

Implementação do Odd-Even Sort Paralelizado Utilizando a Biblioteca OpenMP

Tiago Maia de Lacerda Brasil
619031065

Odd-Even Sort Paralelo

```
...
for (int i = 0; i < 2 * size; i++)
{
    if (i % 2 == 0)
    {
        #pragma omp parallel num_threads(size)
        {
            int n = array_length, k = omp_get_num_threads(), id = omp_get_thread_num();
            int s1, s2, f1, f2;
            workload(n, 2 * k, 2 * id + 0, &s1, &f1);
            workload(n, 2 * k, 2 * id + 1, &s2, &f2);
            quicksort(&(*array)[s1]), (f2 - s1) + 1);
        }
    }
}
...
}
```

Odd-Even Sort Paralelo

```
...
for (int i = 0; i < size; i++)
{
    ...
else
{
    #pragma omp parallel num_threads(size)
    {
        int n = array_length, k = omp_get_num_threads(), id = omp_get_thread_num();
        int s1, s2, f1, f2;
        if (id < k - 1)
        {
            workload(n, 2 * k, 2 * (id + 0) + 1, &s1, &f1);
            workload(n, 2 * k, 2 * (id + 1) + 0, &s2, &f2);
            quicksort(&(*array)[s1]), (f2 - s1) + 1);
        }
    }
}
...
...
```

Odd-Even Sort Paralelo

```
void workload(int n, int k, int i, int *s, int *f)
{
    if (n <= 0 || k <= 0 || i < 0)
    {
        *s = -1;
        *f = -1;
    }

    *s = (n / k) * i + min((n % k), i);
    *f = *s + max(0, (n / k) - 1) + (n % k > i);
}
```

Cálculo do Tempo de Execução

O tempo de execução dos algoritmos foi calculado utilizando a função **clock_gettime** da biblioteca **time**, com resolução de nanosegundos. Foi levado em consideração o tempo real (*wall-clock time*) decorrido durante a execução dos algoritmos.

```
double odd-even( . . . )
{
    struct timespec begin, end;
    clock_gettime(CLOCK_REALTIME, &begin);
    . . .
    clock_gettime(CLOCK_REALTIME, &end);
    long seconds = end.tv_sec - begin.tv_sec;
    long nanoseconds = end.tv_nsec - begin.tv_nsec;
    double elapsed = seconds + nanoseconds * 1e-9;
    return elapsed;
}
```

Método de Comparação

Foram gerados vetores de tamanhos variando entre **1000** e **40000** a cada **1000**, com elementos do tipo double variando no intervalo **[0.0, 1.0]**.

O algoritmo implementado com OpenMPI foi executado 10 vezes para cada vetor, com 1, 2, 4 e 8 processos.

O algoritmo implementado com Open MP foi executado 10 vezes para cada vetor, com 1, 2, 4 e 8 *threads*.

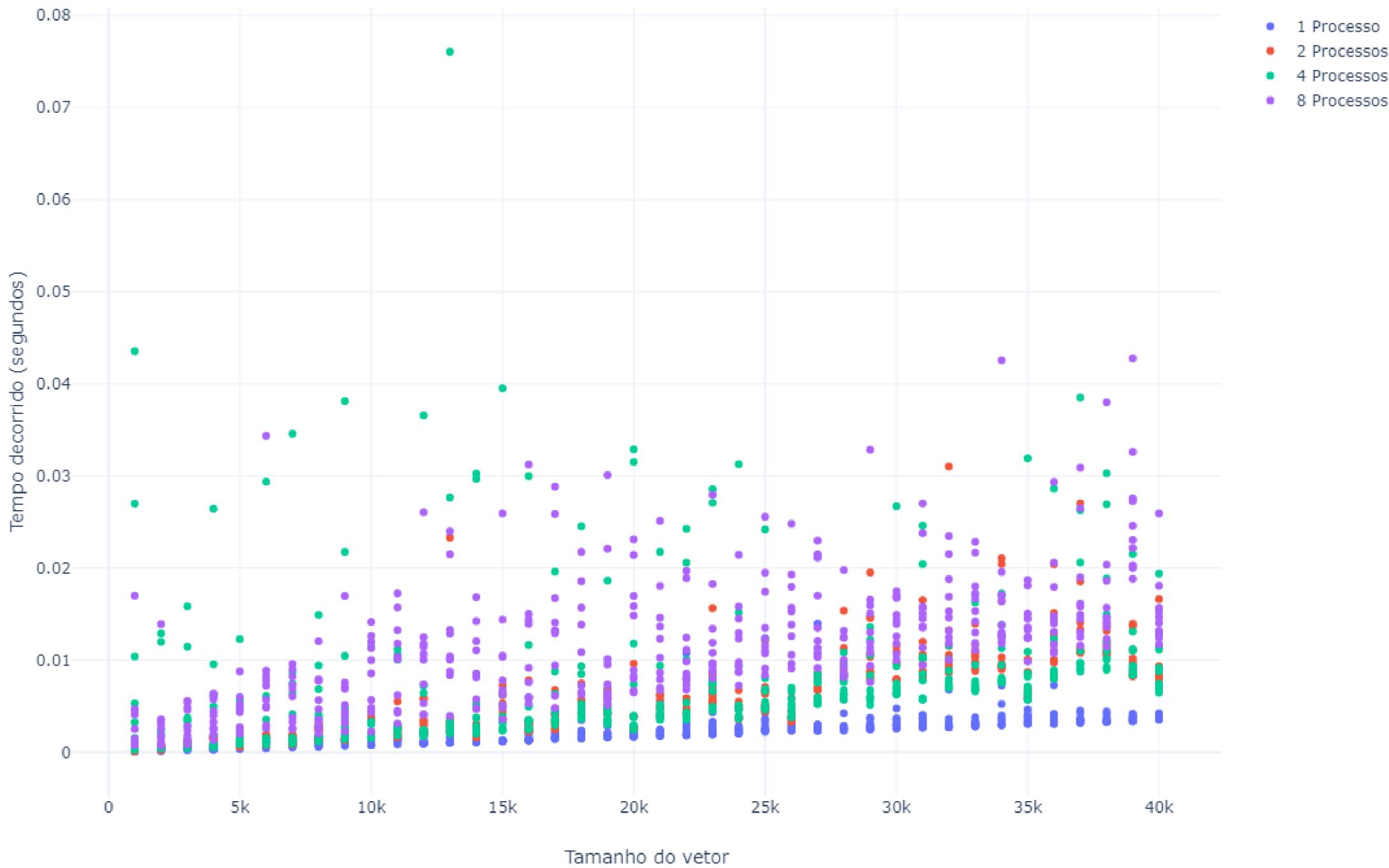
Resultados

O algoritmo implementado com Open MPI e executado com apenas **1** processo se comporta como o quicksort.

Com mais processos, o *overhead* de comunicação apenas piora o desempenho.

É importante notar que todos estes processos se encontram na mesma máquina física, e que este *overhead* seria maior ainda se fosse utilizado um canal de comunicação por redes, por exemplo.

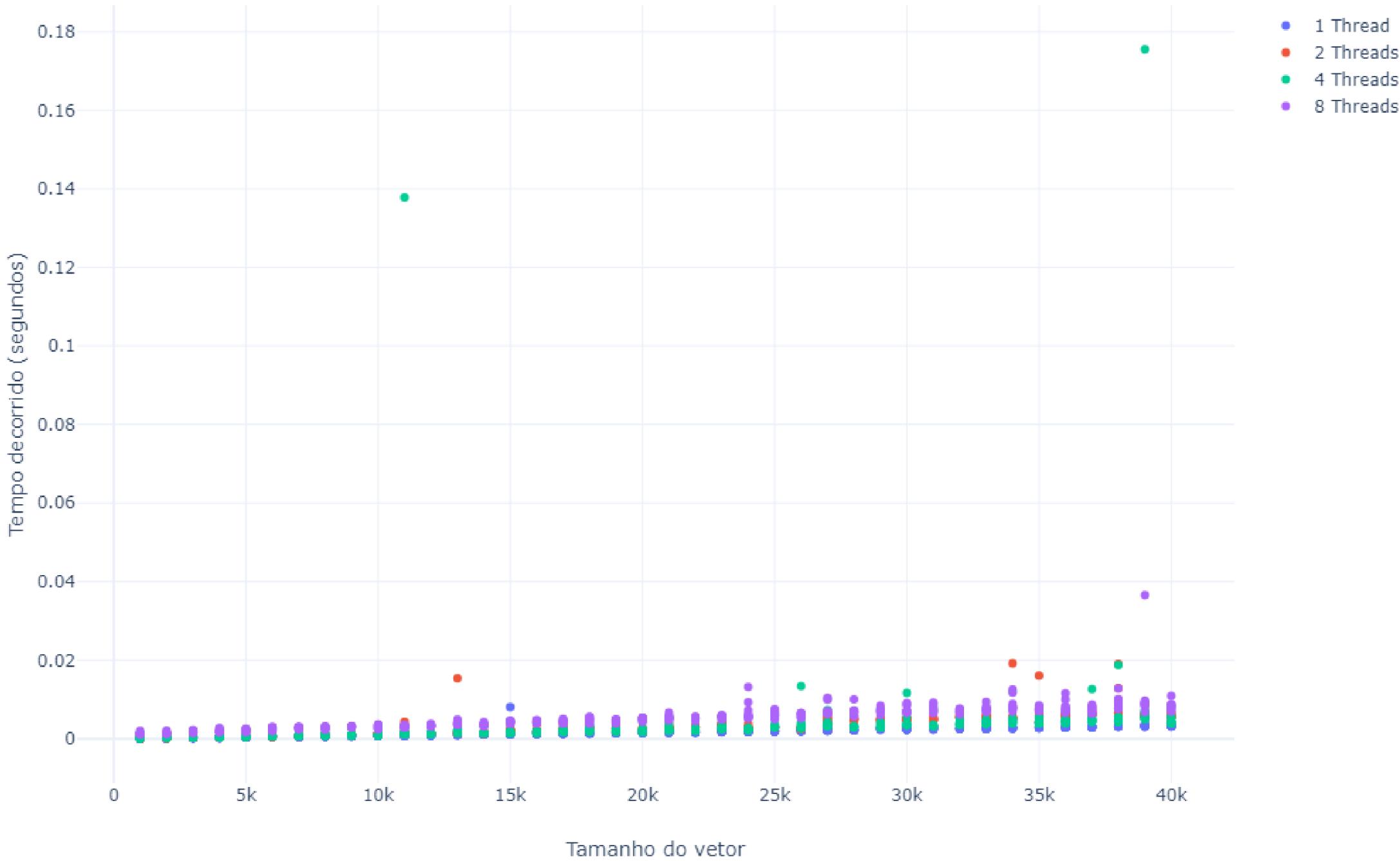
Tempo decorrido de cada execução do algoritmo implementado com Open MPI



Resultados

O algoritmo implementado com Open MP e executado com apenas **1 thread** também se comporta como o quicksort.

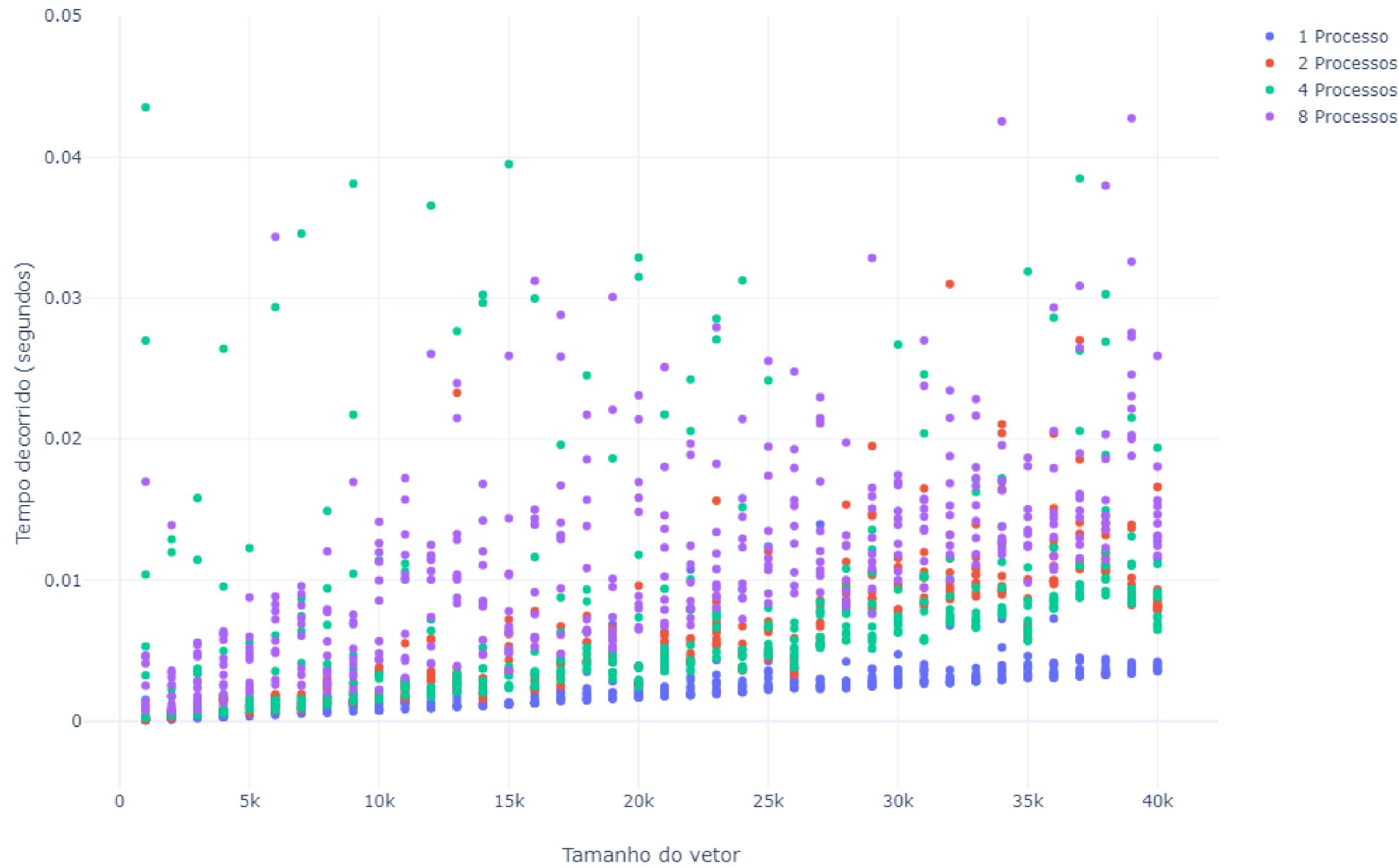
Tempo decorrido de cada execução do algoritmo implementado com OpenMP



Resultados

Há grande variação do tempo decorrido de cada execução do algoritmo implementado com Open MPI.

Tempo decorrido de cada execução do algoritmo implementado com Open MPI



Resultados

Já o algoritmo implementado com OpenMP apresenta maior estabilidade no tempo decorrido de cada execução.

Tempo decorrido de cada execução do algoritmo implementado com OpenMP

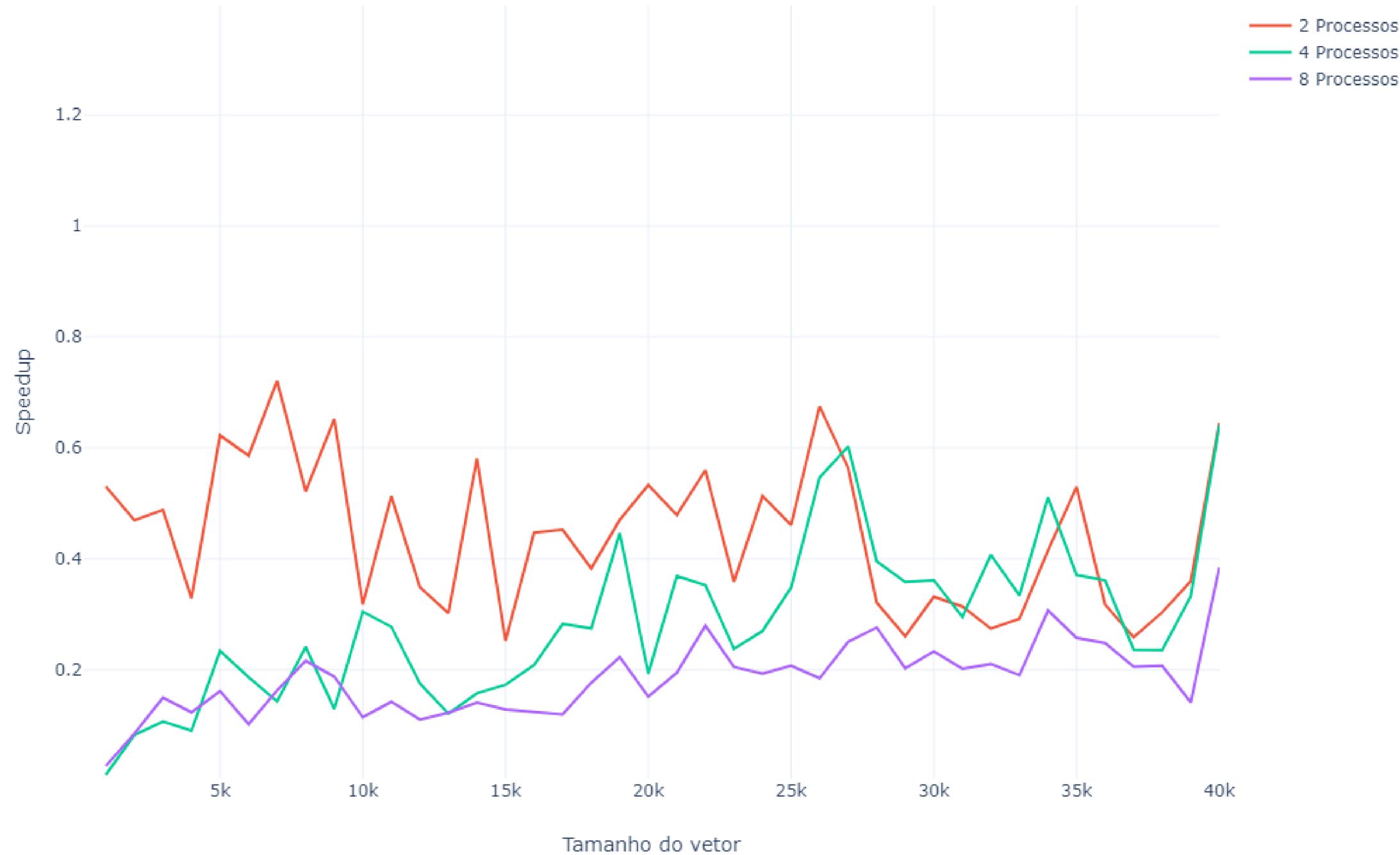


Resultados

No algoritmo implementado com Open MPI, para cada tamanho de vetor foi calculado o **speedup** como a razão entre a média do tempo decorrido do algoritmo **sequencial (1 processo)** pela média do tempo decorrido do algoritmo **paralelo**, ou:

$$\frac{t_s}{t_p}$$

Speedup do algoritmo implementado com Open MPI

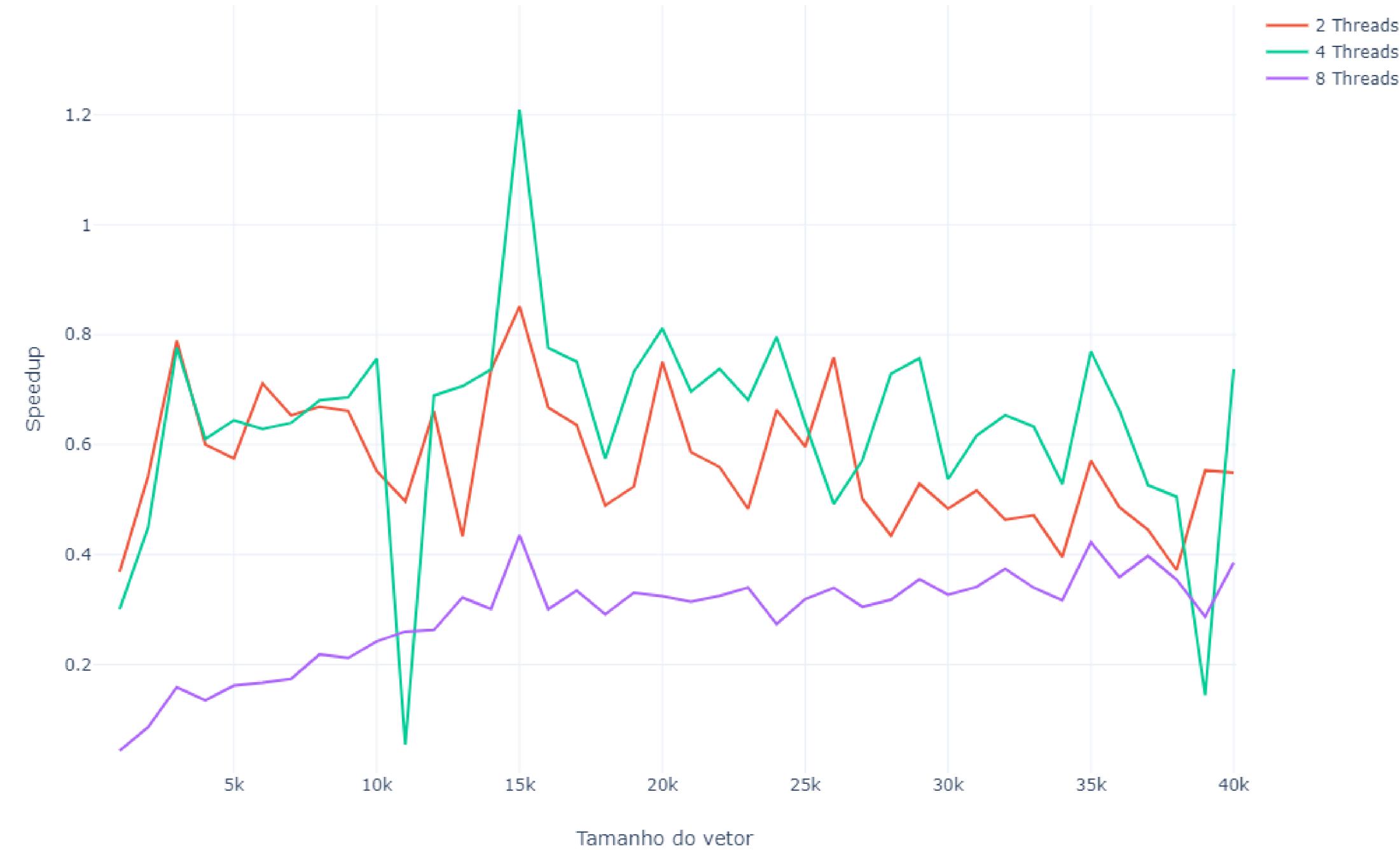


Resultados

No algoritmo implementado com OpenMP, para cada tamanho de vetor foi calculado o **speedup** como a razão entre a média do tempo decorrido do algoritmo **sequencial (1 thread)** pela média do tempo decorrido do algoritmo **paralelo**, ou:

$$\frac{t_s}{t_p}$$

Speedup do algoritmo implementado com OpenMP

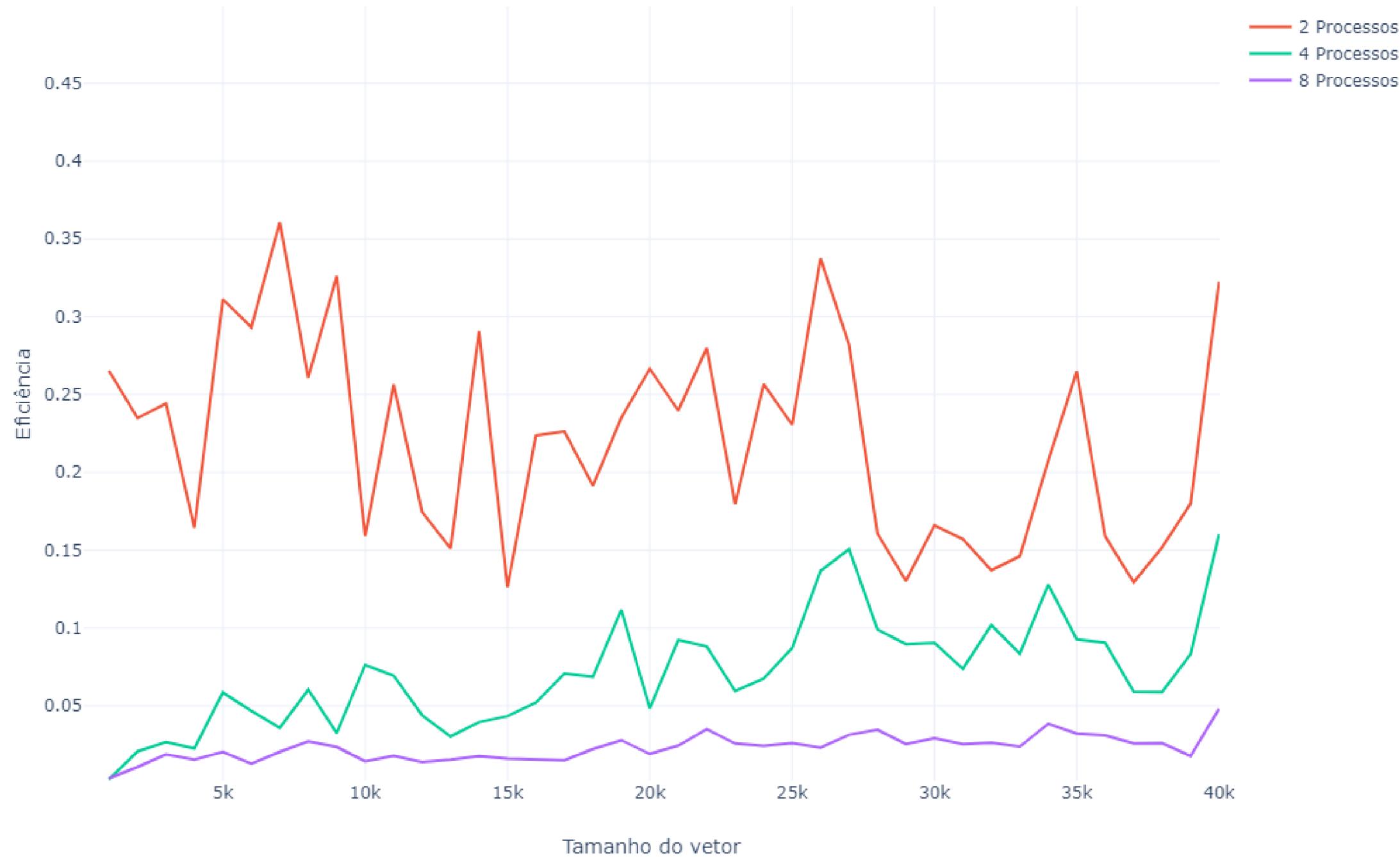


Resultados

No algoritmo implementado com Open MPI, para cada tamanho de vetor foi calculado a **eficiência** como a razão entre a média do tempo decorrido do algoritmo **sequencial (1 processo)** pela média do tempo decorrido do algoritmo **paralelo** vezes o número de **processos**, ou:

$$\frac{t_s}{t_p \times n_p}$$

Eficiência do algoritmo implementado com Open MPI

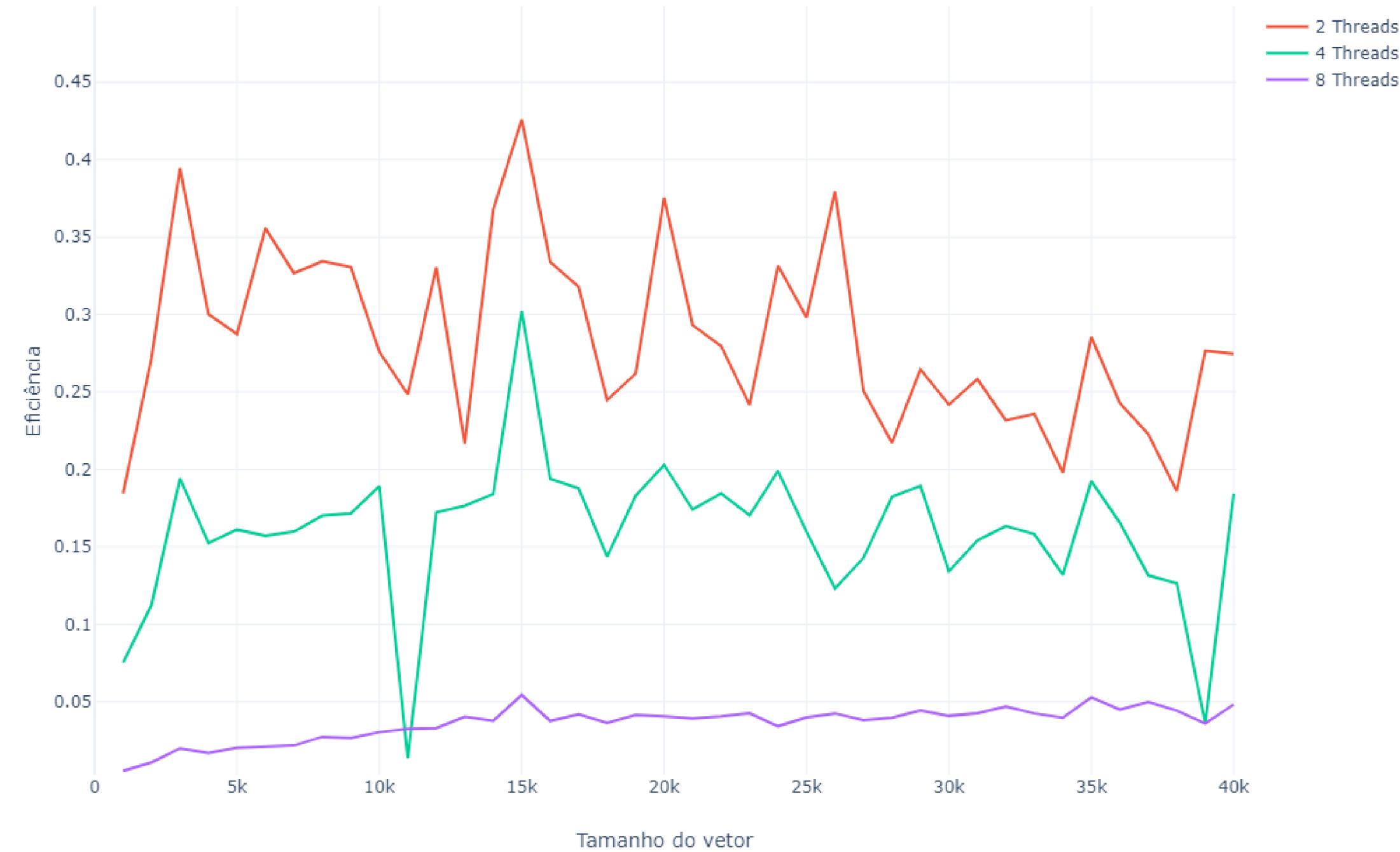


Resultados

No algoritmo implementado com OpenMP, para cada tamanho de vetor foi calculado a **eficiência** como a razão entre a média do tempo decorrido do algoritmo **sequencial (1 threads)** pela média do tempo decorrido do algoritmo **paralelo** vezes o número de **threads**, ou:

$$\frac{t_s}{t_p \times n_t}$$

Eficiência do algoritmo implementado com OpenMP



Código Fonte



Referências

-
- 1 pp10.pdf
 - 2 en.wikipedia.org/wiki/Odd-even_sort
 - 3 linux.die.net/man/3/clock_gettime
-