research/sparse/matrices

CUSOLVER

GPU-accelerated dense and sparse direct solvers

Dense and Sparse Factorizations & Solvers

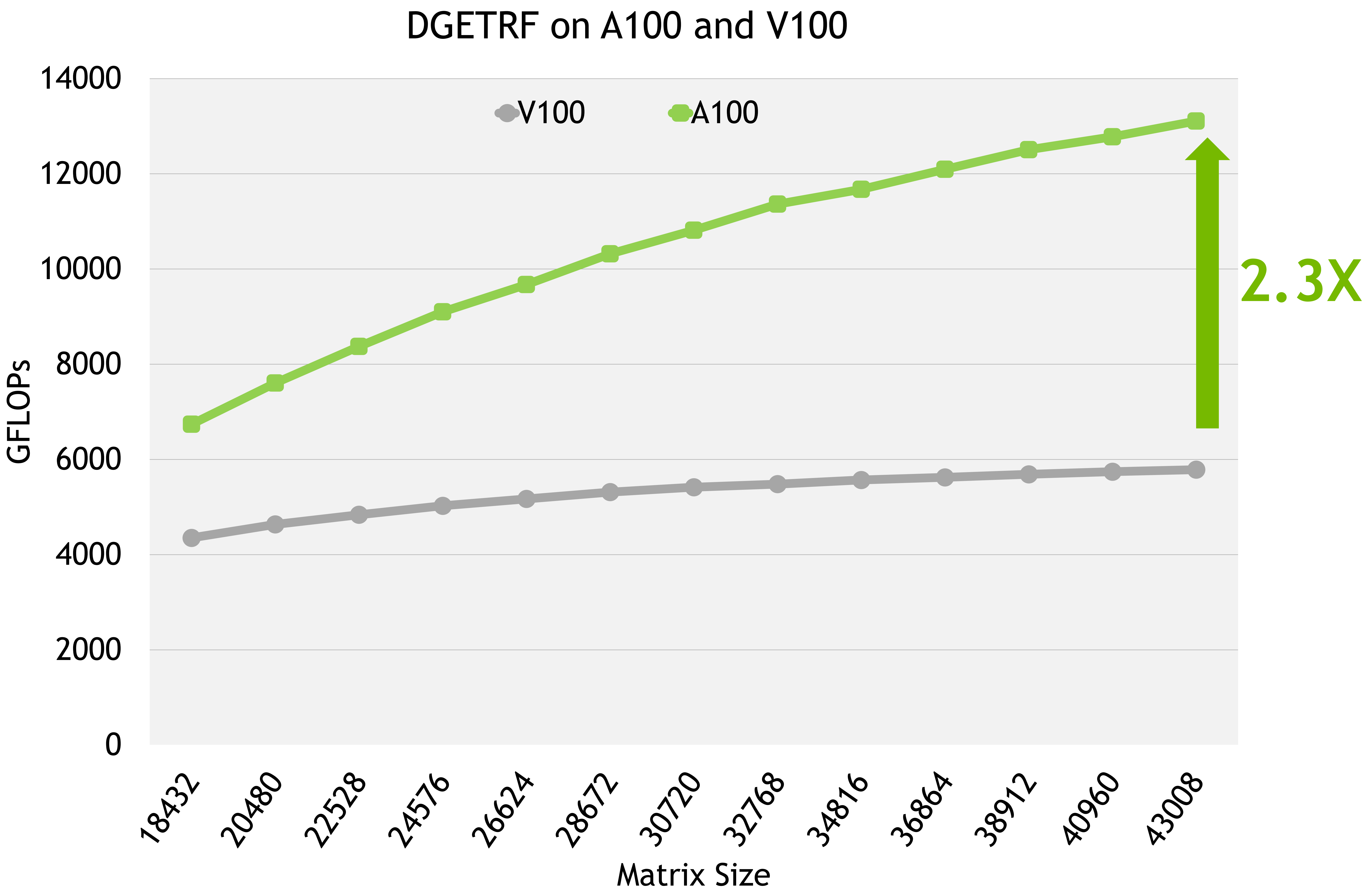
- LU, Cholesky, QR
- Symmetric and Generalized Eigensolvers
- Tensor Core Accelerated Iterative Refinement Solvers
- Multi GPU Support

Accelerating a wide range of applications

- HPC & Scientific Computing
- Data Analytics

Features

- Automatic DMMA acceleration for factorizations and linear solvers
- A100 vs V100
  - Up to 2.3X Speedup



A100 data collected with pre-production hardware and software

CUTENSOR

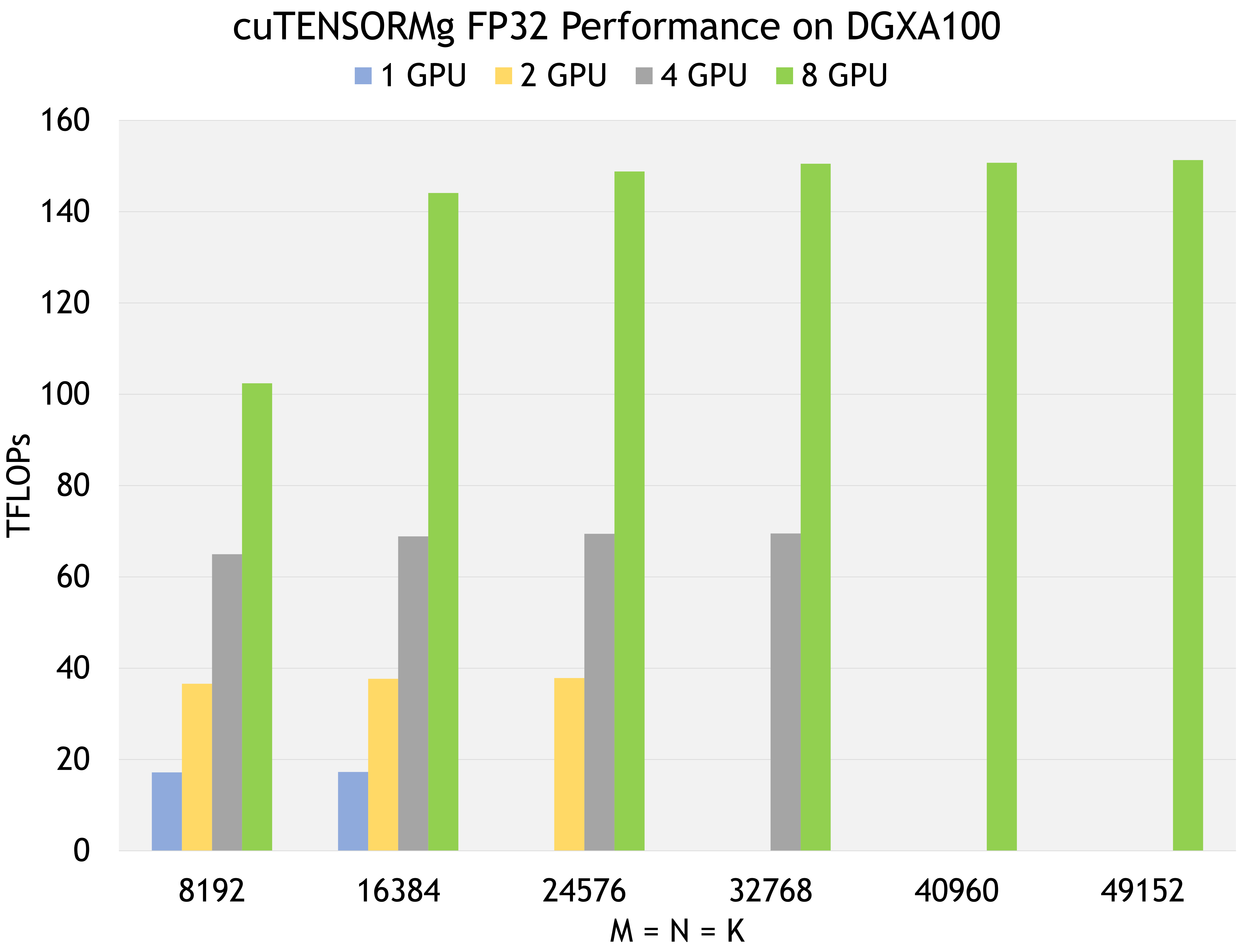
A New High Performance CUDA Library for Tensor Primitives

Available at developer.nvidia.com/cutensor & in HPC SDK

- Tensor Contractions & Reductions
- Elementwise Operations & Elementwise Fusion
- Mixed Precision Support & Tensor Core Acceleration
- Multi-GPU Tensor Contractions

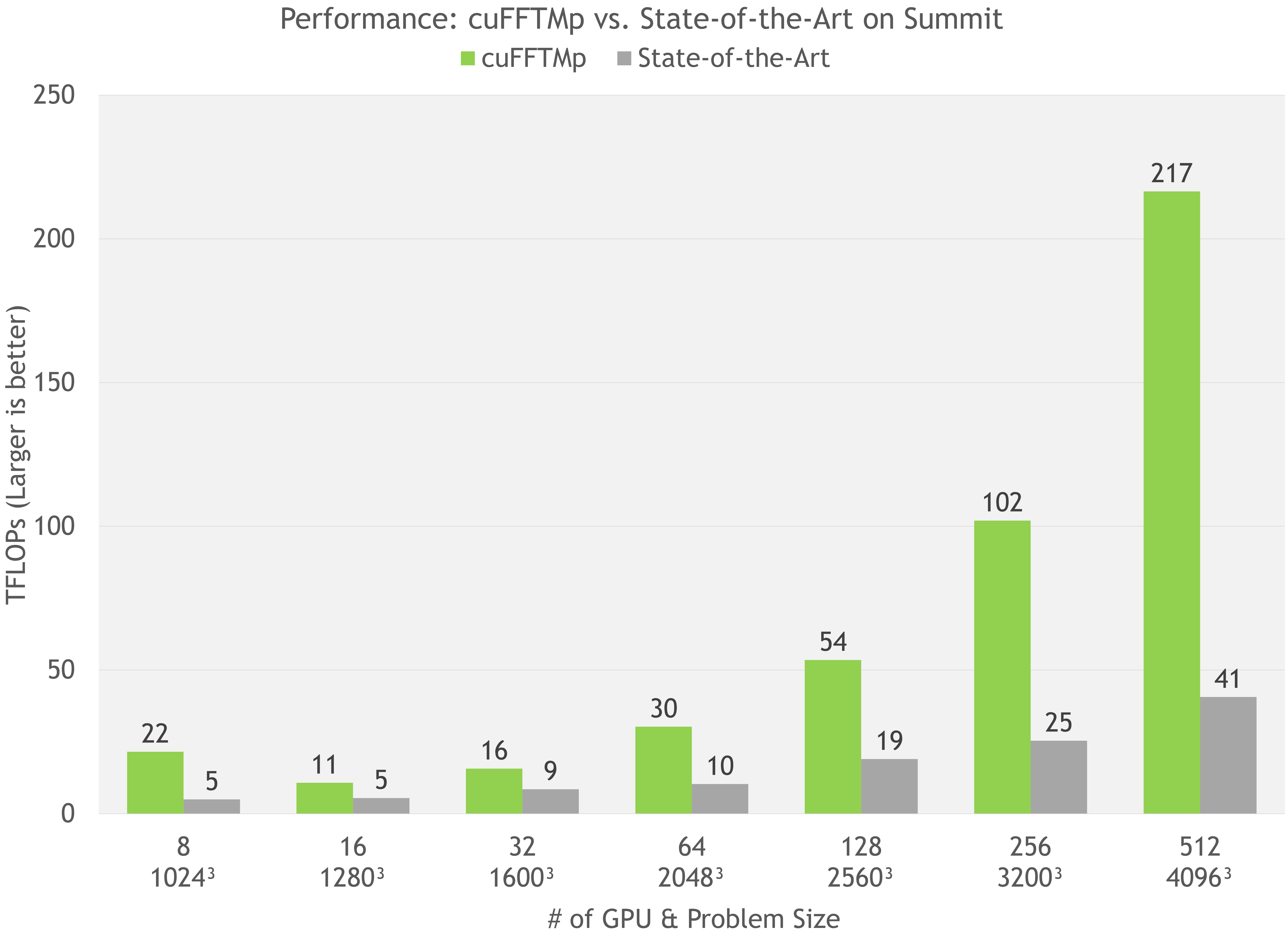
Impact

- DL frameworks aggressively adopting elementwise operations
- Up to 23X application end-to-end speedup over previously CPU-only Quantum Chemistry simulations from drop-in contraction API



CUFFT

GPU-accelerated library for Fast Fourier Transforms



cuFFTMp

A distributed-memory multi-node and multiGPU solution for solving FFTs at scale.

EA release available in Fall ’21
https://developer.nvidia.com/cudamathlibraryea

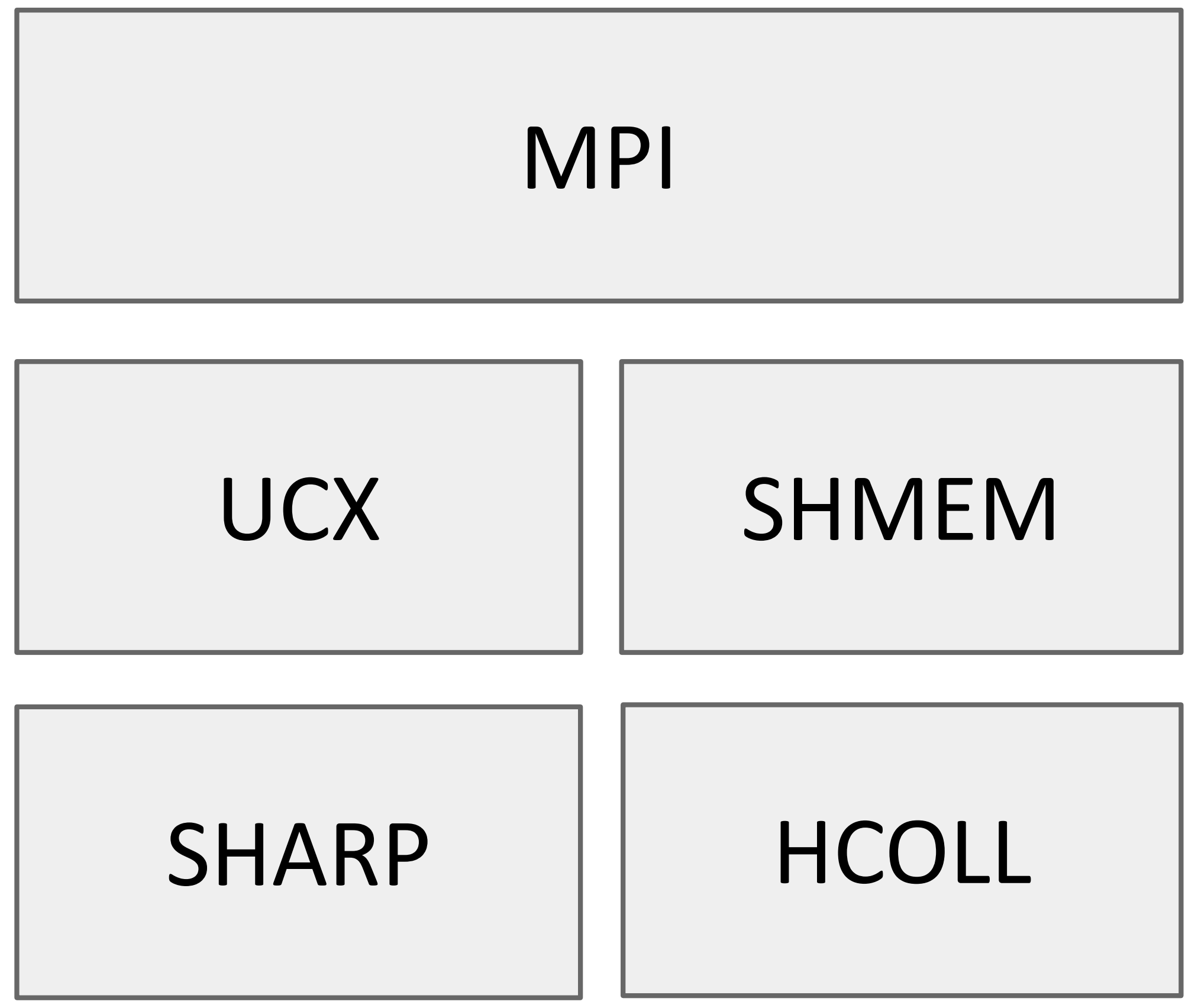
Initial release to 2D & 3D with Slab composition

HPC-X

Accelerated Communications APIs and Operations for HPC

HPC-X is a comprehensive software package that includes Accelerated Communications API and Operations for HPC

- CUDA-aware MPI library based on Open MPI with support for GPUDirect™
- Fully compatible with CUDA C++, CUDA Fortran and the NVIDIA HPC Compilers
- Optimized for NVIDIA InfiniBand interconnect solutions
- Integrated into the NVIDIA HPC SDK



MPI

UCX SHMEM

SHARP HCOLL

Components	Description
MPI/SHMEM	•Open MPI and OpenSHMEM •MPI profiler (IPM - open source tool from *http://ipm-hpc.org* /) •MPI tests (OSU, IMB, random ring, etc.)
HPC Acceleration Package	•HCOLL •UCX •Scalable Hierarchical Aggregation and Reduction Protocol (SHARP) •nccl-rdma-sharp-plugin

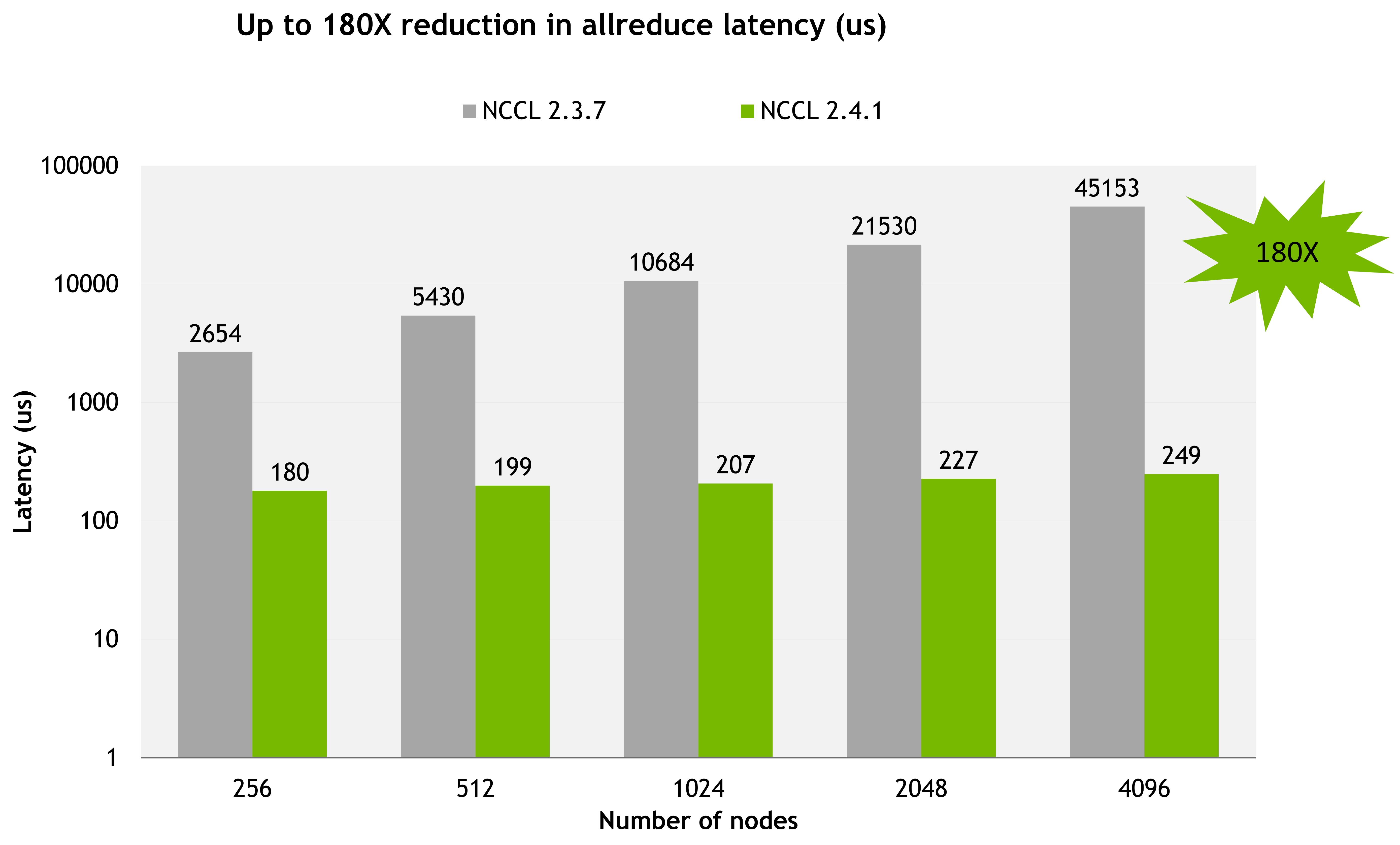
NCCL

NVIDIA Collective Communication Library

- Multi-GPU and Multi-Node Collectives Optimized for NVIDIA GPUs
- Automatic Topology Detection
- Easy to integrate | MPI Compatible
- Minimize latency | Maximize bandwidth

Impact

- Accelerates leading deep learning frameworks
- Adoption in HPC accelerating



AllReduce 8-byte (float) latency, Summit Supercomputer at Oak Ridge National Lab
4096 nodes, 6xV100, 2x IB EDR

NSIGHT SYSTEMS

System profiler

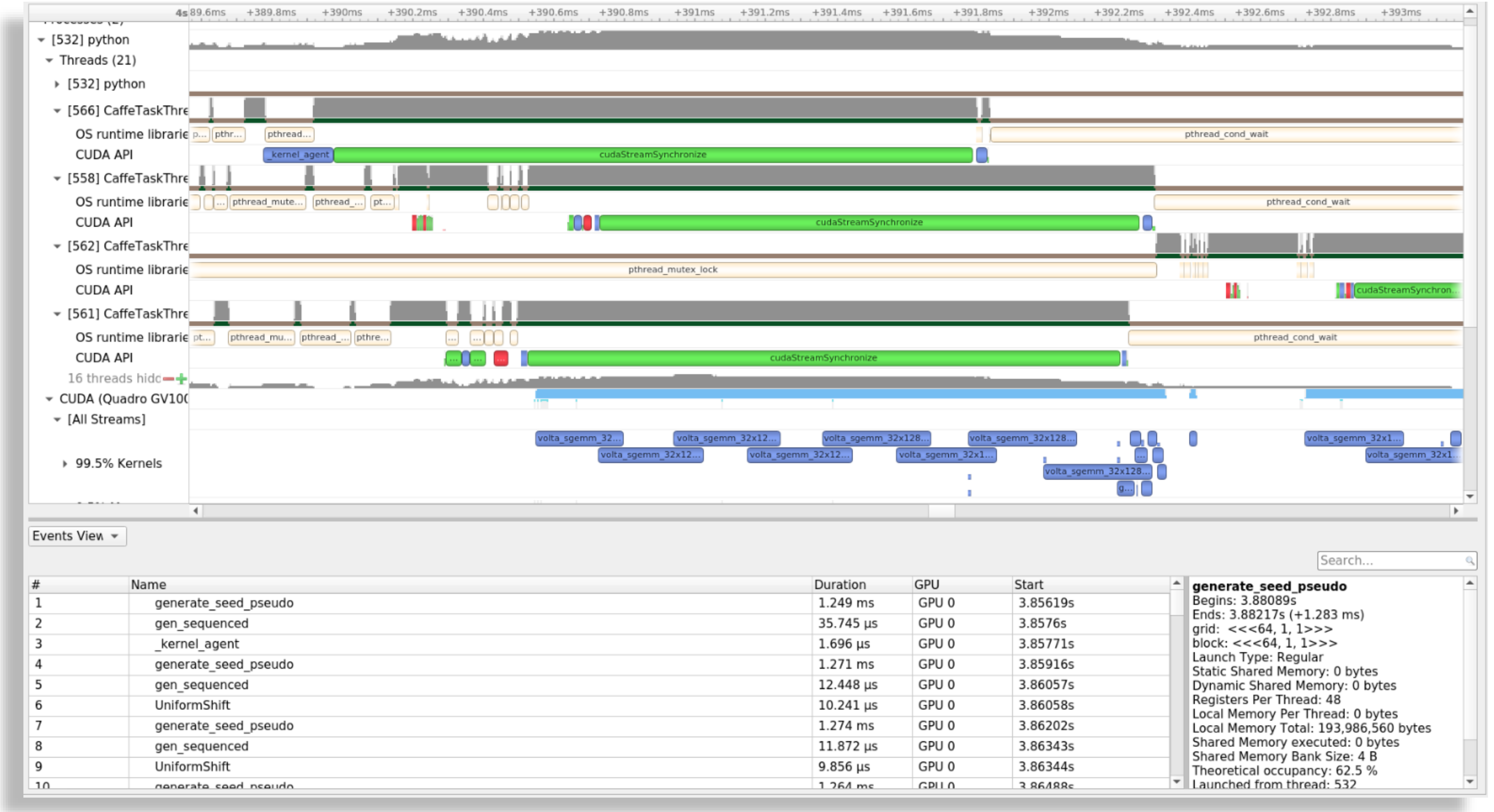
Key Features:

- System-wide application algorithm tuning
  - Multi-process tree support
- Locate optimization opportunities
  - Visualize millions of events on a very fast GUI timeline
  - Or gaps of unused CPU and GPU time
- Balance your workload across multiple CPUs and GPUs
  - CPU algorithms, utilization and thread state
    GPU streams, kernels, memory transfers, etc
- Command Line, Standalone, IDE Integration

OS: Linux (x86, **Power**, Arm SBSA, Tegra), Windows, MacOSX (host)

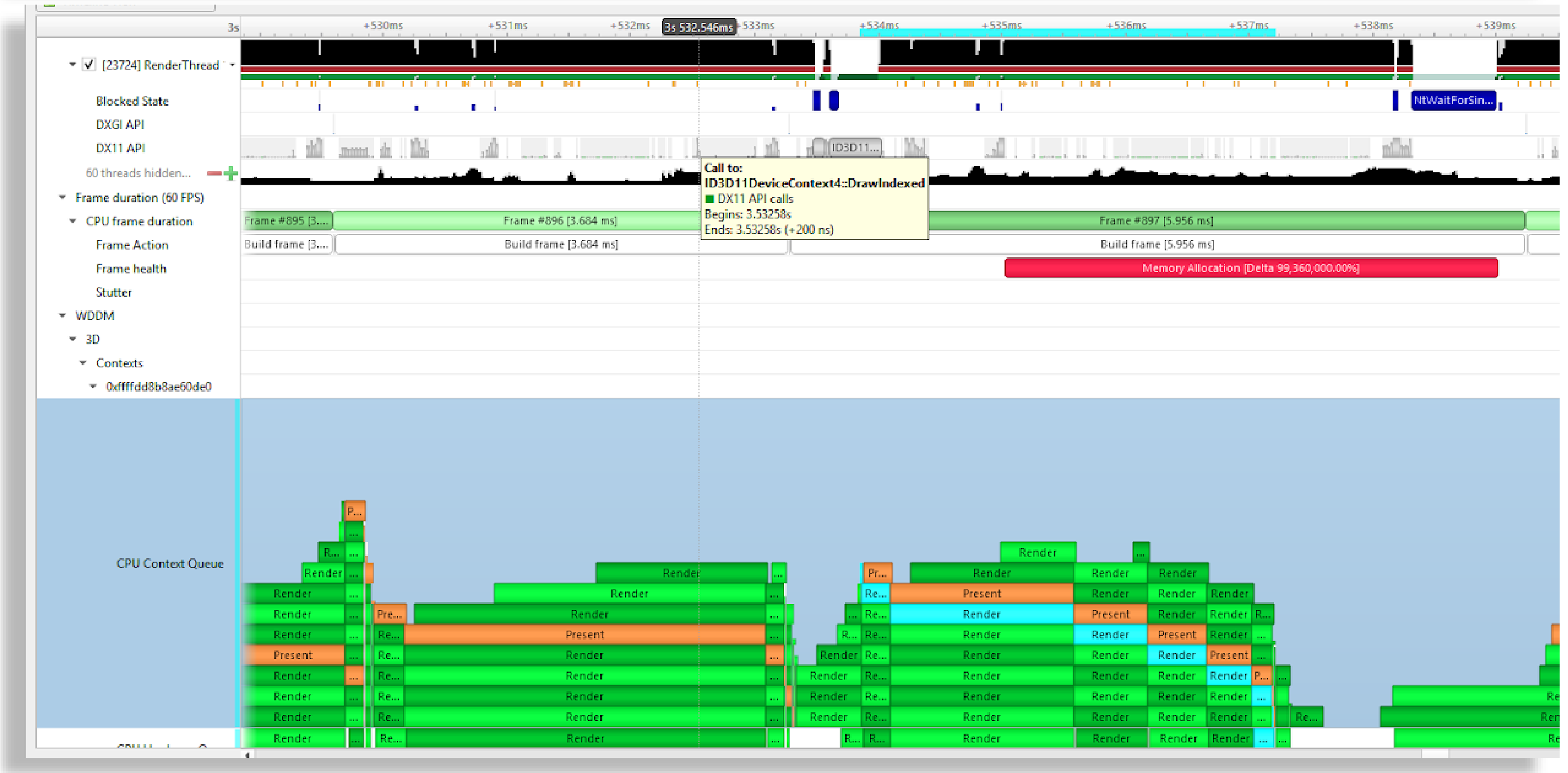
GPUs: Pascal+

Docs/product: https://developer.nvidia.com/nsight-systems

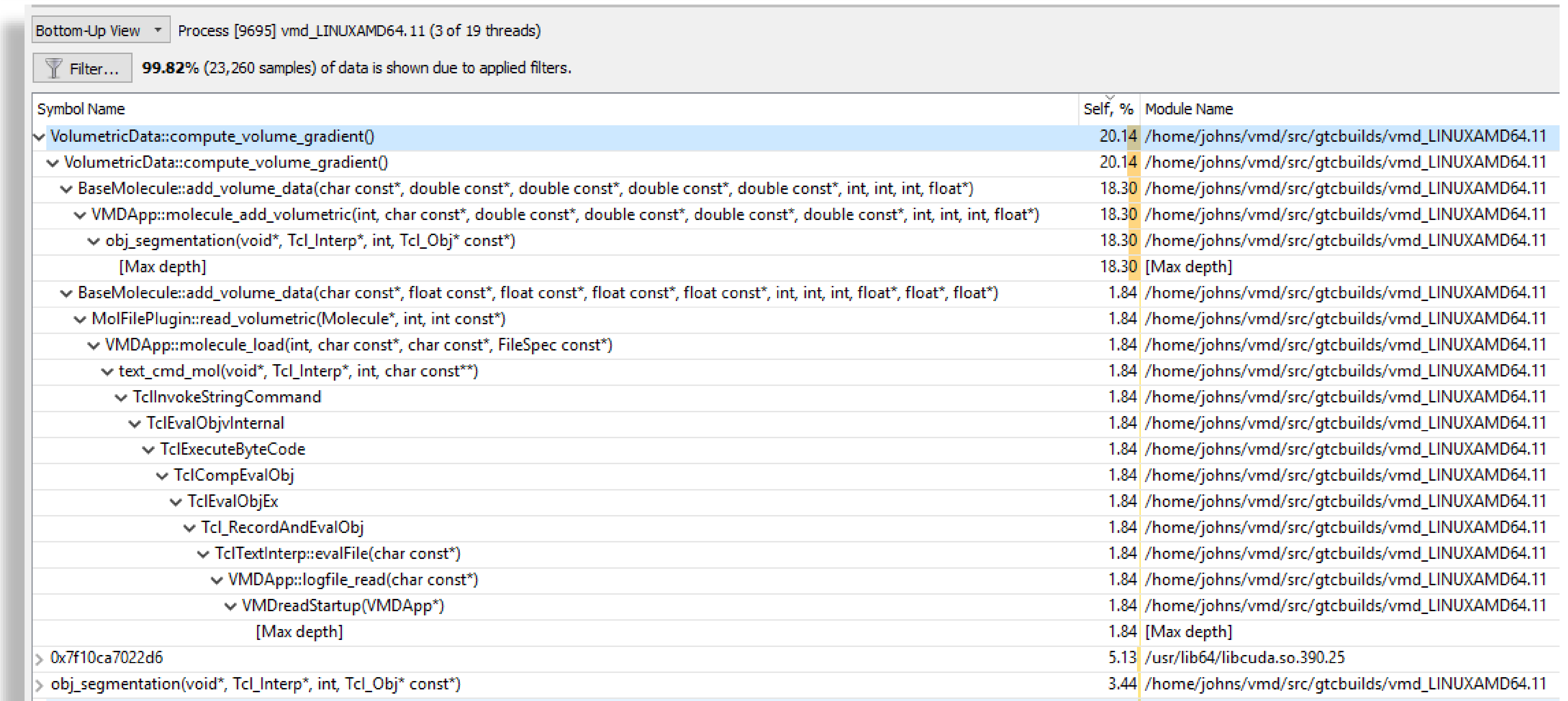


#	Name	Duration	GPU	Start
1	generate_seed_pseudo	1.249 ms	GPU 0	3.85619s
2	gen_sequenced	35.745 µs	GPU 0	3.8576s
3	_kernel_agent	1.696 µs	GPU 0	3.85771s
4	generate_seed_pseudo	1.271 ms	GPU 0	3.85916s
5	gen_sequenced	12.448 µs	GPU 0	3.86057s
6	UniformShift	10.241 µs	GPU 0	3.86058s
7	generate_seed_pseudo	1.274 ms	GPU 0	3.86202s
8	gen_sequenced	11.872 µs	GPU 0	3.86343s
9	UniformShift	9.856 µs	GPU 0	3.86344s
10	generate_seed_pseudo	1.264 ms	GPU 0	3.86488s

generate_seed_pseudo
Begins: 3.88089s
Ends: 3.88217s (+1.283 ms)
grid: <<<64, 1, 1>>>
block: <<<64, 1, 1>>>
Launch Type: Regular
Static Shared Memory: 0 bytes
Dynamic Shared Memory: 0 bytes
Registers Per Thread: 48
Local Memory Per Thread: 0 bytes
Local Memory Total: 193,986,560 bytes
Shared Memory executed: 0 bytes
Shared Memory Bank Size: 4 B
Theoretical occupancy: 62.5 %
Launched from thread: 532



Call to:
ID3D11DeviceContext4::DrawIndexed
DX11 API calls
Begins: 3.53258s
Ends: 3.53258s (+200 ns)



Bottom-Up View | Process [9695] vmd_LINUXAMD64.11 (3 of 19 threads)

Filter... **99.82%** (23,260 samples) of data is shown due to applied filters.

Symbol Name	Self, %	Module Name
VolumetricData::compute_volume_gradient()	20.14	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VolumetricData::compute_volume_gradient()	20.14	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
BaseMolecule::add_volume_data(char const*, double const*, double const*, double const*, double const*, int, int, int, float*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::molecule_add_volumetric(int, char const*, double const*, double const*, double const*, double const*, int, int, int, float*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
obj_segmentation(void*, Tcl_Interp*, int, Tcl_Obj* const*)	18.30	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
[Max depth]	18.30	[Max depth]
BaseMolecule::add_volume_data(char const*, float const*, float const*, float const*, float const*, int, int, int, float*, float*, float*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
MolFilePlugin::read_volumetric(Molecule*, int, int const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::molecule_load(int, char const*, char const*, FileSpec const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
text_cmd_mol(void*, Tcl_Interp*, int, char const**)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclInvokeStringCommand	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclEvalObjvInternal	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclExecuteByteCode	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclCompEvalObj	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclEvalObjEx	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
Tcl_RecordAndEvalObj	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
TclTextInterp::evalFile(char const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDApp::logfile_read(char const*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
VMDreadStartup(VMDApp*)	1.84	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11
[Max depth]	1.84	[Max depth]
0x7f10ca7022d6	5.13	/usr/lib64/libcuda.so.390.25
obj_segmentation(void*, Tcl_Interp*, int, Tcl_Obj* const*)	3.44	/home/johns/vmd/src/gtcbuilds/vmd_LINUXAMD64.11

GPU WORKLOAD

See CUDA workloads execution time

Locate idle GPU times

7s
+956.2ms
+956.4ms
+956.6ms
+956.8ms
+957ms
+957.2ms
+957.4ms
+957.6ms
+957.8ms
+958ms
+958.2ms
+958.4
CUDA (Tesla V100-SXM3-32GB, 0000:34:00.0)
Default stream (7)
Kernels
W...
C...
wgrad_alg0_engine
calc_bias_diff
elementwise_kernel
volta_scudnn_128x32_relu_interior_nn_v1
volta_scudnn_128x64_relu_interior_nn_v1
29 kernel groups hidden...
Memory
Memset
HtoD memcpy
DtoH memcpy

GPU WORKLOAD

See CUDA workloads execution time

Locate idle GPU times

7s
+956.2ms
+956.4ms
+956.6ms
+956.8ms
+957ms
+957.2ms
+957.4ms
+957.6ms
+957.8ms
+958ms
+958.2ms
+958.4
CUDA (Tesla V100-SXM3-32GB, 0000:34:00.0)
Default stream (7)
Kernels
W...
C...
wgrad_alg0_engine
calc_bias_diff
elementwise_kernel
volta_scudnn_128x32_relu_interior_nn_v1
volta_scudnn_128x64_relu_interior_nn_v1
29 kernel groups hidden...
Memory
Memset
HtoD memcpy
DtoH memcpy

GPU WORKLOAD

See CUDA workloads execution time

Locate idle GPU times

7s
+956.2ms
+956.4ms
+956.6ms
+956.8ms
+957ms
+957.2ms
+957.4ms
+957.6ms
+957.8ms
+958ms
+958.2ms
+958.4
CUDA (Tesla V100-SXM3-32GB, 0000:34:00.0)
Default stream (7)
Kernels
W...
C...
wgrad_alg0_engine
calc_bias_diff
elementwise_kernel
volta_scudnn_128x32_relu_interior_nn_v1
volta_scudnn_128x64_relu_interior_nn_v1
29 kernel groups hidden...
Memory
Memset
HtoD memcpy
DtoH memcpy

GPU WORKLOAD

See CUDA workloads execution time

Locate idle GPU times

7s
+956.2ms
+956.4ms
+956.6ms
+956.8ms
+957ms
+957.2ms
+957.4ms
+957.6ms
+957.8ms
+958ms
+958.2ms
+958.4
CUDA (Tesla V100-SXM3-32GB, 0000:34:00.0)
Default stream (7)
Kernels
W...
C...
wgrad_alg0_engine
calc_bias_diff
elementwise_kernel
volta_scudnn_128x32_relu_interior_nn_v1
volta_scudnn_128x64_relu_interior_nn_v1
29 kernel groups hidden...
Memory
Memset
HtoD memcpy
DtoH memcpy

NSIGHT COMPUTE

Kernel Profiling Tool

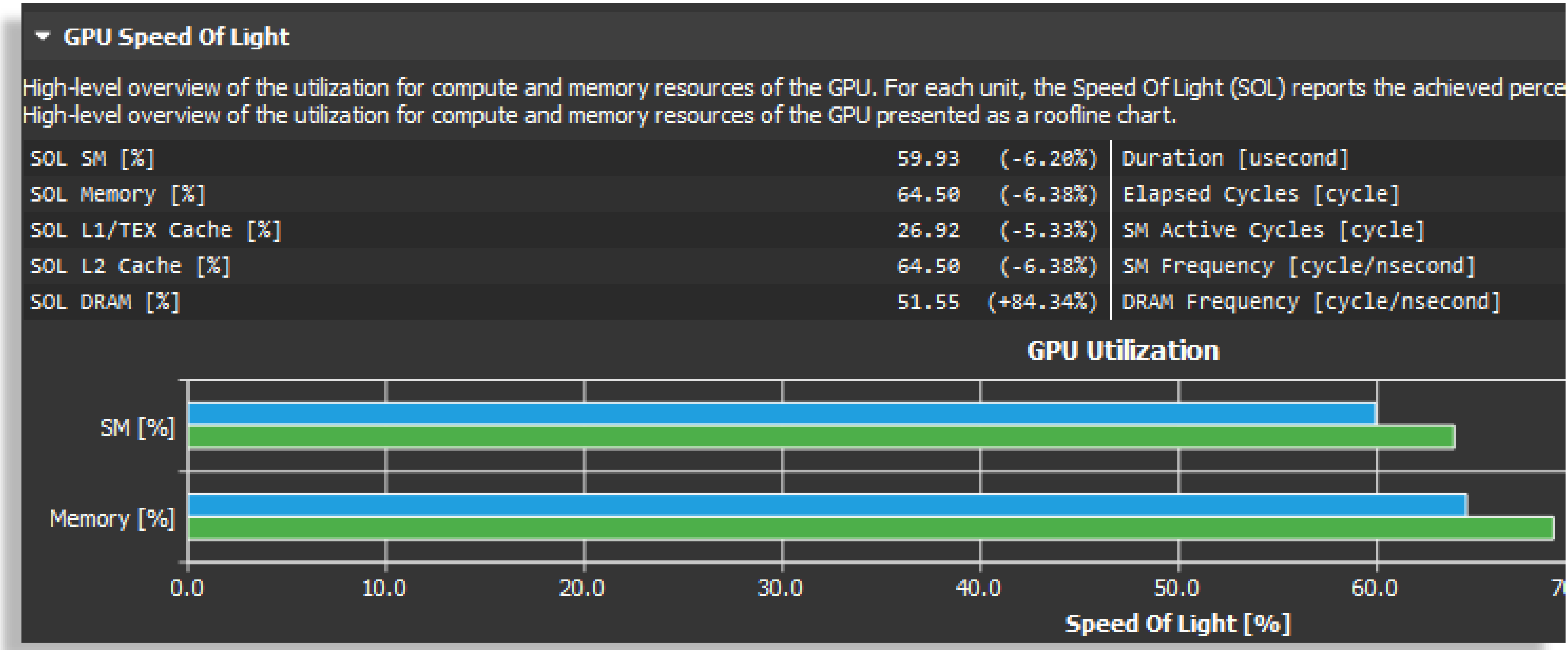
Key Features:

- Interactive CUDA API debugging and kernel profiling
- Fast Data Collection
- Compare performance metrics across different runs
- Fully Customizable (Programmable UI/Guided Analysis)
- Command Line, Standalone, IDE Integration

OS: Linux (x86, Power, Tegra, Arm SBSA), Windows, MacOSX (host only)

GPUs: Volta, Turing, A100 GPUs

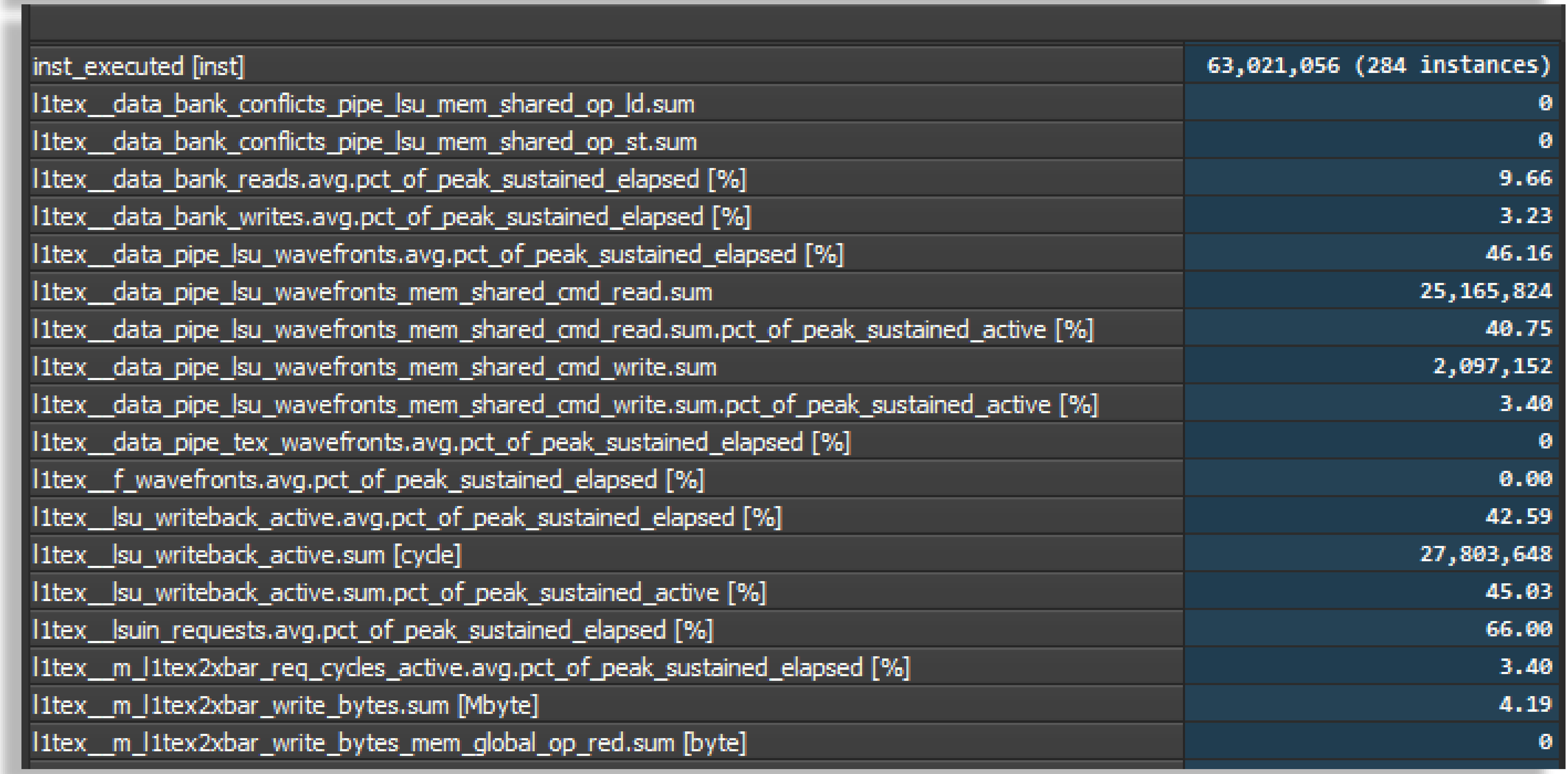
Docs/product: https://developer.nvidia.com/nsight-compute



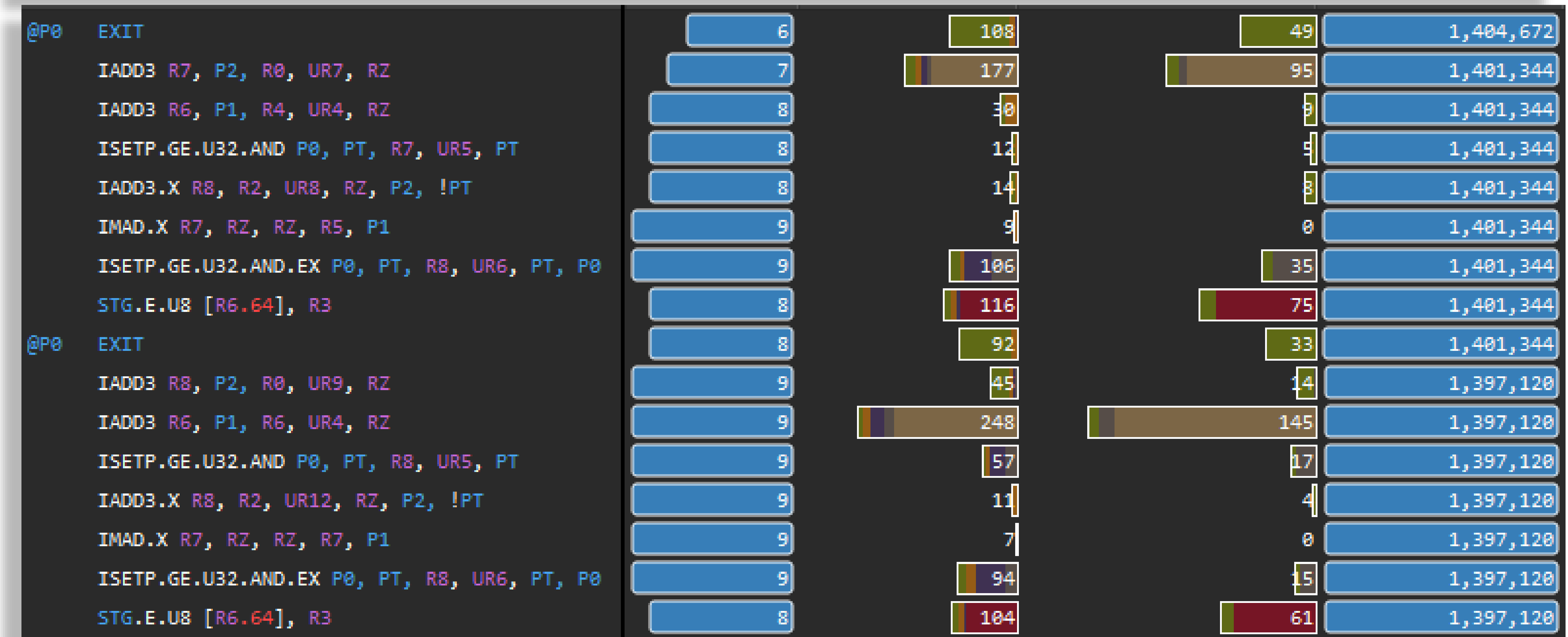
▾ GPU Speed Of Light

High-level overview of the utilization for compute and memory resources of the GPU. For each unit, the Speed Of Light (SOL) reports the achieved perce
High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

```
SOL SM [%]                 59.93   (-6.20%) | Duration [usecond]
SOL Memory [%]             64.50   (-6.38%) | Elapsed Cycles [cycle]
SOL L1/TEX Cache [%]       26.92   (-5.33%) | SM Active Cycles [cycle]
SOL L2 Cache [%]           64.50   (-6.38%) | SM Frequency [cycle/nsecond]
SOL DRAM [%]               51.55  (+84.34%) | DRAM Frequency [cycle/nsecond]
```



inst_executed [inst]	63,021,056 (284 instances)
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_ld.sum	0
l1tex__data_bank_conflicts_pipe_lsu_mem_shared_op_st.sum	0
l1tex__data_bank_reads.avg.pct_of_peak_sustained_elapsed [%]	9.66
l1tex__data_bank_writes.avg.pct_of_peak_sustained_elapsed [%]	3.23
l1tex__data_pipe_lsu_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	46.16
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_read.sum	25,165,824
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_read.sum.pct_of_peak_sustained_active [%]	40.75
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_write.sum	2,097,152
l1tex__data_pipe_lsu_wavefronts_mem_shared_cmd_write.sum.pct_of_peak_sustained_active [%]	3.40
l1tex__data_pipe_tex_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	0
l1tex__f_wavefronts.avg.pct_of_peak_sustained_elapsed [%]	0.00
l1tex__lsu_writeback_active.avg.pct_of_peak_sustained_elapsed [%]	42.59
l1tex__lsu_writeback_active.sum [cycle]	27,803,648
l1tex__lsu_writeback_active.sum.pct_of_peak_sustained_active [%]	45.03
l1tex__lsuin_requests.avg.pct_of_peak_sustained_elapsed [%]	66.00
l1tex__m_l1tex2xbar_req_cycles_active.avg.pct_of_peak_sustained_elapsed [%]	3.40
l1tex__m_l1tex2xbar_write_bytes.sum [Mbyte]	4.19
l1tex__m_l1tex2xbar_write_bytes_mem_global_op_red.sum [byte]	0



```
@P0  EXIT                                         6    108    49   1,404,672
     IADD3 R7, P2, R0, UR7, RZ                    7    177    95   1,401,344
     IADD3 R6, P1, R4, UR4, RZ                    8     30     9   1,401,344
     ISETP.GE.U32.AND P0, PT, R7, UR5, PT         8     12     5   1,401,344
     IADD3.X R8, R2, UR8, RZ, P2, !PT             8     14     3   1,401,344
     IMAD.X R7, RZ, RZ, R5, P1                    9      9     0   1,401,344
     ISETP.GE.U32.AND.EX P0, PT, R8, UR6, PT, P0  9    106    35   1,401,344
     STG.E.U8 [R6.64], R3                         8    116    75   1,401,344
@P0  EXIT                                         8     92    33   1,401,344
     IADD3 R8, P2, R0, UR9, RZ                    9     45    14   1,397,120
     IADD3 R6, P1, R6, UR4, RZ                    9    248   145   1,397,120
     ISETP.GE.U32.AND P0, PT, R8, UR5, PT         9     57    17   1,397,120
     IADD3.X R8, R2, UR12, RZ, P2, !PT            9     11     4   1,397,120
     IMAD.X R7, RZ, RZ, R7, P1                    9      7     0   1,397,120
     ISETP.GE.U32.AND.EX P0, PT, R8, UR6, PT, P0  9     94    15   1,397,120
     STG.E.U8 [R6.64], R3                         8    104    61   1,397,120
```

2022.10.12_ncu-GEMV1-K_unroll1_block8x4_BelemLDG_L2PersistCache_StreamingB1_ldca_ldcs.ncu-rep
Page: Details
Result: 1 - 22425 - Kernel
Add Baseline
Apply Rules
Occupancy Calculator
Copy as Image
All

GPU SOL Section

Compute Workload Analysis Section

Memory Workload Analysis Section

Scheduler Statistics Section

	Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	22425 - Kernel (1, 16, 256)x(8, 4, 1)	336.06 usecond	464,057	36	0 - NVIDIA GeForce RTX 3090	1.38 cycle/nsecond	8.6	[1251049] 2d_Gemv

The report contains imported source files.

GPU Speed Of Light Throughput

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor. High-level overview of the utilization for compute and memory resources of the GPU presented as a roofline chart.

Compute (SM) Throughput [%]	11.07	Duration [usecond]	336.06
Memory Throughput [%]	89.89	Elapsed Cycles [cycle]	464,057
L1/TEX Cache Throughput [%]	22.64	SM Active Cycles [cycle]	451,791.39
L2 Cache Throughput [%]	39.70	SM Frequency [cycle/nsecond]	1.38
DRAM Throughput [%]	89.89	DRAM Frequency [cycle/nsecond]	9.40

High Throughput The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing DRAM in the Memory Workload Analysis section.

Roofline Analysis The ratio of peak float (fp32) to double (fp64) performance on this device is 64:1. The kernel achieved 2% of this device's fp32 peak performance and 0% of its

Compute Workload Analysis

Detailed analysis of the compute resources of the streaming multiprocessors (SM), including the achieved instructions per clock (IPC) and the utilization of each available pipeline. Pipelines with very high utilization might limit the overall performance.

Executed Ipc Elapsed [inst/cycle]	0.15	SM Busy [%]	3.84
Executed Ipc Active [inst/cycle]	0.15	Issue Slots Busy [%]	3.84
Issued Ipc Active [inst/cycle]	0.15		

Low Utilization All compute pipelines are under-utilized. Either this kernel is very small or it doesn't issue enough warps per scheduler. Check the Launch Statistics and Sched

Memory Workload Analysis

Detailed analysis of the memory resources of the GPU. Memory can become a limiting factor for the overall kernel performance when fully utilizing the involved hardware units (Mem Busy), exhausting the available communication bandwidth between those units (Max Bandwidth), or by reaching the maximum throughput of issuing memory instructions (Mem Pipes Busy). Detailed chart of the memory units. Detailed tables with data for each memory unit.

Memory Throughput [Gbyte/second]	810.93	Mem Busy [%]	39.70
L1/TEX Hit Rate [%]	50.58	Max Bandwidth [%]	89.89
L2 Hit Rate [%]	3.07	Mem Pipes Busy [%]	11.07
L2 Compression Success Rate [%]	0	L2 Compression Ratio	0

L1TEX Global Load Access Pattern The memory access pattern for global loads in L1TEX might not be optimal. On average, this kernel accesses 13.6 bytes per thread per memory request; but the address pattern, possibly caused by the stride between threads, results in 16.0 sectors per request, or 16.0*32 = 512.0 bytes of cache data transfers per request. The optimal thread address pattern for 13.6 byte accesses would result in 13.6*32 = 435.2 bytes of cache data transfers per request, to maximize L1TEX cache performance. Check the Source Counters section for uncoalesced global loads.

L2 Store Access Pattern The memory access pattern for stores from L1TEX to L2 is not optimal. The granularity of an L1TEX request to L2 is a 128 byte cache line. That is 4 c 4 sectors per cache line. Check the Source Counters section for uncoalesced stores and try to minimize how many cache lines need to be accessed p

Scheduler Statistics

Summary of the activity of the schedulers issuing instructions. Each scheduler maintains a pool of warps that it can issue instructions for. The upper bound of warps in the pool (Theoretical Warps) is limited by the launch configuration. On every cycle each scheduler checks the state of the allocated warps in the pool (Active Warps). Active warps that are not stalled (Eligible Warps) are ready to issue their next instruction. From the set of eligible warps the scheduler selects a single warp from which to issue one or more instructions (Issued Warp). On cycles with no eligible warps, the issue slot is skipped and no instruction is issued. Having many skipped issue slots indicates poor latency hiding.

Active Warps Per Scheduler [warp]	3.86	No Eligible [%]	96.02
Eligible Warps Per Scheduler [warp]	0.04	One or More Eligible [%]	3.98
Issued Warp Per Scheduler	0.04		

Issue Slot Utilization Every scheduler is capable of issuing one instruction per cycle, but for this kernel each scheduler only issues an instruction every 25.2 cycles. This might leave hardware resources underutilized and may lead to less optimal performance. Out of the maximum of 12 warps per

SOL SECTION

Sections

GPU Utilization
SM [%]
Memory [%]
0.0
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.0
Speed Of Light [%]

SOL SM Breakdown

SOL SM: Inst Executed Pipe Lsu [%]	14.18
SOL SM: Issue Active [%]	7.10
SOL SM: Inst Executed [%]	7.09
SOL SM: Mio Inst Issued [%]	6.30
SOL SM: Pipe Al	4.73
SOL SM: Inst Ex	4.73
SOL SM: Mio2rf Writeback Active [%]	3.54
SOL SM: Inst Executed Pipe Cbu Pred On Any [%]	2.45
SOL SM: Pipe Fma Cycles Active [%]	2.36
SOL SM: Mio Pq Write Cycles Active [%]	2.36
SOL SM: Mio Pq Read Cycles Active [%]	2.36
SOL IDC: Request Cycles Active [%]	0
SOL SM: Inst Executed Pipe Xu [%]	0
SOL SM: Inst Executed Pipe Uniform [%]	0
SOL SM: Inst Executed Pipe Tex [%]	0
SOL SM: Inst Executed Pipe Ipa [%]	0
SOL SM: Inst Executed Pipe Fp16 [%]	0
SOL SM: Pipe Fp64 Cycles Active [%]	0
SOL SM: Pipe Shared Cycles Active [%]	0
SOL SM: Pipe Tensor Cycles Active [%]	0

sm__mio_inst_issued.avg.pct_of_peak_sustained_elapsed
# of instructions issued from MIOC to MIO

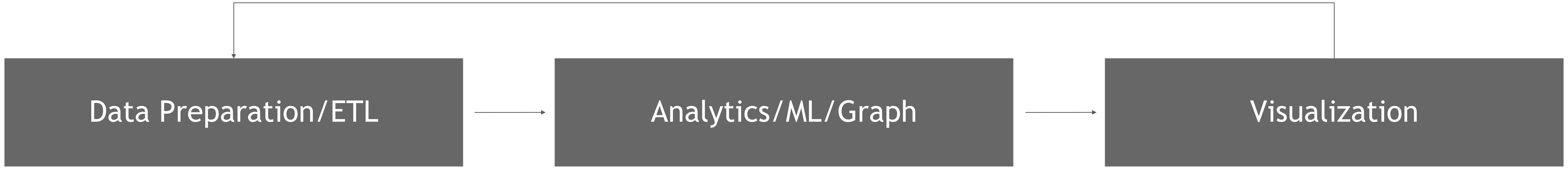
SOL Memory Breakdown

SOL GPU: Dram Throughput [%]	81.56
SOL L2: T Sectors [%]	30.69
SOL L2: Xbar2lts Cycles Active [%]	17.81
SOL L2: D Sectors Fill Device [%]	15.77
SOL L2: Lts2xbar Cycles Active [%]	15.36
SOL L2: D Sectors [%]	15.00
SOL L1: Lsuin Requests [%]	14.18
SOL L1: M L1tex2xbar Req Cycles Active [%]	10.72
SOL L1: M Xbar2l1tex Read Sectors [%]	9.45
SOL L2: T Tag Requests [%]	7.69
SOL L1: Data Pipe Lsu Wavefronts [%]	7.50
SOL L1: Lsu Writeback Active [%]	4.73
SOL L1: Data Bank Writes [%]	2.36
SOL L1: Data Bank Reads [%]	2.36
SOL L1: Texin Sm2tex Req Cycles Active [%]	0.00
SOL L1: F Wavefronts [%]	0.00
SOL L2: D Sectors Fill Sysmem [%]	0
SOL L2: D Atomic Input Cycles Active [%]	0
SOL L1: Data Pipe Tex Wavefronts [%]	0
SOL L1: Tex Writeback Active [%]	0

PYTHON ECOSYSTEM

The RAPIDS suite of open source software libraries gives you the freedom to execute end-to-end data science and analytics pipelines entirely on NVIDIA GPUs.

General Purpose and Domain-Specific Libraries



cuDF

- GPU-accelerated ETL functions
- Tracks Pandas and other common PyData APIs
- Dask + UCX integration for scaling

RAPIDS for Apache Spark

- RAPIDS accelerator for Apache Spark

RAPIDS ML

- GPU-native cuML library, plus XGBoost, FIL, HPO, and more

cuGraph

- GPU graph analytics, including Louvain, PageRank, and more
- Multi node Multi GPU features

cuxfilter

- GPU-accelerated cross-filtering

Viz integration

- pyViz: Plotly Dash, Bokeh, Datashader, HoloViews, hvPlot
- Node-RAPIDS bindings for node.js

Domain-Specific Libraries

CLX + Morpheus
Cyber log processing + anomaly detection

cuStreamz
Streaming analytics

cuSignal
Signals processing

cuCIM
Computer vision & image processing primitives

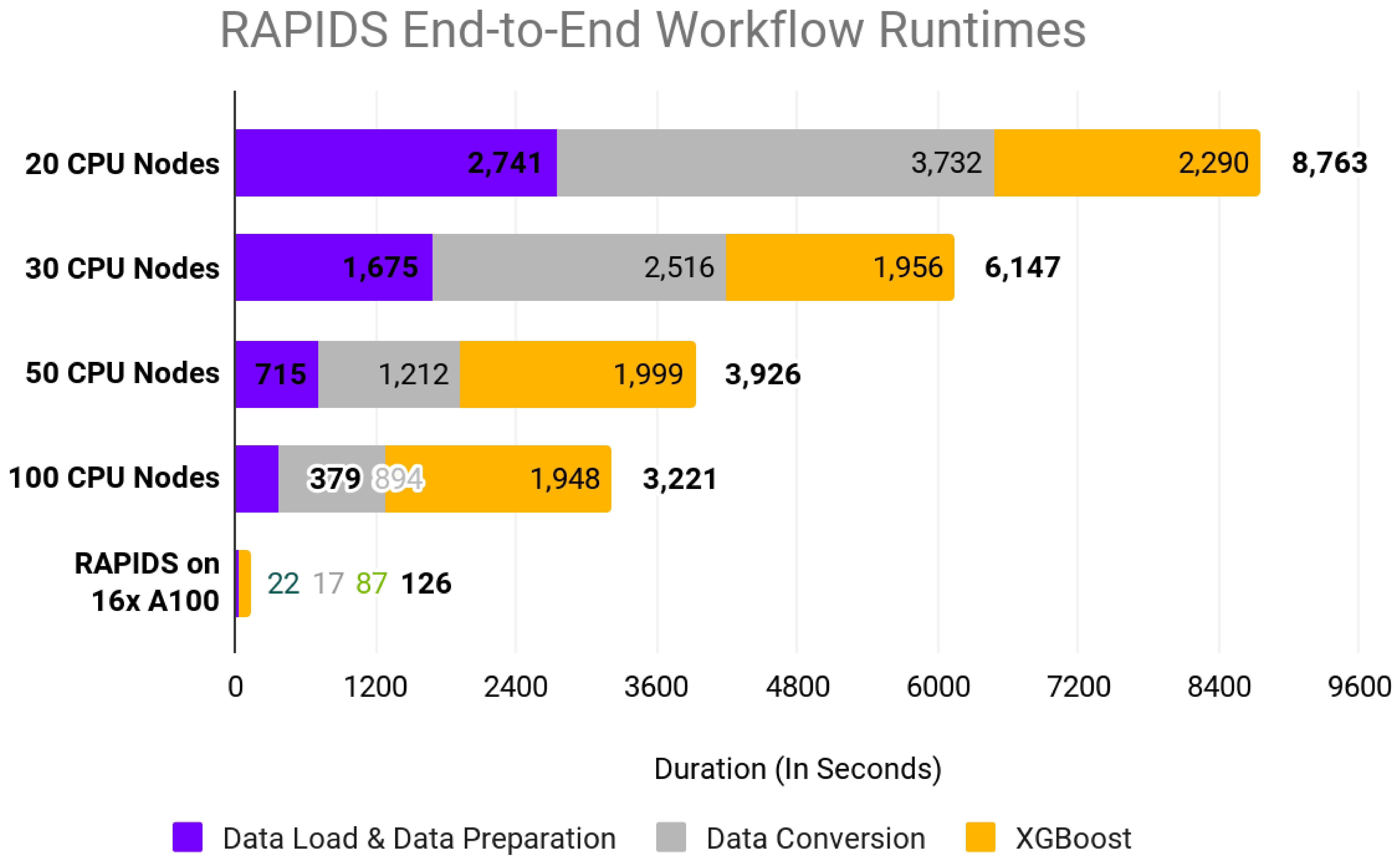
cuSpatial
Spatial analytics

NVTabular
RecSys ETL library

...and more!

Lightning-Fast End-to-End Performance

Reducing Data Science Processes from Hours to Seconds



16 A100s Provide More Power than 100 CPU Nodes

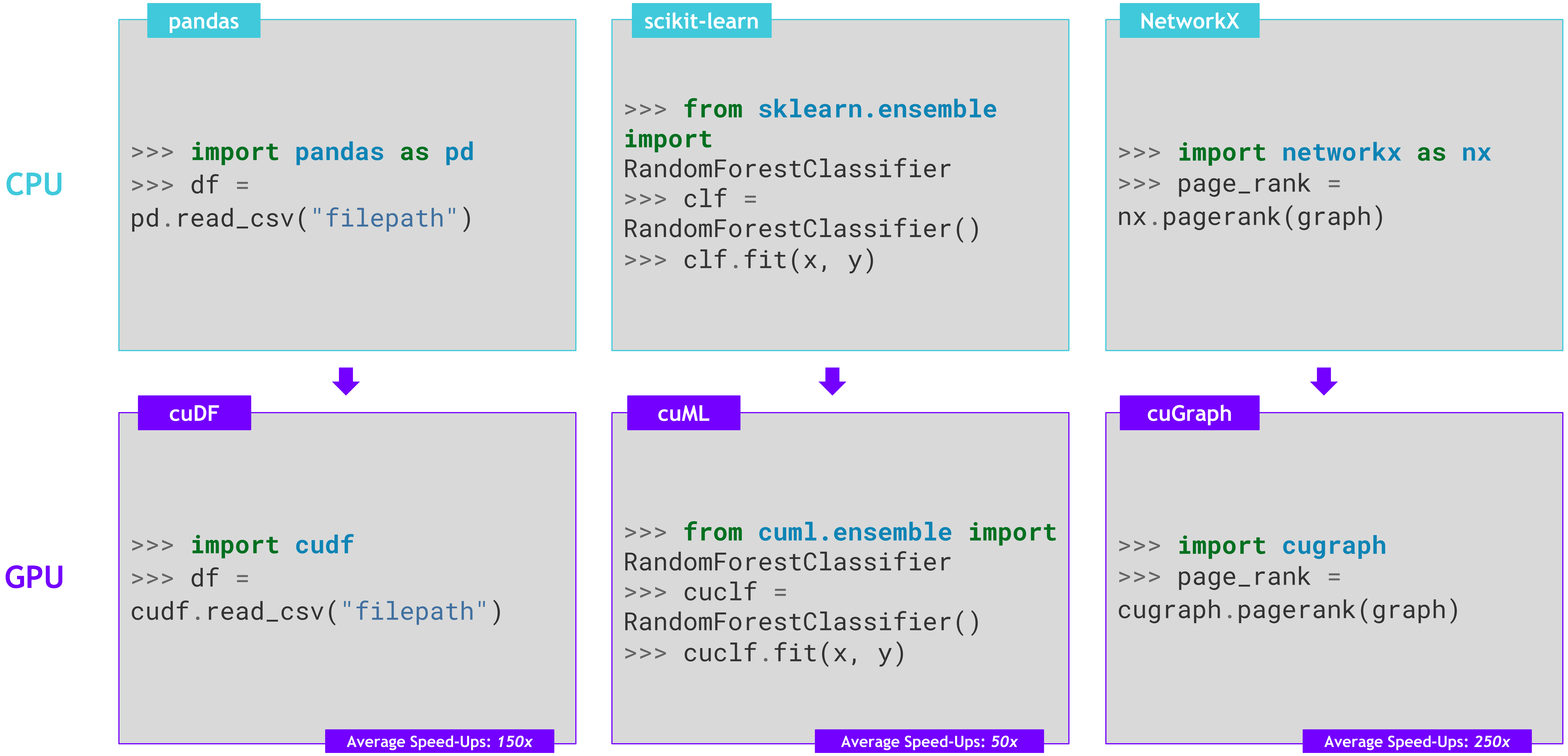
70x Faster Performance than Similar CPU Configuration

20x More Cost-Effective than Similar CPU Configuration

*CPU approximate to n1-highmem-8 (8 vCPUs, 52GB memory) on Google Cloud Platform. TCO calculations-based on Cloud instance costs.

Minor Code Changes for Major Benefits

Abstracting Accelerated Compute through Familiar Interfaces



CPU

pandas

```
>>> import pandas as pd
>>> df =
pd.read_csv("filepath")
```

scikit-learn

```
>>> from sklearn.ensemble
import
RandomForestClassifier
>>> clf =
RandomForestClassifier()
>>> clf.fit(x, y)
```

NetworkX

```
>>> import networkx as nx
>>> page_rank =
nx.pagerank(graph)
```

GPU

cuDF

```
>>> import cudf
>>> df =
cudf.read_csv("filepath")
```

Average Speed-Ups: *150x*

cuML

```
>>> from cuml.ensemble import
RandomForestClassifier
>>> cuclf =
RandomForestClassifier()
>>> cuclf.fit(x, y)
```

Average Speed-Ups: *50x*

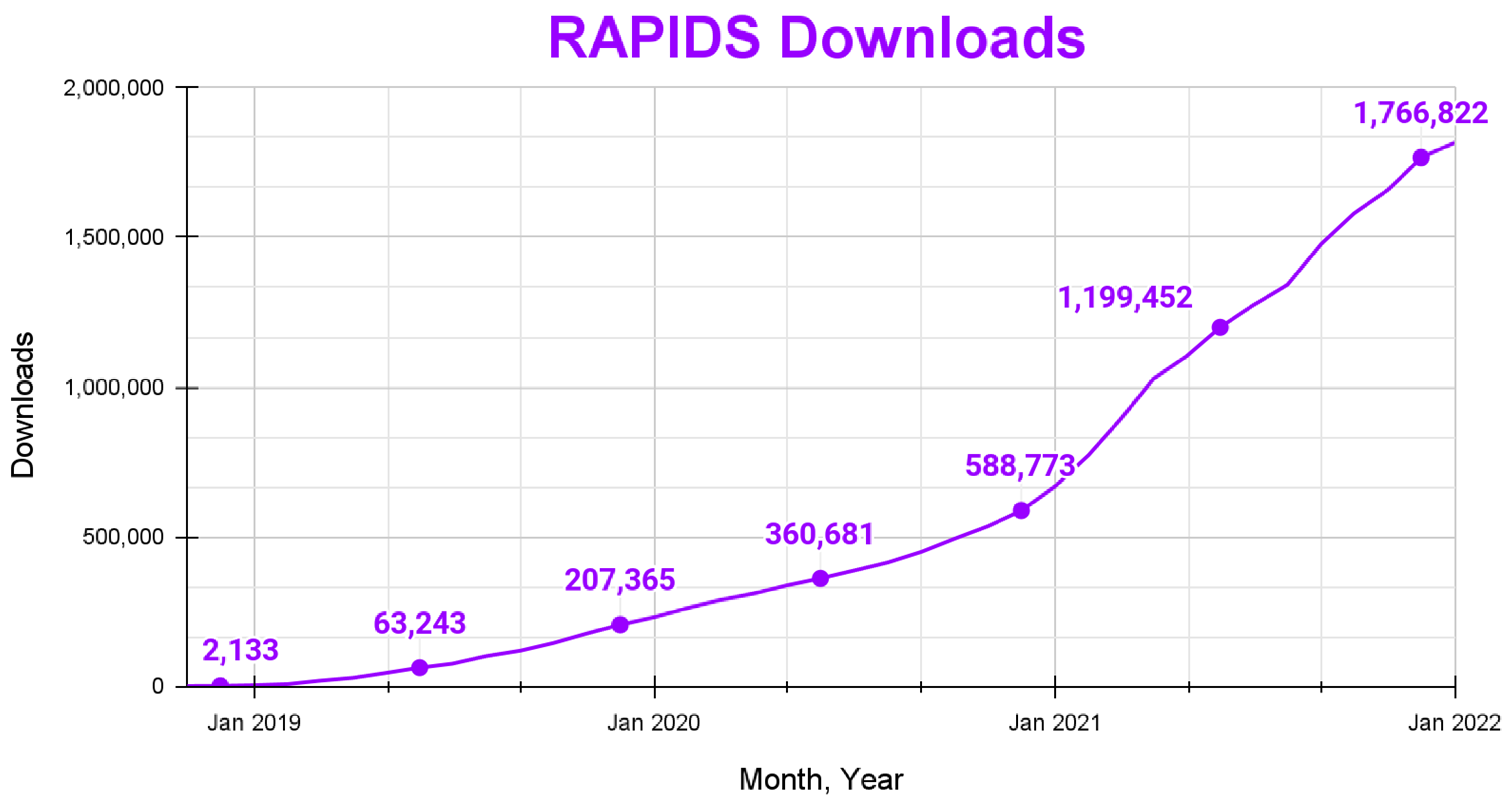
cuGraph

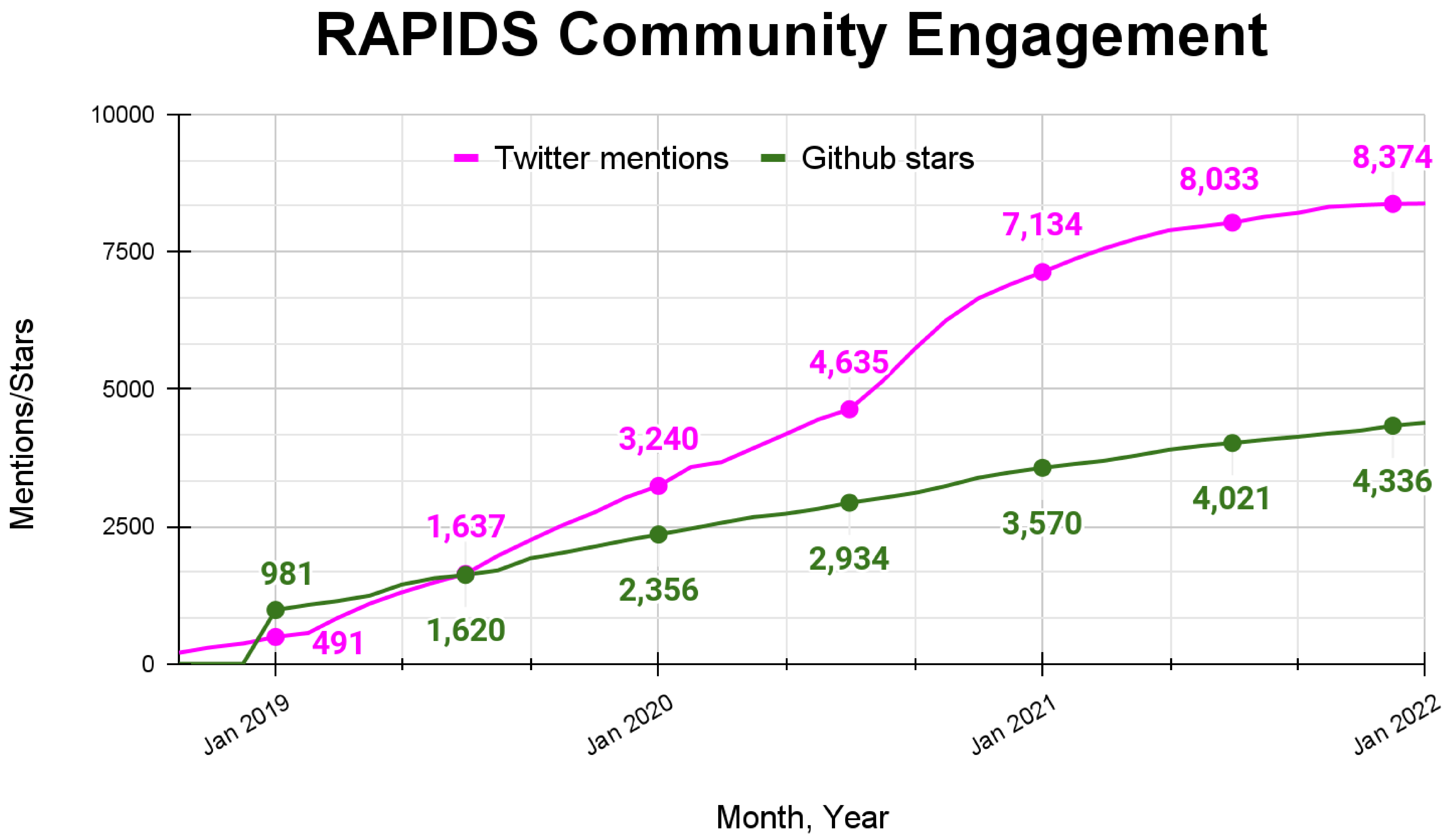
```
>>> import cugraph
>>> page_rank =
cugraph.pagerank(graph)
```

Average Speed-Ups: *250x*

3 Year Anniversary

More than **1.5M** downloads and accelerating





NVIDIA INNOVATIONS IN SPARK 3.0

Accelerate data science pipelines without code changes

RAPIDS Accelerator for Spark 3.0

Intercepts and accelerates SQL and DataFrame operations, dramatically improving ETL performance

Spark Shuffle

Operations that sort, group or join data by value. Data is moved between partitions via the new RAPIDS Accelerator Shuffle in a process called ‘shuffle’

Modifications to Spark Components

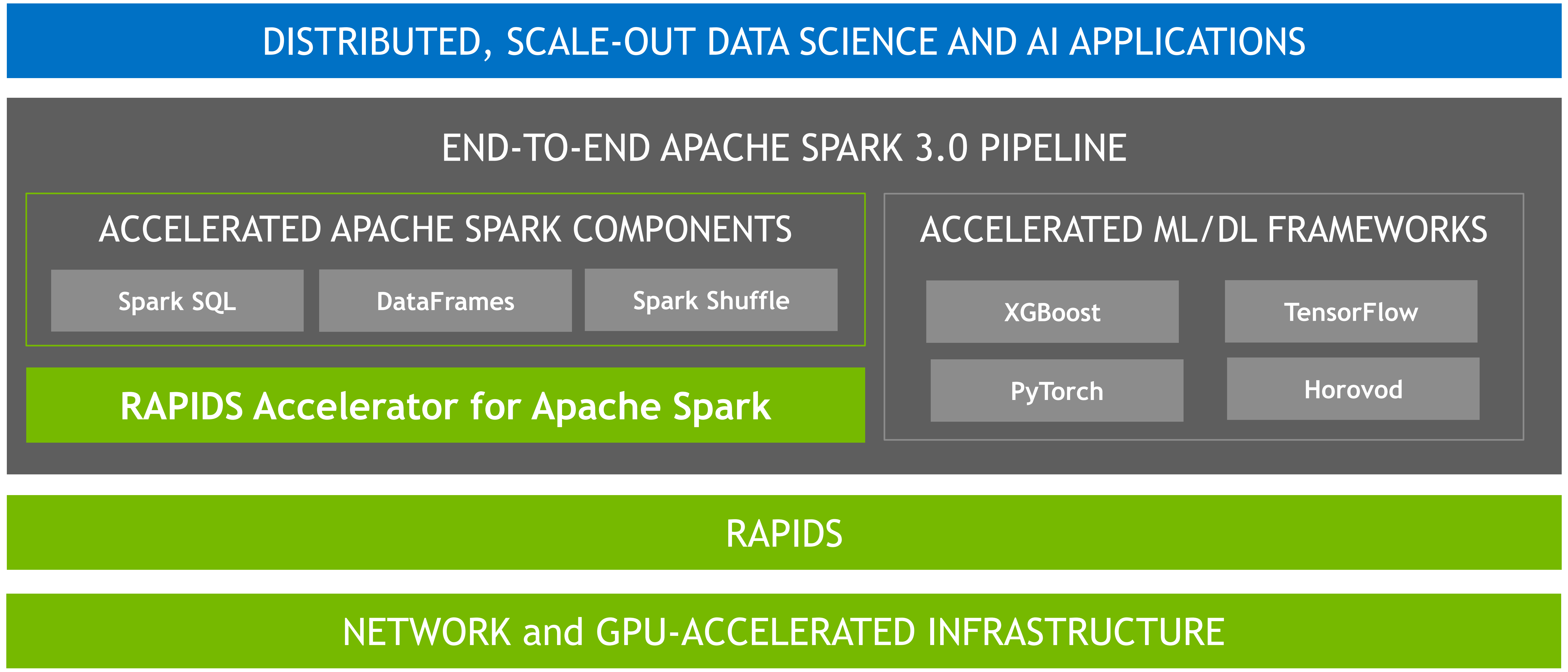
Columnar processing support in the Catalyst query optimizer

Spark shuffle implementation that optimizes the data transfer between Spark processes using RDMA/RoCE and GPU Direct

GPU-Aware Scheduling in Spark

Spark 3.0 places GPU-accelerated workloads directly onto servers containing the necessary GPU resources

Spark standalone, YARN, and Kubernetes clusters


DISTRIBUTED, SCALE-OUT DATA SCIENCE AND AI APPLICATIONS
END-TO-END APACHE SPARK 3.0 PIPELINE
ACCELERATED APACHE SPARK COMPONENTS
Spark SQL
DataFrames
Spark Shuffle
RAPIDS Accelerator for Apache Spark
ACCELERATED ML/DL FRAMEWORKS
XGBoost
TensorFlow
PyTorch
Horovod
RAPIDS
NETWORK and GPU-ACCELERATED INFRASTRUCTURE

SCALE OUT PYTHON TOOLS WITH RAPIDS + DASK

DISTRIBUTE & ACCELERATE COMPUTATION FOR PRODUCTION WORKLOADS

Scale Up / Accelerate

RAPIDS

Accelerates PyData on NVIDIA GPUs

NumPy -> CuPy/PyTorch/..
Pandas -> cuDF
Scikit-Learn -> cuML
Numba -> Numba

RAPIDS + DASK

Distributes and accelerates PyData

Can be distributed across Multi-GPU on single node (DGX) or across a cluster

Provides easy to use tooling enabling HPC-level performance

PYDATA

Provides accessible, easy to use tooling

NumPy, Pandas, Scikit-Learn, Numba and many more

Single CPU core, in-memory data

DASK

Distributes PyData across multiple cores

NumPy -> Dask Array
Pandas -> Dask DataFrame
Scikit-Learn -> Dask-ML
… -> Dask Futures

Scale Out / Parallelize

CUNUMERIC

Automatic NumPy Acceleration and Scalability

cuNumeric

CuNumeric transparently accelerates and scales existing Numpy workloads

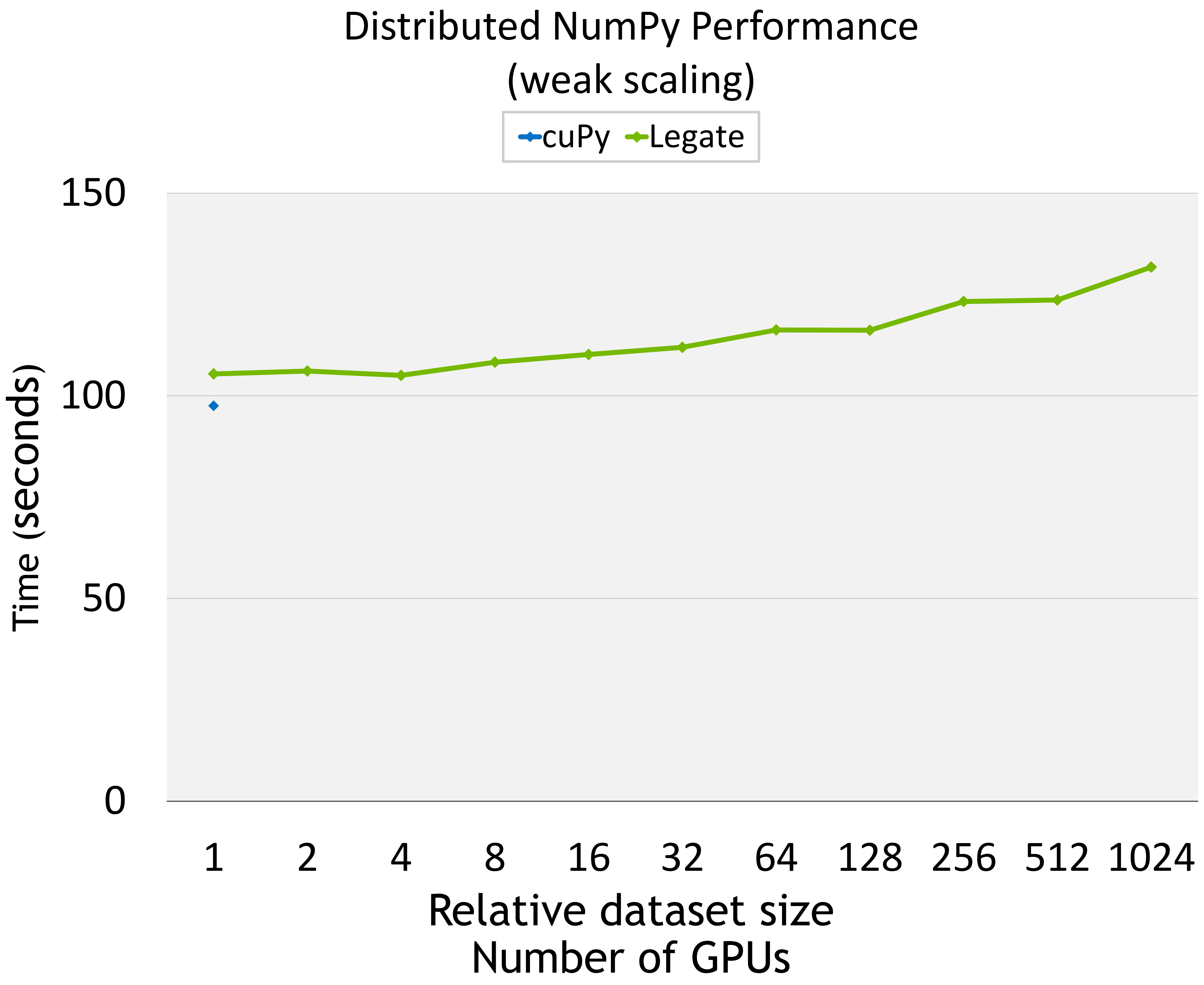
Program from the edge to the supercomputer in Python by changing 1 import line

Pass data between Legate libraries without worrying about distribution or synchronization requirements

Alpha release available at github.com/nv-legate

```
import cunumeric as np

a = np.random.randn(160_000).reshape(400, 400)
b = a + a.T
b
```


Distributed NumPy Performance (weak scaling)

QUANTUM COMPUTING

A NEW COMPUTING MODEL - QUANTUM

NEW COMPUTING MODEL

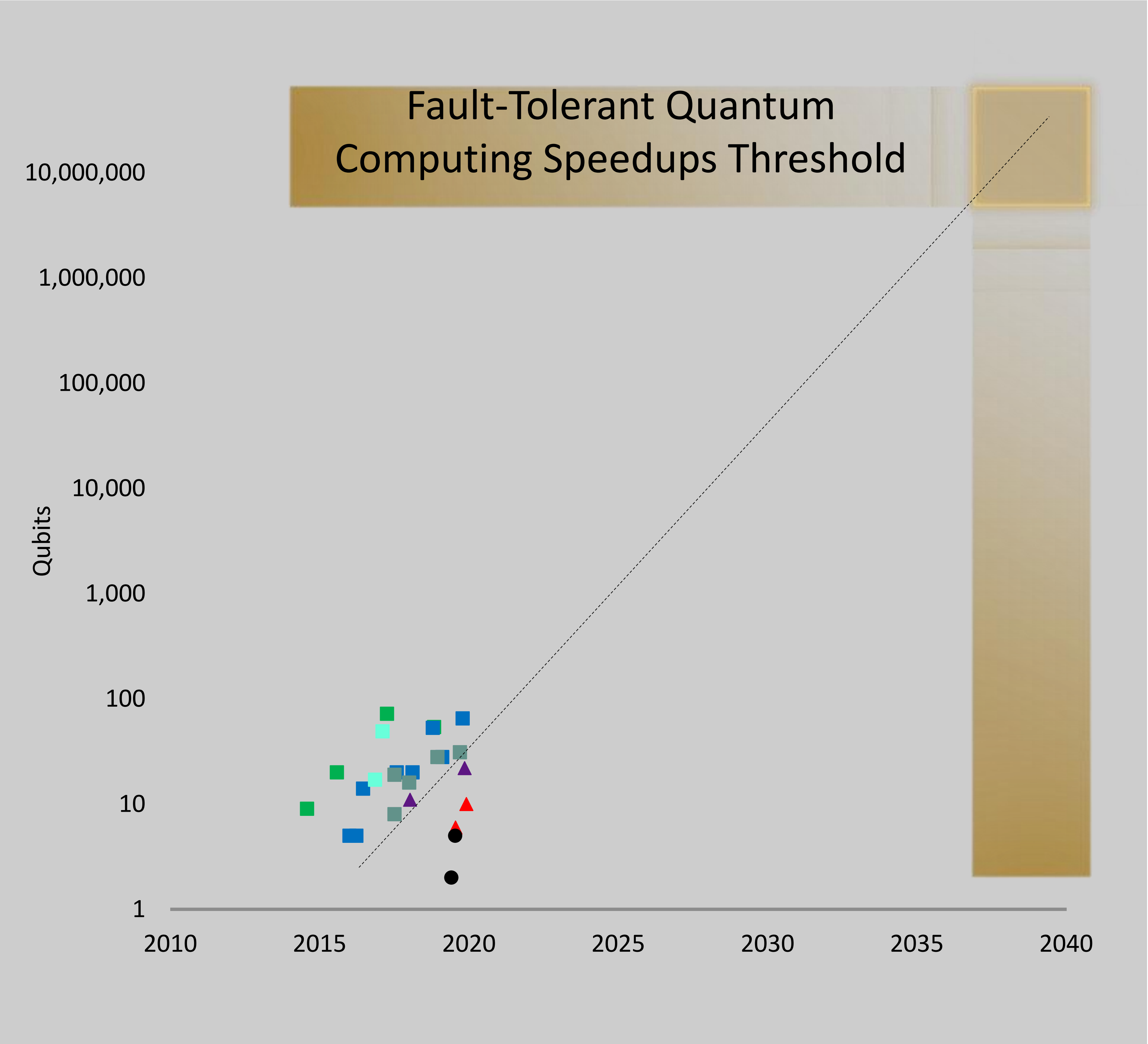
Computational Finance

Quantum Chemistry

Cryptography

Optimization

POTENTIAL USE CASES



QUANTUM SYSTEMS SCALING EXPONENTIALLY

DGX cuQuantum Appliance

Qiskit Integration with multi-node, multi-GPU support

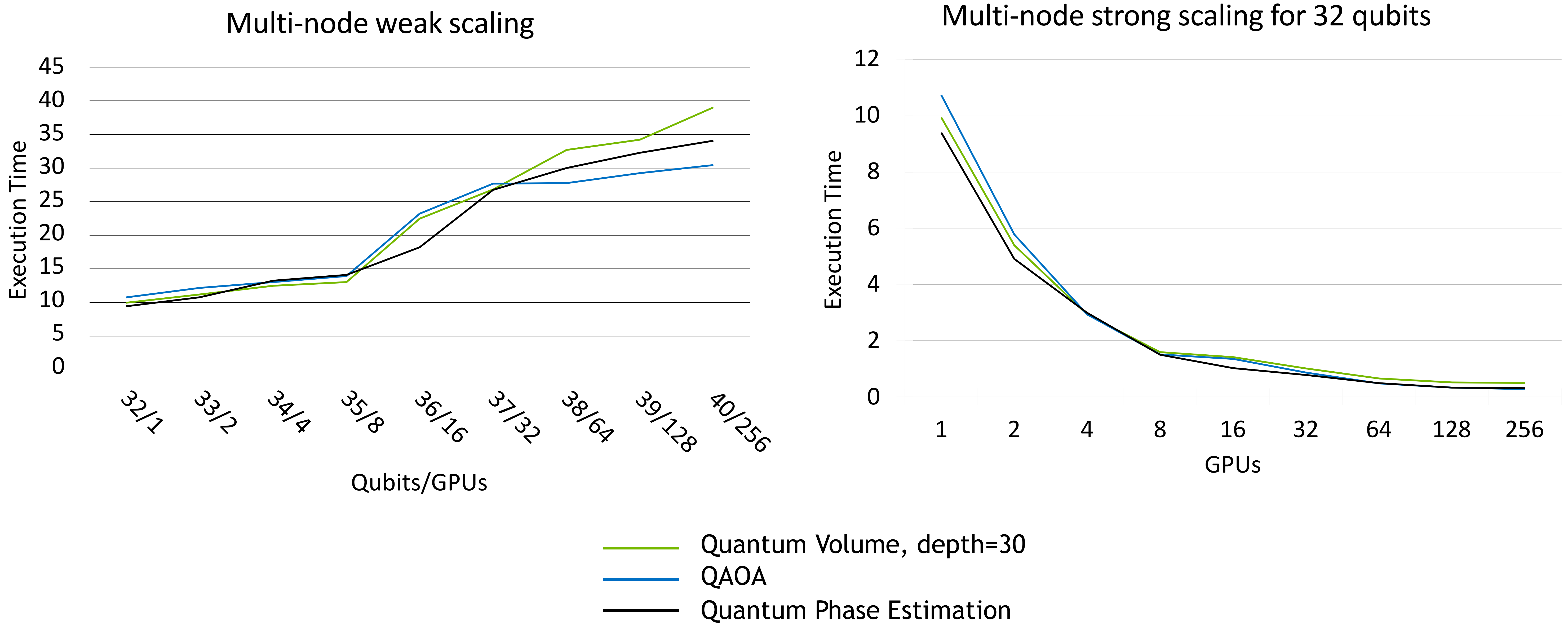
Fully integrated quantum simulation solution

- State-of-the-art ***performance***
- Unmatched simulation ***scale***

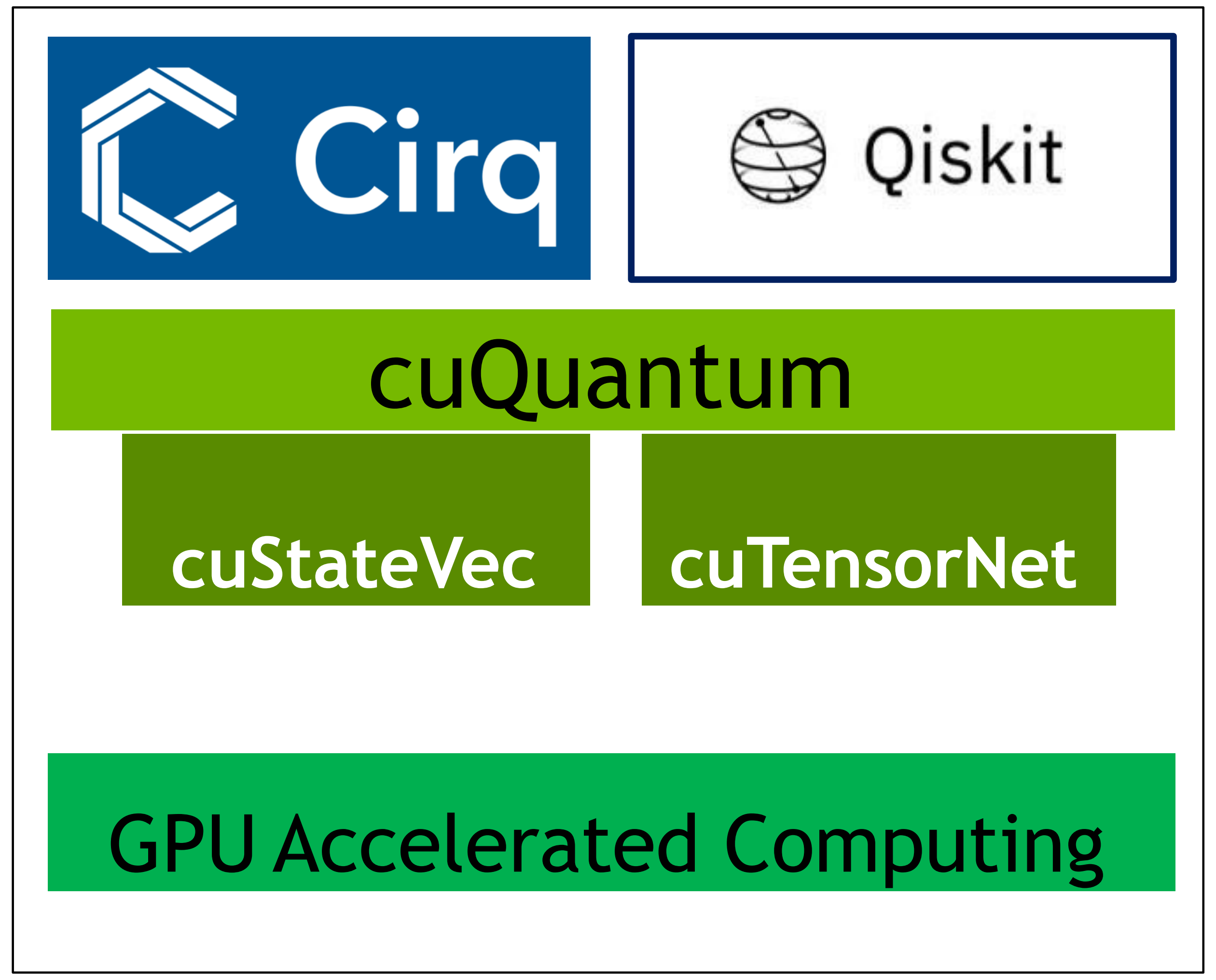
Reduce the simulation time by ***orders of magnitude***

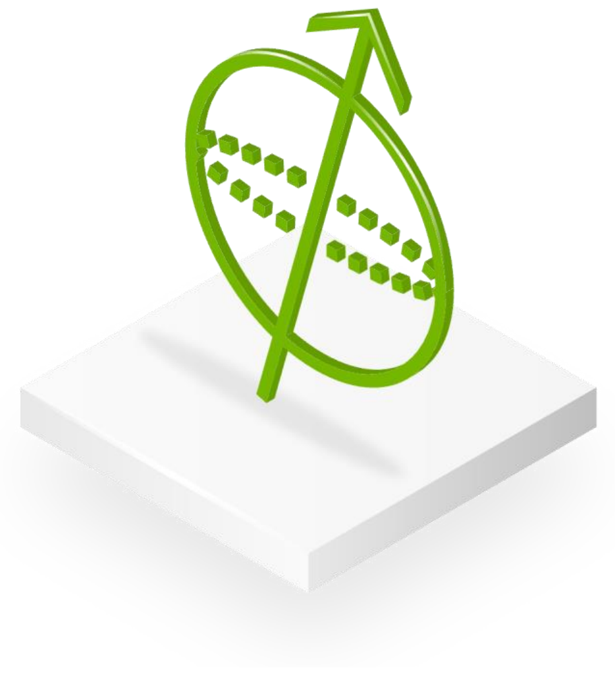
Simulate ***thousands*** of perfect or noisy qubits

Trivially Scale Quantum Algorithms with Industry Leading Performance

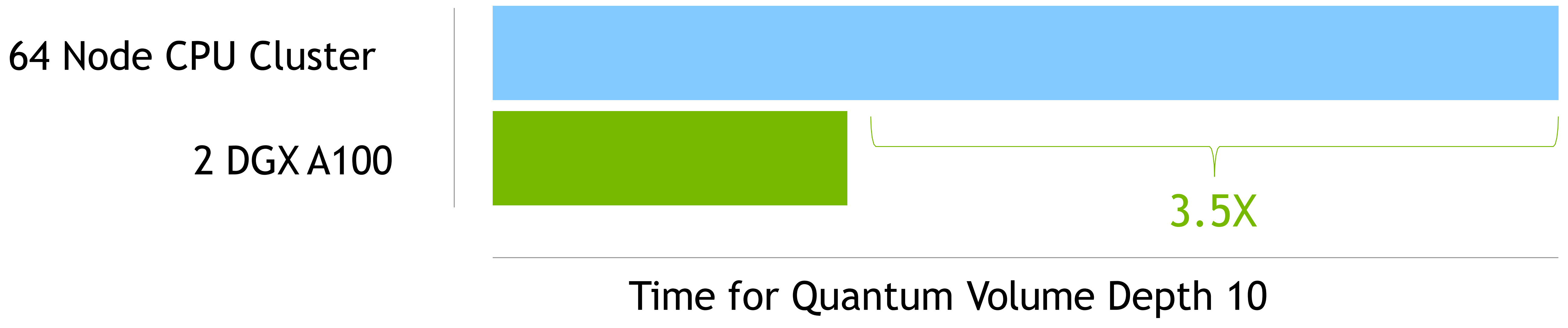


DGX cuQuantum Appliance



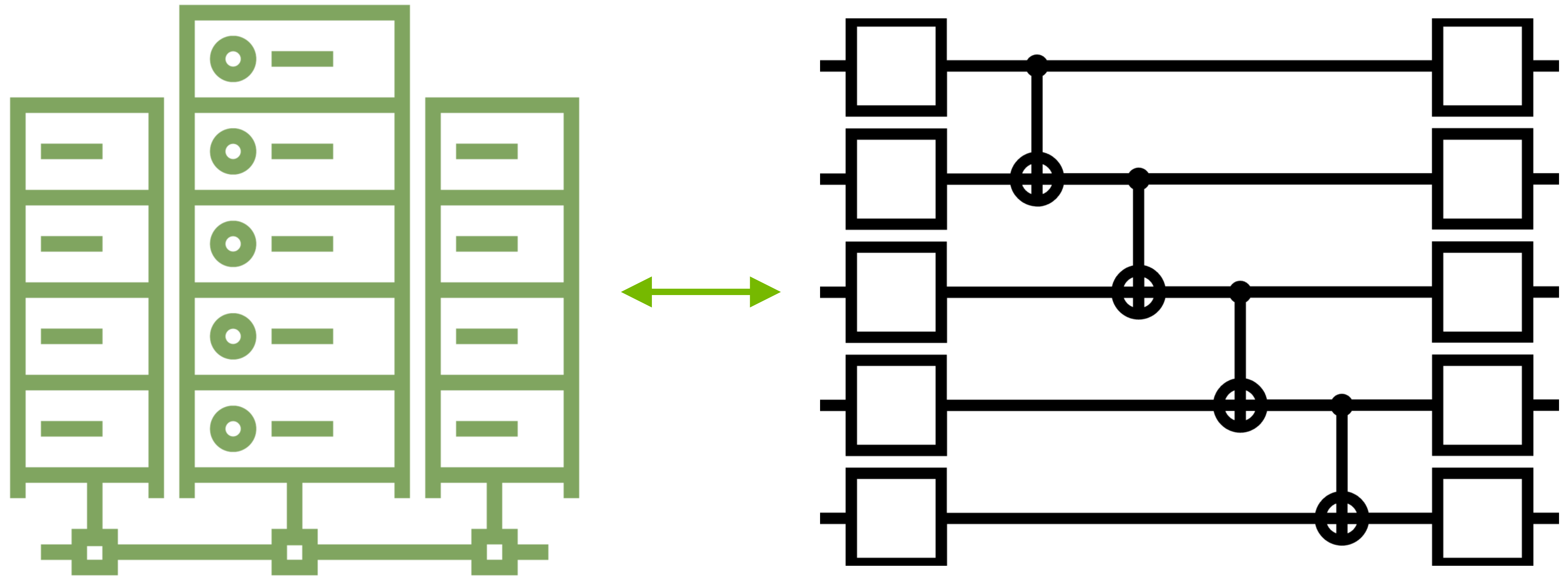


Record breaking performance
2 DGX A100 vs previous best on 64 node CPU cluster



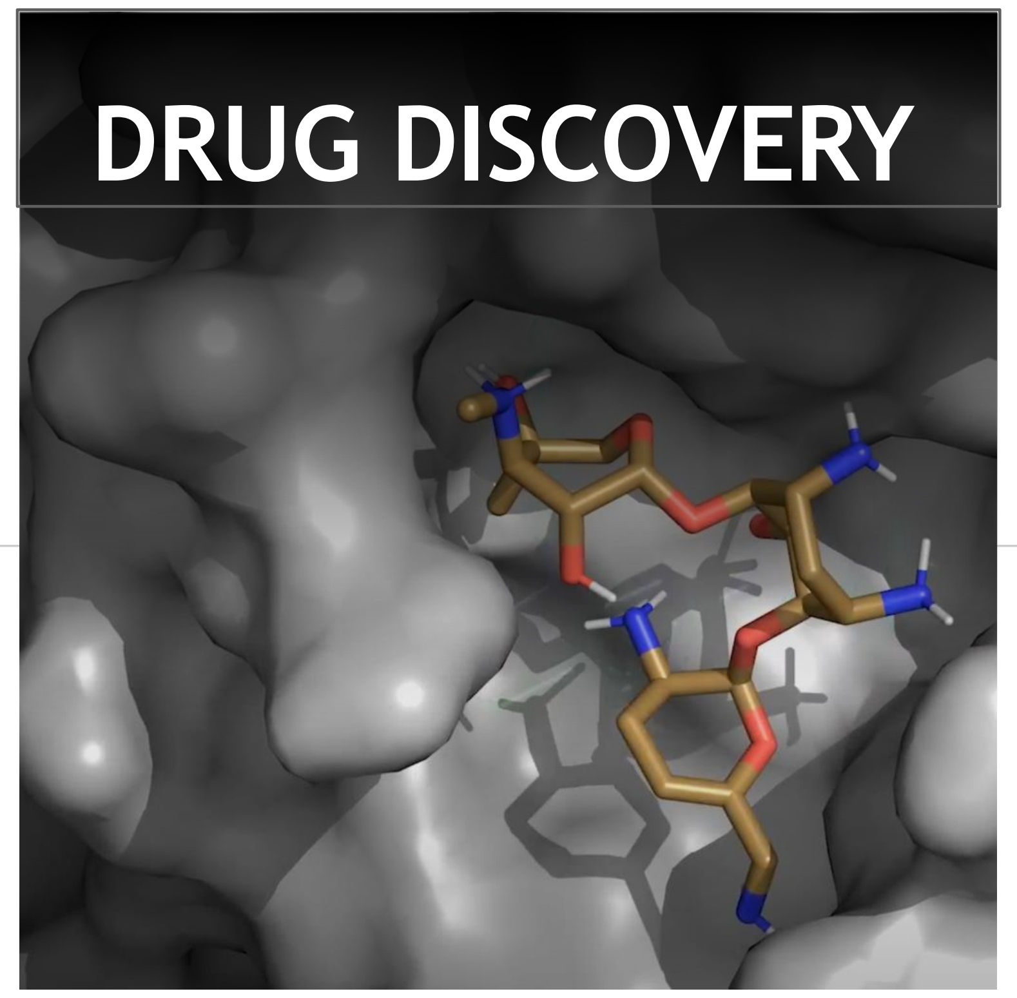
ALL VALUABLE QUANTUM APPLICATIONS WILL BE HYBRID

Various Scientific Domains

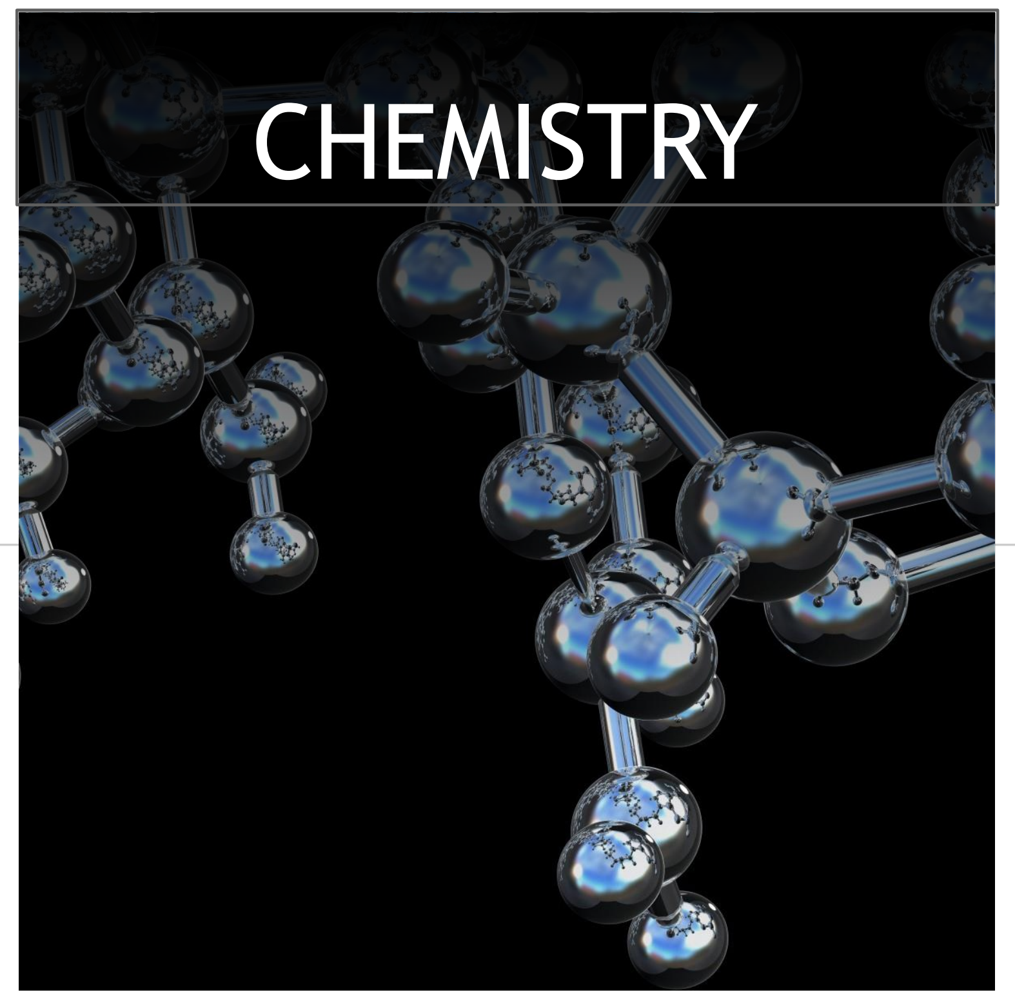


Classical Supercomputer

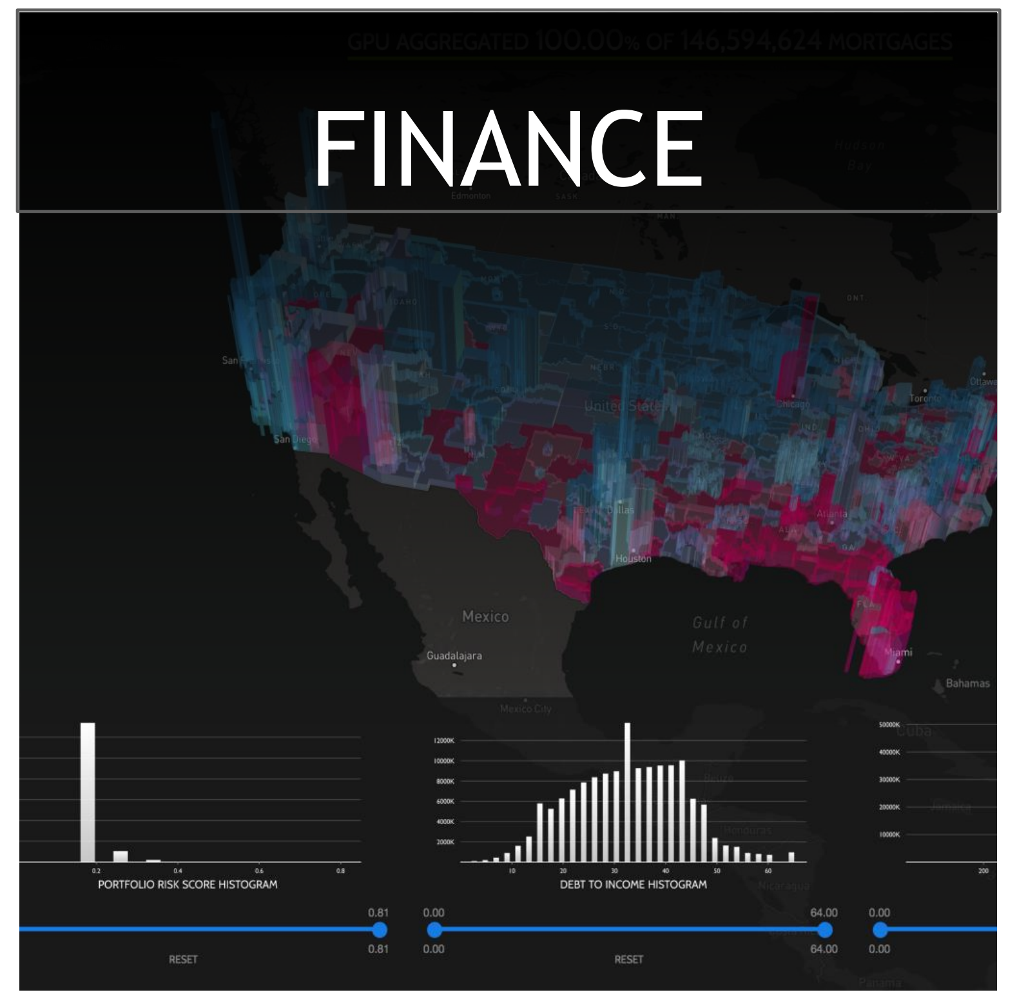
Quantum Computer



DRUG DISCOVERY



CHEMISTRY



FINANCE

OPTIMIZATION

Introducing NVIDIA QODA

A Platform For Hybrid Quantum-classical Computing

NVIDIA QODA PLATFORM

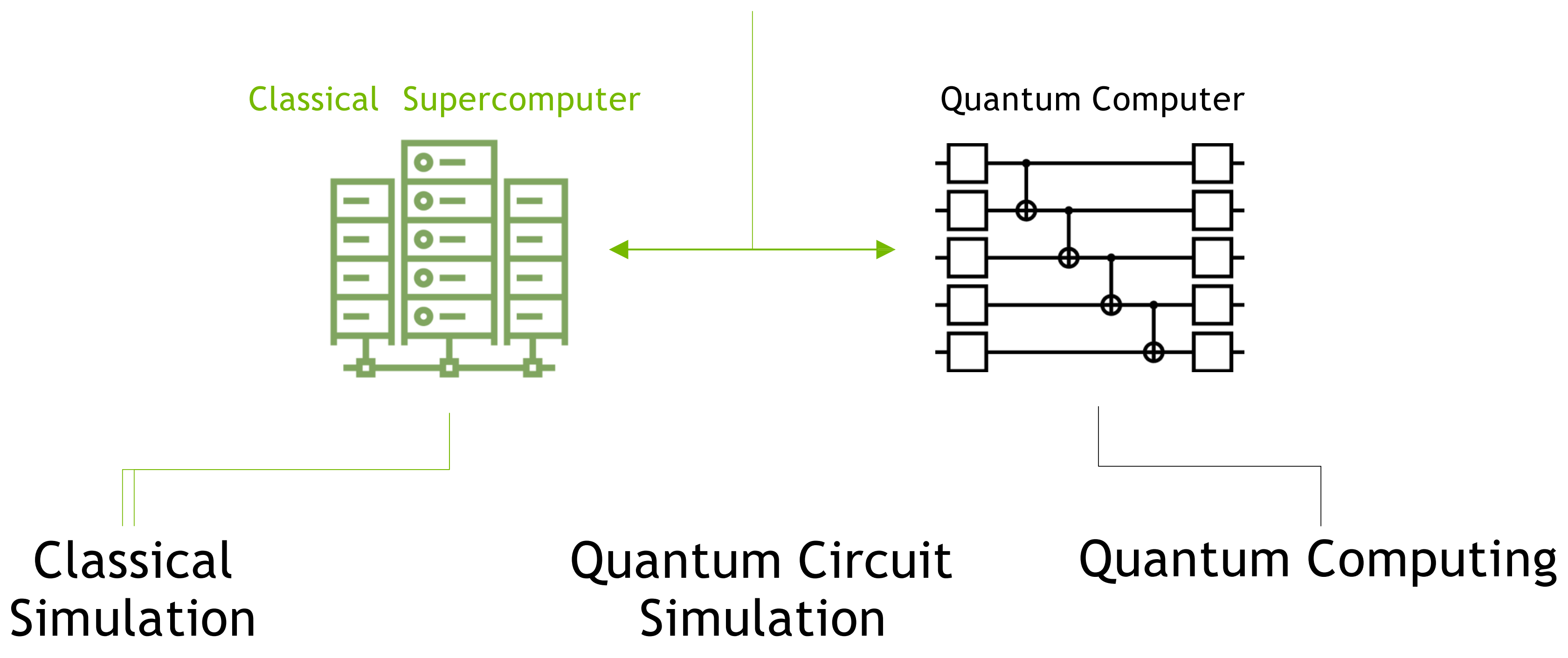
HYBRID APPLICATIONS

Drug Discovery, Chemistry, Weather, Finance, Logistics, and More

NVIDIA QODA

Hybrid Quantum-Classical Programming Platform

SYSTEM-LEVEL COMPILER TOOLCHAIN (NVQ++)



Classical Supercomputer

Quantum Computer

Classical Simulation

Quantum Circuit Simulation

Quantum Computing

QODA FEATURES

Supports any kind of QPU, emulated or physical

Compiler for hybrid systems

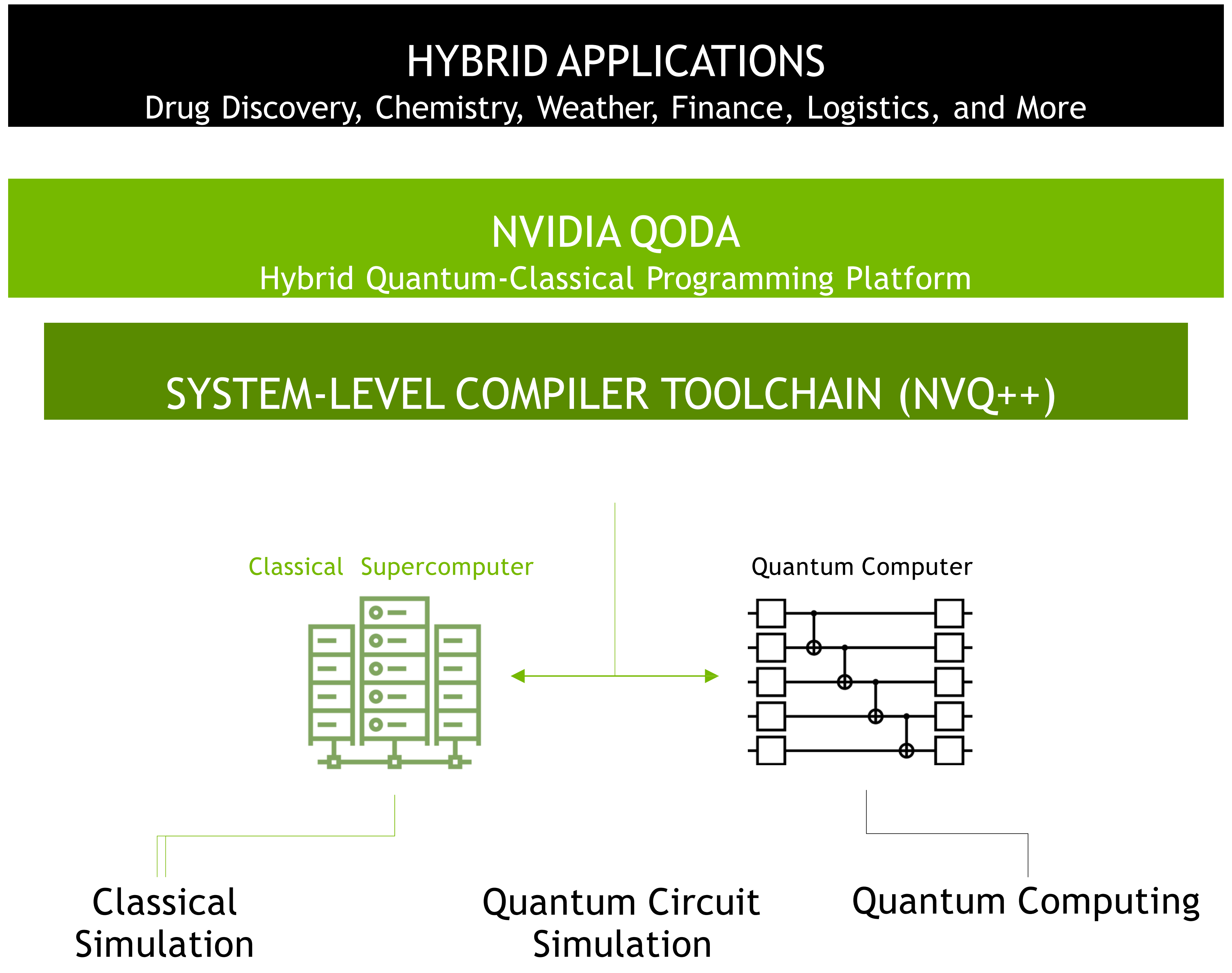
Open and interoperable with today’s applications

Single source C++ and Python programming model

Introducing NVIDIA QODA

Adopted by Community’s Global Leaders to Enable Quantum-Accelerated Applications

NVIDIA QODA PLATFORM



HYBRID APPLICATIONS

Drug Discovery, Chemistry, Weather, Finance, Logistics, and More

NVIDIA QODA

Hybrid Quantum-Classical Programming Platform

SYSTEM-LEVEL COMPILER TOOLCHAIN (NVQ++)

Classical Supercomputer

Quantum Computer

Classical Simulation

Quantum Circuit Simulation

Quantum Computing

QODA PLATFORM ECOSYSTEM

Thanks