AGENDA

Accelerated Computing with Standard Languages
GPU Supercomputing in the PyData Ecosystem
Advancements in HPC Libraries
NVIDIA Developer Tools
# Programming the NVIDIA Platform

**CPU, GPU, and Network**

## Accelerated Standard Languages

ISO C++, ISO Fortran

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

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

```python
import cunumeric as np
```

```python
def saxpy(a, x, y):
y[:] += a*x
```

## Incremental Portable Optimization

OpenACC, OpenMP

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

```cpp
#pragma omp target data map(x,y) {
...
}
```

```python
@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];
}
```

```python
int main(void) {
...
cudaMemcpy(d_x, x, ...);
cudaMemcpy(d_y, y, ...);
saxpy<<<(N+255)/256,256>>>(...);
cudaMemcpy(y, d_y, ...);
}
```

## Platform Specialization

CUDA

## Acceleration Libraries

- Core
- Math
- Communication
- Data Analytics
- AI
- Quantum
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)
endo

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
## 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++23 Extended Floating Point Types
- C++23 Range Based Parallel Algorithms
- C++2X Executors
- C++2X Linear Algebra
- Fortran 202X DO CONCURRENT Reduction
HPC COMPILERS
NVC | NVC++ | NVFORTRAN

Accelerated
A100
Automatic

Programmable
Standard Languages
Directives
CUDA

CPU Optimized
Directives
Vectorization

Multi-Platform
x86_64
Arm
OpenPOWER
NVIDIA HPC SDK
Available at developer.nvidia.com/hpc-sdk, on NGC, via Spack, and in the Cloud

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
- HPC-X
- MPI
- UCX
- SHMEM
- SHARP
- HCOLL
- NVSHMEM
- NCCL

ANALYSIS

Profilers
- Nsight
- Systems

Debugger
- cuda-gdb
- Host
- Device

Develop for the NVIDIA Platform: GPU, CPU and Interconnect Libraries | Accelerated C++ and Fortran | Directives | CUDA x86_64 | Arm | OpenPOWER 7-8 Releases Per Year | Freely Available
PARALLEL PROGRAMMING WITH ISO C++
HPC PROGRAMMING IN ISO C++
ISO is the place for portable concurrency and parallelism

**C++17 & C++20**

- **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

- **Ranges**
  - Simplifies iterating over a range of values

- **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**

- **std::mdspan/mdarray**
  - HPC-oriented multi-dimensional array abstractions.

- **Range-Based Parallel Algorithms**
  - Improved multi-dimensional loops

- **Extended Floating Point Types**
  - First-class support for formats new and old: `std::float16_t/float64_t`

**And Beyond**

- **Executors / Senders-Recievers**
  - Simplify launching and managing parallel work across CPUs and accelerators

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

---

**Preview support coming to NVC++**

- Linear Algebra
- Scalable Synchronization Library
- Memory Model Clarifications
- Forward Progress Guarantees
- Parallel and vector concurrency
- Parallel Algorithms
- Ranges
- Scavable Synchronization Library
- Memory Model Clarifications
- Forward Progress Guarantees
- Parallel and vector concurrency
- Parallel Algorithms
- Ranges
- Scalable Synchronization Library
- Memory Model Clarifications
- Forward Progress Guarantees
- Parallel and vector concurrency
- Parallel Algorithms
- Ranges
- Scalable Synchronization Library
- Memory Model Clarifications
- Forward Progress Guarantees
- Parallel and vector concurrency
- Parallel Algorithms
- Ranges
PARALLELISM IN C++ ROADMAP

**C++ 11**
- Memory model enhancements
- Lambdas

**C++ 14**
- Atomics extensions
- Generic Lambda Expressions

**C++ 17**
- Parallel algorithms
- Forward progress guarantees
- Memory model clarifications

**C++ 20**
- Scalable synchronization library
- Ranges
- Span

**C++ PIPELINE**
- Mdspan
- Range-based parallel algorithms
- Extended floating-point types

- Linear algebra algorithms
- Asynchronous parallel algorithms
- Senders-receivers

---

How users run C++ code on GPUs today

Co-designed with V100 hardware support

N-dimensional loops and usability

Custom algorithms and async. control flow

Extended C++ interface to BLAS/Lapack

General usability of performance provided by executors

General parallelism user facing feature
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) 
{ 
#pragma omp parallel firstprivate(length, dvovmax) 
   { 
      Real_t dthydro_tmp = dthydro; 
      Index_t hydro_elem = -1; 
   } 
#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 dtvovv = dvovmax / (fabs(domain.vdov(indx)) + Real_t(1.e-20)); 
         if (dthydro_tmp > dtvov) { 
            dthydro_tmp = dtvov; 
            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]; 
      } 
   } 
   if (hydro_elem_per_thread[0] != -1) { 
      dthydro = dthydro_per_thread[0]; 
   } 
   return; 
}
C++ STANDARD PARALLELISM

Lulesh Performance

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Lulesh Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>g++</td>
<td>1.19X</td>
</tr>
<tr>
<td>icpc</td>
<td>1.00X</td>
</tr>
<tr>
<td>nvc++ (CPU)</td>
<td>1.98X</td>
</tr>
<tr>
<td>nvc++ (GPU)</td>
<td>14.75X</td>
</tr>
<tr>
<td>ISO C++</td>
<td>1.91X</td>
</tr>
</tbody>
</table>

Same ISO C++ Code

AMD EPYC 7742 CPU, NVIDIA A100 GPU. g++ version 10.3.0, icpc version 2021.5.0, nvc++ version 22.3
C++ with OpenMP

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

```cpp
#pragma omp parallel // OpenMP parallel region
{
#pragma omp for // OpenMP for loop
for (MInt i = 0; i < noCells; i++) { // Loop over all cells
  if (timeStep % ipow[maxLevel_ - clevel[i] * distLevel]) == 0) { // Multi-grid loop
    const MInt distStartId = i * nDist; // More offsets for 1D accesses // Local offsets
    const MInt distStartId = i * distNeighbors;
    const MFloat* const distributionsStart = &distributions[distStartId];
    for (MInt j = 0; j < nDist - 1; j += 2) { // Unrolled loop distributions (factor 2)
      if (neighborId[I * distNeighbors + j] > -1) { // First unrolled iteration
        const MInt n1StartId = neighborId[distStartId + j] * nDist;
        oldDistributions[n1StartId + j] = distributionsStart[j]; // 1D access AoS format
      }
      if (neighborId[I * distNeighbors + j + 1] > -1) { // Second unrolled iteration
        const MInt n2StartId = neighborId[distStartId + j + 1] * nDist;
        oldDistributions[n2StartId + j + 1] = distributionsStart[j + 1];
      }
    }
    oldDistributions[distStartId + lastId] = distributionsStart[lastId]; // Zero-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
  }
});
```
<table>
<thead>
<tr>
<th>Fortran 2018</th>
<th>Fortran 202x</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fortran Array 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, IEOR.</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></td>
</tr>
<tr>
<td><strong>Co-Arrays</strong></td>
<td></td>
</tr>
<tr>
<td>➢ Not currently available</td>
<td></td>
</tr>
<tr>
<td>➢ Accelerated co-array images</td>
<td></td>
</tr>
</tbody>
</table>

ISO is the place for portable concurrency and parallelism.

Preview support available now in NVFORTRAN
MiniWeather

Mini-App written in C++ and Fortran that simulates weather-like fluid flows using Finite Volume and Runge-Kutta methods.

Existing parallelization in MPI, OpenMP, OpenACC, ...

Included in the SPEChpc benchmark suite*

Open-source and commonly-used in training events.

https://github.com/mrnorman/miniWeather/

```
do concurrent (i=1:NUM_VARS, k=1:nz, l=1:nx)
  local(x,z,x0,z0,xrad,zrad,amp,dist,wpert)
  if (data_spec_int == DATA_SPEC_GRAVITY_WAVES) then
    x = (i_beg-1 + i*0.5_rp) * dx
    z = (k_beg-1 + k*0.5_rp) * dz
    x0 = xlen/8
    z0 = 1000
    xrad = 500
    zrad = 500
    amp = 0.01_rp
    dist = sqrt((x-x0)/xrad)**2 + ((z-z0)/zrad)**2
    * pi / 2_rp
    if (dist <= pi / 2_rp) then
      wpert = amp * cos(dist)**2
    else
      wpert = 0_rp
    endif
    tend(i,k,ID_WMOM) = tend(i,k,ID_WMOM) + wpert*hy_dens_cell(k)
  endif
  state_out(i,k,l) = state_init(i,k,l)
  + dt * tend(i,k,l)
enddo
```


OpenACC version uses -gpu=managed option.

*SPEChpc is a trademark of The Standard Performance Evaluation Corporation
POT3D: DO CONCURRENT + LIMITED OPENACC

POT3D is a Fortran application for approximating solar coronal magnetic fields.

Included in the SPEChpc benchmark suite*

Existing parallelization in MPI & OpenACC

Optimized the DO CONCURRENT version by using OpenACC solely for data motion and atomics

https://github.com/predsci/POT3D

One A100 GPU
(lower is better) σ (over 4 runs)

<table>
<thead>
<tr>
<th>Wall Clock Time (seconds)</th>
<th>Original (no managed)</th>
<th>STDPAR + Min ACC (managed)</th>
<th>Original (managed)</th>
<th>STDPAR + ACC (no managed)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1481.3</td>
<td></td>
<td>1644.5</td>
<td>1634.3</td>
<td>1486.2</td>
</tr>
</tbody>
</table>

!$acc enter data copyin(phi,dr_i)
!$acc enter data create(br)
do concurrent (k=1:np,j=1:nt,i=1:nrm1)
br(i,j,k)=(phi(i+1,j,k)-phi(i,j,k ))*dr_i(i)
enddo
!$acc exit data delete(phi,dr_i,br)

Data courtesy of Predictive Science Inc.

*SPEChpc is a trademark of The Standard Performance Evaluation Corporation
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
...

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

do nt = 1, ntimes
  d = c + matmul(a,b)
end do

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
  d = c + matmul(a,b)
end do

real(8), dimension(ni,nk) :: a
real(8), dimension(nk,nj) :: b
real(8), dimension(ni,nj) :: c
...
HPC PROGRAMMING IN ISO FORTRAN
Examples of Patterns Accelerated in NVFORTRAN

\[
d = 2.5 \times \text{ceil}(\text{transpose}(a)) + 3.0 \times \text{abs}(\text{transpose}(b))
\]
\[
d = 2.5 \times \text{ceil}(\text{transpose}(a)) + 3.0 \times \text{abs}(b)
\]
\[
d = \text{reshape}(a, \text{shape}=[ni,nj,nk])
\]
\[
d = \text{reshape}(a, \text{shape}=[ni,nk,nj])
\]
\[
d = 2.5 \times \text{sqrt}(\text{reshape}(a, \text{shape}=[ni,nk,nj], \text{order}=[1,3,2]))
\]
\[
d = \alpha \times \text{conjg}(\text{reshape}(a, \text{shape}=[ni,nk,nj], \text{order}=[1,3,2]))
\]
\[
d = \text{reshape}(a, \text{shape}=[ni, nk, nj], \text{order}=[1,3,2])
\]
\[
d = \text{reshape}(a, \text{shape}=[nk, ni, nj], \text{order}=[2,3,1])
\]
\[
d = \text{reshape}(a, \text{shape}=[ni*nj, nk])
\]
\[
d = \text{reshape}(a, \text{shape}=[nk, ni*nj], \text{order}=[2,1])
\]
\[
d = \text{reshape}(a, \text{shape}=[64,2,16,16,64], \text{order}=[5,2,3,4,1])
\]
\[
d = \text{abs}(\text{reshape}(a, \text{shape}=[64,2,16,16,64], \text{order}=[5,2,3,4,1]))
\]
\[
c = \text{matmul}(a,b)
\]
\[
c = \text{matmul}(\text{transpose}(a),b)
\]
\[
c = \text{matmul}(\text{reshape}(a, \text{shape}=[m,k], \text{order}=[2,1]),b)
\]
\[
c = \text{matmul}(a,\text{transpose}(b))
\]
\[
c = \text{matmul}(a,\text{reshape}(b, \text{shape}=[k,n], \text{order}=[2,1]))
\]
\[
c = \text{matmul}(\text{transpose}(a),\text{transpose}(b))
\]
\[
c = \text{matmul}(\text{transpose}(a),\text{reshape}(b, \text{shape}=[k,n], \text{order}=[2,1]))
\]
\[
c = \text{matmul}(a,b)
\]
\[
c = \alpha \times \text{matmul}(a,b) + c
\]
\[
c = \alpha \times \text{matmul}(a,b) + \beta \times c
\]
AGENDA

Accelerated Computing with Standard Languages

GPU Supercomputing in the PyData Ecosystem

Advancements in HPC Libraries

NVIDIA Developer Tools
import numpy as np
da = np.random.randn(16).reshape(4, 4)
b = a + a.T

import dask.array as da
import numpy as np
da = 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

da = 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
da = np.random.randn(160_000).reshape(400, 400)
b = a + a.T
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
COMPUTATIONAL FLUID DYNAMICS

- CFD codes like:
  - Shallow-Water Equation Solver
- Oil Pipeline Risk Management: Geoclaw-landspill simulations
- Python Libraries: Jupyter, NumPy, SciPy, SymPy, Matplotlib

```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](https://github.com/barbagroup/CFDPython)
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
MICRO-JOIN

size = num_rows_per_gpu * num_gpus
key_l = np.arange(size)
val_l = np.random.randn(size)
lhs = pd.DataFrame({ "key": key_l, "val": val_l })
key_r = key_l // 3 * 3    # selectivity: 0.33
payload_r = np.random.randn(size)
rhs = pd.DataFrame({ "key": key_r, "val": val_r })
out = lhs.merge(rhs, on="key")
AGENDA

Accelerated Computing with Standard Languages
GPU Supercomputing in the PyData Ecosystem
Advancements in HPC Libraries
NVIDIA Developer Tools
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

Recently Introduced
- Improved heuristics (cache)
- Improved FP64 SYRK, TRMM, SYMM
- Batched GEMV Extensions
- Helper functions for improved error management

Maximum Speedups OF CTK 11.6u1 over CTK 11.1: Sizes < 2k

* A100 80GB @ 1095 MHz: CTK 11.1 vs. CTK 11.6U1
cuSOLVER
GPU Optimized Factorizations & Solvers

Dense and Sparse Factorizations & Solvers
- LU, Cholesky, QR
- Symmetric and Generalized Eigensolvers
- Tensor Core Accelerated Iterative Refinement Solvers
- Multi GPU & Multi-node Support

Accelerating a wide range of applications
- HPC & Scientific Computing
- Data Analytics

Recently Improvements
- Improvements for small (D/Z)SYGVD/SYEVD
- Multi-node Multi-GPU (LU w/ & w/o pivoting)

Speedups for latest cuSOLVER versus 11.0

* A100 80GB Default clocks: CTK 11.0 vs. CTK 11.6
GPU Optimized FFT

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

Accelerating a wide range of applications

- HPC & Scientific Computing
- Data Analytics

Recent Improvements

- Optimizations for large 3D FFT
- Uniform performance improvement for size < 32k
- Performance improvements for all sizes (up to 10x)

---

Speedups (over 10%) for latest cuFFT versus CTK 11.0

* A100 80GB Default clocks: CTK 11.0 vs. CTK 11.7U1
cuSOLVER
GPU Optimized Factorizations & Solvers

Recent Improvements
- First Released in HPC SDK 21.11
- LU Decomposition
- Cholesky

LU Decomposition (GETRF+GETRS) w/ Pivoting on Summit

* Summit: 6x V100 16GB per node
Recent Improvements

- Released in HPC SDK 22.3
- Distributed 2D/3D FFTs
- Slab Decomposition
- Pencil Decomposition (Preview)
- Helper functions: Pencils <-> Slabs

**cuFFTMp**

Distributed 2D/3D FFTs at Scale

**Distributed 3D FFT Performance: Comparison by Priceison**

C2C | Z2Z
---|---
61 | 851
29 | 410
109 | 429
52 | 210
226 | 105
512 | 1024
8192 | 8192
12288 | 12288
2048 | 2048
4096 | 4096
12288 | 12288
2048 | 2048
4096 | 4096
128 | 128
5120 | 16384
256 | 6144
6144 | 8192
512 | 1024
1024 | 2048
1024 | 2048
12288 | 12288
2048 | 2048
4096 | 4096

* Selene: A100 80GB @ 1410 MHz
**MATH LIBRARIES DEVICE EXTENSIONS**

Enabling kernel fusion of high-performance numerical methods

**cuFFTDx: In MathDx**
- Retain and reuse on-chip data
- Inline FFTs in user kernel up to 32k (A100)
- Combine FFT operations

---

**Convolutions Performance**

FFT Sizes (1D)

- cuFFTDx
- cuFFT
- cuFFT (callbacks)

* A100 80GB @ 1410 MHz

---

**Convolution Kernel Fusion**

- FFT
- Custom Op
- FFT

AGENDA

Accelerated Computing with Standard Languages

GPU Supercomputing in the PyData Ecosystem

Advancements in HPC Libraries

NVIDIA Developer Tools
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
```

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

```plaintext
(cuda-gdb) info cuda threads breakpoint all
   BlockIdx  ThreadIdx          Virtual PC  Dev  SM  Wp  Ln  Filename   Line
                (0, 0, 0) 0x00000000000948e58  0  11  0  0  infoCommands.cu  12
(1, 0, 0) (0, 0, 0) 0x00000000000948e58  0  11  0  0  infoCommands.cu  12
(1, 0, 0) (1, 0, 0) 0x00000000000948e58  0  11  0  1  infoCommands.cu  12
(1, 0, 0) (2, 0, 0) 0x00000000000948e58  0  11  0  2  infoCommands.cu  12
(1, 0, 0) (3, 0, 0) 0x00000000000948e58  0  11  0  3  infoCommands.cu  12
(1, 0, 0) (4, 0, 0) 0x00000000000948e58  0  11  0  4  infoCommands.cu  12
(1, 0, 0) (5, 0, 0) 0x00000000000948e58  0  11  0  5  infoCommands.cu  12

(cuda-gdb) info cuda threads breakpoint 2 lane 1
   BlockIdx  ThreadIdx          Virtual PC  Dev  SM  Wp  Ln  Filename   Line
                (1, 0, 0) 0x00000000000948e58  0  11  0  1  infoCommands.cu  12
(1, 0, 0) (1, 0, 0) 0x00000000000948e58  0  11  0  1  infoCommands.cu  12
```

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

DEVELOPMENT

Programming Models
- Standard C++ & Fortran
- OpenACC & OpenMP
- CUDA

Compilers
- nvcc
- nvc
- nvc++
- nvfortran

Core Libraries
- libc++
- Thrust
- CUB

Math Libraries
- cuBLAS
- cuTENSOR
- cuSPARSE
- cuSOLVER
- cuFFT
- cuRAND

Communication Libraries
- HPC-X
- MPI
- UCX
- SHMEM
- SHARP
- HCOLL
- NVSHMEM
- NCCL

ANALYSIS

Profilers
- Nsight

Debugger
- cuda-gdb

Develop for the NVIDIA Platform: GPU, CPU and Interconnect Libraries | Accelerated C++ and Fortran | Directives | CUDA
x86_64 | Arm | OpenPOWER
7-8 Releases Per Year | Freely Available
GTC SPRING 2022 SESSIONS TO REWATCH

For more information on these topics

- No More Porting: Coding for GPUs with Standard C++, Fortran, and Python [S41496]
- A Deep Dive into the Latest HPC Software [S41494]
- C++ Standard Parallelism [S41960]
- Future of Standard and CUDA C++ [S41961]
- Shifting through the Gears of GPU Programming: Understanding Performance and Portability Trade-offs [S41620]
- From Directives to DO CONCURRENT: A Case Study in Standard Parallelism [S41318]
- Evaluating Your Options for Accelerated Numerical Computing in Pure Python [S41645]
- How to Develop Performance Portable Codes using the Latest Parallel Programming Standards [S41618]