UNIFESP
UNIVERSIDADE FEDERAL DE SÃO PAULO
1933

Relatório - Atividade 5
Programação Concorrente e Distribuída

Rosangela Miyeko Shigenari 92334

1. Desenvolva programas concorrentes em C/PThreads e Java, ambos utilizando-se de semáforos, 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.

a. (Programas em Anexo)
b. Calculando o tempo de execução com N de tamanho 10^5 e 10^7 com 1, 2, 4 e 8 *threads,* obtendo os valores da tabela abaixo.

O programa foi executado em uma máquina com as seguintes especificações:

- Processador Intel Core i5 dual core de 1,8 GHz
- Turbo Boost de até 2,9 GHz
- Memória LPDDR3 de 8 GB com 1600 MHz
- Armazenamento SSD de 128 GB

Em seguida, são mostrados gráficos da relação da eficiência e *speedUp* com o número de *threads* de acordo com os 2 valores de N testados.

N/ Número de *threads*	1	2	4	8
10^5	3 ms	1 ms	1 ms	1 ms
10^7	46 ms	32 ms	27 ms	31 ms

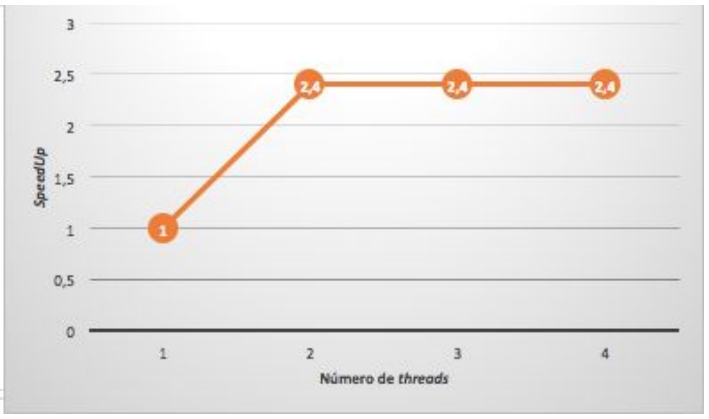


Gráfico 1: *SpeedUp* x Número de *threads* no programa em *pthread* para N = 100000

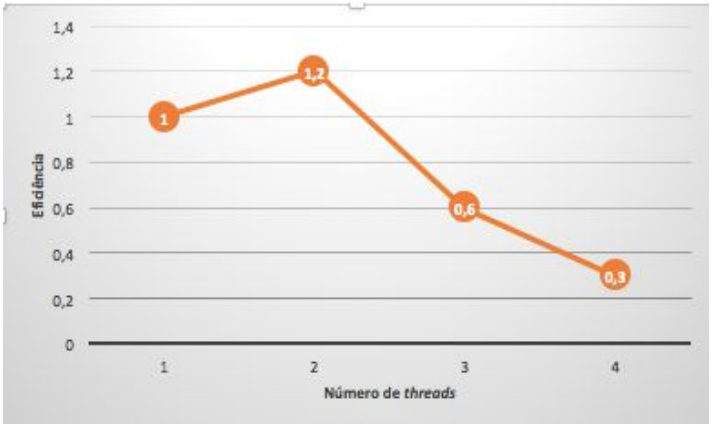


Gráfico 2: Eficiência x Número de *threads* no programa em *pthread* para N = 100000

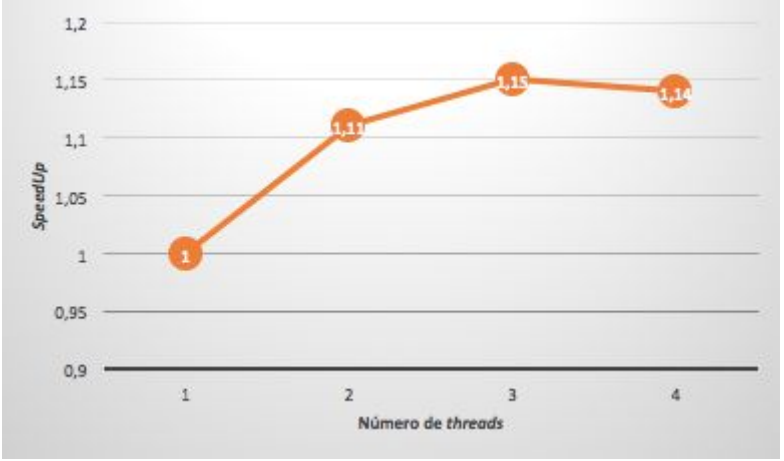


Gráfico 3: *SpeedUp* x Número de *threads* no programa em *pthread* para N = 10000000

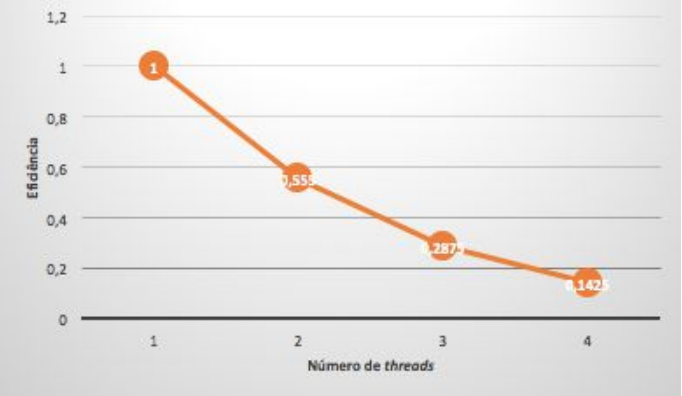


Gráfico 4: Eficiência x Número de *threads* no programa em *pthread* para N = 10000000

Para o programa em *java thread* temos os seguintes resultados

N/ Número de *threads*	1	2	4	8
10^5	3 ms	3ms	2 ms	2 ms
10^7	38 ms	33 ms	31ms	30 ms

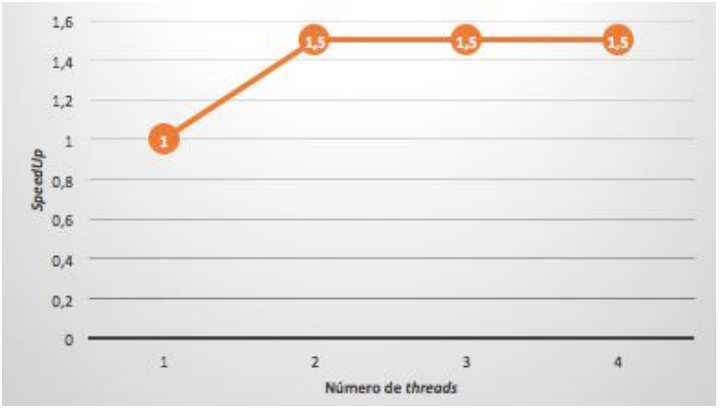


Gráfico 5: *SpeedUp* x Número de *threads* no programa em *JavaThread* para N = 100000

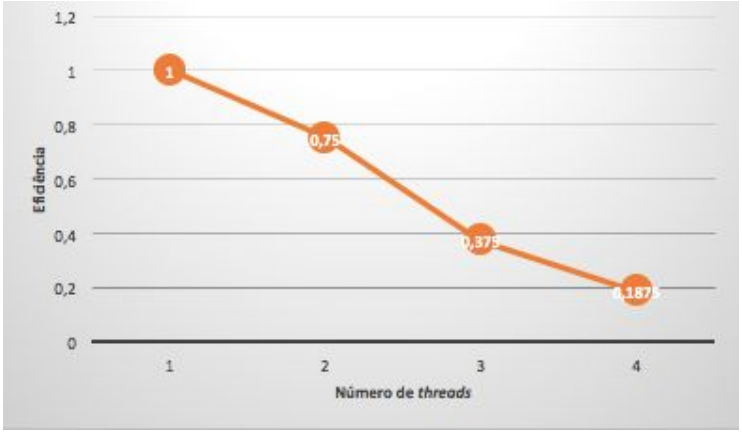


Gráfico 6: Eficiência x Número de *threads* no programa em *JavaThread* para N = 100000

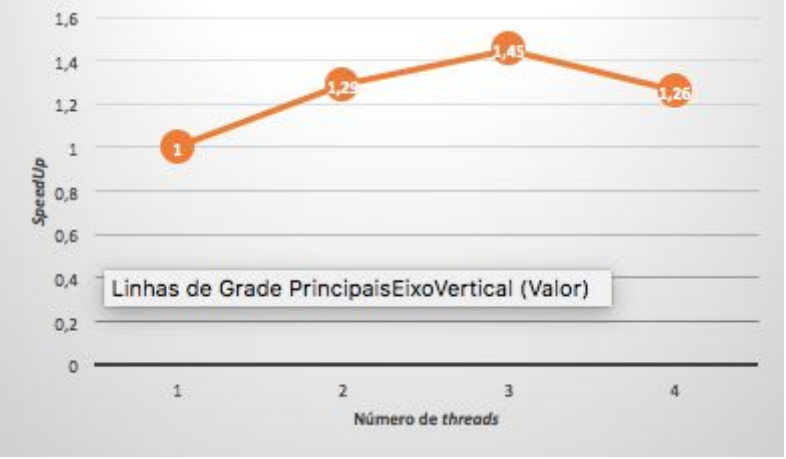


Gráfico 7: *SpeedUp* x Número de *threads* no programa em *JavaThread* para N = 10000000

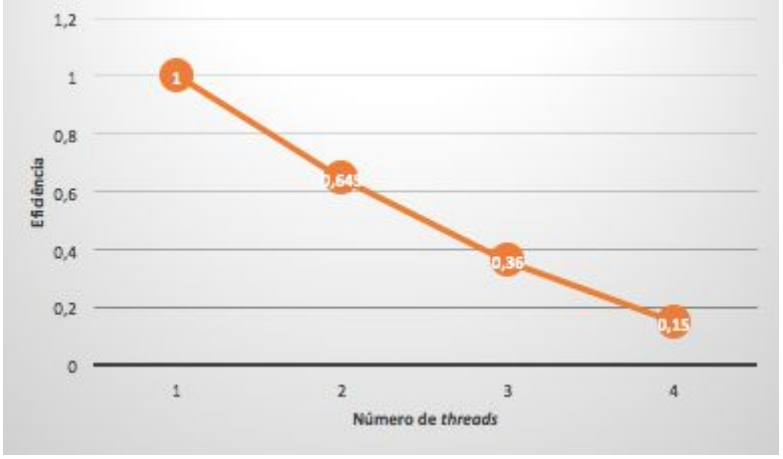


Gráfico 8: Eficiência x Número de *threads* no programa em *JavaThread* para N = 10000000

A partir da análise dos resultados, pode se observar que o programa em *pthread* possui maior desempenho levando em consideração o *speedUp* e a eficiência atingida.

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

1. a) seção crítica (#pragma omp critical) para controlar a totalização dos resultados parciais em uma variável global; (em anexo como)
2. b) redução (reduction (...)). (em anexo como)
3. Faça um gráfico com o *Speed-up* e a eficiência obtido de ambos.

A tabela seguinte mostra os valores obtidos na execução do programa para 1, 2, 4 e 8 *threads,* para N=100000 e N=10000000.

a) Para #pragma omp critical

N/ Número de *threads*	1	2	4	8
10^5	0.98 ms	0.87 ms	1.11 ms	1.06 ms
10^7	66 ms	41 ms	34 ms	32 ms

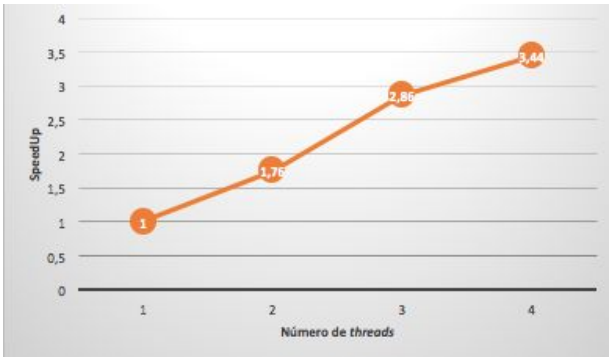


Gráfico 9: *SpeedUp* x Número de *threads* no programa em *OpenMP* com uso do *critical* para N = 100000

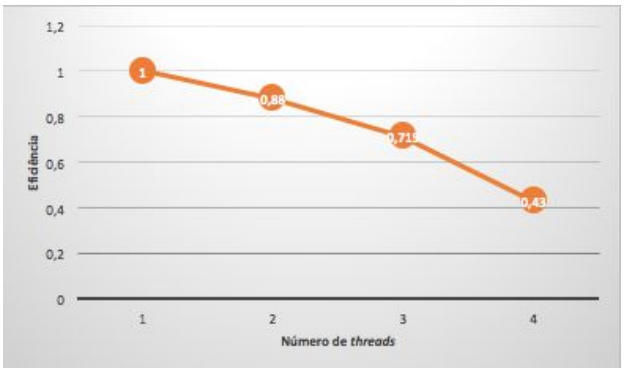


Gráfico 10: Eficiência x Número de *threads* no programa em *OpenMP* com uso do *critical* para N = 100000

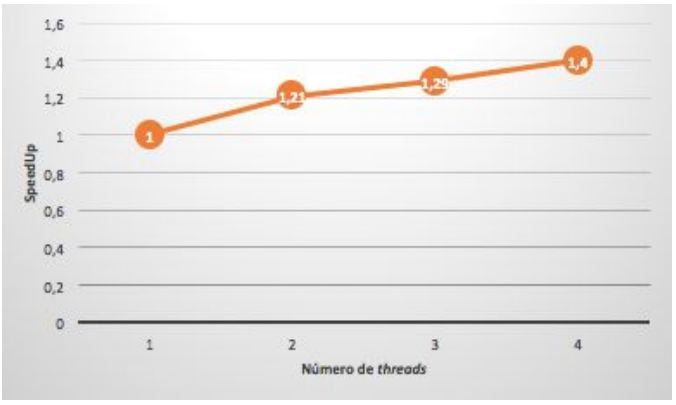


Gráfico 11: *SpeedUp* x Número de *threads* no programa em *OpenMP* com uso do *critical* para N = 10000000

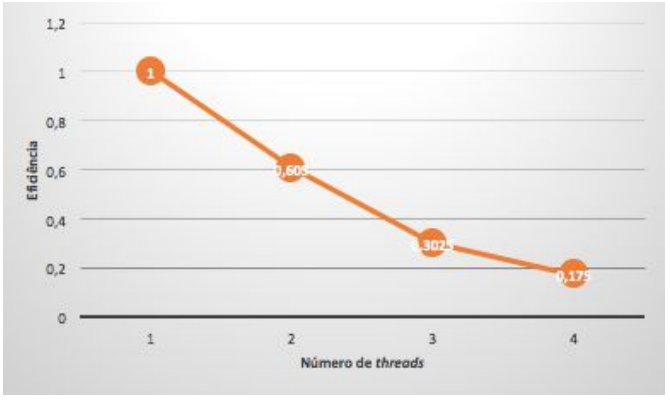


Gráfico 12: Eficiência x Número de *threads* no programa em *OpenMP* com uso do *critical* para N = 10000000

b) Para *reduction*

N/ Número de *threads*	1	2	4	8
10^5	0.71 ms	0.63 ms	0.52 ms	0.51 ms
10^7	88 ms	95 ms	113 ms	146 ms

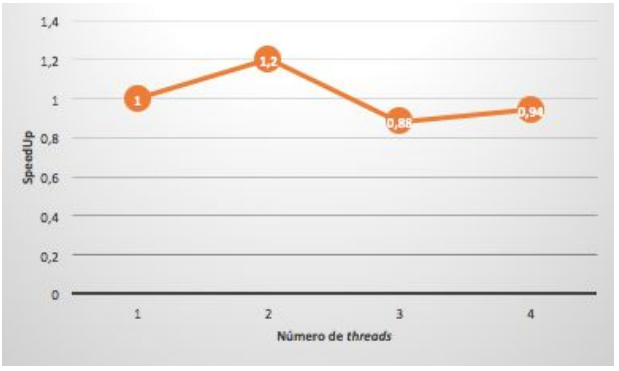


Gráfico 13: *SpeedUp* x Número de *threads* no programa em *OpenMP* com uso do *reduction* para N = 100000

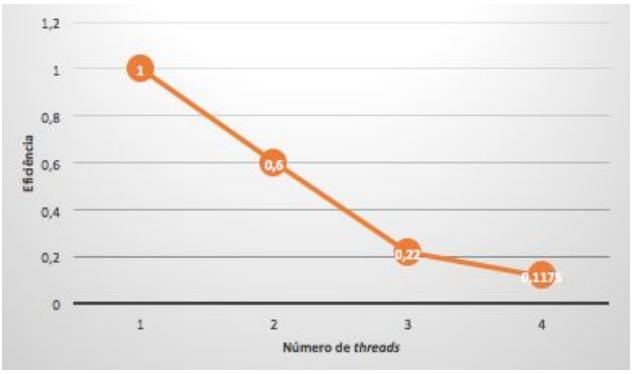


Gráfico 14: Eficiência x Número de *threads* no programa em *OpenMP* com uso do *reduction* para N = 100000

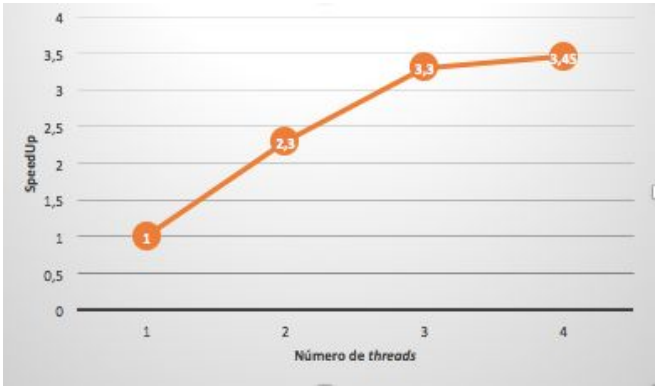


Gráfico 15: *SpeedUp* x Número de *threads* no programa em *OpenMP* com uso do *reduction* para N = 10000000

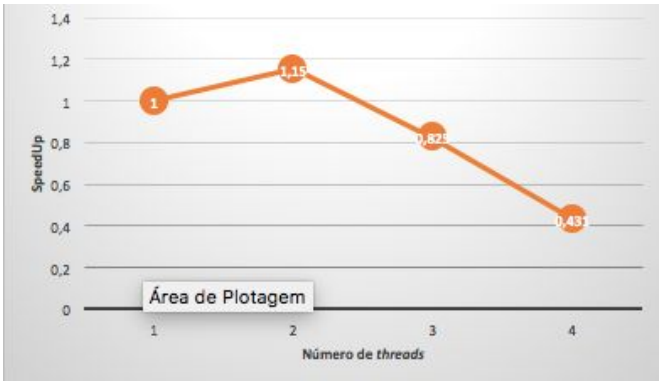


Gráfico 16: Eficiência x Número de *threads* no programa em *OpenMP* com uso do *reduction* para N = 10000000

A seção crítica é uma alternativa não eficiente. Já o *reduction* divide o código para realizar a soma, e cada uma dessas partes possui uma cópia da variável, possibilitando a execução paralela de todas as partes, não ocorrendo o mesmo na seção crítica, piorando o desempenho quando se aumentava o número de *threads.*

3. Considere dois processos concorrentes:

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

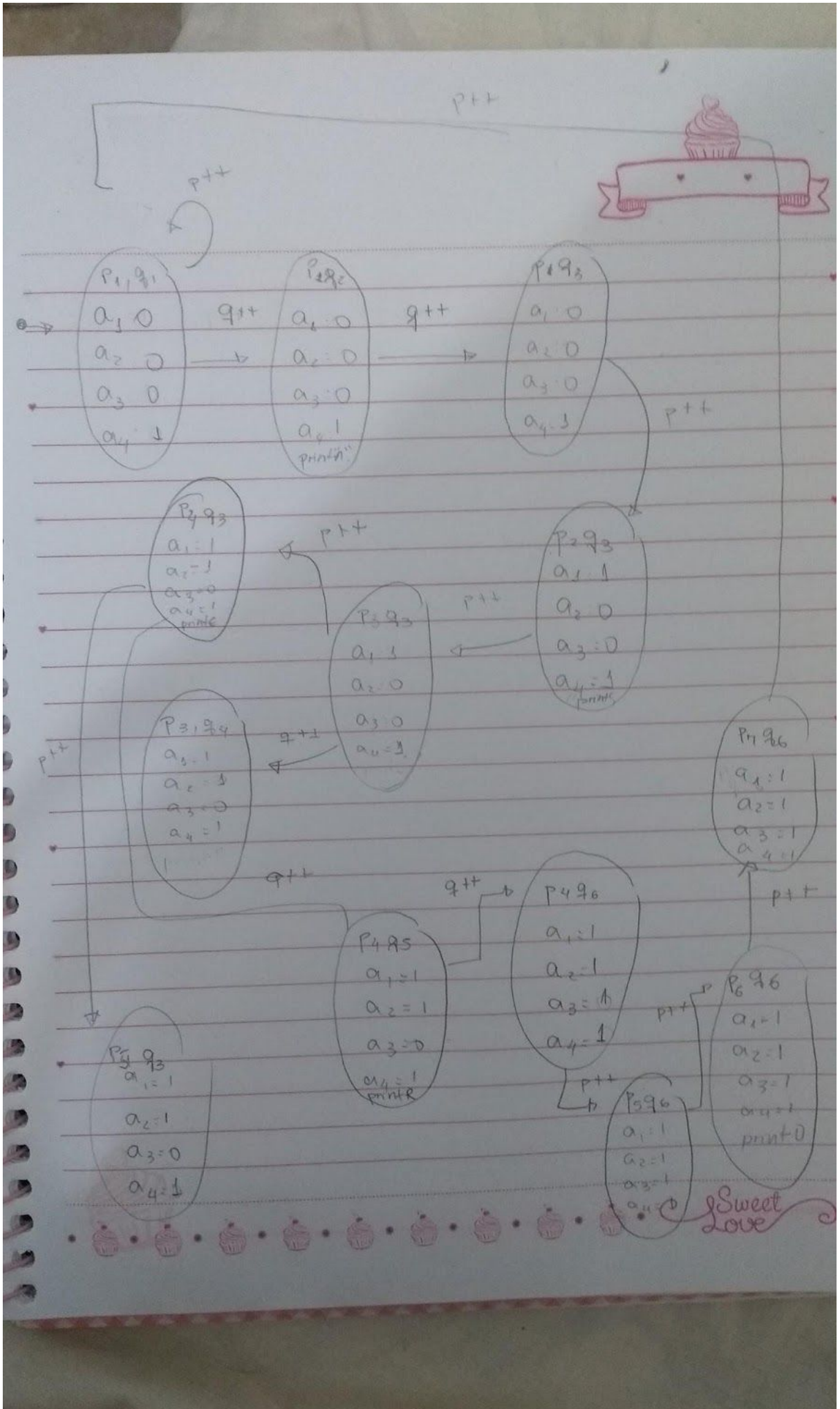
Faça os seguintes exercícios:

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).

Valores iniciais:
a1, a2, a3 são locked
a4 é unlocked

Processo A	**Processo B**
P1: wait(a1)	Q1: wait(a4)
P2: print(“C”)	Q2: print(“A”)
P3: signal(a2)	Q3: signal(a1)
P4: print(“E”)	Q4: wait(a2)
P5: wait(a3)	Q5: print(“R”)
P6: print(“O”)	Q6: signal(a3)
P7: signal(a4)	

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


p++
p++
P_1, q_1
a_1 0
a_2 0
a_3 0
a_4 1
q++
$P_1 q_2$
a_1 0
a_2 0
a_3 0
a_4 1
print"
q++
$P_1 q_3$
a_1 0
a_2 0
a_3 0
a_4 1
p++
$P_2 q_3$
a_1 1
a_2 0
a_3 0
a_4 1
print
p++
$P_2 q_3$
a_1 1
a_2 1
a_3 0
a_4 1
print
p++
$P_3 q_3$
a_1 1
a_2 0
a_3 0
a_4 1
q++
$P_3 q_4$
a_1 1
a_2 1
a_3 0
a_4 1
p++
$P_4 q_5$
a_1 1
a_2 1
a_3 0
a_4 1
printR
q++
$P_4 q_6$
a_1 1
a_2 1
a_3 1
a_4 1
p++
$P_5 q_6$
a_1 1
a_2 1
a_3 1
a_4 1
p++
$P_6 q_6$
a_1 1
a_2 1
a_3 1
a_4 1
print 0
p++
$P_7 q_6$
a_1 1
a_2 1
a_3 1
a_4 1
$P_5 q_3$
a_1 1
a_2 1
a_3 0
a_4 1
Sweet Love

b) Implemente o programa em linguagem C/PThreads ou Java que faça o que o pseudo-código determine. (em anexo como semaforo.c)

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. (código em anexo)

N/ Número de *threads*	1	2	4	8
10^5	2 ms	1 ms	0.86 ms	0.85 ms
10^7	88 ms	82 ms	77 ms	91 ms

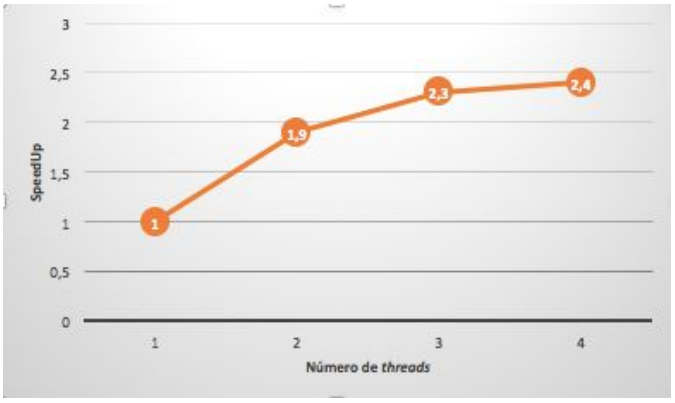


Gráfico 17: *SpeedUp* no programa do cálculo do produto escalar em Java para N=100000

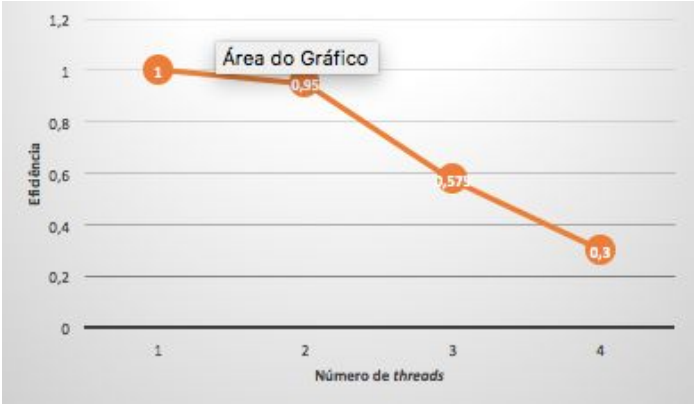


Gráfico 18: Eficiência no programa do cálculo do produto escalar em Java para N=100000

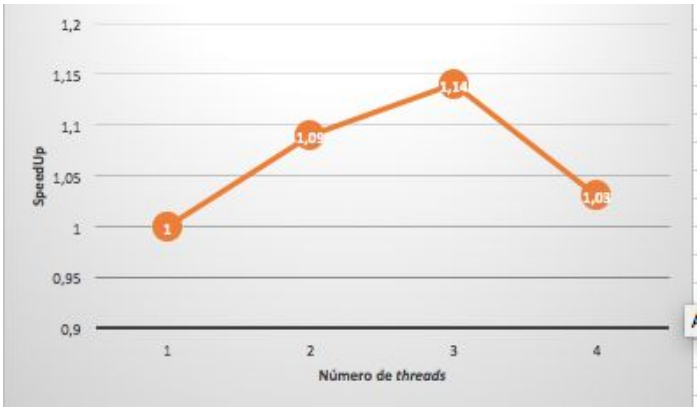


Gráfico 19: *SpeedUp* no programa do cálculo do produto escalar em Java para N=10000000

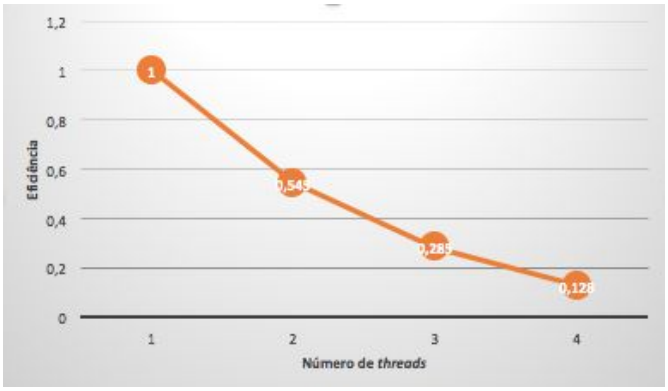


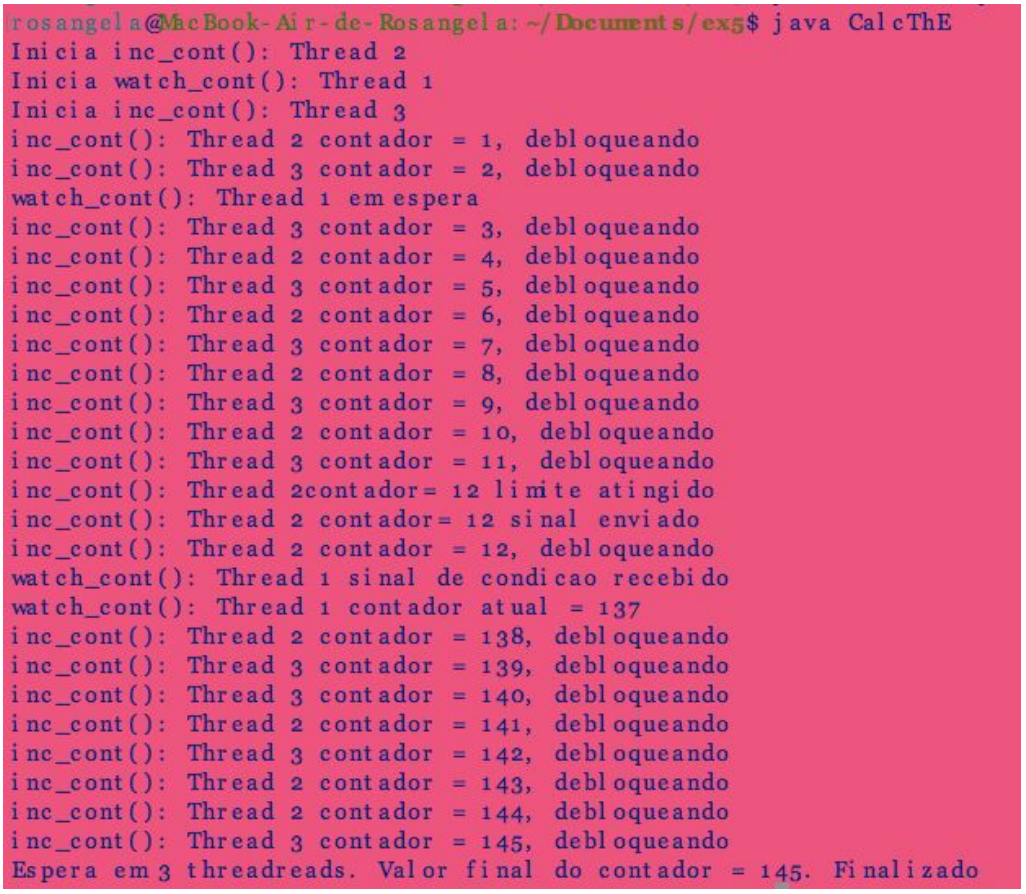
Gráfico 18: Eficiência no programa do cálculo do produto escalar em Java para N=10000000

5. A partir do seguinte programa exemplo disponível no Moodle: "Exemplo de variável de condição em C com PThreads", o qual demonstra o uso de um código com semáforos e variáveis de condição em C/PThreads, faça um programa similar usando linguagem Java utilizando os seguintes recursos de controle de fluxo de execução de threads:

a. Trava (lock()) e variável de condição;
b. Código Monitor (método com cláusula *sychronized*) e variável de condição.
c. Mostre saídas provando que o código funciona.

(código em anexo na pasta ex5)

Ao executar o programa se obteve a seguinte saída.



Obs: no caso do item (b) deve-se ter o mesmo código monitor rodando procedimentos distintos de acordo com a identificação da *Thread*.