Answers.md

CO332 - Heterogenous Parallel Computing

Assignment 2


```
Sagar Bharadwaj - 15CO141
Aneesh Aithal   - 15CO107
```


Q1

1. N operations to add N numbers.
2. 2 * N reads (N reads for one vector)
3. N writes (For the final vector)
4. Thrust allows you to implement high performance parallel applications with minimal programming effort through a high-level interface that is fully interoperable with CUDA C.

Q2

1. Approximately 9 * Width * Height * (No. of channels) additions + Width * Height * (No. Of Channels) divisions.
2. 9 * Width * Height reads. If each color is read separately then 9 * Width * Height * (No. Of Channels)
3. Width * Height writes
4. Shared memory can be used to reduce global memory accesses. The stencil can be stored in shared memory

Q3

1. Width * Height * 5 operations. 3 multiplications and 2 additions per RGB pixel.
2. 2D matrix is better for applications like color conversion where data from all the three channels are required for every pixel. However for applications like blurring an image where data from different channels are required separately, it's convinient to store them in different layers and parallelly process them separately.
   An example where 3D matrix is better would be an image manipulation software working separately on different color channels.
3. Width * Height reads. If each color is read separately, then Width *Height * (No. Of Channels).
4. Width * Height writes.
5. SIMD instructions can be implemented. Branching statements like if can be removed and two kernel functions can be used instead.

Q4

K is the common dimension of the two matrices. i.e. M x K and K x N are the dimensions of the input matrices.

1. M * N * (2K-1) floating point operations. K multiplications and K-1 additions per cell in the output matrix.
2. 2 * M * N * K reads.
3. M*N writes.
4. This can optimized by using shared memory to implement tiled matrix multiplication.

Q5

1. M * N * (2K-1) floating point operations. K multiplications and K-1 additions per cell in the output matrix.
2. M * N * K/(Tile Width) global memory reads. Tiling reduces global memory reads by a factor of the tile width.
3. M*N writes.
4. The Multiplications can also run parallely. The sum can be then calculated using parallel reduction for each cell in the output matrix.
5. The major difficulty involved working with border cases. Finding when the threads should be synced was also difficult.
6. Matrix can be divided into tiles with dimensions smaller than the thread limit. Multiple kernels calls can then be made to facilitate complete matrix multiplication.
7. A version of tiled matrix multiplicaton can be used. Load only a tile onto the global memory and split it into further sub-tiles using shared memory. Call the kernel several times to operate on each tile.