

CUDA Libraries & OpenCL

Albert García García

agarcia @ dtic.ua.es

Contents

CUDA libraries

OpenCL

Thrust

Objectives

- Know the existing CUDA libraries
 - Math, Vision, Helpers,
 - Scope and lifetime
- OpenCL
 - CUDA equivalents of OpenCL functions
- Thrust C++ template library
 - Common data types & functions

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
 - error functions and their inverses
 - 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 various exponential and log functions
- Includes matrix-vector and matrix-matrix product
- It is possible to call cuBLAS routines from user kernels
- Support for using CUDA streams to do a large number of small tasks concurrently
- cuBLAS is a set of routines to be called by user host code:
 - memory allocation
 - Data copying CPU-GPU
 - Error reporting
 - Matrix-matrix product
 - Matrix-vector product
- simpleCUBLAS example in SDK is a good example code to start using it

CUDA Libraries

cuFFT

- Fast Fourier Transform
- 1D, 2D, 3D
- Similar to FFTW and other CPU libraries
- Like cuBLAS, it is a set of routines called by user host code:
 - 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

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

cuRAND

- Random number generation
- XORWOW, mrg32k3a, Mersenne Twister and Philox 4x32 10 pseudo-random generators
- Uniform, Normal, log-Normal, Poisson outputs
- Sobol quasi-random generator (with optimal scrambling)

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

cuDNN

- library for Deep Neural Networks
- Used by most DL libraries: Tensorflow, Caffee, Torch, ...

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

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

<http://developer.nvidia.com/gpu-accelerated-libraries>

and other tools:

<http://developer.nvidia.com/tools-ecosystem>

Applications

Libraries

Easy to use
Most Performance

Compiler
Directives

Easy to use
Portable code

Programming
Languages

Most Performance
Most Flexibility

Tools

Other languages:

- **Fortran**: PGI (Portland Group) 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
- **Mathematica**: similar to MATLAB
- **Python**: <http://mathematician.de/software/pycuda>
- **C#**
- **R**: <http://www.fuzzyl.com/products/gpu-analytics/>
- **Java**
- ...

Tools

OpenACC

- 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/>
- Similar to OpenMP pragmas

OpenMP 4.0:

- strongly pushed by Intel to accommodate Xeon Phi and unify things, in some sense
- <http://on-demand.gputechconf.com/gtc/2016/presentation/s6510-jeff-larkin-targeting-gpus-openmp.pdf>

OpenCL

- Cross platform for parallel computing on heterogeneous devices
- Ensure proper execution but not max performance on different devices
- There are OpenCL implementations for NVIDIA, AMD GPUs, x86 CPUs, and other devices (FPGAs).
- Support for multiple devices from different manufacturers

OpenCL

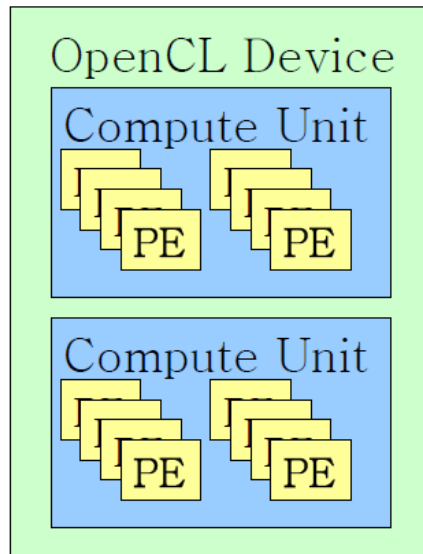
- OpenCV - CUDA concepts

OpenCL Parallelism Concept	CUDA Equivalent
kernel	kernel
host program	host program
NDRange (index space)	grid
work item	thread
work group	block

OpenCL

– OpenCL Hardware abstraction

- Handles multi-core CPUs, GPUs, and other accelerators as devices
- Each device contains one or more 'Computing Units'. (SMs)
- Each 'Computing Unit' contains one or more SIMD processing elements



OpenCL

– OpenCL - CUDA - Memories

OpenCL Memory Types	CUDA Equivalent
global memory	global memory
constant memory	constant memory
local memory	shared memory
private memory	registers

OpenCL

- The code running on accelerator devices is analogous to CUDA kernels
- ```
__kernel void vadd(__global const float *a,
__global const float *b, __global float
*result){
 int id = get_global_id(0);
 result[id] = a[id] + b[id];
}
```
- As in CUDA, each OpenCL work unit gets its own indexes



# OpenCL

- List of OpenCL - CUDA dimensions and indexes

| OpenCL API Call                  | Explanation                                                           | CUDA Equivalent                                 |
|----------------------------------|-----------------------------------------------------------------------|-------------------------------------------------|
| <code>get_global_id(0);</code>   | global index of the work item in the x dimension                      | <code>blockIdx.x*blockDim.x+ threadIdx.x</code> |
| <code>get_local_id(0)</code>     | local index of the work item within the work group in the x dimension | <code>threadIdx.x</code>                        |
| <code>get_global_size(0);</code> | size of NDRange in the x dimension                                    | <code>gridDim.x × blockDim.x</code>             |
| <code>get_local_size(0);</code>  | Size of each work group in the x dimension                            | <code>blockDim.x</code>                         |

# OpenCL

## – Conclusions

- OpenCL intrinsically evolves slower than CUDA
- Although OpenCL is correct in terms of execution, the performance of one kernel is not guaranteed on different devices
- Application programming in OpenCL requires more testing than in the CUDA Runtime API
- In the future it should become the standard for GPU programming

# Thrust

- Thrust is a C++ template library for CUDA based on the Standard Template Library (STL)
- Minimal programming effort through a high-level interface that is fully interoperable with CUDA C

```
int main(void)
{
 // generate random data on the host
 thrust::host_vector<int> h_vec(100);
 thrust::generate(h_vec.begin(), h_vec.end(), rand);

 // transfer to device and compute sum
 thrust::device_vector<int> d_vec = h_vec;
 int x = thrust::reduce(d_vec.begin(), d_vec.end(), 0, thrust::plus<int>());
 return 0;
}
```

## Data Structures

- `thrust::device_vector`
- `thrust::host_vector`
- `thrust::device_ptr`
- Etc.

## Algorithms

- `thrust::sort`
- `thrust::reduce`
- `thrust::exclusive_scan`
- Etc.



© NVIDIA Corporation 2011

# Thrust

- Data types:
  - Host:
    - `thrust::host_vector<int> H(4);`
  - Device:
    - `thrust::device_vector<int> D(5)`

# Thrust

- Algorithms:

- Transformations

- // fill Z with twos

- `thrust::fill(Z.begin(), Z.end(), 2);`

- // compute  $Y = X \bmod 2$

- `thrust::transform(X.begin(), X.end(), Z.begin(), Y.begin(), thrust::modulus<int>());`

- Reductions

- Prefix-Sums

- Reordering

- Sorting

# Thrust

- Algorithms:
  - Transformations
  - Reductions
    - `int sum = thrust::reduce(D.begin(), D.end(), (int) 0, thrust::plus<int>());`
  - Prefix-Sums
  - Reordering
  - Sorting

# Thrust

- Algorithms:
  - Transformations
  - Reductions
  - Prefix-Sums
    - `thrust::inclusive_scan(data, data + 6, data);`
    - `thrust::exclusive_scan(data, data + 6, data);`
  - Reordering
  - Sorting

# Thrust

- Algoritmos:
  - Transformations
  - Reductions
  - Prefix-Sums
  - Reordering
    - `copy_if` : copies only the items that meet the condition
    - `partition` : sort the elements according to a predicate (true first)
    - `remove`, `remove_if` : removes items that do not meet the condition
    - `unique`: removes consecutive duplicates from a collection
  - Sorting



# Thrust

- Algoritmos:
  - Transformations
  - Reductions
  - Prefix-Sums
  - Reordering
  - Sorting
    - `thrust::sort(A, A + N);`
    - `thrust::sort_by_key(keys, keys + N, values);`

# Thrust

- Iterators: they are useful structures to operate on data and algorithms

- `constant_iterator`

- `thrust::constant_iterator<int> first(10);`  
`thrust::constant_iterator<int> last = first + 3;`

- `first[0] // returns 10`  
`first[1] // returns 10`  
`first[100] // returns 10`

- `counting_iterator`
  - `transform_iterator`
  - `permutation_iterator`
  - `zip_iterator`

# Thrust

- Iterators: they are useful structures to operate on data and algorithms

- `constant_iterator`

- `counting_iterator`

- `thrust::counting_iterator<int> first(10);`  
`thrust::counting_iterator<int> last = first + 3;`

- `first[0] // returns 10`

- `first[1] // returns 11`

- `first[100] // returns 110`

- `transform_iterator`

- `permutation_iterator`

- `zip_iterator`

# Thrust

- Iterators: they are useful structures to operate on data and algorithms
  - constant\_iterator
  - counting\_iterator
  - transform\_iterator
    - ```
thrust::device_vector<int> vec(3);  
vec[0] = 10; vec[1] = 20; vec[2] = 30;  
... first = thrust::make_transform_iterator(vec.begin(), negate<int>());  
... last  = thrust::make_transform_iterator(vec.end(),  negate<int>());  
first[0]  // returns -10  
first[1]  // returns -20  
first[2]  // returns -30
```
 - permutation_iterator
 - zip_iterator

Thrust

- Iteradores: son estructuras útiles para operar sobre los datos y los algoritmos

- `constant_iterator`
- `counting_iterator`
- `transform_iterator`
- `zip_iterator`

```
A[0] = 10; A[1] = 20; A[2] = 30;
B[0] = 'x'; B[1] = 'y'; B[2] = 'z';
first = thrust::make_zip_iterator(thrust::make_tuple(A.begin(), B.begin()));
last = thrust::make_zip_iterator(thrust::make_tuple(A.end(), B.end()));
first[0] // returns tuple(10, 'x')
first[1] // returns tuple(20, 'y')
first[2] // returns tuple(30, 'z')
// maximum of [first, last)
thrust::maximum< tuple<int,char> > binary_op;
thrust::tuple<int,char> init = first[0];
thrust::reduce(first, last, init, binary_op); // returns tuple(30, 'z')
```

<https://github.com/thrust/thrust/wiki/Quick-Start-Guide>

```

#include <thrust/host_vector.h>
#include <thrust/device_vector.h>

#include <thrust/copy.h>
#include <thrust/fill.h>
#include <thrust/sequence.h>

#include <iostream>

int main(void)
{
    // initialize all ten integers of a device_vector to 1
    thrust::device_vector<int> D(10, 1);

    // set the first seven elements of a vector to 9
    thrust::fill(D.begin(), D.begin() + 7, 9);

    // initialize a host_vector with the first five elements of D
    thrust::host_vector<int> H(D.begin(), D.begin() + 5);

    // set the elements of H to 0, 1, 2, 3, ...
    thrust::sequence(H.begin(), H.end());

    // copy all of H back to the beginning of D
    thrust::copy(H.begin(), H.end(), D.begin());

    // print D
    for(int i = 0; i < D.size(); i++)
        std::cout << "D[" << i << "] = " << D[i] << std::endl;

    return 0;
}

```

Where To From Here?

- NVIDIA developer blog
 - [How to Implement Performance Metrics in CUDA C++](#)
 - [How to Query Device Properties and Handle Errors in CUDA C++](#)
 - [How to Optimize Data Transfers in CUDA C++](#)
 - [How to Overlap Data Transfers in CUDA C++](#)
 - [How to Access Global Memory Efficiently in CUDA C++](#)
 - [Using Shared Memory in CUDA C++](#)
 - [An Efficient Matrix Transpose in CUDA C++](#)
 - [Finite Difference Methods in CUDA C++, Part 1](#)
 - [Finite Difference Methods in CUDA C++, Part 2](#)
- Online courses on CUDA programming
 - <https://www.udacity.com/course/intro-to-parallel-programming--cs344>
- Parallel for all blog (Mark Harris, NVIDIA)
 - <https://devblogs.nvidia.com/>

CUDA Libraries & OpenCL

Thanks for your attention!

These slides have been modified/remixed using the TeachingKit licensed by NVIDIA and the University of Illinois under the [Creative Commons Attribution-NonCommercial 4.0 International License](https://creativecommons.org/licenses/by-nc/4.0/).



Albert García García

agarcia @ dtic.ua.es