ADD - INVERT
Labo Geavanceerde Computertechniek

(JLIZNM)

Jona Cappelle & Jonas Bolle

March 9, 2020

Session Date:	January 1, 2017
Partners:	Jona Cappelle
	Jonas Bolle
Class:	MELICTEES
Instructor:	Stijn Crul

Contents

List of Listings

1 Inleiding

2 Code

```
1  ////////////////////////////////////////////////////////////////////////////
2  // ADD / INVERT -- Jona Cappelle -- Jonas Bolle
3  ////////////////////////////////////////////////////////////////////////////
4
5  // includes, system
6  #include <stdlib.h>
7  #include <stdio.h>
8  #include <string.h>
9  #include <math.h>
10
11 // includes CUDA
12 #include <cuda_runtime.h>
13
14 // includes, project
15 #include <helper_cuda.h>
16 #include <helper_functions.h> // helper functions for SDK examples
17
18 // eigen includes
19 #include "iostream"
20 #include "cstdlib"
21 #include "time.h"        // timing on cpu
22
23 extern "C"
24 #define ARRAYSIZE 100000000 // Is also the number of threads that will be used
25
26
27 // HELPER FUNCTIONS
28 void init_array(int *a)
29 {
30         for (int i = 0; i < ARRAYSIZE; i++)
31         {
32                 a[i] = i;
33         }
34 }
35
36
37 ////////////////////////////////////////////////////////////////////////////
38 // KERNEL ADD
39 ////////////////////////////////////////////////////////////////////////////
40 int BLOCKSIZE;
41
42 // GPU
43 __global__ void add(int *a, int *b, int *out)
44 {
45         int idx = blockIdx.x * blockDim.x + threadIdx.x;
```

```
46          if (idx < ARRAYSIZE)
47          {
48                  out[idx] = a[idx] + b[idx];
49          }
50  }
51
52  // CPU
53  void cpu_add(int *a, int *b, int *out)
54  {
55          for (int i = 0; i < ARRAYSIZE; i++)
56          {
57                  out[i] = a[i] + b[i];
58          }
59  }
60
61  ////////////////////////////////////////////////////////////////////////////////
62  // KERNEL INVERT
63  ////////////////////////////////////////////////////////////////////////////////
64
65  // GPU
66  __global__ void invert(int *a, int *out)
67  {
68          int idx = blockIdx.x * blockDim.x + threadIdx.x;
69          if (idx < ARRAYSIZE)
70          {
71                  out[idx] = a[ARRAYSIZE - 1 - idx];
72          }
73  }
74
75  // CPU
76  void cpu_invert(int *a, int *out)
77  {
78          for (int i = 0; i < ARRAYSIZE; i++)
79          {
80                  out[i] = a[ARRAYSIZE - 1 - i];
81          }
82  }
83
84  ////////////////////////////////////////////////////////////////////////////////
85  // Program main
86  ////////////////////////////////////////////////////////////////////////////////
87  int main()
88  {
89
90          //declare variables
91          int *a_host, *b_host, *out_host;
92          int *a_dev, *b_dev, *out_dev;
93
94
```

```
95          //allocate arrays on host
96          a_host = (int *)malloc(ARRAYSIZE * sizeof(int));
97
98          b_host = (int *)malloc(ARRAYSIZE * sizeof(int));
99          out_host = (int *)malloc(ARRAYSIZE * sizeof(int));
100
101         init_array(a_host);
102         init_array(b_host);
103
104         //allocate arrays on device
105         cudaMalloc((void **)&a_dev, ARRAYSIZE * sizeof(int));
106         cudaMalloc((void **)&b_dev, ARRAYSIZE * sizeof(int));
107         cudaMalloc((void **)&out_dev, ARRAYSIZE * sizeof(int));
108
109         cudaEvent_t start, stop;
110         cudaEventCreate(&start);
111         cudaEventCreate(&stop);
112
113         // Timer on CPU
114 //        clock_t start, end;
115 //        double cpu_time_used;
116
117 //         Initialize data file where the timing results will be stored
118          FILE *f = fopen("data.csv", "w");
119
120         for (int BLOCKSIZE = 1; BLOCKSIZE < 300; BLOCKSIZE++)
121         {
122                 float millis = 0;
123                 // Calculate amount of blocks needed
124                 int nBlocks = ARRAYSIZE / BLOCKSIZE + (ARRAYSIZE % BLOCKSIZE ==
                    ↪  0 ? 0 : 1);
125                 printf("Nblocks: %i", nBlocks);
126
127                 // Start timer
128                 StopWatchInterface *timer = 0;
129                 sdkCreateTimer(&timer);
130                 sdkStartTimer(&timer);
131 //                cudaEventRecord(start);
132                 //Step 1: Copy data to GPU memory
133                 cudaMemcpy(a_dev, a_host, ARRAYSIZE * sizeof(int),
                    ↪  cudaMemcpyHostToDevice);
134                 cudaMemcpy(b_dev, b_host, ARRAYSIZE * sizeof(int),
                    ↪  cudaMemcpyHostToDevice);
135                 cudaMemcpy(out_dev, out_host, ARRAYSIZE * sizeof(int),
                    ↪  cudaMemcpyHostToDevice);
136
137                 ////////////////////////////////////////////////////////////////////////////
138                 // GPU -- comment / uncomment to run 'ADD' / 'INVERT'
139                 ////////////////////////////////////////////////////////////////////////////
```

```
140
141 //                add<<<nBlocks, BLOCKSIZE>>>(a_dev, b_dev, out_dev);
142                 invert <<< nBlocks, BLOCKSIZE >>> ( a_dev, out_dev );
143 //                cudaEventRecord(stop);
144
145                 ////////////////////////////////////////////////////////////////////////////
146                 // CPU -- comment / uncomment to run 'ADD' / 'INVERT'
147                 ////////////////////////////////////////////////////////////////////////////
148
149 //                start = clock();
150 //                cpu_add( a_host, b_host, out_host);
151 //                cpu_invert ( a_host, out_host );
152 //                end = clock();
153 //                cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
154 //                printf("%f", cpu_time_used);
155
156                 //Step 4: Retrieve result
157                 cudaMemcpy(a_host, a_dev, ARRAYSIZE * sizeof(int),
                    ↪  cudaMemcpyDeviceToHost);
158                 cudaMemcpy(b_host, b_dev, ARRAYSIZE * sizeof(int),
                    ↪  cudaMemcpyDeviceToHost);
159                 cudaMemcpy(out_host, out_dev, ARRAYSIZE * sizeof(int),
                    ↪  cudaMemcpyDeviceToHost);
160
161 //                cudaEventSynchronize(stop);
162 //                cudaEventElapsedTime(&millis, start, stop);
163
164                 // Stop timer
165                 sdkStopTimer(&timer);
166
167                 // Print time to console
168                 printf("Processing time: %f (ms)\n", sdkGetTimerValue(&timer));
169 //                printf("Processing time: %f (ms)\n", millis);
170
171                 // Write timing results to file
172                 fprintf(f, "%d,%f\n", BLOCKSIZE, sdkGetTimerValue(&timer));
173 //                fprintf(f, "%f\n", sdkGetTimerValue(&timer));
174 //                fprintf(f, "%d,%f\n", BLOCKSIZE, millis);
175
176                 // Verwijder timer
177                 sdkDeleteTimer(&timer);
178
179         } //End for
180
181         // Close the file
182         fclose(f);
183
184
185         // Free up the used memory
```

```
186         free(a_host);
187         free(b_host);
188         free(out_host);
189         cudaFree(a_dev);
190         cudaFree(b_dev);
191         cudaFree(out_dev);
192 
193         return 0;
194 }
```

3 Conclusion

References