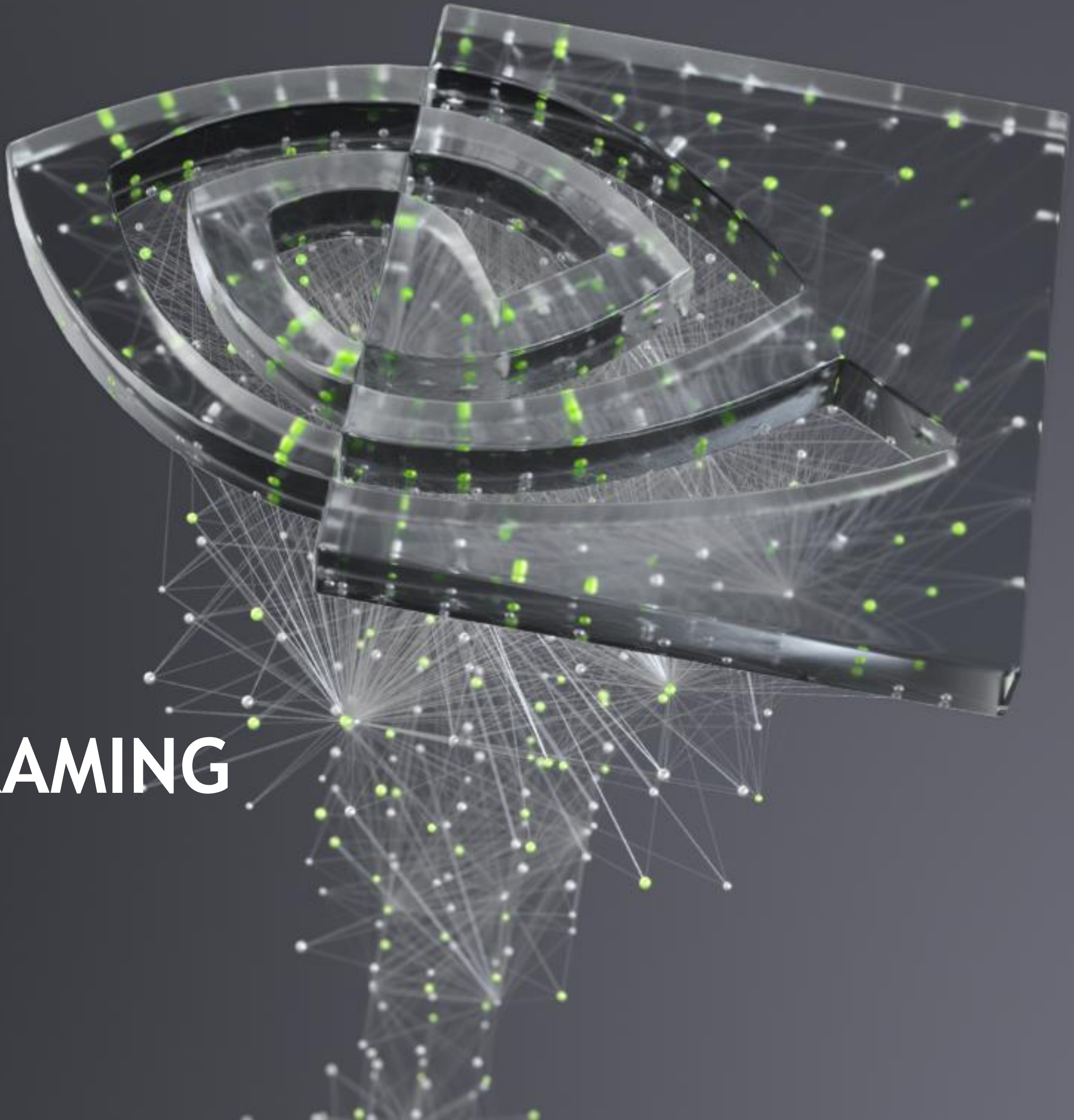
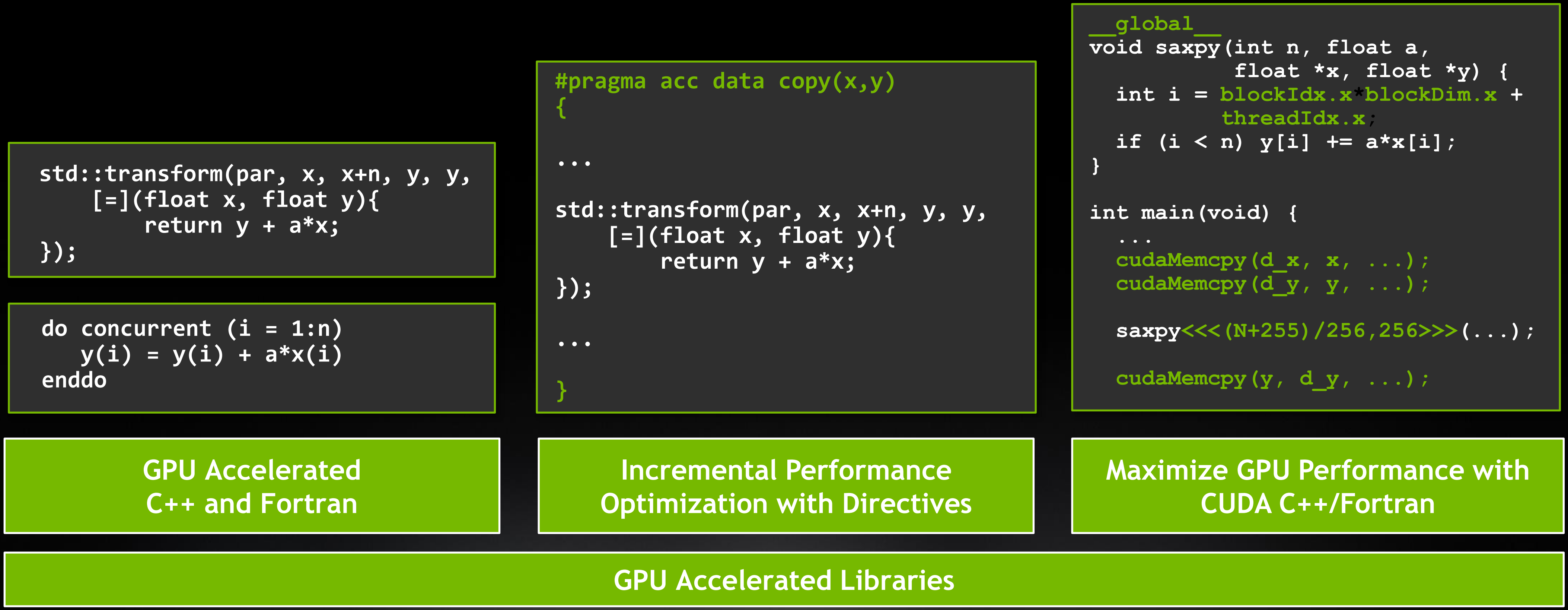
# GPU ACCELERATION PRINCIPLE AND PROGRAMING - HPC SDK

Ryan Jeng | Tech Lead, DevTech APAC



# GPU PROGRAMMING IN 2020 AND BEYOND

Math Libraries | Standard Languages | Directives | CUDA



```
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
```

GPU Accelerated C++ and Fortran

```
#pragma acc data copy(x,y)
{

...

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

...

}
```

Incremental Performance Optimization with Directives

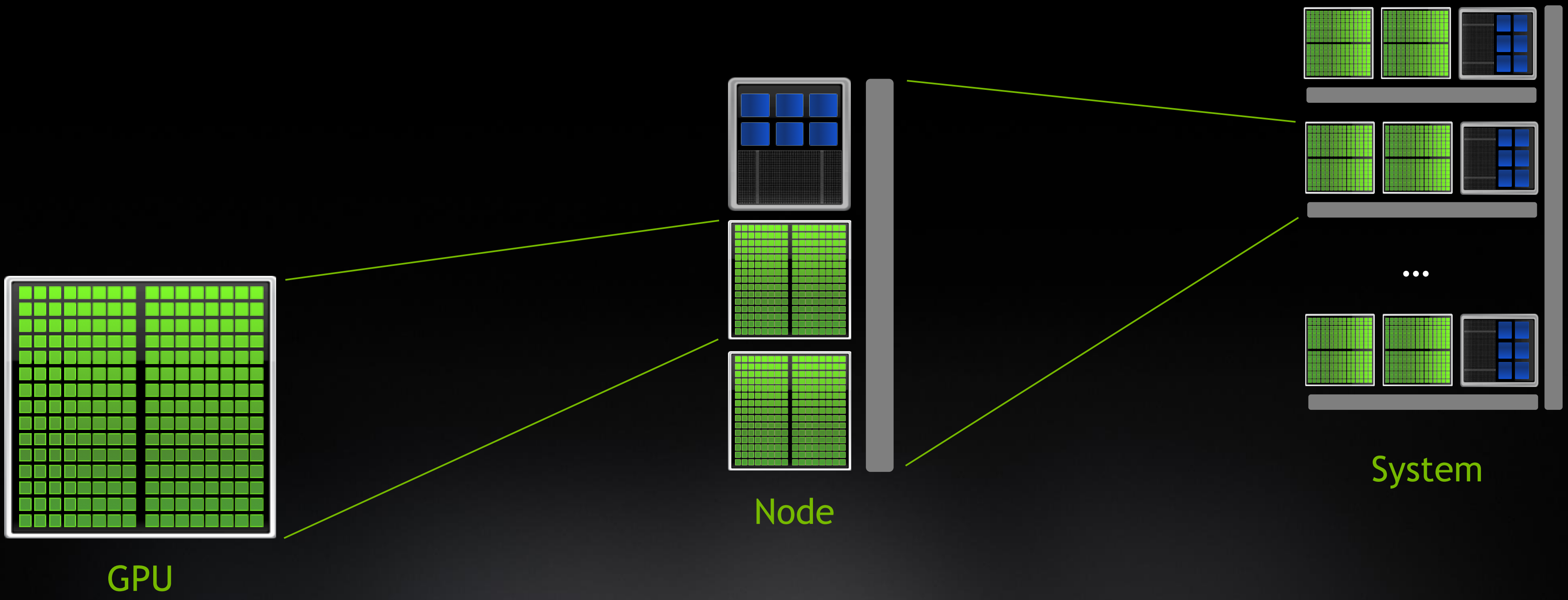
```
__global__
void saxpy(int n, float a,
           float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

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

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

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

Maximize GPU Performance with CUDA C++/Fortran

GPU Accelerated Libraries

# PROGRAMMING GPU-ACCELERATED HPC SYSTEMS

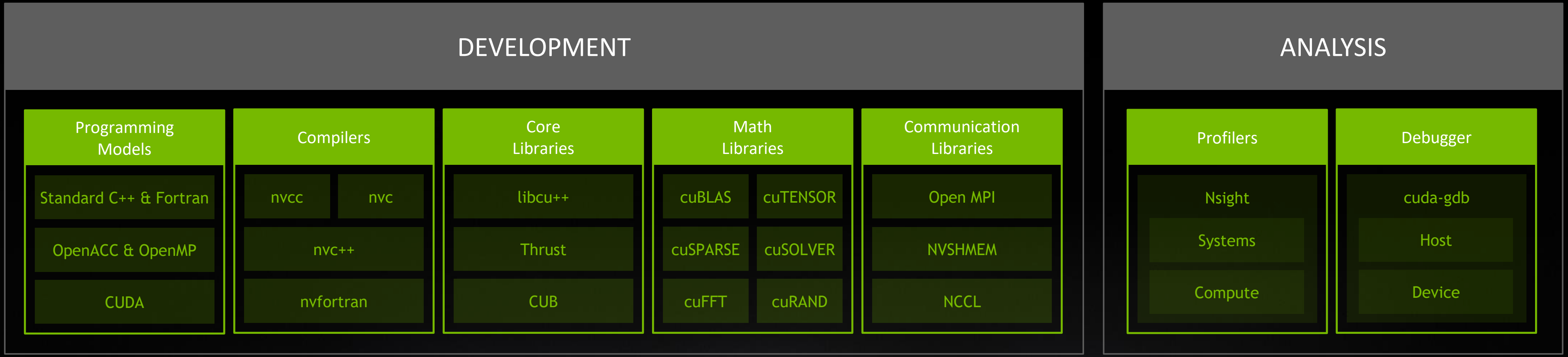
GPU | CPU | Interconnect



# AVAILABLE NOW: THE NVIDIA HPC SDK

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

## NVIDIA HPC SDK



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
  - Open MPI
  - NVSHMEM
  - NCCL

ANALYSIS

- Profilers
  - Nsight
    - Systems
    - Compute
- Debugger
  - cuda-gdb
    - Host
    - Device

Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect

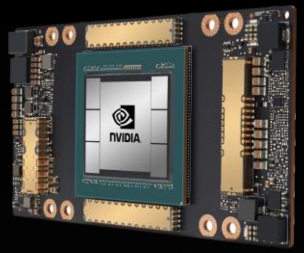
HPC Libraries | GPU Accelerated C++ and Fortran | Directives | CUDA

7-8 Releases Per Year | Freely Available

# HPC COMPILERS & STANDARD LANGUAGE PARALLELISM

# HPC COMPILERS

NVC | NVC++ | NVFORTRAN



Accelerated

A100

Automatic

Fortran

NVIDIA CUDA

C++

OpenACC

More Science, Less Programming

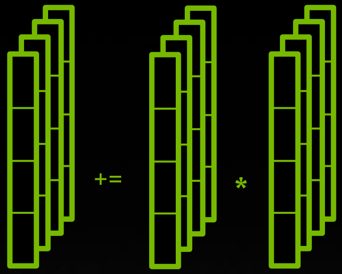
OpenMP

Programmable

Standard Languages

Directives

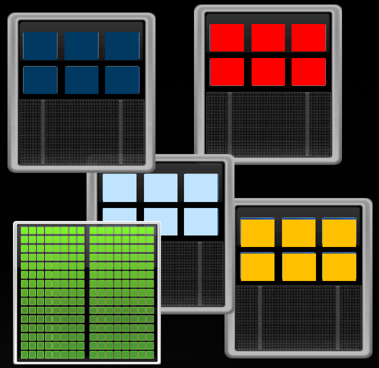
CUDA



Multicore

Directives

Vectorization



Multi-Platform

x86_64

Arm

OpenPOWER

# HPC PROGRAMMING IN ISO C++

ISO is the place for portable concurrency and parallelism

## 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

- Simplify launching and managing parallel work across CPUs and accelerators

### `std::mdspan/mdarray`

- HPC-oriented multi-dimensional array abstractions.

### 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`

# HPC PROGRAMMING IN ISO C++

## C++ Parallel Algorithms

```
std::sort(std::execution::par, c.begin(), c.end());

std::unique(std::execution::par, c.begin(), c.end());
```

- Introduced in C++17
- Parallel and vector concurrency via execution policies

  `std::execution::par, std::execution::par_seq, std::execution::seq`
- Several new algorithms in C++17 including
  - `std::for_each_n(POLICY, first, size, func)`
- Insert `std::execution::par` as first parameter when calling algorithms
- NVC++ 20.5: automatic GPU acceleration of C++17 parallel algorithms
  - Leverages CUDA Unified Memory

# PARALLEL 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)
{
#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 ;
}
```

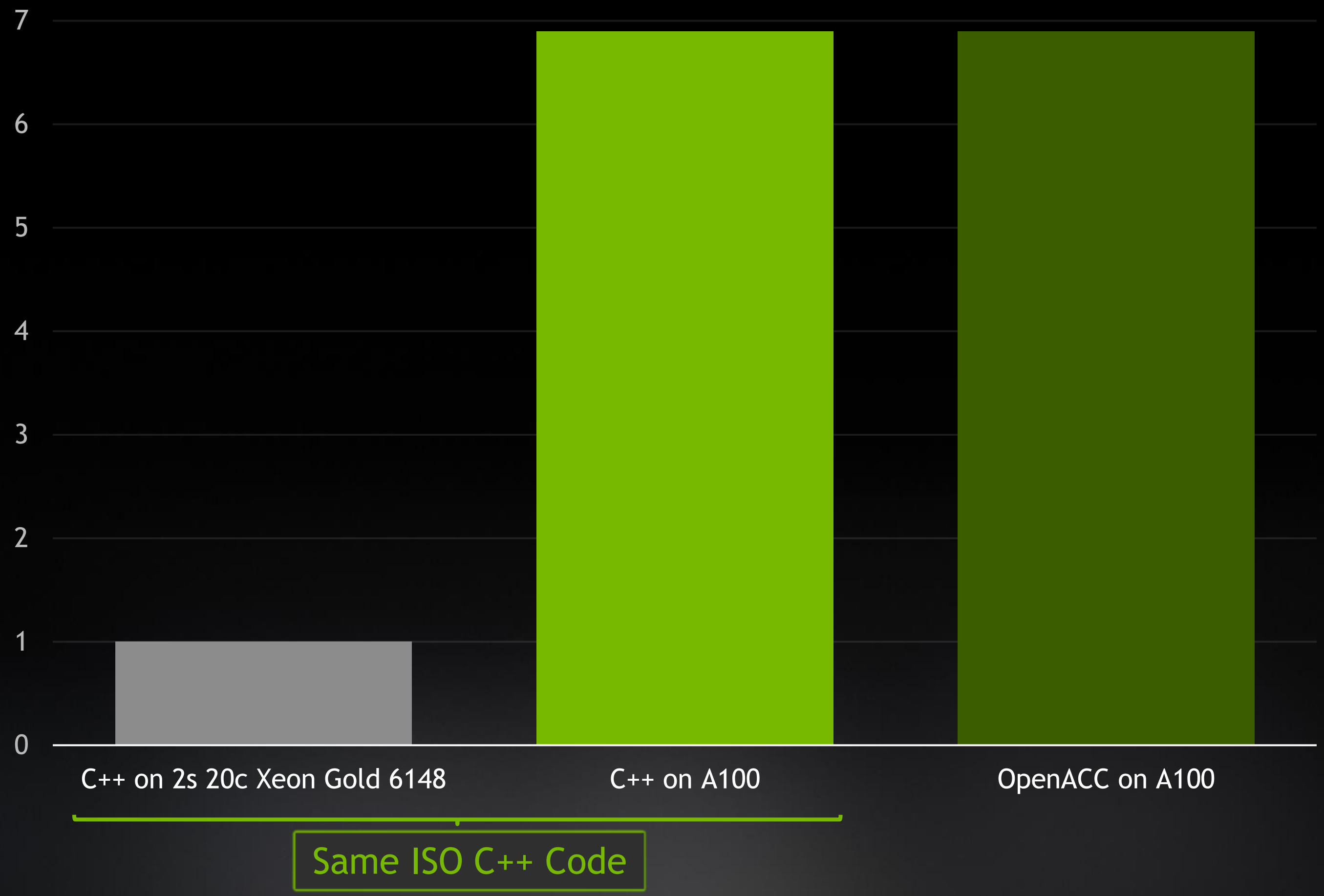
C++ with OpenMP

```
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));
      }
  });
}
```

Parallel C++17

# LULESH PERFORMANCE

Speedup – Higher is Better



# 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, etc

### DO CONCURRENT

- NVFORTRAN 20.x
- Auto-offload & multi-core

### Co-Arrays

- Coming Soon
- Accelerated co-array images

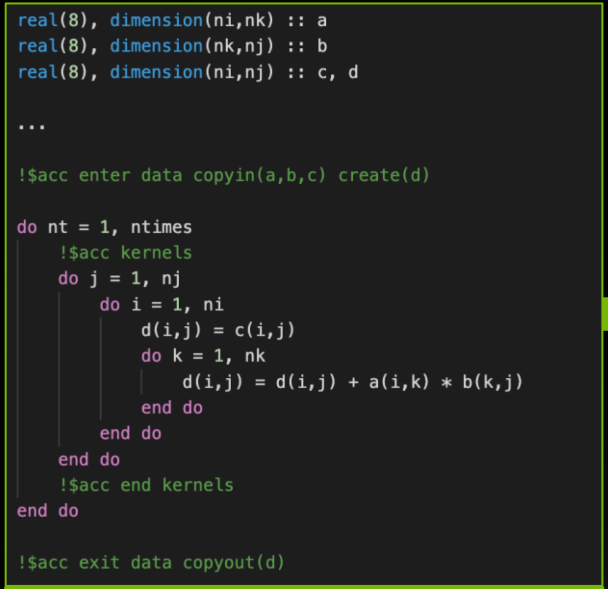
## Fortran 202x

### DO CONCURRENT Reductions

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

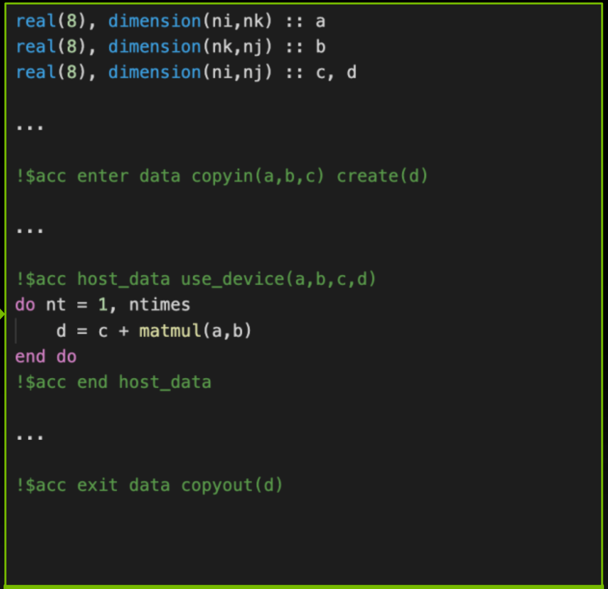
# HPC PROGRAMMING IN ISO FORTRAN

## NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend



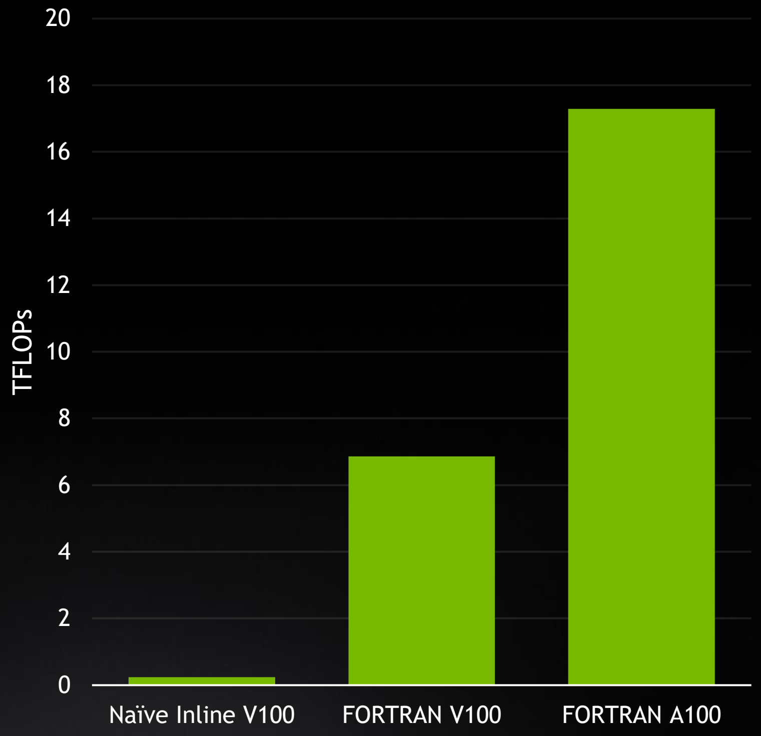
```
real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c, d

...

!$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, d

...

!$acc enter data copyin(a,b,c) create(d)

...

!$acc host_data use_device(a,b,c,d)
do nt = 1, ntimes
    d = c + matmul(a,b)
end do
!$acc end host_data

...

!$acc exit data copyout(d)
```

MATMUL FP64 matrix multiply



# HPC LIBRARIES

# A100 FEATURES IN MATH LIBRARIES

Automatic Acceleration of Critical Routines in HPC and AI

**cuBLAS**

BF16, TF32 and FP64 Tensor Cores

**cuSPARSE**

Increased memory BW, Shared Memory and L2

**cuTENSOR**

BF16, TF32 and FP64 Tensor Cores

**cuSOLVER**

BF16, TF32 and FP64 Tensor Cores

**cuFFT**

Increased memory BW, Shared Memory and L2

**CUDA Math API**

BF16 Support

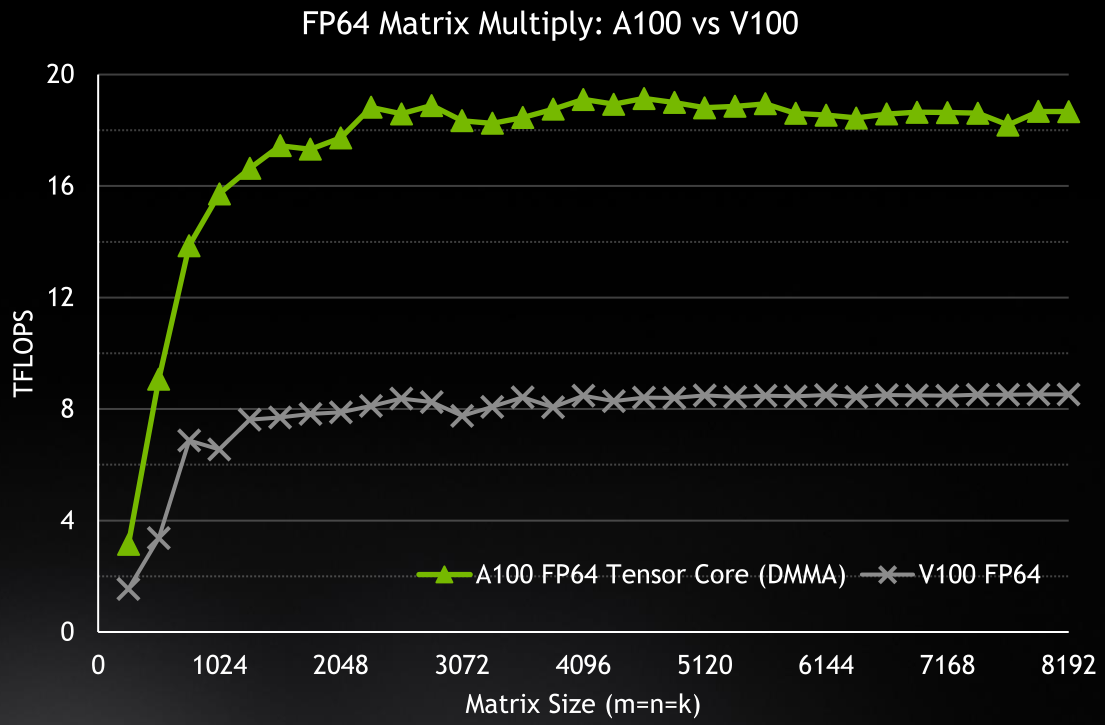
# A100 TENSOR CORES IN LIBRARIES

## cuBLAS

- Automatic Tensor Core acceleration
- Removed matrix size restrictions for Tensor Core acceleration

**DGEMM on A100**

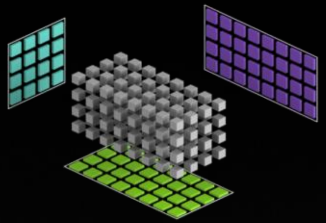
- Up to 19 TFLOPs, 2.4x V100


FP64 Matrix Multiply: A100 vs V100

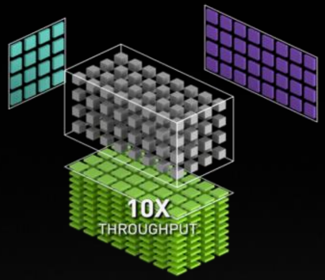
# A100 TENSOR CORES IN LIBRARIES

## cuBLAS

NVIDIA V100 FP32



NVIDIA A100 Tensor Core TF32



TF32 MMA Dimensions: m,n,k = 16x8x8

10X

NVIDIA V100 FP64

NVIDIA A100 Tensor Core FP64

2.5X
THROUGHPUT

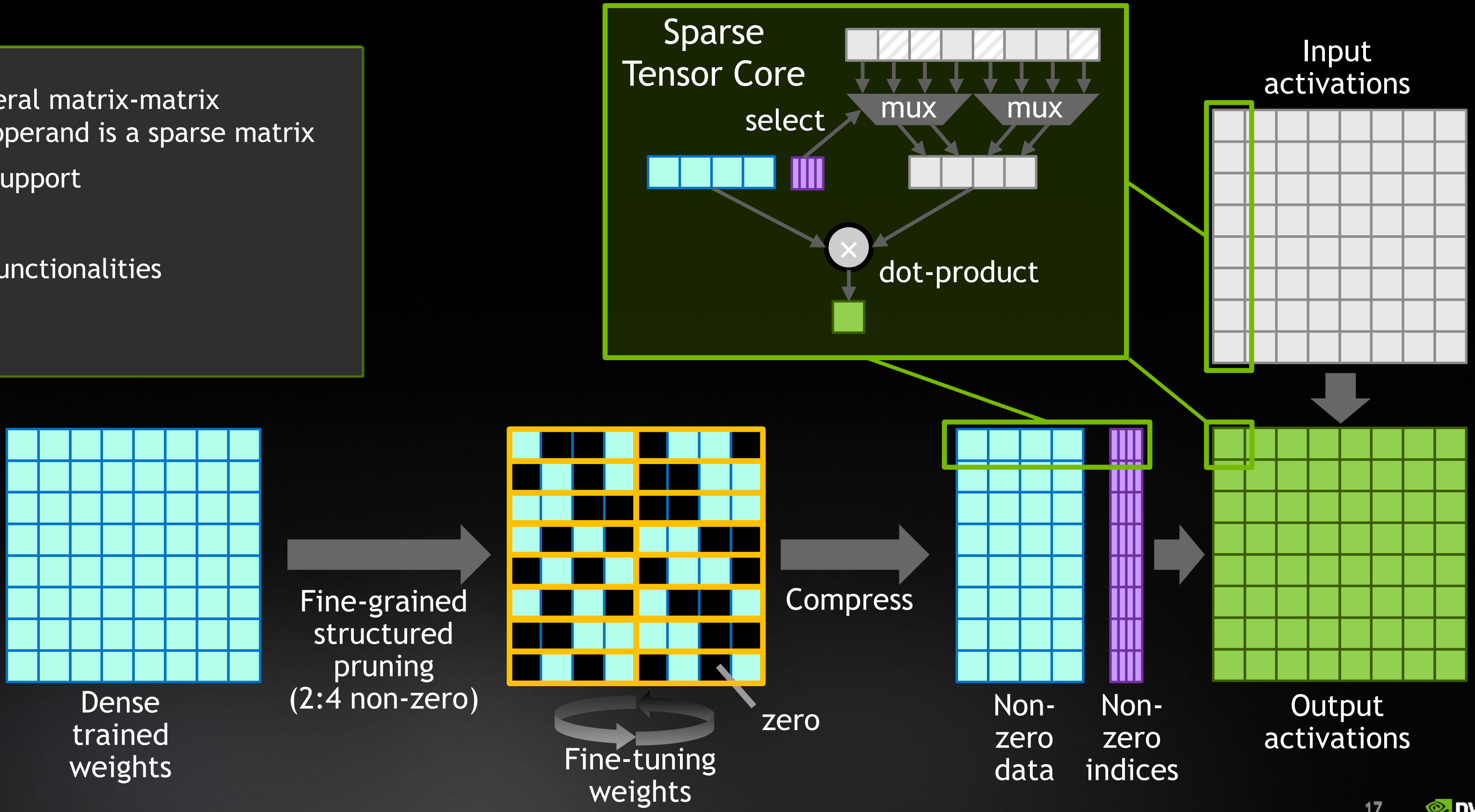
DMMA Dimensions: m,n,k = 8x8x4

2.5X

# cuSPARSELt

## Extension Library with Sparse Matmul

- High-performance library for general matrix-matrix operations in which at least one operand is a sparse matrix
- Ampere Sparse MMA tensor core support
- Mixed-precision support
- Matrix pruning and compression functionalities
- Auto-tuning functionality



# A100 TENSOR CORE

| | INPUT OPERANDS | ACCUMULATOR | TOPS | X-factor vs. FFMA | SPARSE TOPS | SPARSE X-factor vs. FFMA |
|---|---|---|---|---|---|---|
| *V100* | FP32 | FP32 | 15.7 | 1x | - | - |
| | FP16 | FP32 | 125 | 8x | - | - |
| *A100* | FP32 | FP32 | 19.5 | 1x | - | - |
| | TF32 | FP32 | 156 | 8x | 312 | 16x |
| | FP16 | FP32 | 312 | 16x | 624 | 32x |
| | BF16 | FP32 | 312 | 16x | 624 | 32x |
| | FP16 | FP16 | 312 | 16x | 624 | 32x |
| | INT8 | INT32 | 624 | 32x | 1248 | 64x |
| | INT4 | INT32 | 1248 | 64x | 2496 | 128x |
| | BINARY | INT32 | 4992 | 256x | - | - |
| | IEEE FP64 | | 19.5 | 1x | - | - |

# MATH LIBRARY DEVICE EXTENSIONS

## Introducing cuFFTDx: Device Extension

**Available in Math Library EA Program**

Device callable library

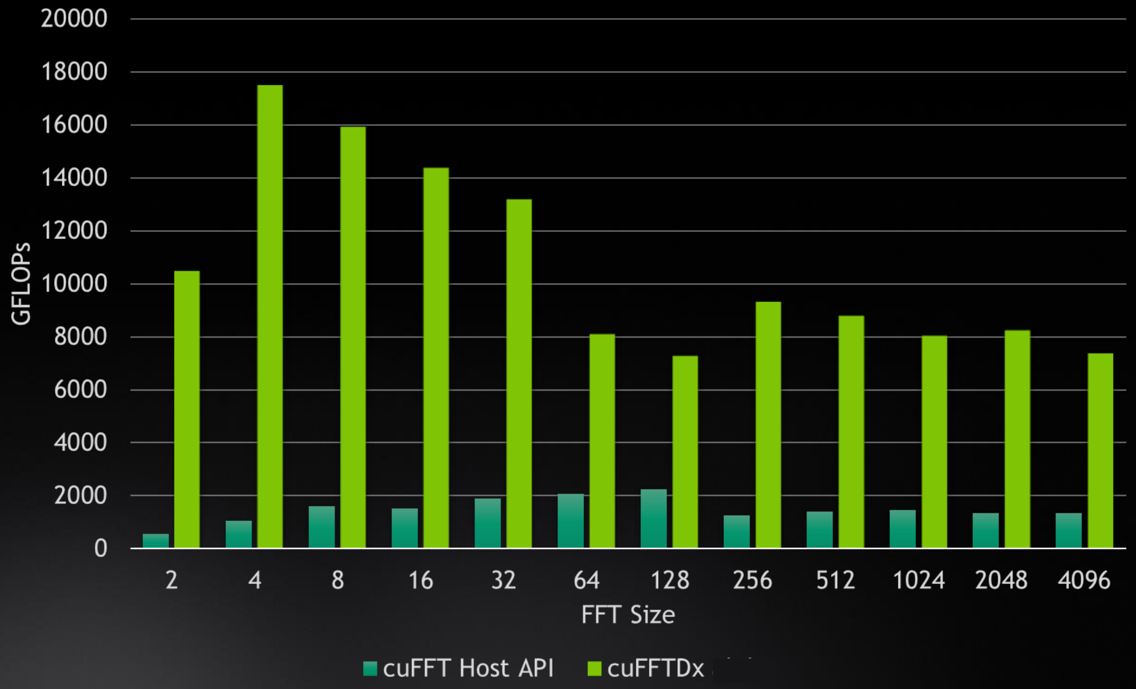
Retain and reuse on-chip data

Inline FFTs in user kernels

Combine multiple FFT operations

### cuFFTDx Device API V100 Performance

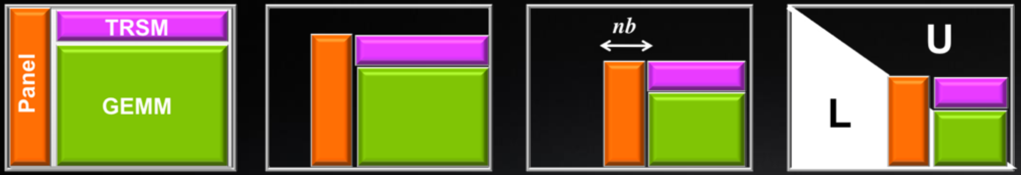
Small-size FFTs

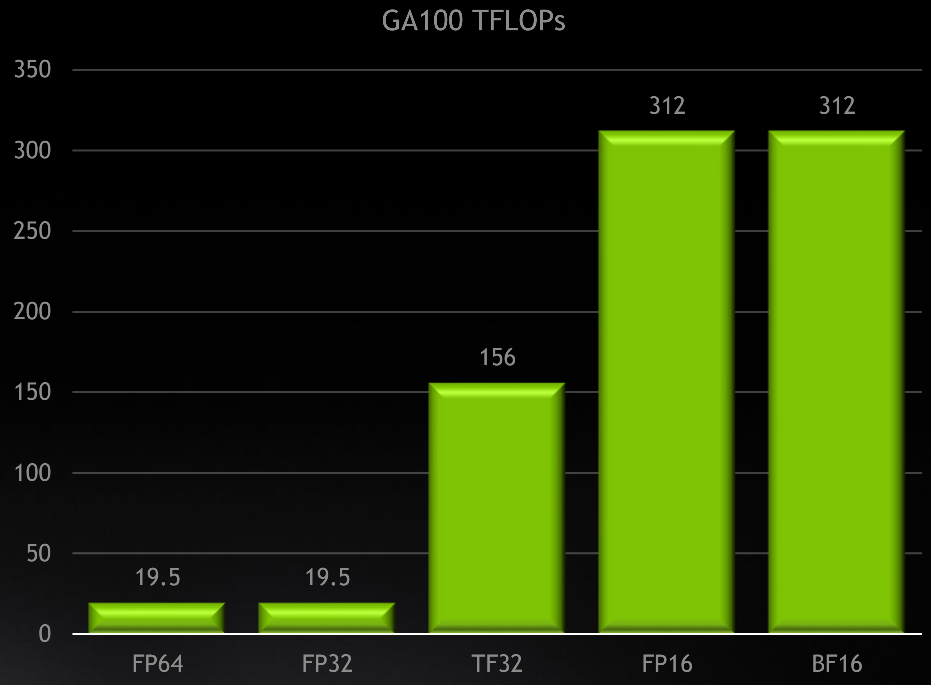


# TENSOR CORE ACCELERATED LINEAR SOLVERS

## Mixed Precision Dense Linear Solvers

- Common HPC Solvers dominated by matrix multiplication
  - LU, QR
- Can we accelerate with FP16 Tensor Core and retain FP64 accuracy? Yes!

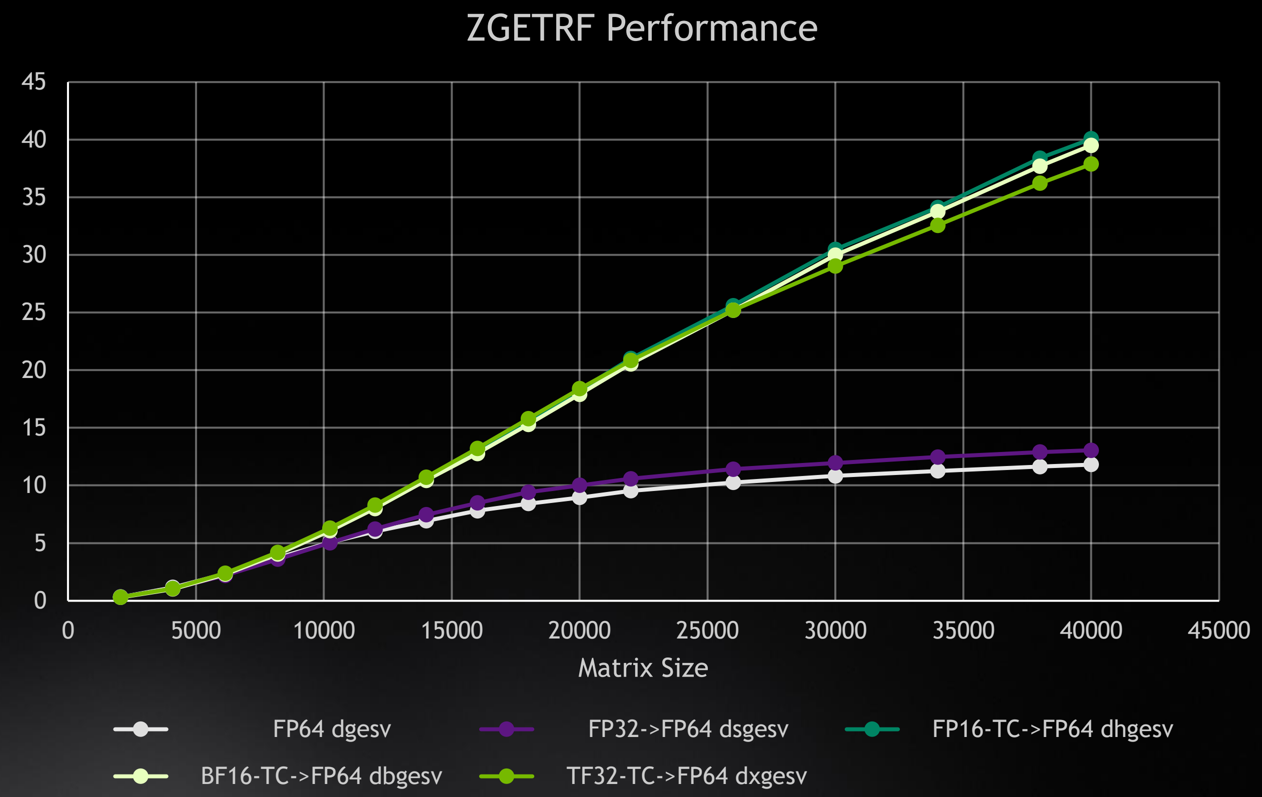




# TENSOR CORE ACCELERATED LINEAR SOLVERS

## Mixed Precision Dense Linear Solvers

- LU & QR Solvers available in cuSOLVER
- FP64 input, FP64 output, black-box mixed precision acceleration
- Real and Complex, single and multi-RHS



# MULTI GPU SUPPORT IN LIBRARIES

## Linear Algebra and FFT

### cuFFT

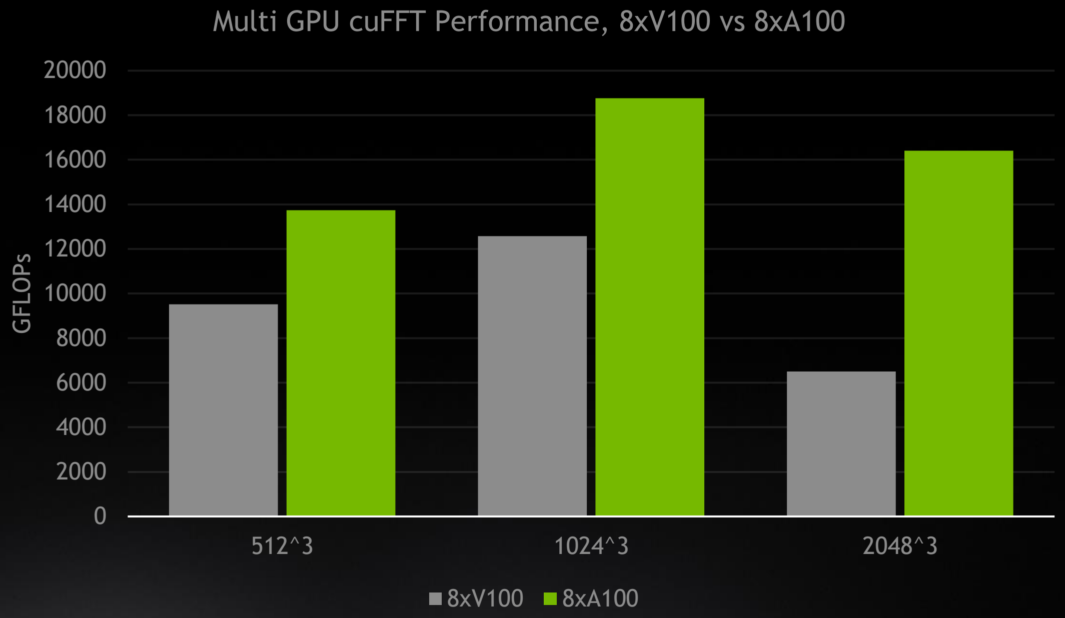
- Single Process Multi-GPU FFT
- Multi Node Multi-GPU FFT Coming Soon

### cuSOLVER

- Single Process Multi-GPU Eigensolver
- Single Process Multi-GPU LU
- Single Process Multi-GPU Cholesky
- Multi Node Multi-GPU LU Coming Soon

### cuBLAS

- Improved Single Process Multi-GPU GEMM



# MULTI GPU WITH THE NVIDIA HPC SDK

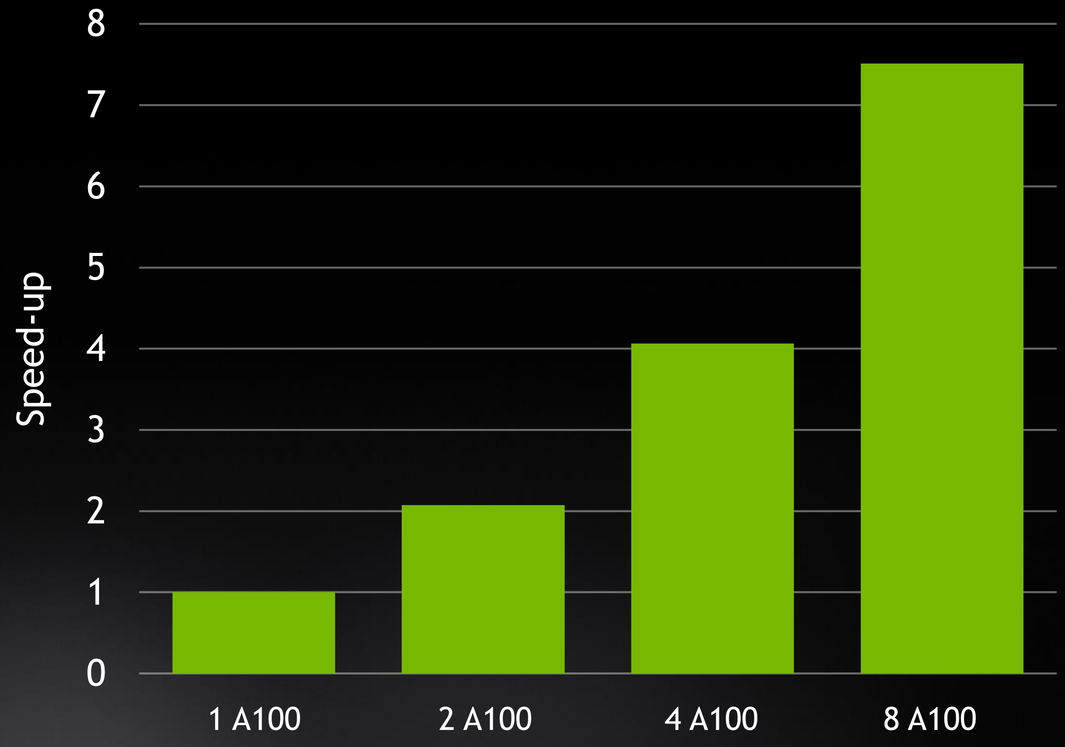
## Cloverleaf Hydrodynamics Mini-App

### Full Integration provided by HPC SDK

- Fortran + OpenACC + Open MPI

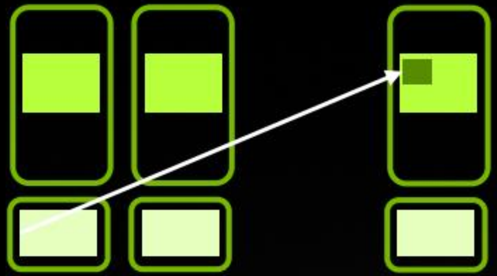
### Strong Scaling - Cloverleaf BM128

- Perfect scaling to 4 A100 GPUs
- 7.5X speed-up on 8 A100 GPUs

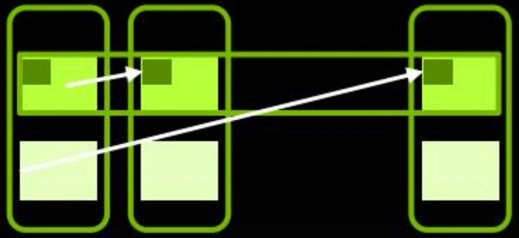


# COMMUNICATION LIBRARIES

Single GPU, Multi GPU, and Multi Node



Open MPI + UCX



NVSHMEM

NCCL

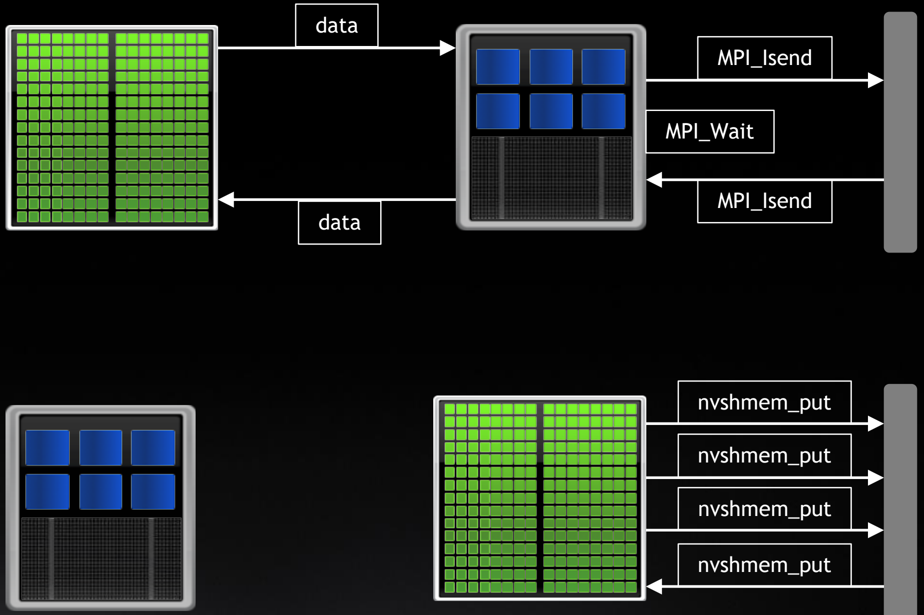
# INTRODUCING NVSHMEM

## GPU Optimized OpenSHMEM

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

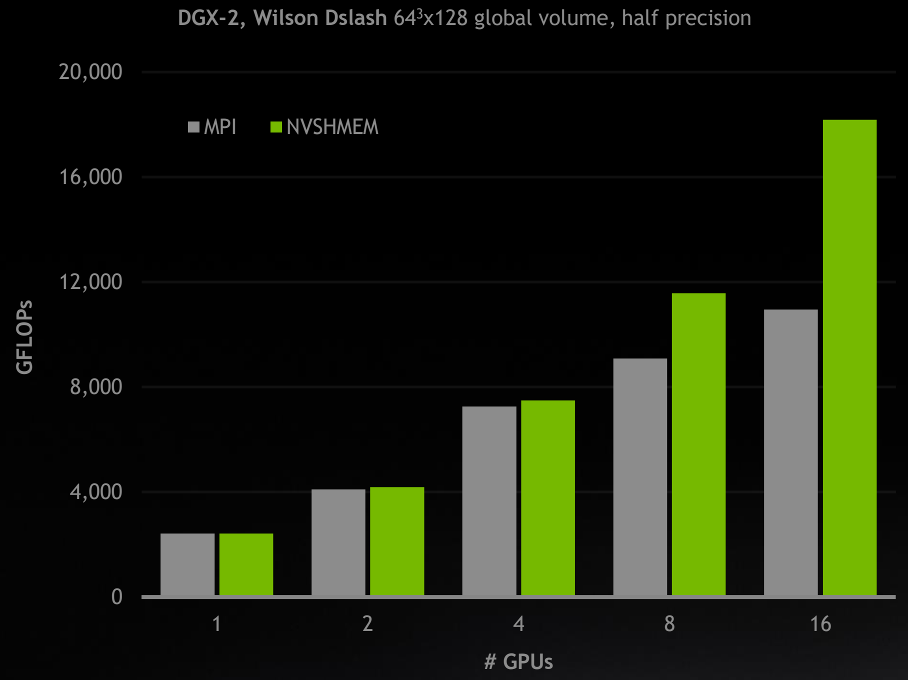
Pre-release Impact

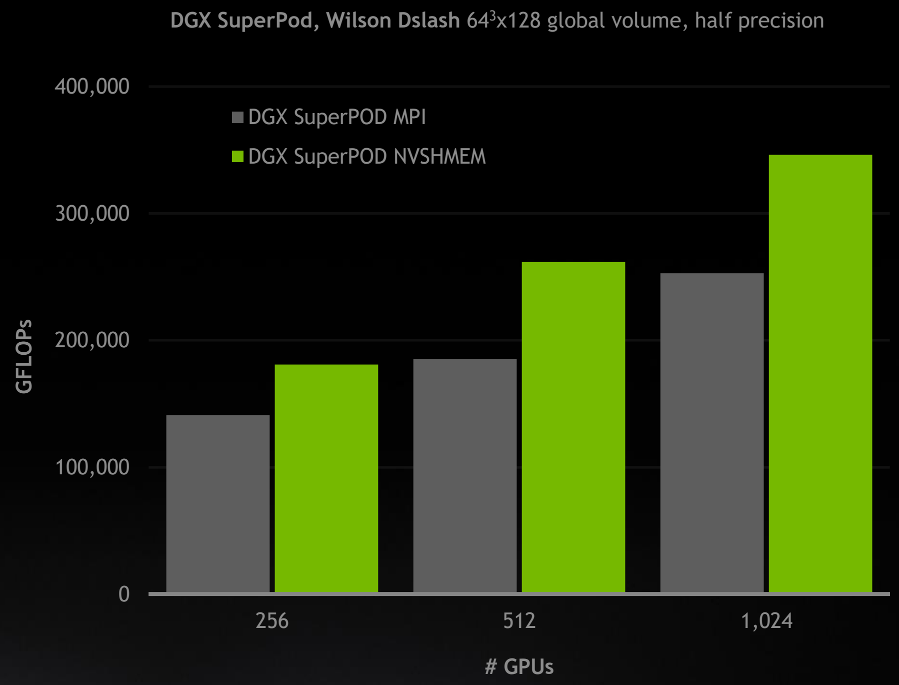
- LBANN, Kokkos/CGSolve, QUDA



# INTRODUCING NVSHMEM

## Impact in HPC Applications





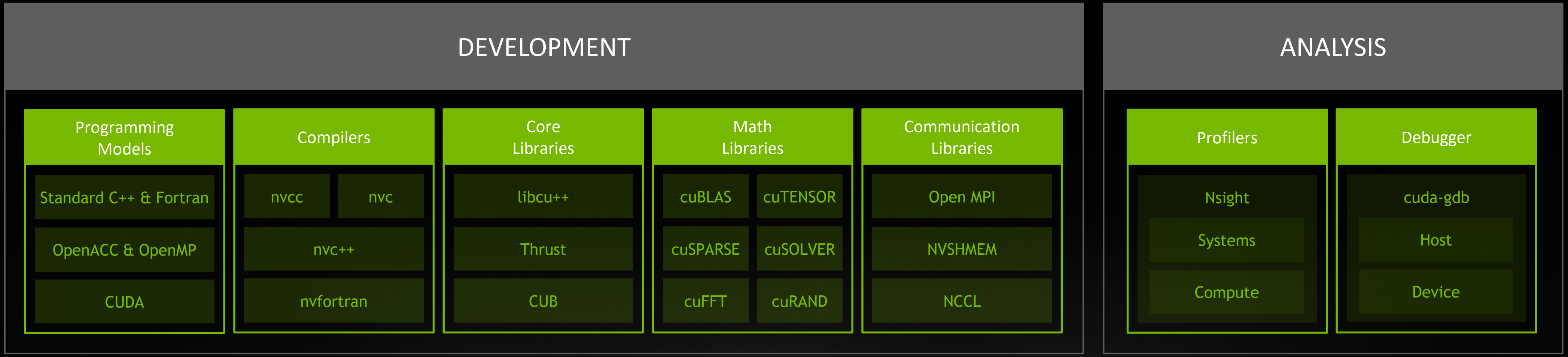
**QUDA: Quantum Chromodynamics on CUDA**

- Up to 1.7X Single Node Speedup
- Up to 1.4X Multi Node Speedup

# AVAILABLE NOW: THE NVIDIA HPC SDK

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

## NVIDIA HPC SDK



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
  - Open MPI
  - NVSHMEM
  - NCCL

ANALYSIS

- Profilers
  - Nsight
    - Systems
    - Compute
- Debugger
  - cuda-gdb
    - Host
    - Device

Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect

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

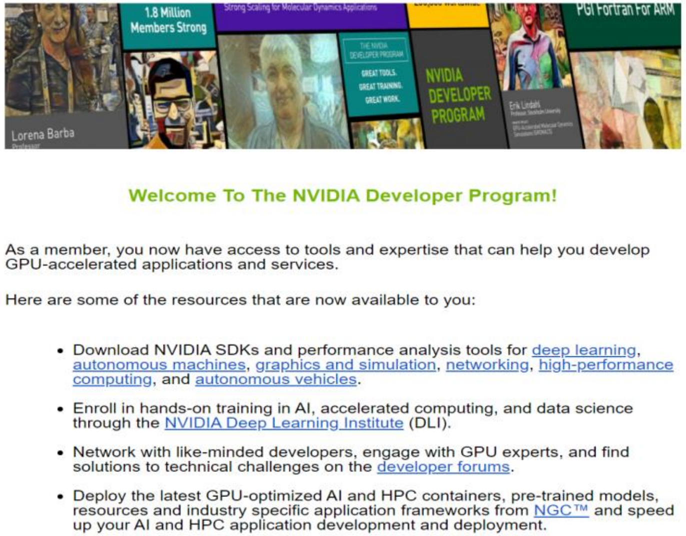
7-8 Releases Per Year | Freely Available

# 加入 NVIDIA 開發者計畫

developer.nvidia.com/developer-program

請掃描左邊 QR code 加入 NVIDIA 開發者計畫，活動結束前完成註冊，即可換取精美小禮物。

請向 NVIDIA 工作人員出示主旨為「Your application for the program NVIDIA Developer Program is approv



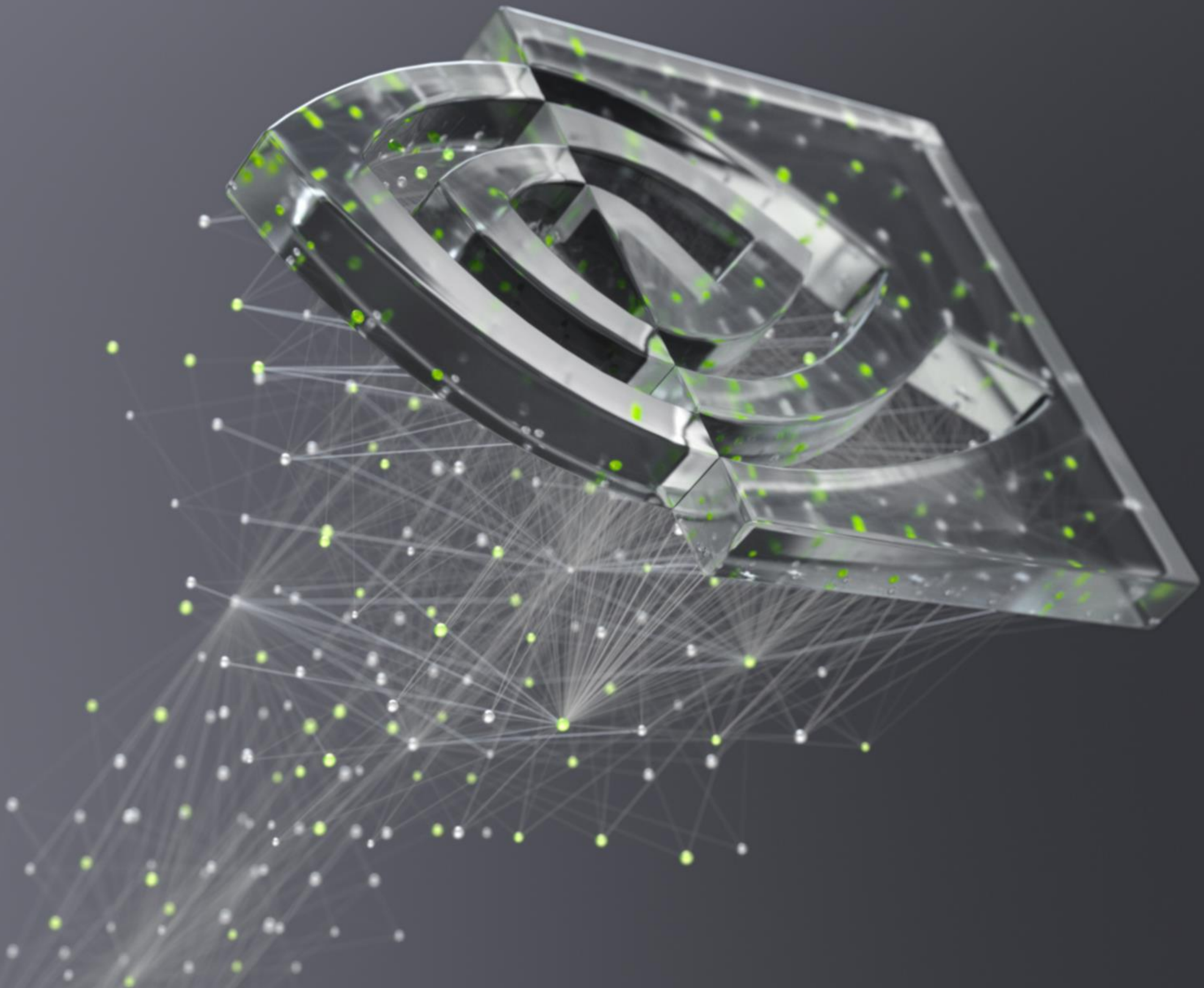
1.8 Million Members Strong
Strong Scaling for Molecular Dynamics Applications
THE NVIDIA DEVELOPER PROGRAM
GREAT TOOLS. GREAT TRAINING. GREAT WORK.
NVIDIA DEVELOPER PROGRAM
PGI Fortran For ARM
Lorena Barba
Professor
Erik Lindahl

Welcome To The NVIDIA Developer Program!

As a member, you now have access to tools and expertise that can help you develop GPU-accelerated applications and services.

Here are some of the resources that are now available to you:

- Download NVIDIA SDKs and performance analysis tools for deep learning, autonomous machines, graphics and simulation, networking, high-performance computing, and autonomous vehicles.
- Enroll in hands-on training in AI, accelerated computing, and data science through the NVIDIA Deep Learning Institute (DLI).
- Network with like-minded developers, engage with GPU experts, and find solutions to technical challenges on the developer forums.
- Deploy the latest GPU-optimized AI and HPC containers, pre-trained models, resources and industry specific application frameworks from NGC™ and speed up your AI and HPC application development and deployment.



NVIDIA