Part 1

Hardware specifications:
Compute Compatibility: 6.1 (Pascal Architecture)
GTX 1080 Nvidia
GPU name: GP104
CUDA cores: 2560
Base Clock speed: 1708
Memory Bandwidth: 320.3 GB/s GDDR5X
Memory Size: 8 GB
Bus Interface: PCIe 3.0
Bus Width: 256 bit
L1 Cache: 48 KB (per SM)
L2 Cache: 2 MB

Active Blocks Device Limit: 32
Active Warps Device Limit: 64
Active Threads Device Limit: 2048
Threads/Block Device Limit: 1024

Registers/Thread Device Limit: 255
Registers/Block && Registers/SM Limit: 65536

Shared Memory/Block Device Limit: 49152
Shared Memory/SM Device Limit: 98304

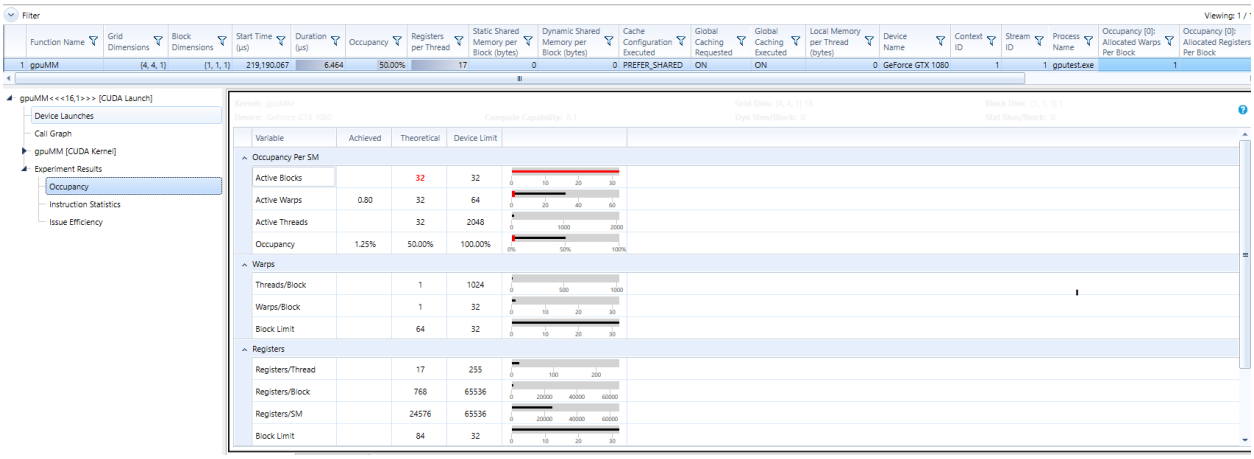
Execution configurations:

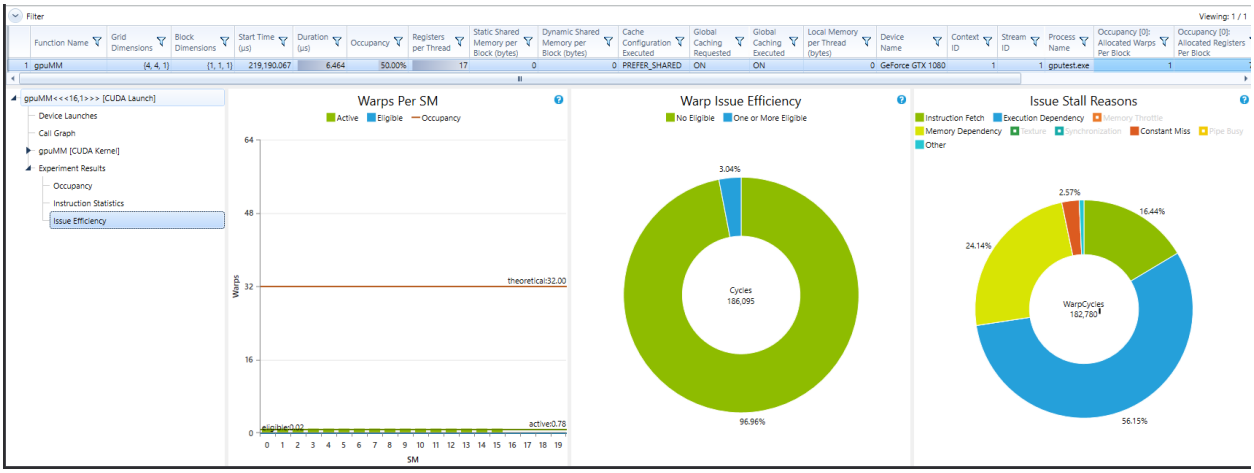
Size: 4 // 1 block // 4 threads
Size: 4 // 4 block // 1 threads
Size: 64 // 2 block // 32 threads
Size: 64 // 32 block // 2 threads
Size: 256 // 8 block // 32 threads
Size: 256 // 32 block // 8 threads

Size: 1024 // 32 block // 32 threads
Size: 1024 // 32 block // 32 threads

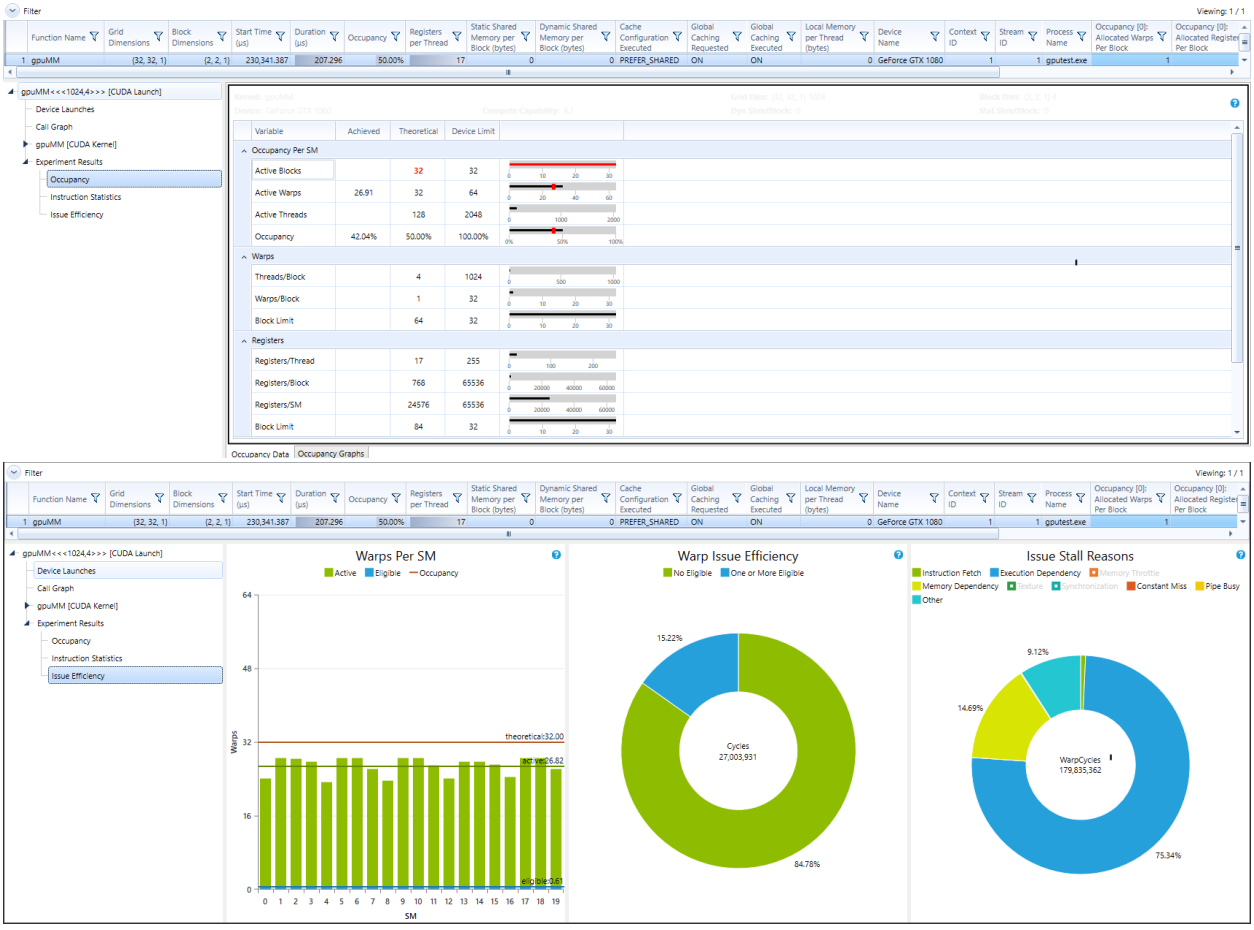
Profiling Results (screenshots)

NAIVE// Size: 4 // 4 block // 1 threads

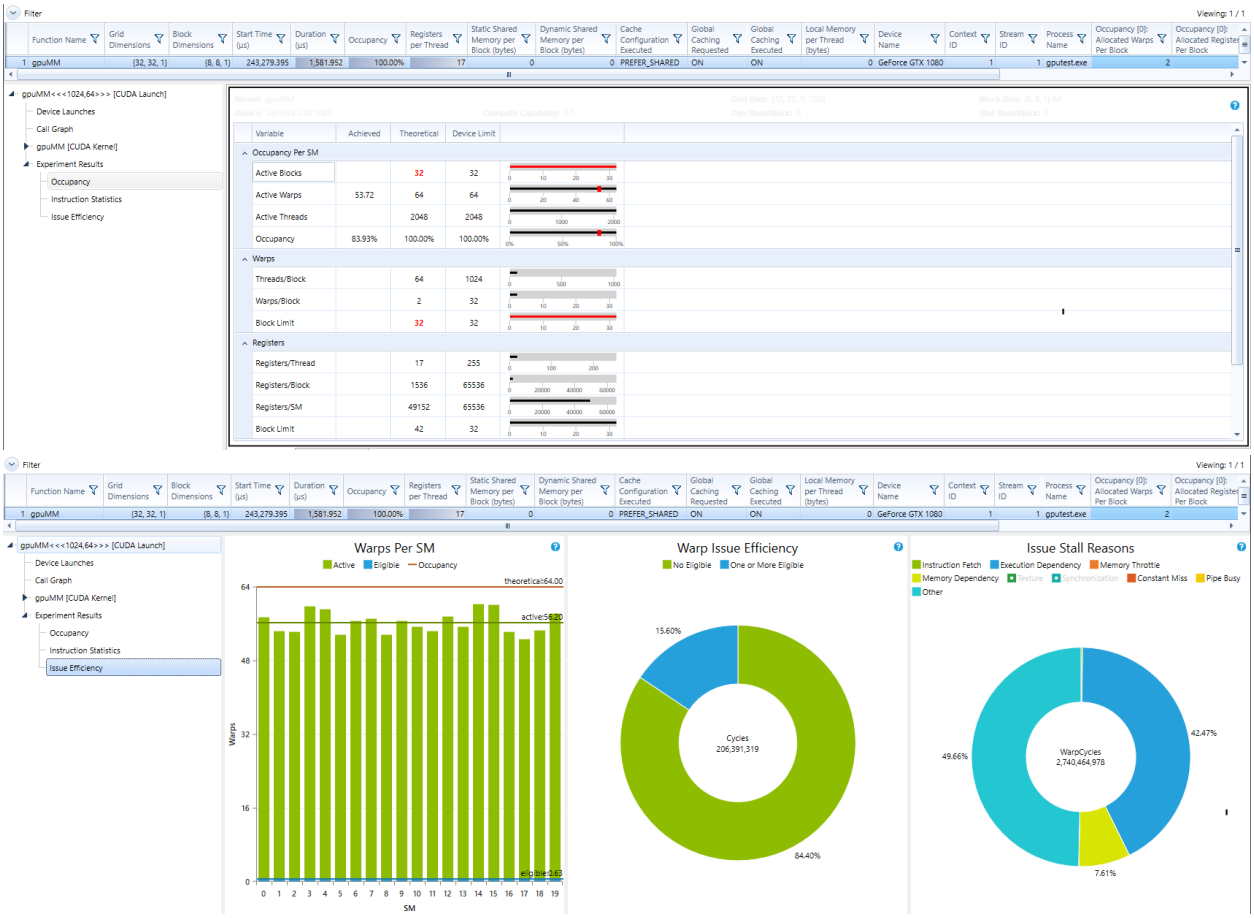




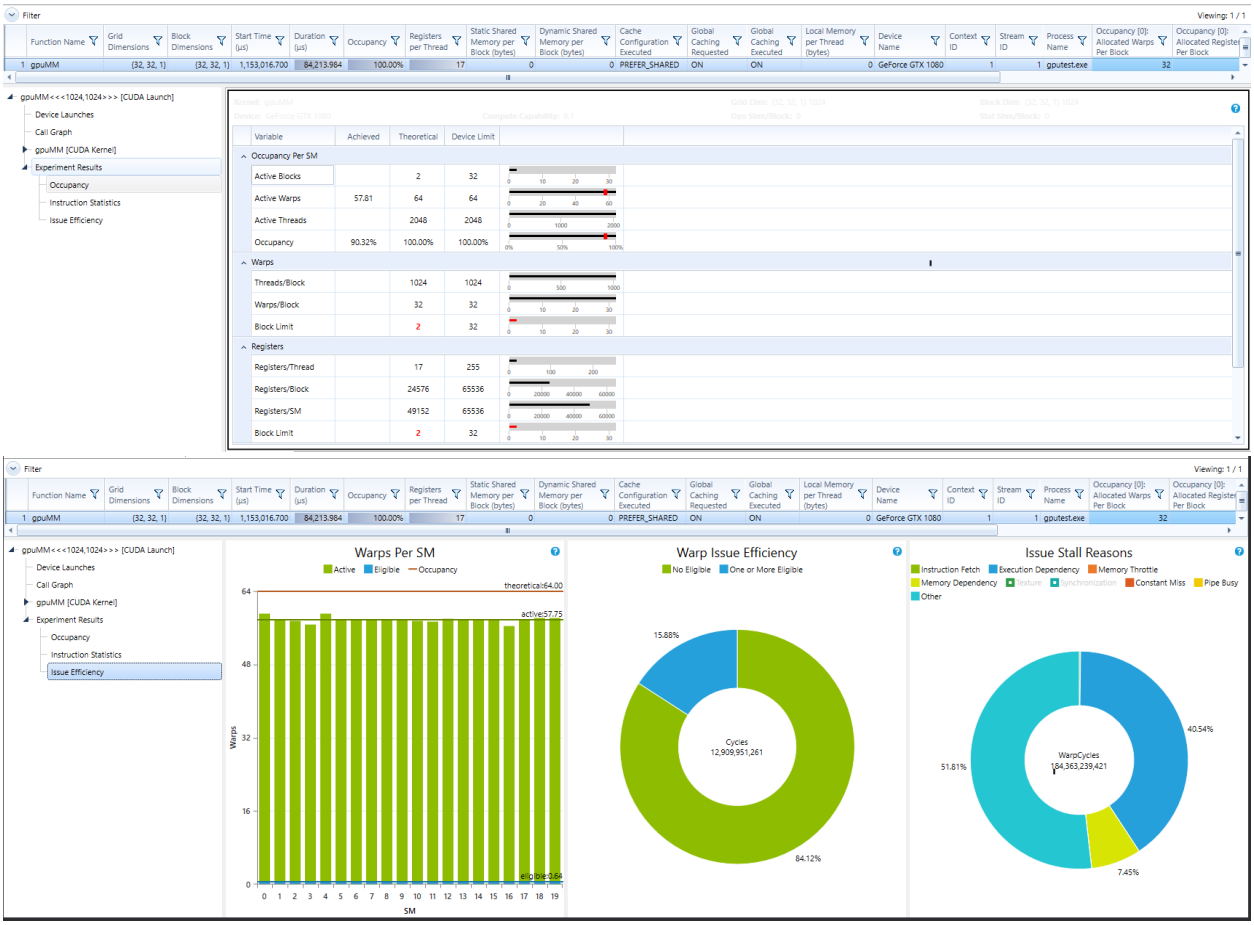
NAIVE// Size: 64 // 32 block // 2 threads



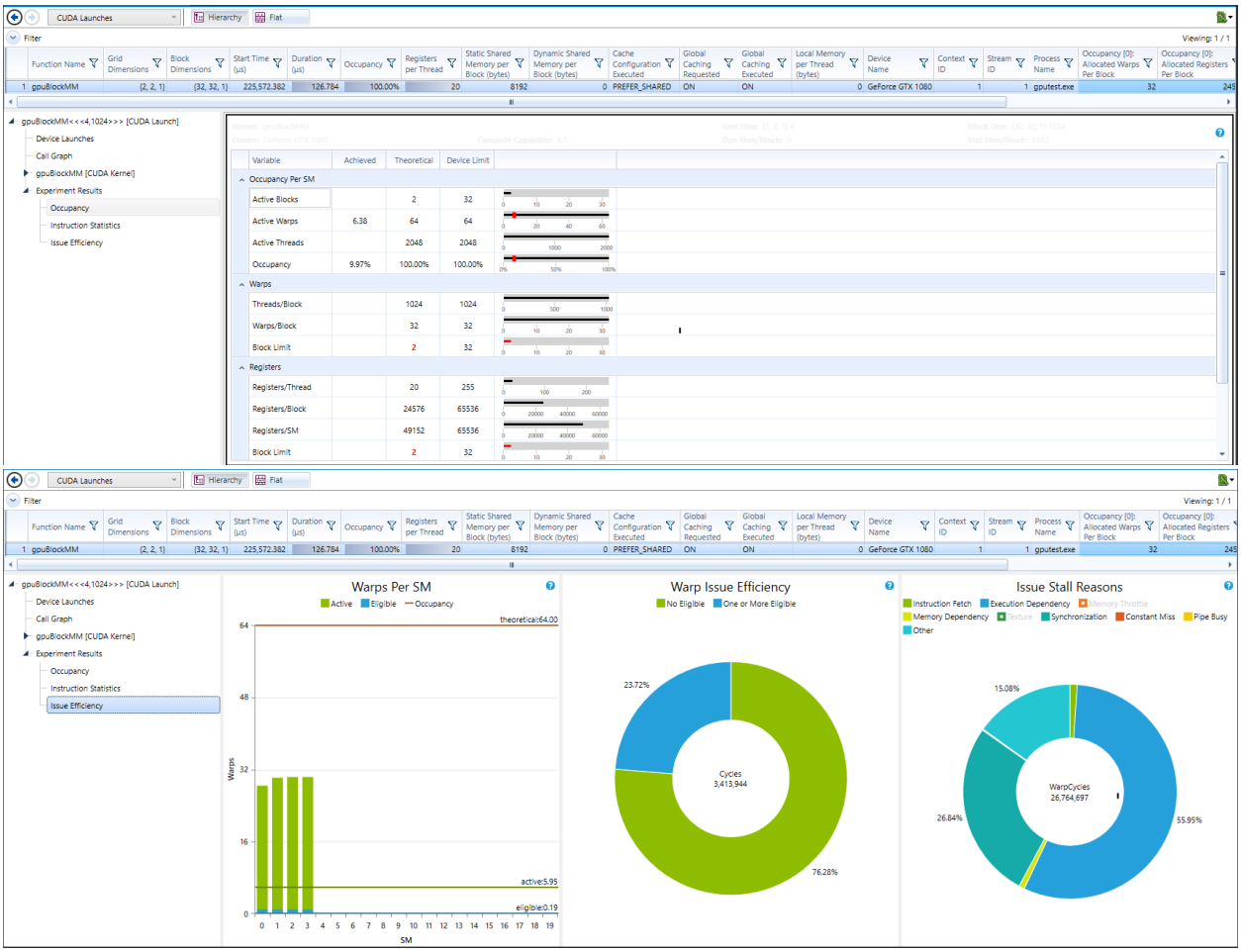
NAIVE// Size: 256 // 32 block // 8 threads



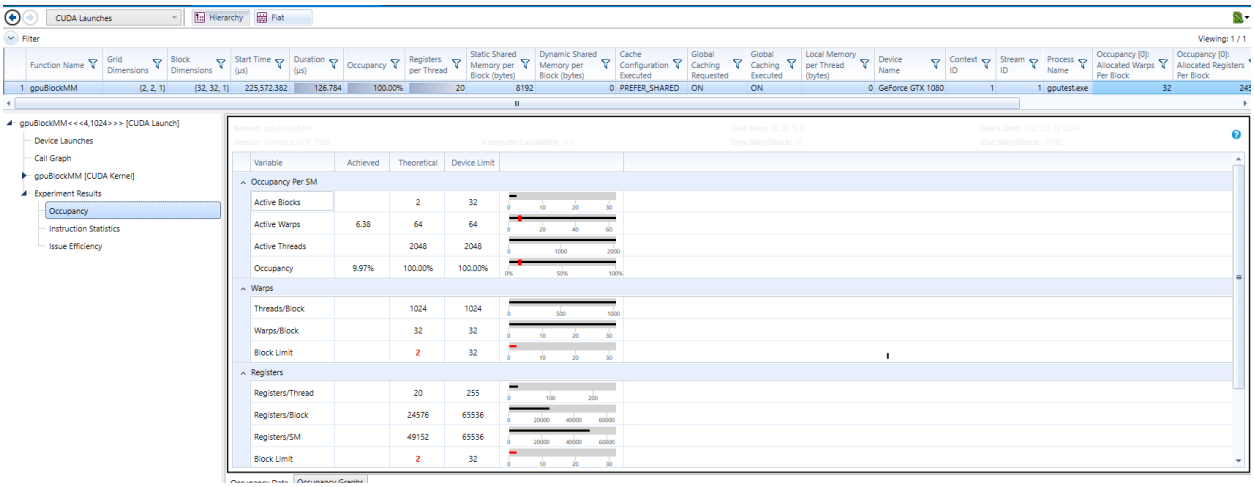
NAIVE// Size: 1024 // 32 block // 32 threads

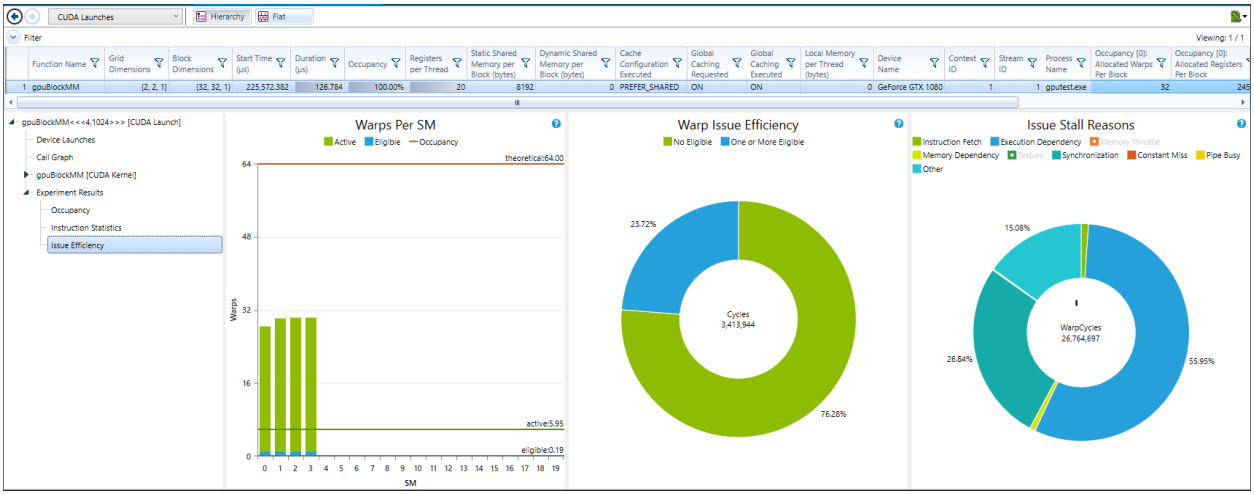


Size: 64 // 2 block // 32 threads

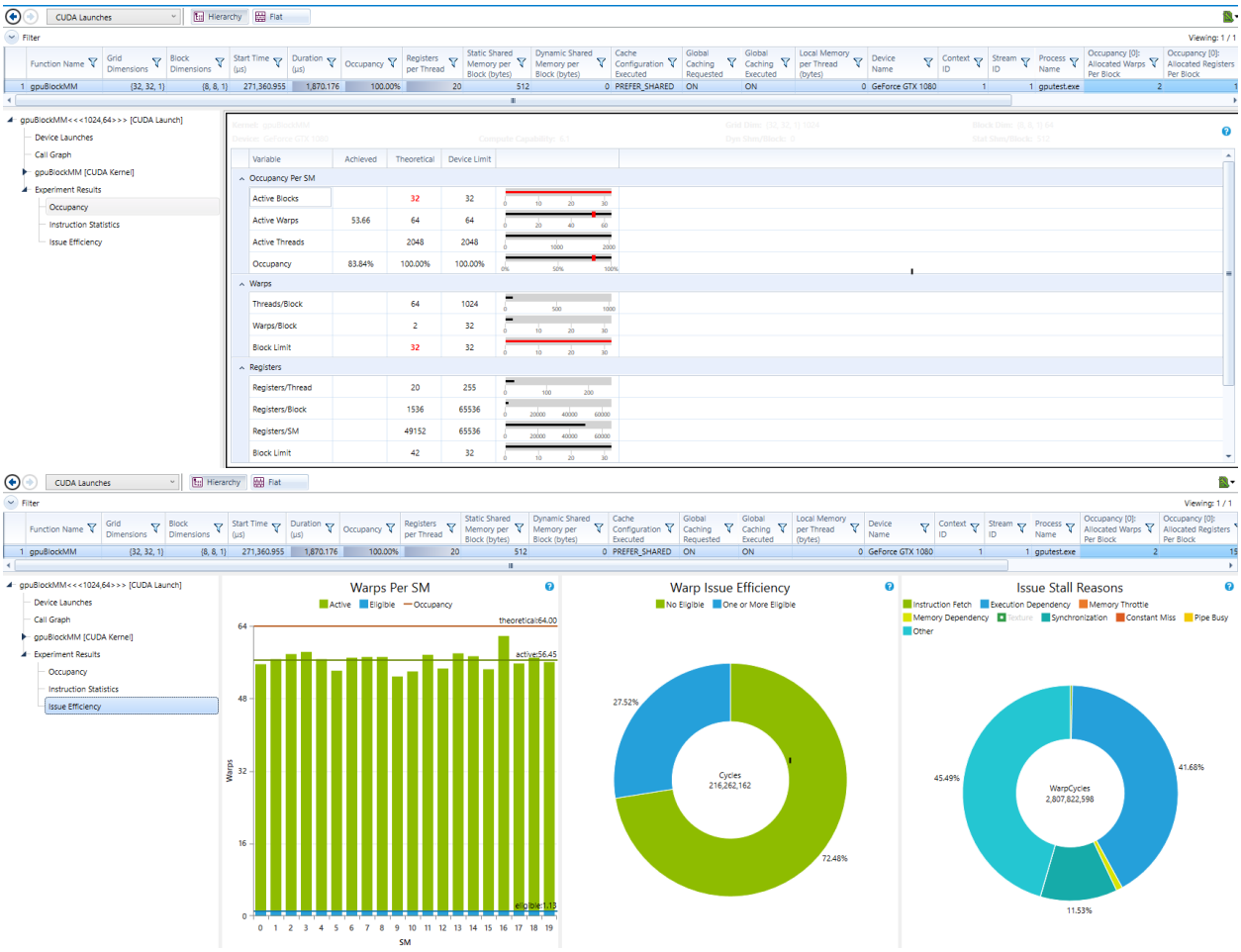


Size: 64 // 32 block // 2 threads

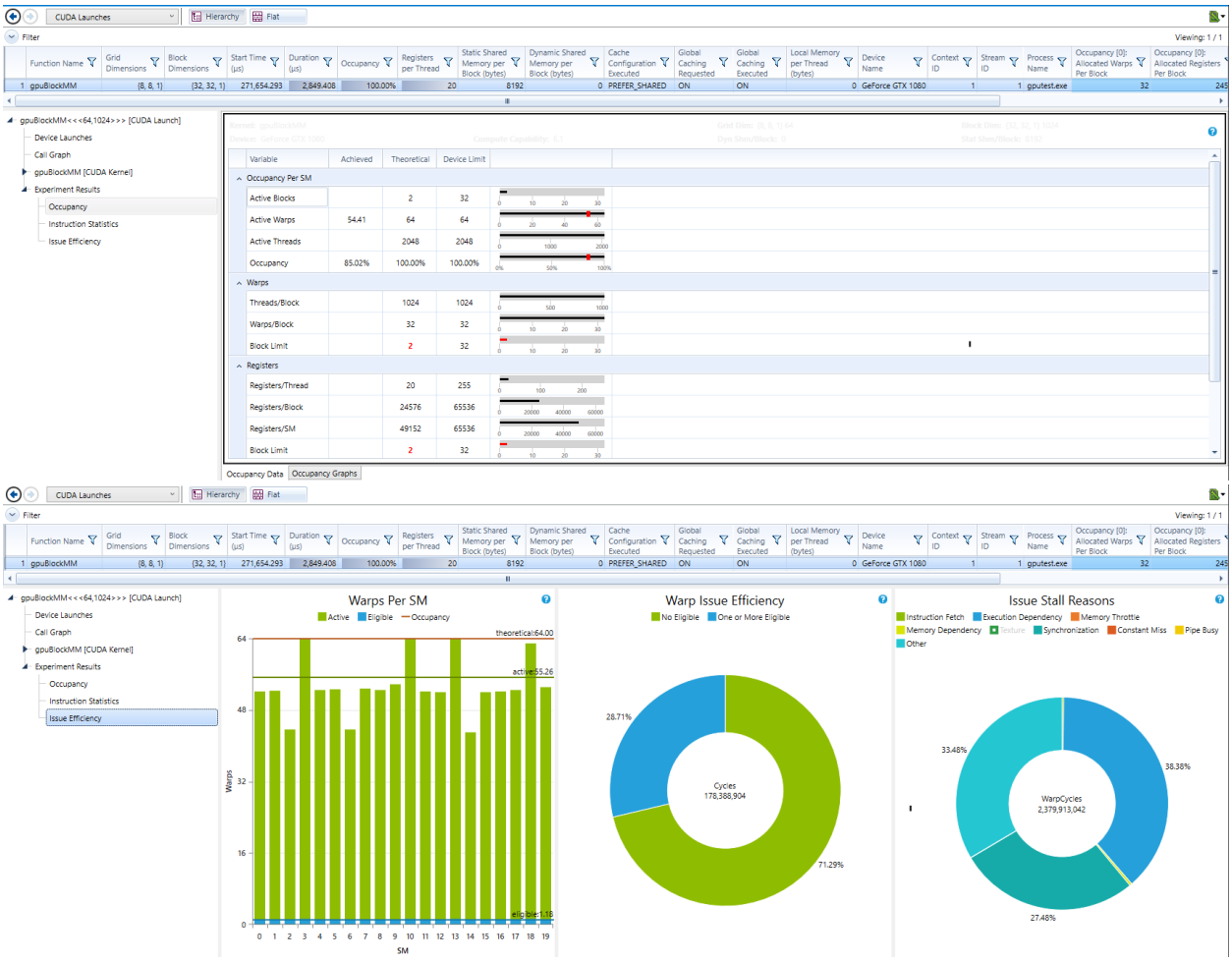




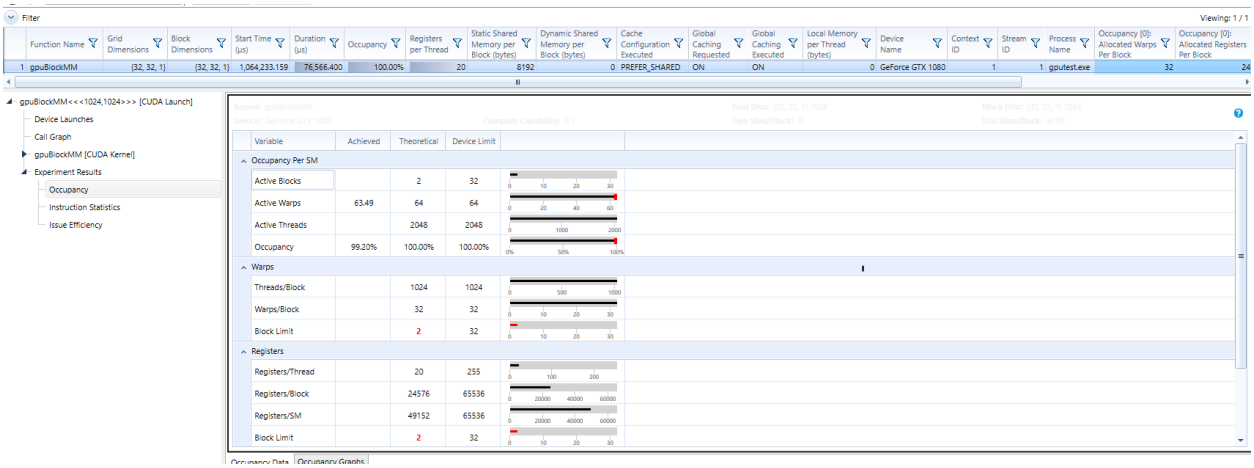
Size: 256 // 8 block // 32 threads

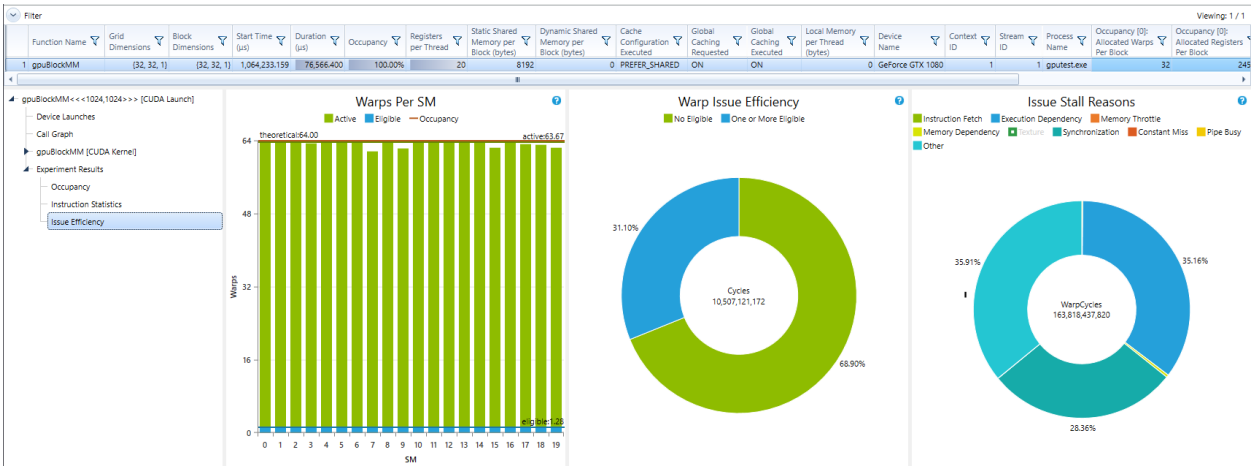


Size: 256 // 32 block // 8 threads

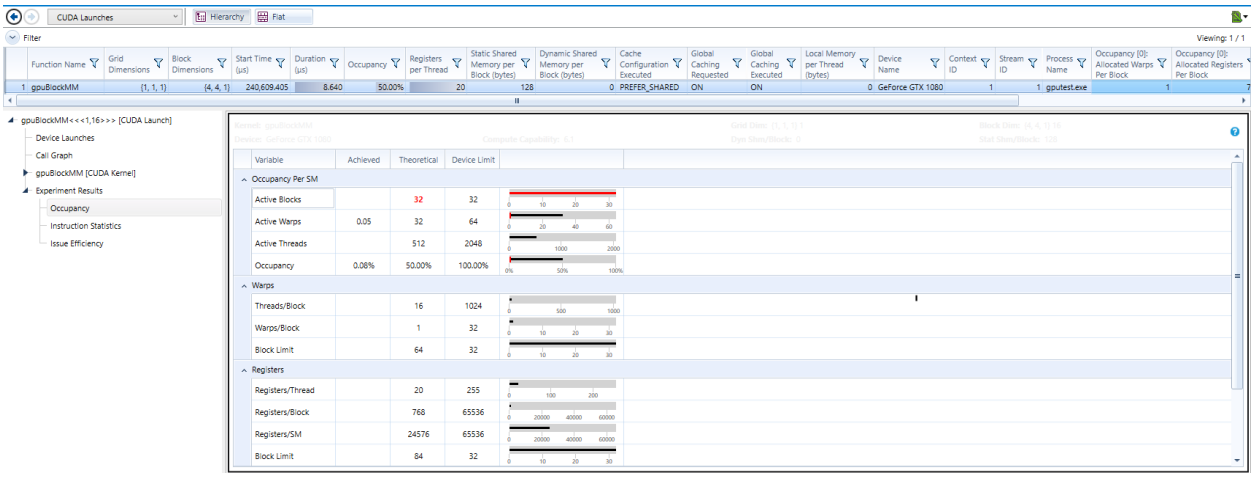


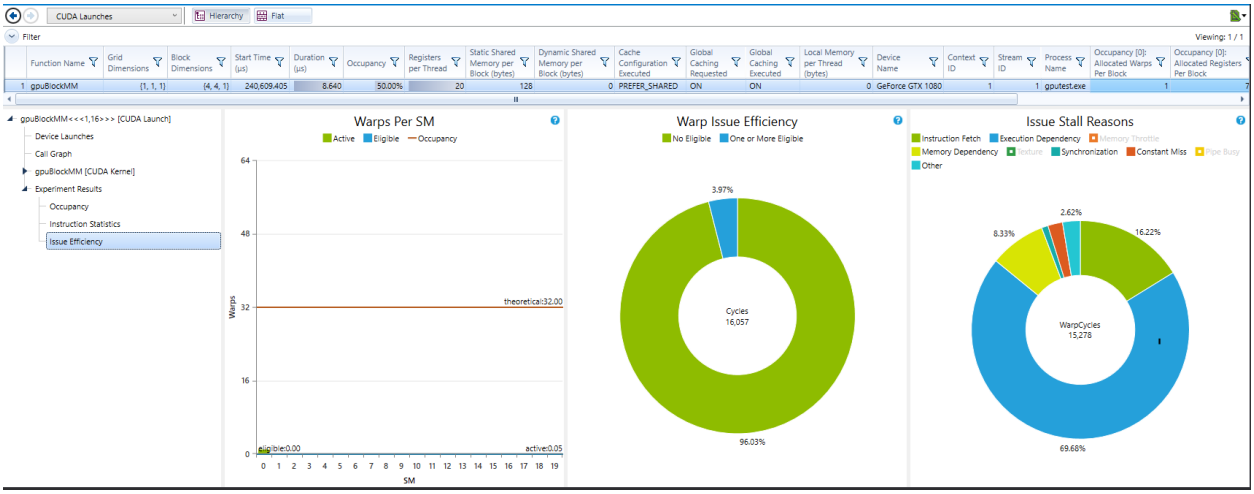
Size: 1024 // 32 block // 32 threads



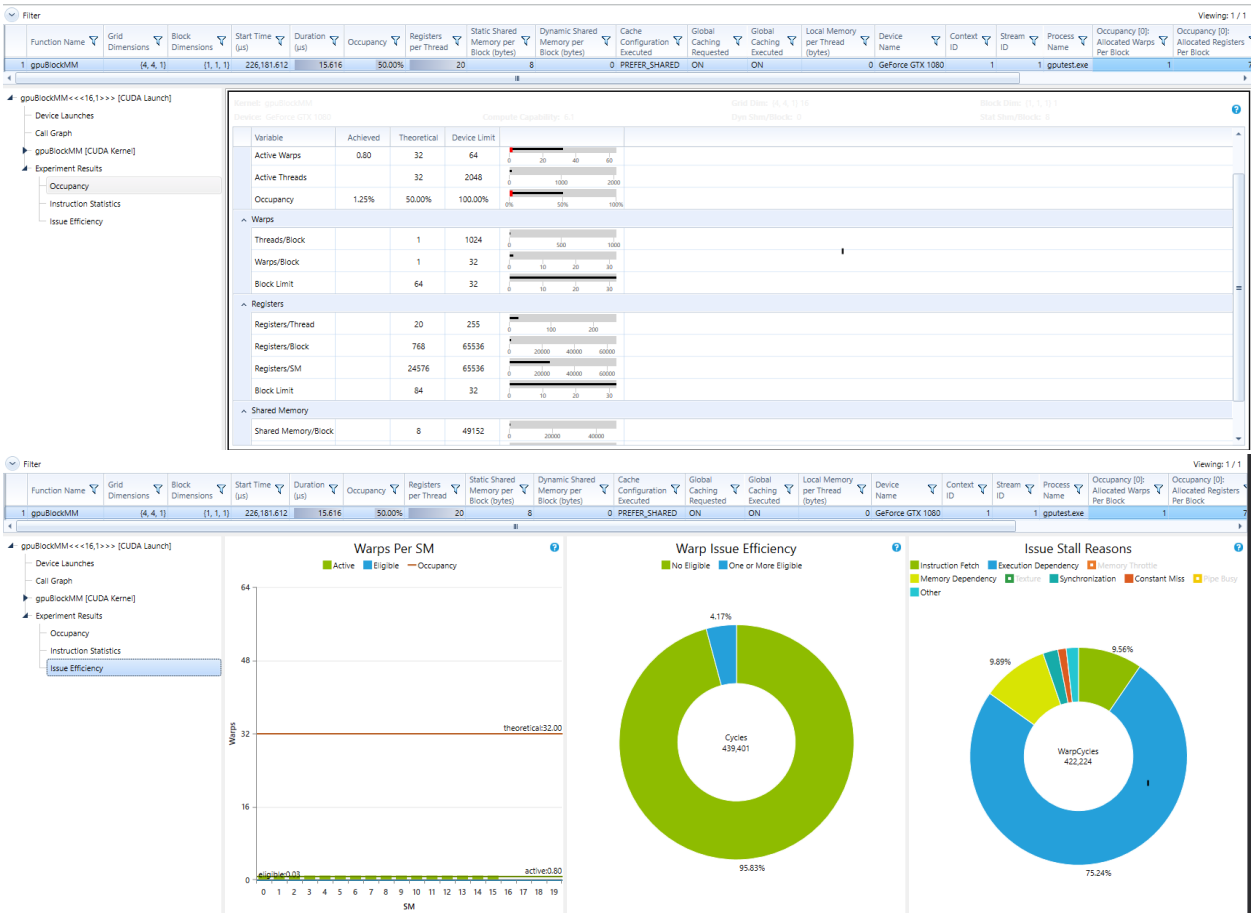


Size: 4 // 1 block // 4 threads





Size: 4 // 4 block // 1 threads



Describe which bottlenecks are an issue

When we were testing to figure out why the values and occupancy was low, we recognized that the values that showed a common trend were the block sizes being lower. This was expressed when the comparison of the CPU implementation vs GPU shared memory, when the sizes of the block and array were small. By effectively increasing register usage and increasing the size of the block, the rate of occupancy was increasing to the point of reaching almost perfect occupancy by trying to reach the theoretical limits of the device. We also recognized the issues of having a bottleneck early on, so we tried to make sure that the Streaming Multiprocessor was fully loaded and allocated enough space for the multiple blocks available to each test case.

Discuss possible optimizations

To have a better peak global memory one thing that can be done is increase the occupancy. Low occupancy can come from code efficiency, so the most efficient code will help give a higher occupancy. This is because the code determines the best ratio of threads and blocks used within the program to determine the best occupancy. The code can be more focused on having more threads rather than more blocks and the program can fail to increase the block size which can lead to lower occupancy levels in the program.

Another way to help reach the peak global memory bandwidth is to increase ILP. Instruction level parallelism(ILP) helps with the peak global memory bandwidth because it helps to improve the performance of the kernels by changing the use of thread local storage. By storing the values in a thread-local location the instruction becomes non-blocking. By becoming non-blocking the process becomes asynchronous and this helps each thread proceed to other instructions without waiting for the load operation. This lowers the number of transactions and hides the latency.