

Lecture 5: libraries and tools

Prof. Mike Giles

mike.giles@maths.ox.ac.uk

Oxford University Mathematical Institute

Lecture 5 – p. 1/30

CUDA libraries

• cuBLAS

- basic linear algebra subroutines for dense matrices
- includes matrix-vector and matrix-matrix product
- it is possible to call cuBLAS routines from user kernels – 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
- `simpleCUBLAS` example in Practical 5 – taken from NVIDIA sample codes

Lecture 5 – p. 3/30

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

Lecture 5 – p. 2/30

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

cuBLASxt extends cuBLAS to multiple GPUs

Lecture 5 – p. 4/30

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
 - `simpleCUFFT` example in Practical 5 – taken from NVIDIA sample codes

Lecture 5 – p. 5/30

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
 - could be used for iterative solution
 - also has solution of sparse triangular system
 - note: batched tridiagonal solver is in cuBLAS not cuSPARSE

Lecture 5 – p. 6/30

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

Lecture 5 – p. 7/30

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

Lecture 5 – p. 8/30

CUDA Libraries

- NCCL
 - NVIDIA Collective Communications Library
 - multi-GPU over both PCIe and NVlink
 - multi-node over NVIDIA/Mellanox NICs
- cuDNN
 - library for Deep Neural Networks
- nvGraph
 - Page Rank, Single Source Shortest Path, Single Source Widest Path
- NPP (NVIDIA Performance Primitives)
 - library for imaging and video processing
 - includes functions for filtering, JPEG decoding, etc.

Figure 5.1, p. 9/30

CUDA 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
 - again I've not used it, but possibly worth investigating
 - for more information see
 - <https://github.com/kokkos/kokkos/wiki>
 - <https://trilinos.org/packages/kokkos/>

CUDA Libraries

- Thrust
 - high-level C++ template library with an interface based on the C++ Standard Template Library (STL)
 - very different philosophy 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

Lecture 5 – p. 10/30

Useful header files

- `dbl_dbl.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

- 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/>
 - developed by Jack Dongarra, Jim Demmel and others

Lecture 5 – p. 13/30

The 7 dwarfs

- Phil Colella, senior researcher at Lawrence Berkeley National Laboratory, talked about “7 dwarfs” of numerical computation in 2004
- expanded to 13 by a group of UC Berkeley professors in a 2006 report: “A View from Berkeley”
www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf
- key algorithmic kernels in many scientific computing applications
- very helpful to focus attention on HPC challenges and development of libraries and problem-solving environments/frameworks.

Lecture 5 – p. 15/30

Other libraries

- ArrayFire from Acclereyes:
 - was commercial software, but now open source
 - supports both CUDA and OpenCL execution
 - C, C++ and Fortran interfaces
 - wide range of functionality including linear algebra, image and signal processing, random number generation, sorting
 - www.acclereyes.com/products/arrayfire

NVIDIA maintains webpages with links to a variety of CUDA libraries:

developer.nvidia.com/gpu-accelerated-libraries
and other tools:

developer.nvidia.com/tools-ecosystem

Lecture 5 – p. 14/30

The 7 dwarfs

- dense linear algebra
- sparse linear algebra
- spectral methods
- N-body methods
- structured grids
- unstructured grids
- Monte Carlo

Lecture 5 – p. 16/30

Dense linear algebra

- cuBLAS
- cuSOLVER
- CUTLASS
- MAGMA
- ArrayFire

Lecture 5 – p. 17/30

Spectral methods

- cuFFT
 - library provided / maintained by NVIDIA
- nothing else needed?

Lecture 5 – p. 19/30

Sparse linear algebra

- iterative solvers:
 - some available in PetSc
 - others can be implemented using sparse matrix-vector multiplication from cuSPARSE
 - NVIDIA has AmgX, an algebraic multigrid library
- direct solvers:
 - NVIDIA's cuSOLVER
 - SuperLU and STRUMPACK:
<https://www.exascaleproject.org/wp-content/uploads/2022/06/LiSherrySparseBofSlides.pdf>

Lecture 5 – p. 18/30

N-body methods

- OpenMM
 - <http://openmm.org/>
 - open source package to support molecular modelling, developed at Stanford
- Fast multipole methods:
 - 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

Lecture 5 – p. 20/30

Structured grids

- lots of people have developed one-off applications
- no great need for a library for single block codes (though possible improvements from “tiling”?)
- multi-block codes could benefit from a general-purpose library, mainly for MPI communication
- Oxford OPS project has developed a high-level open-source framework for multi-block codes, using GPUs for code execution and MPI for distributed-memory message-passing

all implementation details are hidden from “users”, so they don’t have to know about GPU/MPI programming

Lecture 5 – p. 21/30

Monte Carlo

- NVIDIA cuRAND library
- some use examples among NVIDIA sample codes
- Acclereyes ArrayFire library
- nothing else needed except for more output distributions?

Lecture 5 – p. 23/30

Unstructured grids

In addition to GPU implementations of specific codes there are projects to create high-level solutions which others can use for their application codes:

- Alonso, Darve and others (Stanford)
- Oxford / Imperial College project developed OP2, a general-purpose open-source framework based on a previous framework built on MPI

See <https://op-dsl.github.io/> for both OPS and OP2

Lecture 5 – p. 22/30

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

Lecture 5 – p. 24/30

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>

Lecture 5 – p. 25/30

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:

- strongly pushed by Intel to accommodate Xeon Phi and unify things, in some sense
- <https://www.openmp.org/wp-content/uploads/20210924-OpenMP-update-for-DOE.pdf>

Lecture 5 – p. 26/30

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

Lecture 5 – p. 27/30

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>

Lecture 5 – p. 28/30

Tools

GPU Direct:

- webpage:
`https://developer.nvidia.com/gpudirect`
- software support for direct data transfers from one GPU to another
- works across PCIe within a single machine
- works across PCIe-connected network adapters between different systems
- includes capabilities to work with cameras and other video input devices (e.g. for self-driving cars)
- very important in applications which might otherwise be limited by PCIe bandwidth

Summary

- active work on all of the dwarfs
- in most cases, 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 a webpage with links to CUDA tools/libraries:
`developer.nvidia.com/cuda-tools-ecosystem`
- the existence of this ecosystem is part of why I think CUDA will remain more used than OpenCL for HPC