

Report Lab 3

Andres Calderon - SID:861243796

November 12, 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 need to
12 __global__ void scan(float *out, float *in, unsigned size){
13     // Declare share memory space...
14     __shared__ float section[2 * BLOCK_SIZE];
15     int t = blockDim.x * blockIdx.x + threadIdx.x;
16
17     // Load data in memory share taking into account that exclusive scan is required...
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     // First phase: reduction step...
26     for(int stride = 1; stride <= BLOCK_SIZE; stride = stride * 2){
27         int index = (threadIdx.x + 1) * stride * 2 - 1;
28         if(index < 2 * BLOCK_SIZE)
29             section[index] += section[index - stride];
30         __syncthreads();
31     }
32
33     // Second phase: post-scan step...
34     for(int stride = BLOCK_SIZE / 2; stride > 0; stride /= 2){
35         int index = (threadIdx.x + 1) * stride * 2 - 1;
36         if(index + stride < 2 * BLOCK_SIZE)
37             section[index + stride] += section[index];
38         __syncthreads();
39     }
40
41     // Copy back to global memory...
42     if(t < size)
43         out[t] = section[threadIdx.x];
44 }
45
46 // This function takes the output of the scan function and add the accumulated sum (stored in n) to
47 // ↪ positions in each block...
48 __global__ void post(float *out, float *n, unsigned size){
49     int t = blockDim.x * blockIdx.x + threadIdx.x;
50
51     out[t] += n[t / BLOCK_SIZE];
52 }
53
54 /*****
55 Setup and invoke your kernel(s) in this function. You may also allocate more
56 GPU memory if you need to
57 *****/
58 // Functions to define block a grid dimensions and lunch the kernels...
59 void preScan(float *out, float *in, unsigned size){
60     dim3 dim_block(BLOCK_SIZE, 1, 1);

```

```

60     dim3 dim_grid(size/BLOCK_SIZE + 1, 1, 1);
61     scan<<<dim_grid, dim_block>>>(out, in, size);
62 }
63
64 void postScan(float *out, float *n, unsigned size){
65     dim3 dim_block(BLOCK_SIZE, 1, 1);
66     dim3 dim_grid(size/BLOCK_SIZE + 1, 1, 1);
67     post<<<dim_grid, dim_block>>>(out, n, size);
68 }

```

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 = cudaMemcpy(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, which is in kernel.cu
70  preScan(out_d, in_d, num_elements);
71  cuda_ret = cudaDeviceSynchronize();
72  if(cuda_ret != cudaSuccess) FATAL("Unable to launch/execute kernel");
73  stopTime(&timer); printf("%f s\n", elapsedTime(timer));
74
75  // Copy device variables from host
76  printf("Copying data from device to host..."); fflush(stdout);
77  startTime(&timer);
78  cuda_ret = cudaMemcpy(out_h, out_d, num_elements*sizeof(float), cudaMemcpyDeviceToHost);
79  if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to host");
80  cudaDeviceSynchronize();
81  stopTime(&timer); printf("%f s\n", elapsedTime(timer));
82
83  /* Now we need to traverse the out_h array to extract the last accumulated sum for each block and then add
   ↪ them to its corresponding positions... */
84
85  // Let's declare a auxiliar array to store the accumulated sums of each block...
86  float *partial_h, *partial_d;
87  // Allocate memory for the new array...
88  partial_h = (float *) malloc((num_elements/BLOCK_SIZE + 1) * sizeof(float));
89  // We do not add nothing to the first block
90  partial_h[0] = 0;
91  int n = 1;
92  // Iterate through the array out_h extracting partial sums...
93  for(int i = BLOCK_SIZE - 1; i < num_elements; i += BLOCK_SIZE){
94      partial_h[n] = partial_h[n - 1] + out_h[i];
95      n++;
96  }
97  // Print the partial_h array just for debugging purposes...
98  if((num_elements/BLOCK_SIZE + 1) <= 10){
99      for(int i = 0; i < n; i++){
100          printf("\nPARTIAL[%d] = %0.3f", i, partial_h[i]);
101      }
102      printf("\n");
103  }
104  // Allocate memory and copy the array in the device...
105  cuda_ret = cudaMalloc((void**)&partial_d, (num_elements/BLOCK_SIZE + 1) * sizeof(float));
106  if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");
107  cuda_ret = cudaMemcpy(partial_d, partial_h, (num_elements/BLOCK_SIZE + 1) * sizeof(float),
   ↪ cudaMemcpyHostToDevice);
108  if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to the device");
109
110  // Invoke a new kernel (it is in kernel.cu) where each thread add the accumulated sum in partial_d to its
   ↪ corresponding position in out_d...
111  postScan(out_d, partial_d, num_elements);
112  cuda_ret = cudaDeviceSynchronize();
113  if(cuda_ret != cudaSuccess) FATAL("Unable to launch/execute kernel");
114
115  // Copy back the new results...
116  cuda_ret = cudaMemcpy(out_h, out_d, num_elements*sizeof(float), cudaMemcpyDeviceToHost);
117  if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to host");
118
119  // Verify correctness
120  printf("Verifying results..."); fflush(stdout);
121  verify(in_h, out_h, num_elements);
122
123  // Printing results (just for debugging purposes)...
124  if(num_elements <= 100){
125      printf("\nPrinting IN (%d elements)...\n", num_elements);

```

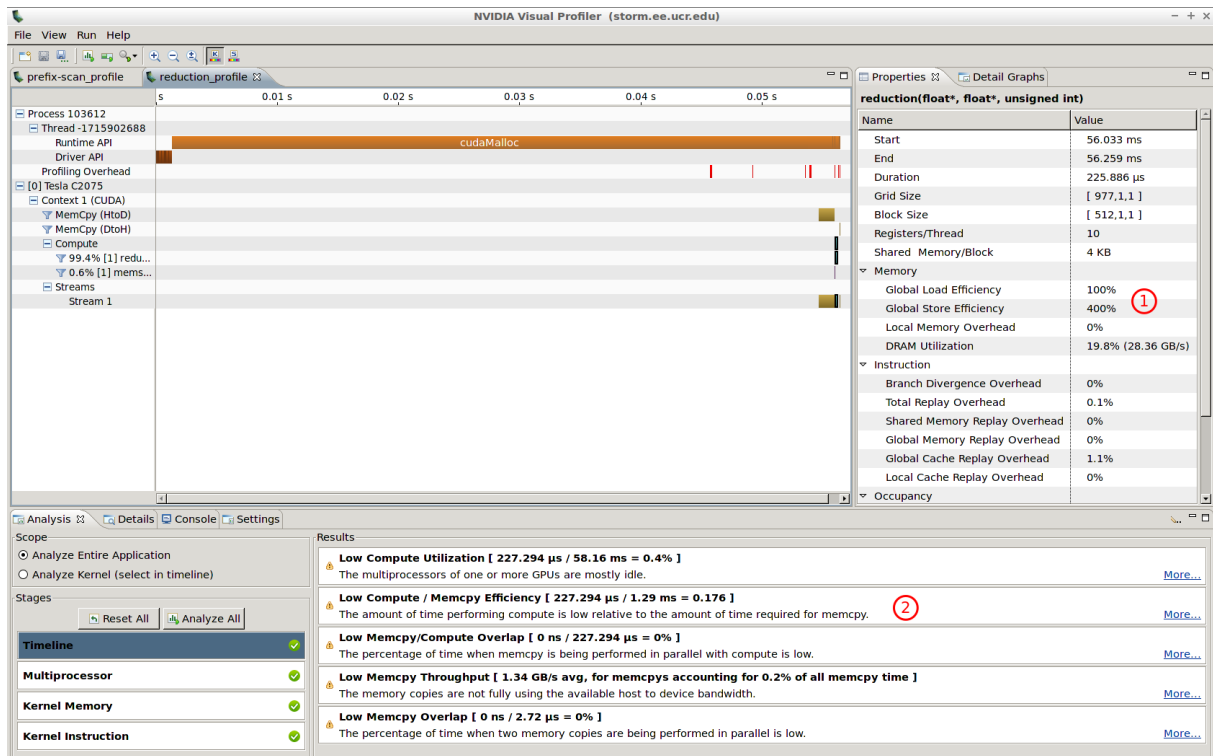


Figure 1: NVVP performance analysis for *reduction*.

```

126     for(int i = 0; i < num_elements; i++){
127         printf("%0.3f ", in_h[i]);
128     }
129     printf("\n");
130
131     printf("\nPrinting OUT (%d elements)...\n", num_elements);
132     for(int i = 0; i < num_elements; i++){
133         printf("%0.3f ", out_h[i]);
134     }
135     printf("\n");
136 }
137
138 // Free memory
139 cudaFree(in_d); cudaFree(out_d); cudaFree(partial_d);
140 free(in_h); free(out_h); free(partial_h);
141
142 return 0;
143 }

```

2 Answers to Questions

1. Use visual profiler to report relevant statistics about the execution of your kernels. Did you find any surprising results?
2. For each of reduction and prefix scan, suggest one approach to speed up your implementation.

For reduction, chunk input data to perform more `memcpy` operations and take advantage of the time of each of them to perform computation using streams. So, when the program is copying some data it already can computing some of them asynchronously. In this way, it is possible to overlap the copy and compute stages.

For prefix-scan: Harris' algorithm...

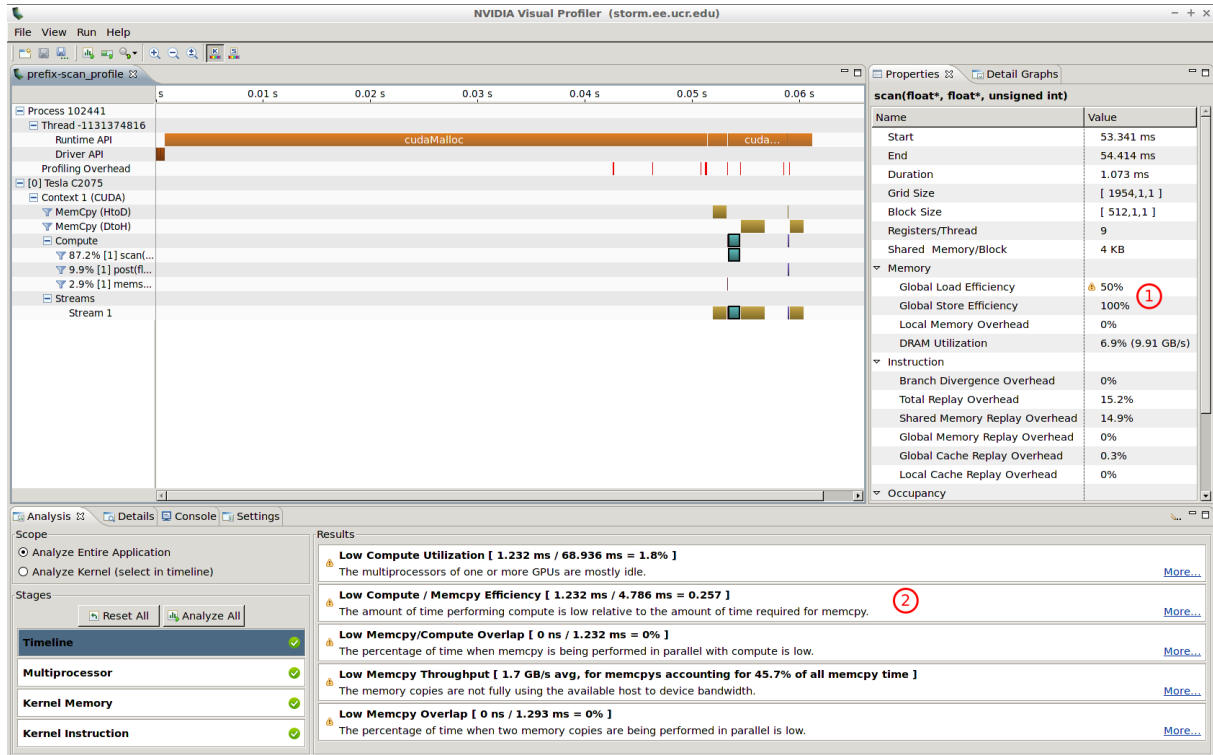


Figure 2: NVVP performance analysis for *prefix-scan*.

References

- [1] Andres Calderon. *GitHub Personal Repository*, 2015. <https://github.com/aocalderon/PhD/tree/master/Y1Q1/GPU/1ab2>.
- [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-%20202.8-%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.