Harness the Power of GPUs:
An Introduction to GPGPU Programming
Lecture 4: GPU Memory Hierarchies and Management

Guido Juckeland
Visiting Scholar
Technische Universität Dresden, Germany

June 17, 2014
GPU memory hierarchy

- Registers
- Shared Memory
- Global Memory

Bandwidth

Amount of memory
GPU memory bandwidth

**Tesla C2050 (FERMI)**

Shared memory: $4 \times 32 \text{ Banks} \times 14 \text{ SM} \times 0.5 \times 1.15 \text{GHz} = 1.03 \text{Tbyte/s}$

Registers ($a \times b + c$): $3 \times 4 \times 14 \text{ SM} \times 32 \text{ Threads} \times 1.15 \text{ GHz} = 6.18 \text{Tbyte/s}$
Kepler GPUs have 65,536 32-bit wide registers per SMX
What: Registers are local memory for the compute units
How: every local variable in your kernels is put into registers (e.g. int i)
Who: Registers are attached to one thread and only accessible by this thread
Lifetime: data lives as long as the thread lives
Access: read and write
Zero Latency!!
Kepler GPUs have up to 48kB of shared memory per SMX

What: Memory for fast data exchange or common access within a thread block

How: Declared as __shared__ float commondata[threadsPerBlock] in the kernel

Who: All threads of one thread block

Lifetime: data lives as long as the thread block lives

Access: read and write

Using shared memory efficiently is the key to high performance!
Shared memory organization

Consecutive 32 bit words are placed in consecutive memory banks (*interleaving*).

Number of shared memory banks
Kepler GPUs: 32

32 bit word

Shared Memory

- read
- write
Bank conflicts

- Serializes shared memory accesses
- Increases the one clock cycle access time

- **Shared Memory**
- **Thread i**
  - read
  - write
- **Thread j**
Broadcasts

- No conflict
- Read accesses to multiple threads can be broadcast
Concurrent read/write accesses

- No conflict
- Write access must be to “free” banks to be carried out within one clock cycle

**Diagram:**

- **Shared Memory**
- **Thread i**
- **Thread j**

**Legend:**
- Blue: read
- Red: write
Strided memory accesses

- **Left:**
  Every thread accesses consecutive data, linear addressing with a 1x32bit stride, no bank conflict

- **Middle:**
  Every thread accesses every other data element, linear addressing with a 2x32bit stride, 2-way bank conflict

- **Right:**
  Every thread accesses every third data element, linear addressing with a 3x32bit stride, no bank conflict
Global memory

- Kepler GPUs have between 4 and 12 GB of global device memory.
- What: Large memory for streaming data that is not used multiple times.
- How: All device pointers, manually via `__global__ float common[n]`
- Who: All threads on the device and host
- Lifetime: runtime of the program
- Access: read and write
- This is where you `cudaMemcpy`'s go

If you are only using global memory, you might just want to buy more CPUs.
Data alignment

Device memory (128 byte alignment)

Transferred data (1 x 128 Bytes, not aligned)

Actually transmitted data (2 x 128 Byte)

50% memory bandwidth thrown away
Ensuring data alignment

- Memory can be aligned by using the “right” CUDA mallocs
- **1D array**
  ```c
  cudaMalloc() // start address is aligned
  ```
- **2D array**
  ```c
  cudaMallocPitch() // start address of every row is // aligned
  ```
- **Other mallocs**
  ```c
  cudaMemcpy3D(), cudaMemcpyToArray(), cudaMemcpy3DArray()
  ```
Alignment of structs or classes

struct Foo {
    float a;
    float b;
    float c;
} // sizeof(Foo) == 12

struct __align__(16) Bar {
    float a;
    float b;
    float c;
} // sizeof(Bar) == 16
Automatic host-device transfers

- **Mapped memory**
  - `cudaHostAllocMapped()`
  - Does not require device memory allocations
  - Data is transferred automatically to the device and back
  - Does not do sections or asynchronous copying
Kepler GPUs have up to 512 kB local memory per thread.

What: Register spilling memory for unknown sizes or excessive use

How: see above, manually via __local__ int foo

Who: only attached to one thread

Lifetime: lifetime of the thread

Access: read and write

If you use local memory, you will see a large performance drop.
Kepler GPUs have 64kB of constant memory.

**What:** Cached part of device memory with own data path

**How:** `__constant__` float `readOnlyData[n]`

**Who:** All threads on the device and host

**Lifetime:** Runtime of the program

**Access:** Read only
Wrap-up
QUESTIONS?