

Desempenho do Crivo de Eratóstenes com Bitset e OpenMP

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 evaluates a segmented Sieve of Eratosthenes using bitset representation and OpenMP parallelism. Two parallel versions (BOOL and BITSET, both with 8 threads) were tested for values from 10^8 to 10^{12} , achieving speedups between $0.93\times$ and $1.07\times$, while the BITSET version reduces memory usage by 71%.*

Resumo. *Este trabalho avalia o Crivo de Eratóstenes segmentado com representação bitset e OpenMP. Duas versões paralelas (BOOL e BITSET, ambas com 8 threads) foram testadas de 10^8 a 10^{12} , obtendo speedup entre $0,93\times$ e $1,07\times$, com a versão BITSET reduzindo o uso de memória em 71%.*

1. Introdução

A enumeração de números primos é um problema clássico em teoria dos números com aplicações diretas em criptografia, testes de segurança e geração de chaves em sistemas como RSA e curvas elípticas (Crandall and Pomerance 2012). O Crivo de Eratóstenes é o algoritmo canônico para essa tarefa, e sua versão segmentada (Sorenson 1998) reduz o tamanho da memória alocada de $O(N)$ para $O(\sqrt{N} + B)$, na qual B é o tamanho do bloco de processamento.

A representação por *BITSET*, na qual cada bit representa a primalidade de um número ímpar, reduz o tráfego de memória em $8\times$ em relação a vetores de *bytes booleans*, favorecendo a localidade de *cache* (Oliveira e Silva et al. 2014). A combinação dessa representação com *OpenMP* (OpenMP 2020) permite explorar os múltiplos núcleos presentes em *CPUs* atuais, sendo ambas as implementações avaliadas neste trabalho paralelizadas com *OpenMP*. Este trabalho investiga o desempenho do crivo de Eratóstenes segmentado nas versões *BOOL* e *BITSET*, ambas paralelas com *OpenMP*, avaliando tempos de execução, *speedup* relativo, tamanho máximo alocado de memória e número de marcações para entradas de até 10^{12} , contribuindo com evidências empíricas sobre o impacto da representação por *bitset* no contexto paralelo.

Crivos paralelos têm sido estudados sob diversas estratégias, como a verificação exaustiva da conjectura de Goldbach até 4×10^{18} (Oliveira e Silva et al. 2014) e bibliotecas como o *primesieve*¹, que combina representação por *bits*, *SIMD* e múltiplas *threads* para alto *throughput*. Este trabalho tem escopo mais restrito: isolar o efeito da representação de dados (*bool* vs. *bitset*) sob uma mesma estratégia de paralelização com *OpenMP*, sem buscar superar o estado da arte em desempenho absoluto.

¹<https://github.com/kimwalisch/primesieve>

2. Fundamentação Teórica

O crivo segmentado divide o intervalo $[2, N]$ em blocos de tamanho B . Para cada bloco, utiliza-se a lista de primos até $\sqrt{N}$ para marcar compostos. Com $B \approx \sqrt{N}$, o bloco cabe inteiramente em *cache L1/L2*, reduzindo falhas de *cache* (Sorenson 1998). A complexidade temporal permanece $O(N \log \log N)$, mas o uso de memória torna-se mais eficiente. A potencialização por números ímpares elimina os múltiplos de 2 explicitamente: apenas os índices i correspondentes a $2i + 1$ são armazenados, reduzindo o espaço em 2× em relação à representação completa.

No processador utilizado, cada núcleo possui L1d de 32 KiB e L2 de 256 KiB. Com bloco de 2^{20} elementos ímpares na representação *bitset*, cada segmento ocupa $2^{20}/8 = 128$ KiB, cabendo inteiramente na L2 de cada núcleo e favorecendo a reutilização de linhas de *cache* durante o peneiramento paralelo. Na representação *BITSET*, cada *byte* armazena o estado de 8 números ímpares consecutivos. O acesso ao *bit* i é feito com $\text{GET}(A, i) = A[i \gg 3] \& (1 \ll (i \& 7))$ e $\text{SET}(A, i): A[i \gg 3] \mid= (1 \ll (i \& 7))$.

A fase segmentada do crivo é paralela, pois cada bloco pode ser processado independentemente, os primos pequenos são pré-computados antes do *loop* paralelo. A diretiva `#pragma omp parallel for schedule(dynamic, 1)` distribui os segmentos entre as *threads*, com cada uma mantendo seu próprio *buffer* local para evitar condições de corrida (OpenMP 2020; Dagum and Menon 1998). Ambas as implementações avaliadas neste trabalho, *BOOL* e *BITSET*, utilizam essa estratégia de paralelização.

3. Metodologia Experimental

Foram desenvolvidas duas implementações em C com aritmética de precisão arbitrária via *GMP* (GNU Multiple Precision Arithmetic Library): uma versão paralela baseada em vetor `bool` (*BOOL*) e uma versão paralela com representação *bitset* (*BITSET*). Ambas utilizam crivo segmentado, processam apenas números ímpares, empregam blocos de 2^{20} elementos e são paralelizadas com *OpenMP*. As implementações foram compiladas com otimização `-O3 -lm -lgmp`, executadas com 8 *threads* e compiladas com `-fopenmp`.

Na versão *BOOL*, cada elemento do bloco é representado por uma posição independente de um vetor `bool` (1 *byte* por número ímpar), com acesso direto por indexação. Já a versão *BITSET* codifica 8 números ímpares por *byte*, exigindo as operações das Equações 1 e 2. Ambas compartilham o mesmo crivo segmentado, bloco de 2^{20} elementos e estratégia de paralelização, diferindo apenas na granularidade de armazenamento.

Os experimentos foram realizados em um processador *Intel Core i7-4790* (4 núcleos físicos, 8 *threads* lógicas, frequência base de 3,60 GHz e *Turbo Boost* de até 4,00 GHz), com hierarquia de *cache L1 de 128 KiB, L2 de 1 MiB e L3 de 8 MiB*, sob *Ubuntu 22.04.5 LTS* e compilador *GCC 11.4.0*. Foram avaliados valores $N = 10^8, 10^9, 10^{10}, 10^{11}$ e 10^{12} . Cada configuração foi executada 10 vezes e as métricas reportadas correspondem à média aritmética. As métricas analisadas foram: tempo de execução (segundos), *speedup* relativo (razão entre tempo médio de *BOOL* e tempo médio de *BITSET*), máximo alocado de memória RSS, número de marcações de compostos e contagem de primos. Ambas produziram contagens de primos idênticas, confirmando a corretude dos resultados.²

²Códigos disponíveis no repositório <https://github.com/gimenezhiago/Conjecturas/tree/main/src/Crivo>.

4. Resultados e Análise

A razão do *speedup* foi definida como BOOL/BITSET por a versão *BOOL* representar a abordagem convencional de armazenamento, servindo como referência para avaliar o impacto da representação compacta. O foco da comparação é o efeito da representação de dados sob uma mesma estratégia de paralelização; tempos da versão sequencial não foram incluídos por não serem objeto direto desta análise, ficando como extensão natural deste trabalho.

A Figura 1 apresenta o *speedup* relativo (BOOL/BITSET) para cada valor de N . Como ambas as implementações são paralelas com 8 *threads*, os valores ficam próximos de $1\times$, variando entre $0,93\times$ e $1,07\times$. Para $N = 10^{10}$ e $N = 10^{12}$, a versão *BITSET* apresenta leve vantagem de tempo, enquanto para os demais valores as diferenças são mínimas e dentro da margem de variação experimental. Esse comportamento é esperado: o ganho de localidade de *cache* do *bitset* é parcialmente compensado pelo custo das operações de manipulação de bits, resultando em desempenho equivalente ao *BOOL* paralelizado.

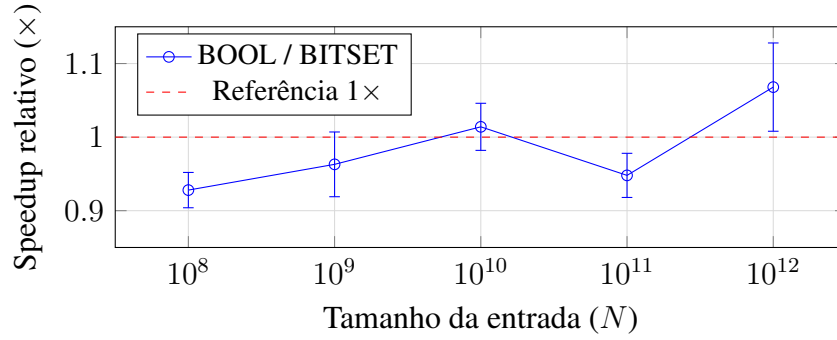


Figura 1. Speedup relativo (BOOL/BITSET) com barras de erro ($\pm\sigma$) para cada valor de N . Ambas as implementações utilizam OpenMP com 8 *threads*.

A Tabela 1 detalha os tempos médios e o *speedup* relativo. Os tempos de execução das duas implementações são muito próximos em todos os casos, com diferença máxima de 7% em $N = 10^{12}$. O desvio padrão menor da versão *BITSET* indica maior estabilidade, provavelmente pela menor pressão sobre a *cache*.

Tabela 1. Tempos médios de execução e *speedup* relativo (BOOL/BITSET), com 8 *threads* em ambas as versões.

N	Tempo BOOL (s)	Tempo BITSET (s)	Speedup ($\times$)
10^8	$0,0552 \pm 0,0014$	$0,05949 \pm 0,00047$	$0,928 \pm 0,024$
10^9	$0,5988 \pm 0,0274$	$0,62191 \pm 0,00109$	$0,963 \pm 0,044$
10^{10}	$6,6425 \pm 0,2116$	$6,55153 \pm 0,00992$	$1,014 \pm 0,032$
10^{11}	$65,951 \pm 2,091$	$69,555 \pm 0,133$	$0,948 \pm 0,030$
10^{12}	$816,84 \pm 46,06$	$765,16 \pm 4,02$	$1,068 \pm 0,060$

A Tabela 2 evidencia o principal diferencial da representação *bitset*: a redução de memória RSS é de aproximadamente 71% em todos os valores de N , chegando a 71,5% para $N = 10^8$ e mantendo-se em 68,0% mesmo para $N = 10^{12}$. Ao contrário da versão sequencial anterior, em que a lista de primos pequenos crescia com N e diluía o ganho

Tabela 2. Tamanho médio de alocação de memória RSS (megabytes, média de 10 rodadas).

N	BOOL (MB)	BITSET (MB)	Redução (%)
10^8	$8,618 \pm 0,034$	$2,454 \pm 0,078$	71,5
10^9	$8,646 \pm 0,041$	$2,479 \pm 0,041$	71,3
10^{10}	$8,664 \pm 0,033$	$2,499 \pm 0,031$	71,2
10^{11}	$8,940 \pm 0,030$	$2,562 \pm 0,039$	71,3
10^{12}	$9,325 \pm 0,038$	$2,988 \pm 0,031$	68,0

relativo, a memória de ambas as versões paralelas permanece praticamente estável com o crescimento de N (variando de 8,6 MB a 9,3 MB para *BOOL* e de 2,5 MB a 3,0 MB para *BITSET*), confirmando o comportamento segmentado com bloco fixo. O desvio padrão da memória RSS foi inferior a 0,08 MB em todos os casos.

5. Conclusão

Este trabalho avaliou o impacto da representação *bitset* no desempenho do Crivo de Eratóstenes segmentado paralelizado com *OpenMP*, comparando duas implementações paralelas com 8 *threads*: *BOOL* e *BITSET*. Os tempos de execução das duas versões mostraram-se equivalentes para entradas de 10^8 a 10^{12} , com *speedup* relativo entre $0,93\times$ e $1,07\times$, indicando que o custo das operações de manipulação de bits compensa o ganho de localidade de *cache* no contexto paralelo.

O principal benefício do *bitset* reside no tamanho utilizado de memória: a versão *BITSET* reduziu o uso de memória RSS em aproximadamente 71% em todos os valores de N avaliados, mantendo esse ganho estável mesmo para $N = 10^{12}$. As duas implementações produziram as mesmas contagens de números primos, confirmando a correteza dos resultados. Dessa forma, conclui-se que a representação *bitset* é especialmente vantajosa quando a eficiência de memória é prioritária, sendo uma alternativa compacta e equivalente em desempenho à representação booleana no contexto paralelo com *OpenMP*. Trabalhos futuros incluem vetorização *SIMD*, diferentes políticas de escalonamento e implementações híbridas *CPU-GPU*.

Referências

- [Crandall and Pomerance 2012] Crandall, R. and Pomerance, C. (2012). *Prime Numbers: A Computational Perspective*. Mathematics and Statistics. Springer New York.
- [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.
- [Oliveira e Silva et al. 2014] Oliveira e Silva, T., Herzog, S., and Pardi, S. (2014). Empirical verification of the even Goldbach conjecture and computation of prime gaps up to $4 \cdot 10^{18}$. *Mathematics of Computation*, 83(288):2033–2060.
- [OpenMP 2020] OpenMP (2020). *OpenMP Application Program Interface Version 5.1*.
- [Sorenson 1998] Sorenson, J. P. (1998). Trading time for space in prime number sieves. In *Proceedings of the 3rd International Algorithmic Number Theory Symposium (ANTS-III)*, volume 1423 of *Lecture Notes in Computer Science*, pages 179–195, Berlin, Heidelberg. Springer.