CS610: Programming For Performance

Assignment 4
Devesh Shukla (200322)

November 5, 2024

Problem 1

Following are the results for all the kernel versions with N = 64:

Part (i)
Naive CUDA kernel: 0.09744 ms

Part (ii)
In this version, I used a for loop inside the kernel to use each thread for operating on multiple elements ranging from the set {1, 2, 4, 8}. The best times were observed when each thread computed a single element of the *out* array and the following times were reported with this configuration only. For shared memory tiling, I used tiles of size 8 X $TILE\_DIM$ X 8 to be computed by each thread block, where $TILE\_DIM$ took values from the set $\{1, 2, 4, 8\}$.

Shared memory kernel with $TILE\_DIM = 1$: 0.019808 ms
Shared memory kernel with $TILE\_DIM = 2$: 0.01808 ms
Shared memory kernel with $TILE\_DIM = 4$: 0.019392 ms
Shared memory kernel with $TILE\_DIM = 8$: 0.01968 ms

Clearly, all the above tile dimensions give a speedup of around 5 times as compared to the naive CUDA kernel. Moreover, the best kernel speedup was observed with TILE_DIM = 2 i.e., for tiles of size 8 X 2 X 8.

Part (iii)
In this version, I unrolled the for loop inside the kernel from part (ii) 4 times so that each thread computes 4 elements of the *out* array.

Shared memory kernel with loop transformation (TILE_DIM = 1): 0.0256 ms
Shared memory kernel with loop transformation (TILE_DIM = 2): 0.018592 ms
Shared memory kernel with loop transformation (TILE_DIM = 4): 0.018048 ms
Shared memory kernel with loop transformation (TILE_DIM = 8): 0.018944 ms

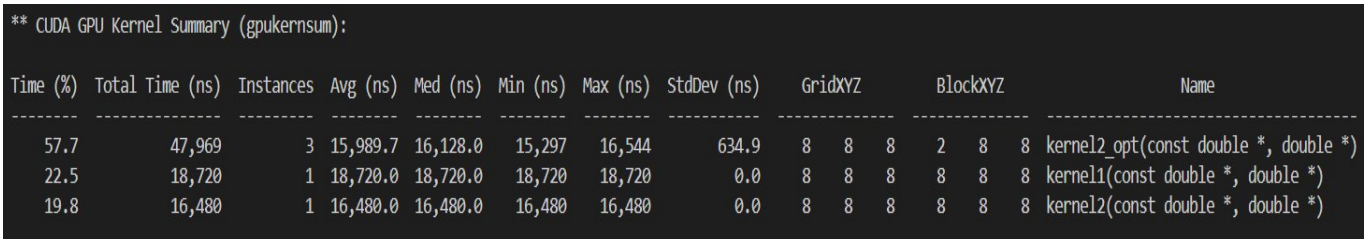
Upon applying loop unrolling transformation, we see minor improvements in the kernel

for bigger tile dimensions while kernel from part (ii) performs slightly better for TILE_DIM equal to 1 or 2. Again, all the tile dimensions give a speedup of around 4-5 times as compared to the naive CUDA kernel.

Part (iv)
Pinned memory kernel: 0.032096 s
This kernel, considered in isolation, performs slightly worse than the kernels from parts (ii) and (iii) but, the overall program runs more than 2x faster while using this kernel as compared to the previous two kernels. This is because using pinned memory reduces the data transfer time between CPU and GPU which is much significant as compared to the kernel computation time in all these cases.

Part (v)
Unified virtual memory (UVM) kernel: 0.02976 ms
This kernel also performs slightly worse than the kernels from parts (ii) and (iii). But overall, the program takes around 1.14 ms as compared to around 1.30 ms for parts (ii) and (iii).

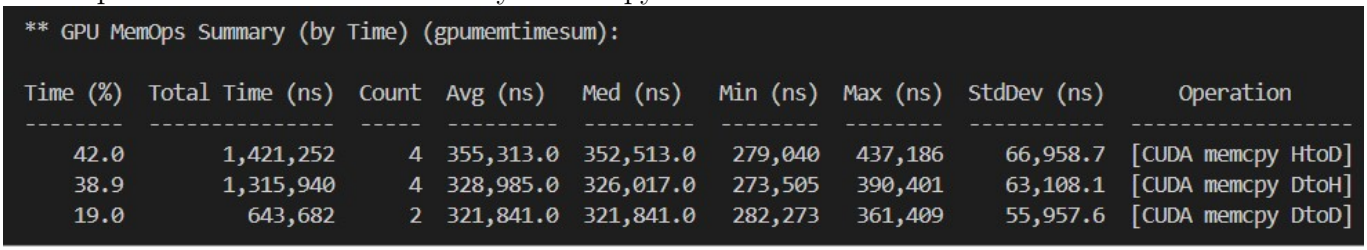
The following image shows the time taken by each kernel during a single execution of the program which launches all 5 versions sequentially. This was obtained using Nsight Systems. As the image shows, kernel2_opt was instantiated 3 three times, once each for each of the last three versions. Moreover, the average time taken by each run of kernel2_opt is found to be lower than those for kernel1 and kernel2 as kernel2_opt is an optimised version of kernel1 and kernel2.



** CUDA GPU Kernel Summary (gpukernsum):

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	GridXYZ			BlockXYZ			Name
57.7	47,969	3	15,989.7	16,128.0	15,297	16,544	634.9	8	8	8	2	8	8	kernel2_opt(const double *, double *)
22.5	18,720	1	18,720.0	18,720.0	18,720	18,720	0.0	8	8	8	8	8	8	kernel1(const double *, double *)
19.8	16,480	1	16,480.0	16,480.0	16,480	16,480	0.0	8	8	8	8	8	8	kernel2(const double *, double *)

The following image shows the time taken for data transfer between CPU and GPU for N = 64. It was also obtained using Nsight Systems. Clearly, the time taken for data transfer between host and GPU is significantly larger as compared to the average time taken by any kernel exclusively. Hence, the overall times taken by almost all the kernels don't differ much as computation time is dominated by data copy time.



** GPU MemOps Summary (by Time) (gpumemtimesum):

Time (%)	Total Time (ns)	Count	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Operation
42.0	1,421,252	4	355,313.0	352,513.0	279,040	437,186	66,958.7	[CUDA memcpy HtoD]
38.9	1,315,940	4	328,985.0	326,017.0	273,505	390,401	63,108.1	[CUDA memcpy DtoH]
19.0	643,682	2	321,841.0	321,841.0	282,273	361,409	55,957.6	[CUDA memcpy DtoD]

Problem 4

GPU server used: gpu3

Part (i)
The following table shows the run times (in milliseconds) for various 2D convolution versions with a convolution filter of dimensions 5 X 5.
N represents the length along each dimension of the input 2D array. t_{serial}, t_{basic} and $t_{optimised}$ represent the times taken by the serial version, the basic 2D kernel version and the optimised 2D kernel version (utilising shared memory) respectively. t''_{basic} and $t''_{optimised}$ represent the times taken by the basic and optimised kernels respectively, without taking into account the cudaMemcpy time between CPU and GPU.

N	t_{serial}	t_{basic}	$t_{optimised}$	t''_{basic}	$t''_{optimised}$
1024	69.4549	8.8927	5.3313	0.158432	0.022336
2048	295.857	17.8878	16.9765	0.166912	0.057984
4096	1680.18	69.462	68.8372	2.66467	0.298368

As is clear from the above table, as N increases the time taken along each column increases due to increase in the computation because of increase in the input array size. Furthermore, the basic and optimised 2D kernel versions are significantly faster than the serial version. Among the kernels, we observe that the time taken exclusively by the optimised kernel (sixth column)is smaller by an order of magnitude as compared to the basic version (fifth column). As N increases, the overall time taken by the basic and optimised kernel versions is nearly the same. This is because for large values of N, the time taken for copying memory between the CPU and GPU dominates significantly over the kernel completion times.

Part (ii)
The following table shows the run times (in milliseconds) for various 3D convolution versions with a convolution filter of dimensions 5 X 5 X 5.
N represents the length along each dimension of the input 3D array. t_{serial}, t_{basic} and $t_{optimised}$ represent the times taken by the serial version, the basic 3D kernel version and the optimised 3D kernel version (utilising shared memory) respectively. t''_{basic} and $t''_{optimised}$ represent the times taken by the basic and optimised kernels respectively, without taking into account the cudaMemcpy time between CPU and GPU.

N	t_{serial}	t_{basic}	$t_{optimised}$	t''_{basic}	$t''_{optimised}$
64	51.806	1.37405	1.33139	0.04144	0.009056
128	586.425	9.3607	9.40582	0.240064	0.01296
256	5966.85	73.1878	70.4974	3.75011	0.678528

Similar to part (i), as N increases the time taken along each column increases due to increase in the computation because of increase in the input array size. Furthermore, the basic and optimised 3D kernel versions are significantly faster than the serial version. Among the kernels, we observe that the time taken exclusively by the optimised kernel (sixth column) is

smaller by an order of magnitude as compared to the basic version (fifth column). As N and hence the size of the input and output arrays increases, the overall time taken by the basic and optimised kernel versions is dominated very heavily by the data transfer time between the CPU and the GPU.