De La Salle University, Manila
Computer Technology Department

Term 2, A.Y. 2022-2023

CEPARCO

GPU CUDA:
Cooperative Groups

GROUP 3

Roleda, Jan Carlo S.

Solis, Frances Danielle B.

Geronimo, Michael Robert G.

Barrion, Ryan Onil C.

June 5, 2023

CEPARCO - Cooperative Group Results

Cooperative Groups are a feature that was introduced in CUDA 9 and is another way to synchronize threads within a thread block. A cooperative group is a subset of threads within a thread block that can synchronize with each other. Using this method is more beneficial than traditional CUDA programming because it offers a simpler way to synchronize threads, allows for dynamic creation and destruction of thread groups within a kernel, and improves performance due to its efficient inter-thread communication and coordination. Overall, cooperative groups provide a higher-level programming model for efficient thread collaboration and synchronization within a thread block. It makes it simpler to create parallel GPU code that is both scalable and efficient.

In this GPU project, a program for getting the dot product of two floating-point vectors x and y with the result in a scalar variable is created in three different versions: a C++ program, a CUDA program using grid-stride loop with prefetching+mem advise, and a CUDA program using cooperative groups and a grid-stride loop with prefetching+mem advise.

C++ PROGRAM

NumElem	Time (uS)
2^20	122.9
2^22	488.2
2^24	2002.2

Figure 1. Runtime of C++ Program

BASIC CUDA

	Block Size		
NumElem	256	512	1024
2^20	149.34	391.12	2359.4
2^22	405.66	613.69	2585.4
2^24	1398.7	1513.5	3494.3

*VALUES ARE IN MICROSECONDS

Figure 2. Runtime of Basic CUDA program with Mem prefetch and Mem Advise

COOPERATIVE GROUP (NO SUBGROUP)

	Block Size		
NumElem	256	512	1024
2^20	48.911	93.248	388.41
2^22	140.7	164.26	446.12
2^24	513.53	535.81	657.29

*VALUES ARE IN MICROSECONDS

Figure 3. Runtime of Cooperative Group Code with no subgroup

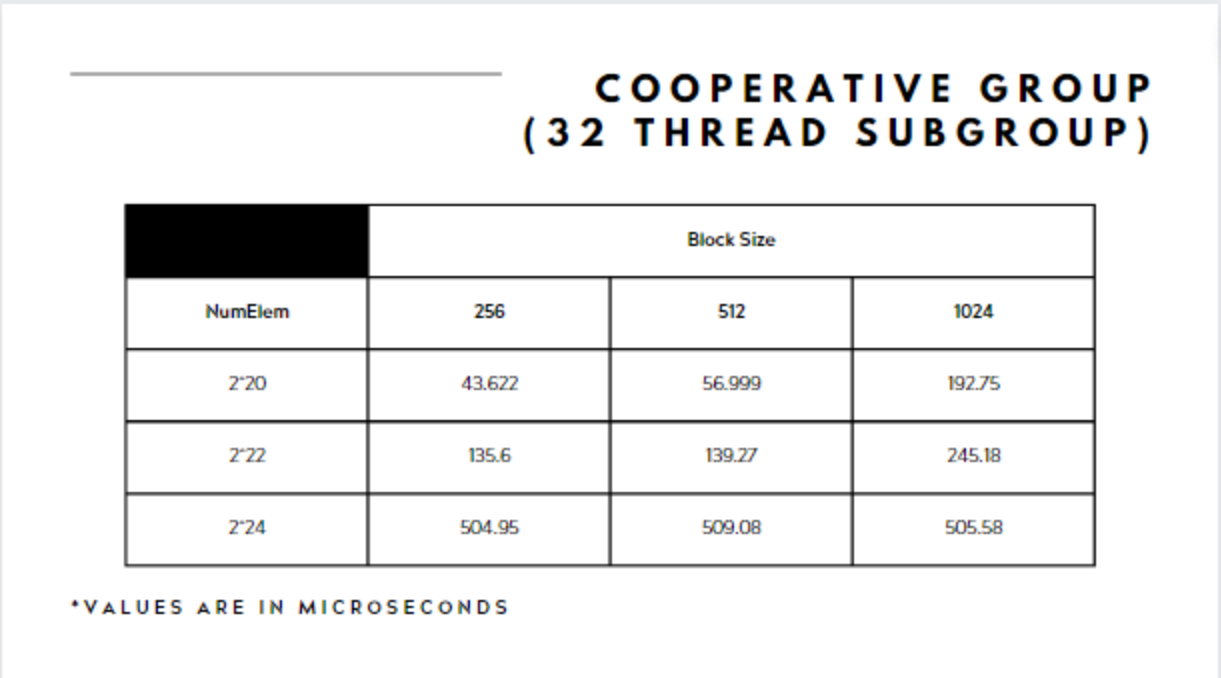

Figure 4. Cooperative Group with a 32 Thread Subgroup

These are the runtime results of C++ program, a CUDA program using grid-stride loop with prefetching+mem advise, and a CUDA program using cooperative groups and a grid-stride loop with prefetching+mem advise. It is noticeable that The cuda program using cooperative groups achieved a better performance in runtime compared with the C++ version and the CUDA code without cooperative groups. Cooperative groups are much faster due to it having sub threads. Furthermore, to further optimize, using 32 Thread subgroups increases performance especially with the addition of more elements.

Lastly these are some of the observations that were determined:

1. matching thread count to block count achieves faster results vs larger allocation of block count (for some reason)
2. subgrouping into tiles provides very granular control over synchronization of threads
3. fixed thread group size lets you unroll parallel loops to optimize a little further (in 3.2)