LSU EE 7722 **Homework 2** Solution **Due: 6 April 2020**

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.*

Each block uses $4 \times 2048 = 8192\,\mathrm{B}$ of shared memory. The RTX 2080 is of CC 7.5, and those devices have $64\,\mathrm{kiB} = 2^{16}\,\mathrm{B}$ per SM, and so shared memory limits the number of blocks to no more than $\lfloor 64\,\mathrm{kiB}/8\,\mathrm{kiB} \rfloor = 8$ blocks.

For a block size of 32 threads (or 1 warp) there would be 8 blocks and so 8 warps resident. For 64 threads there would be $2 \times 8 = 16$ warps. For 256 threads shared memory would set an upper limit of 8 blocks (or 64 warps) but CC 7.5 devices can have no more than 32 resident warps, so there would be only 4 blocks resident and so $4 \times 8 = 32$ resident warps. Similar reasoning applies to the 1024-thread block: there would be one block or 32 warps resident.

(*b*) What is the maximum value on `Dj` 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`.

Variable `App dapp` is in the constant address space. For all CUDA devices the address space size is $64\,\mathrm{kiB}$. The six pointers (`h_in`, etc.) use $6 \times 8\,\mathrm{B}$, the int $4\,\mathrm{B}$, leaving $2^{16} - 52 = 65484$, enough for 16371 floats, and so that is the maximum value of `Dj`.

Problem 2: A GeForce RTX 2080 (TU104) has an off-chip data bandwidth of $\Theta_M = 496.1\,\text{GB/s}$ 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.

Volta V100: $\Theta_M = 898\,\text{GB/s}$ and $\Theta_{\text{IF}} = 80 \times 64 \times 1.38\,\text{GHz} = 7065.6\,\text{SP GFLOP/s}$ SP GFLOPS. (That was based on 80 SMs, 64 FP32 units per SM, and a 1.38 GHz clock frequency.) For a P100: $\Theta_M = 549.1\,\text{GB/s}$ and $\Theta_{\text{IF}} = 56 \times 64 \times 1.33\,\text{GHz} = 4766.7\,\text{SP GFLOP/s}$.

(*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} the following GPUs: Turing TU104, Volta V100, and a Pascal P100.

Based on the table in Chapter 5 of the CUDA Programming Guide 10.2, the number of type-conversion functional units are 16 for all the devices: TU104, V100, P100. The throughputs for each device in instructions per second are:

TU104: $\Theta_{\text{IF}} = 48 \times 16 \times 1.81\,\text{GHz} = 1390 \times 10^9\,\text{insn/s}$
V100: $\Theta_{\text{IF}} = 80 \times 16 \times 1.38\,\text{GHz} = 1766 \times 10^9\,\text{insn/s}$
P100: $\Theta_{\text{IF}} = 56 \times 16 \times 1.33\,\text{GHz} = 1192 \times 10^9\,\text{insn/s}$

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 `Dj`, and they are also compiled for specific block sizes. Let J indicate the value of `Dj` (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 $t_M(N) = 2N(4\,\mathrm{B})/\Theta_M$ and $t_F(N, J) = NJ/\Theta_{\mathrm{IF}}$. 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.

It would be better approximated by $\max\{\, t_M(N),\, t_F(N, J)\,\}$ because the memory accessed performed by one warp can be overlapped with the computation performed by another. If $t_F(N, J) > t_M(N)$ we could hope that there would always be a warp ready to peform multiply adds, even when other warps are awaiting data. If that's the case (and if the contribution of other instructions is tiny) execution time is based only on FP throughput.

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.

Expression $t_M(N) + t_F(N, J)$ is the better approximation because the `syncthreads` will force FP operations to wait for memory accesses and shared memory stores to complete and vice versa. So there is no way to overlap the two activities.

Suppose now each kernel were launched in a configuration with 4 blocks per MP and the block size were 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?

Kernel `conv_inbuf_block` would benefit because it would now be possible to overlap computation and data access. While one block in an SM is waiting for data another block in the SM can be computing. Therefore the bound should be closer to $\max\{\, t_M(N),\, t_F(N, J)\,\}$.

(*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.)

Equating and solving for J yields $J = 8\frac{\Theta_{\mathrm{IF}}}{\Theta_{\mathrm{M}}}$ B. For a T104: $J = 8\frac{5576\,\mathrm{SP\,GFLOP/s}}{486.1\,\mathrm{GB/s}}\,\mathrm{B} = 91$.

(*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.

On recent NVidia devices the throughput of memory instructions is $\frac{1}{4}$ that of FP32 operations. Let Θ_{IM} denote the throughput of memory instructions. Then $t_F(N, J) = NJ(1/\Theta_{\mathrm{IF}} + 1/\Theta_{\mathrm{IM}})$. For cases in which $\Theta_{\mathrm{IM}} = \Theta_{\mathrm{IF}}/4$ we have $t_F(N, J) = NJ\,(1/\Theta_{\mathrm{IF}} + 4/\Theta_{\mathrm{IF}}) = 5NJ/\Theta_{\mathrm{IF}}$. That of course is substantially longer.

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

Equating and solving for J yields $J = 1.6\frac{\Theta_{\mathrm{IF}}}{\Theta_{\mathrm{M}}}$ B. For a T104: $J = 1.6\frac{5576\,\mathrm{SP\,GFLOP/s}}{486.1\,\mathrm{GB/s}}\,\mathrm{B} = 18.35$.