Aluno: *Sérgio Luciano de Oliveira Soares*

RA: *263560*

Faculdade de Tecnologia

Relatório para disciplina FT077A - Processamento de Alto Desempenho.

Prof. André Leon S. Gradvohl, Dr.

12/06/2021

SUMÁRIO

1. INTRODUÇÃO

Com base nos problemas propostos anteriormente realizado nos laboratório 01 em PThreads, laboratório 02 OpenAcc, e laboratório 03 com OpenMP, este laboratório 04 que consiste em criar um programa serial e um programa paralelo com OpenMPI que calcule a operação matricial D = A * B + C, onde todas as matrizes (A, B, C e D) têm dimensões n x n.

2. METODOLOGIA

Para esse exercício foram realizadas adequações do programa desenvolvido anteriormente em PThreads e OpenMP reutilizando-se todas as otimizações e procedimentos que já haviam sido desenvolvidas no trabalho anterior, bem como os recursos de verificações de resultado desenvolvidos na tarefa 03 em OpenAcc.

A divisão da tarefa entre os processos e o esquema de comunicação entre eles teve que ser repensado e remodelado por inteiro, dado à diferença entre a arquitetura para a execução com memória distribuída e com memória compartilhada.

3. DESENVOLVIMENTO

Otimizações utilizadas para o cálculo:
D = A*B+C = C+A*B

soma de matrizes: $s_{ij} = a_{ij} + b_{ij}$
multiplicação de matrizes: $m_{ik} = a_{i1} . b_{1k} + a_{i2} . b_{2k} + ... + a_{in} . b_{nk}$

Para o cálculo do resultado D (=A*B+C), temos a fórmula para cada elemento:
$D_{ik} = C_{i1} + A_{i1} . B_{1k} + A_{i2} . B_{2k} + ... + A_{in} . B_{nk}$

A matriz foi alocada em uma única etapa (como um vetor).

O acesso aos elementos para o loop acima (1..n) causa o acesso aos elementos em B de forma não sequencial na memória.

Como o acesso a posições próximas de memória são mais interessantes, foi utilizada a abordagem de calcular a matriz transposta de B, de forma que o loop cause o acesso aos elementos de B de forma sequencial, assim como em A.

Bt = transposta de B

Com isso, temos:

$D_{ik} = C_{i1} + A_{i1} . B_{k1} + A_{i2} . B_{k2} + ... + A_{in} . B_{kn}$

O código todo foi feito em um único arquivo (exerc4_OpenMPI.c).

Para compilação:

mpicc exec4_OpenMPI.c -o exerc4_OpenMPI

https://github.com/s263560/pad/blob/master/MPI/exerc4_OpenMPI.c

A rotina para o cálculo serial foi mantida a mesma dos exercícios anteriores.

A rotina para verificar se o resultado do cálculo serial em D bate com o cálculo paralelo em D2 feita no exercício com OpenACC também foi mantida, pois haviam várias possibilidades de erros durante a criação do programa para o processamento paralelo com OpenMPI.

A verificação do resultado ajudou a validar as alterações feitas no algoritmo, bem como a necessidade do tratamento do 'resto' da matriz ao dividir as linhas igualmente entre os processos, já que nos exercícios anteriores foi utilizada outra abordagem para a divisão.

O algoritmo foi alterado de modo que o laço passasse a calcular 'por linha', ao invés de calcular por elemento da matriz. Isso foi devido ao alto custo de comunicação existente no MPI usando processos em hosts diferentes.

A lógica usada para a distribuição do processamento foi enviar a matriz transposta Bt para todos os nodos via broadcast, e o envio parcial das matrizes A e C para os nodos, sendo a distribuição do processamento baseada em blocos de 'linhas' das matrizes A e C.

Como a distribuição dos blocos nem sempre é exata, foi adicionada uma rotina para calcular, no nodo master, as linhas restantes da divisão do processamento. Por exemplo, ao dividir 10 linhas em 3 processos, cada processo fica com 3 linhas, sobrando 1 linha ainda a ser processada. Essa(s) linha(s) restantes foram calculadas no nodo master de modo serial após o processamento em paralelo.

Para a execução com até 8 processos, o arquivo 'hosts' utilizado foi:

```
masternode slots=1
workernode1 slots=4
workernode2 slots=4
```

Essa configuração é compatível com a configuração ‘ideal’ no ambiente utilizado, pois o número de slots em cada nodo equivale ao número de processadores lógicos disponíveis.

Para a execução com 16 processos, não foi possível utilizar a configuração ‘ideal’, então o arquivo ‘hosts2’ foi criado para o equivalente a 2 slots para cada processador lógico:

```
masternode slots=2
workernode1 slots=8
workernode2 slots=8
```

Após a compilação, também foi necessário a cópia do executável para cada worknode, sendo essa cópia feita através dos comandos:

```
scp exerc4_OpenMPI workernode1:${PWD}/exerc4_OpenMPI
scp exerc4_OpenMPI workernode2:${PWD}/exerc4_OpenMPI
```

Após a cópia, as execuções foram feitas, na sequência, para 1, 2, 4, 8 e 16 processos com os seguintes comentos:

```
mpirun -n 1 -hostfile ./hosts ./exerc4_OpenMPI
mpirun -n 2 -hostfile ./hosts ./exerc4_OpenMPI
mpirun -n 4 -hostfile ./hosts ./exerc4_OpenMPI
mpirun -n 8 -hostfile ./hosts ./exerc4_OpenMPI
mpirun -n 16 -hostfile ./hosts2 ./exerc4_OpenMPI
```

A execução com 1 processo, não pedida no enunciado do exercício, foi feita para se ter uma ideia do overhead adicionado ao utilizar a biblioteca MPI mesmo realizando a execução em um único host.

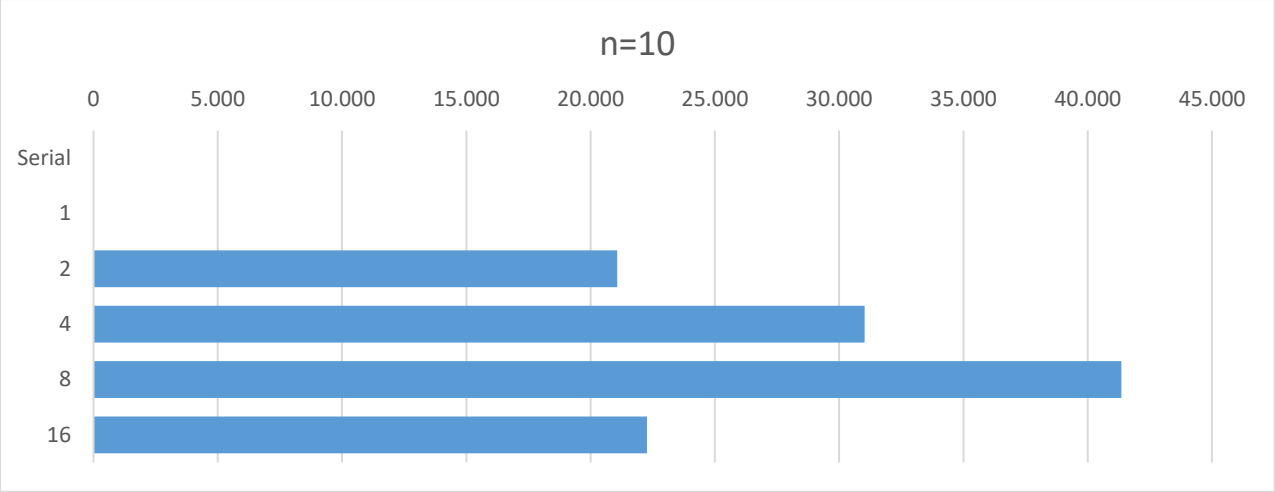
4. RESULTADOS E GRÁFICOS

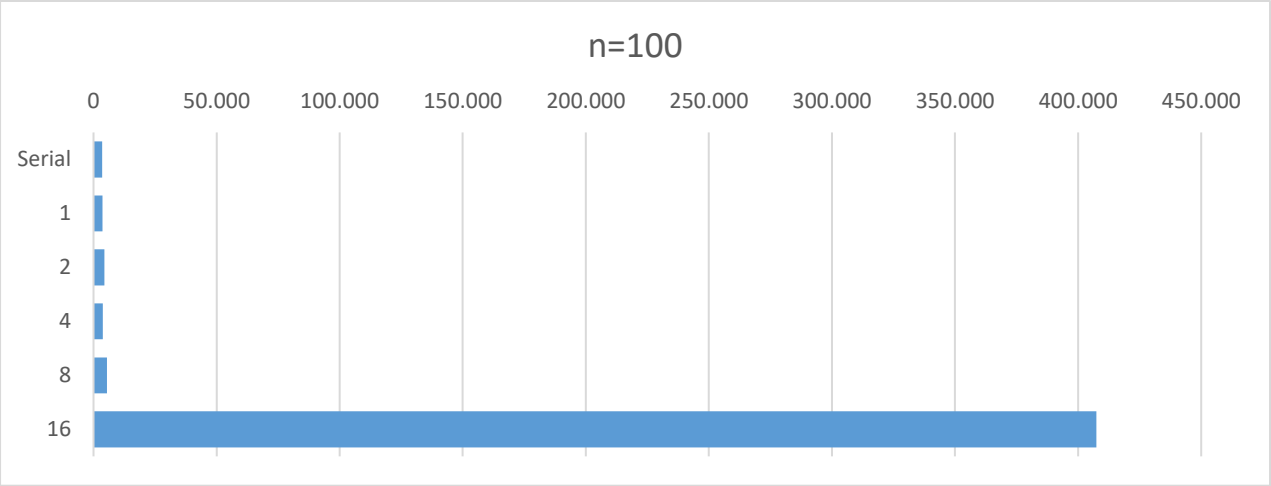
Calcule o tempo de execução do programa serial para matrizes de tamanho n x n, onde n = 10, 100 e 1000.

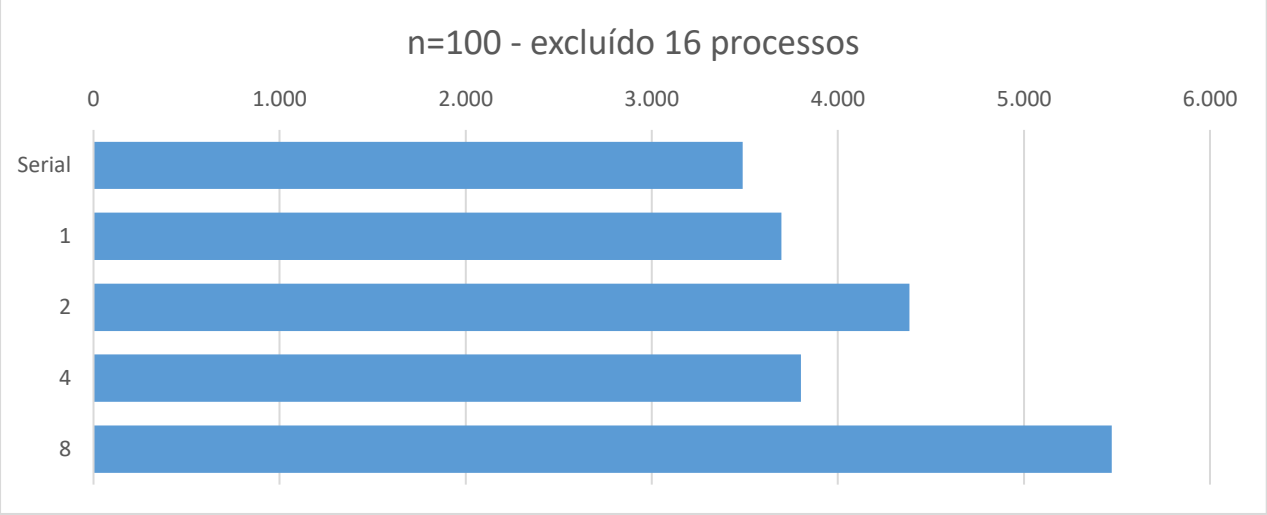
Calcule o tempo de execução do programa paralelo para matrizes de tamanho n x n, onde n = 10, 100 e 1000, cada uma com 2, 4, 8 e 16 processos MPI.

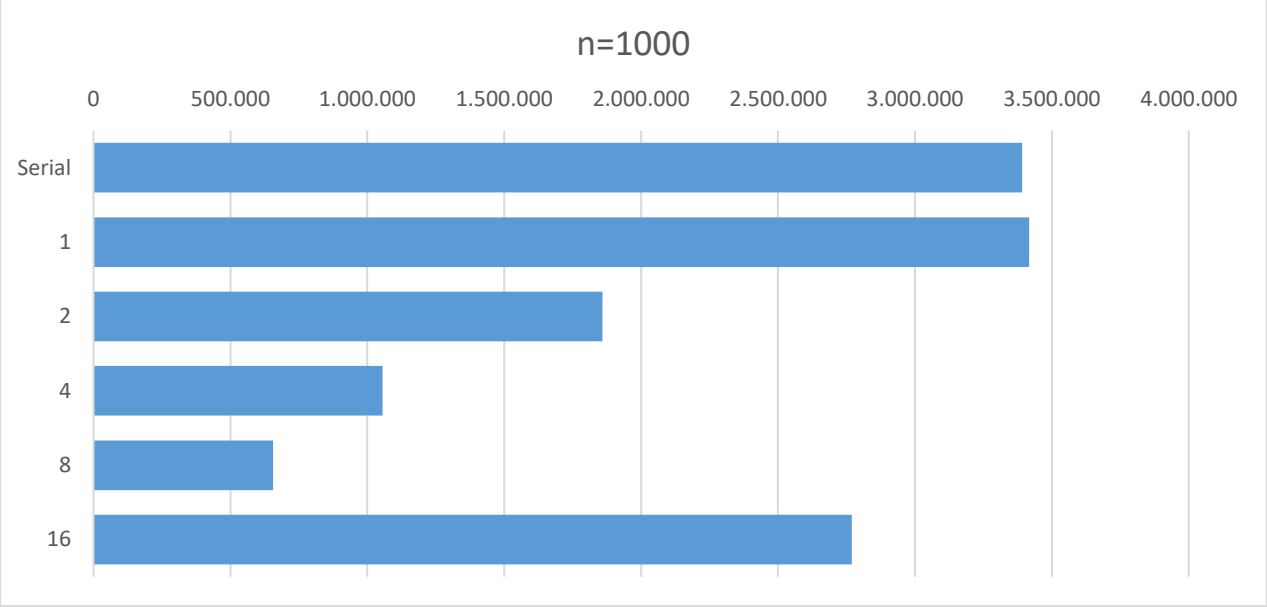
Tempos de execução em microssegundos:

	Processos MPI					
n	Serial	1	2	4	8	16
10	5	16	21.074	31.025	41.353	22.263
100	3.488	3.696	4.384	3.801	5.472	407.480
1000	3.391.990	3.416.835	1.859.096	1.055.263	655.799	2.769.141


n=10
0
5.000
10.000
15.000
20.000
25.000
30.000
35.000
40.000
45.000
Serial
1
2
4
8
16


n=100
0
50.000
100.000
150.000
200.000
250.000
300.000
350.000
400.000
450.000
Serial
1
2
4
8
16


n=100 - excluído 16 processos
0
1.000
2.000
3.000
4.000
5.000
6.000
Serial
1
2
4
8



5. RESPOSTAS SOLICITADAS

I) Há necessidade de sincronização entre as threads para resolver as operações?

A utilização das funções “bloqueantes” do MPI foram suficientes para a sincronização entre os processos criados, então não foi necessário utilizar nenhum recurso a mais que as operações básicas do MPI.

II) Qual foi o speedup em relação ao programa serial em cada uma das execuções?

Speedup:

	Processos				
n	1	2	4	8	16
10	0,3125	0,0002	0,0002	0,0001	0,0002
100	0,9437	0,7956	0,9177	0,6374	0,0086
1000	0,9927	1,8245	3,2144	5,1723	1,2249

Obs: a coluna com 1 processo se refere ao algoritmo utilizando MPI com apenas 1 processo em relação ao processamento serial puro.

III) Houve algum caso em que não houve speedup em relação ao programa serial? Se houve, qual a razão para isso?

Sim.

Com n=10, não houve speedup em nenhuma situação, pois o overhead de comunicação (a partir de 2 processos) deixou a execução mais de 5.000 vezes mais lento. O overhead de controle do MPI observado na execução com 1 processo mostrou um overhead baixo – apesar de 3 vezes o tempo de processamento, foram adicionados apenas 10 microssegundos para os controles.

Com n=100, a velocidade com até 8 processos não apresentou speedup maior que 1, porém não foi inferior a 0,5. Para o caso de 16 processos – onde haviam mais slots que processadores lógicos para cada nodo, a perda de desempenho foi da ordem de 100.000 vezes!

Com n=1000, houve speedup próximo ao ideal com 2 e 4 processos, um bom speedup com 8 processos, e um speedup acima de 1 mesmo com 16 processos.

IV) Em relação ao laboratório anterior, a solução com MPI foi melhor ou pior do que a solução com o OpenMP? Por que?

Houve um ganho no tempo de execução para o cálculo com n=1000 usando MPI em relação ao OpenMP quando da utilização de 8 processos. No exercício de OpenMP haviam 4 processadores disponíveis, e nessa situação de 8 processos, foram utilizados 8 processadores em 3 máquinas diferentes (1 no masternode, 4 no workernode1 e 3 no workernode2).

Apesar do overhead de comunicação necessário, o tempo total de execução foi menor nessa situação.

Porém, nas outras situações (até 4 processos/threads) o OpenMP demonstrou uma melhor performance.

6. CONCLUSÕES

As alterações para o MPI no código foram muito mais significativas do que para a utilização do OpenMP, OpenACC e Pthreads.

Porém, foi possível verificar que é possível obter ganhos de performance com a utilização de mais nodos adicionados ao sistema, e, apesar da lógica para a criação do algoritmo para a utilização do MPI considerando todas as necessidades de comunicação, alocação de memória, e modificações necessárias, não houve necessidade de uma refatoração completa do código inicial.