Universidade Estadual de Campinas - UNICAMP
Faculdade de Tecnologia - FT

LABORATÓRIO II - OpenACC

GRUPO: **OPTIMUS TECH**

Integrantes:
Carolina da Silva Sancho 214376
Gabriella Carlos do Nascimento 260587

LIMEIRA, SÃO PAULO
30 DE NOVEMBRO DE 2020

SUMÁRIO COM HYPERLINKS

1. Introdução
2. Compilação e Execução
3. Análise e Gráficos
4. Conclusão

1. Introdução

Este laboratório tem como objetivo resolver um problema utilizando biblioteca para programação de alto desempenho (PAD), onde neste trabalho será utilizado a biblioteca `<openacc.h>`, cujo código foi realizado obrigatoriamente na Linguagem C e executado no MobaXterm (para Windows).

O problema a ser resolvido neste laboratório, e nos demais, consiste em calcular a multiplicação entre matrizes e a redução pela soma dos elementos da matriz resultante dessa multiplicação. Como requisitos importantes para a realização da multiplicação das matrizes temos:

- As matrizes precisam ser alocadas dinamicamente (com o comando malloc ou equivalente).
- Todas as matrizes necessárias para os programas precisam ser alocadas em uma única etapa.

2. Compilação e Execução

1) Para compilar e executar os programas em C **sem o script** no MobaXterm ou em outro terminal siga os seguintes passos:

PASSO 1: Compilar e executar o código em "`criaArquivos.c`" , onde serão gerados os arquivos com os dados que irão compor as matrizes do código em "`matrizOpenACC.c`", que serão usadas para o cálculo da multiplicação das matrizes e da redução pela soma da `matrizD`.

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos y w v arqA.dat arqB.dat arqC.dat
```

As variáveis "`y w v`" representam o tamanho das matrizes (`matrizA[y][w]`, `matrizB[w][v]` e `matrizC[v][1]`), e "`arqA.dat arqB.dat arqC.dat`" são os arquivos que foram criados no código "`criaArquivos.c`". Para os valores de "`y w v`", utilizaremos: `y=10 w=10 v=10`, `y=100 w=100 v=100` e `y=1000 w=1000 v=1000`, então basta substituir esses valores para gerar novos arquivos.

PASSO 2: Compilar e executar o código em "`matrizOpenACC.c`", para realizar o cálculo da multiplicação das matrizes e da redução pela soma da `matrizD`, para isso, digite os seguintes comandos:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos y w v arqA.dat arqB.dat arqC.dat
```

```
gcc -o matrizOpenACC -fopenacc matrizOpenACC.c
./matrizOpenACC y w v arqA.dat arqB.dat arqC.dat arqD.dat
```

Em que, "matrizOpenACC.c" é o nome do arquivo em C, "-fopenacc" é o comando para executar a biblioteca <openacc.h>, "y w v" são as variáveis do tamanho das matrizes (matrizA[y][w], matrizB[w][v] e matrizC[v][1]), e "arqA.dat arqB.dat arqC.dat" são os arquivos que foram criados no código "criaArquivos.c" e o "arqD.dat" que foi criado no arquivo "matrizOpenACC.c", todos com valores aleatórios, conforme quantidade determinada pelas variáveis digitadas, cujos valores serão utilizados para realizar a multiplicação das matrizes no código "matrizOpenACC.c". Abaixo, estão os exemplos dos comando com os valores das variáveis determinadas pelas especificações dos laboratório:

Para y=10, w=10 e y=10, temos:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos 10 10 10 arqA.dat arqB.dat arqC.dat
```

```
gcc -o matrizOpenACC -fopenacc matrizOpenACC.c
./matrizOpenACC 10 10 10 arqA.dat arqB.dat arqC.dat arqD.dat
```

Para y=100, w=100 e y=100, temos:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos 100 100 100 arqA.dat arqB.dat arqC.dat
```

```
gcc -o matrizOpenACC -fopenacc matrizOpenACC.c
./matrizOpenACC 100 100 100 arqA.dat arqB.dat arqC.dat arqD.dat
```

Para y=1000, w=1000 e y=1000, temos:

```
gcc criaArquivos.c -o criaArquivos
./criaArquivos 1000 1000 1000 arqA.dat arqB.dat arqC.dat
```

```
gcc -o matrizOpenACC -fopenacc matrizOpenACC.c
./matrizOpenACC 1000 1000 1000 arqA.dat arqB.dat arqC.dat
arqD.dat
```

2) Para compilar e executar o programa em C **com o script** digite os seguintes comandos (em que novamente "y w v" são as variáveis do tamanho das matrizes):

PASSO 1: Compilar e executar o código em "`criaArquivo.c`", seguindo os passos disponíveis acima em **PASSO 1** onde não é utilizado o script.

PASSO 2: Escrever os seguintes comandos:

```
chmod +x matriz.sh
./matriz.sh
./matrizOpenACC y w v arqA.dat arqB.dat arqC.dat arqD.dat
```

Exemplo:

```
chmod +x matriz.sh
./matriz.sh
./matrizOpenACC 10 10 10 arqA.dat arqB.dat arqC.dat arqD.dat
```

3. Análise e Gráficos

A partir da execução dos códigos em linguagem C disponíveis em: https://github.com/carolsancho/Laboratorios-PAD, os tempos de processamento da multiplicação e do cálculo da redução por soma da matriz D, descontadas as operações de entrada e saída, para os valores das seguintes variáveis: y = 10, w = 10, v = 10 **(1)**; y =100, w = 100, v = 100 **(2)**; y =1000, w = 1000, v = 1000 **(3)**, em que **não foi utilizado a biblioteca `<openacc.h>`** tiveram os seguintes resultados:

VARIÁVEIS	TEMPO DE PROCESSAMENTO (ms)
(1)	0.03
(2)	4.98
(3)	6197.52

Agora, os tempos de processamento para as mesmas variáveis e **utilizando a biblioteca `<openacc.h>`** foram obtidos os seguintes resultados:

VARIÁVEIS	TEMPO DE PROCESSAMENTO (ms)
(1)	0.04
(2)	4.82
(3)	5671.31

Assim, a partir da análise dos tempos de processamento das tabelas acima foi possível perceber que, quando é utilizado o OpenACC, devido a programação ser paralela,

e o código ser executado por um acelerador conectado, possível através da programação híbrida de CPU + GPU, os tempos de processamentos foram inferiores ao serem comparados quando não utilizado a biblioteca `<openacc.h>`, com exceção de valores muitos baixos para as variáveis, como por exemplo o caso encontrado em **(1)**, onde os tempos de processamento não apresentam significativa diferença, porém para resultados superiores ao valor máximo das variáveis encontradas em **(2)** e **(3)**, nota-se uma melhora nos tempos de processamento quando usado a biblioteca `<openacc.h>`, por conta da paralelização, que podemos ver sua utilização nas linhas 22, 34 e 45 do arquivo "`matrizOpenACC.c`" apresentadas a seguir e no Gráfico 1 abaixo.

Imagem 1 - paralelização utilizada para realizar a multiplicação da matrizA com a matrizB e salva o resultado em uma matriz temporária chamada matrizTemp.

```
22      #pragma acc parallel loop collapse(2)
23          for (int lin = 0; lin < y; lin++){
24              for (int col = 0; col < v; col++){
25                  matrizTemp[posicao(lin, col, v)] = 0;
26                  #pragma acc loop seq
27                  for (int i = 0; i < w; i++){
28                      matrizTemp[posicao(lin, col, v)] += matrizA[posicao(lin, i, w)] * matrizB[posicao(i, col, v)];
29                  }
30              }
31          }
```

Fonte: imagem retirada do arquivo "`matrizOpenACC.c`" elaborado pelas autoras deste documento.

Imagem 2 - paralelização utilizada para realizar a multiplicação da matrizTemp com a matrizC e salva o resultado na matrizD.

```
34      #pragma acc parallel loop seq
35          for (int lin = 0; lin < y; lin++){
36              int col = 0;
37              matrizD[posicao(lin, col, 1)] = 0;
38              for (int i = 0; i < v; i++){
39                  matrizD[posicao(lin, col, 1)] += matrizTemp[posicao(lin, i, v)] * matrizC[posicao(i, col, 1)];
40              }
41          }
```

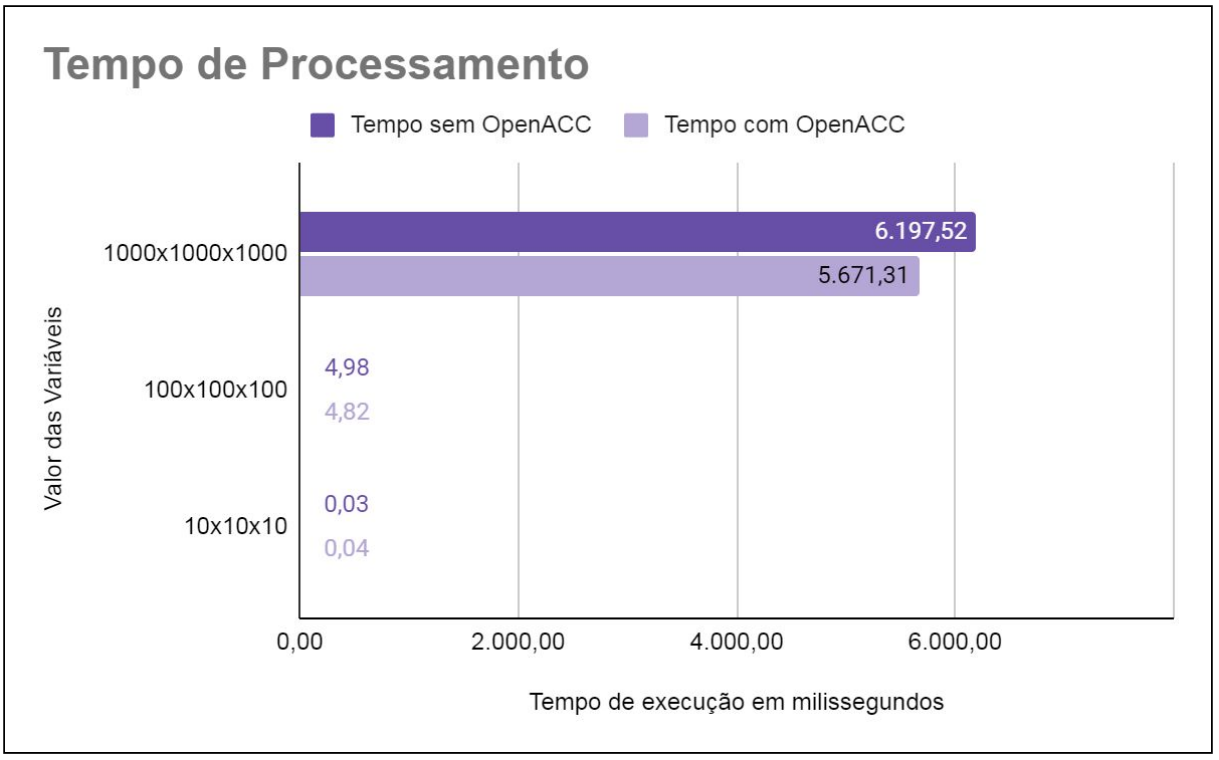
Fonte: imagem retirada do arquivo "`matrizOpenACC.c`" elaborado pelas autoras deste documento.

Imagem 3 - cálculo da redução pela soma dos valores apresentados na matrizD.

```
45      #pragma acc parallel loop collapse(2) reduction(+:soma)
46          for (i = 0; i < y; i++){
47              for (j = 0; j < 1; j++){
48                  soma += matrizD[posicao(i,j,1)];
49              }
50          }
51          printf("%lf\n", soma);
52  }
```

Fonte: imagem retirada do arquivo "`matrizOpenACC.c`" elaborado pelas autoras deste documento.

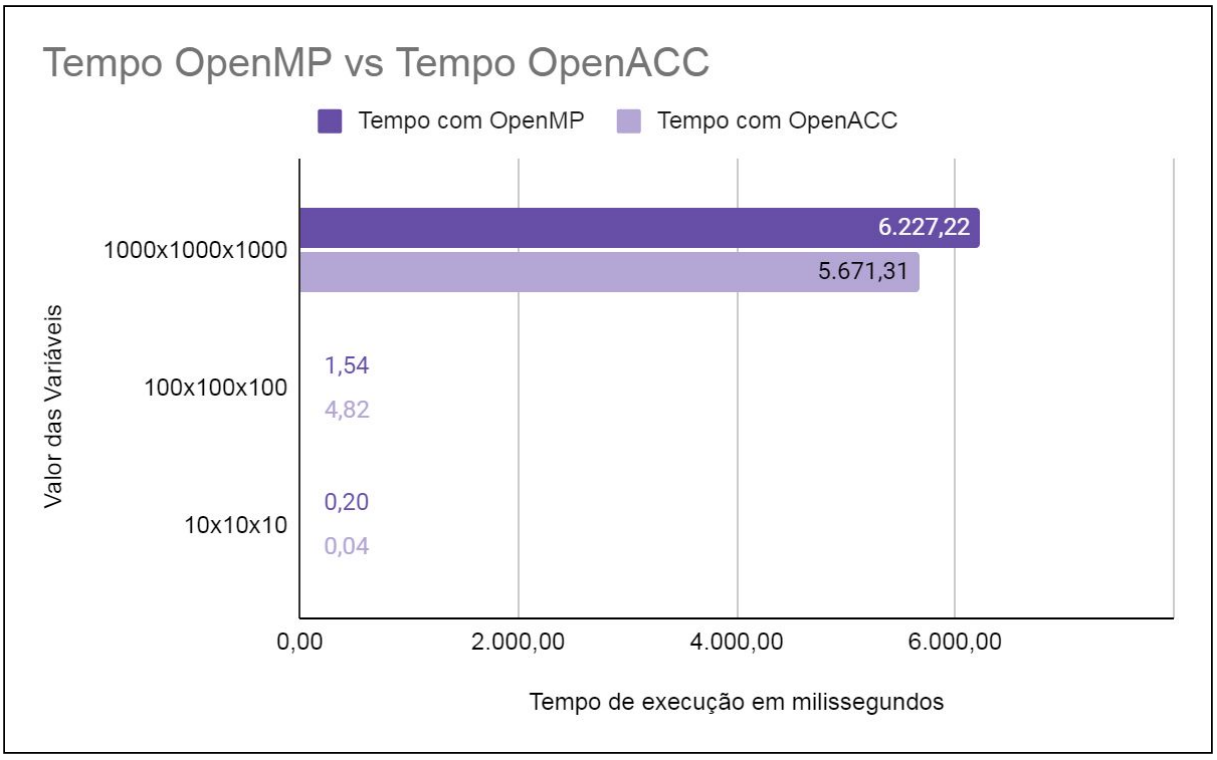
Gráfico 1 - Tempos de Processamento da multiplicação e cálculo da redução pela soma da matriz D.



Fonte: elaborado pelas autoras deste documento.

Tanto OpenMP (visto no laboratório anterior) quanto o OpenACC permitem a programação paralela baseada em diretivas, onde o OpenMP permite a programação paralela em plataformas de computação de memória compartilhada, e o OpenACC permite que o código seja executado por um acelerador conectado, possibilitando a programação híbrida de CPU + GPU (como dito anteriormente), o que melhora a sequenciação e o paralelismo, visto que o OpenACC utiliza a CPU para otimizar a parte sequencial do código e usa o acelerador para executar a parte paralela do código, o que pode ser visto no gráfico abaixo, em que o tempo de processamento para variáveis iguais a **(3)** ou superiores a esse valor, apresentam melhor desempenho quando utilizado o OpenACC. Porém, para valores menores entre **(2)** e **(3)**, é expressiva a diferença entre os tempos, o que mostra que um melhor desempenho do uso de multi-thread no código.

Gráfico 2 - Comparação entre os tempos de Processamento utilizando OpenMP e OpenACC.



Fonte: elaborado pelas autoras deste documento.

4. Conclusão

A partir do Laboratório II de OpenACC da disciplina de Tópicos Especiais em Telecomunicações I (Programação de Alto Desempenho), o grupo Optimus Tech foi capaz de ter como experiência a aplicação de conceitos estudados durante as vídeo aulas de OpenACC e pelos exemplos apresentados no repositório disponibilizado pelo professor no github, e assim correlacionando-os com a prática, sendo possível através da criação de um código em linguagem C que utilizou o OpenACC para exemplificar a simplificação da programação paralela de sistemas heterogêneos de CPU / GPU, onde no código realizado pelo grupo utilizou-se em grande parte `#pragma acc parallel loop`.

Portanto, através da prática foi possível ter uma melhor fixação dos conteúdos trabalhados na disciplina, além de ter explorado o emprego do OpenACC e utilizado novamente a ferramenta MobaXterm como suporte.