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

INCREMENTAL PORTABLE OPTIMIZATION
OpenACC, OpenMP

PLATFORM SPECIALIZATION
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

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

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

__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

Programmable
Standard Languages
Directives
CUDA

CPU Optimized
Directives
Vectorization

Multi-Platform
x86_64
Arm
OpenPOWER
Scientists Need On-Ramps

- Code that doesn’t run on NVIDIA platforms
- Standard Parallelism
- 10x Perf vs Host-Only Lane
- CUDA C++ & Fortran
- GPU Speed of Light Lane
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

<table>
<thead>
<tr>
<th>C++17</th>
<th>C++20</th>
<th>C++23 and Beyond</th>
</tr>
</thead>
</table>
| **Parallel Algorithms** | ➢ In NVC++  
➢ Parallel and vector concurrency | **Scalable Synchronization Library**  
➢ Express thread synchronization that is portable and scalable across CPUs and accelerators  
➢ In libc+++:  
➢ std::atomic<T>  
➢ std::barrier  
➢ std::counting_semaphore  
➢ std::atomic<T>::wait/notify_*  
➢ std::atomic_ref<T> | **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 |

**Forward Progress Guarantees**  
➢ Extend the C++ execution model for accelerators  
**Memory Model Clarifications**  
➢ Extend the C++ memory model for accelerators

Preview support coming to NVC++
C++17 PARALLEL ALGORITHMS
Lulesh Hydrodynamics Mini-app

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

codesign.llnl.gov/lulesh
static inline void CalcHydroConstraintForElems(Domain &domain, Index_t length, Index_t *regElemlist, Real_t dvovmax, Real_t &dthydro) {

#if _OPENMP
  const Index_t threads = omp_get_max_threads();
  Index_t dthydro_per_thread[threads];
  Real_t dthydro_per_thread[threads];
#else
  Index_t threads = 1;
  Index_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(0.0)) {
      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++ STANDARD PARALLELISM
Lulesh Performance

Same ISO C++ Code

- OpenMP on 64c EPYC 7742
- Standard C++ on 64c EPYC 7742
- Standard C++ on A100

NVC++
GCC
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 REFACTURING

M-AIA (ZFS) INTRODUCTION

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

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

<table>
<thead>
<tr>
<th></th>
<th>OpenMP (EPYC 7742)</th>
<th>Standard C++ (EPYC 7742)</th>
<th>Standard C++ (A100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Speed-up</td>
<td>1.0X</td>
<td>48.5X</td>
<td>225.0X</td>
</tr>
</tbody>
</table>

Same Standard C++ Code
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
GTC Fall Session A31329
# HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

<table>
<thead>
<tr>
<th>Fortran 2018</th>
<th>Fortran 202x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array Syntax and Intrinsics</strong></td>
<td><strong>DO CONCURRENT Reductions</strong></td>
</tr>
<tr>
<td>➢ NVFORTRAN 20.5</td>
<td>➢ NVFORTRAN 21.11</td>
</tr>
<tr>
<td>➢ Accelerated matmul, reshape, spread, ...</td>
<td>➢ REDUCE subclause added</td>
</tr>
<tr>
<td><strong>DO CONCURRENT</strong></td>
<td>➢ Support for +, *, MIN, MAX, IAND, IOR, IOR.</td>
</tr>
<tr>
<td>➢ NVFORTRAN 20.11</td>
<td>➢ Support for .AND., .OR., .EQV., .NEQV on LOGICAL values</td>
</tr>
<tr>
<td>➢ Auto-offload &amp; multi-core</td>
<td>➢ Atomics</td>
</tr>
</tbody>
</table>

**Co-Arrays**

- Coming Soon
- Accelerated co-array images

Preview support available now in NVFORTRAN
HPC PROGRAMMING IN ISO FORTRAN

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

```fortran
subroutine smooth( a, b, w0, w1, w2, n, m, niter )
  real, dimension(:,:) :: a,b
  real :: w0, w1, w2
  integer :: n, m, niter
  integer :: i, j, iter
  do iter = 1,niter
    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
```

Jacobi Performance

Same ISO Fortran Code
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)
```

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

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

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)
```

MATMUL FP64 matrix multiply
Inline FP64 matrix multiply

Naive Inline V100
FORTRAN V100 FORTRAN A100
HPC PROGRAMMING IN ISO FORTRAN
Examples of Patterns Accelerated in NVFORTRAN

\[
d = 2.5 \times \text{ceil}(\text{transpose}(a)) + 3.0 \times \text{abs}(\text{transpose}(b))
\]
\[
d = 2.5 \times \text{ceil}(\text{transpose}(a)) + 3.0 \times \text{abs}(b)
\]
\[
d = \text{reshape}(a, \text{shape}=\text{[ni,nj,nk]})
\]
\[
d = \text{reshape}(a, \text{shape}=\text{[ni,nk,nj]})
\]
\[
d = 2.5 \times \sqrt{\text{reshape}(a, \text{shape}=\text{[ni,nk,nj]}, \text{order}=\text{[1,3,2]})}
\]
\[
d = \text{alpha} \times \text{conjg}(\text{reshape}(a, \text{shape}=\text{[ni,nk,nj]}, \text{order}=\text{[1,3,2]})}
\]
\[
d = \text{reshape}(a, \text{shape}=\text{[nk,ni,nj]}, \text{order}=\text{[2,3,1]})
\]
\[
d = \text{reshape}(a, \text{shape}=\text{[ni*nj,nk]}]
\]
\[
d = \text{spread}(a, \text{dim}=3, \text{ncopies}=\text{nk})
\]
\[
d = \text{spread}(a, \text{dim}=1, \text{ncopies}=\text{ni})
\]
\[
d = \text{spread}(a, \text{dim}=2, \text{ncopies}=\text{nx})
\]
\[
d = \text{abs}(\text{spread}(a, \text{dim}=2, \text{ncopies}=\text{nx}))
\]
\[
d = \text{transpose}(a)
\]
\[
d = \text{alpha} \times \text{transpose}(a)
\]
\[
d = \text{alpha} \times \text{ceil}(\text{transpose}(a))
\]
\[
d = \text{alpha} \times \text{conjg}(\text{transpose}(a))
\]
\[
c = \text{c + matmul(a,b)}
\]
\[
c = \text{c - matmul(a,b)}
\]
\[
c = \text{c + alpha} \times \text{matmul(a,b)}
\]
\[
d = \text{alpha} \times \text{matmul(a,b)} + \text{c}
\]
\[
d = \text{alpha} \times \text{matmul(a,b)} + \beta \times \text{c}
\]
\[
c = \text{matmul}(\text{transpose}(a),\text{transpose}(b))
\]
\[
c = \text{matmul}(\text{transpose}(a),\text{reshape}(b, \text{shape}=\text{[k,n]}, \text{order}=\text{[2,1]})]
\]
\[
d = \text{spread}(a, \text{dim}=3, \text{ncopies}=\text{nk})
\]
\[
d = \text{spread}(a, \text{dim}=1, \text{ncopies}=\text{ni})
\]
\[
d = \text{spread}(a, \text{dim}=2, \text{ncopies}=\text{nx})
\]
\[
d = \text{abs}(\text{spread}(a, \text{dim}=2, \text{ncopies}=\text{nx}))
\]
\[
d = \text{transpose}(a)
\]
\[
d = \text{alpha} \times \text{transpose}(a)
\]
\[
d = \text{alpha} \times \text{ceil}(\text{transpose}(a))
\]
\[
d = \text{alpha} \times \text{conjg}(\text{transpose}(a))
\]
\[
c = \text{c + matmul(a,b)}
\]
\[
c = \text{c - matmul(a,b)}
\]
\[
c = \text{c + alpha} \times \text{matmul(a,b)}
\]
\[
d = \text{alpha} \times \text{matmul(a,b)} + \text{c}
\]
\[
d = \text{alpha} \times \text{matmul(a,b)} + \beta \times \text{c}
\]
ACCELERATED STANDARD LANGUAGES
Parallel performance for wherever your code runs

ISO C++

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

ISO Fortran

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

Python

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

CPU

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

GPU

- nvc++ -stdpar=gpu
- nvfortran -stdpar=gpu
- legate -gpus 1 saxpy.py
BRINGING GPU SUPERCOMPUTING TO PYDATA ECOSYSTEM

1 CPU Core  
2000

Multicore CPU  
2005

10s of Nodes  
2010

1000s of Nodes  
2015

GPU Accelerated With Native NumPy APIs  
2020

GPU Supercomputing with all native PyData APIs  
Future

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
PRODUCTIVE

Have Your Cake and Eat It Too

**Productivity**

```python
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

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

```python
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
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

Familiar Domain-Specific Interfaces
- cuNumeric
- cuDF
- SciPy
- Legate

Runtime System for Scalable Execution
- Legion

GPU-Accelerated CUDA-X Libraries
- cuBLAS, cuDF, NCCL, cuTENSOR, cuML, ...
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

```python
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

CUNUMERIC
Automatic NumPy Acceleration and Scalability
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 L2S & SMEM

Scaling Up
Multi-GPU and Multi-Node Libraries

Composability
Device Functions
NVIDIA MATH LIBRARIES
Linear Algebra, FFT, RNG and Basic Math

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

<table>
<thead>
<tr>
<th>Library and Tensor Core Functionality</th>
<th>INT4</th>
<th>INT8</th>
<th>FP16</th>
<th>BF16</th>
<th>TF32</th>
<th>FP64</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dense</td>
<td>Sparse</td>
<td>Dense</td>
<td>Sparse</td>
<td>Dense</td>
<td>Sparse</td>
</tr>
<tr>
<td>cuBLAS &amp; cuBLASLt Dense GEMM</td>
<td></td>
<td></td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cuTENSOR Tensor Contractions</td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cuSOLVER Linear System Solvers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>cuSPARSE Block-SpMM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>✅</td>
</tr>
<tr>
<td>cuSPARSELt SpMM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CUTLASS Dense GEMM and SpMM</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>CUTLASS Convolutions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 GEMM Performance

TFLOPS

<table>
<thead>
<tr>
<th></th>
<th>TFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GV100 - 10.2 - FP16 HMMA</td>
<td>118.3</td>
</tr>
<tr>
<td>GA100 - 11.0 - FP16 HMMA</td>
<td>286.1</td>
</tr>
<tr>
<td>GA100 - 11.0 - TF32 TF32MMA</td>
<td>110.7</td>
</tr>
</tbody>
</table>

TFLOPS

<table>
<thead>
<tr>
<th></th>
<th>TFLOPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GV100 - 10.2 - FP32 FMA</td>
<td>16.7</td>
</tr>
<tr>
<td>GA100 - 11.0 - FP32 FMA</td>
<td>18.7</td>
</tr>
<tr>
<td>GV100 - 10.2 - FP64 FMA</td>
<td>7.8</td>
</tr>
<tr>
<td>GA100 - 11.0 - DP64 DMMA</td>
<td>18.9</td>
</tr>
</tbody>
</table>
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

DGETRF on A100 and V100

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

cusparseSpSV (New) vs. cusparseXcsrsrv2

Speed-up on A100 GPU

261 SuiteSparse matrices sorted by increasing speed-up

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

![cuTENSORmGF Performance on DGXA100](image)
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

Performance: cuFFTMp vs. State-of-the-Art on Summit

- cuFFTMp
- State-of-the-Art

TFLOPs (Larger is better)

# of GPU & Problem Size

- 8 1024³
- 16 1280³
- 32 1600³
- 64 2048³
- 128 2560³
- 256 3200³
- 512 4096³

- 22
- 11
- 16
- 30
- 54
- 102
- 217

- 5
- 5
- 9
- 10
- 19
- 25
- 41

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

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
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 Primitives
- Utility Operators
  - Iterators
  - Thread/Block I/O
  - PTX Intrinsics
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();
LIBCU++: A GPU-ENABLED STL

<table>
<thead>
<tr>
<th>Host Compiler’s Standard Library</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>#include &lt;...&gt;</code></td>
</tr>
<tr>
<td><code>std::</code></td>
</tr>
<tr>
<td><code>#include &lt;cuda/std/...&gt;</code></td>
</tr>
<tr>
<td><code>cuda::std::</code></td>
</tr>
<tr>
<td><code>#include &lt;cuda/...&gt;</code></td>
</tr>
<tr>
<td><code>cuda::</code></td>
</tr>
</tbody>
</table>

**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>

**1.1.0 (CUDA 11.0)**

**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, ...>
...

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>
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 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

<table>
<thead>
<tr>
<th>Components</th>
<th>Description</th>
</tr>
</thead>
</table>
| MPI/SHMEM                   | • Open MPI and OpenSHMEM  
   • MPI profiler (IPM - open source tool from [http://ipm-hpc.org](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

180X reduction in allreduce latency (us)
DEVELOPER TOOLS

**Debuggers:** cuda-gdb, Nsight Visual Studio Edition

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

**Correctness Checker:** Compute Sanitizer

```bash
$ compute-sanitizer --leak-check full memcheck_demo
========== COMPUTE-SANITIZER
Mallocing memory
Running unaligned_kernel: no error
Sync: no error
Running 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 0x40010000 is misaligned
```

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

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

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!

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
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.

```bash
$ make &
$ make -j8
```

### Memcheck

```
$ make
```

- Configuring done
- Generating done
- Build files have been written to: /home/wayne/cuda_sanitizer/cuda build

```
$ make
```

- Confusing done
- Generating done
- Build files have been written to: /home/wayne/cuda_sanitizer/cuda build

```bash
$ make clean
$ make
```

- Configuring done
- Generating done
- Build files have been written to: /home/wayne/cuda_sanitizer/cuda build

```bash
$ make all
```

- Configuring done
- Generating done
- Build files have been written to: /home/wayne/cuda_sanitizer/cuda build

### Memcheck Output

```
[memcheck]
```

- Found 1 error and 4 warnings.
- Memory check project: /home/wayne/cuda/cuda_sanitizer/cuda build
- Start 1: verify
  1/1 MemCheck: #1 verify .......................... Passed: 6.77 sec
- 0 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: 0
- Memory check log files can be found here: (N corresponds to test number) /home/wayne/cuda/cuda_sanitizer/cuda build/Testing/Temporary/MemChecker/write.log
- Memory checking results:
  - invalid_global_read: 1
  - cudaErrorLaunchFailure: 3
  - Submit files
    - Submit files
      - Submit files
        - Submit files
```

---

**NVIDIA**
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