Estratégias de Paralelização da Inversão de Matrizes utilizando o Algoritmo de Gauss-Jordan com OpenMP

Herlon A. Santos[1], **Thaiany Santana**[1]

[1]Programa de Pós-Graduação em Ciência da Computação (UEFS)
Av. Transnordestina, s/n, Feira de Santana – BA – Brasil

Abstract. *This technical paper explores parallelization strategies for matrix inversion using the Gauss-Jordan algorithm. Implementations were developed in C, including a sequential version with row and column orientation, and a parallel version using OpenMP. Performance analysis was conducted by varying the matrix size and number of threads. The experiments demonstrate significant execution time gains when using parallelization, validating the effectiveness of the strategy.*

Resumo. *Este artigo técnico explora estratégias de paralelização para a inversão de matrizes utilizando o algoritmo de Gauss-Jordan. Foram desenvolvidas implementações em C, incluindo uma versão sequencial com orientação por linhas e colunas, e uma versão paralela utilizando OpenMP. A análise de desempenho foi realizada variando o tamanho da matriz e o número de threads. Os experimentos demonstram ganhos significativos de tempo de execução com o uso da paralelização, validando a eficácia da estratégia.*

1. Introdução

A inversão de matrizes é uma operação fundamental em diversas áreas da computação científica, aprendizado de máquina e sistemas de controle. Contudo, seu custo computacional é elevado, especialmente para matrizes de grande porte. Neste contexto, a paralelização surge como uma estratégia promissora para acelerar esse processo.

Este trabalho propõe uma abordagem didática para análise de desempenho de diferentes estratégias de inversão utilizando o algoritmo de Gauss-Jordan, comparando versões sequenciais (orientadas a linha e coluna) com uma versão paralela baseada em OpenMP.

2. Fundamentação Teórica

O algoritmo de Gauss-Jordan é um método direto para obtenção da matriz inversa. Ele transforma uma matriz aumentada $[A|I]$ em $[I|A^{-1}]$ através de operações de linha elementares.

Na versão sequencial, a inversão foi implementada com duas variações: orientação por linhas e orientação por colunas. Já na versão paralela, utilizou-se a biblioteca OpenMP para acelerar a normalização e a eliminação de Gauss por meio de múltiplas threads.

3. Metodologia

A metodologia adotada neste trabalho consistiu na implementação, execução e análise comparativa de diferentes versões do algoritmo de inversão de matrizes baseado no método de Gauss-Jordan. As etapas realizadas são descritas a seguir:

3.1. Implementação dos Algoritmos

Foram desenvolvidos dois programas em linguagem C:

- **Versão sequencial** (`im_serial.c`): permite selecionar a orientação da eliminação (por linhas ou por colunas) via argumento de linha de comando.
- **Versão paralela** (`im_parallel.c`): utiliza a biblioteca OpenMP para paralelizar as etapas de normalização da linha pivô e a eliminação de Gauss, adotando orientação por linhas.

Ambas as versões utilizam uma mesma matriz de entrada, gerada aleatoriamente e garantidamente inversível, a fim de garantir a comparabilidade dos experimentos.

- Sistema Operacional: Ubuntu 22.04
- Compilador: `gcc` com suporte a OpenMP (`-fopenmp`)
- CPU: Intel Core i7 com 8 núcleos

3.2. Descrição dos Programas

Os códigos foram implementados em linguagem C. A versão sequencial (`im_serial.c`) permite escolher a orientação da inversão (linha ou coluna) via linha de comando. A versão paralela (`im_parallel.c`) utiliza OpenMP para acelerar as operações do algoritmo Gauss-Jordan com orientação por linha.

As matrizes utilizadas são geradas aleatoriamente e garantidamente inversíveis. A verificação de correção é feita multiplicando a matriz original por sua inversa e comparando com a matriz identidade.

4. Estratégias de Paralelização

Conforme verificado nos resultados, o algoritmo de Gauss-Jordan orientado a linhas demonstrou ser superior em todos os testes quando comparado ao orientado a colunas. Portanto, as estratégias de paralelização utilizadas foram baseadas apenas no algoritmo com orientação a linhas:

- Normalização da linha do pivô
- Eliminação de Gauss
- Inicialização da matriz identidade
- Busca do pivô

4.1. Coleta de Dados

Durante a execução, os programas registram o tempo de processamento e salvam os dados em arquivos CSV:

- `results_row.csv`
- `results_col.csv`
- `results_omp.csv`

Esses dados foram usados para análise gráfica dos resultados.

Tamanho da Matriz	Orientado a linhas	Orientado a colunas
	Tempo de execução (s)	Tempo de execução (s)
10	0,000029	0,000075
100	0,008665	0,020241
500	1,068074	1,642287
1000	7,319517	15,256861
2000	55,554869	179,556352
3000	228,676957	600,686618
4000	480,362635	2911,000828

Figura 1. Comparativo do código serial (orientado a linhas x colunas)

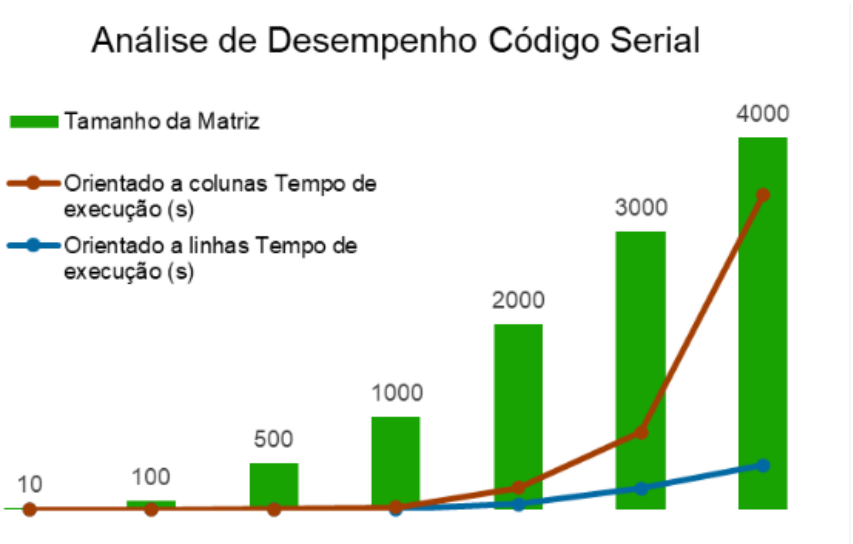


Figura 2. Análise de Desempenho Código Serial

5. Resultados e Discussão

Os testes foram realizados gerando matrizes de diferentes dimensões (10×10, 100×100, 500×500, 1000×1000, 2000×2000, 3000×3000, 4000×4000). A Figura 1 apresenta os resultados obtidos comparando os tempos de execução das matrizes geradas quando orientadas a linhas ou colunas.

Nas Figuras 1 e 2 pode-se observar a relação entre o tempo de execução e o tamanho da matriz do código serializado orientado a linhas e a colunas. À medida que a dimensão da matriz cresce, o tempo de execução aumenta exponencialmente, com comportamento visivelmente mais discrepante na versão orientada a colunas.

5.1. Processo de Paralelização

No processo de paralelização utilizando OpenMP foram implementadas as seguintes estratégias:

- Cada thread recebe um subconjunto das linhas para inicialização (Figura 3)
- Redução para encontrar o pivô com seção crítica (Figura 4)
- Normalização eficiente da linha do pivô (Figura 5)
- Eliminação de Gauss com balanceamento dinâmico (Figura 6)
- Validação paralelizada da matriz inversa (Figura 7)

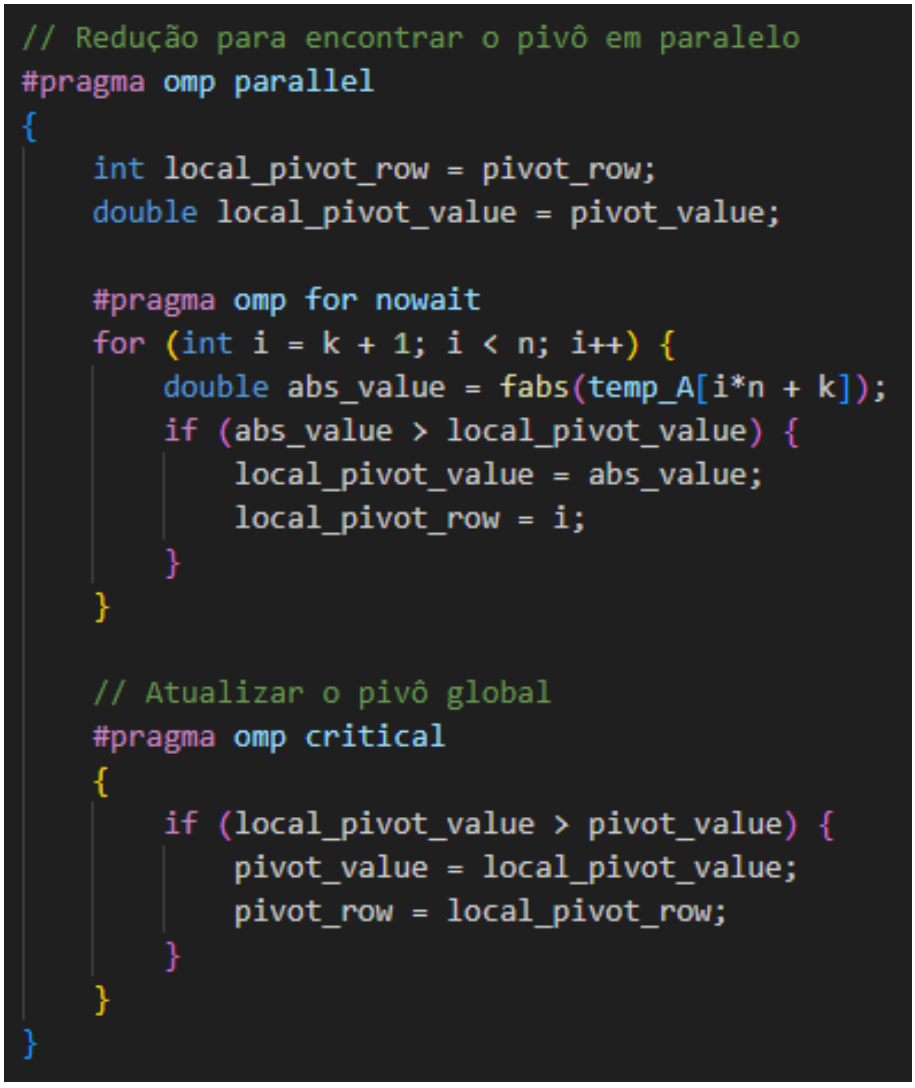
```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        Ainv[i*n + j] = (i == j) ? 1.0 : 0.0;
    }
}
```

Figura 3. Paralelização do Loop de Inicialização da Matriz Identidade

5.2. Análise dos Resultados

Como evidenciado pelas Figuras 8 e 9, observa-se que:

- A versão paralela apresenta aceleração significativa, especialmente em matrizes maiores
- A orientação por linha é ligeiramente mais eficiente que por coluna na versão sequencial
- O desempenho escala linearmente com o número de threads até o limite de 8 threads

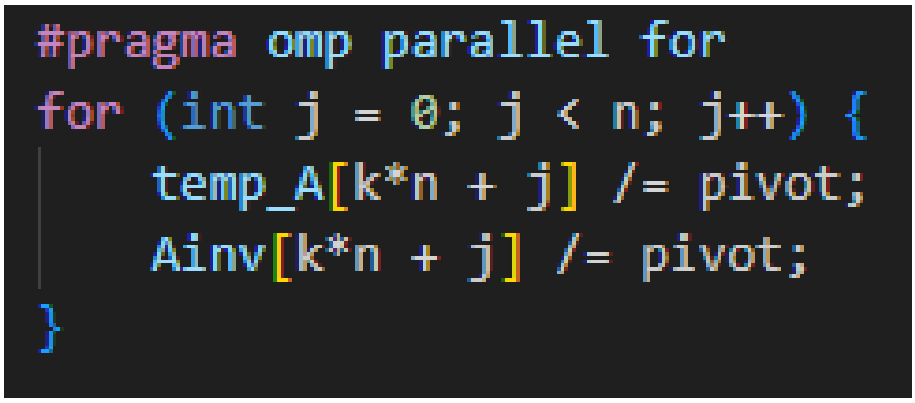


```
// Redução para encontrar o pivô em paralelo
#pragma omp parallel
{
    int local_pivot_row = pivot_row;
    double local_pivot_value = pivot_value;

    #pragma omp for nowait
    for (int i = k + 1; i < n; i++) {
        double abs_value = fabs(temp_A[i*n + k]);
        if (abs_value > local_pivot_value) {
            local_pivot_value = abs_value;
            local_pivot_row = i;
        }
    }

    // Atualizar o pivô global
    #pragma omp critical
    {
        if (local_pivot_value > pivot_value) {
            pivot_value = local_pivot_value;
            pivot_row = local_pivot_row;
        }
    }
}
```

Figura 4. Paralelização da Busca do Pivô com Redução



```
#pragma omp parallel for
for (int j = 0; j < n; j++) {
    temp_A[k*n + j] /= pivot;
    Ainv[k*n + j] /= pivot;
}
```

Figura 5. Paralelização da Normalização da Linha do Pivô

```
#pragma omp parallel for schedule(dynamic)
for (int i = 0; i < n; i++) {
    if (i != k) {
        double factor = temp_A[i*n + k];
        for (int j = 0; j < n; j++) {
            temp_A[i*n + j] -= factor * temp_A[k*n + j];
            Ainv[i*n + j] -= factor * Ainv[k*n + j];
        }
```

Figura 6. Paralelização da Eliminação de Gauss com `schedule(dynamic)`

```
#pragma omp parallel for
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        result[i*n + j] = 0.0;
        for (int k = 0; k < n; k++) {
            result[i*n + j] += A[i*n + k] * Ainv[k*n + j];
        }
    }
}
```

Figura 7. Paralelização da Validação da Matriz Inversa

Tempo médio por tamanho de matriz e número de threads				
Tamanho da Matriz	Número de threads			
	1	2	3	4
10	0.000033	0.000813	0.000286	0.003197
100	0.001053	0.001618	0.010372	0.010792
500	0.191714	0.174658	0.115036	0.115187
1000	2.009424	1.885357	2.071090	2.748241
2000	15.425779	14.172381	14.402178	14.735922
3000	50.131242	47.123314	47.662517	49.289630
4000	113.222778	104.475110	107.448410	108.424802

Figura 8. Métricas de Desempenho por Tamanho de Matriz

Tabela de melhorias por tamanho de matriz					
Tamanho da Matriz	Tempo Serial (s)	Tempo Paralelo (s)	Melhoria (%)	Speedup	Threads Ótimo
10	0.000029	0.000033	-12,64	0.887755	1
100	0.008665	0.001053	87,85	8.228870	1
500	1.068.074	0.115036	89,23	9.284693	3
1000	7.319.517	1.885.357	74,24	3.882298	2
2000	55.554.869	14.172.381	74,49	3.919939	2
3000	228.676.957	47.123.314	79,39	4.852735	2
4000	480.362.635	104.475.110	78,25	4.597867	2

Figura 9. Métricas de Desempenho por Quantidade de Threads

6. Conclusão

Este trabalho demonstrou, de forma prática e didática, como a paralelização com OpenMP pode melhorar substancialmente o desempenho da inversão de matrizes pelo método de Gauss-Jordan. A versão paralela se mostrou eficiente e escalável, especialmente em cenários com matrizes de grandes dimensões.

Como trabalhos futuros, propõe-se:

- Comparar com bibliotecas otimizadas (ex: LAPACK)
- Utilizar técnicas de paralelização em GPU com CUDA

Referências

[1] Golub, Gene H. and Van Loan, Charles F. *Matrix Computations*. 4th ed. Johns Hopkins University Press, 2013.

[2] Mehtre, V. V. and Tiwari, Shubham. *Review on Gauss Jordan Method and it's Applications*. 4th ed. International Journal of Advances in Engineering and Management (IJAEM), 2022, pp: 540-546.

[3] Barney, Blaise. *OpenMP Tutorial*. Lawrence Livermore National Laboratory, 2010. Disponível em: `https://hpc.llnl.gov/tutorials/openMP/`.

[4] Gupta, Ravi and Misra, Rajiv. *Parallel matrix inversion algorithm and its implementation using OpenMP*. In: Proceedings of the ICCCCT, 2010, IEEE, pp. 345–349.