Advanced Features of CUDA
Objective

- Introduce some of the most relevant “advanced” features of CUDA
  - The majority of features here will probably not be necessary or useful for any particular application
- CUDA Programming Guide (CPG) 3.1 sections will be referenced
Agenda

- Tools
- A note on pointer-based data structures
- Warp-level intrinsics
- Streams
- Events
- Textures
- Atomic operations
- Page-locked memory & zero-copy access
- Multi-GPU
- Graphics interoperability
- Dynamic compilation
Tools: nvcc

Some nvcc features:

- --ptxas-options=-v
- Print the smem, register and other resource usages
- #pragma unroll X
- You can put a pragma right before a loop to tell the compiler to unroll it by a factor of X
  - Doesn't enforce correctness if the loop trip count isn't a multiple of X
- CPG E.2
The cuda profiler can be used from a GUI or on the command line.

Cuda profiler collects information from specific counters for things like branch divergence, global memory accesses, etc.
Tools: Debugger

- printf and cuprintf in kernel function
- cudagdb
  - Debugger with gdb-like interface that lets you set breakpoints in kernel code while it's executing on the device, examine kernel threads, and contents of host and device memory
- Parallel Nsight for Visual Studio
  - Build-in interfaces for debug in GPU
    - Break points
    - Local variables
- Multi-GPU support
- Video tutorial:
  - http://developer.download.nvidia.com/tools/ParallelNsight/Videos/Parallel_Nsight_1.0_CUDADebug.wmv
Device pointers and host pointers are not the same.

For an internally-consistent data structure on the device, you need to write data structures with device pointers on the host, and then copy them to the device.
Warp-level intrinsics

- **warpsize**
  - Another built-in variable for the number of threads in a warp
  - If you *have to* write code dependent on the warp size, do it with this variable rather than “32” or something else

- **Warp voting**
  - Warp And, Warp Or (\_\_all and \_\_any)
    - Allows you to do a one-bit binary reduction in a warp with one instruction, returning the result to every thread
  - CPG B.2
**Streams**

- **All device requests made from the host code are put into a queue**
- **Queue is read and processed asynchronously by the driver and device**
- **Driver ensures that commands in the queue are processed in sequence.**
  Memory copies end before kernel launch, etc.
To allow concurrent copying and kernel execution, you need to use multiple queues, called “streams”

Cuda “events” allow the host thread to query and synchronize with the individual queues.

CPG 3.2.7.5
Events

CUDA uses Events for timing purpose and synchronization
  - GPU timer
  - Synchronization (wait until an event is recorded)
  - CPG 3.2.7.6
Textures

- `texture<Type, Dim, ReadMode> texRef(norm, fMode, aMode)`
- Creates a reference to a texture object
- **Type**: the element type of the stored texture
  - Can be short vector types, like `char4` or `uint2`
- **Dim**: the dimensionality of the texture
- **ReadMode**: choice of return type from fetch functions
  - `cudaReadModeElementType`: fetches the “real” elements
  - `cudaReadModeNormalizedFloat`: elements automatically converted to normalized floats with magnitude [0,1] when fetched
Textures cont.

- `texture<Type, Dim, ReadMode> texRef(norm, fMode, aMode)`
- `norm`: selects normalized indexes or not
  - 0: texture indexes are integers [0,width-1]
  - 1: texture indexes are floats [0,1]
- `fMode`: filtering mode
  - `cudaFilterModePoint`: fetch nearest element
  - `cudaFilterModeLinear`: linearly interpolate result from nearest points – only for floating-point Type
- `aMode`: addressing mode
  - `cudaAddressModeClamp` or `cudaAddressModeModeWrap`, for whether accesses are clamped to image edge wrap around
Texture binding

- After creating a texture reference, you must bind it to a region or memory before use.
- The best way to allocate memory for textures is to use cudaArrays.
- Compared to global memory, textures have some extra overhead, but have some bandwidth benefits.
  - Cached: gives bandwidth benefit when locality exists
  - Latency still high, even if cached
  - Coalescing requirements do not apply
Atomic Operations

- Integer atomic ops to global memory
  - Supported for compute capability 1.1 and higher (G92 on)
  - Fundamentally has the same bandwidth and coalescing attributes as normal global memory accesses
    - Consumes bandwidth for read and write
    - Uncoalesced accesses still burn excess bandwidth
    - Non-blocking instructions

- Integer atomic ops to shared memory
  - Supported for compute capability 1.2 and higher (GT200 on)

- Major features to look into for doing histograms
Advanced Features

Page-locked memory and zero-copy access

- Page-locked memory is memory guaranteed to actually be in memory
  - In general, the operating system is allowed to “page” your memory to a hard disk if it's too big, not currently in use, etc.

- `cudaMallocHost()` / `cudaFreeHost()`
  - Allocates page-locked memory on the host
    - Significantly faster for copying to and from the GPU
  - Beginning with CUDA 2.2, a kernel can directly access host page-locked memory – no copy to device needed
    - Useful when you can't predetermine what data is needed
    - Less efficient if all data will be needed anyway
    - Could be worthwhile for pointer-based data structures as well
Multi-GPU Computing

- One workstation can support multiple GPUs, each of which should be controlled by a CPU thread in different contexts (as least the same number of CPU cores as the number of GPUs)
- Select GPU by calling `cudaSetDevice()`
- Inter-GPU communication needs to go through host, using `memcpy` (pinned memory and async)
- The CPU code can use OpenMP and MPI interface
➢ Want to render and compute with the same data?
  ➢ CUDA allows you to map OpenGL and Direct3D buffer objects into CUDA
  ➢ Render to a buffer, then pass it to CUDA for analysis
  ➢ Or generate some data in CUDA, and then render it directly, without copying it to the host and back
Dynamic compilation

- The CUDA driver has a just-in-time compiler built in
  - Currently only compiles PTX code
  - Still, you can dynamically generate a kernel in PTX, then pass it to the driver to compile and run
  - Some applications have seen significant speedup by compiling data-specific kernels