Practicum Parallel Programmeren: eerste twee sessies

In de eerste twee practicumsessies maak je twee "kleinere" oefeningen om vertrouwd te geraken met OpenCL. Je kan hierbij vertrekken van een aantal bestanden die je hier kan downloaden:

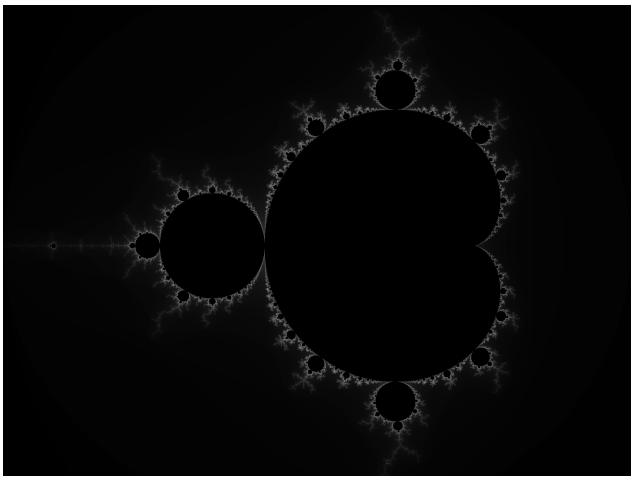
`http://www.cs.kuleuven.be/~joost/DN/ocl.tar.gz`

Na het uitpakken (`tar -xvzf ocl.tar.gz`) van dit bestand vind je o.a. de volgende nuttige bestanden om van te vertrekken:

- De bestanden `common/ocl_utils.c` en `.h` bevatten de standaard *boilerplate* code van OpenCL.
- In `ocl_add_numbers/main.c` en `ocl_add_numbers/kernel.cl` wordt deze *boilerplate* code gebruikt om twee vectoren van getallen met elkaar op te tellen (zoals in het voorbeeld uit de eerste theorieles).

Zorg om te beginnen dat je het programma `ocl_add_numbers/main.c` alvast eens kan compileren en uitvoeren.

1 Mandelbrot



De welbekende Mandelbrot tekeningen worden als volgt gemaakt: je neemt een complex getal $c = a + bi$, doet er herhaaldelijk een bepaalde operatie op, en als het resultaat uiteindelijk groter wordt dan een bepaalde grenswaarde, dan kleur je het punt (a, b) zwart; anders blijft (a, b) wit. Om de tekeningen nog wat mooier te maken, kan je ook werken met grijswaardes, gebaseerd op het aantal iteraties van de operatie dat je nodig hebt om de grenswaarde te overschrijden.

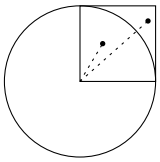
Het bestand `mandelbrot/main.c` bevat een C-programma dat een dergelijke Mandelbrot figuur tekent. In de functie `render_mandelbrot` bevindt zich een dubbele `for`-lus, die itereert over coördinaten `(pos_x, pos_y)` en voor elk paar de functie `calc_mandel_pixel` oproept om de grijswaarde te berekenen die op deze positie getekend moet worden.

Parallelliseer dit programma door deze dubbele `for`-lus te vervangen door een OpenCL kernel. Elke thread moet voor één pixel de grijswaarde berekenen.

Meet of je OpenCL programma sneller/trager is dan het originaal. Hiervoor kan je de functies in `common/time_utils.c` en `.h` gebruiken. Je roept de functie `time_measure_start(string)` op om een timer te starten met naam `string`. Daarna nog eens de functie `time_measure_stop(string)` oproepen met dezelfde string als argument geeft je het aantal seconden dat sindsdien verstreken is.

2 Berekenen van π

Op basis van het eerste kwadrant van een eenheidscirkel kan je de waarde van π schatten. De oppervlakte van een kwart van een eenheidscirkel is immers $\frac{\pi}{2}$. De oppervlakte van het vierkant op onderstaande figuur is 1. We genereren nu een heel aantal willekeurige punten (x, y) in dit vierkant, door telkens een willekeurige $x \in [0, 1]$ en $y \in [0, 1]$ te nemen. Voor zo'n punt berekenen we dan de afstand tot de oorsprong $\sqrt{x^2 + y^2}$. Als deze ≤ 1 is, dan ligt het punt binnen de eenheidscirkel. Als we n willekeurige punten gegenereerd hebben en i ervan lagen binnen de eenheidscirkel, dan geeft de verhouding $\frac{i}{n}$ ons een schatting voor de waarde van $\frac{\pi}{2}$. Naarmate n groter wordt, wordt deze schatting steeds exacter.



In het bestand `pi/main.c` vind je C-code die dit algoritme implementeert. In de functie `calc_pi` staat een `for`-lus die n keer de aanmaak en test van een willekeurig punt (x, y) uitvoert. Deze moet je nu parallelliseren. Doe dit in verschillende stappen. Meet bij elke stap of de nieuwe code sneller/trager is dan de oude.

- Genereer op de host n willekeurige punten (x, y), laat dan n threads op de GPU in parallel beslissen of elk van deze punten in de eenheidscirkel ligt en het resultaat (0 of 1) wegschrijven naar een array van grootte n. De host telt dan het aantal enen in deze array en berekent daarmee de schatting voor π.
- Laat daarna elke thread ipv. 1 sample er onmiddellijk 64 genereren en het aantal van deze 64 samples tellen dat in de cirkel valt. De host moet dan opnieuw al deze aantallen optellen en delen door 64∗`aantal threads`.
- Gebruik dezelfde methode als in de les om per workgroup de resultaten van de verschillende workitems al op te tellen, zodat de host alleen nog maar de resultaten van al de workgroups moet combineren.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	·	5	·	9	·	13	·	17	·	21	·	25	·	29	·
6		·		22		·		38		·		54		·	
28				·				92				·			
120								·							