CUDA C/C++ Hands-On

Dr. Timo Stich
Developer Technology Group

Getting Started

- Remote Login:
  - /> ssh –X stich@mpc606.mata
  - Password: CUDAisfun

- Add cuda toolkit to PATH and LD_LIBRARY_PATH
  - `/> source add_cuda_path.sh`

- `Test:`
  - `/> nvcc --version`
  - nvcc: NVIDIA (R) Cuda compiler driver
  - Copyright (c) 2005-2011 NVIDIA Corporation
  - Built on Thu_May_12_11:09:45_PDT_2011
  - Cuda compilation tools, release 4.0, V0.2.1221

CUDA 101 - Getting Started

- CU-Files
  - CUDA C/C++ files, like .c/.cpp
  - Can contain both GPU and CPU code
- NVCC
  - Compiler frontend
  - Separates CPU and GPU code
  - GPU handled by NVIDIA compiler, CPU code passed to other compiler (GCC, Intel, MS)
- Executable
  - Single binary that contains all the code for CPU and GPU
  - Simple to run and distribute

CUDA 101 - Tasks

1. Compile & Run
   a) nvcc –arch sm_20 cuda101.cu –o cuda101
   b) ./cuda101

2. Launch helloKernel
   a) Grid: 2 Blocks x 3 Threads

3. Compute global thread index in the kernel
   a) Use special variables: blockDim.x, blockIdx.x, threadIdx.x

CUDA 101 - Tasks

4. Print Message from Kernel
   a) standard printf calls work on the GPU (sm_20 and newer)

Other Buildsystems

- Microsoft Visual Studio
  - Rules distributed with the CUDA Toolkit
- CMake
  - FindCUDA part of standard distribution (since v2.8)
- Eclipse
  - CUDA Plugin from fixstars (also includes support for cuda-gdb)

Vectoradd - Background

- Simple computation
  - $\underline{z} = a * \underline{x} + \underline{y}$
- Different ways to implement CPU – GPU memory transfer
  - Simple IO
  - Fast IO
  - Zero-Copy IO

Vectoradd - Kernel

- CUDA memory pointers work just like C memory pointers
- __global__ void kernel(float* data)
  - `float a = data[i];         // ith element`
  - `float a = *data;           // first element`
  - `float* pa = data + i;      // pointer to ith element`

Vectoradd – Simple IO

- Allocate Device Memory
  - cudaMalloc(void** pointerToGPUMem, size_t numberOfBytes);
  - `float* d_x;`
  - `cudaMalloc(&d_x, 100 * sizeof(float));`
- Copy Data Between Host and Device
  - cudaMemcpy(void* dest, void* source, size_t nBytes, cudaMemcpyDir)
  - with Dir = HostToDevice / DeviceToHost / DeviceToDevice
  - `cudaMemcpy(h_x, d_x, Nbytes, cudaMemcpyDeviceToHost);`

Vectoradd – Fast IO

- Pinned Host Memory
  - Lock CPU memory to specific Physical address (OS function)
  - Enables GPU DMA engine to do the copying -> faster + asynchronous
- cudaHostAlloc(void** pointerToCPUMem, size_t nBytes, options)

```
float* h_x;
cudaHostAlloc(&h_x, 100 * sizeof(float), cudaHostAllocMapped);
```

- (since CUDA 4.0: Also possible to pin pre-allocated memory using cudaHostRegister/cudaHostUnregister)

Vectoradd – Zero-Copy IO

- Map CPU memory into GPU memory space
  - No copying involved
  - Dereferecing pointer on GPU results in PCIe traffic

- cudaHostGetDevicePointer(void** pGPU, void* pCPU, options);

```
float* h_x, d_x;
cudaHostAlloc(&h_x, 100 * sizeof(float), cudaHostAllocMapped);
cudaHostGetDevicePointer(&d_x, h_x, 0);
```

Other Options (Tesla & Quadro)

- Unified Memory Address Space
  - CPU & GPU memory is mapped into on contiuous memory space
  - From address location is recovered at runtime
- Peer to Peer memory access
  - Dereference pointer to memory in other GPUs
  - Can be combined with UVA

Debugging

- cuda-gdb (part of the CUDA toolkit)
  - CUDA enabled gdb
  - Debug both CPU and GPU code
- cuda-memcheck (part of the CUDA toolkit)
  - Reports out of bounds and misaligned memory accesses
- Compile with debug symbols
  - nvcc –G -g minimum.cu –lcurand –o minimum

Debugging - Memcheck

- cuda-memcheck ./minimum

```
========= CUDA-MEMCHECK
========= Invalid __global__ read of size 4
=========     at 0x000002c0 in minimum.cu:18:findMinimum
=========     by thread (62,0,0) in block (0,0,0)
=========     Address 0x2003002f8 is out of bounds
=========
========= Invalid __global__ read of size 4
=========     at 0x000002c0 in minimum.cu:18:findMinimum
=========     by thread (63,0,0) in block (0,0,0)
=========     Address 0x2003002fc is out of bounds
=========
========= ERROR SUMMARY: 2 errors
```

Debugging – CUDA-GDB

- cuda-gdb ./minimum
- Useful commands:
  - set cuda memcheck on
  - break <line number or kernel/function name>
  - run
  - print <variable name>

Other Debuggers

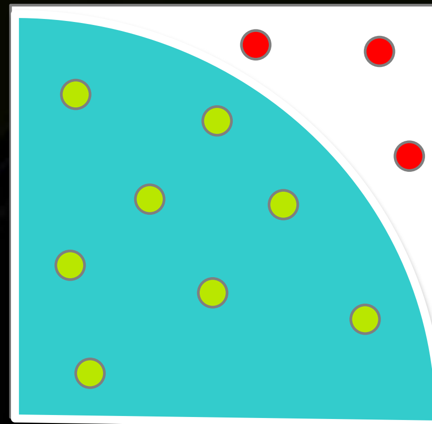
- Debugger / Profiler functionality is exposed via API
  - CUPTI API
  - CUDA Debugger API
  - /usr/local/cuda/extra/
- Other CUDA debuggers
  - Allinea Debugger
  - TotalView Debugger
  - NVIDIA Parallel Nsight (MS Visual Studio Plugin)

Libraries

- CUDA Toolkit comes with many optimized libraries
  - CUBLAS
  - CUFFT
  - CUSparse
  - CURAND (GPU + CPU)
  - Thrust (GPU + CPU)
  - NPP

Libraries

- Example: Estimate PI
  - Estimate area of quarter circle with radius 1 (mul by 4 to get total area)
  - Draw random samples from uniform [0..1] distribution
  - Count samples that are inside the circle



Libraries - CURAND

- Parallel Random Number Generation
  - GPU via Host API
  - GPU via Device API (in kernel calls)
  - CPU

- Device API:
  - #include <curand_kernel.h>
  - curandState s;
  - curand_init(seed, 0, 0, &s);
  - float v = curand_uniform(&s);

Libraries - Thrust

- C++ Template Library, like STL
  - Dynamic Memory Class (Vector)
  - Iterators
  - Algorithms (transform, sort, reduce, scan)
- Can be used on CPU via OpenMP backend:
  - nvcc -O2 -o monte_carlo monte_carlo.cu -Xcompiler -fopenmp -DTHRUST_DEVICE_BACKEND=THRUST_DEVICE_BACKEND_OMP -lcudart -lgomp

Libraries - Thrust

- thrust::counting_iterator (int value)
  - generate a sequence of numbers without the need to store them to memory
- thrust::reduce(iter_start, iter_end, offset_value, operator)
  - combine all elements provided by iterators using operator, e.g. thrust::plus<float> will sum all elements
- thrust::transform_reduce(iter_start, iter_end, transform_functor, offset_value, operator)
  - like reduce but call functor on each element before using it in the reduction