COMS4040A Assignment 2 – Report

Wesley Earl Stander - 1056114 - Coms Hons

May 8, 2020

1 Matrix Transpose

As shown in figure 1, the shared kernel is slightly faster than the global kernel with the opposite true at 512 width. Both kernels provide a significant increase to throughput compared to sequential implementation. All implementations swap the elements required for the transpose to occur.

Figure 1: Vector Reduction throughput

1.1 Running Outcome

```
CUDA Driver Version / Runtime Version 10.2 / 9.1
---Device 0---
Name: "GeForce RTX 2060"
CUDA Capability Major/Minor version number: 7.5
--- Memory information for device ---
Total global mem: 5926 MB
Total constant mem: 65536 B
The size of shared memory per block: 49152 B
The maximum number of registers per block: 65536
The number of SMs on the device: 30
The number of threads in a warp: 32
The maximal number of threads allowed in a block: 1024
Max thread dimensions (x,y,z): (1024, 1024, 64)
Max grid dimensions (x,y,z): (2147483647, 65535, 65535)

Matrix size: 2^9x2^9
Block size: 8x8
Average sequential time: 0.000837201s
Average global kernel time: 0.0147712ms
Average shared kernel time: 0.0157696ms
Throughput of sequential implementation: 2.50496GB/s
Throughput of global kernel: 141.976GB/s
Throughput of shared kernel: 132.987GB/s
Performance speedup: global over sequential 56.6779x
Performance speedup: shared over sequential 53.0895x
Performance speedup: shared over global 0.936688x

Matrix size: 2^10x2^10
Block size: 8x8
Average sequential time: 0.00355181s
Average global kernel time: 0.0585312ms
Average shared kernel time: 0.052848ms
Throughput of sequential implementation: 2.36179GB/s
Throughput of global kernel: 143.319GB/s
Throughput of shared kernel: 158.731GB/s
Performance speedup: global over sequential 60.6823x
Performance speedup: shared over sequential 67.208x
Performance speedup: shared over global 1.10754x

Matrix size: 2^11x2^11
Block size: 8x8
Average sequential time: 0.0530804s
Average global kernel time: 0.192176ms
Average shared kernel time: 0.189219ms
Throughput of sequential implementation: 0.632143GB/s
Throughput of global kernel: 174.603GB/s
Throughput of shared kernel: 177.331GB/s
Performance speedup: global over sequential 276.207x
Performance speedup: shared over sequential 280.523x
```

```
Performance speedup: shared over global 1.01563x

Matrix size: 2^12x2^12
Block size: 8x8
Average sequential time: 0.228584s
Average global kernel time: 0.76873ms
Average shared kernel time: 0.697402ms
Throughput of sequential implementation: 0.587171GB/s
Throughput of global kernel: 174.597GB/s
Throughput of shared kernel: 192.454GB/s
Performance speedup: global over sequential 297.352x
Performance speedup: shared over sequential 327.765x
Performance speedup: shared over global 1.10228x
```

2 Vector Reduction

As shown in figure 2, the shared memory kernel is consistently 3 times faster than the global memory kernel, with the same implementation of binary reduction. The kernels provide huge increases as compared to the sequential implementation. Harris provided insight on how to optimize the shared kernel implementation. Unrolling the last warp helped increase speed-up alongside adding on the first load for the shared implementation.

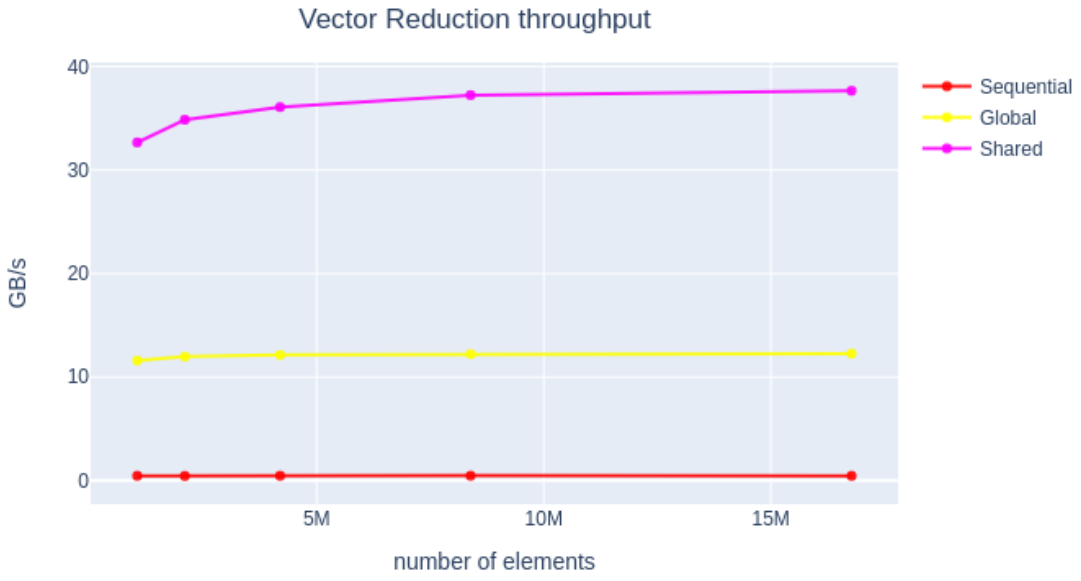

Figure 2: Vector Reduction throughput

2.1 Running Outcome

```
CUDA Driver Version / Runtime Version 10.2 / 9.1
---Device 0---
Name: "GeForce RTX 2060"
CUDA Capability Major/Minor version number: 7.5
--- Memory information for device ---
Total global mem: 5926 MB
Total constant mem: 65536 B
The size of shared memory per block: 49152 B
The maximum number of registers per block: 65536
The number of SMs on the device: 30
The number of threads in a warp: 32
The maximal number of threads allowed in a block: 1024
Max thread dimensions (x,y,z): (1024, 1024, 64)
Max grid dimensions (x,y,z): (2147483647, 65535, 65535)

Problem size: 2^20 elements
Average sequential time: 0.00229612s
Average global time: 0.0904864ms
Average shared time: 0.0320992ms
Throughput of sequential: 0.456672 GB/s
Throughput of global kernel: 11.5882 GB/s
Throughput of shared kernel: 32.6667 GB/s
Global kernel performance speed-up over sequential: 25.3753x
Shared kernel performance speed-up over sequential: 71.5321x
Shared kernel performance speed-up over global kernel: 2.81896x

Problem size: 2^21 elements
Average sequential time: 0.00475901s
Average global time: 0.175222ms
Average shared time: 0.0601568ms
Throughput of sequential: 0.440669 GB/s
Throughput of global kernel: 11.9685 GB/s
Throughput of shared kernel: 34.8614 GB/s
Global kernel performance speed-up over sequential: 27.1599x
Shared kernel performance speed-up over sequential: 79.1102x
Shared kernel performance speed-up over global kernel: 2.91276x

Problem size: 2^22 elements
Average sequential time: 0.00918494s
Average global time: 0.345869ms
Average shared time: 0.11623ms
Throughput of sequential: 0.45665 GB/s
Throughput of global kernel: 12.1269 GB/s
Throughput of shared kernel: 36.0861 GB/s
Global kernel performance speed-up over sequential: 26.5561x
Shared kernel performance speed-up over sequential: 79.0235x
Shared kernel performance speed-up over global kernel: 2.97572x

Problem size: 2^23 elements
```

```
Average sequential time: 0.0181909s
Average global time: 0.687066ms
Average shared time: 0.225242ms
Throughput of sequential: 0.461142 GB/s
Throughput of global kernel: 12.2093 GB/s
Throughput of shared kernel: 37.2427 GB/s
Global kernel performance speed-up over sequential: 26.4763x
Shared kernel performance speed-up over sequential: 80.7618x
Shared kernel performance speed-up over global kernel: 3.05035x

Problem size: 2^24 elements
Average sequential time: 0.0370151s
Average global time: 1.36836ms
Average shared time: 0.445472ms
Throughput of sequential: 0.453253 GB/s
Throughput of global kernel: 12.2608 GB/s
Throughput of shared kernel: 37.6617 GB/s
Global kernel performance speed-up over sequential: 27.0507x
Shared kernel performance speed-up over sequential: 83.0918x
Shared kernel performance speed-up over global kernel: 3.07171x
```

3 Matrix Multiplication

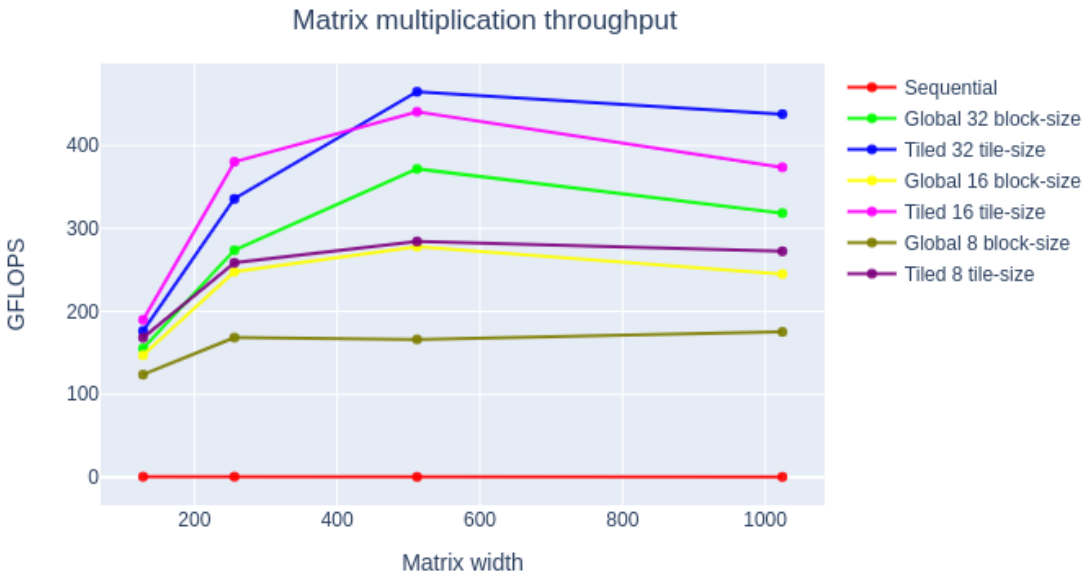

Figure 3: Vector Reduction throughput

As shown in figure 3, the tiled kernel is faster than the global kernel and the throughput increases with the size of the tile. The global kernel throughput also increases with the size of the block. It is to be noted that the 16 width tiled kernel is faster than the 32 width global kernel, showing dramatic increases of throughput with the tiled kernel. The kernels provide huge increases as compared to the sequential implementation. The tiled kernel computes partial dot products and then computes those into the global space. The global implementation computes the global dot product and the sequential as well.

3.1 Running Outcome

3.1.1 Tile-width = 32 & block-size = 32

```
CUDA Driver Version / Runtime Version 10.2 / 9.1
---Device 0---
Name: "GeForce RTX 2060"
CUDA Capability Major/Minor version number: 7.5
--- Memory information for device ---
Total global mem: 5926 MB
Total constant mem: 65536 B
The size of shared memory per block: 49152 B
The maximum number of registers per block: 65536
The number of SMs on the device: 30
The number of threads in a warp: 32
The maximal number of threads allowed in a block: 1024
Max thread dimensions (x,y,z): (1024, 1024, 64)
Max grid dimensions (x,y,z): (2147483647, 65535, 65535)

Matrix size: 2^7x2^7
Tile-size: 32x32, Block-size: 32x32
Average sequential time: 0.00718268s
Average global kernel time: 0.027008ms
Average shared kernel time: 0.023792ms
Throughput of sequential implementation: 0.583947 GFLOPS
Throughput of global kernel: 155.299 GFLOPS
Throughput of shared kernel: 176.291 GFLOPS
Performance improvement: global over seqential 265.946x
Performance speed-up: shared over seqential 301.895x
Performance speed-up: shared over global 1.13517x

Matrix size: 2^8x2^8
Tile-size: 32x32, Block-size: 32x32
Average sequential time: 0.0581249s
Average global kernel time: 0.12264ms
Average shared kernel time: 0.1ms
Throughput of sequential implementation: 0.577281 GFLOPS
Throughput of global kernel: 273.601 GFLOPS
Throughput of shared kernel: 335.544 GFLOPS
Performance improvement: global over seqential 473.948x
Performance speed-up: shared over seqential 581.249x
Performance speed-up: shared over global 1.2264x
```

```
Matrix size: 2^9x2^9
Tile-size: 32x32, Block-size: 32x32
Average sequential time: 0.746576s
Average global kernel time: 0.722104ms
Average shared kernel time: 0.577616ms
Throughput of sequential implementation: 0.359555 GFLOPS
Throughput of global kernel: 371.741 GFLOPS
Throughput of shared kernel: 464.73 GFLOPS
Performance improvement: global over seqential 1033.89x
Performance speed-up: shared over seqential 1292.51x
Performance speed-up: shared over global 1.25015x

Matrix size: 2^10x2^10
Tile-size: 32x32, Block-size: 32x32
Average sequential time: 6.72731s
Average global kernel time: 6.74206ms
Average shared kernel time: 4.90537ms
Throughput of sequential implementation: 0.319219 GFLOPS
Throughput of global kernel: 318.52 GFLOPS
Throughput of shared kernel: 437.782 GFLOPS
Performance improvement: global over seqential 997.812x
Performance speed-up: shared over seqential 1371.42x
Performance speed-up: shared over global 1.37443x
```

3.1.2 Tile-width = 16 & block-size = 16

```
CUDA Driver Version / Runtime Version 10.2 / 9.1
---Device 0---
Name: "GeForce RTX 2060"
CUDA Capability Major/Minor version number: 7.5
--- Memory information for device ---
Total global mem: 5926 MB
Total constant mem: 65536 B
The size of shared memory per block: 49152 B
The maximum number of registers per block: 65536
The number of SMs on the device: 30
The number of threads in a warp: 32
The maximal number of threads allowed in a block: 1024
Max thread dimensions (x,y,z): (1024, 1024, 64)
Max grid dimensions (x,y,z): (2147483647, 65535, 65535)

Matrix size: 2^7x2^7
Tile-size: 16x16, Block-size: 16x16
Average sequential time: 0.00732539s
Average global kernel time: 0.028528ms
Average tiled kernel time: 0.022112ms
Throughput of sequential implementation: 0.572571 GFLOPS
Throughput of global kernel: 147.024 GFLOPS
Throughput of tiled kernel: 189.685 GFLOPS
```

```
Performance improvement: global over seqential 256.779x
Performance speed-up: tiled over seqential 331.286x
Performance speed-up: tiled over global 1.29016x

Matrix size: 2^8x2^8
Tile-size: 16x16, Block-size: 16x16
Average sequential time: 0.0708874s
Average global kernel time: 0.135448ms
Average tiled kernel time: 0.08824ms
Throughput of sequential implementation: 0.473348 GFLOPS
Throughput of global kernel: 247.729 GFLOPS
Throughput of tiled kernel: 380.263 GFLOPS
Performance improvement: global over seqential 523.355x
Performance speed-up: tiled over seqential 803.348x
Performance speed-up: tiled over global 1.535x

Matrix size: 2^9x2^9
Tile-size: 16x16, Block-size: 16x16
Average sequential time: 0.738332s
Average global kernel time: 0.966432ms
Average tiled kernel time: 0.609256ms
Throughput of sequential implementation: 0.36357 GFLOPS
Throughput of global kernel: 277.759 GFLOPS
Throughput of tiled kernel: 440.595 GFLOPS
Performance improvement: global over seqential 763.977x
Performance speed-up: tiled over seqential 1211.86x
Performance speed-up: tiled over global 1.58625x

Matrix size: 2^10x2^10
Tile-size: 16x16, Block-size: 16x16
Average sequential time: 6.1771s
Average global kernel time: 8.76858ms
Average tiled kernel time: 5.74649ms
Throughput of sequential implementation: 0.347652 GFLOPS
Throughput of global kernel: 244.907 GFLOPS
Throughput of tiled kernel: 373.704 GFLOPS
Performance improvement: global over seqential 704.459x
Performance speed-up: tiled over seqential 1074.94x
Performance speed-up: tiled over global 1.5259x
```

3.1.3 Tile-width = 8 & block-size = 8

```
CUDA Driver Version / Runtime Version 10.2 / 9.1
---Device 0---
Name: "GeForce RTX 2060"
CUDA Capability Major/Minor version number: 7.5
--- Memory information for device ---
Total global mem: 5926 MB
Total constant mem: 65536 B
The size of shared memory per block: 49152 B
```

```
The maximum number of registers per block: 65536
The number of SMs on the device: 30
The number of threads in a warp: 32
The maximal number of threads allowed in a block: 1024
Max thread dimensions (x,y,z): (1024, 1024, 64)
Max grid dimensions (x,y,z): (2147483647, 65535, 65535)

Matrix size: 2^7x2^7
Tile-size: 8x8, Block-size: 8x8
Average sequential time: 0.00711099s
Average global kernel time: 0.033912ms
Average tiled kernel time: 0.024936ms
Throughput of sequential implementation: 0.589834 GFLOPS
Throughput of global kernel: 123.682 GFLOPS
Throughput of tiled kernel: 168.203 GFLOPS
Performance improvement: global over seqential 209.69x
Performance speed-up: tiled over seqential 285.17x
Performance speed-up: tiled over global 1.35996x

Matrix size: 2^8x2^8
Tile-size: 8x8, Block-size: 8x8
Average sequential time: 0.0699515s
Average global kernel time: 0.199336ms
Average tiled kernel time: 0.129808ms
Throughput of sequential implementation: 0.479682 GFLOPS
Throughput of global kernel: 168.331 GFLOPS
Throughput of tiled kernel: 258.493 GFLOPS
Performance improvement: global over seqential 350.922x
Performance speed-up: tiled over seqential 538.884x
Performance speed-up: tiled over global 1.53562x

Matrix size: 2^9x2^9
Tile-size: 8x8, Block-size: 8x8
Average sequential time: 0.73921s
Average global kernel time: 1.61529ms
Average tiled kernel time: 0.9446ms
Throughput of sequential implementation: 0.363138 GFLOPS
Throughput of global kernel: 166.184 GFLOPS
Throughput of tiled kernel: 284.179 GFLOPS
Performance improvement: global over seqential 457.634x
Performance speed-up: tiled over seqential 782.564x
Performance speed-up: tiled over global 1.71002x

Matrix size: 2^10x2^10
Tile-size: 8x8, Block-size: 8x8
Average sequential time: 6.52058s
Average global kernel time: 12.2579ms
Average tiled kernel time: 7.88581ms
Throughput of sequential implementation: 0.329339 GFLOPS
Throughput of global kernel: 175.191 GFLOPS
Throughput of tiled kernel: 272.323 GFLOPS
Performance improvement: global over seqential 531.948x
```

```
Performance speed-up: tiled over seqential 826.875x
Performance speed-up: tiled over global 1.55443x
```

References

Mark Harris. *Optimizing Parallel Reduction in CUDA*. Retrieved 8/05/2020, from https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf

A Question 1

A.1 Sequential

```
void matrix_transpose_seq(float *a, float *b, size_t width){
   for (int i =0; i < width; ++i) {
      for (int j = 0; j < width; ++j) {
         b[i*width+j] = a[j*width+i];
      }
   }
}
```

A.2 Global Kernel

```
__global__ void matrix_transpose_simple(const float *a, float *b, size_t width){
   int row = blockIdx.y*blockDim.y + threadIdx.y;
   int col = blockIdx.x*blockDim.x + threadIdx.x;
   b[row*width+col] = a[col*width+row];
}
```

A.3 Shared Kernel

```
__global__ void matrix_transpose_shared (const float *a, float *b, size_t width) {
   __shared__ float s[BLOCK_SIZE][BLOCK_SIZE];
   int bx = blockIdx.x;
   int by = blockIdx.y;
   int tx = threadIdx.x;
   int ty = threadIdx.y;

   int row = bx*BLOCK_SIZE + tx;
   int col = by*BLOCK_SIZE + ty;
   int in = row + col * width;

   int rowOut = by*BLOCK_SIZE + tx;
   int colOut = bx*BLOCK_SIZE + ty;
   int out = rowOut + colOut * width;

   s[ty][tx] = a[in];
   __syncthreads();
   b[out] = s[tx][ty];
}
```

B Question 2

B.1 Sequential

```
void vector_reduction_seq(const float *a,
                          float *c,
                          const size_t n){
    c[0] = 0;
   for(int i = 0; i < n; i++){
      c[0] += a[i];
   }
}
```

B.2 Global Kernel

```
__global__ void vector_reduction(float *a,
                                 float *c,
                                 const size_t n){
  // compute the global element index this thread should process
  unsigned int tid = threadIdx.x;
  unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
  for(unsigned int s=blockDim.x/2; s > 0; s >>= 1) { //binary reduction
      if (tid < s) {
          a[i] += a[i + s];
      }
      __syncthreads();
  }

  if (tid == 0) atomicAdd(c, a[i]);
}
```

B.3 Shared Kernel

```
__device__ void warp_reduce(volatile float* sD, int tid) { //unroll last warp (32 threads)
    sD[tid] += sD[tid + 32];
    sD[tid] += sD[tid + 16];
    sD[tid] += sD[tid + 8];
    sD[tid] += sD[tid + 4];
    sD[tid] += sD[tid + 2];
    sD[tid] += sD[tid + 1];
}

__global__ void vector_reduction_shared(const float* a, float* c, const size_t n) {
    extern __shared__ float sD[];
    unsigned int tid = threadIdx.x;
    unsigned int blockSize = blockDim.x;
    unsigned int i = blockIdx.x*(blockSize*2) + tid;
    sD[tid] = a[i] + a[i+blockSize]; //add on first load
    __syncthreads();
```

```
    for(unsigned int s=blockSize/2; s > 32; s >>= 1) { //binary reduction
        if (tid < s) {
            sD[tid] += sD[tid + s];
        }
        __syncthreads();
    }

    if (tid < 32) warp_reduce(sD, tid); //unroll last warp for block

    if (tid == 0) atomicAdd(c,sD[0]); //add each block value to final value
}
```

C Question 3

C.1 Sequential

```
void matrix_multiply_seq(float *a, float *b, float *ab, size_t width){
   int i, j, k;
   for(i=0; i<width; i++) {
      for(j=0; j<width; j++){
         ab[i*width+j]=0.0;
         for(k=0; k<width; k++){
            ab[i*width+j] += a[i*width+k] * b[k*width+j]; //dot product
         }
      }
   }
}
```

C.2 Global Kernel

```
__global__ void matrix_multiply_simple(const float *a,const float *b, float *ab, size_t width){

  int row = blockIdx.y*blockDim.y + threadIdx.y;
  int col = blockIdx.x*blockDim.x + threadIdx.x;
  float result = 0;

  for(int k = 0; k < width; ++k){
    result += a[row*width+k] * b[k*width+col]; //dot product
  }

  ab[row*width+col] = result;
}
```

C.3 Shared Kernel

```
__global__ void matrix_multiply_tiled (const float *a, const float *b, float *ab, size_t width) {
```

```
   __shared__ float ds_a[TILE_WIDTH][TILE_WIDTH];
   __shared__ float ds_b[TILE_WIDTH][TILE_WIDTH];
   int bx = blockIdx.x;
   int by = blockIdx.y;
   int tx = threadIdx.x;
   int ty = threadIdx.y;

   int row = by * TILE_WIDTH + ty;
   int col = bx * TILE_WIDTH + tx;
   float pvalue = 0;

   for (int ph = 0; ph < width/TILE_WIDTH; ++ph) {
      ds_a[ty][tx] = a[row*width + ph*TILE_WIDTH + tx];
      ds_b[ty][tx] = b[col+(ph*TILE_WIDTH+ty)*width];
      __syncthreads();
      for (int i = 0; i < TILE_WIDTH; ++i)
         pvalue += ds_a[ty][i] * ds_b[i][tx]; //dot product
      __syncthreads();
   }

   ab[row*width+col] = pvalue;
}
```