



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

## Relatório do Laboratório de OpenACC

---


Vitor Roberto Kogawa de Moraes - 245582

Eric Satoshi Suzuki Kishimoto - 233974

---

## Sumário

Introdução	3
<b>Desenvolvimento</b>	<b>3</b>
<b>Dificuldades</b>	<b>5</b>
<b>Comparativo OpenACC, OpenMP e nenhuma das duas bibliotecas</b>	<b>5</b>
<b>Instruções de uso</b>	<b>7</b>
<b>Conclusão</b>	<b>8</b>



---

## Introdução


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

## Desenvolvimento

Para realização do trabalho, utilizamos alguns recursos disponibilizados pelo professor. Primeiramente, utilizamos a biblioteca `MatrixIO.h` para ler e escrever as matrizes. Também utilizamos o programa `GenerateRandomMatrix.c` para realizar testes de desempenho da biblioteca OpenACC. Também utilizamos a instância virtual da AWS disponibilizada pelo professor para utilizarmos a biblioteca OpenACC e o compilador gcc para criar o programa.

Assim como no laboratório anterior, este laboratório consiste em realizar a multiplicação de matrizes, porém, com a diferença da tecnologia utilizada. Desta vez, utilizamos uma GPU para paralelizar o código. Inicialmente, utilizamos o código de multiplicação de matrizes do laboratório anterior para realizar o trabalho. Porém, em vez de usar as diretivas do OpenMP, utilizamos diretivas do OpenACC para paralelizar o código.

Para realizar a multiplicação de matrizes com a GPU, utilizamos uma abordagem um pouco diferente da que utilizamos com a CPU. Em vez de inicializarmos a matriz durante a multiplicação, inicializamos ela antes com zeros.



```
#pragma acc parallel loop collapse(2) copyin(matrix3[0:lin3*col3])
for ( i=0; i<lin1; i++ )
{
    for ( j=0; j<col2; j++)
    {

        // Multiplicação de matrizes
        matrix3[posicao(i, j, col3)] = 0;

    }
}
```

No trecho de código acima, usamos 2 loops for para atribuímos 0 à todos os elementos da matriz. Com a diretiva parallel loop, essa inicialização é dividida entre os mini núcleos de processamento da GPU e em seguida, colocamos a diretiva collapse para transformar esses 2 loops em apenas 1, otimizando esse trecho. Também utilizamos a diretiva copyin para mover a matriz 3 (matriz que será o resultado da multiplicação) para a GPU. Com isso, temos a matriz inicializada na GPU e podemos trabalhar com ela.

Depois disso, criamos uma região paralela que irá realizar a multiplicação em si. Usamos 3 loops for aninhados para isso.

```
#pragma acc collapse(3) data copyout(matrix3[0:lin3*col3]) copyin(matrix1[0:lin1*col1], matrix2[0:lin2*col2])
for (i=0; i<lin1; i++)
{
    for (j=0; j<col2; j++)
    {
        for (k=0; k<col1; k++)
        {
            matrix3[posicao(i, j, col3)] += matrix1[posicao(i, k, col1)] * matrix2[posicao(k, j, col2)];
        }
    }
}
```

Utilizamos a diretiva collapse para transformar esses 3 laços em 1. Utilizamos a diretiva data para movimentar os dados entre a CPU e a GPU. Primeiramente, temos o copyout que, irá copiar a matriz 3 para a CPU após o processamento. Depois, temos o copyin para copiar as matrizes 1 e 2 para a GPU e assim, realizar as operações dos loops for dentro da GPU.

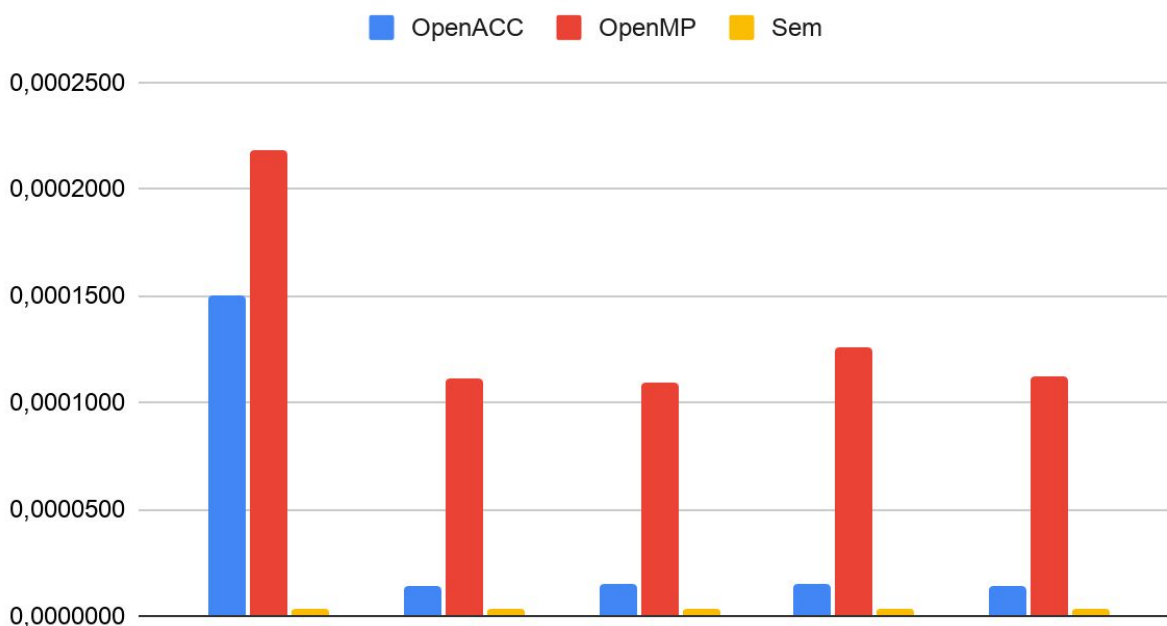
## Dificuldades

Tivemos algumas dificuldades em instalar a biblioteca OpenACC junto com 1 dos compiladores gcc ou PGI, mas uma instância da AWS foi disponibilizadas para o grupo com as ferramentas necessárias para o trabalho. Também tivemos dificuldades em otimizar o código pois o código estava mais lento com a paralelização do que o sem a paralelização. Mas, modificamos o modo com que os dados eram transferidos entre a CPU e a GPU.

## Comparativo OpenACC, OpenMP e nenhuma das duas bibliotecas

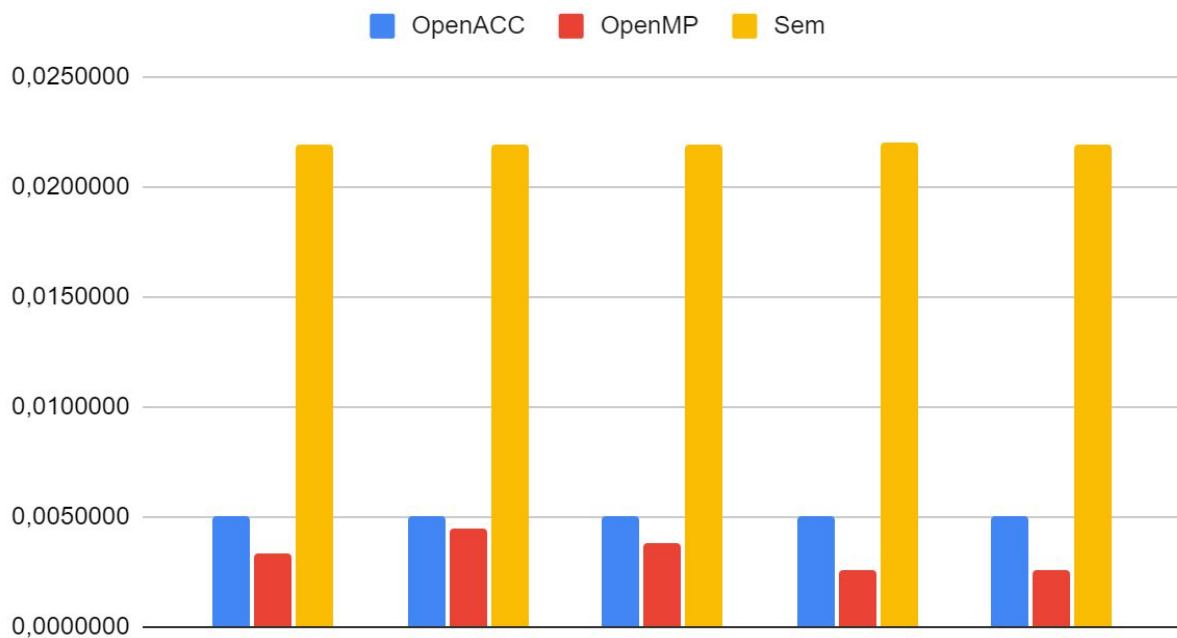
Assim como no laboratório anterior percebemos que para pequenas cargas de trabalho como a multiplicação e redução de uma matriz 10x10, a utilização de bibliotecas de paralelização não é a abordagem mais indicada quando o foco é performance, a imagem abaixo mostra graficamente este comparativo.

### Matriz 10x10

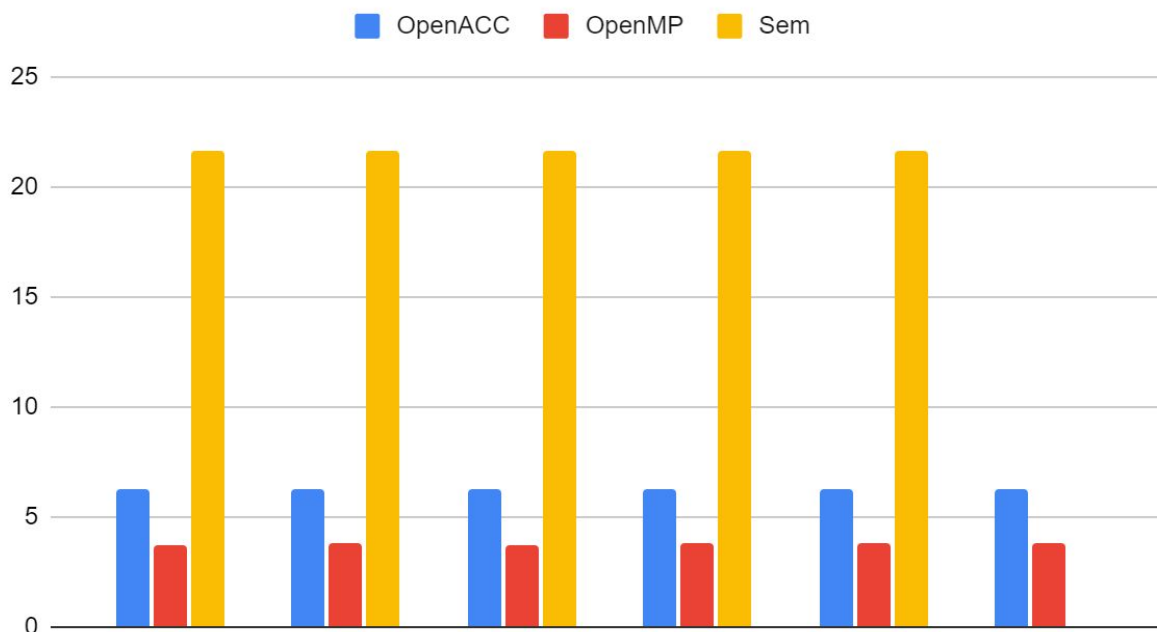


Para tarefas que exigem mais cargas de trabalho como a multiplicação e redução de matrizes 100x100 e/ou 1000x1000 as bibliotecas de paralelização são mais indicadas, porém cada uma trabalho a sua maneira e nos testes que realizamos a biblioteca OpenMP se saiu melhor do que a biblioteca OpenACC.

### Matrix 100x100



## Matrix 1000x1000



Tanto para matrizes como a 100x100 como para a 1000x1000 podemos ver que em alguns dos testes o OpenACC demorou quase que o dobro do tempo que o OpenMP.

Acreditamos que esta diferença de desempenho se dá pelo fato do OpenMP trabalhar somente com base na CPU, diferentemente do OpenACC que trabalha tanto com a GPU quanto com CPU e a passagem de dados entre as zonas de memória da CPU e GPU pelo IO Bus acaba afetando de maneira direta a performance do algoritmo, pois como dito em aula não é recomendado o uso em excesso deste barramento. Entretanto, o código sem paralelização demorou muito mais para processar os dados em relação ao OpenACC para as matrizes 100x100 e 1000x1000.

## Instruções de uso

Para utilizar o programa, basta clonar o repositório

[https://github.com/satoshi-eric/TT005-High\\_Performance\\_Computing](https://github.com/satoshi-eric/TT005-High_Performance_Computing) na máquina local, entrar no diretório OpenACC, entrar em bin e após colocar os arquivos das matrizes dentro do diretório bin basta executar o comando abaixo:

---

```
./main <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

Ao comparar as matrizes utilizando o comando diff, é possível que ele aponte diferenças entre os 2 arquivos. Isso pode ocorrer devido à presença de caracteres de escape. Para comparar apenas os valores da matriz, utilizar o argumento -w para que o comando diff não acuse diferenças apenas de valores e não de caracteres de escape.

Para compilar o arquivo, é necessário navegar até a pasta src e executar o make. Como o programa já foi gerado na pasta bin, é provável que ele diga que o main está atualizado. Caso seja necessário recompilar o código, basta executar o comando make clean e executar o make novamente.

## Conclusão

Portanto, durante as análises dos tempos de execução, podemos ver que para estes testes em específico e a maneira com que o algoritmo foi implementado utilizando as ferramentas de cada biblioteca, o OpenMP foi mais performático que o OpenACC, porém reforçando acreditamos que essa diferença se dá pela responsabilidade a mais que o OpenACC tem com



---

relação a passagem de dados pelo IO Bus, e que para tarefas que exigem pouca carga de trabalho a paralelização não é a estratégia mais recomendada.