LABORATÓRIO II
BIBLIOTECA OPENACC

Grupo: **Not Found**

Julia De Souza Dos Santos - 238166
Matheus Sales Oliveira - 203577

SUMÁRIO

1. PROBLEMA

1.1. DEFINIÇÃO

Dadas três matrizes A_{yxw}, B_{wxv}, C_{vx1}, calcule a matriz D_{yx1} tal que D_{yx1}= (A_{yxw} X B_{wxv}) X C_{vx1}. Além disso, calcule a redução pela soma dos elementos na matriz D_{yx1}, isto é, a soma de todos os elementos em D_{yx1}.

1.2. REQUISITOS

- ✓ O programa deve ser escrito em linguagem C e no ambiente Linux.
- ✓ O código fonte deve estar disponível em um repositório no GitHub.
- ✓ O problema deverá ser resolvido utilizando a biblioteca OpenACC.
- ✓ As dimensões das matrizes são definidas pelas variáveis w, v e y (informadas pela linha de comando).
- ✓ Os dados das matrizes devem ser números reais, com até duas casas decimais, entre -10 e +10. Esses números devem ser gerados aleatoriamente.
- ✓ As matrizes devem ser alocadas dinamicamente em uma única etapa.
- ✓ Os valores de cada matriz devem estar armazenados sequencialmente em um arquivo do tipo “dat”.
- ✓ O programa deve ser executado da seguinte forma através do terminal:

```
./programa y w v arqA.dat arqB.dat arqC.dat arqD.dat
```

Onde:

- *./programa* → nome do programa que realiza a multiplicação das matrizes
- *y* → número de linhas da matriz A.
- *w* → número de colunas da matriz A e número de linhas da matriz B.
- *v* → número de colunas da matriz B e número de linhas da matriz C.
- *arqA.dat* → nome do arquivo que contém a matriz A.
- *arqB.dat* → nome do arquivo que contém a matriz B.
- *arqC.dat* → nome do arquivo que contém a matriz C.
- *arqD.dat* → nome do arquivo que contém a matriz resultante B.

2. OBJETIVO

O objetivo deste trabalho é avaliar a influência da paralelização de um programa sobre o seu tempo de execução. Nesse cenário, desenvolveu-se um programa na linguagem C que propõe uma solução para o problema apresentado acima utilizando a paralelização. Desse modo, o paralelismo foi implementado utilizando a biblioteca OpenACC, que possibilita a criação de regiões paralelas através do uso de diretivas de compilação para processamento em GPUs.

3. PROGRAMA

O programa desenvolvido para esta aplicação apresenta uma estrutura semelhante ao programa desenvolvido no laboratório 1. As principais diferenças deste em relação ao programa anterior são: retirada da função *gerar_valores_aleatorios* (para esse laboratório, a criação dos arquivos de entrada com valores aleatórios é feita por outro programa chamado: arquivos_aleatorios.c), inclusão de uma condição que verifica a existência dos arquivos passados como parâmetro na linha de comando e, por fim, adaptação das funções *multiplicar_matrizes* e *reducao_pela_soma* para a biblioteca OpenACC.

Dessa forma, o novo programa é composto por 5 funções e, dentre elas, apenas duas fazem o uso de paralelismo, visto que são as responsáveis pelas principais funcionalidades do software: *multiplicar_matrizes* e *reducao_pela_soma*. As outras 3 funções são responsáveis por gerar as matrizes e o arquivo de saída e pela coordenação de todas as funcionalidades (*main*).

a) `multiplicar_matrizes`: essa função é responsável por multiplicar duas matrizes e armazenar o resultado dessa multiplicação em uma matriz auxiliar que será retornada pela função. Nela criou-se uma região paralela antes do primeiro laço *for* e utilizou-se a diretiva `collapse(2)` para combinar os dois laços aninhados mais próximos a fim de obter um maior paralelismo.

   Dentro dessa região paralelizada, após estes dois primeiros laços, foi utilizado a diretiva `reduction(+:soma)` para possibilitar que as *gangs* que executam esse trecho de código consigam acessar e atualizar o valor da variável *soma* simultaneamente.

```
float* multiplicar_matrizes(float *matA, float *matB, unsigned int linA, unsigned int colA, unsigned int linB, unsigned int colB){
    float *matAux = (float *) malloc(linA * colB * sizeof(float));

    register unsigned int linha, coluna, i;
    register float soma=0;

    CRIA A REGIÃO PARALELA
    #pragma acc parallel loop collapse(2)
    for(linha=0; linha<linA; linha++){
        for(coluna=0; coluna < colB; coluna++){
            soma=0;
            #pragma acc loop reduction(+:soma)
            for(i=0; i < linB; i++){
                soma += matA[linha*colA + i] * matB[i*colB + coluna];
            }
            matAux[linha*colB + coluna] = soma;
        }
    }

    return matAux;
}
```

b) `reducao_pela_soma`: essa função é responsável por calcular a redução pela soma dos elementos armazenados na matriz resultante D_{yx1}. Para isto, a função recebe como parâmetros a matriz D_{yx1}, seu número de linhas e seu número de colunas e utiliza estes dados em um laço de repetição do tipo *for* aninhado para somar cada elemento da matriz a uma variável auxiliar chamada *soma*. Por fim, o programa retorna o valor armazenado nesta variável após o fim dos laços.

Nessa função foi utilizado o paralelismo no segundo for aninhado, onde as *gangs* acessam a variável *soma* e adicionam a ela o elemento armazenado na respectiva posição da matriz. Para isso, utilizou-se a cláusula `#pragma acc parallel loop reduction(+:soma)` que permitirá que as *gangs* que executam esse trecho do código realizem a operação de adição sobre a variável *soma* de forma simultânea.

```
double reducao_pela_soma(float *matriz, unsigned int linM, unsigned int colM){
    register int linha, coluna;
    register double soma = 0;

    for(linha=0; linha < linM; linha++){
        #pragma acc parallel loop reduction(+:soma)
        for(coluna=0; coluna < colM; coluna++){
            soma += matriz[linha*colM + coluna];
        }
    }

    return soma;
}
```

c) `gerar_matriz`: essa função é responsável por ler o arquivo que contém todos os elementos da matriz e armazenar cada um dos dados escritos nele na matriz alocada dinamicamente. Essa função sofreu uma alteração em relação ao código

desenvolvido no laboratório 1. Considerando o feedback do professor sobre a entrega anterior e visando aprimorar a qualidade do software, foi adicionado uma condição nessa função que verifica se o arquivo de entrada passado como parâmetro existe e, caso contrário, é enviado uma mensagem ao usuário e o programa é finalizado sem realizar as demais funções.

```
float *gerar_matriz (char *nomedoArquivo, float *matriz, unsigned int linM, unsigned int colM){
    FILE  *arquivo = fopen(nomedoArquivo,"r");

    if(arquivo == NULL){
        printf("O arquivo %s nao existe!\n", nomedoArquivo);
        fclose(arquivo);
        exit(0);
    }

    register unsigned int linha, coluna;

    for (linha=0; linha<linM;linha++){
        for (coluna=0; coluna<colM;coluna++){
            fscanf(arquivo,"%f",&matriz[linha*colM +coluna]);
        }
    }

    fclose(arquivo);

    return matriz;
}
```

d) `gravar_matriz`: essa função é responsável por escrever o resultado da multiplicação das matrizes no arquivo de saída. Para isso, ela recebe como parâmetro o nome do arquivo que será criado, a matriz que será gravada no arquivo e o número de linhas e colunas desta matriz. Assim como a função anterior, a gravar_matriz também sofreu algumas alterações. Como o arquivo é aberto no modo “w”, não faria sentido verificar se o arquivo existe pois esse modo de abertura cria o arquivo caso ele não exista (lembrando que: caso já exista algum arquivo com esse nome, o arquivo existente será substituído). Dessa forma, adicionamos apenas uma verificação para o caso de ocorrer algum outro problema com o arquivo que o impeça de ser criado.

```
void gravar_matriz(char *nomeArquivo, float *matriz, unsigned int linM, unsigned int colM){
    FILE *arquivo = fopen(nomeArquivo, "w");

    if(arquivo == NULL){
        printf("Erro ao criar arquivo: %s\n", nomeArquivo);
        fclose(arquivo);
        exit(0);
    }

    register unsigned int linha, coluna;

    for (linha=0; linha<linM; linha++){
        for (coluna=0; coluna<colM;coluna++){
            fprintf(arquivo,"%.2f\n", matriz[linha*colM +coluna]);
        }
    }

     fclose(arquivo);
}
```

e) `main`: essa função é responsável por coordenar todas as demais funções presentes no programa. Nela são feitas as alocações dinâmicas de todas as matrizes, os cálculos do tempo de execução das funções multiplicar_matrizes e reducao_pela_soma e a criação e preenchimento das matrizes utilizando arquivos de dados externos. Além disso, é onde as funções que fazem uso da paralelização são chamadas.

Em relação aos detalhes dessa função, o nosso programa faz a alocação de dinâmica de 5 matrizes: matrizA, matrizB, matrizC, matrizD e matrizAB. Essa ultima é alocada com o mesmo número de linhas da matrizA e o mesmo número de colunas da matrizB e gerada a partir da multiplicação das matrizes A e B. Posteriormente, essa matriz AB é multiplicada com a matrizC para gerar a resultante matrizD. A única alteração necessária nessa função foi a retirada das chamadas da função *gerar_valores_aleatorios*.

4. RESULTADOS E CONCLUSÃO

Para avaliar como a paralelização de um programa utilizando a biblioteca openACC influencia no seu desempenho, foram realizados 5 testes com o código desenvolvido. Em cada teste, foi atribuído o mesmo valor para as variáveis y, w, v que representam os números de linhas e colunas de cada matriz. Após definido os valores, o mesmo código foi executado porém, na primeira execução, não considerou-se o paralelismo (compilado com o comando: `gcc -o programa programa.c`). Já na segunda execução, a paralelização foi considerada (compilado com o comando: `gcc -o programa` **`-fopenacc`**

`programa.c`). Para cada execução, os respectivos valores do tempo de execução foram registrados na tabela abaixo.

y	w	v	Tempo de Execução (sem paralelização)	Tempo de Execução (com paralelização)	Diferença % dos Tempos de Execução
10	10	10	0,000004	0,000014	250,00%
100	100	100	0,002190	0,003519	60,68%
1000	1000	1000	2,581925	3,595931	39,27%
1500	1500	1500	11,147819	15,190705	36,27%
2000	2000	2000	29,854409	37,267058	24,83%
					40,26%

Figura 1 - Tabela de Análise

Após o registro dos tempos de execução, foi calculado a diferença percentual entre os tempos obtidos executando o programa sem paralelização e executando o programa com paralelização. Esse cálculo percentual foi feito utilizando-se a seguinte fórmula:

$$\left(\frac{Tempo\ de\ Execução\ com\ Paralelização - Tempo\ de\ Execução\ sem\ Paralelização}{Tempo\ de\ Execução\ sem\ Paralelização} \right) \times 100$$

Após as execuções, foi possível notar através da avaliação da tabela de análise e do cálculo apresentado acima que, para todos os valores de y, w e v, o programa paralelizado levou um tempo maior para executar do que o programa não paralelizado. Realizando uma média entre a diferença percentual dos tempos de execução considerando os valores de y, w e v maiores que 10, percebe-se que o programa não paralelizado executou as mesmas operações 40,26% mais rápido do que o programa paralelizado. Porém, apesar disso, pode-se notar também que quanto maior os valores para y, w e v menor é a diferença percentual entre os programas.

Abaixo, encontram-se três gráficos que demonstram essa diferença entre os tempos de execução. Como há uma diferença considerável entre os tempos de execução para y=w=v=10 e y=w=v=100 em relação aos demais valores, foram criados gráficos exclusivos para esses valores de x, y e w e, posteriormente, foi criado um gráfico que inclui todos os valores testados.

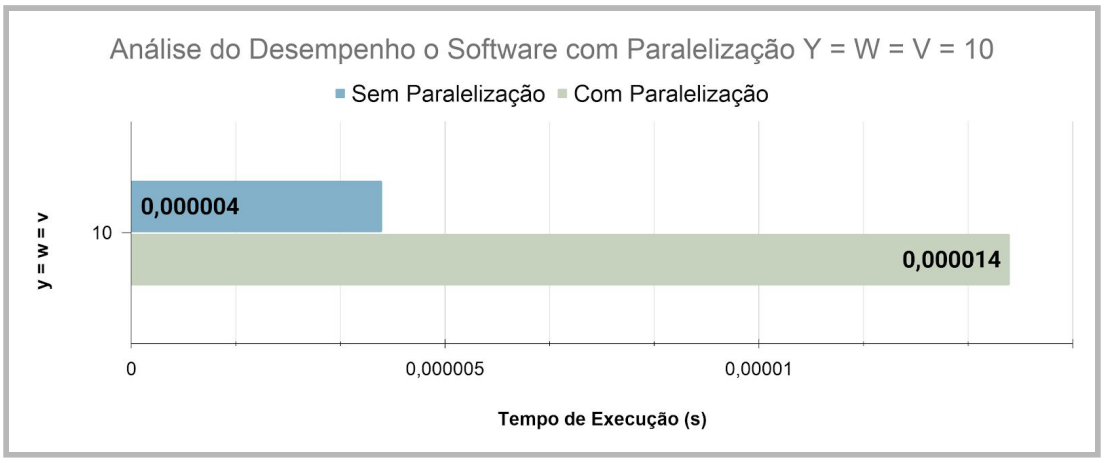

Figura 2 - Gráfico de desempenho para y = w = v =10

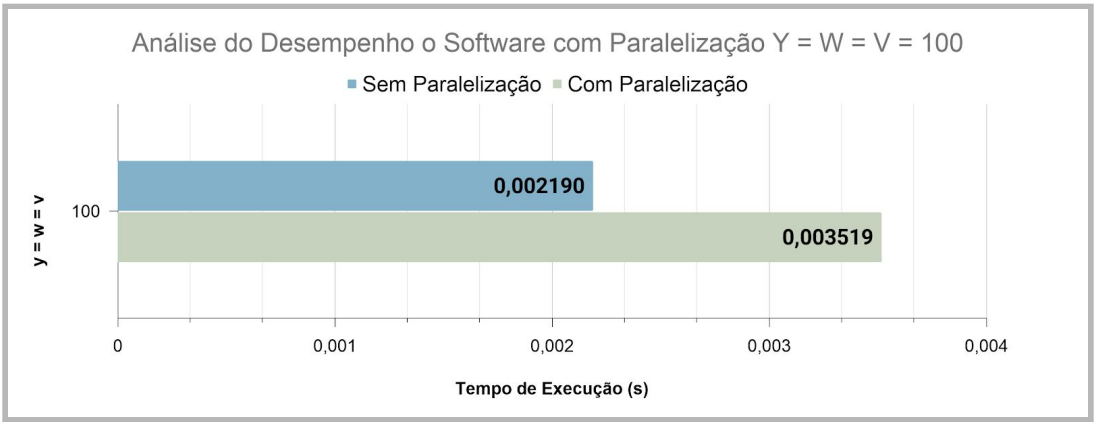

Figura 3 - Gráfico de desempenho para y = w = v =100

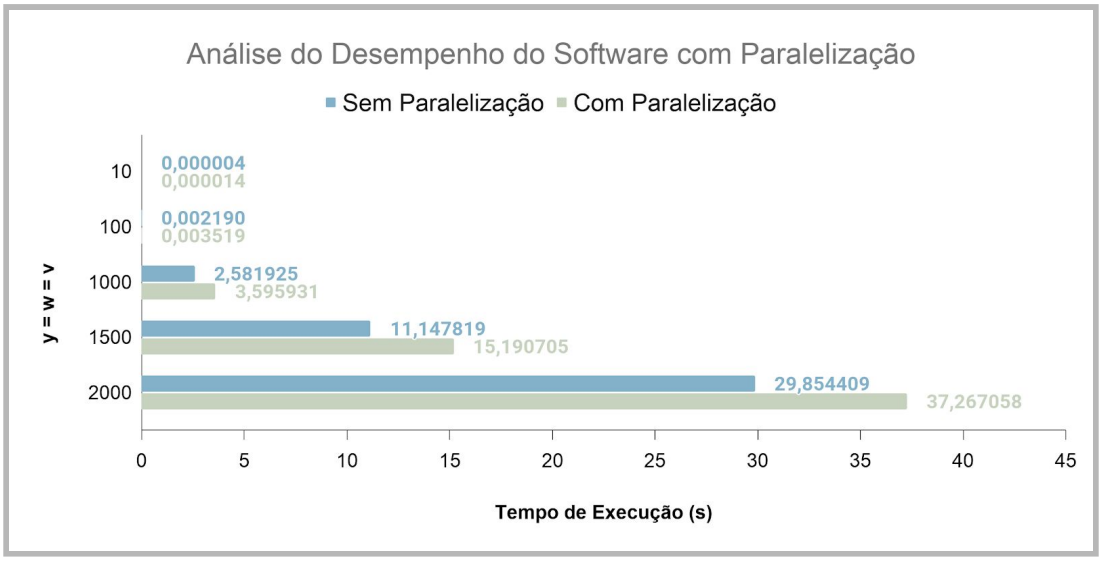

Figura 4 - Gráfico de desempenho para todos os valores de y, w e v

É interessante notar que, se compararmos as duas bibliotecas utilizadas até agora: openMP e openACC, a openMP obteve um resultado muito mais satisfatório uma vez que ela executou o programa que realiza as mesmas funções duas vezes mais rápido que a abordagem serial e, portanto, mais rápido que a abordagem utilizando openACC. Esse fato nos levou a tentar outras duas abordagens com a biblioteca openACC: a primeira delas utilizando as diretivas *copyin* e *copyout* e a outra utilizando a diretiva *tile*. Nas imagens abaixo é possível ver os códigos com essas abordagens e o respectivo tempo de execução que cada abordagem para y, w e v igual a 10, 100 e 1000:

“Essa primeira abordagem visava utilizar as diretivas copyin e copyout para controlar o movimento de dados entre a CPU e a GPU e, assim, melhorar o tempo de execução. Então, utilizamos o copyin para copyout as matrizes A e B na GPU e, uma vez que as operações delas fossem feitos na GPU dentro a região paralela, acreditávamos que obteríamos um menor tempo de execução. Mas, como podemos observar na imagem abaixo, não foi o que ocorreu”

```
float* multiplicar_matrizes(float *matA, float *matB, unsigned int linA, unsigned int colA, unsigned int linB, unsigned int colB){
    float *matAux = (float *) malloc(linA * colB * sizeof(float));

    register unsigned int linha, coluna, i;
    register float soma=0;

    #pragma acc parallel loop collapse(2) copyin(matA[0:linA*colA], matB[0:colA*colB]) copyout(matAux[0:linA*colB])
    for(linha=0; linha<linA; linha++){
        for(coluna=0; coluna < colB; coluna++){
            soma=0;
            #pragma acc loop reduction(+:soma)
            for(i=0; i < linB; i++){
                soma += matA[linha*colA + i] * matB[i*colB + coluna];
            }
            matAux[linha*colB + coluna] = soma;
        }
    }

    return matAux;
}
```

```
[j238166@ip-172-31-22-24 ~]$ ./programa 10 10 10 arqA.dat arqB.dat arqC.dat arqD.dat
As operacoes com matrizes levaram 0.000012 segundos para executar
-2294.055634
```

```
[j238166@ip-172-31-22-24 ~]$ ./programa 100 100 100 arqA.dat arqB.dat arqC.dat arqD.dat
As operacoes com matrizes levaram 0.003517 segundos para executar
-650871.024109
```

```
[j238166@ip-172-31-22-24 ~]$ ./programa 1000 1000 1000 arqA.dat arqB.dat arqC.dat arqD.dat
As operacoes com matrizes levaram 3.625941 segundos para executar
-119736106.847641
```

“Essa segunda abordagem visava utilizar a diretiva tile para que os dados fossem localizados com mais facilidade e para que os blocos fossem executados simultaneamente, visando melhorar o tempo de execução. Mas, apesar de se mostrar mais rápido que a abordagem anterior, ainda não foi mais rápido que a abordagem serial.”

```
float* multiplicar_matrizes(float *matA, float *matB, unsigned int linA, unsigned int colA, unsigned int linB, unsigned int colB){
    float *matAux = (float *) malloc(linA * colB * sizeof(float));

    register unsigned int linha, coluna, i;
    register float soma=0;

    #pragma acc parallel loop tile(2,2)
    for(linha=0; linha<linA; linha++){
        for(coluna=0; coluna < colB; coluna++){
            soma=0;
            #pragma acc loop reduction(+:soma)
            for(i=0; i < linB; i++){
                soma += matA[linha*colA + i] * matB[i*colB + coluna];
            }
            matAux[linha*colB + coluna] = soma;
        }
    }

    return matAux;
}
```

```
[j238166@ip-172-31-22-24 ~]$ ./programa 10 10 10 arqA.dat arqB.dat arqC.dat arqD.dat
As operacoes com matrizes levaram 0.000013 segundos para executar
-2294.055634
```

```
[j238166@ip-172-31-22-24 ~]$ ./programa 100 100 100 arqA.dat arqB.dat arqC.dat arqD.dat
As operacoes com matrizes levaram 0.003646 segundos para executar
-650871.024109
```

```
[j238166@ip-172-31-22-24 ~]$ ./programa 1000 1000 1000 arqA.dat arqB.dat arqC.dat arqD.dat
As operacoes com matrizes levaram 3.789651 segundos para executar
-119736106.847641
```

Assim, baseando-se nos tempos de execução obtidos, nos dados apresentados na tabela, na análise dos gráficos e nos testes realizados com diferentes abordagens, concluiu-se que a paralelização do programa desenvolvido utilizando a biblioteca openACC não apresentou vantagem, uma vez que o tempo de execução obtido foi maior para o programa paralelizado do que para o programa serial. Um dos motivos para isso seria que a transferência de dados entre a CPU e GPU está consumindo muito tempo, o que anula a vantagem ocasionada pela paralelização.

5. TUTORIAIS

5.1. COMPILANDO

Há várias formas de compilar um programa escrito em linguagem C, abaixo encontram-se algumas destas formas.

5.1.1 COMPILANDO VIA TERMINAL DO LINUX

Para compilar o programa via terminal do Linux usando o compilador GCC deve-se seguir os seguintes passos:

1. *Digite o comando abaixo no terminal do Linux para compilar o programa das matrizes:*

```
gcc -o programa -fopenacc programa.c
```

Onde:

- `gcc` → executa o compilador GCC.
- `-o programa` → define que o programa executável se chamará programa.
- `-fopenacc` → garante que a biblioteca OpenAcc será incluída no programa durante a compilação.
- `programa.c` → nome do arquivo fonte escrito em linguagem C que será utilizado para gerar o arquivo executável.

5.1.2. COMPILANDO VIA SCRIPT

Para automatizar o processo de compilação foi criado um script do tipo SH. O arquivo de script recebeu o nome de `script.sh` e, antes de executá-lo, é preciso **garantir que ele esteja na mesma pasta que o programa escrito em linguagem C**. Após isso, é necessário alterar a permissão de administrador ao script para garantir que não haja nenhuma falha durante a fase a compilação. Para conferir as devidas permissões ao script basta digitar e executar o código abaixo no terminal do Linux:

```
chmod +x  script.sh
```

Por fim, para executar o programa basta digitar e executar o seguinte código:

```
./script.sh
```

5.1.2. COMPILANDO VIA MAKEFILE

Para automatizar o processo de compilação foi criado um arquivo makefile. O arquivo de makefile recebeu o nome padrão `makefile` e, antes de executá-lo, é preciso **garantir que ele esteja na mesma pasta que o programa escrito em linguagem C**. Após isso, é necessário digitar e executar o seguinte comando no terminal do Linux para compilar o programa:

```
make
```

5.2. EXECUTANDO O PROGRAMA

1. *Execute o programa das matrizes utilizando os comandos abaixo:*

```
./programa y w v arqA.dat arqB.dat arqC.dat arqD.dat
```

Onde:

- `./programa` → o programa gerado anteriormente.
- `y` → número de linhas da matriz A.
- `w` → número de colunas da matriz A e número de linhas da matriz B.
- `v` → número de colunas da matriz B e número de linhas da matriz C.
- `arqA.dat` → nome do arquivo que contém os valores da matriz A_{yxw}.
- `arqB.dat` → nome do arquivo que contém os valores da matriz B_{wxv}.
- `arqC.dat` → nome do arquivo que contém os valores da matriz C_{vx1}.
- `arqD.dat` → nome do arquivo que contém os valores da matriz resultante D_{yx1}.

5.3. GERANDO ARQUIVOS ALEATÓRIOS

Para auxiliar na execução do programa principal, foi criado um segundo programa chamado “arquivos_aleatorios.c” que é responsável por gerar os arquivos com valores aleatórios. Para compilar o programa auxiliar, basta digitar os comandos abaixo no terminal do Linux:

```
gcc -o arquivos_aleatorios arquivos_aleatorios.c
```

Onde:

- `gcc` → executa o compilador GCC.
- `-o arquivos_aleatorios` → define que o programa executável se chamará *arquivos_aleatorios*.
- `arquivos_aleatorios.c` → nome do arquivo fonte escrito em linguagem C que será utilizado para gerar o arquivo executável.

Para executar o programa gerado anteriormente basta utilizar o código abaixo no terminal do Linux:

```
./arquivos_aleatorios y w v arqA.dat arqB.dat arqC.dat
```

Onde:

- `./arquivos_aleatorios` → o programa gerado anteriormente.
- `y` → número de linhas da matriz A.
- `w` → número de colunas da matriz A e número de linhas da matriz B.
- `v` → número de colunas da matriz B e número de linhas da matriz C.
- `arqA.dat` → nome do arquivo que contém os valores da matriz A_{yxw}.
- `arqB.dat` → nome do arquivo que contém os valores da matriz B_{wxv}.

- `arqC.dat` → nome do arquivo que contém os valores da matriz C_{vx1}.

O programa principal pode utilizar os arquivos com valores aleatórios gerados pelo programa auxiliar “arquivos_aleatorios.c” para preencher suas matrizes desde que:

- Os valores de `y,w,v` na execução do programa principal sejam iguais ao valores de `y,w,v` na execução do programa *arquivos_aleatorios.*
- O nome dos arquivos `arqA.dat, arqB.dat e arqC.dat` sejam iguais na execução do programa principal e na execução do programa auxiliar para que, dessa forma, o programa principal possa ler os dados armazenados nos arquivos gerado pelo programa auxiliar *arquivos_aleatorios.*

5.4. GITHUB

O código fonte escrito em Linguagem C assim como o arquivo makefile e o script SH encontram-se no seguinte repositório no GitHub na pasta(diretório) OpenACC:

https://github.com/matheusaleso/PAD

6. REFERÊNCIAS

GRADVOHL, André Leon Sampaio. Teste de alocação de matrizes. GitHub. Disponível em: <https://gradvohl.github.io/alocaMatrizes/>. Acesso em: 22 de out. de 2020