Introduction to Scientific Programming using GPGPU and CUDA

Day 2

Sergio Orlandini
s.orlandini@cineca.it

Mario Tacconi
m.tacconi@cineca.it
- Tools from CUDA-Toolkit
  - Profiler
  - CUDA-GDB
  - CUDA-memcheck
  - Parallel Nsight

- CUDA-Enabled Libraries
  - CUBLAS
  - CUFFT
  - CUSPARSE
  - CURAND
  - MAGMA, THRUST, CUDDP, ...

- Hands on
Profiling tools: built-in

- CUDA toolkit provides useful tools for profiling your code

```bash
export CUDA_PROFILE=1
export CUDA_PROFILE_CONFIG=$HOME/.config
```

// Contents of config

gld_coherent
gld_incoherent
gst_coherent
gst_incoherent

gld_incoherent: Number of non-coalesced global memory loads
gld_coherent: Number of coalesced global memory loads
gst_incoherent: Number of non-coalesced global memory stores
gst_coherent: Number of coalesced global memory stores
local_load: Number of local memory loads
local_store: Number of local memory stores
branch: Number of branch events taken by threads
divergent_branch: Number of divergent branches within a warp
instructions: instruction count
warp_serialize: Number of threads in a warp that serialize based on address conflicts to shared or constant memory
cta_launched: executed thread blocks

```bash
method,gputime,cputime,occupancy,gld_incoherent,gld_coherent,gst_incoherent,gst_coherent
method=[ memcopy ] gputime=[ 438.432 ]
gld_incoherent=[ 0 ] gld_coherent=[ 1952 ] gst_incoherent=[ 62464 ] gst_coherent=[ 0 ]
method=[ memcopy ] gputime=[ 349.344 ]
... ...
```
Profiling: Visual Profiler

- Traces hosts and device calls, data transfers, kernel launches, shows overlapping streams and measure performances
- supports automated analysis (hardware counters)
CUDA Toolkit also provides a cuda-gdb text debugger

- the traditional gdb enhanced with CUDA extensions

```
(cuda-gdb) info cuda threads
BlockIdx ThreadIdx To BlockIdx ThreadIdx Count Virtual PC Filename Line
Kernel 0* (0,0,0) (0,0,0) (0,0,0) (255,0,0) 256 0x000000000866400
bitreverse.cu 9
(cuda-gdb) thread
[Current thread is 1 (process 16738)]
(cuda-gdb) thread 1
[Switching to thread 1 (process 16738)]
#0 0x000019d5 in main () at bitreverse.cu:34
34 bitreverse<<<1, N, N*sizeof(int)>>>(d);
(cuda-gdb) backtrace
#0 0x000019d5 in main () at bitreverse.cu:34
(cuda-gdb) info cuda kernels
Kernel Dev Grid SMs Mask GridDim BlockDim Name Args
0 0 1 0x00000001 (1,1,1) (256,1,1) bitreverse data=0x110000
```
### Debugging: CUDA-MEMCHECK

- Very useful to find out-of-bound, misaligned global memory accesses
- Stand alone or integrated in CUDA-GDB

```
$ cuda-memcheck --continue ./memcheck_demo
======== CUDA-MEMCHECK
Mallocing memory
Running unaligned_kernel
Ran unaligned_kernel: no error
Sync: no error
Running out_of_bounds_kernel
Ran out_of_bounds_kernel: no error
Sync: no error
======== Invalid __global__ write of size 4
======== at 0x00000038 in memcheck_demo.cu:5:unaligned_kernel
======== by thread (0,0,0) in block (0,0,0)
======== Address 0x200200001 is misaligned
========
======== Invalid __global__ write of size 4
======== at 0x00000030 in memcheck_demo.cu:10:out_of_bounds_kernel
======== by thread (0,0,0) in block (0,0,0)
======== Address 0x87654320 is out of bounds
========
======== ERROR SUMMARY: 2 errors
```
Parallel NSight

- Plug-in available for Eclipse and Microsoft Visual Studio IDE
- Aggregates all external functionalities:
  - Debugger (fully integrated)
  - Visual Profiler
  - Memory correctness checker
- As a plug-in, it extends all the convenience of IDE development to CUDA
- Very fast growing community of users and developers
- A number of very nice (and useful) features:
  - supports multi-GPU
  - remote debug and profiling
  - PTX assembly view
  - warp inspector
Parallel NSight
Parallel NSight
CUDA Enabled Libraries

CUDA Toolkit includes many useful libraries:

- **CUBLAS**: Basic Linear Algebra Subprograms
- **NVBLAS**: multi-GPUs accelerated drop-in BLAS built on top of cuBLAS
- **CUFFT**: Fast Fourier Transform
- **CUSPARSE**: sparse matrix linear algebra
- **CURAND**: pseudorandom and quasirandom number generator
- **NPP**: image, signal processing, statistic (nVIDIA Performance Primitives)
- **THRUST**: vector-based library for parallel algorithms in C++

Other CUDA enabled libraries outside the CUDA Toolkit:

- **MAGMA** (Matrix Algebra on GPU and Multicore Architectures)
- **CUDPP** (Data Parallel Primitives): parallel prefix-sum, sort, reduction
- **CULA**: CUDA LAPACK API (single precision version is freely available)
- **CUSP**: CUDA Sparse Solver and graph:

For a complete list, visit CUDA-Zone and look for GPU-Accelerated Libraries
CUDA Math Library

- Of course, CUDA Toolkit provides mathematical functions as other higher level language
  - by simply adding “#include math.h” in your source code
  - Complete support for all C99 standard float and double math functions

- IEEE-754 accurate for float, double, and all rounding modes
  - Extended Trigonometry and Exponential Functions
    - cospi, sincos, sinpi, exp10
  - Inverse Error Functions (erfinv, erfcinv)
  - Optimized Reciprocal Functions (rsqrt, rcbrt)
  - Bessel Functions (j0, j1, jn, y0, y1, yn)

- Floating Point Data Attributes (signbit, isfinite, isinf, isnan)
- intrinsic versions are also provided
cuBLAS Library

- BLAS is a standard in terms of interface and accuracy for most other libraries which implements linear algebra operations
  
  - BLAS Level 1:
  - BLAS Level 2:
  - BLAS Level 3:

- There are vendor optimized implementation for many hardware architecture (Intel, Power, ARM, etc)

- CUDA Toolkit provide a CUDA-enabled implementation of all BLAS

- WARNING: data layout in memory follows column-major ordering and 1-based indexing

- Function naming convention: cublas + BLAS name
Here we show a simple performance study case for DGEMM (double-precision dense matrix matrix product - old system)
CUBLAS: DGEMM performance

- Performance versus matrix size dependency
cuBLASXT

- Starting with CUDA 6.0, the cuBLAS Library exposes two API
  - the regular cuBLAS API
  - the new cuBLASXT API

- With cuBLAS API
  - the application must allocate the required matrices and vectors in the GPU memory space
  - fill them with data, call the sequence of desired cuBLAS functions,
  - then upload the results from the GPU memory space back to the host

- With cuBLASXT API
  - the application must allocate data using managed memory
  - the library will take care of dispatching the operations to one or multiple GPUs present in the system
cuFFT

- cuFFT is the CUDA version of the Fast Fourier Transform
  - based on Cooley-Turkey and Bluestein algorithm
- cuFFT API is very similar to the FFTW one
  - as FFTW does, cuFFT use the workplan concept to optimize its work
  - once a workplan is computed, the library itself maintains necessary information to execute FFT operation on data many times efficiently
  - WARNING: cuFFT follow row-major convention for data in memory

- Other key features:
  - provides 1D, 2D, 3D transform
    - for many real and complex types (single, double, quad precision)
    - in-place and out-of-place transforms
  - non-normalized output:
    - \( \text{IFFT(FFT(A))} = \text{len(A)} \times A \)
  - support for asynchronous operation on CUDA streams
  - thread-safe (CUDA 4.1)
cuFFT sample: 2D complex-complex

```c
#define NX 256
#define NY 128

cufftHandle plan;
cufftComplex *idata, *odata;
cudaMalloc((void**)&idata, sizeof(cufftComplex)*NX*NY);
cudaMalloc((void**)&odata, sizeof(cufftComplex)*NX*NY);
...

/* create a plan for FFT 2D */
cufftPlan2d(&plan, NX, NY, CUFFT_C2C);
/* use plan for “out of place” transform */
cufftExecC2C(plan, idata, odata, CUFFT_FORWARD);
/* back transform “in place” */
cufftExecC2C(plan, odata, odata, CUFFT_INVERSE);
/* if input output pointers differ, “out of place” is implied */

/* destroy plan and free resources */
cufftDestroy(plan);
cudaFree(idata), cudaFree(odata);
```
cuFFT: performances of FFT1D

- Measured on sizes that are exactly powers-of-2
- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz
Performance may vary based on OS version and motherboard configuration
cuFFT: performances of FFT3D

- cuFFT 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz

Performance may vary based on OS ver. and motherboard config.
cuSPARSE

- Support for dense, COO, CSR, CSC, ELL/HYB and Blocked CSR sparse matrix formats
- Level 1 routines for sparse vector x dense vector operations
- Level 2 routines for sparse matrix x dense vector operations
- Level 3 routines for sparse matrix x multiple dense vectors (tall matrix)
- Routines for sparse matrix by sparse matrix addition and multiplication
- Conversion routines that allow conversion between different matrix formats
- Sparse Triangular Solve
- Tri-diagonal solver
- Incomplete factorization preconditioners ilu0 and ic0
- **Flexible usage model**
  - Host API for generating random numbers in bulk on the GPU
  - Inline implementation allows use inside GPU functions/kernels, or in your host code

- **Four high-quality RNG algorithms**
  - MRG32k3a
  - MTGP Merseinne Twister
  - XORWOW pseudo-random generation
  - Sobol’ quasi-random number generators, including support for scrambled and 64-bit RNG

- **Multiple RNG distribution options**
  - Uniform distribution
  - Normal distribution
  - Log-normal distribution
  - Single-precision or double-precision
1. Create a generator:
   curandCreateGenerator()

2. Set a seed:
   curandSetPseudoRandomGeneratorSeed()

3. Generate the data from a distribution:
   curandGenerateUniform()/curandGenerateUniformDouble() : Uniform
   curandGenerateNormal()/cuRandGenerateNormalDouble() : Gaussian
   curandGenerateLogNormal/curandGenerateLogNormalDouble() : Log-Normal

4. Destroy the generator:
   curandDestroyGenerator()
```c
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand.h>

int main() {

int i, n = 100;
curandGenerator_t gen;
float *devData, *hostData;

// Allocate n floats on host
hostData = (float *) calloc (n, sizeof(float));

// Allocate n floats on device
cudaMalloc((void **) &devData, n * sizeof(float));

// Create psudo-random number generator
curandCreateGenerator(&gen,
CURAND_RNG_PSEUDO_DEFAULT);

// set seed
curandSetPseudoRandomGeneratorSeed(gen, 1234ULL);

// generate n float on device
curandGenerateUniform(gen, devData, n);

// copy device memory to host
cudaMemcpy(hostData, devData, n * sizeof(float),
cudaMemcpyDeviceToHost);

// show result
for (i = 0; i < n; i++) {
    printf("%1.4f ", hostData[i]);
}
printf("\n");

// Cleanup
curandDestroyGenerator(gen);

cudaFree(devData)
free(hostData)
return 0;
}
```
```c
#include <stdio.h>
#include <stdlib.h>
#include <cuda.h>
#include <curand_kernel.h>

__global__ void setup_kernel(curandState *state)
{
    int id = threadIdx.x - blockIdx.x * 64;
    // each thread gets same seed
    curand_init(1234, id, 0, &state[id]);
}

__global__ void generate_kernel(
    curandState *state, int *result)
{
    int id = threadIdx.x + blockIdx.x * 64;
    int count = 0;
    unsigned int x;
    curandState localState = state[id];
    for (int n = 0; n < 1000000; n++) {
        x = curand(&localState);
    }
    // copy state back to global memory
    state[id] = localState;
    // store results
    result[id] += count;
}
```
CUDPP

- CUDPP: CUDA Data Parallel Primitives library
- collection of many *data-parallel* algorithms:
  - prefix-sum ("scan")
  - parallel sort
  - reduction
- Important building blocks for a wide variety of data-parallel algorithms, including sorting, stream compaction, and building data structures such as trees and summed-area tables
- provides primitives and other complex operation functions such as:
  - hash table
  - array compaction
  - tridiagonal linear system solver
  - sparse matrix-vector product
- Specifications
  - open source project in C/C++
  - Support for Windows, Linux and OSX
  - open source: http://cudpp.github.io/
MAGMA: Matrix Algebra on GPU and Multicore Architectures

- LAPACK (Linear Algebra PACKage) is the de facto standard linear algebra operations
  - built on BLAS

- MAGMA is essentially a re-implementation of standard legacy LAPACK on heterogeneous architectures such as GPU + multicore CPU
  - built on top of cuBLAS

- MAGMA 1.x support multi-GPU CUDA enabled environment, and its able to overlap computation on CPU cores (essentially through optimized multithreaded version of BLAS and LAPACK for the CPU side)

- Developed by the ICL group (Innovative Computing Laboratory) + many external collaborations + users community

- open source: http://icl.cs.utk.edu/projectsfiles/magma/

- WARNING: memory data layout follow the FORTRAN (column-major) convention
MAGMA: C/C++ usage

- MAGMA is entirely developed in C. So it’s usage is very easy in a C/c++ code
- The library interface is just in one file:
  - magma.h
- user must explicitly manage memory on host and device using traditional CUDA runtime APIs

```c
// Reduction of a symmetrix matrix into tridiagonal form
#include <cuda.h>  //
#include <magma.h> //

// magma_int_t magma_dsytrd( char uplo, magma_int_t n, double *A,
//      magma_int_t lda, double *d, double *e,
//      double *tau, double *work, magma_int_t *lwork,
//      double *da, double *dc, magma_int_t *info);

cudaError_t stat;
double *da, *dwork;
stat = cudaMalloc((void**)&da, n*n*sizeof(double));
stat = cudaMalloc((void**)&dwork, workSize* sizeof(double));
magma_dsytrd(‘U’, n, A, lda, diagonal, offdiagonal, tau, work, lwork, da, dwork,
    &info)
```
MAGMA: F90/2003 usage

- In order to use MAGMA with F90/2003 requires the programmer to provide interface and the ISO_C_BINDING module.

```cpp
!! Native C interface:
!! magma_int_t magma_dsytrd( char uplo, magma_int_t n, double *A,
!!     magma_int_t lda, double *d, double *e,
!!     double *tau, double *work, magma_int_t *lwork,
!!     double *da, double *dc, magma_int_t *info);
!! Interface for F90/2003:
subroutine magma_dsytrd(uplo, n, a, lda, d, e, tau, work, lwork, da, dc, info)

bind(C, name="magma_dsytrd")
use iso_c_binding
implicit none

character, value:: uplo
integer(C_INT), value :: n, lda
integer(C_INT) :: info, lwork
type(C_PTR), value :: a, d, e, tau, work, da, dc

! NB: type(C_PTR), value == void*
end subroutine magma_dsytrd
```
CUDA Thrust

A C++ template library for CUDA

- Mimics the C++ STL

  - Two containers
    - Manage memory on host and device:
      - `thrust::host_vector<T>`
      - `thrust::device_vector<T>`

d- Algorithms

  - Sorting, reduction, scan, etc:
    - `thrust::sort()`
    - `thrust::reduce()`
    - `thrust::inclusive_scan()`

  - act on ranges of the container data by pair of iterators (a sort of pointers)
```c
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <thrust/copy.h>
#include <cstdlib>

int main(void)
{
    // generate 32M random numbers on the host
    thrust::host_vector<int> h_vec(32 << 20);
    thrust::generate(h_vec.begin(), h_vec.end(), rand);

    // transfer data to the device
    thrust::device_vector<int> d_vec = h_vec;

    // sort data on the device (846M keys per second on GeForce GTX 480)
    thrust::sort(d_vec.begin(), d_vec.end());

    // transfer data back to host
    thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

    return 0;
}
```
```cpp
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/reduce.h>
#include <thrust/functional.h>
#include <cstdlib>

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;
}
```
CUDA Thrust

Various Algorithms (32M integer samples)

- TBB
- Thrust

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>reduce</td>
<td></td>
</tr>
<tr>
<td>transform</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td></td>
</tr>
<tr>
<td>sort</td>
<td>14</td>
</tr>
</tbody>
</table>

Sort (32M integer samples)

- TBB
- Thrust

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>char</td>
<td>35</td>
</tr>
<tr>
<td>short</td>
<td>25</td>
</tr>
<tr>
<td>int</td>
<td>10</td>
</tr>
<tr>
<td>long</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>

- CUDA 4.1 on Tesla M2090, ECC on
- MKL 10.2.3, TYAN FT72-B7015 Xeon x5680 Six-Core @ 3.33 GHz
- Performance may vary based on OS ver. and motherboard config.
Lapack for CUDA: CULA Library

http://www.cula tools.com

Proprietary library that implements the LAPACK in CUDA, which is available in several versions.

The speed-up of the picture on the right refers to:

**CPU**: Quad-core Intel Core i7 930 @ 2.8 GHZ CPU

**GPU**: NVIDIA Tesla C1060

**GPU**: NVIDIA Tesla C2050
Bibliography

✓ CUDA C Programming Guide
✓ NVIDIA CUDA Library Documentation (Doxygen –generated Reference Manual)
✓ Tuning CUDA Applications for Fermi, Kepler, Maxwell, Pascal, etc
✓ Kirk and Hwu, Programming Massively Parallel Processors
✓ P. Micikevicius, Fundamental and Analysis-Driven Optimization, GPU Technology Conference 2010 (GTC 2010)
✓ V. Volkov, Benchmarking GPUs to tune dense linear algebra
✓ V. Volkov, Better performance at lower occupancy, GPU Technology Conference 2010 (GTC 2010)
✓ J. Dongarra et al. “An Improved MAGMA GEMM for Fermi GPUs”
Exercise 1: naive version

```
HOST
h_A
h_B

Copy Host To Device

DEVICE
d_A
d_B
d_C

OP =

HOST
h_C
```
Exercise 2: cudaStream version

Copy Host To Device

Copy Device To Host
Excercise 2: cudaStream version

Copy Host To Device

Copy Device To Host
Exercise 2: cudaStream version

Execution Time Lines

Sequential Version

- H2D Engine: Stream 0
- Kernel Engine
- D2H Engine

Asynchronous Versions

- H2D Engine: 1 2 3 4
- Kernel Engine: 1 2 3 4
- D2H Engine: 1 2 3 4

Time
Excercise 2: CUDA runtime functions

You need to use the following functions (C for CUDA):

- `cudaError_t cudaStreamCreate(cudaStream_t *stream)`
- `cudaError_t cudaStreamDestroy(cudaStream_t stream)`
- `cudaError_t cudaDeviceSynchronize(void)`
- `cudaError_t cudaMemcpyAsync(void* dst, void* src, size_t nbyte, enum cudaMemcpyKind kind, cudaStream_t stream)`

You need too use the following functions (CUDA FORTRAN):

- `integer function cudaStreamCreate(stream)
  integer :: stream`
- `integer function cudaStreamDestroy(stream)
  integer :: stream`
- `integer function cudaDeviceSynchronize()
  integer :: stream`
- `integer function cudaMemcpyAsync(dst, src, nelements, kind, stream)"