UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

Sistemas Operacionais

Trabalho Prático da Unidade 1 – Processos e Threads

Relatório

Professor: Gustavo Girão

Aluno: André Augusto Fernandes

Natal / 2022

Sumário

1. Introdução

Sistemas operacionais permitem utilizar diferentes métodos de processamento para otimizar o desempenho da execução da uma tarefa pretendida, como processamento sequencial, paralelo por threads ou paralelo por processos.

Há situações em que um problema maior pode ser fracionado em partes menores, independentes entre si, de forma que seja possível sua resolução por execução paralela, alcançando o resultado em um tempo menor do que se fosse feito de forma sequencial.

Um exemplo deste tipo de utilização é a multiplicação de Matrizes, cujos elementos da matriz resultado não se comunicam entre si, necessitando apenas de partes dos dados das matrizes de origem, permitindo assim a paralelização da computação do resultado.

2. Objetivo

Esta atividade se propõe a analisar o comportamento do sistema quando submetido à multiplicação de matrizes utilizando as formas sequencial e paralelas (threads e processos).

Para isso foram desenvolvidos três programas que implementam cada uma das formas de programação sugeridas acima, utilizando diferentes tamanhos de matrizes e de quantidades variáveis de paralelização, a fim de observar o comportamento de cada e identificar os métodos que têm os melhores e piores desempenhos.

3. Programas

3.1. Auxiliar

O programa auxiliar será responsável por criar duas matrizes de dimensões fornecidas pelo usuário, que terá seus elementos preenchidos de forma aleatória pelo programa. As matrizes deverão ser gravadas nos arquivos matriz_1.txt e matriz_2.txt.

Compilação:

```
gcc auxiliar.c -o auxiliar
```

Execução:

```
./auxiliar <linhasA> <colunasA> <linhasB> <colunasB>
```

3.2. Sequencial

O programa deverá multiplicar, de forma sequencial, duas matrizes fornecidas pelo usuário e gravar os elementos em um arquivo chamado matriz_3.txt.

Compilação:

```
gcc sequencial.c -o sequencial
```

Execução:

```
./sequencial matriz_1.txt matriz_2.txt
```

3.3. Paralelo Threads

Programa para multiplicar duas matrizes com paralelização por threads. Deverão ser fornecidas, na linha de comando, duas matrizes (arquivos .txt) e um valor que corresponde ao número de elementos (P) que cada thread irá calcular. Cada thread é responsável por calcular e gerar um arquivo com parte da matriz resultado. P será obtido por [n1xm2 / N]. Em que N corresponde ao número de threads ou arquivos que se deseja criar.

Compilação:

```
gcc paralelo_threads.c -o paralelo_threads -lm
```

Execução:

```
./paralelo_threads matriz_1.txt matriz_2.txt <valor de P>
```

3.4. Paralelo Processos

Semelhante ao anterior, mas que faz a implementação através da criação de processos ao invés de threads.

Compilação:

```
gcc paralelo_processos.c -o paralelo_processos -lm
```

Execução:

```
./paralelo_processos matriz_1.txt matriz_2.txt <valor de P>
```

3.5. ORIENTAÇÕES

Felizmente, pra facilitar a vida, as implementações foram automatizadas com o uso do Makefile. Para executar, basta digitar: “**make e1**” ou “**make e2**” para as implementações E1 ou E2 respectivamente. ;-)

4. Implementações

4.1. Implementação E1

Nesta implementação deverão ser executados os três programas (sequencial, threads e processos) com os tamanhos de matrizes: 150x150, 300x300, 600x600, 1200x1200, 2400x2400. Utilizando um valor de N = 8 no caso dos programas paralelos, ou seja, P = [n1xm2 / 8].

Repetir 10 vezes cada multiplicação.

Analisar resultados.

4.2. Implementação E2

Utilizando as matrizes de tamanho 2400x2400, realizar a multiplicação por paralelização com threads e processos, desta vez, utilizando diferentes tamanhos de N: 2, 4, 8, 16 e 32.

Repetir 10 vezes para cada N.

Analisar resultados.

5. Resultados

E1)

Os resultados obtidos na implementação E1 possibilitaram observar que os tempos de processamento nos programas de multiplicação paralela foram bem inferiores aos de multiplicação sequencial (Tabela 1).

Tempo médio de execução para P = n1xm2 / 8			
Matriz Resultado	Paralelo_Processos	Paralelo_Threads	Sequencial
150 x 150	0,0s	0,0s	0,0s
300 x 300	0,2s	0,0s	0,2s
600 x 600	1,0s	1,0s	2,0s
1200 x 1200	4,6s	4,6s	15,4s
2400 x 2400	40,0s	38,6s	143,2s

Tabela 1: Resultados da implementação E1

Pode-se observar também que a evolução da curva de tempo médio de execução em relação ao tamanho da matriz apresentou uma tendência exponencial de crescimento, com coeficiente de determinação muito próximo de 100% (Gráfico 1).

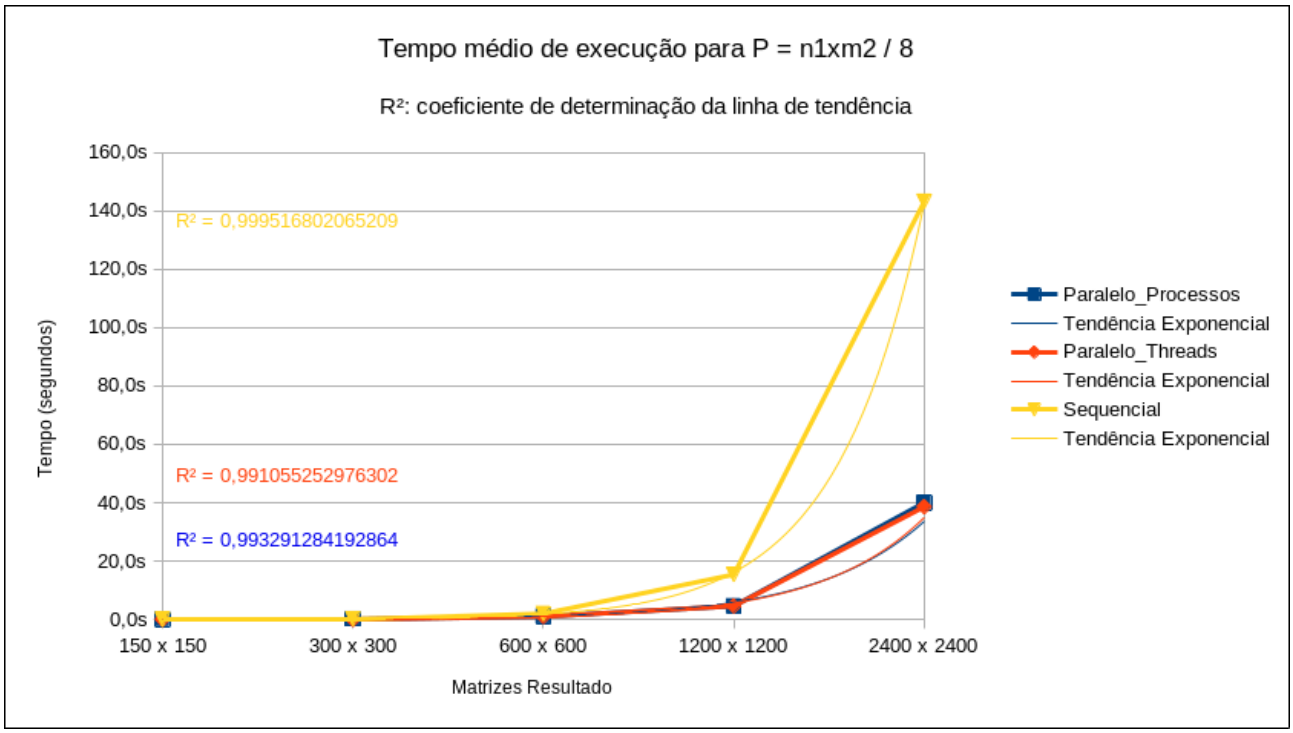

Figura 1: Tempo médio de execução para P = n1xm2 / 8

E2)

Na implementação E2 foi possível observar que na paralelização, tanto com threads quanto com processos, os tempo de execução são menores quando o valor de N (quantidade de threads/processos) é maior ou igual a 8, com pouca variação entre os respectivos tempos (Tabela 2).

Matriz: 2400 x 2400		
P	Paralelo_Processos	Paralelo_Threads
180000 = [n1xm2/32]	43,8	41
360000 = [n1xm2/16]	41,6	39
720000 = [n1xm2/8]	43,4	39,4
1440000 = [n1xm2/4]	50,2	50
2880000 = [n1xm2/2]	79,4	79,8

Tabela 2: Execuções Paralelas

Já nas execuções com N menor que 8, os tempos aumentaram conforme aquele diminuía, em uma tendência aparentemente exponencial a partir do N = 8 pra baixo (Gráfico 2).

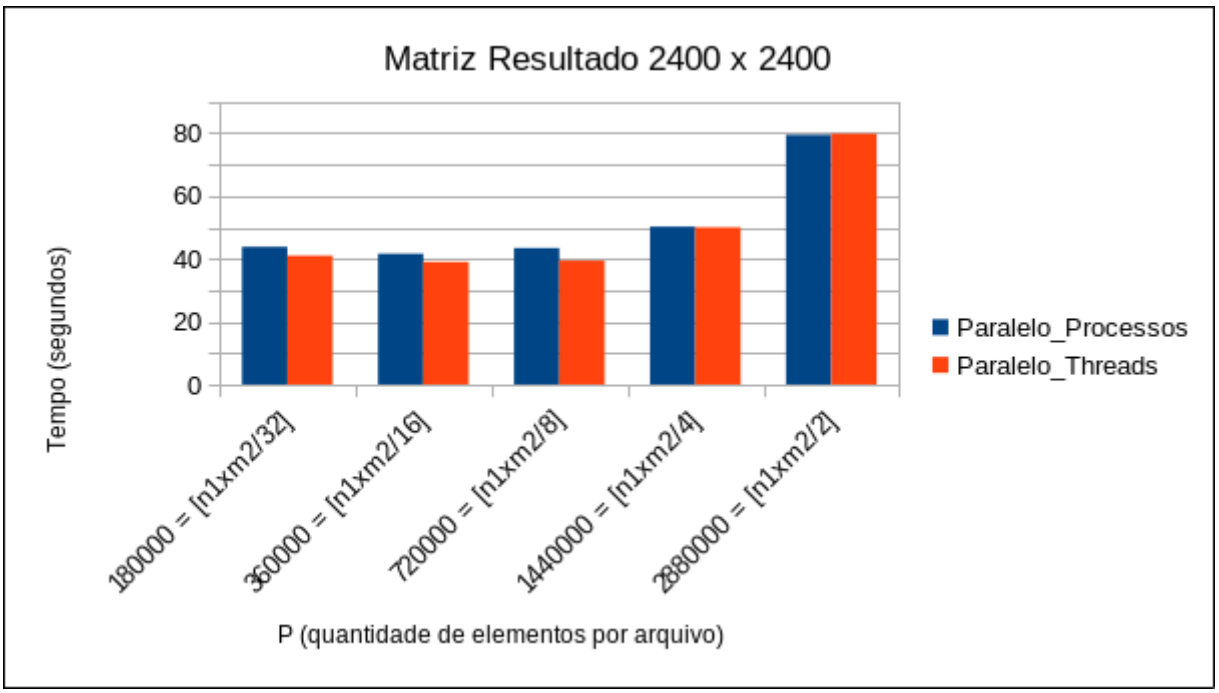

Figura 2: Relação Tempo vs Paralelização

Cabe destacar que as paralelizações com threads e processos tiveram desempenhos muito semelhantes, no entanto, percebe-se uma leve vantagem de eficiência para threads.

6. Análise dos Resultados

Tendo em vista os resultados e observando o comportamento das CPUs no monitor de sistema, pode-se concluir que:

Na execução sequencial o sistema utiliza apenas uma unidade de CPU por vez, não passando de 20% da capacidade total de processamento da máquina (Figura 3).

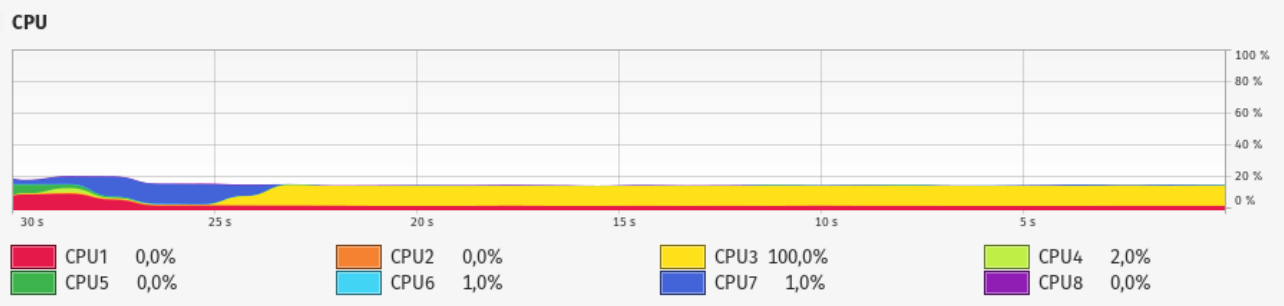

Figura 3: Sequencial

Já na multiplicação por processos ou threads, o sistema utiliza todas as CPUs em capacidade máxima quando N é igual ou maior que a quantidade daquelas (Figura 4). Percebe-se aqui, que a partir de N = 8, não se obtém maiores ganhos em aumentar a paralelização, o que indica que N = #CPUs otimiza o trabalho da máquina e do usuário, tendo em vista que aumentar a paralelismo implica em mais computação sem retorno em eficiência.

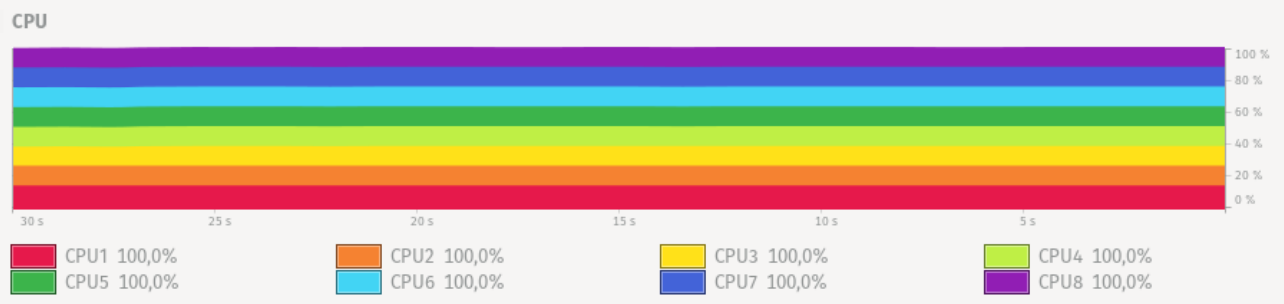

Figura 4: Processos com N = 8

Quando o valor de N é menor que o #CPUs, o sistema continua a utilizar todas estas, mas não a 100% de suas capacidades, indicando uma perda de eficiência por subutilização do sistema (Figura 5).

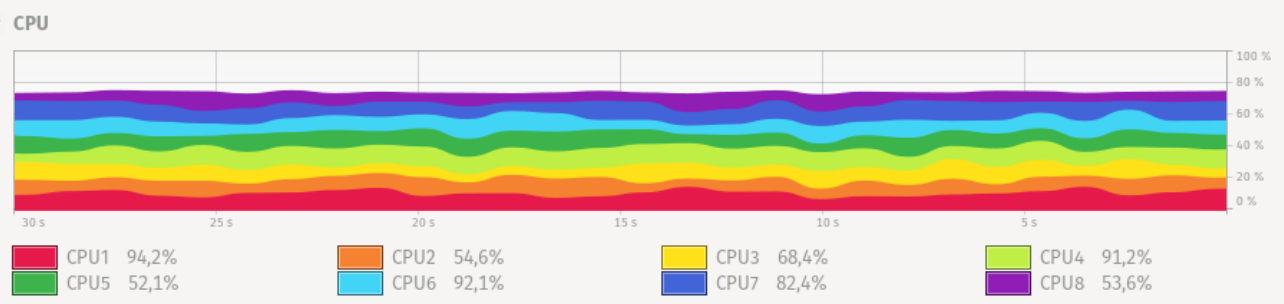

Figura 5: Processos com N = 4

Tal perda se justifica pois ao enviar o processo para mais de uma CPU, ambas terão de interagir entre si e buscar informações em espaços de memória mais distantes que as contidas na sua própria unidade, estas mais próximas e de mais fácil acesso.

Conforme mencionado anteriormente, a eficiência do programa diminui exponencialmente à medida que se reduz N abaixo do #CPUs, como se pode observar na Figura 6. O valor foi diminuído para 2, ocasionando a utilização de apenas 40% da capacidade do sistema.

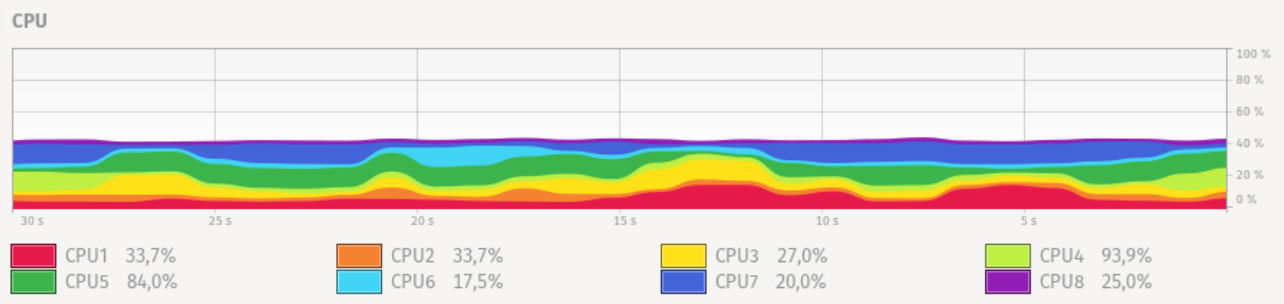

Figura 6: Processos com N = 2

Assim como observado anteriormente, a análise do monitor de sistema também mostrou uma leve vantagem de eficiência para threads, em comparação com processos, quando a paralelização foi inferior ao número de CPUs.

7. Conclusão

Conclui-se que a utilização de threads e processos contribui para a eficiência do programa à medida que a tarefa possa ser dividida em um número igual ou superior às unidades de processamento, pois possibilita que toda a execução seja feita integralmente alí, sem que haja necessidade de acessos a memórias mais distantes ou interações com outros processos.

Em relação à vantagem de threads em relação a processos, penso que isso se justifica pois a primeira possui uma estrutura mais “leve” de dados, já que uma thread não cria novos processos e, apesar de serem independentes entre si, compartilham o mesmo processo pai que as criou. Já o paralelismo por processos envolver a criação de inteiras estruturas de dados novas, exigindo mais capacidade de memória e computação do sistema.

Por fim, cabe destacar que a arquitetura da máquina influencia diretamente a performance do programa. Sem multiprocessamento, não é possível se beneficiar pela paralelização.