Introduction to Local Barriers of GPGPU

Synchronization is often implemented by establishing a barrier within an application where a task may not proceed further until other tasks reaches the same point. The NVIDIA CUDA provides a barrier synchronization mechanism within a thread block. However it does not allow threads in different blocks to perform barrier synchronization with each other. Such synchronization is only available through CPU by launching the kernel repeatedly. In addition, two approaches have been proposed to implement global (inter-block) barrier synchronizations: 1) lock-based barrier through atomic operation and 2) lock-free barrier through memory flags.

Why local barriers?
However a global barrier is an overkill for applications with wavefront parallelisms[1] as shown in Figure 1(a). The major drawback is unnecessary waiting time. For example, in Figure 1(b), a global barrier (repeated kernel launching) is used for thread blocks TB4, TB5 and TB6 (of kernel i) so that they complete before the next diagonal: TB7-TB10 (of kernel $i+1$). This global barrier is not necessary since TB7 only depends on TB4 and TB10 only depends on TB6. If TB6 finishes before TB4, the corresponding SM can start to process TB10 but it is forced to be idle by the global synchronization barrier.
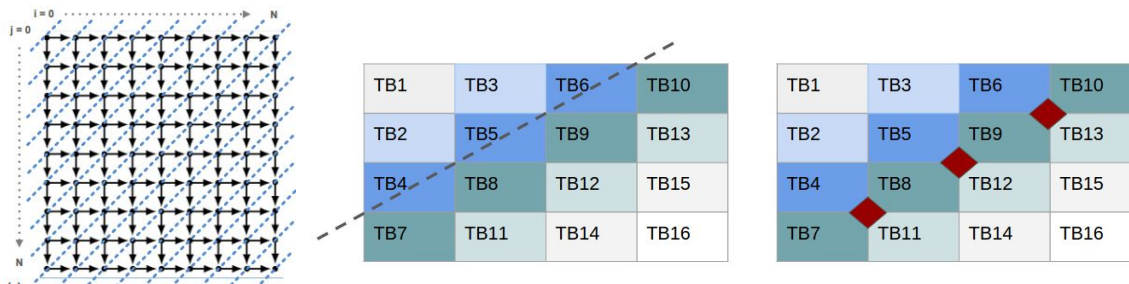

Figure 1. (a) Wavefront parallelism. (b) A global barrier example. (c) A local barrier example

In contrast, local barriers shown in Figure 1(c) avoid repeated kernel launching and improve SM utilization. Local barriers can be implemented in the same fashion as the global barrier through either atomic operations or memory flags. Instead of one global barrier, three local barriers (the diamonds) are used to ensure data dependency, where TB7 only waits for TB4, TB8 waits for TB4 and TB5 and so on. Therefore new TBs can be executed immediately when a local synchronization barrier is reached by TBs in the previous diagonal.

Basics
1. Install gpgpu-sim, setup configuration
2. Find and compile wavefront applications
3. Understand kernel launching
4. Understand the concept of within-TB synchronization and inter-TB synchronization
5. Understand the implementation of local barriers using memory flags or atomic operation.

Implementation of local/global barriers. Left: memory flag. Right: Atomic operation [2]

Experiments

We can compare 3 schemes: 1) repeated kernel launching, 2) global barrier using memory flags 3) local barriers using memory flags from the following perspectives:

1. Study the effect of the number of TBs (data size) of a kernel. Ideally, the performance improvement of local barriers against global barriers should increase with more TBs. Note that TB dimensions should stay the same.

2. Study the dimension of the TBs. The dimension of the TBs decide how many TBs can fit in one SM. A small TB means more TBs per SM which improves load balancing among SMs. A large TB has more within-TB synchronization and less inter-TB synchronization.

3. Study the effect of the granularity of local barriers. Fine grained local barriers reduce unnecessary waiting time but also introduce memory flag accessing overhead. By changing the number of local barriers per tile/TB, we can study its impacts.

All these experiments can be done with simulator only.

[1] Belviranli, Mehmet E., et al. "PeerWave: Exploiting Wavefront Parallelism on GPUs with Peer-SM Synchronization." *Proceedings of the 29th ACM on International Conference on Supercomputing*. ACM, 2015.

[2] Xiao, Shucai, and Wu-chun Feng. "Inter-block GPU communication via fast barrier synchronization." *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*. IEEE, 2010.