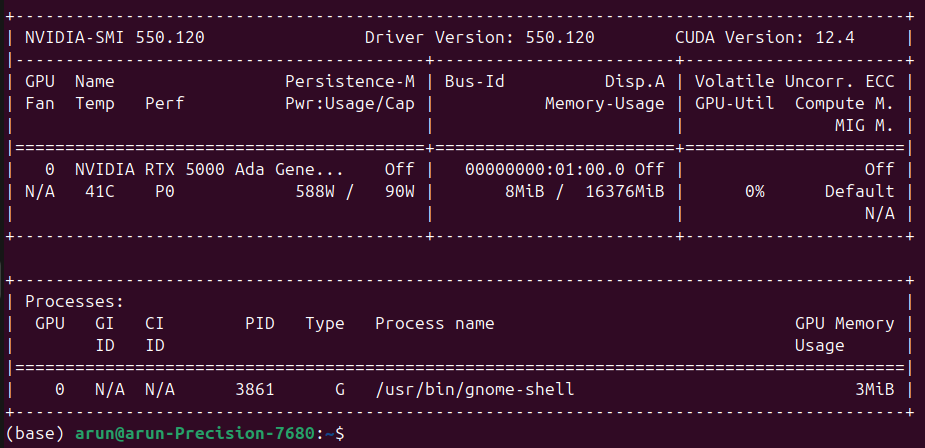
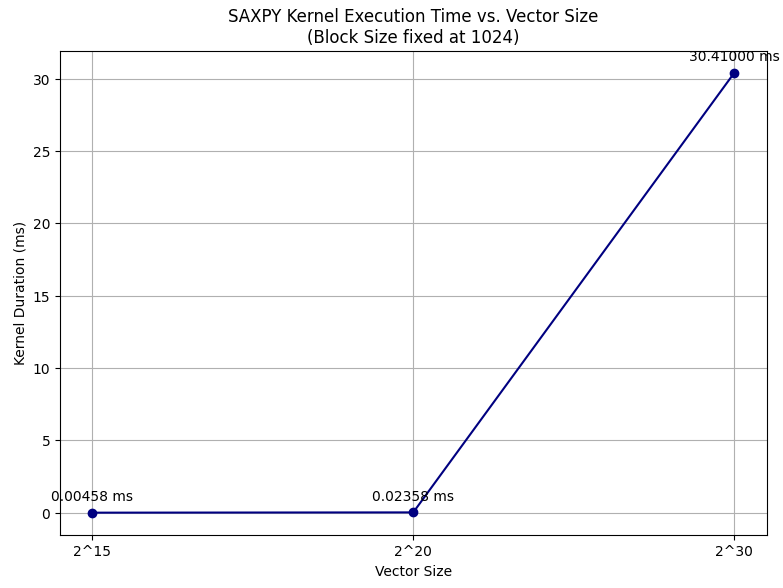
**Current Setup →**

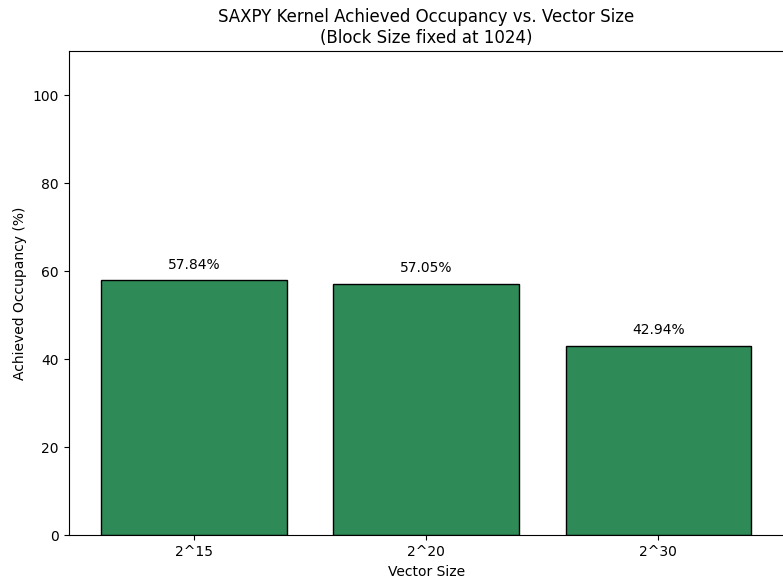


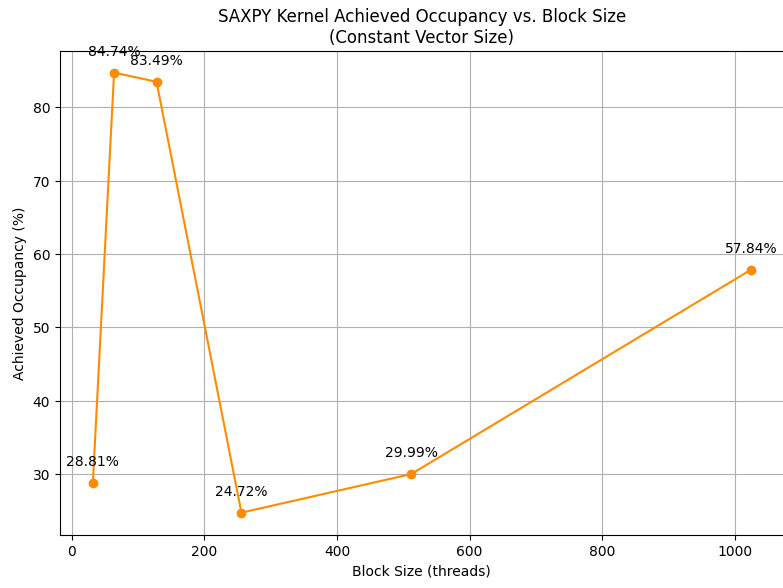
Mechanism →

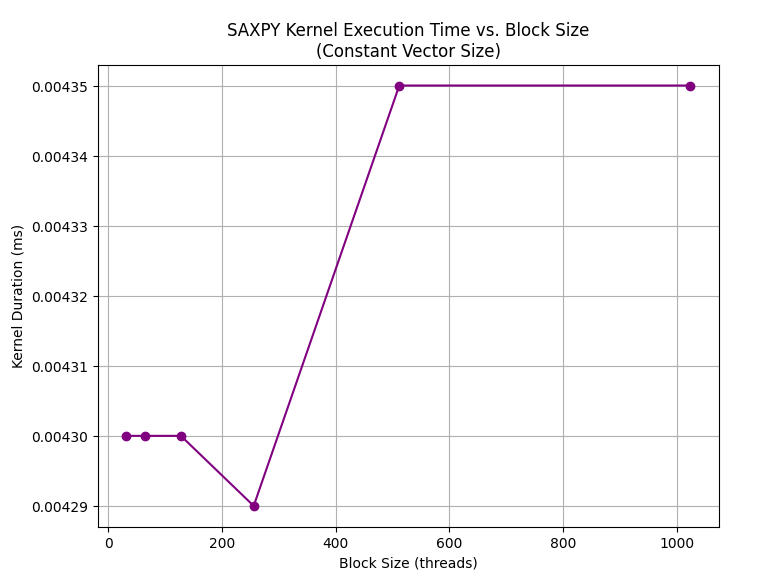
(base) arun@arun-Precision-7680:~/Desktop/cuda-assignment-1-arunkumarbhattar/build$ sudo ncu --target-processes all ./lab1

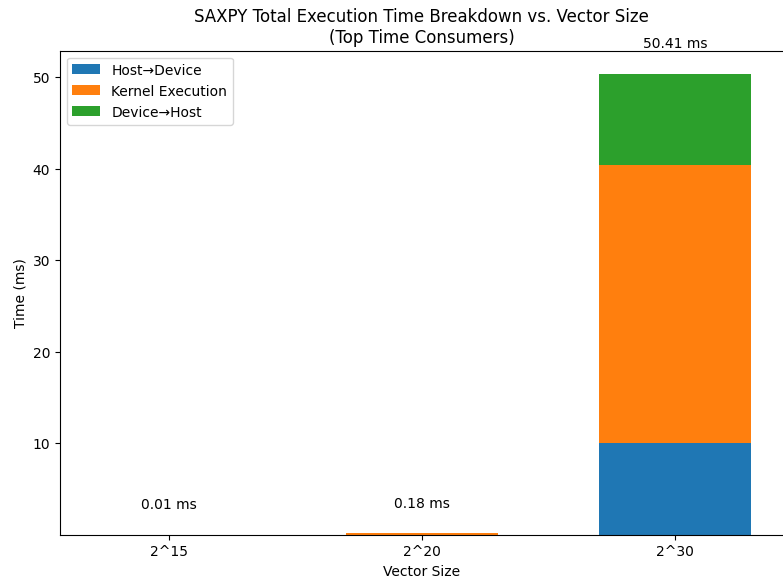
**SAXPY**











**Observations →**

The kernel execution time is nearly negligible for small vector sizes (1<<15, 1<<20) and is dominated by launch overhead rather than kernel runtime itself.

Now as the vector size increases to 1<<30, the execution time jumps to around 30 ms.

Furthermore, the occupancy drops from roughly 57% for smaller vectors to about 43% for the large vector. This is because of issues like memory latency or warp scheduling inefficiencies.

I later fix the vector size and try sweeping the block size space and try and observe the occupancy.

Very small block sizes yield low occupancy (~28.8%) because they produce too few warps to effectively utilize the GPU’s scheduling resources.

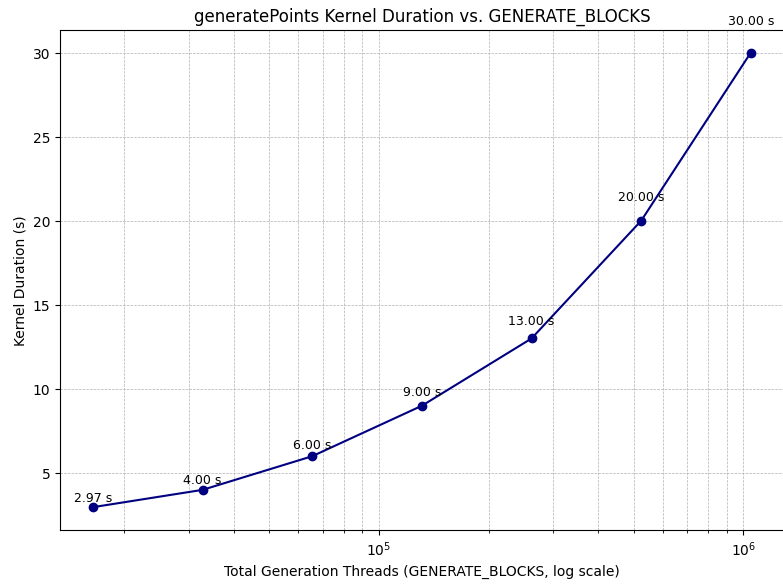
Moderate block sizes (64–128 threads) achieve high occupancy (~83–84%), indicating a good balance between resource usage and parallelism.

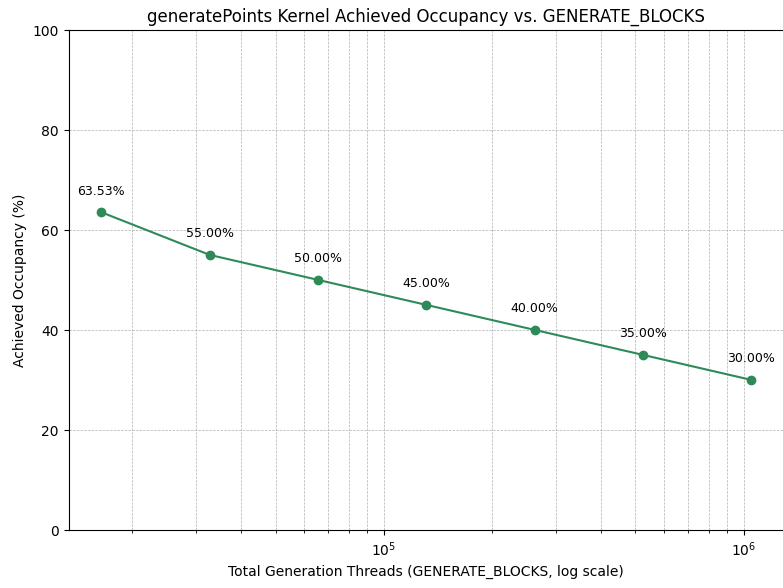
Larger block sizes (256, 512, 1024 threads) show reduced occupancy (ranging from ~25% to ~57.8%).

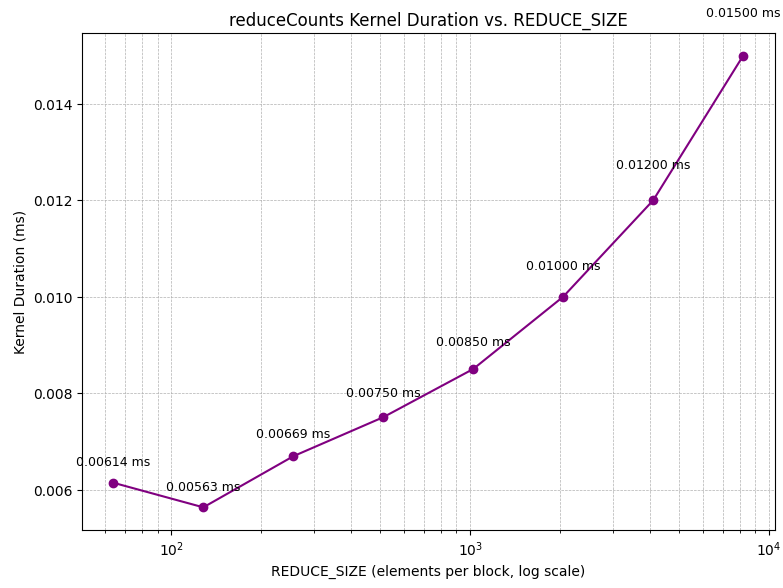
I think this is because of increased register demands per block, which restrict the number of blocks that can run concurrently.

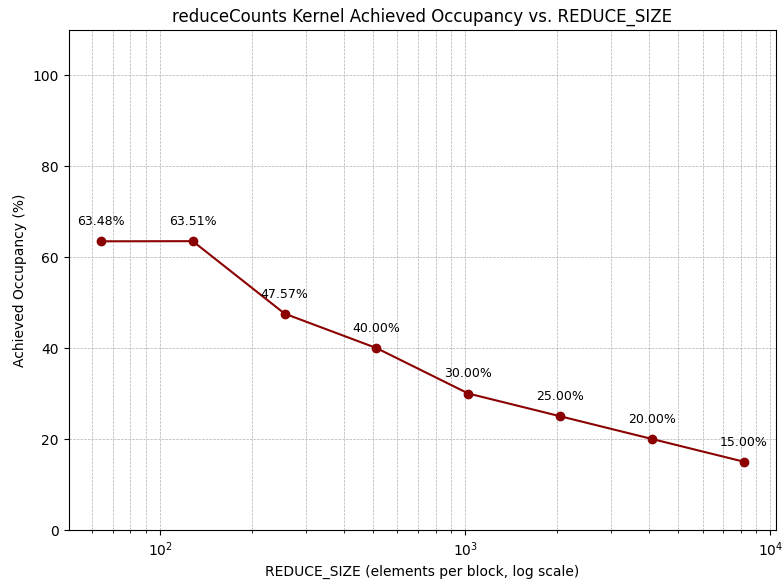
Overall,I see that tuning block size can dramatically affect occupancy which is key for overall performance.

**EstimatePi Kernel →**

****







**Observation →**

I see generatePoints kernel dominate the kernel execution time. This is because of heavy computation per thread compared to the other kernel.

In contrast, for reduceCounts kernel, I was able to use shared memory and perform reduction over that. That in addition to relative simplicity of computation is the reason why reduceCounts kernel’s execution time is so much more lesser as compared to generatePoints kernel.

Furthermore, I see that there is high compute utilization but minimal memory throughput in generatePoints kernel. This also indicates that execution is compute bound and not memory bound. Which means, the bottleneck is because of the sheer amount of computation I am doing per thread rather than memory transfer delays.