Projeto 1 – Simulação de Física 2D

Super Computação – 2018/2

André De Marco Toyama
https://github.com/AToyama/supercomp

Introdução

Este projeto tem como objetivo realizar um estudo de diferentes técnicas de supercomputação e sua diferença de performance. Para realizar os testes adotamos um modelo de simulação baseado em conceitos explorados em Física do Movimento, em que simulamos a colisão de corpos em um plano 2D.

Modelo de Simulação

Para nossa simulação consideramos que cada corpo simulado possui massa, aceleração, raio, velocidade e posição, assim adotamos um modelo elástico para tratar colisão entre dois corpos. Esse modelo obedece a regra de que a quantidade de movimento e a energia cinética se mantém constantes após uma colisão, então se considerarmos uma colisão entre dois objetos de m1, m2 e velocidade v1 e v2, vemos que o momento permanece constante após a colisão na seguinte equação:

$$m1 * v1 + m2 * v2 = m1 * v1' + m2 * v2'$$

Para obtermos a velocidade do objeto após a colisão, nos baseamos na seguinte equação:

$$\mathbf{v}_1' = \mathbf{v}_1 - \frac{2m_2}{m_1 + m_2} \frac{\langle \mathbf{v}_1 - \mathbf{v}_2, \mathbf{x}_1 - \mathbf{x}_2 \rangle}{\|\mathbf{x}_1 - \mathbf{x}_2\|^2} (\mathbf{x}_1 - \mathbf{x}_2),$$
$$\mathbf{v}_2' = \mathbf{v}_2 - \frac{2m_1}{m_1 + m_2} \frac{\langle \mathbf{v}_2 - \mathbf{v}_1, \mathbf{x}_2 - \mathbf{x}_1 \rangle}{\|\mathbf{x}_2 - \mathbf{x}_1\|^2} (\mathbf{x}_2 - \mathbf{x}_1)$$

https://en.wikipedia.org/wiki/Elastic_collision

Nos aproveitamos do fato do fato de que nosso programa utiliza as componentes direcionais Vx e Vy da velocidade, então decompomos a equação de velocidade após colisão, assim usamos para o calculo as seguintes fórmulas.

$$Vx1' = \frac{Vx1 * (m1 - m2) + 2 * m2 * Vx2}{m1 + m2}$$
$$Vy1' = \frac{Vy1 * (m1 - m2) + 2 * m2 * Vy2}{m1 + m2}$$
$$Vx2' = \frac{Vx2 * (m2 - m1) + 2 * m1 * Vx1}{m1 + m2}$$
$$Vy2' = \frac{Vy2 * (m2 - m1) + 2 * m1 * Vy1}{m1 + m2}$$

Por fim temos a nossa "mesa", onde ocorre a simulação dos corpos, ela tem uma largura, altura e um coeficiente de atrito dinâmico, que é usado para o cálculo da aceleração dos corpos. Para cálcularmos o módulo da aceleração utilizamos:

$$a = mU * m * g$$

Onde mU é o coeficiente de atrito dinâmico, m é a massa e g é a acelaração gravitacional, que consideramos como $9{,}8\,m/s^2$. Assim como atualizamos a posição dos corpos baseado na velocidade, atualizamos a velocidade com base na aceleração.

Técnicas de Supercomputação

- SIMD

  SIMD (Single Instruction, Multiple Data), é um método que aplica a mesma instrução a multiplos dados simultâneamente, adequado para operações em conjunto de dados como vetores e matrizes.

- OpenMP

  OpenMP (Open Multi-Processing) é uma API para programação multi-processo. Contém um conjunto de ferramentas que permite o desenvolvimento de aplicações paralelas, influenciando o comportamento e tempo de execução.

Executando o Projeto

O projeto é compilado através de CMake, com os comandos:

- $ cmake .

- $ make

São gerados quatro diferentes arquivos executaveis no diretório:

- *visualizador* → para ver a simulação gráfica
- *mov_seq* → realizar o teste de tempo na versão sequencial
- *mov_par* → realizar o teste de tempo na versão paralelizada (OpenMP)
- *mov_simd* → realizar o teste de tempo com instruções SIMD

Os executáveis requerem um arquivo de entrada com as informações necessárias para a simulação, tanto da mesa quanto dos diferentes corpos a serem simulados. O arquivo de entrada deve seguir o seguinte formato:

W H N

mu alpha_w alpha_b

<id> <raio> <massa> <x0> <y0> <vx0> <vy0>

Onde:

- *W e H* → “width” e “height”, dimensões do campo de simulação
- *mu* → coeficiente de atrito dinâmico
- *alpha_w e alpha_b* → proporção de energia cinética mantida após colisão entre a parede e entre dois corpos.
- *N* → número de corpos a serem simulados

Enfim, as linhas seguintes devem seguir o padrão mostrado, onde cada uma representa um corpo a ser simulado e suas propriedades (id, raio, massa, x0, y0, vx0 e vy0).

Para o arquivo de entrada temos alguns exemplos prontos para testar diferentes situações:

- *input1.txt* → teste de um corpo colidindo com a parede
- *input2.txt* → teste de colisão entre dois corpos de massas diferentes
- *input3.txt* → teste de colisão simultânea entre três corpos
- *input4.txt* → teste de colisão entre vários corpos de diferentes tamanhos e massas.

Para melhor visualização, nesse específicos testes consideramos o coeficiente de atrito como 0, para podermos observar o comportamento dos corpos por um período maior.

Por fim, temos no projeto o arquivo "input_generator.py", que é basicamente um scrip de python que gera um arquivo chamado "input.txt" no qual se pode pré definir os dados da simulação. Para mudar os dados do arquivo gerado basta mudar os valores no começo do código:

```
# ------ INPUT DEFINITIONS -------

table_width = 100
table_height = 100
number_of_bodies = 100
radius = 1
mu = 0.01
speed_range = 6
mass = 1
mass_range = 0

# --------------------------------
```

- *table_width* → largura da mesa
- *table_heigth* → altura da mesa
- *number_of_bodies* → número de corpos a ser simulados
- *radius* → raio dos corpos (nesse caso todos os corpos sempre terão o mesmo raio)
- *mu* → coeficiente de atrito dinâmico
- *speed_range* → atribui uma velocidade aleatória para os corpos entre x e -x

- *mass* → massa dos corpos
- *mass_range* → atribui uma massa aleátoria entre 1 e x, se esse atributo for deixado como 0, todos os corpos terão a mesma massa com o valor de "mass".

Após compilar o projeto e gerar os executáveis, os mesmos devem ser executados da seguinte maneira:

$ ./[executável] [arquivo de entrada]

A simulação necessita de um Δt, que serve como padrão de tempo para a simulação. O mesmo pode ser passado na hora de executar o programa como uma váriavel de ambiente, se não for passado o progama assumira Δt = 0.01

$ DELTA_T=[valor] ./[executável] [arquivo de entrada]

Pode ser passado também a variável de ambiente *GUI,* se for passada a váriavel *GUI=0* então o progama irá imprimir a cada segundo, o tempo real de simulação e uma lista com os dados de todos os corpos sendo simulados. Essa váriavel de ambiente porém só pode ser utilizada no visualizador, já que os prints interferem o tempo de execução e a função dos outros arquivos é realizar teste de tempo.

$ DELTA_T=[valor] GUI=0 ./visualizador [arquivo de entrada]

Sistema

CPU: Intel Core i5-4210U CPU @ 2.7GHz

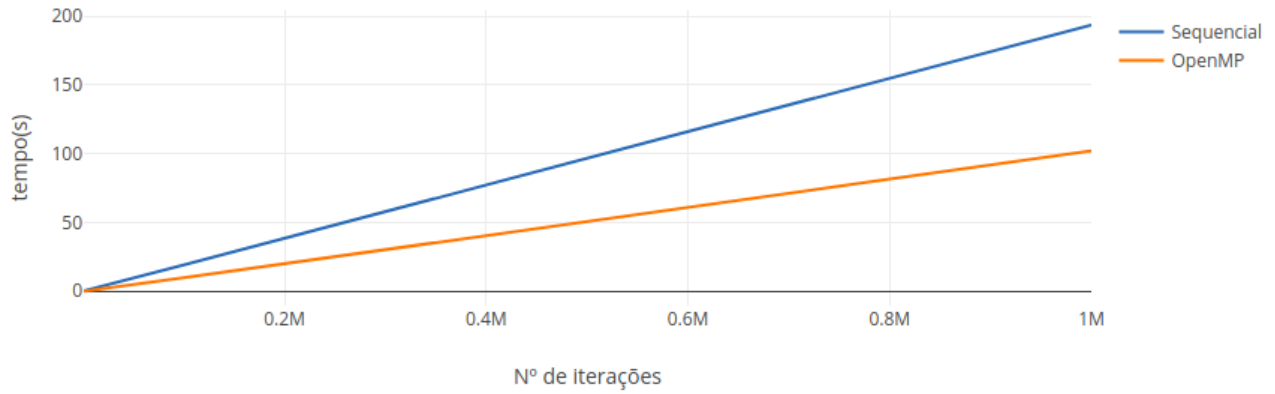
GPU: Mesa DRI Intel(R) Haswell Mobile

RAM: 8GB DDR3

Análise de Tempo

Por algum motivo, os testes de tempo realizados para o modelo SIMD foram muito inconsistentes, então resolvi realizar uma comparação apenas entre a versão sequêncial e a em que foi utilizada OpenMP, pois podemos ver o comportamento esperado. Ficará como melhoria para o futuro o entedimento do comportamento estranho do modelo SIMD, pois em vez de melhorar o tempo de execução do programa, ele foi piorado cerca de 5 vezes.

O gráfico abaixo mostra os resultados de tempo para diferentes números de iterações do programa.



Podemos ver que a versão paralelizada seguiu uma constância de melhoria de aproximadamente 50% do tempo conforme o número de iterações aumenta. Temos um resultado similar no gráfico abaixo, onde temos o tempo em segundos de acordo com a quantidade de corpos sendo simulados, em um cenário de 100.00 iterações.