Lecture 5: tensor cores, libraries and tools

Prof. Mike Giles

mike.giles@maths.ox.ac.uk

Oxford University Mathematical Institute

Tensor cores

Starting with the Volta generation of GPUs, NVIDIA introduced “tensor cores” to greatly accelerate matrix multiplication for Machine Learning.

WMMA = Warp Matrix Multiply and Accumulate

A single `wmma` instruction, executed by all threads in the warp, performs a small matrix-matrix multiplication and addition:

$$\underbrace{D}_{M\times N} = \underbrace{A}_{M\times K} * \underbrace{B}_{K\times N} + \underbrace{C}_{M\times N}$$

Volta GPUs only supported 16×16 matrices ($M{=}K{=}N{=}16$) with A, B of type `half` (fp16) and C, D of type `float` (fp32).

Tensor cores

The small matrices are referred to as “fragments” (because they’re usually parts of larger matrices) and stored within the warp’s threads in a way which is “opaque” (not visible to programmer).

In the user’s kernel code, the warp

- loads in the fragments from shared memory
- performs the MMA matrix-multiplication-addition instruction
- stores the resulting fragment in shared memory

Tensor cores

Part of the kernel code for $C = A * B$ for a single warp:

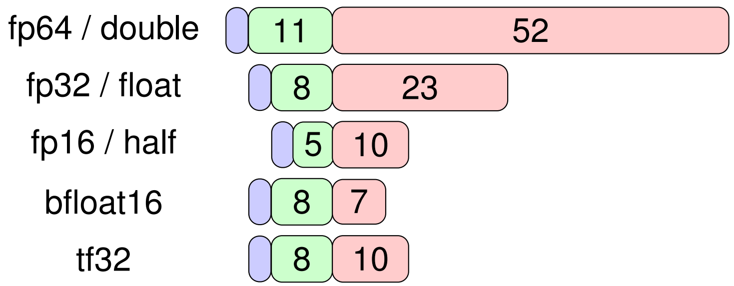
```
int M=16, N=16, K=16;

// Declare the fragments
wmma::fragment<wmma::matrix_a, M, N, K, half, wmma::col_major> a_frag;
wmma::fragment<wmma::matrix_b, M, N, K, half, wmma::col_major> b_frag;
wmma::fragment<wmma::accumulator, M, N, K, float> c_frag;

// Initialize the output to zero
wmma::fill_fragment(c_frag, 0.0f);

// Load the inputs
wmma::load_matrix_sync(a_frag, a, M);
wmma::load_matrix_sync(b_frag, b, K);

// Perform the matrix multiplication
wmma::mma_sync(c_frag, a_frag, b_frag, c_frag);

// Store the output
wmma::store_matrix_sync(c, c_frag, M, wmma::col__major);
```

Tensor cores

New GPUs have `wmma` instructions for various datatypes.

Floating point variables have the form $(-1)^s \times m \times 2^e$ where s is a sign bit, and m and e are the mantissa and exponent, represented by a varying number of bits

more exponent bits = more range (from small to big)
more mantissa bits = more accuracy



Tensor cores

In the latest Hopper GPUs, the `wmma` instructions support the following combinations of types and matrix sizes:

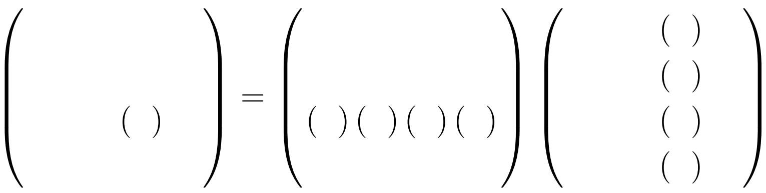
A, B	C, D	$M \times K \times N$
half	half	$16{\times}16{\times}16$, $32{\times}8{\times}16$, $8{\times}32{\times}16$
half	float	$16{\times}16{\times}16$, $32{\times}8{\times}16$, $8{\times}32{\times}16$
bfloat16	float	$16{\times}16{\times}16$, $32{\times}8{\times}16$, $8{\times}32{\times}16$
tf32	float	$16{\times}16{\times}8$
double	double	$8{\times}8{\times}4$
char	int	$16{\times}16{\times}16$, $32{\times}8{\times}16$, $8{\times}32{\times}16$

My guess is that the `wmma` instructions take the same amount of time for each of these combinations.

Tensor cores

The `wmma` instructions do a small matrix product within a single warp.

When doing a large matrix multiplication, you have to "tile" the output, and then different warps process different tiles, independently.



$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Tensor cores

To see how to implement tiling read
`developer.nvidia.com/blog/programming-tensor-cores-cuda-9/`
but this is an example code, not optimal.

Doing it optimally is hard, and NVIDIA has also introduced some tricky new features to improve performance (asynchronous loading directly into shared memory).

In general, best to leave this to NVIDIA experts and use the cuBLAS library!

CUDA libraries

Originally, NVIDIA planned to provide only one or two maths libraries, but over time these have steadily increased

- CUDA math library
  all of the standard math functions you would expect
  (i.e. very similar to what you would get from Intel)
  - various exponential and log functions
  - trigonometric functions and their inverses
  - hyperbolic functions and their inverses
  - error functions and their inverses
  - Bessel and Gamma functions
  - vector norms and reciprocals (esp. for graphics)
  - mainly single and double precision – a few in half precision

CUDA libraries

- cuBLAS
  - basic linear algebra subroutines for dense matrices
  - includes matrix-vector and matrix-matrix product
  - uses tensor cores by default for performance
  - user can specify the accuracy required (e.g. allowing use of TF32 for floats)
  - routines called by either host or kernel (device API)
  - some support for a single routine call to do a “batch” of smaller matrix-matrix multiplications
  - also support for using CUDA streams to do a large number of small tasks concurrently

CUDA libraries

cuBLAS is a set of routines to be called by user host code:

- helper routines:
  - memory allocation
  - data copying from CPU to GPU, and vice versa
  - error reporting
- compute routines:
  - matrix-matrix and matrix-vector product
  - Warning! Some calls are asynchronous, i.e. the call starts the operation but the host code then continues before it has completed

cuBLASLt is a new lightweight version

cuBLASDx is a new (preview) device side version

cuBLASXt extends cuBLAS to multiple GPUs

CUDA libraries

- cuFFT
  - 1D, 2D, 3D Fast Fourier Transform
  - has most variations found in FFTW and elsewhere
  - like cuBLAS, routines called by user host code:
    - helper routines include “plan” construction
    - compute routines perform 1D, 2D, 3D FFTs
    - it supports doing a “batch” of independent transforms, e.g. applying 1D transform to a 3D dataset

CUDA libraries

- cuTENSOR
  - tensor linear algebra library
  - makes extensive use of new tensor cores
- cuSPARSE
  - various routines to work with sparse matrices
  - includes sparse matrix-vector and matrix-matrix products
  - also has solution of sparse triangular system
  - batched tridiagonal solver in cuBLAS not cuSPARSE
- cuDSS
  - new (preview) Direct Sparse Solver library

CUDA libraries

- cuRAND
  - random number generation
  - XORWOW, `mrg32k3a`, Mersenne Twister and `Philox_4x32_10` pseudo-random generators
  - Sobol quasi-random generator (with optional scrambling)
  - uniform, Normal, log-Normal, Poisson outputs
  - also device level routines for RNG within kernels
- cuSOLVER:
  - key LAPACK dense solvers, 3 – 6x faster than MKL
  - sparse direct solvers, 2–14x faster than CPU
  - latest version uses iterative refinement with low-precision tensor core operations

CUDA libraries

- CUB
  - collection of basic building blocks (e.g. sort, scan, reduction) at three levels: device, thread block, warp
  - available from `github.com/NVIDIA/cub`
- CUTLASS (CUDA Templates for Linear Algebra Subroutines)
  - collection of CUDA C++ template abstractions for implementing matrix-multiplication (GEMM)
  - available from `github.com/NVIDIA/cutlass`
- AmgX
  - library for algebraic multigrid
  - available from `developer.nvidia.com/amgx`

CUDA libraries

- cuDNN
  - library for Deep Neural Networks
- NCCL
  - NVIDIA Collective Communications Library
  - multi-GPU over both PCIe and NVlink
  - multi-node over NVIDIA/Mellanox NICs
- nvGraph
  - Page Rank, Single Source Shortest Path, Single Source Widest Path

CUDA libraries

- Thrust
  - high-level C++ template library with an interface based on the C++ Standard Template Library (STL)
  - very different philosopy to other libraries; users write standard C++ code (no CUDA) but get the benefits of GPU parallelisation
  - also supports x86 execution
  - relies on C++ object-oriented programming; certain objects exist on the GPU, and operations involving them are implicitly performed on the GPU
  - I've not used it, but for some applications it can be very powerful – e.g. lots of built-in functions for operations like sort and scan
  - also simplifies memory management and data movement

Useful header files

- `dbldbl.h` available from
  `https://gist.github.com/seibert/5914108`
  Header file for double-double arithmetic for quad-precision (developed by NVIDIA, but published independently under the terms of the BSD license)
- `cuComplex.h` part of the standard CUDA distribution
  Header file for complex arithmetic – defines a class and overloaded arithmetic operations.
- `helper_math.h` available with NVIDIA sample codes
  Defines operator-overloading operations for CUDA intrinsic vector datatypes such as `float4`

Other libraries

- Kokkos
  - another high-level C++ template library
  - developed in the US DoE Labs, so considerable investment in both capabilities and on-going software maintenance
  - I've not used it, but possibly worth investigating
  - for more information see
    `https://github.com/kokkos/kokkos/wiki`
    `https://trilinos.org/packages/kokkos/`

Other libraries

- MAGMA
  - a new LAPACK for GPUs – higher level numerical linear algebra, layered on top of cuBLAS
  - open source – freely available from
    `https://icl.utk.edu/magma/`
- OpenMM
  - `http://openmm.org/`
  - open source package to support molecular modelling at Stanford

Other libraries

- Fast multipole methods for N-body problems:
  - ExaFMM by Yokota and Barba:
    `http://www.bu.edu/exafmm/`
    `https://lorenabarba.com/figshare/exafmm-10-years-7-re-writes-the-tortuous-progress-of-computational-research/`
  - FMM2D by Holm, Engblom, Goude, Holmgren:
    `http://user.it.uu.se/~stefane/freeware`
  - software by Takahashi, Cecka, Fong, Darve:
    `onlinelibrary.wiley.com/doi/10.1002/nme.3240/pdf`
  - new solver within GROMACS:
    `www.mpinat.mpg.de/634623/Kohnke_2021_IJHPCA.pdf`
  - not clear to me which of these is still developed/maintained

Other libraries

- OP2 and OPS
  - high-level frameworks for unstructured (OP2) and multi-block (OPS) codes
  - uses CUDA on GPUs, OpenMP on CPUs, and MPI for message-passing on multiple systems
  - all implementation details are hidden from "users", so they don't have to know about CUDA/OpenMP/MPI programming
  - originally developed in Oxford; development continued now by Gihan Mudalige (Warwick) and Istvan Reguly (PPCU in Budapest)
  - code available on `https://op-dsl.github.io/`

Tools

Debugging using NVIDIA Compute Sanitizer:

- `compute-sanitizer --tool memcheck`
  detects array out-of-bounds errors, and mis-aligned device memory accesses
- `compute-sanitizer --tool racecheck`
  checks for shared memory race conditions:
  - Write-After-Write (WAW): two threads write data to the same memory location but the order is uncertain
  - Read-After-Write (RAW), Write-After-Read (WAR): one thread writes & one reads, with uncertain order
- `compute-sanitizer --tool initcheck`
  detects reading of uninitialised device memory
- `compute-sanitizer --tool synccheck`
  detects incorrect use of `__syncthreads` and related intrinsics

Tools

Other languages:

- CUDA Fortran: available from NVIDIA
- Python:
  `https://developer.nvidia.com/cuda-python`
  `https://numba.pydata.org/`
- MATLAB: can call kernels directly, or use OOP like Thrust to define MATLAB objects which live on the GPU
  `https://www.mathworks.com/solutions/gpu-computing.html`
- Mathematica: similar to MATLAB?
  `https://reference.wolfram.com/language/CUDALink/tutorial/Overview.html`
- R:
  `https://developer.nvidia.com/blog/accelerate-r-applications-cuda/`
  `http://www.r-tutor.com/gpu-computing`

Tools

OpenACC (“More Science, Less Programming”):

- like Thrust, aims to hide CUDA programming by doing everything in the top-level CPU code
- programmer takes standard C/C++/Fortran code and inserts pragmas saying what can be done in parallel and where data should be located
- `https://www.openacc.org/`

OpenMP 5.0 is similar but newer:

- pushed strongly by Intel
- `https://www.openmp.org/wp-content/uploads/20210924-OpenMP-update-for-DOE.pdf`

Tools

Integrated Development Environments (IDE):

- Nsight Visual Studio edition – NVIDIA plug-in for Microsoft Visual Studio
  `developer.nvidia.com/nsight-visual-studio-edition`
- Nsight Eclipse edition – IDE for Linux systems (now distributed as plug-ins for standard Eclipse)
  `developer.nvidia.com/nsight-eclipse-edition`
- these come with editor, debugger, profiler integration

Tools

NVIDIA Nsight Compute CLI profiler `ncu`:

- standalone software for Linux and Windows systems
- uses hardware counters to collect a lot of useful information
- I think only 1 SM is instrumented – implicitly assumes the others are behaving similarly
- lots of things can be measured, but a limited number of counters, so it runs the application multiple times if necessary to get full info
- see practical 3 for an example of its use
- can also visualise output using `ncu-ui`
  `https://docs.nvidia.com/nsight-compute/NsightCompute/index.html`

Summary

- significant effort to develop general purpose libraries or frameworks, to enable users to get the benefits without being CUDA experts
- too much going on for one person (e.g. me) to keep track of it all
- NVIDIA maintains webpages with links to CUDA libraries and tools:
  `developer.nvidia.com/gpu-accelerated-libraries`
  `developer.nvidia.com/tools-ecosystem`
- the existence of this ecosystem is a key part of CUDA’s success