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

Histograma Utilizando Pthreads

Natal/RN

2020

SUMÁRIO

1. INTRODUÇÃO

Histogramas são representações gráficas muito utilizadas para se analisar a frequência em que um determinado dado ocorre em meio a um conjunto de dados. São, geralmente, gráficos de barras de simples construção onde o eixo x representa a classe do dado e o eixo y representa a frequência dessa classe.

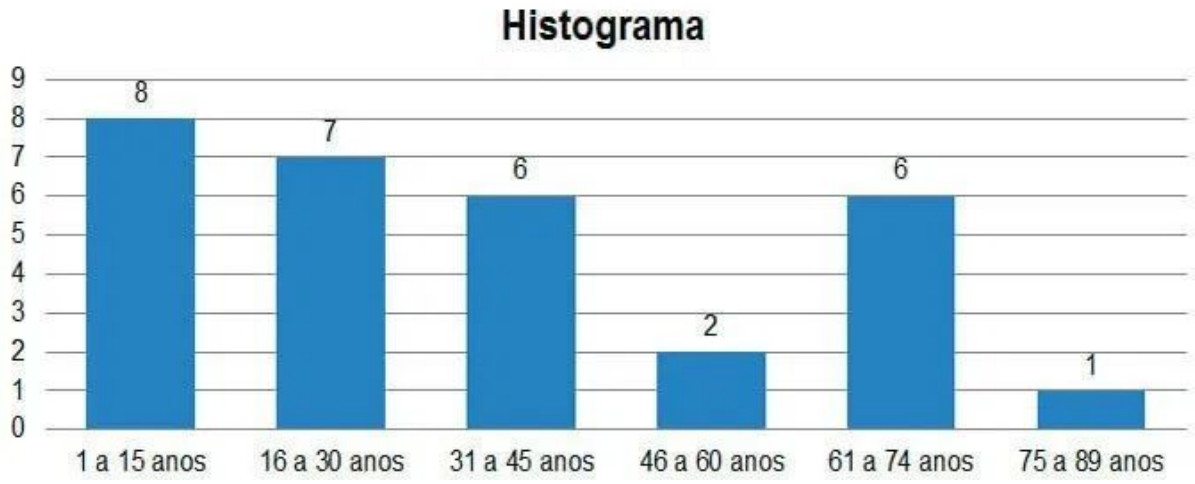


Figura 1: Exemplo de histograma.

Na Figura 1 temos um exemplo de histograma simples que traz a frequência de idades de um determinado grupo de pessoas em que o eixo x representa uma classe de idades que vai de 1 a 89 anos e varia de 15 em 15 anos. O eixo y representa a quantidade de pessoas naquela faixa de idade.

Este trabalho tem como objetivo a implementação de um algoritmo de geração de números aleatórios seguindo uma distribuição normal e criação de um histograma de incidência por intervalos dos números gerados. Tal problema será implementado utilizando os paradigmas de programação serial e paralela com memória compartilhada. Ao final será realizado testes de comparação de eficiência entre os dois paradigmas.

2. METODOLOGIA

Será implementado dois algoritmos, um no paradigma serial e outro no paradigma paralelo com memória compartilhada. Para tal implementação, utilizarei a linguagem de programação C que possui suporte a biblioteca *POSIX Threads* (PThreads). 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 serão realizados no Super Computador da UFRN mantido pelo *Núcleo de Processamento de Alto Desempenho* (NPAD) e 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 uma quantidade de pontos a serem gerados pseudo-aleatoriamente diferentes seguindo uma distribuição normal e será contabilizado o tempo das 4 execuções. Da mesma forma será feito com o algoritmo paralelo, só que agora, para cada tamanho de problema, o código será executado com 4, 8, 16 e 32 threads e em cada execução será contabilizado o tempo. Para que haja uma precisão maior, essa bateria de testes será realizada 10 e será feita uma média aritmética dos tempos contabilizados.

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.

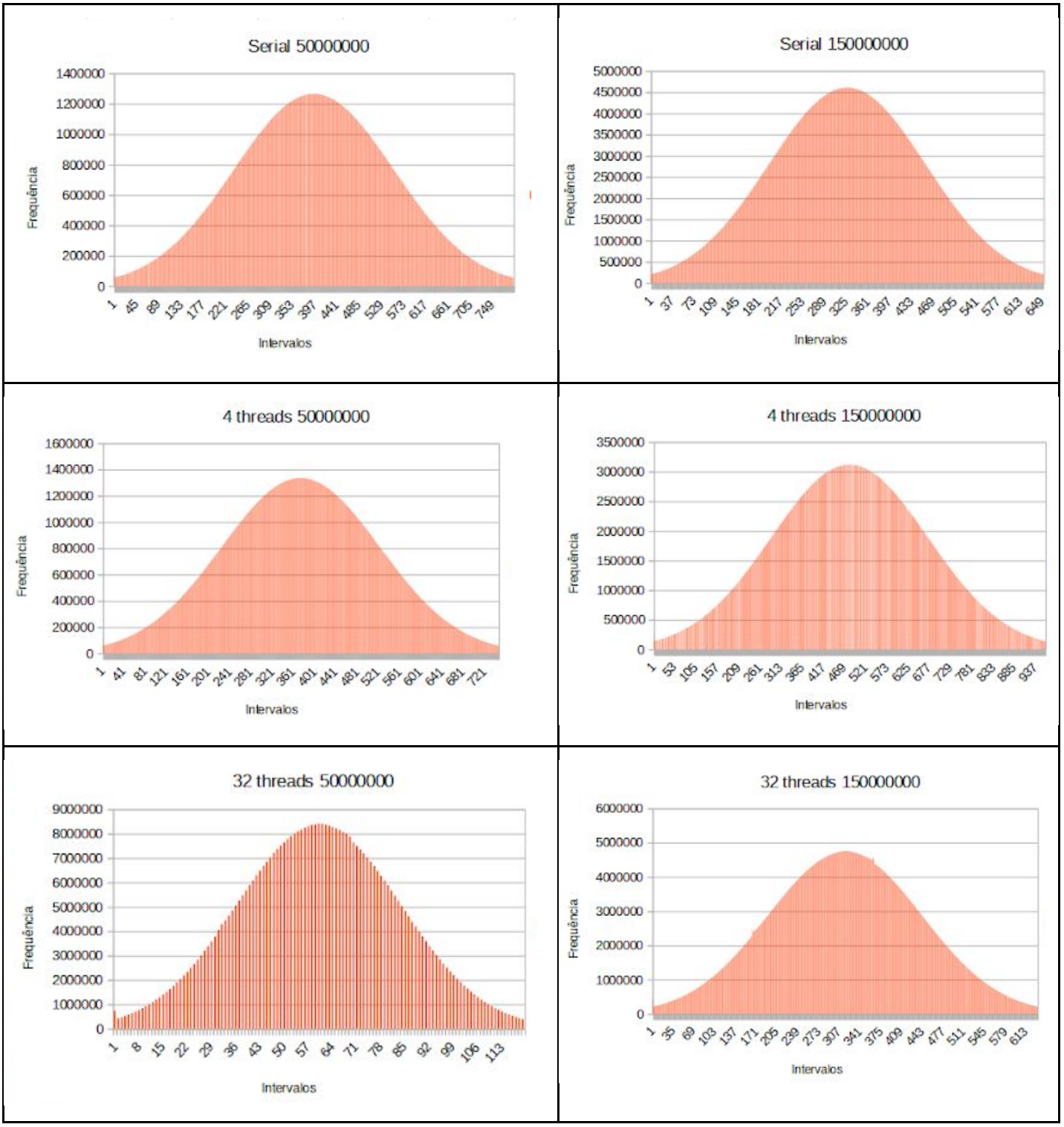
```
σ = 0.2
μ = 0.5

normalRandom( )
    return (cos(2* π *rand_r( ) )*sqrt(-2.*log(rand_r()))* σ + μ )

main(  )
    //Gerar numero de intervalos ate 1000
    intervals = rand_r( ) % 1000
    //Criar vetor histrograma
    histogram[intervals]
    //Resultado
    for ( index = 0; index < size_problem; index++ )
        generated_value = normalRandom( )
        histogram[(generated_value*size_problem/(size_problem/intervals)]
```

Código 1: Pseudocódigo do algoritmo serial.

O Código 1 apresenta uma implementação simples do algoritmo serial, ele gera números aleatórios seguindo a distribuição normal e armazena no vetor resultado que representa o histograma, de forma que cada posição do vetor é relativo a um intervalo de números seguindo os limites da distribuição.

3.2 CORRETUDE

Dada a implementação dos dois algoritmos, foi analisado a corretude dos mesmos com os seus tamanhos de problemas.



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 no super computador do NPAD.

Tamanho do problema	Tempo de execução (s)
50000000	81.67
70000000	114.28
90000000	146.96
150000000	245.29

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.

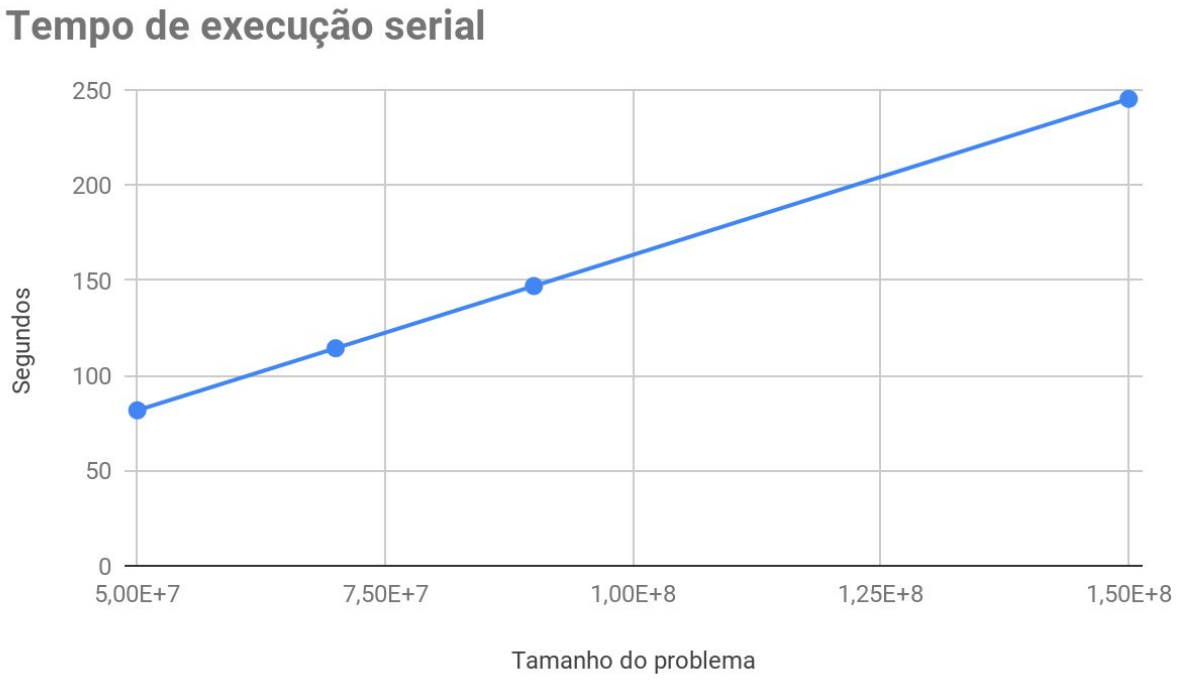


Gráfico 1: Tempo de execução da implementação serial.

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.

Threads	Tamanho do problema	Tempo de execução (s)
4	50000000	39.80
4	70000000	55.28
4	90000000	71.59
4	150000000	120.98
8	50000000	39.36
8	70000000	55.09
8	90000000	69.11
8	150000000	111.06
16	50000000	35.05
16	70000000	50.47
16	90000000	68.57
16	150000000	111.02
32	50000000	26.10
32	70000000	36.10
32	90000000	46.58
32	150000000	77.84

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 4, 8, 16 e 32 núcleos de processamento simultâneos conforme a Tabela 2.

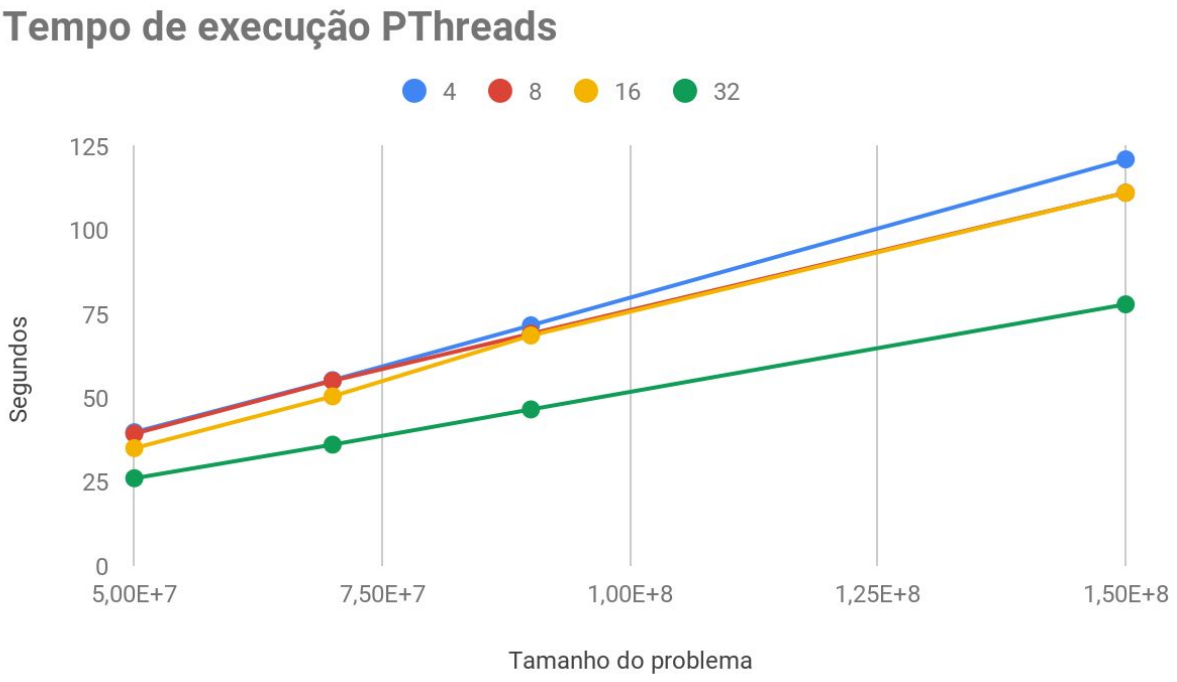


Gráfico 2: Tempo de execução da implementação paralela.

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

3.4 SPEEDUP, EFICIÊNCIA E ESCALABILIDADE

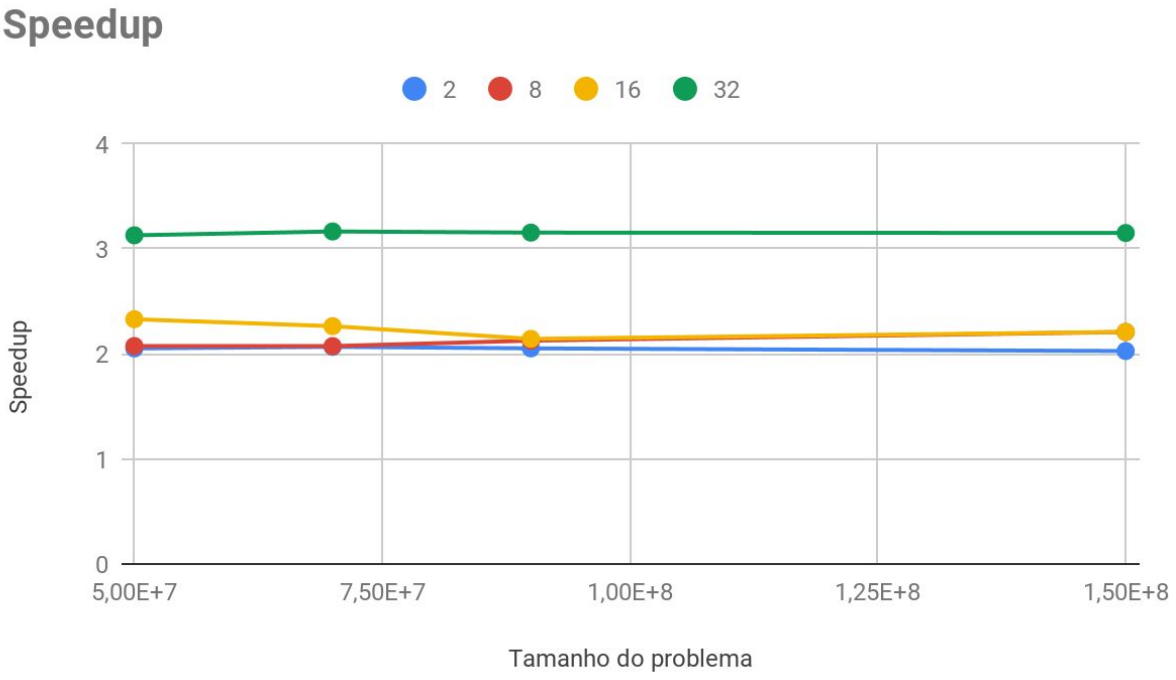


Gráfico 3: Speedup da implementação paralela.

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 é alto, logo o código paralelo consegue ser até 3x mais rápido do que o serial.

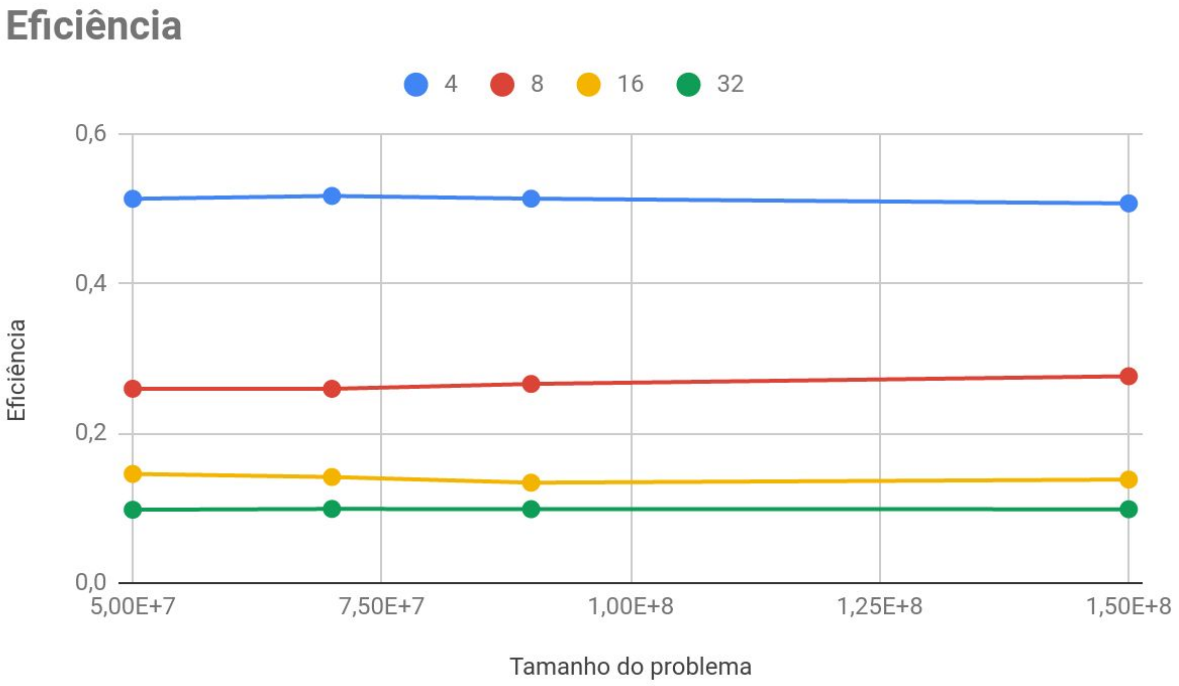


Gráfico 4: Eficiência da implementação em paralelo.

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 boa eficiência deste algoritmo em paralelo. Também é perceptível a escalabilidade do mesmo pois a eficiência se mantém dado o aumento do número de núcleos.

4. CONCLUSÃO

Foi implementado os dois paradigmas do algoritmo para o problema apresentado. Ficou perceptível pelas análises feitas que tal problema pode ser muito bem explorado pela computação paralela.