Agenda

- Programming the NVIDIA Platform
- NVIDIA HPC SDK
- Standards-Based Parallel Programming
- Conclusions
NVIDIA HPC SOFTWARE
Major Initiatives

Seamless Acceleration
Standard Languages, Tensor Cores, GH C2C

Scaling Up
Multi-GPU and Multi-Node Libraries

Domain Libraries
Quantum, LQCD, Signal Processing

Arm Software
Compilers, Libraries, Ecosystem
Programming the NVIDIA Platform
Programming the NVIDIA Platform
CPU, GPU, and Network

ACCELERATED STANDARD LANGUAGES
ISO C++, ISO Fortran

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

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

import cunumeric as np

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

INCREMENTAL PORTABLE OPTIMIZATION
OpenACC, OpenMP

#pragma acc data copy(x,y) {
   ...
   std::transform(par, x, x+n, y, y,
      [=](float x, float y) {
         return y + a*x;
      });
   ...
}

#pragma omp target data map(x,y) {
   ...
   std::transform(par, x, x+n, y, y,
      [=](float x, float y) {
         return y + a*x;
      });
   ...
}

PLATFORM SPECIALIZATION
CUDA

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

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

ACCELERATION LIBRARIES
Core Math Communication Data Analytics AI Quantum
### Accelerated Standard Languages

Parallel performance for wherever your code runs

<table>
<thead>
<tr>
<th>ISO C++</th>
<th>ISO Fortran</th>
<th>Python</th>
</tr>
</thead>
</table>
| 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 |

CPU

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

GPU

- nvc++ -stdpar=gpu
- nvfortran -stdpar=gpu
- legate -gpus 1 saxpy.py
NVIDIA HPC SDK
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
x86_64 | AArch64 | OpenPOWER
7-8 Releases Per Year | Freely Available
HPC Compilers

NVC | NVC++ | NVFORTRAN

Accelerated
- GPU
- Automatic

Programmable
- Standard Languages
- Directives
- CUDA

CPU Optimized
- Directives
- Vectorization

Multi-Platform
- x86_64
- AArch64
- OpenPOWER
NVIDIA PERFORMANCE LIBRARIES
Core and Math Library Directions

Seamless Acceleration
Tensor Cores, GH C2C

Scaling Up
Multi-GPU and Multi-Node Libraries

Composability
Device Functions

Arm Execution
High Performance CPU Libraries
NVIDIA Math Libraries

- cuBLAS
- cuSPARSE
- cuTENSOR
- cuSOLVER
- CUTLASS
- AMGX
- cuRAND
- cuFFT
- Math API

 cuFFT
 cuSPARSE
 cuTENSOR
 cuSOLVER
 CUTLASS
 AMGX
 cuRAND
 cuFFT
 Math API
Recent Improvements

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

- Released in HPC SDK 22.3
- Distributed 2D/3D FFTs
- Slab Decomposition
- Pencil Decomposition (Preview)
- Helper functions: Pencils <-> Slabs
Standards-based Parallel Programming
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.
  - Preview Implementation In Progress!
- **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- Receivers**
  - Simplify launching and managing parallel work across CPUs and accelerators
  - Preview Implementation In Progress!
- **Linear Algebra**
  - C++ standard algorithms API to linear algebra
  - Maps to vendor optimized BLAS libraries
  - Preview Implementation In Progress!
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];
    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(0.0)) {
                #pragma omp atomic
                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;
        }
    }
    if (hydro_elem_per_thread[0] != -1) {
        dthydro = dthydro_per_thread[0];
    } else {
        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));
                }
            });
    }
}
### C++ Standard Parallelism

#### Lulesh Speed-up

<table>
<thead>
<tr>
<th>Compiler</th>
<th>OpenMP (default)</th>
<th>ISO C++</th>
<th>NVC++ (H100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>g++</td>
<td>1.19X</td>
<td>1.98X</td>
<td>19.15X</td>
</tr>
<tr>
<td>icpc</td>
<td>1.00X</td>
<td>1.91X</td>
<td></td>
</tr>
<tr>
<td>nvc++</td>
<td>1.00X</td>
<td>2.71X</td>
<td></td>
</tr>
</tbody>
</table>

**AMD EPYC 7742 CPU, NVIDIA H100 GPU.** g++ version 10.3.0, icpc version 2021.5.0, nvc++ version 22.x
M-AIA

Multi-physics simulation framework developed at the Institute of Aerodynamics, RWTH Aachen University

- Hierarchical grids, complex moving geometries
- Adaptive meshing, load balancing
- Numerical methods: FV, DG, LBM, FEM, Level-Set, ...
- Physics: aeroacoustics, combustion, biomedical, ...
- Developed by ~20 PhDs (Mech. Eng.), ~500k LOC+
- **Programming model:** MPI + ISO C++ parallelism

![Graph showing relative speed-up and strong efficiency](image-url)
# HPC PROGRAMMING IN ISO FORTRAN

ISO is the place for portable concurrency and parallelism

### Fortran 2018

**Fortran Array Intrinsics**
- NVFORTRAN 20.5
- Accelerated matmul, reshape, spread, ...

**DO CONCURRENT**
- NVFORTRAN 20.11
- Auto-offload & multi-core

**Co-Arrays**
- Not currently available
- Accelerated co-array images

### Fortran 202x

**DO CONCURRENT Reductions**
- NVFORTRAN 21.11
- REDUCE subclause added
- Support for +, *, MIN, MAX, IAND, IOR, IEOR.
- Support for .AND., .OR., .EQV., .NEQV on LOGICAL values

Preview support available now in NVFORTRAN
MiniWeather
Standard Language Parallelism in Climate/Weather Applications

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 (l=1:NUM_VARS, k=1:nz, i=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_r) * dx
        z = (k_beg - 1 + k*0.5_r) * dz
        x0 = xlen/8
        z0 = 1000
        xrad = 500
        zrad = 500
        amp = 0.01_r
        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,ll) = state_init(i,k,ll) + dt * tend(i,k,ll)
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

Data courtesy of Predictive Science Inc.

*SPEChpc is a trademark of The Standard Performance Evaluation Corporation
• GAMESS is a popular Quantum Chemistry application.
• More than 40 years of development in Fortran and C
• MPI + OpenMP baseline code
• Hartree-Fock rewritten in Do Concurrent

Fock Build

<table>
<thead>
<tr>
<th>Speed-Up</th>
<th>0.0X</th>
<th>0.5X</th>
<th>1.0X</th>
<th>1.5X</th>
<th>2.0X</th>
<th>2.5X</th>
<th>3.0X</th>
<th>3.5X</th>
<th>4.0X</th>
<th>4.5X</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenMP (CPU)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenMP (GPU)</td>
<td>1.0X</td>
<td>1.3X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Do Concurrent (GPU)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3.9X</td>
</tr>
</tbody>
</table>

nvfortran 22.7, NVIDIA A100 GPU, AMD “Milan” CPU

* Courtesy of Melisa Alkan, Iowa State University. Not yet published.
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)

Utilizes A100 Tensor Cores automatically

MATMUL FP64 matrix multiply

**Figure:** 
- **Naive Inline V100:** 0 TFLOPs
- **FORTRAN V100:** 2 TFLOPs
- **FORTRAN A100:** 20 TFLOPs
Conclusions

• NVIDIA HPC SDK is a complete and portable toolkit for HPC developers
• NVIDIA supports a wide range of programming models to enable you to choose the right balance of productivity, portability, and performance for your project
• For more information on these topics, see the following on-demand GTC sessions:
  • A Deep Dive into the Latest HPC Software
  • CUDA: New Features and Beyond
  • How CUDA Programming Works
  • Developing HPC Applications with Standard C++, Fortran, and Python