

ADD - INVERT

Labo Geavanceerde Computertechniek

(JLIZNM)

Jona CAPPELLE & JONAS BOLLE

23 maart 2020



Sessie Datum: 9 Maart, 2020

Partners: Jona Cappelle

Jonas Bolle

Klas: MELICTEES

Begeleider: Stijn Crul

Inhoudsopgave

1	Inleiding	2
2	Probleemstelling	2
3	Oplossing	3
3.1	Flowchart code	3
3.2	Code	4
4	Analyse performantie	4
4.1	ADD	4
4.1.1	Tijd op GPU	4
4.1.2	Tijd op CPU	4
4.2	INV	4
4.2.1	Tijd op GPU	4
4.2.2	Tijd op CPU	4
4.3	Vergelijking	4
5	Besluit	5
A	Grafieken	6
B	CODE	8

Lijst van figuren

1	Vergelijking architectuur CPU vs GPU [1]	2
2	Grids, blocks en threads [1]	2
3	Flowchart ADD - INV	3
4	ADD op GPU met memory copy (10 miljoen elementen)	6
5	ADD op GPU zonder memory copy (10 miljoen elementen)	6
6	INV op GPU met memory copy (100 miljoen elementen)	7
7	INV op GPU zonder memory copy (100 miljoen elementen)	7

Lijst van tabellen

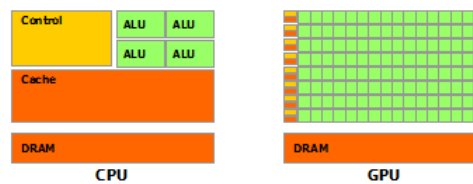
1	Vergelijkende tabel tijden GPU en CPU (met en zonder kopiëren van de data) .	5
---	------------------------------------------------------------------------------	---

1 Inleiding

In dit eerste labo geavanceerde computerarchitectuur maken we kennis met CUDA, een extensie van de C programmeertaal gemaakt door nVidia. Het grote voordeel hierbij is dat de code op de GPU kan uitgevoerd worden. Hierdoor kan de programmeur gebruik maken van de zeer grote parallelle rekenkracht van een nVidia grafische kaart. Cuda is enkel geschikt voor algoritmes die geparalleliseerd kunnen worden.

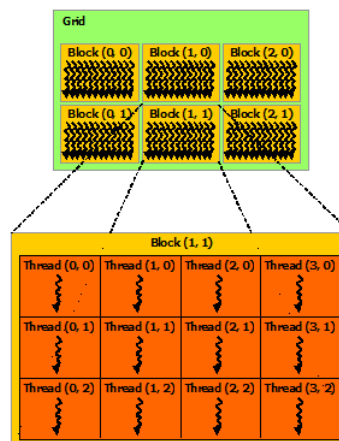
In dit eerste labo gaan we twee kleine cuda programmaatjes schrijven. Het eerste is om twee arrays op te tellen, het tweede is om een array te inverteren. [2]

De architectuur van een CPU en GPU verschilt drastisch (zie figuur 1).



Figuur 1: Vergelijking architectuur CPU vs GPU [1]

Een GPU bestaat uit grids, blocks en threads. Een grid bestaat uit verschillende blocks, die op hun beurt bestaan uit verschillende threads. Nu is het aan de programmeur om deze threads op zo'n efficiënt mogelijke manier te gebruiken. [1]



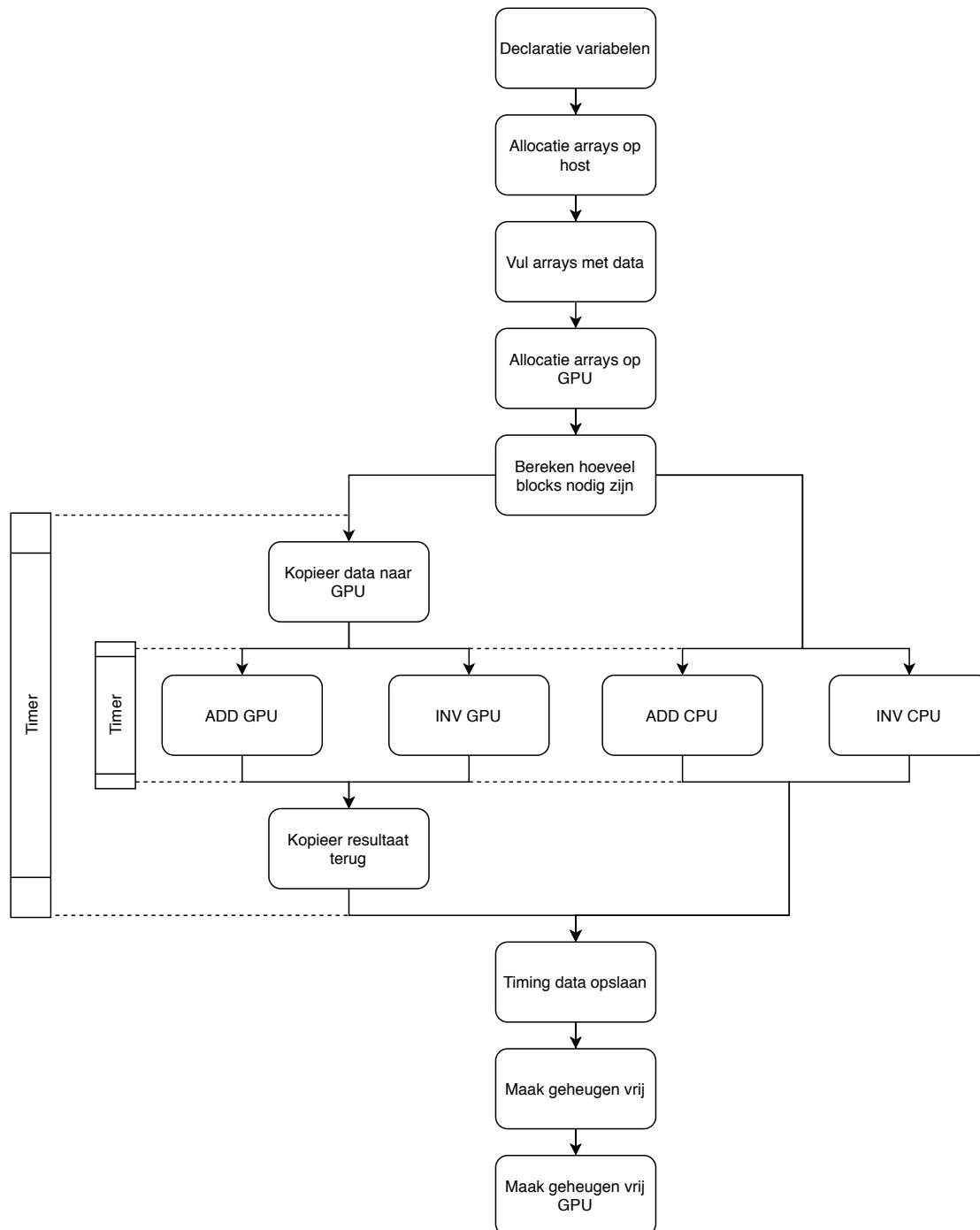
Figuur 2: Grids, blocks en threads [1]

2 Probleemstelling

We zullen in dit labo twee arrays optellen en een array inverteren. We zullen dit een aantal keren doen. De timing om al deze operaties uit te voeren zal opgenomen worden, dit zowel op de GPU als op de CPU. Hieruit zullen we later conclusies kunnen trekken. We onderzoeken ook de invloed van de blocksize op de GPU. De operaties op de GPU zullen getimed worden met het kopiëren van data naar de GPU en terug en zonder het kopiëren van de data.

3 Oplossing

3.1 Flowchart code



Figuur 3: Flowchart ADD - INV

3.2 Code

De volledige cuda code kan u terug vinden in bijlage B op pagina 8.

4 Analyse performantie

4.1 ADD

4.1.1 Tijd op GPU

In bijlage A op pagina 6 zijn grafieken terug te vinden van uitvoeringstijd in functie van het aantal blocks. Zie hiervoor figuur 4 voor deze met kopiëren van de data naar de GPU en figuur 5 voor de tijd zonder kopiëren van de data. We zien dat de snelste GPU tijden rond de $\pm 380 \mu\text{s}$ liggen met mee kopiëren van de data naar de GPU. Zonder het kopiëren van de data bekomen we een resultaat van $\pm 25 \mu\text{s}$.

4.1.2 Tijd op CPU

Met een array grootte van 10 000 000 bekomen we op de CPU een tijd van:

$$\text{Tijd op CPU} \Rightarrow 37.12 \mu\text{s}$$

4.2 INV

4.2.1 Tijd op GPU

In bijlage A op pagina 6 zijn grafieken terug te vinden van uitvoeringstijd in functie van het aantal blocks. Zie hiervoor figuur 6 voor deze met kopiëren van de data naar de GPU en figuur 7 voor de tijd zonder kopiëren van de data. We zien dat de snelste GPU tijden rond de $\pm 375 \mu\text{s}$ liggen met het mee kopiëren van de data naar de GPU. Zonder het kopiëren van de data bekomen we een sneller resultaat van $\pm 22 \mu\text{s}$.

4.2.2 Tijd op CPU

Met een array grootte van 100 000 000 bekomen we op de CPU een tijd van:

$$\text{Tijd op CPU} \Rightarrow 33.33 \mu\text{s}$$

4.3 Vergelijking

Onderstaande tabel 1 geeft een duidelijk overzicht van de verschillende tijden op de GPU (met en zonder kopiëren van de data) en op de CPU.

	ADD	INV
Tijd GPU (met kopiëren) [μ s]	380	375
Tijd GPU (zonder kopiëren) [μ s]	25	22
Tijd CPU [μ s]	37.12	33.33

Tabel 1: Vergelijkende tabel tijden GPU en CPU (met en zonder kopiëren van de data)

5 Besluit

Uit bovenstaande tabel 1 kunnen we besluiten dat het uitvoeren van een bewerking op de GPU veel sneller kan verlopen dan op een CPU. Dit is net omdat een GPU gebruik kan maken van zeer veel parallele processoren om het rekenwerk te verdelen, terwijl een CPU krachtigere processoren heeft maar het werk meer sequentiëel moet werken.

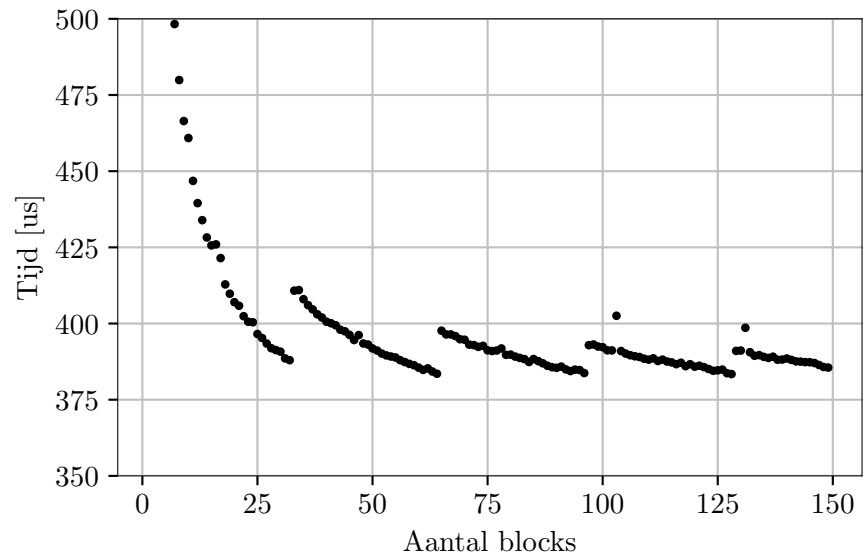
We zien duidelijk dat het kopiëren van data van en naar de GPU zeer veel tijd in beslag neemt. Dit weegt echter niet op tegen de prestatieverbetering van het uitvoeren van een parallel algoritme op de GPU. Bij grote workloads wordt het snelheidsverlies door het kopiëren wordt meer dan goed gemaakt door het sneller berekenen.

Bij veelvouden van 32 blocks werkt de GPU het snelste. Alle threads in één block steken is zeker niet aan te raden, dit gaf de langste uitvoeringstijd. We zien ook naar mate het aantal blocks stijgt het minder uitmaakt of we met een veelvoud van 32 aan het werken zijn.

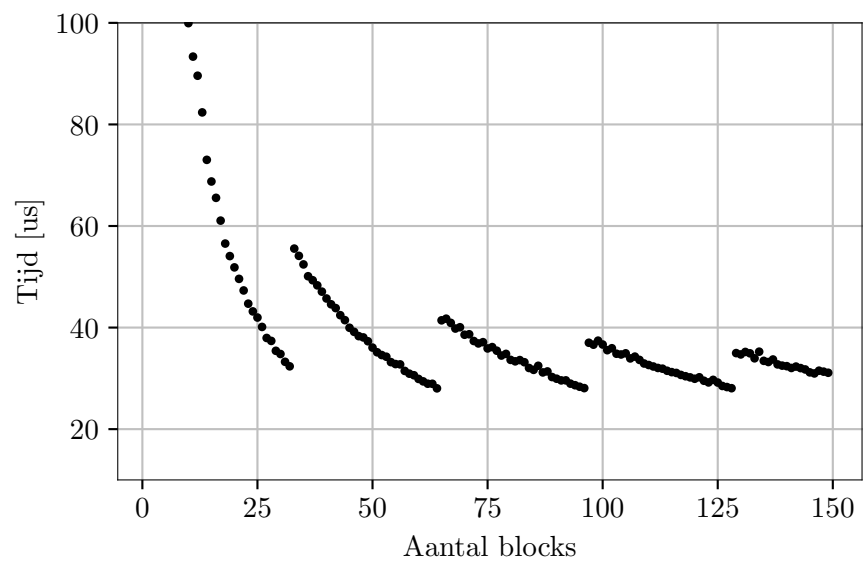
We kunnen ook besluiten dat een optelling meer bewerkingstijd in beslag neemt dan het realloceren van geheugenplaatsen. We zien dit verschil duidelijk tussen INV, die sneller uitvoert dan ADD.

Het besluit is dus, mits het correct alloceren van data op de GPU, het voor grote parallele workloads veel interessanter is om een GPU te gebruiken dan een CPU.

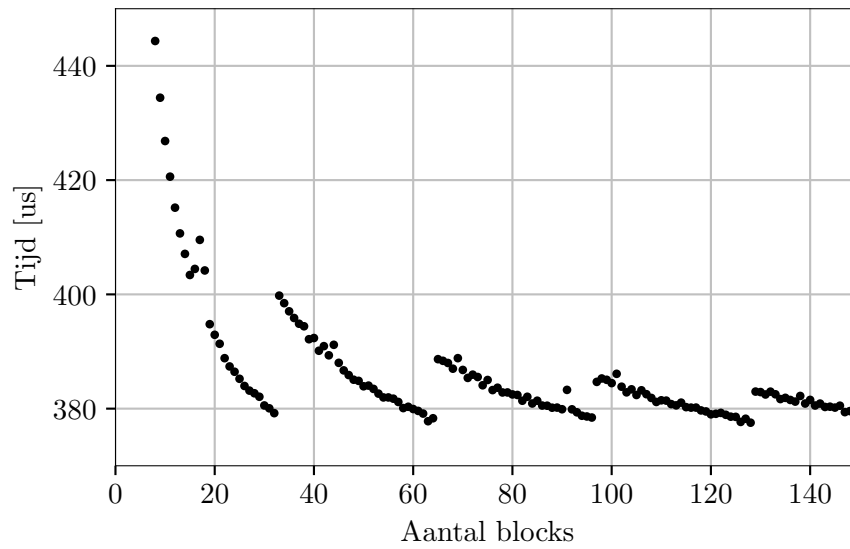
A Grafieken



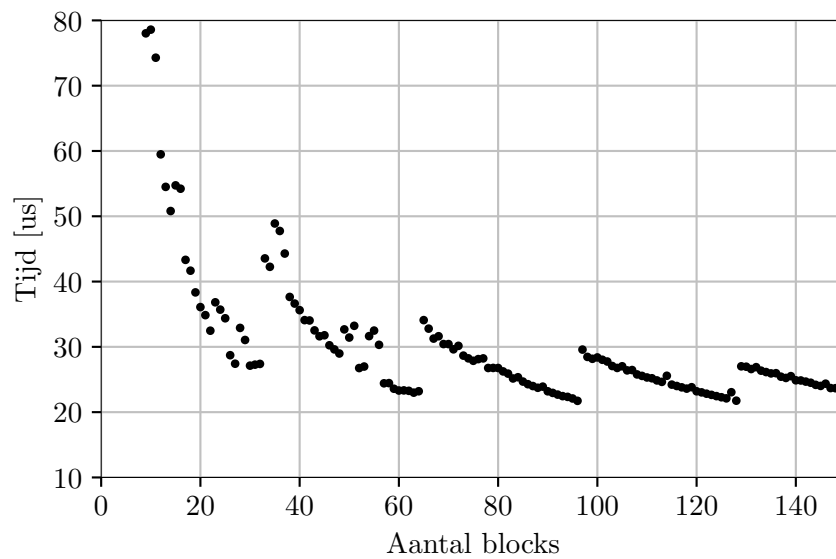
Figuur 4: ADD op GPU met memory copy (10 miljoen elementen)



Figuur 5: ADD op GPU zonder memory copy (10 miljoen elementen)



Figuur 6: INV op GPU met memory copy (100 miljoen elementen)



Figuur 7: INV op GPU zonder memory copy (100 miljoen elementen)

B CODE

```

1  //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
2  // ADD / INVERT -- Jona Cappellet -- 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 // own 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 // SELECT GPU - CPU TIMING
28 // #define GPU
29 // #define CPU
30 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
31 // RUN ADD - INV
32 // #define ADD
33 // #define INV
34 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
35
36 // Helper function
37 void init_array(int *a)
38 {
39     for (int i = 0; i < ARRAYSIZE; i++)
40     {
41         a[i] = i;
42     }
43 }
44
45 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
46 // KERNEL ADD
47 //////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```
48  int BLOCKSIZE;
49
50  // GPU
51  __global__ void add(int *a, int *b, int *out)
52  {
53      int idx = blockIdx.x * blockDim.x + threadIdx.x;
54      if (idx < ARRAYSIZE)
55      {
56          out[idx] = a[idx] + b[idx];
57      }
58  }
59
60  // CPU
61  void cpu_add(int *a, int *b, int *out)
62  {
63      for (int i = 0; i < ARRAYSIZE; i++)
64      {
65          out[i] = a[i] + b[i];
66      }
67  }
68
69  ///////////////////////////////////////////////////
70  // KERNEL INVERT
71  ///////////////////////////////////////////////////
72
73  // GPU
74  __global__ void invert(int *a, int *out)
75  {
76      int idx = blockIdx.x * blockDim.x + threadIdx.x;
77      if (idx < ARRAYSIZE)
78      {
79          out[idx] = a[ARRAYSIZE - 1 - idx];
80      }
81  }
82
83  // CPU
84  void cpu_invert(int *a, int *out)
85  {
86      for (int i = 0; i < ARRAYSIZE; i++)
87      {
88          out[i] = a[ARRAYSIZE - 1 - i];
89      }
90  }
91
92  ///////////////////////////////////////////////////
93  // Program main
94  ///////////////////////////////////////////////////
95  int main()
96  {
```

```
97      // declare variables
98      int *a_host, *b_host, *out_host;
99      int *a_dev, *b_dev, *out_dev;
100
101      // allocate arrays on host
102      a_host = (int *)malloc(ARRAYSIZE * sizeof(int));
103      b_host = (int *)malloc(ARRAYSIZE * sizeof(int));
104      out_host = (int *)malloc(ARRAYSIZE * sizeof(int));
105
106      // initialize arrays with zeros
107      init_array(a_host);
108      init_array(b_host);
109
110      // allocate arrays on device
111      cudaMalloc((void **)&a_dev, ARRAYSIZE * sizeof(int));
112      cudaMalloc((void **)&b_dev, ARRAYSIZE * sizeof(int));
113      cudaMalloc((void **)&out_dev, ARRAYSIZE * sizeof(int));
114
115      // Record time on GPU with cuda events
116      #ifdef GPU
117          cudaEvent_t start, stop;
118          cudaEventCreate(&start);
119          cudaEventCreate(&stop);
120      #endif
121
122      // Timer on CPU
123      #ifdef CPU
124          clock_t start, end;
125          double cpu_time_used;
126      #endif
127
128      // Initialize data file where the timing results will be stored
129      FILE *f = fopen("data.csv", "w");
130
131      #ifdef GPU
132          for (int BLOCKSIZE = 1; BLOCKSIZE < 300; BLOCKSIZE++)
133          {
134      #endif
135
136      #ifdef CPU
137          float millis = 0;
138      #endif
139
140      // Calculate amount of blocks needed
141      int nBlocks = ARRAYSIZE / BLOCKSIZE + (ARRAYSIZE % BLOCKSIZE ==
142          ↪ 0 ? 0 : 1);
143      printf("Nbblocks: %i", nBlocks);
144
145      // Start timer
```

```

145  #ifdef CPU
146      StopwatchInterface *timer = 0;
147      sdkCreateTimer(&timer);
148      sdkStartTimer(&timer);
149  #endif
150
151  #ifdef GPU // Start cuda event on GPU
152      cudaEventRecord(start);
153  #endif
154
155      //Step 1: Copy data to GPU memory
156      cudaMemcpy(a_dev, a_host, ARRAYSIZE * sizeof(int),
157          ↪ cudaMemcpyHostToDevice);
158      cudaMemcpy(b_dev, b_host, ARRAYSIZE * sizeof(int),
159          ↪ cudaMemcpyHostToDevice);
160      cudaMemcpy(out_dev, out_host, ARRAYSIZE * sizeof(int),
161          ↪ cudaMemcpyHostToDevice);
162
163      ////////////////////////////////////
164      // GPU -- define ADD - INV to run
165      ////////////////////////////////////
166  #ifdef ADD && GPU
167      add<<<nBlocks, BLOCKSIZE>>>(a_dev, b_dev, out_dev);
168  #endif
169  #ifdef INV && GPU
170      invert <<< nBlocks, BLOCKSIZE >>> ( a_dev, out_dev );
171  #endif
172
173  #ifdef GPU // Stop GPU event timer
174      cudaEventRecord(stop);
175  #endif
176
177      ////////////////////////////////////
178      // CPU -- define ADD - INV to run
179      ////////////////////////////////////
180  #ifdef CPU // CPU timer
181      start = clock();
182  #endif
183
184  #ifdef ADD && CPU
185      cpu_add( a_host, b_host, out_host);
186  #endif
187  #ifdef INV && CPU
188      cpu_invert ( a_host, out_host );
189  #endif
190
191  #ifdef CPU
192      end = clock();
193      cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

```

```
191         printf("%f", cpu_time_used);
192     #endif
193
194     //Step 4: Retrieve result
195     cudaMemcpy(a_host, a_dev, ARRAYSIZE * sizeof(int),
196         ↪ cudaMemcpyDeviceToHost);
197     cudaMemcpy(b_host, b_dev, ARRAYSIZE * sizeof(int),
198         ↪ cudaMemcpyDeviceToHost);
199     cudaMemcpy(out_host, out_dev, ARRAYSIZE * sizeof(int),
200         ↪ cudaMemcpyDeviceToHost);
201
202     #ifdef GPU
203         cudaEventSynchronize(stop);
204         cudaEventElapsedTime(&millis, start, stop);
205     #endif
206
207     // Stop timer
208     #ifdef CPU
209         sdkStopTimer(&timer);
210     #endif
211
212     // Print time to console
213     #ifdef CPU
214         printf("Processing time: %f (ms)\n", sdkGetTimerValue(&timer));
215     #endif
216     #ifdef GPU
217         printf("Processing time: %f (ms)\n", millis);
218     #endif
219
220     // Write timing results to file
221     #ifdef CPU
222         fprintf(f, "%f\n", sdkGetTimerValue(&timer));
223     #endif
224     #ifdef GPU
225         fprintf(f, "%d,%f\n", BLOCKSIZE, millis);
226     #endif
227
228     // Verwijder timer
229     #ifdef CPU
230         sdkDeleteTimer(&timer);
231     #endif
232
233     #ifdef GPU
234         } //End for
235     #endif
236
237     // Close the file
238     fclose(f);
239
240     // Free up the used memory
241     free(a_host);
```

```
237         free(b_host);
238         free(out_host);
239
240     #ifdef GPU // Free up the cuda memory
241         cudaFree(a_dev);
242         cudaFree(b_dev);
243         cudaFree(out_dev);
244     #endif
245
246     return 0;
247 }
```

Referenties

- [1] “Programming Guide :: CUDA Toolkit Documentation.” [Online]. Available: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/>
- [2] “What is CUDA? An Introduction — The Supercomputing Blog.” [Online]. Available: <http://supercomputingblog.com/cuda/what-is-cuda-an-introduction/>