

Report Lab 3

Andres Calderon - SID:861243796

November 11, 2015

1 Code

The following code was used to complete the report:

1.1 Reduction

1.1.1 kernel.cu

```
1  /*****
2  *cr
3  *cr          (C) Copyright 2010 The Board of Trustees of the
4  *cr          University of Illinois
5  *cr          All Rights Reserved
6  *cr
7  *****/
8
9  #define BLOCK_SIZE 512
10
11 __global__ void reduction(float *out, float *in, unsigned size)
12 {
13     /*****
14     Load a segment of the input vector into shared memory
15     Traverse the reduction tree
16     Write the computed sum to the output vector at the correct index
17     *****/
18
19     // Declare an array for share memory...
20     __shared__ float partialSum[2 * BLOCK_SIZE];
21
22     // Initialize some variables to access data...
23     unsigned int t = threadIdx.x;
24     unsigned int start = 2 * blockIdx.x * blockDim.x;
25
26     // Validation to avoid load data outside of the input array...
27     if(start + t < size)
28         partialSum[t] = in[start + t];
29     else
30         partialSum[t] = 0.0f;
31
32     // Same validation for the other position...
33     if(start + blockDim.x + t < size)
34         partialSum[blockDim.x + t] = in[start + blockDim.x + t];
35     else
36         partialSum[blockDim.x + t] = 0.0f;
37
38     // Iterate through share memory to compute the sum...
39     for (int stride = blockDim.x; stride > 0; stride /= 2){
40         __syncthreads(); // Synchronize the share memory load and each iteration...
41         if (t < stride)
42             partialSum[t] += partialSum[t + stride];
43     }
44     // Do not forget to synchronize last iteration...
45     __syncthreads();
46 }
```

```

47  // Copy back the result...
48  out[blockIdx.x] = partialSum[0];
49  }

```

There are not significant changes in the other files.

1.2 Prefix-scan

1.2.1 kernel.cu

```

1  /*****
2  *cr
3  *cr          (C) Copyright 2010 The Board of Trustees of the
4  *cr          University of Illinois
5  *cr          All Rights Reserved
6  *cr
7  *****/
8
9  #define BLOCK_SIZE 512
10
11 // Define your kernels in this file you may use more than one kernel if you
12 // need to
13
14 __global__ void scan(float *out, float *in, unsigned size){
15     __shared__ float section[2 * BLOCK_SIZE];
16     int t = blockDim.x * blockIdx.x + threadIdx.x;
17
18     if(t < size)
19         if(t == 0)
20             section[0] = 0.0f;
21         else
22             section[threadIdx.x] = in[t - 1];
23     __syncthreads();
24
25     for(int stride = 1; stride <= BLOCK_SIZE; stride = stride * 2){
26         int index = (threadIdx.x + 1) * stride * 2 - 1;
27         if(index < 2 * BLOCK_SIZE)
28             section[index] += section[index - stride];
29         __syncthreads();
30     }
31
32     for(int stride = BLOCK_SIZE / 2; stride > 0; stride /= 2){
33         int index = (threadIdx.x + 1) * stride * 2 - 1;
34         if(index + stride < 2 * BLOCK_SIZE)
35             section[index + stride] += section[index];
36         __syncthreads();
37     }
38     //__syncthreads();
39     if(t < size)
40         out[t] = section[threadIdx.x];
41 }
42
43 __global__ void post(float *out, float *n, unsigned size){
44     int t = blockDim.x * blockIdx.x + threadIdx.x;
45
46     out[t] += n[t / BLOCK_SIZE];
47 }
48
49 /*****
50 Setup and invoke your kernel(s) in this function. You may also allocate more
51 GPU memory if you need to
52 *****/
53 void preScan(float *out, float *in, unsigned size){
54     dim3 dim_block(BLOCK_SIZE, 1, 1);
55     dim3 dim_grid(size/BLOCK_SIZE + 1, 1, 1);
56     scan<<<dim_grid, dim_block>>>(out, in, size);
57 }
58
59 void postScan(float *out, float *n, unsigned size){
60     dim3 dim_block(BLOCK_SIZE, 1, 1);

```

```

61     dim3 dim_grid(size/BLOCK_SIZE + 1, 1, 1);
62     post<<<dim_grid, dim_block>>>(out, n, size);
63 }

```

1.2.2 main.cu

```

1  /*****
2  *cr
3  *cr      (C) Copyright 2010 The Board of Trustees of the
4  *cr      University of Illinois
5  *cr      All Rights Reserved
6  *cr
7  *****/
8
9  #include <stdio.h>
10 #include "support.h"
11 #include "kernel.cu"
12
13 int main(int argc, char* argv[])
14 {
15     Timer timer;
16     // Initialize host variables
17     printf("\nSetting up the problem..."); fflush(stdout);
18     startTime(&timer);
19
20     float *in_h, *out_h;
21     float *in_d, *out_d;
22     unsigned num_elements;
23     cudaError_t cuda_ret;
24
25     /* Allocate and initialize input vector */
26     if(argc == 1) {
27         num_elements = 1000000;
28     } else if(argc == 2) {
29         num_elements = atoi(argv[1]);
30     } else {
31         printf("\n Invalid input parameters!"
32             "\n Usage: ./prefix-scan    # Input of size 1,000,000 is used"
33             "\n Usage: ./prefix-scan <m> # Input of size m is used"
34             "\n");
35         exit(0);
36     }
37     initVector(&in_h, num_elements);
38
39     /* Allocate and initialize output vector */
40     out_h = (float*)calloc(num_elements, sizeof(float));
41     if(out_h == NULL) FATAL("Unable to allocate host");
42
43     stopTime(&timer); printf("%f s\n", elapsedTime(timer));
44     printf("Input size = %u\n", num_elements);
45
46     // Allocate device variables
47     printf("Allocating device variables..."); fflush(stdout);
48     startTime(&timer);
49     cuda_ret = cudaMalloc((void**)&in_d, num_elements*sizeof(float));
50     if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");
51     cuda_ret = cudaMalloc((void**)&out_d, num_elements*sizeof(float));
52     if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");
53     cudaDeviceSynchronize();
54     stopTime(&timer); printf("%f s\n", elapsedTime(timer));
55
56     // Copy host variables to device
57     printf("Copying data from host to device..."); fflush(stdout);
58     startTime(&timer);
59     cuda_ret = cudaMemcpy(in_d, in_h, num_elements*sizeof(float), cudaMemcpyHostToDevice);
60     if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to the device");
61     cuda_ret = cudaMemset(out_d, 0, num_elements*sizeof(float));
62     if(cuda_ret != cudaSuccess) FATAL("Unable to set device memory");
63     cudaDeviceSynchronize();
64     stopTime(&timer); printf("%f s\n", elapsedTime(timer));

```

```

65
66 // Launch kernel
67 printf("Launching kernel..."); fflush(stdout);
68 startTime(&timer);
69 // Set up and invoke your kernel inside the preScan function,
70 // which is in kernel.cu
71 preScan(out_d, in_d, num_elements);
72 cuda_ret = cudaDeviceSynchronize();
73 if(cuda_ret != cudaSuccess) FATAL("Unable to launch/execute kernel");
74 stopTime(&timer); printf("%f s\n", elapsedTime(timer));
75
76 // Copy device variables from host
77 printf("Copying data from device to host..."); fflush(stdout);
78 startTime(&timer);
79 cuda_ret = cudaMemcpy(out_h, out_d, num_elements*sizeof(float), cudaMemcpyDeviceToHost);
80 if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to host");
81 cudaDeviceSynchronize();
82 stopTime(&timer); printf("%f s\n", elapsedTime(timer));
83
84 // My code...
85 float *partial_h, *partial_d;
86 partial_h = (float *) malloc((num_elements/BLOCK_SIZE + 1) * sizeof(float));
87 partial_h[0] = 0;
88 int n = 1;
89 for(int i = BLOCK_SIZE - 1; i < num_elements; i += BLOCK_SIZE){
90     partial_h[n] = partial_h[n - 1] + out_h[i];
91     n++;
92 }
93 if((num_elements/BLOCK_SIZE + 1) <= 10){
94     for(int i = 0; i < n; i++){
95         printf("\nPARTIAL[%d] = %0.3f", i, partial_h[i]);
96     }
97     printf("\n");
98 }
99 //
100 cuda_ret = cudaMalloc((void**)&partial_d, (num_elements/BLOCK_SIZE + 1) * sizeof(float));
101 if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");
102 cuda_ret = cudaMemcpy(partial_d, partial_h, (num_elements/BLOCK_SIZE + 1) * sizeof(float),
103     ↪ cudaMemcpyHostToDevice);
104 if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to the device");
105 //
106 postScan(out_d, partial_d, num_elements);
107 cuda_ret = cudaDeviceSynchronize();
108 if(cuda_ret != cudaSuccess) FATAL("Unable to launch/execute kernel");
109 //
110 cuda_ret = cudaMemcpy(out_h, out_d, num_elements*sizeof(float), cudaMemcpyDeviceToHost);
111 if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to host");
112
113 // Verify correctness
114 printf("Verifying results..."); fflush(stdout);
115 verify(in_h, out_h, num_elements);
116
117 // Printing results (just for debugging purposes)...
118 if(num_elements <= 100){
119     printf("\nPrinting IN (%d elements)...\n", num_elements);
120     for(int i = 0; i < num_elements; i++){
121         printf("%0.3f ", in_h[i]);
122     }
123     printf("\n");
124
125     printf("\nPrinting OUT (%d elements)...\n", num_elements);
126     for(int i = 0; i < num_elements; i++){
127         printf("%0.3f ", out_h[i]);
128     }
129     printf("\n");
130 }
131
132 // Free memory
133 cudaFree(in_d); cudaFree(out_d); cudaFree(partial_d);

```

```

134     free(in_h); free(out_h); free(partial_h);
135
136     return 0;
137 }

```

2 Answer to Questions

1. In your kernel implementation, how many threads can be simultaneously executing? Assume a GeForce GTX 280 GPU which has 30 streaming multiprocessors.

References

- [1] Andres Calderon. *GitHub Personal Repository*, 2015. <https://github.com/aocalderon/PhD/tree/master/Y1Q1/GPU/lab2>.
- [2] David Kirk and Wen-Mei Hwu. *Programming Massively Parallel Processors: A Hands-On Approach*. Morgan Kaufmann, 2012.
- [3] Wen-Mei Hwu. *A Tiled Kernel for Arbitrary Matrix Dimensions - Heterogeneous Parallel Programming*. Coursera Course, 2015. <https://www.dropbox.com/s/4y06b1m6dozp2kt/2%20-%208%20-%202.8-%20A%20Tiled%20Kernel%20for%20Arbitrary%20Matrix%20Dimensions.mp4?dl=0>.
- [4] David Luebke, John Owens, Mike Roberts and Cheng-Han Lee. *Using NVVP Part1 and Part 2 - Intro to Parallel Programming*. Udacity Course, 2015. <https://www.youtube.com/watch?v=hyKA5fb5ZJI>.
- [5] Nvidia Corporation. *GeForce® GTX 200 GPU Architectural Overview*. Technical Brief #TB-04044-001_v01, 2008. http://www.nvidia.com/docs/IO/55506/GeForce_GTX_200_GPU_Technical_Brief.pdf.