UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO

Perfilagem do algoritmo de Decomposição LU

Discentes: Flávio Henrique Lopes Barbosa**,** Jaysa Keylla Siqueira Barbosa, José Augusto Agripino de Oliveira.

1. Introdução

Um dos motivos para introduzir o algoritmo de decomposição LU é que ele fornece uma maneira eficiente de calcular a matriz inversa, a qual tem muitas aplicações na engenharia; ele também fornece um meio de avaliar o condicionamento do sistema.

Em álgebra linear, a decomposição LU (onde LU vem do inglês lower upper) é uma forma de fatoração de uma matriz não singular como o produto de uma matriz triangular inferior (por isso o lower) e uma matriz triangular superior (que é o upper). Às vezes se deve multiplicar a matriz a ser decomposta por uma matriz de permutação. Esta decomposição se usa em análise numérica para resolver sistemas de equações (mais eficientemente) ou encontrar as matrizes inversas.

Foi analisado o desempenho do algoritmo de decomposição LU, na forma sequencial. Na implementação do código, são usados vetores alocados dinamicamente e que são tratados como matrizes, sendo as "n" primeiras posições a primeira linha, onde "n" é o número de colunas, de "$n + 1$" a "$2 * n$" é a segunda linha, e assim sucessivamente.

Foi feita a perfilagem do código, que é a ação de traçar um perfil. No nosso caso é descobrir quanto tempo o código como um todo leva para executar e o tempo gasto em cada método ou função, além disso, examinar o que está causando isso.

2. Metodologia

Para fazer o perfilamento utilizamos a ferramenta GProf (GNU profiling), que está inserida no GCC(GNU Compiler Collection). O GProf, serve para medir o tempo gasto pelas funções de um algoritmo, como também mostra quantas vezes uma função foi chamada.

Perfilagem, ou profiling, ajuda a investigação de onde o programa está gastando mais tempo de processamento. Assim permitindo que o seu desempenho seja melhorado, reescrevendo as funções que estão sendo mais custosas. Podemos também verificar as funções que estão sendo mais requisitadas .

Para executar o Gprof, primeiro é preciso compilar e linkar o programa que se quer analisar, com a opção profiling habilitada. Depois, é só executar o programa e em seguida o GProf. Segue um exemplo abaixo:

1. *$ gcc –pg -o decomposicaoLU.out decomposicaoLU.c*
2. *$ ./decomposicaoLU.out*
3. *$ gprof ./decomposicaoLU.out gmon.out > decomposicaoLU.txt*

A saída gerada pelo GProf é dividida em *Flat Profile* e *Call Graph.* O *Flat Profile* nos mostra quanto tempo o programa gastou em cada função, como também quantas vezes a função foi chamada. Já o *Call Graph* enumera para cada função quais funções a chamaram, como quais outras funções ela chamou e quantas vezes.

Para que pudéssemos analisar de maneira variada, para sabermos o comportamento do algoritmo de decomposição LU com diferentes tamanhos de matrizes de entrada, utilizamos diferentes matrizes com ordens variadas de 1000, 2000, 3500, 5500 e 7000. Sendo os valores dessas matrizes gerados automaticamente ao utilizar a função rand() presente em C.

3. Resultados

3.1 Saída do programa para matriz quadrada de ordem 1000

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.00      1.84     1.84        1     1.84     1.84  gauss
  0.00      1.84     0.00        2     0.00     0.00  preencheMatriz
```

Call graph:

```
granularity: each sample hit covers 4 byte(s) for 0.54% of 1.84 seconds

index % time    self  children    called     name
                1.84    0.00       1/1          main [2]
[1]    100.0    1.84    0.00       1          gauss [1]
                0.00    0.00       1/2          preencheMatriz [3]
-----------------------------------------------
                                               <spontaneous>
[2]    100.0    0.00    1.84                 main [2]
                1.84    0.00       1/1          gauss [1]
                0.00    0.00       1/2          preencheMatriz [3]
-----------------------------------------------
                0.00    0.00       1/2          gauss [1]
                0.00    0.00       1/2          main [2]
[3]      0.0    0.00    0.00       2          preencheMatriz [3]
-----------------------------------------------
```

3.2 Saída do programa para matriz quadrada de ordem 2000

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.79     14.35    14.35        1    14.35    14.37  gauss
  0.21     14.38     0.03        2     0.01     0.01  preencheMatriz
```

Call graph:

```
granularity: each sample hit covers 4 byte(s) for 0.07% of 14.38 seconds

index % time    self  children    called     name
                                               <spontaneous>
[1]    100.0    0.00   14.38                 main [1]
               14.35    0.01       1/1          gauss [2]
                0.01    0.00       1/2          preencheMatriz [3]
-----------------------------------------------
               14.35    0.01       1/1          main [1]
[2]     99.9   14.35    0.01       1          gauss [2]
                0.01    0.00       1/2          preencheMatriz [3]
-----------------------------------------------
                0.01    0.00       1/2          gauss [2]
                0.01    0.00       1/2          main [1]
[3]      0.2    0.03    0.00       2          preencheMatriz [3]
-----------------------------------------------
```

3.3 Saída do programa para matriz quadrada de ordem 3500

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.88     69.37    69.37        1    69.37    69.41  gauss
  0.12     69.45     0.08        2     0.04     0.04  preencheMatriz
```

Call graph:

```
granularity: each sample hit covers 4 byte(s) for 0.01% of 69.45 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00   69.45                 main [1]
                69.37    0.04       1/1           gauss [2]
                 0.04    0.00       1/2           preencheMatriz [3]
-----------------------------------------------
                69.37    0.04       1/1           main [1]
[2]     99.9   69.37    0.04       1         gauss [2]
                 0.04    0.00       1/2           preencheMatriz [3]
-----------------------------------------------
                 0.04    0.00       1/2           gauss [2]
                 0.04    0.00       1/2           main [1]
[3]      0.1    0.08    0.00       2         preencheMatriz [3]
-----------------------------------------------
```

3.4 Saída do programa para matriz quadrada de ordem 5500

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.96    294.44   294.44        1   294.44   294.50  gauss
  0.04    294.56     0.12        2     0.06     0.06  preencheMatriz
  0.00    294.57     0.01                             _init
```

Call graph:

```
granularity: each sample hit covers 4 byte(s) for 0.00% of 294.57 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00  294.56                 main [1]
                294.44    0.06       1/1           gauss [2]
                  0.06    0.00       1/2           preencheMatriz [3]
-----------------------------------------------
                294.44    0.06       1/1           main [1]
[2]    100.0  294.44    0.06       1         gauss [2]
                  0.06    0.00       1/2           preencheMatriz [3]
-----------------------------------------------
                  0.06    0.00       1/2           gauss [2]
                  0.06    0.00       1/2           main [1]
[3]      0.0    0.12    0.00       2         preencheMatriz [3]
-----------------------------------------------
                                                 <spontaneous>
[4]      0.0    0.01    0.00                 _init [4]
-----------------------------------------------
```

3.5 Saída do programa para matriz quadrada de ordem 7000

Flat profile:

```
Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
 99.95    569.65   569.65        1   569.65   569.77  gauss
  0.04    569.89     0.24        2     0.12     0.12  preencheMatriz
  0.00    569.91     0.02                             _init
```

Call graph:

```
granularity: each sample hit covers 4 byte(s) for 0.00% of 569.91 seconds

index % time    self  children    called     name
                                                 <spontaneous>
[1]    100.0    0.00  569.89                 main [1]
              569.65    0.12       1/1           gauss [2]
                0.12    0.00       1/2           preencheMatriz [3]
-----------------------------------------------
              569.65    0.12       1/1           main [1]
[2]    100.0  569.65    0.12       1         gauss [2]
                0.12    0.00       1/2           preencheMatriz [3]
-----------------------------------------------
                0.12    0.00       1/2           gauss [2]
                0.12    0.00       1/2           main [1]
[3]      0.0    0.24    0.00       2         preencheMatriz [3]
-----------------------------------------------
                                                 <spontaneous>
[4]      0.0    0.02    0.00                 _init [4]
-----------------------------------------------
```

Ao compararmos os tempos de execução do código para as diferentes entradas, é possível notar que o tempo de execução cresce de maneira muito acentuada. Isso ocorre pelo crescimento ter ordem quadrática, justamente por ser uma matriz de duas dimensões.

Além disso, é evidente o tempo extremamente longo gasto na função “gauss” - próximo de 100% do tempo em todos os testes -, por ser onde são criadas as matrizes LU, ou seja, ocorre iterações sucessivas e o cálculo de cada componente das matrizes resultante.

Ademais, com base nos resultados, é possível notar que a entrada da matriz quadrada de ordem 7000 faz com que o algoritmo leve aproximadamente 9 minutos e 30 segundos para que seja finalizada a execução, que é o tempo ideal para que o algoritmo seja analisado e paralelizado.

4.Conclusões

A partir do teste de perfilagem, pudemos observar e comprovar empiricamente através das simulações com várias amostras, que o maior gargalo do algoritmo de Decomposição LU é o método “gauss”. Tal análise baseou-se na variável tempo de execução de cada método específico da aplicação.