CS336 : Parallel & Distributed processing

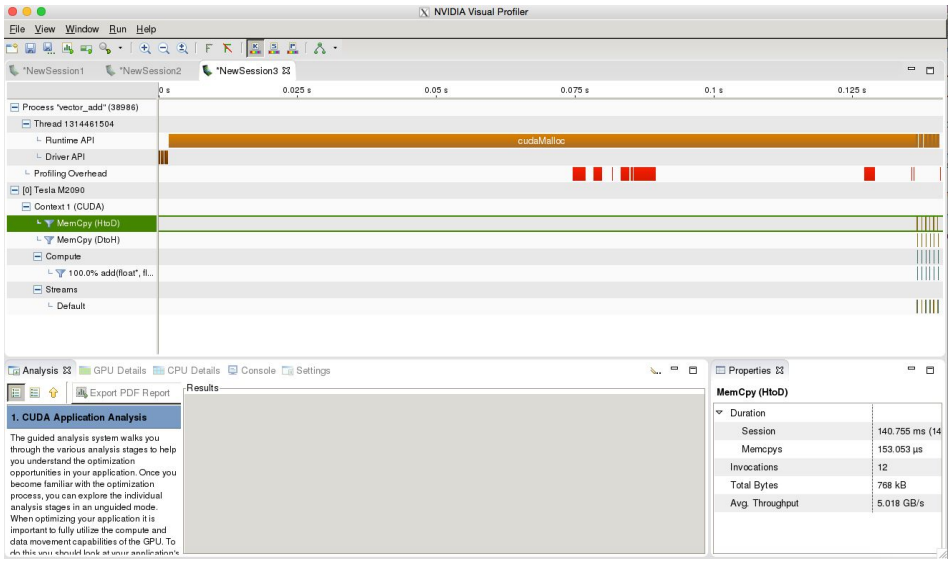
Project 7 Report

Summary of tasks

For this project the main aim was to write a CUDA C program to add two vectors and then to find the dot product of two vectors. The goal of the project is becoming familiar with the basic CUDA C components and with memory-handling.

Tasks

1. Vector addition
   a. Wrote an vector add global function for the device for my cuda
   b. I ran my main program for vector addition six times, and the longest process is the first CUDAmalloc. The first malloc seems independent of size, as the other CUDAmalloc are approximately the same size. As seen from the print statement, the first main program to run takes the longest time.




```
[mggamedz@n1 ~/proj07]$ ./vector_add
We did it!

Run time:  0.033728 ms

We did it!

Run time:  0.012864 ms

We did it!

Run time:  0.012672 ms

We did it!

Run time:  0.012736 ms

We did it!

Run time:  0.012672 ms

We did it!

Run time:  0.012704 ms

[mggamedz@n1 ~/proj07]$
```

c. From my data attached below for run time in seconds for 2048 additions, using about a quarter of the thread limit [128] minimises the run time, and thus, depending on the size of the problem, midrange threads from my data seem to give the optimal solution.

Num Blocks	Num Threads	Run Time
2	1024	0.012864
4	512	0.012224
8	256	0.012224
16	128	0.012128
32	64	0.012512
64	32	0.01232
128	16	0.012608
256	8	0.013344
512	4	0.015008
1024	2	0.017408
	lowest	0.012128

d. Using 90000 blocks, above the 65535 limit, I get errors. Using more threads than 1024 threads, my code also breaks, as that raises error during compile time.

2. I created a CUDA dot product global function using the naive strategy and tree structure to sum the products. Timing the two versions, I got this data:

N=33 * 1024	Time (s)	
threads per block	Naive strategy	tree structure
1	2.538784	0.533344
2	1.456704	0.283808
4	0.904128	0.155392
8	0.631744	0.092928
16	0.492192	0.059456
32	0.260512	0.046368
64	0.139488	0.042112
128	0.080096	0.031488
256	0.065056	0.030464
512	0.063584	0.031904
1024	0.065728	0.034656
2048	0.013312	0.009472
8192	0.014144	0.009216

The Naive strategy for counting is significantly slower than the tree structure for counting and I had no case where the scales turned and thus my conclusion is that the tree structure is the optimal solution.

Extensions

None

Collaborators

I worked alone.