Master Educational Program
“Information technology in applications”

Mathematical computations with GPUs

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

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

FORTRAN

* Special directives
  * OpenACC
* Compilers with support of CUDA Fortran
  * PGI accelerated fortran
  * CRAY
* Linking FORTRAN program with CUDA C modules

OpenACC

* Developers: **NVIDIA, PGI, CRAY, CAPS**
* Compiler directives. Looks like OpenMP
* API
* Languages: C/Fortran
* Documentation:
  * http://www.openacc-standard.org
  * Programmer's guide. PGI compiler

OpenACC: execution model

* Host
  * Execute major part of the code;
  * Allocate/free memory on GPU;
  * Manage data transfer operation;
  * Launch kernels and synchronize streams;
* GPU
  * Execute kernels.

OpenACC: directives

* **Fortran**

```
!$acc directive [clause [, clause] …]
     structured block
!$acc end directive
```

* **C**

```
#pragma acc directive [clause [, clause] …]
     structured block
```

* **Compiling:**

```
pgfortran -acc -Minfo=accel -ta=nvidia <file_name>
pgcc -acc -Minfo=accel -ta=nvidia <file_name>
```

OpenACC: directives

* **directives**
  * parallel
  * loop
  * kernels
  * etc.
* **clauses**
  * if(condition)
  * async[(exp)]
  * num_gangs(exp)
  * num_workers(exp)
  * vector_length(exp)
  * reduction(operator:list)
  * etc.

* **Data clauses**
  * copy(list)
  * copyin(list)
  * copyout(list)
  * create(list)
  * present(list)
  * present_or_copy(list)
  * present_or_copyin(list)
  * present_or_copyout(list)
  * present_or_create(list)
  * deviceptr(list)
  * private(list)
  * firstprivate(list)

OpenACC. resourses

* Standard
  * http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf
* Recommendations
  * http://www.nvidia.com/docs/IO/117377/directives-tips-for-fortran.pdf
  * http://www.nvidia.com/docs/IO/117377/directives-tips-for-c.pdf
* Quick Reference card
  * http://www.openacc.org/sites/default/files/OpenACC_API_QuickRefGuide.pdf

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>>>( параметры )
```

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 -fast -O3 -Minfo=all -Mcuda=cc13 -ta=nvidia:4.0 -o test test.f90`
* Links
  * PGI CUDA Fortran Compiler [http://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 in Java

* AMD Aparapi
  * AMD cards only
  * Java код → OpenCL
  * http://developer.amd.com/zones/java/aparapi/Pages/default.aspx
* java-gpu
  * CUDA supported cards
  * http://code.google.com/p/java-gpu/

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);
```

java-gpu (example)

```
@Parallel(loops = {"y", "x"})
  public void compute() {
    for(int y = 0; y < height; y++) {
      for(int x = 0; x < width; x++) {
        float Zr = 0.0f;
        float Zi = 0.0f;
        …
        data[y][x] = (short)((i * 255) / iterations);
      }
    }
  }
```

```
java -jar Parallel.jar samples.Mandelbrot
```

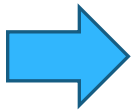
```
java -cp .:Parallel.jar samples.Mandelbrot 2000 2000 out.png
```

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
* JavaCL
  * Wrappers for OpenCL API
  * Support AMD and NVidia cards
  * http://code.google.com/p/javacl/
* 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#

* CUDA.NET
  * Wrappers for CUDA runtime API
  * Web-page removed
* GPU.NET
  * Directives for C#
  * CUDA supported device only
  * Project closed
* Accelerator v2
  * Research project from Microsoft
  * Web page modified in 2011
  * http://research.microsoft.com/en-us/projects/Accelerator/
* Cloo
  * Wrappers for OpenCL
  * http://cloo.sourceforge.net/

GPU programming in C#

* CUDAfy
  * http://www.hybriddsp.com/Products/CUDAfyNET.aspx
  * Developed intensively
    * Dynamic parallelism
    * Direct GPU programming not required
  * Examples: http://gpuscience.com/code-examples/cudafy-me-traveling-salesman-problem-with-cuda-from-c/
* NMath – libraries set
  * http://www.centerspace.net
* Hybridizer successor for CUDA.NET
  * http://www.altimesh.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);
```