CMP SCI 635: Modern Computer Architecture

# A Variable Warp Size Architecture

Name: Kunjal Panchal Date: 21th Nov, 2019 Student ID: 32126469

Paper: 21 – Data Parallel [Rogers15]

**Strengths:**

1. Having multiple thread blocks would benefit when there are is global memory latency involved.

2. A smaller WARP size may be advantageous because the number of threads affected by the divergence caused by some flow control instructions is smaller. Finer granularity for programs needing flow control.

3. If I have a Kernel that without many flow control instructions, then it may be advantageous for the WARP size to be 32, or some other relatively large value. On the other hand, if I have a Kernel that needs some flow control instructions, then divergence is an issue that influences the performance. For such a case, a WARP size of 1 may make more sense.

**Weaknesses:**

1. If we want to check whether a group of threads are executing in the same warp, the synchronization primitives are discouraged from being used directly as independent thread scheduling rolls out - they can be tricky to get right once lockstep warp execution isn't guaranteed.

2. Within the limitations imposed by hardware, what thread and block configuration results in the highest performance for a given GPU depends on the code that is being run. Block configuration in particular interacts with other resource limits in terms of occupancy. There is no universal formula for the "best" configuration.

**Questions/Assertions:**

1. In a one-dimensional launch configuration, the Programming Guide always calculates the lane_id (thread index within a warp) as

   ```
   int lane_id = thread_index & (32 - 1)
   ```

   For multi-dimensional launch configurations, the multi-dimensional thread indices map to linear indices starting from the x-dimension, over y and then to z

   ```
   (int thread_index = threadIdx.x + (threadIdx.y * blockdim.x) + (threadIdx.z * blockDim.x * blockDim.y))
   ```

   Thus, threads within the same block are split up into warps such that threads 0-31, 32-63, 64-95, 96-127, ... are always within a warp.

2. For Nvidia the work group sizes are always set to values like 4, 8, 16, 32. Because we don't want any of those threads to be idle so the local "volume" (x*y*z) should be a multiple of the warp or wavefront size.

3. If a warp size is 32 does that mean that all threads in the warp are actively executing at a given time. For example, my device has only 16 cores per multiprocessor. So, if all threads have to be active, does each core run 2 threads?

4. A side question from Q3: how many threads can be active at a given time in total? Will it be 32(warp size) * 16 (number of cores)? Or just 16 (number of cores) since each core will execute only 1 instruction per clock per thread? Is it correct to say that each core "simultaneously executes 4 threads"?

5. How does the processor handle the scheduling of blocks if there are multiple blocks available? I guess the GPU picks one block out of the stack of blocks and makes it active on some MP, or does it grab as many as it has resources for, then allocate them on the MP and start scheduling warps from one of them?