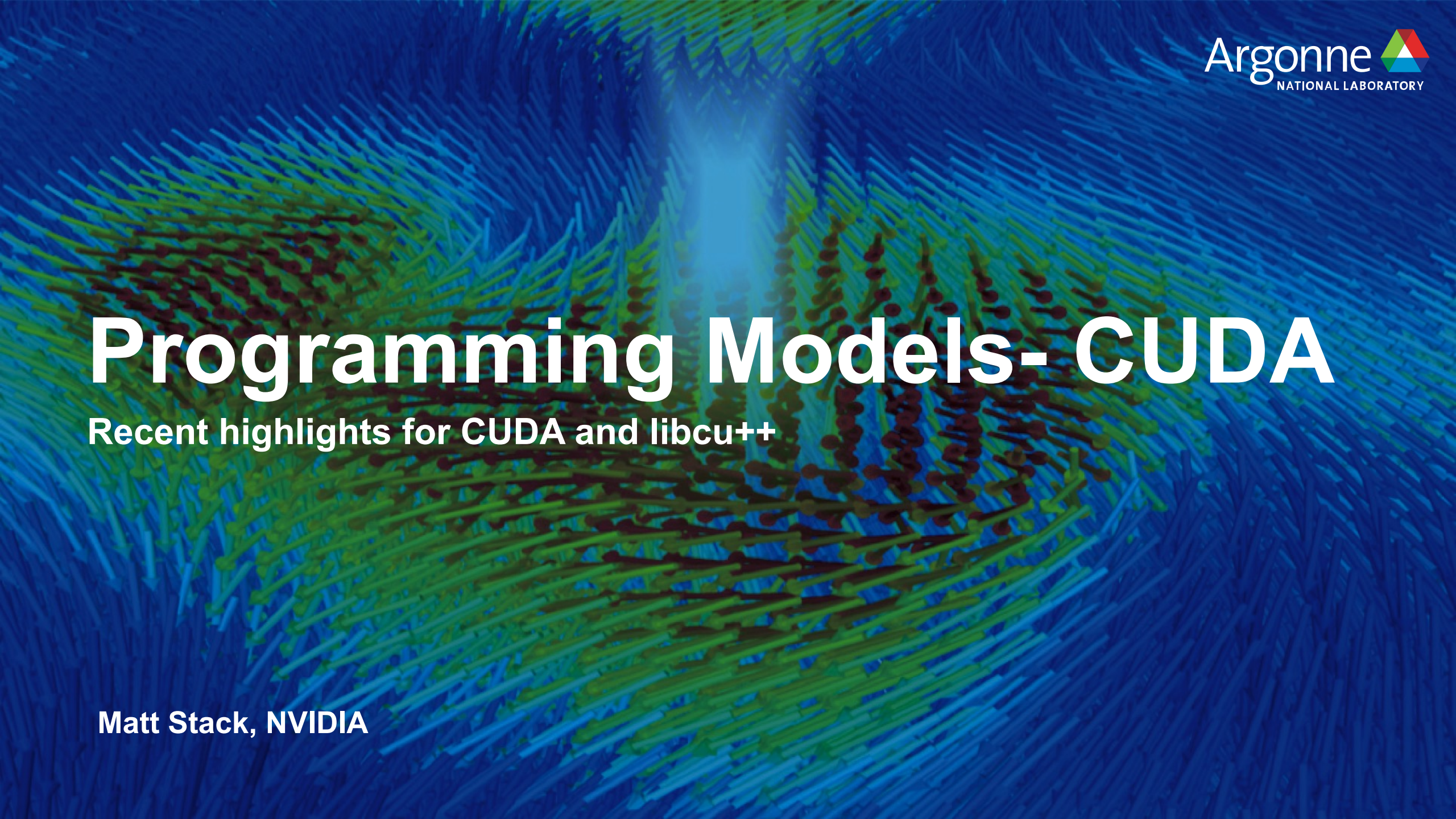

October 10-12, 2023
ALCF Hands-on HPC Workshop



Programming Models- CUDA

Recent highlights for CUDA and libcu++

Matt Stack, NVIDIA

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

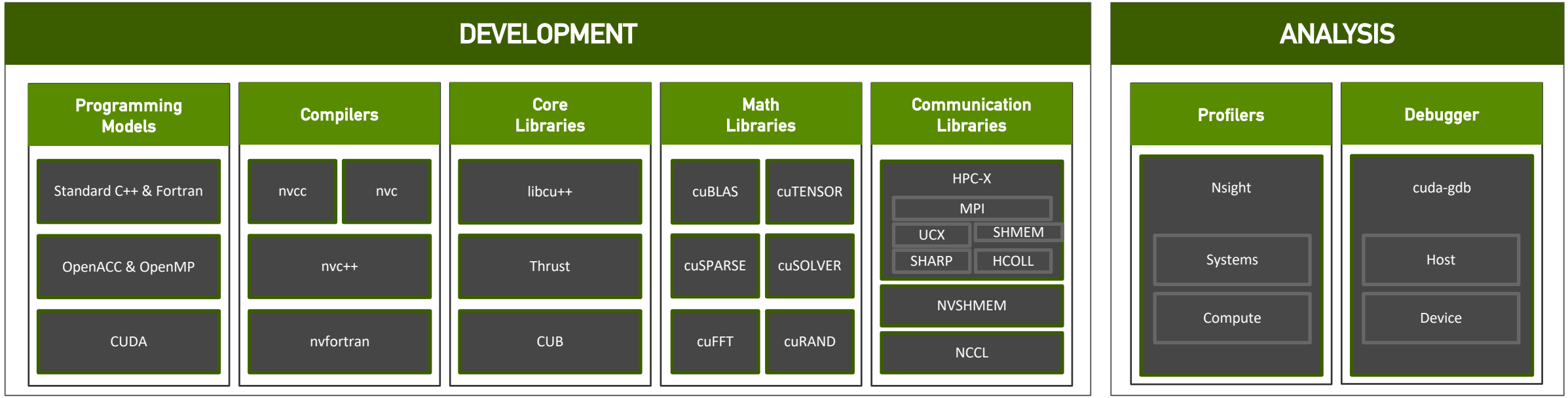
```
__global__
void saxpy(int n, float a,
            float *x, float *y) {
  int i = blockIdx.x*blockDim.x +
          threadIdx.x;
  if (i < n) y[i] += a*x[i];
}

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

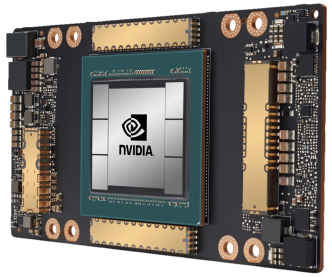
Available at developer.nvidia.com/hpc-sdk, on NGC, via Spack, and in the Cloud



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

Fortran

NVIDIA CUDA

OpenACC

More Science, Less Programming

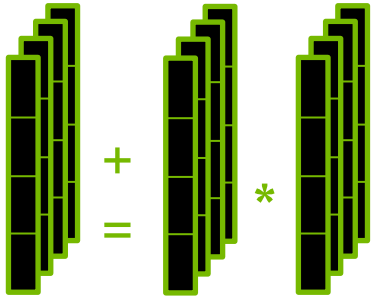
OpenMP

Programmable

Standard Languages

Directives

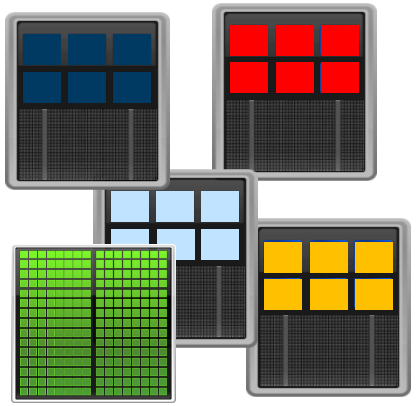
CUDA



CPU Optimized

Directives

Vectorization



Multi-Platform

x86_64

Arm

OpenPOWER

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

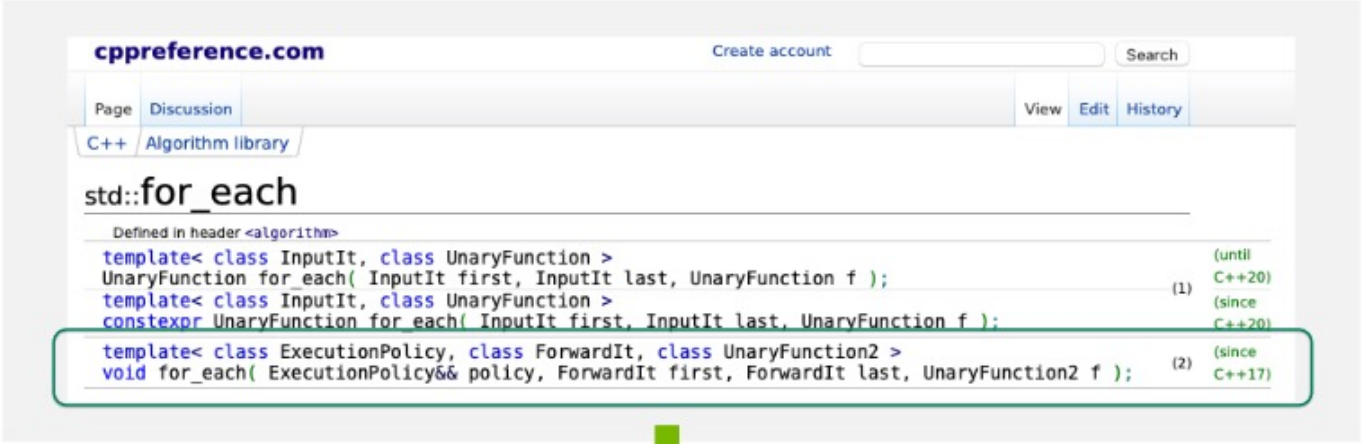
Linear Algebra

- 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`

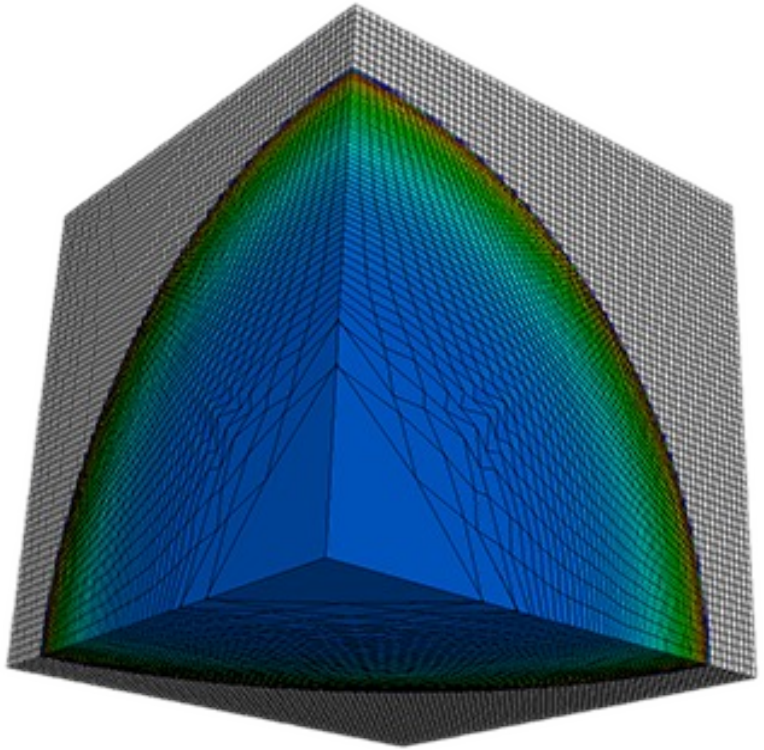
HPC PROGRAMMING IN ISO C++




```
#include <algorithm> // std::for_each and other functions
#include <execution> // seq, par, par_unseq, un_seq
. . .
std::vector<double> vec = . . .
std::for_each(std::execution::par, vec.begin(), vec.end(), [=](auto i){
        . . . // doing work for each element in the vector
});
```

C++17 PARALLEL ALGORITHMS

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



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++

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

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

ACCELERATED STANDARD LANGUAGES

ISO C++

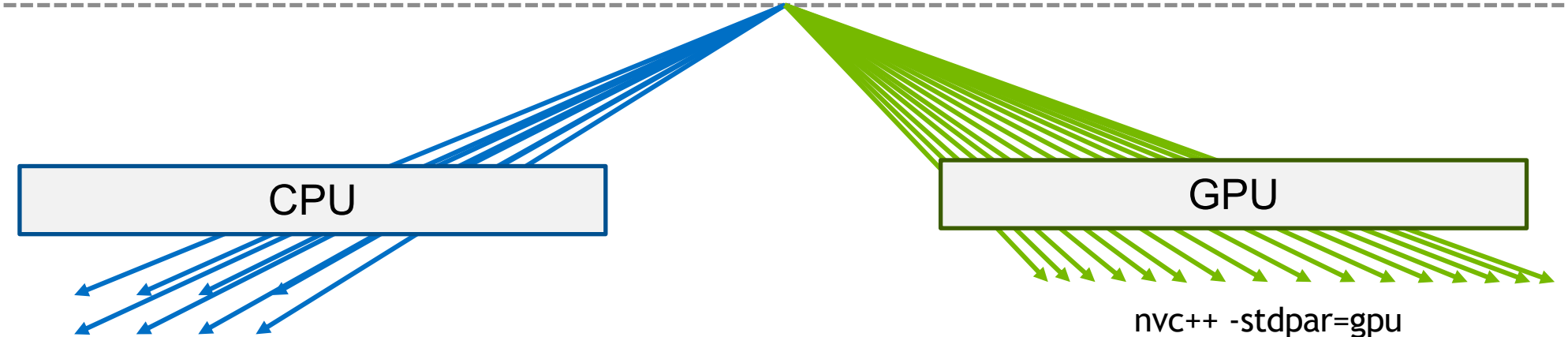
```
std::transform(par, x, x+n, y,
   y,[=](float x, float y){
      return y + a*x;
    }
);
```

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
```

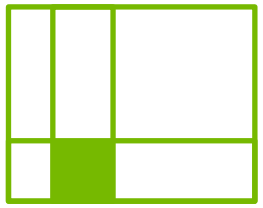


nvc++ -stdpar=multicore
nvfortran -stdpar=multicore
legate -cpus 16 saxpy.py

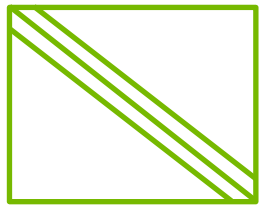
nvc++ -stdpar=gpu
nvfortran -stdpar=gpu
legate -gpus 1
saxpy.py

NVIDIA MATH LIBRARIES

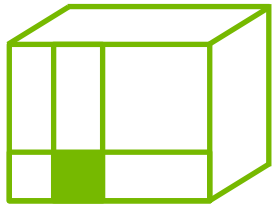
Linear Algebra, FFT, RNG and Basic Math



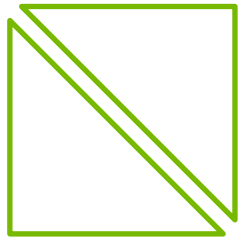
cuBLAS



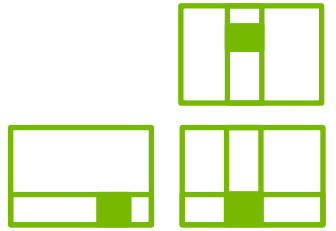
cuSPARSE



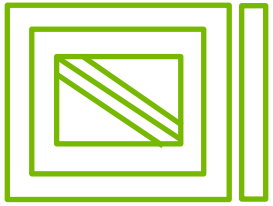
cuTENSOR



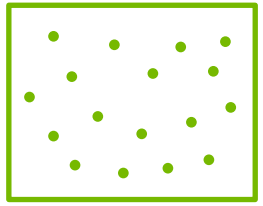
cuSOLVER



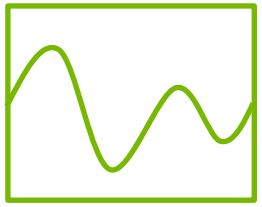
CUTLASS



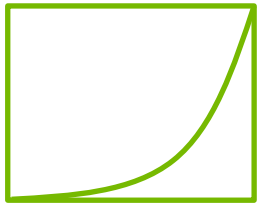
AMGX



cuRAND



cuFFT

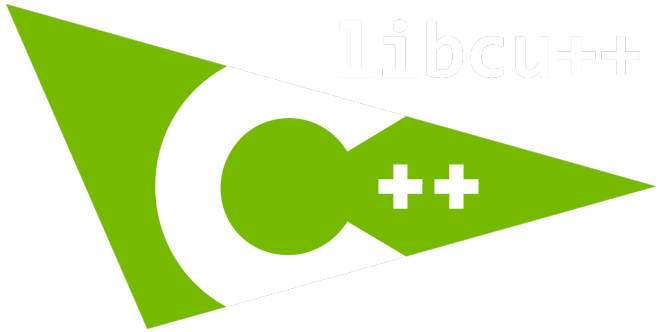


CUDA Math API

TENSOR CORE SUPPORT IN MATH LIBRARIES

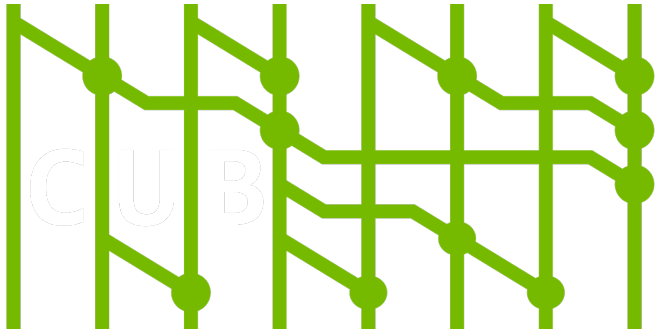
Library and Tensor Core Functionality	INT4		INT8		FP16		BF16		TF32		FP64
	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense	Sparse	Dense
cuBLAS & cuBLASLt Dense GEMM			✅		✅		✅		✅		✅
cuTENSOR Tensor Contractions					✅		✅		✅		✅
cuSOLVER Linear System Solvers					✅		✅		✅		✅
cuSPARSE Block-SpMM	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

C++ Core Compute Libraries (CCCL)



https://github.com/NVIDIA/libcudacxx

https://github.com/NVIDIA/thrust



https://github.com/NVIDIA/cub

https://github.com/nvidia/cccl

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.

```cpp
#include <cstdio>
#include <thrust/device_vector.h>
#include <cub/block/block_reduce.cuh>
#include <cuda/atomic>

constexpr int block_size = 256;

__global__ void sumKernel(int const* data, int* result, std::size_t N)
{
   using BlockReduce = cub::BlockReduce<int, block_size> ;

   __shared__ typename BlockReduce::TempStorage temp_storage;

   int index = threadIdx.x + blockIdx.x * blockDim.x;

   int sum = 0;
   if (index < N) {
      sum += data[index];
   }

   sum = BlockReduce(temp_storage).Sum(sum);

   if (threadIdx.x == 0){
      cuda::atomic_ref<int, cuda::thread_scope_device> atomic_result(*result);
      atomic_result.fetch_add(sum, cuda::memory_order_relaxed);
   }
}

int main()
{
   std::size_t N = 1000;
   thrust::device_vector<int> data(N, 1);
   thrust::device_vector<int> result(1);

   int num_blocks = (N + block_size - 1) / block_size;

   sumKernel<<<num_blocks, block_size>>>(thrust::raw_pointer_cast(data.data()),
                                         thrust::raw_pointer_cast(result.data()), N);

   auto err = cudaDeviceSynchronize();
   if(err != cudaSuccess){
      std::cout << "Error: " << cudaGetErrorString(err) << std::endl;
      return -1;
   }
return 0;
}
```

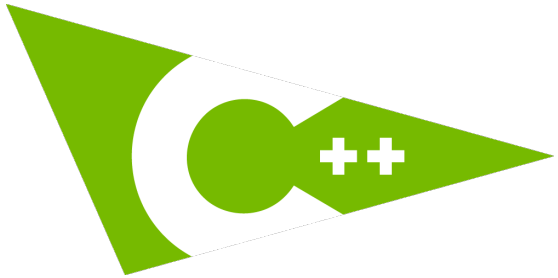
Thrust include,
Cub include,
libcu++ (non-strictly conforming)
include

Cub block reduce

Libcu++ atomic add

Thrust used for memory

https://github.com/NVIDIA/cccl/blob/main/examples/example_project/example.cu



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

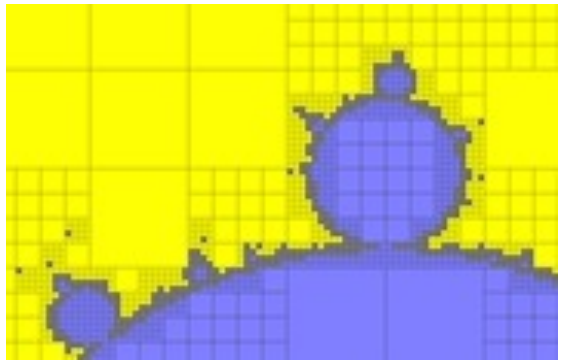
`atomic_ref<T>` (SM60+)
Memory Resources & Allocators
`cuda::stream_view`

Future

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

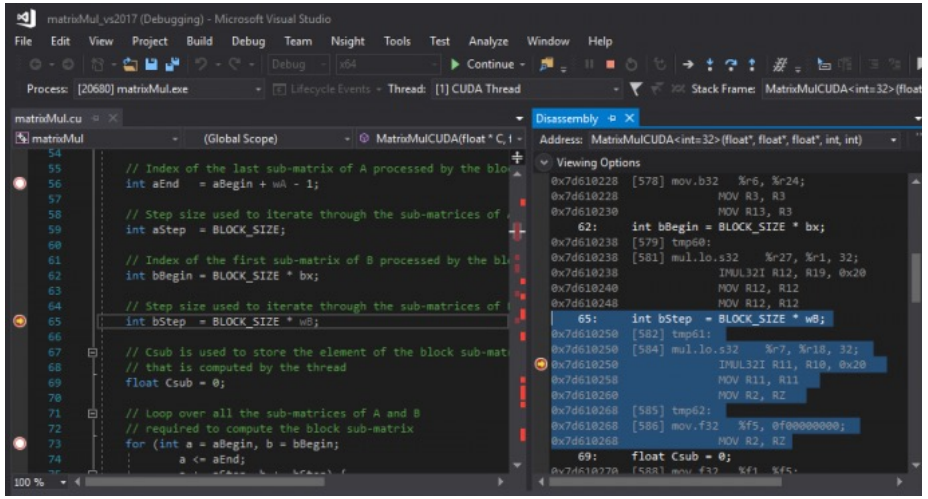
CUDA 12.0 HIGHLIGHTS

Dynamic Programming and Lazy Loading



- Dynamic Programming
  - 3 new ways to launch a child kernel
    - Per-thread stream, Fire-and-forget, Tail launch
  - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cdp2-vs-cdp1
- Lazy Loading (11.7, 11.8+)
  - Technique for loading functions into host or device memory until they are called
    - Saves memory, and loading time
  - set env CUDA_MODULE_LOADING to LAZY (It is enabled by default in 12.2+)
- C++ 20 Support for conforming host compilers (nvc++ 22.x)

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