AGENDA

• Tensor Cores Architecture

• Programming Approaches
  • DL Framework
  • Libraries
  • WMMA CUDA
  • CUTLASS

• HPC Case Studies
  • Particle in Cell
  • Spherical Harmonics - IFS
VOLTA ARCHITECTURE AND TENSOR CORES
TENSOR CORE
Mixed Precision Matrix Math
4x4 matrices

\[ D = \begin{pmatrix}
A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\
A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\
A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\
A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3}
\end{pmatrix} \begin{pmatrix}
B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\
B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\
B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\
B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3}
\end{pmatrix} + \begin{pmatrix}
C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\
C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\
C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\
C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3}
\end{pmatrix}

D = AB + C
VOLTA TENSOR OPERATION

FP16 storage/input  Full precision product  Sum with FP32 accumulator  Convert to FP32 result

Also supports FP16 accumulator mode for inferencing
TENSOR CORE
PROGRAMMING
MODELS
AUTOMATIC MIXED PRECISION

Insert ~ two lines of code to introduce Automatic Mixed-Precision and get upto 3X speedup

AMP uses a graph optimization technique to determine FP16 and FP32 operations

Support for TensorFlow, PyTorch and MXNet

export TF_ENABLE_AUTO_MIXED_PRECISION=1

TensorFlow
Upto 3X Speedup
CUBLAS TENSOR CORE HOW-TO

<table>
<thead>
<tr>
<th>CUBLAS functions</th>
<th>mathMode = CUBLAS_DEFAULT_MATH</th>
<th>mathMode = CUBLAS_TENSOR_OP_MATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>cublasHgemm, cublasSgemm,</td>
<td>Disallowed</td>
<td>Allowed</td>
</tr>
<tr>
<td>cublasGemmEx(algo=DEFAULT)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cublasGemmEx(algo=*_TENSOR_OP</td>
<td>Allowed</td>
<td>Allowed</td>
</tr>
</tbody>
</table>


Volta and Turing family Tensor Core can be used with in mixed precision (FP16 inputs, FP32 accumulation, FP16 or FP32 output) routines.

Pure single precision routines use tensor core (when allowed) by down-convertting inputs to half (FP16) precision on the fly.

Constraint: M,N,K,LDA,LDB,LDC and A,B,C pointers must ALL be aligned to 8 because of high memory bandwidth needed to efficiently use Tensor Cores.
HGEMM VS GEMMEX

cublasSetMathMode(handle, CUBLAS_TENSOR_OP_MATH);
const __half *A = ...;
const __half *B = ...;
__half *C = ...;
cublasHgemm(handle, transa, transb, m, n, k,
  alpha, A, lda, B, ldb,
  beta, C, ldc);

vs.

... float *C = ...;
cublasGemmEx(handle, transa, transb, m, n, k,
  alpha, A, CUDA_R_16F, lda, B, CUDA_R_16F, ldb,
  beta, C, CUDA_R_32F, ldc,
  CUDA_R_32F,
  CUBLAS_GEMM_DEFAULT_TENSOR_OP);

accumulates in FP16
exact results only up to N < 2048

accumulates in FP32
nearly as fast as cublasHgemm
(same datapath, just a bit more I/O)

make sure to ask for tensor cores!
PERFORMANCE TIPS FOR CUBLAS

When N gets large, A and B matrices can get very long and skinny

Prefer the memory layout that keeps lda and ldb small

Change the transa/transb parameters on cublas*Gemm* to match

Your caches and TLBs will thank you!

\[
\begin{align*}
\text{lda} &= 1000 \\
\text{ldb} &= 1000
\end{align*}
\]

vs.

\[
\begin{align*}
\text{lda} &= 1000 \\
\text{ldb} &= 8000000
\end{align*}
\]
CUDA WMMA
CUDA TENSOR CORE PROGRAMMING

WMMA Matrix Multiply and Accumulate Operation

Warp-level operation to perform matrix multiply and accumulate

\[
\text{wmma::mma_sync}(\text{Dmat}, \text{Amat}, \text{Bmat}, \text{Cmat});
\]

\[
D = \begin{pmatrix}
\end{pmatrix} + \begin{pmatrix}
\end{pmatrix}
\]
TENSOR SYNCHRONIZATION
Full Warp 16x16 Matrix Math

Warp-synchronizing operation
Composed Matrix Multiply and Accumulate for 16x16 matrices
Result distributed across warp
Per-Thread fragments to hold components of matrices for use with Tensor Cores

```
wmma::fragment<matrix_a, ...> Amat;
```
CUDA TENSOR CORE PROGRAMMING

WMMA load and store operations

Warp-level operation to fetch components of matrices into fragments

```cpp
wmma::load_matrix_sync(Amat, a, stride);
```
CUDA TENSOR CORE PROGRAMMING

WMMA load and store operations

Warp-level operation to fetch components of matrices into fragments

\[ \text{wmma::store_matrix_sync}(d, \text{Dmat}, \text{stride}); \]
__device__ void tensor_op_16_16_16(
    float *d, half *a, half *b, float *c)
{
    wmma::fragment<
        matrix_a, ...
    > Amat;
    wmma::fragment<
        matrix_b, ...
    > Bmat;
    wmma::fragment<
        matrix_c, ...
    > Cmat;

    wmma::load_matrix_sync(Amat, a, 16);
    wmma::load_matrix_sync(Bmat, b, 16);
    wmma::fill_fragment(Cmat, 0.0f);

    wmma::mma_sync(Cmat, Amat, Bmat, Cmat);

    wmma::store_matrix_sync(d, Cmat, 16,
        wmma::row_major);
}

CUDA C++
Warp-Level Matrix Operations
TENSOR CORES IN CUDA FORTRAN

Similar to CUDA C WMMA API, with some name changes

real(2) support for half-precision data available (on both host and device) in PGI 19.7 compilers

Requires `wmma` Fortran module and macros in `cuf_macros.CUF` file
CUDA FORTRAN TENSOR CORE EXAMPLE

Device Code

```
#include "cuf_macros.CUF"
module m
contains
  attributes(global) subroutine wmma_16x16(a, b, c)
  use wmma
  real(2), intent(in) :: a(16,*), b(16,*)
  real(4) :: c(16,*)
  WMMASubMatrix(WMMAMatrixA, 16, 16, 16, Real, WMMAColMajor) :: sa
  WMMASubMatrix(WMMAMatrixB, 16, 16, 16, Real, WMMAColMajor) :: sb
  WMMASubMatrix(WMMAMatrixC, 16, 16, 16, Real, WMMAKind4) :: sc
  sc = 0.0_4
  call wmmaLoadMatrix(sa, a(1,1), 16)
  call wmmaLoadMatrix(sb, b(1,1), 16)
  call wmmaMatMul(sc, sa, sb, sc)
  call wmmaStoreMatrix(c(1,1), sc, 16)
end subroutine wmma_16x16
end module m
```
CUTLASS 1.3
CUDA C++ Template Library for Matrix Algebra

CUTLASS template library for GEMM computations
• Blocked structure to maximize data reuse
• Software pipelined to hide latency
• Conflict-free Shared Memory access to maximize data throughput

See CUTLASS GTC 2018 talk.
Tensor Cores

- **8x speedup** for mixed-precision matrix multiply
- Programmable via WMMA API  (CUDA 9)

Direct access to Volta Tensor Cores: **mma.sync**  (new instruction in CUDA 10.1)

- Maximum efficiency on Volta SM Architecture
- New in CUTLASS 1.3

Volta Tensor Cores - Performance Relative to cuBLAS

CUTLASS 1.3 - CUDA 10.1 - V100

Performance relative to cuBLAS

<table>
<thead>
<tr>
<th></th>
<th>mma</th>
<th>WMMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>F16 accum, NN</td>
<td>91%</td>
<td>79%</td>
</tr>
<tr>
<td>F16 accum, NT</td>
<td>96%</td>
<td>71%</td>
</tr>
<tr>
<td>F16 accum, TN</td>
<td>92%</td>
<td>78%</td>
</tr>
<tr>
<td>F16 accum, TT</td>
<td>93%</td>
<td>68%</td>
</tr>
<tr>
<td>F32 accum, NN</td>
<td>97%</td>
<td>63%</td>
</tr>
<tr>
<td>F32 accum, NT</td>
<td>92%</td>
<td>57%</td>
</tr>
<tr>
<td>F32 accum, TN</td>
<td>98%</td>
<td>71%</td>
</tr>
<tr>
<td>F32 accum, TT</td>
<td>94%</td>
<td>57%</td>
</tr>
</tbody>
</table>
PROFILING
TENSOR CORES WITH NSIGHT COMPUTE

- The Nsight Compute CLI allows collecting several metrics related to tensor core usage

- This data can be viewed from the CLI or via the Nsight Compute GUI

```
nv-nsight-cu-cli --metrics sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active ./cudaTensorCoreGemm
```

```
compute_gemm, 2019-Aug-08 12:48:39, Context 1, Stream 7
Section: Command line profiler metrics

<table>
<thead>
<tr>
<th>sm__pipe_tensor_cycles_active.avg.pct_of_peak_sustained_active</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>43.44</td>
</tr>
</tbody>
</table>
```

![Nsight Compute GUI screenshot showing metrics](image.png)
HPC CASE STUDIES
PARTICLE PUSH IN MAGNETIC FIELD USING TENSOR CORE (PARTICLE IN CELL)
PARTICLE PUSH

• The governing equation for particle velocity in magnetic field is given by:

\[
\frac{dv}{dt} = \frac{q}{m(v \times B)}, \quad v = velocity, \quad q = charge, \quad m = mass, \quad B = magnetic \ field
\]

• *Discretizing the above equation in 2 dimension can lead to:

```c
/* grab magnetic field at current position*/
B=EvalB(x);

/* get new velocity at n+1*/
v2[0] = v[0] + q/m*B*v[1]*dt;
v2[1] = v[1] - q/m*B*v[0]*dt;

/* update position*/
x2[0] = x[0] + v2[0]*dt;

/* push down*/
v[0]=v2[0];
v[1]=v2[1];
```

Gather by magnetic forces from the cell vertices.

*Ref: https://www.particleincell.com/2011/vxb-rotation/*
SCATTER PARTICLE INSTEAD OF GATHER

- We separate velocity direction and magnitude. Magnitude in FP32 while directions in FP16.
- We pack velocity, magnetic field vectors into Tensor Core format. This is basically the scatter operation.
- The GEMM updates velocities and add them back to particle final velocity at a given time step in FP32.

Gather by interpolation forces from the cell vertices.

Scatter particle properties to nodes and add compute at nodes.

\[(x_1, y_1) \quad (x_2, y_2) \quad (x_3, y_3) \quad (x_4, y_5)\]
BORIS METHOD

*Boris method is the *de facto* standard for particle pushing in plasma simulation codes. It is an explicit technique.

The following equations summarize Boris method.

In the absence of Electric Field, V+ acts as velocity update. Electric field can be easily added.

\[
\begin{align*}
\frac{v^+ - v^-}{\Delta t} &= \frac{q}{2m} (v^+ + v^-) \times B \\
v' &= v^- + v^- \times t \\
v^+ &= v^- + v' \times s \\
v' &= v^- + -1 \times (f1(B) \times v^-) \\
v^+ &= v^- + -1 \times (f2(B) \times v')
\end{align*}
\]

MINI-APP PICTC PERFORMANCE COMPARISON

![Chart showing performance comparison between Reference and Tensor Cores for FP16.]  

- **Source:** CUDA 10.1, Summit
- **Source:** [GitHub](https://github.com/vishalmehtha1991/pictc)
SPHERICAL HARMONICS IN IFS
(WEATHER & CLIMATE)
Spherical harmonics are eigen functions of the Laplacian in spherical co-ordinates.

Grid point space → FFT → Legendre Transform (DGEMM) → Spectral Space

Spectral Space → Inv Legendre Transform (DGEMM) → IFFT → Grid point space

https://doi.org/10.1145/3324989.3325711
SUMMARY

• When used appropriately Tensor Cores can achieve as much as an 8X performance increase.
• A variety of High and Low-level entry points are available for programming Tensor Cores
• Rethinking data layouts, mixed precision and algorithmic patterns is the key to Tensor Core utilization.
ADDITIONAL RESOURCES


cuTensor & CUTLASS -


CUDA Tensor Core Sample - https://docs.nvidia.com/cuda/cuda-samples/index.html#cudatensor-core-gemm