NVIDIA HPC SOFTWARE - ALCF COMPUTATIONAL PERFORMANCE WORKSHOP

JEFF LARKIN, PRINCIPAL HPC APPLICATION ARCHITECT, NVIDIA

AGENDA

Accelerated Computing with Standard Languages

GPU Supercomputing in the PyData Ecosystem

Advancements in HPC Libraries

NVIDIA Developer Tools

AGENDA

Accelerated Computing with Standard Languages

GPU Supercomputing in the PyData Ecosystem

Advancements in HPC Libraries

NVIDIA Developer Tools

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++

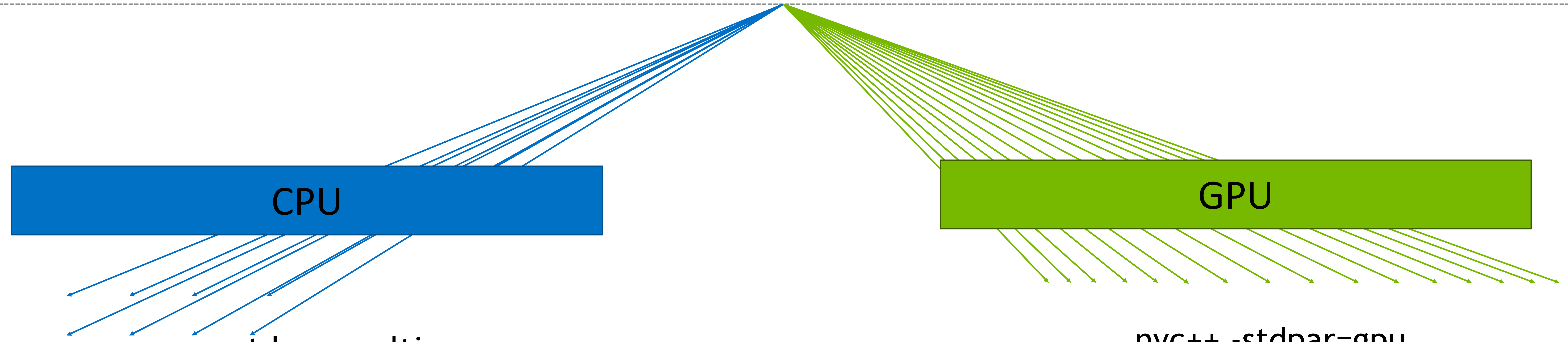
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

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
```



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

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

FUTURE OF CONCURRENCY AND PARALLELISM IN HPC: STANDARD LANGUAGES

How did we get here?

ON-GOING LONG-TERM INVESTMENT

ISO committee participation from industry, academia and government labs.

Fruit born in 2020 was planted over the previous decade.

Focus on enhancing concurrency and parallelism for all.

Open collaboration between partners and competitors.

Past investments in directives enabled rapid progress.

MAJOR FEATURES

Memory Model Enhancements

C++14 Atomics Extensions

C++17 Parallel Algorithms

C++20 Concurrency Library

C++23 Multi-Dim. Array Abstractions

C++23 Extended Floating Point Types

C++23 Range Based Parallel Algorithms

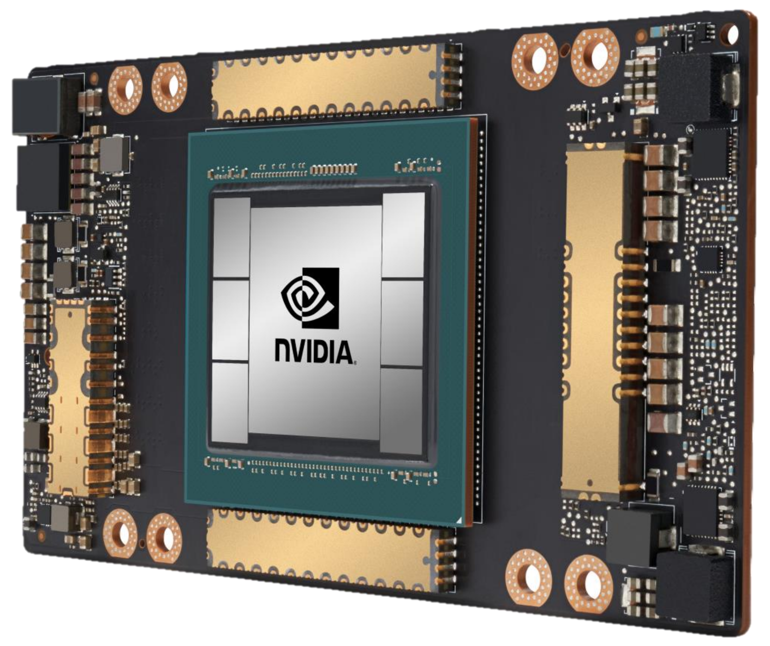
C++2X Executors

C++2X Linear Algebra

Fortran 202X DO CONCURRENT Reduction

HPC COMPILERS

NVC | NVC++ | NVFORTRAN



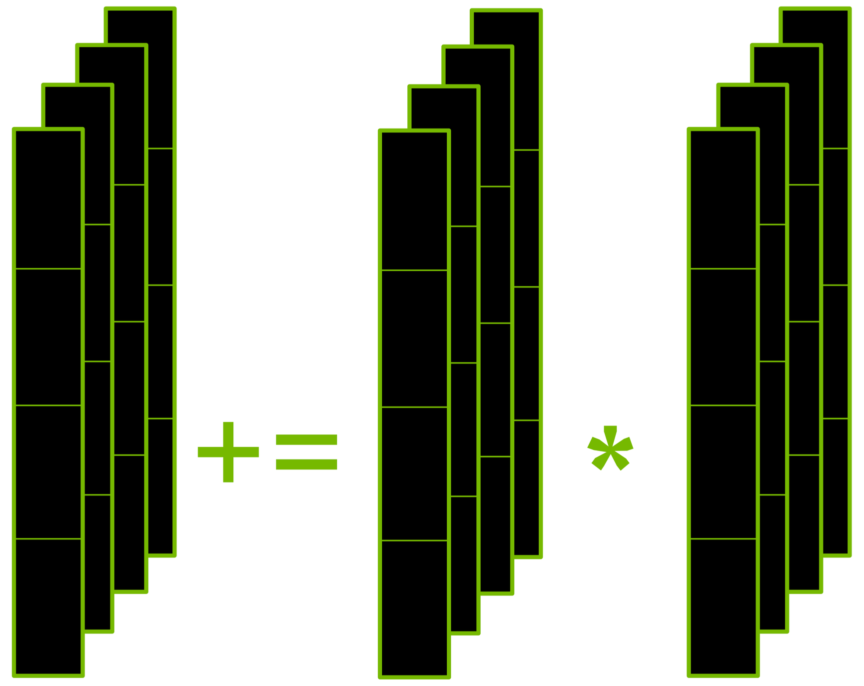
Accelerated

A100
Automatic

Fortran
NVIDIA CUDA
OpenACC
More Science, Less Programming
OpenMP

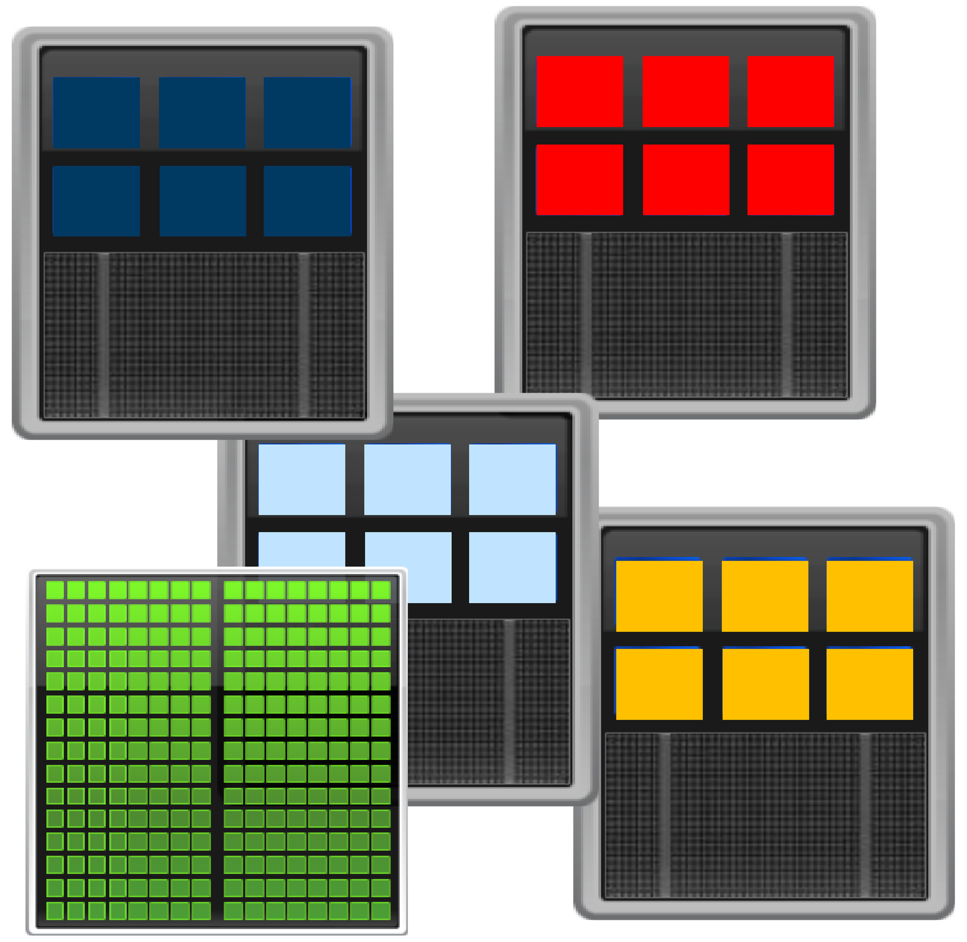
Programmable

Standard Languages
Directives
CUDA



CPU Optimized

Directives
Vectorization



Multi-Platform

x86_64
Arm
OpenPOWER

NVIDIA HPC SDK

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

Develop for the NVIDIA Platform: GPU, CPU and Interconnect

Libraries | Accelerated C++ and Fortran | Directives | CUDA

x86_64 | Arm | OpenPOWER

7-8 Releases Per Year | Freely Available

PARALLEL PROGRAMMING WITH ISO C++

HPC PROGRAMMING IN ISO C++

ISO is the place for portable concurrency and parallelism

C++17 & C++20

Parallel Algorithms

- In NVC++
- Parallel and vector concurrency

Forward Progress Guarantees

- Extend the C++ execution model for accelerators

Memory Model Clarifications

- Extend the C++ memory model for accelerators

Ranges

- Simplifies iterating over a range of values

Scalable Synchronization Library

- Express thread synchronization that is portable and scalable across CPUs and accelerators
- In libcu++:
  - `std::atomic<T>`
  - `std::barrier`
  - `std::counting_semaphore`
  - `std::atomic<T>::wait/notify_*`
  - `std::atomic_ref<T>`

Preview support coming to NVC++

C++23

`std::mdspan/mdarray`

- HPC-oriented multi-dimensional array abstractions.

Range-Based Parallel Algorithms

- Improved multi-dimensional loops

Extended Floating Point Types

- First-class support for formats new and old: `std::float16_t/float64_t`

And Beyond

Executors / Senders-Recievers

- Simplify launching and managing parallel work across CPUs and accelerators

Linear Algebra

- C++ standard algorithms API to linear algebra
- Maps to vendor optimized BLAS libraries

PARALLELISM IN C++ ROADMAP

C++ 11

- Memory model enhancements
- Lambdas

C++ 14

- Atomics extensions
- Generic Lambda Expressions

C++ 17

- Parallel algorithms
- Forward progress guarantees
- Memory model clarifications

C++ 20

- Scalable synchronization library
- Ranges
- Span

C++ PIPELINE

- Mdspan
- Range-based parallel algorithms
- Extended floating-point types
- Linear algebra algorithms
- Asynchronous parallel algorithms
- Senders-receivers

How users run C++ code on GPUs today

Co-designed with V100 hardware support

N-dimensional loops and usability

Custom algorithms and async. control flow

Extended C++ interface to BLAS/Lapack

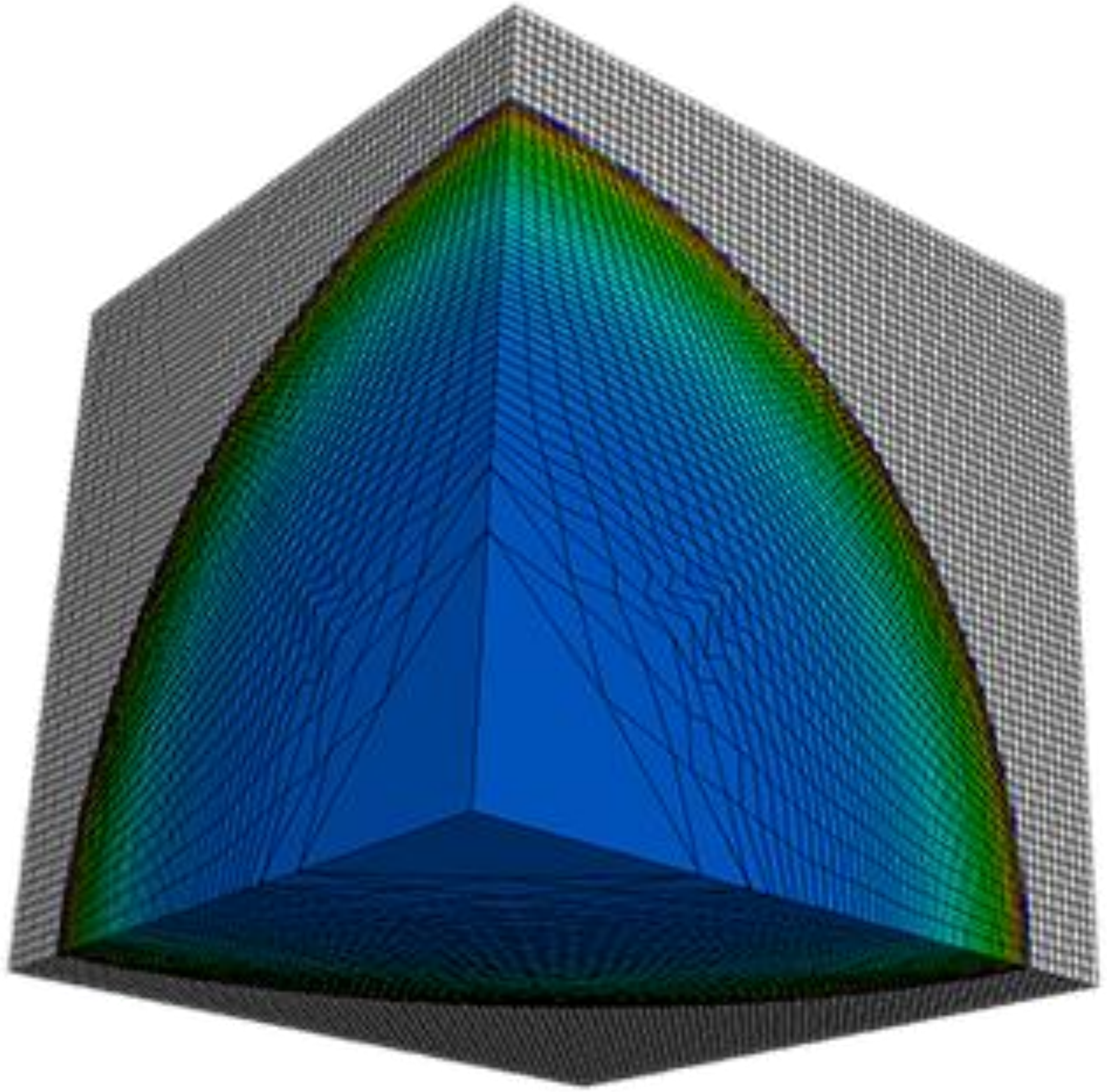
General usability of performance provided by executors

General parallelism user facing feature

C++17 PARALLEL ALGORITHMS

Lulesh Hydrodynamics Mini-app

- ~9000 lines of C++
- Parallel versions in MPI, OpenMP, OpenACC, CUDA, RAJA, Kokkos, ISO C++…
- Designed to stress compiler vectorization, parallel overheads, on-node parallelism



codesign.llnl.gov/lulesh

```
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

STANDARD C++

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

```
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++

NVIDIA

C++ STANDARD PARALLELISM

Lulesh Performance

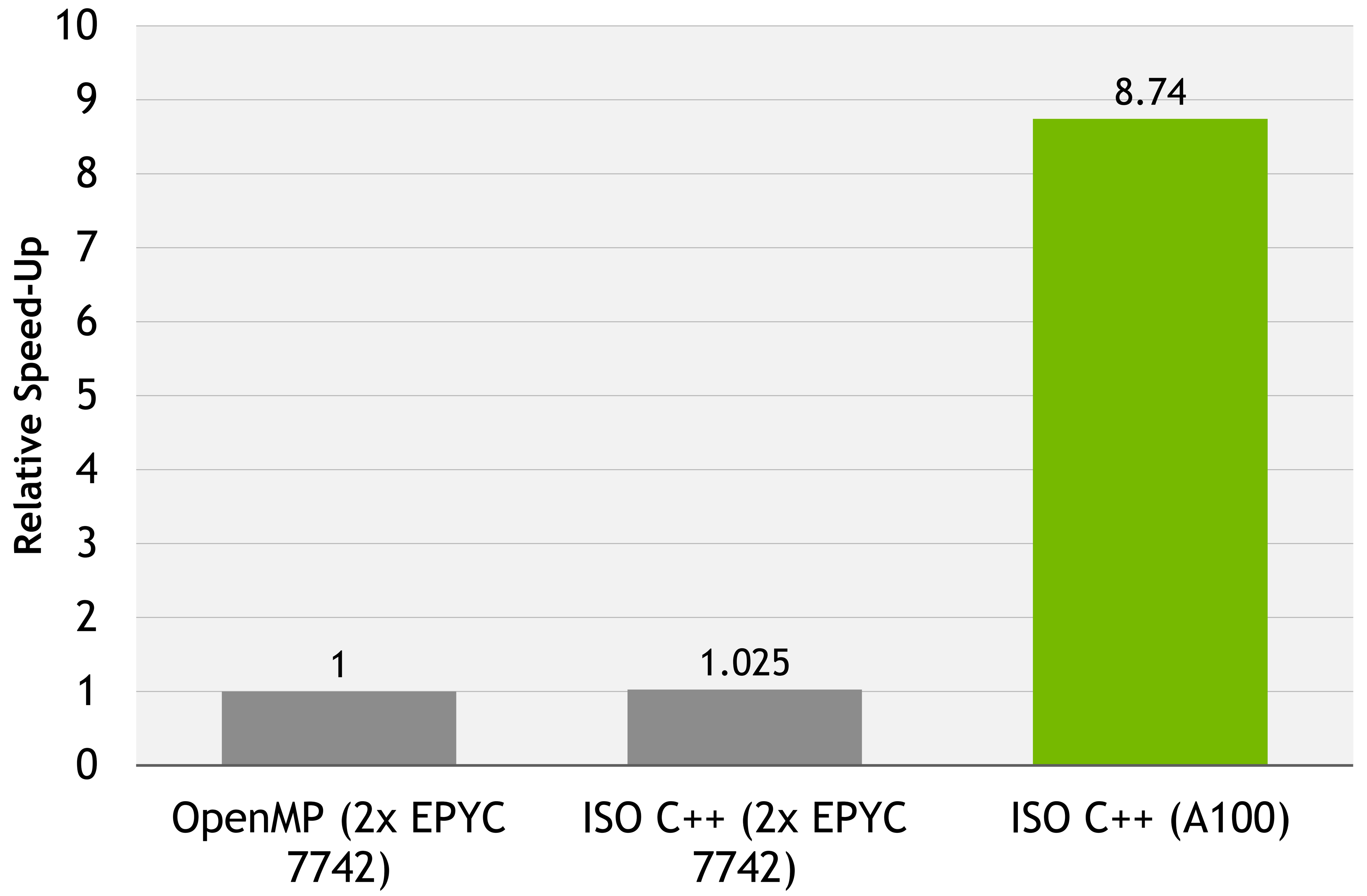
AMD EPYC 7742 CPU, NVIDIA A100 GPU. g++ version 10.3.0, icpc version 2021.5.0, nvc++ version 22.3

M-AIA WITH C++17 PARALLEL ALGORITHMS

Multi-physics simulation framework
from RWTH Aachen University

```
#pragma omp parallel // OpenMP parallel region
{
  #pragma omp for // OpenMP for loop
  for (MInt i = 0; i < noCells; i++) { // Loop over all cells
    if (timeStep % ipow2[maxLevel_ - clevel[i * distLevel]] == 0) { // Multi-grid loop
      const MInt distStartId = i * nDist; // More offsets for 1D accesses // Local offsets
      const MInt distNeighStartId = i * distNeighbors;
      const MFloat* const distributionsStart = &[distributions[distStartId];
      for (MInt j = 0; j < nDist - 1; j += 2) { // Unrolled loop distributions (factor 2)
        if (neighborId[I * distNeighbors + j] > -1) { // First unrolled iteration
          const MInt n1StartId = neighborId[distNeighStartId + j] * nDist;
          oldDistributions[n1StartId + j] = distributionsStart[j]; // 1D access AoS format
        }
        if (neighborId[I * distNeighbors + j + 1] > -1) { // Second unrolled iteration
          const MInt n2StartId = neighborId[distNeighStartId + j + 1] * nDist;
          oldDistributions[n2StartId + j + 1] = distributionsStart[j + 1];
        }
      }
      oldDistributions[distStartId + lastId] = distributionsStart[lastId]; // Zero-th distribution
    }
  }
}
```

C++ with OpenMP



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

```
std::for_each_n(par_unseq, start, noCells, [=](auto i) { // Parallel for
  if (timeStep % IPOW2[maxLevel_ - a_level(i)] != 0) // Multi-level loop
    return;
  for (MInt j = 0; j < nDist; ++j) {
    if (auto n = c_neighborId(i, j); n == -1) continue;
    a_oldDistribution(n, j) = a_distribution(i, j); // SoA or AoS mem_fn
  }
});
```

Standard C++

PARALLEL PROGRAMMING WITH ISO FORTRAN

HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

Preview support available now in NVFORTRAN

Fortran 2018

Fortran Array Intrinsics

- NVFORTRAN 20.5
- Accelerated matmul, reshape, spread, …

DO CONCURRENT

- NVFORTRAN 20.11
- Auto-offload & multi-core

Co-Arrays

- Not currently available
- Accelerated co-array images

Fortran 202x

DO CONCURRENT Reductions

- NVFORTRAN 21.11
- `REDUCE` subclause added
- Support for `+`, `*`, `MIN`, `MAX`, `IAND`, `IOR`, `IEOR`.
- Support for `.AND.`, `.OR.`, `.EQV.`, `.NEQV` on `LOGICAL` values

MINIWEATHER

Standard Language Parallelism in Climate/Weather Applications

MiniWeather

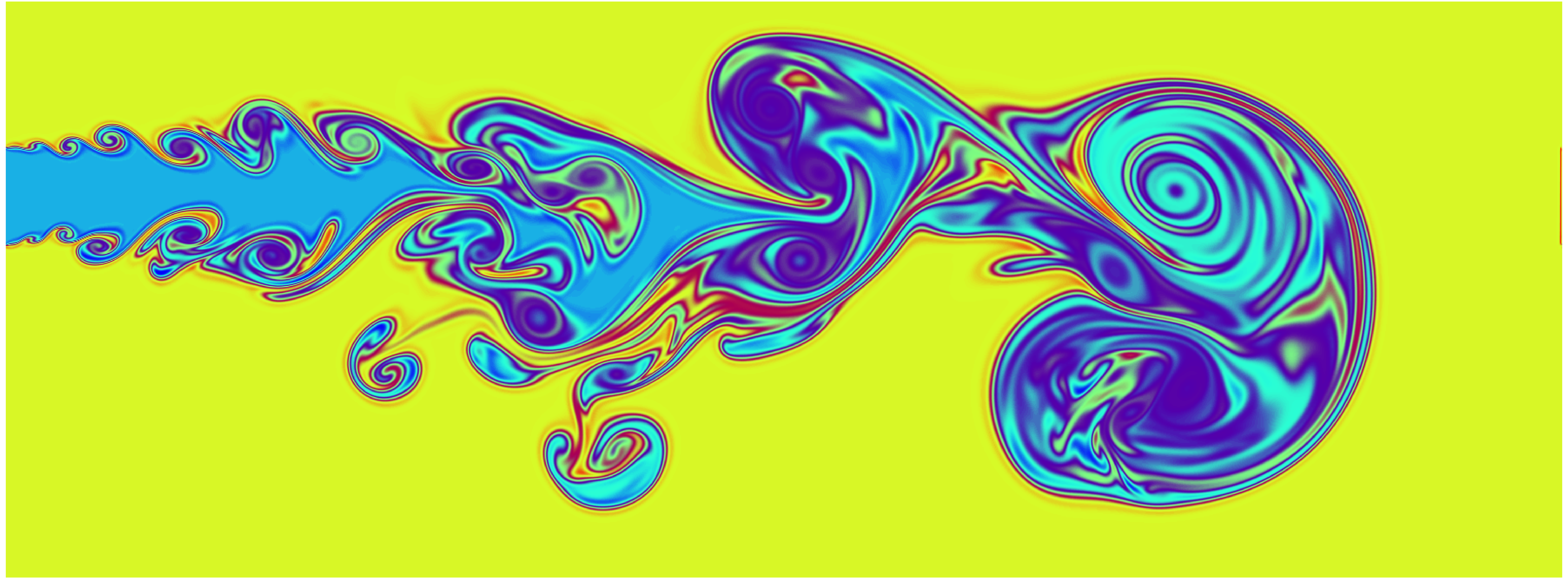
Mini-App written in C++ and Fortran that simulates weather-like fluid flows using Finite Volume and Runge-Kutta methods.

Existing parallelization in MPI, OpenMP, OpenACC, …

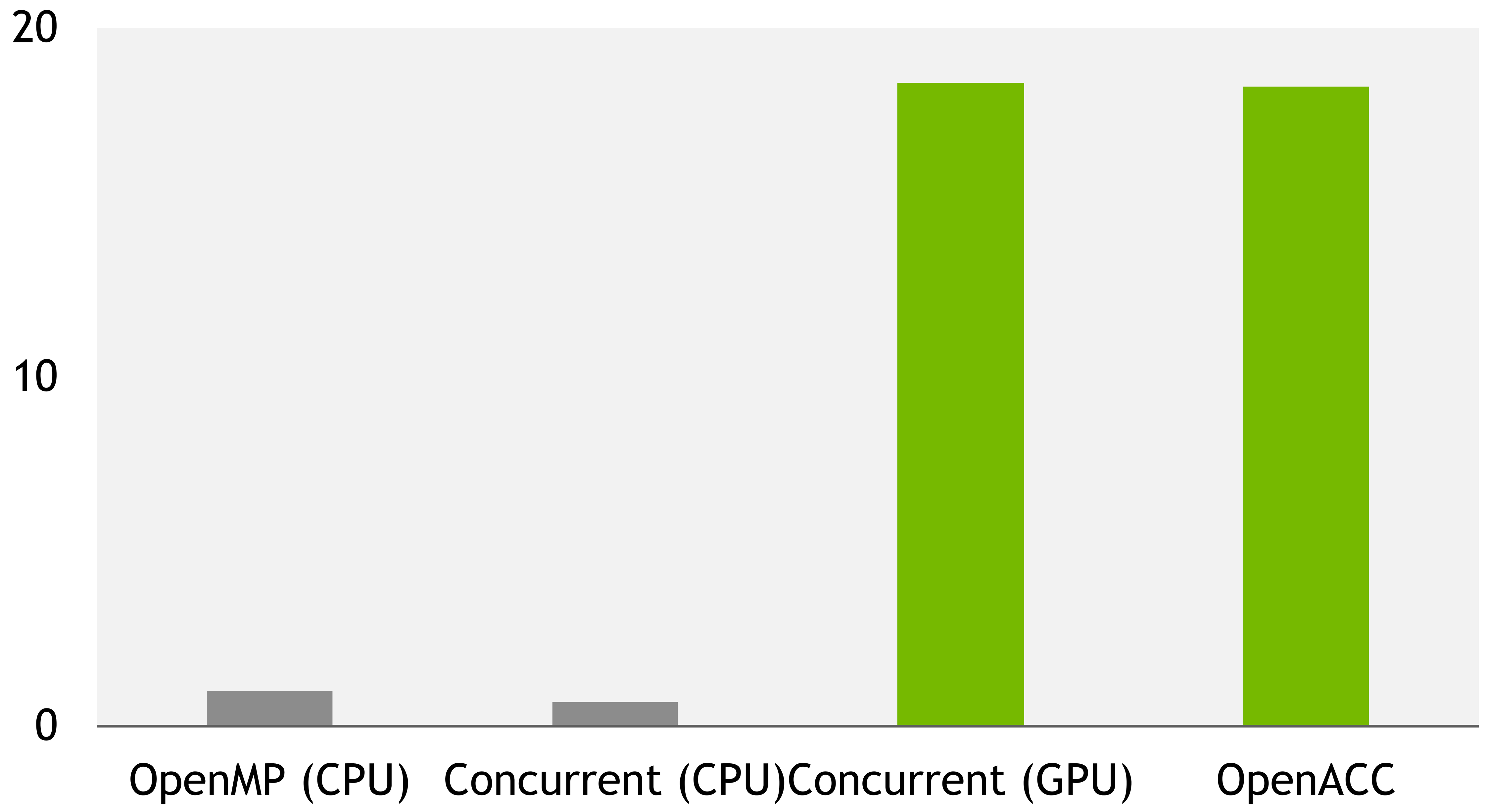
Included in the SPEChpc benchmark suite*

Open-source and commonly-used in training events.

https://github.com/mrnorman/miniWeather/



```
do concurrent (ll=1:NUM_VARS, k=1:nz, i=1:nx)
      local(x,z,x0,z0,xrad,zrad,amp,dist,wpert)

  if (data_spec_int == DATA_SPEC_GRAVITY_WAVES) then
    x = (i_beg-1 + i-0.5_rp) * dx
    z = (k_beg-1 + k-0.5_rp) * dz
        x0 = xlen/8
    z0 = 1000
    xrad = 500
    zrad = 500
    amp = 0.01_rp
    dist = sqrt( ((x-x0)/xrad)**2 + ((z-z0)/zrad)**2 )
         * pi / 2._rp
    if (dist <= pi / 2._rp) then
      wpert = amp * cos(dist)**2
    else
      wpert = 0._rp
    endif
    tend(i,k,ID_WMOM) = tend(i,k,ID_WMOM)
                      + wpert*hy_dens_cell(k)
  endif
  state_out(i,k,ll) = state_init(i,k,ll)
                    + dt * tend(i,k,ll)

enddo
```



*SPEChpc is a trademark of The Standard Performance Evaluation Corporation

Source: HPC SDK 22.1, AMD EPYC 7742, NVIDIA A100. MiniWeather: NX=2000, NZ=1000, SIM_TIME=5. OpenACC version uses -gpu=managed option.

NVIDIA

POT3D: DO CONCURRENT + LIMITED OPENACC

POT3D

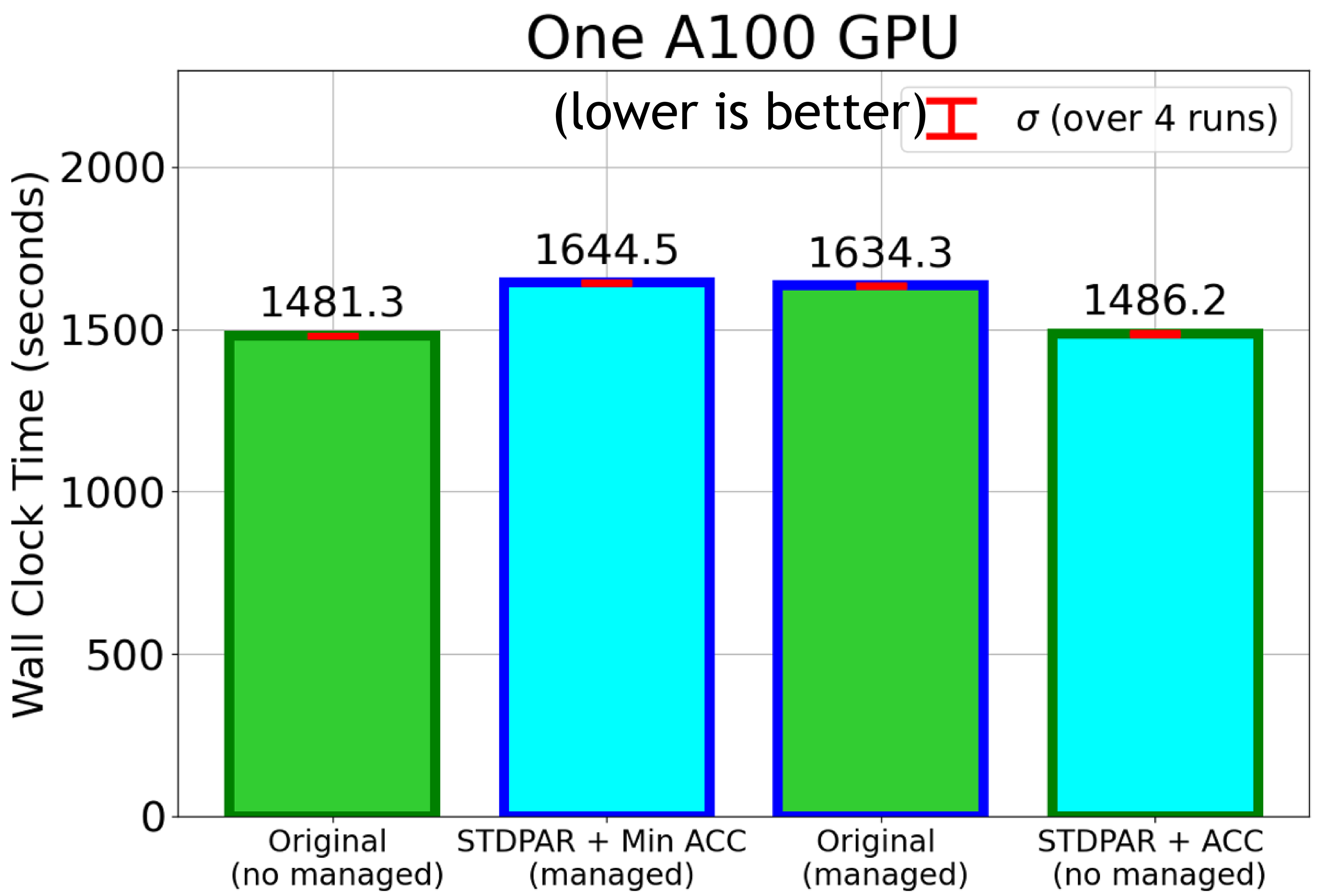
POT3D is a Fortran application for approximating solar coronal magnetic fields.

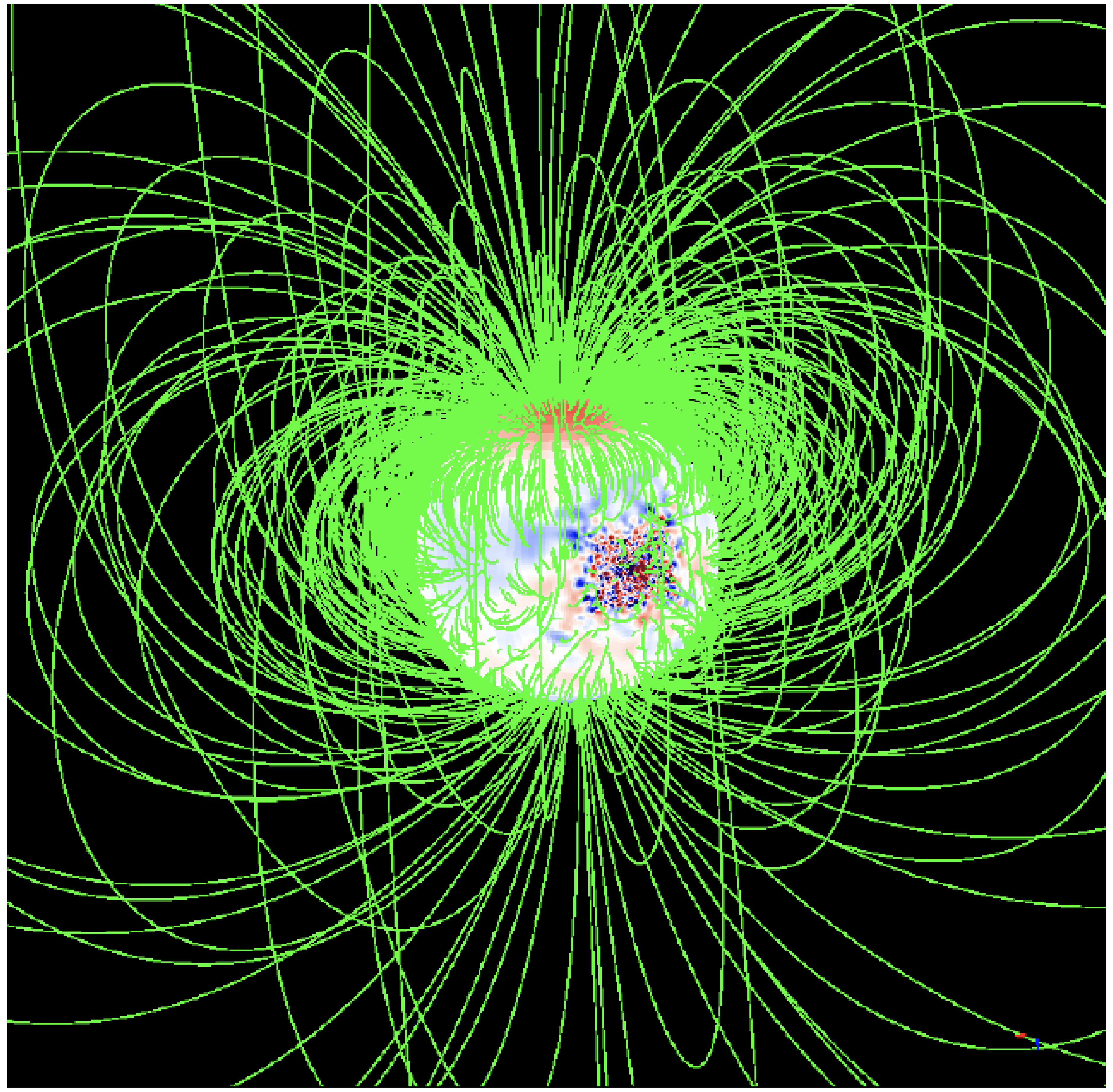
Included in the SPEChpc benchmark suite*

Existing parallelization in MPI & OpenACC

Optimized the DO CONCURRENT version by using OpenACC solely for data motion and atomics

https://github.com/predsci/POT3D





```
!$acc enter data copyin(phi,dr_i)
!$acc enter data create(br)
do concurrent (k=1:np,j=1:nt,i=1:nrm1)
  br(i,j,k)=(phi(i+1,j,k)-phi(i,j,k ))*dr_i(i)
enddo
!$acc exit data delete(phi,dr_i,br)
```

Data courtesy of Predictive Science Inc.

*SPEChpc is a trademark of The Standard Performance Evaluation Corporation

ACCELERATED PROGRAMMING IN ISO FORTRAN

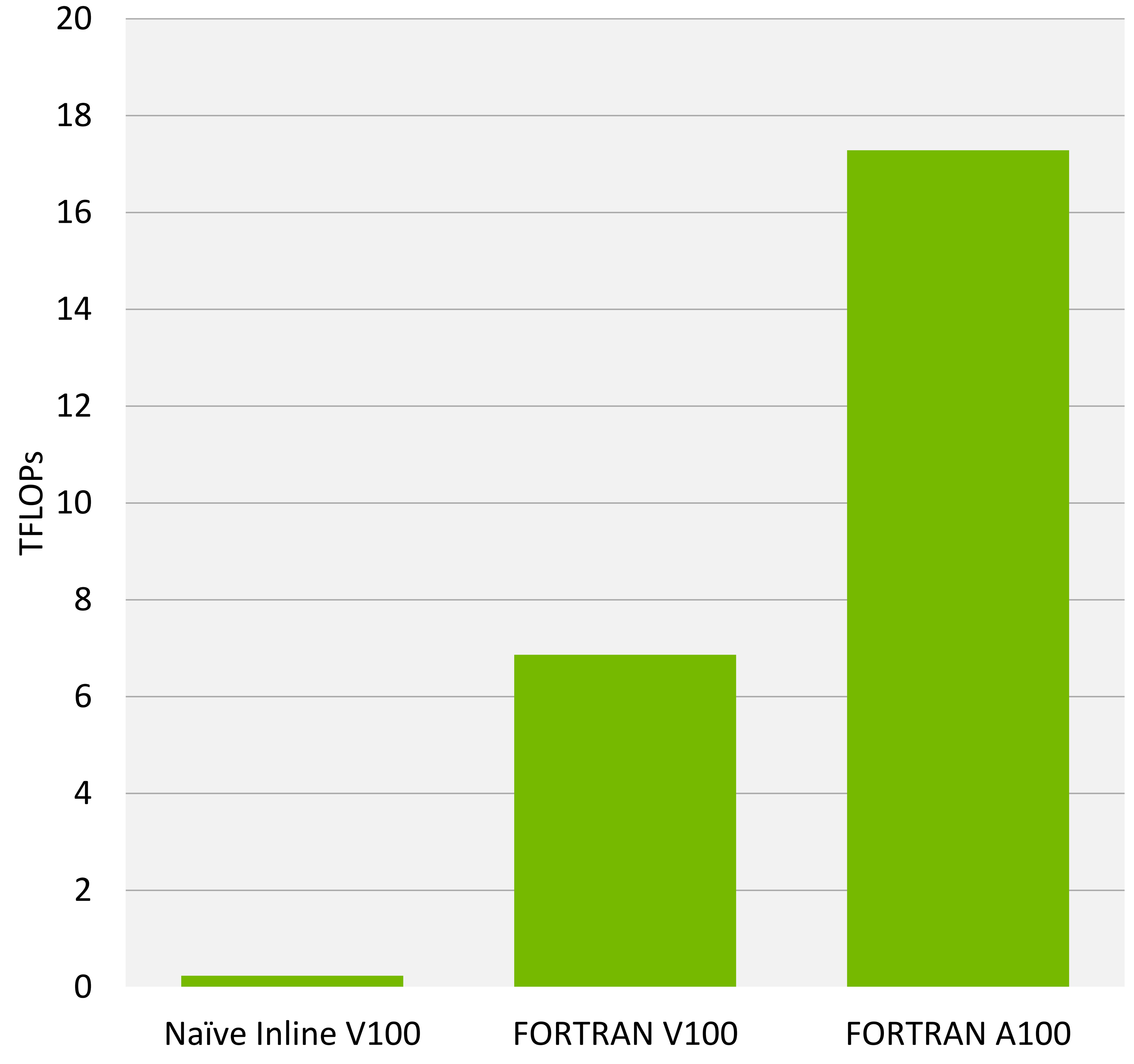
NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c
...
!$acc enter data copyin(a,b,c) create(d)

do nt = 1, ntimes
  !$acc kernels
  do j = 1, nj
    do i = 1, ni
      d(i,j) = c(i,j)
      do k = 1, nk
        d(i,j) = d(i,j) + a(i,k) * b(k,j)
      end do
    end do
  end do
   !$acc end kernels
end do
!$acc exit data copyout(d)
```

Inline FP64 matrix multiply

```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c

...

do nt = 1, ntimes
  d = c + matmul(a,b)
end do
```

MATMUL FP64 matrix multiply



HPC PROGRAMMING IN ISO FORTRAN

Examples of Patterns Accelerated in NVFORTRAN

```
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(transpose(b))
d = 2.5 * ceil(transpose(a)) + 3.0 * abs(b)
d = reshape(a,shape=[ni,nj,nk])
d = reshape(a,shape=[ni,nk,nj])
d = 2.5 * sqrt(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = alpha * conjg(reshape(a,shape=[ni,nk,nj],order=[1,3,2]))
d = reshape(a,shape=[ni,nk,nj],order=[1,3,2])
d = reshape(a,shape=[nk,ni,nj],order=[2,3,1])
d = reshape(a,shape=[ni*nj,nk])
d = reshape(a,shape=[nk,ni*nj],order=[2,1])
d = reshape(a,shape=[64,2,16,16,64],order=[5,2,3,4,1])
d = abs(reshape(a,shape=[64,2,16,16,64],order=[5,2,3,4,1]))
c = matmul(a,b)
c = matmul(transpose(a),b)
c = matmul(reshape(a,shape=[m,k],order=[2,1]),b)
c = matmul(a,transpose(b))
c = matmul(a,reshape(b,shape=[k,n],order=[2,1]))
```

```
c = matmul(transpose(a),transpose(b))
c = matmul(transpose(a),reshape(b,shape=[k,n],order=[2,1]))
d = spread(a,dim=3,ncopies=nk)
d = spread(a,dim=1,ncopies=ni)
d = spread(a,dim=2,ncopies=nx)
d = alpha * abs(spread(a,dim=2,ncopies=nx))
d = alpha * spread(a,dim=2,ncopies=nx)
d = abs(spread(a,dim=2,ncopies=nx))
d = transpose(a)
d = alpha * transpose(a)
d = alpha * ceil(transpose(a))
d = alpha * conjg(transpose(a))
c = c + matmul(a,b)
c = c - matmul(a,b)
c = c + alpha * matmul(a,b)
d = alpha * matmul(a,b) + c
d = alpha * matmul(a,b) + beta * c
```

AGENDA

Accelerated Computing with Standard Languages

GPU Supercomputing in the PyData Ecosystem

Advancements in HPC Libraries

NVIDIA Developer Tools

BRINGING GPU SUPERCOMPUTING TO PYDATA ECOSYSTEM


```
import numpy as np

a = np.random.randn(16).reshape(4, 4)
b = a + a.T
b
```

```
import dask.array as da
import numpy as np

a = da.from_array(
    np.random.randn(160_000).reshape(400, 400),
    chunks=(100, 100))
b = a + a.T
b.compute()
```

```
import dask.array as da
import cupy as cp

a = da.from_array(
    cp.random.randn(160_000).reshape(400, 400),
    chunks=(100, 100),
    asarray=False)
b = a + a.T
b.compute()
```

```
import cunumeric as np

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

PRODUCTIVITY

Sequential and Composable Code

```
def cg_solve(A, b, conv_iters):
    x = np.zeros_like(b)
    r = b - A.dot(x)
    p = r
    rsold = r.dot(r)
    converged = False
    max_iters = b.shape[0]

    for i in range(max_iters):
        Ap = A.dot(p)
        alpha = rsold / (p.dot(Ap))
        x = x + alpha * p
        r = r - alpha * Ap
        rsnew = r.dot(r)

        if i % conv_iters == 0 and \
            np.sqrt(rsnew) < 1e-10:
            converged = i
            break

        beta = rsnew / rsold
        p = r + beta * p
        rsold = rsnew
```

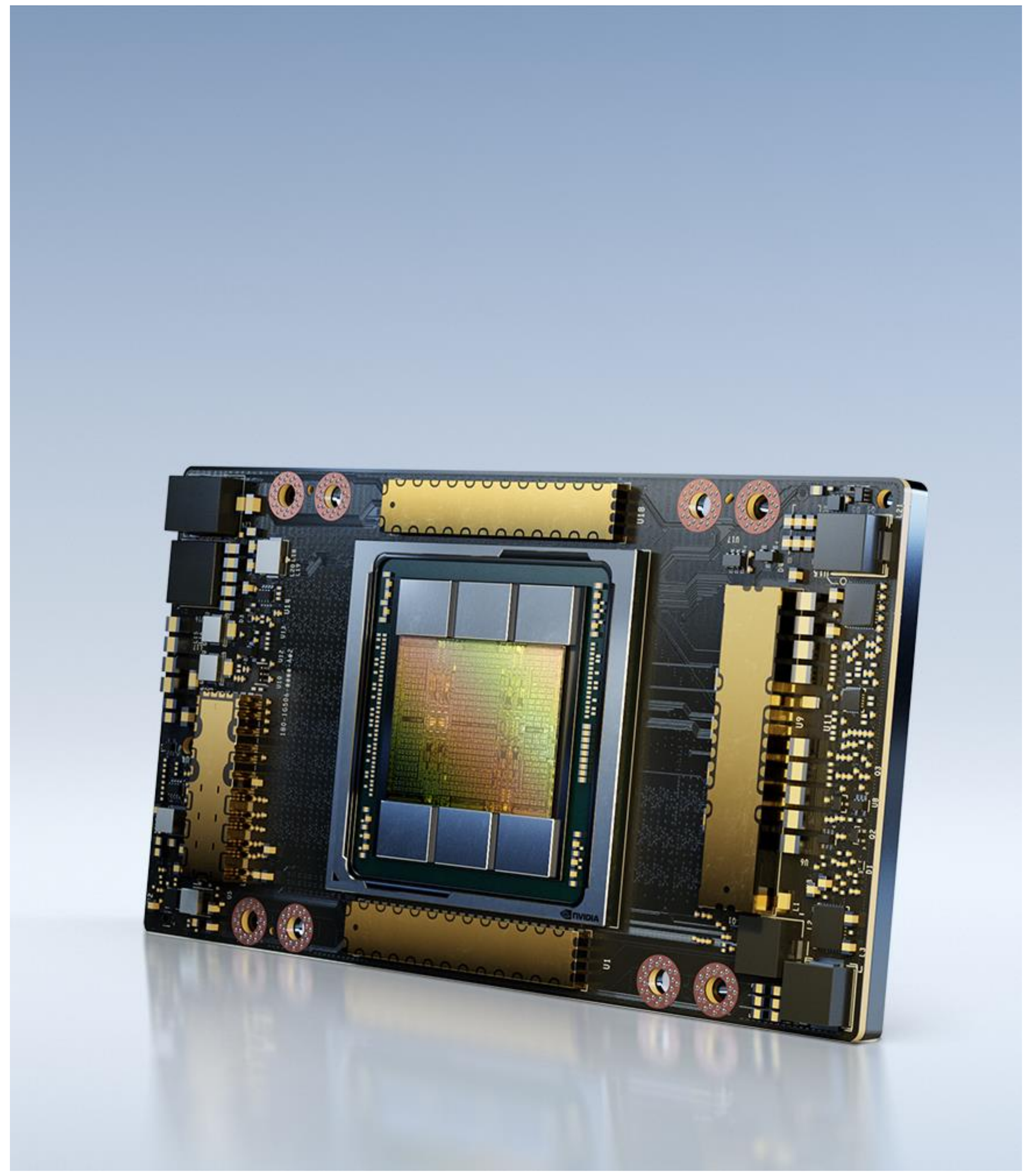
- Sequential semantics - no visible parallelism or synchronization
- Name-based global data – no partitioning
- Composable – can combine with other libraries and datatypes

PERFORMANCE

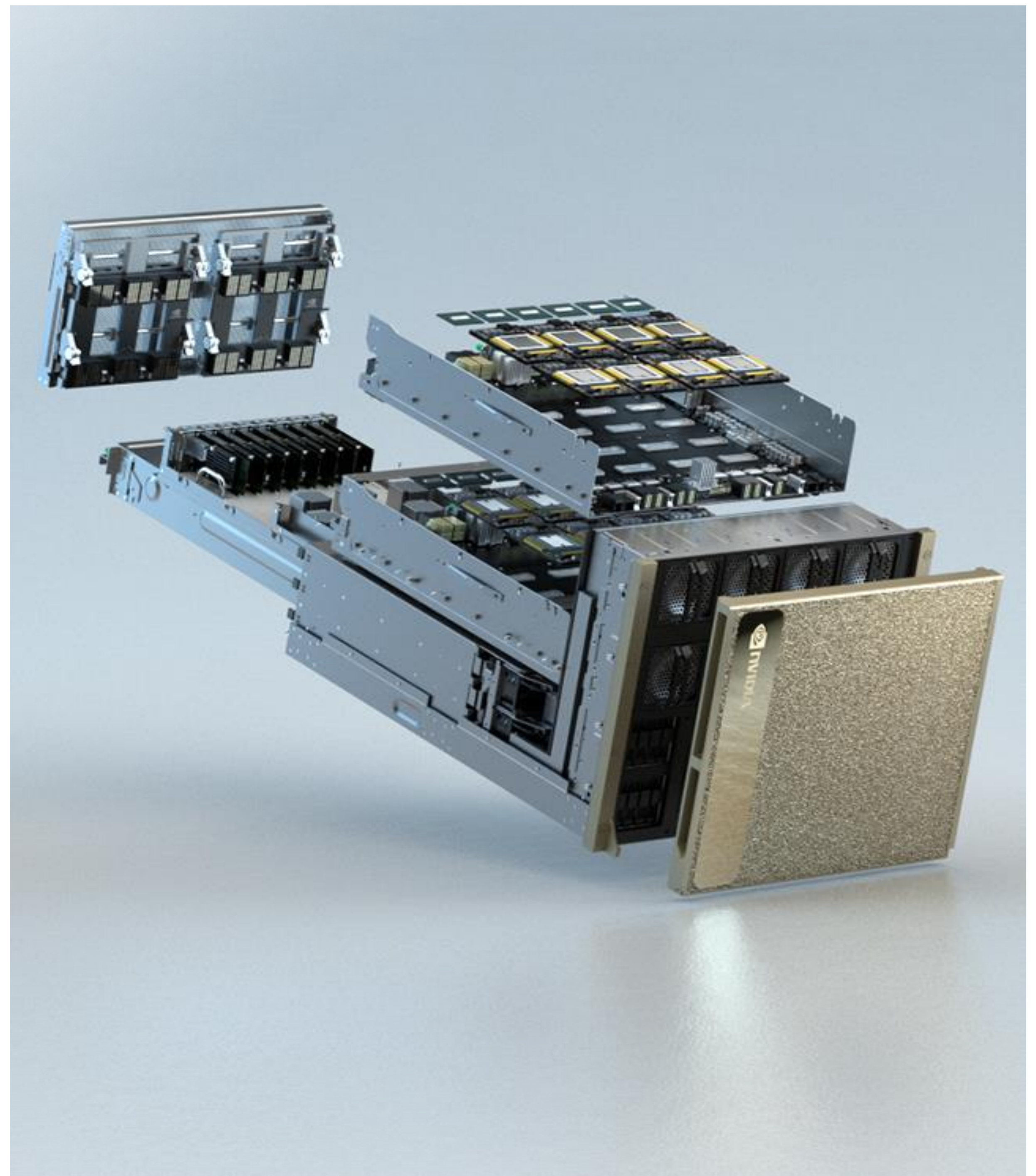
Transparent Acceleration

- Transparently run at any scale needed to address computational challenges at hand
- Automatically leverage all the available hardware

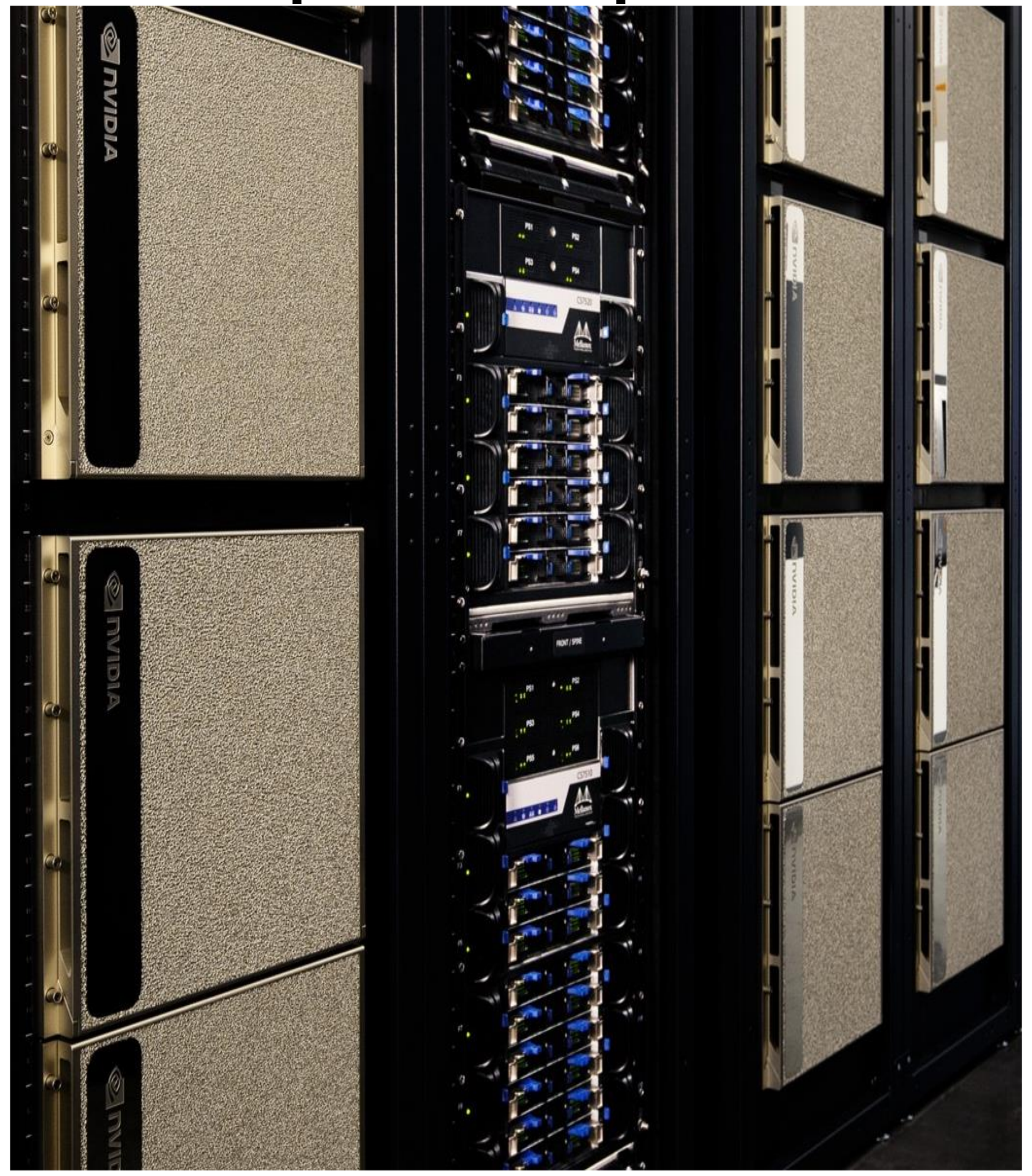
GPU



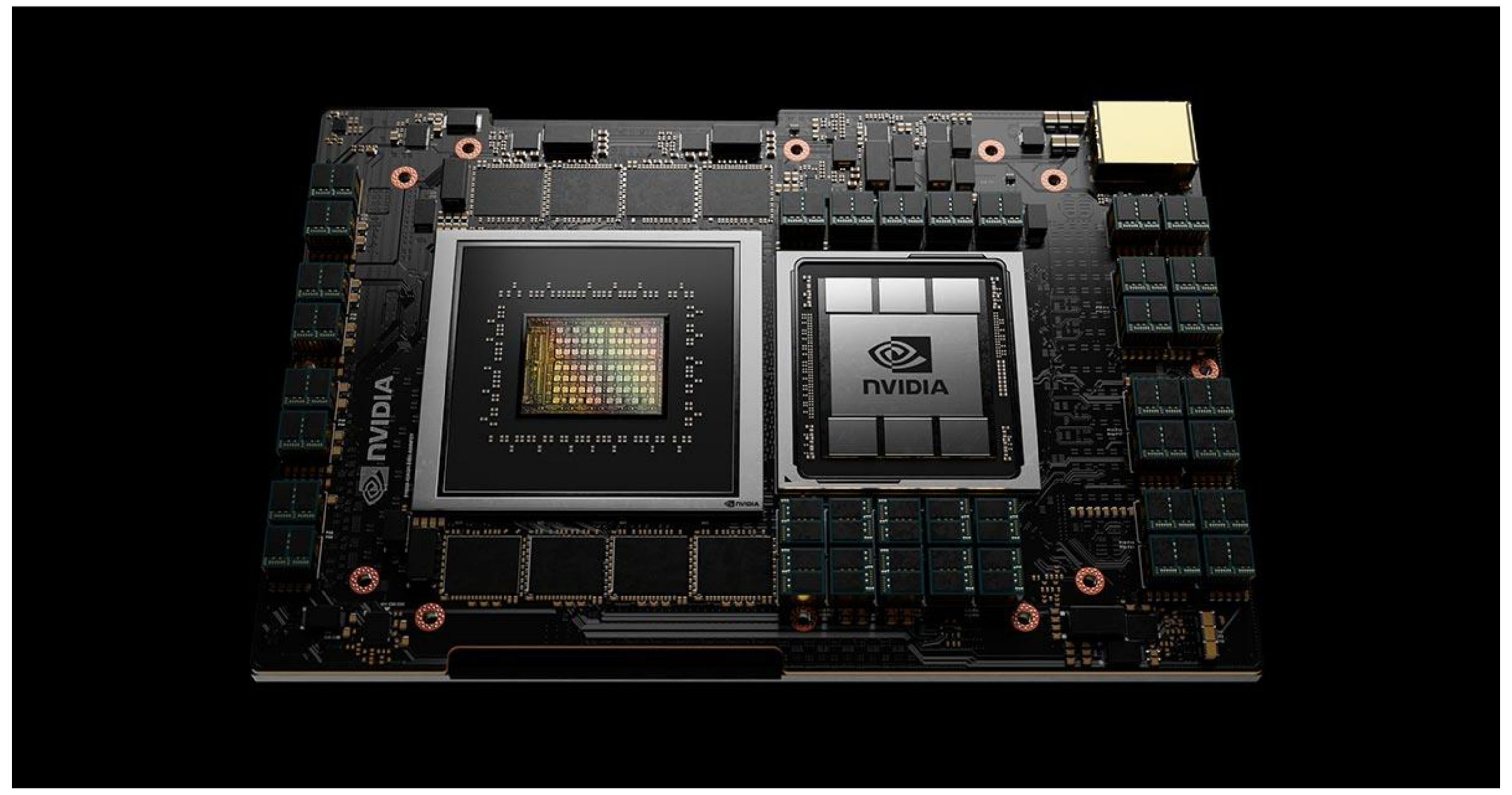
Multi-GPU



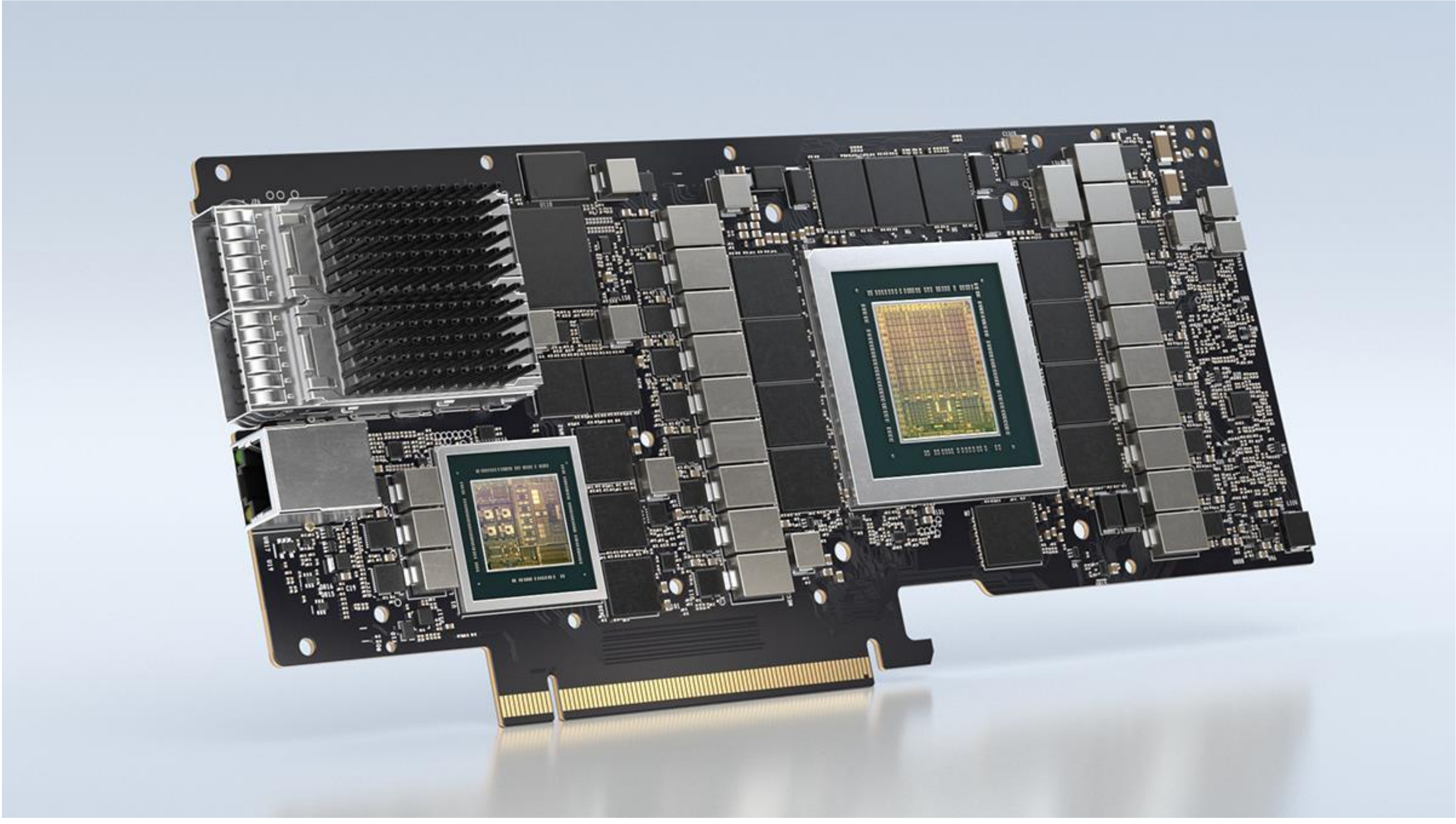
Supercomputer



Grace CPU

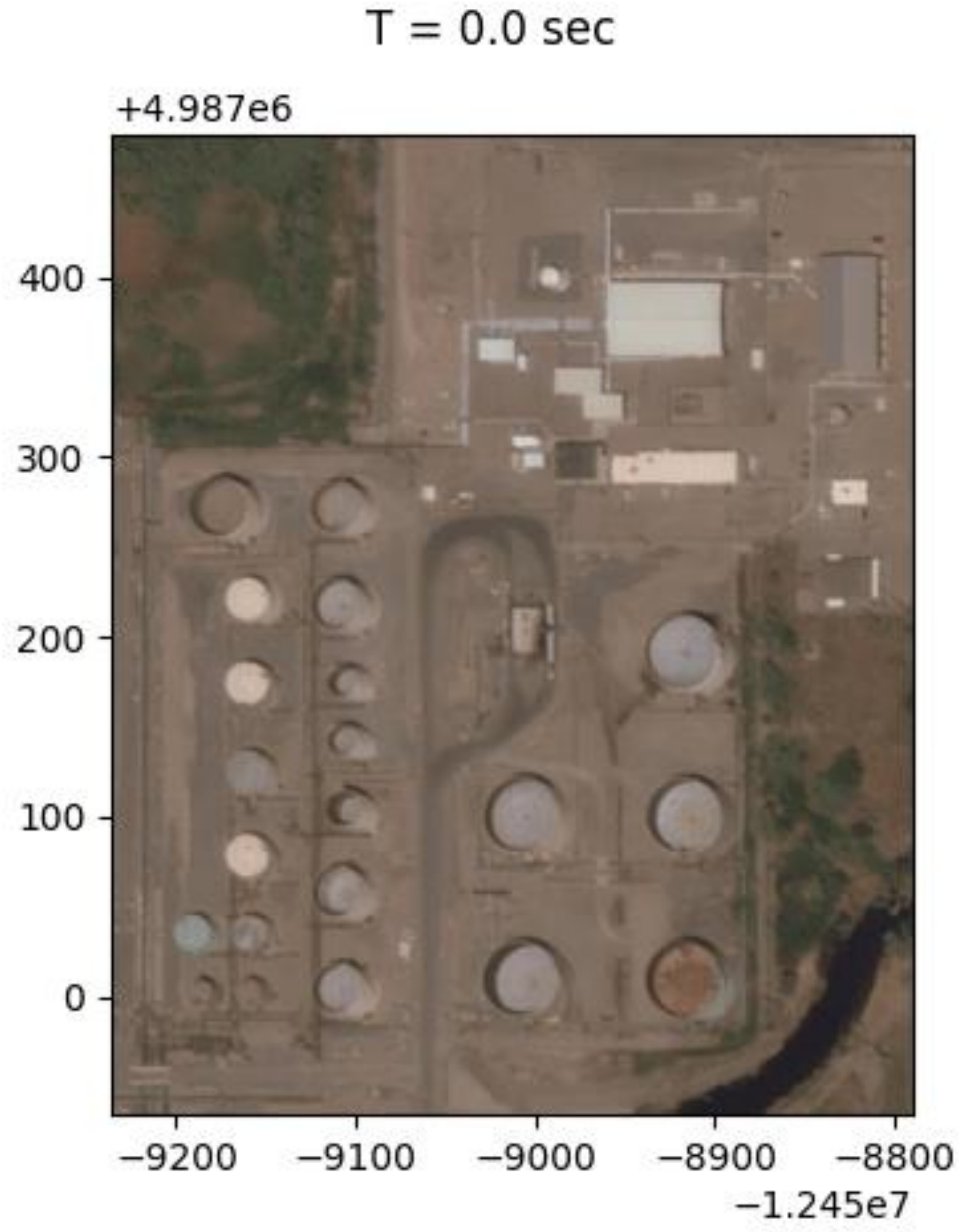


DPU



COMPUTATIONAL FLUID DYNAMICS

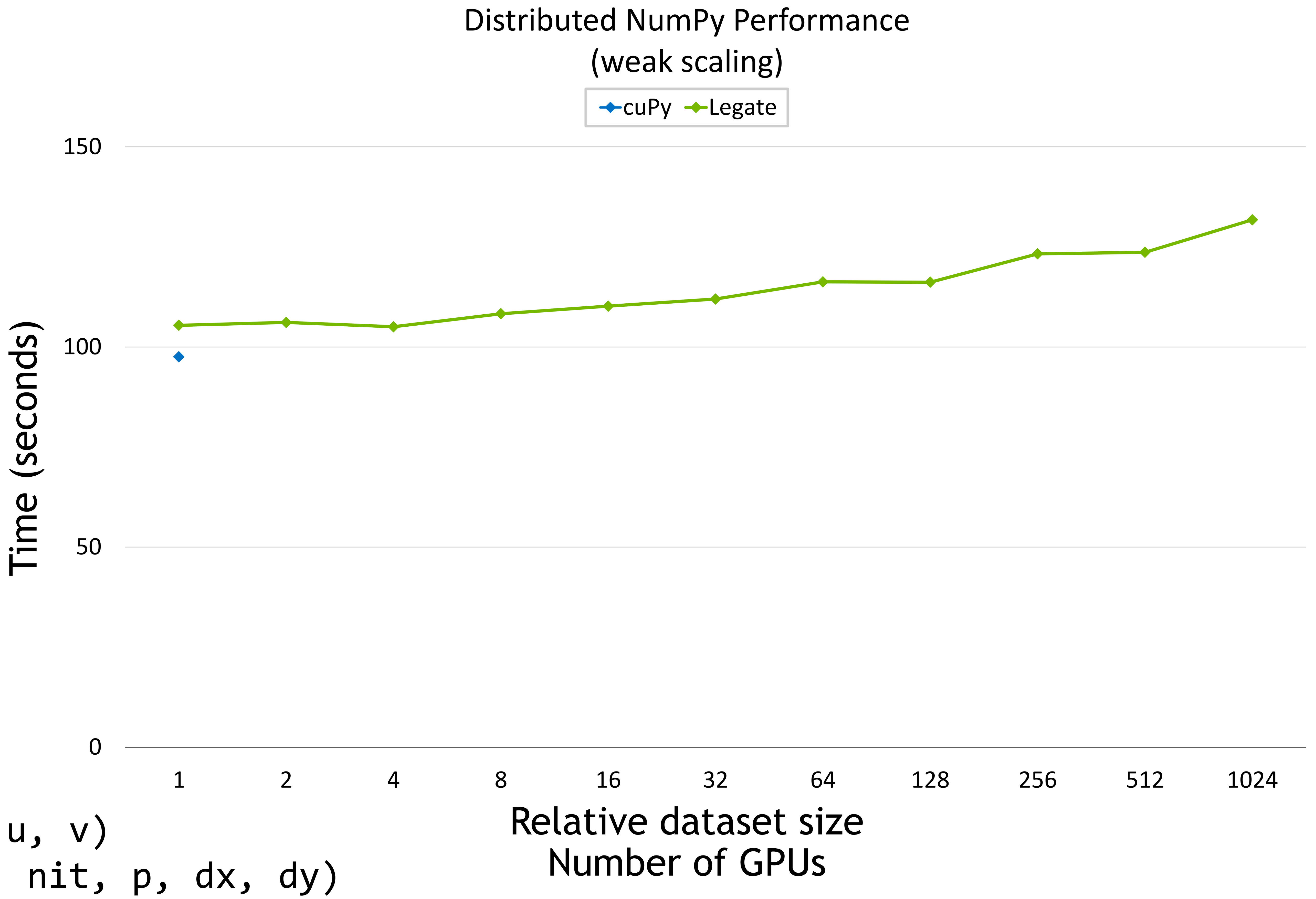
- CFD codes like:
  - Shallow-Water Equation Solver
- Oil Pipeline Risk Management: Geoclaw-landspill simulations
- Python Libraries: Jupyter, NumPy, SciPy, SymPy, Matplotlib




```
for _ in range(iter):
    un = u.copy()

    vn = v.copy()
    b = build_up_b(rho, dt, dx, dy, u, v)
    p = pressure_poisson_periodic(b, nit, p, dx, dy)
```

…

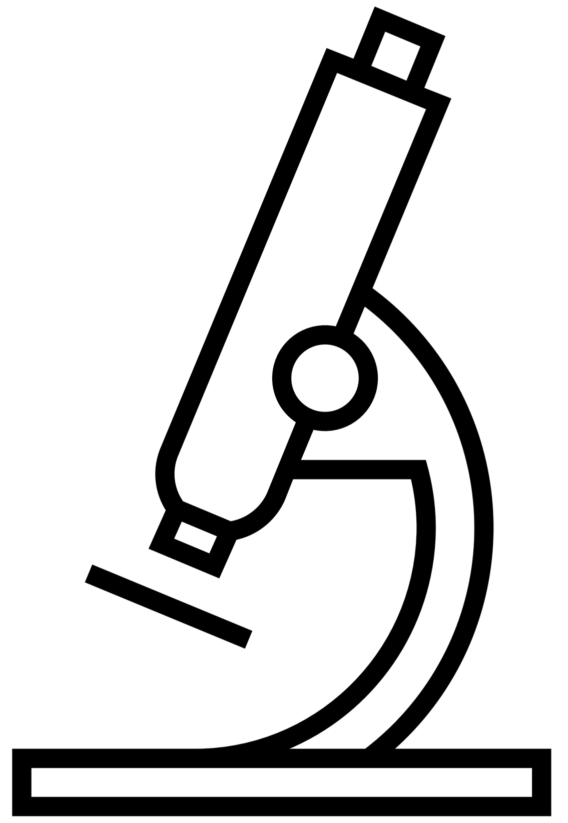
CFD Python on cuNumeric!



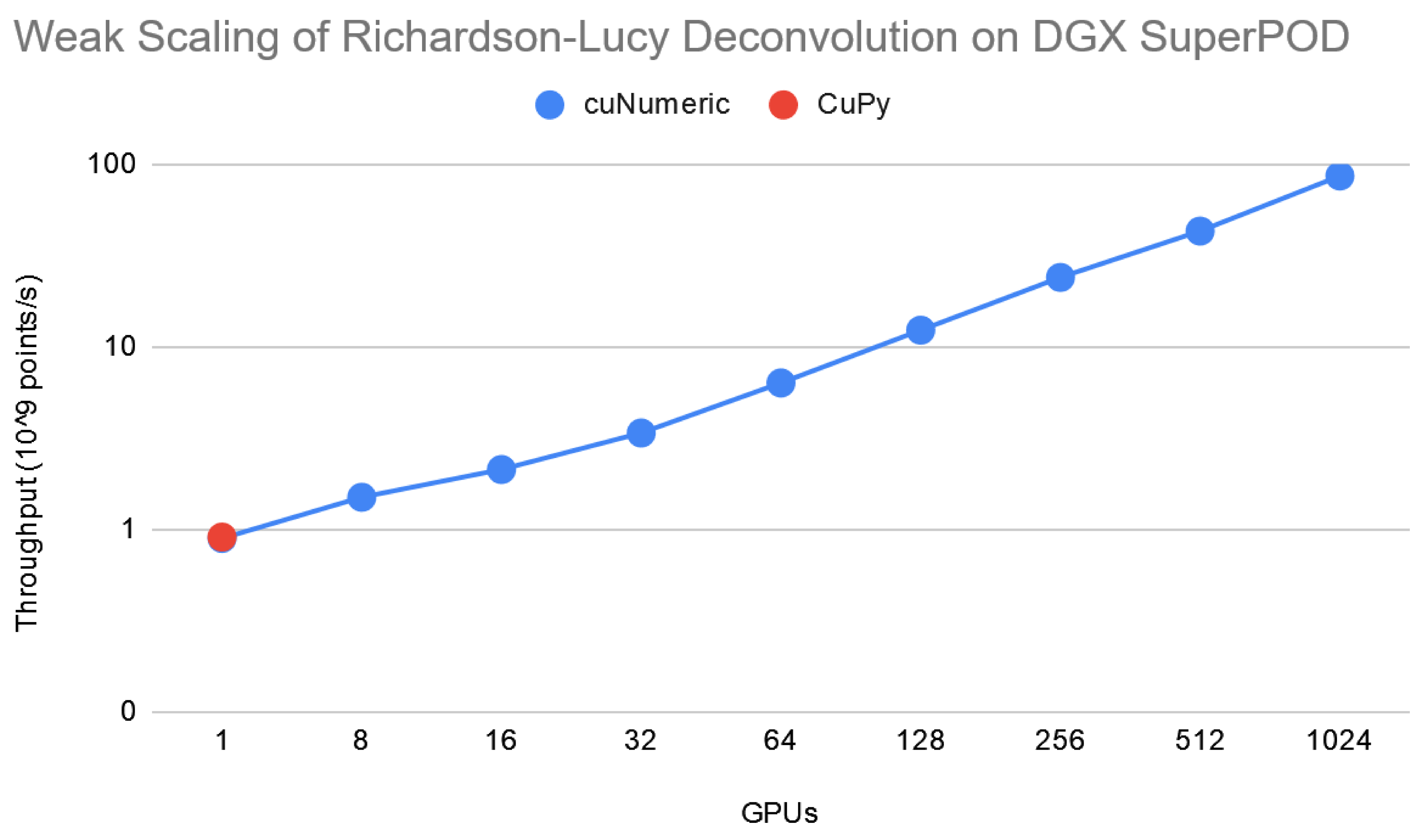
Extracted from “CFD Python” course at https://github.com/barbagroup/CFDPython
Barba, Lorena A., and Forsyth, Gilbert F. (2018). CFD Python: the 12 steps to Navier-Stokes equations. *Journal of Open Source Education*, **1**(9), 21, https://doi.org/10.21105/jose.00021

NVIDIA

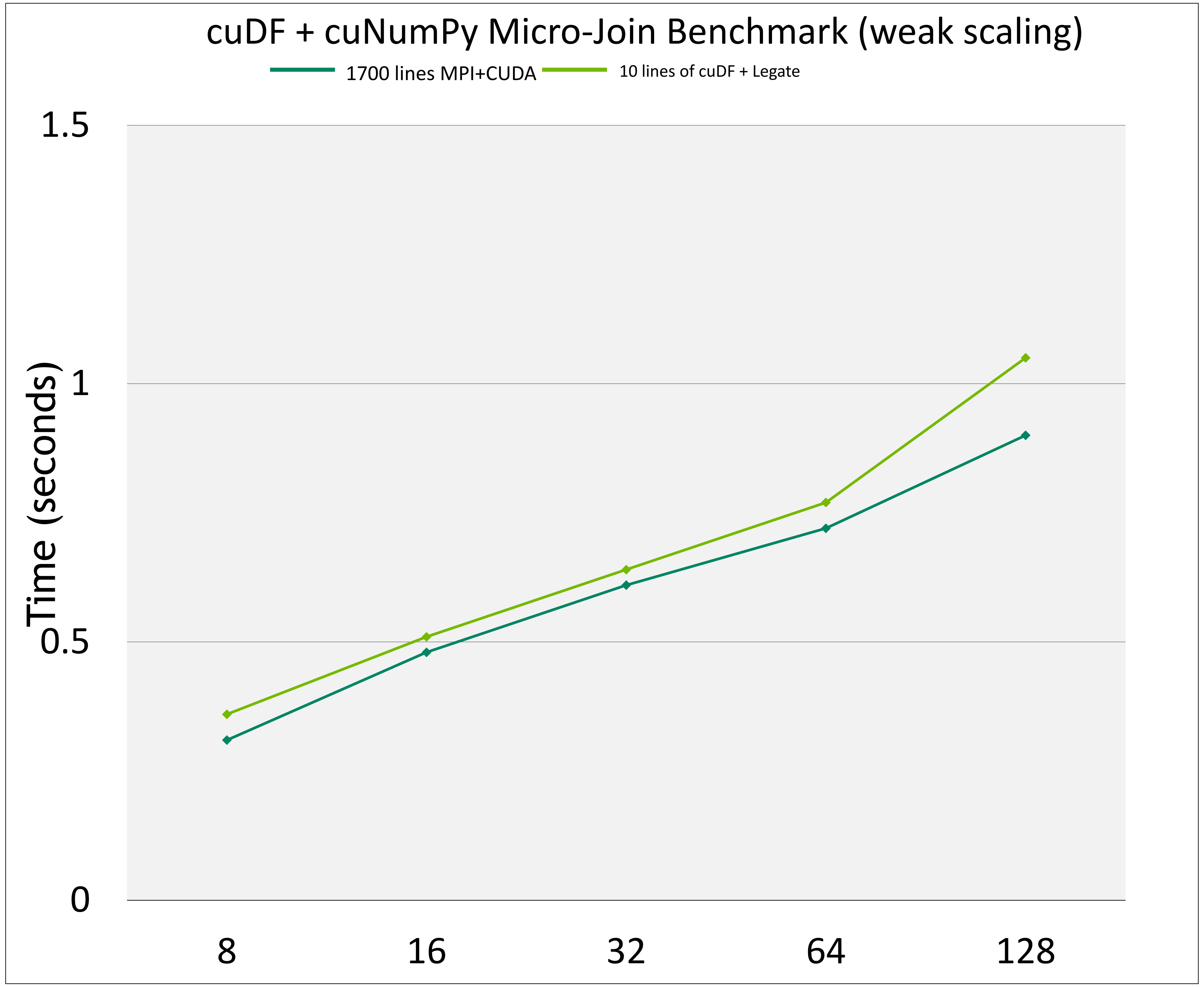
MICROSCOPY WITH RICHARDSON-LUCY DECONVOLUTION



```
def richardson_lucy(image, psf, num_iter=50,
                    clip=True, filter_epsilon=None):
  float_type = _supported_float_type(image.dtype)
  image = image.astype(float_type, copy=False)
  psf = psf.astype(float_type, copy=False)
  im_deconv = np.full(image.shape, 0.5, dtype=float_type)
  psf_mirror = np.flip(psf)

  for _ in range(num_iter):
    conv = convolve(im_deconv, psf, mode='same')
      if filter_epsilon:
        with np.errstate(invalid='ignore'):
          relative_blur = np.where(conv < filter_epsilon, 0,
                                   image / conv)
      else:
        relative_blur = image / conv
      im_deconv *= convolve(relative_blur, psf_mirror,
                            mode='same')

   if clip:
      im_deconv[im_deconv > 1] = 1
      im_deconv[im_deconv < -1] = -1

  return im_deconv
```

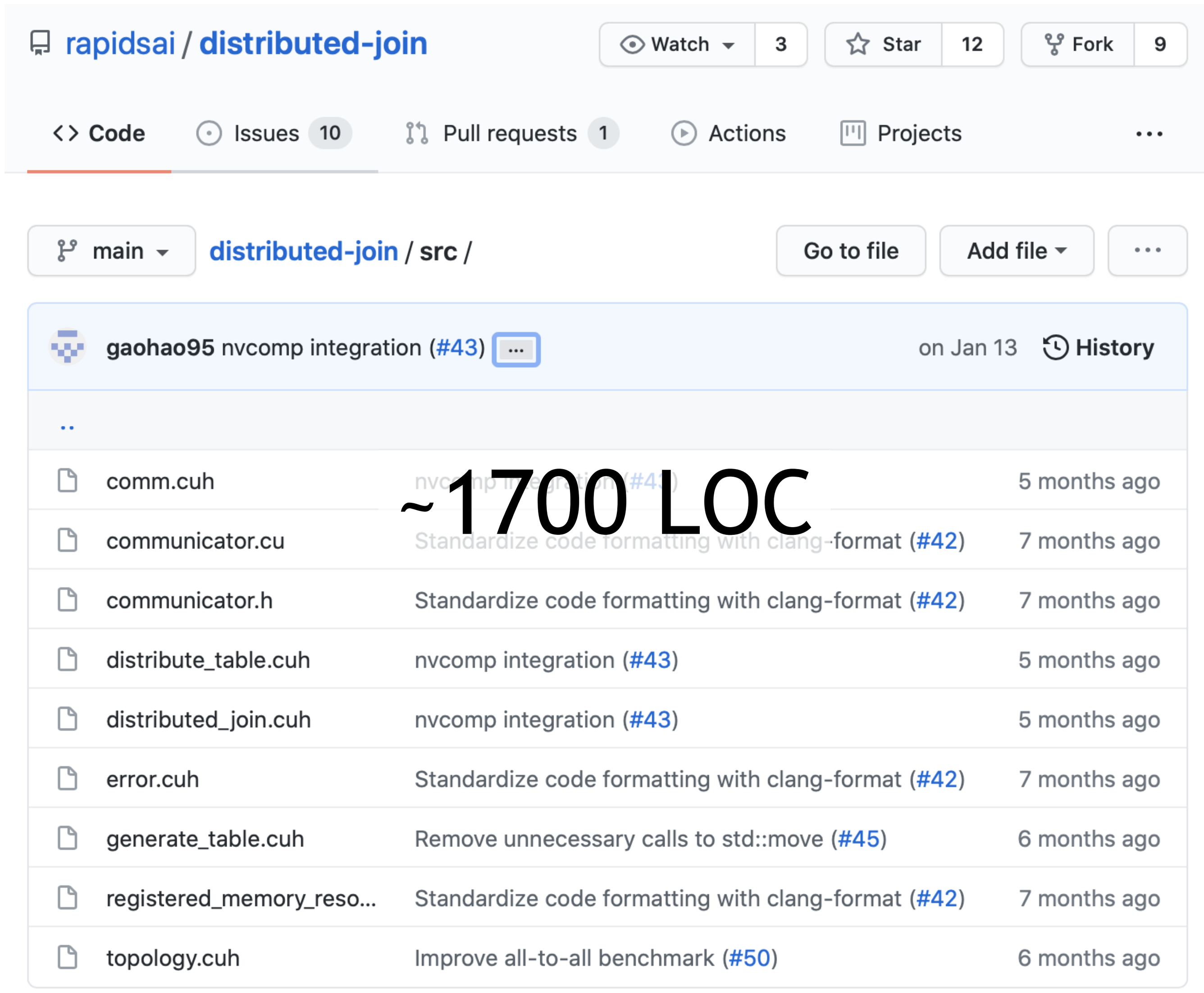


MICRO-JOIN




```
size = num_rows_per_gpu * num_gpus

key_l = np.arange(size)
val_l = np.random.randn(size)
lhs = pd.DataFrame({ "key": key_l, ”val": val_l })

key_r = key_l // 3 * 3     # selectivity: 0.33
payload_r = np.random.randn(size)
rhs = pd.DataFrame({ "key": key_r, ”val": val_r })

out = lhs.merge(rhs, on="key")
```

vs.



Machine: DGX SuperPOD with A100-80GB GPUs

AGENDA

Accelerated Computing with Standard Languages

GPU Supercomputing in the PyData Ecosystem

Advancements in HPC Libraries

NVIDIA Developer Tools

cuBLAS

GPU Optimized BLAS Implementation

Full BLAS implementation + extensions

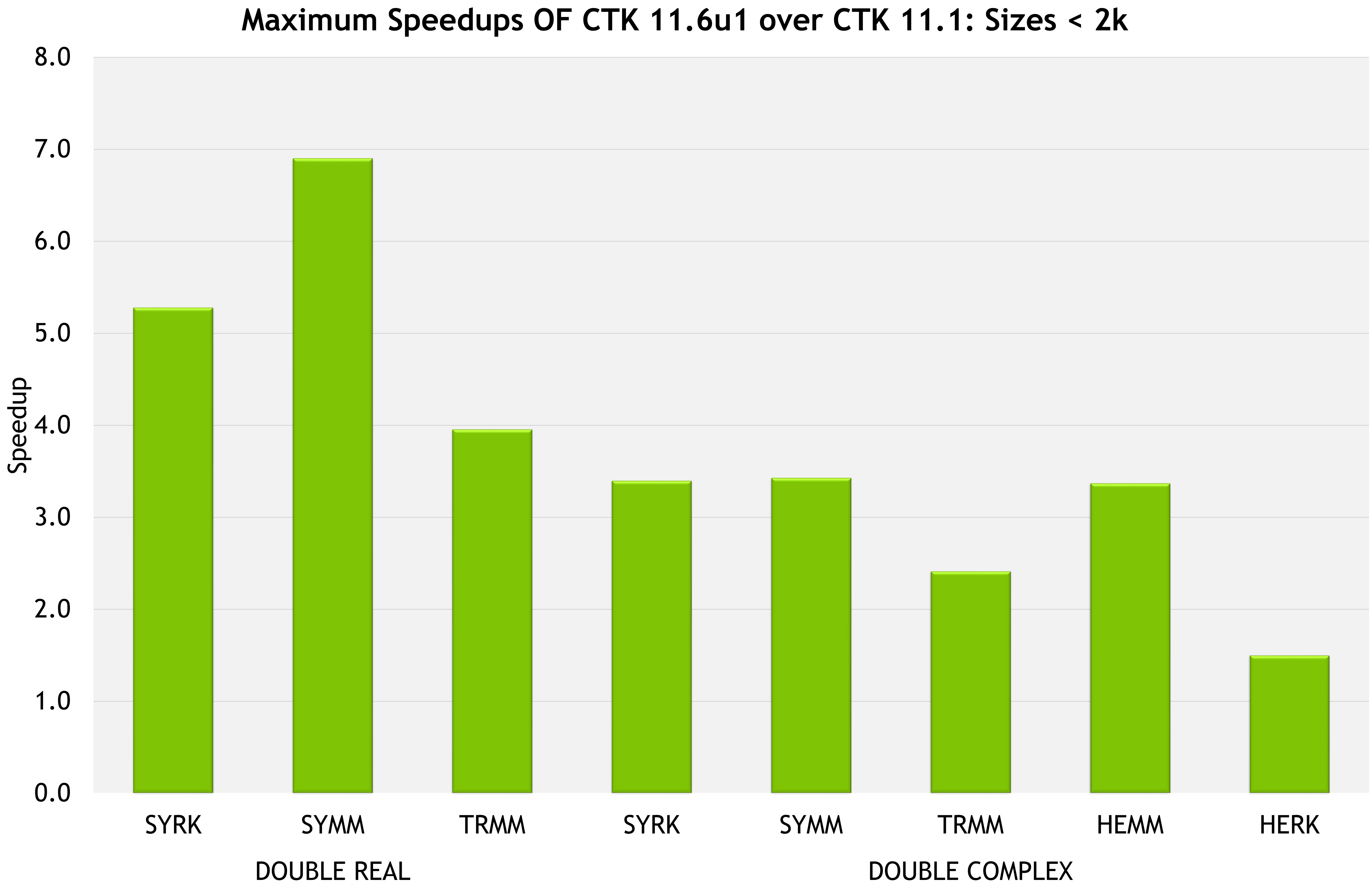
- Vector Vector / Matrix Vector / Matrix Matrix
- Mixed Precision / Multiple GPUs / Batched APIs

Accelerating a wide range of applications

- HPC & Scientific Computing
- Data Analytics & Deep Learning

Recently Introduced

- Improved heuristics (cache)
- Improved FP64 SYRK, TRMM, SYMM
- Batched GEMV Extensions
- Helper functions for improved error management



* A100 80GB @ 1095 MHz: CTK 11.1 vs. CTK 11.6U1

cuSOLVER

GPU Optimized Factorizations & Solvers

Dense and Sparse Factorizations & Solvers

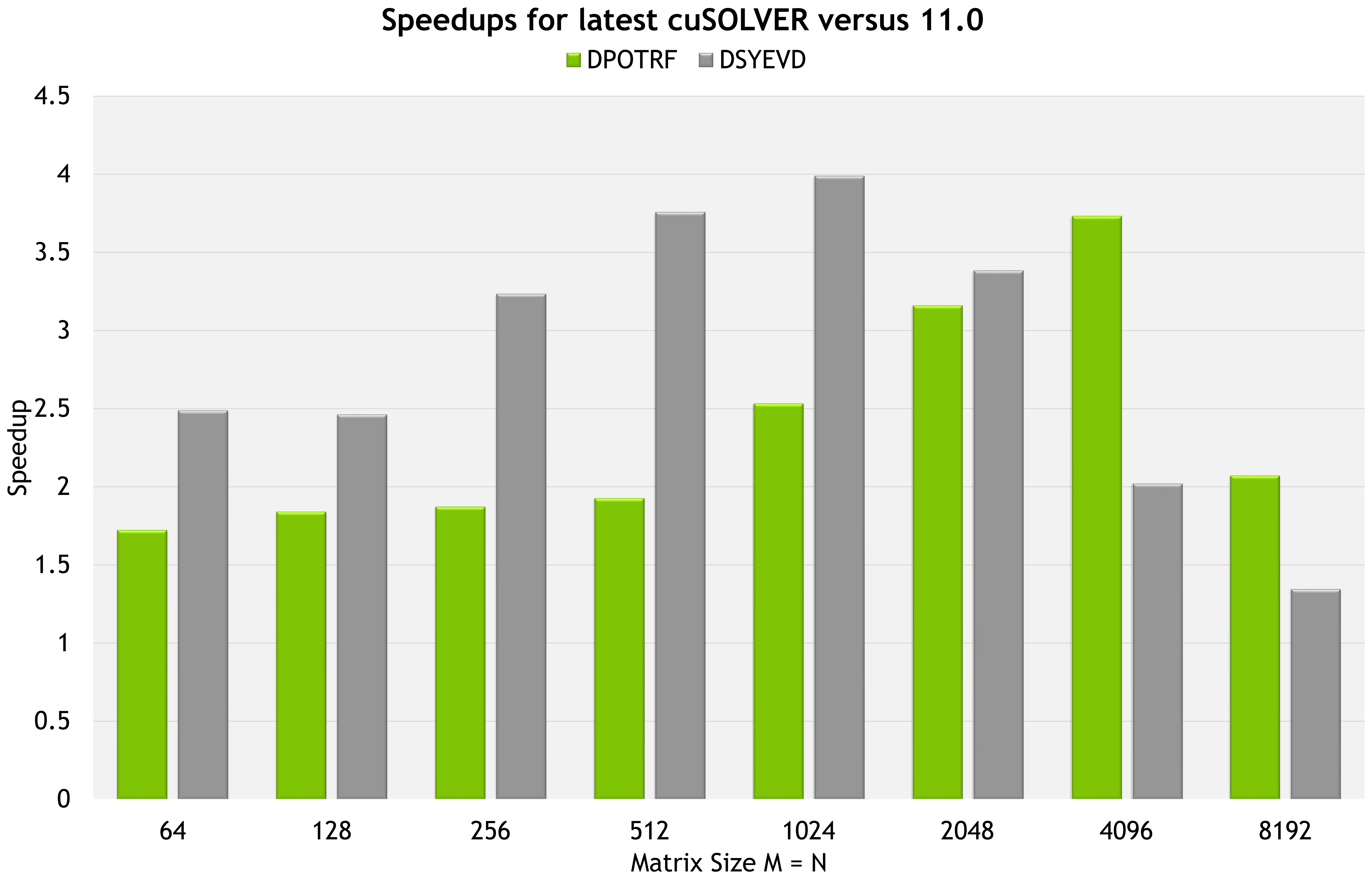
- LU, Cholesky, QR
- Symmetric and Generalized Eigensolvers
- Tensor Core Accelerated Iterative Refinement Solvers
- Multi GPU & Multi-node Support

Accelerating a wide range of applications

- HPC & Scientific Computing
- Data Analytics

Recently Improvements

- Improvements for small (D/Z)SYGVD/SYEVD
- Multi-node Multi-GPU (LU w/ & w/o pivoting)



* A100 80GB Default clocks: CTK 11.0 vs. CTK 11.6

cuFFT

GPU Optimized Fast Fourier Transforms

GPU Optimized FFT

- 1D, 2D and 3D FFT
- Single Process Multi-GPU Support

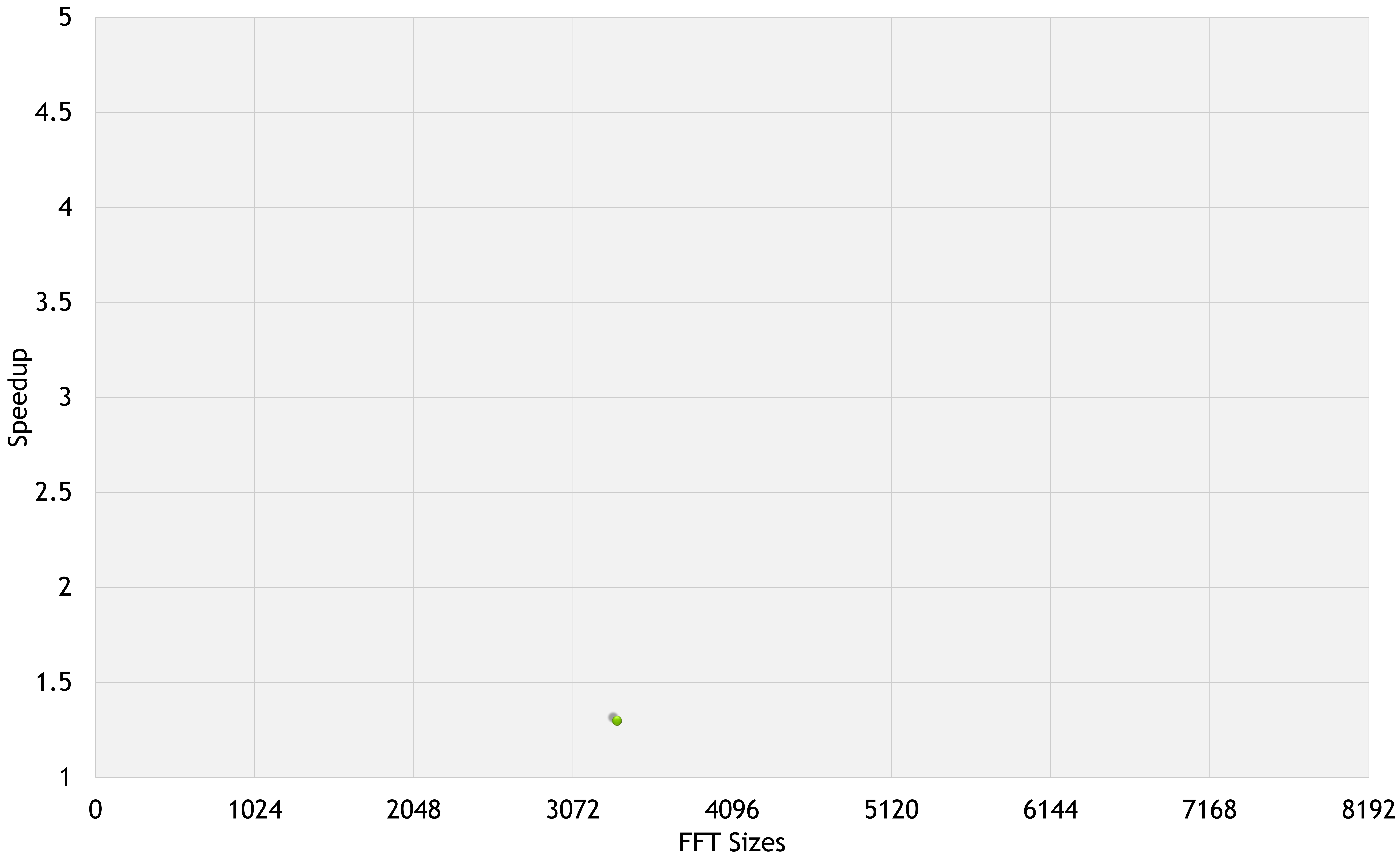
Accelerating a wide range of applications

- HPC & Scientific Computing
- Data Analytics

Recent Improvements

- Optimizations for large 3D FFT
- Uniform performance improvement for size < 32k
- Performance improvements for all sizes (up to 10x)

Speedups (over 10%) for latest cuFFT versus CTK 11.0



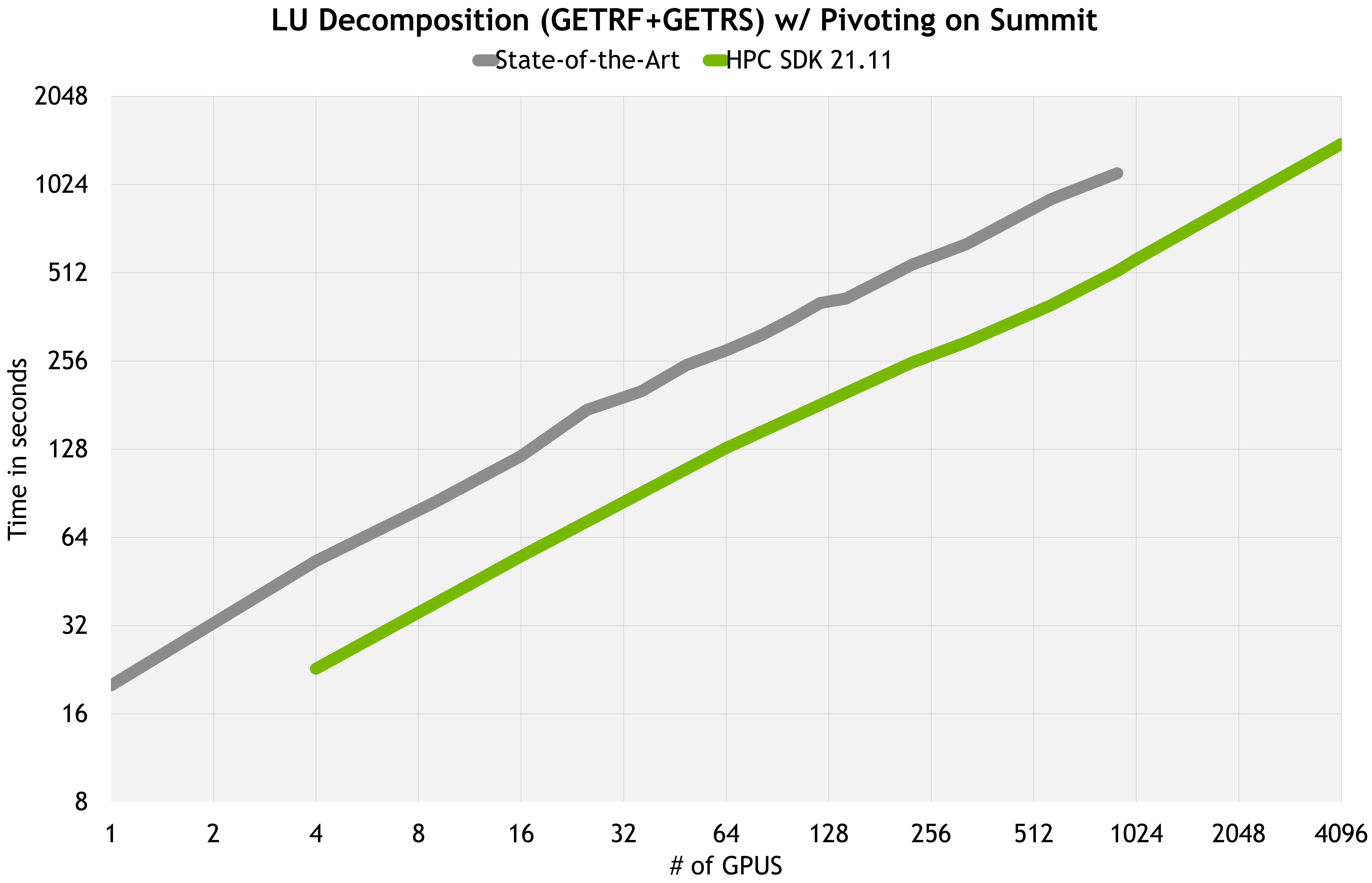
* A100 80GB Default clocks: CTK 11.0 vs. CTK 11.7U1

cuSOLVER

GPU Optimized Factorizations & Solvers

Recent Improvements

- First Released in HPC SDK 21.11
- LU Decomposition
- Cholesky



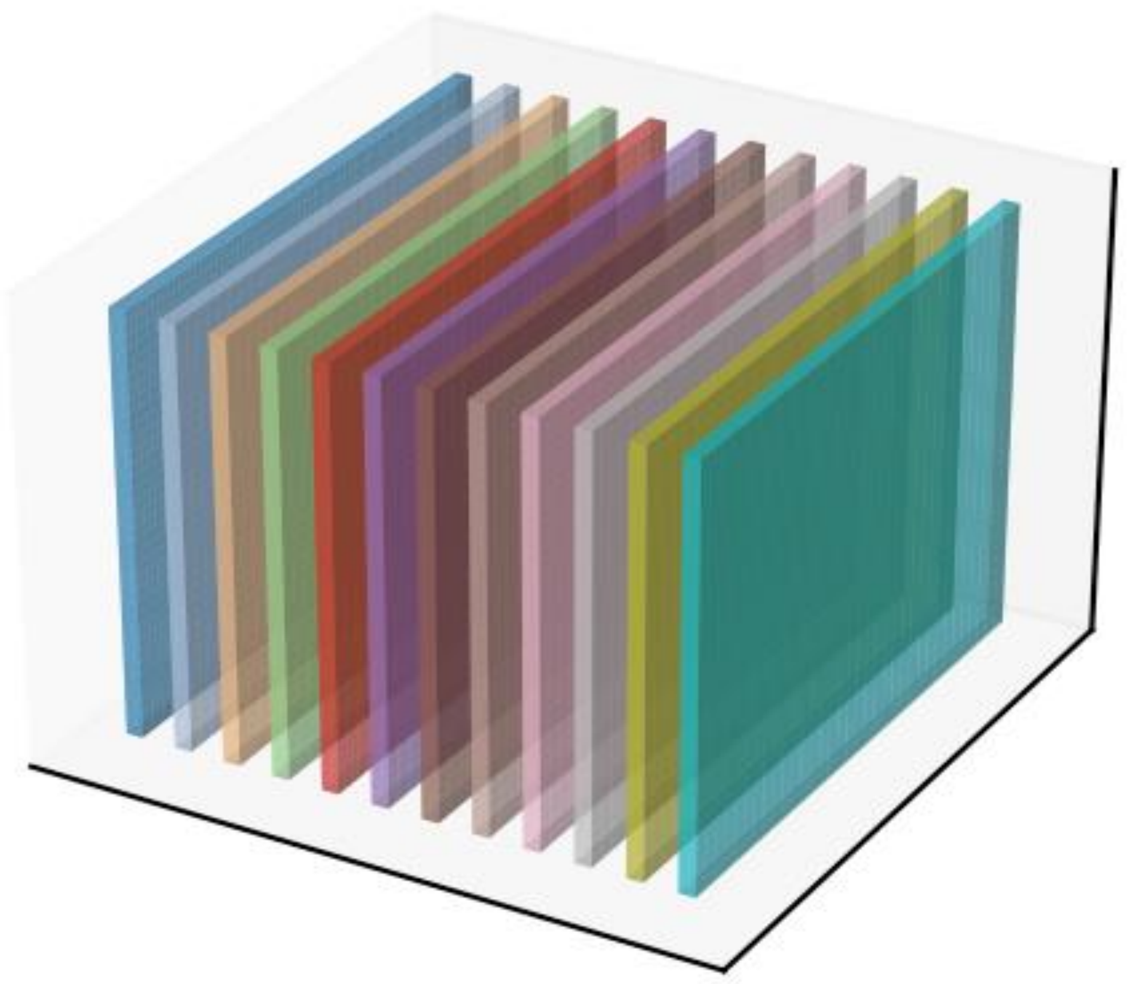
* Summit: 6x V100 16GB per node

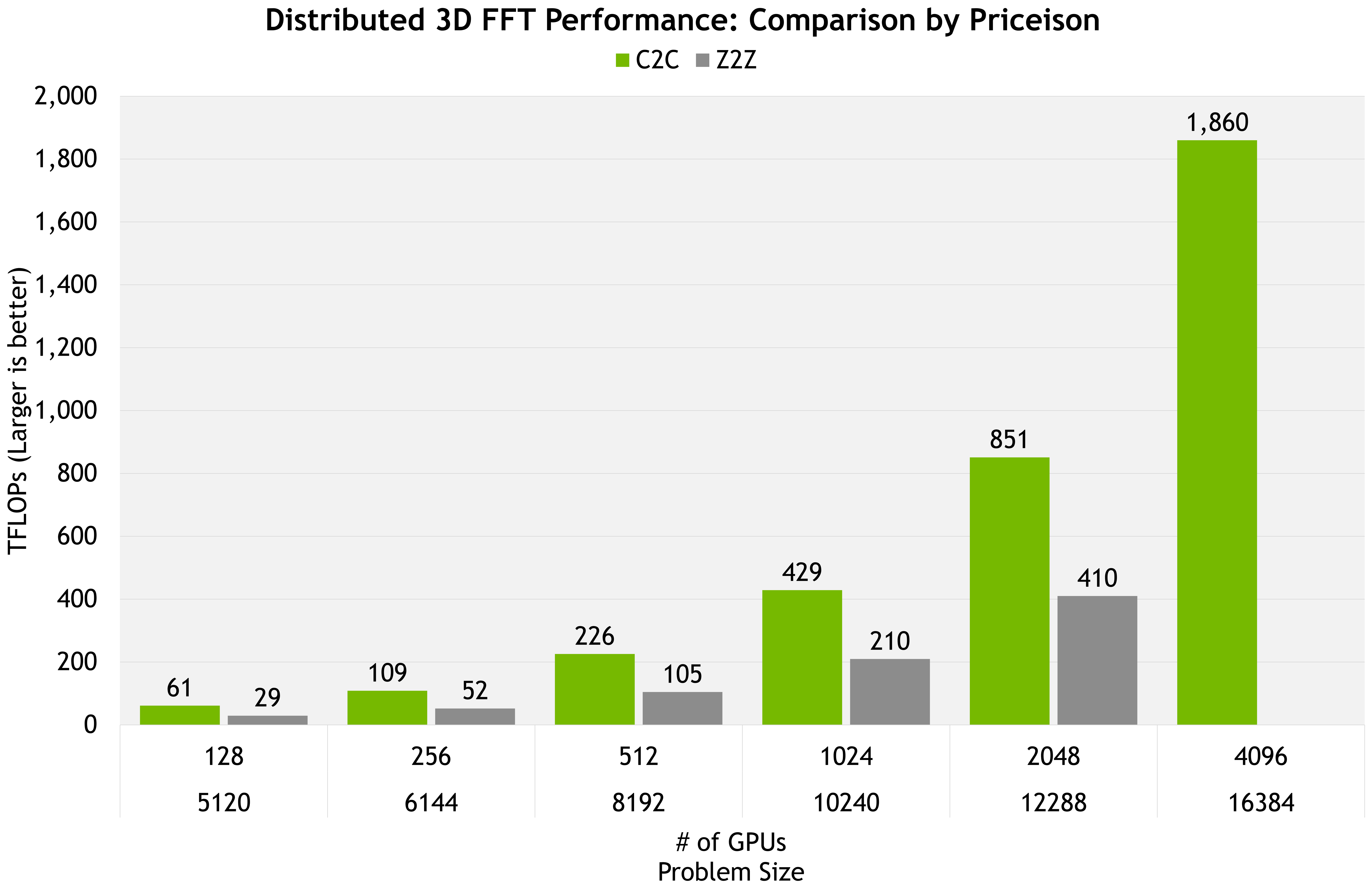
cuFFTMp

Distributed 2D/3D FFTs at Scale

Recent Improvements

- Released in HPC SDK 22.3
- Distributed 2D/3D FFTs
- Slab Decomposition
- Pencil Decomposition (Preview)
- Helper functions: Pencils <-> Slabs





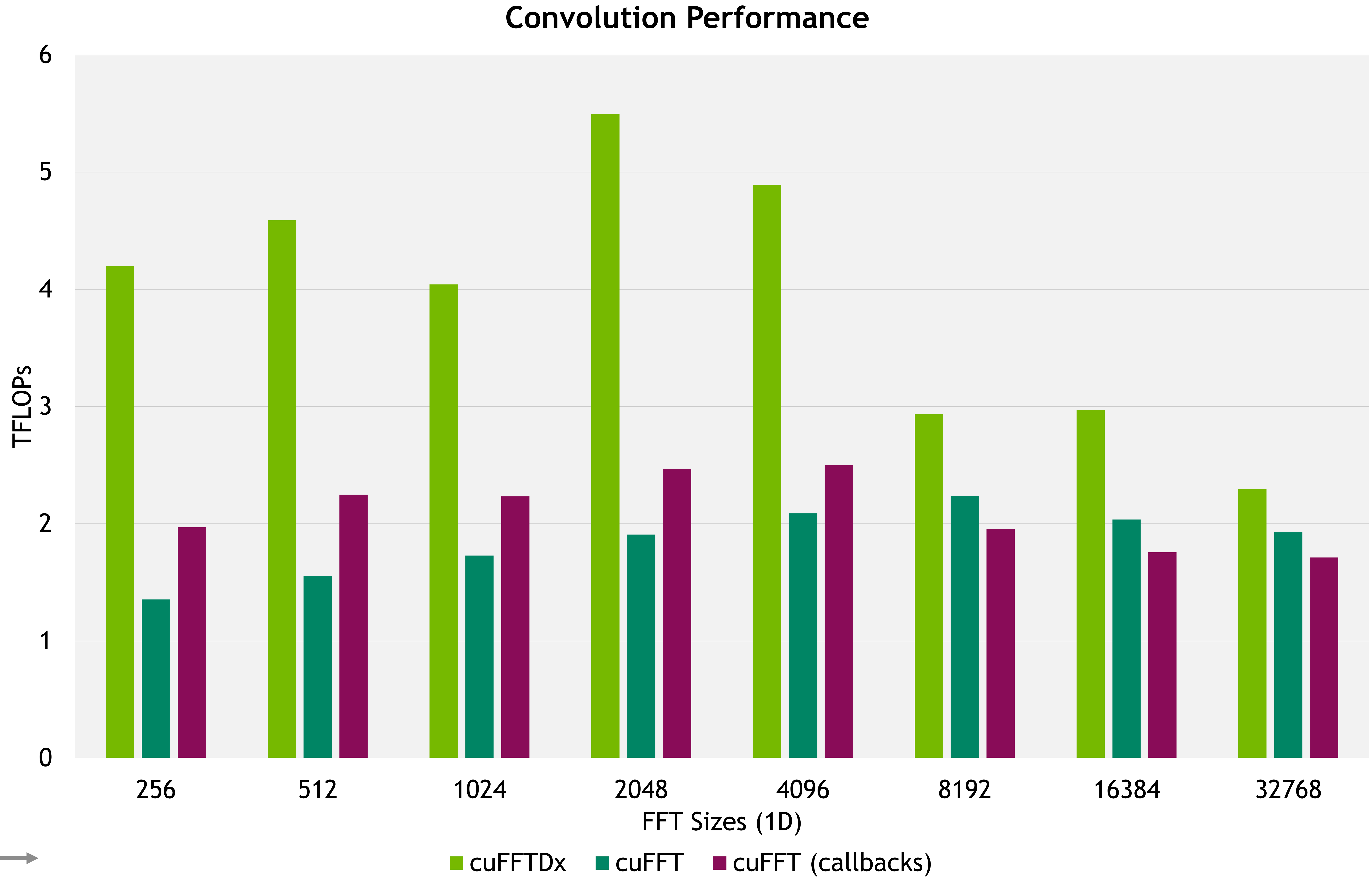
* Selene: A100 80GB @ 1410 MHz

MATH LIBRARIES DEVICE EXTENSIONS

Enabling kernel fusion of high-performance numerical methods

cuFFTDx: In MathDx

- https://developer.nvidia.com/mathdx
- Retain and reuse on-chip data
- Inline FFTs in user kernel up to 32k (A100)
- Combine FFT operations



* A100 80GB @ 1410 MHz

* A100 80GB @ 1410 MHz

Download: MathDx 22.02 at https://developer.nvidia.com/mathdx

AGENDA

Accelerated Computing with Standard Languages

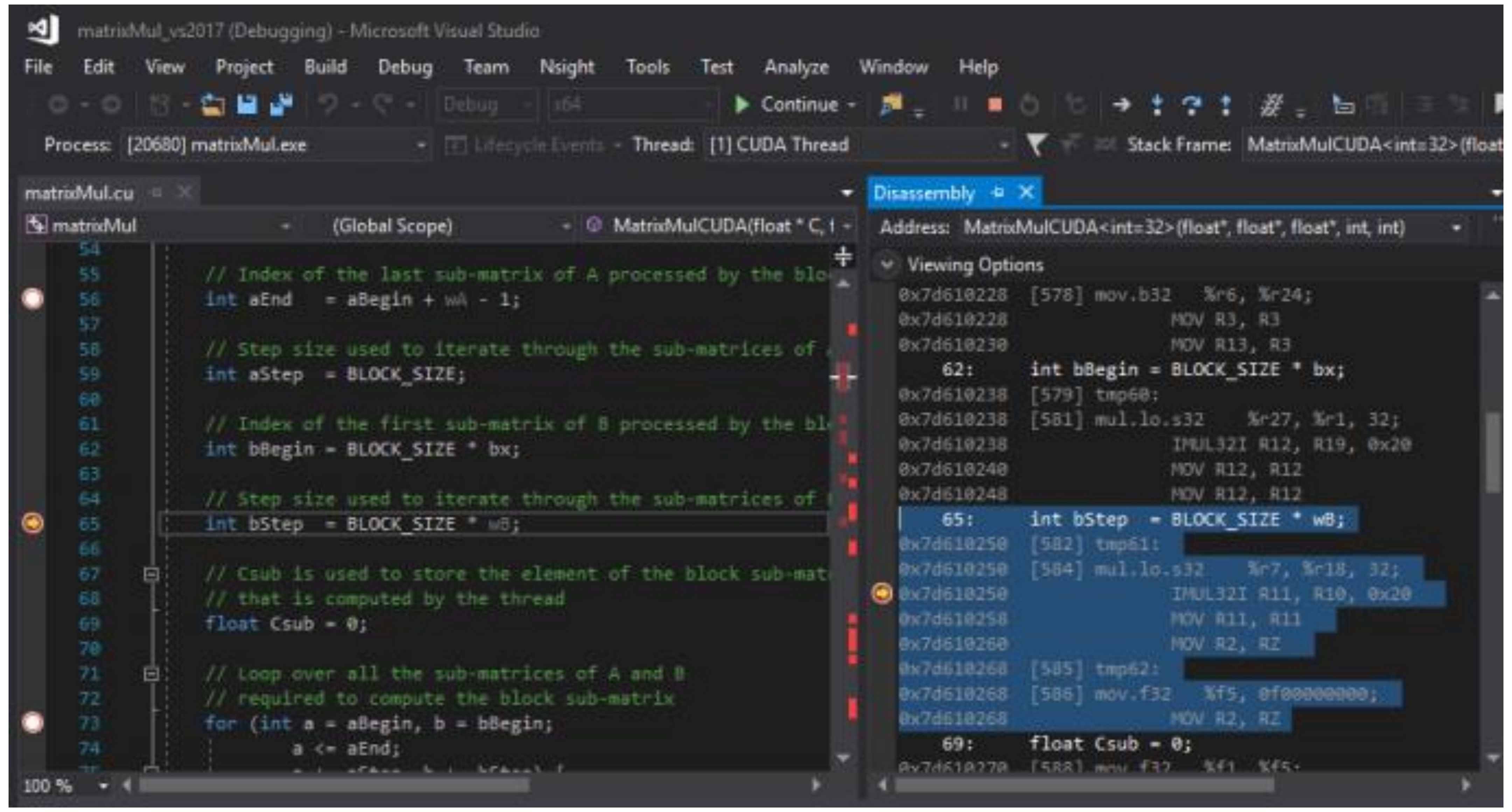
GPU Supercomputing in the PyData Ecosystem

Advancements in HPC Libraries

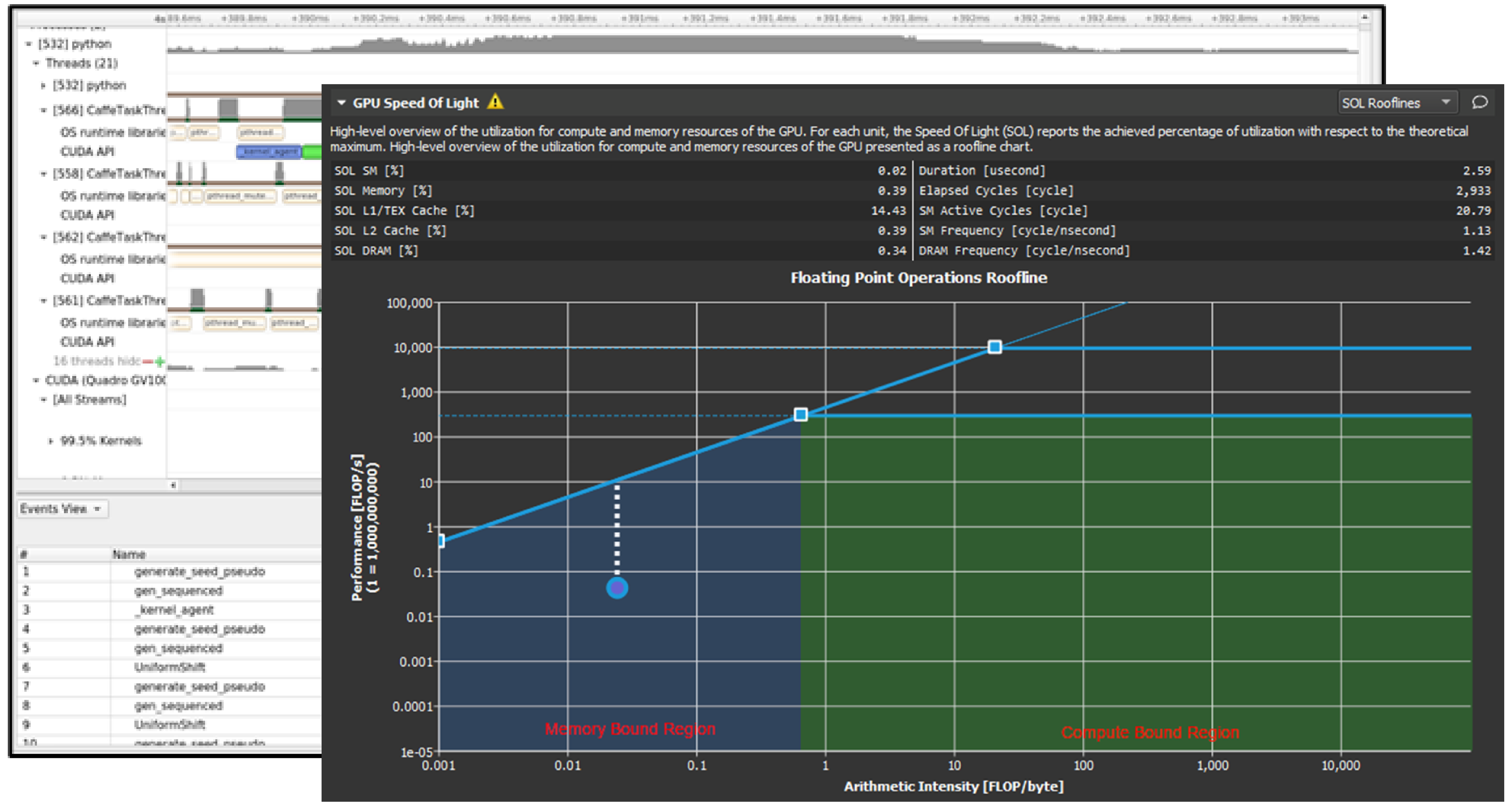
NVIDIA Developer Tools

DEVELOPER TOOLS

Debuggers: cuda-gdb, Nsight Visual Studio Edition



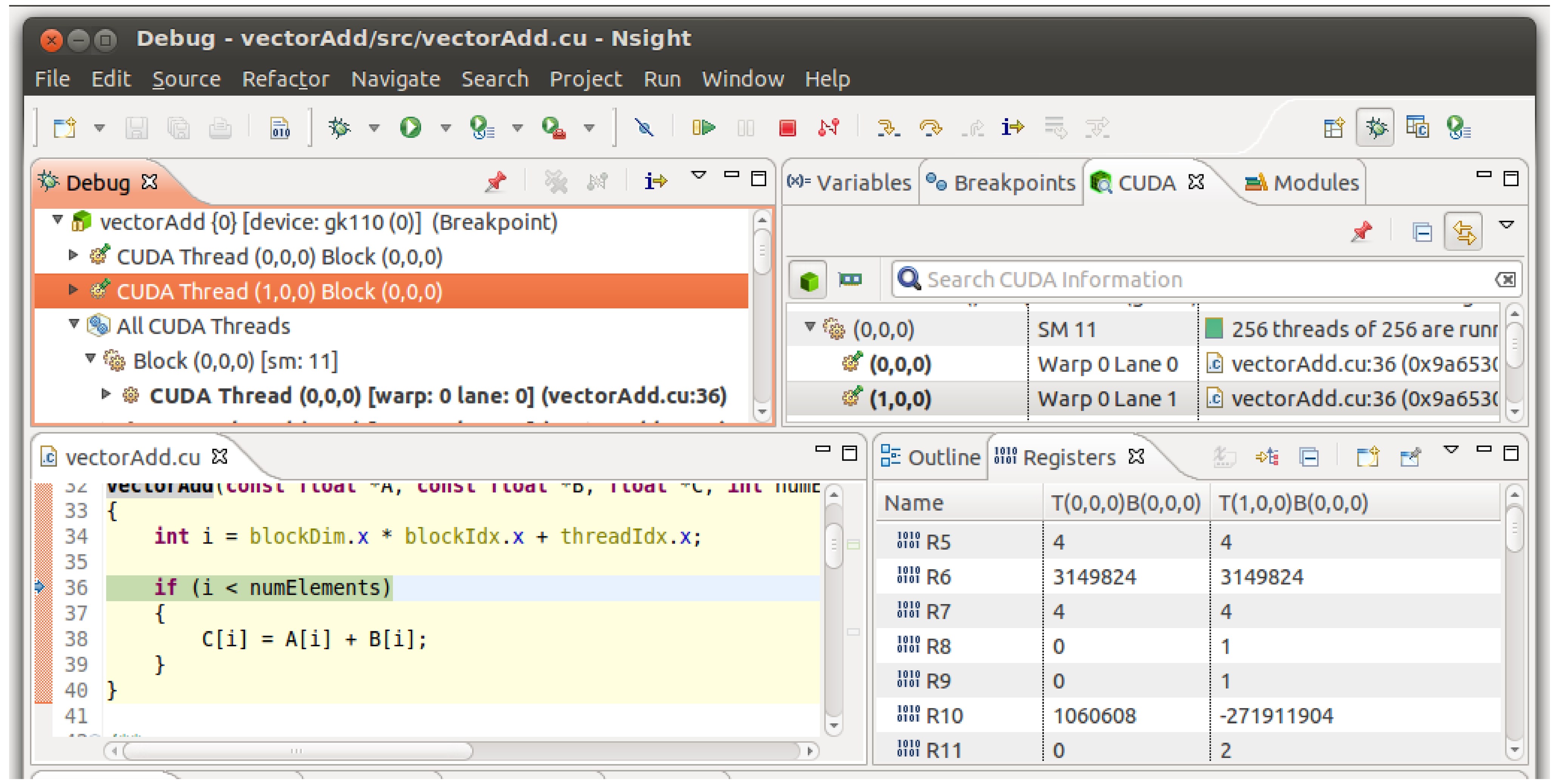
Profilers: Nsight Systems, Nsight Compute, CUPTI, NVIDIA Tools eXtension (NVTX)



Correctness Checker:: Compute Sanitizer

```
$ compute-sanitizer --leak-check full memcheck_demo
========= COMPUTE-SANITIZER
Mallocing memory
Running unaligned_kernel
Ran unaligned_kernel: no error
Sync: no error
Running out_of_bounds_kernel
Ran out_of_bounds_kernel: no error
Sync: no error
========= Invalid __global__ write of size 4 bytes
=========     at 0x60 in memcheck_demo.cu:6:unaligned_kernel(void)
=========     by thread (0,0,0) in block (0,0,0)
=========     Address 0x400100001 is misaligned
```

IDE integrations: Nsight Eclipse Edition
Nsight Visual Studio Edition
Nsight Visual Studio **Code** Edition



NVIDIA

NSIGHT SYSTEMS

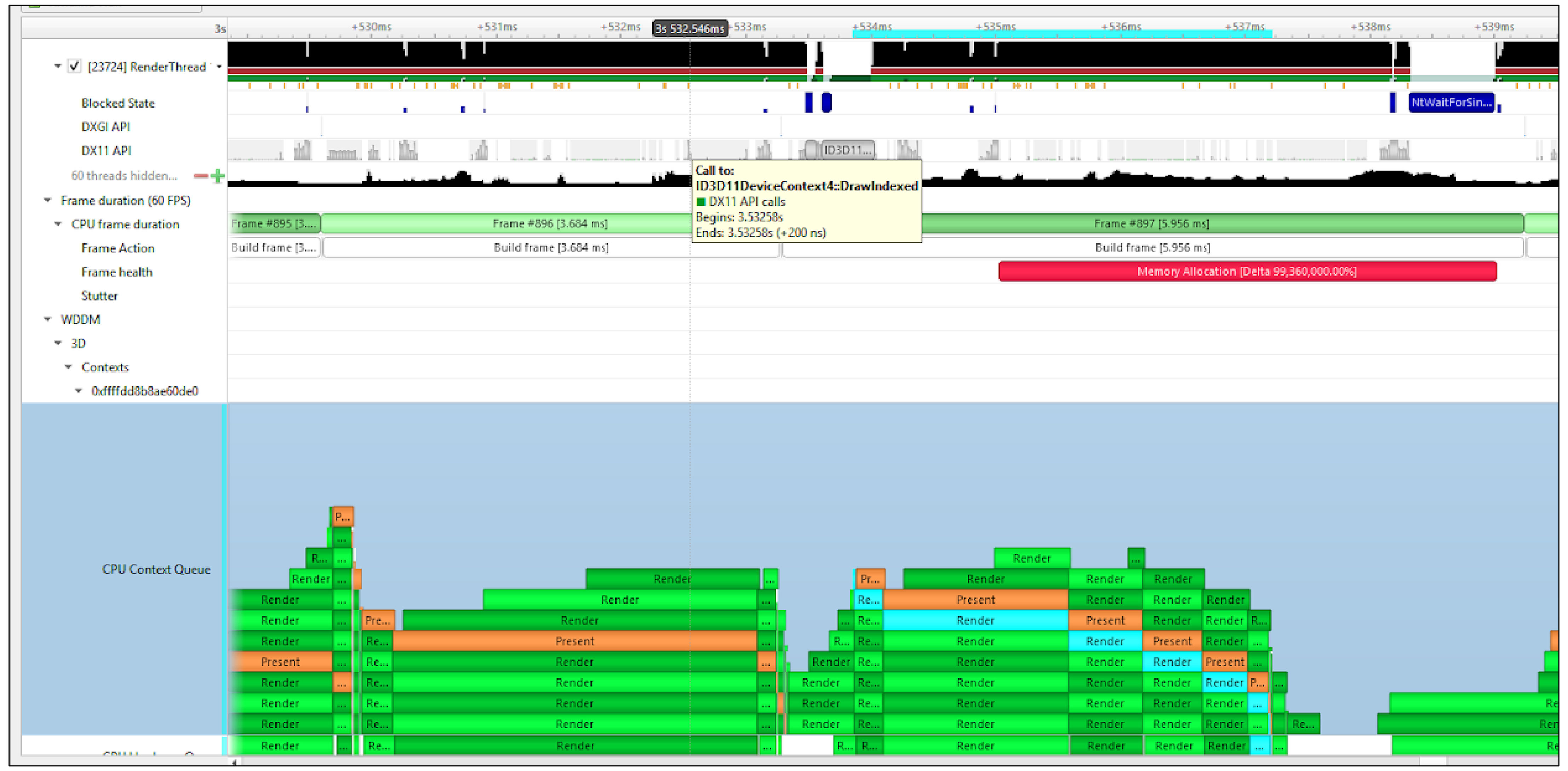
SYSTEM PROFILER

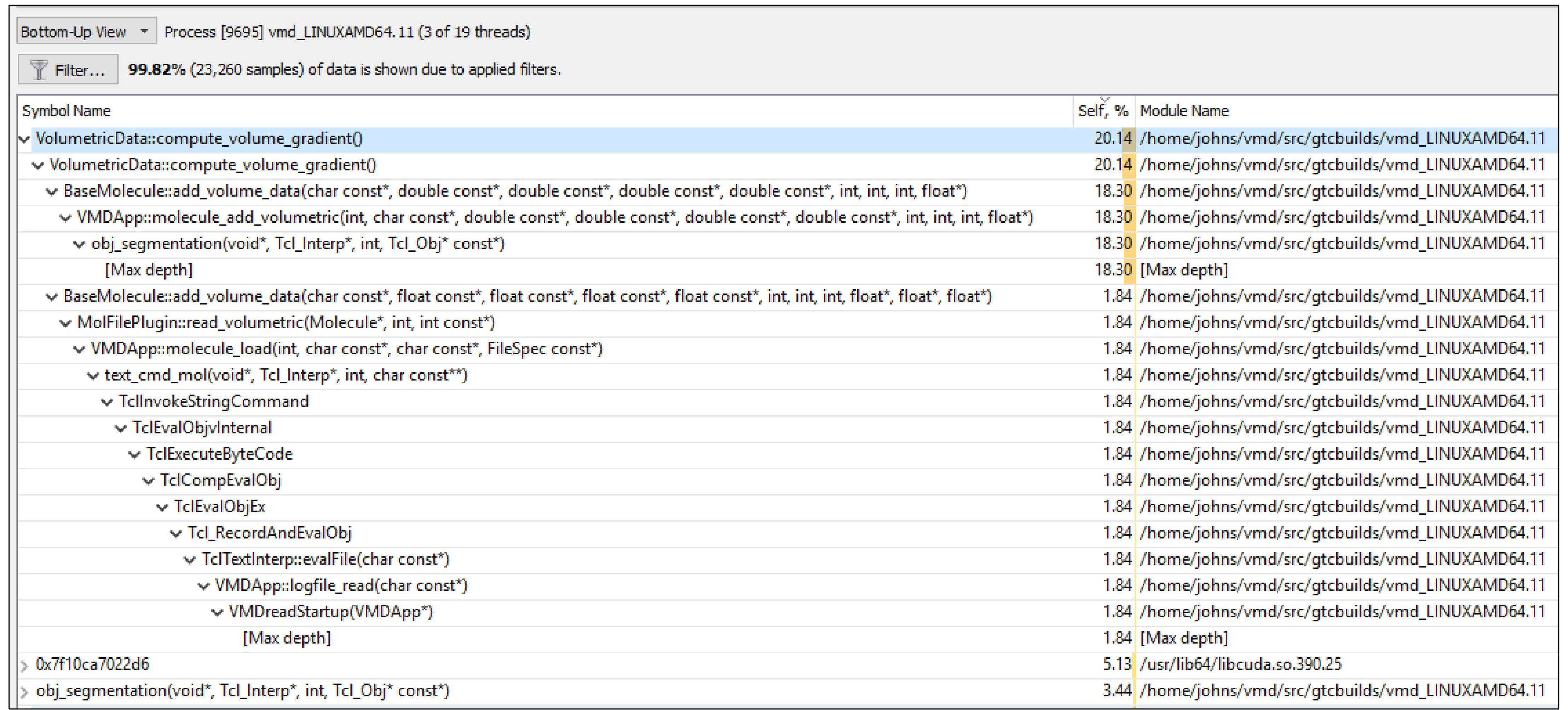
Key Features:

- 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





NSIGHT COMPUTE

KERNEL PROFILING TOOL

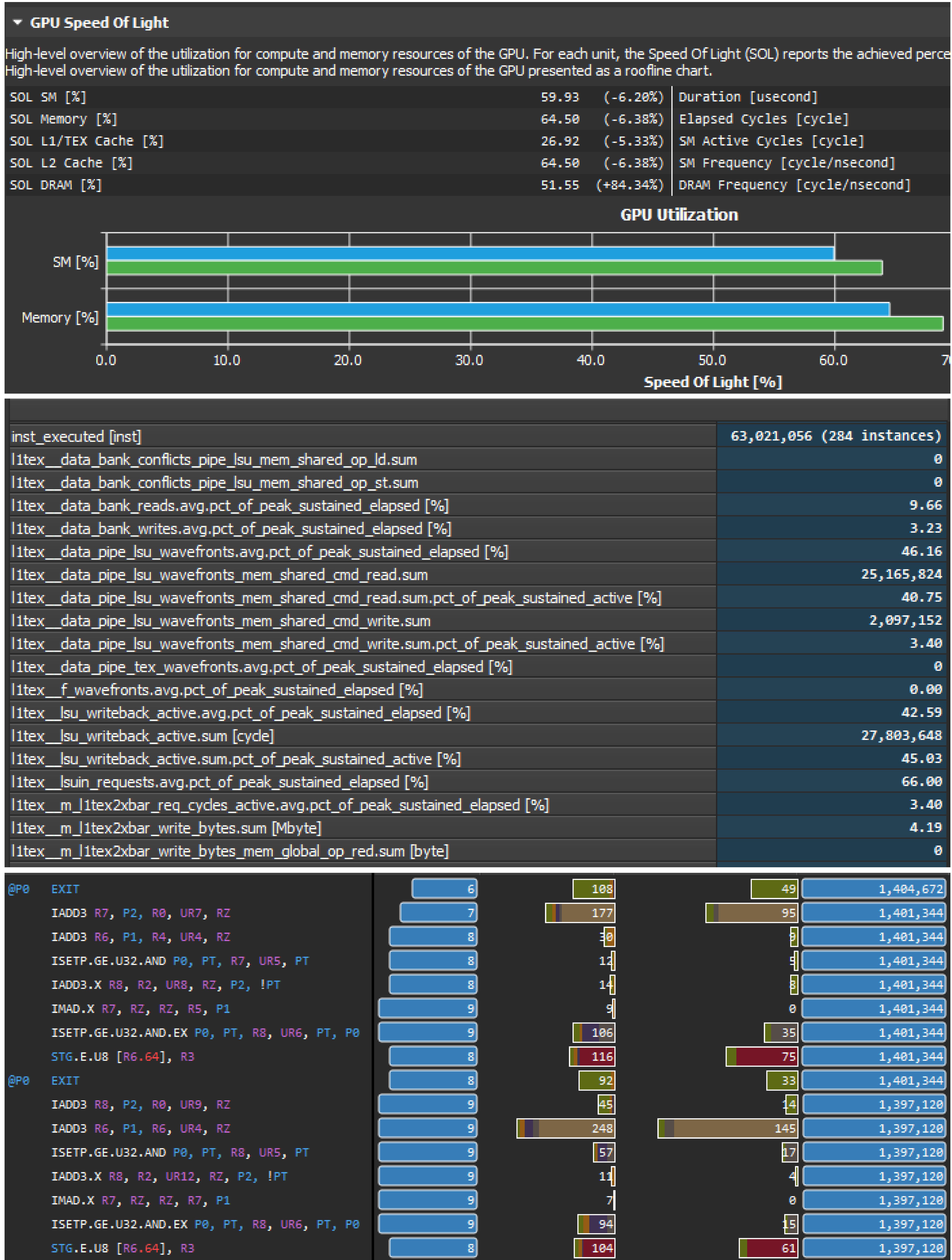
Key Features:

- Interactive CUDA API debugging and kernel profiling
- Built-in rules expertise
- Fully customizable data collection and display
- Command Line, Standalone, IDE Integration, Remote Targets

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

GPUs: Volta, Turing, Ampere GPUs

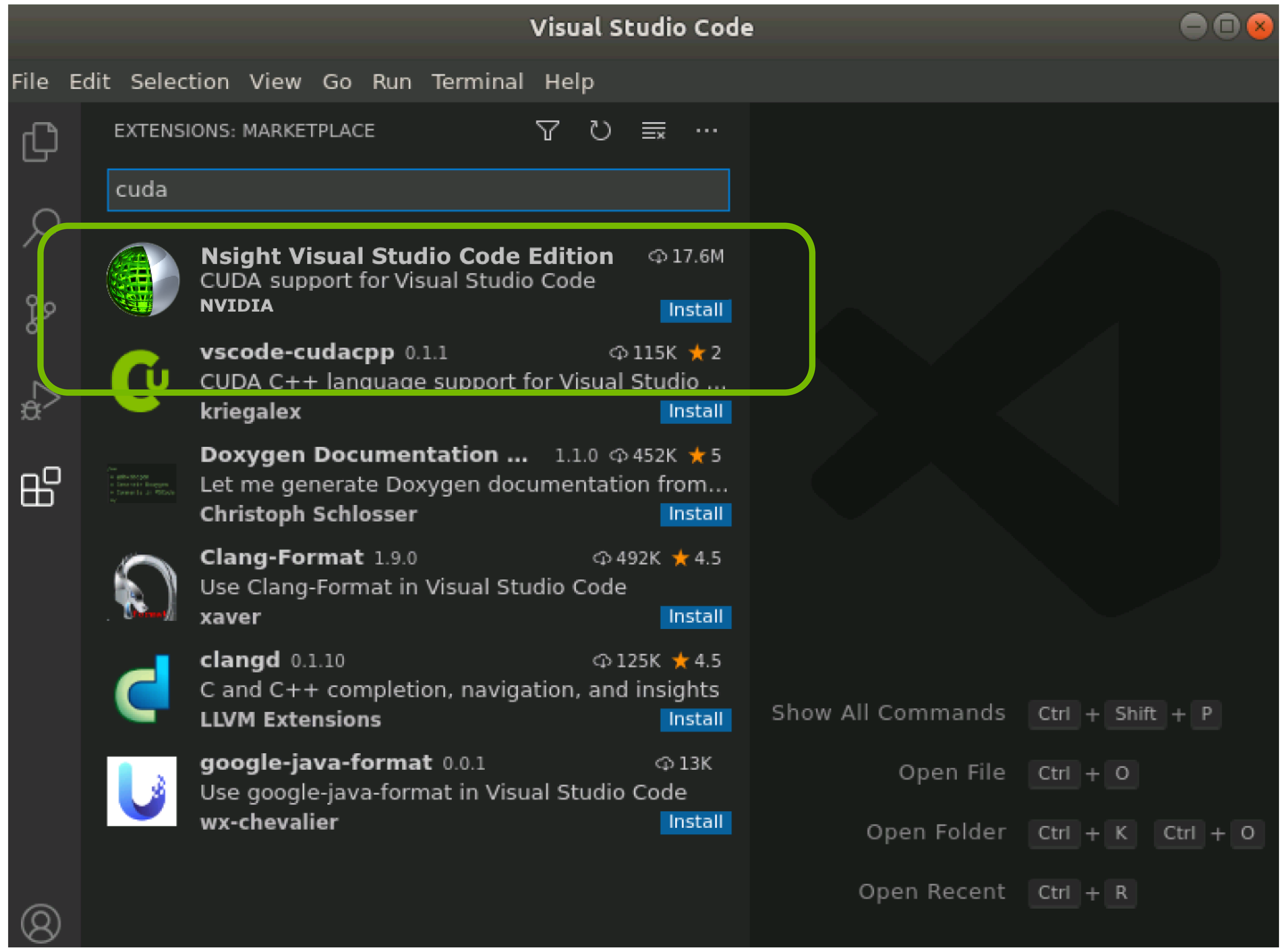
Docs/product: https://developer.nvidia.com/nsight-compute

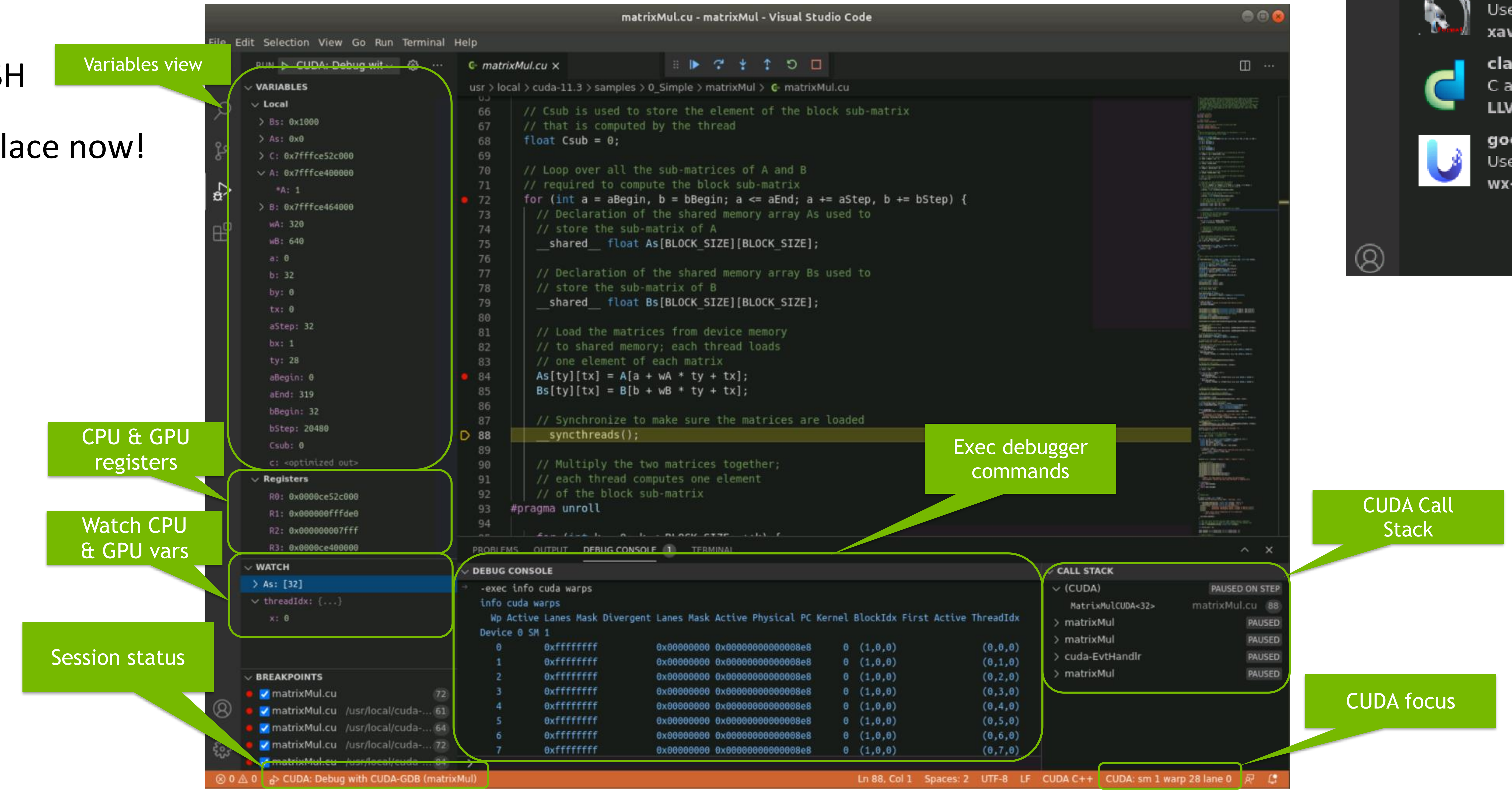


NSIGHT VISUAL STUDIO CODE EDITION

Visual Studio Code extensions that provides:

- CUDA code syntax highlighting
- CUDA code completion
- Build warning/errors
- Debug CPU & GPU code
- Remote connection support via SSH
- Available on the VS Code Marketplace now!





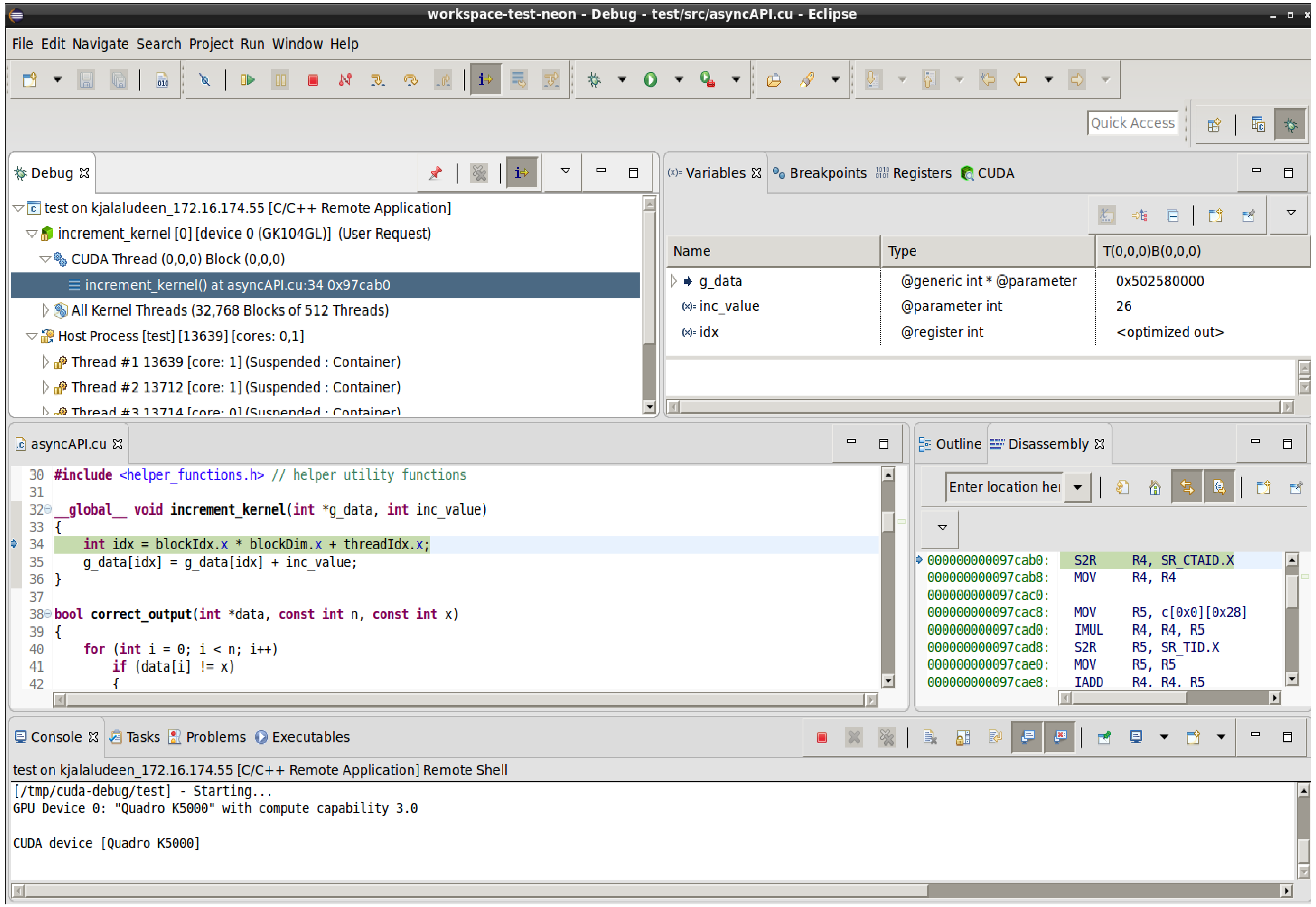
https://developer.nvidia.com/nsight-visual-studio-code-edition

NVIDIA

NSIGHT ECLIPSE EDITION

INTEGRATED CUDA APPLICATION DEVELOPMENT

- Edit, build and Debug CUDA applications
- Seamless CPU and CUDA Debugging
- Native Eclipse plugin
- Docker container support



CUDA GDB

COMMAND LINE AND IDE BACKEND DEBUGGER

- Unified CPU and CUDA Debugging
- CUDA-C/PTX/SASS support
- Built on GDB and uses many of the same CLI commands

```
(cuda-gdb) info cuda threads breakpoint all
  BlockIdx ThreadIdx         Virtual PC Dev SM Wp Ln         Filename  Line
Kernel 0
   (1,0,0)    (0,0,0) 0x0000000000948e58   0 11  0  0 infoCommands.cu    12
   (1,0,0)    (1,0,0) 0x0000000000948e58   0 11  0  1 infoCommands.cu    12
   (1,0,0)    (2,0,0) 0x0000000000948e58   0 11  0  2 infoCommands.cu    12
   (1,0,0)    (3,0,0) 0x0000000000948e58   0 11  0  3 infoCommands.cu    12
   (1,0,0)    (4,0,0) 0x0000000000948e58   0 11  0  4 infoCommands.cu    12
   (1,0,0)    (5,0,0) 0x0000000000948e58   0 11  0  5 infoCommands.cu    12

(cuda-gdb) info cuda threads breakpoint 2 lane 1
  BlockIdx ThreadIdx         Virtual PC Dev SM Wp Ln         Filename  Line
Kernel 0
   (1,0,0)    (1,0,0) 0x0000000000948e58   0 11  0  1 infoCommands.cu    12
```

COMPUTE SANITIZER

AUTOMATICALLY SCAN FOR BUGS AND MEMORY ISSUES

- Compute Sanitizer checks correctness issues via sub-tools:
- *Memcheck* – The memory access error and leak detection tool.
- *Racecheck* – The shared memory data access hazard detection tool.
- *Initcheck* – The uninitialized device global memory access detection tool.
- *Synccheck* – The thread synchronization hazard detection tool.

```
~/W/m/c/build $ cmake ../ && cmake --build .
-- Configuring done
-- Generating done
-- Build files have been written to: /home/rmaynard/Work/misc/cuda_sanitizer_ctest/build
[2/2] Linking CUDA executable demo
~/W/m/c/build $ ctest -D MemoryCheck
   Site: RMAYNARD-DT
   Build name: Linux-unknown
Create new tag: 20210325-1346 - Experimental
Configure project
   Each . represents 1024 bytes of output
    . Size of output: 0K
Build project
   Each symbol represents 1024 bytes of output.
   '!' represents an error and '*' a warning.
    . Size of output: 0K
   0 Compiler errors
   0 Compiler warnings
Performing coverage
 Cannot find any coverage files. Ignoring Coverage request.
Memory check project /home/rmaynard/Work/misc/cuda_sanitizer_ctest/build
    Start 1: verify
1/1 MemCheck #1: verify ..........................   Passed    6.77 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   6.77 sec
-- Processing memory checking output:
1/1 MemCheck: #1: verify ..........................   Defects: 4
MemCheck log files can be found here: (<#> corresponds to test number)
/home/rmaynard/Work/misc/cuda_sanitizer_ctest/build/Testing/Temporary/MemoryChecker.<#>.log
Memory checking results:
Invalid __global__ read - 1
cudaErrorLaunchFailure - 3
Submit files
   SubmitURL: http://my.cdash.org/submit.php?project=CMakeTutorial
   Uploaded: /home/rmaynard/Work/misc/cuda_sanitizer_ctest/build/Testing/20210325-1346/Config
   Uploaded: /home/rmaynard/Work/misc/cuda_sanitizer_ctest/build/Testing/20210325-1346/Build.
   Uploaded: /home/rmaynard/Work/misc/cuda_sanitizer_ctest/build/Testing/20210325-1346/Dynami
   Uploaded: /home/rmaynard/Work/misc/cuda_sanitizer_ctest/build/Testing/20210325-1346/Done.x
   Submission successful
~/W/m/c/build $
```

NVIDIA HPC SDK

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

Develop for the NVIDIA Platform: GPU, CPU and Interconnect

Libraries | Accelerated C++ and Fortran | Directives | CUDA

x86_64 | Arm | OpenPOWER

7-8 Releases Per Year | Freely Available

GTC SPRING 2022 SESSIONS TO REWATCH

For more information on these topics

- No More Porting: Coding for GPUs with Standard C++, Fortran, and Python [S41496]
- A Deep Dive into the Latest HPC Software [S41494]
- C++ Standard Parallelism [S41960]
- Future of Standard and CUDA C++ [S41961]
- Shifting through the Gears of GPU Programming: Understanding Performance and Portability Trade-offs [S41620]
- From Directives to DO CONCURRENT: A Case Study in Standard Parallelism [S41318]
- Evaluating Your Options for Accelerated Numerical Computing in Pure Python [S41645]
- How to Develop Performance Portable Codes using the Latest Parallel Programming Standards [S41618]

NVIDIA®