November 21st Lecture Summary

2023-12675 박지호

In today's lecture, we picked up where we left off on the previous chapter and continued studying how to optimize a GPU program. In the last lecture, we learnt about pipelining as one way to achieve this. In today's lecture, blocking was presented as another way to optimize a GPU program. Blocking refers to a strategy where we divide the problem into smaller blocks which can be evaluated in parallel. This strategy can be used for matrix multiplication, which is a topic that was brought up multiple times in this class already. Also, we talked about how certain parts of the matrix can be tiled to minimize memory access. This time, however, we went into more detail about this optimization technique and how it can be pushed further by utilizing multi-level tiling, where we exploit every level of memory for this storage.

Next, we talked about yet another GPU optimization technique: data layout & memory request coalescing. When threads from the same warp access the cache, if there they attempt to access the block in the same bank, multiple cycles will be required to process every request. We call this the bank conflict. One way to solve this is by adjusting the memory layout, and changing how the data is mapped to memory bank. Common strategies used for this solution is padding and permutation. Other way to solve this is by coalescing the memory requests, where we merge the duplicate cache miss and send in only one reqeust for this.

Then, we looked at many processors in increasing complexity. First we looked at single-core, single-threaded processors: one that can perform single scalar operation per clock, one that can perform two scalar operation per clock, one that can perform one vector operation per clock via SIMD, and one that can perform one scalar operation and one vector operation per clock. Then we looked at single-core, multi-threaded processors, and then the multi-core, multi-threaded processors. Lastly, we looked at GPU. Unlike the other processors we looked at before, GPU has a single ALU, but multiple execution contexts, allowing it to support multiple threads.

Then we moved on to the topic of shared memory vs. message passing. Shared memory multiprocessor (SMP) refers to a multiprocessor that uses single physical address space for all processors, which is the architecture we've been discussing in this class. An alternative to this approach is message passing, where each processor has private physical address space, and informations are passed around by explicitly sending and receiving message. This approach allows for more freedom in optimization, as the programmer has full control over the connections. There were more details about the specifics of networks, which would be discussed further in the coming lectures, when we talk about accelerators.

The final topic of the lecture was a roofline diagram. This refers to a diagram of relation between arithmetic operation's speed and its intensity. Here, intensity means how well memory access is utilized for computation. When the intensity is moderate, the memory bandwith is a limiting factor, but after reaching the certain speed, the speed is bottlenecked by a peak FP performance. We used matrix multiplication as an example to see how roofline model is drawn. Then, we looked at how poor memory usage shifts a data point in roofline diagram. We also looked at how change in memory bandwidth shifts the diagonal line in the diagram, and how memory usage needs to change to adapt to a different architecture.