Parallel Programming Assignment 3 (CUDA)

Team Members
Sandeep N Menon : 14CO140
Matthew George : 14CO225

Q1

Questions and Answers :

1) What is the architecture and compute capability of your GPU?

A. Architecture = Kepler
Compute Capability = 3.5

2) What are the maximum block dimensions for your GPU?

A. Maximum dimension 0 of block: 1024
Maximum dimension 1 of block: 1024
Maximum dimension 2 of block: 64

3) Suppose you are launching a one dimensional grid and block. If the hardware's maximum grid dimension is 65535 and the maximum block dimension is 512, what is the maximum number threads can be launched on the GPU?

A. 512

4) Under what conditions might a programmer choose not want to launch the maximum number of threads?

A. The number of threads required for the execution of the given code is much lesser than the maximum number of threads.
For e.g., addition of two 1-dimensional arrays, each containing 10 elements requires only 10 threads. (Each thread will perform one addition operation.)
The remaining threads which are spawned consumes some time,causing unnecessary performance losses.

5) What can limit a program from launching the maximum number of threads on a GPU?

A. The number of threads that can be launched depends on the amount of resources used by each thread. SMs have finite resources of threads and local memory. If the amount of resources used per thread is high, we may not be able to hit the theoretical maximum no. of threads.

6) What is shared memory? How much shared memory is on your GPU?

A. Every thread in its block shares a memory, but threads cannot see or modify the copy of this variable that is seen within other blocks. This provides an excellent means by which threads within a block can communicate and collaborate on computations. Furthermore, shared memory buffers reside physically on the GPU. Because of this, the latency to access shared memory tends to be far lower than typical buffers, making shared memory very effective.

Total shared memory per block = 49152 bytes

7) What is global memory? How much global memory is on your GPU?

A. Threads always have the option of writing to and reading from global memory. Global memory is much slower than accessing shared memory; however, global memory is much larger.

Total global memory = 3405643776 bytes

8) What is constant memory? How much constant memory is on your GPU?

A. We use constant memory for data that will not change over the course of a kernel execution. In some situations, using constant memory rather than global memory will reduce the required memory bandwidth. The constant memory space is cached. As a result, a read from constant memory costs one memory read from device memory only on a cache miss; otherwise, it just costs one read from the constant cache.

Total constant memory = 65536 bytes

9) What does warp size signify on a GPU? What is your GPU's warp size?

A. Warp size is the number of threads in a warp, which is a sub-division used in the hardware implementation to coalesce memory access and instruction dispatch.

Warp Size = 32 threads

10) Is double precision supported on your GPU?

A. Yes.
Peak Double Precision floating point performance (GFLOP) = 1.43 Tflops

Q3

Questions :

1. How many floating operations are being performed in the matrix addition kernel?
2. How many global memory reads are being performed by your kernel?
3. How many global memory writes are being performed by your kernel?

Answers:

Let the no. of rows and no. of columns be 1000 each
Total no. of threads = 1000*1000 = 10^6

1) Total no. of floating point operations:

```
__global__ void MatrixAddKernel(int* mat1_d, int* mat2_d, int* sum_d, int nrow, int ncol)
{
        int row=blockIdx.y*blockDim.y+threadIdx.y;          -> 2
        int col=blockIdx.x*blockDim.x+threadIdx.x;          ->2

        if(row<nrow && col<ncol)
```

sum_d[row*ncol+col]=mat1_d[row*ncol+col]+mat2_d[row*ncol+col];
-> 4*3 [2 for row*nrow+col, 2 for memory acces]

}

Total no. of floating point operation per thread = 2+2+12=16
Total no. of floating point operations = 16*10^6

2) No. of global memory reads per thread = 2
Total = 2*10^6

3) No. of global memory writes per thread = 1
Total = 10^6

Q4

Questions

1. How many floating operations are being performed in your color conversion kernel?
2. How many global memory reads are being performed by your kernel?
3. How many global memory writes are being performed by your kernel?
4. Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.

Answers

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
In this way, global memory reads would reduce.

Q5

Questions.

1. How many floating operations are being performed in your color conversion kernel?
2. Which format would be more efficient for color conversion: a 2D matrix where each entry is an RGB value or a
3D matrix where each slice in the Z axis representes a color. I.e. is it better to have color interleaved in this
application? can you name an application where the oposite is true?
3. How many global memory reads are being performed by your kernel?
4. How many global memory writes are being performed by your kernel?
5. Describe what possible optimizations can be implemented to your kernel to achieve a performance speedup.

Answers:

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.

Q6

Questions:

1. How many floating operations are being performed in your matrix multiply kernel? explain.
2. How many global memory reads are being performed by your kernel? explain.
3. How many global memory writes are being performed by your kernel? explain.
4. Describe what further optimizations can be implemented to your kernel to achieve a performance speedup.
5. Compare the implementation difficulty of this kernel compared to the previous MP. What difficulties did you have with this implementation?
6. Suppose you have matrices with dimensions bigger than the max thread dimensions. Sketch an algorithm that would perform matrix multiplication algorithm that would perform the multiplication in this case.
7. Suppose you have matrices that would not fit in global memory. Sketch an algorithm that would perform matrix multiplication algorithm that would perform the multiplication out of place.

Answers :

1. If N is the number of rows/columns, then 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. If N is the number of rows/columns, then 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. 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. The naive algorithm is then used over the block matrices, computing products of sub matrices entirely in fast memory.
7. Splitting the matrix into a grid will allow the product of each grid to be computed independantly.

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
```

Q7

Questions:

1. Describe all optimizations you tried regardless of whether you committed to them or abandoned them and
whether they improved or hurt performance.
2. Were there any difficulties you had with completing the optimization correctly.
3. Which optimizations gave the most benifit?
4. For the histogram kernel, how many global memory reads are being performed by your kernel? explain.
5. For the histogram kernel, how many global memory writes are being performed by your kernel? explain.
6. For the histogram kernel, how many atomic operations are being performed by your kernel? explain.
7. For the histogram kernel, what contentions would you expect if every element in the array has the same value?
8. For the histogram kernel, what contentions would you expect if every element in the input array has a random
value?

Answers:

4. Each thread reads one value from input array hence number of global reads equal to input length.

5. Each thread writes one value from input array hence number of global writes equal to number of elements in NUM_BINS that is 4096

6. Since atomic operation is done for each location in number of atomicAdd is equal to number of elements in NUM_BINS that is 4096

7. If each value in input array is same then we can use shared memory that each block gets a copy of array of size num_bins it updates the values locally. Then reduce it to global memory.

8. Another way we can use is sort the input array and take upper bound array index of a particular value. Then find frequency of each value by using adjacent difference.

Q8

Questions :

1. Name 3 applications of convolution.
2. How many floating operations are being performed in your convolution kernel? explain.
3. How many global memory reads are being performed by your kernel? explain.
4. How many global memory writes are being performed by your kernel? explain.
5. What is the minimum, maximum, and average number of real operations that a thread will perform? Real
operations are those that directly contribute to the final output value.

6. What is the measured floating-point computation rate for the CPU and GPU kernels in this application? How do they each scale with the size of the input?
7. How much time is spent as an overhead cost for using the GPU for computation? Consider all code executed within your host function with the exception of the kernel itself, as overhead. How does the overhead scale with the size of the input?
8. What do you think happens as you increase the mask size (say to 1024) while you set the block dimensions to 16x16? What do you end up spending most of your time doing? Does that put other constraints on the way you'd write your algorithm (think of the shared/constant memory size)?
9. Do you have to have a separate output memory buffer? Put it in another way, why can't you perform the
convolution in place?
10. What is the identity mask?

Answers :
1. Blur filters, Edge detection algorithms and fundamental concepts in signal processing and analysis

2. 3*N FLOPS
N Additions
2*N Multiplications

3. 2*N

4. N

5.
min = TILE_WIDTH * TILE_WIDTH.
max = image_height*image_width.
Avg = MASK_HEIGHT*MASK_WIDTH

7. 3*MASK_HEIGHT*MASK_WIDTH

8. Yes, If not used, too many conditional statements are needed and too much in place calculation will lead to slow running code which we don't want.

9. 64x64 matrix which is passed with keyword (__restrict). [to the kernel!]
10. Identity mask
[0 0 0]
[0 1 0]
[0 0 0]