

**Programação para Sistemas Paralelos e Distribuídos**

Questão 4) Para uma determinada dimensão do referido fractal, qual dos três programas montados apresenta melhor performance? Qual o percentual de ganho de uma solução em relação a outra? Para essa resposta, montar um experimento controlado, com simulações de execução dos códigos, levando-se em conta os parâmetros que influenciam a resposta (número de threads, de núcleos, de máquinas, etc.). Ao final dos testes, monte uma tabela comparativa e mostre os tempos de execução de cada programa, considerando os parâmetros que influenciam a performance e apresente uma resposta conclusiva.

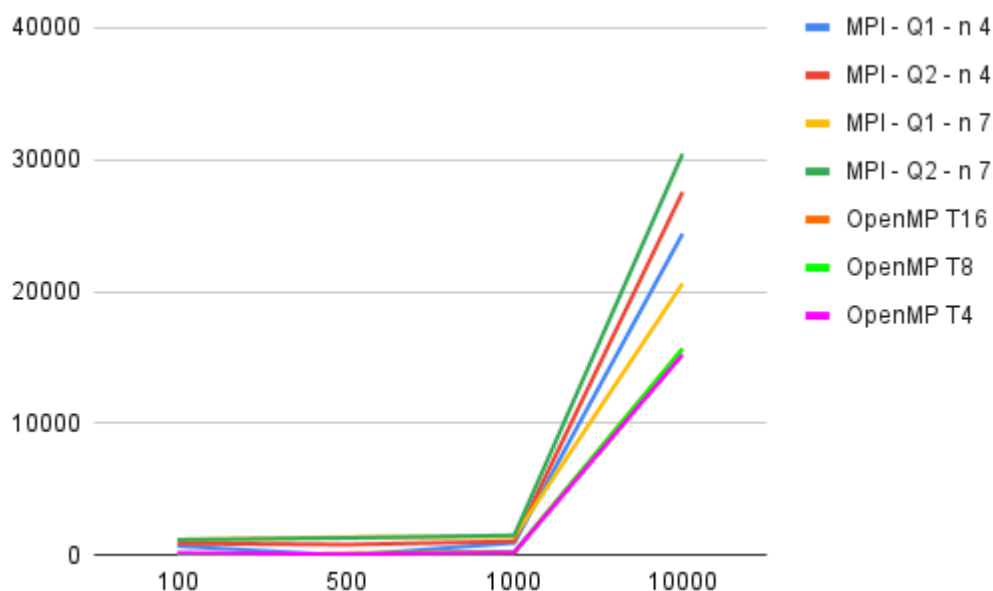
Fractal	100	500	1000	10000
MPI - Questão 1 -n 4	0.724s	0.825s	0.962s	24.370s
MPI - Questão 2 -n 4	0.923s	0.829s	1.076s	27.520s
MPI - Questão 1 -n 7	1.195s	1.388s	1.421s	20.598s
MPI - Questão 2 -n 7	1.193s	1.337s	1.531s	30.419s
OpenMP \$OMP_NUM_THREADS = 16	0.016s	0m0.157s	0.249s	15.648s
OpenMP \$OMP_NUM_THREADS = 8	0.020s	0.132s	0.239s	15.660s
OpenMP \$OMP_NUM_THREADS = 4	0m0.205s	0m0.102s	0m0.232s	15.213s

A partir da tabela podemos concluir que o **MPI** (Message Passing Interface) que permite vários processos em diferentes hosts se comunicam entre si e coordenam seu trabalho em uma tarefa paralela. Quando se utiliza MPI em mais de um host, o tempo de execução pode ser afetado por diversos fatores, como a latência e a largura de banda da rede, a carga de trabalho distribuída entre os hosts e a eficiência do algoritmo paralelo implementado. Assim, conforme podemos observar o tempo de execução para 7 hosts seja superior ao tempo de execução para 4 hosts. Isso pode ocorrer devido a vários motivos, como: **Sobrecarga da rede**, quanto mais hosts são utilizados, maior é o tráfego de dados na rede e maior é a latência de comunicação entre os hosts. Isso pode levar a atrasos na troca de mensagens entre os processos MPI e, conseqüentemente, aumentar o tempo de execução e **Balanceamento de carga**, ao utilizar mais hosts, é necessário distribuir a carga de trabalho entre eles de forma equilibrada. Se essa distribuição não for eficiente, alguns hosts podem ficar ociosos enquanto outros estão sobrecarregados,

o que pode diminuir a eficiência do algoritmo paralelo e aumentar o tempo de execução.

Em relação ao **OpenMP** é a capacidade de executar tarefas em paralelo por meio do uso de threads. Quando se utiliza OpenMP, aumentar o número de threads pode melhorar o desempenho em certas condições. Isso ocorre porque, em geral, o aumento do número de threads leva a uma distribuição mais equilibrada da carga de trabalho entre os processadores e reduz o tempo de espera ocioso para acesso à memória compartilhada.

Portanto, conforme observado na tabela, o OpenMP permite que essas partes do código sejam executadas em paralelo por meio do uso de threads, aproveitando a memória compartilhada disponível no sistema e alcançando uma escalabilidade muito boa e um desempenho alto. Por outro lado, o MPI é mais adequado para aplicações que requerem comunicação e coordenação entre múltiplos processos em sistemas distribuídos. O MPI permite que os processos se comuniquem e coordenem entre si, permitindo que eles realizem cálculos paralelos em vários sistemas.





Exemplos de Execução:

Os tempos da tabela acima foram obtidos mediante chococino.

Questão 1)

Local:

```
gcc -c lib_julia.c
```

```
mpicc lib_julia.o fractalmpiserial.c -o fractalmpiserial -lm
```

```
mpirun -n 8 ./fractalmpiserial 1000
```

ou

Chococcino:

```
gcc -c lib_julia.c
```

```
mpicc lib_julia.o fractalmpiserial.c -o fractalmpiserial -lm
```

```
mpirun -host cm1,cm2,cm3,cm4,gpu1,gpu2,gpu3 -n 7 ./fractalmpiserial 1000
```

ou

```
make
```

```
mpirun -host cm1,cm2,cm3,cm4,gpu1,gpu2,gpu3 -n 7 ./fractalmpiserial 1000
```

Questão 2)

```
gcc -c lib_julia.c
```

```
mpicc lib_julia.o fractalmpi_io.c -o fractalmpi_io -lm
```

```
mpirun -n 4 ./fractalmpi_io 1000
```

ou

```
gcc -c lib_julia.c
```

```
mpicc lib_julia.o fractalmpi_io.c -o fractalmpi_io -lm
```

```
mpirun -host cm1,cm2,cm3,cm4 -n 4 ./fractalmpi_io 1000
```

ou

```
make
```

```
mpirun -host cm1,cm2,cm3,cm4,gpu1,gpu2,gpu3 -n 7 ./fractalmpi_io 1000
```

Questão 3)

```
make
```

```
make run V_THREADS=8 N_FRACTAL=1000
```