Implementação Sequencial e Paralela do Método de Gauss-Jacobi Utilizando OpenMP

Eros Henrique Lunardon Andrade[1], Igor de Souza Pinto[1], Victor Henrique Vechi[1]

[1]Campus Pinhais – Instituto Federal do Paraná (IFPR)
Pinhais - PR - Brasil

eroshla.13@gmail.com, igorpinto103@gmail.com, victorvechi@gmail.com

Abstract. *This work presents the sequential and parallel implementation of the Gauss-Jacobi method for solving linear systems. The sequential version was used as the baseline, while the parallel version was developed with the OpenMP library, applying parallelization directives in independent loops. Experiments were conducted on a single machine, using different input sizes for the sequential version and varying the number of threads for the parallel version, with five executions per configuration. Performance metrics such as speedup, efficiency, and percentage gain were computed, in addition to a comparison with the theoretical maximum speedup estimated at 18. The results show significant performance improvements, highlighting the scalability of the method up to a certain point, as well as the limitations imposed by Amdahl's Law.*

Resumo. *Este trabalho apresenta a implementação sequencial e paralela do método de Gauss-Jacobi para resolução de sistemas lineares. A versão sequencial foi utilizada como base de comparação, enquanto a versão paralela foi desenvolvida com a biblioteca OpenMP, explorando diretivas de paralelização em laços independentes. Os experimentos foram realizados em uma mesma máquina, com variação de entradas na versão sequencial e de número de threads na versão paralela, totalizando cinco execuções por configuração. Foram calculadas as métricas de desempenho de speedup, eficiência e ganho percentual, além da comparação com o speedup máximo teórico estimado em 18. Os resultados demonstram ganhos significativos de desempenho, evidenciando a escalabilidade do método até certo ponto, mas também as limitações impostas pela Lei de Amdahl.*

1. Introdução

O projeto teve como proposta a implementação do método de Gauss-Jacobi para resolução de sistemas lineares. Inicialmente, desenvolvemos uma versão sequencial do algoritmo, que serviu como base para análise de desempenho. Em seguida, realizamos a paralelização do código utilizando a biblioteca OpenMP, com o objetivo de comparar a eficiência das abordagens.

As atividades envolveram a medição de tempos de execução em diferentes tamanhos de entrada e variações no número de threads, permitindo calcular métricas como speedup e eficiência. A partir desses resultados, foi possível discutir a escalabilidade do método e avaliar os impactos da paralelização no desempenho do algoritmo.

2. Fundamentação Teórica

O método de Gauss-Jacobi é um algoritmo iterativo para resolução de sistemas lineares da forma Ax=bAx = bAx=b. Ele se baseia na decomposição da matriz em elementos diagonais e não diagonais, atualizando os valores das incógnitas a cada iteração até que a solução convirja dentro de uma tolerância definida. Embora seja simples de implementar, sua eficiência depende de condições como diagonal dominante da matriz.

Em termos de desempenho, a análise de algoritmos paralelos considera métricas como **speedup** e **eficiência**. O speedup é definido pela razão entre o tempo de execução da versão sequencial e o da versão paralela, enquanto a eficiência indica o aproveitamento do paralelismo em relação ao número de processadores utilizados. A Lei de Amdahl fornece um limite teórico para o ganho de desempenho, considerando a fração do código que pode ser paralelizada.

Para explorar o paralelismo, este trabalho utiliza a biblioteca **OpenMP**, que fornece diretivas simples para dividir o processamento em múltiplos threads em arquiteturas de memória compartilhada. O uso de construções como *#pragma omp parallel* `for` permite que laços independentes sejam executados de forma concorrente, aproveitando melhor os recursos de hardware disponíveis.

3. Metodologia

A implementação do trabalho foi dividida em duas etapas: a versão sequencial do método de Gauss-Jacobi e a versão paralela utilizando a biblioteca OpenMP. Ambas foram desenvolvidas em C++, com foco na análise comparativa de desempenho.

3.1 Versão Sequencial

A versão sequencial do algoritmo foi utilizada como referência de desempenho.O método realiza iterações sucessivas até que o erro entre duas soluções consecutivas seja inferior a um limite de tolerância (tol = 1e-8) ou até atingir o número máximo de iterações (max_iter = 1.000.000).

A medição de tempo foi realizada utilizando a biblioteca <chrono>, considerando apenas a região de execução do algoritmo.

Para avaliação de desempenho, foram realizadas **5 execuções com diferentes tamanhos de entrada**, registrando o tempo médio obtido.

3.2 Versão Paralela com OpenMP

A paralelização foi realizada utilizando diretivas do OpenMP em regiões críticas do algoritmo. As principais otimizações foram:

- **Cálculo da soma dos elementos do vetor** com redução aditiva (reduction(+:sum_x));

- **Determinação do erro máximo** com redução por máximo (reduction(max:maxdiff));

- **Atualização do vetor solução** em paralelo com #pragma omp parallel for.

Também foi utilizada uma segunda versão (jacobi_multi_parallel), na qual dois vetores (x e x_new) foram empregados para evitar sobrescrita de valores durante as iterações.

Os testes foram conduzidos variando o número de threads disponíveis (1, 2, 4, 8, até o limite da máquina), permitindo observar o impacto do paralelismo no desempenho.

3.3 Configuração Experimental

Todos os experimentos foram realizados na mesma máquina, conforme orientação do enunciado, com as seguintes especificações de hardware e software:

- **Sistema Operacional:** Windows 11

- **Processador:** Intel Core i7-13650HX (14 núcleos / 20 threads)

- **Memória RAM:** 16 GB DDR5

- **Compilador:** g++ com suporte a OpenMP

O compilador utilizado foi o **g++** com suporte a OpenMP, e os testes foram configurados para diferentes quantidades de threads (1, 2, 4, 8, até o máximo suportado pela máquina).

3.4 Procedimento de Medição

Cada configuração foi executada **5 vezes**, e a média dos tempos foi considerada para análise. Na versão sequencial, variaram-se **os tamanhos de entrada** para observar a evolução do tempo de execução. Na versão paralela, utilizaram-se **as mesmas entradas**, mas variando o número de threads, para avaliar a escalabilidade.

As métricas avaliadas foram:

- **Speedup**:

$$S = \frac{T_{seq}}{T_{par}}$$

- **Eficiência**:

$$E = \frac{S}{p}$$

- **Ganho percentual**:

$$G = \frac{T_{seq} - T_{par}}{T_{seq}} \times 100$$

onde Tseq representa o tempo médio da versão sequencial, Tpar o tempo médio da versão paralela e p o número de threads utilizados.

3.5 Referencial Máximo Teórico

De acordo com a Lei de Amdahl, considerando a fração de código paralelizável no método de Gauss-Jacobi, o **speedup máximo teórico estimado é próximo de 18**. Este valor será utilizado como referência para comparar a escalabilidade real obtida nas execuções experimentais.

4 Resultados e Discussões

4.1 Resultados Experimentais

Os experimentos foram realizados conforme a metodologia descrita, considerando a versão sequencial como base de comparação. A Tabela 1 resume os tempos médios obtidos em cada configuração, bem como os valores de speedup, eficiência e ganho percentual em relação à versão sequencial.

Cálculos Serial:

- Speedup: S = 409,235 ÷ 126,034 ≈ 3,248
- Eficiência: E = 3,248 ÷ 18 ≈ 0,1804
- Ganho percentual: ((409,235 – 126,034) ÷ 409,235) × 100 ≈ 69,2%
- Máximo teórico esperado: próximo de 18

Cálculos Paralelo:

Execução Paralela (9 threads)

- Média dos testes: 189,396 s

Cálculos Paralelo:

- Speedup: S = 409,235 ÷ 189,396 ≈ 2,161
- Eficiência: E = 2,161 ÷ 9 ≈ 0,2401
- Ganho percentual: ((409,235 – 189,396) ÷ 409,235) × 100 ≈ 53,74%
- Máximo teórico esperado: próximo de 9

4.2 Resumo Comparativo

Configuração	Tempo médio (s)	Speedup	Eficiência	Ganho (%)	Máx. Teórico
Sequencial	409,235	1,0	1,0	–	–
18 threads	126,034	3,248	0,1804	69,2%	~18
9 threads	189,396	2,161	0,2401	53,74%	~9

Análise dos resultados

- A paralelização trouxe ganhos significativos: com 18 threads, o tempo caiu de ~409s para ~126s, representando uma redução de 69%.
- O speedup máximo observado foi 3,248, bem abaixo do valor teórico de ~18, o que evidencia limitações práticas como overhead de sincronização, custo das operações de redução e porções não paralelizáveis do código.
- A eficiência caiu conforme o número de threads aumentou: 24% com 9 threads e apenas 18% com 18 threads. Isso era esperado pela Lei de Amdahl, que indica que a escalabilidade se reduz à medida que cresce o número de processadores.
- O melhor equilíbrio entre speedup e eficiência foi obtido com 9 threads, enquanto a configuração com 18 threads apresentou maior redução de tempo absoluto, mas com menor aproveitamento relativo dos recursos.

5 Conclusão

Este trabalho apresentou a implementação sequencial e paralela do método de Gauss-Jacobi utilizando a biblioteca OpenMP. A versão sequencial serviu como referência para comparação, enquanto a versão paralela explorou a divisão de laços independentes em múltiplos threads.

Os resultados mostraram que a paralelização trouxe ganhos significativos de desempenho, com redução de até 69% no tempo de execução ao utilizar 18 threads. No entanto, o speedup máximo observado (3,248) ficou distante do limite teórico de 18, evidenciando os impactos de fatores como overhead de sincronização, operações de redução e a parcela do código que não pode ser paralelizada.

A análise também mostrou que a eficiência cai à medida que aumenta o número de threads, o que está de acordo com as previsões da Lei de Amdahl. O melhor equilíbrio entre tempo de execução e eficiência foi obtido com 9 threads, enquanto o uso de 18 threads resultou em maior ganho absoluto de tempo, mas menor aproveitamento relativo dos recursos.

Como trabalhos futuros, sugere-se investigar estratégias de paralelização híbrida, explorar otimizações de memória e realizar testes em diferentes arquiteturas de hardware, incluindo GPUs, a fim de avaliar o potencial de escalabilidade do método em ambientes mais diversos.

6 Referências

Amdahl, G. M. (1967). *Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities*. AFIPS Conference Proceedings, 30, 483–485.

Saad, Y. (2003). *Iterative Methods for Sparse Linear Systems*. 2. ed. Philadelphia: SIAM.

OpenMP Architecture Review Board. (2021). *OpenMP Application Programming Interface, Version 5.1*. Disponível em: https://www.openmp.org.