EEP524
Oct 31, 2022
Jaidon Lybbert
Homework #3

Questions

1) **If the SM of a CUDA device can take up to 1536 threads and up to 4 thread blocks. Which of the following block configurations would result in the largest number of threads in the SM?**

512 threads per block will maximize the number of threads per SM. Each SM containing 3 blocks of 512 threads results in 3x512=1536 threads per SM.

2) **For a vector addition, assume that the vector length is 2000, each thread calculates one output element, and the thread block size is 512 threads. How many threads will be in the grid?**

There will be 2048 threads in the grid, consisting of 4 blocks x 512 threads. 4 blocks are needed as you can't allocate partial thread blocks. The extra 48 threads may be disabled or the outputs ignored.

3) **With reference to the previous question, how many warps do you expect to have divergence due to the boundary check on vector length?**

One warp. The boundary check condition would apply to the last 48 threads in the 4th block. Since warps are 32 threads, the very last warp of the last block would contain only threads that are "out of bounds" so there would be no control divergence. The second to last warp would contain 16 threads out of bounds, and 16 within bounds. This would be the only warp with control divergence.

4) **Assume a device that allows up to 64 blocks per SM and 2048 threads per SM. Indicate which of the following assignments per SM are possible. IN the cases in which it is possible, indicate the occupancy level.**

   A. 8 blocks with 128 threads each

   Possible. 8x128=1024 active threads. 1024/32=32 active warps. 32 active/64 max warps per SM=50% occupancy.

   B. 16 blocks with 64 threads each

Possible. 16x64=1024 active threads. 1024/32=32 active warps. 32/64=50% occupancy.

C. 32 blocks with 32 threads each

Possible. 32x32=1024 active threads. 32 active warps. 50% occupancy.

D. 64 blocks with 32 threads each

Possible. 64x32=2048 active threads. 2048/32=64 active warps. 100% occupancy.

E. 32 blocks with 64 threads each

Possible. 32x64=2048 active threads. 2048/32=64 active warps. 100% occupancy.

5) **Consider a GPU with the following hardware limits: 2048 threads per SM, 32 blocks per SM, and 64K (65,536) registers per SM. For each of the following kernel characteristics, specify whether the kernel can achieve full occupancy. If not, specify the limiting factor.**

A. kernel uses 128 threads per block, and 30 registers per thread.

The kernel can achieve full occupancy. 2048/128=16 blocks per SM. 16*128*30=61,440 registers per SM.128/32=4 warps per block. 4*16=64 warps per SM. 100% occupancy.

B. kernel uses 32 threads per block, and 29 registers per thread.

The kernel can not achieve full occupancy. Blocks per SM is the limiting factor. 2048/32=64 blocks per SM is over the maximum of 32 blocks per SM. 32x32x29=29,696 registers per SM. 32/32 = 1 warps per block. 1x32 blocks = 32 warps per SM. 50% occupancy.

C. kernel uses 256 threads per block, and 34 registers per thread.

The kernel can not achieve full occupancy. Registers per SM is the limiting factor. 2048/256=8 blocks per SM. 8x256x34=69,632 registers per SM is over the limit of 65,536 registers per SM. Reducing to 7 blocks per SM. 7x256x34=60,928 registers per SM is within the limit. 256/32 = 8 warps per block. 8x7=56 warps per SM. 56/64=84% occupancy.

Analysis

Calculate the impact of control divergence for the N tiles in the tiled MMULT algorithm.

Assume 16x16 tiles and thread blocks.
Each thread block has 8 warps.
Assume square matrices of 100x100.
Each warp will go through 7 phases.
7x7 blocks.

Type 1 blocks: 42
Type 2 blocks: 7

In the last phase of each Type 1 block, (16x7)-100=12 rows of the block will cross the boundary. This means the last 12x16=192 threads will not pass the conditional boundary check, or 192/32=6 warps. This is an even number of warps, so they won't in fact suffer control divergence at any phase. **0 warp phases will have control divergence** for Type 1 blocks. The **performance reduction will be 0%** for Type 1 blocks.

In each of the first 6 phases of each Type 2 block, the last 12 threads of each row will not pass the conditional boundary check. This will affect every warp, which will each contain 8 threads which do pass the conditional and 24 threads which do not. Each block contains 16x16/32=8 warps, so there will be 8x7=56 warps which suffer from control divergence for each of the first six phases, for a total of 6x56=336 phases with control divergence. In the last phase, the last 12 rows of threads will also not pass the boundary check. This means there will only be 2 warps which suffer from control divergence per block or 7x2=14 phases with control divergence. In total there will be 14+336=**350 warp phases with control divergence** for Type 2 blocks. Out of a total of 8x7x7=392 total warp phases, this makes for a 302/392=**77% performance reduction** for Type 2 blocks.

To calculate total performance impact, we take the total number of warp phases with control divergence (350) and divide by the total number of warp phases 8x7x7x7=2744. **The total performance reduction is 350/2744=12.8%.**

Control Divergence

Based on the configuration of your kernels for control divergence, can you explain how the Nsight Compute metrics can capture or identify when divergence is happening?

Occupancy

Theoretical vs. Achieved Occupancy

For all three conditions (no divergence, single branch, and nested branch divergence), the theoretical occupancy was 100%. I was surprised to see that the achieved occupancy decreased with increased divergence.

The kernel with no divergence had the lowest occupancy of 59.55% and the kernel with nested branch divergence had the greatest occupancy of 62.15%.

Theoretical vs. Achieved Active Warps Per SM

For all kernels, the theoretical active warps per SM was 48. Similar to the occupancy, the kernel with no warp divergence had the least achieved active warps per SM of 28.59, and the kernel with the most divergence had the highest active warps per SM of 29.83.

Analysis

I am hesitant to conclude that divergence is positively correlated to occupancy. I could see in some cases how control divergence could result in greater occupancy, if warps which otherwise would have been blocked (or waiting on a memory access) are active for longer due to the fact that they are busy computing for longer. The resulting ratio of active warps to max warps averaged over time would be higher, but this doesn't improve performance and no time is saved.

Warp State Statistics

Avg. Active Threads Per Warp

All three kernels had an average active thread count per warp of 32.

Avg. Not Predicated Off Threads Per Warp

The kernel with no divergence, and the kernel with single branch divergence both had an Avg. Not Predicated Off Threads Per Warp count of 32 threads. The kernel with nested branch divergence had a count of 27.36.

Analysis

The Avg. Not Predicated Off Threads Per Warp value was the first value that was affected as I would expect it to be. To me, this seems to be the most reliable value to look at when determining control divergence, as it specifies the behavior of threads in relation to warps, rather than the behavior of warps in relation to SMs (occupancy).

MMULT

Host Verification Results

Matrix Size	Max Error
P: 100x100 k=100	3.519E-7%
P: 1000x1000 k=1000	2.660E-7%
P: 2000x3000 k=2500	4.038E-7%

Table 1 Maximum error between host and device calculations for 3 matrix sizes.

Performance Comparison for P: 2000x3000 k=2500

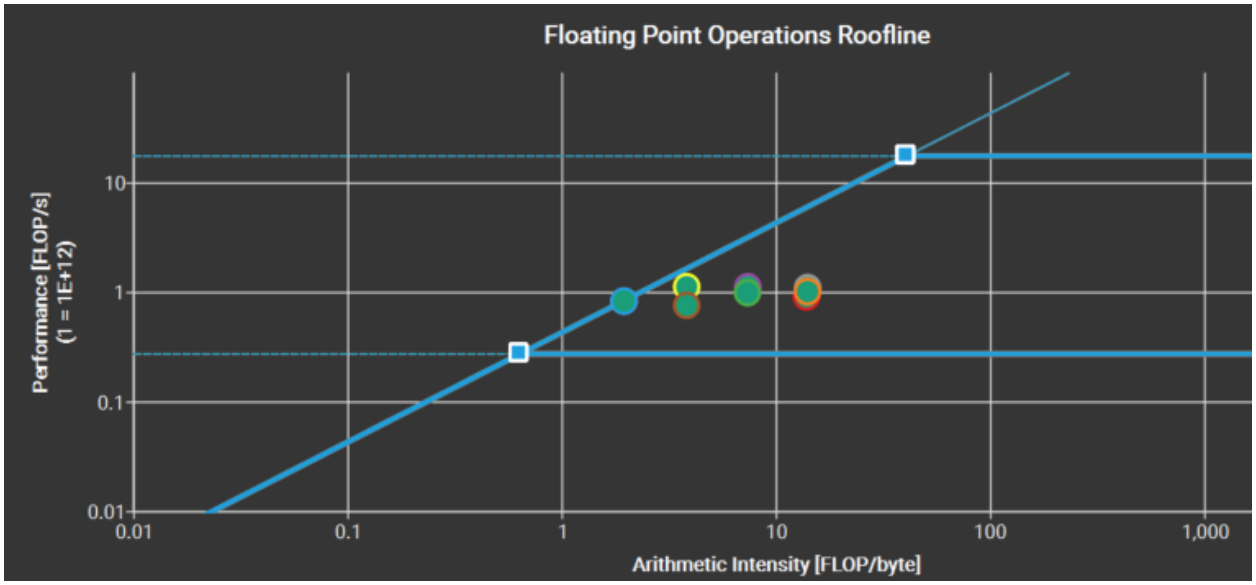

Fig. 1 Roofline chart showing the spread of Arithmetic Intensity between varied kernel runs in table 2 for the 2000x3000 matrix result.

Kernel	AI	Compute Throughput (%)	Execution Duration (ms)	Occupancy (%)
Shared Memory 8x8 Tile	3.87	84.64	40.64	66.74
Shared Memory 16x16 Tile	7.45	98.75	31.21	99.52
Global Memory 16x16 CTA	7.48	99.48	27.41	99.52
Shared Memory 32x32 Tile	14.05	85.92	34.35	66.68
Global Memory 32x32 CTA	14.22	89.74	30.36	66.45
Global Memory 8x32 CTA	14.22	97.44	27.91	99.37
Global Memory 32x8 CTA	3.88	99.40	27.43	99.53
Global Memory 64x4 CTA	1.98	73.61	37.05	98.73

Table 2 Results for various kernels calculating the 2000x3000 matrix result.

Analysis

I was surprised to find that the naive kernels using only global memory often outperformed the kernels using shared memory. I believe the reason is that the hardware for my particular device (GeForce RTX 3070) has been optimized to make good use of the L1 cache. I noticed that the cache hit rate for my shared memory kernels was very nearly 0%, while the global memory counterparts was nearly 100%. In essence, I think both kernels are taking equal advantage of the same physical L1 cache / shared memory on the device.

Kernels with a 32x32 CTA achieved very good arithmetic intensity, since the amount of compute is no different, this must be from improved memory traffic. This makes sense because 32 threads accessing contiguous memory results in coalesced access. In my case, it seems this had a much greater effect on memory performance than the tiling method.

Disabling Fused Instruction Equivalent

Changing the command line argument seemed to have little effect on the performance of the highest performing kernel. The most significant percentage change was in the AI, with a 1.5% increase.

Kernel	AI	Compute Throughput (%)	Execution Duration (ms)	Occupancy (%)
Global Memory 8x32 CTA	14.22	97.44	27.91	99.37
Global Memory 8x32 CTA (fmad=false)	14.44	96.88	28.06	99.42

Table 3 Comparison of the best performing kernel with and without “fused instruction equivalent” disabled.