Lecture 5: Libraries and tools

Prof Wes Armour

wes.armour@eng.ox.ac.uk

Prof Mike Giles

mike.giles@maths.ox.ac.uk

Oxford e-Research Centre

Department of Engineering Science

Learning outcomes

In this fifth lecture we will learn about GPU libraries and tools for GPU programming.

You will learn about:

- NVIDIA GPU libraries and there usefulness in scientific computing.

- Third party libraries.

- Directives based approaches to GPU computation.

- Tools for GPU programming.

Software overview

NVIDIA provides a **rich ecosystem of software tools** that allow you to easily utilise GPUs in your project.

During this lecture we will focus on a range of **libraries and software tools that will make your life easier** when either writing CUDA code or when utilising GPUs in your projects.
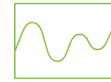
Math Libraries

GPU-accelerated math libraries lay the foundation for compute-intensive applications in areas such as molecular dynamics, computational fluid dynamics, computational chemistry, medical imaging, and seismic exploration.

cuBLAS

GPU-accelerated basic linear algebra (BLAS) library

Learn More >

cuFFT

GPU-accelerated library for Fast Fourier Transforms

Learn More >

CUDA Math Library

GPU-accelerated standard mathematical function library

Learn More >

cuRAND

GPU-accelerated random number generation (RNG)

Learn More >

cuSOLVER

GPU-accelerated dense and sparse direct solvers

Learn More >

cuSPARSE

GPU-accelerated BLAS for sparse matrices

Learn More >

cuTENSOR

GPU-accelerated tensor linear algebra library

Learn More >

AmgX

GPU-accelerated linear solvers for simulations and implicit unstructured methods

Learn More >

Dependencies – Advantages / Disadvantages

Some advantages:

- Simple to use – you don't need to write your own complex code to perform a specific task.

- Well maintained – always benefit from the latest optimisations and improvements.

- Easier(?) to move from CPU code to GPU code (for example see cuFFTW).

Some disadvantages:

- Can make installing your code on another system harder (the user also needs to have the dependency installed).

- If the dependency isn't maintained it could break your code as other things (e.g. compiler) are updated.

- If your code is very dependent on it and the developers stop supporting it – you become the owner (not a great position to be in).

CUDA math library

The CUDA Math Library contains all of the typical mathematical functions that you will need for your projects. It is very similar to Intel's MKL library.

- 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)

- To use - "#include math.h"

The library supports standard int, float and double types, but in recent years has also added support for fp8, fp16 and bfloat16.

Typecasting and SIMD intrinsics are also included in this library.

CUDA Math Library

GPU-accelerated standard mathematical function library

cuTENSOR

Tensor cores originally introduced on Volta in 2017 provided hardware enabled acceleration for matrix-matrix multiplies.

- Originally performed a 4x4 matrix multiply-accumulate (think FMA for matrices) using wmma:: instruction.

- Matrices A and B would be lower precision and the accumulators would be the same or higher precision

- With Ampere and Hopper some of these restrictions have been relaxed.

https://arxiv.org/pdf/2206.02874.pdf
https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/
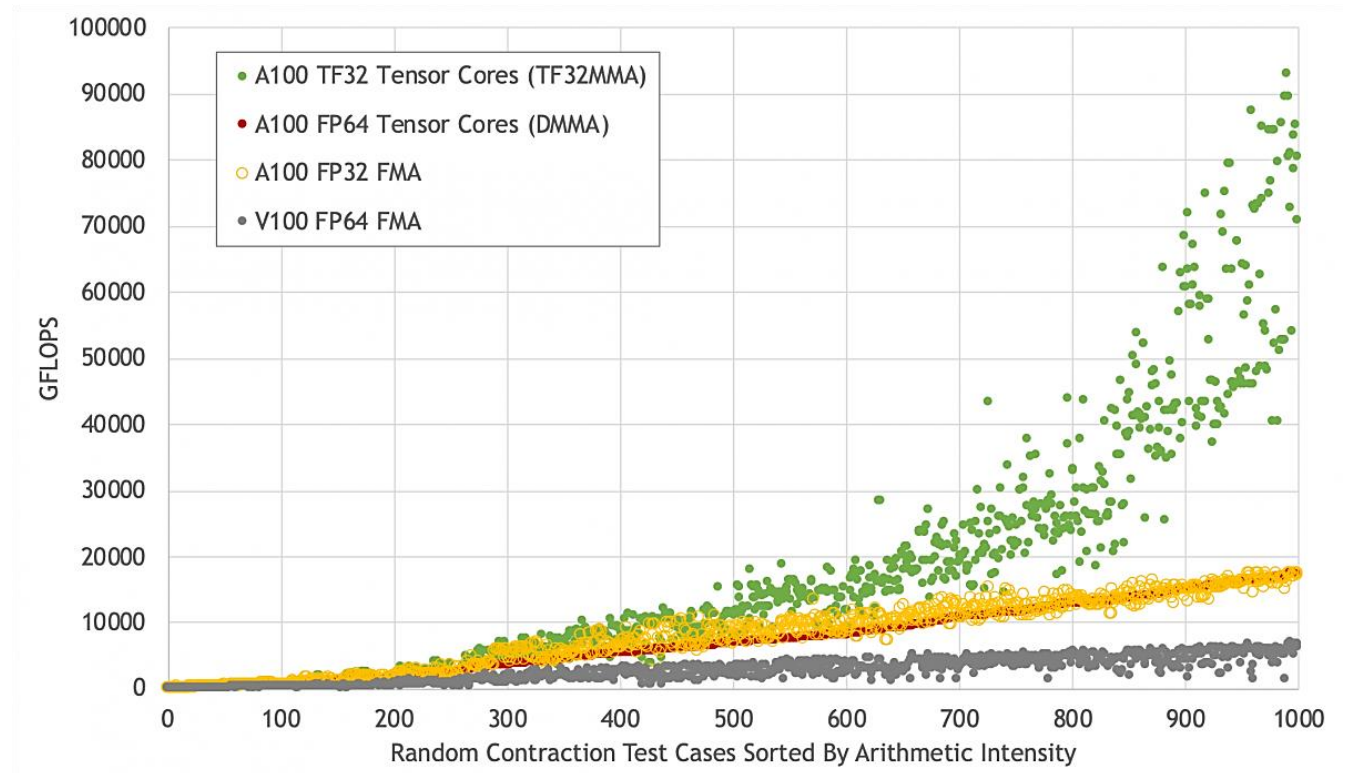https://developer.nvidia.com/blog/nvidia-automatic-mixed-precision-tensorflow/
https://developer.download.nvidia.com/video/gputechconf/gtc/2019/presentation/s9593-cutensor-high-performance-tensor-operations-in-cuda-v2.pdf

cuTENSOR

Current tensor cores (Hopper) are able to use various precisions and exploit sparsity to gain further acceleration.

- FP64 inputs with FP32 compute (DMMA).

- FP32 inputs with FP16, BF16, or TF32 compute.

- Complex-times-real operations.

- Conjugate (without transpose) support.

- Support for up to 64-dimensional tensors.

https://developer.nvidia.com/cutensor

https://docs.nvidia.com/cuda/cutensor/index.html

https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/

https://github.com/NVIDIA/CUDALibrarySamples/blob/master/cuTENSOR/reduction.cu

cuBLAS Library

The cuBLAS (CUDA Basic Linear Algebra Subprograms) library provides CUDA accelerated standard BLAS APIs (for 152 different routines) for dense matrices.

- includes matrix-vector and matrix-matrix product.

- it is possible to call cuBLAS routines from user kernels (via a 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.

- has support for multi-GPU operation (cuBLASTxt or cuBLASMg).

- has mixed / low precision implementations.

cuBLAS

GPU-accelerated basic linear algebra
(BLAS) library

cuBLAS Library

To use cuBLAS in your codes, a set of routines are called from your host code. These come in two forms, helper routines and compute routines.

Helper routines for:

- memory allocation

- data copying from CPU to GPU, and vice versa

- error reporting

Compute routines for:

- e.g. matrix-matrix and matrix-vector product

- **Warning!** Some calls are asynchronous, i.e. the call starts the operation

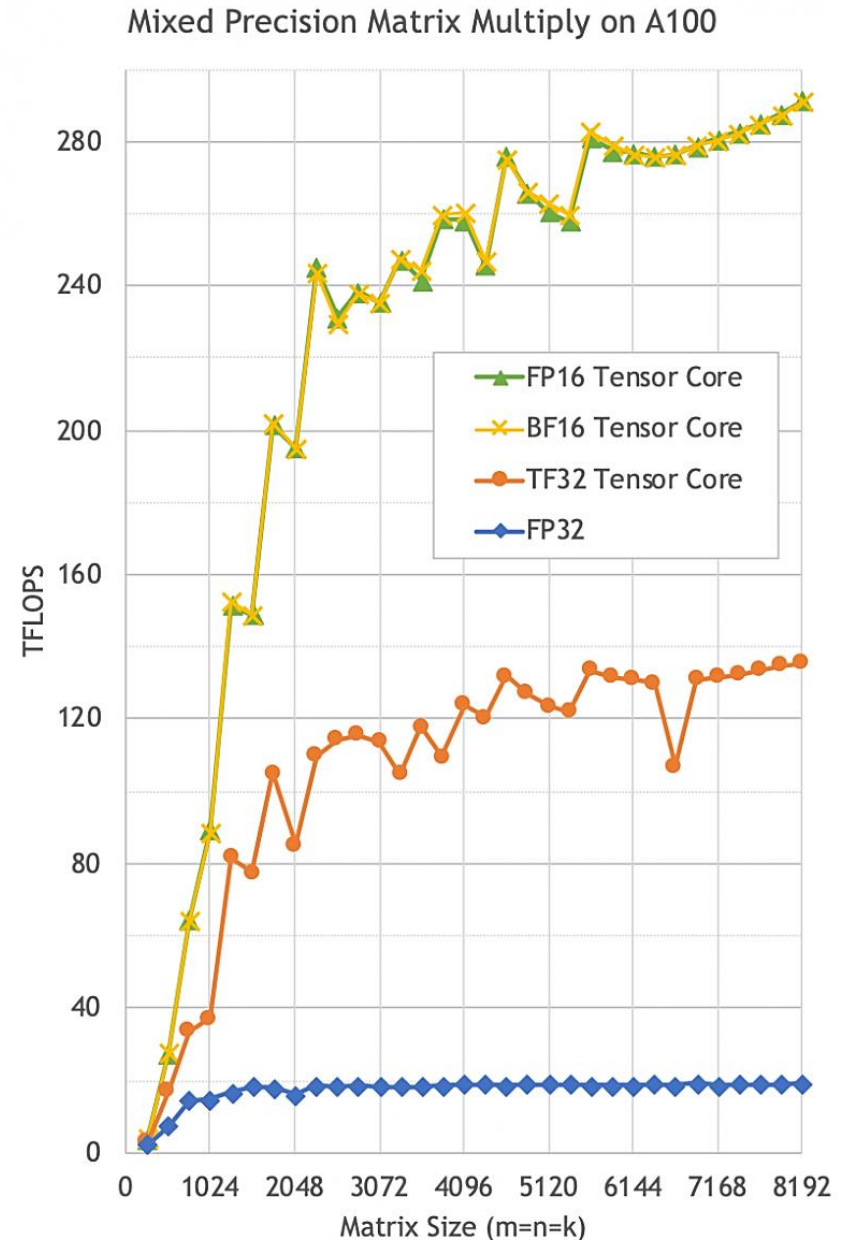 but the host code then continues before it has completed!!

cuBLAS

GPU-accelerated basic linear algebra
(BLAS) library

cuBLAS Library

cuBLAS is one of three libraries that use "tensor cores". Tensor cores are different to a standard processing core, the are designed to perform very specific operations and these operations are executed on mixed precision data.

If you are able to reduce the precision of your matrix / vector operations in cuBLAS you can gain significant acceleration.
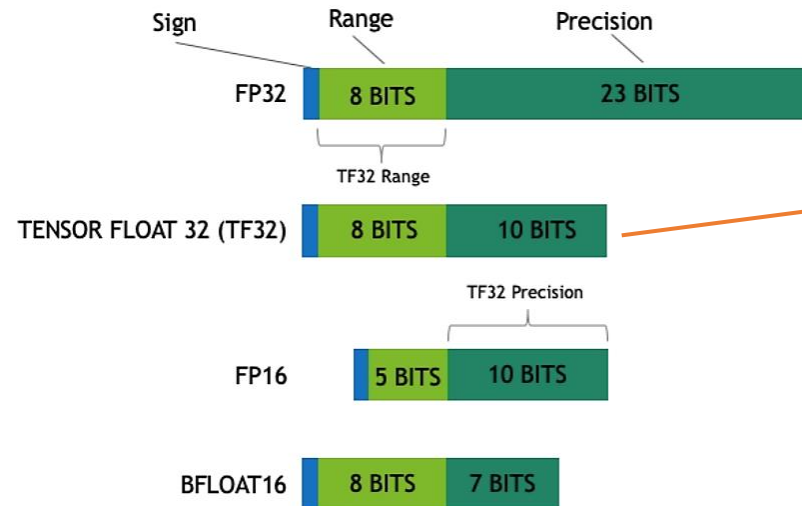
https://developer.nvidia.com/cublas

Mixed Precision Matrix Multiply on A100

cuBLAS Library

Since Ampere, there has been a "TF32" variable that is a compromise between FP32 and BFLOAT16. It allows for a compromise between lower precision and FP32 matrix-matrix operations.
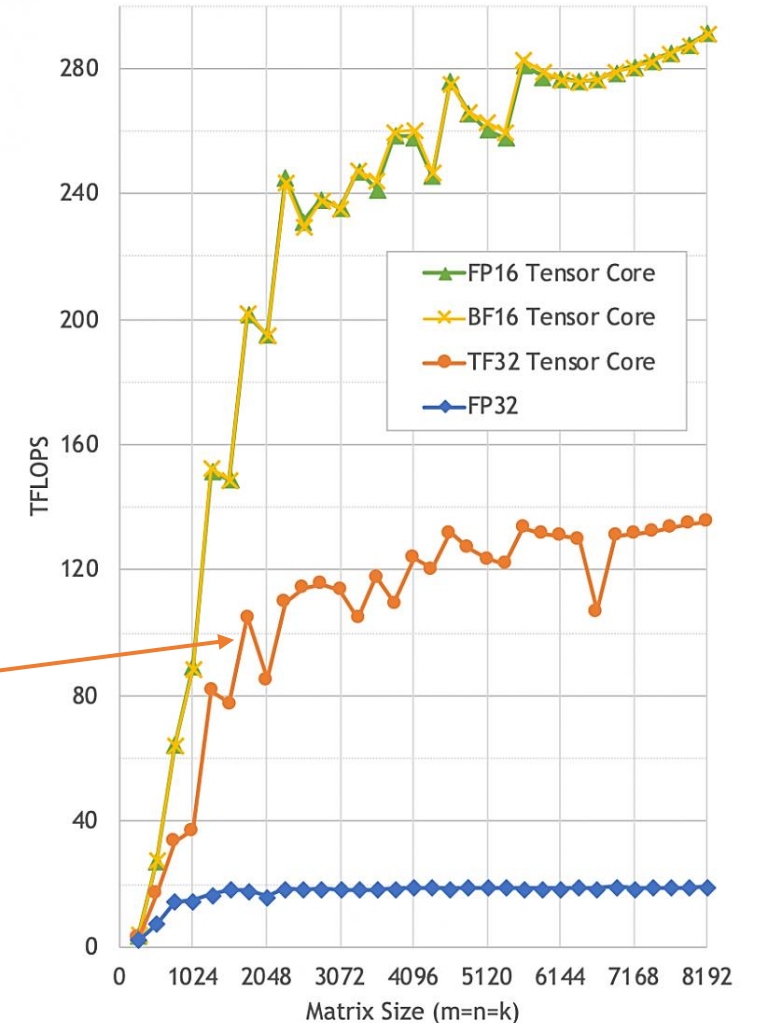
Also DMMA64 for double precision – uses AI magic?

Useful for:

- AI Training

- Liner solvers

cuFFT Library

The cuFFT library is a GPU accelerated library that provides Fast Fourier Transforms.

- Provides 1D, 2D and 3D FFTs.

- Has almost all of the variations found in FFTW and other CPU libraries.

- Includes the cuFFTW library, a porting tool, to enable users of FFTW to start using GPUs with minimal effort.

- Provides some device level functionality. *If this is something of interest ask Karel – he has already produced shared memory device level FFTs for our projects.*
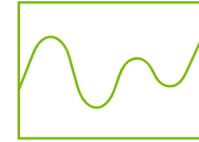
cuFFT

GPU-accelerated library for Fast Fourier Transforms

cuFFT Library

cuFFT is used exactly like cuBLAS - it has a set of routines called by host code:

Helper routines include "plan" construction.

Compute routines perform 1D, 2D, 3D FFTs:

- `cufftExecC2C()` - complex-to-complex.
- `cufftExecR2C()` - real-to-complex.
- `cufftExecC2R()` - complex-to-real.

(double precision routines have different function calls, e.g. `cufftExecZ2Z()`)

It supports doing a "batch" of independent transforms, e.g. applying 1D transform to a 3D dataset

The `simpleCUFFT` example in SDK is a good starting point.

cuFFT

GPU-accelerated library for Fast Fourier Transforms

https://docs.nvidia.com/cuda/cufft/index.html#introduction
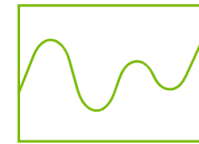
cuSPARSE Library

cuSPARSE is a GPU accelerated library that provides various routines to work with sparse matrices.

- Includes sparse matrix-vector and matrix-matrix products.

- Can be used for iterative solution (but see cuSOLVER for an easy life).

- Also has solution of sparse triangular system

- Note: batched tridiagonal solver is in cuBLAS not cuSPARSE

cuSPARSE

GPU-accelerated BLAS for sparse matrices

cuRAND Library

The cuRAND library is a GPU accelerated library for random number generation.

It has many different algorithms for pseudorandom and quasi-random number generation.

Pseudo: XORWOW, mrg32k3a, Mersenne Twister and Philox 4x32_10
Quasi: SOBOL and Scrambled SOBOL

Uniform, Normal, log-Normal and Poisson outputs

This library also includes device level routines for RNG within user kernels.
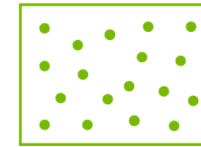
cuRAND

GPU-accelerated random number generation (RNG)

cuSOLVER

cuSOLVER brings together cuSPARSE and cuBLAS.

Has solvers for dense and sparse systems.

Key LAPACK dense solvers, 3 – 6x faster than MKL.

Sparse direct solvers, 2–14x faster than CPU equivalents.

cuSOLVER

GPU-accelerated dense and sparse direct solvers

Other notable libraries

CUB (CUDA Unbound): https://nvlabs.github.io/cub/

- Provides a collection of basic building blocks at three levels: device, thread block, warp.
- Functions include sort, scan and reduction.
- Thrust uses CUB for CUDA versions of key algorithms.
- Last update over a year ago…

http://on-demand.gputechconf.com/gtc/2014/presentations/S4566-cub-collective-software-primitives.pdf

AmgX (originally named NVAMG): http://developer.nvidia.com/amgx

- Library for algebraic multigrid

AmgX

GPU-accelerated linear solvers for simulations and implicit unstructured methods

Other notable libraries

cuDNN

- Library for Deep Neural Networks
- Some parts developed by Jeremy Appleyard (NVIDIA) working in Oxford
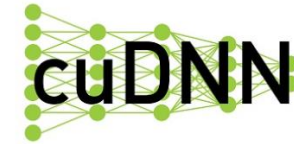
nvGraph

- 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.

CUDA Video Decoder API…

GPU-accelerated library of primitives for deep neural networks

nvGRAPH

GPU-accelerated library for graph analytics
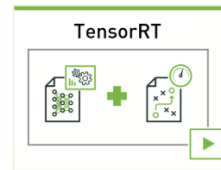

NVIDIA Performance Primitives

GPU-accelerated library for image and signal processing
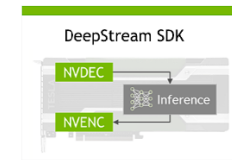
Other notable libraries

Deep Learning Libraries

GPU-accelerated library of primitives for deep neural networks

GPU-accelerated neural network inference library for building deep learning applications

Advanced GPU-accelerated video inference library

Signal, Image and Video Libraries
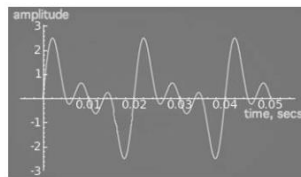
cuFFT

GPU-accelerated library for Fast Fourier Transforms

NVIDIA Performance Primitives

GPU-accelerated library for image and signal processing

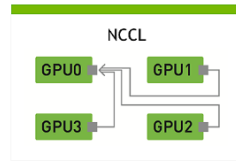### NVIDIA Codec SDK

High-performance APIs and tools for hardware accelerated video encode and decode

Other notable libraries

Parallel Algorithm Libraries

NCCL

Collective Communications Library for scaling apps across multiple GPUs and nodes

nvGRAPH

GPU-accelerated library for graph analytics

Thrust

GPU-accelerated library of parallel algorithms and data structures

MAGMA

MAGMA (Matrix Algebra on GPU and Multicore Architectures) has been available for a few years (See nice SC17 handout: http://www.icl.utk.edu/files/print/2017/magma-sc17.pdf)

- LAPACK for GPUs – higher level numerical linear algebra, layered on top of cuBLAS.

- Open source – freely available.

- Last updated February 2023.

https://icl.utk.edu/magma/
https://developer.nvidia.com/magma

ArrayFire

Originally a commercial software (from Accelereyes), but is now open source.

- Supports both CUDA and OpenCL execution.

- C, C++ and Python interfaces.

- Working to incorporate OneAPI (So should support most Intel hardware).

- Supports NVIDA and AMD GPUs/APUs, Intel processors and mobile devices from ARM, Qualcomm…

- Wide range of functionality including linear algebra, image and signal processing, random number generation, sorting…

- Actively developed.

https://arrayfire.com/
https://github.com/arrayfire/arrayfire

Thrust

Thrust is a high-level C++ template library with an interface based on the C++ Standard Template Library (STL).

Thrust has a very different philosophy to other libraries - users write standard C++ code (*no CUDA*) but get the benefits of GPU acceleration.

Thrust relies on C++ object-oriented programming – certain objects exist on the GPU, and operations involving them are implicitly performed on the GPU.

It has lots of built-in functions for operations like sort and scan.

It also simplifies memory management and data movement.

https://thrust.github.io/

Thrust

GPU-accelerated library of parallel algorithms and data structures

Kokkos

**Kokkos is another high-level C++ template library, similar to Thrust**.

It has been developed in the US DoE Labs, so there is considerable investment in both capabilities and on-going software maintenance.

Could be worth investigating if you are considering using Thrust in your projects.

For more information see
https://kokkos.github.io/kokkos-core-wiki/
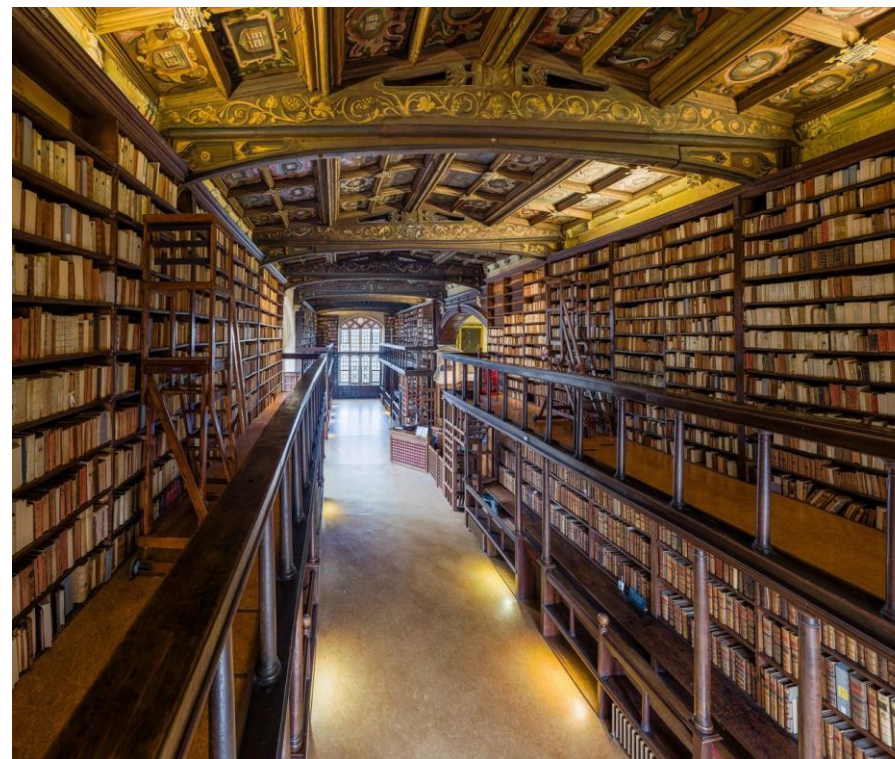https://kokkos.org/about/
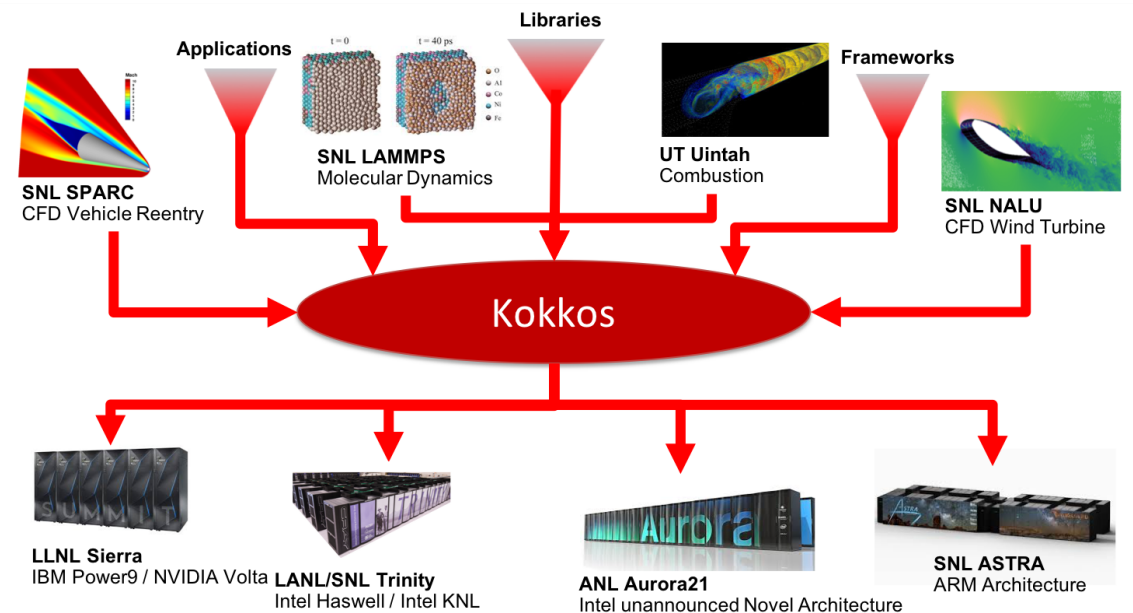
A final word on libraries

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

www.developer.nvidia.com/gpu-accelerated-libraries

and other tools:

www.developer.nvidia.com/tools-ecosystem

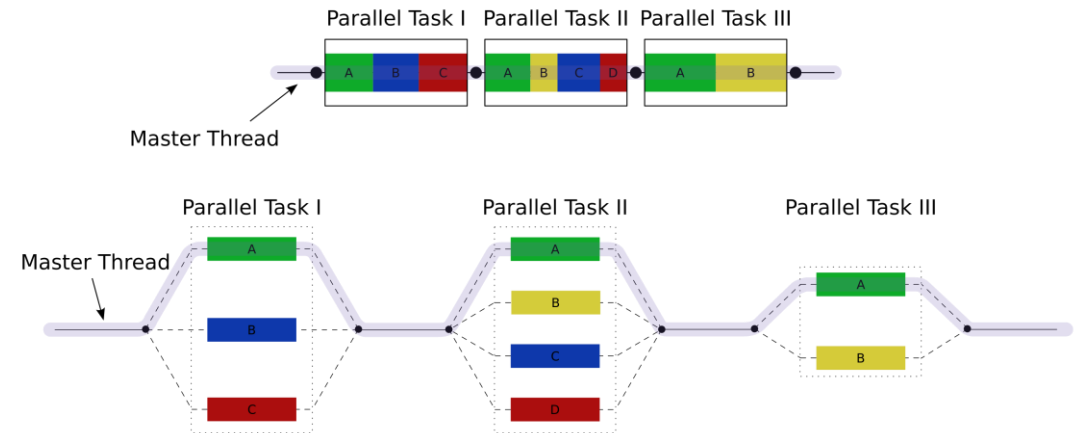https://en.wikipedia.org/wiki/Duke_Humfrey%27s_Library

A note on directive based approaches and other languages

OpenMP

OpenMP 5.0 is a directive based approach to parallelisation.

- It uses a fork-join model.

- Can be used in C / C++ and FORTRAN codes.

- It supports both CPU and GPU hardware.

Is now becoming the industry standard for in node CPU parallelisation.

SYCL

SYCL ("sickle") - C++ Single-source Heterogeneous Programming for OpenCL.

From KHRONOS Group (responsible for OpenCL).

Provides an abstraction layer that builds on OpenCL.

It enables **code for heterogeneous processors** to be written in a **"single-source" style** using completely standard C++.

Supported by Intel, NVIDIA and AMD.
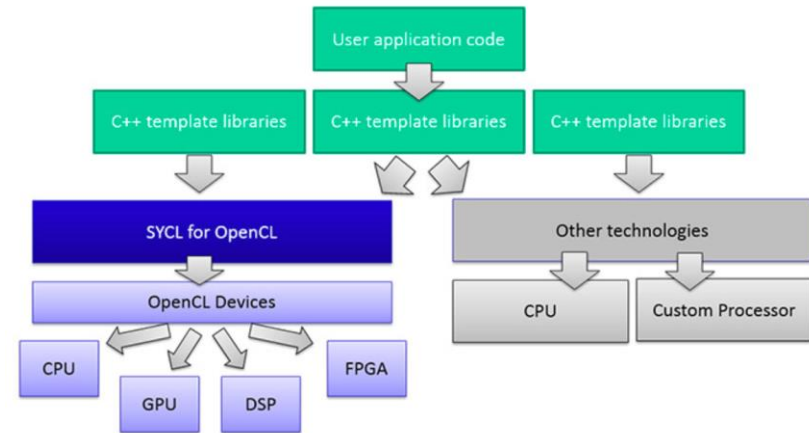
Other Languages

FORTRAN: CUDA FORTRAN compiler with natural FORTRAN equivalent to CUDA C.

MATLAB: can call kernels directly, or use OOP like Thrust to define MATLAB objects which live on the GPU
https://uk.mathworks.com/help/parallel-computing/run-matlab-functions-on-a-gpu.html

Mathematica: similar to MATLAB?

Python: CuPy (compatible with NumPy – acceleration for array computations), Numba and CUDA python
http://mathema.tician.de/software/pycuda
https://store.continuum.io/cshop/accelerate/
https://developer.nvidia.com/cuda-python
https://developer.nvidia.com/how-to-cuda-python
https://nvidia.github.io/cuda-python/overview.html

Other useful things…

CUDA Toolkit

Provides a comprehensive environment for C/C++ developers building GPU-accelerated applications.

OpenACC

Directives for parallel computing, is a new open parallel programming standard designed to enable all scientific and technical programmers.

CUDA FORTRAN

Enjoy GPU acceleration directly from your Fortran program using CUDA Fortran from The Portland Group.

PyCUDA

Gives you access to CUDA fuctionality from your Python code.

OpenCL™

OpenCL is a low-level API for GPU computing that can run on CUDA-powered GPUs.

Alea GPU

This is a novel approach to develop GPU applications on .NET, combining the CUDA with Microsoft's F#.

https://developer.nvidia.com/tools-ecosystem
https://developer.nvidia.com/language-solutions
https://developer.nvidia.com/hpc-sdk
https://developer.nvidia.com/hpc-compilers

Which library should I use for my problem?

The seven dwarfs

Phil Colella a 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

The seven dwarfs

These 13 dwarfs define key algorithmic kernels in many scientific computing applications.

They have been **very helpful to focus attention on HPC challenges and development of libraries and problem-solving environments/frameworks**.

The seven dwarfs

1. Dense linear algebra

2. Sparse linear algebra

3. Spectral methods

4. N-body methods

5. Structured grids

6. Unstructured grids

7. Monte Carlo

1. Dense Linear Algebra

Many tools available, some from NVIDIA, some third party:

- cuBLAS

- cuSOLVER

- MAGMA

- ArrayFire

CUTLASS, an NVIDIA tool for Fast Linear Algebra in CUDA C++ might also be worth a look if you can't use the above libraries for any reason.

https://devblogs.nvidia.com/cutlass-linear-algebra-cuda/

cuBLAS

GPU-accelerated basic linear algebra (BLAS) library

cuSOLVER

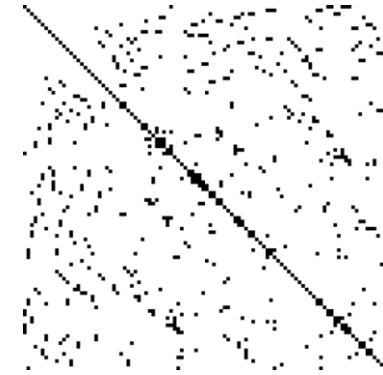GPU-accelerated dense and sparse direct solvers

MAGMA

ArrayFire

2. Sparse Linear Algebra

Iterative solvers

- Some available in PetSc (Portable, Extensible Toolkit for Scientific Computation, for solving PDEs) - https://petsc.org/release/overview/nutshell/
- Others can be implemented using sparse matrix-vector multiplication from cuSPARSE (is also now in PETSc).
- NVIDIA has AmgX, an algebraic multigrid library.

Direct solvers

- NVIDIA's cuSOLVER.
- SuperLU project (Gaussian elimination with partial pivoting) https://portal.nersc.gov/project/sparse/superlu/
- STRUMPACK (ask Mike) https://portal.nersc.gov/project/sparse/strumpack//
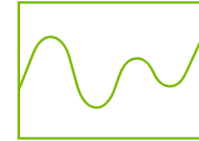

cuSOLVER

GPU-accelerated dense and sparse direct solvers

3. Spectral methods

cuFFT /cuFFTW

Library provided / maintained by NVIDIA

For those interested in FFTs on GPUs – ask karel…

cuFFT

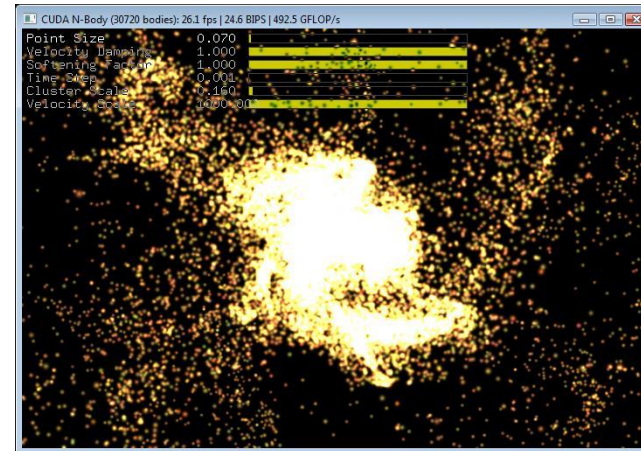GPU-accelerated library for Fast Fourier Transforms

4. 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/

- FMM2D by Holm, Engblom, Goude, Holmgren: http://user.it.uu.se/~stefane/freeware
  https://lorenabarba.com/figshare/exafmm-10-years-7-re-writes-the-tortuous-progress-of-computational-research/

- Software by Takahashi, Cecka, Fong, Darve:
  http://onlinelibrary.wiley.com/doi/10.1002/nme.3240/pdf

https://docs.nvidia.com/cuda/cuda-samples/index.html#cuda-n-body-simulation

https://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/nbody/doc/nbody_gems3_ch31.pdf

5. 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.

For those interested – ask Mike…

Slffea, Mysid - Drawn by Slffea, vectorized by Mysid.
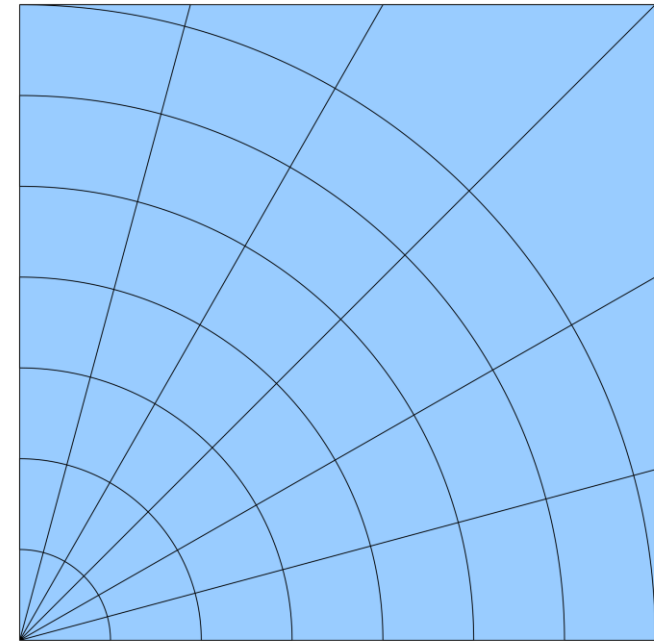
6. Unstructured grids

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 / Warwick project developed OP2, a general-purpose open-source framework based on a previous framework built on MPI.

- If there's interest Mike could talk about OP2 and OPS in lecture 8/9.

See https://op-dsl.github.io/ for both OPS and OP2
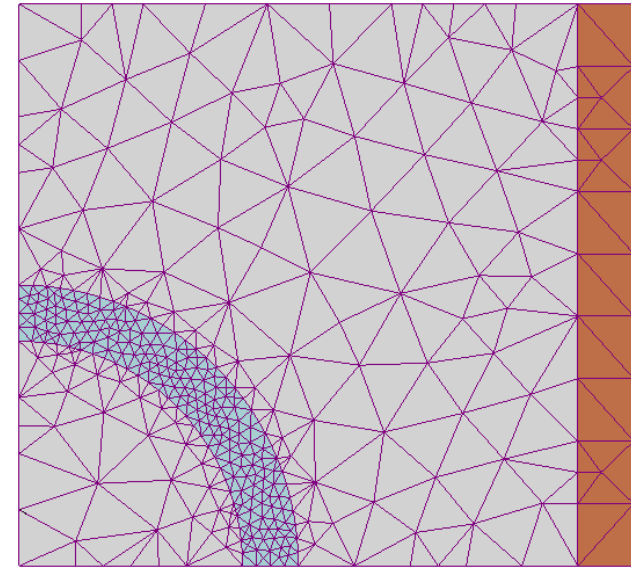
7. Monte Carlo methods

- NVIDIA cuRAND library.

- ArrayFire library.

- Some examples in CUDA SDK distribution.

- Nothing else needed except for more output distributions?

cuRAND

GPU-accelerated random number
generation (RNG)

Useful tools

Tools - Debugging

`compute-sanitizer –tool memcheck`

A command line tool that detects array out-of-bounds errors, and mis-aligned device memory accesses – very useful because such errors can be tough to track down otherwise.

`compute-sanitizer --tool racecheck`

This checks for shared memory race conditions:
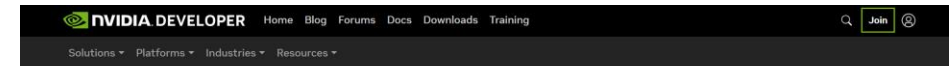
- Write-After-Write (WAW): two threads write data to the same memory location but the order is uncertain.

- Read-After-Write (RAW) and Write-After-Read (WAR): one thread writes and another reads, but the order is uncertain.

`compute-sanitizer --tool initcheck`

This detects the reading of uninitialised device memory.

`compute-sanitizer --tool synccheck`

This detects incorrect use of __syncthreads() and related intrinsics.

NVIDIA Compute Sanitizer

Compute Sanitizer Tools & API

Compute Sanitizer is a functional correctness checking suite. This suite contains multiple tools that can perform different type of checks. Tool features are described below.

The Compute Sanitizer API enables the creation of sanitizing and tracing tools that target CUDA applications. Examples of such tools are memory and race condition checkers. The Compute Sanitizer API is composed of three APIs: the callback API, the patching API and the memory API. It is delivered as a dynamic library on supported platforms.

Memcheck	Racecheck	Initcheck	Synccheck
The *memcheck* tool is a run time error detection tool for CUDA applications. The tool can precisely detect and report out of bounds and misaligned memory accesses to global, local and shared memory in CUDA applications. It can also detect and report hardware reported error information. In addition, the memcheck tool can detect and report memory leaks in the user application.	The *racecheck* tool is a run time shared memory data access hazard detector. The primary use of this tool is to help identify memory access race conditions in CUDA applications that use shared memory. In CUDA applications, storage declared with the __shared__ qualifier is placed on chip *shared memory*. All threads in a thread block can access this per block shared memory. Shared memory goes out of scope when the thread block completes execution. As shared memory is on chip, it is frequently used for inter-thread communication and as a temporary buffer to hold data being processed. As this data is being accessed by	The *initcheck* tool is a run time uninitialized device global memory access detector. This tool can identify when device global memory is accessed without it being initialized via device side writes, or via CUDA memcpy and memset API calls. Currently, this tool only supports detecting accesses to device global memory.	The *synccheck* tool is a runtime tool that can identify whether a CUDA application is correctly using synchronization primitives, specifically __syncthreads() and __syncwarp() intrinsics and their Cooperative Groups API counterparts.

https://developer.nvidia.com/compute-sanitizer

Tools – CUDA-GDB

For those familiar with the GNU debugger – GDB, this is an extension of GDB that allows users to debug both GPU and CPU code.

All existing GDB debugging features are included for debugging host code and then further functionality allows the user to debug device code.

Supports C/C++ and Fortran (that includes CUDA code).

CUDA-GDB

Delivers a seamless debugging experience that allows you to debug both the CPU and GPU portions of your application simultaneously. Use CUDA-GDB on Linux or MacOS, from the command line, DDD or EMACS.

Tools - IDEs

Integrated Development Environments (IDE):

Nsight Systems – Unified IDE for Windows/Linux/Mac/Jetson:
https://developer.nvidia.com/nsight-systems

Nsight Visual Studio edition – NVIDIA plugin for Microsoft Visual Studio
http://developer.nvidia.com/nvidia-nsight-visual-studio-edition

Nsight Eclipse plugins
https://docs.nvidia.com/cuda/nsight-eclipse-plugins-guide/index.html

these come with editor, debugger, profiler integration
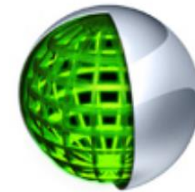
NVIDIA® Nsight™

The ultimate development platform
for heterogeneous computing. Work
with powerful debugging and profiling
tools, optimize the performance of
your CPU and GPU code. Find out
about the Ecilipse Edition and the
graphics debugging enabled Visual
Studio Edition.

Tools - Profiling

NVIDIA Profiler `ncu` or `ncu-ui` for a graphical interface.

- This is a standalone piece of software for Linux and Windows systems.

- It uses hardware counters to collect <u>a lot</u> of useful information.

- Lots of things can be measured, but the limited number of counters means that, for some larger applications, it runs the application multiple to gather necessary information.

- The `ncu` CLI can be useful if you want to profile on a machine that you don't have a graphical interface to.

- Do `ncu --help` for more info on different options.
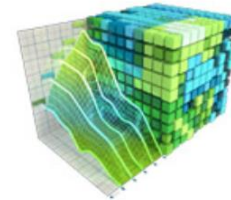
NVIDIA Visual Profiler

This is a cross-platform performance profiling tool that delivers developers vital feedback for optimizing CUDA C/C++ applications. First introduced in 2008, Visual Profiler supports all CUDA capable NVIDIA GPUs shipped since 2006 on Linux, Mac OS X, and Windows.

https://docs.nvidia.com/nsight-compute/NsightCompute/index.html

https://docs.nvidia.com/cuda/profiler-users-guide/index.html

What have we learnt?

In this lecture we've looked at the **wide software ecosystem that now surrounds GPU computing** and how that can be used to make your life as a programmer easier.

We've looked at **directives based approaches** and how these are useful.

Finally we've looked at **tools** that allow us to develop CUDA code in an easy and maintainable way.