

Análise de Desempenho da Verificação de Convergência de Sequências Numéricas com OpenMP e CUDA

Hiago Gimenez Vieira, Maria Fernanda Fuchs Lopes,
Fernando C. G. Manso, João Fabrício Filho

¹Universidade Tecnológica Federal do Paraná – Campus Campo Mourão

hiagogimenez@alunos.utfpr.edu.br, mfuchslopes@gmail.com

{fmanso, joaof}@utfpr.edu.br

Abstract. *This work analyzes sequential, OpenMP, and CUDA implementations for computationally verifying the conjecture that every positive natural number converges to the cycle 3, 7, 15. Using inputs of up to 1 billion numbers, OpenMP (8 threads) achieved a speedup of 5.13× and CUDA reached 69.09×, demonstrating the feasibility of parallelism for large-scale verification.*

Resumo. *Este trabalho analisa implementações sequencial, OpenMP e CUDA para verificação computacional da conjectura de convergência ao ciclo 3, 7, 15, segundo regras de transformação baseadas em primalidade, divisibilidade e paridade. Com entradas de até 10^9 , OpenMP (8 threads) atingiu speedup de 5,13× e CUDA alcançou 69,09×, evidenciando a viabilidade do paralelismo em verificações de larga escala.*

1. Introdução

Conjecturas sobre sequências numéricas demandam verificação computacional em larga escala, como ilustra a Conjectura de *Collatz* (Tao 2022), que propõe que, partindo de qualquer número natural positivo e aplicando repetidamente a regra "se par, divida por 2; se ímpar, multiplique por 3 e some 1", a sequência sempre alcança o ciclo $\{1, 2, 4\}$. (Tao 2022) demonstrou, em sentido probabilístico, que quase todas as órbitas da iteração $3n+1$ atingem valores quase limitados, o avanço teórico mais significativo sobre a conjectura na última década, com a qual a conjectura aqui investigada guarda semelhança estrutural.

Do ponto de vista computacional, (Barina 2021) propôs um algoritmo eficiente para verificação de *Collatz* em *GPU*, substituindo tabelas pré-computadas de $O(2^N)$ entradas por tabelas de apenas $O(N)$ elementos, atingindo $2,2 \times 10^{11}$ números por segundo. Para paralelismo em *CPU*, (Dagum and Menon 1998) estabelecem *OpenMP* como padrão da indústria para sistemas de memória compartilhada, com ganhos proporcionais aos núcleos físicos disponíveis. (Gimenez et al. 2025) fundamentam a abordagem metodológica deste trabalho em paralelismo com sistemas *multicore*.

Este trabalho investiga uma conjectura que propõe: todo número natural n pertencente aos naturais não-nulos converge para o ciclo 3, 7, 15 quando submetido iterativamente a cinco regras de transformação aplicadas hierarquicamente:

1. se n é primo: $f(n) = 2n+1$;

2. se $n \equiv 0 \pmod{5}$: $f(n) = \frac{n}{5}$;
3. se $n \equiv 0 \pmod{9}$: $f(n) = n + 1$;
4. se n é par: $f(n) = \frac{n}{2}$;
5. caso contrário: $f(n) = n + s(n)$, na qual $s(n)$ é a soma dos algarismos de n .

O objetivo deste trabalho é verificar computacionalmente a conjectura para os primeiros n números naturais, fornecendo evidências experimentais sobre sua validade e analisando a viabilidade do uso de técnicas de paralelismo para acelerar esse processo de verificação.

A verificação computacional dessa conjectura apresenta desafios significativos: números primos causam crescimento exponencial (dobrando e somando 1), enquanto divisões reduzem os valores. Esse comportamento pode gerar milhares de iterações antes da convergência, demandando técnicas de programação paralela para viabilizar análises em larga escala. Em sistemas *multicore*, *OpenMP* (OpenMP Architecture Review Board 2020) é uma interface de programação paralela baseada em memória compartilhada que permite a criação de aplicações paralelas via diretivas no código-fonte, distribuindo tarefas entre múltiplas *threads*. Em *GPU*, *CUDA* (Nickolls et al. 2008) é uma plataforma que explora o paralelismo massivo de *GPUs*, cujos milhares de núcleos executam múltiplas operações simultaneamente, em contraste com as *CPUs* otimizadas para execução sequencial.

A avaliação experimental testou a conjectura para n de até 1×10^9 , na qual a implementação com OpenMP obteve *speedup* de $5,13 \times$ e CUDA atingiu *speedup* de $69 \times$. A análise considera diferentes tamanhos de entrada (10 a 10^9), métricas de *throughput* (números/segundo) e estatísticas de convergência, revelando padrões sobre distribuição de regras e número médio de passos.

2. Metodologia Experimental

Os experimentos foram conduzidos em um sistema com: *CPU*: Intel Core i7-4790 (4 cores, 8 threads), Sistema Operacional: Linux Ubuntu 22.04.5 LTS, compilador GCC 11.4.0 com as flags `-O3 -fopenmp`, *GPU*: NVIDIA GeForce RTX 3060 e Compilador CUDA NVCC 12.5.82 com flag de otimização `-O3`.

Foram desenvolvidas três versões do algoritmo. A primeira consiste em uma implementação sequencial em linguagem C, utilizada como referência de desempenho. A segunda utiliza *OpenMP* para paralelizar o processamento em *CPU*, distribuindo os valores entre 8 threads via `schedule(dynamic, 1000)`, para mitigar o desbalanceamento causado pela variação no número de iterações por número. A terceira implementação utiliza *CUDA* para explorar o paralelismo massivo da *GPU*, com blocos de 256 threads e lotes de até 10^6 números por iteração do laço externo, devido a limitações de memória da GPU. Cada thread processa um número independente, e os resultados de cada lote são copiados de forma síncrona para o host antes do próximo lote, gerando *overhead* de transferência em entradas grandes. Os experimentos foram realizados com diferentes tamanhos de entrada, variando de 10 até 1 bilhão de números. Cada configuração foi executada 10 vezes, considerando a média dos tempos de execução e o desvio padrão. As métricas avaliadas incluem tempo de execução, *speedup* e *throughput*.¹

¹Códigos nesse repositório: <https://github.com/gimenezhiago/Conjecturas/tree/main/src/Manso>.

3. Resultados e Análise

Os resultados obtidos, mostrados na Figura 1, revelam que o uso de paralelismo com *OpenMP* proporciona ganhos significativos de desempenho. Para 100 mil números, a versão com 8 *threads* reduziu o tempo de 0,09s para 0,02s, alcançando *speedup* de $4,39\times$. Esse ganho se intensifica com entradas maiores: 1 milhão de números apresentou *speedup* de $4,81\times$ (0,38s vs 1,82s sequencial), 10 milhões atingiram $5,06\times$ (8,84s vs 44,79s), e 100 milhões alcançaram *speedup* de $5,13\times$ (225,51s vs 1157,22s). O *throughput* aumentou de 86414 números/segundo no sequencial para 443447 números/segundo com *OpenMP* 8 *threads*, mantendo-se consistente até 1 bilhão de números (*speedup* $5,13\times$).

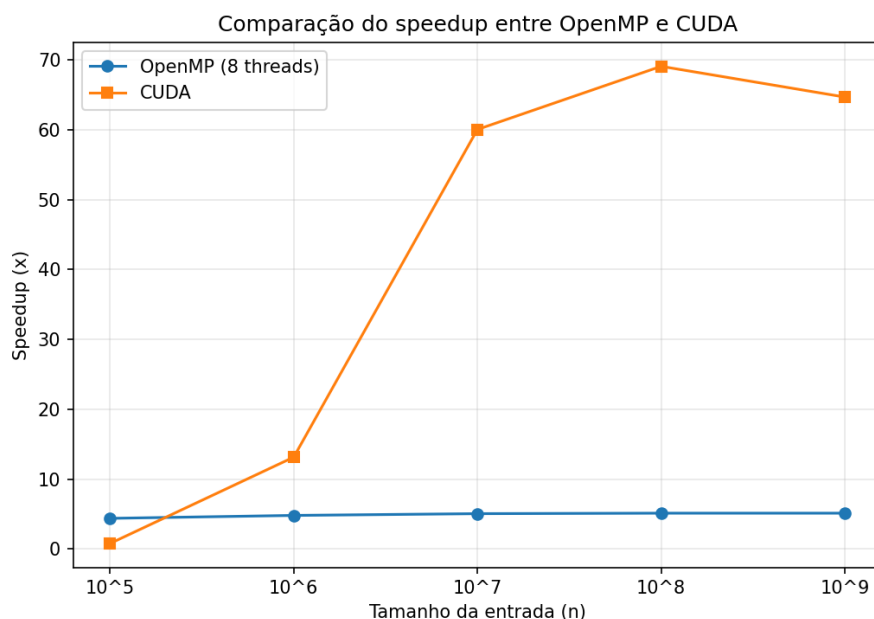


Figura 1. Comparação do speedup entre OpenMP (8 threads) e CUDA.

A implementação *CUDA* apresentou desempenho significativamente superior para grandes volumes de dados. Embora o *overhead* de inicialização (aproximadamente 110ms) limite o desempenho para entradas pequenas, *speedup* de apenas $0,79\times$ para 100 mil números, a *GPU* demonstra elevada eficiência para entradas maiores. O ponto de distinção ocorre em 1 milhão de números (*speedup* $13,15\times$, 0,14s vs 1,83s), superando as implementações *CPU* pela primeira vez.

O desempenho da *GPU* cresce dramaticamente: para 10 milhões, *speedup* atinge $60,05\times$ (0,74s vs 44,79s), $11,8\times$ mais rápido que *OpenMP*. O pico ocorre em 100 milhões de números com *speedup* de $69,09\times$ (16,74s vs 1157,22s), processando 5970782 números/segundo, $69\times$ superior ao sequencial e $13,46\times$ superior ao *OpenMP*. Para 1 bilhão, o *speedup* reduz para $64,69\times$ (475,79s) devido a limitações de memória *GPU* que exigem processamento em lotes, aumentando *overhead* de transferências.

A Tabela 1 analisa o comportamento ao longo de 1 milhão de números e revelou padrões importantes. Das 30.219.675 iterações totais, a regra par foi responsável por 13.964.365 aplicações (46,2%), atuando como principal mecanismo de redução. A regra que soma os algarismos contribuiu com 10902355 aplicações (36,1%), criando um

Tabela 1. Distribuição das regras para diferentes tamanhos de entrada

Regra	100	10.000	100.000	1.000.000	%
Primo	241	22.437	230.887	2.330.556	7,7
Div. por 5	83	16.358	211.125	2.576.755	8,5
Div. por 9	29	4.378	44.339	445.644	1,5
Par	467	92.968	1.165.526	13.964.365	46,2
Contrário	321	68.013	885.604	10.902.355	36,1
Total	1.141	204.154	2.537.481	30.219.675	100

crescimento intermediário controlado. A regra primo, aplicada 2.330.556 vezes (7,7%), é responsável pelo crescimento exponencial que gera sequências longas. As regras de divisibilidade por 5 e 9 contribuíram com 8,5% e 1,5%, respectivamente.

4. Conclusão

Este trabalho investigou técnicas de programação paralela para verificação computacional da conjectura de convergência ao ciclo 3, 7, 15, confirmada para o intervalo [1, 1.000.000.000]. *OpenMP* com 8 *threads* alcançou *speedup* de 5,13×, representando o melhor ponto para *CPUs multicore*, enquanto *CUDA* atingiu 69,09× para 10^8 , estabelecendo-se como solução dominante. A regra par domina com 46,2% das aplicações, enquanto a regra primo (7,7%) gera maiores magnitudes. O throughput de 5970782 números/segundo é inferior ao $2,2 \times 10^{11}$ de (Barina 2021), que emprega tabelas pré-computadas, inviáveis aqui devido à regra de primalidade, verificada em tempo de execução. Como trabalho futuro, destacam-se a prova matemática formal e implementações com precisão arbitrária (*GMP*) em arquiteturas heterogêneas *CPU+GPU* para verificação de números maiores, e diferenciar o tamanho dos lotes na *GPU*.²

Referências

- [Barina 2021] Barina, D. (2021). Convergence verification of the collatz problem. *The Journal of Supercomputing*, 77(3):2681–2688.
- [Dagum and Menon 1998] Dagum, L. and Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55.
- [Gimenez et al. 2025] Gimenez, H., Gonçalves, R., and Filho, J. F. (2025). Desempenho da multiplicação de matrizes em sistemas multicore com pthreads, openmp, blocking, simd e blas. In *Anais da XVI Escola Regional de Alto Desempenho de São Paulo*, pages 38–41, Porto Alegre, RS, Brasil. SBC.
- [Nickolls et al. 2008] Nickolls, J., Buck, I., Garland, M., and Skadron, K. (2008). Scalable parallel programming with cuda. *Queue*, 6:40–53.
- [OpenMP Architecture Review Board 2020] OpenMP Architecture Review Board (2020). *OpenMP Application Program Interface Version 5.1*.
- [Tao 2022] Tao, T. (2022). Almost all orbits of the collatz map attain almost bounded values. *Forum of Mathematics, Pi*, 10:e12.

²Agradecimento ao Professor Fernando Manso e João Fabrício pela orientação e auxílio.