

Problem 1: As we discussed in class an important factor in GPU performance is the number of resident warps in an MP (also called an SM). There are several factors that determine the number of resident warps per MP, including the block size, grid size, the amount of shared memory used per block, and the number of registers used per thread.

(a) Consider `conv_inbuf_block` from the Homework 1 solution. This kernel lazily sizes `in_buffer` to 2048 elements, which wastes a considerable amount of memory.

Find the number of resident warps per SM on a RTX 2080 if this kernel is launched with a grid size of 48000 blocks and block size of: 32 threads, 64 threads, 256 threads, and 1024 threads.
Hint: This is an easy problem.

(b) What is the maximum value on D_j for kernel `conf_wbuf` from Homework 2? (Base this on the kernel, not limits used elsewhere in the file.) *Hint: The limit is due to the address space used for w .*

Problem 2: A GeForce RTX 2080 (TU104) has an off-chip data bandwidth of $\Theta_M = 496.1$ GB/s and a single-precision floating-point bandwidth of $\Theta_{\text{IF}} = 5576$ SP GFLOPS, with a multiply/add counted as one operation.

(a) Find the off-chip data bandwidth, Θ_M , and single-precision floating-point bandwidth, Θ_{IF} , for a Volta V100 GPU and a Pascal P100 GPU. Remember to count a multiply/add as one instruction.

(b) Let Θ_{IT} denote the throughput of integer type conversions of size up to 32 bits. (For example, converting a 32-bit integer to an 8-bit integer.) Find Θ_{IT} for the following GPUs: Turing TU104, Volta V100, and a Pascal P100.

Problem 3: The kernels in `hw02.cu` compute the same kind of convolution as those from Homework 1 but they are compiled for several values of D_j , and they are also compiled for specific block sizes. Let J indicate the value of D_j (the number of weights) and N indicate the array size (value of `app.array_size`). The amount of computation per element (one value of `h`) in the code is J multiply/adds, the number of data items read is $2J$ (counting both the inputs and weights), and one output element is written. Of course we know that a well-written routine will read fewer than $2JN$ items. For a large N the number of items read and written per thread will approach $2N$.

For a given value of J and N two lower bounds on execution time can be computed, one using data bandwidth and the other using FP bandwidth. They are $t_M(N) = 2N(4B)/\Theta_M$ and $t_F(N, J) = NJ/\Theta_{IF}$. The $t_M(N)$ bound is closer to reality for CC 7.X GPUs because it is easy to predict the amount of data crossing the chip boundary. The t_F bound above is much less realistic because it only counts multiply/add instructions. On CC 3.X to CC 6.X GPUs $t_M(N)$ will underestimate `conv_wbuf` because the `dapp.d_in` accesses won't use an L1 cache. On all GPUs we have considered t_F will be way off because we are only considering multiply/add instructions.

(a) Suppose `conv_wbuf` and `conv_inbuf_block` were launched on CC 7.5 (Turing) GPUs.

Which would better approximate `conv_wbuf`'s run time in a one-block-per-MP launch configuration, $\max\{t_M(N), t_F(N, J)\}$ or $t_M(N) + t_F(N, J)$? Explain.

Which would better approximate `conv_inbuf_block`'s run time in a one-block-per-MP launch configuration, $\max\{t_M(N), t_F(N, J)\}$ or $t_M(N) + t_F(N, J)$? Explain.

Suppose now each kernel were launched in a configuration with 4 blocks per MP and the block size chosen so that 4 blocks can be resident. (That is, four blocks can be active on one MP at the same time.) How does that change the answers to the questions above?

(b) Suppose $t(N, J) = \max\{t_M(N), t_F(N, J)\}$ was a good bound on execution time. A modeled execution is called *compute-bound* when $t_M(N) < t_F(N, J)$ and *data-bound* when $t_M(N) > t_F(N, J)$. (Those terms are also used for actual executions.)

For a compute-bound execution, effort to increase data bandwidth will have no effect on execution time, and for a data-bound execution effort to improve execution rate will have no effect.

Using the bounds above find the value of J for which execution is neither compute- nor data-bound. (Don't worry about J not being an integer.)

(c) The bound t_F is crude because the kernels execute more than floating-point instructions. Based on an inspection of the SASS code there is a load instruction for each multiply/add (which makes sense). For `conv_wbuf` the load is from global memory, and for `conv_inbuf_block` the load is from shared memory.

Find a bound $t_I(N, J)$ that accounts for both the multiply/add and the load instructions. Use class materials and other resources to find the throughput of load and store instructions.

Use this new bound to find J for which the instruction time bound and data time bound are equal.