NVIDIA HPC SOFTWARE

JEFF LARKIN, JANUARY 2022, NERSC/OLCF/ALCF NVIDIA HPC-SDK TRAINING

TOPICS

Programming the NVIDIA Platform With Standard Languages

Bringing GPU Supercomputing to the Python Ecosystem

Math, Core, and Communication Libraries

Developer Tools Update

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

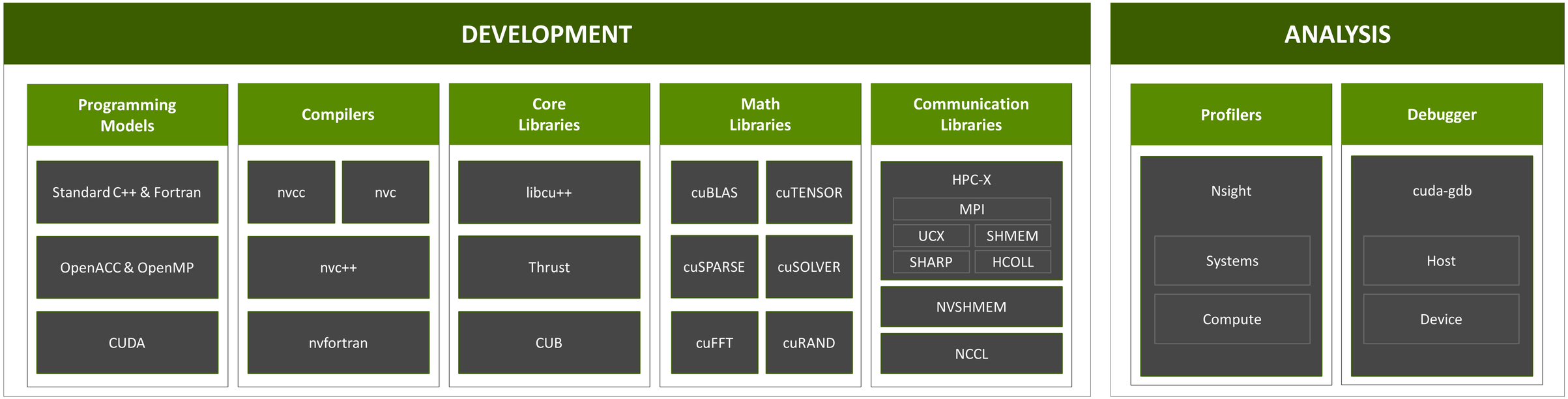
```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

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

NVIDIA HPC SDK

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



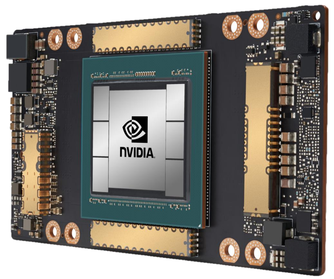
Develop for the NVIDIA Platform: GPU, CPU and Interconnect

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

7-8 Releases Per Year | Freely Available

HPC COMPILERS

NVC | NVC++ | NVFORTRAN



Accelerated

A100
Automatic

Fortran

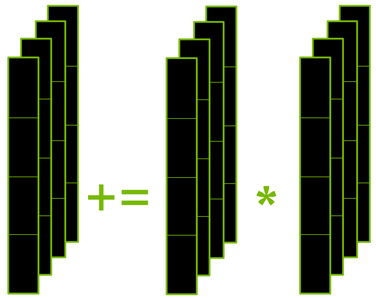
NVIDIA
CUDA

OpenACC
More Science, Less Programming

OpenMP

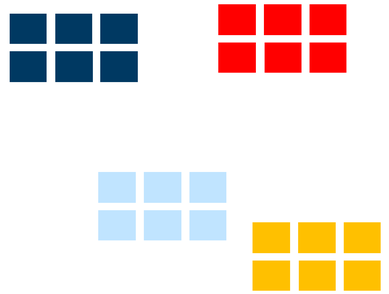
Programmable

Standard Languages
Directives
CUDA



CPU Optimized

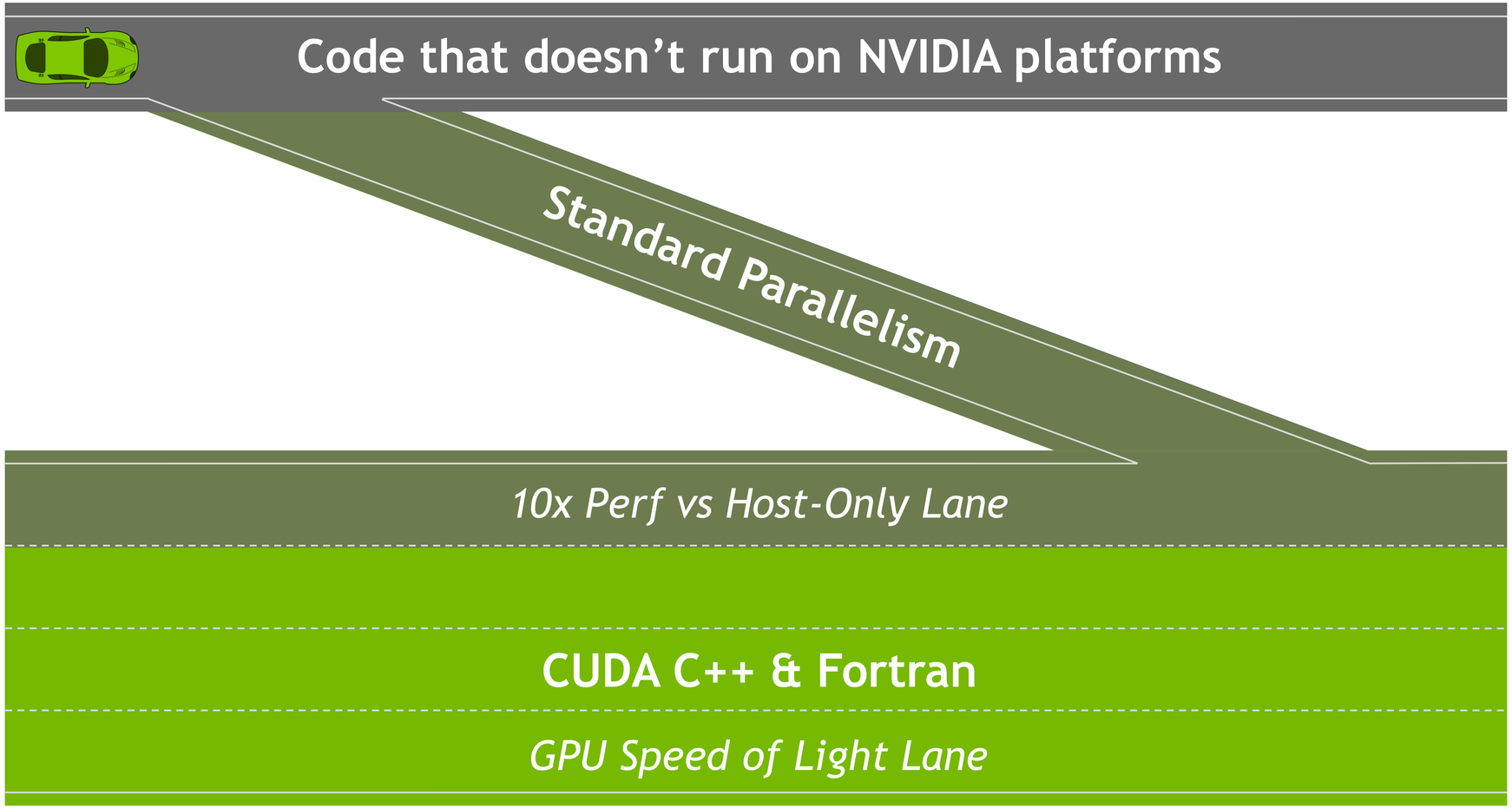
Directives
Vectorization



Multi-Platform

x86_64
Arm
OpenPOWER

Scientists Need On-Ramps



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++2X Executors

C++2X Linear Algebra

C++2X Extended Floating Point Types

C++2X Range Based Parallel Algorithms

Fortran 2X DO CONCURRENT Reduction

HPC PROGRAMMING IN ISO C++

ISO is the place for portable concurrency and parallelism

Preview support coming to NVC++

C++17

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

C++20

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>`

C++23 and Beyond

Executors / Senders-Recievers

- Simplify launching and managing parallel work across CPUs and accelerators

`std::mdspan/mdarray`

- HPC-oriented multi-dimensional array abstractions.

Range-Based Parallel Algorithms

- Improved multi-dimensional loops

Linear Algebra

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

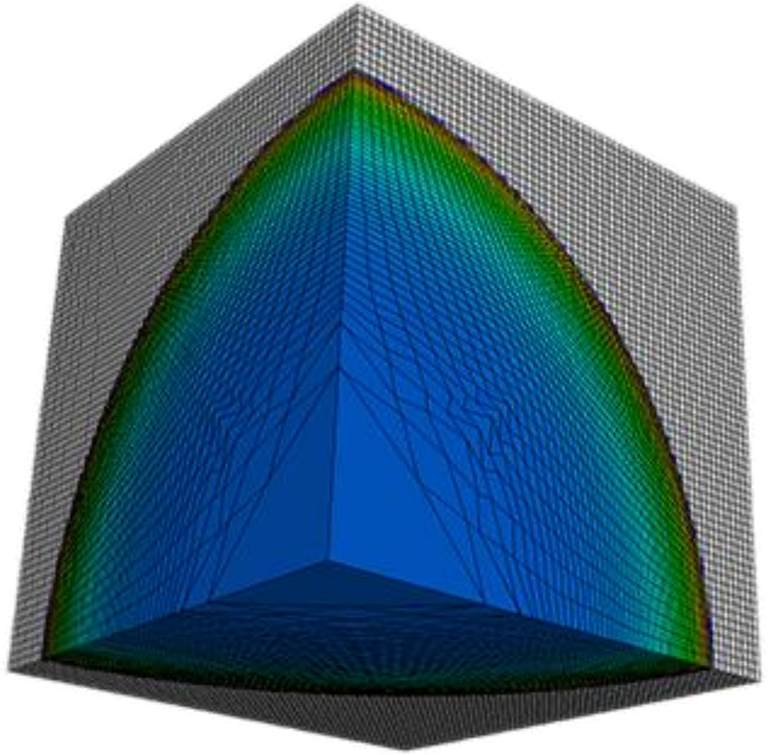
Extended Floating Point Types

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

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

```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

STANDARD C++

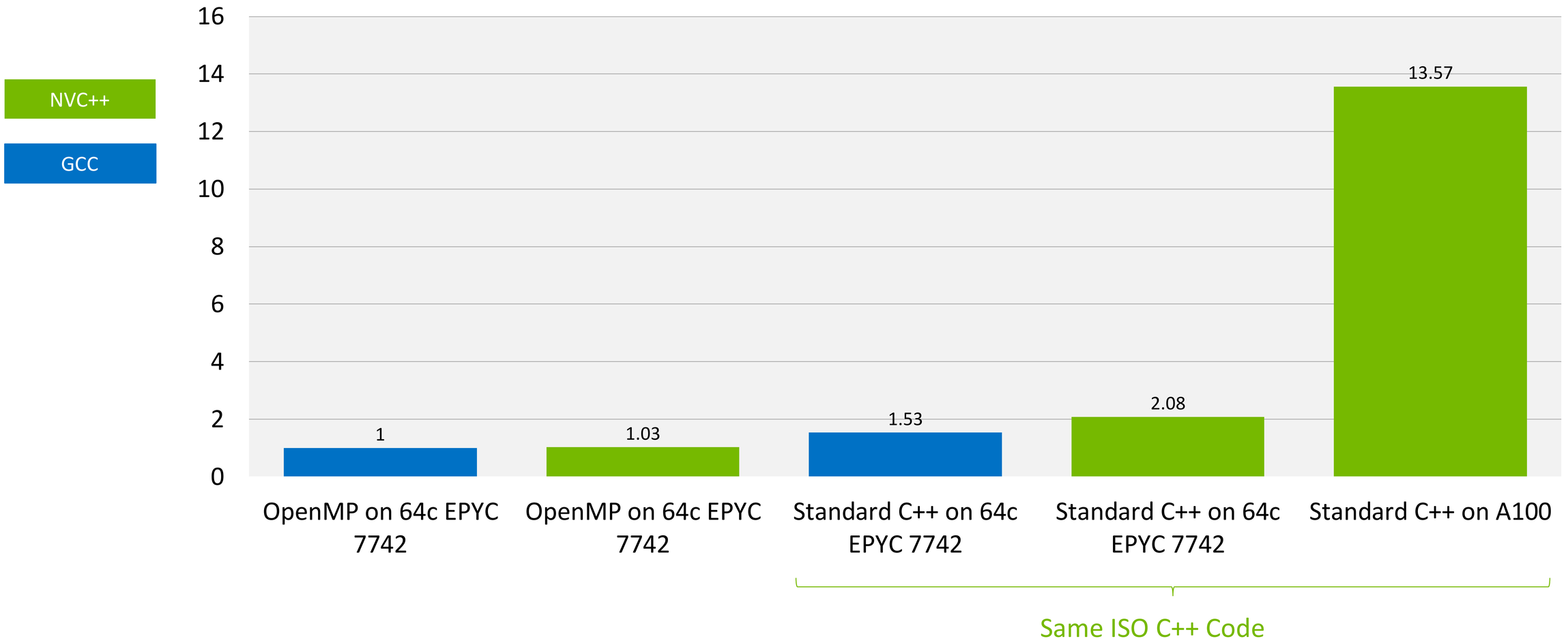
- Composable, compact and elegant
- Easy to read and maintain
- ISO Standard
- Portable – nvc++, g++, icpc, MSVC, …

```cpp
static inline
void CalcHydroConstraintForElems(Domain &domain, Index_t length,
    Index_t *regElemlist, Real_t dvovmax, Real_t &dthydro)
{
  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++

C++ STANDARD PARALLELISM

Lulesh Performance



M-AIA (ZFS) INTRODUCTION

“Multiphysics AIA”

Hierarchical Cartesian grids

Fluid flow, Heat transfer, Combustion, Aeroacoustics

Complex Moving Geometries

~400k LOC, MPI+OpenMP

FV, DG, LBM

LBM solver accelerated with Standard C++

M-AIA REFACTORING

M-AIA (ZFS) INTRODUCTION

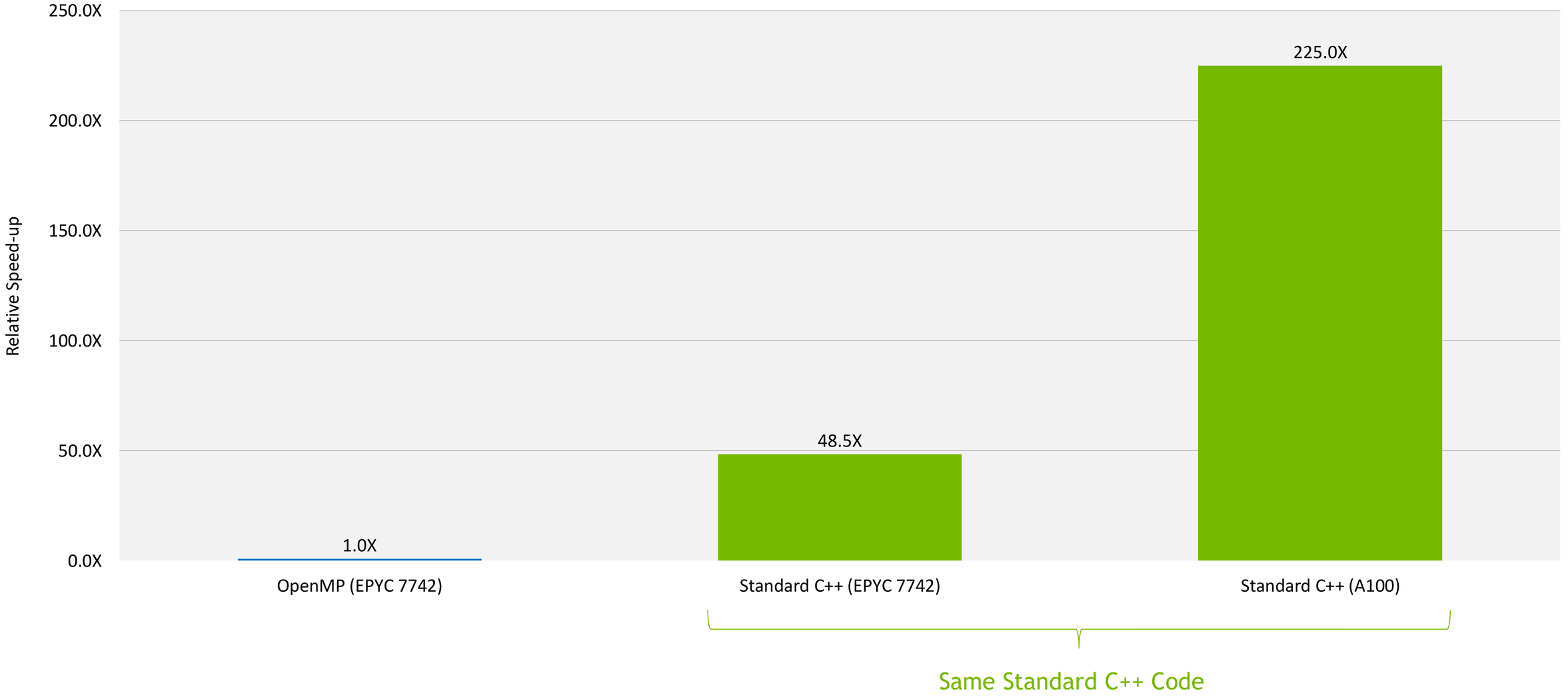
```
#pragma omp parallel // OpenMP parallel region
{
  #pragma omp for // OpenMP for loop
  for (MInt i = 0; i < noCells; i++) { // Loop over all cells
    if (timeStep % ipow2[maxLevel_ - clevel[i * distLevel]] == 0) { // Multi-grid loop
      const MInt distStartId = i * nDist; // More offsets for 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
    }
  }
}
```

Single ISO C++ code that runs in parallel on CPU and GPU

```
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
  }
});
```

C++ STANDARD PARALLELISM

MAIA Performance



STLBM

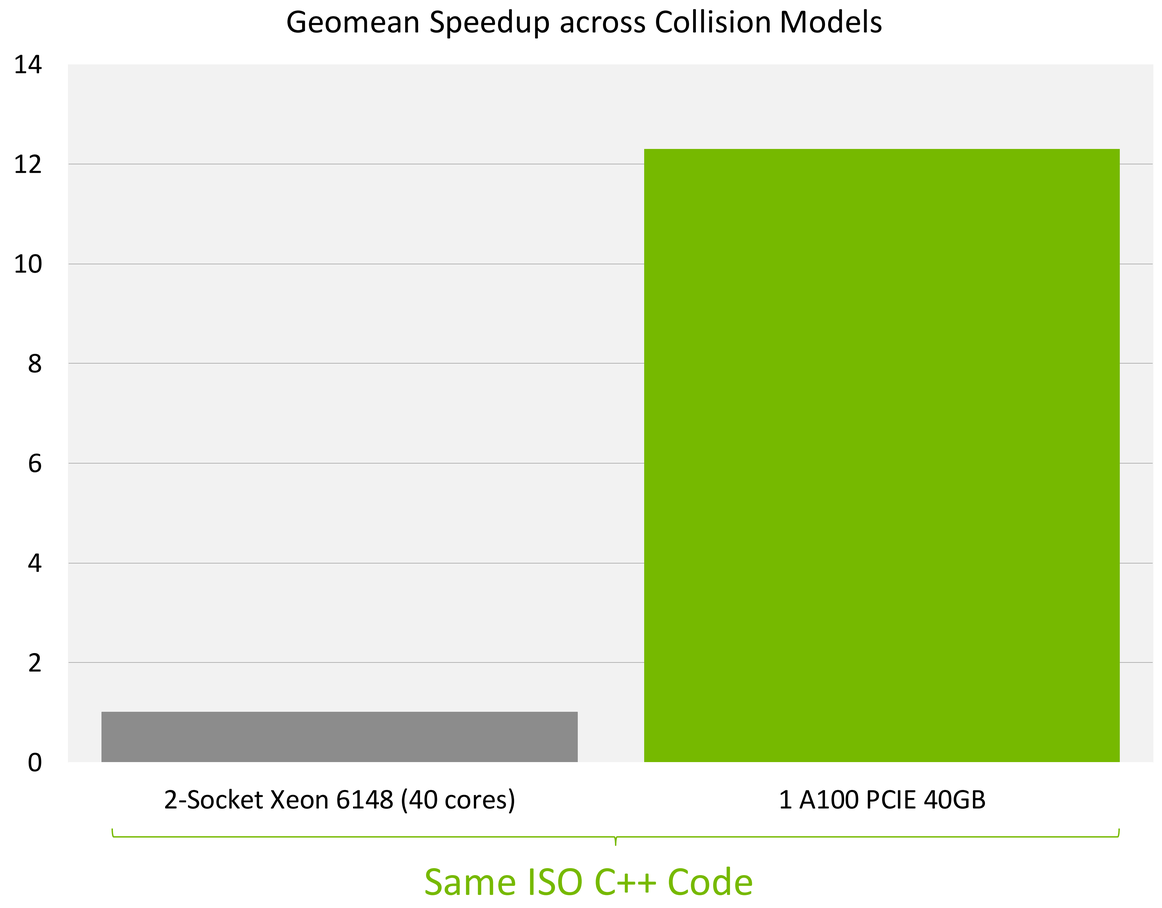
Many-core Lattice Boltzmann with C++ Parallel Algorithms

- 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

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



HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

Fortran 2018

Array Syntax and Intrinsics

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

DO CONCURRENT

- NVFORTRAN 20.11
- Auto-offload & multi-core

Co-Arrays

- Coming Soon
- Accelerated co-array images

Preview support available now in NVFORTRAN

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
- Atomics

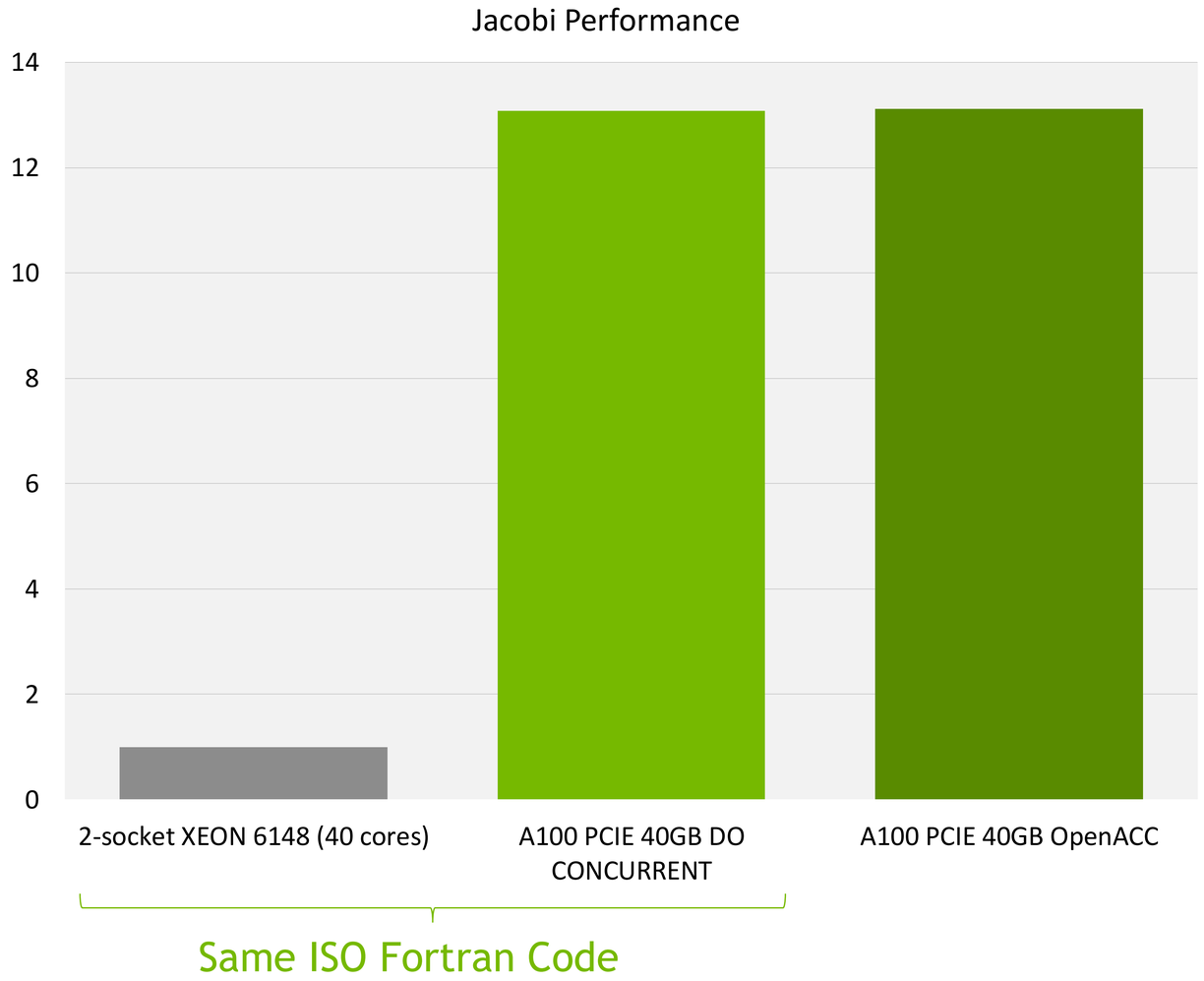
HPC PROGRAMMING IN ISO FORTRAN

DO CONCURRENT

DO CONCURRENT in NVFORTRAN

- Available in NVFORTRAN 20.11
- Automatic GPU acceleration & multi-core support
- Syntax for nested parallelism / loop collapse; expose more parallelism to the compiler

```
subroutine smooth( a, b, w0, w1, w2, n, m, niters )
 real, dimension(:,:) :: a,b
 real :: w0, w1, w2
 integer :: n, m, niters
 integer :: i, j, iter
  do iter = 1,niters
   do concurrent(i=2 : n-1, j=2 : m-1)
     a(i,j) = w0 * b(i,j) + &
              w1 * (b(i-1,j) + b(i,j-1) + b(i+1,j) + b(i,j+1)) + &
              w2 * (b(i-1,j-1) + b(i-1,j+1) + b(i+1,j-1) + b(i+1,j+1))
   enddo
   do concurrent(i=2 : n-1, j=2 : m-1)
     b(i,j) = w0 * a(i,j) + &
              w1 * (a(i-1,j) + a(i,j-1) + a(i+1,j) + a(i,j+1)) + &
              w2 * (a(i-1,j-1) + a(i-1,j+1) + a(i+1,j-1) + a(i+1,j+1))
   enddo
  enddo
```



ACCELERATED PROGRAMMING IN ISO FORTRAN

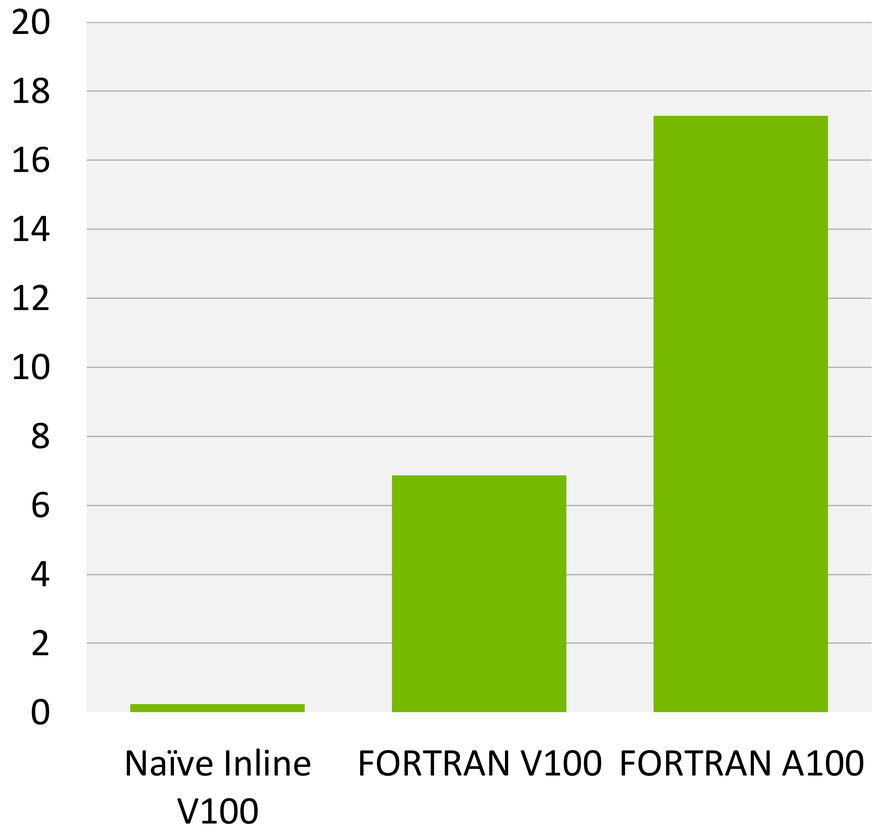
NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

```
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
```

ACCELERATED STANDARD LANGUAGES

Parallel performance for wherever your code runs

ISO C++

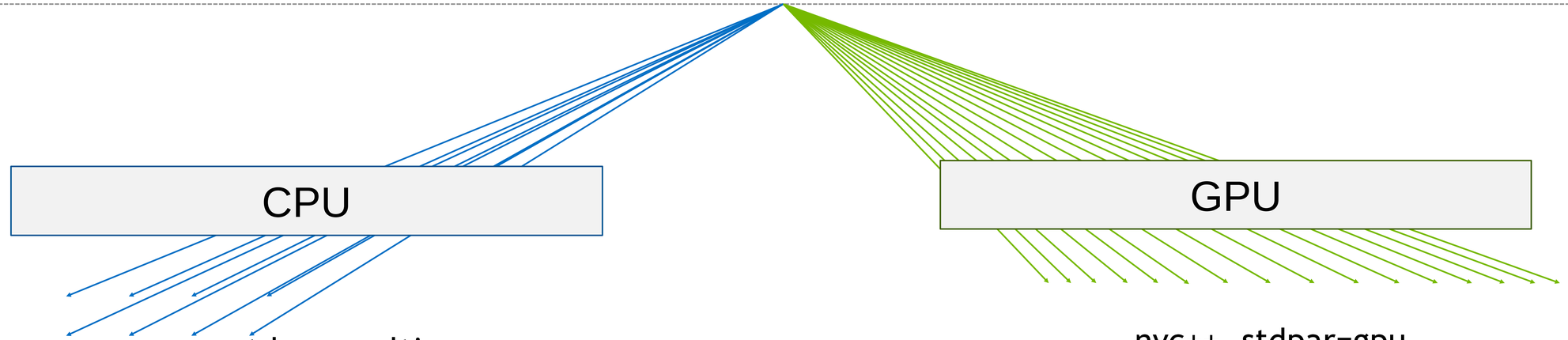
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

ISO Fortran

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

Python

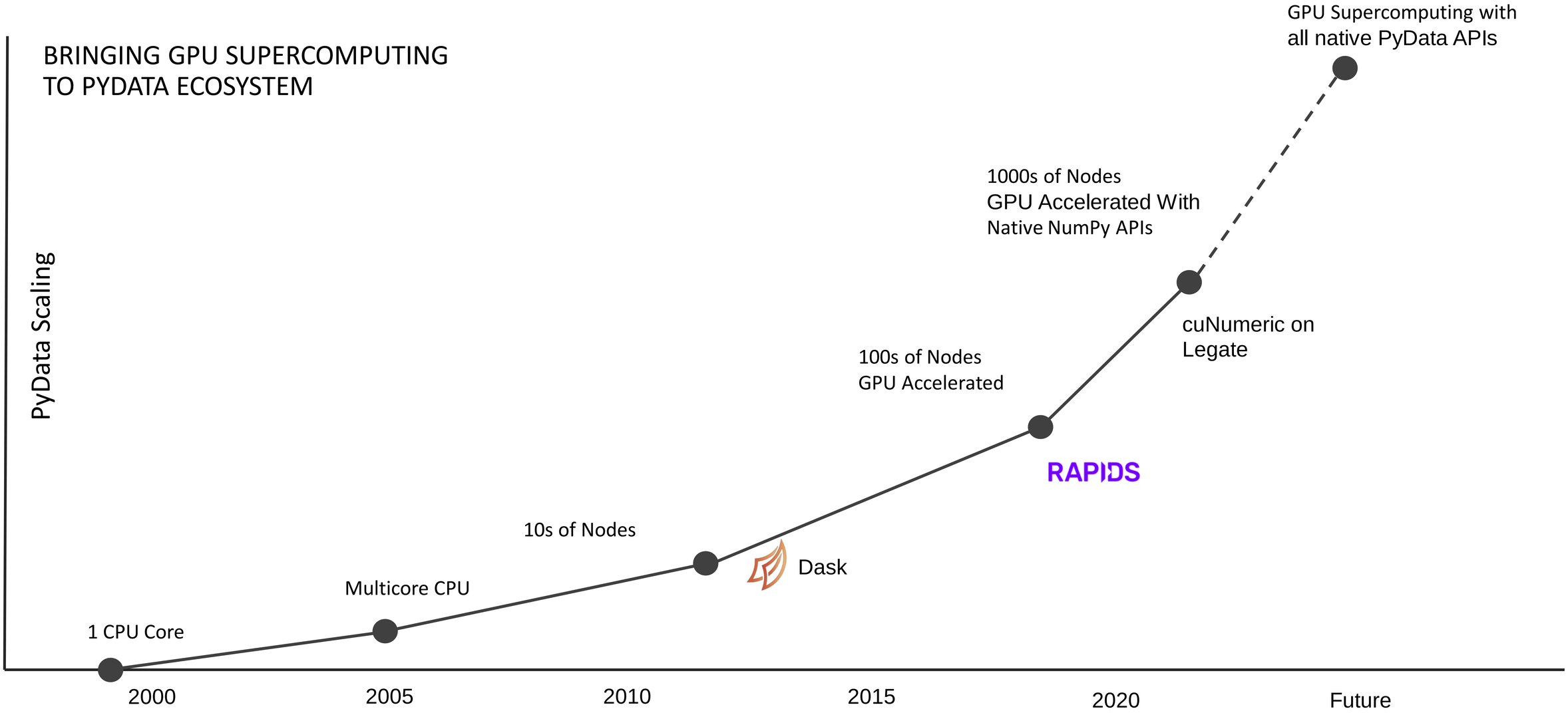
```
import cunumeric as np
…
def saxpy(a, x, y):
    y[:] += a*x
```



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

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

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
```

NVIDIA

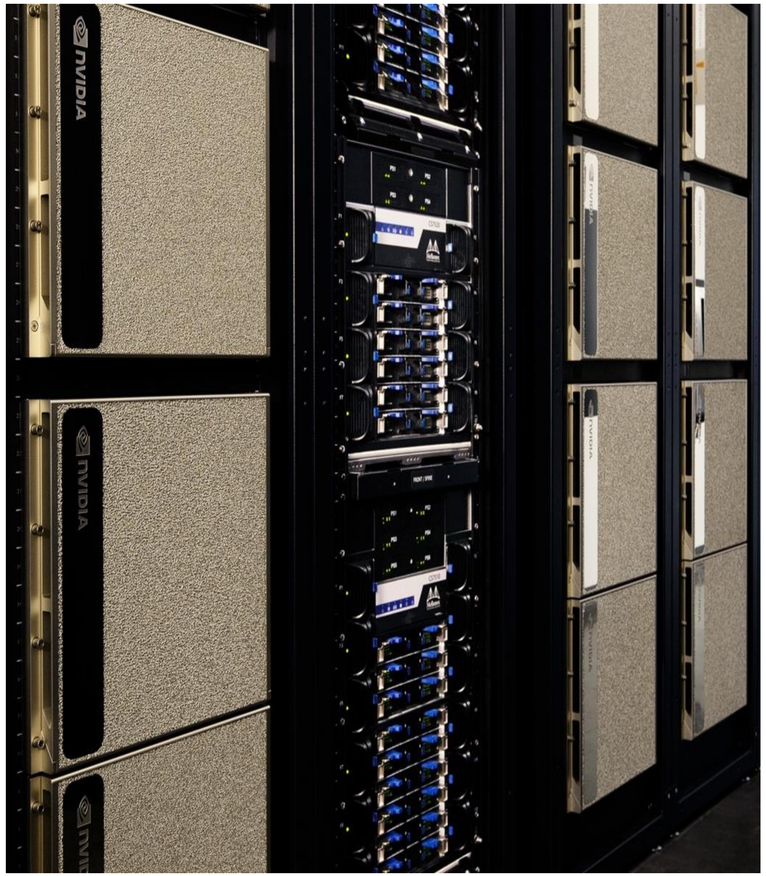
PYTHON ECOSYSTEM GOALS

Have Your Cake and Eat It Too

Productivity

```
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
```

Performance



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
```

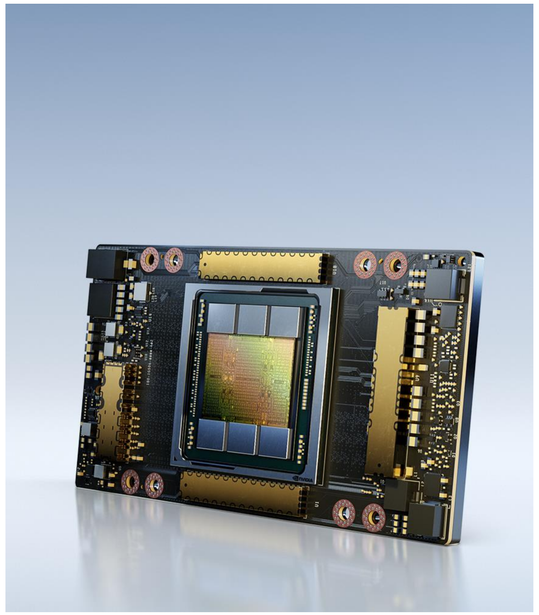
- Sequential semantics - no visible parallelism or synchronization
- Name-based global data – no partitioning
- Composable – can combine with other libraries and datatypes

PERFORMANCE

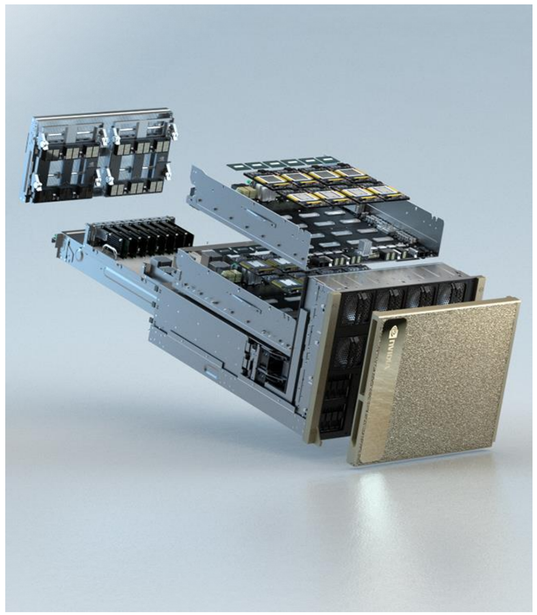
Transparent Acceleration

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

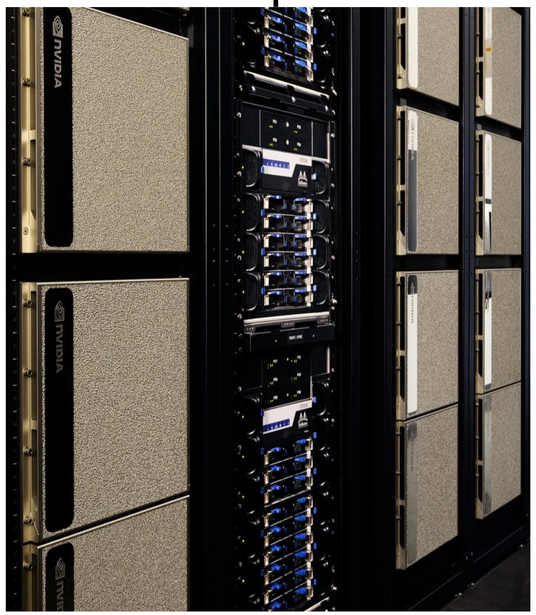
GPU



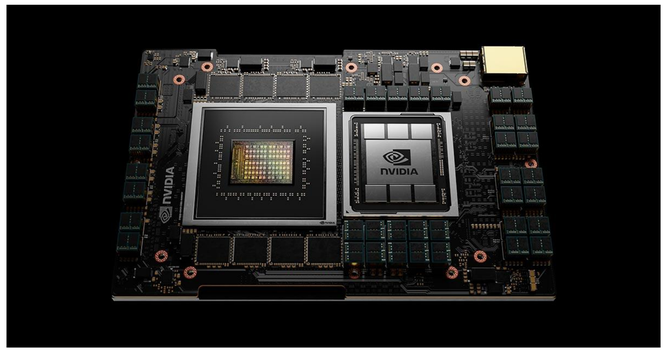
DGX-2



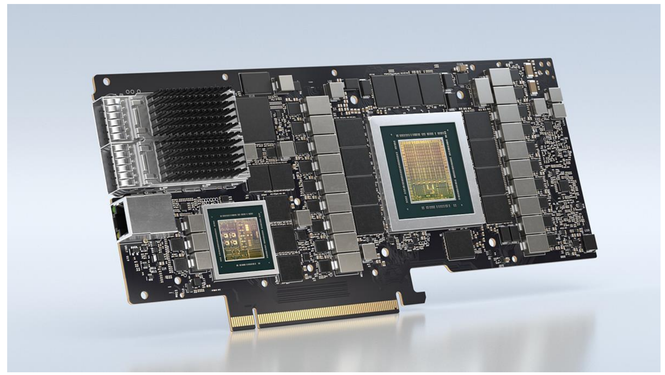
DGX SuperPod



Grace CPU

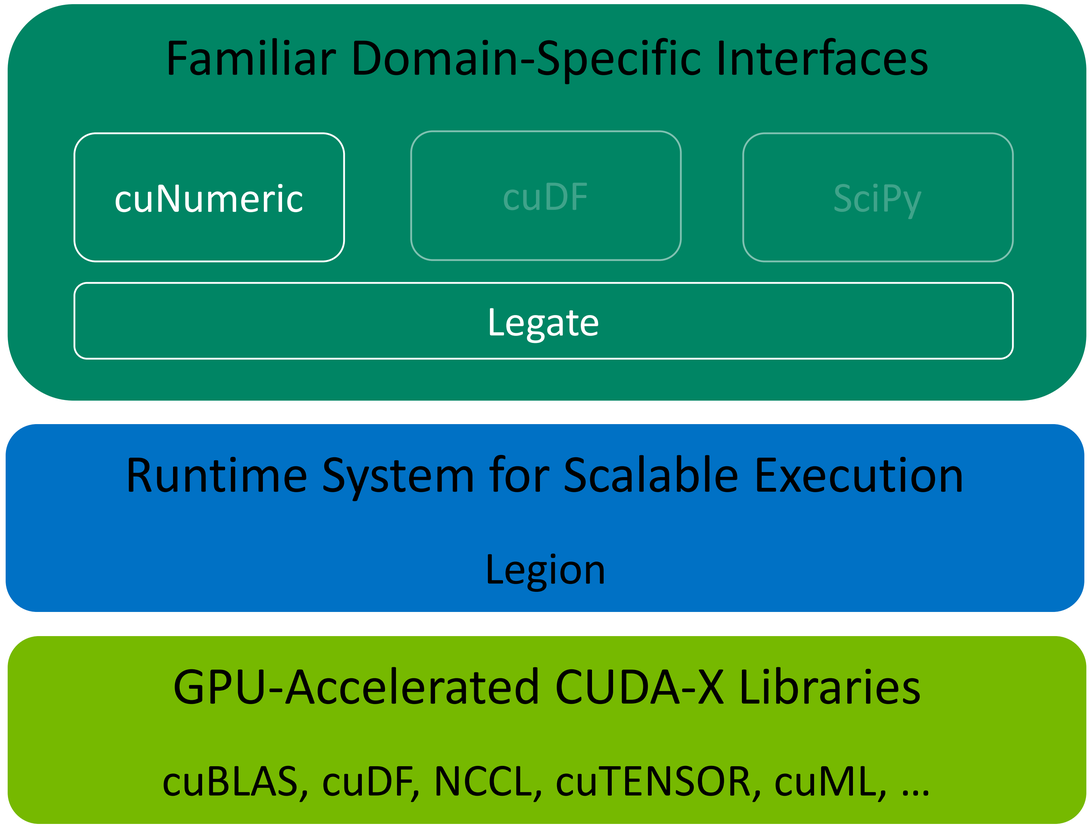


DPU



LEGATE ECOSYSTEM ARCHITECTURE

Scalable implementations of popular domain-specific APIs



CUNUMERIC

Automatic NumPy Acceleration and Scalability

cuNumeric

CuNumeric transparently accelerates and scales existing Numpy workloads

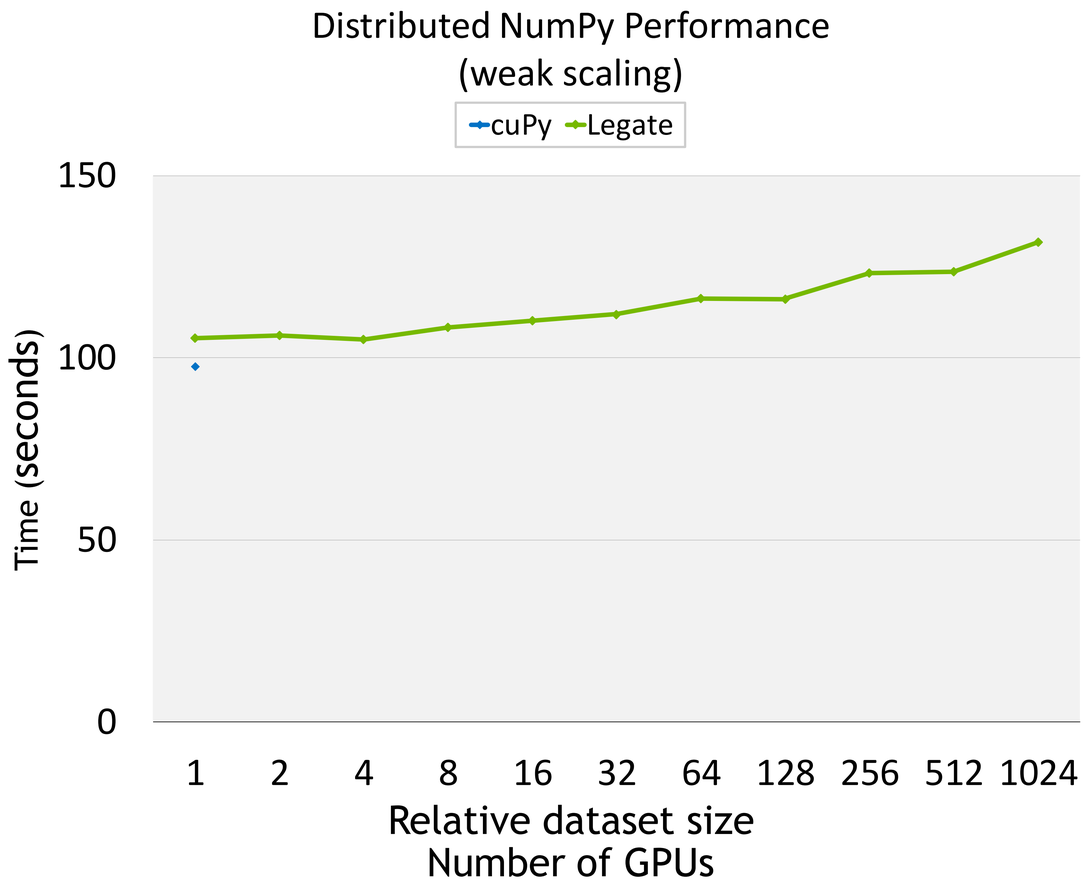
Program from the edge to the supercomputer in Python by changing 1 import line

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

Alpha release available at github.com/nv-legate

```
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)
```

…

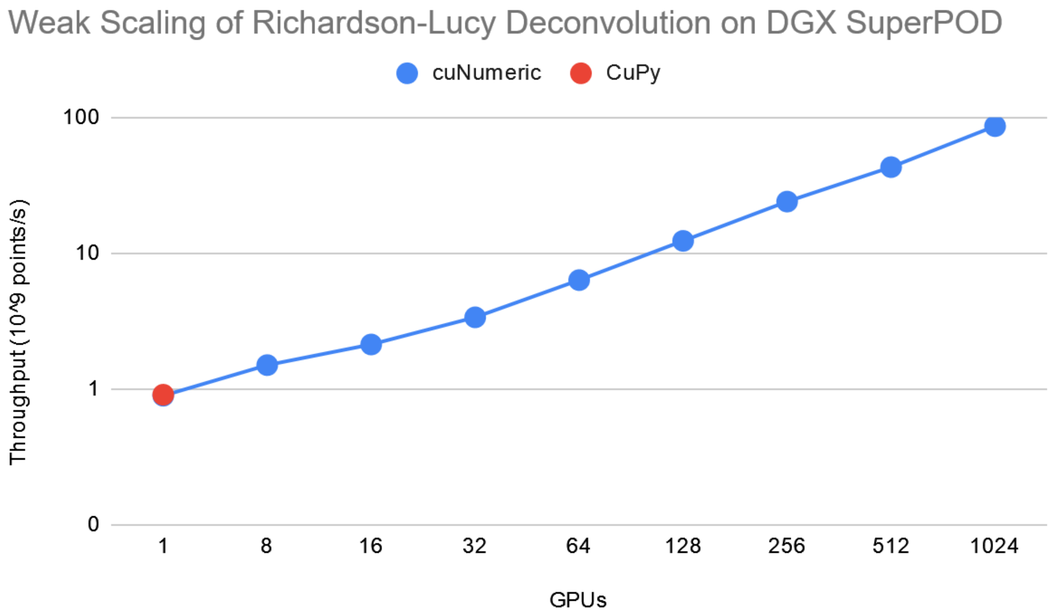


Extracted from “CFD Python” course at https://github.com/barbagroup/CFDPython
Barba, Lorena A., and Forsyth, Gilbert F. (2018). CFD Python: the 12 steps to Navier-Stokes equations. *Journal of Open Source Education*, **1**(9), 21, https://doi.org/10.21105/jose.00021

PERFORMANCE RESULTS

Microscopy Demo with Richardson-Lucy Deconvolution

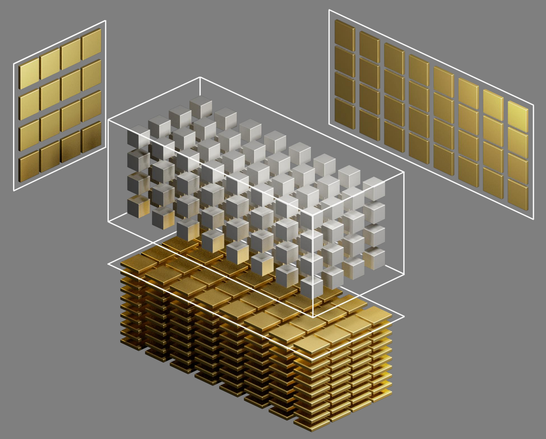
```
def richardson_lucy(image, psf, num_iter=50,
                    clip=True, filter_epsilon=None):
  float_type = _supported_float_type(image.dtype)
  image = image.astype(float_type, copy=False)
  psf = psf.astype(float_type, copy=False)
  im_deconv = np.full(image.shape, 0.5, dtype=float_type)
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



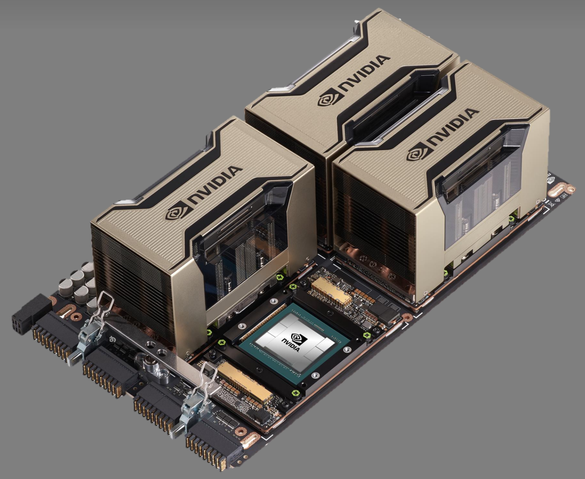
NVIDIA PERFORMANCE LIBRARIES

Major Directions

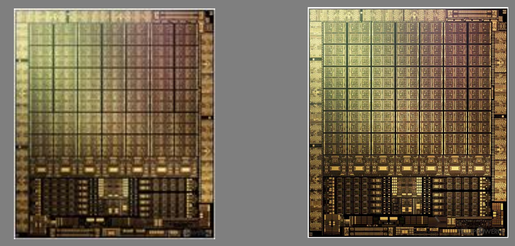
Seamless Acceleration
Tensor Cores, Enhanced L2$ & SMEM



Scaling Up
Multi-GPU and Multi-Node Libraries

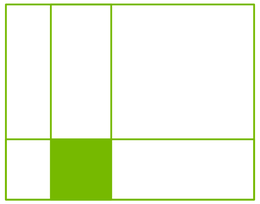


Composability
Device Functions

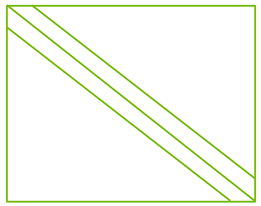


NVIDIA MATH LIBRARIES

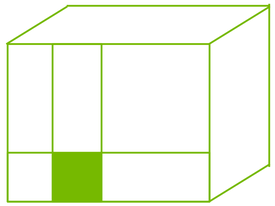
Linear Algebra, FFT, RNG and Basic Math



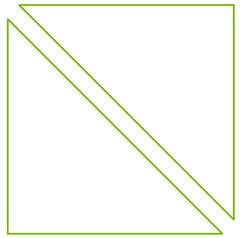
cuBLAS



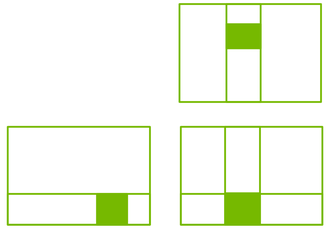
cuSPARSE



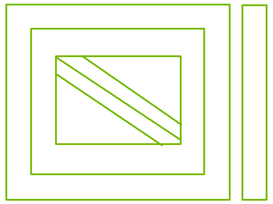
cuTENSOR



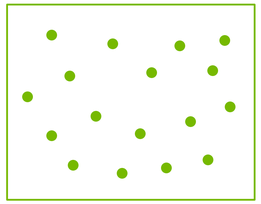
cuSOLVER



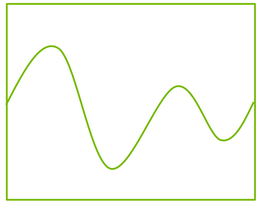
CUTLASS



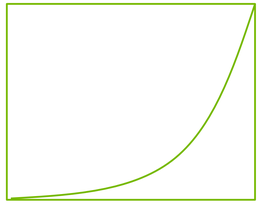
AMGX



cuRAND



cuFFT



CUDA Math API

TENSOR CORE SUPPORT IN MATH LIBRARIES

High-level overview of supported functionality by each library

Library and Tensor Core Functionality	INT4		INT8		FP16		BF16		TF32		FP64
	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense
cuBLAS & cuBLASLt Dense GEMM			✓		✓		✓		✓		✓
cuTENSOR Tensor Contractions					✓		✓		✓		✓
cuSOLVER Linear System Solvers					✓		✓		✓		✓
cuSPARSE Block-SpMM			✓		✓		✓		✓		✓
cuSPARSELt SpMM				✓		✓		✓		✓	
CUTLASS Dense GEMM and SpMM	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
CUTLASS Convolutions	✓		✓		✓		✓		✓		

CUBLAS

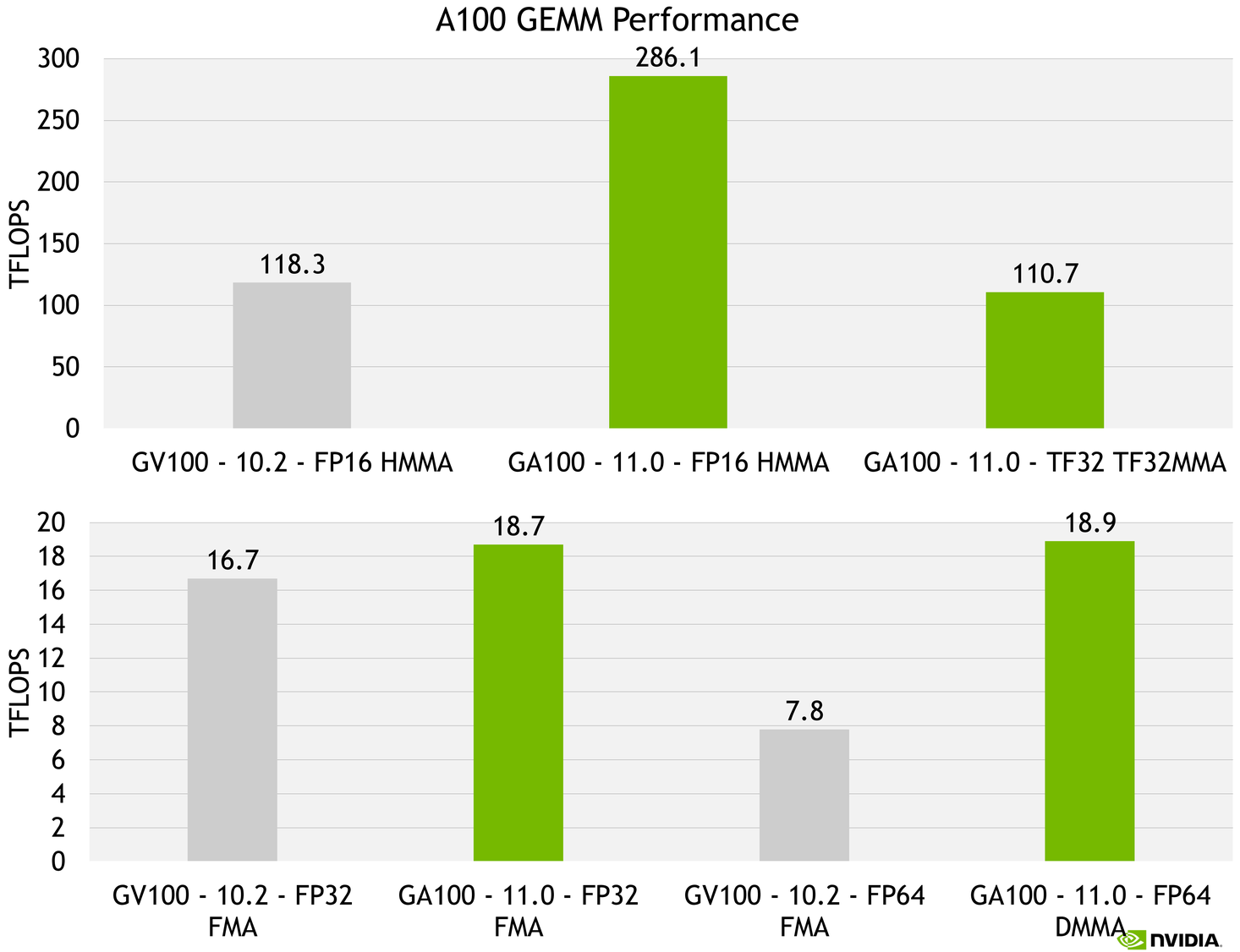
GPU Optimized BLAS Implementation

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



A100 TENSOR CORES IN LIBRARIES

cuSOLVER Linear Solvers

Dense and Sparse Factorizations & Solvers

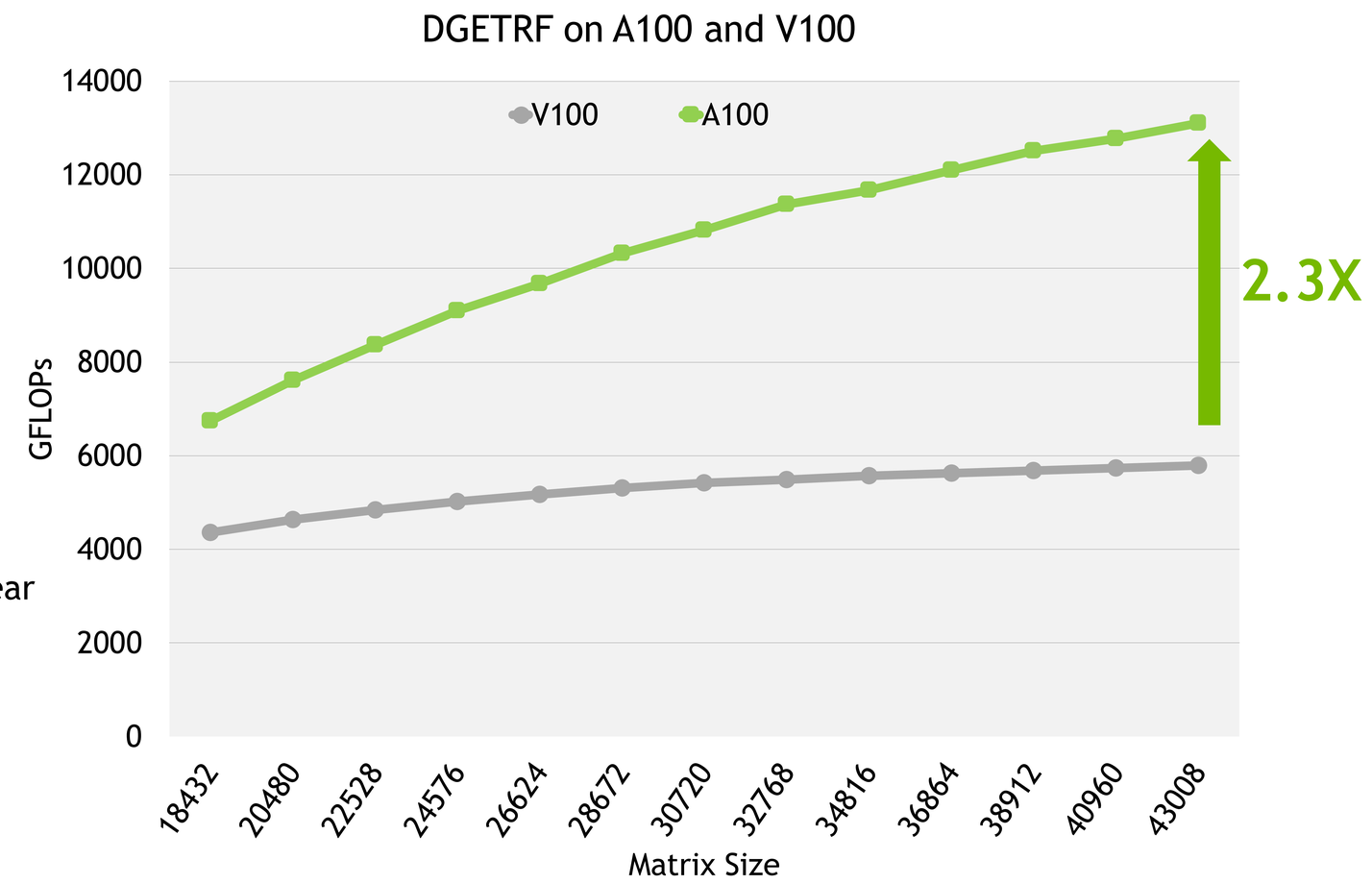
- LU, Cholesky, QR
- Symmetric and Generalized Eigensolvers
- Tensor Core Accelerated Iterative Refinement Solvers
- Multi GPU Support

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 2.8X Speedup



A100 data collected with pre-production hardware and software

CUSPARSE

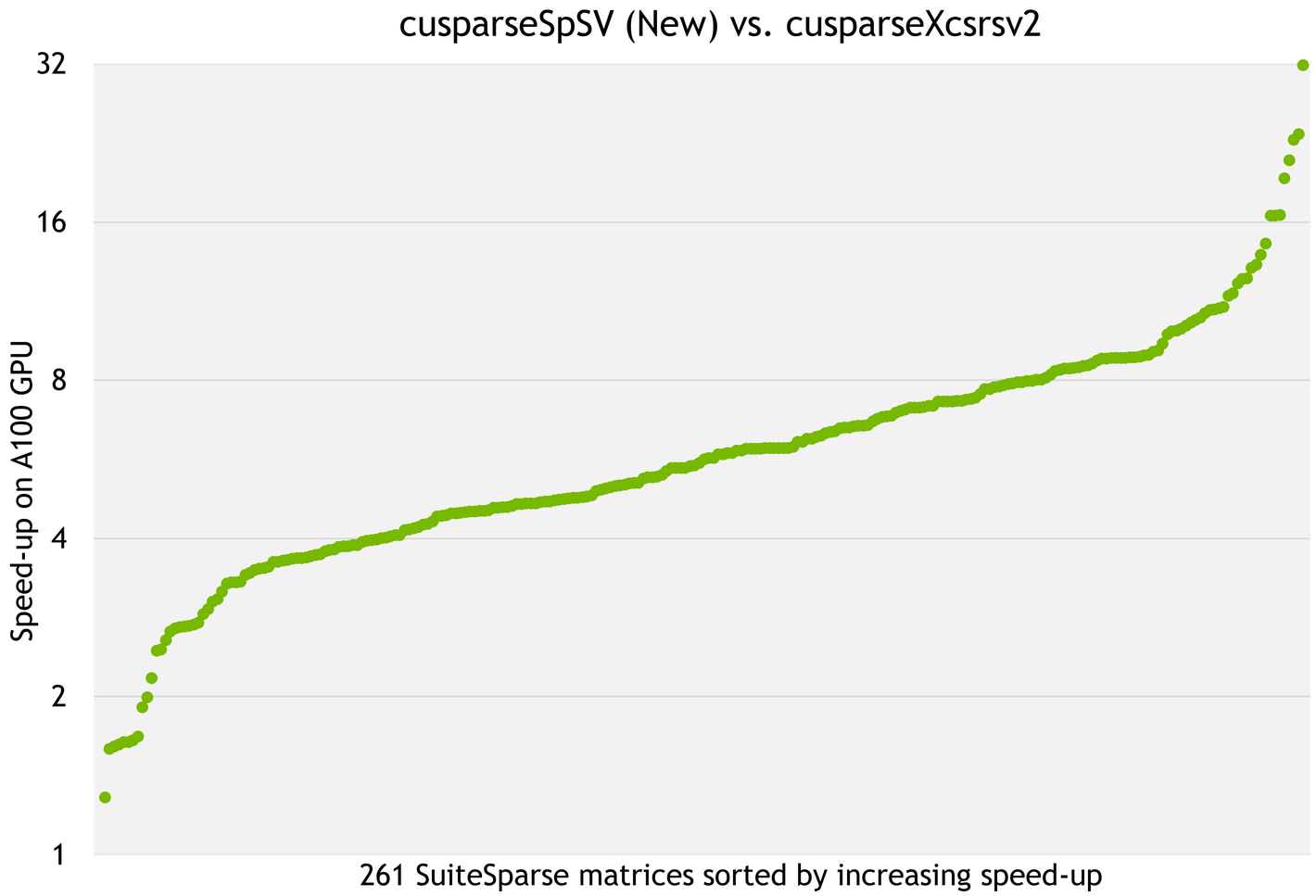
GPU Optimized Sparse BLAS

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 11.2U1 performance collected on GA100 @ 1095,1213GHz
- MKL 2020 performance collected on Xeon Platinum 8280 @ 2.7GHz

CUFFT

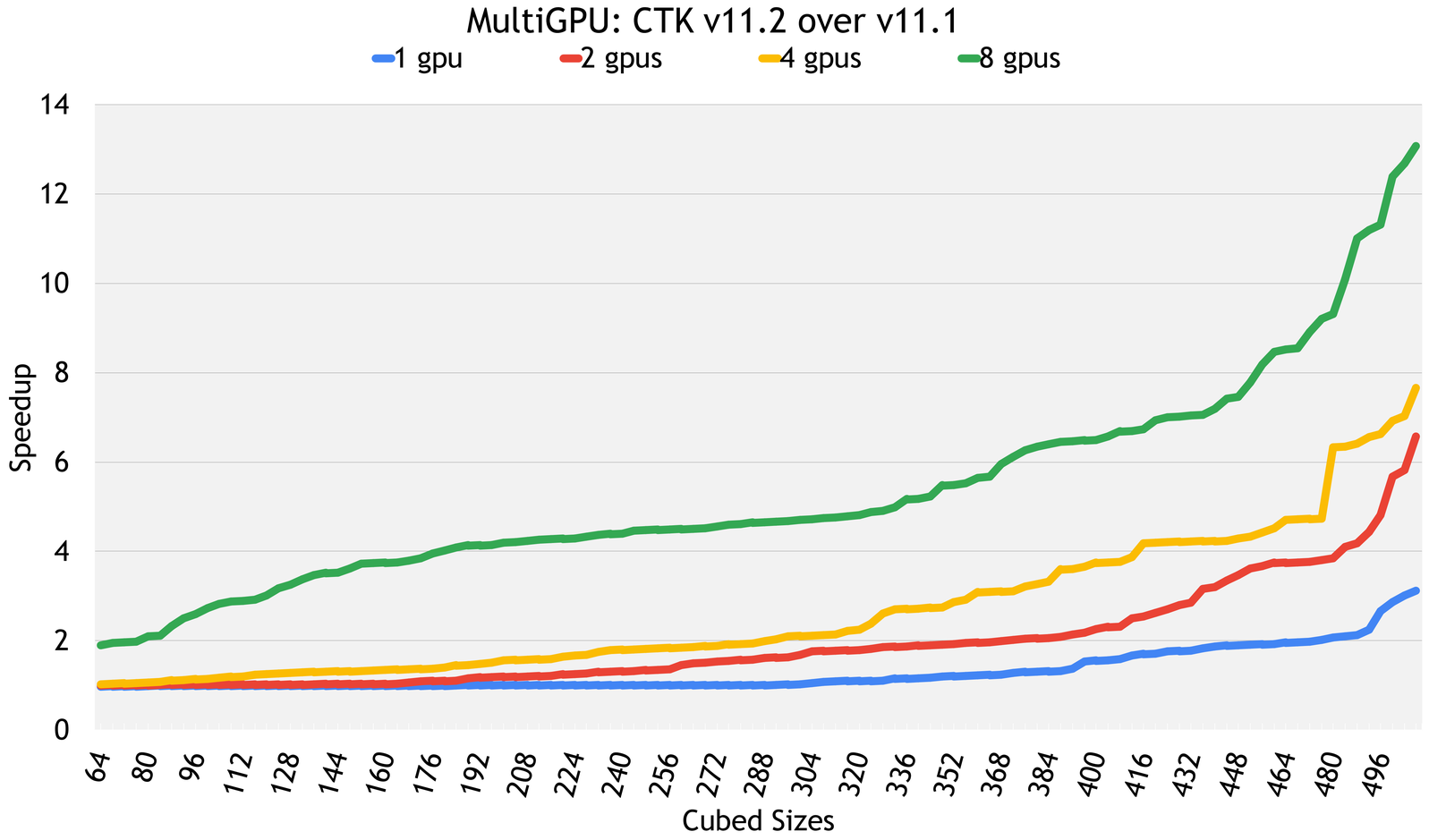
GPU Optimized Fast Fourier Transforms

GPU Optimized FFT

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

Accelerating a wide range of applications

- HPC & Scientific Computing
- Data Analytics



* Benchmarks ran on DGXA100

CUTENSOR

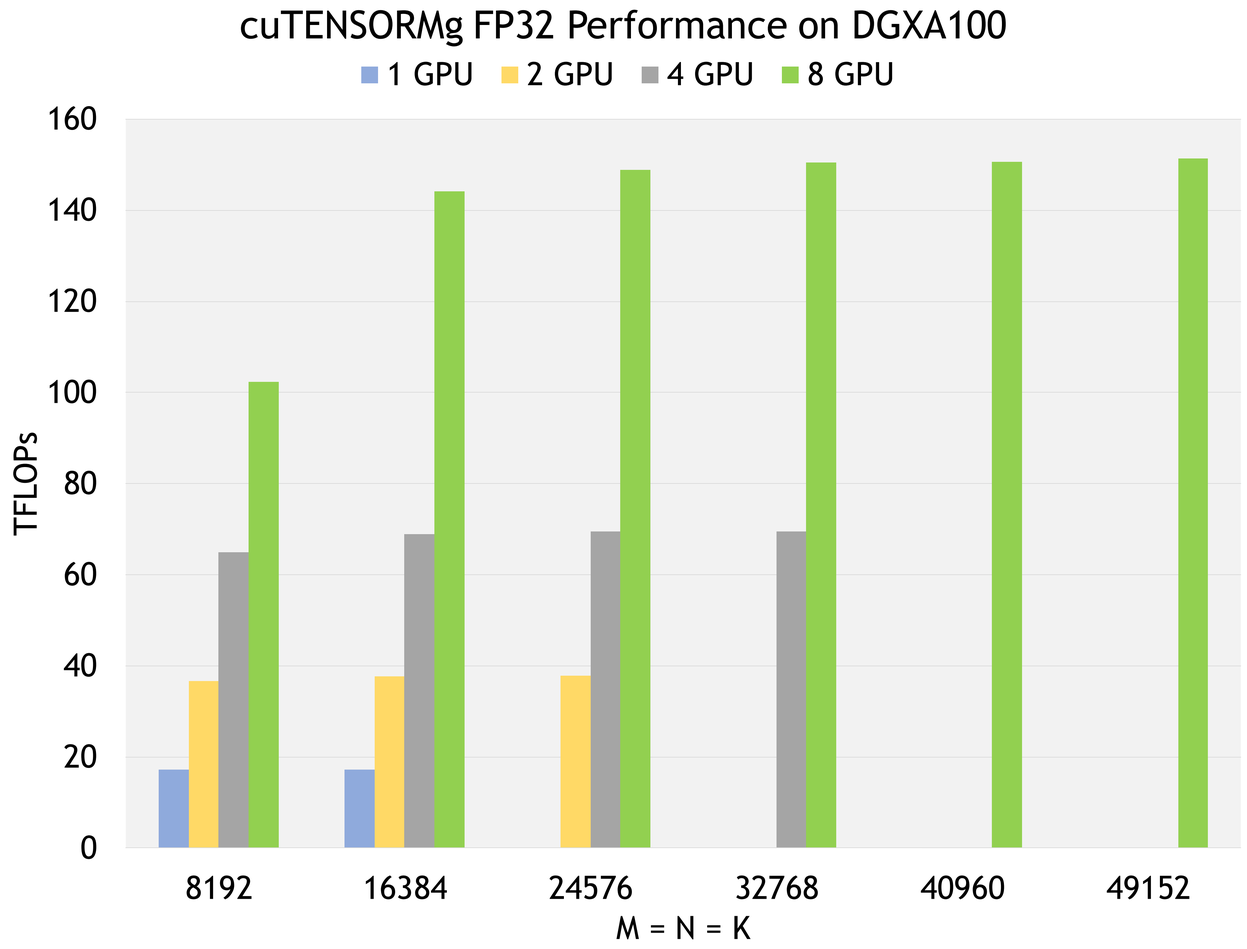
A New High Performance CUDA Library for Tensor Primitives

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



MULTI-NODE MATH LIBRARIES

cuSOLVERMp: Dense Linear Algebra at scale

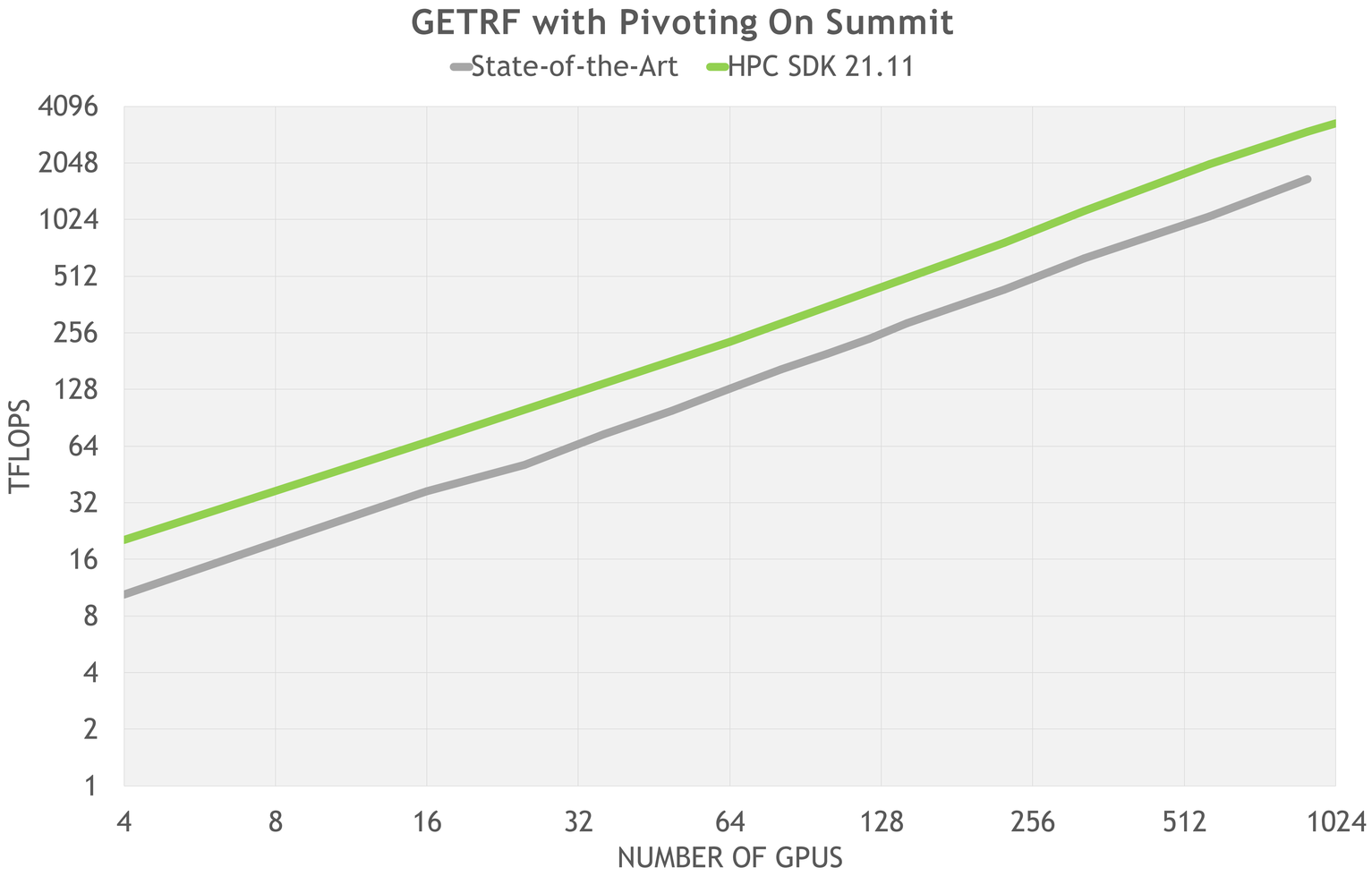
cuSOLVERMp

A distributed-memory multi-node and multiGPU solution for solving systems of linear equations at scale.

GA release available in HPC SDK 21.11

Initial release to support LU Decomposition, with and without pivoting, and Cholesky.

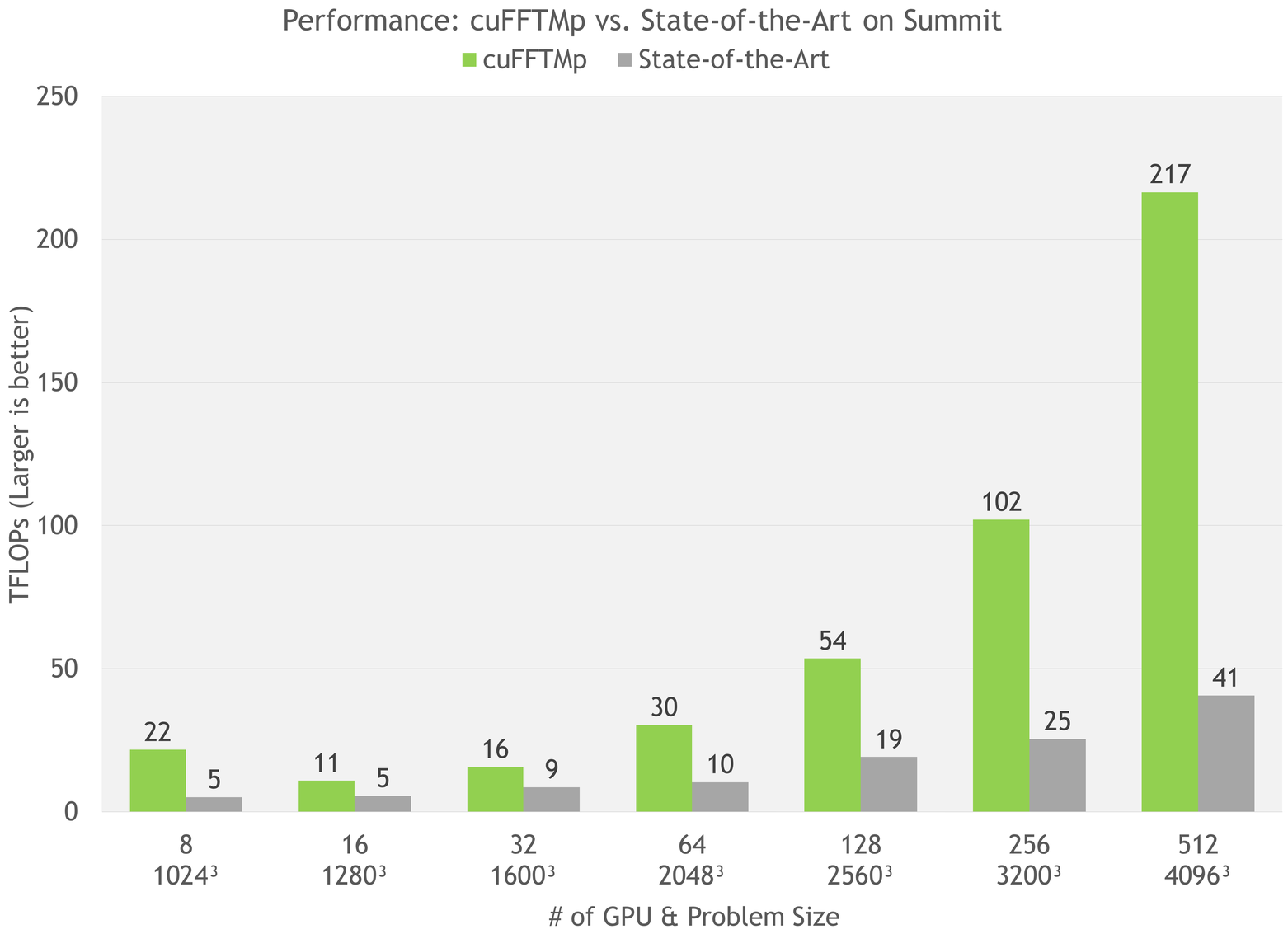
Multiple RHS coming soon!



* Problem size is increased with number of GPUs

MULTI-NODE MATH LIBRARIES

cuFFTMp: Fast Fourier Transforms at scale



cuFFTMp

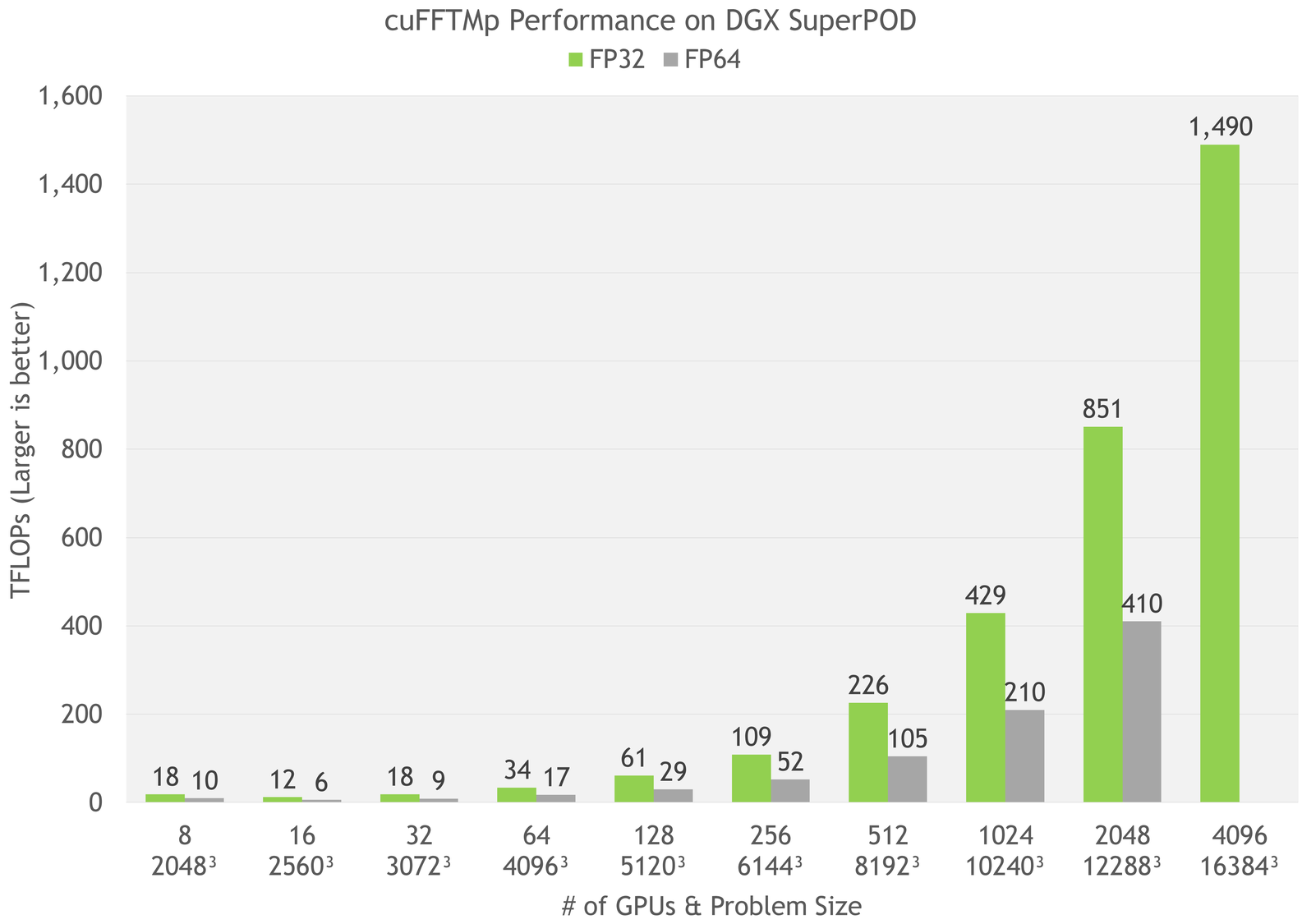
A distributed-memory multi-node and multiGPU solution for solving FFTs at scale.

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

Initial release to 2D & 3D with Slab composition

MULTI-NODE MATH LIBRARIES

cuFFTMp: Fast Fourier Transforms at scale



cuFFTMp

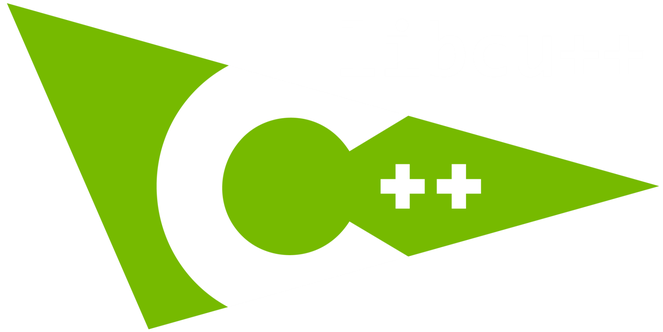
A distributed-memory multi-node and multiGPU solution for solving FFTs at scale.

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

Initial release to 2D & 3D with Slab composition

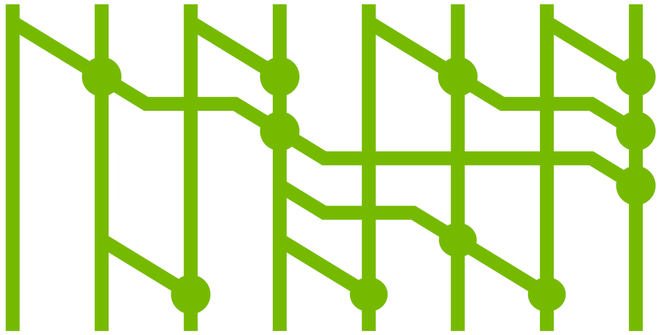
NVIDIA

C++ Core Compute Libraries



The Standard Library for your entire system
https://github.com/NVIDIA/libcudacxx

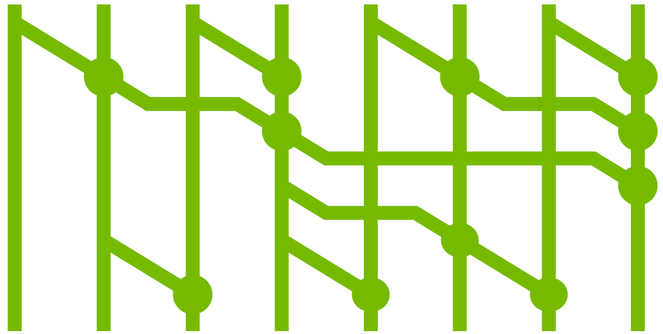
The C++ parallel algorithms library
https://github.com/NVIDIA/thrust



Cooperative primitives for CUDA C++
https://github.com/NVIDIA/cub

The C++ parallel algorithms library
https://github.com/NVIDIA/thrust

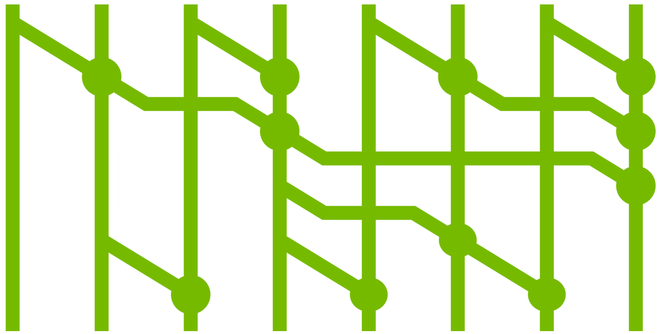
- High-Level Container Classes
  - Host_Vector
  - Device_Vector
- High-Level Algorithms
  - Transform
  - Fill
  - Copy
  - …
- Iterator Classes
  - Counting Iterator
  - Constant Iterator
  - …



Cooperative primitives for CUDA C++
https://github.com/NVIDIA/cub

- Parallel Communication Primitives
  - Warp-wide Collectives
  - Blow-wide Collectives
  - Device-wide Primatives
- Utility Operators
  - Iterators
  - Thread/Block I/O
  - PTX Intrinsics

The C++ parallel algorithms library
https://github.com/NVIDIA/thrust



Cooperative primitives for CUDA C++
https://github.com/NVIDIA/cub

```
thrust::universal_vector<float> h;
thrust::universal_vector<float> d;

thrust::event e = thrust::async::copy(
  par, h.begin(), h.end(), d.begin());
thrust::future<float> f = thrust::async::reduce(
  par.after(e), y.begin(), y.end());

…

float r = f.get();
```

```
template <typename R>
__global__ void f(R, …) {
  __shared__ typename R::TempStorage tmp;
  int local_s(…);
  int s = R(tmp).Sum(local_s);
}

f<<<32, 1>>>(cub::WarpReduce<int>{}, …);
f<<<128, 1>>>(cub::BlockReduce<int, 128>{}, …);
```

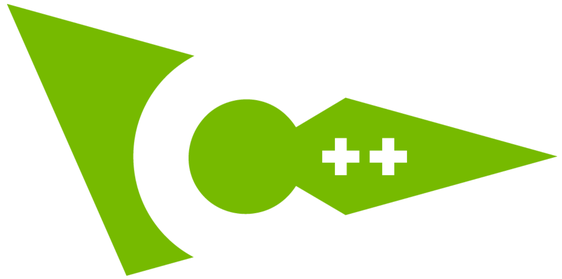
LIBCU++: A GPU-ENABLED STL

Host Compiler's Standard Library	
`#include <…>`	ISO C++, `__host__` only.
`std::`	Complete, strictly conforming to Standard C++.

`#include <cuda/std/…>`	CUDA C++, `__host__ __device__`.
`cuda::std::`	Subset, strictly conforming to Standard C++.
`#include <cuda/…>`	CUDA C++, `__host__ __device__`.
`cuda::`	Conforming extensions to Standard C++.

libcu++

libcu++ does not interfere with or replace your host Standard Library.



The NVIDIA C++ Standard Library

https://github.com/NVIDIA/libcudacxx

1.0.0 (CUDA 10.2)

`atomic<T>` (SM60+)
Type Traits

1.1.0 (CUDA 11.0)

`atomic<T>::wait/notify` (SM70+)
`barrier` (SM70+)
`latch` (SM70+)
`*_semaphore` (SM70+)
`cuda::memcpy_async` (SM70+)
`chrono::` Clocks & Durations
`ratio<Num, Denom>`

1.2.0 (CUDA 11.1)

`cuda::pipeline` (SM80+)

1.3.0 (CUDA 11.2)

`tuple<T0, T1, ...>`

1.4.1 (CUDA 11.3)

`complex`
`byte`
`chrono::` Dates & Calendars

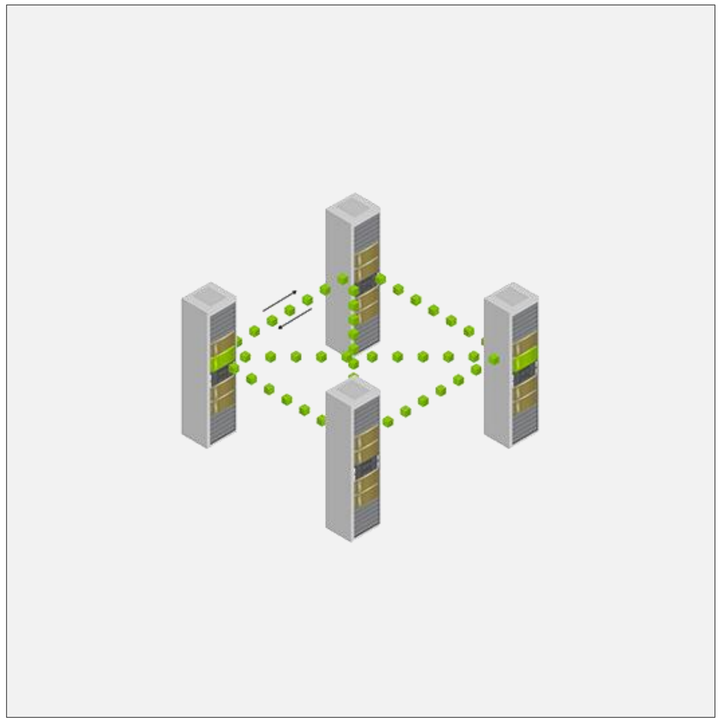
2.0.0

`atomic_ref<T>` (SM60+)
Memory Resources & Allocators
`cuda::stream_view`

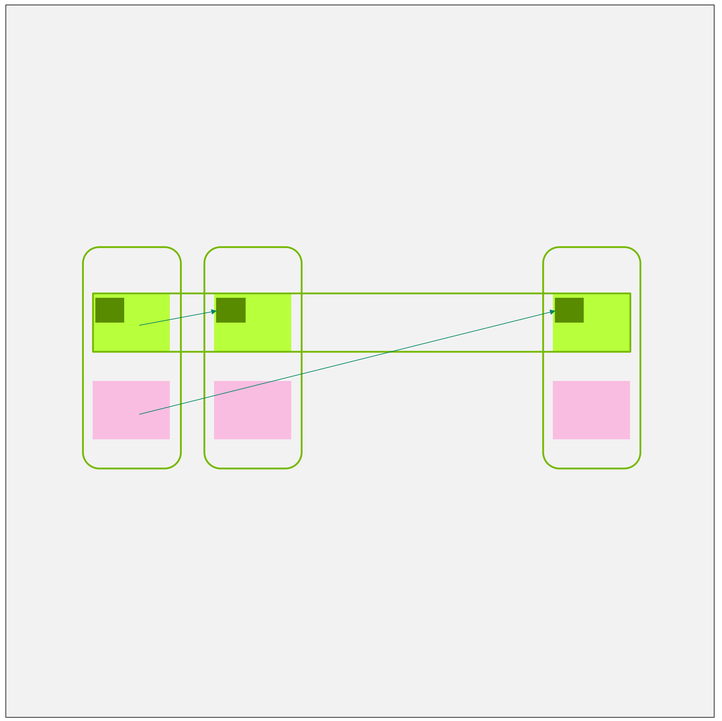
Future

Executors
Range Factories & Adaptors
Parallel Range Algorithms
Parallel Linear Algebra Algorithms
`mdspan<T, ...>`
…

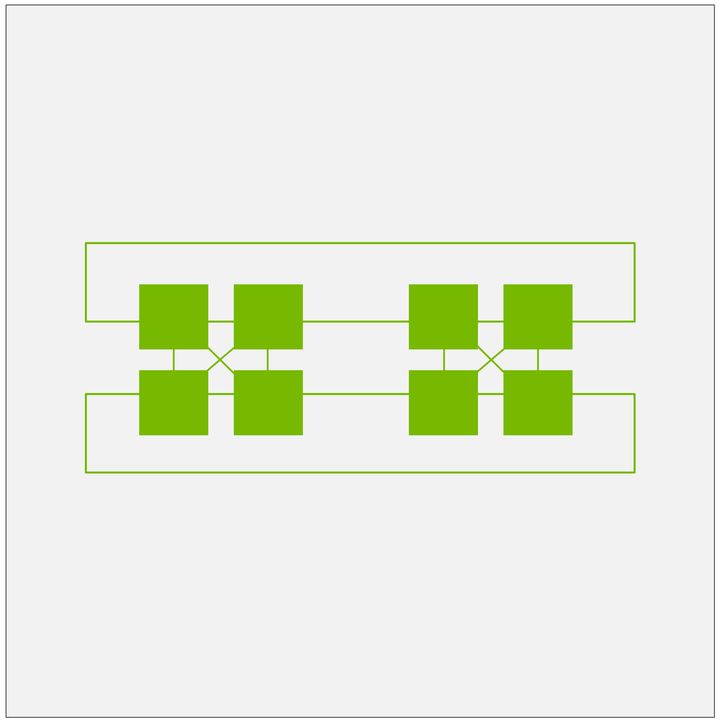
NVIDIA COMMUNICATION LIBRARIES



Optimized whole-system communications



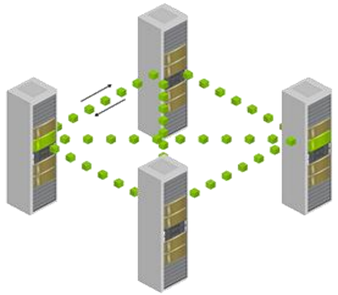
Low-latency PGAS programming



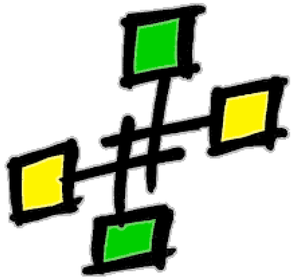
Multi-node collectives for accelerators

Multi-GPU Programming Models [S31050]

SOFTWARE AND ACCELERATION PACKAGES



HPC-X MPI



HPC-X OpenSHMEM

In-Network Computing

Unified Communication X

Unified Collective Communication

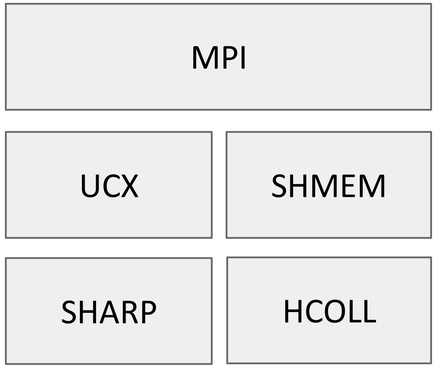
NCCL-SHARP and UCX Support

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



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

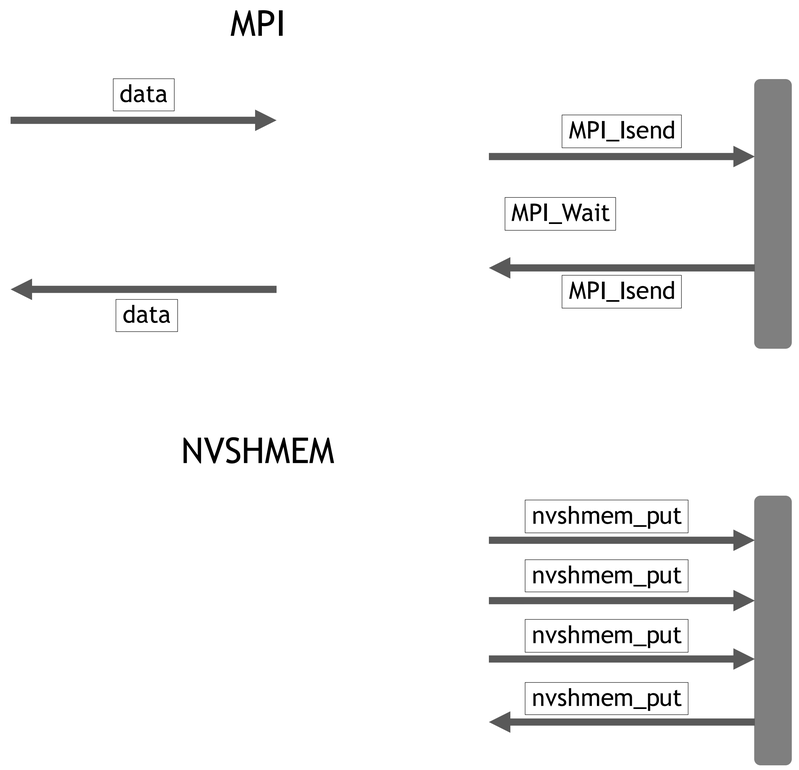
NVSHMEM

GPU Optimized SHMEM

- Initiate from CPU or GPU
- Initiate from within CUDA kernel
- Issue onto a CUDA stream
- Interoperable with MPI & OpenSHMEM

Application Impact

- LBANN, Kokkos/CGSolve, QUDA



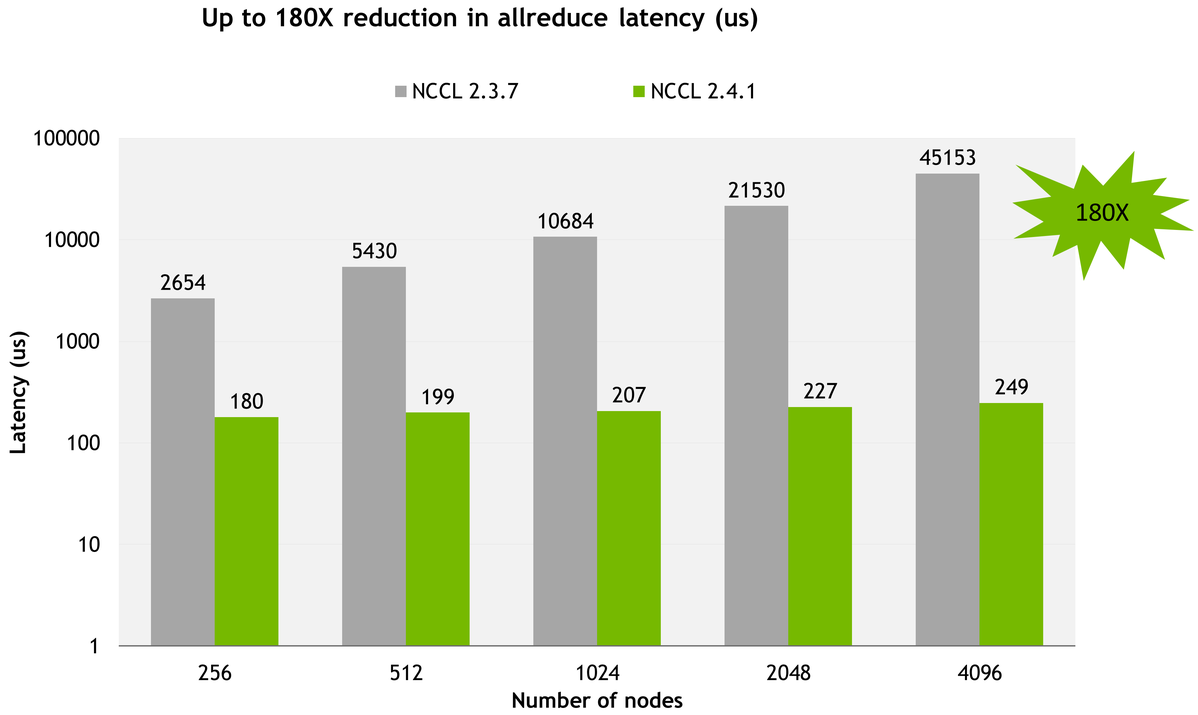
NCCL

GPU-Optimized Collectives

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

Impact

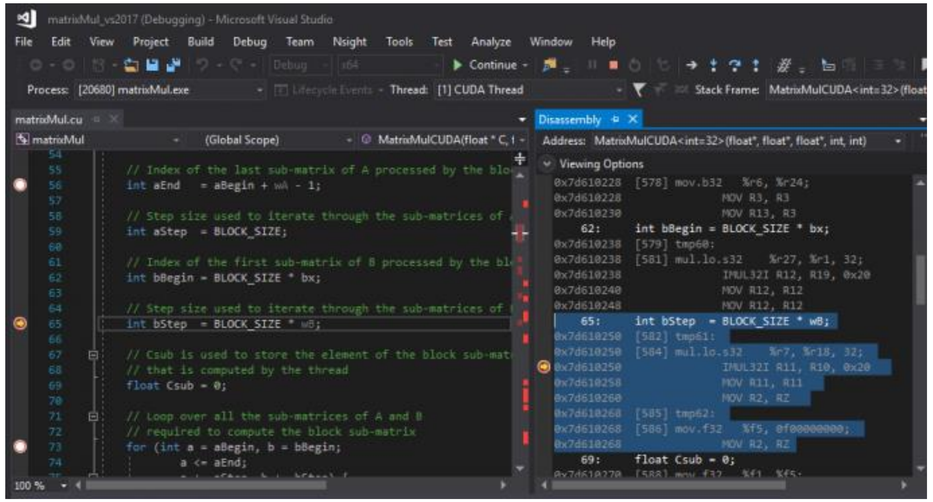
- Accelerates leading deep learning frameworks
- Adoption in HPC accelerating



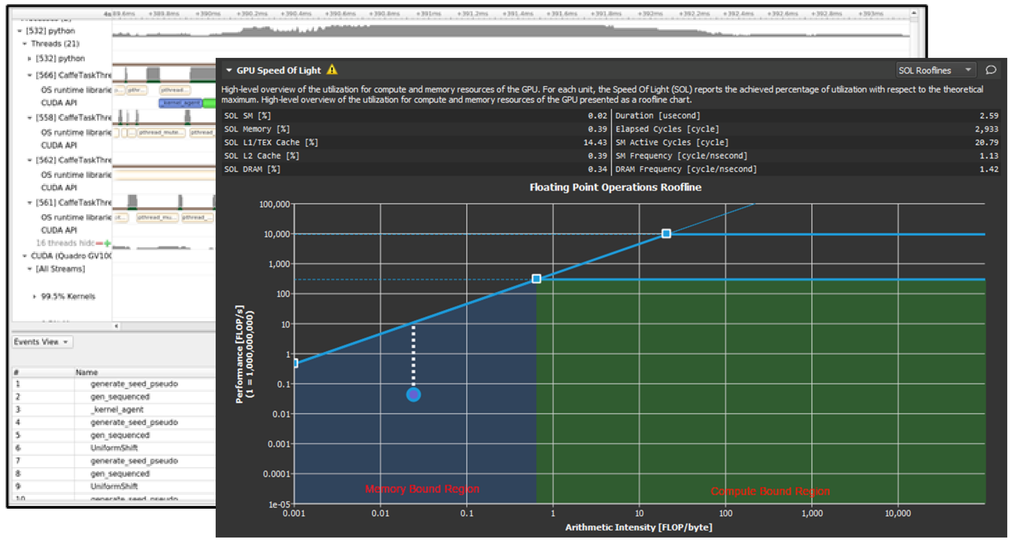
AllReduce 8-byte (float) latency, Summit Supercomputer at Oak Ridge National Lab
4096 nodes, 6xV100, 2x IB EDR

DEVELOPER TOOLS

Debuggers: cuda-gdb, Nsight Visual Studio Edition



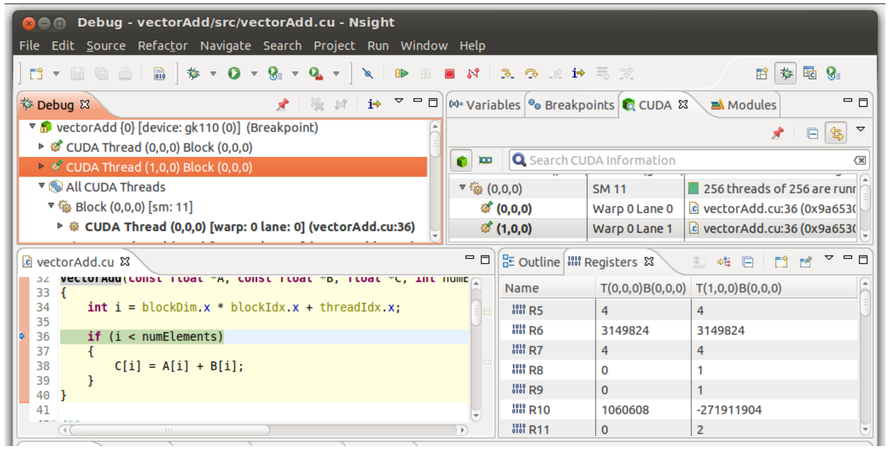
Profilers: Nsight Systems, Nsight Compute, CUPTI, NVIDIA Tools eXtension (NVTX)



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

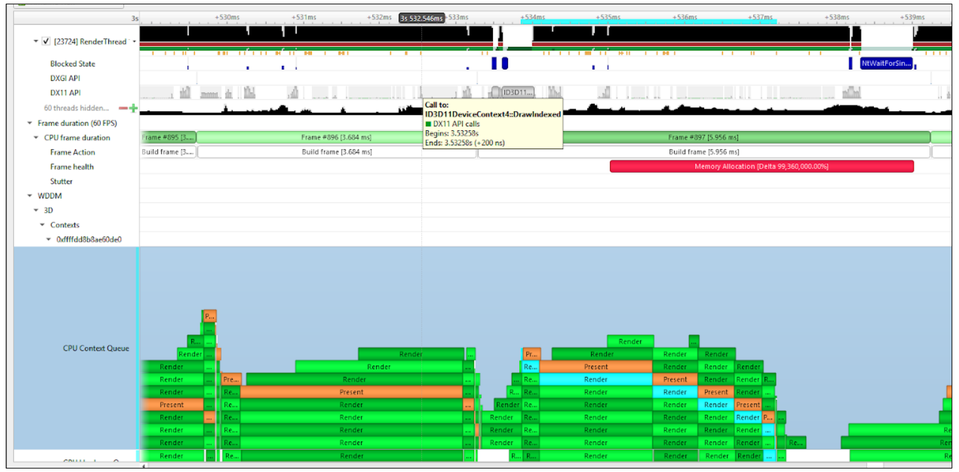
SYSTEM PROFILER

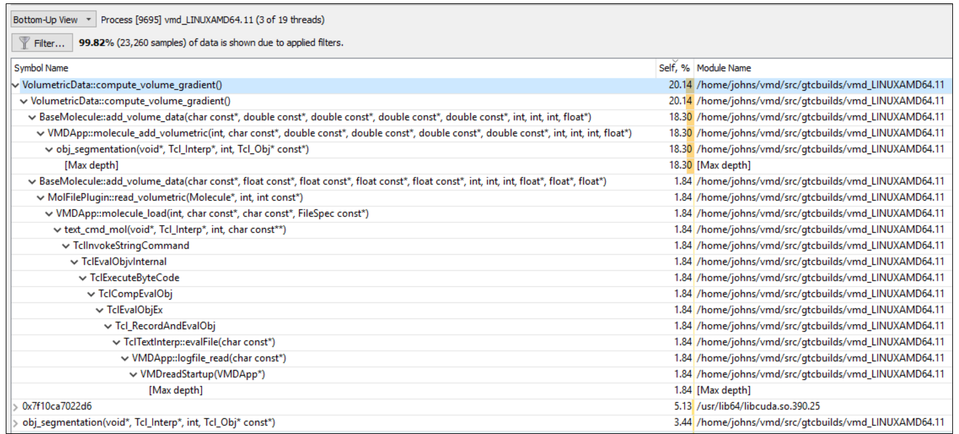
Key Features:

- 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

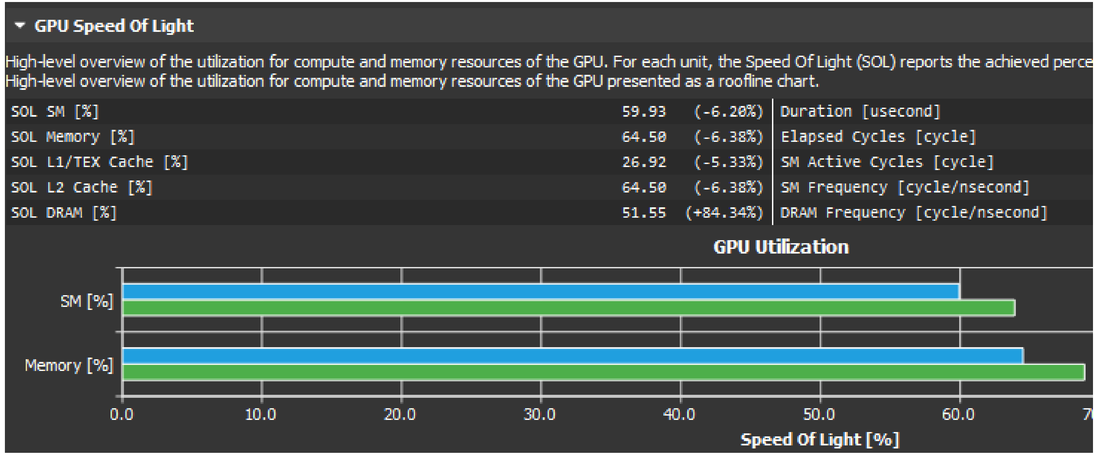
Key Features:

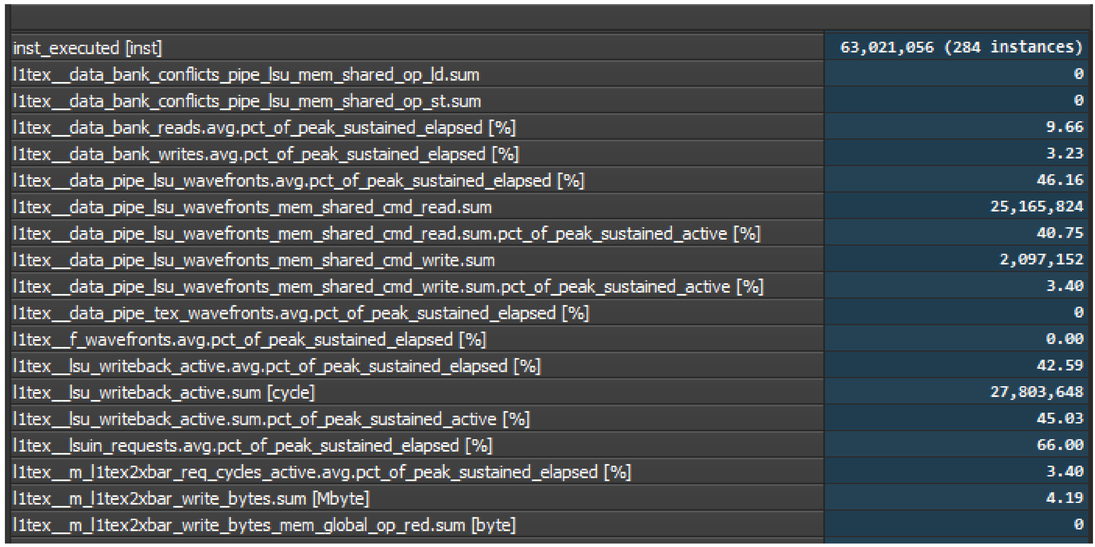
- Interactive CUDA API debugging and kernel profiling
- Built-in rules expertise
- Fully customizable data collection and display
- Command Line, Standalone, IDE Integration, Remote Targets

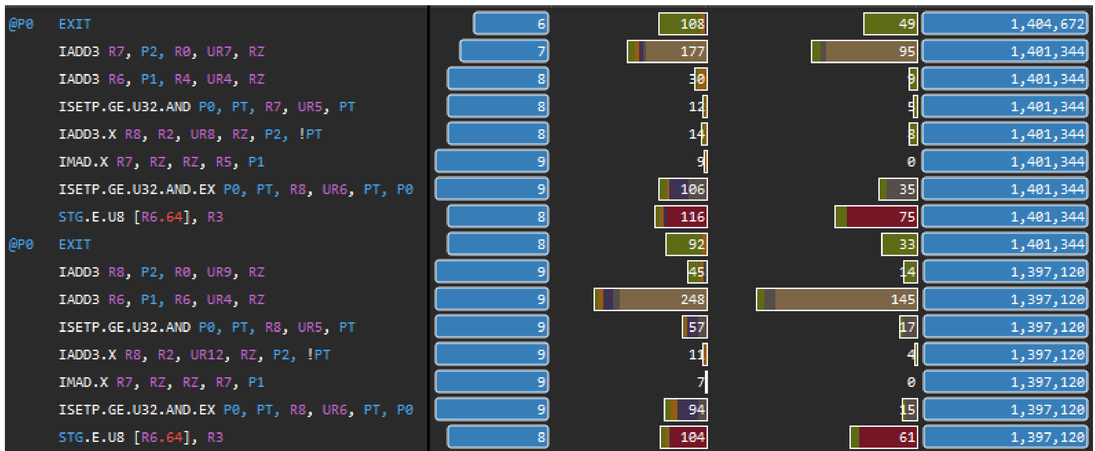
OS: Linux (x86, Power, Tegra, Arm SBSA), Windows, MacOSX (host only)

GPUs: Volta, Turing, Ampere GPUs

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



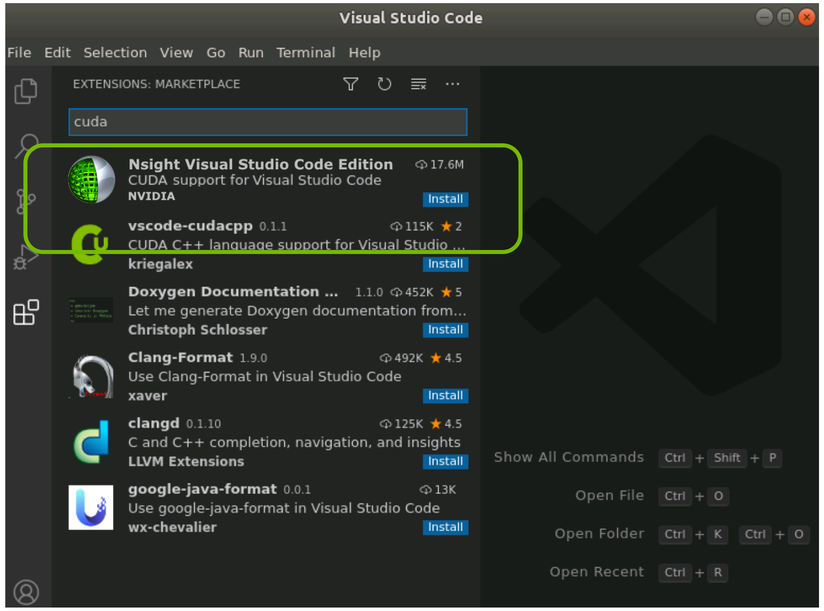


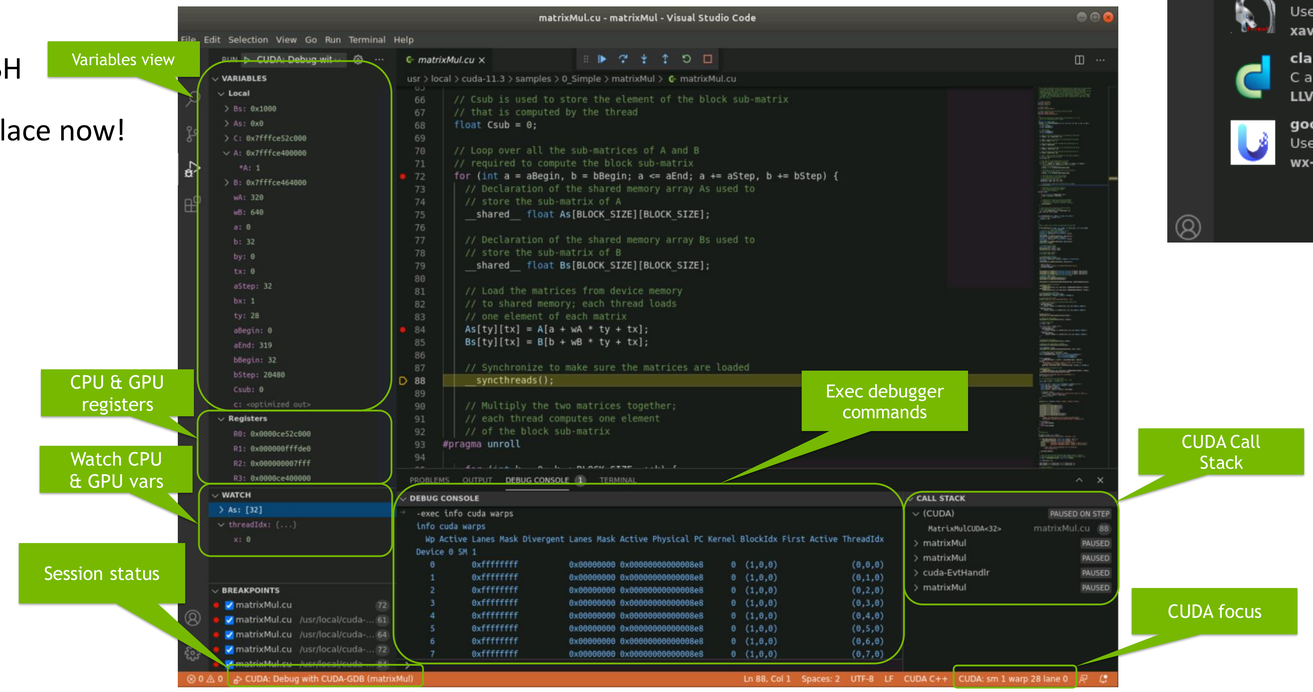


NSIGHT VISUAL STUDIO CODE EDITION

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!



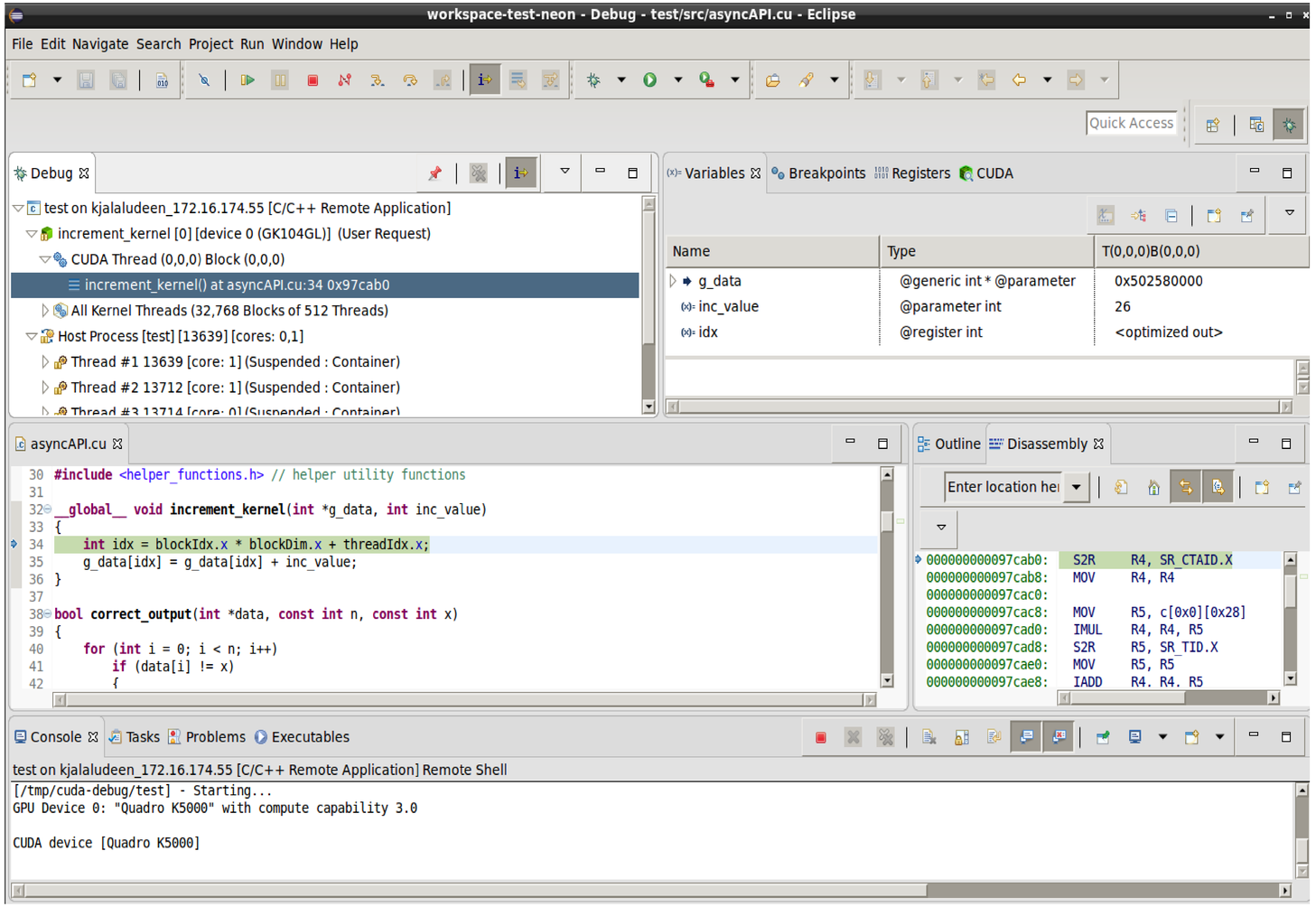


https://developer.nvidia.com/nsight-visual-studio-code-edition

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

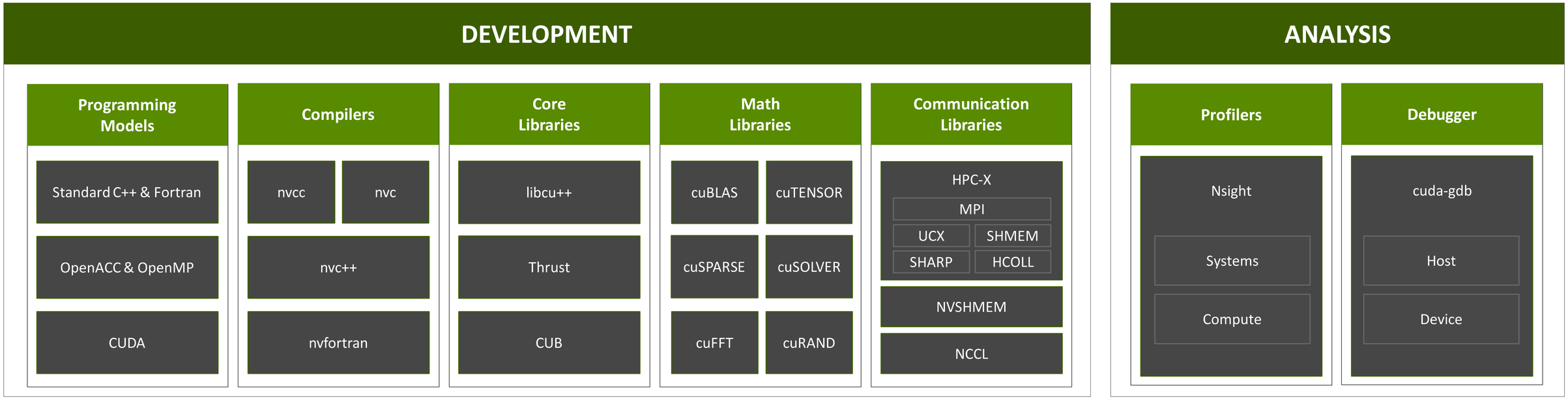
AUTOMATICALLY SCAN FOR BUGS AND MEMORY ISSUES

- 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

7-8 Releases Per Year | Freely Available