Trabalho Prático 0
Operações com matrizes alocadas dinamicamente

Vitor Emanuel Ferreira Vital
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

vitorvital@dcc.ufmg.br

1. Introdução

Neste trabalho prático foi posto como objetivo desenvolver um programa que realiza operações sobre matrizes alocadas dinamicamente, sendo elas: soma, multiplicação e transposição. A partir do programa foi necessário avaliar o desempenho, custo computacional e o padrão de acesso à memória, transcrevendo sobre os resultados obtidos no processo. Além disso, conceitos sobre tipos abstratos de dados e sua robustez, abstração e desempenho, são temas do trabalho.

2. Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++ da GNU Compiler Collection. Foi utilizado o sistema operacional Linux, a partir de um dual boot no notebook.

2.1. Tipos Abstratos de Dados

A implementação do programa foi baseada em Tipos Abstratos de Dados (TADs). Sua preferência se deu pela facilidade de divisão dos problemas, pois separa a especificação e implementação. Portanto, é possível a especificação de um conjunto de dados, no caso as matrizes, e operações sobre elas.

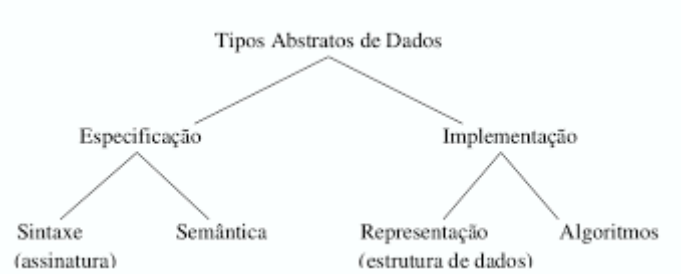


Figura 1- Tipos abstratos de dados

As funções e dados das matrizes foram declaradas no arquivo cabeçalho, nesse caso, arquivo mat.hpp. As principais funções utilizadas, desse cenário em específico, são as de operações de matrizes.

2.2. Manipulando matrizes

Por tratar-se de operações em matrizes, foram criadas diversas funções para obtê-las e manipulá-las, tendo como resultado as matrizes desejadas. Dessa forma, conseguindo lidar com matrizes de diversas dimensões e arquivando seus resultados.

2.2.1. Criação de matrizes

As matrizes que serão tratadas no programa devem ser inicialmente criadas. Por se tratar de matrizes alocadas dinamicamente, é necessário alocar os espaços de memória de acordo com as dimensões passadas no arquivo. Todos os dados das matrizes são provenientes de arquivos .txt passados durante a inicialização do executável.

Para a criação de matrizes em específico é necessário passar a referência para uma struct do tipo matriz, além das dimensões desejadas e seu respectivo ID para futuras verificações.

Com tais dados, verificam-se se as dimensões são válidas para as matrizes, devendo possuir dimensões maiores que zero. Dado a corretude dos dados passados, são alocadas N linhas, respectivas as quantidade passada, e para cada linha são alocadas M colunas, respectivas a quantidade passada. Dessa forma, há a inicialização de um ponteiro para ponteiros que possui N ponteiros apontando, cada um deles, para M ponteiros para doubles. Com isso, criamos o espaço para inicialização dos valores da matriz. Na figura 2 podemos visualizar como ficaria essa alocação.

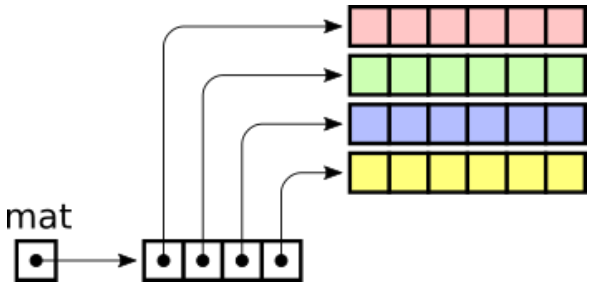


Figura 2 - Alocação de matrizes

2.2.2. Inicialização de matrizes

Nessa etapa, com posse de uma matriz criada, precisamos inicializar os valores conforme submetido na execução do programa. Nessa parte, é necessário pegar os dados dos arquivos .txt passados, tais arquivos possuem as dimensões das matrizes e seus valores.

Após sua abertura para leitura, é necessário pegar todos os valores internos do arquivo passando-os para sua posição da matriz. É necessário verificar se as dimensões correspondem à quantidade passada e se tais valores condizem com o tipo double. Dessa forma, obtemos uma matriz com seus respectivos valores, pronta para realização de operações.

2.2.3. Acesso à matrizes

O acesso às matrizes nos auxiliam na verificação da localidade de referência, visto que ao acessá-las damos ao programa tal informação que é posteriormente mostrada graficamente, permitindo a visualização dos acessos em memória realizados durante todas as operações.

Nessa parte, utilizamos de aninhamento de estruturas de repetição [for e for] para acessar sequencialmente cada elemento da matriz.

2.2.4. Impressão de matrizes

Conforme solicitado, toda matriz resultante deve ser armazenada em um arquivo de saída .txt, com as mesmas configurações das matrizes de entradas. Dessa forma, na primeira linha encontram-se as dimensões da matriz e nas linhas seguintes os valores respectivos da matriz resultante. Portanto, em uma operação de soma, por exemplo, a matriz resultante será igual a soma de elemento a elemento de $A_{\text{MxN}} + B_{\text{MxN}}$.

Portanto, para realização correta dessa função devemos passar a matriz resultante da operação realizada e o nome do arquivo de saída no formato .txt. Dessa forma, todo conteúdo da matriz é lido e passado elemento a elemento ao arquivo, respeitando as dimensões da matriz.

2.2.5. Soma de matrizes

Na operação de soma de matrizes são passadas 3 matrizes, duas inicializadas com valores passados por arquivo, conforme visto na inicialização de matrizes e uma cujos valores são nulos. A última matriz, por ser a resultante da soma das duas matrizes, deve possuir as mesmas dimensões, assim como as matrizes que serão somadas, já que a soma de matrizes demanda que todas as duas tenham as mesmas dimensões.

Nessa etapa, passamos por todos NxM elementos das duas matrizes somamos os respectivos valores e jogamos o resultando na matriz resultante. Dessa forma, possuímos a matriz que desejamos, cujos valores são a soma de A e B .

2.2.6. Multiplicação de matrizes

Na operação de multiplicação de matrizes são passadas também 3 matrizes, seguindo a mesma lógica das matrizes da soma: duas que serão multiplicadas e a última que recebe os valores resultantes.

Conforme opera a multiplicação de matrizes devemos respeitar o critério das dimensões das matrizes. A primeira matriz da multiplicação deve possuir o número de colunas igual ao número de linhas da segunda matriz, ou seja, $A_{\text{MxP}}\, B_{\text{PxN}}$, como resultado temos uma matriz A_{MxN}.

Nessa parte, temos o aninhamento de 3 for's, isso porque, na multiplicação de matrizes, cada linha da matriz A é multiplicada por cada coluna da matriz B, a cada operação de LinhaXColuna, somamos os produtos resultantes e o resultado é o valor respectivo para um elemento da matriz C.

2.2.7. Transposição de matrizes

A última operação solicitada, a transposição de matrizes, provém da conversão de linhas de A_{MxN} em colunas, ou seja, a matriz AT (A transposta) possui dimensão NxM, todos os elementos da primeira coluna de A passam a estar na primeira linha de A, os elementos da segunda coluna na segunda linha e assim por diante.

Com base nisso, como não há um critério de dimensão, passamos para a chamada dessa função a matriz A a ser transposta e uma segunda matriz inicializada com a dimensões de A invertida, ou seja, se A_{MxN} AT_{NxM} conforme explicado anteriormente. Dessa forma, a partir de um aninhamento de estruturas de repetição [for e for], obtém-se, passando por todos elementos de A, a matriz resultante.

2.2.8. Destruição de matrizes

Por fim, encerrando as operações de matrizes, temos suas respectivas destruições. Por se tratar de alocações dinâmicas, isso porque as variáveis alocadas continuam a existir mesmo após o findar da execução, diferente da alocação estática, o que gera um acúmulo de lixo na memória, pois continua a ser ocupada por tais variáveis.

Antes de iniciar a destruição da matriz precisamos verificar se essa ainda existe, caso contrário não é necessário fazer a desalocação, posteriormente , caso ela ainda exista realiza-se sua destruição.

Dessa forma, passamos a referência para a matriz a qual desejamos eliminar e deslocamos inicialmente todas as linhas da matriz, ou seja, para uma matriz $A_{\text{NxM,}}$ fazemos N desalocações e por fim desalocamos a matriz A criada na operação

3. Análise de Complexidade

3.1. Tempo

A partir de cada operação, será feita uma análise de custo temporal de cada operação realizada com as matrizes, baseando-se no número de vezes que a operação considerada relevante foi executada durante a operação.

Com isso será possível fazer a medição do custo em números de operações para executar um algoritmo de uma matriz de tamanho nxm.

3.1.1. Operação de soma

Para realizar a soma de matrizes bidimensionais devemos respeitar o critério da igualdade dimensional, ou seja, para duas matrizes $A = [a_{\text{ij}}]_{\text{mxn}}$ e $B = [b_{\text{ij}}]_{\text{mxn}}$ é possível realizar a soma entre elas se o número de linhas de A forem iguais a de B e o número de colunas de A iguais às de B, possuindo, portanto, a mesma ordem.

Com posse de matrizes que respeitem esse critério, basta que realizemos a soma entre as respectivas casas entre as duas matrizes, como podemos ver na figura abaixo. Dessa forma, obtemos uma terceira matriz resultante igual à soma das matrizes A e B.

$$A \pm B = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \pm \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{pmatrix} = \begin{pmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & \cdots & a_{1n} \pm b_{1n} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & \cdots & a_{2n} \pm b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} \pm b_{m1} & a_{m2} \pm b_{m2} & \cdots & a_{mn} \pm b_{mn} \end{pmatrix}$$

Figura 1 - Soma de Matrizes

Portanto, ao selecionar a operação de soma de matrizes, consideramos relevante apenas a parte do código que realiza essa operação, sendo, nesse caso, a soma entre a matriz A e matriz a B nas estruturas de repetição aninhadas [for e for].

Como não temos pior ou melhor caso, a operação será realizada por todos os $N * M$ elementos das matrizes, portanto a função de complexidade de tempo é $O(n^2)$.

3.1.2. Operação de multiplicação

Da mesma forma que a operação de adição, a multiplicação de matrizes bidimensionais exige que alguns critérios sejam respeitados. Nesse caso, o número de colunas da primeira matriz seja igual ao número de linhas da segunda, ou seja, $A = [a_{ij}]_{mxp}$ e $B = [b_{ij}]_{pxn}$.

Com posse de matrizes que respeitem esse critério, precisamos multiplicar cada linha de A por cada coluna de B, dessa maneira, cada elemento de A será multiplicado pelo elemento em B que possui o mesmo valor de i (na mesma linha) e os valores dessa operação são somados resultando no elemento da matriz resultante, como pode-se visualizar na figura abaixo. Dessa forma, obtemos uma terceira matriz resultante igual à multiplicação das matrizes A e B.

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \quad B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} & a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + a_{13} \cdot b_{32} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} & a_{21} \cdot b_{12} + a_{22} \cdot b_{22} + a_{23} \cdot b_{32} \end{pmatrix}$$

Figura 2 - Multiplicação de matrizes

Portanto, ao selecionar a operação de multiplicação de matrizes, consideramos relevante apenas a parte do código que realiza essa operação, sendo, nesse caso, a multiplicação entre a matriz A e matriz a B nas estruturas de repetição aninhadas [for, for e for].

Como não temos pior ou melhor caso, a função de complexidade de tempo é $O(n^3)$.

3.1.3. Operação de transposição

Por fim, temos a transposição de uma matriz bidimensional. Nesse caso, não há um critério exato para que ela seja transposta. Portanto, seja a matriz $A = [a_{ij}]_{mxn}$ sua transposta é a matriz $AT = [b_{ij}]_{nxm}$ no qual suas linhas passam a ser suas colunas e suas colunas suas linhas.

É possível ver essa operação na figura abaixo.

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \Leftrightarrow A^{\mathrm{T}} = \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{m,n} \end{bmatrix}$$

Figura 3 - Transposição de matriz

Portanto, ao selecionar a operação de transposição de matriz, consideramos relevante apenas a parte do código que realiza essa operação, sendo, nesse caso, a transposta de A na estrutura de repetição [for].

Como não temos pior ou melhor caso, permutamos os $N * M$ elementos da matriz, resultando em uma função de complexidade de tempo de $O(n^2)$.

3.2. Espaço

Considerando que cada elemento da matriz ocupe uma unidade de espaço, para nossa análise. Dessa forma, temos 3 casos, a soma de matrizes, que precisamos de 3 matrizes, as duas que vão ser somadas e a matriz resultante. Posterior a isso, a multiplicação, na qual também teremos 3 matrizes, seguindo a mesma lógica da soma e, por fim, a transposição, nela temos apenas duas matrizes, a matriz inicial e a resultante.

Em todas essas operações utilizamos matrizes e são elas os elementos armazenados. Portanto, teremos para a soma e a multiplicação $O(3n^2) = O(n^2)$ e, para a transposição, $O(2n^2) = O(n^2)$.

4. Estratégias de Robustez

Os mecanismos de programação defensiva e tolerância a falhas são de extrema importância na realização de programas, isso porque evita que o usuário passe entradas incorretas, utilize dessas falhas para danificar de alguma forma o programa, ou obtenha resultados inesperados na execução das funções do programa.

Nessa etapa foram utilizadas as funções __erroassert e __avisoassert para informar ao usuário problemas durante a execução do programa ou por introdução de valores incorretos.

4.1. Controle de opções

Durante a execução do programa utilizam-se switch's para verificação das opções de entrada do usuário, por restringirmos as opções e por permitimos a realização de apenas uma operação[Soma, Multiplicação ou Transposição] por chamada do programa é necessário verificar sempre se o usuário escolheu apenas uma das operações possíveis e, caso tenha escolhido apenas uma, se é válido.

4.2. Controle de dimensões

Conforme informado anteriormente, as matrizes utilizadas nas operações devem seguir critérios para sua correta execução. Dessa forma, durante todo o programa são verificados diversas vezes se a ordem das matrizes satisfazem aos critérios de operação

4.2.1. Critério de dimensão das matrizes

De acordo com o enunciado do trabalho "todas as matrizes sobre as quais operadores devem ser implementados são bidimensionais", dessa forma, deve-se controlar a dimensão da matriz impedindo que valores nulos ou negativos sejam utilizados, além disso, por ter duas dimensões, não é permitido matriz com valor único. Portanto, casos que não respeitem tais critérios ocasionam no término da execução do programa por erro. Como podemos ver um exemplo abaixo.

```
bin/matop -1 test/m1.txt -2 test/m2.txt -o /tmp/mem/somaresult.out -s -p /tmp/mem/somalog.out -l
src/matop.cpp:63: Erro 'opt->optx_mat1 > 1 && opt->opty_mat1 > 1' - Dimensoes de matriz invalida.
```

Nesse caso, inseriu-se dimensões negativas gerando em um inconsistência para o programa ocasionando em seu término.

4.2.2. Soma e multiplicação critérios não respeitados

Como citado anteriormente existem critérios para realização das operações entre matrizes, dimensões iguais e número de linhas igual ao número de colunas, nesses casos é importante verificarmos tais casos, evitando erros durante a execução do programa.

Para isso, há a verificação da igualdade das dimensões para evitar qualquer erro, podemos ver um exemplo na imagem abaixo.

```
bin/matop -1 test/m1.txt -2 test/m2.txt -o /tmp/mem/somaresult.out -s -p /tmp/mem/somalog.out -l
src/mat.cpp:158: Erro 'a->cols == b->cols' - Dimensões incompatíveis
```

4.2.3. Dimensão x valores passados

Durante a passagem da matriz por meio do arquivo podem haver discordância na quantidade de valores passados e a dimensão passada, para isso é importante verificar se todos valores passados são válidos para então adicioná-los à matriz. Além de verificar se a disposição dos valores estão de acordo.

Por exemplo, em uma situação na qual há a passagem de uma matriz 10x10, porém há uma linha com apenas 9 elementos. Nesse caso, haverá um resultado inesperado, portanto é preferível encerrar a execução para evitar futuros problemas. Podemos ver um exemplo abaixo.

```
bin/matop -1 test/m1.txt -2 test/m2.txt -o /tmp/mem/somaresult.out -s -p /tmp/mem/somalog.out -l
src/mat.cpp:108: Erro 'istr.eof()' - Arquivo de matriz nao segue a especificacao.
```

4.2.4. Matriz inexistente

Durante a destruição de uma matriz é padronizado a atribuição de suas linhas e colunas a -1, isso para evitar a tentativa posterior de desalocar ou para não haver a manipulação de uma matriz já destruída.

Dessa forma, é importante, antes de destruir a matriz, verificar se ela existe, evitando problemas no código. Um exemplo da tentativa de destruição dupla de uma matriz pode ser vista abaixo.

```
src/mat.cpp:218: Aviso: '((a->linhas>0)&&(a->cols>0))' - Matriz já foi destruida
double free or corruption (!prev)
```

4.3. Verificação de alocação

A alocação dinâmica de memória possibilita “reserva” de espaços de memória na heap para atribuir endereços posteriormente. Entretanto, essa ferramenta, além de muito útil e prestativa, oferece muitos riscos à integridade do programa quando utilizada incorretamente, por isso é importante verificar a corretude da sua execução evitando futuras falhas.

Podemos ver dois casos de erros abaixo, uma quando aloca-se menos espaços do que esperado e no segundo caso, quando há uma falha, por algum motivo, na alocação.

```
bin/matop -1 test/m1.txt -2 test/m2.txt -o /tmp/mem/somaresult.out -s -p /tmp/mem/somalog.out -l
make: *** [Makefile:33: mem] Falha de segmentação (arquivo core criado)
```

```
bin/matop -1 test/m1.txt -2 test/m2.txt -o /tmp/mem/somaresult.out -s -p /tmp/mem/somalog.out -l
src/mat.cpp:41: Erro 'mat->m!=NULL' - Malloc falhou
```

Dessa forma, é muito importante atentar-se a todas manipulações dinâmicas realizadas no programa evitando possíveis erros durante a execução.

4.4. Verificação de abertura de arquivos

Na implementação desse programa foi solicitado pegar os dados de matrizes de arquivos, além de gravar as matrizes resultantes em outros, ou seja, a utilização de arquivos é de extrema importância no decorrer do programa. Entretanto, podem haver falhas na abertura desses, o que impossibilita a execução do programa. Portanto, é de suma importância verificar se os arquivos foram abertos corretamente para utilização no programa.

Durante a execução podemos passar valores incorretos na função de abrir arquivos, dando informações até então inexistentes no computador, como por exemplo, ao tentar abrir o arquivo “matriz.txt” pedir para abrir o “matruz.txt”, por não haver um arquivo com esse nome é importante

que exista um aviso ao usuário sobre o erro, ou mesmo por impossibilidade de abrir um arquivo existente por estar corrompido, por exemplo. Podemos ver um exemplo de falha abaixo.

```
bin/matop -1 test/m1.txt -2 test/m2.txt -o /tmp/mem/somaresult.out -s -p /tmp/mem/somalog.out -l
src/mat.cpp:70: Erro 'arq' - Arquivo de matriz inexistente ou invalido.
```

5. Testes

Para as diversas análises do programa criado, foram introduzidas matrizes de diferentes tamanhos. Foram testados ao todo 5 diferentes entradas, sendo essas matrizes: $5x5, 10x10, 25x25, 50x50$ e $100x100$, todas com valores aleatórios e reais do intervalo $[-100, 100]$ com duas casas decimais.

A partir desses dados, foram obtidos mapas de acesso, histogramas, tempos de execução, custos relativos, dentre outros dados. Tais informações foram obtidas através da utilização das ferramentas: gprof, gnuplot, memlog e analisamem.

Os testes não ultrapassaram a dimensão $100x100$ por problemas obtidos no computador onde foram realizados os testes, tais problemas serão discutidos posteriormente.

Para exemplificar a saída das matrizes usaremos a $5x5$. Isso decorre da complexidade de analisar operações com grandes matrizes e que é possível verificar a corretude do código e suas operações com matrizes pequenas.

5.1. Matriz 5x5

Nesse caso geramos duas matrizes $5x5$ das quais foram feitas as operações de soma, multiplicação e transposição, a transposição foi realizada apenas com a primeira matriz. As matrizes utilizadas e os resultados das operações podem ser vistas abaixo:

Matriz 1	Matriz 2
-98.04 -16.94 71.86 26.37 39.78 48.82 23.96 19.31 -0.04 -69.29 -59.38 -71.43 73.49 70.72 68.91 -75.76 -26.60 -75.87 29.66 -14.98 -86.96 -65.79 -96.31 -62.80 33.20	-23.14 34.66 -24.10 -61.80 -40.12 42.98 -87.23 92.02 19.12 0.72 57.80 -18.50 5.63 25.53 -29.77 32.98 -97.98 96.00 49.38 63.62 88.17 54.88 -34.44 -11.67 -18.95

Soma [Matriz 1 + Matriz 2]	Multiplicação [Matriz 1 * Matriz 2]
-121.18 17.72 47.76 -35.43 -0.34 91.8 -63.27 111.33 19.08 -68.57 -1.58 -89.93 79.12 96.25 39.14 -42.78 -124.58 20.13 79.04 48.64 1.21 -10.91 -130.75 -74.47 14.25	10071.2 -3650.41 2370.01 8407.48 2705.72 -5094.39 -4553.88 3519.46 -1259.34 -1205.77 10959.9 -334.202 -312.322 6868.12 3336.46 -4118.07 -2630.12 2314.21 3875.84 7449.83 -5526.02 12481.7 -11672.7 -1831.08 1684.14

Transposição [Matriz 1^t]

```
-98.04 48.82 -59.38 -75.76 -86.96
-16.94 23.96 -71.43 -26.6 -65.79
71.86 19.31 73.49 -75.87 -96.31
26.37 -0.04 70.72 29.66 -62.8
39.78 -69.29 68.91 -14.98 33.2
```

5.2. Matrizes maiores que 100x100

Com objetivo de obter testes com grandes tempos de execução houveram tentativas de realizar operações com matrizes cada vez maiores, comparando-se também com exemplos utilizados em outros trabalhos práticos 0. Entretanto, possivelmente pela máquina utilizada foram verificadas interrupções ou tempos muito longos, anormais comparados aos testes de outras pessoas.

A partir disso, foi realizada a contagem de tempo das mesmas operações em uma matriz 150x150, porém o programa ainda estava em execução mesmo após 6min48seg do início dela. Além disso, havia uma demanda imediata de 100% da CPU após a inicialização das operações. Outro fator observado é que sempre paravam após o analisamem do arquivo de saída multlog.out. Podemos ver tais casos nas figuras abaixo.

Figura 6 - Cronometro (Google) início à para forçada da execução

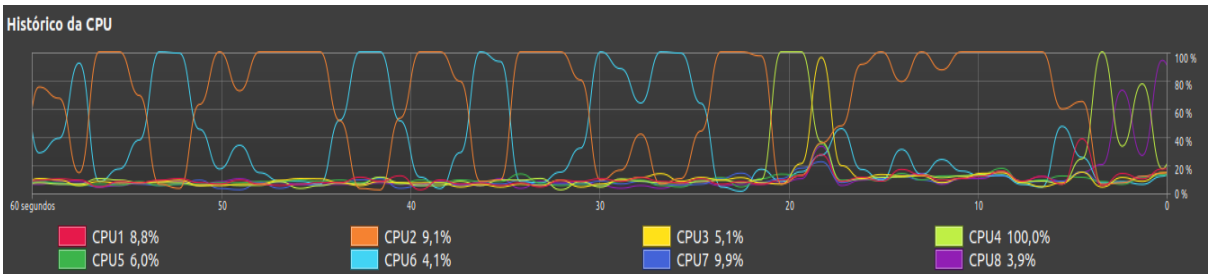


Figura 7 - Histórico da CPU durante a execução dos programas

```
2 2 0 22500
2 2 22500 22500
bin/matop -1 test/m1.txt -2 test/m2.txt -o /tmp/mem/multresult.out -m -p /tmp/mem/multlog.out -l
rm -rf /tmp/multin
mkdir /tmp/multin
../analisamem/bin/analisamem -i /tmp/mem/multlog.out -p /tmp/multin/multin
minend 94331446857952 maxend 94331447413816 range 555864 numend 69485
minfase 0 maxfase 2 range 2 numfase 3
minid 0 maxid 2 numid 3
```

Figura 8 - Momento no qual a execução trava

Além desse teste, foi feita a tentativa de execução em matrizes da ordem de 10000x10000, porém houve o alerta de capacidade de memória atingida, mesmo havendo muita memória disponível, possivelmente porque a alocação dinâmica de todo esses elementos acabou por preencher todo espaço disponível para o programa ou para o sistema Linux no computador.

Na ausência de outra máquina para realização dos testes, as análises foram obtidas a partir das 5 citadas anteriormente, o que teve impacto significativo nos resultados de tempo de execução.

6. Análise Experimental

Para a análise do programa, baseando-se nas entradas citadas anteriormente foi possível gerar diversos gráficos e obter informações acerca dos algoritmos utilizados. Por maior facilidade de visualização os resultados são respectivos às operações com a matriz 10x10. As operações possuem a seguinte organização:

$A_{NxM} + B_{NxM} = C_{NxM}$

$A_{NxP} * B_{PxM} = C_{NxM}$

$A_{NxM} \rightarrow AT(\text{A transposto}) = B_{MxN}$

Sendo A codificado com o $ID = 0$, B com $ID = 1$ e C com $ID = 2$. A partir disso podemos

Todos os mapas e histogramas se baseiam nessa organização e implementação conforme o código de mat.cpp disponibilizado

6.1. Mapa de acesso

Mapas de acesso são gráficos de pontos mostrando quais endereços foram acessados ao longo do tempo, sendo X o tempo e Y o endereço. Pontos próximos em X indicam endereços acessados em tempos próximos.

6.1.1. Soma

O mapa de acesso da matriz *A* na soma possui inicialmente duas sequências de acesso de escrita, isso decorre de sua inicialização. Todos os elementos de *A* são inicialmente setados em 0, para evitar erros, ou qualquer outro problema posterior, depois é inicializado com os valores passados no arquivo respectivo a matriz 1. Podemos ver que os elementos são acessados sequencialmente, iniciando da primeira linha primeira coluna, primeira linha segunda coluna e assim por diante.

Posteriormente, temos um novo acesso à matriz A, respectiva a leitura, realizada para se definir as distâncias de pilha utilizadas nas representações dos histogramas.

Por fim, temos a uma sequência de acessos respectivas a leitura, nesse após de inicializada e acessada a matriz é lida elemento a elemento para somá-los aos respectivos elementos de B.

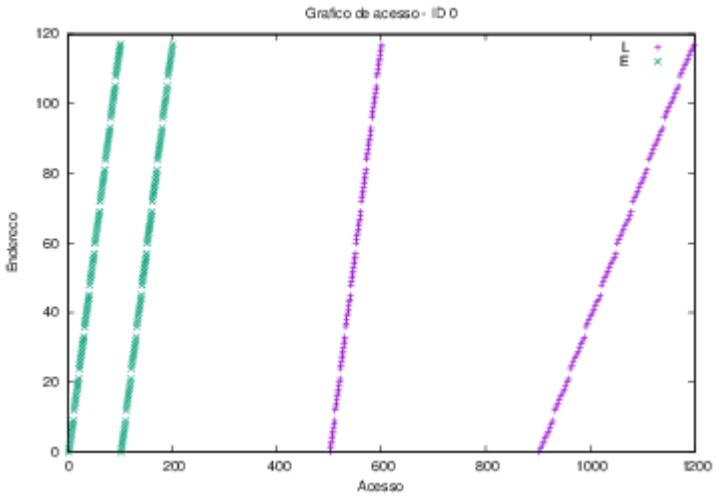


Figura 9 - Mapa de acesso à memória Matriz A na soma

O mapa de acesso da matriz *B* na soma possui inicialmente duas sequências de acesso de escrita, isso decorre de sua inicialização. Todos os elementos de *B* são inicialmente setados em 0,

assim como a matriz A, depois é inicializado com os valores passados no arquivo respectivo a matriz 2. Podemos ver que os elementos são acessados sequencialmente, iniciando da primeira linha primeira coluna, primeira linha segunda coluna e assim por diante.

Posteriormente, temos um novo acesso à matriz B, respectiva a leitura, realizada para se definir as distâncias de pilha utilizadas nas representações dos histogramas.

Por fim, temos a uma sequência de acessos respectivas a leitura, nesse após de inicializada e acessada a matriz é lida elemento a elemento para somá-los aos respectivos elementos de A.

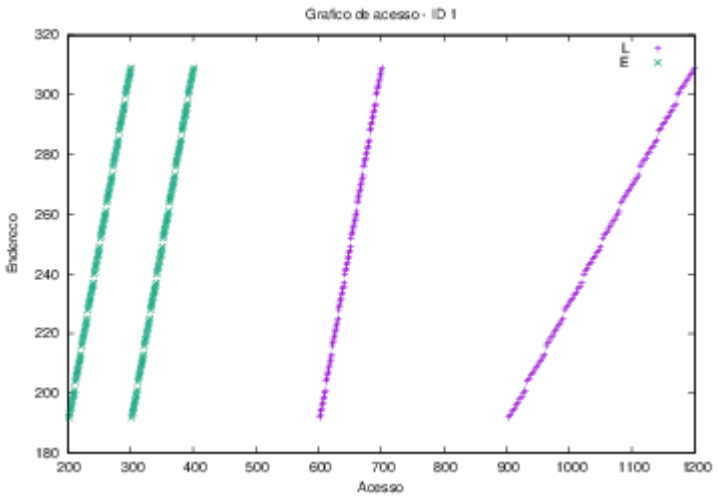


Figura 10 - Mapa de acesso à memória Matriz B na soma

Por fim, o mapa de acesso da matriz resultante na soma possui inicialmente uma sequências de acesso de escrita, isso decorre de sua inicialização nula, assim como a matriz A e B.

Posteriormente, temos um novo acesso à matriz C, respectiva a leitura, realizada para se definir as distâncias de pilha utilizadas nas representações dos histogramas.

Na sequência a matriz C é reinicializada com 0, evitando qualquer modificação durante a execução de outras funções e por fim os elementos da operação de soma são alocados a ele.

Por fim temos um novo acesso a C e um posterior respectivo a escrita dos seus valores no arquivo de saída

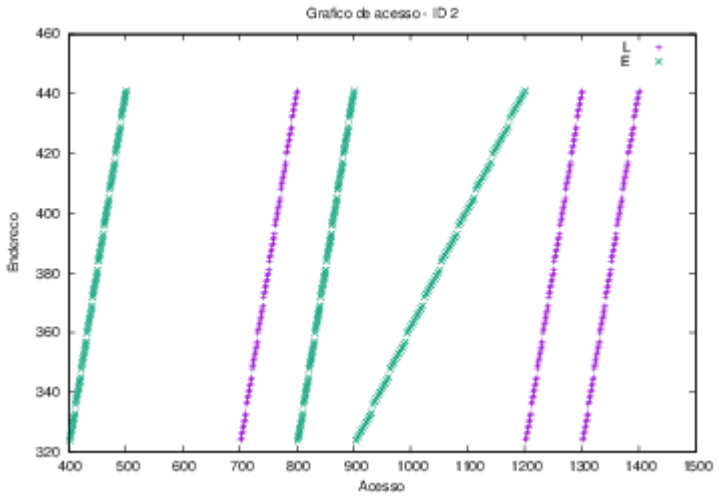


Figura 11 - Mapa de acesso à memória Matriz C (resultante) na soma

6.1.2. Multiplicação

Seguindo a lógica da soma de matrizes, a matriz A é inicializada com 0 e posteriormente com os valores provindos do arquivo respectivo aos seus elementos. Em seguida, faz-se a leitura de seus dados para definição das distâncias de pilha analisadas posteriormente nos histogramas.

É possível ver, acessos sequenciais de mesmos endereços no mapa. Isso provém do fato de que na operação de multiplicação de matrizes cada linha de A é multiplicada por todas as colunas de B, nesse caso, como B possui 10 colunas, há o acesso a mesma linha da matriz A 10 vezes.

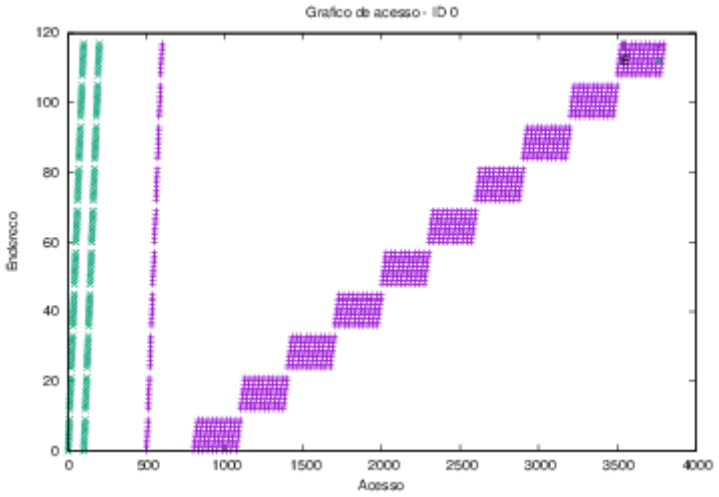


Figura 12 - Mapa de acesso à memória Matriz A na multiplicação

Nesse sentido, a matriz B é inicializada com 0 e posteriormente com os valores provindos do arquivo respectivo aos seus elementos. Posteriormente, faz-se a leitura de seus dados para definição das distâncias de pilha analisadas posteriormente nos histogramas.

É possível ver acessos consecutivos de endereços não tão próximos, porém com uma distância parecida. Isso provém do fato de que na operação de multiplicação de matrizes cada coluna de B é multiplicada por todas as linhas de A, ou seja, há o acesso do primeiro elemento da primeira linha e depois do primeiro elemento da segunda linha, entretanto, a escrita da matriz B é feita linha por linha,

portanto, a distância entre os elementos de cada linha da mesma coluna é igual a quantidade de colunas da matriz. Nesse caso, a distância é igual a 10.

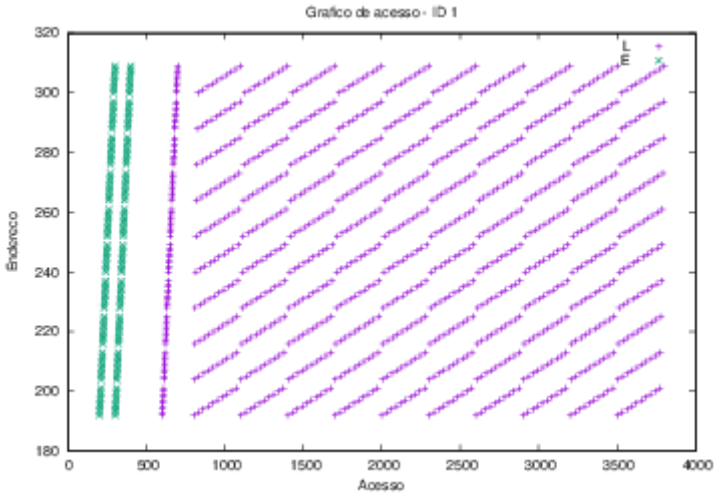


Figura 13 - Mapa de acesso à memória Matriz B na multiplicação

Por fim, a matriz C é inicializada com 0 e então acessada para definição das respectivas distâncias de pilha. Posteriormente, faz-se a escrita de seus dados, sendo estes resultantes da multiplicação de $A * B$.

Podemos perceber que a um condensamento maior de escritas para um mesmo endereço de memória para só então passar para o próximo endereço. Isso decorre do fato de a operação de multiplicação possuir a soma dos elementos de linhas por colunas de $A * B$, como a 10 elementos na linha de A e 10 elementos nas colunas de B, há a soma de também 10 elementos na matriz resultante C.

Posteriormente ela é reacessada e então impressa no arquivo resultante.

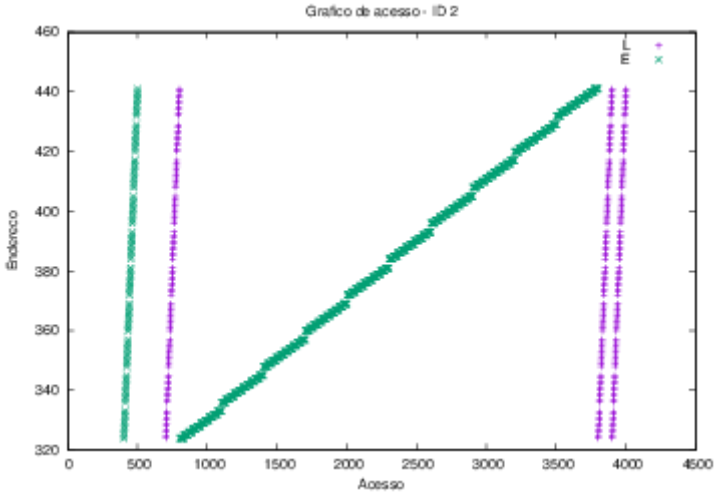


Figura 14 - Mapa de acesso à memória Matriz C (resultante) na multiplicação

6.1.3. Transposição

Seguindo a lógica das duas outras operações de matrizes, a matriz A é inicializada com 0 e posteriormente com os valores provindos do arquivo respectivo aos seus elementos. Em seguida, faz-se a leitura de seus dados para definição das distâncias de pilhas.

Podemos ver acessos dispersos semelhante as dispersões ocorridas na matriz *B* da multiplicação de matrizes, isso decorre do fato do acesso ser feito por colunas. Pois, precisamos passar as colunas de A para as linhas de AT, resultando nesse tipo de acesso.

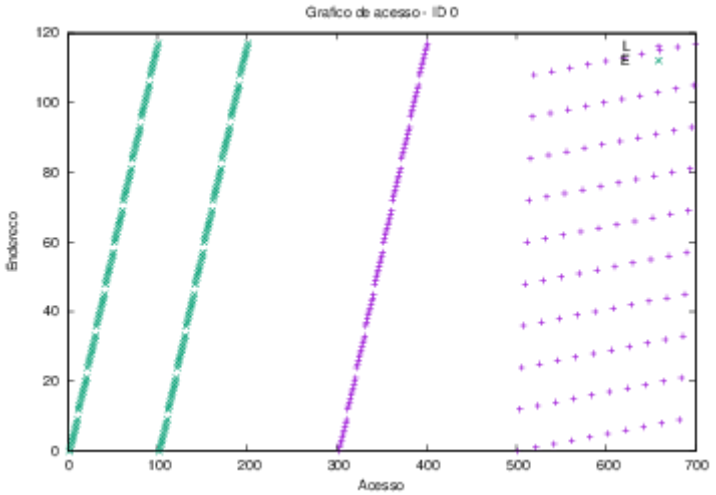


Figura 15 - Mapa de acesso à memória Matriz A na transposição

Por fim, na matriz resultante da transposição são realizadas inicialização com valores nulos e posteriormente os dados das colunas de *A* escritos nas linhas de *B*. Pois, precisamos passar as colunas de A para as linhas de AT. E ao findar da operação, há o acesso em *B* e posteriormente ele é escrito no arquivo resultante.

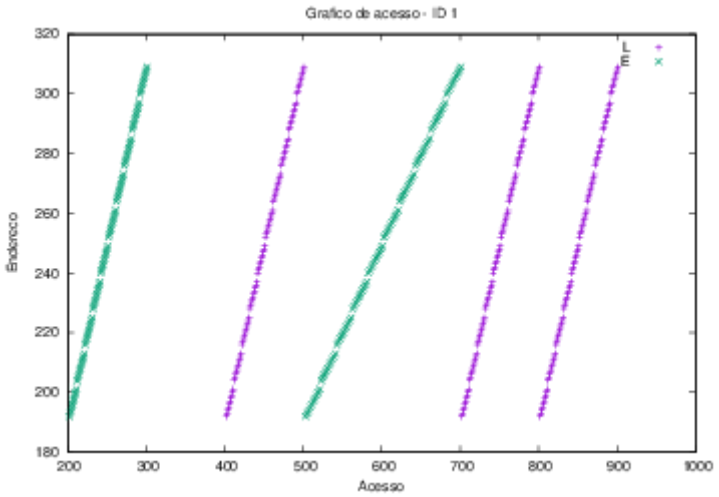


Figura 16 - Mapa de acesso à memória Matriz B (resultante) na transposição

6.2. Histogramas

Histogramas são gráficos de barras quais distâncias de pilha foram acessados ao longo da execução do programa, sendo X a distância e Y a quantidade de vezes que essa distância foi acessada.

Todos os histogramas foram divididos em três fases:

1ª - respectiva à criação e inicialização das matrizes.

2ª - respectiva aos acessos e operações das matrizes.

3ª - respectiva ao acesso e impressão da matriz resultante e desalocação das matrizes.

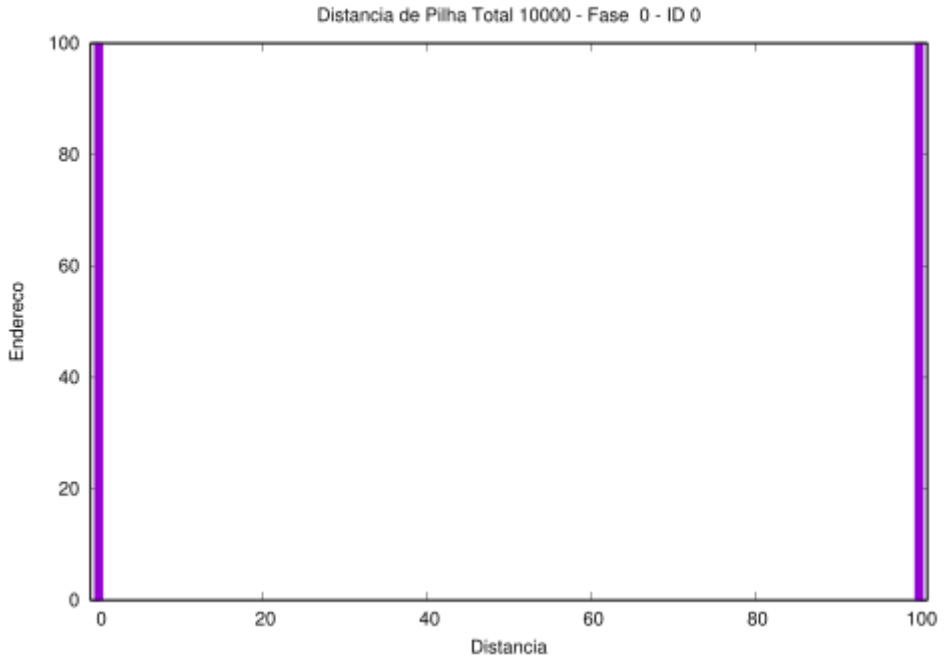
6.2.1. Soma

6.2.1.1. Fase 0

Por se tratar de inicialização tanto a matriz *A* quanto a matriz *B* são inicialmente inicializadas com valor 0, como é o primeiro acesso à matriz não há distância de pilha. Posteriormente, as matrizes são inicializadas e o primeiro valor a ter o valor 0 atribuído é o último na pilha, neste caso, possui distância igual a 100. Após a inicialização do primeiro elemento o segundo a ser atribuído o 0, porém passou a ser o último após o acesso do primeiro elemento, e assim sucessivamente até todos serem atribuídos ao valor esperado.

Nessa fase, podemos perceber que o histograma da matriz C só possui distância de pilha igual a 0, isso ocorre pois, nessa etapa ela é acessada apenas uma vez, não possuindo as distâncias de pilha para a análise.

Figura 17 - Histograma



Matriz A na soma - Fase 0

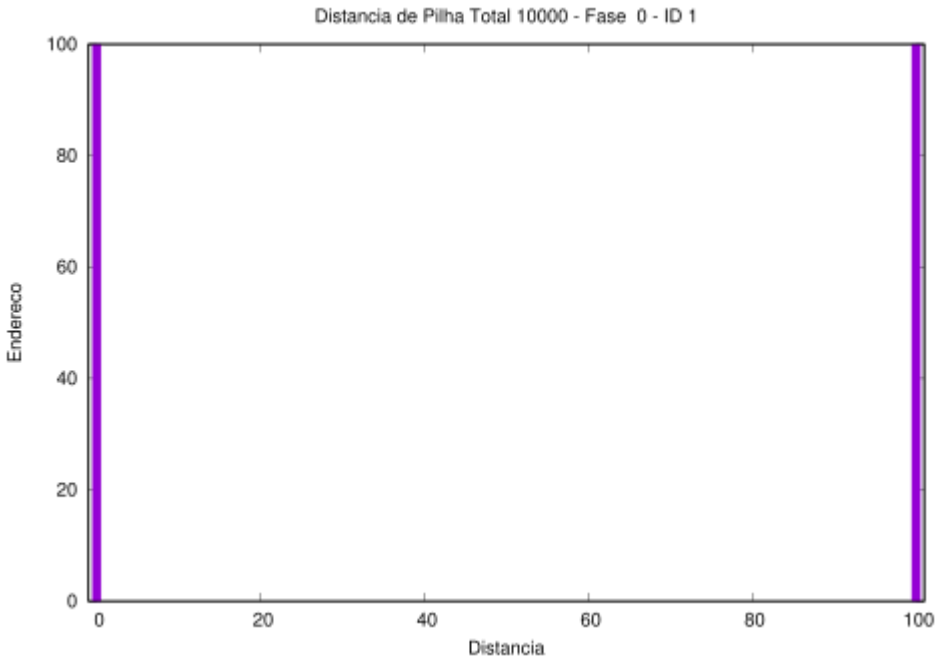


Figura 18 - Histograma Matriz B na soma - Fase 0

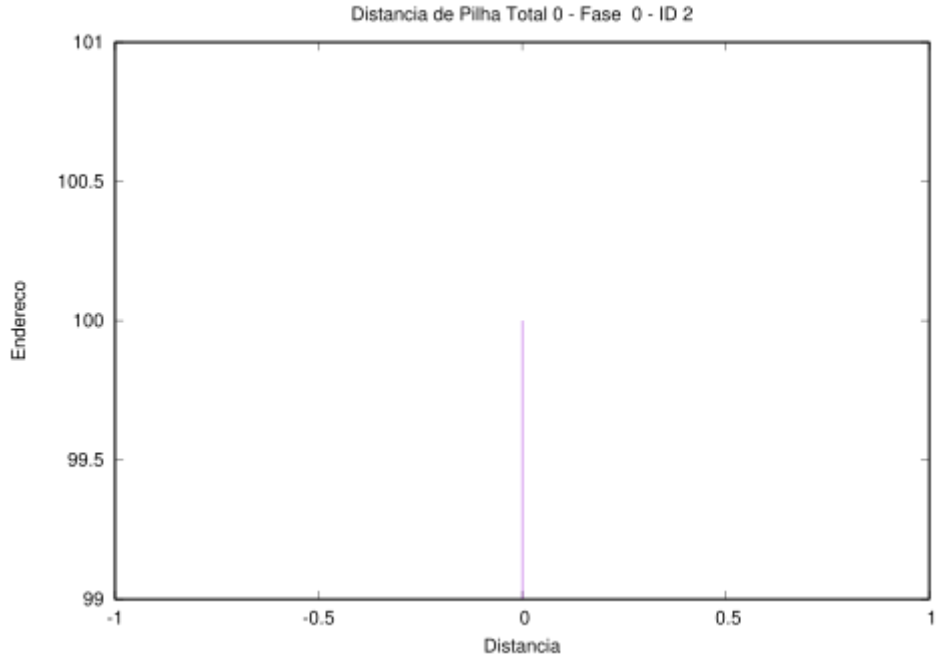


Figura 19 - Histograma Matriz C (resultante) na soma - Fase 0

6.2.1.2. Fase 1

Tratando-se de soma tanto a matriz A quanto a matriz B são percorridas da mesma forma. Dessa maneira, após o acesso às matrizes percorremos cada elemento de A e B, iniciando do primeiro elemento a ser inicializado ao último, como visto esse forma de percorrer leva a uma concentração de acessos na distância 100.

Nessa fase, podemos perceber que o histograma da matriz C possui distância de pilha igual a 0, isso ocorre pois, nessa etapa ela é acessada não possuindo as distâncias de pilha para a análise. Depois disso, a distância 100 é acessada 200 vezes, isso decorre do fato da matriz ser acessada, inicializada com zero e posteriormente atribui-se os valores resultantes da soma.

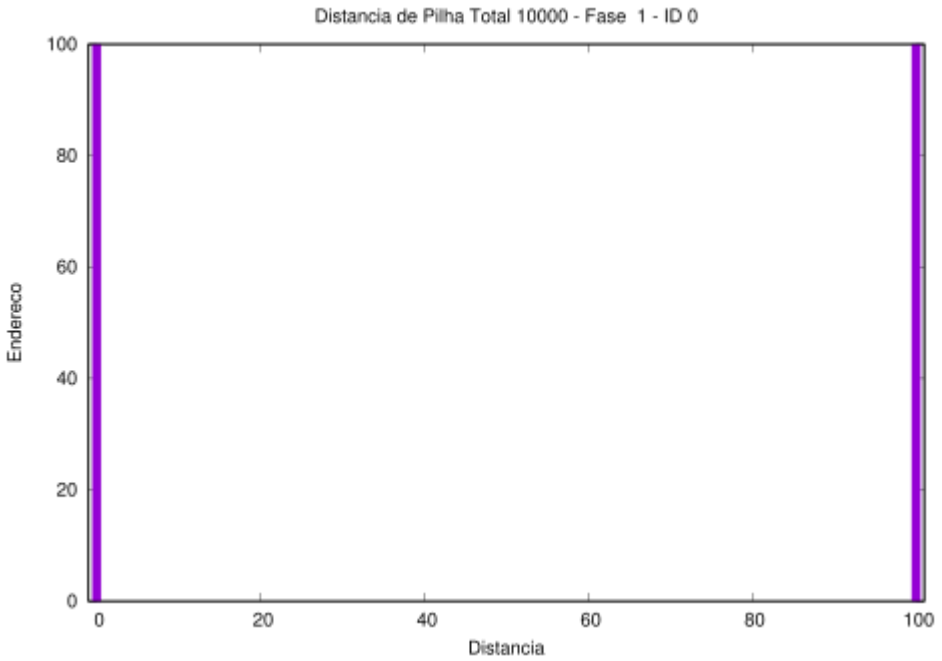


Figura 17 - Histograma Matriz A na soma - Fase 1

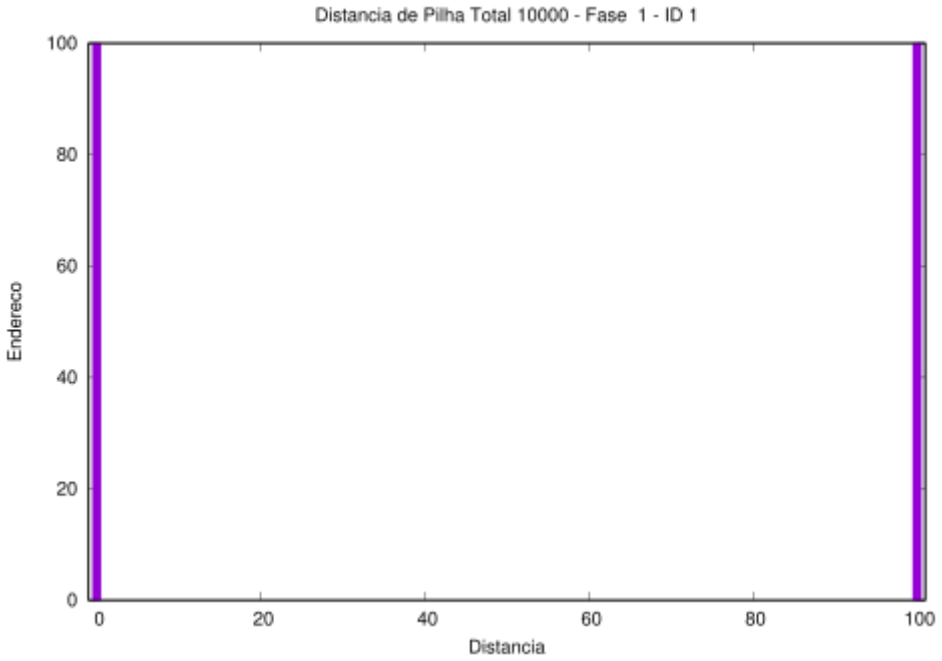


Figura 18 - Histograma Matriz B na soma - Fase 1

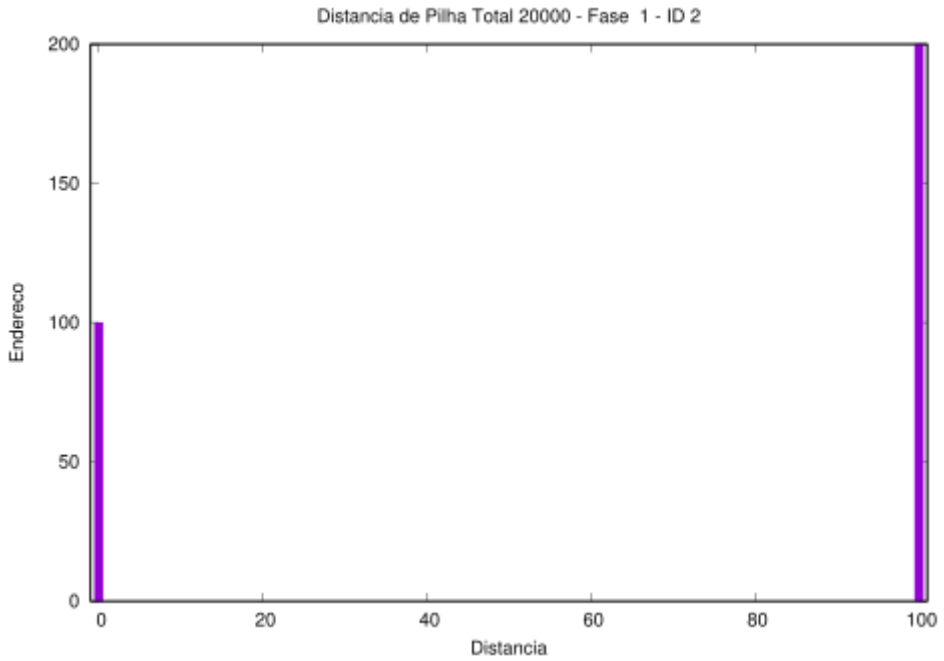


Figura 19 - Histograma Matriz C (resultante) na soma - Fase 1

6.2.1.3. Fase 2

Por fim, na última fase, temos operações apenas com a matriz resultante, nessa ela é acessada e posteriormente impressa no arquivo resultante, imprimindo o primeiro elemento acessado primeiro, ou seja, o último da pilha seguindo a mesma lógica de acesso das fases passadas, gerando a concentração no 100.

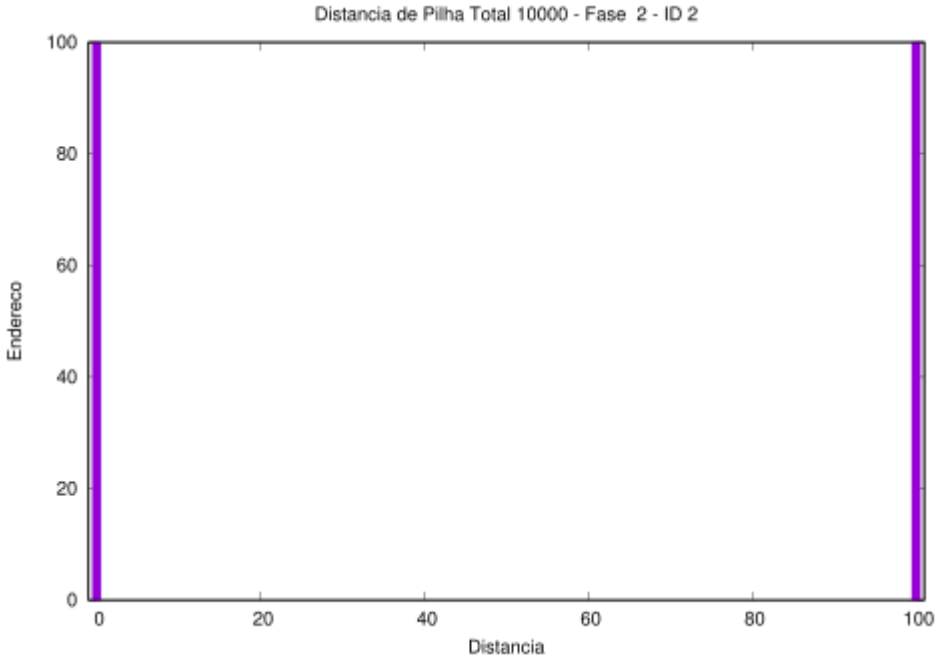


Figura 20 - Histograma Matriz B(resultante) na soma - Fase 2

6.2.2. Multiplicação

6.2.2.1. Fase 0

Por se tratar de inicialização tanto a matriz *A* quanto a matriz *B* são inicialmente inicializadas com valor 0, como é o primeiro acesso à matriz não há distância de pilha. Posteriormente, as matrizes são inicializadas e o primeiro valor a ter o valor 0 atribuído é o último na pilha, neste caso, possui distância igual a 100. Após a inicialização do primeiro elemento o segundo a ser atribuído o 0, porém passou a ser o último após o acesso do primeiro elemento, e assim sucessivamente até todos serem atribuídos ao valor esperado. Tal lógica é a mesma que ocorre na inicialização das matrizes na operação de soma.

Nessa fase, podemos perceber que o histograma da matriz C só possui distância de pilha igual a 0, isso ocorre pois, nessa etapa ela é acessada apenas uma vez, não possuindo as distâncias de pilha para a análise. Conforme visto na operação de soma.

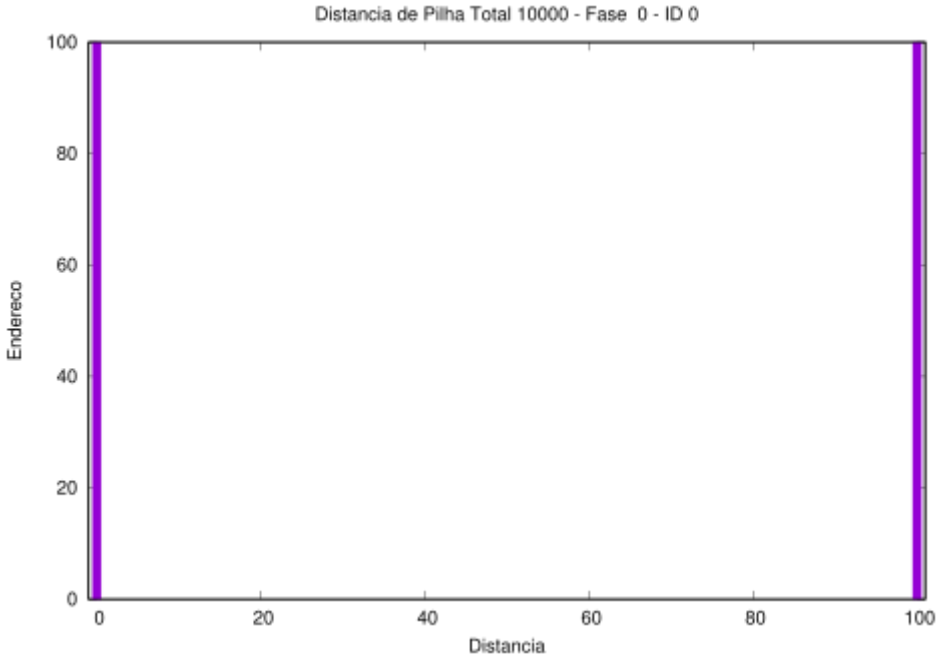


Figura 21 - Histograma Matriz A na multiplicação - Fase 0

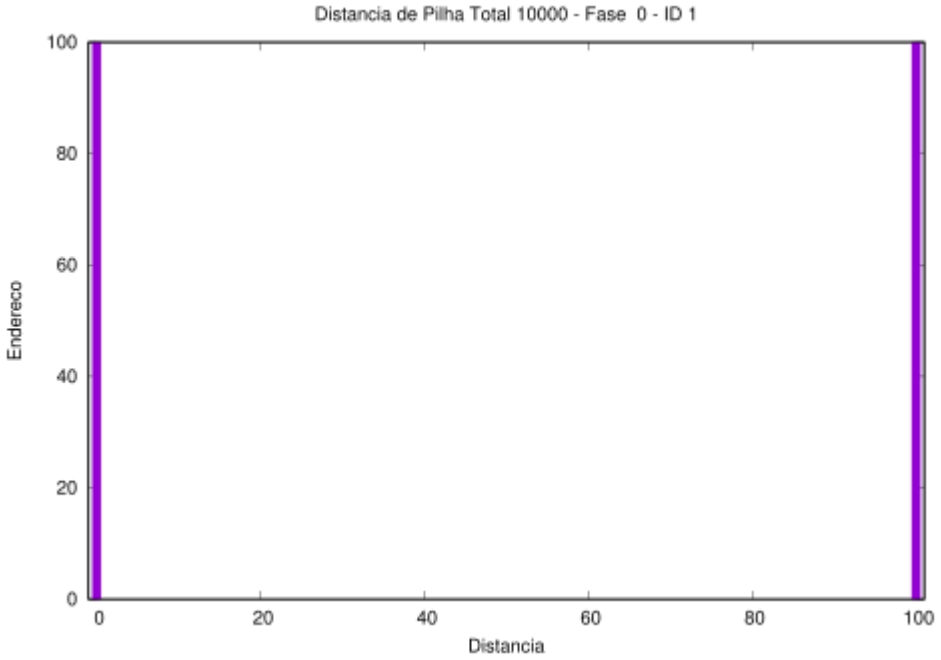


Figura 22 - Histograma Matriz B na multiplicação - Fase 0

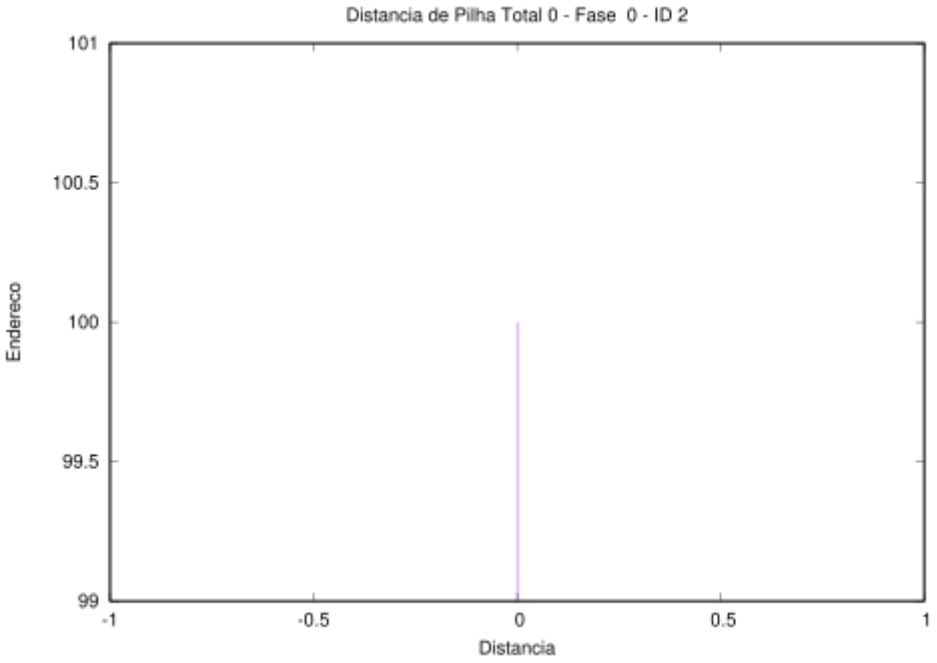


Figura 23 - Histograma Matriz C(resultante) na multiplicação - Fase 0

6.2.2.2. Fase 1

Nessa etapa, a matriz A, após ser inicializada, é acessada por linhas e temos o acesso ao primeiro elemento inicializado de cada linha, que possui distância de pilha 10, posteriormente acessamos o segundo elemento dessa mesma linha, que inicialmente possuía distância 9, após o acesso ao elemento anterior passa a ser o 10º, e assim sucessivamente, acessando sempre o elemento com a maior distância de pilha, que para uma matriz 10x10 é o 10 elemento. Por tal motivo uma concentração de distâncias igual a 10.

A matriz B, possui um comportamento distinto, visto que acessamos-a por colunas. Após sua inicialização, ao acessar seus elementos, temos um comportamento não uniforme, como pode ser visto no gráfico, isso porque o primeiro elemento da primeira coluna foi o primeiro a ser inicializado e por isso possui distância de pilha igual a 100, no primeiro momento. Posteriormente, acessamos o segundo elemento da primeira coluna, sendo ele o nonagésimo elemento da pilha, porém, com o acesso ao primeiro ele passa a ser o nonagésimo primeiro, o terceiro passa de octogésimo à octogésimo segundo causando um acesso mais irregular, se comparado ao da matriz A.

Nessa fase, podemos perceber que o histograma da matriz C possui distância de pilha igual a 0, isso ocorre pois, nessa etapa ela é acessada não possuindo as distâncias de pilha para a análise. Depois disso, a distância 1 é acessada, isso decorre do fato da matriz ser inicializada com elemento a elemento, linha por linha, acessando sempre o primeiro elemento da lista e, por fim, é acessada iniciando pelo primeiro elemento a ser inicializado, seguindo a mesma lógica das últimas operações e concentrando acessos à distância 100.

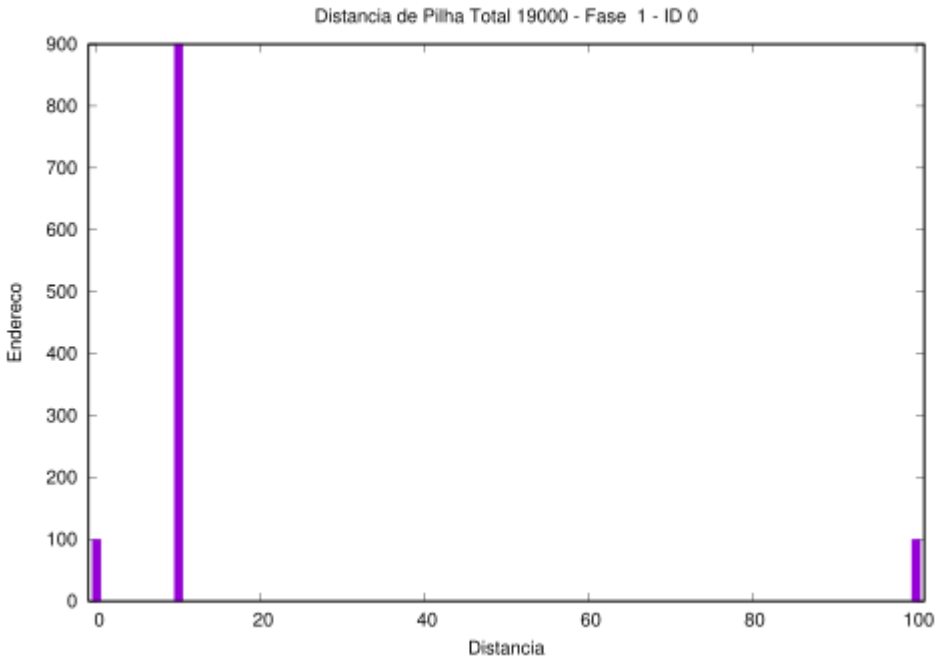


Figura 24 - Histograma Matriz A na multiplicação - Fase 1

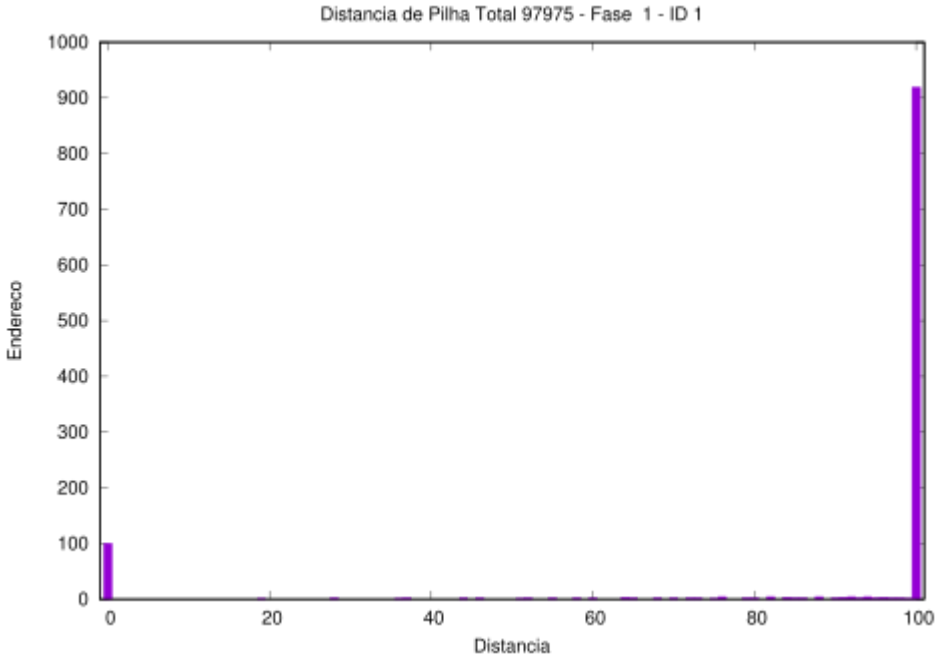


Figura 25 - Histograma Matriz B na multiplicação - Fase 1

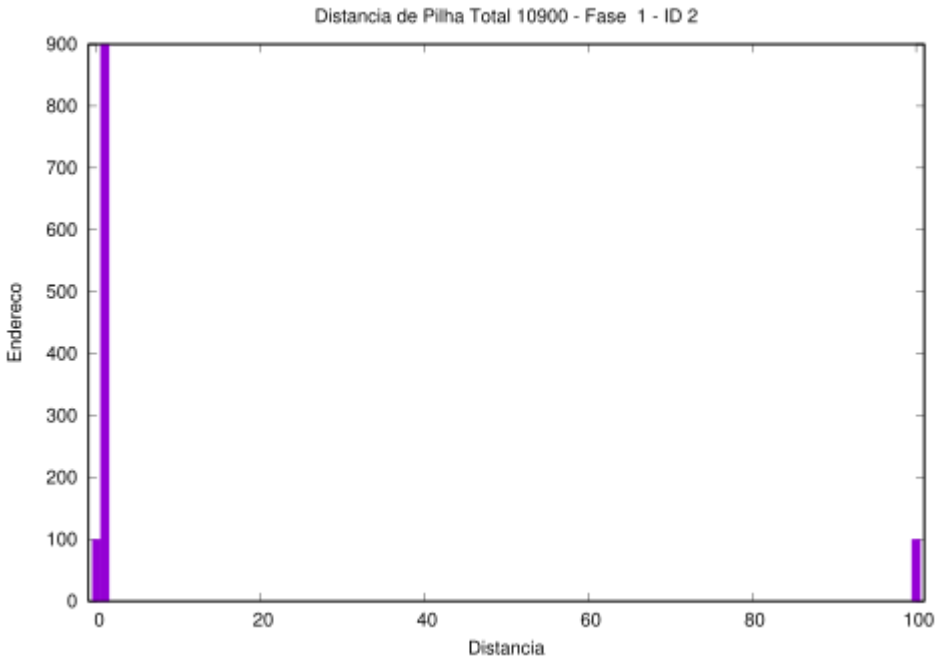


Figura 26 - Histograma Matriz C(resultante) na multiplicação - Fase 1

6.2.2.3. Fase 2

Por fim, na última fase, temos operações apenas com a matriz resultante, nessa ela é acessada e posteriormente impressa no arquivo resultante, imprimindo o primeiro elemento acessado primeiro, ou seja, o último da pilha seguindo a mesma lógica de acesso das fases passadas, gerando a concentração no 100. Seguindo a mesma lógica da fase 2 da soma.

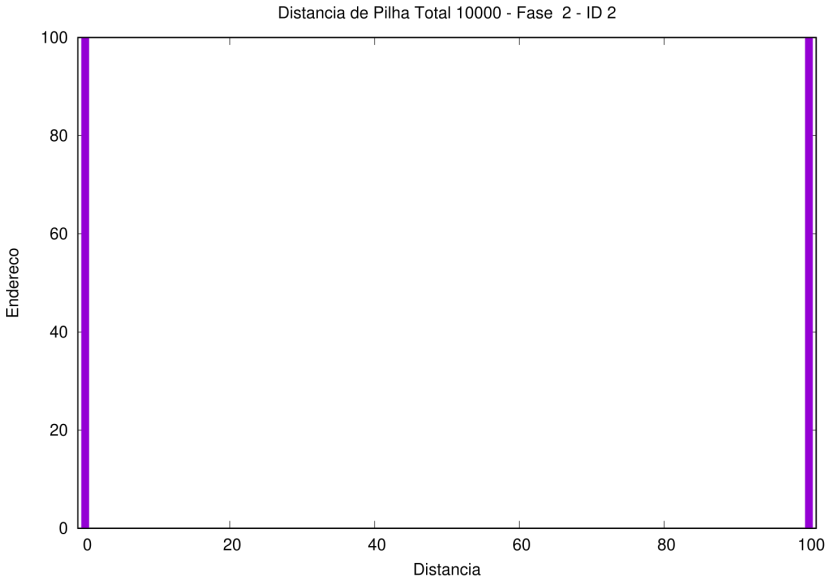


Figura 27 - Histograma Matriz C(resultante) na multiplicação - Fase 2

6.2.3. Transposição

6.2.3.1. Fase 0

Seguindo a lógica das últimas duas operações com matrizes a matriz *A* é inicialmente inicializada com valor 0, como é o primeiro acesso à matriz não há distância de pilha. Posteriormente, a matriz é inicializada e o primeiro valor a ter o valor 0 atribuído é o último na pilha, neste caso,

possui distância igual a 100. Após a inicialização do primeiro elemento o segundo a ser atribuído o 0, porém passou a ser o último após o acesso do primeiro elemento, e assim sucessivamente até todos serem atribuídos ao valor esperado.

A matriz resultante é apenas inicializada com 0 nessa primeira fase, portanto não há distância de pilha definida.

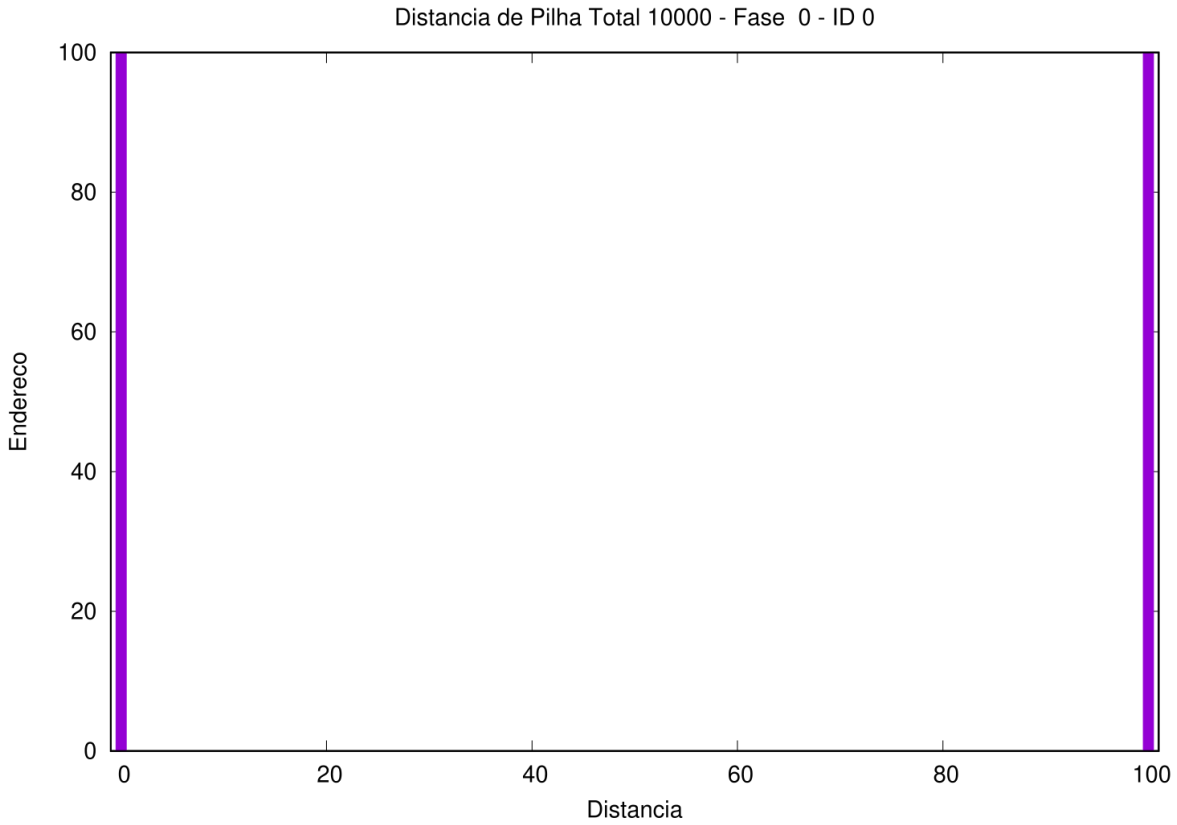


Figura 28 - Histograma Matriz A na transposição- Fase 0

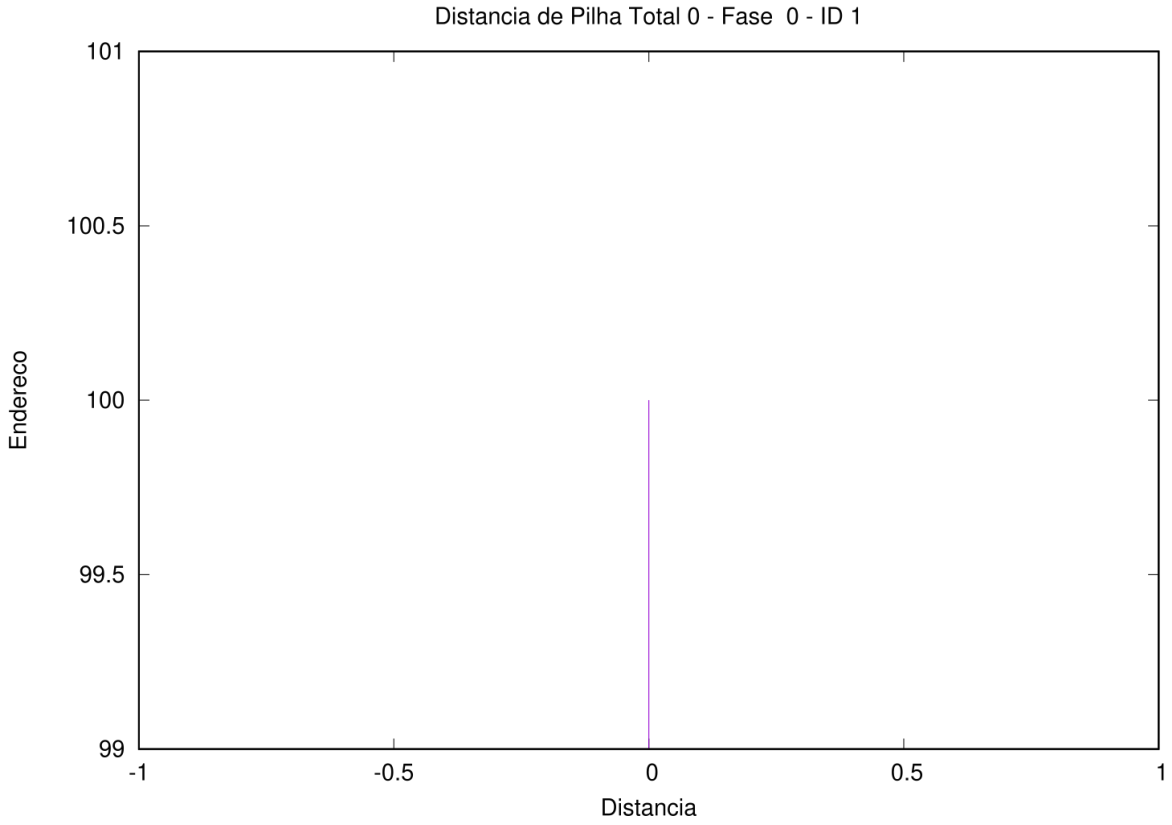


Figura 29 - Histograma Matriz B(resultante) na transposição - Fase 0

6.2.3.2. Fase 1

A matriz A, possui um comportamento distinto, visto que acessamos-a por colunas. Após sua inicialização, ao acessar seus elementos, temos um comportamento não uniforme, como pode ser visto no gráfico, isso porque o primeiro elemento da primeira coluna foi o primeiro a ser inicializado e por isso possui distância de pilha igual a 100, no primeiro momento. Posteriormente, acessamos o segundo elemento da primeira coluna, sendo ele o nonagésimo elemento da pilha, porém, com o acesso ao primeiro ele passa a ser o nonagésimo primeiro, o terceiro passa de octogésimo à octogésimo segundo causando um acesso mais irregular. Similar a matriz B da operação de multiplicação de matrizes

Já a matriz resultante B é acessada sequencialmente e por linhas, mantendo a organização 0 e 100, assim como as matrizes ao serem inicializadas.

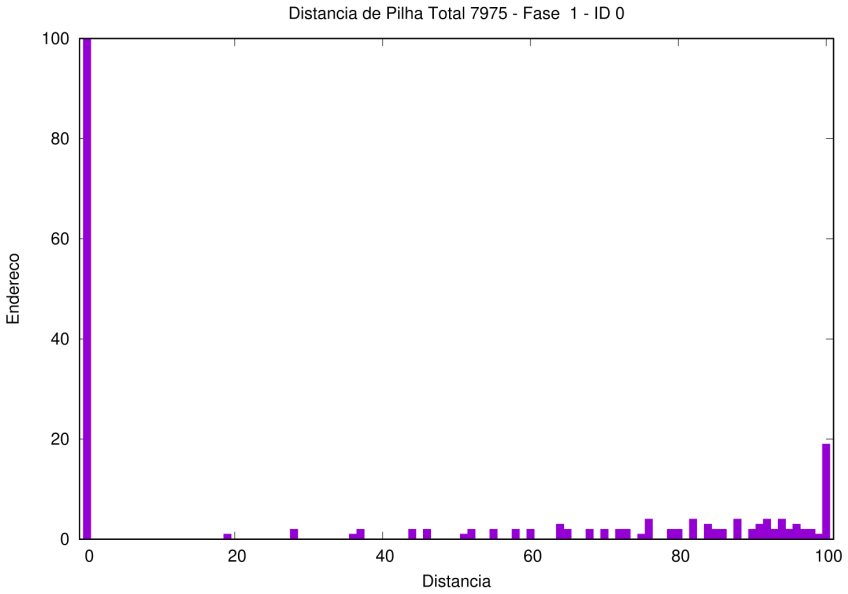


Figura 30 - Histograma Matriz A na transposição - Fase 1

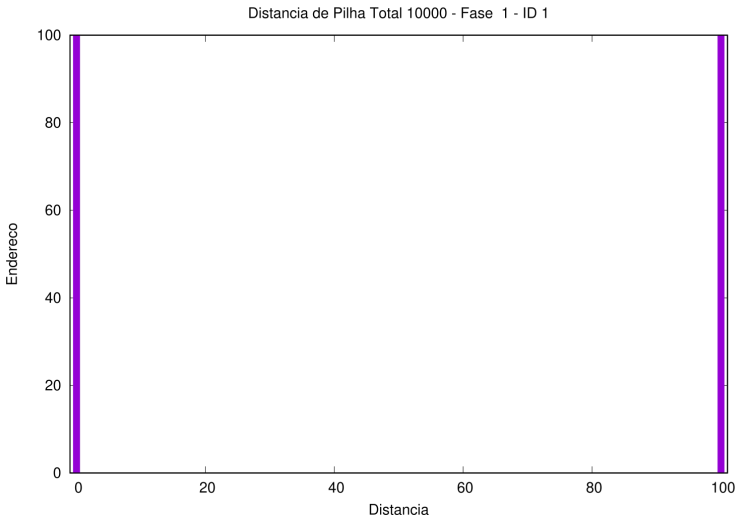


Figura 31 - Histograma Matriz B(resultante) na transposição - Fase 1

6.2.3.3. Fase 2

Por fim, na última fase, temos operações apenas com a matriz resultante, nessa ela é acessada e posteriormente impressa no arquivo resultante, imprimindo o primeiro elemento acessado primeiro, ou seja, o último da pilha seguindo a mesma lógica de acesso das fases passadas, gerando a concentração no 100.

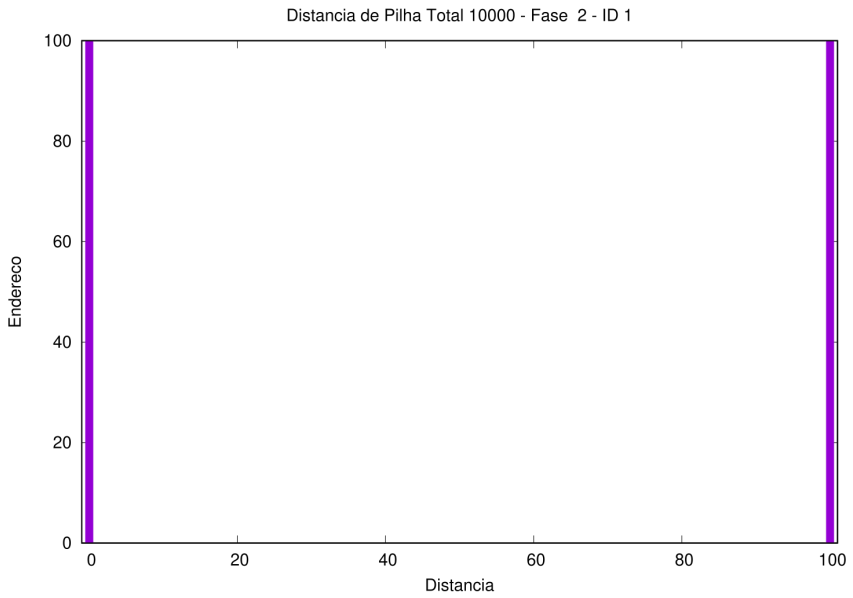


Figura 32 - Histograma Matriz B(resultante) na transposição - Fase 2

6.3. Tempo de execução

Conforme citado na parte dos testes as análise do tempo de execução foi afetada pelo limite baixo de operações de multiplicação fornecidas pelo computador utilizado, nesse sentido todos os tempos foram inferiores a 0.1 seg, tempo relativamente baixo que pode sofrer com execuções externas, gerando resultados não tão confiáveis.

Entretanto, a partir dos tempos de execução do programa para diferentes matrizes e operações, podemos recolher algumas informações. Tais dados foram obtidos a partir da ferramenta gprof, os resultados foram transformados nesse gráfico abaixo.

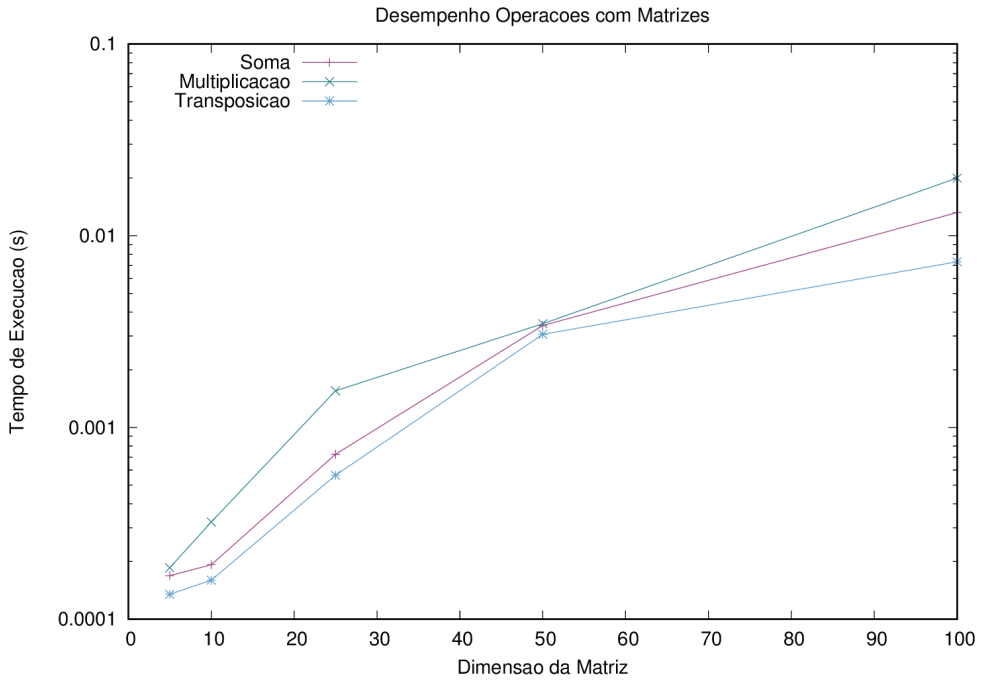


Figura 33 - Tempo de execução x Dimensão da matriz para diferentes operações

1. A multiplicação, como esperando, demanda mais tempo de execução que as demais operações;
2. Quanto maior a entrada n, maior o custo de tempo para execução do problema;

3. O custo de execução para pequenas matrizes é afetada por outros programas em execução, podendo gerar algumas falhas, como podemos ver um decréscimo do tempo esperado na operação de multiplicação para matrizes 50x50.

7. Conclusão

Após a implementação do programa, pode-se notar que havia duas partes distintas para o desenvolvimento do trabalho. A primeira era implementar as operações em matrizes alocadas dinamicamente, e as estruturas do código utilizado. A segunda era utilizar programas para avaliação do desempenho e corretude do código, com base na análise do desempenho de tempo, do uso da memória e do funcionamento do algoritmo para diferentes entradas.

A grande dificuldade encontrada foi na implementação do código, por haver diversos programas a serem utilizados, tive dificuldade na debugação quando havia problemas na compilação.

Após o funcionamento do código, a incapacidade de fazer testes com grandes entradas acabou gerando frustrações durante o desenvolvimento do trabalho, diversos fatores vieram à tona após a descoberta desse entrave, porém nem a análise gerada pelo computador me permitiu visualizar o local onde poderia estar originando tal problema. Visto que, o analisamem estava a utilizar 12% da CPU e haviam poucos programas que utilizam tanto da CPU funcionando em paralelo, entretanto, o problema se mantinha.

Porém, ao fim desse trabalho, ficou ainda mais evidente a importância da utilização das ferramentas de análise de desempenho e memória de programas. Tal prática possibilitou fixar mais sobre utilização de alocações dinâmicas, utilizações de TAD's e matrizes. Além disso, difere-se a perspectiva de apenas programar para resolver problemas, para programar com o objetivo de encontrar a solução mais eficaz para os problemas, entendendo a implementação e seu comportamento na memória e em custo de tempo.

8. Bibliográfica

"TAD: tipos abstratos de dados" - decom-ufop

Ziviani, N. (2006). Projetos de Algoritmos com Implementacões em Pascal e C

Cormen, T. Leiserson, C, Rivest, R., Stein, C.(2009) Introduction to Algorithms

Aulas e materiais disponibilizados no moodle UFMG.

Instruções para compilação e execução

1. Necessário arquivos teste: durante a execução do programa é necessário passar arquivos que possuem as matrizes, no Makefile essas implementações estão em uma pasta chamada test. A nomenclatura dos arquivos que possuem as matrizes do Makefile são m1.txt para a matriz 1 e m2.txt para a matriz 2.
2. Analisamem - no mesmo terminal onde está a pasta TP, ou seja, onde estão as implementações do trabalho prático, deve haver a pasta com o analisamem cedido pelo professor Wagner Meira no moodle na parte Estrutura de Dados - Metaturma → Material Adicional.
3. Pastas com resultados - Todos os dados resultantes são alocados a pastas específicas na pasta /tmp do computador, sua divisão está feita de acordo com a ferramenta utilizada para ele e de quais operações são tais dados.
4. Compilação - Foi utilizado o G++ para compilar os códigos