ELEC 374: GPU Machine Problem #4

Tiled Matrix Multiplication

Due Date: 2022-03-27

Isabel Frolick – 20155540

This routine focuses on optimizing the kernel algorithm by implementing a 'tiled' version of the matrix multiplication routine. The tile method reduces the number of accesses to the global memory by efficiently utilizing the device's shared memory. To utilize the tiled version of matrix multiplication, a kernel function, 'kernel_mutliplication' was utilized, as seen below.

```
#define BLOCK_WIDTH 16
#define TILE_WIDTH 16

__global__ void kernel_multiplication(int* N, int* M, int* P, int nums)
{
    __shared__ int Mds[TILE_WIDTH][TILE_WIDTH];
    __shared__ int Nds[TILE_WIDTH][TILE_WIDTH];

    int threadx = threadIdx.x;
    int thready = threadIdx.y;
    int blockx = blockIdx.x;
    int blocky = blockIdx.y;
    int sum;
    int ROW = blocky * TILE_WIDTH + thready;
    int COL = blockx * TILE_WIDTH + threadx;

    for (int q = 0; q < numElements / TILE_WIDTH; q++) {
        Mds[thready][threadx] = N[row * numElements + q * TILE_WIDTH + threadx];
        Nds[thready][threadx] = M[(q * TILE_WIDTH + thready) * nums + COL];

        __syncthreads();

        for (int k = 0; k < TILE_WIDTH; k++)
            sum += Mds[thready][k] * Nds[k][threadx];

        __syncthreads();
    }
    P[ROW * nums + COL] = sum;

}
```

This function calls sufficient shared memory as variables- Mds, Nds -which allocate sufficient space for a 2D matrix the size of the tile width (which is declared as 4, 16, or 256). Shared memory is an on-chip GPU memory, that is fast but limited in its storage capacity. The original matrices, N and M begin in global memory and are loaded in 'tile size' into the shared memory attributes. The shared memory attributes act as temporary storage for the input matrices during computation, to be loaded back into the output matrix. In the kernel function, the output matrix P is set equal to the running product (called sum) in the shared memory, not the global memory, allowing the data to be reused among all threads in the block, improving access speed.

Matrix Size	Tile Width	GPU Performance	CPU Performance
16x16	4	0.000228832	0.043
	16	0.00018832	0.028
	256	0.000152768	0.035
256x256	4	0.00227162	0.232
	16	0.00222916	0.213
	256	0.00235301	0.206
4096x4096	4	0.093032	0.453
	16	0.0903145	0.881
	256	0.0901743	0.892

Within a block, shared memory is shared by all threads such that all threads can 'see' what the other threads are loading. Without tiling, each thread must access an entire row of the first input matrix and an entire column of the second input matrix, necessitating WIDTH x WIDTH memory accesses. When using tiling, each thread loads elements from each input matrix, computes the product using shared memory, and stores the temporary result. Each thread can then load the computed product into the output matrix. All threads in all blocks within the system compute the product in parallel, resulting in a much faster computation with less global memory accesses. Resultantly, the results of the tiled version of matrix multiplication are much faster than the baseline matrix multiplication in Machine Problem 3.

In the kernel implementation of this routine, assuming there are 14 streaming multiprocessors as the program is utilizing a NVIDIA Tesla C2050 device, and the warp size is 32, 448 threads can run simultaneously. The warp size is the number of threads per group per processor, such that 14x32 = 448 threads.

The resource usage of the kernel can be accessed by using the Nsight functionality within Visual Studio 2015. This module includes GPU device, CUDA Device, OpenCL device, and system information. An example of the GPU Device attributes of the devices within the GPU devices in the system is below. In the second image below, an example of the kernel resource usage can be seen, including the maximum number of threads per block, shared memory per block, and number of blocks per SM.

Attribute	Tesla C2075	Quadro 600
Driver Version	376.51	376.51
Driver Model	TCC	WDDM
CUDA Device Index	0	1
GPU Family	GF110	GF108GL
Compute Capability	2.0	2.1
Number of SMs	14	2
Frame Buffer Physical Size (MB)	5316	1024
Frame Buffer Bandwidth (GB/s)	150.336	25.6
Frame Buffer Bus Width (bits)	384	128
Frame Buffer Location	Dedicated	Dedicated
Graphics Clock (Mhz)	573	640
Memory Clock (Mhz)	1566	800
Processor Clock (Mhz)	1147	1280
RAM Type	GDDR5	DDR3
Attached Monitors	0	0