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

Laboratório 06

Decomposição recursiva de tarefas e Map-Reduce

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

SÃO JOSÉ DOS CAMPOS
2018

Especificações da Máquina

Cada nó (máquina) do cluster é composto por dois processadores Intel Xeon E5-2660v4 de 2.0-GHz e 128 GB de memória principal.

Cara processador citado tem 14 núcleos, portanto, cada nó tem 28 núcleos/processadores.

Exercício 1

Desenvolva um programa para fazer ordenação de valores numéricos dispostos em um vetor de tamanho: $N = 10^7$, com dados do tipo Double. Utilize, necessariamente o algoritmo de MergeSort. Faça a decomposição de tarefas de forma recursiva, sendo que deve-se definir um limite para a quantidade de Threads abertas ao mesmo tempo, ou seja, ao se atingir o limite pode-se continuar a gerar recursões mas não novas threads. A quantidade de threads abertas está especificada a seguir.

Coloque em um gráfico os tempos de processamento, Speedup e Eficiência obtidos variando-se o número máximo de threads abertas simultaneamente em: 1, 2, 4, 8, 16, 32 e 64 (a partir de 2 threads, considerar 2 threads mais o programa principal). A medição de tempo deve concentrar-se apenas no processamento da ordenação e desconsiderar o tempo gasto para alocação e preenchimento do vetor. Pode-se utilizar de JavaThreads, C+PThreads ou C+OpenMP (utilizando-se da funcionalidade Tasks em OpenMP, ver conjunto específico de slides).

Resultados do Exercício 1

Para o exercício 1, foi implementado um código em linguagem C, utilizando OpenMP, com a funcionalidade Tasks.

Abaixo, temos um exemplo de ordenação realizada:

```
343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.3438
00 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 2
6.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.34
3800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800
 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.
343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.3438
00 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 2
6.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.34
3800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800
 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.
343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.3438
00 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 2
6.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.34
3800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800
 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.
343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.3438
00 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 2
6.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.34
3800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800
 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.
343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.3438
00 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 2
6.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.34
3800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800 26.343800

Ordenacao concluida. Tempo de Execução: 13.434021
```

Figura 1: Exemplo de ordenação feita com linguagem C+OpenMP, no terminal do Ubuntu.

Os resultados obtidos em simulações locais foram:

Figura 2: Resultados de tempo de execução obtidos no Exercício 1.

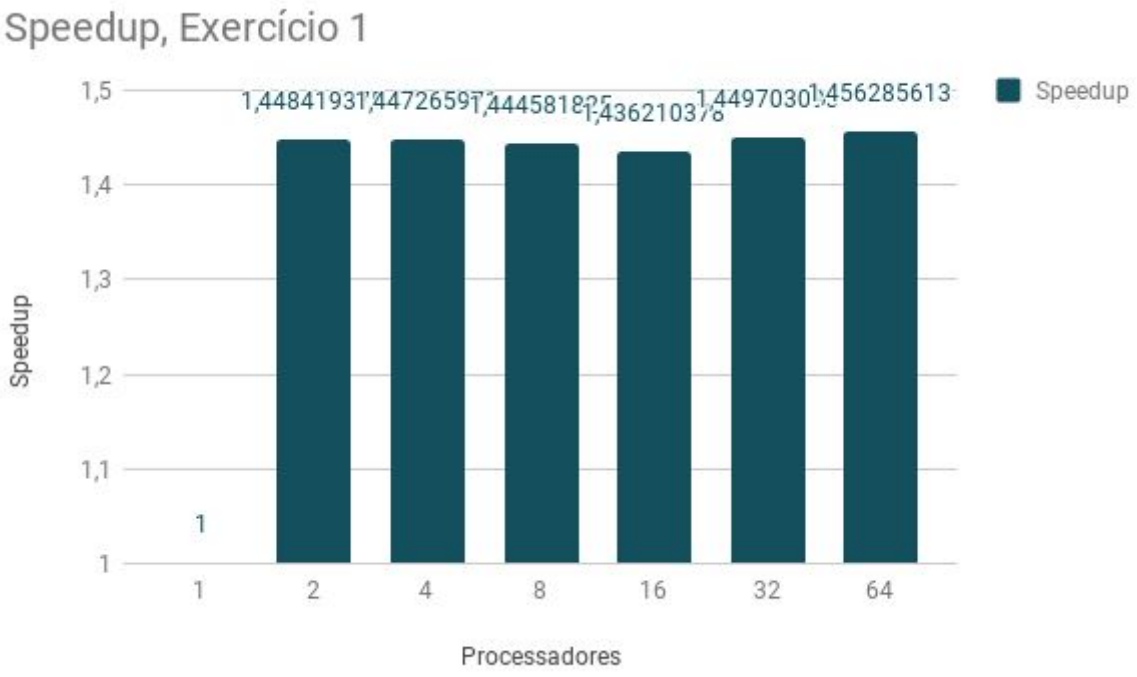

Figura 3: Resultados de Speedup obtidos no Exercício 1.

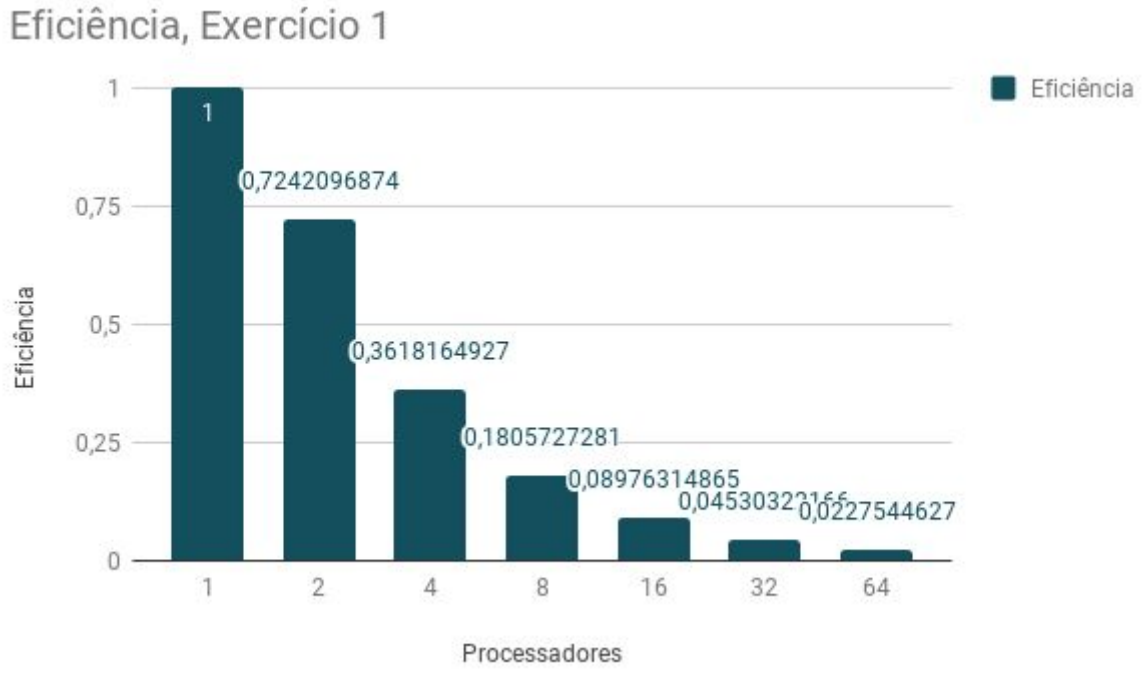

Figura 4: Resultados de Eficiência obtidos no Exercício 1.

Exercício 2

A partir do código-fonte exemplo "mapreduce_serial.c", implemente uma versão paralela em OpenMP que conte a quantidade de ocorrências de cada código de gastos (numSubCota) do arquivo csv fornecido. O código-fonte exemplo conta ocorrências do arquivo "data.txt" (também fornecido), que contém apenas números, desta forma, deve-se adaptar ou criar um novo código-fonte para realizar o que se pede. Faça uma tabela ou gráfico mostrando tempos de execução, Speedup e Eficiência, variando-se o número de threads para: 1, 2, 4, 7, 14 e 28 (capacidade máxima de processadores em um nó do cluster).

Observação: Os códigos gerados foram levemente diferentes (apareceu um código a mais) devido ao não tratamento de divisão de números quebrados em uma linha, durante a separação da leitura do arquivo-texto.

Resultados do Exercício 2

Veja abaixo uma demonstração do funcionamento da versão do código em OpenMP:

```
willian@willian-dihanster:~$ gcc -o lab mapreduce_openmp.c -fopenmp
willian@willian-dihanster:~$ ./lab
codigo 1: 23642 ocorrencias
codigo 3: 70566 ocorrencias
codigo 4: 2659 ocorrencias
codigo 5: 13358 ocorrencias
codigo 8: 952 ocorrencias
codigo 9: 2318 ocorrencias
codigo 10: 28234 ocorrencias
codigo 11: 25928 ocorrencias
codigo 12: 1005 ocorrencias
codigo 13: 20223 ocorrencias
codigo 14: 6925 ocorrencias
codigo 119: 304 ocorrencias
codigo 120: 6521 ocorrencias
codigo 121: 29 ocorrencias
codigo 122: 28869 ocorrencias
codigo 123: 1469 ocorrencias
codigo 137: 30 ocorrencias
codigo 999: 108392 ocorrencias

Tempo de Execução: 0.036250
```

Figura 5: Exemplo de funcionamento do código em OpenMP, no terminal do Ubuntu.

Neste exercício, foram obtidos os seguintes resultados:

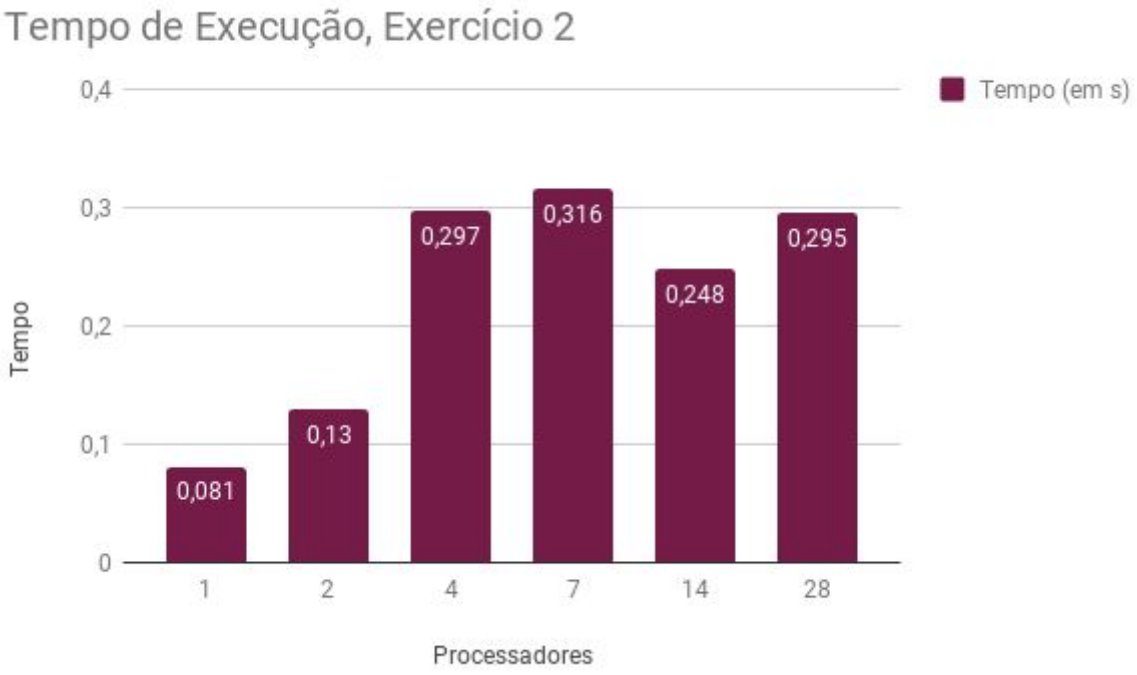

Figura 6: Resultados de tempo de execução obtidos no Exercício 2.

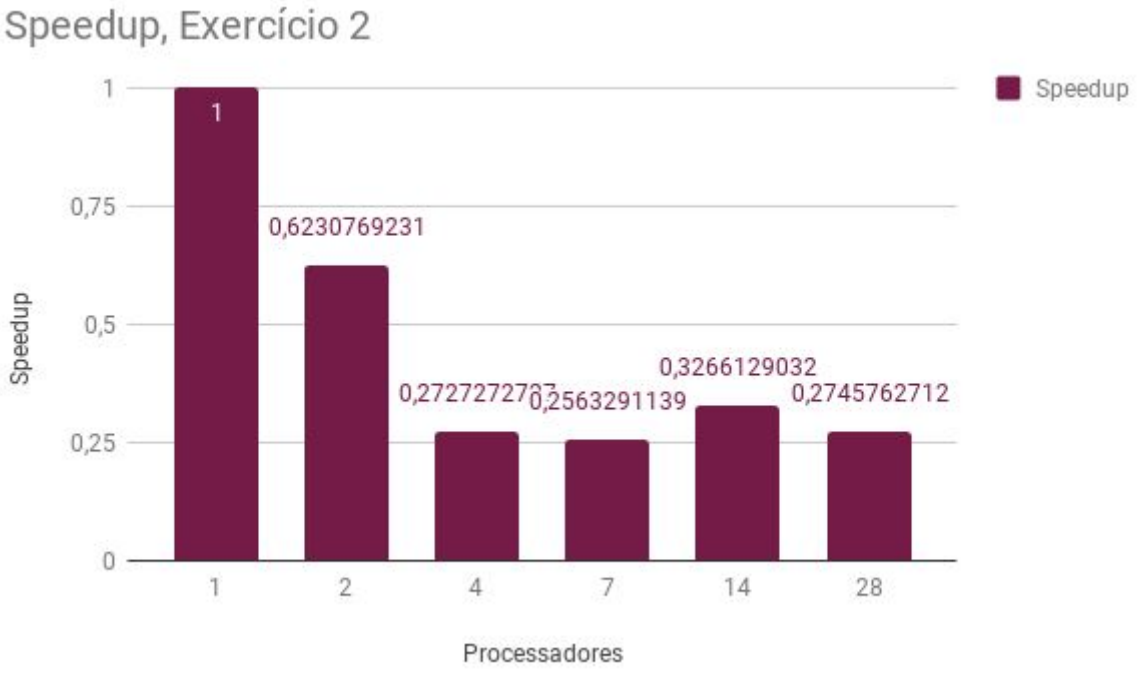

Figura 7: Resultados de Speedup obtidos no Exercício 2.

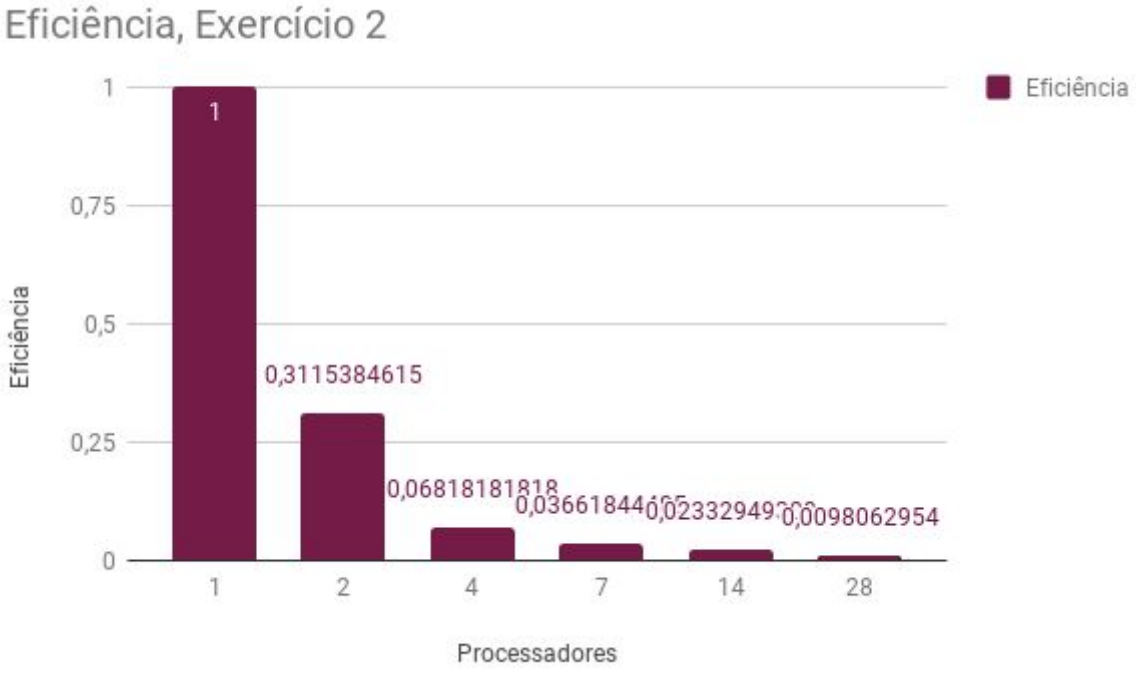

Figura 8: Resultados de Eficiência obtidos no Exercício 2.

Exercício 3

Crie uma versão MPI do exercício 2 acima. Faça as mesmas medidas de tempo do caso anterior e amplie para medir o desempenho com até dois nós, correspondente à 56 processos/processadores.

Resultados do Exercício 3

A seguir, temos uma demonstração do funcionamento com MPI:

```
anajbellini@SteveRogers:~/Workspaces/OpenMP/Atividade6$ mpirun -np 1 map3
Inicializando Map-Reduce com 1 processos...
codigo 0: 714971 ocorrencias
codigo 1: 23642 ocorrencias
codigo 3: 70566 ocorrencias
codigo 4: 2659 ocorrencias
codigo 5: 13358 ocorrencias
codigo 8: 952 ocorrencias
codigo 9: 2318 ocorrencias
codigo 10: 28234 ocorrencias
codigo 11: 25928 ocorrencias
codigo 12: 1005 ocorrencias
codigo 13: 20223 ocorrencias
codigo 14: 6925 ocorrencias
codigo 119: 304 ocorrencias
codigo 120: 6522 ocorrencias
codigo 121: 29 ocorrencias
codigo 122: 28869 ocorrencias
codigo 123: 1469 ocorrencias
codigo 137: 30 ocorrencias
codigo 999: 108392 ocorrencias
Map-Reduce concluido. Tempo gasto 0.027450 segundos
anajbellini@SteveRogers:~/Workspaces/OpenMP/Atividade6$
```

Figura 9: Exemplo de funcionamento do código em MPI, no terminal do Ubuntu.

E por fim, os resultados:

Figura 10: Resultados de tempo de execução obtidos no Exercício 3.

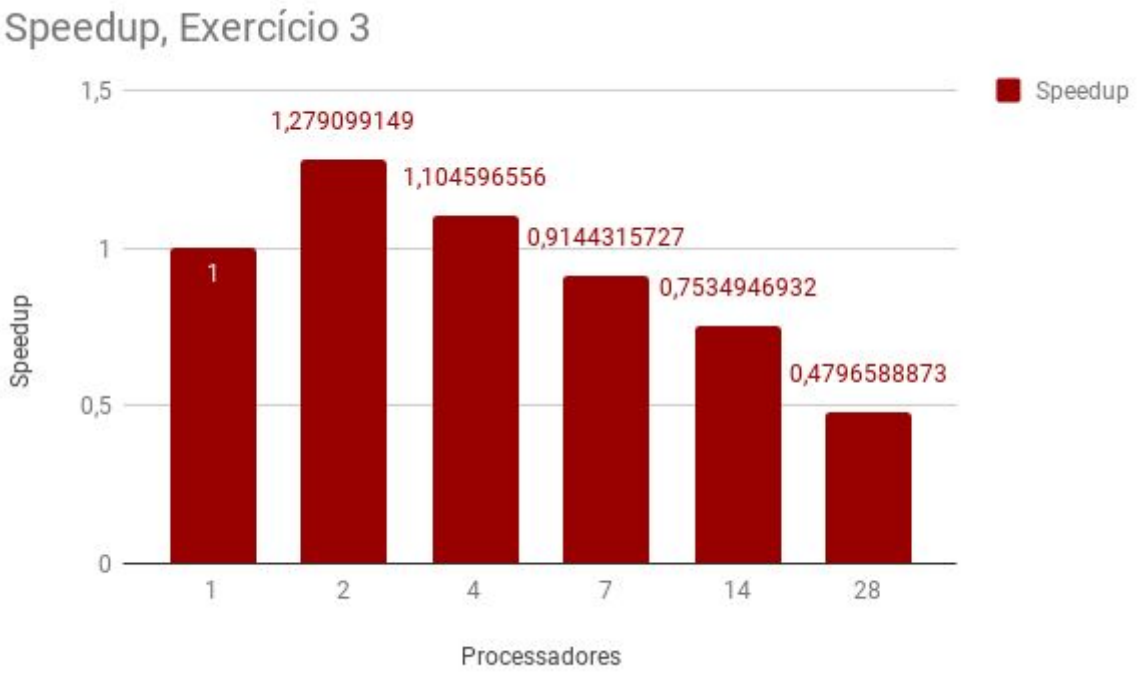

Figura 11: Resultados de Speedup obtidos no Exercício 3.

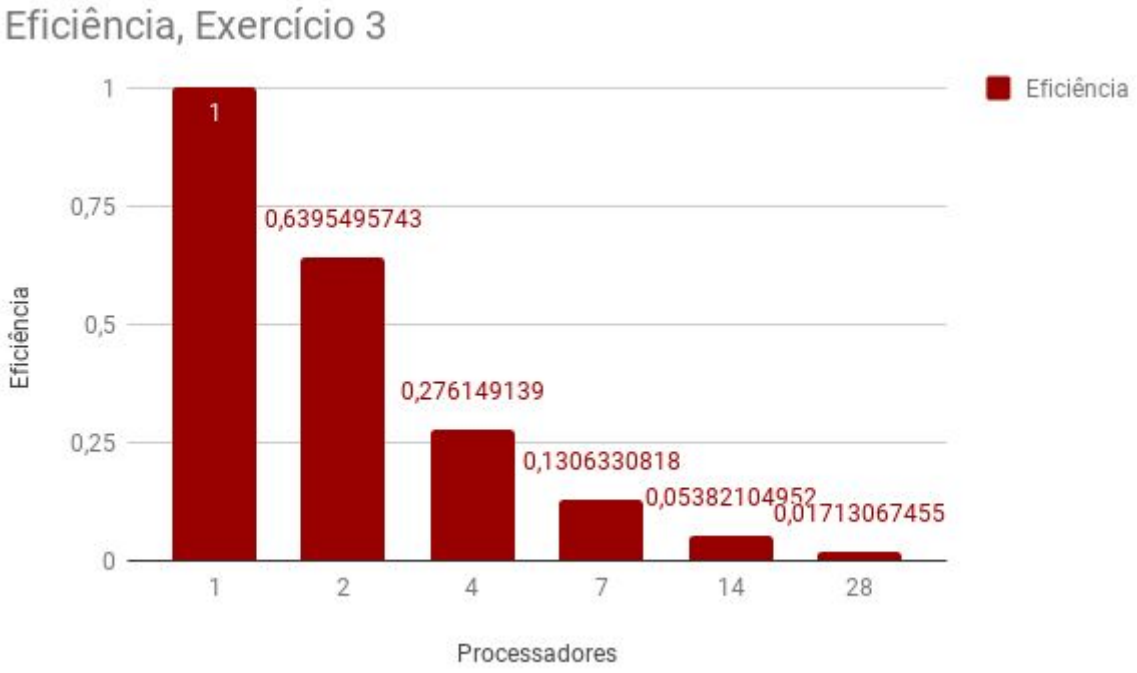

Figura 12: Resultados de Eficiência obtidos no Exercício 3.