



TT005 - Tópicos Especiais em Telecomunicações I - Programação de Alto desempenho

Relatório do Laboratório de OpenMP

Vitor Roberto Kogawa de Moraes - 245582

Eric Satoshi Suzuki Kishimoto - 233974

Sumário

Introdução	3
Arquitetura de hardware e Sistemas Operacionais	3
Desenvolvimento	3
Dificuldades	6
Gráficos de Tempo de Execução	10
Multiplicação de matrizes 10x10	11
Multiplicação de matrizes 100x100	12
Multiplicação de matrizes 1000x1000	13
Instruções de Uso	13
Conclusão	14

Introdução

O presente trabalho tem como objetivo relatar os conhecimentos e experiências adquiridas pelos alunos durante o desenvolvimento de algoritmos paralelos utilizando a biblioteca OpenMP.

Arquitetura de hardware e Sistemas Operacionais

Para realização do laboratório de OpenMP, utilizamos uma máquina virtual com 6GB de memória RAM, 6 processadores (no programa foram utilizadas apenas 4 threads) com o sistema operacional Kali Linux. Também utilizamos uma máquina física com 8GB, 8 processadores com o sistema operacional Ubuntu instalado.

Desenvolvimento

Para a realização do laboratório, utilizamos alguns recursos disponibilizados pelo professor. Utilizamos a biblioteca MatrixIO para leitura de matrizes em arquivos e a escrita no arquivo de saída. Também utilizamos o programa generateRandomMatrix para realizar testes com nosso programa.

O laboratório consistia em basicamente realizar uma multiplicação de matrizes lidas de outros arquivos e paralelizar essa tarefa entre 4 threads. Para fazer isso, criamos um programa de multiplicação de matrizes. Entretanto, o laboratório exigia alguns requisitos. As matrizes deveriam estar alocadas em uma única etapa, então, o processo de acesso à matriz é um tanto diferente. Para realizar o acesso aos elementos das matrizes, precisamos criar uma diretiva de pré-processamento.

```
#define posicao(I, J, COLUNAS) ((I)*(COLUNAS) + (J))
```

Ela define que o elemento com posição i, j está localizado em $i * \text{num_col} + j$. Com isso, podemos acessar os elementos da matriz e realizar a multiplicação.

```
/* Verificando se é possível multiplicar as 2 matrizes */
if (col1 != lin2)
{
    printf("The number of matrix 1 columns is different to number of matrix 2 lines \n");
    exit(EXIT_FAILURE);
}
```

Para realizar a multiplicação, a primeira coisa a fazer foi verificar se o número de colunas da primeira matriz é igual ao número de linhas da segunda matriz. Se forem diferentes, uma mensagem de erro será disparada.

Logo após, realizamos a multiplicação propriamente dita. Para isso, criamos 3 laços for aninhados, 1 para percorrer as linhas da matriz 1, um para percorrer as colunas da matriz 2 e outro para somar as multiplicações da matriz 1 pela 2. Paralelizamos essa região pois ela era a que demandava maior processamento e, com isso, as threads tiveram uma carga de trabalho bem definida. veremos o porquê nos gráficos de tempo de execução.

```
#pragma omp parallel for num_threads(4) private(i, j, k)
// Percorre as linhas da matriz1
for (i = 0; i < lin1; i++)
{
    // Percorre as colunas da matriz2
    for (j = 0; j < col2; j++)
    {
        matrix3[posicao(i, j, col3)] = 0;
        // Somatório de cada elemento da multiplicação de matrizes
        for (k = 0; k < col1; k++)
        {
            matrix3[posicao(i, j, col3)] += (matrix1[posicao(i, k, col1)] * matrix2[posicao(k, j, col2)]);
        }
    }
}
```

Também criamos uma função para realizar a redução pela soma. Não colocamos ela junta da multiplicação para modularizar o código.

```
double reductionSum(int lin, int col, float *matrix)
{
    float sum = 0;

    #pragma parallel for num_threads(4) reduction(+:sum)
    // Itrando pelos elementos da matriz
    for (int i = 0; i < lin*col; i++)
    {
        sum += matrix[i];
    }

    return sum;
}
```

Na main, usamos os argumentos padrão da linguagem C `argc` e `argv` para ler dados diretamente do terminal como demandado pelo professor. Verificamos se a quantidade corresponde a quantidade de argumentos necessários para realizar o processamento. Se não, uma mensagem de como o programa deve ser usado será disparada para o usuário.

```
// Tratamento de erro de argumentos fornecidos pelo terminal
if (argc < 8)
{
    printf("Uso:\n");
    printf("%s <numero de linhas da matriz A> <numero de linhas da matriz B> <numero de linhas da matriz C> <arquivo A> <arquivo B> <arquivo C> <arquivo D>\n", argv[0]);
}
```

Se houverem argumentos para o processamento, os números de linhas das matrizes são convertidos em inteiros e verificados se são nulos ou negativos. Caso, sejam, uma mensagem de erro será disparada.

```
// argv: ./main y w v arqA.dat arqB.dat arqC.dat arqD.dat
int y = atoi(argv[1]);
int w = atoi(argv[2]);
int v = atoi(argv[3]);
```

Caso tudo esteja de acordo, as matrizes são lidas utilizando funções disponibilizadas pelo professor.

```
float *matrix1 = readMatrixFloat(y, w, argv[4]);
float *matrix2 = readMatrixFloat(w, v, argv[5]);
float *matrix3 = readMatrixFloat(v, 1, argv[6]);
```

Após isso, as 2 multiplicações são realizadas medindo-se o tempo de execução com a função `clock()`.

```
clock_t begin = clock(); // Calculando tempo de execução da multiplicação

float *matrixRes = multMatrix(y, v, multMatrix(y, w, matrix1, w, v, matrix2), v, 1, matrix3); // Multiplicação de matrizes

clock_t end = clock(); // Calculando tempo de execução da multiplicação
```

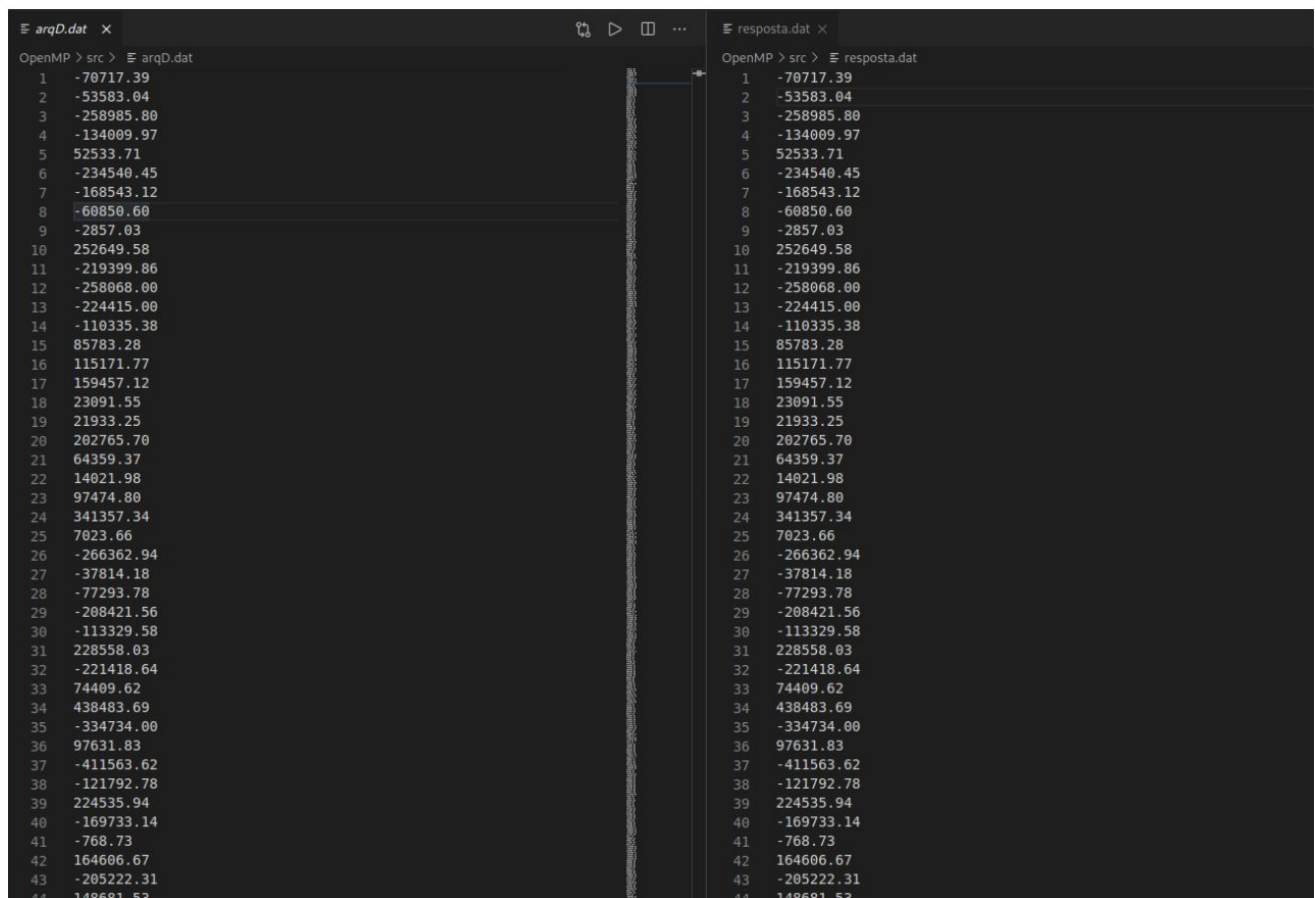
Depois disso, o tempo de execução e a redução pela soma são imprimidos no terminal e a matriz resultante da multiplicação é escrita no arquivo de saída informado pelo usuário.

Dificuldades

Após chegar ao programa executando corretamente, enfrentamos algumas dificuldades. Utilizamos os arquivos de testes disponibilizados pelo professor para rodar o programa. Ao compararmos os 2 programas usando o comando diff, o arquivo de resposta e o arquivo de saída do programa mostravam diferenças em quase todos os valores, porém, as diferenças eram pequenas entre os 2 arquivos.

arqD.dat	resposta.dat
1 -70717.39	1 -70717.39
2 -53582.76	2 -53583.04
3 -258985.93	3 -258985.80
4 -134009.84	4 -134009.97
5 52533.78	5 52533.71
6 -234540.46	6 -234540.45
7 -168542.90	7 -168543.12
8 -60850.54	8 -60850.60
9 -2857.07	9 -2857.03
10 252649.53	10 252649.58
11 -219399.64	11 -219399.86
12 -258068.28	12 -258068.00
13 -224414.87	13 -224415.00
14 -110335.18	14 -110335.38
15 85783.38	15 85783.28
16 115171.68	16 115171.77
17 159457.04	17 159457.12
18 23091.53	18 23091.55
19 21933.05	19 21933.25
20 202765.94	20 202765.70
21 64359.49	21 64359.37
22 14021.99	22 14021.98
23 97474.88	23 97474.80
24 341357.01	24 341357.34
25 7023.66	25 7023.66
26 -266363.17	26 -266362.94
27 -37814.16	27 -37814.18
28 -77293.70	28 -77293.78
29 -208421.67	29 -208421.56
30 -113329.58	30 -113329.58
31 228557.92	31 228558.03
32 -221418.76	32 -221418.64
33 74409.56	33 74409.62
34 438483.55	34 438483.69
35 -334733.90	35 -334734.00
36 97631.80	36 97631.83
37 -411563.54	37 -411563.62
38 -121792.70	38 -121792.78
39 224535.94	39 224535.94
40 -169733.16	40 -169733.14
41 -768.70	41 -768.73
42 164606.46	42 164606.67
43 -205221.99	43 -205222.31
44 148681.45	44 148681.53

Inicialmente, pensamos que o algoritmo poderia estar errado, mas depois de alguns pequenos testes com pequenas matrizes no site <https://matrixcalc.org/pt/>, percebemos que o algoritmo estava certo. Então, utilizamos outra abordagem. Se as diferenças entre os 2 arquivos eram mínimas, poderia haver algum problema envolvendo imprecisão de ponto flutuante. Então, na biblioteca MatrixIO, mudamos os tipos de double para float para verificar se haveria alguma diferença. Após realizar essa mudança, o programa criou o arquivo de saída com os mesmos valores do arquivo de resposta do professor.



```
OpenMP > src > arqD.dat
1 -70717.39
2 -53583.04
3 -258985.80
4 -134009.97
5 52533.71
6 -234540.45
7 -168543.12
8 -60850.60
9 -2857.03
10 252649.58
11 -219399.86
12 -258068.00
13 -224415.00
14 -110335.38
15 85783.28
16 115171.77
17 159457.12
18 23091.55
19 21933.25
20 202765.70
21 64359.37
22 14021.98
23 97474.80
24 341357.34
25 7023.66
26 -266362.94
27 -37814.18
28 -77293.78
29 -208421.56
30 -113329.58
31 228558.03
32 -221418.64
33 74409.62
34 438483.69
35 -334734.00
36 97631.83
37 -411563.62
38 -121792.78
39 224535.94
40 -169733.14
41 -768.73
42 164606.67
43 -205222.31
44 148681.53

OpenMP > src > resposta.dat
1 -70717.39
2 -53583.04
3 -258985.80
4 -134009.97
5 52533.71
6 -234540.45
7 -168543.12
8 -60850.60
9 -2857.03
10 252649.58
11 -219399.86
12 -258068.00
13 -224415.00
14 -110335.38
15 85783.28
16 115171.77
17 159457.12
18 23091.55
19 21933.25
20 202765.70
21 64359.37
22 14021.98
23 97474.80
24 341357.34
25 7023.66
26 -266362.94
27 -37814.18
28 -77293.78
29 -208421.56
30 -113329.58
31 228558.03
32 -221418.64
33 74409.62
34 438483.69
35 -334734.00
36 97631.83
37 -411563.62
38 -121792.78
39 224535.94
40 -169733.14
41 -768.73
42 164606.67
43 -205222.31
44 148681.53
```

Entretanto, ao usar o comando `diff` no terminal e direcioná-lo para um arquivo de saída, percebemos que o comando ainda apontava diferenças entre os arquivos. Isso ocorreu pois havia espaços extras no arquivo de resposta, antes e depois de cada valor.

```
resposta.dat X
OpenMP > src > resposta
1 -70717.39
2 -53583.04
3 -258985.80
4 -134009.97
5 52533.71
```

Removemos esses espaços e ao executar novamente o comando diff, ele não apontou nenhuma diferença entre os 2 arquivos. Então, para comparar os 2 arquivos é necessário que o arquivo a ser comparado não possua nenhum espaço ou caractere desse tipo antes ou depois de cada linha.

```
eric@kali:~/Documentos/TT005-High_Performance_Computing/OpenMP/src$ ls
arqA_997x981.dat arqB_981x991.dat arqC_991x1.dat arqD.dat auxMatrix main main.c makefile multMatrix.h resposta.dat runTest.sh
eric@kali:~/Documentos/TT005-High_Performance_Computing/OpenMP/src$ diff resposta.dat arqD.dat
eric@kali:~/Documentos/TT005-High_Performance_Computing/OpenMP/src$
```

Então diferenças entre os arquivos de saída do programa e os de testes podem ocorrer à um problema de precisão de ponto flutuante, dado que o tipo float possui 4 bytes para armazenamento enquanto o tipo double possui 8 bytes. Isso pode gerar uma diferença significativa dependendo do tipo de programa que se cria.

Outra dificuldade foi medir o tempo de execução da multiplicação. Como dito anteriormente, utilizamos a função clock para medir o tempo de execução da multiplicação. Para verificar se o tempo estava correto, também utilizamos o comando time do linux e os tempos não correspondiam um com o outro. Então executamos com diferentes números de threads e percebemos que o número de threads influenciam no tempo do clock. Testamos com alguns números de threads e obtivemos os seguintes resultados

Com 2 threads

```
eric@kali:~/Documentos/TT005-High_Performance_Computing/OpenMP/src$ time ./main 997 981 991 arqA_997x981.dat arqB_981x991.dat arqC_991x1.dat arqD.dat
Time to multiply the matrices 18.99663200 s
Reduction sum: -4129376.75

real    0m10.948s
user    0m20.352s
sys     0m0.069s
```

Com 4 threads


```
eric@kali:~/Documentos/TT005-High_Performance_Computing/OpenMP/src$ time ./main 997 981 991 arqA_997x981.dat arqB_981x991.dat arqC_991x1.dat arqD.dat
Time to multiply the matrices 24.97373300 s
Reduction sum: -4129376.75

real    0m7.800s
user    0m26.347s
sys     0m0.065s
```

Com 8 threads

```
eric@kali:~/Documentos/TT005-High_Performance_Computing/OpenMP/src$ time ./main 997 981 991 arqA_997x981.dat arqB_981x991.dat arqC_991x1.dat arqD.dat
Time to multiply the matrices 32.07983700 s
Reduction sum: -4129376.75

real    0m6.929s
user    0m33.174s
sys     0m0.157s
```

Com isso, temos a seguinte tabela

Medição de tempo		
Número de threads	Clock	Time
2	18,9966	10,9480
4	24,9737	7,8000
8	32,0798	6,929
Média	25,3501	8,5590

Arredondando os valores, temos o seguinte

Medição de tempo		
Número de threads	Clock	Time
2	19	11
4	25	8
8	32	7
Média	25	9

Com isso, percebemos um padrão se formando. Ao que aparenta ser, o tempo mostrado pelo programa é multiplicado pelo número de threads. Então, para resolver o problema, dividimos o

tempo de execução da multiplicação pelo número de threads usadas. Com isso, obtivemos o tempo real de execução da multiplicação de matrizes. Ou seja, a função clock não possui uma integração boa com o OpenMP.

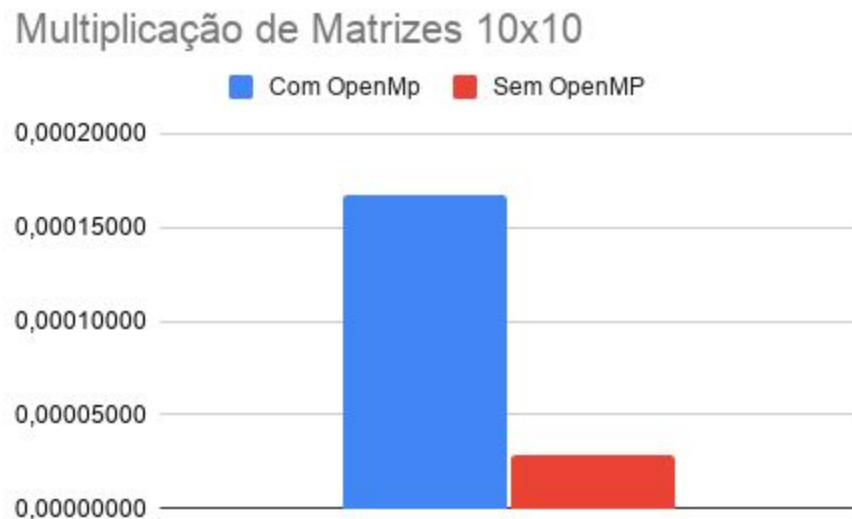
Gráficos de Tempo de Execução

Para plotar gráficos de tempo de execução, primeiramente, criamos scripts para gerar matrizes aleatórias 10x10, 100x100 e 1000x1000 com e sem o OpenMP. Desse modo podemos comparar o efeito que a adição de novas threads teve na execução. Para gerar os gráficos, criamos um script para pegar as saídas do programa e armazená-las em arquivos de texto.

<pre>time10.txt × ... OpenMP > tests > time10.txt 1 Time to multiply the matrices 0.00009650 s 2 Reduction sum: 153144.00 3</pre>	<pre>time10sem.txt × ... OpenMP > tests > time10sem.txt 1 Time to multiply the matrices 0.00002000 s 2 Reduction sum: 153144.00 3</pre>
<pre>time100.txt × ... OpenMP > tests > time100.txt 1 Time to multiply the matrices 0.00393525 s 2 Reduction sum: 134948577280.00 3</pre>	<pre>time100sem.txt × ... OpenMP > tests > time100sem.txt 1 Time to multiply the matrices 0.01375200 s 2 Reduction sum: 134948577280.00 3</pre>
<pre>time1000.txt × ... OpenMP > tests > time1000.txt 1 Time to multiply the matrices 6.77590675 s 2 Reduction sum: 123444386653536256.00 3</pre>	<pre>time1000sem.txt × ... OpenMP > tests > time1000sem.txt 1 Time to multiply the matrices 18.92443100 s 2 Reduction sum: 123444386653536256.00 3</pre>

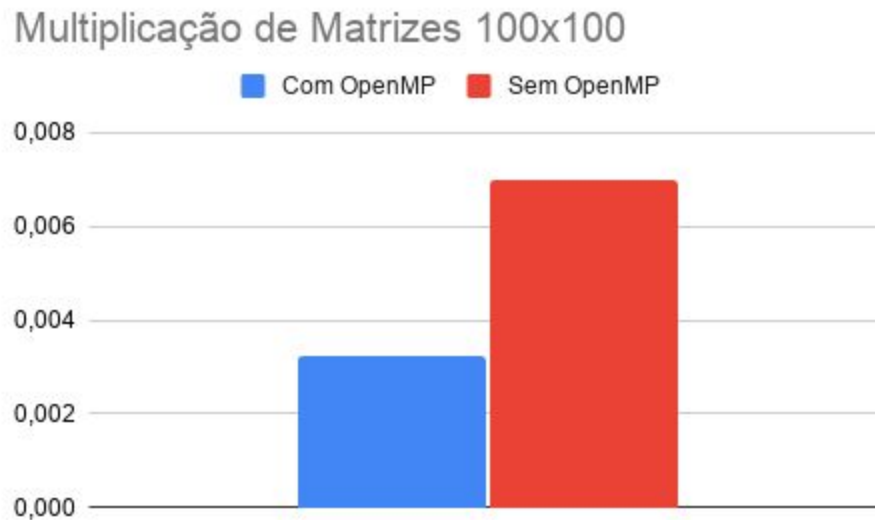
Esses arquivos mostram o tempo para multiplicar a matriz e a redução pela soma. Esses são apenas arquivos para testes. Para o usuário, os tempos de execução e a redução pela soma serão exibidos no terminal.

Multiplicação de matrizes 10x10



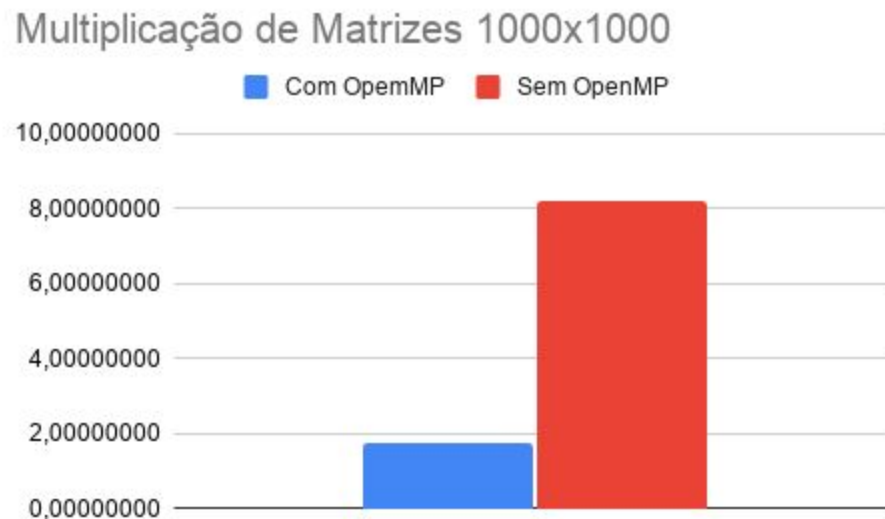
Do gráfico, podemos perceber que o tempo de execução do programa com o OpenMP é maior. Isso acontece pois as matrizes são muito pequenas e, sendo assim, o tempo de criação das threads não compensa o tempo de execução de cada thread. Ou seja, o tempo para criar as 4 threads é maior do que o tempo para que todas elas sejam executadas. Então, uma forma de otimizar o código é criar uma condição para que matrizes pequenas executem sem threads adicionais.

Multiplicação de matrizes 100x100



Do gráfico, podemos perceber que o tempo de execução com o OpenMP é menor para matrizes maiores. Dado que o tempo de execução da multiplicação foi consideravelmente menor com a criação de threads, podemos inferir que tempo para criação de cada thread foi menor que o tempo da carga de trabalho de cada uma das threads. Como o tempo de execução com o OpenMP é um pouco maior que o tempo sem, podemos afirmar que o tempo de execução com e sem o OpenMP são iguais com matrizes de tamanho entre 10x10 e 100x100.

Multiplicação de matrizes 1000x1000



Do gráfico, podemos perceber que o tempo de execução com o OpenMP é muito menor em relação ao tempo sem o OpenMP. Como a razão entre os tempos $\frac{\text{tempo com OpenMP}}{\text{tempo sem OpenMP}}$ diminui com o aumento das dimensões das matrizes, melhor o desempenho para matrizes de tamanhos maiores. Então, é uma boa ideia criar mais threads se a quantidade de dados a serem processados é maior. Entretanto, é uma boa ideia utilizar menos threads se a quantidade de dados for pequena.

Instruções de Uso

Para utilizar o programa, basta clonar o repositório

https://github.com/satoshi-eric/TT005-High_Performance_Computing na máquina local, entrar no diretório OpenMP, entrar em bin e executar o comando

```
./programa <linA> <linB> <linC> <arqA> <arqB> <arqC> <arqD>
```

no terminal e o arquivo `arqD.dat` será gerado. Os argumentos passados no terminal são:

- `linA`: número de linhas da primeira matriz
- `linB`: número de linhas da segunda matriz
- `linC`: número de linhas da terceira matriz
- `arqA`: nome do arquivo `.dat` da primeira matriz
- `arqB`: nome do arquivo `.dat` da segunda matriz
- `arqC`: nome do arquivo `.dat` da terceira matriz
- `arqD`: nome do arquivo `.dat` da matriz resultante

Dentro dos arquivos com as matrizes, para cada elemento da matriz não devem haver nenhum espaço ou caractere do tipo. O arquivo a ser comparado com o arquivo de saída do programa também não deve ter espaço antes ou depois de cada elemento pois o comando do Linux `diff` pode apontar erros por causa desses espaços. Algumas diferenças podem ocorrer ao comparar com arquivos de testes por conta dos tipos `float` e `double`, então pode não estar incorreto, mas impreciso.

Para compilar o programa, é necessário navegar até a pasta `src` e executar o `make`. Um arquivo objeto será criado na pasta `src` e o executável será criado na pasta `bin`.

Caso haja algum tipo de problema para clonar o repositório ou o programa não funcionar, é possível usar a IDE online `gitpod` para acessar e usar o repositório. Para acessá-lo, basta entrar no repositório e digitar `gitpod.io/#` antes da URL. No nosso caso, a URL resultante será `gitpod.io/#https://github.com/satoshi-eric/TT005-High_Performance_Computing`. Com isso, é possível acessar o repositório remotamente e usá-lo sem precisar de um ambiente preparado.

Conclusão

Durante as análises do tempo de execução, tanto para redução da matriz resultante quanto para a multiplicação das matrizes, percebemos sim que o OpenMP melhora bastante a performance do algoritmo, porém para tarefas que exigem pouca demanda com relação a carga de processamento o OpenMP acaba não sendo muito eficaz, o cenário contrário acontece quando a tarefa a ser processada pela máquina exige muita da mesma, nesta situação o OpenMP melhora muito a performance do algoritmo através do paralelismo.