OpenCL n-body

Vandevelde, Simon (simon.vandevelde@student.kuleuven.be)
Van Assche, Dylan (dylan.vanassche@student.kuleuven.be)

20 mei 2018

1 Testomgeving

1.1 Compilatieproblemen

Door het gebruik van verouderde packages zat er een bug in het `GLM` package wat ervoor zorgde dat de code niet gecompileerd kon worden in onze testomgeving. Na het bijwerken van `GLM` waren de problemen van de baan.

1.2 Hardware

- CPU
- GPU
- RAM
- OS

2 OpenCL aanpassingen

2.1 for-lus 1

```
for (int i = 0; i < length; ++i)
{
    for (int j = 0; j < length; ++j)
    {

        if (i == j)
            continue;

        cl_float3 pos_a = host_pos[i];
        cl_float3 pos_b = host_pos[j];
```

```
        float dist_x = (pos_a.s[0] - pos_b.s[0]) *
           distance_to_nearest_star;
        float dist_y = (pos_a.s[1] - pos_b.s[1]) *
           distance_to_nearest_star;
        float dist_z = (pos_a.s[2] - pos_b.s[2]) *
           distance_to_nearest_star;


        float distance = sqrt(
                dist_x * dist_x +
                dist_y * dist_y +
                dist_z * dist_z);

        float force_x = -mass_grav * dist_x / (distance *
            distance * distance);
        float force_y = -mass_grav * dist_y / (distance *
            distance * distance);
        float force_z = -mass_grav * dist_z / (distance *
            distance * distance);

        float acc_x = force_x / mass_of_sun;
        float acc_y = force_y / mass_of_sun;
        float acc_z = force_z / mass_of_sun;

        host_speed[i].s[0] += acc_x * delta_time;
        host_speed[i].s[1] += acc_y * delta_time;
        host_speed[i].s[2] += acc_z * delta_time;
    }

}
```

2.2 for-lus 2

```
for(int i = 0; i < length; ++i)
{
    host_pos[i].s[0] += (host_speed[i].s[0] * delta_time)
        / distance_to_nearest_star;
    host_pos[i].s[1] += (host_speed[i].s[1] * delta_time)
        / distance_to_nearest_star;
    host_pos[i].s[2] += (host_speed[i].s[2] * delta_time)
        / distance_to_nearest_star;
}
```

2.3 for-lus 1 + 2

```
for(int i = 0; i < length; ++i)
```

```
{
    host_pos[i].s[0] += (host_speed[i].s[0] * delta_time)
        / distance_to_nearest_star;
    host_pos[i].s[1] += (host_speed[i].s[1] * delta_time)
        / distance_to_nearest_star;
    host_pos[i].s[2] += (host_speed[i].s[2] * delta_time)
        / distance_to_nearest_star;
}
```

2.4 for-lus 1 atomisch

2.5 for-lus 1 + 2 met afstanden

3 Resultaten

Om onze resultaten te verwerken hebben we een klein Python script geschreven dat alle timestamps inleest en het gemiddelde neemt van de 100 eerste metingen per categorie. Deze gemiddelden worden dan uitgezet op een aantal grafieken welke hieronder zullen worden besproken.

3.1 5 lichamen

3.2 50 lichamen

3.3 500 lichamen

3.4 5 000 lichamen

3.5 50 000 lichamen

3.6 500 000 lichamen

CPU faalt!

4 Mogelijke verbeteringen

Het is nog mogelijk om de laatste versie van het programma nog te versnellen door gebruik te maken van nog enkele OpenCL functies en ontwerppatronen. Deze werden niet geïmplementeerd wegens tijdsgebrek.

4.1 Minder dataoverdrachten

Het programma kopieert nu telkens per kernel de data van de host naar de GPU en omgekeerd. Dit resulteert in 4 dataoverdrachten omdat beide for-lussen draaien als een aparte kernel.

We zouden de for-lussen kunnen overzetten in 1 kernel waardoor de data-overdrachten worden gereduceerd tot slechts 2 overdrachten.

4.2 Lokaal geheugen

Momenteel maakt het programma gebruik van globaal geheugen in de GPU. Elke processor kan aan de hele dataset. Beter gezegd: het geheugen is volledig gedeeld onder elke processor. Hierdoor staat dit globaal geheugen verder weg van de processor kern en duurt het dus langer vooraleer de data arriveert bij de processor.

Als we gebruik maken van het lokaal geheugen van een werkgroep kunnen we dit probleem voorkomen. Maar dit vereist tevens ook dat we de berekeningen zodanig opsplitsen zodat ze in een werkgroep passen en geen invloed hebben elkaar, hierover later meer in paragraaf 4.3.

4.3 Efficiënt opsplitsen in werkgroepen

Beide for-lussen worden simpel weg op de GPU uitgevoerd zonder dat het algoritme geoptimaliseerd werd voor parrallelisme op de GPU.

4.4 Coalesced memory access

5 Conclusie