Eliminação Gaussiana por Pivoteamento Parcial com Paralelização OpenMP

José Carlos Zancanaro

Escola do Mar, Ciência e Tecnologia – Universidade do Vale do Itajaí (UNIVALI)
Itajaí – SC – Brasil

jose.zancanaro@edu.univali.br

Resumo. *Este artigo abordará sobre a implementação do método de Eliminação de Gauss para resolução de sistemas lineares. Demonstrando o uso da API OpenMP para paralelizar as linhas da matriz a ser resolvida, diminuindo o tempo de execução do algoritmo.*

1. Introdução

Este artigo aborda a solução de um problema escolhido na disciplina de Arquitetura e Organização de Computadores do curso de bacharelado em Ciência da Computação. O problema geral consiste em implementar um algoritmo e utilizar uma ferramenta de paralelização de processos definidos pelo autor. Além de comparar seu tempo de execução com e sem o uso da paralelização.

O algoritmo escolhido para realização do trabalho foi a implementação do método de Eliminação de Gauss por Pivoteamento Parcial para resolução de sistemas lineares. O problema foi tratado com o uso da linguagem de programação C++ 11 na IDE QtCreator, e a ferramenta de paralelização selecionada para resolução do experimento foi a OpenMP.

2. Definições

Nesta seção são apresentadas a ferramenta de paralelização escolhida, a definição do problema matemático que foi implementado e as especificações do computador usado para os testes realizados.

2.1. OpenMP

Open Multi Processing (*OpenMP*) é uma Interface de Programação de Aplicativo (*API*) de programação paralela em alto nível. Foi definida e é mantida por diversas empresas de hardware e software denominadas OpenMP ARB (Architecture Review Board).

A OpenMP ARB é uma corporação sem fins lucrativos que visa supervisionar, desenvolver e aplicar melhorias para a OpenMP. Esta API pode ser utilizada para programas escritos nas linguagens de programação C/C++ e Fortran.

Constituída por 3 itens fundamentais:

- Diretivas de compilação
- Biblioteca de execução
- Variáveis de ambiente

2.2. Eliminação de Gauss

Eliminação de Gauss, conhecido também como Eliminação Gaussiana ou Método do Escalonamento, é um método utilizado para a resolução de sistemas lineares. Tendo um sistema linear obtido, o método manipulará o sistema a partir de operações elementares, transformando-o em um sistema de resolução mais simples, mantendo exatamente as mesmas soluções que o sistema original.

O método Eliminação de Gauss manipula o sistema, modificando-o de uma matriz original para uma matriz triangular. Uma vez que é obtida a matriz triangular, a solução poderá ser adquirida através da substituição regressiva.

Existem 3 operações elementares que podem ser aplicadas a um sistema linear sem modificar o resultado do mesmo, são elas:

- Substituir duas linhas entre si
- Multiplicar os elementos de uma linha por uma constante não-nula
- Substituir uma linha por ela mesma somada a um múltiplo de outra linha

Adiante, será demonstrado um exemplo da resolução do sistema linear pela Eliminação Gaussiana indicado na Figura 01:

$$\begin{bmatrix} x & + & 3y & + & 4z & = & -5 \\ 3x & + & 2y & + & z & = & 8 \\ 2x & + & 4y & + & 3z & = & 4 \end{bmatrix}$$

Figura 01 - Exemplo de sistema linear

$$\begin{bmatrix} 1 & 3 & 4 & -5 \\ 3 & 2 & 1 & 8 \\ 2 & 4 & 3 & 4 \end{bmatrix}$$

Figura 02 - Matriz definida a partir do sistema linear

Inicialmente, como a Figura 02 exibe, a matriz definitiva do sistema linear é obtida e em seguida, é necessário começar a zerar os valores abaixo da diagonal principal, ou seja, zerar os valores **3**, **2**, e **4**.

Sequencialmente, a partir do primeiro valor da diagonal principal, neste caso **1**, são zerados os valores abaixo dele, **3** e **2**. Para isto, é definido o **1** como pivô do momento e será encontrado os multiplicadores das respectivas linhas abaixo dele.

$$linha\,x = linha\,x + \left(\frac{-n}{pivo}\right) * linha\,do\,pivo$$

Fórmula 01 - Fórmula geral para obter multiplicador

Conforme a Fórmula 01, são obtidos os seguintes multiplicadores:

$$linha\,2 = linha\,2 + \left(\frac{-3}{1}\right) * linha\,1$$

Fórmula 02 - Multiplicador para modificar a linha 2

$$linha\,3 = linha\,3 + \left(\frac{-2}{1}\right) * linha\,1$$

Fórmula 03 - Multiplicador para modificar a linha 3

Executando os multiplicadores na matriz, é obtida a matriz demonstrada na Figura 03:

$$\begin{bmatrix} 1 & 3 & 4 & -5 \\ 0 & -7 & -11 & 23 \\ 0 & -2 & -5 & 14 \end{bmatrix}$$

Figura 03 - Nova matriz gerada a partir das substituições

A partir de agora, atribuímos o nosso pivô como sendo o valor **-7** e a partir do mesmo, encontraremos o multiplicador para assim poder realizar o zeramento do valor abaixo dele, em nossa matriz, o valor -**2**:

$$linha\,3 = linha\,3 + \left(\frac{-(-2)}{(-7)}\right) * linha\,2$$

Fórmula 04 - Multiplicador para modificar a linha 3

$$\begin{bmatrix} 1 & 3 & 4 & -5 \\ 0 & -7 & -11 & 23 \\ 0 & 0 & \frac{-13}{7} & \frac{52}{7} \end{bmatrix}$$

Figura 04 – Matriz gerada após substituições

A partir deste momento, é necessário zerar acima da diagonal principal, para isso, é definido o pivô como **-13 / 7** e são encontrados os multiplicadores para assim atribuir zero aos valores **-**11, **4** e **3**:

$$linha\,1 = linha + \left(\frac{-4}{\frac{-13}{7}}\right) * linha\,3$$

Fórmula 05 - Multiplicador para modificar a linha 1

$$linha\,2 = linha\,2 + \left(\frac{-(-11)}{\frac{-13}{7}}\right) * linha\,3$$

Fórmula 06 - Multiplicador para modificar a linha 2

A Figura 05 relata como a matriz ficou após as devidas substituições:

$$\begin{bmatrix} 1 & 3 & 0 & 11 \\ 0 & -7 & 0 & -21 \\ 0 & 0 & \frac{-13}{7} & \frac{52}{7} \end{bmatrix}$$

Figura 05 - Matriz após substituições

Seguidamente, atribuímos o pivô como **-7** e encontramos o seguinte multiplicador:

$$linha\,1 = linha\,1 + (\frac{-(3)}{-7}) * linha\,2$$

Fórmula 07 - Multiplicador para modificar a linha 1

Após estas diversas substituições, finalmente conseguimos nossa matriz com os valores zerados acima e abaixo da diagonal principal, como é relatado na Figura 06:

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & -7 & 0 & -21 \\ 0 & 0 & \frac{-13}{7} & \frac{52}{7} \end{bmatrix}$$

Figura 06 - Matriz em sua fase final

Finalizando, é necessário pegar os valores da última coluna e dividir com sua respectiva linha, assim obtemos os valores de x, y e z:

$$linha\,1 = x = \frac{2}{1} \quad ; x = 2$$

Fórmula 08 -Divisão da linha 1

$$linha\,2 = y = \frac{-21}{-7} \quad ; y = 3$$

Fórmula 09 - Divisão da linha 2

$$linha\,3 = z = \frac{\frac{52}{7}}{\frac{-13}{7}} \quad ; z = -4$$

Fórmula 10 - Divisão da linha 3

Se substituímos os valores de x, y e z na Figura 01, é garantido que o resultado final é correto, assim temos o sistema linear devidamente resolvido.

Efetuar o método de Eliminação de Gauss com Pivoteamento Parcial é simplesmente executar a Eliminação Gaussiana normalmente, mas no momento de definir o pivô, é escolhido o maior valor em módulo disponível na coluna do pivô. Caso aconteça este requisito, faz-se necessário a troca da linha completa. Ressaltando que o complemento do Pivoteamento Parcial funciona somente no zeramento de valores abaixo da diagonal principal, no momento de zerar acima da diagonal principal, é necessário que a implementação seja a normal, para não haver conflitos de valores.

2.3. Computador de referência

Para realização da proposta deste experimento, o computador definido pelo autor como referência segue as seguintes especificações:

- Sistema Operacional: Linux Manjaro XFCE Edition (18.0)
- CPU: Intel i5 – 6200U, 4 núcleos, 2.80 GHz
- GPU : Intel Skylake GT2 [HD Graphics 520]
- Memória RAM: 4 GB
- Versão do Kernel Linux: 4.19.4 – 1

3. Algoritmo

Para realização do experimento do uso de threads, foi implementado o algoritmo de Eliminação Gaussiana, que resolverá matrizes independentemente de seu tamanho. A fim de utilizar da OpenMP na IDE escolhida, foi preciso acrescentar duas flags no arquivo **.pro**:

```
QMAKE_CXXFLAGS+= -fopenmp
QMAKE_LFLAGS +=  -fopenmp
```

Figura 07 - Flags necessárias

A implementação da Eliminação de Gauss pode ser dividida em quatro partes principais:

```
// Encontrar o maior pivô disponível (em módulo).
for (size_t pivo = 0; pivo < matriz[0].size() - 2; pivo++) {
    size_t maiorPivo = pivo;
    for (size_t i = pivo + 1; i < matriz.size(); i++) {
        if (fabs(matriz[i][pivo]) > fabs(matriz[maiorPivo][pivo])) {
            maiorPivo = i;
        }
    }
    if (pivo != maiorPivo) {
        matriz[pivo].swap(matriz[maiorPivo]);
    }
```

Figura 08 - Primeira parte principal do código

A primeira parte como é indicado na Figura 08, é a responsável por encontrar o pivô que será responsável pelo zeramento dos valores abaixo dele, para verificar qual atende ao requisito, foi comparado todos os valores da coluna indicada e foi encontrado o maior valor em módulo.

```
        // Zerar abaixo da diagonal principal.
        #pragma omp parallel
        {
            #pragma omp for
            for (size_t linha = pivo + 1; linha < matriz.size(); linha++) {
                double fator = matriz[linha][pivo] / matriz[pivo][pivo];
                for (size_t coluna = 0; coluna < matriz[0].size(); coluna++) {
                    matriz[linha][coluna] -= (fator) * (matriz[pivo][coluna]);
                }
            }
        }
    }
```

Figura 09 - Segunda parte principal do código

A Figura 09 relata o momento que será zerado os valores abaixo da diagonal principal, a partir do pivô definido anteriormente. Nesta parte do código foram implementadas as diretivas de paralelização da OpenMP, responsabilizando cada thread com uma linha da matriz.

```
    // Zerar acima da diagonal principal.
    for (size_t pivo = matriz[0].size() - 2; pivo > 0; pivo--) {
        #pragma omp parallel
        {
            #pragma omp for
            for (size_t linha = 0; linha < pivo; linha++) {
                double fator = matriz[linha][pivo] / matriz[pivo][pivo];
                for (size_t coluna = 0; coluna < matriz[0].size(); coluna++) {
                    matriz[linha][coluna] -= (fator) * (matriz[pivo][coluna]);
                }
            }
        }
    }
```

Figura 10 - Terceira parte principal do código

Para zeramento acima da diagonal principal, também foram utilizadas as diretivas de paralelização, como demonstra a Figura 10. Frisando que, neste momento, o pivô a ser definido não deve atender ao requisito de maior valor em módulo, é ideal seguir com a implementação normal.

Finalizando a implementação, faz-se necessário a divisão da coluna de resultados com a sua respectiva linha da diagonal principal, tendo assim, a seguinte parte relatada na Figura 11:

```
            // Finalizar eliminação.
            std::vector<double> respostas;
            for (size_t i = 0; i < matriz.size(); i++) {
                matriz[i][matriz[i].size() - 1] /= matriz[i][i];
                matriz[i][i] /= matriz[i][i];
                respostas.push_back(matriz[i][matriz[i].size() - 1]);
            }
```

Figura 11 - Quarta parte principal do código

Com a finalidade de determinar o tempo de execução, foi escolhida a biblioteca **chrono**, adicionando duas variáveis que definem o tempo inicial e final, e subtraindo-as para obter o valor em segundos de execução.

```
auto inicio = std::chrono::high_resolution_clock::now();
```

Figura 12 - Variável com tempo inicial de execução

```
auto fim = std::chrono::high_resolution_clock::now();
```

Figura 13 – Variável com tempo final de execução

```
std::chrono::duration<double> tempo = fim - inicio;
std::cout << tempo.count() << "s" << std::endl;
```

Figura 14 - Variável com tempo total de execução

4. Comparativo

Nesta seção, são apresentados os testes obtidos durante a execução da implementação do experimento. Para realização do comparativo, foram utilizadas funções auxiliares para criação de 3 matrizes definidas pelos tamanhos 512, 1024 e 2048. Para cada matriz, foram executadas três testes e extraída uma média. Os testes realizados utilizaram do computador de referência citado na seção 2.3.

A tabela 01 apresenta os dados retirados sem a paralelização com OpenMP, seguindo da tabela 02 que exibe os resultados com a paralelização.

Tamanho	Teste 1	Teste 2	Teste 3	Média (em segundos)
512	2,03692	2,04632	2,03474	2,0
1024	16,1342	16,1445	16,1421	16,1
2048	128,532	128,577	128,628	128,6

Tabela 01 - Dados obtidos sem a paralelização

Gráfico 01 – Sem paralelização OpenMP

Tamanho	Teste 1	Teste 2	Teste 3	Média (em segundos)
512	1,08325	1,10866	1,09139	1,1
1024	8,64378	8,71956	8,62326	8,7
2048	68,1344	68,1042	68,1343	68,1

Tabela 02 - Dados obtidos com paralelização OpenMP

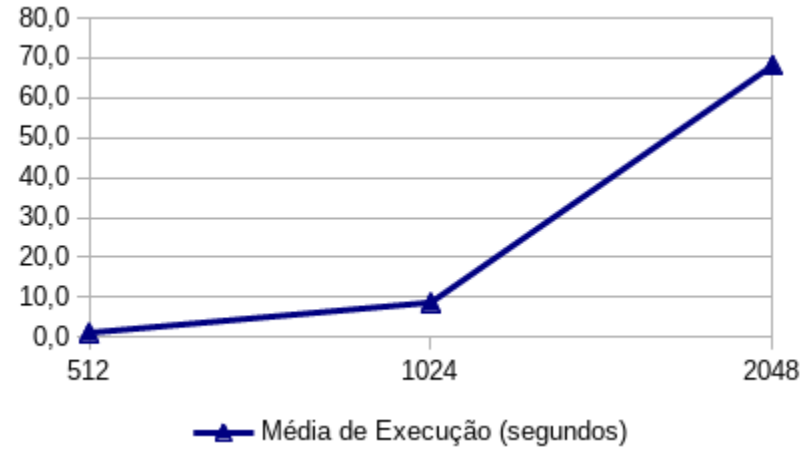

Gráfico 02 - Com paralelização OpenMP

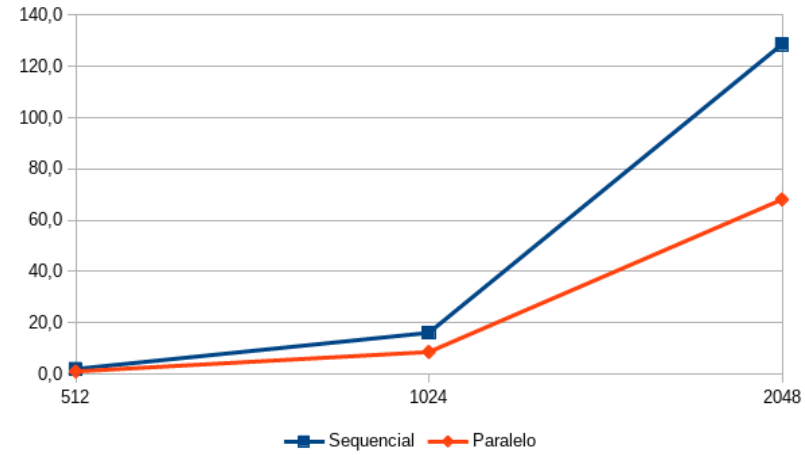

Gráfico 03 – Relação de tempo

Como era de se esperar, utilizando a paralelização, a média de tempo de execução obtida é superior, podendo-se notar que no caso do experimento realizado, a superioridade chega a ser relativamente 2 vezes melhor.

5. Conclusão

Com base nos dados e representações gráficas efetuadas, pode-se afirmar que o uso da paralelização traz benefícios em questão de tempo de execução nas implementações de algoritmos.

Mesmo que outras ferramentas de paralelização possam ser complexas e apresentam uma sintaxe diferente do normal, sempre que possível, um programador deve utilizar da paralelização. Sejam problemas simples ou problemas mais complexos, o uso correto das threads enriquecerá o algoritmo programado.

Para a realização do experimento proposto pela disciplina, o uso da API OpenMP com finalidade no uso de paralelização, mostrou-se extremamente eficiente no algoritmo implementado. Diminuindo drasticamente o tempo de execução para matrizes de tamanhos diferentes.

6. Referências

"Sobre OpenMP", <https://www.openmp.org/about/about-us/>, acesso: 28 de novembro de 2018.

"Introdução ao OpenMP", <https://www.cenapad.unicamp.br/servicos/treinamentos/apostilas/apostila_openmp.pdf> , acesso: 28 de novembro de 2018.

"Eliminação Gaussiana", <https://www.ufrgs.br/reamat/CalculoNumerico/livro-sci/sdsl-eliminacao_gaussiana.html>, acesso: 28 de novembro de 2018.