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
  - significant input from Vasily Volkov at UC Berkeley; one routine contributed by Jonathan Hogg from RAL
  - 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

- SimpleSUGARS example in SDK is a good example code

- cuBLASxt extends cuBLAS to multiple GPUs
CUDA libraries

- **cuFFT**
  - Fast Fourier Transform
  - 1D, 2D, 3D
  - significant input from Satoshi Matsuoka and others at Tokyo Institute of Technology
  - has almost all of the variations found in FFTW and other CPU libraries?
  - nothing yet at device level?

- **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
  - contribution from István Reguly (Oxford)

- **cuRAND**
  - random number generation
  - XORWOW, mrg32k3a, Mersenne Twister and Philox_4x32_10 pseudo-random generators
  - Sobol quasi-random generator (with optimal scrambling)
  - uniform, Normal, log-Normal, Poisson outputs
  - includes device level routines for RNG within user kernels

- **cuSOLVER**:
  - key LAPACK dense solvers, 3 – 6x faster than MKL
  - sparse direct solvers, 2–14x faster than CPU equivalents

CUDA libraries

- Like cuBLAS, it is a set of 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 SDK
CUDA libraries

**CUB**
- provides a collection of basic building blocks at three levels: device, thread block, warp
- functions include sort, scan, reduction
- Thrust uses CUB for CUDA version of key algorithms

**AmgX (originally named NVAMG)**
- library for algebraic multigrid

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

CUDA 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**

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://github.com/kokkos/kokkos/wiki) [https://trilinos.org/packages/kokkos/](https://trilinos.org/packages/kokkos/)
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 in CUDA SDK
  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
  - developed by Jack Dongarra, Jim Demmel and others

Other libraries

- **ArrayFire from Accelereyes**:
  - 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.accelereyes.com/products/arrayfire

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.**

NVIDIA maintains webpages with links to a variety of CUDA libraries:
developer.nvidia.com/gpu-accelerated-libraries
and other tools:
devveloper.nvidia.com/tools-ecosystem
The 7 dwarfs

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

Dense linear algebra

- cuBLAS
- cuSOLVER
- MAGMA
- ArrayFire

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 project at University of Florida (Tim Davis)
  - project at RAL (Jennifer Scott & Jonathan Hogg)

Spectral methods

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

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

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

Monte Carlo

- NVIDIA cuRAND library
- Accelereyes ArrayFire library
- some examples in CUDA SDK distribution
- nothing else needed except for more output distributions?

May be other work I’m not aware of
Tools

Debugging:

- `cuda-memcheck` detects array out-of-bounds errors, and mis-aligned device memory accesses – very useful because such errors can be tough to track down otherwise.
- `cuda-memcheck --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.
- `cuda-memcheck --tool initcheck` detects reading of uninitialised device memory.

Other languages:

- FORTRAN: PGI (Portland Group) CUDA FORTRAN compiler with natural FORTRAN equivalent to CUDA C; also IBM FORTRAN XL for new DoE systems.
- MATLAB: can call kernels directly, or use OOP like Thrust to define MATLAB objects which live on the GPU.
- Mathematica: similar to MATLAB?
- Python: [http://mathema.tician.de/software/pycuda](http://mathema.tician.de/software/pycuda)
- Haskell: [https://hackage.haskell.org/package/cuda](https://hackage.haskell.org/package/cuda)

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/](https://www.openacc.org/)

OpenMP 4.0 is similar but newer:

- strongly pushed by Intel to accommodate Xeon Phi and unify things, in some sense.

Integrated Development Environments (IDE):

- these come with editor, debugger, profiler integration.
**Tools**

NVIDIA Visual Profiler `nvprof`:
- 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
- can also obtain instruction counts from command line: `nvprof --metrics "flops_sp,flops_dp" prac2`
  ```
  do nvprof --help for more info on other options
  ```

**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 eco-system is part of why I think CUDA will remain more used than OpenCL for HPC