N-Body Problem

Fábio A. Bocampagni

[a] *Universidade Federal do Rio de Janeiro,*

Abstract

The N-Body problem, a classical computational physics challenge, involves the simulation of the interactions between N point masses in a gravitational field. Solving this problem efficiently has vast applications in astrophysics, aerospace engineering, and computer graphics, among others. As computational power continues to increase, parallel and concurrent programming techniques become essential for harnessing this power effectively.

This paper explores the use of concurrent programming to address the N-Body problem. We delve into the challenges of parallelizing a problem that exhibits high interdependence and presents a novel solution that leverages concurrent techniques to improve computational efficiency.

Our approach involves breaking down the problem into smaller tasks, utilizing multiple cores, threads, or distributed computing resources. We discuss the trade-offs, synchronization challenges, and load balancing strategies encountered during the concurrent execution of N-Body simulations.

The paper presents a comprehensive study of concurrent programming paradigms, including multithreading, message passing, and GPU acceleration, highlighting the benefits and limitations of each. We discuss methods for mitigating data race conditions, optimizing communication, and adapting algorithms for parallelism.

Additionally, we provide experimental results demonstrating the speedup achieved through concurrent programming on both shared-memory and distributed-memory architectures. We analyze the impact of problem size, hardware configurations, and different concurrent strategies on the performance of N-Body simulations.

Our findings demonstrate that concurrent programming can significantly enhance the efficiency of N-Body simulations, making it possible to handle larger and more complex scenarios. We also discuss the potential challenges and future research directions in the field of parallel and distributed N-Body simulations.

This paper serves as a comprehensive guide for researchers and practitioners interested in harnessing the power of concurrent programming to solve the N-Body problem, shedding light on the path towards more accurate and scalable simulations.

Keywords: n-body problem, Concurrent Programming, Accelerated Programming, Nvidia Cuda

1. Introdução

Na física, o problema dos ”n-corpos” refere-se ao desafio de prever os movimentos individuais de um grupo de objetos celestes influenciados por interações gravitacionais. Esse dilema tem impulsionado a pesquisa científica na busca por compreender as órbitas e movimentos de entidades como o Sol, a Lua, planetas e estrelas visíveis. No século 20, a compreensão da dinâmica de sistemas estelares em aglomerados globulares tornou-se um importante problema de ”n-corpos”.

Ao lidar com o problema de ”n-corpos” no contexto da relatividade geral, a complexidade de encontrar soluções é consideravelmente aumentada devido à inclusão de variáveis adicionais, como distorções no espaço-tempo.

A questão física fundamental pode ser resumida informalmente da seguinte maneira:

Dadas as características orbitais conhecidas e relativamente estáveis (incluindo posição instantânea, velocidade e tempo) de um grupo de corpos celestes, a tarefa é prever as forças que eles exercem uns sobre os outros. Consequentemente, esse conhecimento permite a previsão das trajetórias orbitais reais para todos os momentos futuros.

2. Descrição do problema

O problema dos ”n-corpos”, consiste em calcular a relação entre os N corpos celestes de um sistema. De certa forma, essa relação nos da possibilidade de prever a trajetória dos N corpos celestes e qualquer momento futuro, assim como aceleração, velocidade e posição instantânea. De modo geral, buscamos resolver o seguinte problema:

Dadas as características orbitais dos corpos celestes de um sistema conhecido e relativamente estável, preveja as forças exercidas uns sobres os outros.

Buscaremos, por meio da mecânica clássica, uma fórmula fechada para servir de apoio aos nossos cálculos. Comecemos derivando uma fórmula fechada para o problema, partindo da Lei da Gravitação Universal.

O problema dos ”n-corpos” considera n massas pontuais m_i, $i = 1, 2, \ldots, n$, em um referencial inercial no espaço tridimensional $\mathbb{R}^3$, movendo-se sob a influência da atração gravitacional mútua. Cada massa m_i possui um vetor de posição $\mathbf{q}_i$. A

segunda lei de Newton afirma que a massa multiplicada pela aceleração $m_i \frac{d^2\mathbf{q}_i}{dt^2}$ é igual à soma das forças sobre a massa. A lei da gravidade de Newton estabelece que a força gravitacional sentida pela massa m_i devida a uma única massa m_j é dada por:

$$\mathbf{F}_{ij} = \frac{Gm_i m_j}{\left\|\mathbf{q}_j - \mathbf{q}_i\right\|^2} \cdot \frac{\left(\mathbf{q}_j - \mathbf{q}_i\right)}{\left\|\mathbf{q}_j - \mathbf{q}_i\right\|} = \frac{Gm_i m_j \left(\mathbf{q}_j - \mathbf{q}_i\right)}{\left\|\mathbf{q}_j - \mathbf{q}_i\right\|^3}$$

onde G é a constante gravitacional e $\left\|q_j - q_i\right\|$ é a magnitude da distância entre q_i e q_j (métrica induzida pela norma L^2). Somando todas as massas, obtemos as equações de movimento dos "n-corpos":

$$m_i \frac{d^2\mathbf{q}_i}{dt^2} = \sum_{\substack{j=1 \\ j\neq i}}^{n} \frac{Gm_i m_j \left(\mathbf{q}_j - \mathbf{q}_i\right)}{\left\|\mathbf{q}_j - \mathbf{q}_i\right\|^3} = -\frac{\partial U}{\partial \mathbf{q}_i}$$

onde U é a energia potencial própria:

$$U = -\sum_{1\leq i<j\leq n} \frac{Gm_i m_j}{\left\|\mathbf{q}_j - \mathbf{q}_i\right\|}$$

Definindo o momento como $\mathbf{p}_i = m_i \frac{d\mathbf{q}_i}{dt}$, as equações de movimento de Hamilton para o problema dos "n-corpos" tornam-se:

$$\frac{d\mathbf{q}_i}{dt} = \frac{\partial H}{\partial \mathbf{p}_i} \quad \frac{d\mathbf{p}_i}{dt} = -\frac{\partial H}{\partial \mathbf{q}_i}$$

onde a função Hamiltoniana é:

$$H = T + U$$

e T é a energia cinética:

$$T = \sum_{i=1}^{n} \frac{\|\mathbf{p}_i\|^2}{2m_i}$$

As equações de Hamilton mostram que o problema dos "n-corpos" é um sistema de $6n$ equações diferenciais de primeira ordem, com $6n$ condições iniciais, sendo $3n$ coordenadas iniciais de posição e $3n$ valores iniciais de momento.

Simetrias no problema dos "n-corpos" resultam em integrais de movimento globais que simplificam o problema. A simetria translacional resulta no centro de massa:

$$\mathbf{C} = \frac{\sum_{i=1}^{n} m_i \mathbf{q}_i}{\sum_{i=1}^{n} m_i}$$

movendo-se com velocidade constante, de modo que $\mathbf{C} = \mathbf{L}_0 t + \mathbf{C}_0$, onde $\mathbf{L}_0$ é a velocidade linear e $\mathbf{C}_0$ é a posição inicial. As constantes de movimento $\mathbf{L}_0$ e $\mathbf{C}_0$ representam seis integrais do movimento. A simetria rotacional resulta no momento angular total sendo constante:

$$\mathbf{A} = \sum_{i=1}^{n} \mathbf{q}_i \times \mathbf{p}_i$$

onde $\times$ é o produto vetorial. As três componentes do momento angular total $\mathbf{A}$ fornecem mais três constantes do movimento. A última constante geral do movimento é dada pela conservação de energia H. Portanto, todo problema dos "n-corpos" possui dez integrais de movimento.

Devido ao fato de T e U serem funções homogêneas de grau 2 e -1, respectivamente, as equações de movimento apresentam uma invariância de escala: se $\mathbf{q}_i(t)$ é uma solução, então também o é $\lambda^{-2/3}\mathbf{q}_i(\lambda t)$ para qualquer $\lambda > 0$.

O momento de inércia de um sistema de "n-corpos" é dado por:

$$I = \sum_{i=1}^{n} m_i \mathbf{q}_i \cdot \mathbf{q}_i = \sum_{i=1}^{n} m_i \|\mathbf{q}_i\|^2$$

e o virial é dado por $Q = \frac{1}{2}\frac{dI}{dt}$. Então, a fórmula de Lagrange-Jacobi afirma que:

$$\frac{d^2 I}{dt^2} = 2T - U$$

Para sistemas em equilíbrio dinâmico, a média de longo prazo de $\langle \frac{d^2 I}{dt^2} \rangle$ é zero. Portanto, em média, a energia cinética total é metade da energia potencial total, $\langle T \rangle = \frac{1}{2}\langle U \rangle$, o que é um exemplo do teorema do virial para sistemas gravitacionais. Se M é a massa total e R um tamanho característico do sistema (por exemplo, o raio que contém metade da massa do sistema), então o tempo

3. Dividir para conquistar

A solução sequencial mostra-se naturalmente em uma primeira análise. Para cada par de corpos celestiais iremos calcular a interação de força entre eles. Mais precisamente, deixaremos fixo um dado corpo c_i e o utilizando como referencial, iremos calcular a interação de força entre c_i e todos os outros corpos do sistema.

Como condicionamos o modelo do problema a existência de um ponto fixo como referencial em consonância com o fato de mudarmos somente as características inerentes ao corpo que está fixo, o problema pode também ser modelado de forma concorrente. Naturalmente podemos notar o ganho de performance sem muito esforço. Ora, se um sistema de corpos celestes é resolvido de forma sequencial em uma quantidade finita de tempo, o mesmo sistema será resolvido em uma quantidade menor de tempo se utilizarmos a mesma regra de construção da solução sequencial, porém, calculando T pontos fixos de uma vez, onde T é o número de fluxos de execução. Haja vista que teremos, para o mesmo conjunto de instruções, uma maior quantidade de processamento, ganha-se em performance com a utilização de concorrência.

Uma possível solução concorrente é instanciar N fluxos de execução os quais serão responsáveis por calcular as forças relacionadas aos corpos celestes. Ou seja, a primeira *thread* irá calcular as interações do primeiro corpo, a segunda *thread* do segundo corpo e assim por diante, até chegarmos à última *thread* que calculará a interação do último corpo do sistema.

Como podemos notar, essa solução ingênua não resolve o problema de forma assintótica. Em outras palavras, como N pode crescer de forma indefinida, instanciar mais e mais fluxos de execução não é uma solução possível, apesar de funcionar bem para problemas com quantidades de corpos iguais ou menores que a quantidade de núcleos de processamento.

Um ajuste na solução acima para permitir sua viabilidade é utilizarmos o máximo de *threads* possíveis, porém, ao invés de condicionarmos a existência de um fluxo de execução a um determinado corpo fixo, condicionaremos a existência desse fluxo a existência de um corpo celeste ainda não processado. Ou seja, caso a quantidade de processadores seja igual ou maior que a quantidade de corpos, a solução é igual à mencionada anteriormente. Caso contrário, instanciaremos o máximo de fluxos de execução que conseguirmos e os orquestraremos dividindo o número de corpos de forma igualitária entre os fluxos de execução. Assim, cada fluxo de execução terá assinado um conjunto de corpos celestes os quais precisarão ser resolvidos.

4. Nvidia CUDA

CUDA, originalmente conhecido como "Compute Unified Device Architecture" (Arquitetura de Dispositivo de Computação Unificada), é uma API desenvolvida pela Nvidia para realizar computação paralela e aproveitar o poder das GPUs em tarefas de propósito geral (GPGPU). Essa plataforma é projetada para funcionar principalmente em placas gráficas que suportam a API CUDA, geralmente aquelas com chipsets Nvidia.

A API CUDA inclui um conjunto de instruções conhecido como ISA (Instruction Set Architecture) CUDA e fornece um mecanismo para executar cálculos em paralelo na GPU. Ela permite que os desenvolvedores acessem os diferentes tipos de memória da placa de vídeo e determina como os acessos à memória global, à cache e a organização das threads devem ser configurados. Além disso, os desenvolvedores precisam gerenciar o escalonamento das atividades entre a GPU e a CPU. A plataforma CUDA é compatível com linguagens de programação como C, C++ e Fortran, facilitando a implementação de algoritmos paralelos e cálculos intensivos na GPU.

O uso do CUDA para a resolução do problema dos "n-corpos" é particularmente benéfico em cenários nos quais a simulação envolve um grande número de partículas e cálculos intensivos. No entanto, é importante notar que a otimização e a organização adequada dos dados e dos cálculos são essenciais para obter o máximo desempenho. Utilizaremos um modelo computacional com regras de construção idênticas às mencionadas anteriormente, porém, traduzidas para a utilização do Nvidia Cuda.

5. Balanceamento de carga

Dividir a quantidade de trabalho entre os fluxos de execução é uma tarefa necessária para que um, ou um grupo de fluxos, não tenham sua performance prejudicada devido a uma quantidade de trabalho não distribuída de forma simétrica. Assim, cada *thread* receberá a mesma quantidade de trabalho garantida por uma regra a ser definida antes de sua instanciação.

Chamemos de política a regra pela qual a quantidade de carga será distribuída entre os fluxos de execução. A política a ser criada, de forma especial, precisa se preocupar com problemas clássicos como a não divisibilidade entre o número de corpos e a quantidade de fluxos de execução. Como mencionado anteriormente, a quantidade de corpos pode aumentar indefinidamente. Quanto maior a quantidade de corpos, mais um fluxo de execução irá se prejudicar devido a não divisibilidade da carga de trabalho de forma simétrica.

Consideremos um *buffer* B_N responsável por armazenar os N corpos celestes. Haja vista o caráter concorrente, é necessário que B_N seja acessível por todos os fluxos de execução. Em outras palavras, B_N é uma variável de escopo global. Uma vez que todos os fluxos de execução podem acessar o mesmo endereço, torna-se necessário a implementação de mecanismos de eleição de exclusividade de acesso, exclusão mútua, para evitar problemas clássicos como a condição de corrida, *starvation* e *deadlock*.

A política pode ser definida como $Policy(\sigma, B_N)$, onde σ é a implementação *thread safe* de acesso a variável e B_N o *buffer* de armazenamento dos corpos celestes. Utilizaremos σ toda vez que quisermos acessar B_N e a iremos acessar da seguinte forma:

Cada fluxo de execução irá por meio de σ acessar B_N e pegar um corpo celeste. Ao final, o mesmo fluxo voltará a acessar B_N por meio de σ e dessa vez pegará o corpo celeste indexado pelo mesmo valor do anterior mais um salto (*jump*) j. Assim, podemos redefinir σ como σ_j e a nova política será definida como $Policy(\sigma_j, B_N)$. Naturalmente, a política será definida de forma diferente para cada fluxo de execução, haja vista que j precisa ser diferente entre os fluxos e maior igual a 1 (Não podemos ter $jump = 0$ pois o fluxo de execução nesse caso iria percorrer todos os elementos do *buffer*).

Apesar da sua importância para a fundamentação teórica, a política estará implícita na implementação do código, não possuindo, por exemplo, uma função que a defina a nível de desenvolvimento. Ainda sim, sua existência é condição necessária para a corretude da solução.

6. Avaliação e desempenho

Temos 3 modelos de resolução do mesmo problema: Sequencial, concorrente e acelerado (que, via de regra, também é um modelo concorrente, porém, por questões de identificação, chamemos dessa forma). Iremos validar a corretude desses modelos por meio de formas fechadas como as já mencionadas e também utilizaremos séries de Taylor para dar suporte a resultados obtidos analiticamente.

O desempenho dos 3 modelos será avaliado por métricas estatísticas e gráficos para elucidar tanto o ganho de performance, quanto análise de sensibilidade. Aumentaremos o número de corpos gradativamente nos 3 modelos, experimentando a variação do tempo de execução de cada um.

7. Conclusões

8. Referências

1. Bretscher, Otto. *Linear Algebra with Applications*. 5th ed. Pearson Education, 2013.

2. Bryant, Randal E., and David R. O'Hallaron. *Computer Systems: A Programmer's Perspective*. Third edition. Pearson, 2016.

3. Boyce, William E., and Richard DiPrima. *Elementary Differential Equations and Boundary Value Problems*. 3rd edition. (Referência indireta: "Elementary differential equations and boundary value problems (3rd edition), by Boyce William E. and DiPrima Richard." - *The Mathematical Gazette*).

4. Mor, Harchol Balter. *Introduction to Probability for Computing*.

5. Newton, Isaac, et al. *Newton's Principia: The Mathematical Principles of Natural Philosophy*. First American edition. Carefully revised and Corrected. Franklin Classics, 2018.

6. Nussenzveig, Herch Moyses. *Curso de Física Básica*. 4th ed. rev. Blucher, 2008.

7. Pacheco, Peter S., and Matthew Malensek. *An Introduction to Parallel Programming*. Second edition. Morgan Kaufmann, 2022.

8. Dobrow, Robert P. *Introduction to Stochastic Processes with R*.