UNIVERSIDADE FEDERAL DE SÃO PAULO
INSTITUTO DE CIÊNCIA E TECNOLOGIA
BACHARELADO EM CIÊNCIA E TECNOLOGIA

Laboratório 04

Semáforos, monitores e variáveis de condição

Nomes: Ana Júlia de Oliveira Bellini **RA:** 111774
Willian Dihanster Gomes de Oliveira **RA:** 112269

SÃO JOSÉ DOS CAMPOS
2018

Especificações da Máquina

Para os experimentos realizados em cada um dos exercícios propostos, as especificações da máquina utilizada são descritas a seguir:

Processador: Intel (R) Core (TM) i5-7200U CPU @ 2.50GHz
Núcleos físicos: 2
Memória Cache: 3Mb L3
Memória RAM: 8 GB
Threads: 4
Hyperthreading: Sim
Sistema Operacional: Ubuntu 18.04.1 64 bits
Compilador: gcc 7.3.0

Exercício 1

Desenvolva programas concorrentes em C/PThreads e Java para calcular o Produto Escalar entre 2 vetores (do tipo double) de tamanho N:

$$PE = A1*B1 + A2*B2 + ... + AN*BN$$

Os vetores devem ser preenchidos aleatoriamente e sua alocação e preenchimento não deverão influenciar as medidas de desempenho.

O algoritmo deve ser implementado da seguinte forma: cada thread deve calcular valores parciais, sendo que a totalização deve ocorrer ao final do cálculo, ainda dentro das threads, em uma seção crítica controlada por semáforo, a qual atualiza uma variável global.

Faça um gráfico com o Speed-up e a eficiência obtido considerando N com tamanho de 10^5 e 10^7, e o número de threads de 1, 2, 4 e 8.

Resultados do Exercício 1

Para a versão implementada em C/PThreads, foram obtidos os seguintes valores de Speed-up e eficiência:

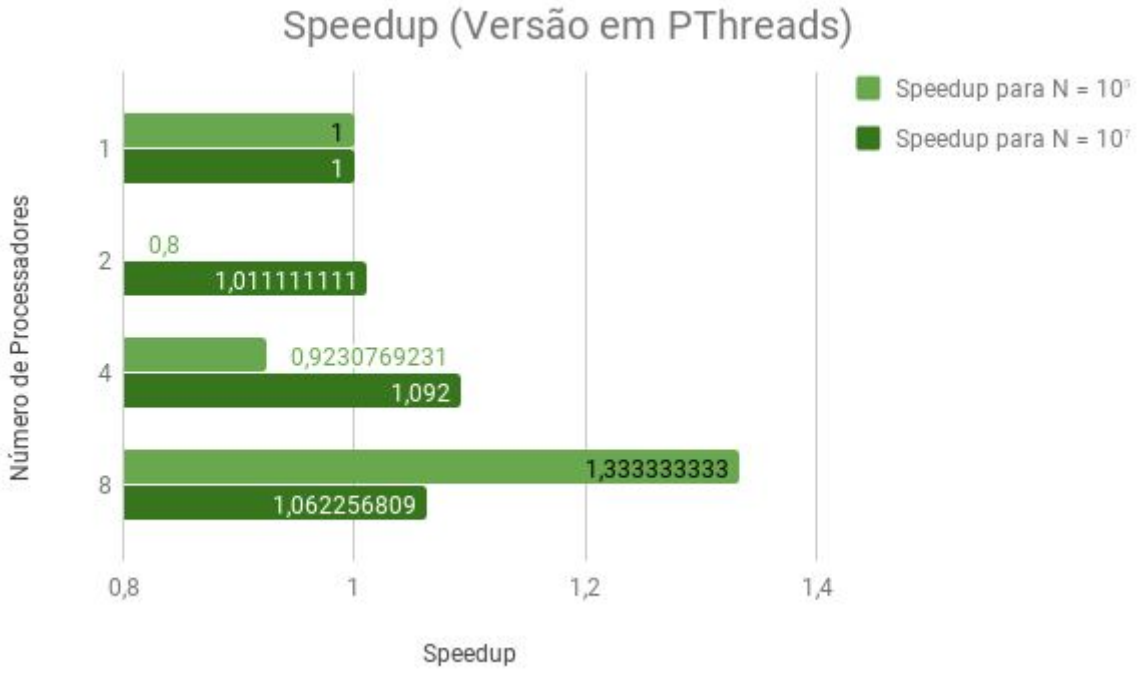


Figura 1: Resultados de Speedup obtidos com a versão do código implementada em PThreads.

Figura 2: Resultados de eficiência obtidos com a versão do código implementada em PThreads.

Já para a versão implementada com JavaThreads, estes foram os resultados:

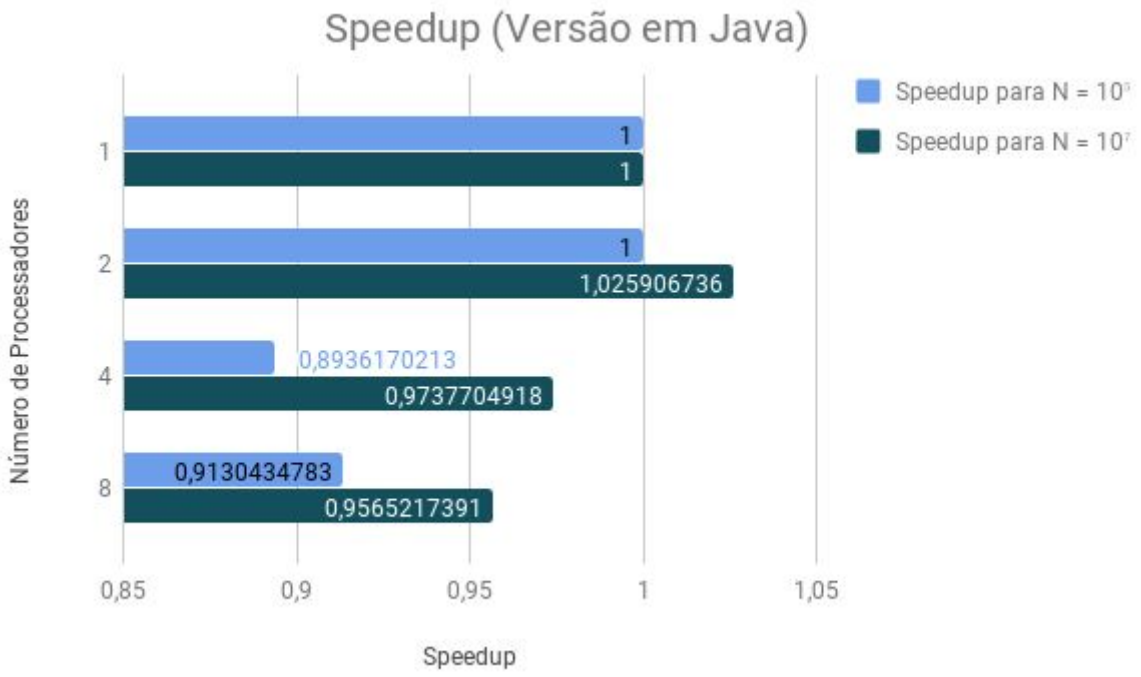


Figura 3: Resultados de Speedup obtidos com a versão do código implementada em Java.

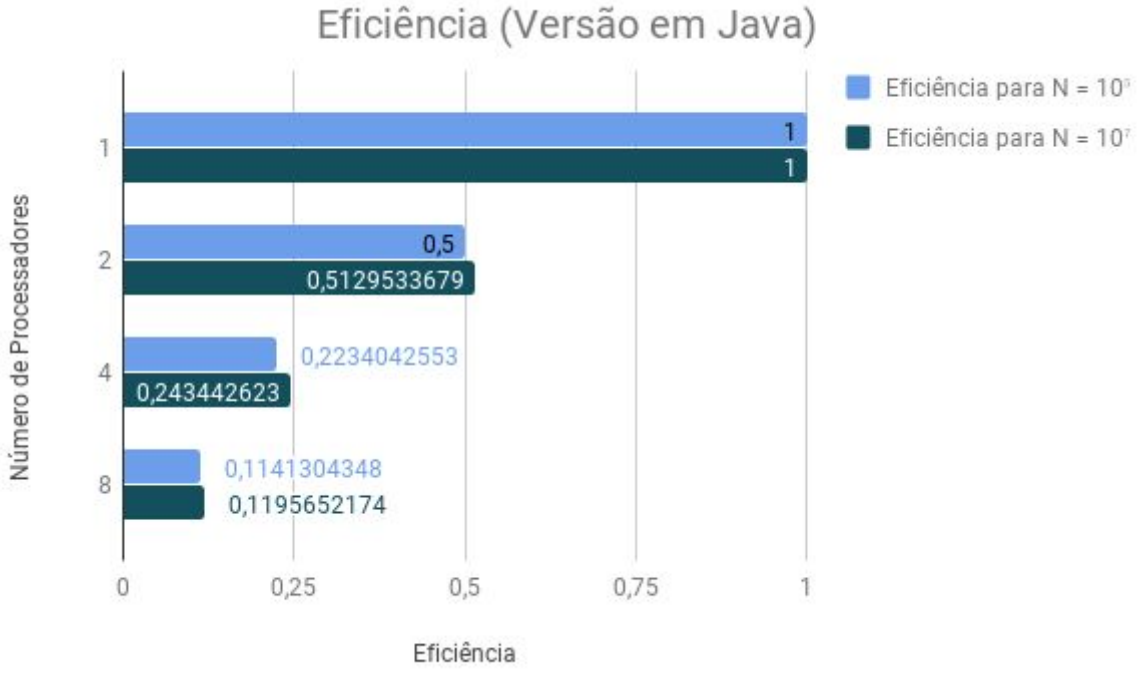


Figura 4: Resultados de eficiência obtidos com a versão do código implementada em Java.

Exercício 2

Faça o mesmo que o exercício 1 com um programa em OpenMP usando:

a) seção crítica (#pragma omp critical) para controlar a totalização dos resultados parciais em uma variável global;

b) redução (reduction (...)).

Faça um gráfico com o Speed-up e a eficiência obtido de ambos.

Resultados do Exercício 2

Utilizando seção crítica, foram obtidos os seguintes Speedups e eficiências:

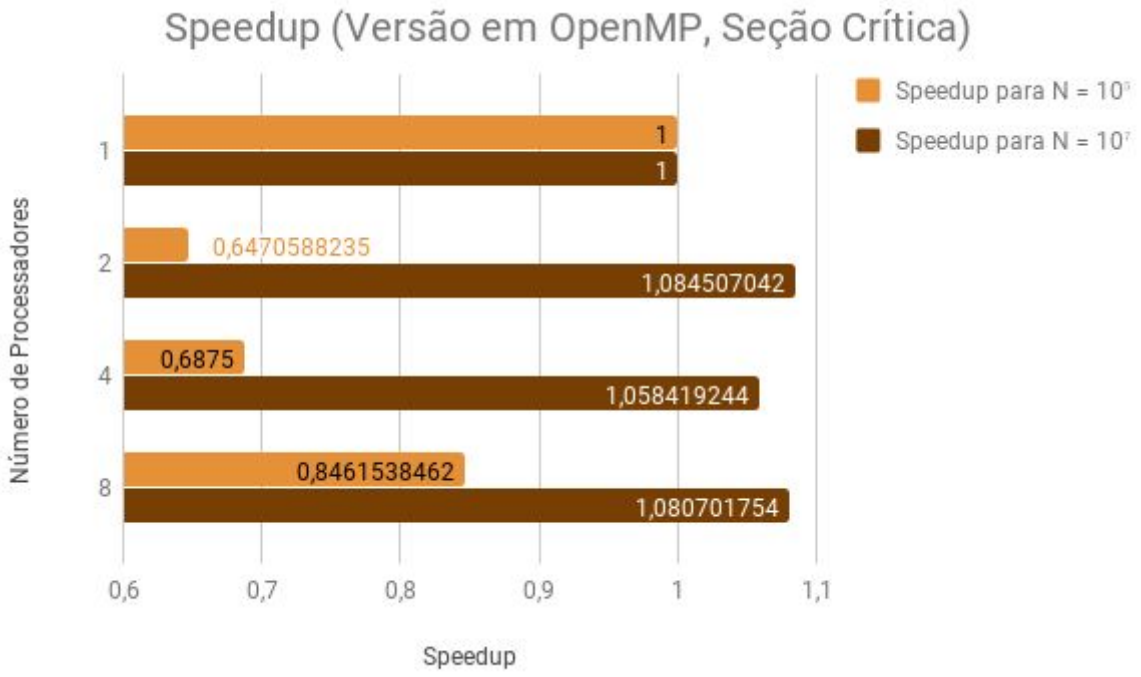


Figura 5: Resultados de Speedup da versão do código implementada em OpenMP, utilizando seção crítica.

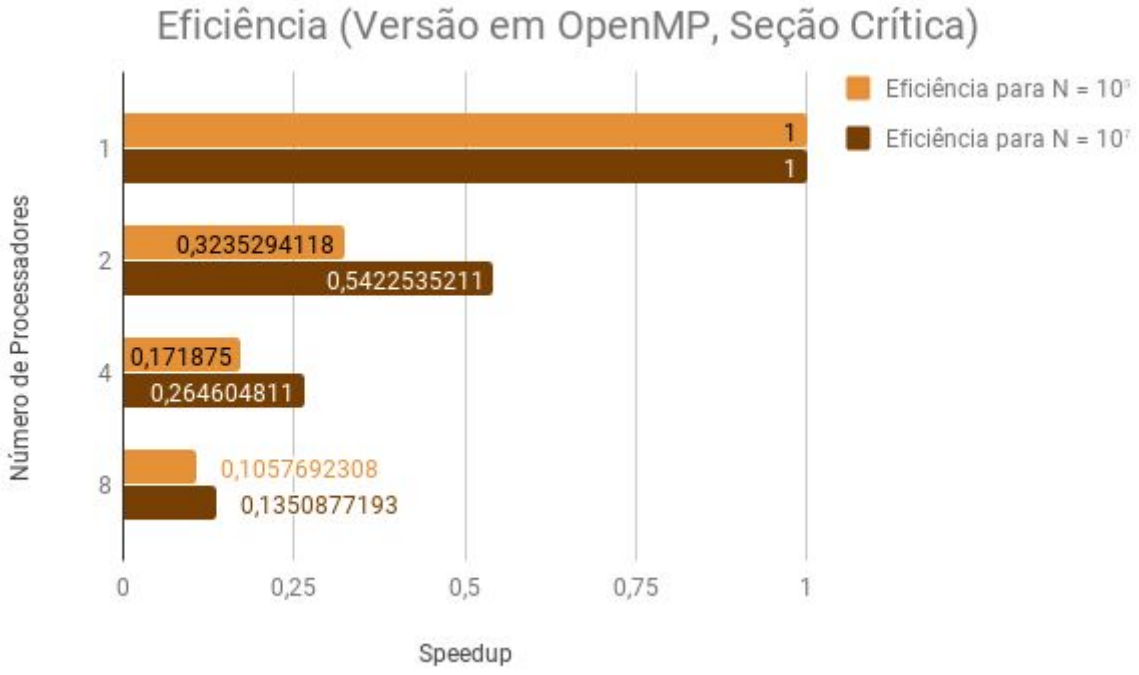


Figura 6: Resultados de eficiência da versão do código implementada em OpenMP, utilizando seção crítica.

E com redução, os resultados foram:

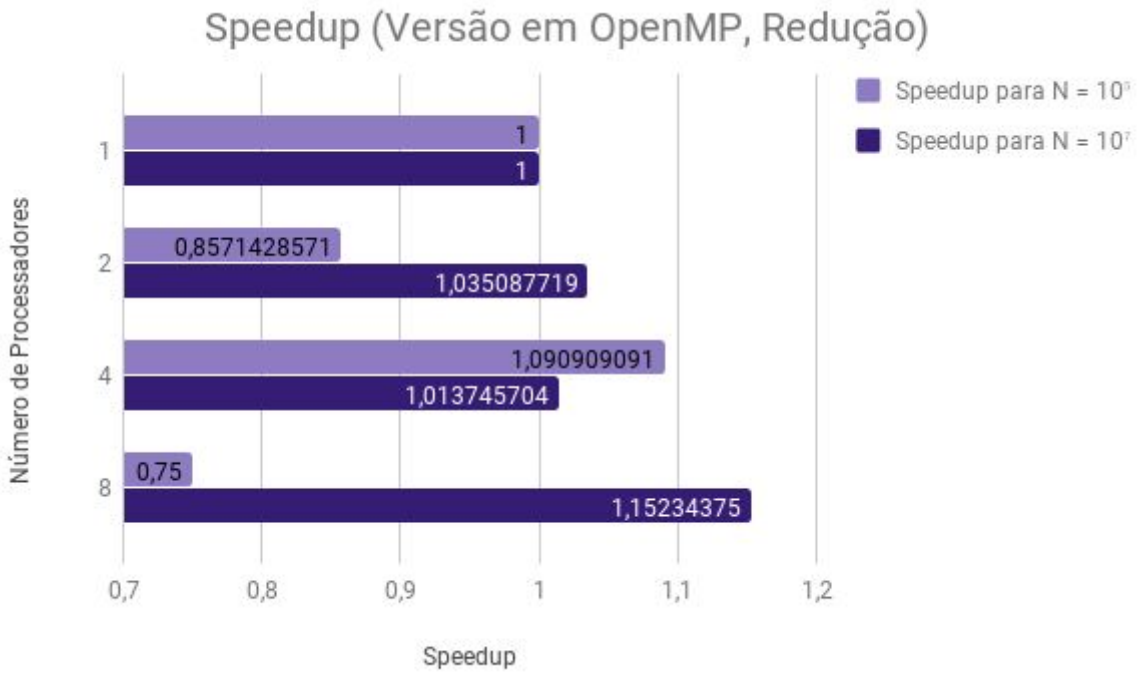


Figura 7: Resultados de Speedup da versão do código implementada em OpenMP, utilizando redução.

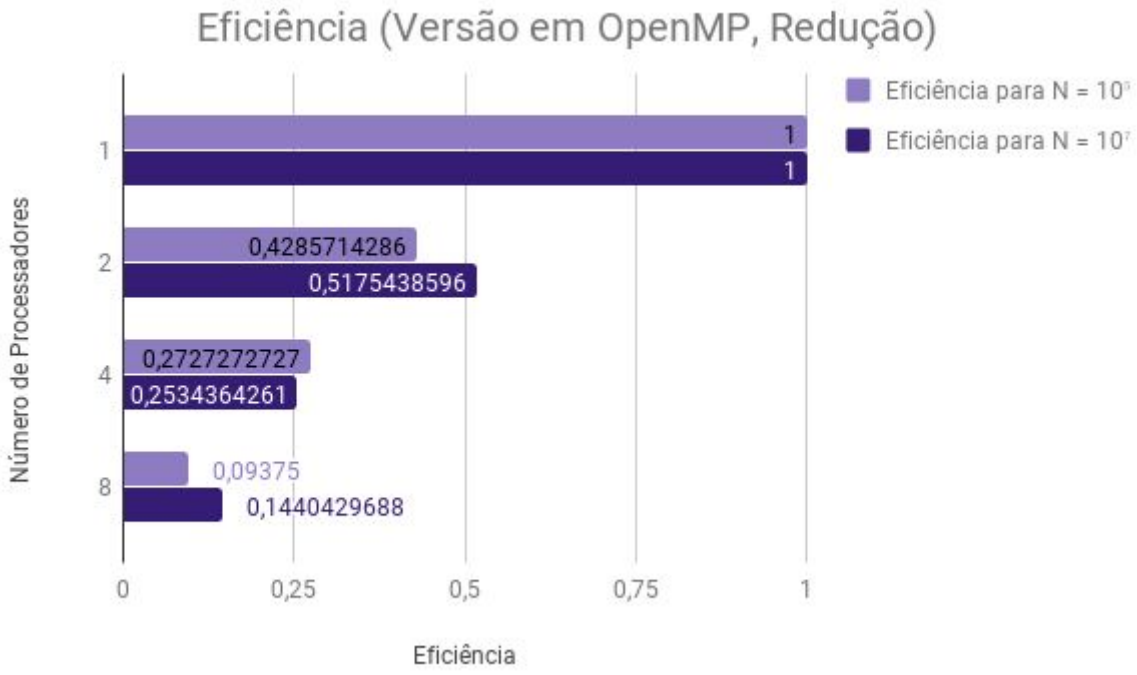


Figura 8: Resultados de eficiência da versão do código implementada em OpenMP, utilizando redução.

Exercício 3

Crie um programa multi-threads com Java que modifique o exercício 1, de forma a usar um código monitor para resolver o mesmo problema. Desta forma, deve-se usar um método contendo a palavra reservada "synchronized", para atualizar o valor do produto escalar após o cálculo dos valores parciais obtidos pelas threads. Faça um gráfico com o Speed-up e a eficiência obtido. Teste com os mesmos parâmetros estabelecidos no exercício original.

Resultados do Exercício 3

Neste exercício, foram obtidos os seguintes resultados:

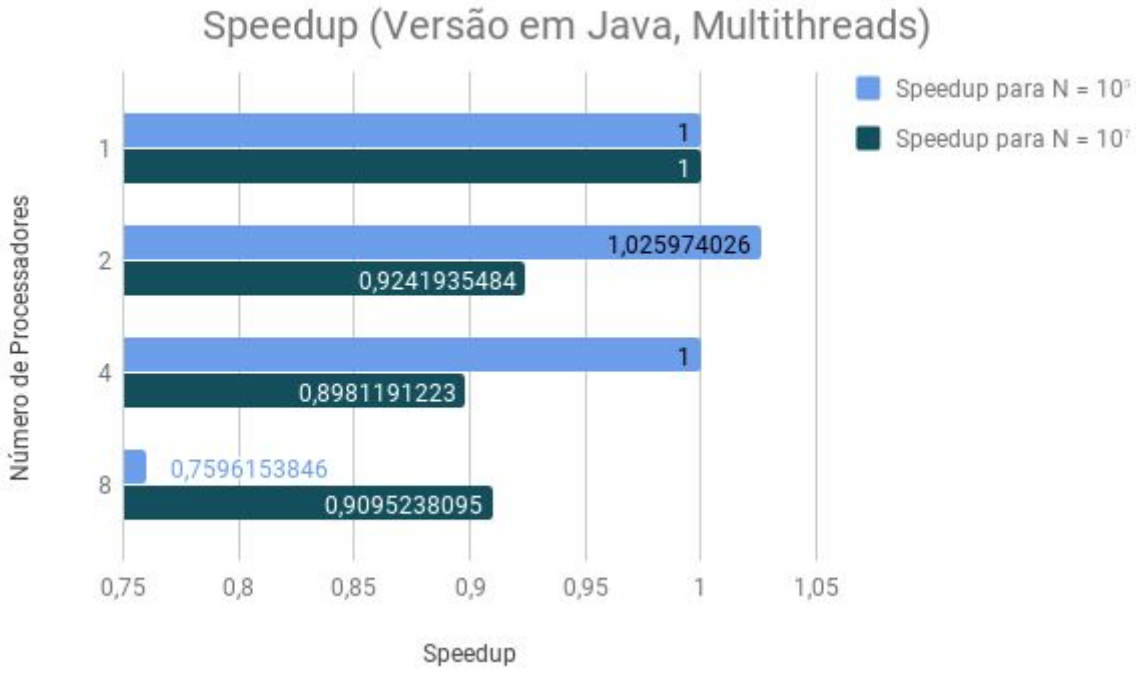


Figura 9: Resultados de Speedup obtidos com a versão do código implementada em Java, com multithreads.

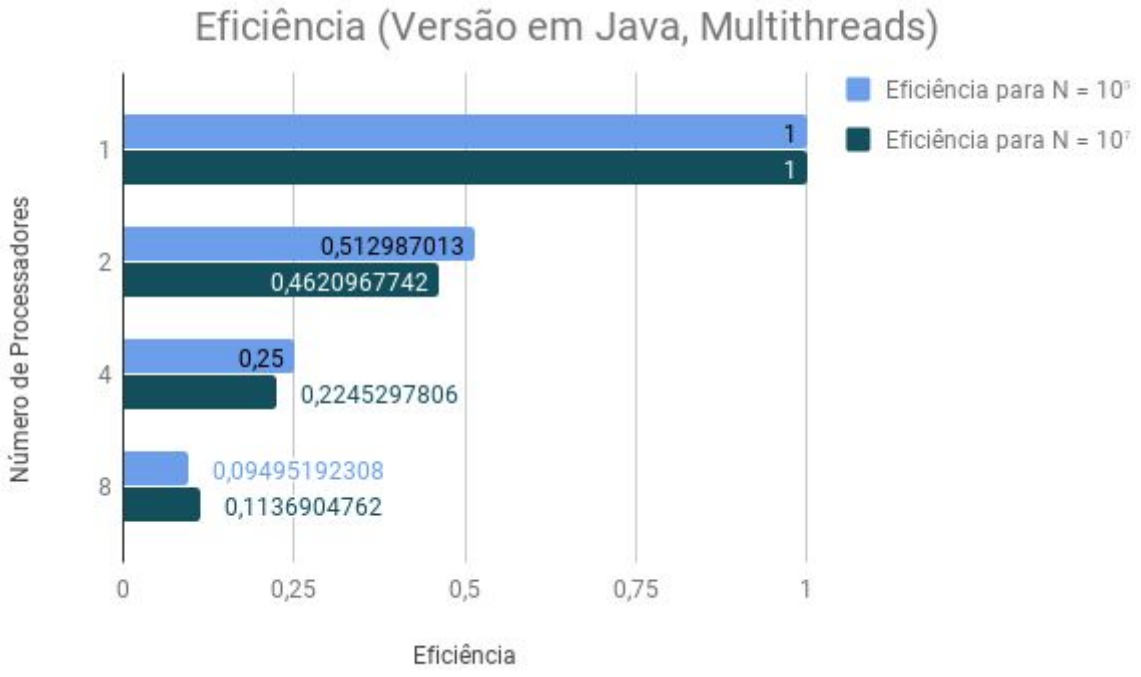


Figura 10: Resultados de eficiência obtidos com a versão do código implementada em Java, com multithreads.

Exercício 4

Considere dois processos concorrentes:

P1	P2
Print(C) Print(E)	Print(A) Print(R) Print(O)

Faça os seguintes exercícios:

a) Crie um pseudo-código que utilize semáforos para que seja possível imprimir a sequência: ACERO ou ACREO. Obs: Não se esqueça de determinar o(s) valor(es) inicial(is) do(s) semáforo(s).

b) Mostre, através de um diagrama de estados ou descrição tabular, que o código funciona como especificado no item a.

c) Implemente o programa em linguagem C/PThreads ou Java que faça o que o pseudo-código determine, e permita imprimir 10 sequências dos "Prints".

Resultados do Exercício 4

Temos o seguinte pseudocódigo:

```
Semáforos Inicio = 1, S1 = 0, S2 = 0

    P1              P2
wait(S1)        wait(inicio)
print(C)        print(A)
signal(s2)      signal(S1)
print(E)        wait(S2)
signal(S2)      print(R)
                wait(s2)
                print(O)
                signal(inicio)
```

Para o pseudocódigo criado acima, temos a seguinte descrição tabular:

i) Para o caso “ACERO”:

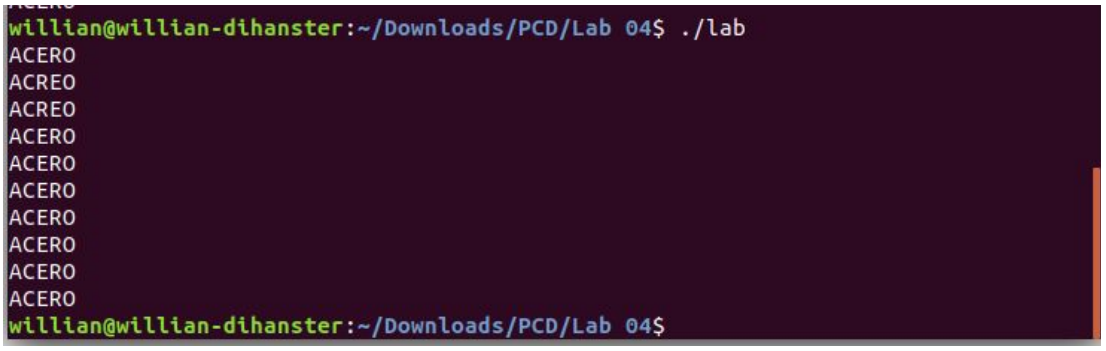
C2	C1	Início	S1	S2	P1	P2
1	1	1	0	0	wait(inicio)	
2	1	1	0	0	print(A)	
3	1	1	0	0	signal(S1)	
4	1	1	1	0	wait(s2)	

4	**2**	**1**	**1**	**0**		**wait(S1)**
4	**3**	**1**	**1**	**0**		**print(C)**
4	**4**	**1**	**1**	**0**		**signal(S2)**
4	**5**	**1**	**1**	**1**		**print(E)**
4	**6**	**1**	**1**	**1**		**signal(S2)**
5	**6**	**1**	**1**	**1**	**print(R)**	
6	**6**	**1**	**1**	**1**	**wait(S2)**	
7	**6**	**1**	**1**	**1**	**print(O)**	
8	**6**	**1**	**1**	**1**	**signal(inicio)**	

ii) Para o caso “ACREO”

C1	C2	Início	S1	S2	P1	P2
1	**1**	**1**	**0**	**0**	**wait(inicio)**	
2	**1**	**1**	**0**	**0**	**print(A)**	
3	**1**	**1**	**0**	**0**	**signal(S1)**	
4	**1**	**1**	**1**	**0**	**wait(s2)**	
4	**2**	**1**	**1**	**0**		**wait(S1)**
4	**3**	**1**	**1**	**0**		**print(C)**
4	**4**	**1**	**1**	**0**		**signal(S2)**
5	**4**	**1**	**1**	**1**	**wait(S2)**	
6	**4**	**1**	**1**	**1**	**print(R)**	
6	**5**	**1**	**1**	**1**		**print(E)**
6	**6**	**1**	**1**	**1**		**signal(S2)**
7	**6**	**1**	**1**	**1**	**wait(s2)**	
8	**6**	**1**	**1**	**1**	**print(O)**	
9	**6**	**1**	**1**	**1**	**signal(inicio)**	

Pode-se conferir na figura a seguir, os resultados para o algoritmo gerado. Como o esperado, foram impressos na tela exemplos de “ACERO” e “ACREO”.



Exercício 5

Considere um programa concorrente, utilizando C+PThreads, ou C+OpenMP, ou ainda JavaThreads, onde existem duas categorias de processos/threads. Os processos da primeira categoria, denominados TPs, deverão produzir N números inteiros aleatórios (todos na faixa entre 1 e 10^8) e inseri-los em uma determinada fila. Os demais processos de outro tipo, denominados TCs, deverão retirar (se houver) dados da fila em comum e verificar se os mesmos são números primos.

Caso os TCs recebam números negativos os mesmos deverão ser finalizados.

Simule o funcionamento do sistema nas seguintes configurações:

a) TP=1, TC=2, N=100

b) TP=2, TC=4, N=100

c) TP=4, TC=2, N=100

Onde N representa a quantidade total de números gerados pelos TPs.

Obs: Considere que após a quantidade total de números aleatórios gerados for igual a N os TPs deverão emitir valores negativos para a quantidade exata de TCs, de forma a encerrá-los e também encerrar-se posteriormente.

Demonstre em relatório o funcionamento correto do código desenvolvido para todos os casos requeridos.

Resultados do Exercício 5

Nas figuras a seguir é possível conferir o funcionamento do algoritmo desenvolvido para os três casos. Nota-se que para todos os casos, os resultados saíram como o esperado. Onde os N números são produzidos e consumidos pelos produtores e consumidores, respectivamente. Além disso, verificam se o número lido é primo ou não e quando os N números foram produzidos, as outras threads foram encerradas, lendo o -1 emitido pela thread consumidor.

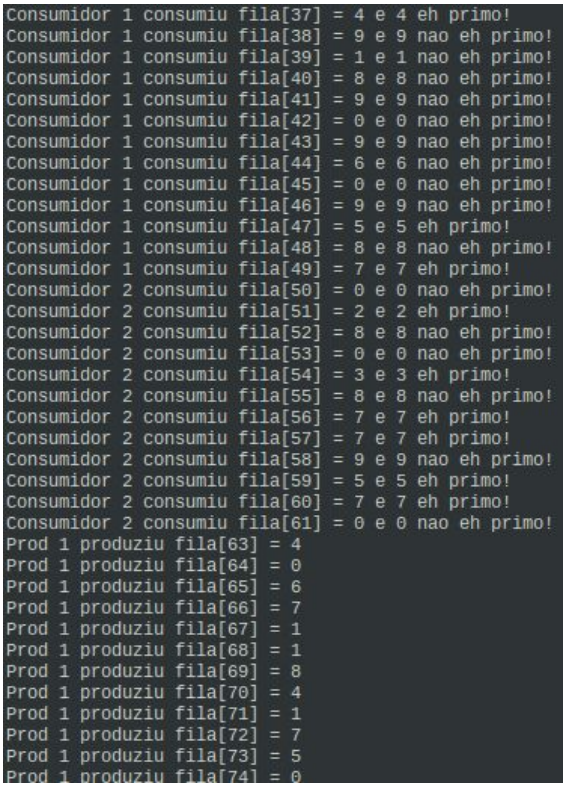


Figura 11: Resultados obtidos no caso A.

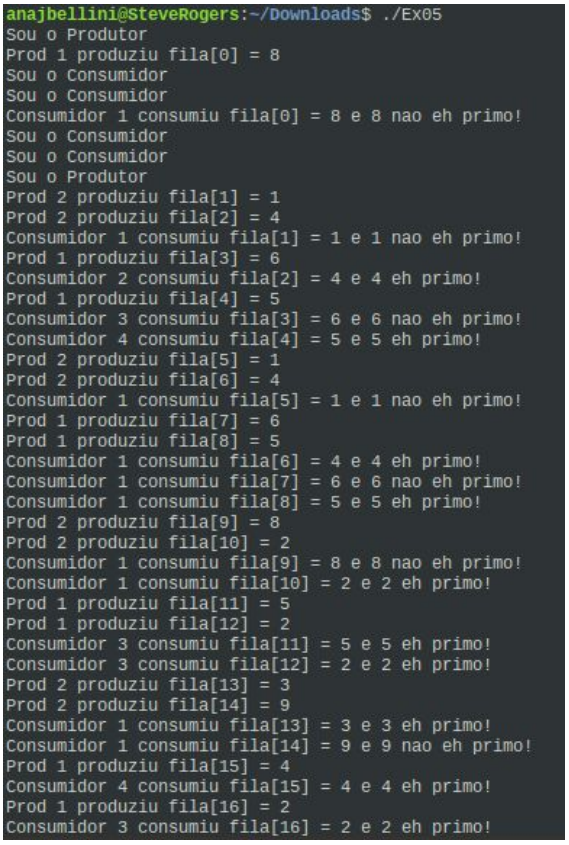


Figura 12: Resultados obtidos no caso B.

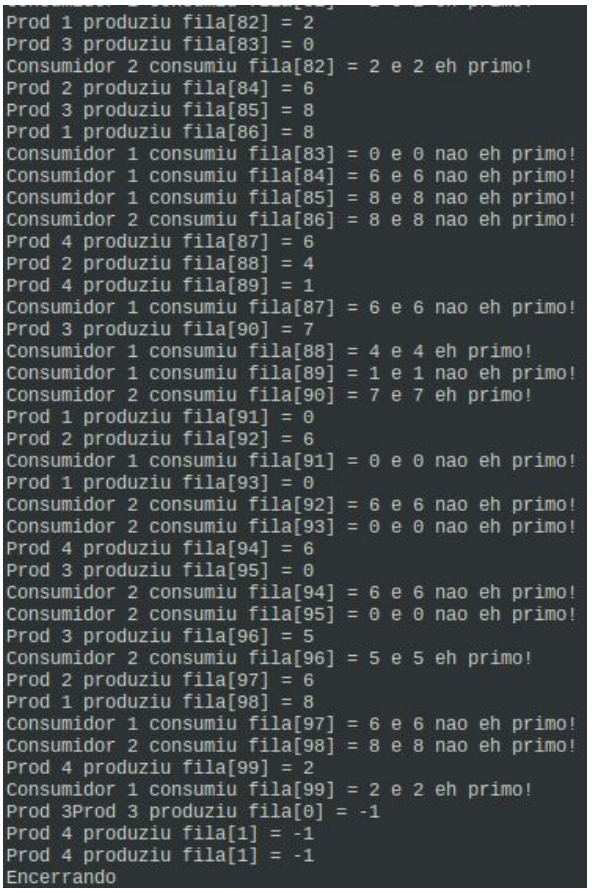


Figura 13: Resultados obtidos no caso C.