Monte Carlo Simulaties - Oefening 1

Ruben Van Assche

10 november 2017

1 Opgave

Het doel van deze oefening was het berekenen van een benadering van het volume met volgende vergelijkingen:

$$\begin{cases} 0 \leq x \leq y \\ 1 \leq y \leq 2 \\ -1 \leq z \leq 3 \\ exp(x) \leq y \\ y * sin(z) \geq 0 \end{cases}$$

Als methode zullen we gebruik maken van Monte Carlo benaderingen.

2 Software

In deze opgave moest gebruik gemaakt worden van de Intel Math Kernel Library[1] en OpenMP voor het berekenen van random numbers. Dit vereiste een andere manier van compileren dan met bijvoorbeeld GSL, meer info hierover in `README.md`.

De versie die ik heb gebruikt bij dit project van de Intel MKL 2018.0.104, voor het gebruik van OpenMP is geen extra library vereist maar de ingebouwde compiler op macOS High Sierra(clang) ondersteunt geen OpenMP. Een oplossing hiervoor was het compileren van dit project met GCC 4.7[2].

3 Random Number Generatie

De Monte Carlo methode vereist een gigantisch aantal random punten voor een berekening van het volume voor de gegeven vergelijkingen. Deze punten genereren we d.m.v. de Intel MKL. Deze library voorziet enkele random number generators in het Vector Statistics gedeelte van de library. MKL gebruikt vectoren omdat deze methode bijna even snel als een scalar random number generator getallen genereert. Met natuurlijk het verschil dat er een vector met meerdere getallen wordt gegenereerd.

Als BRNG koos ik voor MCG59 omdat deze en bredere periode heeft dan MCG31m1 en ondersteuning voor leapfrogging(zie later) biedt. Hiernaast was het ook belangrijk om de random numbers parallel te berekenen, gelukkig is de MKL library hier specifiek voor ontworpen.

3.1 Parallelle Generatie

Elke BRNG in MKL wordt geïnitialiseerd met een stream, een functie waaruit de BRNG initiële waarden haalt. D.m.v. OpenMP zullen we ervoor zorgen dat er in verschillende threads BRNG's draaien die random numbers genereren. Er wordt ook gebruik gemaakt van de reduction methode bij de OpenMP call, deze zal ervoor zorgen dat de variabelen die bijhouden hoeveel punten er in de figuur liggen + het totaal aantal punten niet corrupt worden.

Het corrupt worden van deze variabelen kan komen doordat beide threads naar een gemeenschappelijke variabele willen schrijven tegelijk.

Een zeer belangrijk punt waarbij opgelet moet worden is dat de threads verschillende initiële waarden gebruiken voor hun RNG's anders zou het kunnen dat bepaalde threads random number sequenties gaan hebben die zeer sterk gelijk aan elkaar zijn waardoor de uitkomst van de volume berekening fout kan zijn. MKL heeft twee methoden om dit probleem op te lossen: een eerste is d.m.v. Block Splitting waarbij de random numbers wordt opgesplitst in verschillende niet

[1] `https://software.intel.com/en-us/mkl`
[2] `https://gcc.gnu.org`

overlappende blokken Elke individuele stream wordt toegewezen aan een bepaald blok waaruit dan random numbers komen.

Een tweede methode is leapfrogging waarbij de stream wordt opgesplitst in verschillende sequenties die disjunct zijn. In tegenstelling tot Block Splitting wordt geen volledig blok per stream overgeslagen maar worden er d.m.v. een gegeven individuele stream bepaalde getallen wel of niet geselecteerd. Een minpunt van leapfrogging is dat van zodra het aantal individuele stream groeit er geen zekerheid is dat de getallen wel ëcht"random zijn.

Uiteindelijk koos ik voor leapfrogging, omdat deze niet vereist om op te geven hoe groot een blok gaat zijn + het aantal individuele streams zal nooit groter dan 4 worden omdat dit het aantal cores in mijn computer is. En het mij het meest performant leek om per core een thread toe te wijzen.

4 Monte Carlo

Het principe van Monte Carlo is zeer simpel: genereer een hoop random punten in een bepaalde ruimte en kijk welk percentage van deze punten in de figuur liggen. Dit wil zeggen dat de punten voldoen aan de gegeven vergelijkingen.

4.1 Dimensies voor de ruimte kiezen

Ten eerste moet er geselecteerd worden hoe groot de figuur maximaal kan worden en dus hoe groot de ruimte zal zijn waarin punten worden gegenereerd. We zouden gewoon een ruimte kunnen nemen waarbij x, y, z lopen van $[-10, 10]$ maar het probleem hierbij is dat er dan veel meer random punten vereist zijn voor een goede benadering van het volume. Wanneer we de vergelijkingen bekijken zien we dat $y \in [1, 2]$ en $z \in [-1, 3]$, de x is iets moeilijker gezien $x \in [0, y]$ maar gezien y niet groter kan worden dan 2 kiezen we voor $x \in [0, 2]$.

Deze intervallen voor de x, y, z waarden van een punt zullen dan ook liggen in een ruimte met een volume 8(1*4*2). De random number generator produceert getallen in een range $[0, 1]$ dus voor elk getal zullen we aanpassingen moeten maken zodat deze vallen in het interval van een x, y, z coördinaat:

$$x \to 2 * x$$

$$y \to y + 1$$

$$z \to 4z - 1$$

4.2 Controleren van punten

Nu de punten gegenereerd worden rest het ons enkel nog te checken of deze in het volume liggen dat benadert moet worden. Dit doen we door voor elk punt te checken of het voldoet aan de gegeven vergelijkingen. Het is mogelijk om enkele van de gegeven vergelijkingen te schrappen, dit omdat sowieso al aan deze voorwaarden voldaan wordt door de keuze van de intervallen waarin random numbers vallen. Volgende vergelijkingen laten we achterwege:

$$0 \leq x \leq y$$

$$1 \leq y \leq 2$$

$$-1 \leq z \leq$$

het is mogelijk om $0 \leq x \leq y$
te schrappen omdat $exp(x) \leq y$ de conditie al bevat. Uiteindelijk houden we volgende vergelijkingen over:

$$exp(x) \leq y$$

$$y * sin(z) \geq 0$$

Indien een punt aan deze voorwaarden voldoet rekenen we het als een punt dat in het te berekenen volume valt.

4.3 Berekening Volume

Na het berekenen van vele punten hebben we een variabele `figure_tk`, het aantal punten dat in het volume ligt. En `figure_tn` het totaal aantal punten in de ruimte. Vervolgens berekenen we het volume `volume_tv` alsvolgt:

$$volume\_tv = 8 * \frac{volume\_tk}{volume\_tn}$$

Deze 8 komt uit de berekening in sectie 4.1.

5 Resultaten

Om een goede benadering te maken van het volume zijn veel punten vereist, in de tabel hieronder staat de waarde voor het volume en het aantal berekende punten.

Aantal Punten	**Volume**
40000	1.1566
80000	1.1612
160000	1.15705
320000	1.156
640000	1.15565
1280000	1.15791
2560000	1.15832
5120000	1.15875
10240000	1.15859
20480000	1.15883
40960000	1.15889
81920000	1.15882
163840000	1.15881
327680000	1.15878
655360000	1.15885
1310720000	1.15875

Meer punten heb ik niet meer kunnen berekenen omdat mijn computer haar grootte van integer te klein werd met als gevolg dat ik een negatief aantal punten verkreeg. Een mogelijke oplossing hiervoor is het gebruik van een 64-bit integer, deze zou veel meer punten kunnen opslaan waardoor we een nog nauwkeurigere berekening zouden kunnen krijgen. Maar mijn computer moest voor deze berekeningen al aardig wat CPU-tijd geven waardoor het berekenen van de volgende stap waarschijnlijk zeer lang zou duren. Gezien we zien dat het volume duidelijk converteert naar 1.158 zullen we hierbij de berekening afronden. We vinden dus dat het volume 1.15875 met 1.310.720.000 punten.