HPC ASSIGNMENT - 2

Aswanth P P-15CO112
Arvind Ramachandran -15CO111
Date : 18/07/2017

QUESTION 1

Q1. 1. Number floating operations performed in your vector add kernel =
One operations per thread if size of array is n. My kernel will create n blocks each performs one additon operation in ALU. In total n floating operation in kernel.

2. Number of global memory reads are being performed by your kernel =
if the size of array is n number of global reads will be 2n because each input element should be read.

3. Number of global memory writes are being performed by your kernel =
if the size of array is n number of global writes will be n .each element will be copied back to host memory.

4. Since thrust contains built in function to add vectors it is easier to write kernel code .

QUESTION 2

Q2. 1. The kernel does around BLUR_SIZE * BLUR_SIZE * 4 floating point operations per pixel of image.

2. The kernel performs BLUR_SIZE * BLUR_SIZE * 4 global memory reads per pixel of image.

3. The kernel performs 1 global memory write operation per pixel.

4. We could break the image into tiles, and store the tiled image in shared memory. In this way, global memory reads would reduce.

QUESTION 3

Q3. 1. For every pixel, 2 floating point operations are performed.

2. Interleaving may take up more space due to the alignment needs that exist.

3. 3 global memory reads per pixel.

4. 1 global memory write per pixel.

5. The image in the tiles for each block can be stored in the shared memory. In this way, we can reduce the global memory reads.

QUESTION 4

Q4. 1. N Multiplications and N Addition are being performed by the kernel.

2. 2*N Global reads are being performed by the kernel.

3. 1 Global write is being performed by the kernel.

4. Since the kernel is using a large number of redundant global memory accesses, if we can move the matrices to the shared memory, we could see a speed up in performance.

QUESTION 5

Q5. 1. N Multiplications and N Addition are being performed by the kernel. These happens in sets of <thread_count> number of operations, as is specified by the tiling.

2. 2*N/<thread_count> Global reads are being performed by the kernel. This is because each thread loads a single value in every tile.

3. 1 Global write is being performed by the kernel. This occurs once the final value at that coordinate is computed.

4. It has been shown that unrolling loops can improve performance. Thus, by using separate kernels, one for handling general case(unrolled loop), and others for handling corner cases can be utilized.

5. The issues during implementation were handling corner cases, and determining the loops for handling the data being obtained from the global memory using tiling.

6. Use of Communication-avoiding and distributed algorithm (Cannon's algorithm), which partitions each input matrix into a block matrix whose elements are sub matrices of size $\sqrt{M/3}$ by $\sqrt{M/3}$, where M is the size of fast memory. The naive algorithm is then used over the block matrices, computing products of sub matrices entirely in fast memory. This reduces communication bandwidth to $O(n^3/\sqrt{M})$, which is asymptotically optimal (for algorithms performing $\Omega(n^3)$ computation)

7. Splitting the matrix into a grid will allow the product of each grid to be computed independantly, although this requires that the global memory can atleast store 2 matrices of 1*N, where N is the height/width of the matrices to be multiplied.

ALGORITHM :

```
gridx=width/subimage_width;
gridy=height/subimage_height;

for i from 1 to gridy
  for j from 1 to gridx
       mult(image1[0 to width][i*subimage_height to i+1*subimage_height], \
                    image2[j*subimage_widthto j*1*subimage_width][0 to height]
   end
end
```