Verslag OpenCL : B-Body

# Doel

Het gekregen N-Body programma versnellen met behulp van parallelle programmatie.

In dit gegeven programma wordt de snelheid en de locatie van een object berekend aan de hand van for lussen. Wanneer dit programma wordt uitgevoerd op de CPU zorgt dit voor lange berekentijden (bij hoge hoeveelheden objecten).

# Oplossing

## Versie 1

In de eerste versie hebben we enkel de positie berekening op de GPU uitgevoerd. Dit bracht echter een minieme tijdsversnelling met zich mee doordat we nu telkens de data moeten uitwisselen tussen de CPU en de GPU.

## Versie 2

In deze versie berekenen we enkel de snelheid via de GPU. Aangezien de snelheid een dubbele for lus bevat (en de positie slecht een enkele) bracht dit een zeer grote versnelling met zich mee. Helaas kunnen we deze versie niet gebruiken doordat we “race conditions” hebben. Dit houdt in dat een thread de waarde van een object wil lezen terwijl een andere thread een waarde naar deze thread schrijft. Hierdoor krijgen we elke keer dat we het programma runnen andere waardes.

## Versie 3

In deze versie hebben we de “race conditions” van versie 2 opgelost aan de hand van atomische operaties. Deze operaties worden uitgevoerd worden uitgevoerd zonder tussenkomst van andere thread. Dit betekent dus dat er geen waardes meer overschreven worden. Dit gaat helaas gepaard met een grote vertraging in ons programma aangezien we nu telkens op een andere thread moeten wachten voor we verder kunnen rekenen.

## Versie 4

In deze versie hebben we zowel de positie berekening als de snelheidsberekening op de GPU uitgevoerd. Ook hebben we nu voor de positieberekening een atomic add gebruikt. Dit was eigenlijk niet nodig aangezien er geen “race conditions” zijn bij de positieberekening.

## Versie 5

In deze versie hadden we besloten dat het niet nodig was om een atomic add te gebruiken indien we enkel de buitenste lus van de snelheidsberekening parallel zouden uitvoeren. De binnenste for lus wordt volledig in de kernel geplaatst. Dit bracht een zeer grote versnelling met zich mee.

## Versie 6

In deze versie hebben we ook de atomic add uit de positieberekening gehaald, maar dit had weinig effect op de snelheid.

# Computer Specificities

* CPU : Intel i7-4790 @ 3.6GHz
* GPU : Nvidia GeForce GTX 745
* RAM : 15.6 GB

# Resultaten

Zoals zichtbaar op onderstaande tabellen en grafieken zien we de voordelen van de parallellisatie pas na enkele honderden lichamen zichtbaar wordt. Dit komt omdat de het parallel maken van de programma’s wat overhead met zich meebrengt. Bij weinig lichamen zorgt dit dus voor een relatief grote vertraging maar bij grotere hoeveelheden lichamen is uitvoeringstijd van de overhead code verwaarloosbaar in vergelijking met de totale uitvoeringstijd.

Men kan opmerken dat de grote boosdoener van het N-Body programma de snelheid berekening is. Het parallel maken van de positie berekening heeft maar een zeer kleine invloed op de uitvoeringstijd. Ook is de vertraging van de atomic add duidelijk te zien door naar het tijdsverschil tussen versie 2, 5 en 6 (zonder atomic add) en versie 3 en 4 (met atomic add) te kijken.

Simulatie tijd (seconden)

Aantal lichamen

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | CPU | GPU (1) | GPU (2) | GPU (3) | GPU (4) | GPU (5) | GPU (6) |
| 50 | 0.000056 | 0.000332 | 0.000338 | 0.000333 | 0.000405 | 0.000360 | 0.000385 |
| 100 | 0.000233 | 0.000332 | 0.000338 | 0.000370 | 0.000420 | 0.000400 | 0.000402 |
| 500 | 0.005505 | 0.005480 | 0.000502 | 0.001200 | 0.001059 | 0.000600 | 0.000607 |
| 1000 | 0.021612 | 0.021540 | 0.000972 | 0.002500 | 0.002485 | 0.000881 | 0.000848 |
| 5000 | 0.533493 | 0.534666 | 0.017238 | 0.046554 | 0.045107 | 0.009123 | 0.008035 |
| 10000 | 2.138698 | 2.126394 | 0.066743 | 0.173792 | 0.171723 | 0.031600 | 0.032613 |
| 50000 | 53.050021 | 52.961413 | 1.503233 | 4.224171 | 4.224585 | 0.705520 | 0.705406 |