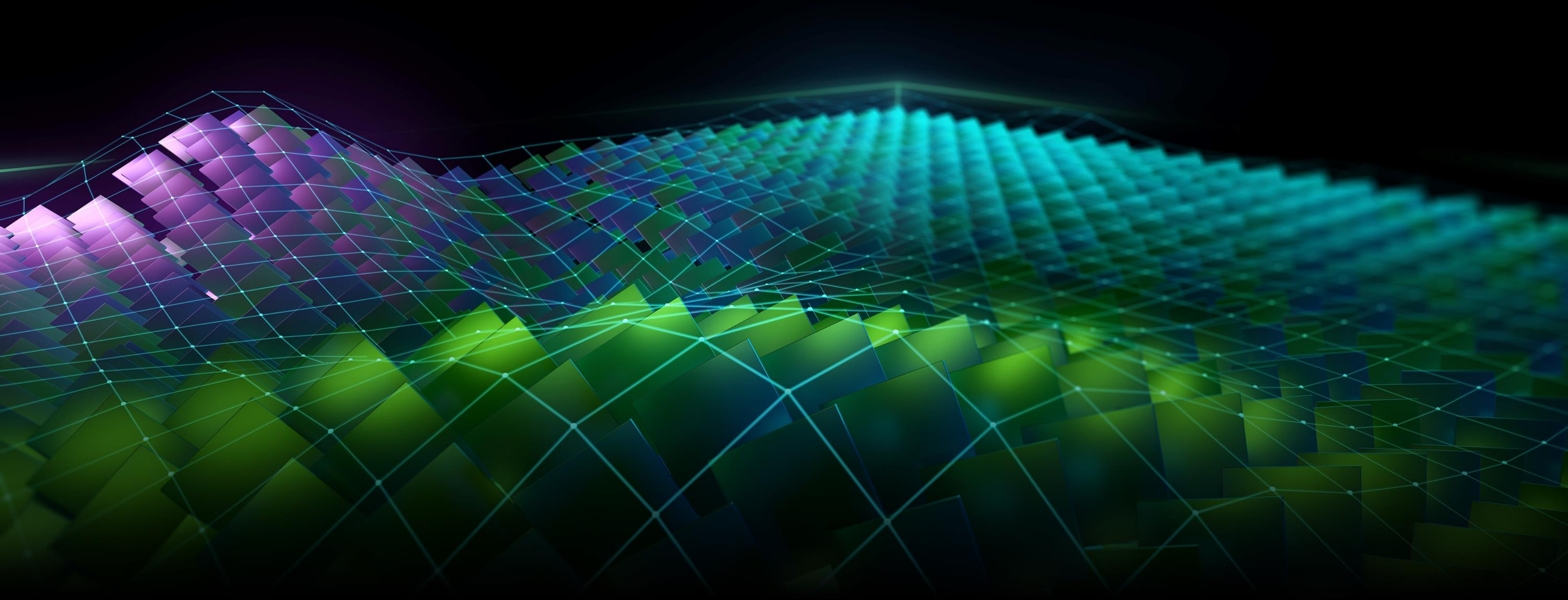
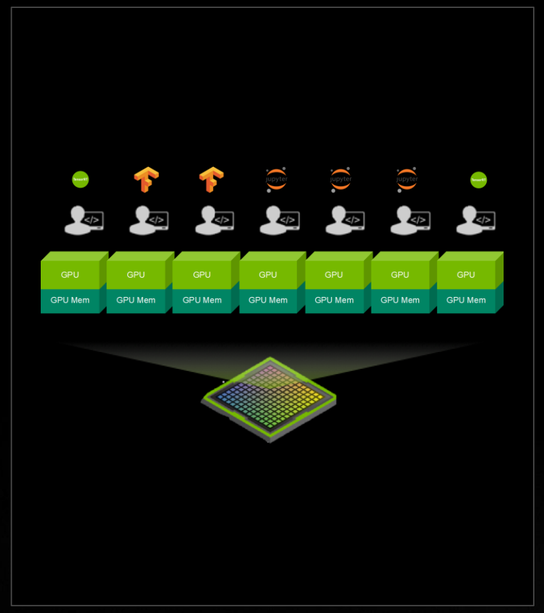


CUDA 11 UPDATE

Jeff Larkin <jlarkin@nvidia.com>

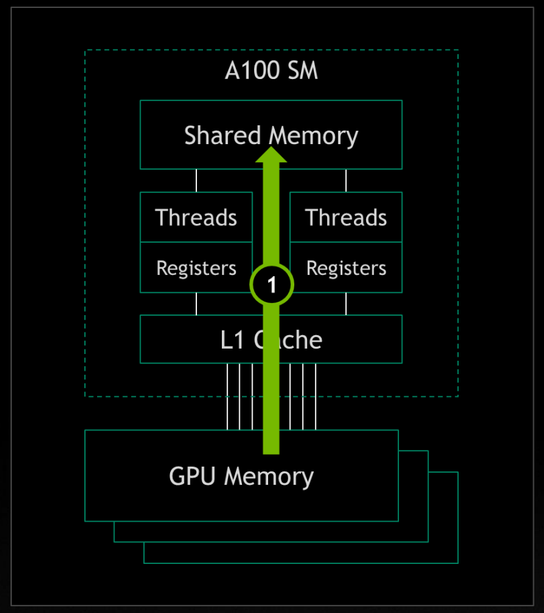
OLCF October Users Call

CUDA KEY INITIATIVES



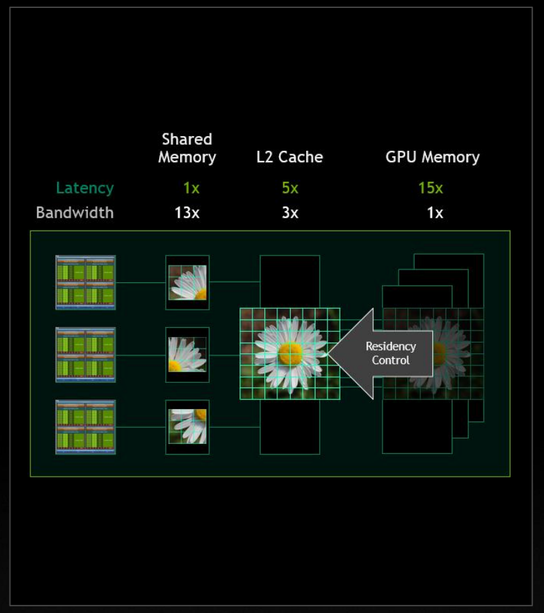
Hierarchy

Programming and running systems at every scale



Asynchrony

Creating concurrency at every level of the hierarchy



Latency

Overcoming Amdahl with lower overheads for memory & processing

```
// ISO C++, __host__ only
#include <atomic>
std::atomic<int> x;

// CUDA C++, __host__ __device__
// Strictly conforming to the ISO C++
#include <cuda/std/atomic>
cuda::std::atomic<int> x;

// CUDA C++, __host__ __device__
// Conforming extensions to ISO C++
#include <cuda/atomic>
cuda::atomic<int, cuda::thread_scope_block> x;
```

Language

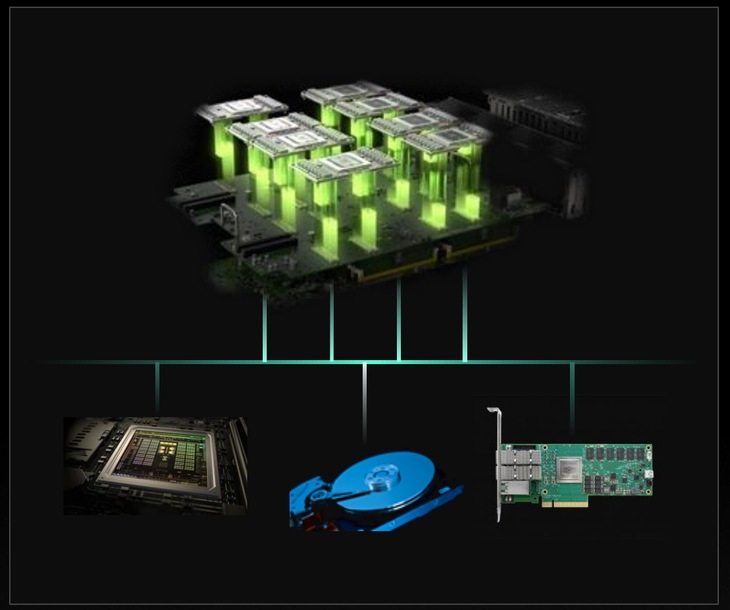
Supporting and evolving Standard Languages

CUDA PLATFORM: TARGETS EACH LEVEL OF THE HIERARCHY

The CUDA Platform Advances State Of The Art From Data Center To The GPU

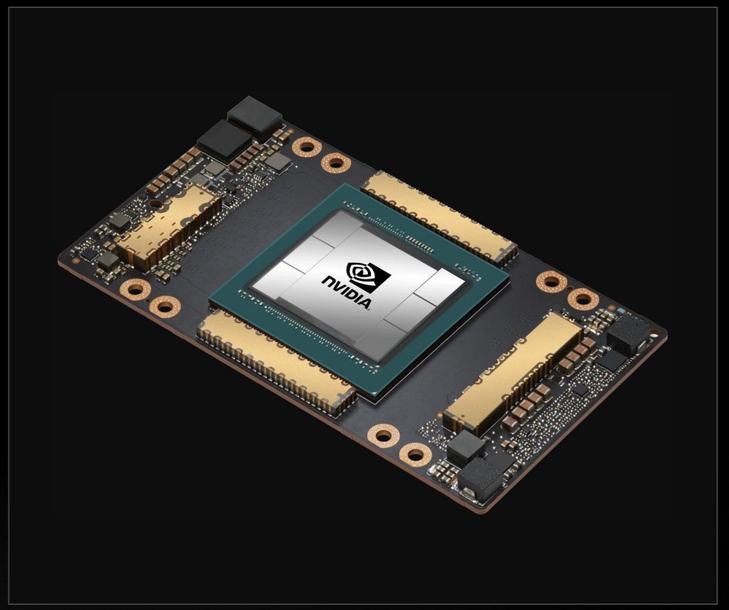
System Scope

FABRIC MANAGEMENT
DATA CENTER OPERATIONS
DEPLOYMENT
MONITORING
COMPATIBILITY
SECURITY



Node Scope

GPU-DIRECT
NVLINK
LIBRARIES
UNIFIED MEMORY
ARM
MIG

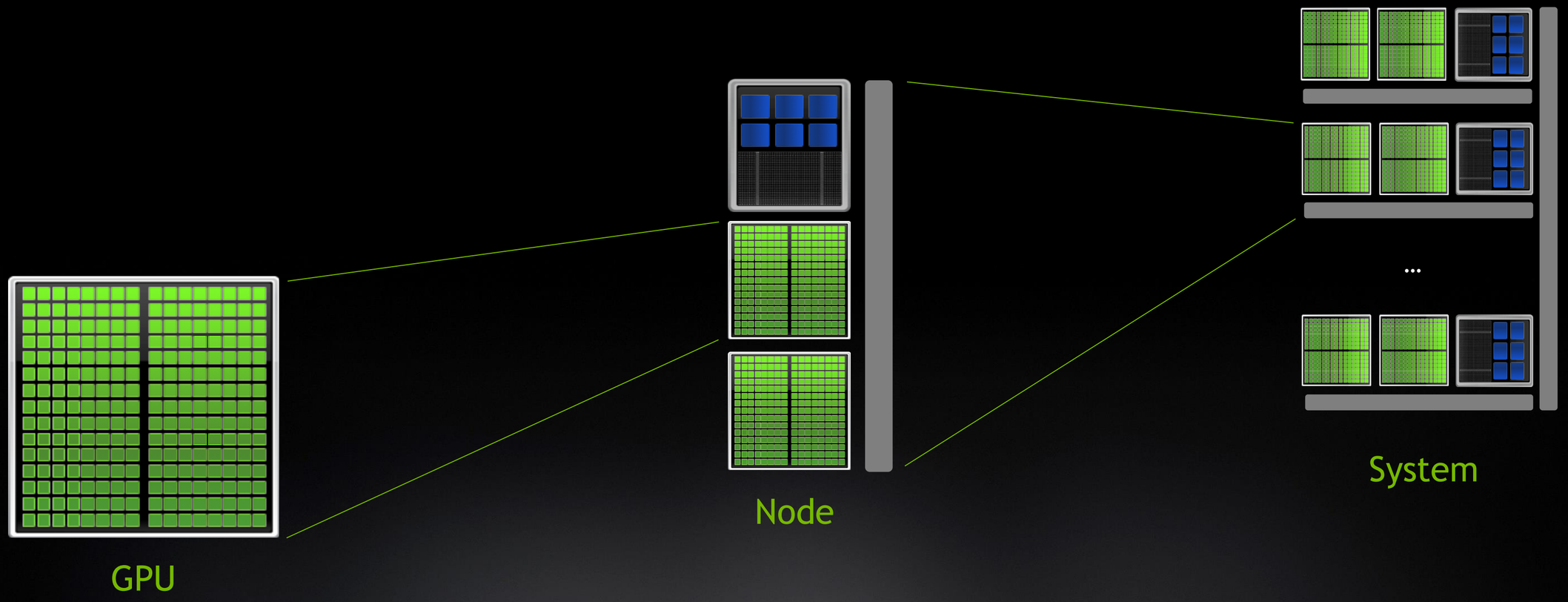


Program Scope

CUDA C++
OPENACC
STANDARD LANGUAGES
SYNCHRONIZATION
PRECISION
TASK GRAPHS

PROGRAMMING GPU-ACCELERATED HPC SYSTEMS

GPU | CPU | Interconnect



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

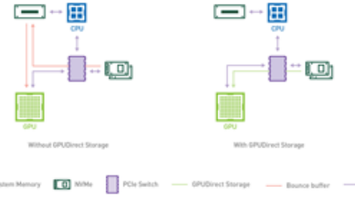
CUDA 11.0
Major Feature Areas

MIG,
TensorCores,
NVLink

arm

DirectX11 Windows 10

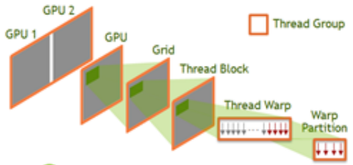
GPUDirect
Storage

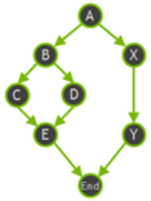


New Platform Capabilities

- *A100 Features*
- *CUDA on Arm Servers*

Cooperative
Groups





Fork-Join
Graphs

New Reduce
Op

```
reduce( tile, value, OP );
```

Asynchronous
Copy

Programming Model Updates

- *Ampere Programming Model*
- *New APIs for CUDA Graphs*
- *Flexible Thread Programming*
- *Memory Management APIs*

Nsight Compute

Kernel Profiling with
Rooflining

Nsight Systems

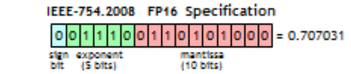
System trace
for Ampere

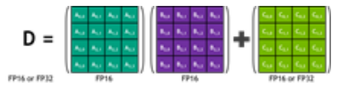
Developer Tools

- *Support for Ampere*
- *Roofline plots with Nsight*
- *Next generation correctness tools*

libcu++

```
atomicAdd(&h, (half)1.15f);

half2 hvec(0.94f, -2.13f);
atomicAdd(&h2, hvec);
```





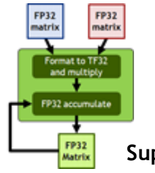
D = AB + C

Link Time
Optimization

CUDA C++

- *C++ Modernization*
- *Parallel standard C++ library*
- *Low precision datatypes and WMMA*

Hardware decoder
acceleration with nvJPEG



Support for BF16 and TF32
datatypes

Strong and weak
scaling on multi-GPU
systems

Math Libraries

- *Low precision datatypes in Ampere*
- *3rd Gen Tensor Core support*
- *Leverage increased memory bandwidth, shared memory and L2 cache*

NVIDIA

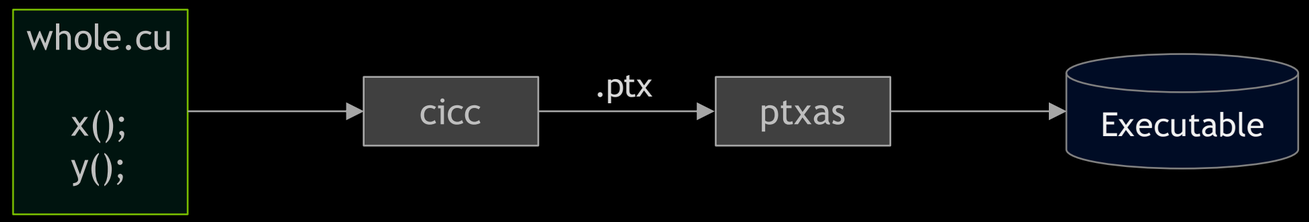
COMPILERS

NVCC HIGHLIGHTS IN CUDA 11.0 TOOLKIT

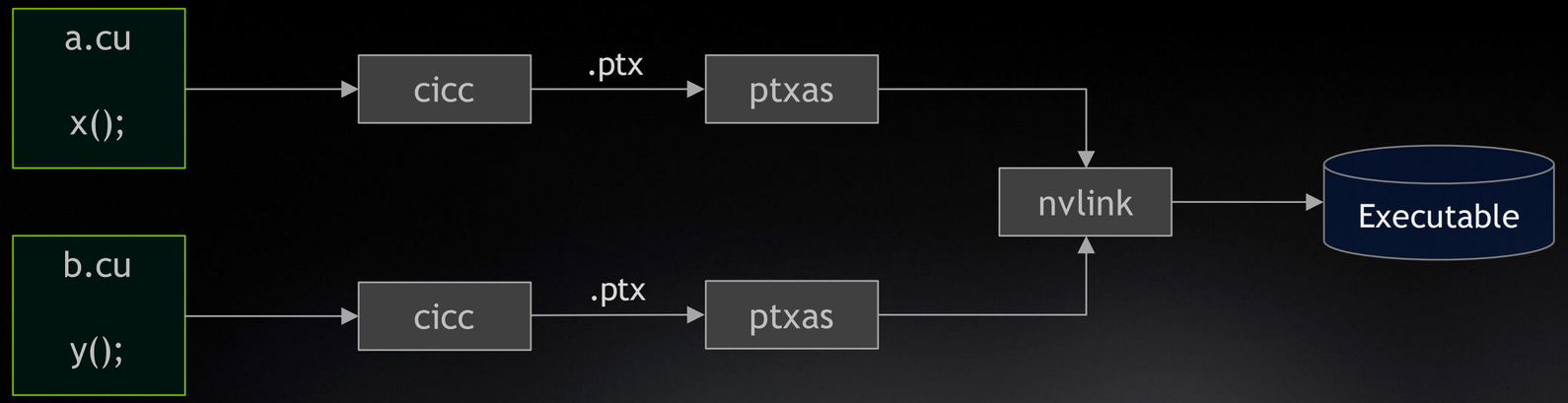
Key Features	
ISO C++ 17 CUDA Support	*Preview feature*
Link-Time Optimization	*Preview feature*

New in CUDA 11.0	
Accept duplicate CLI options across all NVCC sub-components	
Host compiler support for GCC 9, clang 9, PGI 20.1	
Host compiler version check override option	--allow-unsupported-compiler
Native AArch64 NVCC binary with ARM Allinea Studio 19.2 C/C++ and PGI 20 host compiler support	

LINK-TIME OPTIMIZATION



Whole-Program Compilation



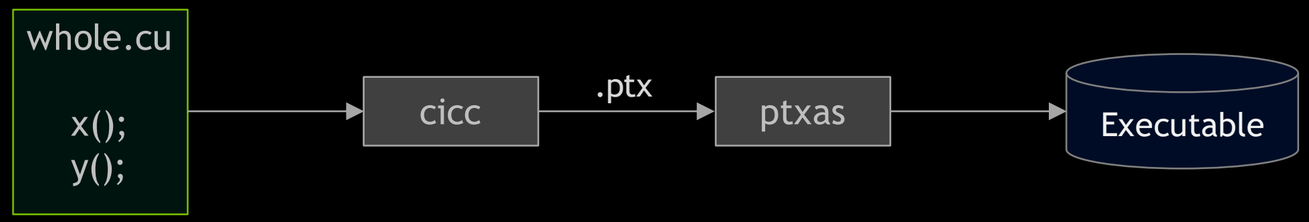
Separate Compilation

All cross-compilation-unit calls must link via ABI, e.g:

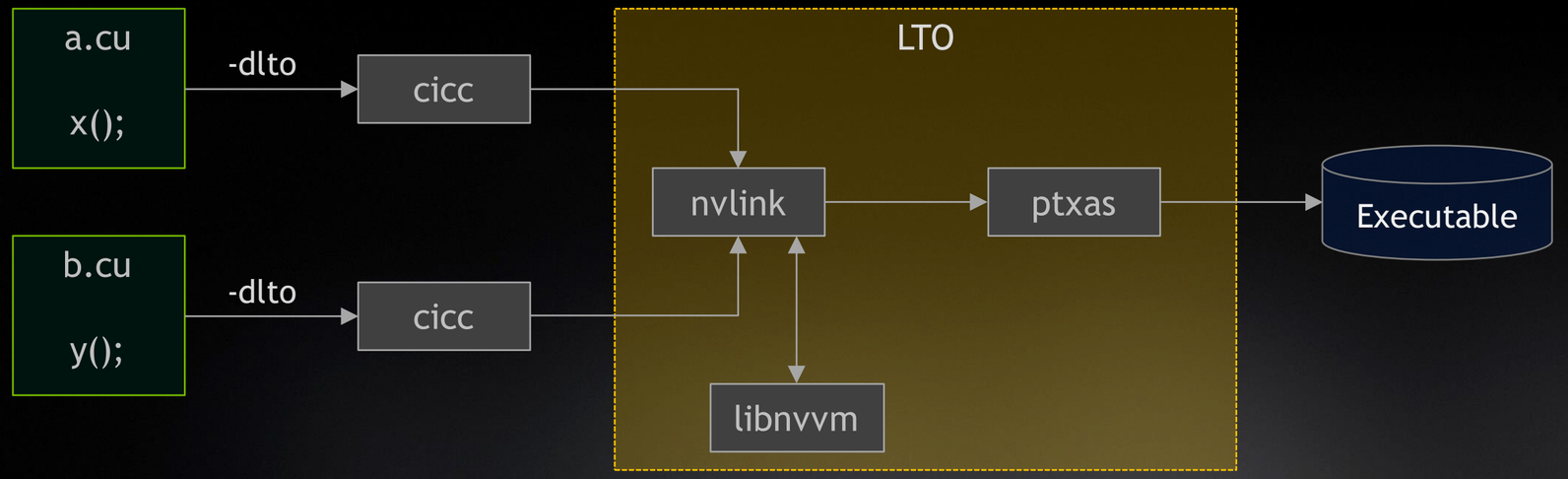
`x()` → `y()`

ABI calls incur call overheads

LINK-TIME OPTIMIZATION



Whole-Program Compilation



Link-Time Optimization

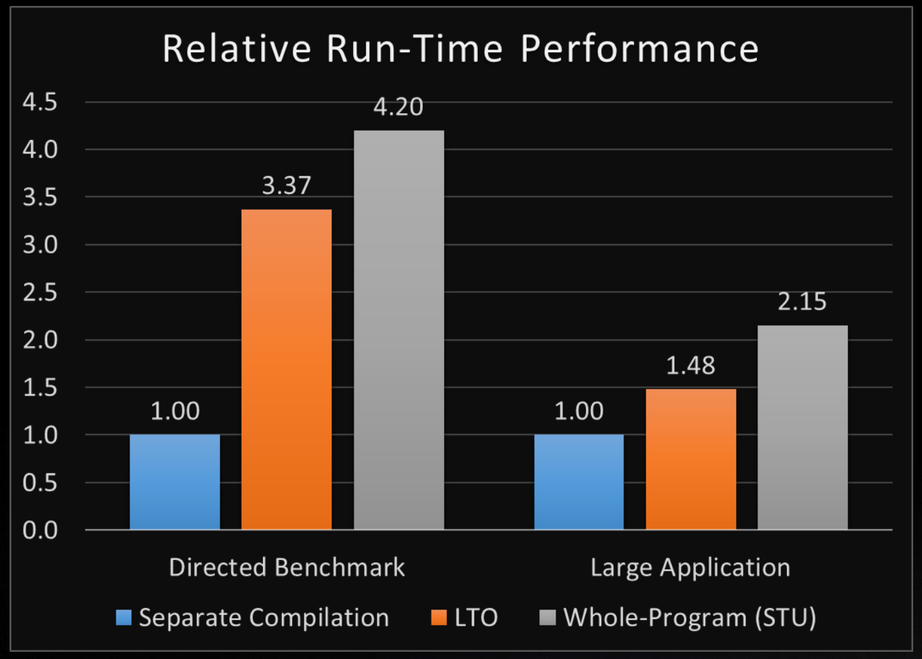
Permits inlining of device functions across modules

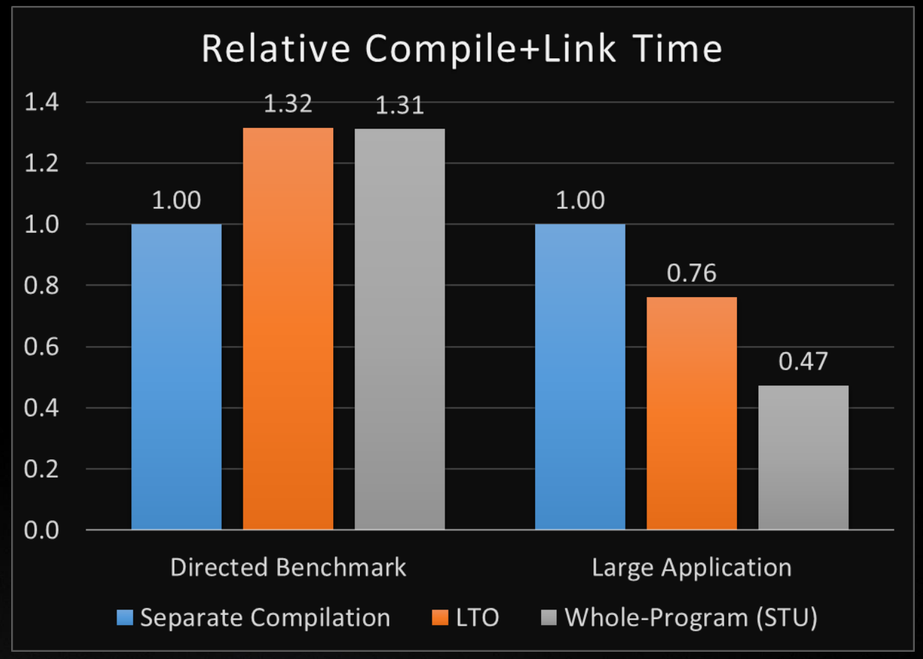
Mitigates ABI call overheads

Facilitates Dead Code Elimination

LINK-TIME OPTIMIZATION

Preview Release in CUDA 11.0



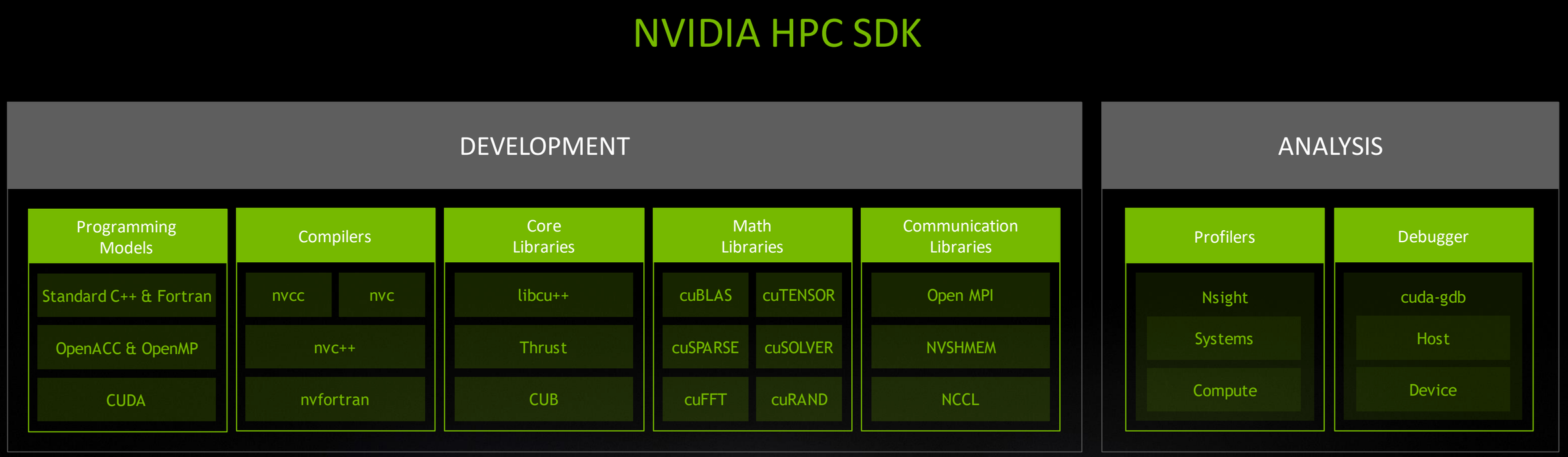


Enabled through ***-dlto*** option for compile and link steps

Partial LTO (mix of separate compilation & LTO) supported

AVAILABLE NOW: THE NVIDIA HPC SDK

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



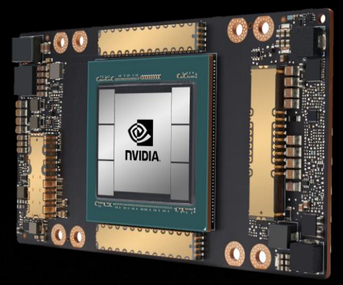
Develop for the NVIDIA HPC Platform: GPU, CPU and Interconnect

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

7-8 Releases Per Year | Freely Available

HPC COMPILERS

NVC | NVC++ | NVFORTRAN

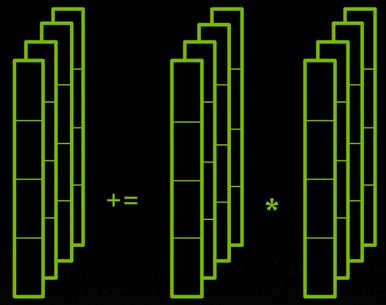


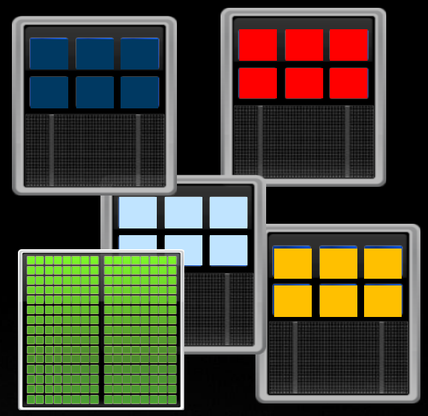
Fortran

NVIDIA CUDA

OpenACC
More Science, Less Programming

OpenMP





Accelerated
Latest GPUs
Automatic Acceleration

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++ 20.5
- 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++ in CUDA 10.2:
  - `std::atomic<T>`
- In libcu++ in CUDA 11.0:
  - `std::barrier`
  - `std::counting_semaphore`
  - `std::atomic<T>::wait/notify_*`
- In libcu++ in the future:
  - `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`

```
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

PARALLEL C++

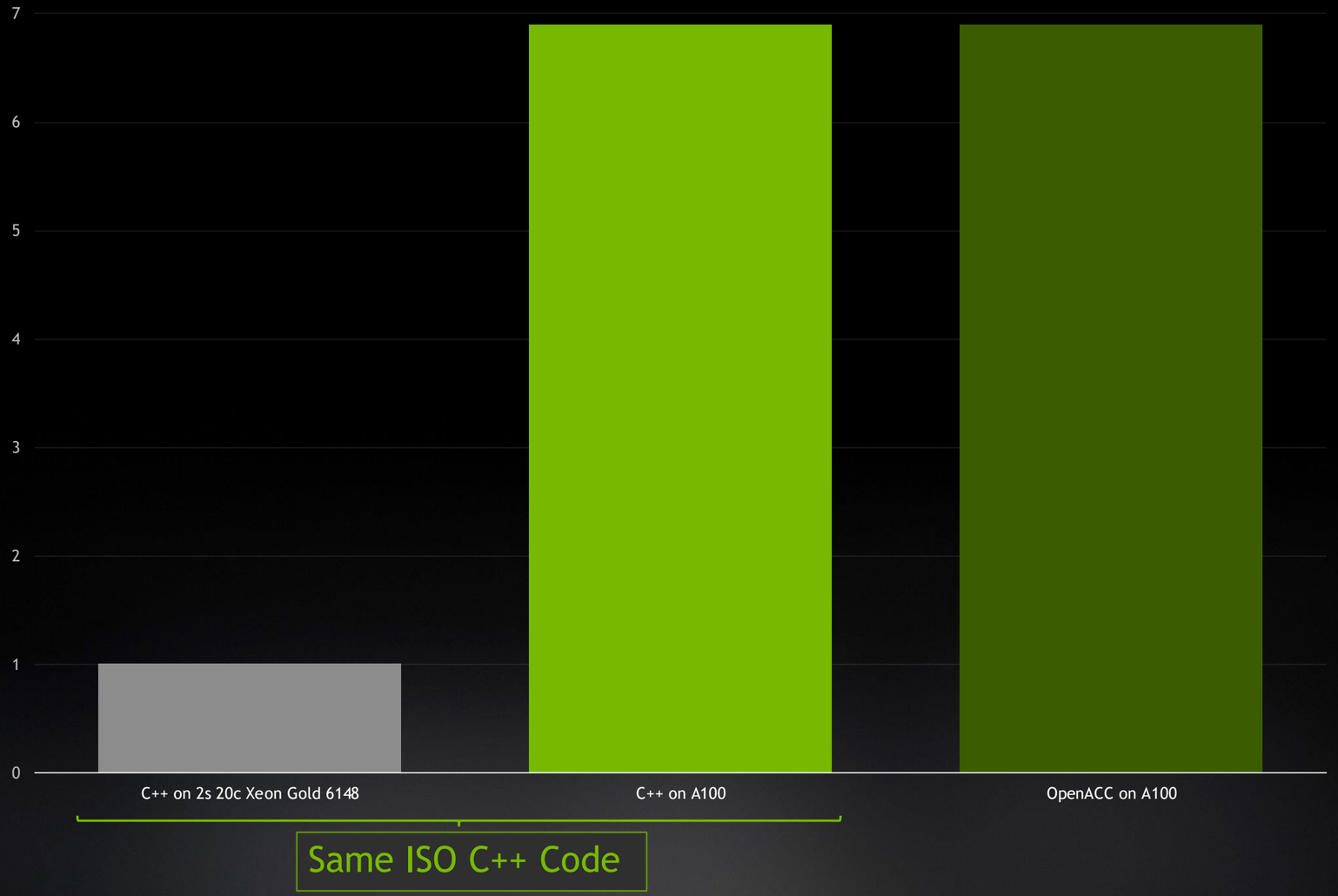
- Composable, compact and elegant
- Easy to read and maintain
- ISO Standard
- Portable – nvc++, g++, icpc, MSVC, …

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



NVIDIA

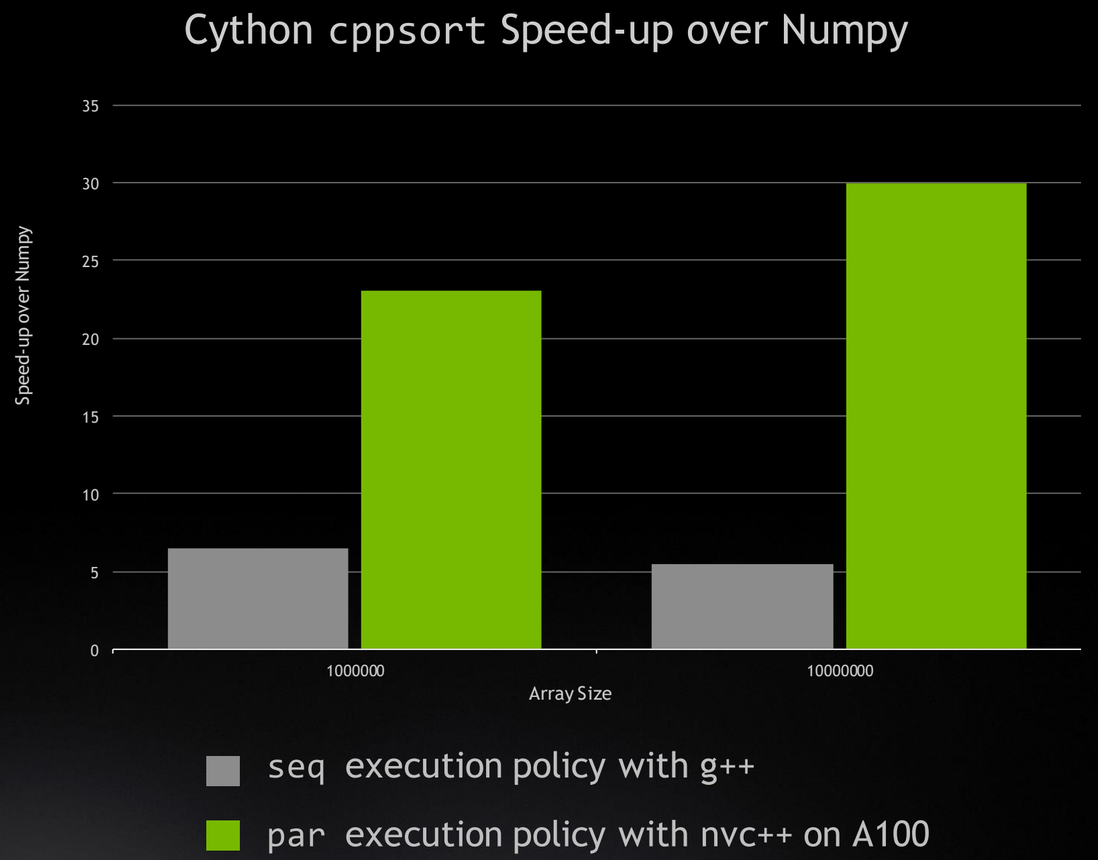
PARALLEL C++ & CYTHON

Using NVC++ and CYTHON to Accelerate Python

A100 Performance for Python

- Access to C++ performance with Cython
- A100 Acceleration with NVC++ stdpar
- Up to 30X Speed-up over Numpy

```
def cppsort(np.ndarray[np.float_t, ndim=1] x):

    cdef vector[float] vec
    vec.resize(x.shape[0])
    copy_n(&x[0], len(x), vec.begin())
    sort(par, vec.begin(), vec.end())
    copy_n(vec.begin(), len(x), &x[0])
```



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

FORTRAN DO CONCURRENT

```
!$ACC KERNELS
!$ACC LOOP INDEPENDENT
        do k=y_min,y_max
!$ACC LOOP INDEPENDENT PRIVATE(right_flux,left_flux,top_flux,bottom_flux,
                        total_flux, min_cell_volume,energy_change,recip_volume)
                do j=x_min,x_max

                        left_flux   = (xarea(j  ,k  )*(xvel0(j,k)+xvel0(j,k+1)                          &
                                      +xvel0(j  ,k  )+xvel0(j  ,k+1)))*0.25_8*dt*0.5
                        right_flux  = (xarea(j+1,k  )*(xvel0(j+1,k  )+xvel0(j+1,k+1)                    &
                                      +xvel0(j+1,k  )+xvel0(j+1,k+1)))*0.25_8*dt*0.5
                        bottom_flux = (yarea(j  ,k  )*(yvel0(j  ,k  )+yvel0(j+1,k  )                    &
                                      +yvel0(j  ,k  )+yvel0(j+1,k  )))*0.25_8*dt*0.5
                        top_flux    = (yarea(j  ,k+1)*(yvel0(j  ,k+1)+yvel0(j+1,k+1)                    &
                                      +yvel0(j  ,k+1)+yvel0(j+1,k+1)))*0.25_8*dt*0.5
                        total_flux=right_flux-left_flux+top_flux-bottom_flux

                        volume_change(j,k) = volume(j,k)/(volume(j,k)+total_flux)

                        min_cell_volume = MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux     &
                                             ,volume(j,k)+right_flux-left_flux                          &
                                             ,volume(j,k)+top_flux-bottom_flux)
                        recip_volume=1.0/volume(j,k)
                        energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*...
                        energy1(j,k)=energy0(j,k)-energy_change
                        density1(j,k)=density0(j,k)*volume_change(j,k)

                enddo
        enddo
!$ACC END KERNELS
```

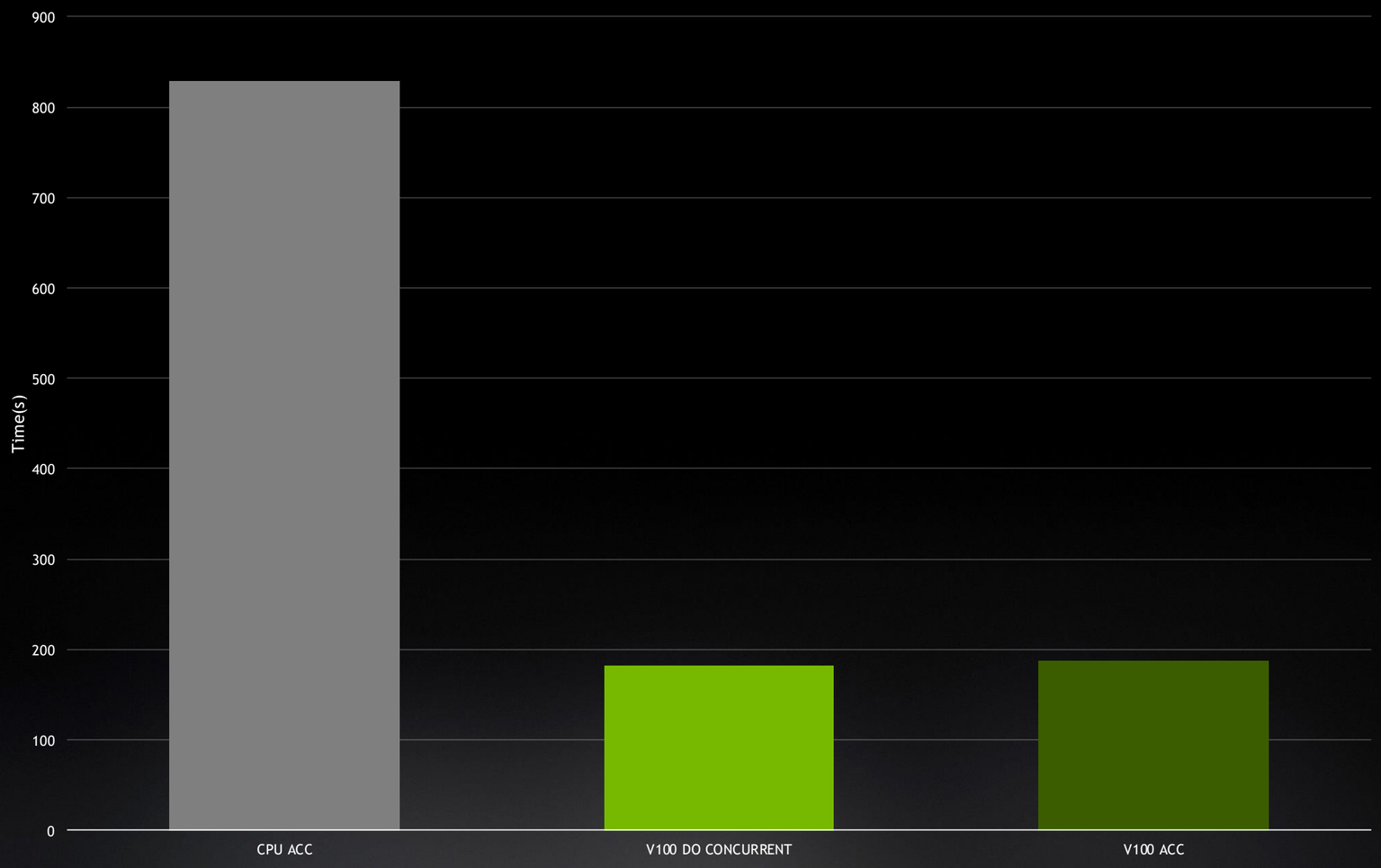
Fortran with OpenACC

```
DO CONCURRENT (k=y_min:y_max, j=x_min:x_max)                                    &
        LOCAL (right_flux,left_flux,top_flux,bottom_flux,total_flux,            &
               min_cell_volume,energy_change,recip_volume)

        left_flux   = (xarea(j  ,k  )*(xvel0(j,k)+xvel0(j,k+1)                  &
                      +xvel0(j  ,k  )+xvel0(j  ,k+1)))*0.25_8*dt*0.5
        right_flux  = (xarea(j+1,k  )*(xvel0(j+1,k  )+xvel0(j+1,k+1)            &
                      +xvel0(j+1,k  )+xvel0(j+1,k+1)))*0.25_8*dt*0.5
        bottom_flux = (yarea(j  ,k  )*(yvel0(j  ,k  )+yvel0(j+1,k  )            &
                      +yvel0(j  ,k  )+yvel0(j+1,k  )))*0.25_8*dt*0.5
        top_flux    = (yarea(j  ,k+1)*(yvel0(j  ,k+1)+yvel0(j+1,k+1)            &
                      +yvel0(j  ,k+1)+yvel0(j+1,k+1)))*0.25_8*dt*0.5
        total_flux=right_flux-left_flux+top_flux-bottom_flux

        volume_change(j,k) = volume(j,k)/(volume(j,k)+total_flux)

        min_cell_volume = MIN(volume(j,k)+right_flux-left_flux+top_flux-bottom_flux &
                             ,volume(j,k)+right_flux-left_flux                      &
                             ,volume(j,k)+top_flux-bottom_flux)
        recip_volume=1.0/volume(j,k)
        energy_change=(pressure(j,k)/density0(j,k)+viscosity(j,k)/density0(j,k))*...
        energy1(j,k)=energy0(j,k)-energy_change
        density1(j,k)=density0(j,k)*volume_change(j,k)

ENDDO
```

ISO Fortran

CLOVERLEAF PERFORMANCE

Time – Lower is Better



CPU System: Skylake 2x20 core Xeon Gold server, one thread per core

NVIDIA

HPC PROGRAMMING IN ISO FORTRAN

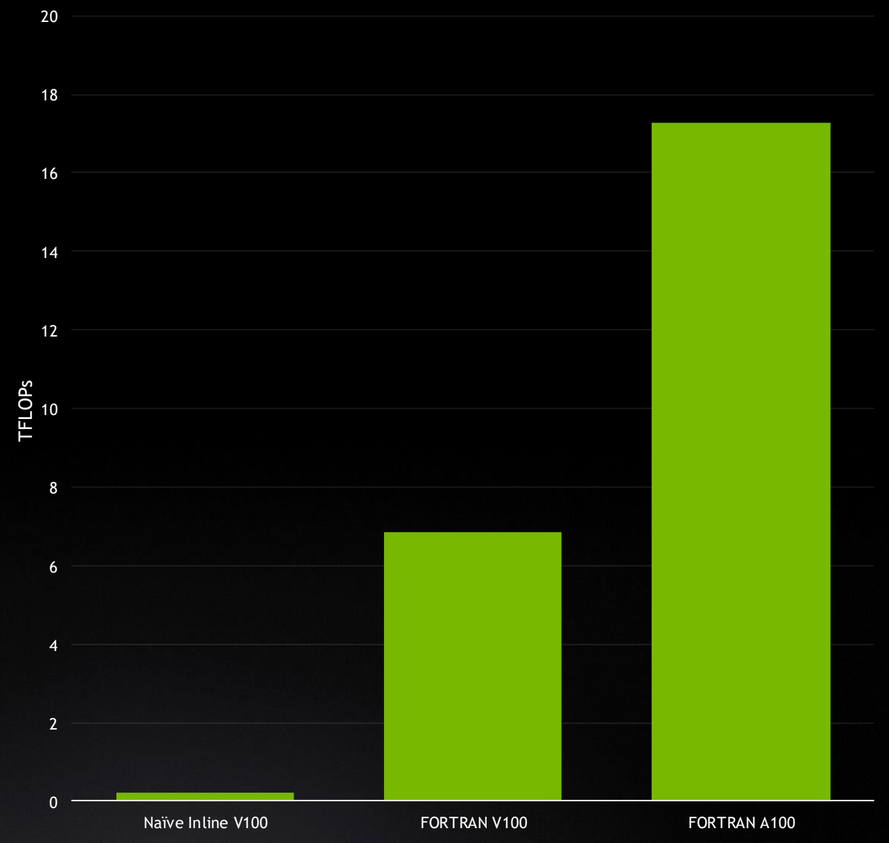
NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

```fortran
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

```fortran
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



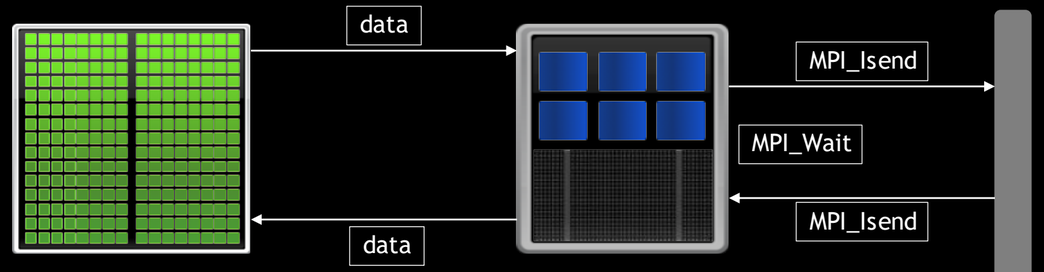
INTRODUCING NVSHMEM

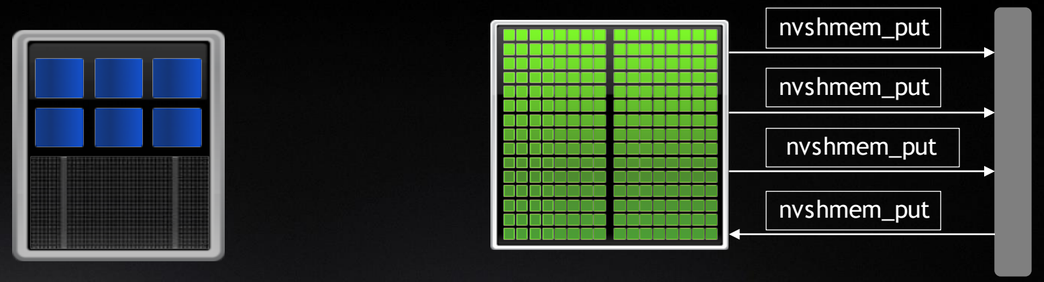
GPU Optimized OpenSHMEM

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

Pre-release Impact

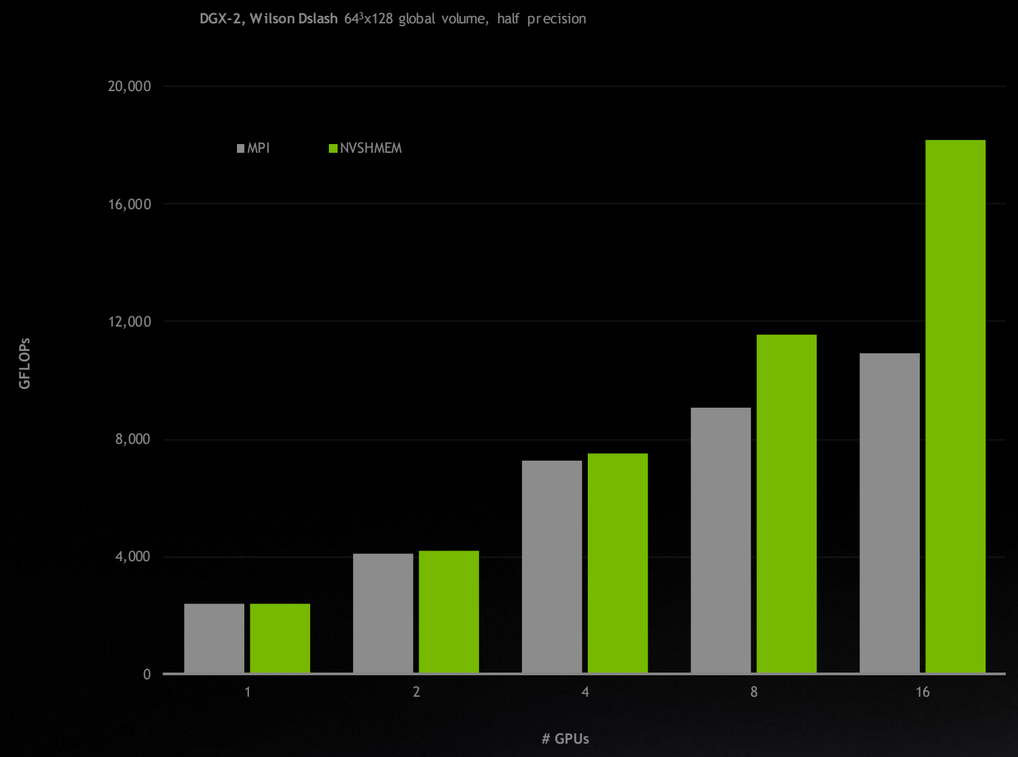
- LBANN, Kokkos/CGSolve, QUDA

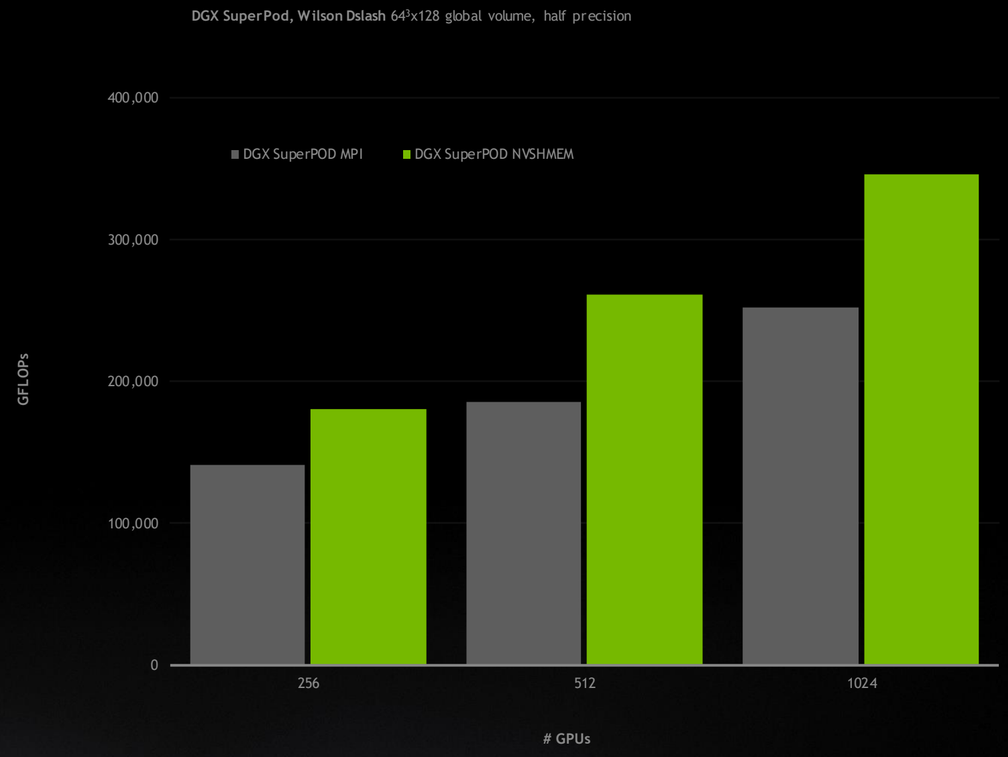




INTRODUCING NVSHMEM

Impact in HPC Applications





QUDA: Quantum Chromodynamics on CUDA

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

MULTI GPU WITH THE NVIDIA HPC SDK

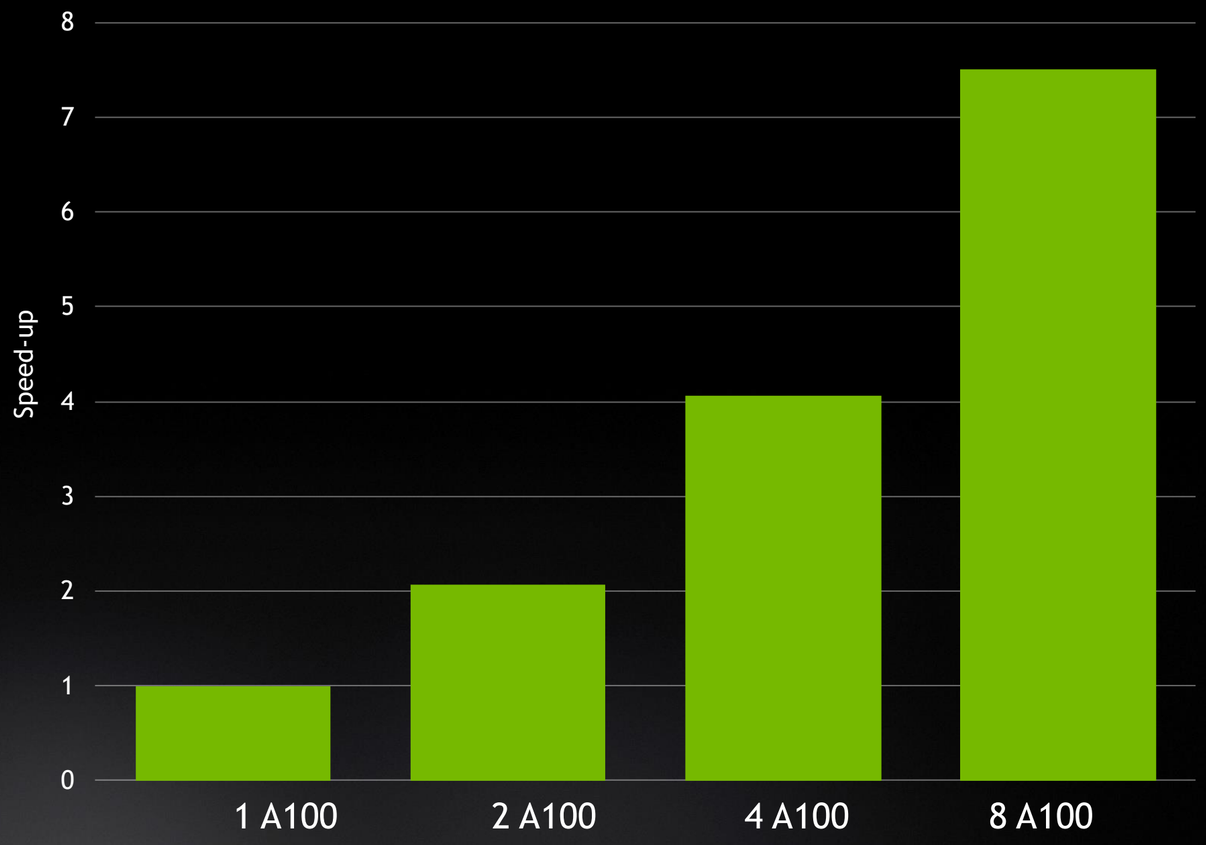
Cloverleaf Hydrodynamics Mini-App

Full Integration provided by HPC SDK

- Fortran + OpenACC + Open MPI

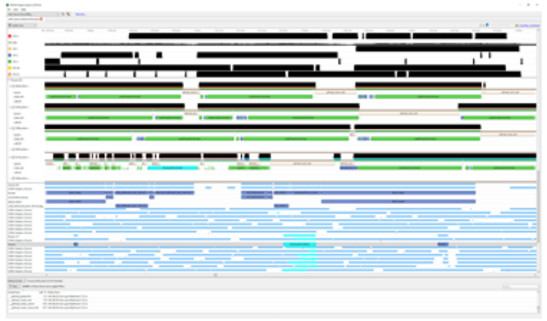
Strong Scaling - Cloverleaf BM128

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



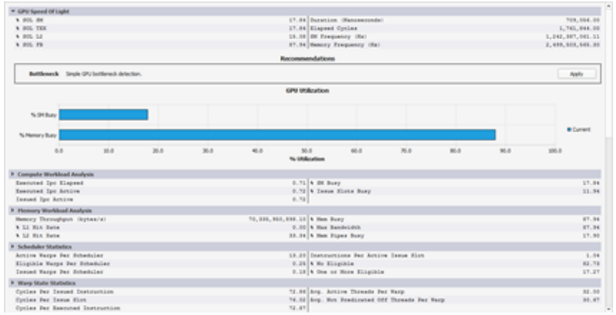
TOOLS

COMPUTE DEVELOPER TOOLS



Nsight Systems

System-wide application algorithm tuning

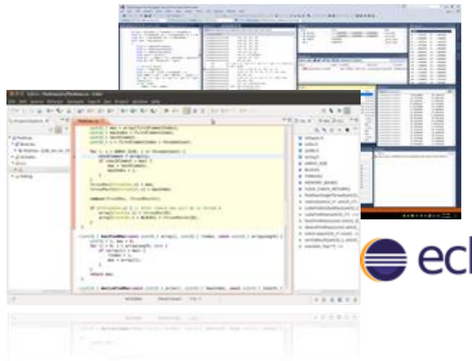


Nsight Compute

CUDA Kernel Profiling and Debugging

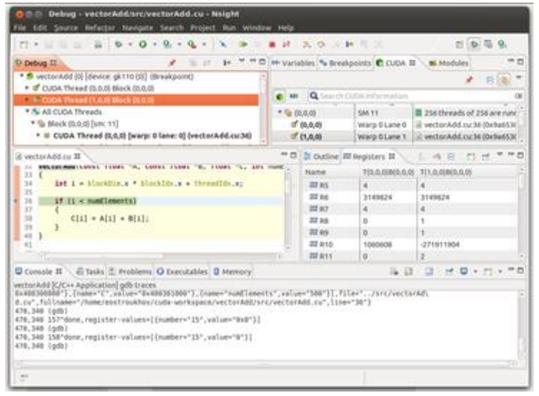
Nsight Graphics

Graphics Shader Profiling and Debugging



IDE Plugins

Nsight Eclipse Edition/Visual Studio (Editor, Debugger)



cuda-gdb

CUDA Kernel Debugging

```
//Out-of-bounds Array Access

__global__ void oobAccess(int* in, int* out)
{
    int bid = blockIdx.x;
    int tid = threadIdx.x;

    if (bid == 4)
    {
        out[tid] = in[dMem[tid]];
    }
}

int main()
{
    ...
    // Array of 8 elements, where element 4 causes the OOB
    std::array<int, Size> hMem = {0, 1, 2, 10, 4, 5, 6, 7};
    cudaMemcpy(d_mem, hMem.data(), size, cudaMemcpyHostToDevice);

    oobAccess<<<10, Size>>>(d_in, d_out);
    cudaDeviceSynchronize();
    ...

$ /usr/local/cuda-11.0/Sanitizer/compute-sanitizer --destroy-on-device-error kernel --show-backtrace no
basic
========= COMPUTE-SANITIZER
Device: Tesla T4
========= Invalid __global__ read of size 4 bytes
=========     at 0x480 in
/tmp/CUDA11.0/ComputeSanitizer/Tests/Memcheck/basic/basic.cu:40:oobAccess(int*,int*)
=========     by thread (3,0,0) in block (4,0,0)
=========     Address 0x7f551f200028 is out of bounds
```

Compute Sanitizer

Memory, Race Checking

NSIGHT SYSTEMS

System profiler

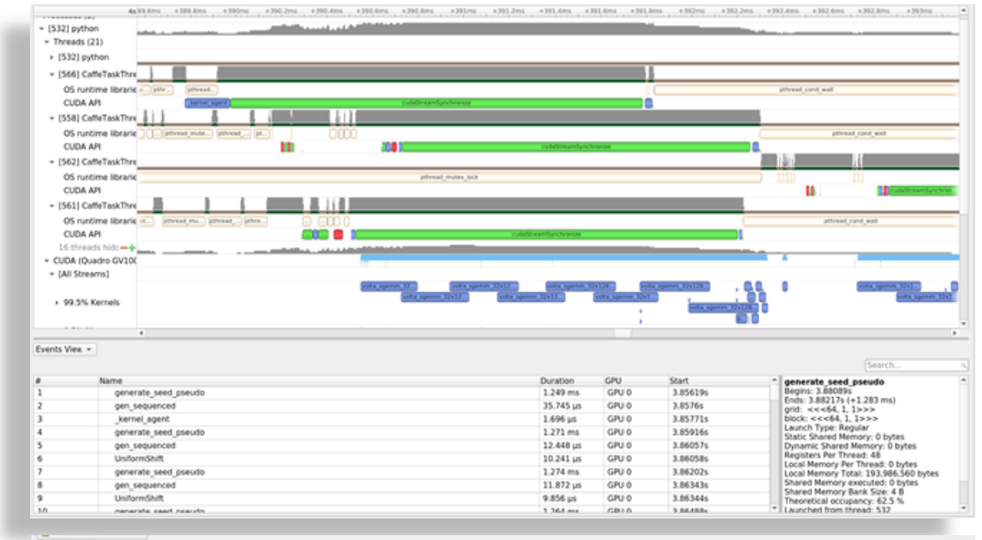
Key Features:

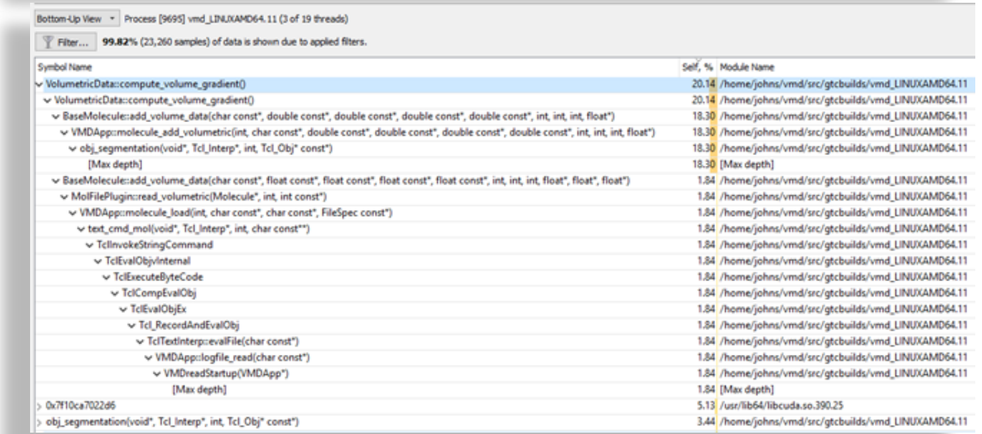
- System-wide application algorithm tuning
  - Multi-process tree support
- Locate optimization opportunities
  - Visualize millions of events on a very fast GUI timeline
  - Or gaps of unused CPU and GPU time
- Balance your workload across multiple CPUs and GPUs
  - CPU algorithms, utilization and thread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

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

GPUs: Pascal+

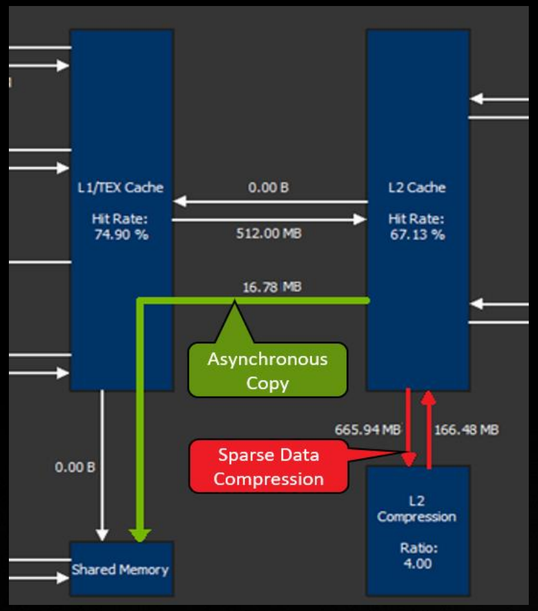
Docs/product: https://developer.nvidia.com/nsight-systems





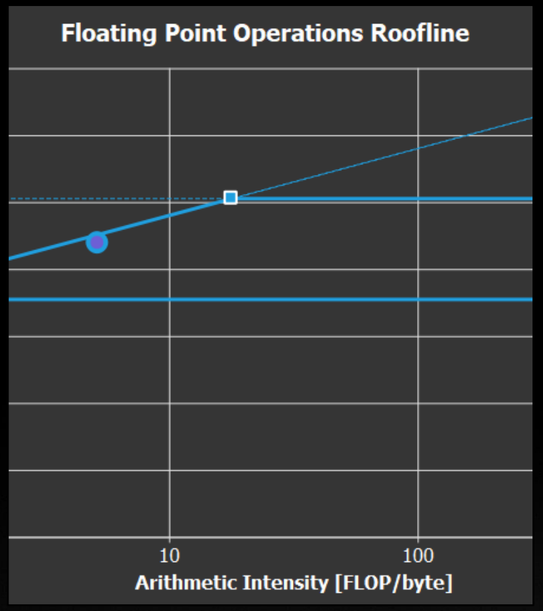
NSIGHT COMPUTE 2020

2020.2 Now available



Chips Update

A100 GPU Support

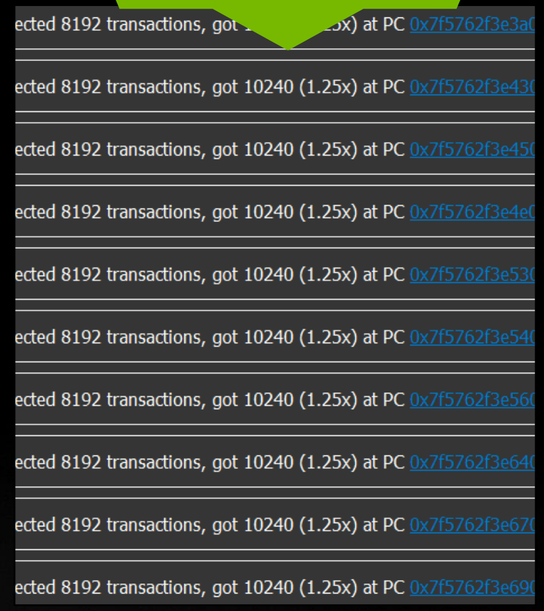


Advanced Analysis

Roofline
New Memory Tables

Workflow Improvements

Hot Spot Tables
Section Links



Other Changes

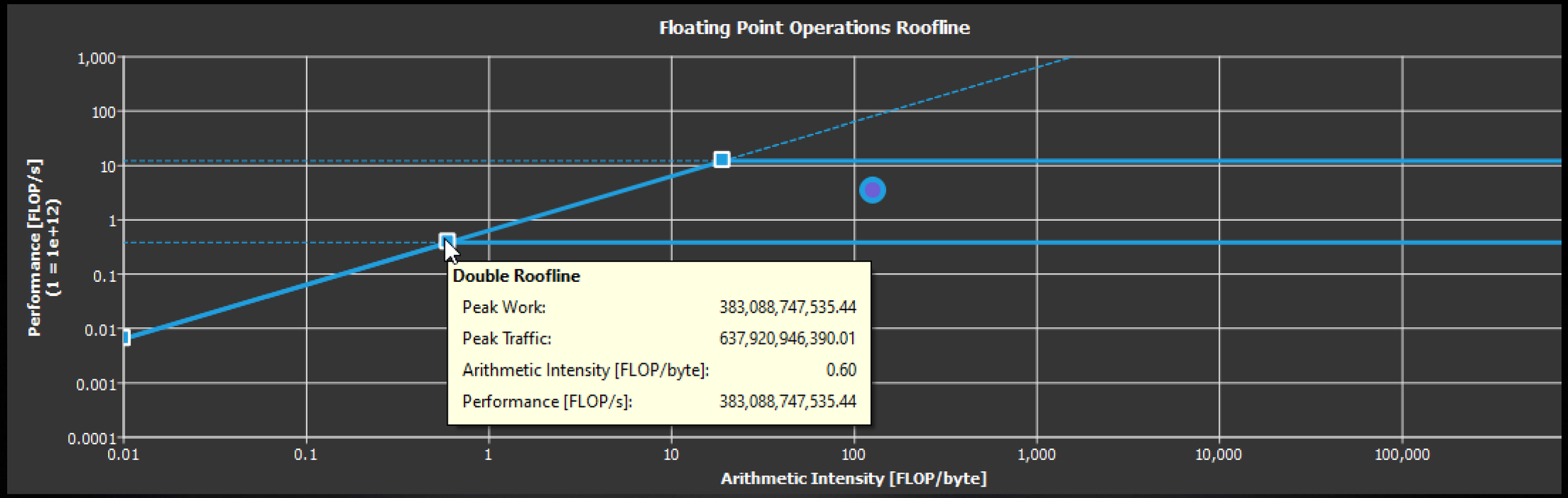
New Rules, Names

For more information see: S21771 - Optimizing CUDA kernels using Nsight Compute

NSIGHT COMPUTE 2020

New Roofline Analysis

Efficient way to evaluate kernel characteristics, quickly understand potential directions for further improvements or existing limiters



Inputs — Arithmetic Intensity (FLOPS/bytes)
Performance (FLOPS/s)

Ceilings — Peak Memory Bandwidth
Peak FP32/FP64 Performance

COMPUTE-SANITIZER

Command Line Interface (CLI) Tool Based On The Sanitizer API

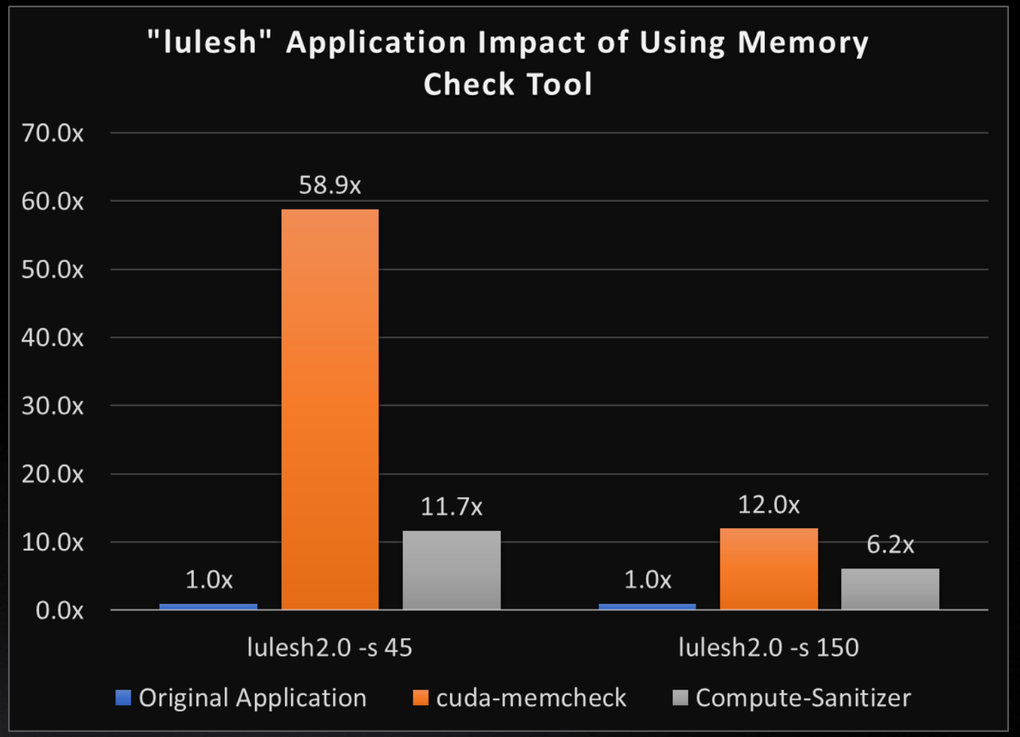
Next-Gen Replacement Tool for `cuda-memcheck`

Significant performance improvement of 2x - 5x compared with `cuda-memcheck` (depending on application size)

Performance gain for applications using libraries such as CUSOLVER, CUFFT or DL frameworks

`cuda-memcheck` still supported in CUDA 11.0 (does not support Arm SBSA)

https://docs.nvidia.com/cuda/compute-sanitizer



For more information see: S22043 – CUDA Developer Tools: Overview and Exciting New Features

NVIDIA

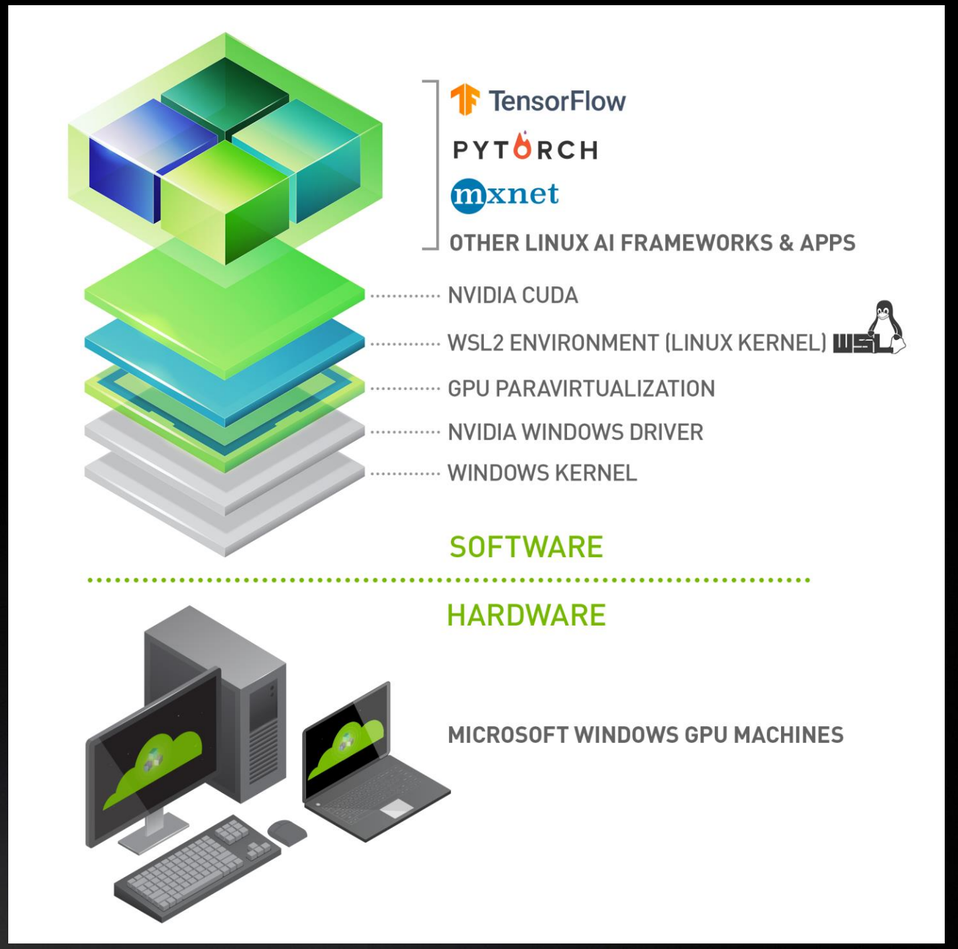
CUDA ON WINDOWS SUBSYSTEM FOR LINUX

Run a Linux kernel **natively** on top of Windows 10

Runs Linux at near full speed **without emulation**

Multi-OS development & testing from a single Windows desktop machine

No need for dual-boot systems - ideal for laptops



GPU-ACCELERATED DATA SCIENCE ON WSL

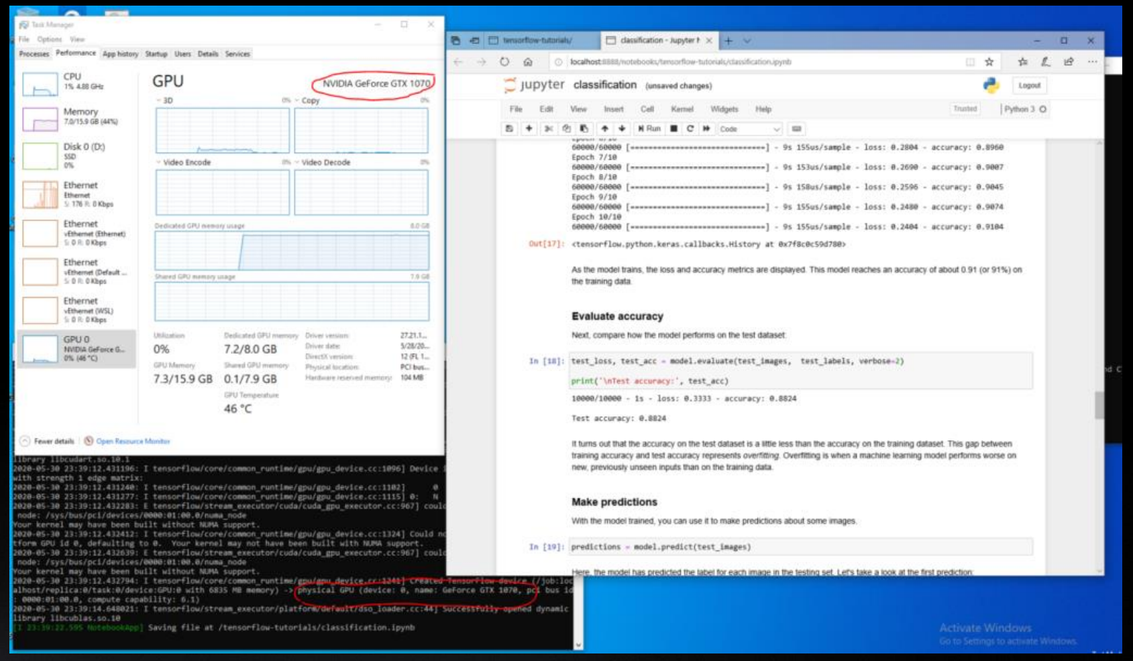
Get the latest version of Docker and run:

- AI Frameworks (PyTorch, TensorFlow)
- RAPIDS & ML Applications
- Jupyter Notebooks

GPU-enabled DirectX, CUDA 11.1 and the NVIDIA Container Toolkit are all available on WSL today

NVML and NCCL support coming soon

See CUDA-on-WSL blog for full details: https://developer.nvidia.com/blog/announcing-cuda-on-windows-subsystem-for-Linux-2/



TensorFlow container running inside WSL 2

NEW FEATURES & IMPROVEMENTS IN CUDA 11

CUTLASS – TENSOR CORE PROGRAMMING MODEL

Warp-Level GEMM and Reusable Components for Linear Algebra Kernels in CUDA

CUTLASS 2.2

Optimal performance on NVIDIA Ampere microarchitecture

New floating-point types: `nv_bfloat16, TF32, double`

Deep software pipelines with async memcopy

CUTLASS 2.1

BLAS-style host API

CUTLASS 2.0

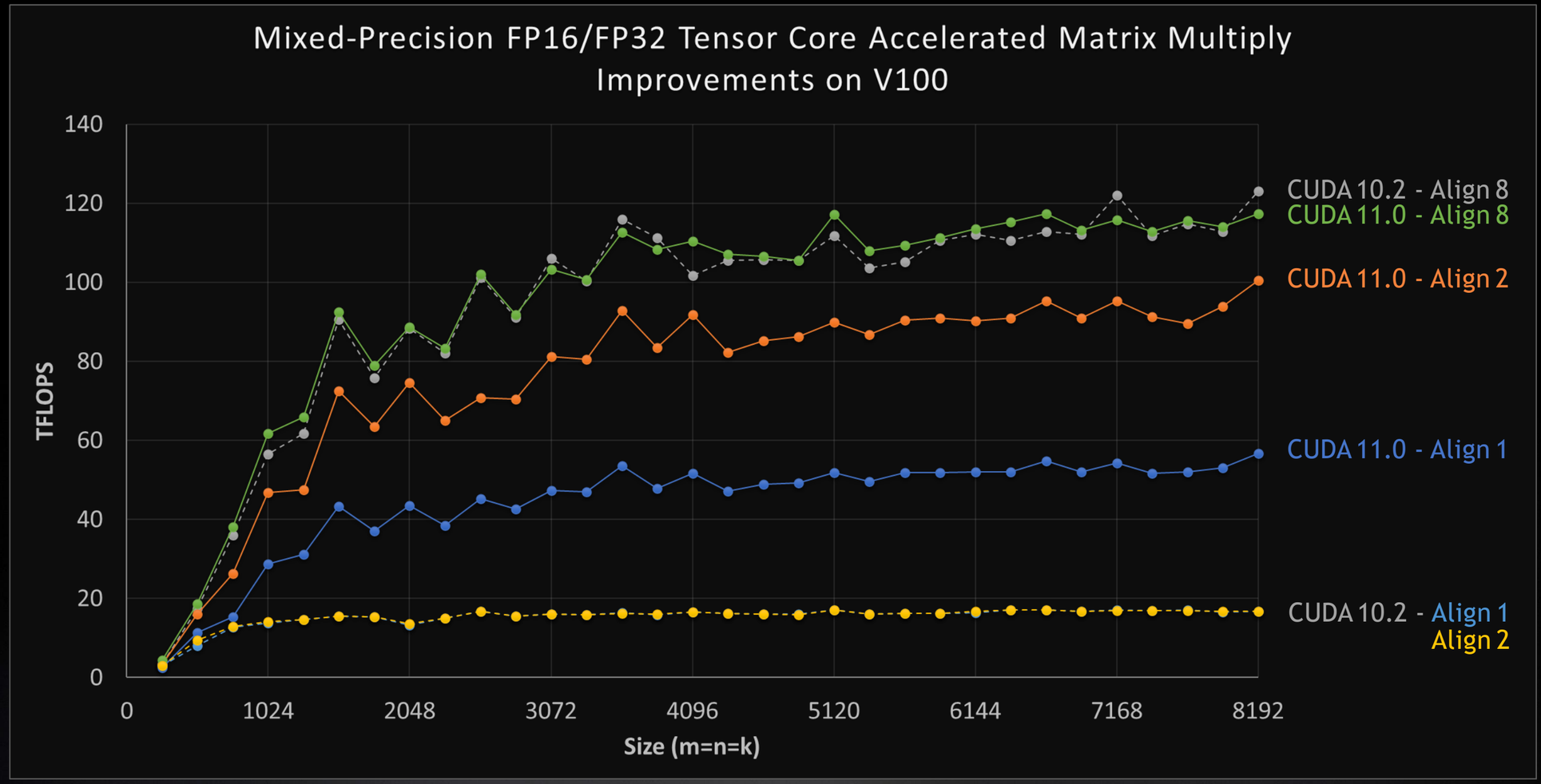
Significant refactoring using modern C++11 programming

```
using Mma = cutlass::gemm::warp::DefaultMmaTensorOp<
  GemmShape<64, 64, 16>,
  half_t, LayoutA,                  // GEMM A operand
  half_t, LayoutB,                  // GEMM B operand
  float, RowMajor                   // GEMM C operand
>;

__shared__ ElementA smem_buffer_A[Mma::Shape::kM * GemmK];
__shared__ ElementB smem_buffer_B[Mma::Shape::kN * GemmK];

// Construct iterators into SMEM tiles
Mma::IteratorA iter_A({smem_buffer_A, lda}, thread_id);
Mma::IteratorB iter_B({smem_buffer_B, ldb}, thread_id);

Mma::FragmentA frag_A;
Mma::FragmentB frag_B;
Mma::FragmentC accum;

Mma mma;

accum.clear();

#pragma unroll 1
for (int k = 0; k < GemmK; k += Mma::Shape::kK) {


  iter_A.load(frag_A);  // Load fragments from A and B matrices
  iter_B.load(frag_B);

  ++iter_A; ++iter_B;   // Advance along GEMM K to next tile in A
                        //   and B matrices

                        // Compute matrix product
  mma(accum, frag_A, frag_B, accum);
}
```

For more information see: S21745 - Developing CUDA Kernels to Push Tensor Cores to the Absolute Limit

cuBLAS

Eliminating Alignment Requirements To Activate Tensor Cores for MMA



AlignN means alignment to 16-bit multiplies of N. For example, align8 are problems aligned to 128bits or 16 bytes.

MATH LIBRARY DEVICE EXTENSIONS

Introducing cuFFTDx: Device Extension

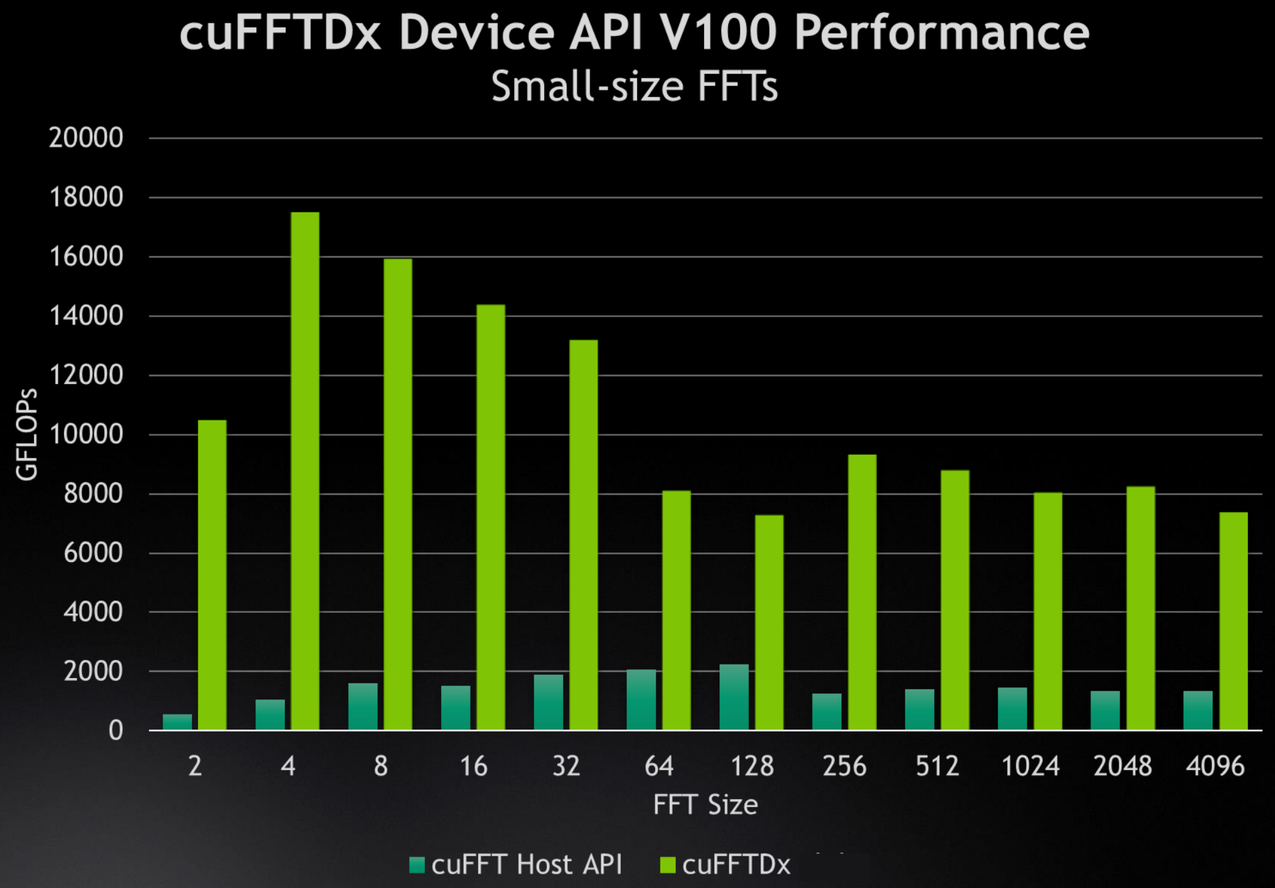
Available in Math Library EA Program

Device callable library

Retain and reuse on-chip data

Inline FFTs in user kernels

Combine multiple FFT operations



https://developer.nvidia.com/CUDAMathLibraryEA

ISO C++ == Language + Standard Library

ISO C++ == Language + Standard Library

CUDA C++ == Language

libcu++ : THE CUDA C++ STANDARD LIBRARY

ISO C++ == Language + Standard Library

CUDA C++ == Language + **libcu++**

Strictly conforming to ISO C++, plus conforming extensions

Opt-in, Heterogeneous, Incremental

cuda::std::

Opt-in	Does not interfere with or replace your host standard library
Heterogeneous	Copyable/Movable objects can migrate between host & device Host & Device can call all member functions Host & Device can concurrently use synchronization primitives*
Incremental	A subset of the standard library today Each release adds more functionality

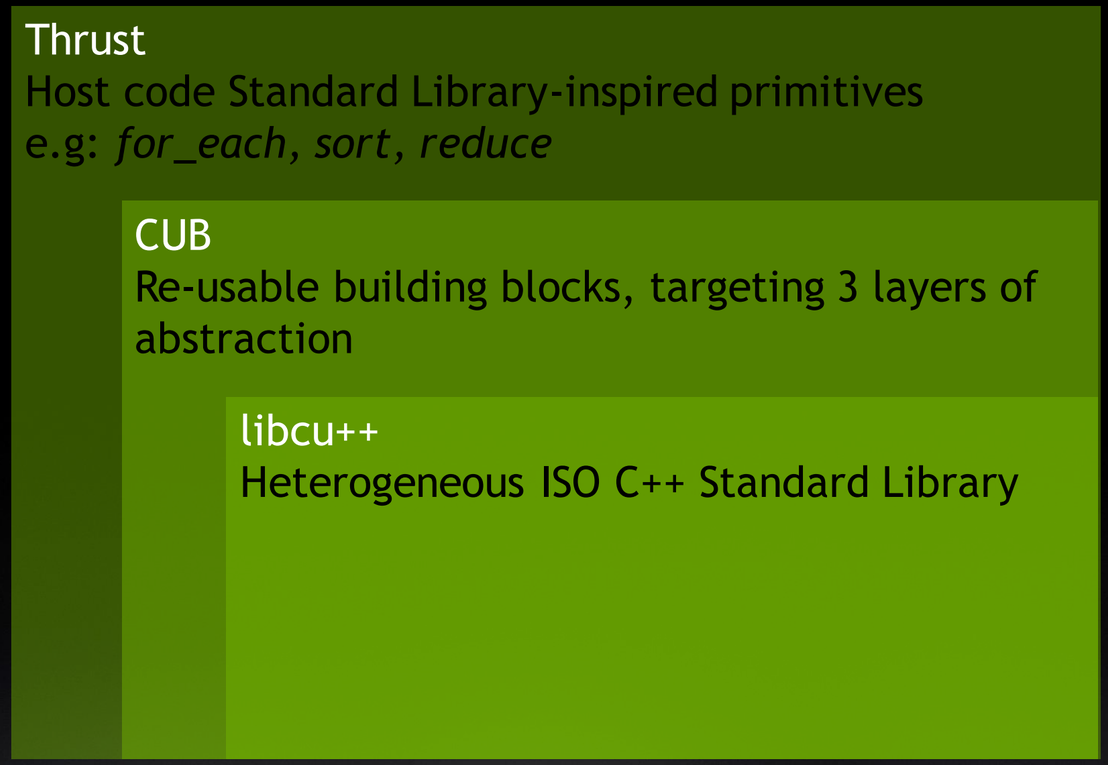
*Synchronization primitives must be in managed memory and be declared with **`cuda::std::thread_scope_system`**

libcu++ NAMESPACE HIERARCHY

```
// ISO C++, __host__ only
#include <atomic>
std::atomic<int> x;

// CUDA C++, __host__ __device__
// Strictly conforming to the ISO C++
#include <cuda/std/atomic>
cuda::std::atomic<int> x;

// CUDA C++, __host__ __device__
// Conforming extensions to ISO C++
#include <cuda/atomic>
cuda::atomic<int, cuda::thread_scope_block> x;
```

For more information see: S21262 - The CUDA C++ Standard Library

CUDA C++ HETEROGENEOUS ARCHITECTURE



CUB is now a fully-supported component of the CUDA Toolkit. Thrust integrates CUB's high performance kernels.

CUB: CUDA UNBOUND

Reusable Software Components for Every Layer of the CUDA Programming Model

Device-wide primitives

Parallel sort, prefix scan, reduction, histogram, etc.
Compatible with CUDA dynamic parallelism

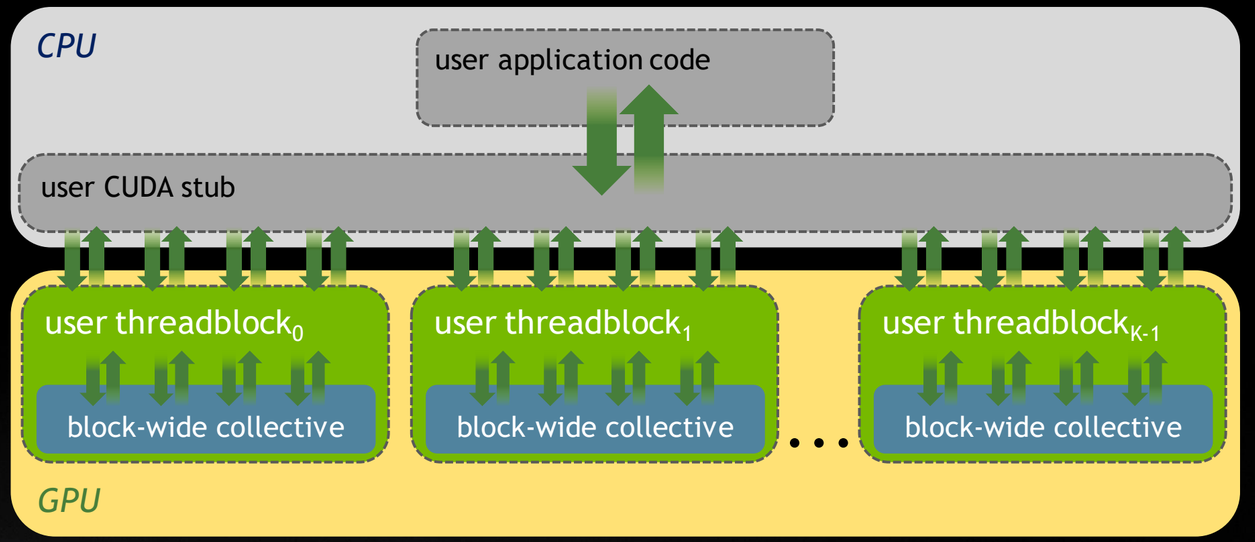
Block-wide "collective" primitives

Cooperative I/O, sort, scan, reduction, histogram, etc.
Compatible with arbitrary thread block sizes and types

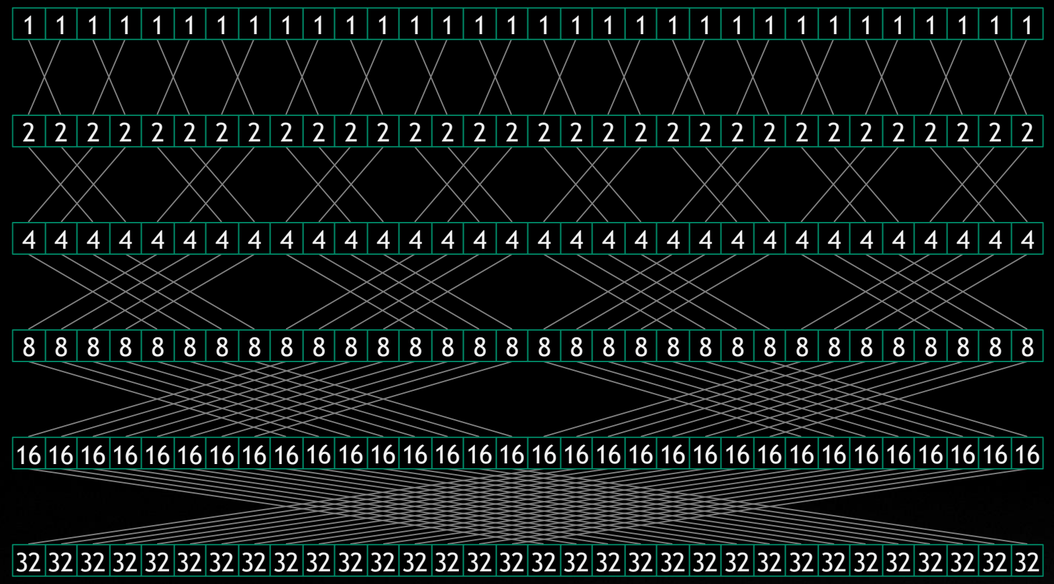
Warp-wide "collective" primitives

Cooperative warp-wide prefix scan, reduction, etc.

Safely specialized for each underlying CUDA architecture

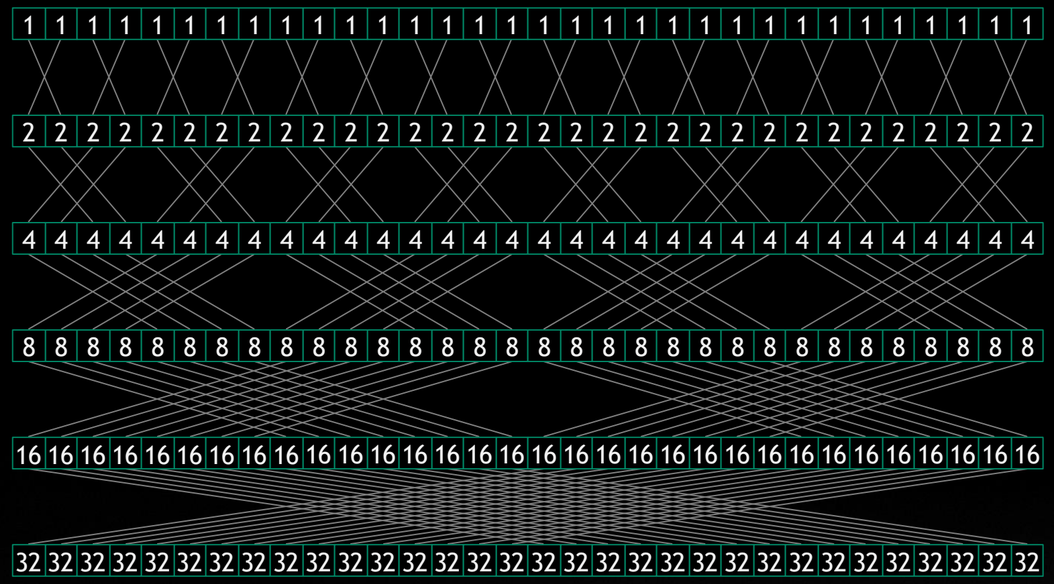


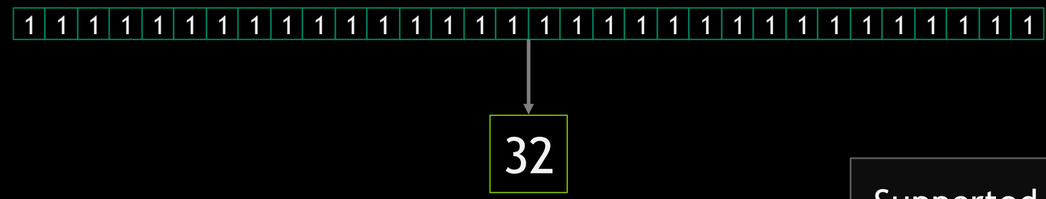
WARP-WIDE REDUCTION USING __shfl




```
__device__ int reduce(int value) {
      value += __shfl_xor_sync(0xFFFFFFFF, value, 1);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 2);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 4);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 8);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 16);

      return value;
}
```

WARP-WIDE REDUCTION IN A SINGLE STEP





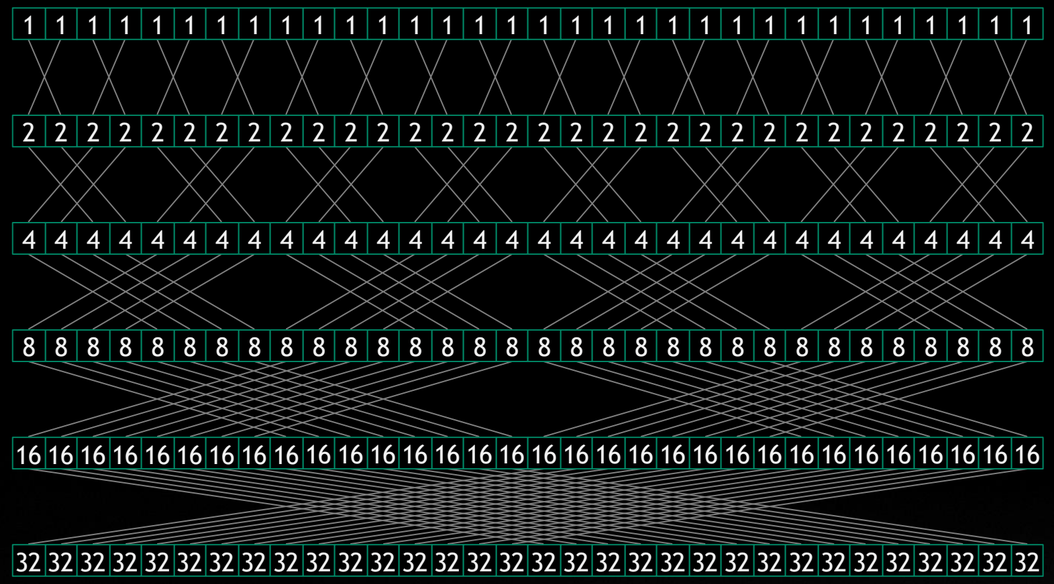
Supported operations

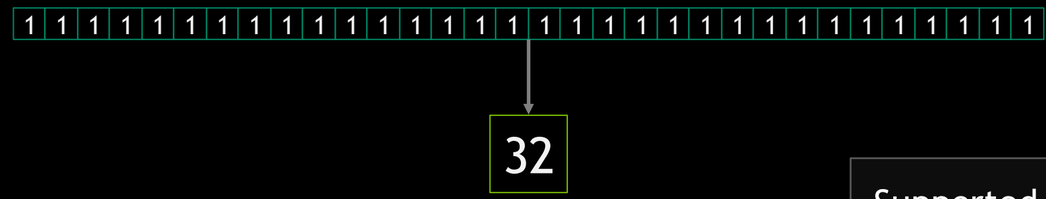
add
min
max
and
or
xor

```
__device__ int reduce(int value) {
      value += __shfl_xor_sync(0xFFFFFFFF, value, 1);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 2);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 4);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 8);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 16);

      return value;
}
```

```
int total = __reduce_add_sync(0xFFFFFFFF, value);
```

WARP-WIDE REDUCTION IN A SINGLE STEP





Supported operations

add
min
max
and
or
xor

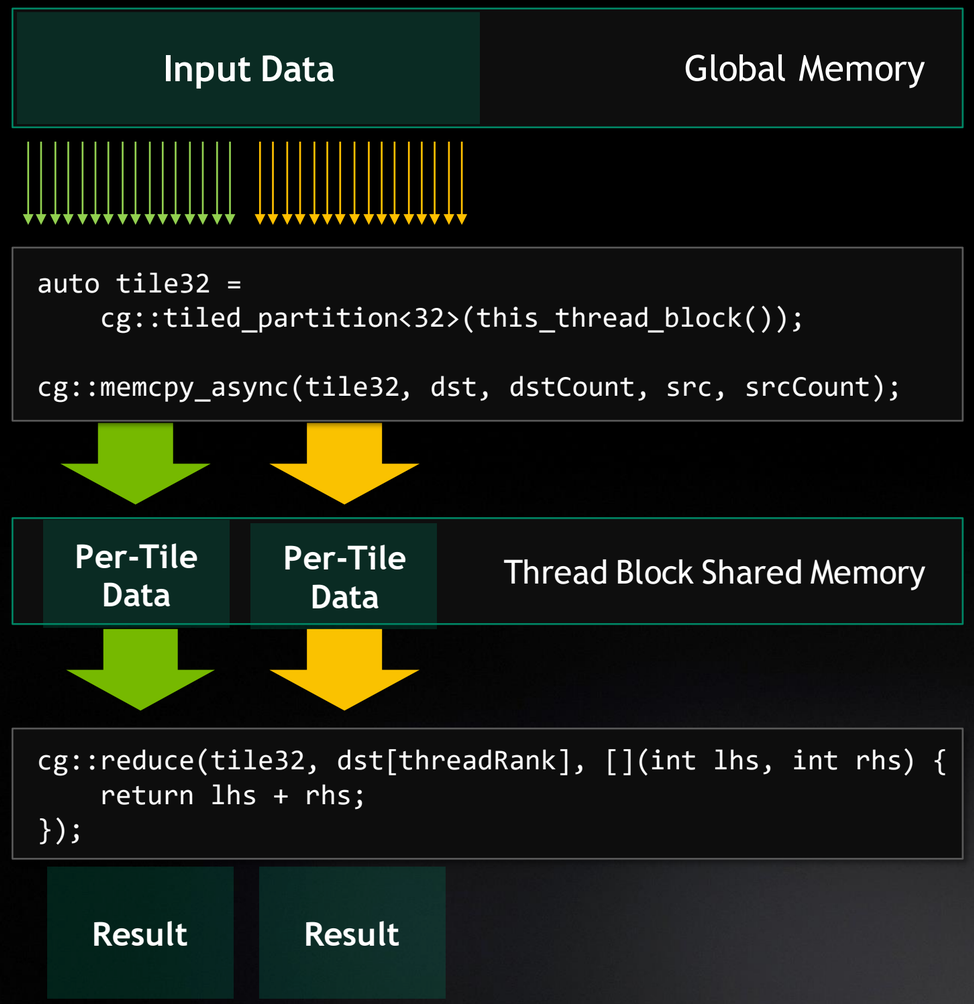
```
__device__ int reduce(int value) {
      value += __shfl_xor_sync(0xFFFFFFFF, value, 1);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 2);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 4);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 8);
      value += __shfl_xor_sync(0xFFFFFFFF, value, 16);

      return value;
}
```

```
int total = __reduce_add_sync(0xFFFFFFFF, value);
```

```
thread_block_tile<32> tile32 =
            tiled_partition<32>(this_thread_block());

// Works on all GPUs back to Kepler
cg::reduce(tile32, value, cg::plus<int>());
```

COOPERATIVE GROUPS

Cooperative Groups Features Work On All GPU Architectures (incl. Kepler)



Cooperative Groups Updates

No longer requires separate compilation

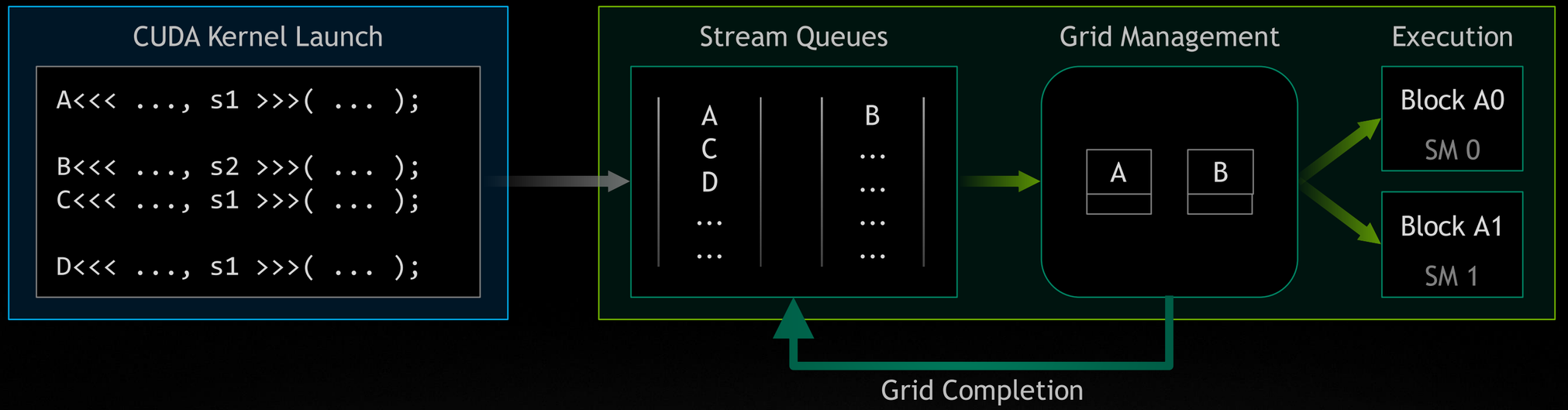
30% faster grid synchronization

New platforms Support (Windows and Linux + MPS)

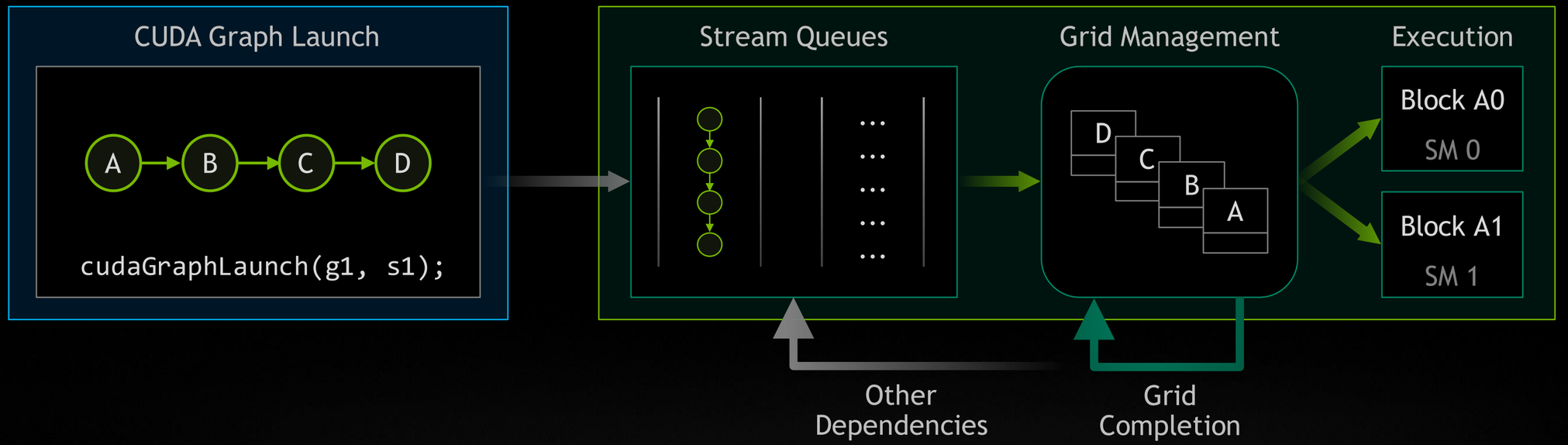
Can now capture cooperative launches in a CUDA graph

cg::reduce also accepts C++ lambda as reduction operation

ANATOMY OF A KERNEL LAUNCH



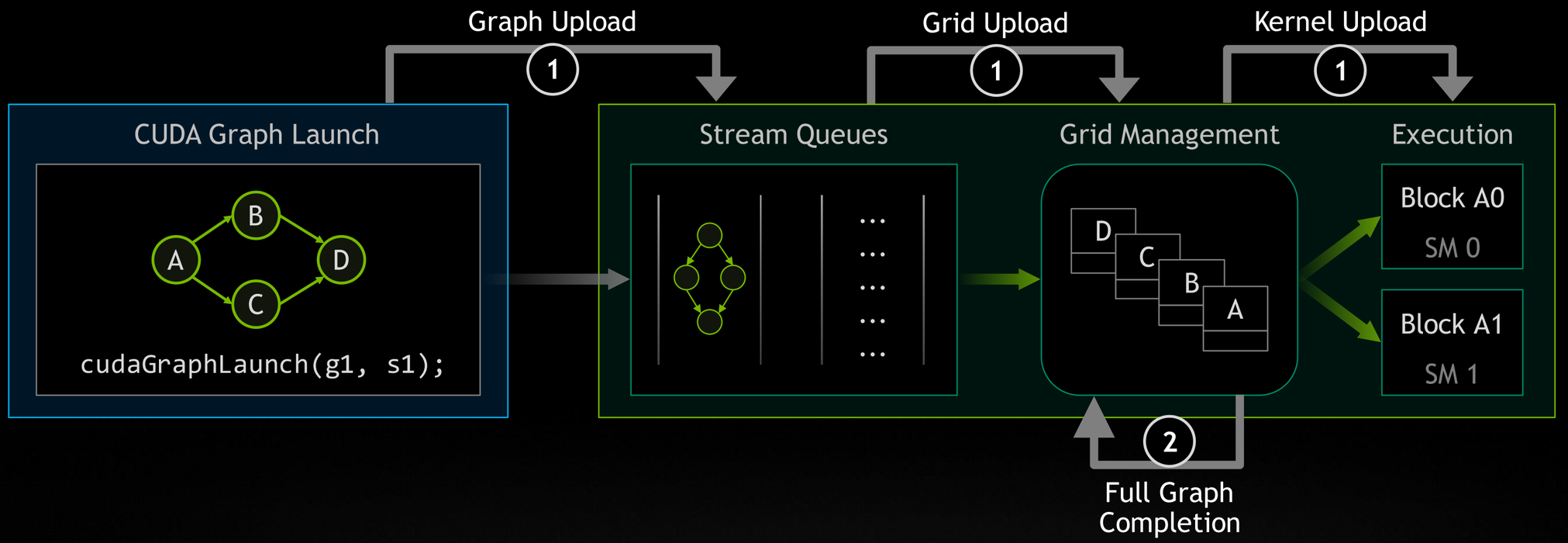
ANATOMY OF A GRAPH LAUNCH



Graph allows launch of multiple kernels in a **single operation**

Graph pushes multiple grids to Grid Management Unit allowing **low-latency dependency resolution**

A100 ACCELERATES GRAPH LAUNCH & EXECUTION

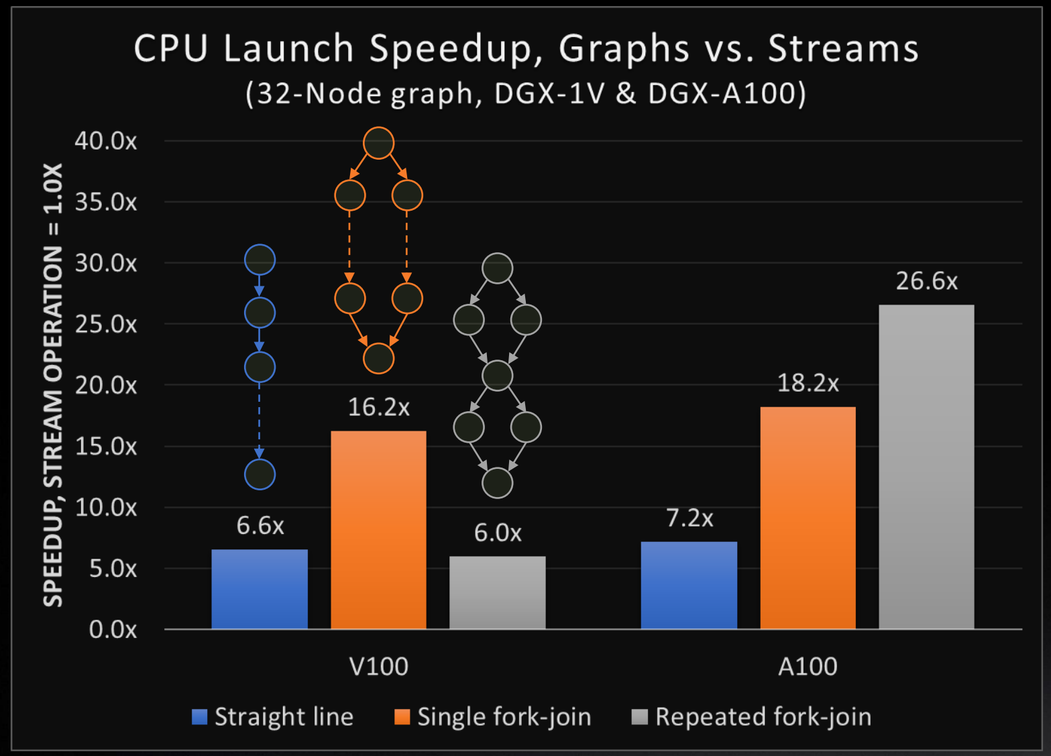


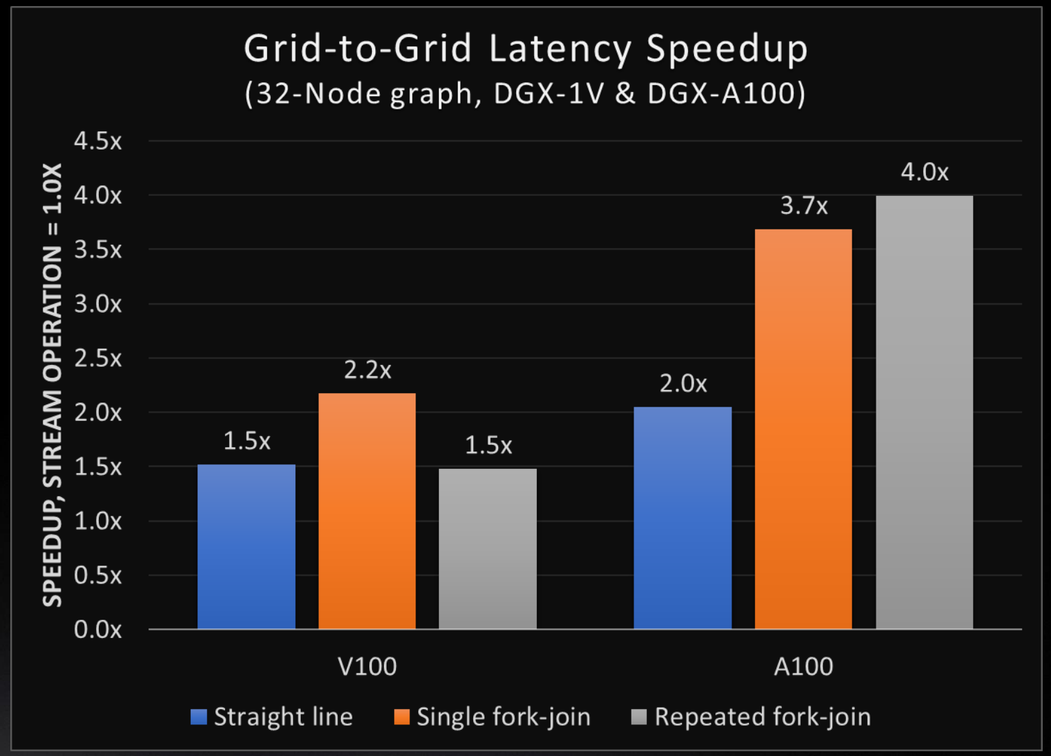
New A100 Execution Optimizations for Task Graphs

1. Grid launch latency reduction via whole-graph upload of grid & kernel data
2. Overhead reduction via accelerated dependency resolution

LATENCIES & OVERHEADS: GRAPHS vs. STREAMS

Empty Kernel Launches – Investigating System Overheads





Note: Empty kernel launches – timings show reduction in latency only

GRAPH PARAMETER UPDATE

Fast Parameter Update When Topology Does Not Change

Graph Update

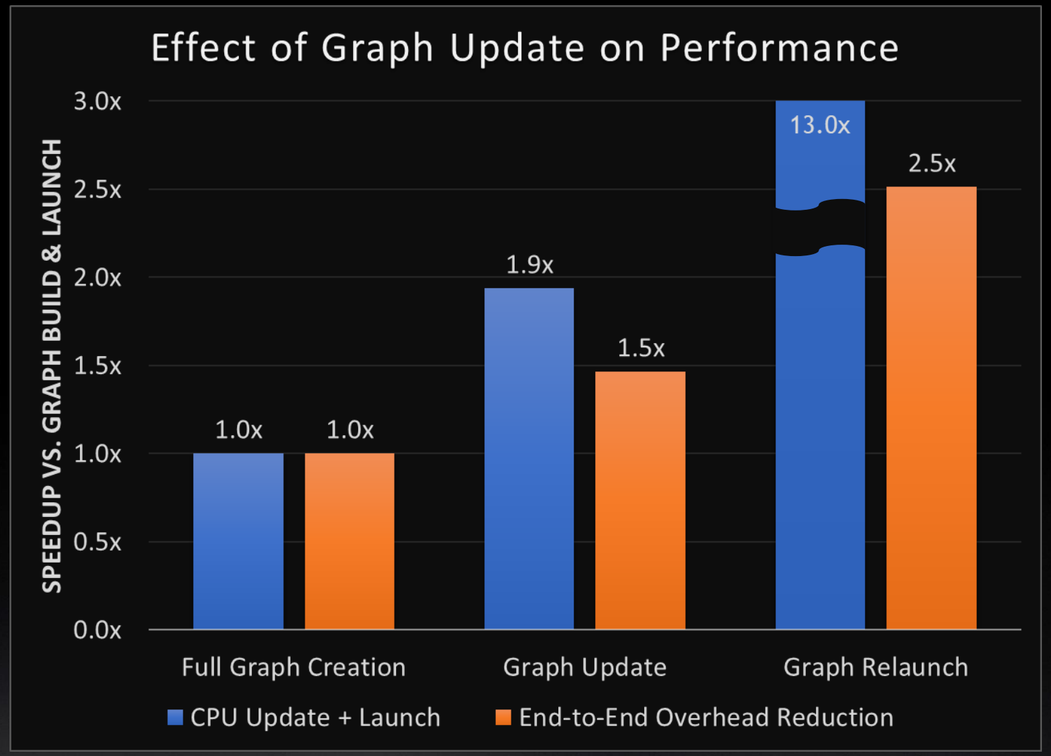
Modify parameters without rebuilding graph

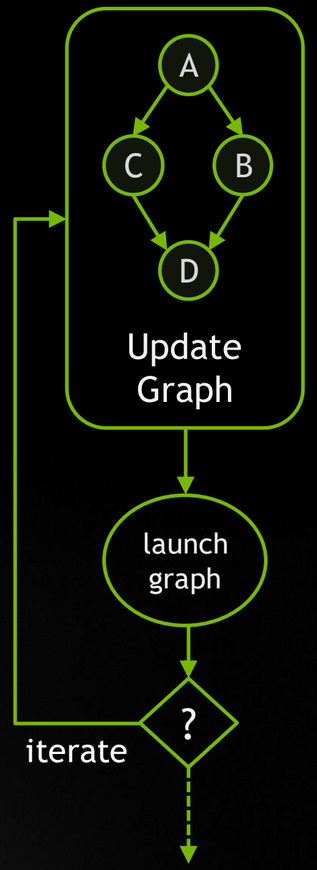
Change launch configuration, kernel parameters, memcopy args, etc.

Topology of graph may not change

Nearly 2x speedup on CPU

50% end-to-end overhead reduction





CUDA VIRTUAL MEMORY MANAGEMENT

Breaking Memory Allocation Into Its Constituent Parts

1. Reserve Virtual Address Range
   `cuMemAddressReserve/Free`

2. Allocate Physical Memory Pages
   `cuMemCreate/Release`

3. Map Pages To Virtual Addresses
   `cuMemMap/Unmap`

4. Manage Access Per-Device
   `cuMemSetAccess`

Control & reserve address ranges

Can remap physical memory

Fine-grained access control

Manage inter-GPU peer-to-peer sharing on a per-allocation basis

Inter-process sharing

For more information see: https://devblogs.nvidia.com/introducing-low-level-gpu-virtual-memory-management/

REFERENCES

Deep dive into any of the topics you've seen by following these links

S21730	Inside the NVIDIA Ampere Architecture
Whitepaper	https://www.nvidia.com/nvidia-ampere-architecture-whitepaper
S22043	CUDA Developer Tools: Overview and Exciting New Features
Developer Blog	https://devblogs.nvidia.com/introducing-low-level-gpu-virtual-memory-management/
S21975	Inside NVIDIA's Multi-Instance GPU Feature
S21170	CUDA on NVIDIA GPU Ampere Architecture, Taking your algorithms to the next level of...
S21819	Optimizing Applications for NVIDIA Ampere GPU Architecture
S22082	Mixed-Precision Training of Neural Networks
S21681	How CUDA Math Libraries Can Help You Unleash the Power of the New NVIDIA A100 GPU
S21745	Developing CUDA Kernels to Push Tensor Cores to the Absolute Limit
S21766	Inside the NVIDIA HPC SDK: the Compilers, Libraries and Tools for Accelerated Computing
S21262	The CUDA C++ Standard Library
S21771	Optimizing CUDA kernels using Nsight Compute