Computação Paralela e Distribuída

Trabalho Académico

Simulação de Partículas

Versão 1.0

Terça-feira, 11/03/2025

Conteúdo

1. Introdução

O propósito deste trabalho académico é ganhar experiência em programação paralela em sistemas *UMA (Uniform Memory Access)* e multicomputadores, usando OpenMP e MPI, respectivamente. Para tal, os estudantes devem escrever uma implementação serial e duas paralelas de um simulador de partículas em movimento no espaço livre.

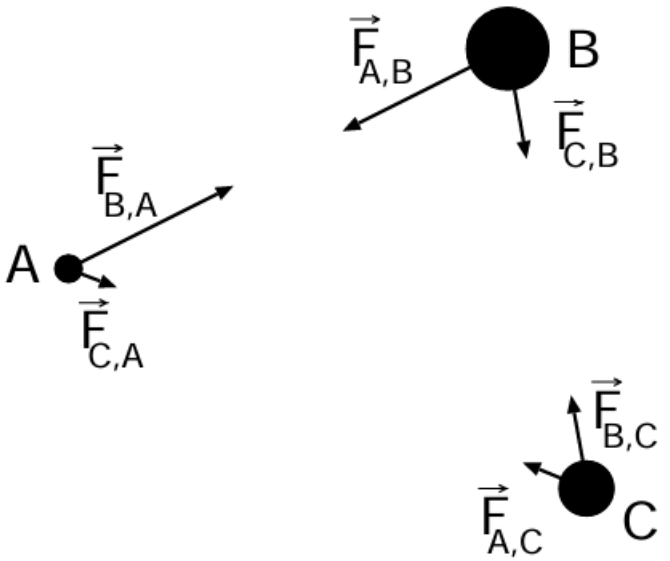
2. Descrição do Problema

Consideramos um cenário onde as partículas se movem livremente no espaço e as únicas forças às quais elas são submetidas são devidas à gravidade umas das outras. Para simplificar, assuma um espaço 2D, um quadrado com lado 1000 (as unidades de medida reais não são relevantes).

A magnitude da força entre as partículas A e B é determinada pela fórmula clássica

$$F_{A,B} = F_{B,A} = G\,\frac{m_A \times m_B}{d_{A,B}^2}$$

onde m_A,m_B são as respectivas massas, $d_{A,B}$ é a distância entre elas e G é a constante gravitacional ($6{,}67408 \times 10^{-11}$). Isso é ilustrado na figura abaixo.



Naturalmente, a força aplicada a cada partícula é a soma da atracção gravitacional de todas as outras partículas. A força resultante determina a aceleração da partícula a cada instante, que é usada para determinar, a cada passo de tempo, os novos valores da velocidade e posição da partícula:

$$\vec{F} = m.\vec{a} \qquad \vec{v}_{t+\Delta} = \vec{v}_t + \vec{a}_t.\Delta$$

$$(x,y)_{t+\Delta} = (x,y)_t + \vec{v}_t.\Delta + \frac{1}{2}\vec{a}_t.\Delta^2$$

Usaremos :

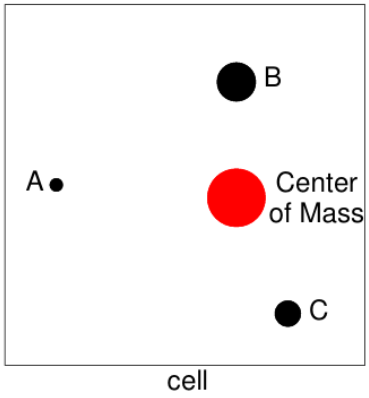
$$\Delta = 0{,}1.$$

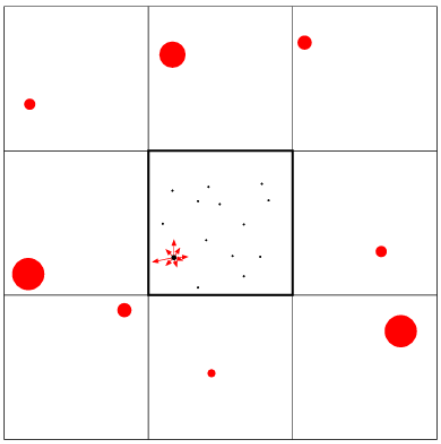
Este problema, conhecido como problema n-corpos, com complexidade $\Theta(n^2)$, é computacionalmente muito desgastante para um grande número de partículas n. Para evitar essa complexidade, para esta tarefa usamos uma aproximação chamada partícula-na-célula (PIC - particle-in-cell). O PIC funciona discretizando o espaço em uma grade (no nosso caso, células quadradas 2D). As partículas em cada célula

definem o centro de massa da célula M (a soma das massas das n partículas na célula) e as coordenadas computadas como:

$$M = \sum_{i=1}^{n} m_i \qquad (X, Y) = \frac{1}{M} \sum_{i=1}^{n} m_i \times (x_i, y_i).$$

Agora a força sobre uma partícula é calculada considerando apenas as partículas na mesma célula e os centros de massa das oito células adjacentes. A figura à esquerda abaixo ilustra a posição do centro de massa em uma célula. A figura à direita considera o espaço dividido em apenas 3×3 células, indica o centro de massas nas células adjacentes e exemplifica as forças que elas causam em uma única partícula.





Em nossa simulação, consideraremos que se duas partículas estiverem a uma distância $\epsilon = 0{,}005$, elas colidem e evaporam, portanto ambas deixam de existir e não devem ser consideradas para o restante da simulação (para simplificar o projeto, ignoramos colisões entre partículas através das células).

A simulação geral é uma sequência de intervalos de tempo, e cada intervalo de tempo consiste na seguinte sequência de operações:

1. determinar o centro de massa de cada célula;
2. calcular a força gravitacional aplicada a cada partícula;
3. calcular a nova posição de cada partícula e então a nova velocidade;
4. verificar colisões.

Assumimos que os lados se enrolam, ou seja, se uma partícula sai pelo lado do espaço, ela entra na posição correspondente no lado oposto do espaço. Note que isso também se aplica à força gravitacional, por exemplo, uma partícula em uma célula do topo é puxada para cima pelo centro de massa da célula correspondente na parte inferior.

As condições iniciais, ou seja, massa, posição inicial e velocidade de cada partícula, são definidas por uma rotina `init_particles()` fornecida no apêndice deste documento.

3. Detalhes de Implementação

3.1. Dados de Entrada

Seu programa deve permitir exactamente cinco parâmetros de linha de comando nesta ordem:

1. semente para o gerador de números aleatórios
2. tamanho do lado do espaço quadrado da simulação
3. tamanho da grade (número de células em cada lado)
4. número de partículas
5. número de passos de tempo

A semente é um inteiro de 31 bits necessário para inicializar o gerador de números aleatórios usado na rotina de inicialização, `init_particles()`. Observe que usaremos valores positivos e negativos aqui, onde o sinal indica uma distribuição de probabilidade uniforme ou normal, respectivamente. O tamanho do espaço de simulação é um `double`. Os três valores restantes são todos inteiros positivos (usarão o tamanho da grade ≥ 3).

3.2. Dados de Saída

A saída do programa consiste em duas linhas. A primeira linha são as coordenadas x e y da posição final da partícula 0, dois valores reais usando três dígitos decimais. A segunda linha é um inteiro indicando o número de partículas que colidiram durante a simulação.

Os programas submetidos devem enviar estas linhas de saída (e **nada mais**!) para a saída padrão, para que o resultado possa ser validado em relação à solução correcta.

ATÊNCÃO: O projecto **não pode ser avaliado** a menos que siga rigorosamente estas regras de entrada e saída!

3.3. Notas de Implementação

Considere o seguinte conjunto de recomendações:

- Conforme dito antes, a massa, posição inicial e velocidade de cada partícula são definidas pela rotina `init_particles()` conforme fornecido (use como está!- pode ser com ajustes apenas para os nomes nos campos da estrutura que descreve as partículas). Os três primeiros parâmetros são os três primeiros parâmetros fornecidos na linha de comando;
- Para minimizar erros numéricos devido a arredondamentos, use o tipo double para números reais;
- Podemos estar a executar simulações muito grandes, certifique-se de usar **inteiros com 64 bits** para contadores relacionados às partículas e intervalos de tempo;
- Para a fórmula que calcula a força gravitacional, use $G = 6{,}67408 \times 10^{-11}$;
- Considere um intervalo de tempo $\Delta = 0{,}1$.

3.4. Amostra de Problema

Para calcular um único passo de tempo em um problema de instância com uma grade 3×3 de tamanho 2 e 10 partículas, o comando e a respectiva saída devem ser:

```
$ ./parsim 1 2 3 10 1

1.570 0.056

0
```

Outros exemplos para ajudar a validar sua solução:

```
$ ./parsim 1 1 5 100 1

0.786 0.027

0

$ ./parsim-10 3 3 100 10

1.733 1.643

2

$ ./parsim-50 10000 200 500000 10

5025.384 5303.928

4
```

3.5. Medição do Tempo de Execução

Para garantir que todos usem a mesma medida para o tempo de execução, será usada a rotina `omp_get_wtime()` do OpenMP, que mede o tempo real (também conhecido como "wall-clock time"). Essa mesma rotina deve ser usada por todas as três versões do seu projeto. Portanto, seus programas devem ter uma estrutura semelhante a esta:

```
#include <omp.h>
<...>

int main(int argc, char *argv[])
{
     double exec_time;

     init_particles(...);
     exec_time =-omp_get_wtime();

     simulation();

     exec_time += omp_get_wtime();
     fprintf(stderr, "%.1fs\n", exec_time);
```

```
    print_result();  // usando a stream stdout!
}
```

Dessa forma, o tempo de execução contabiliza apenas o tempo de execução do algoritmo e é enviado para o erro padrão, stderr. O uso desses dois fluxos de saída permite que a validação dos resultados e a análise do tempo de execução sejam realizadas separadamente.

Como essa rotina de tempo faz parte do OpenMP, precisa incluir `omp.h` e compilar todos os seus programas com o `flag-fopenmp`.

4. Parte 1 – Implementação Serial

Escreva uma implementação serial do algoritmo em C (ou C++). Nomeie o ficheiro de código-fonte desta implementação como `parsim.c`. Conforme descrito acima, seu programa deve esperar exactamente cinco parâmetro de entrada de linha de comando.

ATÊNÇÃO: Esta será a base para comparações e espera-se que seja o mais eficiente possível.

5. Parte 2 – Implementação OpenMP

Escreva uma implementação OpenMP do algoritmo, com as mesmas regras e descrições de entrada/saída. Nomeie este código-fonte como `parsim-omp.c`.

Pode começar simplesmente adicionando diretivas OpenMP, mas é livre e incentivado a modificar o código para tornar a paralelização **mais eficaz** e **mais escalável**.

ATÊNÇÃO: Tenha cuidado com a sincronização e o balanceamento de carga!

Nota importante: para testar a escalabilidade, executaremos este programa atribuindo valores diferentes à variável shell `OMP_NUM_THREADS`. Se substituir esse valor em seu programa, não poderemos avaliar adequadamente a escalabilidade de seu programa.

6. Parte 3 – Implementação MPI

Escreva uma implementação MPI do algoritmo como para o OpenMP e resolva os mesmos problemas. Nomeie este código-fonte como `parsim-mpi.c`.

Para MPI, precisará modificar seu código substancialmente. Além da sincronização e balanceamento de carga, precisará levar em consideração a **minimização do impacto dos custos de comunicação**. É encorajado a explorar diferentes abordagens para a decomposição do problema.

ATÊNÇÃO: Créditos extras serão dados aos grupos que apresentarem uma implementação combinada de MPI + OpenMP.

7. O Que Entregar e Quando Entregar

Deve eventualmente enviar as versões sequenciais e paralelas do seu programa (**por favor, use os nomes dos ficheiros indicados acima**) e os tempos para executar as versões paralelas nos dados de entrada que serão disponibilizados (para 1, 2, 4 e 8 tarefas paralelas para ambos OpenMP e MPI, e adicionalmente 16, 32 e 64 para MPI). Observe que não usaremos nenhum nível de optimizações do compilador para avaliar o desempenho de seus programas, então também não deveria usar.

Também deve enviar um breve relatório sobre os resultados (2-4 páginas) que discuta:

- Qual a abordagem usada para paralelização?
- Que decomposição foi usada?
- Quais foram as preocupações de sincronização e porquê?
- Como foi tratado o balanceamento de carga?
- Quais são os resultados de desempenho? São o que esperava?

Entregará, inicialmente, a versão serial. Entregará a versão serial e a versão paralela do OpenMP na primeira data de entrega, com o relatório resumido, e depois a versão serial novamente (esperamos a mesma) e a versão paralela do MPI na segunda data de entrega, com um relatório actualizado.

O relatório deve ser submetido via Google Classroom, na turma a que pertence. No relatório deve conter o link do repositório GitHub do projecto. Nomeie o ficheiro como `g<n>serial.zip`, `g<n>omp.zip` e `g<n>mpi.zip`, onde `<n>` é o número do seu grupo a ser dado pelo docente. Deve partilhar o repositório **GitHub** do projecto com o docente, através da *username* `joaojdacosta`. **Obs.:** Não deve incluir os ficheiros de instâncias.

7.1. Prazo de Entrega

Data de entrega (serial): **24 de Março de 2025, até as 17H.**
Data de entrega (serial + OMP): **04 de Abril de 2025, até as 17H.**

- Obs: a avaliação do trabalho é realizada na aula após data de entrega.

Data de entrega (serial + MPI): **16 de Maio de 2025, até às 17H.**

- Obs: a avaliação do trabalho é realizada na aula após data de entrega.

A – Rotina `init_particles()`

```
#define _USE_MATH_DEFINES
#include <math.h>
#define G 6.67408e-11
#define EPSILON2 (0.005*0.005)
#define DELTAT 0.1

unsigned int seed;
void init_r4uni(int input_seed)
{
     seed = input_seed + 987654321;
}
double rnd_uniform01()
{
     int seed_in = seed;
     seed ^= (seed << 13);
     seed ^= (seed >> 17);
     seed ^= (seed << 5);
     return 0.5 + 0.2328306e-09 * (seed_in + (int) seed);
}
double rnd_normal01()
{
     double u1, u2, z, result;
     do {
          u1 = rnd_uniform01();
          u2 = rnd_uniform01();
          z = sqrt(-2 * log(u1)) * cos(2 * M_PI * u2);
          result = 0.5 + 0.15 * z;
     } while (result < 0 || result >= 1);
     return result;
}
void init_particles(long seed, double side, long ncside,
long long n_part, particle_t *par))
{
     double (*rnd01)() = rnd_uniform01;
     long long i;

     if(seed < 0) {
          rnd01 = rnd_normal01;
          seed =-seed;
     }

     init_r4uni(seed);

     for(i = 0; i < n_part; i++) {
          par[i].x = rnd01() * side;
          par[i].y = rnd01() * side;
          par[i].vx = (rnd01()- 0.5) * side / ncside / 5.0;
          par[i].vy = (rnd01()- 0.5) * side / ncside / 5.0;
```

```
            par[i].m = rnd01() * 0.01 * (ncside * ncside) /
n_part / G * EPSILON2;
        }
}
```

Bom trabalho!