CUDA: Global Memory

Mark Greenstreet

CpSc 418 – November 19, 2018

- Questions from last week
- Architecture Snapshot
- The Programmer's View of Global Memory
- Summary, preview review

Questions from Piazza (part 1)

- *The diagram on slide 2 is not representative of the GTX 1060 in the numbers of SMs, right? It has a lot more than 9 SMs.*
  - Correct. The slide showed a GTX 1080 – the high end GPU for the "Pascal" generation.
  - The GTX 1060 GPU is the "good-enough but not super-expensive" GPU in the `linXX` machines.
  - The GTX 1080 has 28 SMs, the GTX1060 has 9 SMs. The SM architecture is the same for both.
- *I think I heard that the equation on slide 4 is wrong? The new equation given in class that I've written down is* $\frac{11521.724}{192} = 82$*. Why do we need to multiply by 2? Is it because we need to load 2 floating point numbers per floating point operation?*
  - The slide is now fixed. I had done the calculation for the GTX 1080, updated the slide deck for the GTX 1060, and missed that equation.
  - The multiply by 2 is because an SP can perform a fused multiply-add as a single instruction, but it counts as two floating point operations.
  - Details of the calculation are on the next slide.

CGMA – calculation details

- CGMA: Compute-to-Global-Memory-Access ratio
  - Compute: number of floating point operations
  - Global memory access: number of 32-bit words read and/or written to/from the GPU's DRAM
- The example from the previous slide:

$$\begin{aligned} \text{CGMA} &= \frac{1152\text{SPs} * 1.7 \times 10^9 \frac{\text{instructions}}{\text{SP}\cdot\text{sec}} * 2 \frac{\text{flops}}{\text{instruction}}}{192 \frac{\text{bytes}}{\text{sec}} * 1 \frac{32-\text{bitword}}{\text{byte}}} \\ &\approx 82 \frac{\text{flops}}{\text{memoryaccess}(32-\text{bitword})} \end{aligned}$$

where

- 1152 SPs: GTX 1060 architecture details
- $1.7 \times 10^9 \frac{\text{instructions}}{\text{SP}\cdot\text{sec}}$: GTX 1060 clock frequency
- $2 \frac{\text{flops}}{\text{instruction}}$: fused multiply-add
- $192 \frac{\text{bytes}}{\text{sec}}$: GTX 1060 off-chip memory bandwidth
- $1 \frac{32-\text{bitword}}{\text{byte}}$: `sizeof(float)`

Questions from Piazza (part 2)

On slide 7 (now 8), it says that threads in the same warp can swap registers. What does this mean? And why would they want to "swap" registers?

- Each SP has its own register file. Each thread in a warp executes on a different SP and can only access its own register file.
- However, we can use an instruction that says
  - Threads `i` and `i+4` both read their own register 5.
  - Thread `i` sends its value to thread `i+4`, and thread `i+4` sends its value to thread 5.
  - We could do this in the case where `i & 4 == 0` and thus `(i+4) & 4 = 1`.
- Why would we want to do this?
  - Draw picture on whiteboard
  - It lets the two threads communicate (one word) with a cost of one instruction. This is very fast communication. Compare with the cost of λ from Erlang where λ could be 1000 or more. If we know the communication pattern in advance, for example, as in reduce, scan, or bitonic sort, we can make very good use of such instructions.
  - The limitation is that the threads must be in the same warp. This gives us very fast communication within a warp of 32 threads, but we have to use more expensive mechanisms to communicate between threads in different warps.

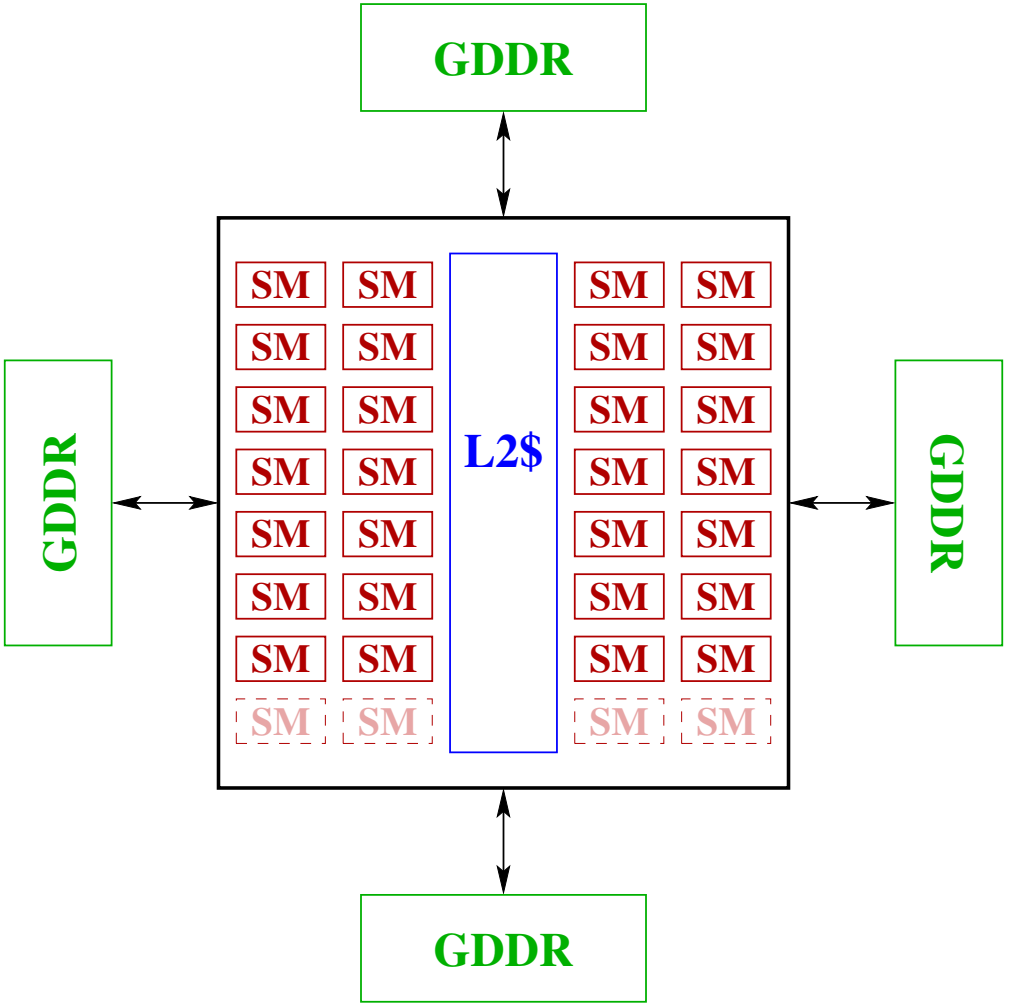
Questions from Piazza (part 3)

- *On slide 9 (now 10), why are there 64 active warps? I tried relating this to the number of SPs/SM and the number of warps that the SM can schedule in a cycle but I couldn't figure it out.*
  - That was a hardware design choice.
  - If the hardware supports more warps, the SM can run more blocks at the same time. This means there are more candidates for runnable warps when executing high-latency instructions, especially when waiting for a read of off-chip memory.
  - However, scheduling 64 warps means being able to chose any of them each cycle. If the designers make that number too large, then the hardware for warp selection becomes too complicated which can limit the clock frequency and increase power consumption.
- *What exactly are banks?*
  - *Is it just shared memory divided up into groups?* **Yes.**
  - *If so, how is it divided?* `bankIndex = (byteAddress div 4) mod 32`.
    - ★ In more detail, the GPU lets the programmer specify whether to compute banks using `div 4` or `div 8`.
    - ★ We'll use `div 4` for everything in class to keep it simple.
  - *When might all threads in a warp access the same bank? Is an example when all threads are getting values from the same column in a matrix.* **Very good example**. **Another example** is random accesses. It won't be the case that all threads will access the same bank, but there will be lots of conflicts.

Questions from Piazza (last question)

- *On slide 12 (now 13), for the number of computes in the CGMA ratio, why do we need to multiply 4096 by 2?*
- **It's that sneaky fused multiply-add again.**

GPU Architecture Review



GTX 1080 Architecture
(high–end, "Pascal" generation)

GPUs in the the `linXX.ugrad.cs.ubc.ca` machines

- GeForce GTX 1060
- 9 SMs (10 on chip, 9 reported by the device):
  - 128 SPs/SM.
  - That's 1152 SPs on the chip.
  - Each SP can read two 32-byte register and write one 32-byte register each clock cycle.
  - With a 1.5GHz clock, that's about 20 TBytes/sec.
- $\sim$ 1.5GHz clock frequency.
- 3 GBytes of GDDR5 memory, $\sim$ 192GBytes/sec. memory bandwidth.
  - Limited by number of off-chip wires.

How DRAM works

- Programmers want memory that is large and fast.
  - Physics means and large memories are slow, and fast memories are small.
  - DRAM is on the “large and slow” end of the scale (but still smaller and faster than flash, or other non-volatile memory technologies).
- A DRAM is an array of “capacitors”
  - A capacitor stores electric charge kind-of like how a glass of water stores liquid.
  - Writing a bit is easy: fill or empty the glass.
  - Reading means connecting the glass through a pipe to a sensor to detect the water level.
    - ★ The pipe is long, and has a much larger volume than the glass.
    - ★ The change in water level in the pipe is tiny.
    - ★ It takes a long time (10s of nanoseconds) to measure the water level.

Big-$\mathcal{O}$ for DRAM

- Arrange water cups in an $N \times N$ array.
- Memory capacity goes as N^2.
- Overhead for circuitry to select and sense bits goes a $O(N)$.
- Make N large to amortize overhead.
  - This should be a familiar story by now.
  - The limit on N is how big can we make the array and still make a reliable measurement.
  - For real DRAMs, N is 1024 (i.e. 2^{10}) or 2048.
  - That would mean we would have 1Mbit $=$ 128Kbyte DRAM chips.
    - ★ How do we get 8Gbit parts?

DRAM Tiles

- Each DRAM chip has **many** of these 1Mbit tiles.
- E.g. an 8Gbit chip would have 8192 such tiles.
- Example: access 128bits of data from a DIMM with 16 DRAM chips.
  - 8 bits from each chip.
  - Each bit from a different tile.
    - ★ Why?
    - ★ Error correction.
- Memory addresses:
  - Some address bits specify which DIMM
  - Some address bits specify which tile(s) on each DIMM.
  - Some address bits specify which row of the tile to access.
  - Some address bits specify which column of the tile to access.
- The book talks about memory tiles.
  - Knowing the details lets you tune your code for maximum performance.
  - We won't do that kind of tuning in this class.

DRAM Performance

- Latency: really bad
  - 20-30ns for GDDR, 60-100ns for typical PC memory.
  - Compare with 1.6GHz GPU clock or 3GHz CPU clock.
  - Memory latency is 30 to 300 or more clock cycles.
- Bandwidth: not great, but we can do something here.
  - When we read a location, we get 1 bit from each tile that is accessed.
  - But, we had $\sim$ 1024 bits available from each tile.
  - Opportunity: read and write large bursts.
    - How?
    - Careful coordination among many threads.

Coalesced Memory Accesses

Other GPU Memory

- Texture Cache: read-only, shared by all blocks
- Caches
  - L1 cache: per SM
  - L2 cache: shared by all SMs
  - No coherence guarantees

The CUDA Memory Philosophy

GPUs are very fast,
But DRAM has bad latency.
Use lots of threads carefully.

Summary

- GPUs can have thousands of execution units, but only a few off-chip memory interfaces.
  - This means that the GPU can perform 10-50 floating point operations for every memory read or write.
  - Arithmetic operations are very cheap compared with memory operations
- To mitigate the off-chip memory bottleneck
  - GPUs have, limited on-chip memory
  - Registers and the per-block, shared-memory will be our main concerns in this class.
- Moving data between different kinds of storage is the programmer's responsibility.
  - The programmer explicitly declares variables to be stored in shared memory.
  - The programmer needs to be aware of the per-thread register usage to achieve good SM utilization.
  - The only way to communicate between thread blocks is to write to global memory, end the kernel, and start a new kernel (ouch!)

Preview

November 21: GPU Performance: Part 1

Reading: Kirk & Hwu – Chapter 5

November 23: GPU Performance: Part 2

November 25: GPU Performance: Part 3

Review

- What is CGMA?
- On slide ?? we computed the CGMA for matrix-multiplication using 16×16 blocks of the A, B, and C matrices.
  - How many such thread-blocks can execute concurrently on an SM with 48KBytes of memory?
  - How does the CGMA change if we use 32×32 blocks?
  - If we use the larger matrix-blocks, how many thread blocks can execute concurrently on an SM with 48Kbytes of memory?
  - If we use the larger matrix-blocks, how many thread blocks can execute concurrently on an SM with 96Kbytes of memory?
- What are bank conflicts?
- How can increasing the number of registers used by a thread improve performance?
- How can increasing the number of registers used by a thread degrade performance?
- What is a “coalesced memory access”?