GPU ACCELERATION PRINCIPLE AND PROGRAMING - HPC SDK

Ryan Jeng | Tech Lead, DevTech APAC
GPU PROGRAMMING IN 2020 AND BEYOND

Math Libraries | Standard Languages | Directives | CUDA

Incremental Performance Optimization with Directives

Maximize GPU Performance with CUDA C++/Fortran

GPU Accelerated Libraries

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;
    });

  ...
}

__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, ...);
  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
```
PROGRAMMING GPU-ACCELERATED HPC SYSTEMS

GPU | CPU | Interconnect

GPU

Node

System
AVAILABLE NOW: THE NVIDIA HPC SDK
Available at developer.nvidia.com/hpc-sdk, on NGC, and in the Cloud

NVIDIA HPC SDK

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 & STANDARD LANGUAGE PARALLELISM
HPC COMPILERS

NVC | NVC++ | NVFORTRAN

Accelerated
A100
Automatic

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++
  - Parallel and vector concurrency

- **Forward Progress Guarantees**
  - Extend the C++ execution model for accelerators

- **Memory Model Clarifications**
  - Extend the C++ memory model for accelerators

**C++20**

- **Scalable Synchronization Library**
  - Express thread synchronization that is portable and scalable across CPUs and accelerators
  - In libcu++:
    - `std::atomic<T>`
    - `std::barrier`
    - `std::counting_semaphore`
    - `std::atomic<T>::wait/notify_*`
    - `std::atomic_ref<T>`

**C++23 and Beyond**

- **Executors**
  - 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`
HPC PROGRAMMING IN ISO C++

C++ Parallel Algorithms

- Introduced in C++17
- Parallel and vector concurrency via execution policies
  - `std::execution::par`, `std::execution::par_seq`, `std::execution::seq`
- Several new algorithms in C++17 including
  - `std::for_each_n(POLICY, first, size, func)`
- Insert `std::execution::par` as first parameter when calling algorithms
- NVC++ 20.5: automatic GPU acceleration of C++17 parallel algorithms
  - Leverages CUDA Unified Memory

```cpp
std::sort(std::execution::par, c.begin(), c.end());
std::unique(std::execution::par, c.begin(), c.end());
```
```cpp
static inline void CalcHydroConstraintForElems(Domain &domain, Index_t length,
                                           Index_t *regElemlist, Real_t dvovmax, Real_t &dthydro)
{

  #if _OPENMP
  const Index_t threads = omp_get_max_threads();
  Index_t hydro_elem_per_thread[threads];
  Real_t dthydro_per_thread[threads];
  #else
  Index_t threads = 1;
  Index_t hydro_elem_per_thread[1];
  Real_t dthydro_per_thread[1];
  #endif

  #pragma omp parallel firstprivate(length, dvovmax)
  {
    Real_t dthydro_tmp = dthydro;
    Index_t hydro_elem = -1;
    #if _OPENMP
    Index_t thread_num = omp_get_thread_num();
    #else
    Index_t thread_num = 0;
    #endif

    #pragma omp for
    for (Index_t i = 0; i < length; ++i) {
      Index_t indx = regElemlist[i];
      if (domain.vdov(indx) == Real_t(0.0)) {
        dthydro_tmp = dvovmax / (fabs(domain.vdov(indx)) + Real_t(1.e-20));
      }
      else {
        dthydro_tmp = dvovmax / (fabs(domain.vdov(indx)) + Real_t(1.e-20));
      }
      dthydro_per_thread[thread_num] = dthydro_tmp;
    }
  }

  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++17**

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

Speedup - Higher is Better

- C++ on 2s 20c Xeon Gold 6148
- C++ on A100
- OpenACC on A100

Same ISO C++ Code
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, IOR.
- Support for .AND., .OR., .EQV., .NEQV on LOGICAL values
- Atomics
HPC PROGRAMMING IN ISO FORTRAN

NVFORTRAN Accelerates Fortran Intrinsics with cuTENSOR Backend

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)

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 exit kernels
end do

!$acc exit data copyout(d)
```

MATMUL 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)
```
HPC LIBRARIES
A100 FEATURES IN MATH LIBRARIES
Automatic Acceleration of Critical Routines in HPC and AI

- cuBLAS: BF16, TF32 and FP64 Tensor Cores
- cuSPARSE: Increased memory BW, Shared Memory and L2
- cuTENSOR: BF16, TF32 and FP64 Tensor Cores
- cuSOLVER: BF16, TF32 and FP64 Tensor Cores
- cuFFT: Increased memory BW, Shared Memory and L2
- CUDA Math API: BF16 Support
A100 TENSOR CORES IN LIBRARIES

cuBLAS

- Automatic Tensor Core acceleration
- Removed matrix size restrictions for Tensor Core acceleration

DGEMM on A100
- Up to 19 TFLOPs, 2.4x V100

FP64 Matrix Multiply: A100 vs V100

TFLOPS vs Matrix Size (m=n=k)
A100 TENSOR CORES IN LIBRARIES

cuBLAS

NVIDIA V100 FP32

NVIDIA A100 Tensor Core TF32

TF32 MMA Dimensions: m,n,k = 16x8x8

NVIDIA V100 FP64

NVIDIA A100 Tensor Core FP64

DMMA Dimensions: m,n,k = 8x8x4

10X

2.5X
cuSPARSELt

Extension Library with Sparse Matmul

- High-performance library for general matrix-matrix operations in which at least one operand is a sparse matrix
- Ampere Sparse MMA tensor core support
- Mixed-precision support
- Matrix pruning and compression functionalities
- Auto-tuning functionality

Dense trained weights

Fine-grained structured pruning (2:4 non-zero)

Fine-tuning weights

Compress

Zero

Non-zero data

Non-zero indices

Output activations

Sparse Tensor Core

Select

Mux

Mux

Dot-product

Input activations
<table>
<thead>
<tr>
<th>INPUT OPERANDS</th>
<th>ACCUMULATOR</th>
<th>TOPS</th>
<th>X-factor vs. FFMA</th>
<th>SPARSE TOPS</th>
<th>SPARSE X-factor vs. FFMA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>V100</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>15.7</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>125</td>
<td>8x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>A100</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP32</td>
<td>FP32</td>
<td>19.5</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TF32</td>
<td>FP32</td>
<td>156</td>
<td>8x</td>
<td>312</td>
<td>16x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>BF16</td>
<td>FP32</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>FP16</td>
<td>FP16</td>
<td>312</td>
<td>16x</td>
<td>624</td>
<td>32x</td>
</tr>
<tr>
<td>INT8</td>
<td>INT32</td>
<td>624</td>
<td>32x</td>
<td>1248</td>
<td>64x</td>
</tr>
<tr>
<td>INT4</td>
<td>INT32</td>
<td>1248</td>
<td>64x</td>
<td>2496</td>
<td>128x</td>
</tr>
<tr>
<td>BINARY</td>
<td>INT32</td>
<td>4992</td>
<td>256x</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IEEE FP64</td>
<td></td>
<td>19.5</td>
<td>1x</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
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

cuFFTDx Device API V100 Performance
Small-size FFTs

- cuFFT Host API
- cuFFTDx
TENSOR CORE ACCELERATED LINEAR SOLVERS

Mixed Precision Dense Linear Solvers

- Common HPC Solvers dominated by matrix multiplication
  - LU, QR
- Can we accelerate with FP16 Tensor Core and retain FP64 accuracy? Yes!
TENSOR CORE ACCELERATED LINEAR SOLVERS

Mixed Precision Dense Linear Solvers

- LU & QR Solvers available in cuSOLVER
- FP64 input, FP64 output, black-box mixed precision acceleration
- Real and Complex, single and multi-RHS

ZGETRF Performance

Matrix Size

0 5000 10000 15000 20000 25000 30000 35000 40000 45000

0 5 10 15 20 25 30 35 40 45

FP64 dgesv
FP32->FP64 dsgesv
FP16-TC->FP64 dhgesv
BF16-TC->FP64 dbgesv
TF32-TC->FP64 dxgesv
MULTI GPU SUPPORT IN LIBRARIES

Linear Algebra and FFT

**cuFFT**
- Single Process Multi-GPU FFT
- **Multi Node Multi-GPU FFT Coming Soon**

**cuSOLVER**
- Single Process Multi-GPU Eigensolver
- Single Process Multi-GPU LU
- Single Process Multi-GPU Cholesky
- **Multi Node Multi-GPU LU Coming Soon**

**cuBLAS**
- Improved Single Process Multi-GPU GEMM

---

**Multi GPU cuFFT Performance, 8xV100 vs 8xA100**

- **GFLOPs**
- **512^3**
- **1024^3**
- **2048^3**

---

NVIDIA
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
COMMUNICATION LIBRARIES

Single GPU, Multi GPU, and Multi Node

Open MPI + UCX

NVSHMEM

NCCL
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

➢ Up to 1.7X Single Node Speedup

QUDA: Quantum Chromodynamics on CUDA

➢ Up to 1.4X Multi Node Speedup
AVAILABLE NOW: THE NVIDIA HPC SDK

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

NVIDIA HPC SDK

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
- Open MPI
- NVSHMEM
- NCCL

ANALYSIS

Profilers
- Nsight
- Systems
- Compute

Debugger
- cuda-gdb
- Host
- Device

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
加入 NVIDIA 開發者計畫

請掃描左邊 QR code 加入 NVIDIA 開發者計畫，活動結束前完成註冊，即可換取精美小禮物。

請向 NVIDIA 工作人員出示主旨為「Your application for the program NVIDIA Developer Program is approved」的email。

developer.nvidia.com/developer-program