Master Educational Program
“Information technology in applications”

Mathematical computations with GPUs

Using GPUs for mathematical problems in Fortran, Python, Java and C#

Alexey A. Romanenko
arom@nsu.ru
Novosibirsk State University

CUDA Fortran

* Reflection of CUDA C to FORTRAN
* All operations supported by
  * Basic FORTRAN syntax
  * FORTRAN extension
  * Runtime API
    * `use cudafor`

CUDA FORTRAN: memory allocation

* Variable definition:

```
real, device,  allocatable :: foo(:)
real, allocatable :: bar(:)
    attributes (device) :: bar
```

* Allocate/free

```
allocate( foo(1:n), bar )
deallocate( foo )
err = cudaMalloc( bar, n )
err = cudaFree( bar )
```

CUDA FORTRAN: memory copying

```
real, device, allocatable :: da(:)
real, allocatable :: ha(:)
integer :: n
...
da(1:n) = ha(1:n)
...
err = cudaMemcpy(ha, da, n)
```

CUDA FORTRAN: launching kernel

```
type(dim3) :: grid, block
...
grid = dim3(256, 1, 1)
block = dim3(512, 1, 1)
...
call kernel<<<grid, block>>>( larams )
```

Kernel is launched asynchronously!

CUDA FORTRAN: kernel

```
attributes(global) subroutine cuj ( a, newa, n, m, w0, w1, w2, cc )
 real, value :: w0, w1, w2
...
 real, shared :: reduce(256)
 j = (blockidx%y-1)*blockdim%y + threadidx%y + 1
 i = (blockidx%x-1)*blockdim%x + threadidx%x + 1

 if( j < n .and. i < m )then
    newa(i,j) = w0 * a(i,j) + &
        w1 * (a(i-1,j) + a(i,j-1) + a(i+1,j) + a(i,j+1) ) + &
        w2 * (a(i-1,j-1) + a(i-1,j+1) + a(i+1,j-1) + a(i+1,j+1) )
    mychange = max( mychange, abs( newa(i,j) - a(i,j) ) )
 endif
 ir = (threadidx%y-1) * blockdim%x + threadidx%x
 reduce(ir) = mychange
 call syncthreads()
...
end subroutine
```

CUDA FORTRAN

* Compiler oprions
  * `pgfortran -O3 -Mcuda=cc60 -ta=nvidia:14.0 -o test test.f90`
* Links
  * PGI CUDA Fortran Compiler [https://www.pgroup.com/resources/cudafortran.htm]

CUDA C/Fortran vs. OpenACC

* **CUDA C/Fortran:**
* + High performance;
* + incremental development;
* - CUDA-platform only;
* - Two versions of code (parallel + sequential).
* **OpenACC:**
* + High performance possible;
* + incremental development;
* + compatible with non-CUDA platform;
* + Only one version of code;
* - It's difficult to control compiler;
* - No free compiler available.

Call of CUDA kernel

* Call of CUDA-C-kernel from CUDA Fortran

```
interface
  attributes(global) subroutine saxpy(a,x,y,n) bind(c)
    real, device :: x(*), y(*)
    real, value  :: a
    integer, value :: n
  end subroutine
end interface
call saxpy<<<grid,block>>>(aa,xx,yy,nn)
```

* Call of CUDA-Fortran-kernel from CUDA C

```
extern __global__ void saxpy_( float a, float* x, float* y, int n );
saxpy_<<<grid,block>>>( a, x, y, n );


attributes(global) subroutine saxpy(a,x,y,n)
  real, value :: a
  real :: x(*), y(*)
  integer, value :: n
```

Call of CUDA kernels from FORTRAN

CUDA C:

```
__global__ kernel(аргументы){
    ....
}
void some_function_(аргументы){
    ....
    kernel <<<GS, BS>>>(аргументы);
    ....
}
```

Fortran:

```
....
call some_function(аргументы);
....
```

GPU programming Python

* Numba
  * JIT compiler for Python
  * Nvidia CUDA. Experimental on AMD ROC
  * https://numba.pydata.org/
* CuPy
  * CuPy is a GPU array backend that implements a subset of NumPy interface
  * https://docs.cupy.dev/

GPU programming Python

	CuPy	PyCUDA	PyTorch	JAX
NVIDIA CUDA support	X	X	X	X
CPU/GPU agnostic coding	X		X	X
Autograd support			X	X
NumPy compatible interface	X			X
Used-defined CUDA kernels	X	X		

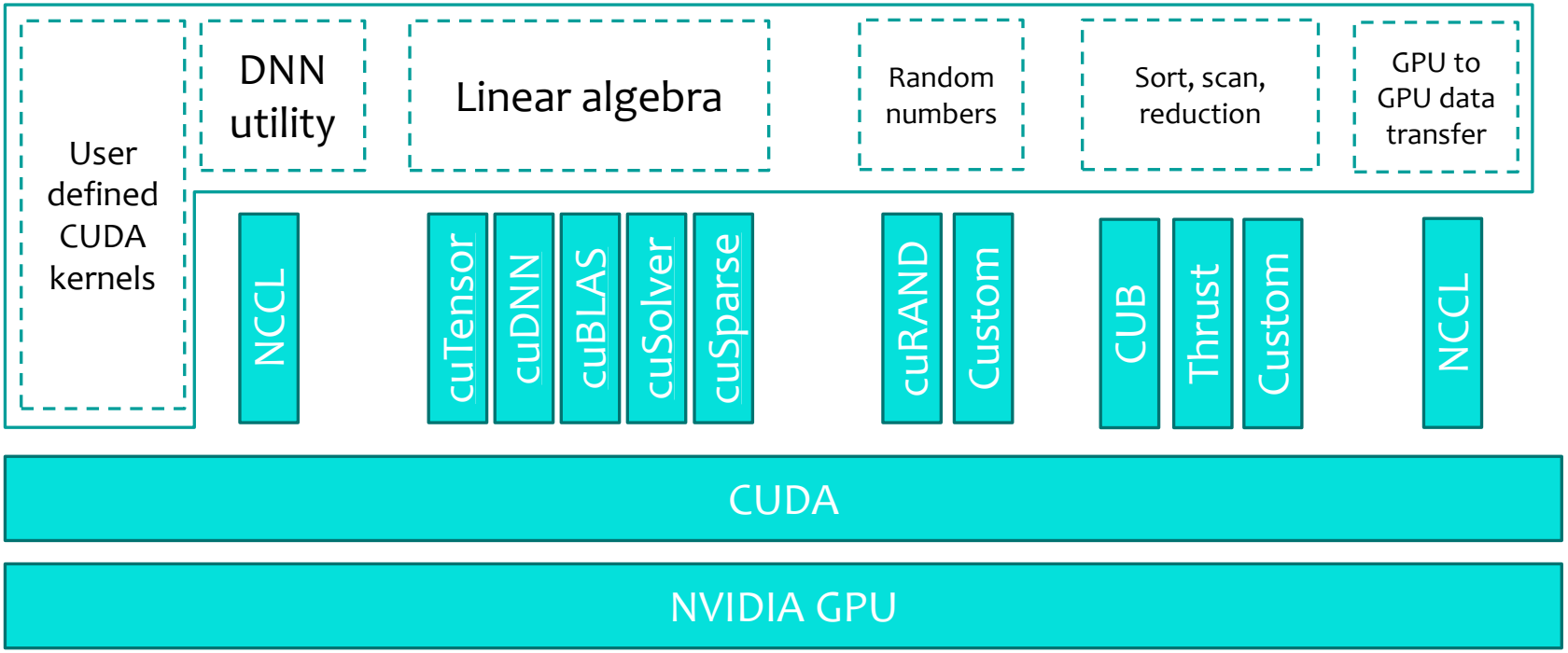
NumPy and CuPy

```
import numpy as np
X_cpu = np.zeros((10,))
W_cpu = np.zeros((10,5))
y_cpu = np.dot(X_cpu,W_cpu)
```

```
import cupy as cp
X_gpu = cp.zeros((10,))
W_gpu = cp.zeros((10,5))
y_gpu = cp.dot(X_cpu,W_cpu)
```

```
for xp in [np, cp]
X = xp.zeros((10,))
W = xp.zeros((10,5))
y = xp.dot(X_cpu,W_cpu)
```

CuPy Architecture



Calling library function from Python

* C
  * Native support (ctype)
  * CFFI
* C++
  * pybind11

Python. ctype

```
// test.c
#include <stdio.h>
int func_ret_int(int val) {
        printf("get func_ret_int: %d\n", val);
        return val;
}

//compile
gcc -fPIC -shared -o libtest.so test.c
```

Python. ctype. Continue

```
import ctypes # load library
test = ctypes.CDLL('./objs/libtest.so’)

# our function returns int
test.func_ret_int.restype = ctypes.c_int
# our function gets its argumet as an int
test.func_ret_int.argtypes = [ctypes.c_int, ]
print('ret func_ret_int: ', test.func_ret_int(101))
```

GPU programming in Java

* Aparapi - Open-source framework for executing native Java code on the GPU
  * Java код → OpenCL (OpenCL 1.2, OpenCL 2.0, and OpenCL 2.1)
  * https://aparapi.com/
* JOCL - Java bindings for OpenCL
  * AMD, NVIDIA GPUs
  * http://www.jocl.org/

Aparapi (example)

```
final float inA[] = ....  // get a float array of data from somewhere
final float inB[] = ....  // get a float array of data from somewhere
                          // (inA.length==inB.length)
final float result = new float[inA.length];

for (int i=0; i<array.length; i++){
    result[i]=intA[i]+inB[i];
}
```

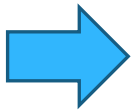
```
Kernel kernel = new Kernel(){
   @Override public void run(){
      int i= getGlobalId();
      result[i]=intA[i]+inB[i];
   }
};
Range range = Range.create(result.length);
kernel.execute(range);
```

GPU programming in Java

* jcuda.org
  * Wrappers for CUDA runtime API and driver API
  * Kernel on CUDA C
  * http://www.jcuda.org/
* jocl.org
  * Wrappers for OpenCL API
  * http://www.jocl.org/
* Lighrweight Java Game Library (LWJGL)
  * Wrappers for OpenCL API
  * http://www.lwjgl.org
* etc.

```
import java.lang.*;
public class Hello {
        static { System.loadLibrary("Hello"); }
        public static native String getMessage();
        public static void main( String[] args ) {
                System.out.println( getMessage() );
                System.exit(0);
        }
}
```

javac Hello.java
javah Hello



```
//Hello.h
#include <jni.h>
#ifndef _Included_Hello
#define _Included_Hello
#ifdef __cplusplus
extern "C" {
#endif
JNIEXPORT jstring JNICALL
        Java_Hello_getMessage (JNIEnv *, jclass);
#ifdef __cplusplus
}
#endif
#endif
```

http://xyplot.com/jni.simple.htm

GPU programming in C#

* CUDAfy.net
  * https://github.com/lepoco/CUDAfy.NET
* ILGPU
  * ILGPU is a JIT compiler for GPU programs (also known as kernels) written in .Net-based languages
  * https://www.ilgpu.net/
* Hybridizer (CUDA.NET successor) - http://www.altimesh.com/
* CAMPY - http://campynet.com/

DLL:

```
#include <thrust/device_vector.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <thrust/detail/type_traits.h>

extern "C" __declspec(dllexport) void __cdecl GPUSort(int*, unsigned int);
extern void GPUSort(int* data, unsigned int numElements) {
        thrust::device_vector<int> d_data(data, data + numElements);
        thrust::stable_sort(d_data.begin(), d_data.end());
        thrust::copy(d_data.begin(), d_data.end(), data);
}
```

C#:

```
[DllImport("GPUSort.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void GPUSort(
        [MarshalAsAttribute(UnmanagedType.LPArray,
                            ArraySubType = UnmanagedType.I4)]
        int[] data, uint numElements);
```