Arquiteturas Paralelas e a Simulação dos N-corpos: Uma Análise

Lucas Menduiña Ramos Evangelista, Gabriel Marinho de Souza, Juliana Nascente, Marcelo Zamith

[1]Departamento de Ciência da Computação – Instituto Multidisciplinar
Universidade Federal Rural do Rio do Janeiro (UFRRJ)
Av. Governador Roberto Silveira s/no – Moquetá – Nova Iguaçu – RJ
CEP: 26285-060 – Brazil

julianananascente@gmail.com, {menduina19, mzamith, gmarinho}@ufrrj.br

Abstract. *The N-body problem, widely studied in astrophysics, involves predicting the motion of celestial bodies under the influence of their mutual gravitational interactions. This study proposes an analisys of two variants of the problem, using high-performance computing, leveraging parallelism on both CPUs and GPUs. The implementations utilize efficient load balancing techniques, based on data and task parallelization, with a focus on OpenMP and Nvidia CUDA. Parallelizing the gravitational force calculations and mass distribution results in significant speedup and reduced execution time, empirically validated and presented graphically. In addition to offering an efficient alternative to complex simulations, this work has potential applications in other domains involving large-scale data and intensive computational tasks.*

Resumo. *O problema dos N-corpos, amplamente estudado na astrofísica, envolve a previsão dos movimentos de corpos celestes sob a influência de suas interações gravitacionais. Este estudo propõe uma análise utilizando computação de alto desempenho para duas variantes desse problema, explorando o paralelismo em CPUs e GPUs. As implementações utilizam técnicas de balanceamento de carga eficientes, baseadas na paralelização de dados e tarefas, com destaque para o uso de OpenMP e Nvidia CUDA. A paralelização dos cálculos de forças gravitacionais e da distribuição de massa resulta em ganhos significativos de aceleração e redução no tempo de execução, validados empiricamente e apresentados graficamente. Além de oferecer uma alternativa eficiente a simulações complexas, este trabalho também possui aplicações potenciais em outros domínios que envolvem grandes volumes de dados e processamento intensivo.*

1. Introdução

Nos últimos anos, a computação tem testemunhado uma revolução nos processadores, como previsto por Moore [Moore 1965]. Atualmente, os processadores contam com a implementação de tecnologias como super pipelines e pipelines superescalares, instruções MMX, e multi-núcleos, que viabilizam um ambiente *multithread* real, produzindo uma vazão maior de instruções, mesmo que com um *clock* menor.

Super pipelines e pipelines superescalares são técnicas que viabilizam a execução de múltiplas instruções por ciclo de *clock*. A arquitetura superescalar tem cada estágio

do seu pipeline replicado, o que possibilita que duas ou mais instruções estejam simultaneamente no mesmo estágio do pipeline. Em uma outra abordagem, a arquitetura de super pipeline emprega mais estágios com uma granularidade menor em cada estágio, e, assim, aumentando a vazão das instruções através a sobreposição dos estágios [Patterson and Hennessy 2013].

A tecnologia MMX, introduzida pela Intel em 1996, explora o paralelismo de dados, executando sobre um mesmo conjunto de dados uma instrução - SIMD, o que aumenta a vazão de processamento, especialmente na parte de imagens. A tecnologia multi-núcleos implementou mais de um processador em um único *chip*, o que permitiu o paralelismo real em um mesmo computador e com uma memória compartilhada [Patterson and Hennessy 2013]. Além disso, tal tecnologia tem potencialmente um maior poder computacional sem aumentar a frequência do *clock* e, assim, contornando as limitações físicas atingidas pelo processador Pentium IV.

Outra tecnologia que atualmente equipa computadores atuais é GPU (Unidade de Processamento Gráfico - *Graphics Processing Unit*). Teve sua evolução impulsionada pela industria dos jogos digitais e, em 2008, surge como uma aceleradora no processamento de propósito geral, mostrando um bom desempenho no tratamento de dados massivamente. Desde então, tornou-se hardware necessário para a computação de alto desempenho e tendo grande protagonismo no avanço da computação cientifica e inteligencia artificial, computando em tempo aceitável grandes massas de dados até então nunca processadas [Clua and Zamith 2015].

Dentre os diversos problemas que demandam um grande poder computacional, há na literatura um problema de mecânica celeste conhecido como N-corpos. O problema consiste em definir, para cada instante de tempo, a posição dos corpos em um sistema. Com até dois corpos é possível obter uma solução analítica, nos demais casos, é preciso trabalhar com um algoritmo para chegar a uma solução. O algoritmo utilizado e a quantidade de corpos tornam esse problema interessante para computação de alto desempenho [Gangavarapu et al. 2019].

Assim, os autores propõem neste artigo uma análise do tempo de execução de CPUs e GPUs utilizando, para isso, o algoritmo do problema dos N-corpos. Este trabalho está dividido em Seções, onde a Seção 2 apresenta a metodologia utilizada no desenvolvimento do trabalho; a Seção 3 discute os resultados alcançados; e, por fim a Seção 4 mostra as conclusões e trabalhos futuros.

2. Metodologia

Os experimentos foram separadas, para fins didáticos, em dois grupos: CPU e GPU, que representam os experimentos em CPU e GPU, respectivamente. A Tabela 1 resume os processadores utilizados nos testes de acordo com o grupo.

A implementação foi feita de forma independente pelos autores e envolveu o desenvolvimento de dois códigos, um para ser computado pela CPU e utilizou o OpenMP; e, outro em GPU, escrito em CUDA. Ambos os códigos são implementações do problema dos N-corpos, que é uma questão clássica da mecânica newtoniana que estuda o movimento de n corpos sob a influência de forças gravitacionais mútuas. A dinâmica do movimento dos corpos é descrita por equações diferenciais baseada na segunda lei de

Tabela 1. Processadores utilizados nos testes

Modelo	Memória	Grupo	Outros
-Intel Xeon/E-2226G/3.40GHz -6 núcleos sem HT	-32K L1 -256K L2 -12MB L3 -32GB RAM DDR4	CPU	-Debian 12 -Kernel 6.1.0-25 AMD64
-Intel i7-4790K/4.00GHz -8 núcleos com HT	-32K L1 -256K L2 -8MB L3 -16GB RAM DDR3	CPU	-Ubuntu 24.04 LTS -Kernel 6.8.0-41 generic
-GeForce GTX TITAN X -3.072 núcleos CUDA	-12 GB DDR5	GPU	-Driver: 535.183.01
-GeForce GTX 1660 -1.408 núcleos CUDA	-6 GB DDR5	GPU	-Driver: 535.183.01

Newton, Eq. 1, e da e da lei da gravitação universal de Newton.

$$F_i = m_i \times a_i \tag{1}$$

Onde: F_i é a força resultante sobre o i-ésimo corpo; m_i e a_i são a massa e aceleração do corpo i. A força gravitacional entre dois corpos i e j é dada pela lei da gravitação universal de Newton, conforme Eq.:

$$F_{i,j} = G\frac{m_i m_j (r_j - r_i)}{|r_j - r_i|^3} \tag{2}$$

Onde a constante gravitacional é dada por G; m_i e m_j representam respectivamente as massas dos corpos i e j; os vetores de posição dos corpos i e j são caracterizados respectivamente por r_i e r_j. Considerando as Eq. 1 e 2, é possível obter a equação do movimento através:

$$m_i \times a_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} G\frac{m_i m_j (r_j - r_i)}{|r_j - r_i|^3} \tag{3}$$

Assim, para cada passo de tempo, é definida a nova posição do i-ésimo corpo, considerando a força gravitacional que os outros corpos exercem sobre o corpo i, conforme apresentado pela Eq.3. Portanto, o algoritmo para solução da simulação é de ordem $O(n^2)$, onde n representa a quantidade de corpos no sistema.

3. Resultados

Para cada quantidade de partículas os testes consideraram a média de 3 execuções em cada hardware apresentado na Tabela 1, a fim de minimizar qualquer flutuação do Sistema Operacional. Além disso, enquanto as iterações foram fixadas em 100, a quantidade de corpos no sistema variou de 1000 até 10000, variando de 1000 corpos para cada instância de teste.

A fim de avaliar a influência do número de iterações na execução do algoritmo, a variável fixada foi invertida. Nessa segunda série de experimentos, as mesmas precauções

para minimizar flutuações do Sistema Operacional foram adotadas. Neste caso, a quantidade de partículas foi mantida fixa em 1000, enquanto o número de iterações variou de 100 a 1000, com incrementos de 100 passos a cada instância. Os testes foram aplicados as implementações *multithread* e CUDA.

A Figura 1 apresenta, em escala logarítmica, a comparação dos tempos de execução de cada implementação. A solução utilizando CUDA demonstrou o melhor desempenho, alcançando um speedup inicial de $3,877$ em relação à versão sequencial, com um crescimento contínuo ao longo dos testes. Para o caso de 10000 corpos, o speedup chegou a $40,056$, resultando em um *speedup* médio de $22,018$ durante os experimentos. O speedup médio consiste na média aritmética de todas as instâncias dos testes representadas no eixo das partículas. Dessa maneira é obtido um valor que representa o ganho médio de speedup entre os testes.

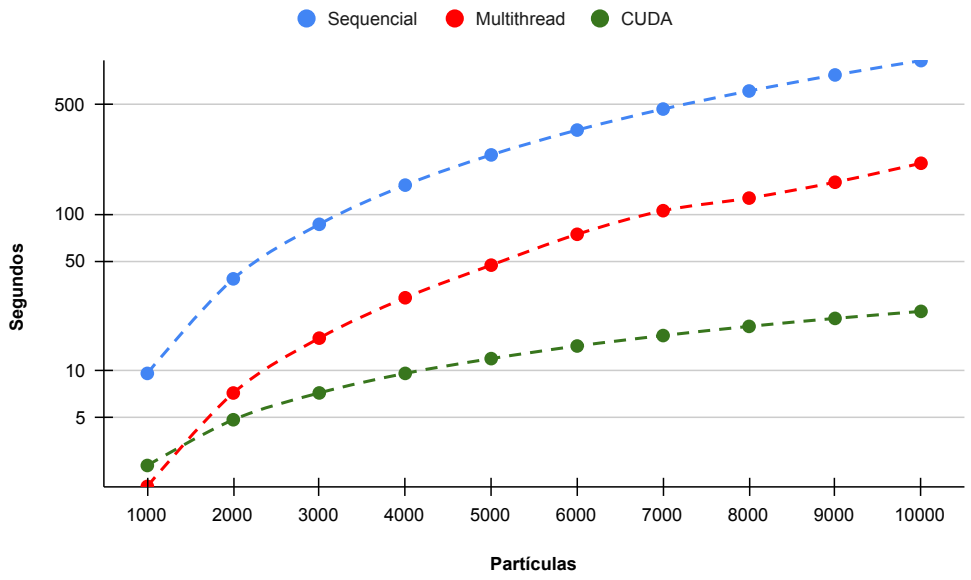

Figura 1. Comparação entre as implementações.

A implementação *multithread*, por sua vez, apresentou um *speedup* médio de $4,957$ no tempo de execução, superando a implementação em CUDA para um número reduzido de partículas. Isso pode ser explicado pelo tempo inicial necessário para copiar os dados da memória principal para a GPU, resultando em uma sobrecarga que só é compensada quando o número de corpos é suficientemente grande, no qual essa sobrecarga se torna desprezível, permitindo que a GPU aproveite plenamente seu poder de processamento paralelo.

Além disso, os resultados também indicam que a variação do número de iterações influencia de maneira linear o tempo de execução do algoritmo. Na Figura 2 é possível observar que essa tendência é mantida para ambos os tipos de processadores. Como mencionado anteriormente neste artigo, a implementação *multithread* apresenta, nesse gráfico, vantagem sobre a implementação em CUDA devido a sobrecarga inicial do carregamento dos dados da memória principal para a GPU, que nesse experimento não foi compensada devido ao baixo número de partículas.

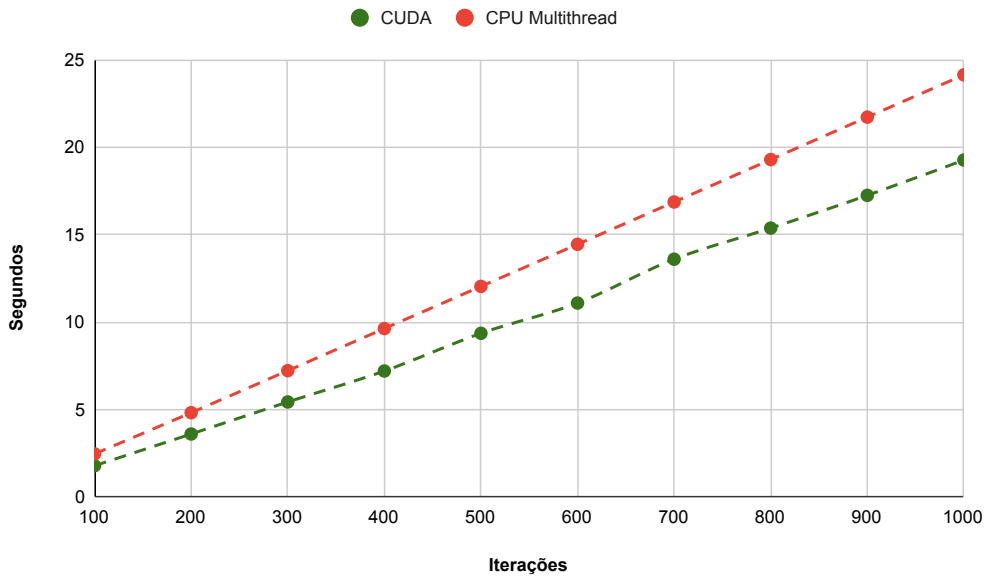

Figura 2. Influência do número de iterações no tempo de execução em segundos.

4. Conclusão

Este trabalho apresenta três implementações do algoritmo para o problema dos N-corpos: uma versão sequencial e duas versões paralelas, utilizando CUDA e OpenMP. Os resultados indicam que a implementação sequencial é significativamente inferior em termos de tempo de execução quando comparada às versões paralelas, tornando-se inadequada para problemas que demandam alta carga de processamento. Em contrapartida, os algoritmos paralelos demonstraram-se eficientes no processamento de grandes volumes de dados. A implementação utilizando OpenMP, ao explorar todas as *threads* disponíveis na CPU, apresentou uma melhora expressiva em relação ao algoritmo sequencial. Por sua vez, a implementação em CUDA mostrou-se ainda mais eficiente para instâncias de grande porte, superando o desempenho da abordagem com OpenMP.

Referências

Clua, E. W. G. and Zamith, M. P. (2015). Programming in cuda for kepler and maxwell architecture. *Revista de Informática Teórica e Aplicada*, 22(2):233–257.

Gangavarapu, T., Pal, H., Prakash, P., Hegde, S., and Geetha, V. (2019). Parallel openmp and cuda implementations of the n-body problem. *In: Misra, S., et al. Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science(), vol 11619. Springer.*

Moore, G. (1965). Moore's law. *Electronics Magazine*, 38(8):114.

Patterson, D. A. and Hennessy, J. L. (2013). *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition.