UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE

INSTITUTO METRÓPOLE DIGITAL

Disciplina: Tópicos Especiais em Internet das Coisas “B”

Aluno: Israel Medeiros Fontes

Professor: Kayo Gonçalves e Silva

Relatório

Odd-Even Sort Paralelo Utilizando MPI

Natal/RN

2020

SUMÁRIO

1. INTRODUÇÃO 3

2. METODOLOGIA 4

3. DESENVOLVIMENTO 5

3.1. IMPLEMENTAÇÃO 5

3.2. CORRETUDE 7

3.3. RESULTADOS 8

3.4. SPEEDUP, EFICIÊNCIA E ESCALABILIDADE 11

4. CONCLUSÃO 13

1. INTRODUÇÃO

Os algoritmos de ordenação são, sem sombra de dúvidas, um dos assuntos mais importantes para a computação. Existem várias técnicas e implementações desses algoritmos para os mais diversos tipos de aplicação, cada um com suas particularidades e fins. Um deles é o *Odd-Even Sort*, ou numa tradução simples para o português: Ordenação Ímpar-Par, ele é um algoritmo que se baseia na comparação de pares ímpar e par de um vetor e ordenação desses pares sucessivas vezes até que o vetor completo seja totalmente ordenado.

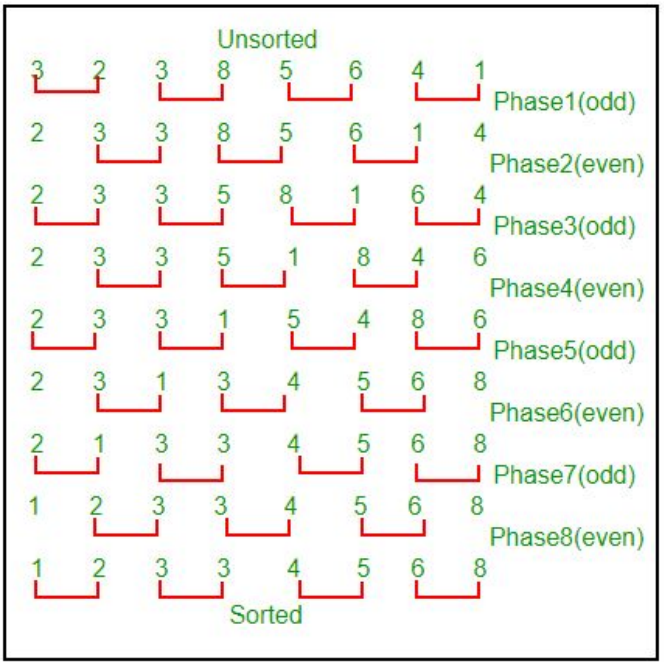

Figura 1: Exemplo de ordenação de um vetor utilizando *Odd-Even Sort.*

Referência: https://www.geeksforgeeks.org/odd-even-sort-brick-sort

Na Figura 1 temos um exemplo de como o algoritmo funciona. Dado um vetor de 8 elementos inteiros, o algoritmo realiza 8 fases de comparação e ordenação. Nas fases pares o algoritmo compara o valor entre os pares nas posições par-ímpar respectivamente, já nas fases ímpares ocorre a comparação entre os pares nas posições ímpar-par respectivamente. Dessa maneira, ao final da 8 fase, todo o vetor estará totalmente ordenado.

Este trabalho tem como objetivo a implementação do algoritmo de ordenação *Odd-Even Sort* utilizando os paradigmas de programação paralela e serial e realização de testes de comparação de eficiência entre os dois paradigmas.

2. METODOLOGIA

Será implementado o algoritmo *Odd-Even Sort* nos paradigmas serial e paralelo. Para tal implementação, utilizarei a linguagem de programação C que possui suporte a biblioteca *Message Passing Interface* (MPI). Ela possibilita a comunicação entre vários processadores, com ela é possível realizar a troca de dados entre processadores que, neste trabalho, serão vetores de números inteiros. Após a implementação dos algoritmos, será realizada uma bateria de testes coordenados por scripts escritos em *Shell Script* para analisar o tempo de execução dos dois algoritmos.

Os testes se darão da seguinte maneira: primeiro serão realizadas 4 execuções do código serial de modo que cada execução corresponderá a um tamanho de problema diferente, ou seja, a um tamanho de vetor diferente que será criado e preenchido de maneira pseudoaleatória, ordenado utilizando o algoritmo *Odd-Even Sort* serial e retornando e armazenando por último o tempo de execução levado pelo algoritmo. Cada vetor criado pelo algoritmo serial será persistido em um arquivo binário para que ele seja posteriormente utilizado pelo algoritmo paralelo de forma que os dois algoritmos possam ordenar o mesmo problema, possibilitando uma comparação mais justa; depois dos testes do algoritmo serial serão realizados os mesmos testes com o algoritmo paralelo utilizando o vetor criado anteriormente pelo serial e também será extraído o tempo de execução. Cada execução serial e paralela será feita 5 vezes e o tempo de execução final será a média dessas execuções.

Ao final será feita as análises de corretude, speedup, eficiência e escalabilidade do algoritmo paralelo em comparação com o serial.

3. DESENVOLVIMENTO

3.1 IMPLEMENTAÇÃO

Na implementação do algoritmo serial, será utilizado o pseudocódigo a seguir em sua implementação.

```
odd-even-sort ( list )
    for (phase = 0; phase < list.length; phase++ )
        // odd-even
        if ( phase%2 == 0 )
            for ( x = 1; x < list.length; x += 2)
                if (list[x-1] > list[x])
                    swap list[x] and list[x-1]
        // even-odd
        else
            for ( x = 1; x < list.length-1; x += 2)
                if (list[x] > list[x+1])
                   swap list[x] and list[x+1]
```

Código 1: Pseudocódigo do algoritmo *Odd-Even Sort*.

O Código 1 apresenta uma implementação simples do algoritmo *Odd-Even Sort*, ele realiza o processo de ordenação como observamos na Figura 1. Ele divide a ordenação em fases que vão até o tamanho da lista, cada fase pode ser par ou ímpar. Nas fases pares é feita a comparação entre as duplas de itens começando da primeira posição par e os ordenando entre si. Nas fases ímpares é feita a comparação entre as duplas de itens começando da primeira posição ímpar e os ordenando entre si. Dessa maneira, ao final da última fase, toda a lista estará ordenada.

```
// Distribui o vetor inicial em tamanhos iguais para todos processos
MPI_Scatter ( list )
// Ordena os vetores locais de cada processo
odd-even-sort ( my_list )
for ( phase = 0; phase < ranks; phase++ )
    // odd-even
    if ( phase%2 != 0 )
        // Processos ímpares enviam seus vetores para os pares
        if ( rank%2 != 0 )
            MPI_Send (my_list, rank-1)
        // Processos pares
        if ( rank%2 == 0 )
            // Recebem os vetores dos ímpares
            MPI_Recv (temp_list, rank+1)
            // Concatenam com seus próprios vetores
            join temp_list and my_list
            // Ordenam
            odd-even-sort (temp_list)
            // Enviam para os ímpares apenas o final do vetor ordenado
            MPI_Send (temp_list[final], rank+1)
        else
            // Processos ímpares apenas recebem o vetor já ordenado dos pares
            MPI_Recv(my_list, rank-1)
    // even-odd
    else
        // Processos pares maiores que 0
        if ( rank%2 == 0 and rank > 0 )
            // Enviam para os ímpares suas listas
            MPI_Send(my_list, rank-1)
            // Recebem a lista já ordenada
            MPI_Recv(my_list, rank-1)
        // Processos ímpares
        else if (rank%2 != 0 and rank < ranks-1)
            // Recebem os vetores dos pares
            MPI_Recv (temp_list, rank+1)
            // Concatenam com seus próprios vetores
            join temp_list and my_list
            // Ordenam
            odd-even-sort (temp_list)
            // Enviam a segunda parte do vetor para os processos pares
            MPI_Send (temp_list[final], rank+1)
```

Código 2: Pseudocódigo do algoritmo *Odd-Even Sort* implementado em paralelo.

O Código 2 apresenta uma abordagem paralela do algoritmo *Odd-Even Sort* que foi implementada neste trabalho. Nesta abordagem foi utilizado a implementação do *Odd-Even Sort* serial para realizar as ordenações internas em cada processo.

3.2 CORRETUDE

Dada a implementação dos dois algoritmos, foi analisado a corretude dos mesmos com tamanhos de problema bem reduzidos.

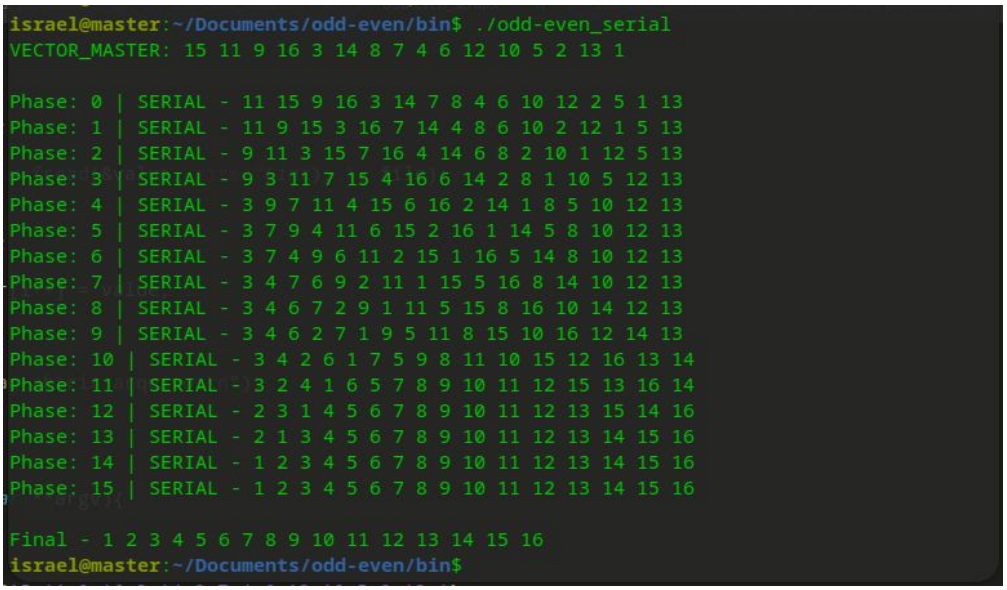

Figura 2: Corretude da implementação em serial.

Conforme a Figura 2, dado um vetor de 16 posições de números inteiros desordenados, o algoritmo serial implementado segundo o Código 1 conseguiu chegar ao final da última fase entregando o vetor já ordenado como era esperado.

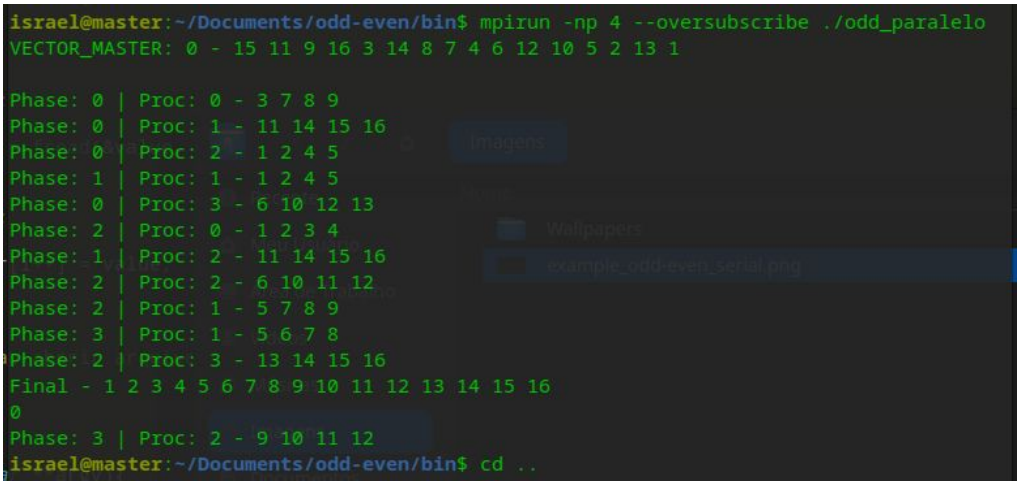

Figura 3: Corretude da implementação em paralelo.

Já na Figura 3 podemos perceber que a implementação do algoritmo paralelo segundo o Código 2 também conseguiu realizar a ordenação corretamente do mesmo vetor que foi inserido na análise de corretude da implementação serial.

3.3 RESULTADOS

Agora que já é sabido que os algoritmos entregam o resultado esperado, serão realizados os testes de comparação de tempo de execução descritos na metodologia. Para cada paradigma foi realizado uma bateria de testes.

Todos os testes descritos a partir de agora foram feitos em um notebook com as seguintes configurações: processador Intel Core i7-3632QM com 4 núcleos físicos e 8 threads com hyperthreading, clock base de 2.2ghz e máximo de 3.2ghz; memória RAM DDR3 com 8gb; sistema operacional Linux, distribuição Deepin com 64bits.

Tamanho do problema	Tempo de execução (s)
120000	36.07
150000	56.56
180000	81.44
210000	111.02

Tabela 1: Tempo de execução da implementação serial.

Dado tamanhos de problemas diferentes, ou seja, tamanhos de vetores diferentes, foi contabilizado o tempo de execução conforme a Tabela 1.

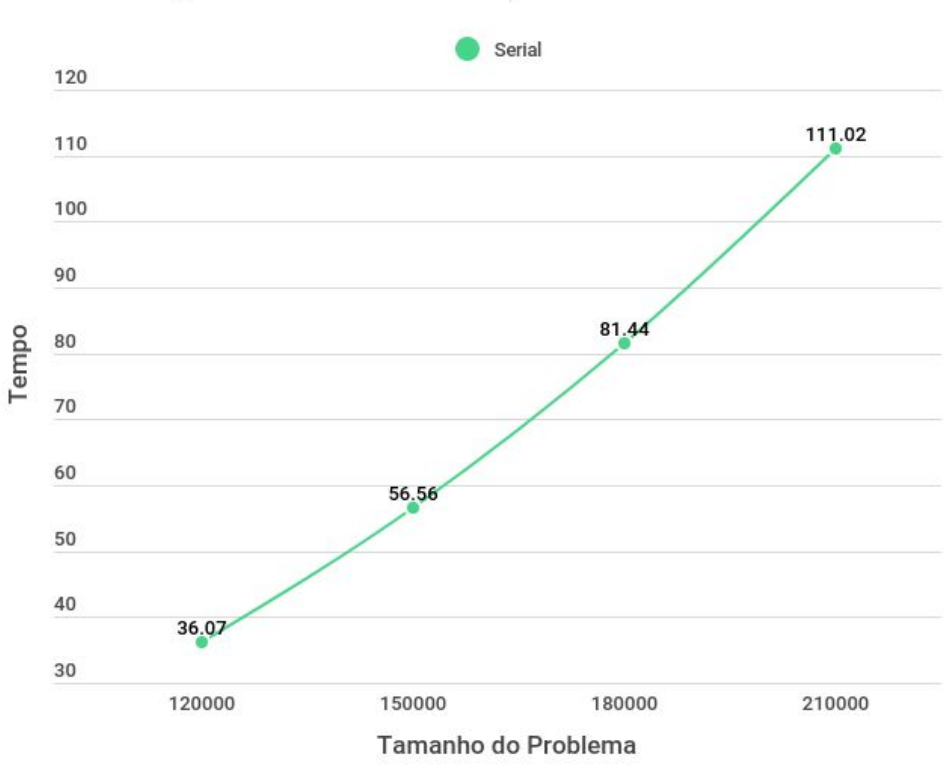

Gráfico 1: Tempo de execução da implementação serial.

Com o Gráfico 1, conseguimos perceber que o algoritmo serial tem um tempo de execução linear de acordo com a mudança do tamanho do problema.

CPUs	Tamanho do problema	Tempo de execução (s)
2	120000	32.91
2	150000	51.51
2	180000	74.05
2	210000	100.95
4	120000	27.64
4	150000	42.86
4	180000	61.98
4	210000	84.04
8	120000	22.93
8	150000	36.39
8	180000	52.90
8	210000	72.42

Tabela 2: Tempo de execução da implementação paralela.

Da mesma maneira foi analisado o tempo de execução na implementação paralela, com os mesmos vetores da implementação serial, mas agora é verificada com 2, 4 e 8 núcleos de processamento simultâneos conforme a Tabela 2.

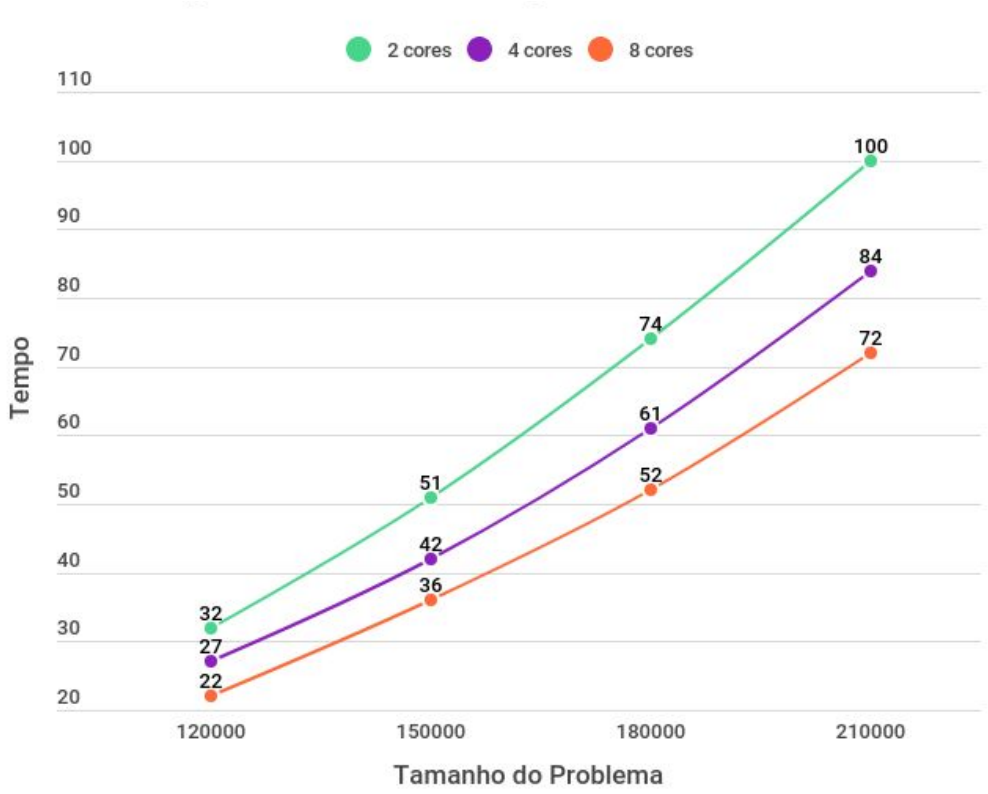

Gráfico 2: Tempo de execução da implementação paralela.

Como podemos perceber no Gráfico 2, o tempo de execução para a implementação em paralelo também segue linear para as três quantidades de núcleos de processamento diferentes.

3.4 SPEEDUP, EFICIÊNCIA E ESCALABILIDADE

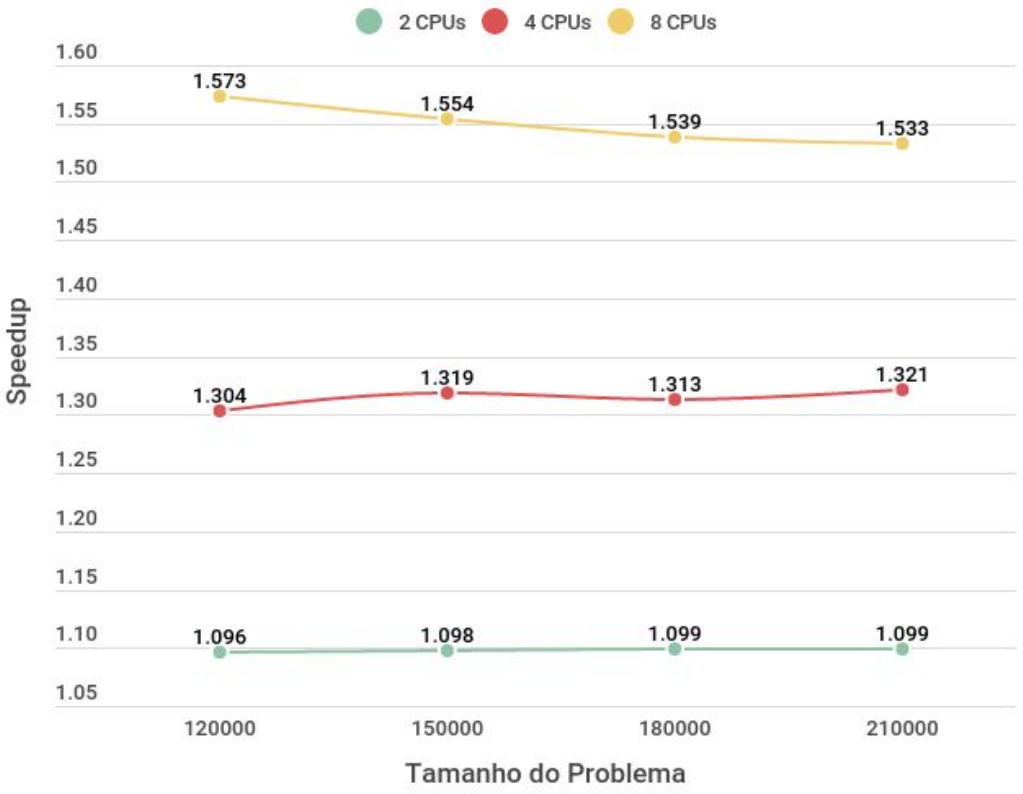

Gráfico 3: Speedup da implementação paralela.

O Speedup é um índice que traduz quantas vezes um algoritmo paralelo é mais rápido do que sua implementação serial. Ele é calculado da seguinte forma:

$$S = \frac{T_s}{T_p}$$

onde S é o Speedup, T_s é o tempo da execução serial e T_p é o tempo da execução em paralelo. O Gráfico 3 apresenta os Speedups para cada execução realizada referente ao tamanho do problema e quantidade de núcleos.

É perceptível que o Speedup para o problema *Odd-Even Sort* implementado segundo o Código 2 é muito baixo, ou seja, não é tão mais rápido do que a implementação serial.

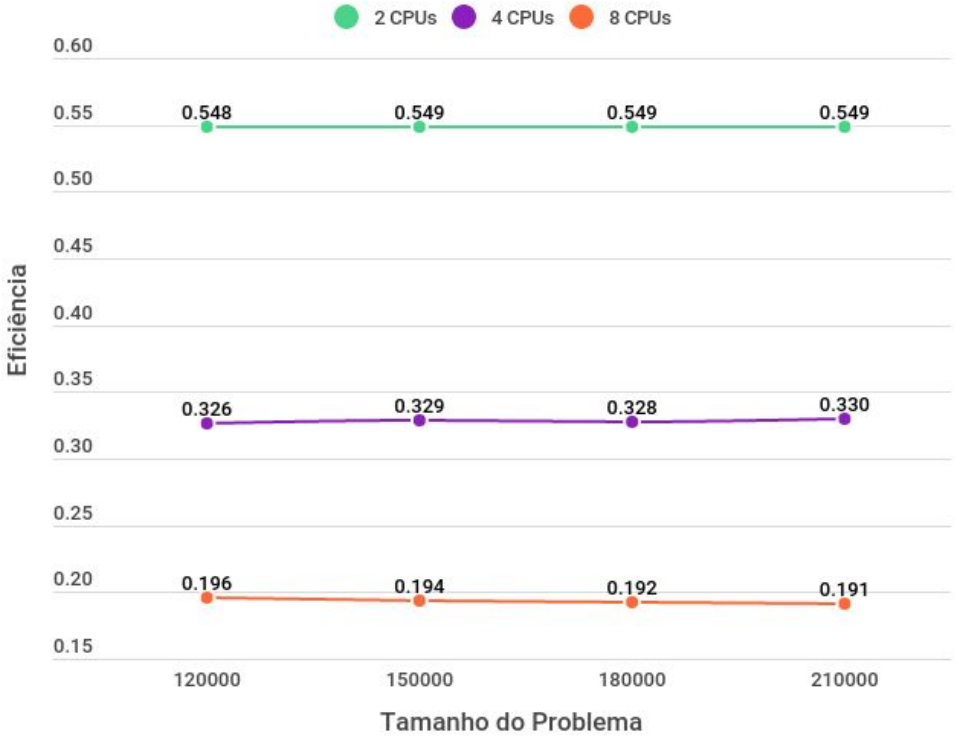

Gráfico 4: Eficiência da implementação em paralelo.

O Gráfico 4 traz a eficiência da implementação paralela que é calculada da seguinte forma:

$$E_f = \frac{S}{m}$$

onde E_f é a eficiência S é o Speedup e m é a quantidade de núcleos de processamento.

Percebemos a baixa eficiência deste algoritmo em paralelo. Também é perceptível a não escalabilidade do mesmo pois a eficiência diminui dado o aumento do número de núcleos.

4. CONCLUSÃO

Foi implementado os dois paradigmas do algoritmo *Odd-Even Sort*. Ficou perceptível pelas análises feitas que tal método de ordenação não consegue explorar muito bem o potencial da computação paralela. Talvez, grande parte desse problema esteja atrelado a utilização do algoritmo *Odd-Even Sort* serial nas sub-ordenações em cada processo. Fica para um próximo trabalho analisar se ao utilizar algoritmos mais rápidos nessas sub-ordenações seja possível aproveitar melhor o poder do paralelismo.