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

```
/******************************************************************************
 *cr
 *cr            (C) Copyright 2010 The Board of Trustees of the
 *cr                        University of Illinois
 *cr                         All Rights Reserved
 *cr
 ******************************************************************************/

#define BLOCK_SIZE 512

__global__ void reduction(float *out, float *in, unsigned size)
{
    /********************************************************************
    Load a segment of the input vector into shared memory
    Traverse the reduction tree
    Write the computed sum to the output vector at the correct index
    ********************************************************************/

  // Declare an array for share memory...
  __shared__ float partialSum[2 * BLOCK_SIZE];

  // Initialize some variables to access data...
  unsigned int t = threadIdx.x;
  unsigned int start = 2 * blockIdx.x * blockDim.x;

  // Validation to avoid load data outside of the input array...
  if(start + t < size)
    partialSum[t] = in[start + t];
  else
    partialSum[t] = 0.0f;

  // Same validation for the other position...
  if(start + blockDim.x + t < size)
    partialSum[blockDim.x + t] = in[start + blockDim.x + t];
  else
    partialSum[blockDim.x + t] = 0.0f;

  // Iterate through share memory to compute the sum...
  for (int stride = blockDim.x; stride > 0; stride /= 2){
    __syncthreads();  // Synchronize the share memory load and each iteration...
    if (t < stride)
      partialSum[t] += partialSum[t + stride];
  }
  // Do not forget to synchronize last iteration...
  __syncthreads();

```

```
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);
```

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

1.2.2 main.cu

```
/******************************************************************************
 *cr
 *cr      (C) Copyright 2010 The Board of Trustees of the
 *cr            University of Illinois
 *cr             All Rights Reserved
 *cr
 ******************************************************************************/

#include <stdio.h>
#include "support.h"
#include "kernel.cu"

int main(int argc, char* argv[])
{
  Timer timer;
  // Initialize host variables
  printf("\nSetting up the problem..."); fflush(stdout);
  startTime(&timer);

  float *in_h, *out_h;
  float *in_d, *out_d;
  unsigned num_elements;
  cudaError_t cuda_ret;

  /* Allocate and initialize input vector */
  if(argc == 1) {
    num_elements = 1000000;
  } else if(argc == 2) {
    num_elements = atoi(argv[1]);
  } else {
    printf("\n  Invalid input parameters!"
      "\n  Usage: ./prefix-scan     # Input of size 1,000,000 is used"
      "\n  Usage: ./prefix-scan <m> # Input of size m is used"
      "\n");
    exit(0);
  }
  initVector(&in_h, num_elements);

  /* Allocate and initialize output vector */
  out_h = (float*)calloc(num_elements, sizeof(float));
  if(out_h == NULL) FATAL("Unable to allocate host");

  stopTime(&timer); printf("%f s\n", elapsedTime(timer));
  printf("Input size = %u\n", num_elements);

  // Allocate device variables
  printf("Allocating device variables..."); fflush(stdout);
  startTime(&timer);
  cuda_ret = cudaMalloc((void**)&in_d, num_elements*sizeof(float));
  if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");
  cuda_ret = cudaMalloc((void**)&out_d, num_elements*sizeof(float));
  if(cuda_ret != cudaSuccess) FATAL("Unable to allocate device memory");
  cudaDeviceSynchronize();
  stopTime(&timer); printf("%f s\n", elapsedTime(timer));

  // Copy host variables to device
  printf("Copying data from host to device..."); fflush(stdout);
  startTime(&timer);
  cuda_ret = cudaMemcpy(in_d, in_h, num_elements*sizeof(float), cudaMemcpyHostToDevice);
  if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to the device");
  cuda_ret = cudaMemset(out_d, 0, num_elements*sizeof(float));
  if(cuda_ret != cudaSuccess) FATAL("Unable to set device memory");
  cudaDeviceSynchronize();
  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),
        ↪  cudaMemcpyHostToDevice);
103     if(cuda_ret != cudaSuccess) FATAL("Unable to copy memory to the device");
104     //
105     postScan(out_d, partial_d, num_elements);
106     cuda_ret = cudaDeviceSynchronize();
107     if(cuda_ret != cudaSuccess) FATAL("Unable to launch/execute kernel");
108
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);
```

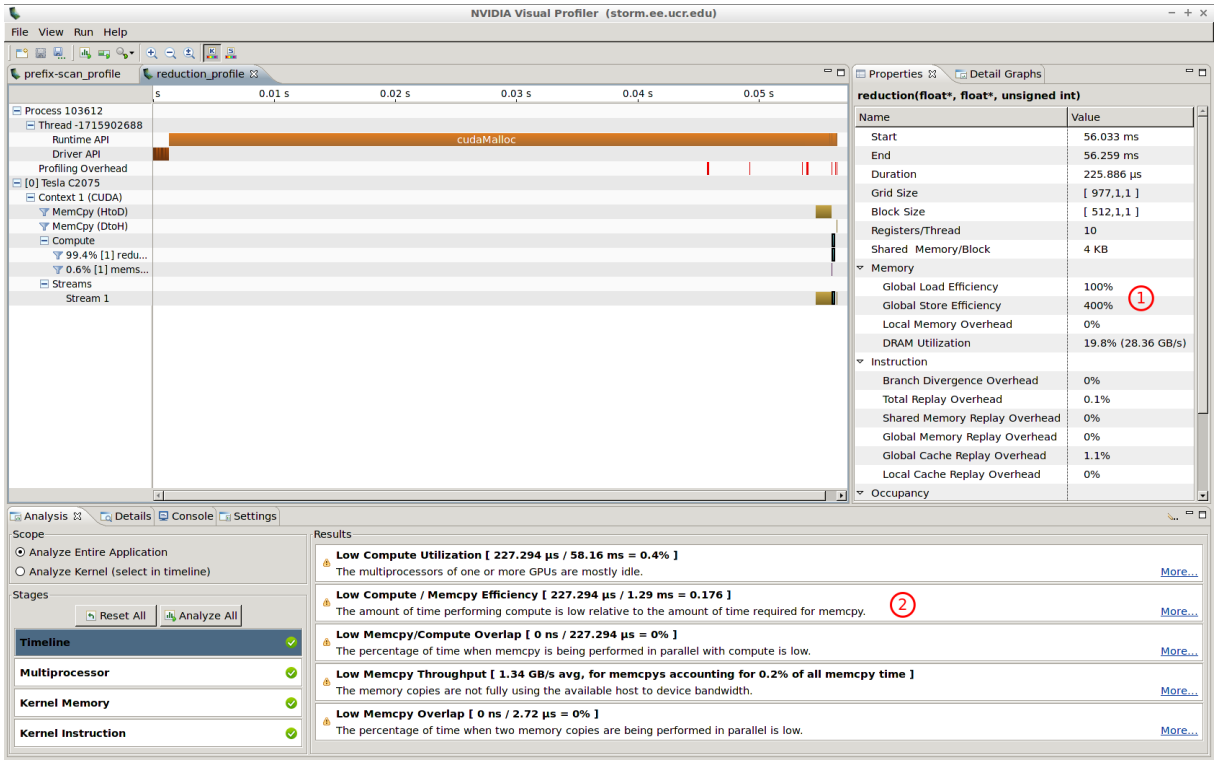

Figure 1: NVVP performance analysis for *reduction*.

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

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...

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`.

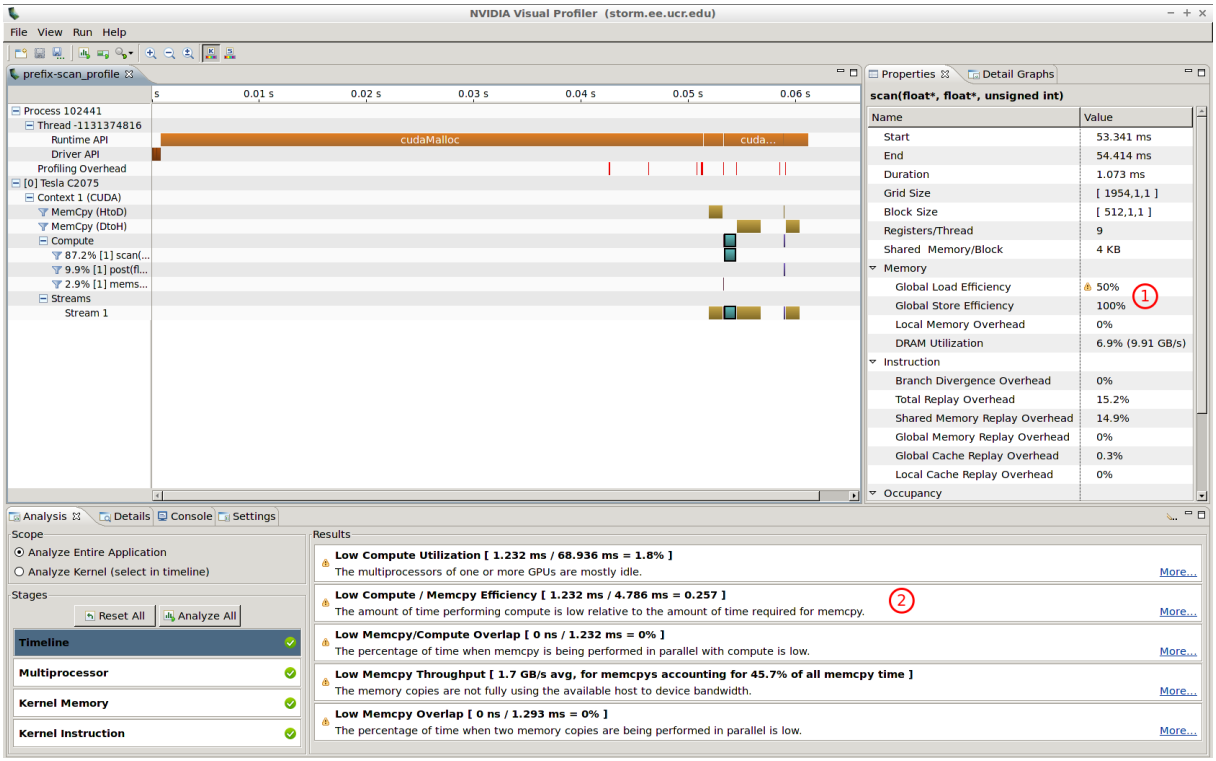

Figure 2: NVVP performance analysis for *prefix-scan*.

[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`.