Multicore Computing

Assignment Four (Theory)

Department of Computer Engineering

Sharif University of Technology

Spring 2022

Lecturer:
Dr. Falahati

Name - Student Number:
Amirmahdi Namjoo - 97107212

1 Question One

a)
- conditional branch instructions (Divergent paths)
- Stalls due to long latency operations

For example, grouping threads into warps is efficient if those threads remain on the same dynamic execution path (i.e., same PC) throughout their execution. For complex applications with complicated control flow behavior, conditional branch instructions can cause threads in a warp to take different dynamic execution paths (or diverge). GPUs allow a warp to have one active PC, so these paths must execute sequentially, and therefore, the warp must execute with a fewer number of active threads than the SIMD width of the core.

The other problem is stalls caused by long latency operations. The main culprit here is scheduling policies. Suppose we use a fair scheduling policy with equal priorities (like a round-robin). In that case, all warps tend to arrive at long latency operations simultaneously, and no warp can hide the latency. If we use policies that allow warps to progress at different rates, starvation and underutilization of data locality may happen. For example, data brought to the cache by one warp is likely to be accessed by other warps, but if warps progress at very different rates, this data may be evicted from the cache when the other warp needs it.

b) The Large Warp Microarchitecture idea is proposed to decrease the performance penalty caused by branch divergence. The idea is to have fewer but larger warps so that the total number of threads and SIMD width stay the same. The key benefit is that we can create sub-warps of active threads, even when we have branch divergence.

The active mask of these large warps is a 2D structure, where the number of columns is equivalent to SIMD width, and the number of rows is equal to Large warp depth. Each cell indicates whether or not the corresponding thread is active. When a large warp is selected to fetch, the fetch and decode are done like the baseline model of GPU, but in parallel to this, a specialized logic tries to examine the two 2D structures and pick one active thread from each column to form a sub-warp.

While the divergent code executes in a baseline core, SIMD resources will be underutilized because each warp contains fewer active threads than the SIMD width. By using large warps, inactive slots are filled with active threads in the sub-warp creation process, therefore increasing utilization and, therefore, the performance will increase. Although, in this case, we will have a smaller number of warps with respect to the baseline model.

c) The baseline GPU scheduler usually uses a round-robin scheduler which gives all warps equal priority. It is beneficial in some ways because, in this way, all warps usually arrive at a specific section of the code at the same time and, therefore, can benefit from data locality; but this could cause problems because if we have a long latency operation, all warps will stall due to these operations and GPU cannot hide these high latency operations.

The idea is to use a two-level round-robin scheduler. In this system, we put warps into fetch groups of fixed size. For example, 32 warps are put into four groups of 8. Warps in a group have the same priority, but the groups have different priorities. For example, at first, group 0 has a higher priority than group 1 and so on.

Once all warps in the highest priority group are stalled due to a long latency operation, a fetch group switch occurs, giving group 1 the highest priority, group 2 the next, then 3, and then group 0, which previously had the highest priority, will now have the lowest (circular change of priorities). Prioritizing each fetch group prevents all warps from stalling; therefore, the effect of long-latency operations is alleviated, and we can get higher performance.

The fetch group size in this mechanism is essential. It must be big enough to keep the pipeline busy while there are no long latency operations and small enough to prevent all warps get stuck in a long latency operation . Actually, Having only one big fetch group is equivalent to the baseline round-robin model.

d)

- As mentioned in the article, the combination of large-warp and two-level scheduling with a fetch group size of 1 can be problematic. In an application that has no stalls after a short warm-up period, the two-level scheduling continues to prioritize a single large warp. Actually, we can end up in a situation where no other large warps are active. This can be problematic because, in this case, for conditional branches, a large warp must wait for all sub-warps to complete before being re-fetched, and this introduces several bubbles.
- Another problem is hardware cost. We must change the usual register file of GPU to implement large warps. The usual register file has a single address decoder, but to allow large warps, we must add an address decoder per SIMD lane, increasing chip area and power usage. Also, in the process of creating sub-warps, some bits in the bit mask must be cleared, but this cannot be done on the actual active mask, and a copy must be created. So for a large warp with 256 threads, additional 256-bit storage is needed.
- Large-warp size needs careful consideration. For example, in the case of `sort` and `blackjack` applications, we can see that 256 threads performed better than 512 because, in this case, most of the threads reach the reconvergence point and wait for a few threads to continue.

2 Question Two

a)

$$\text{warps} = \frac{\text{Total Thread}}{\text{Warp Size}} = \frac{1024}{32} = 32$$

We have 1024 threads because we will have one thread per outer loop iteration (based on the question description).

b) All 1024 threads execute instructions 1,2, and 4.

In the first iteration of `j`, the value of `s` is 1, so threads with `i` divisible by two will execute instruction 3. i.e., half of the threads of each warp will execute this instruction. Therefore

$$\text{SIMD Utilization} = \frac{1024 + 1024 + \frac{1024}{2} + 1024}{1024 + 1024 + 1024 + 1024} = \frac{7}{8}$$

c) All 1024 threads execute instructions 1,2 and 4.

For instruction 3, only threads with $i < 512$ will run this. The difference between this one and the previous one is that in the former, half the threads of each warp were inactive. But in this case, for half of the warps, all threads are inactive, no instructions are issued, and we have no performance loss. i.e., All threads of only half of the warps execute instruction 3.

$$\text{SIMD Utilization} = \frac{1024 + 1024 + \frac{1024}{2} + 1024}{1024 + 1024 + \frac{1024}{2} + 1024} = 1$$

d) All 1024 threads execute instructions 1,2, and 4.

For instruction 3, with $0 \leq j < 5$, all 32 warps are active, and the number of active threads per warp divides by half in each iteration. The denominator, in this case, is always 4096 with $5 \leq j < 10$, only one thread per warp is active, and some of the warps will have no active thread, and therefore scheduler will not issue any instruction for them. Therefore the denominator for these will also change.

$$\text{SIMD Utilization} = \begin{cases} \frac{3072+2^{(9-j)}}{4096}, & \text{if } 0 \leq j < 5 \\ \frac{3072+2^{(9-j)}}{3072+32\times 2^{(9-j)}}, & \text{if } 5 \leq j < 10 \end{cases}$$

e) All 1024 threads execute instructions 1,2 and 4.

For instruction 3, with $0 \leq j < 5$, all 32 threads in each warp are active, but not all of the warps are active. The number of warps active will halve each iteration. With $5 \leq j < 10$, only one warp (the one containing $i = 0$ to $i = 31$) will be active, and the number of active threads will half in each iteration. Therefore, for $0 \leq j < 5$, the value of nominator and denominator is equal, and we have 100% utilization. For $5 \leq j < 10$, the nominator changes. Therefore:

$$\text{SIMD Utilization} = \begin{cases} 1, & \text{if } 0 \leq j < 5 \\ \frac{3072+2^{(9-j)}}{3072+32}, & \text{if } 5 \leq j < 10 \end{cases}$$

f) This could not happen for $0 \leq j < 5$ because the second one has 100% utilization, while the first one has utilization of less than 1. For $5 \leq j < 10$ we can solve the equation

$$\frac{3072 + 2^{(9-j)}}{3072 + 32} = \frac{3072 + 2^{(9-j)}}{3072 + 32 \times 2^{(9-j)}}$$

$$\Rightarrow j = 9$$

Which is reasonable. In this case, only one thread of only one warp is active.

g) Code 2 will be faster; It has less intra-warp branch divergence. In other words, in most cases of Code 2, a warp is either completely active or inactive. Therefore we have high SIMD utilization, and the scheduler will not schedule completely inactive warps. In contrast, in code 1, in lots of cases, all warps are active while some threads inside warps are not active, and therefore scheduler schedules them with less than optimal utilization.