OpenMP - Eficiência por Threads

Algoritmo de MergeSort

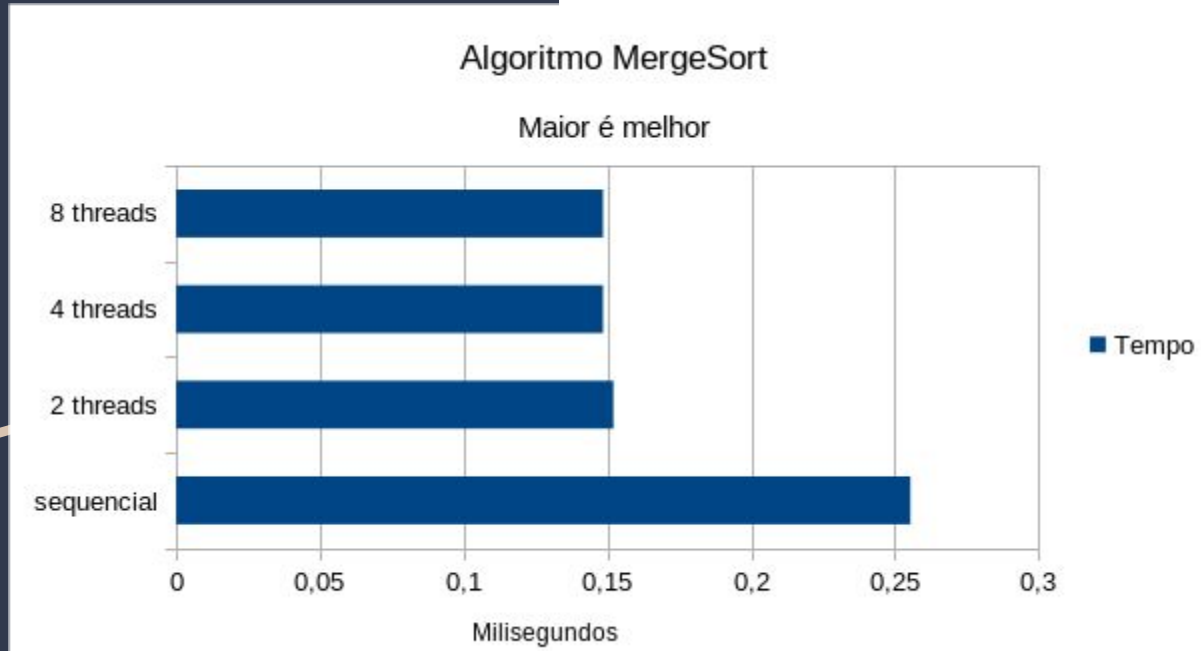
```
void Merge(int vec[], int vecSize) {
        int mid;
        int i, j, k;
        int* tmp;

        tmp = (int*) malloc(vecSize * sizeof(int));
        if (!tmp)
                exit(1);

        mid = vecSize / 2;

        i = 0;
        j = mid;
        k = 0;

        while (i < mid && j < vecSize) {
                if (vec[i] < vec[j])
                        tmp[k] = vec[i++];
                else
                        tmp[k] = vec[j++];
                ++k;
        }

        if (i == mid)
                while (j < vecSize)
                        tmp[k++] = vec[j++];
        else
                while (i < mid)
                        tmp[k++] = vec[i++];

        for (i = 0; i < vecSize; ++i)
                vec[i] = tmp[i];

        free(tmp);
}
```

Região paralelizada

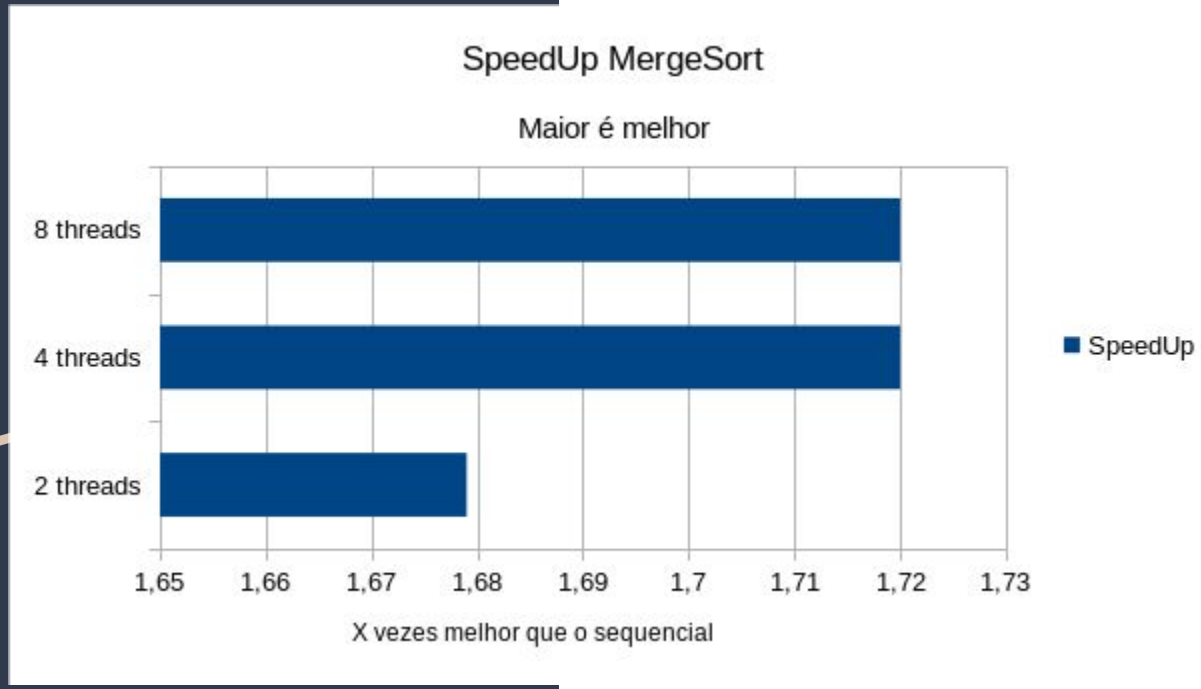
```
void mergesort(int arr[], int size, int thread){
        int mid;
        if(size > 1) {
                mid = size / 2;
                if(thread > 1) {
                        #pragma omp parallel sections
                        {
                                #pragma omp section
                                {
                                        mergesort(arr, mid, thread/2);
                                }
                                #pragma omp section
                                {
                                        mergesort(arr + mid, size - mid, thread/2);
                                }
                        }
                } else {
                        mergesort(arr, mid, 1);
                        mergesort(arr + mid, size - mid, 1);
                }
                Merge(arr, size);
        }
}
```

Algoritmo disponível no Github: https://github.com/cart-pucminas/teaching-parallelism-freshmen/tree/master/Merge%20Sort

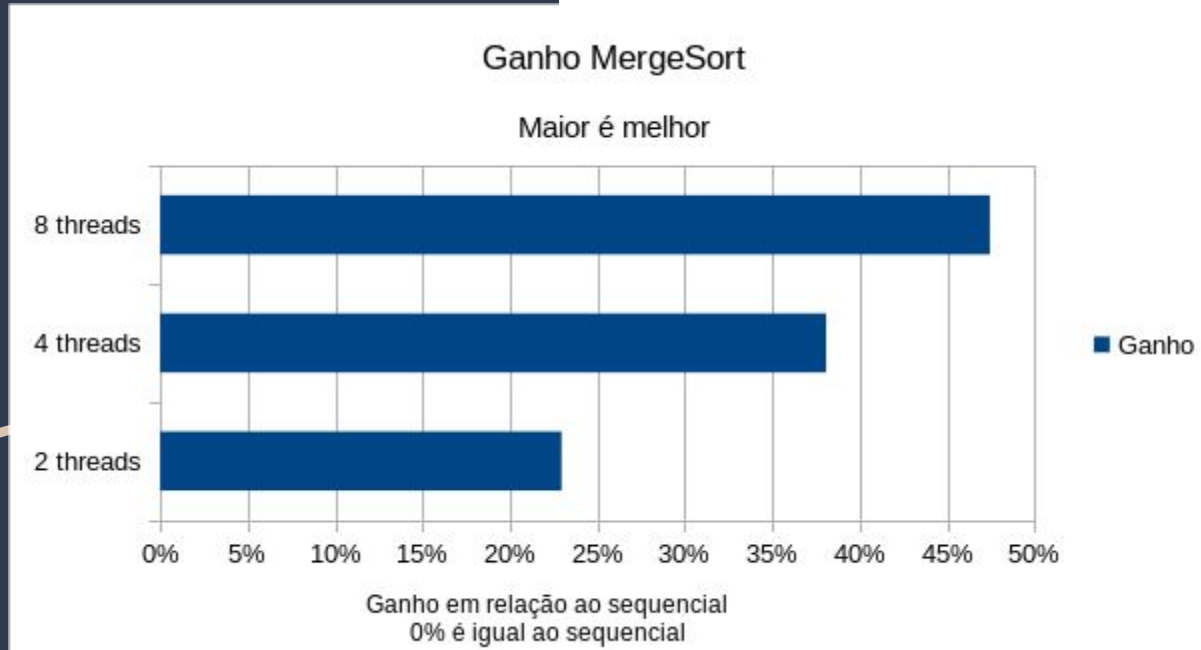
Tempos


Algoritmo MergeSort
Maior é melhor
8 threads
4 threads
2 threads
sequencial
0
0,05
0,1
0,15
0,2
0,25
0,3
Milisegundos
Tempo

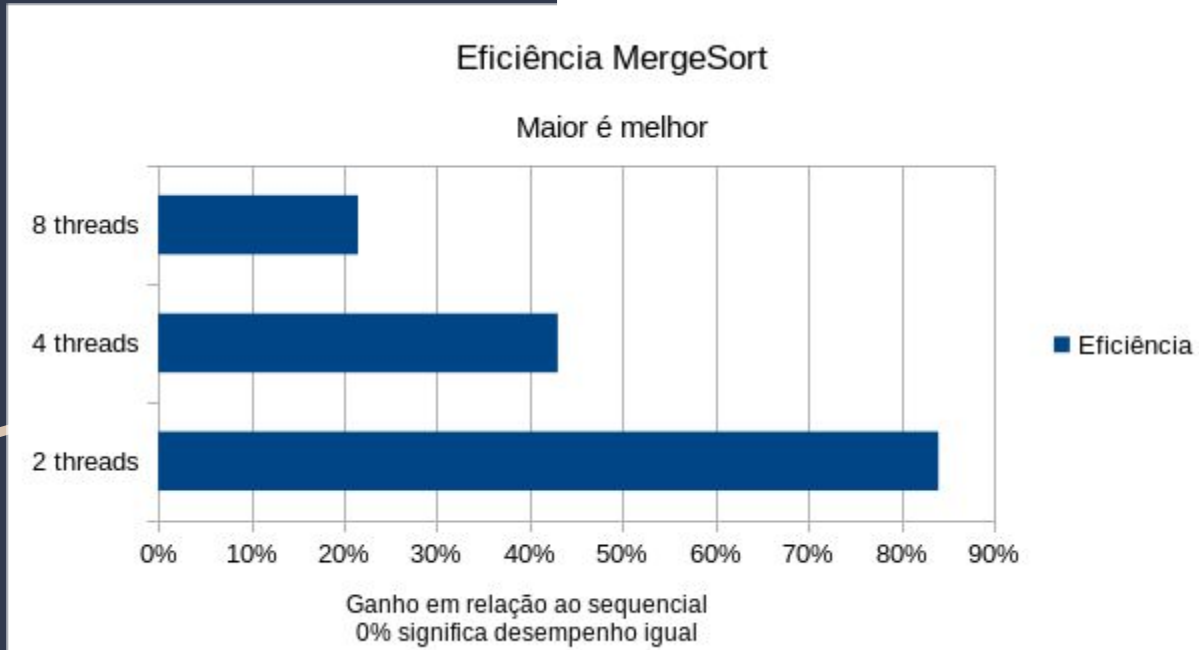
Speed Up


SpeedUp MergeSort
Maior é melhor
8 threads
4 threads
2 threads
1,65
1,66
1,67
1,68
1,69
1,7
1,71
1,72
1,73
X vezes melhor que o sequencial
SpeedUp

Ganho


Ganho MergeSort
Maior é melhor
8 threads
4 threads
2 threads
0%
5%
10%
15%
20%
25%
30%
35%
40%
45%
50%
Ganho
Ganho em relação ao sequencial
0% é igual ao sequencial

Eficiência


Eficiência MergeSort
Maior é melhor
8 threads
4 threads
2 threads
0%
10%
20%
30%
40%
50%
60%
70%
80%
90%
Eficiência
Ganho em relação ao sequencial
0% significa desempenho igual

Algoritmo de SelectionSort

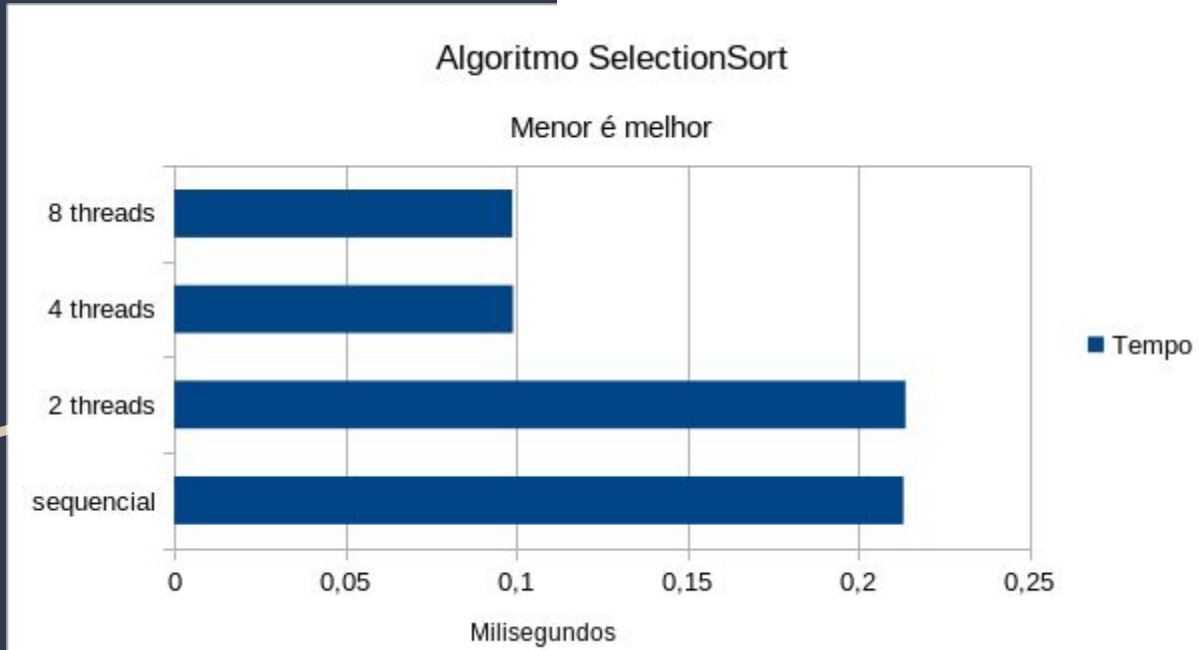
```
void SelectionSort (int *array, int size) {
        int i, j, min;
        for (i = 0; i < (size-1); i++) {
                min = i;
                #pragma omp parallel
                {
                        int local_min = i;
                        #pragma omp for
                        for (j = (i+1); j < size; j++){
                                if(array[j] < array[local_min])
                                        local_min = j;
                        }
                        #pragma omp critical
                        if(array[local_min] < array[min])
                                min = local_min;
                }
                if (i != min) {
                        int swap = array[i];
                        array[i] = array[min];
                        array[min] = swap;
                }
        }
}
```

Região paralelizada

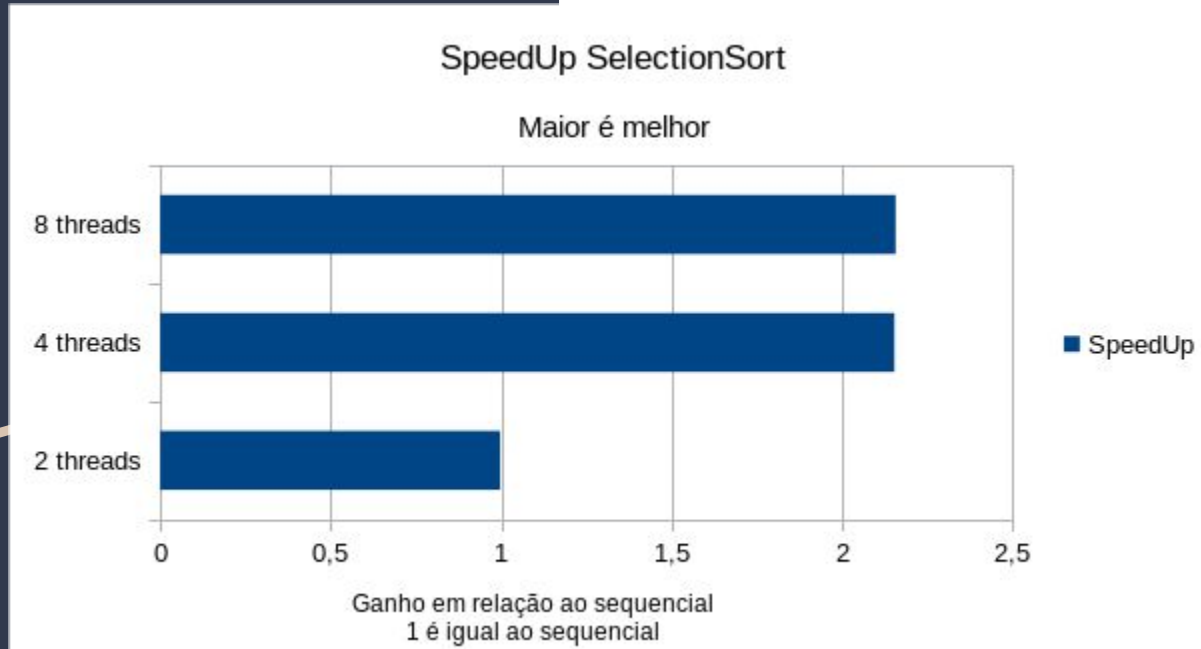
```
int main(int argc, char** argv) {
        int size = 15000, algorithm, i, *arr, opt;
        arr = malloc(size* sizeof(int));
        srand(time(NULL));
        for (i = 0; i < size; i++)
                arr[i] = rand()%size;
        double start, end;

        omp_set_num_threads(8);
        start = omp_get_wtime();

        SelectionSort(arr, size);

        end = omp_get_wtime();

        printf("Tempo: %.3f\n",end - start);
        if(IsSort(arr, size) == 1)
                printf("Result: Sorted\n");
        else
                printf("Result: Not Sorted\n");
        return 0;
}
```

Algoritmo disponível no Github: https://github.com/cart-pucminas/teaching-parallelism-freshmen/tree/master/Selection%20Sort

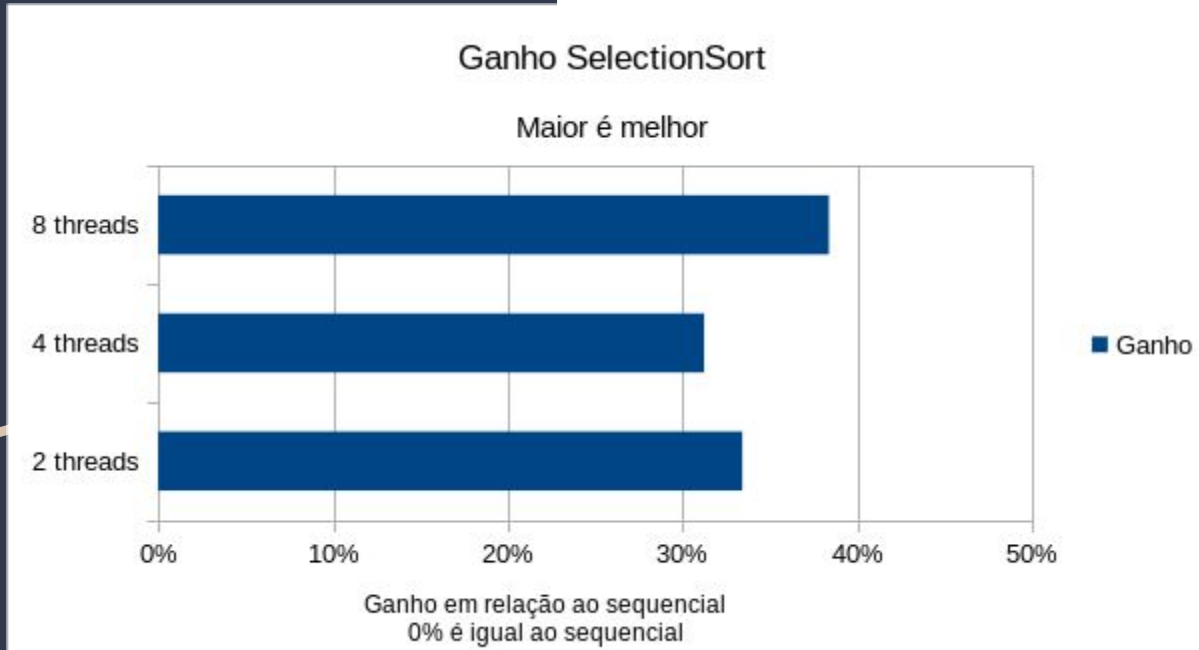
Tempos


Algoritmo SelectionSort
Menor é melhor
8 threads
4 threads
2 threads
sequencial
0
0,05
0,1
0,15
0,2
0,25
Milisegundos
Tempo

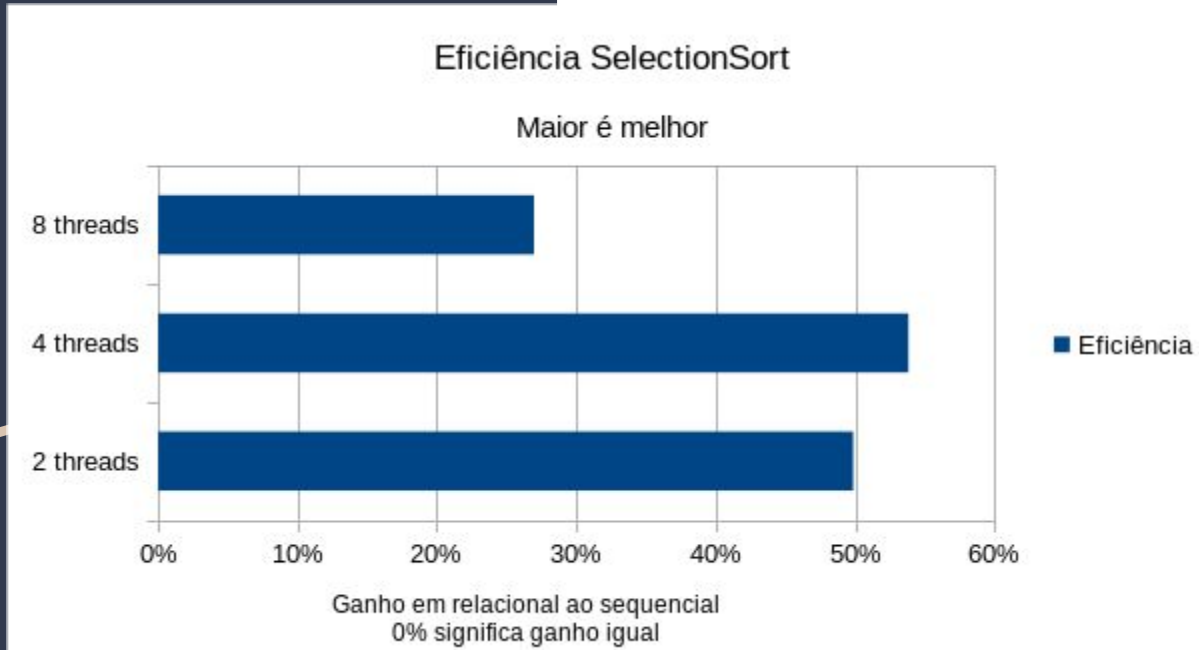
Speed Up


SpeedUp SelectionSort
Maior é melhor
8 threads
4 threads
2 threads
0
0,5
1
1,5
2
2,5
Ganho em relação ao sequencial
1 é igual ao sequencial
SpeedUp

Ganho


Ganho SelectionSort
Maior é melhor
8 threads
4 threads
2 threads
0%
10%
20%
30%
40%
50%
Ganho
Ganho em relação ao sequencial
0% é igual ao sequencial

Eficiência


Eficiência SelectionSort
Maior é melhor
8 threads
4 threads
2 threads
0%
10%
20%
30%
40%
50%
60%
Eficiência
Ganho em relacional ao sequencial
0% significa ganho igual

Algoritmo de ShellSort

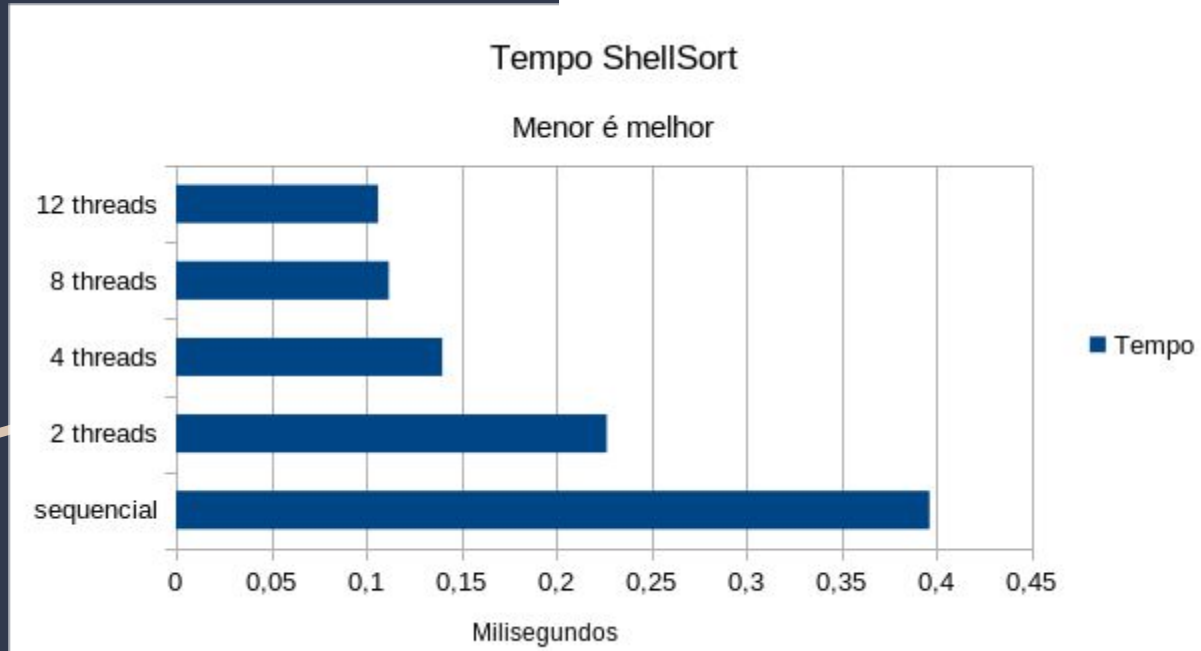
```
void shellsort(int arr[], int n){
        int gap, i, j, grupoId, temp;
        for (gap = n/2; gap > 0; gap /= 2)
                #pragma omp parallel for private(j, i)
                for(grupoId = 0; grupoId < gap; grupoId++)
                        for (i=gap+grupoId; i<n-grupoId; i+=gap) {
                                int key = arr[i];
                                j = i - gap;
                                while (j >= 0 && arr[j] > key) {
                                        arr[j+gap] = arr[j];
                                        j-=gap;
                                }

                                arr[j+gap] = key;
                        }
}
```

Região paralelizada

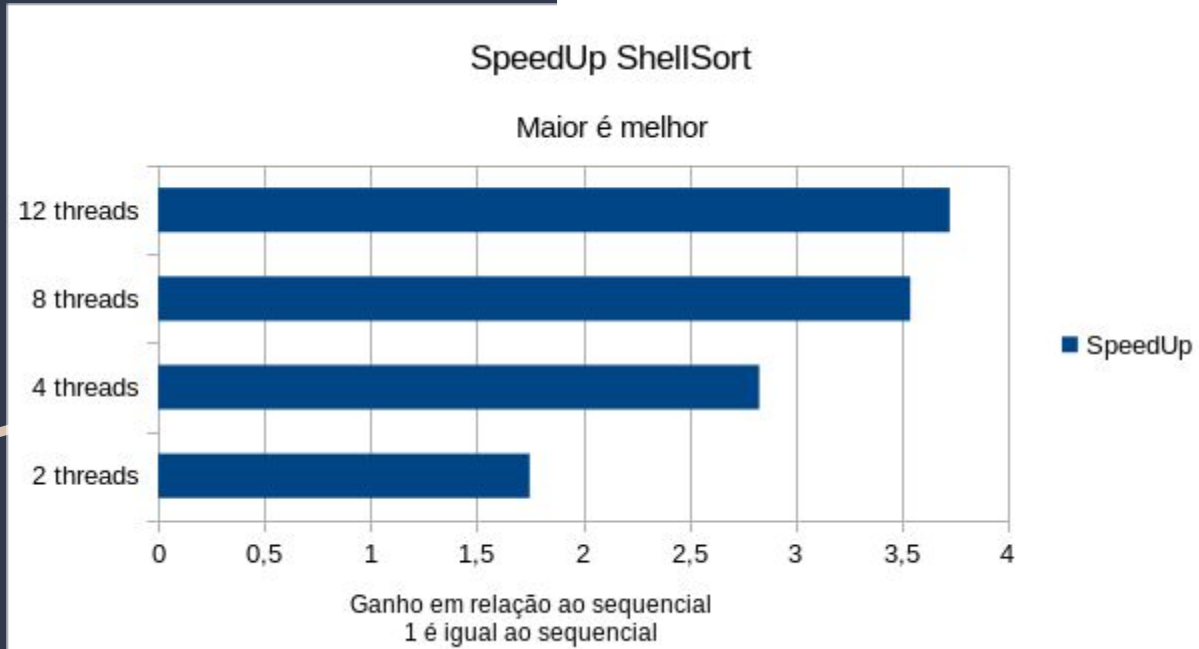
```c
int main(int argc, char** argv) {
        int size = 1500000, algorithm, i, *arr, opt;
        arr = malloc(size* sizeof(int));
        srand(time(NULL));
        for (i = 0; i < size; i++)
                arr[i] = rand()%size;
        double start, end;

        omp_set_num_threads(12);
        start = omp_get_wtime();

        shellsort(arr, size);

        end = omp_get_wtime();

        printf("Tempo: %.3f\n",end - start);
        if(IsSort(arr, size) == 1)
                printf("Result: Sorted\n");
        else
                printf("Result: Not Sorted\n");
        return 0;
}
```

Algoritmo disponível no Github:
https://github.com/cart-pucminas/teaching-parallelism-freshmen/tree/master/Shell%20Sort

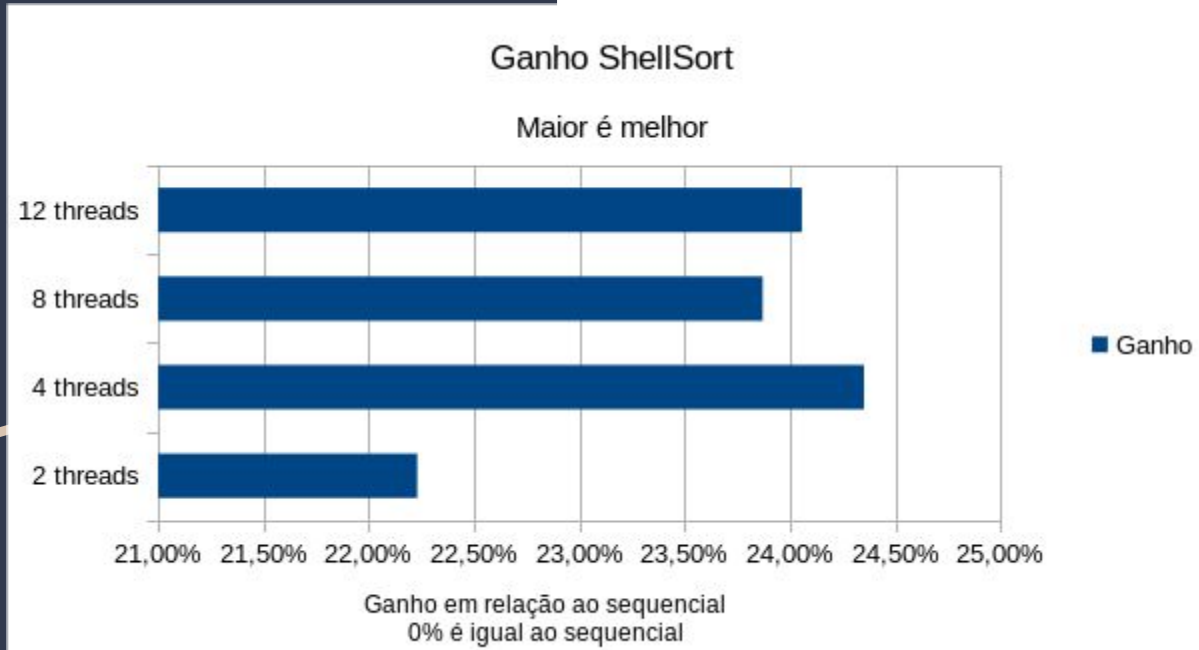
Tempos


Tempo ShellSort
Menor é melhor
12 threads
8 threads
4 threads
2 threads
sequencial
0
0,05
0,1
0,15
0,2
0,25
0,3
0,35
0,4
0,45
Milisegundos
Tempo

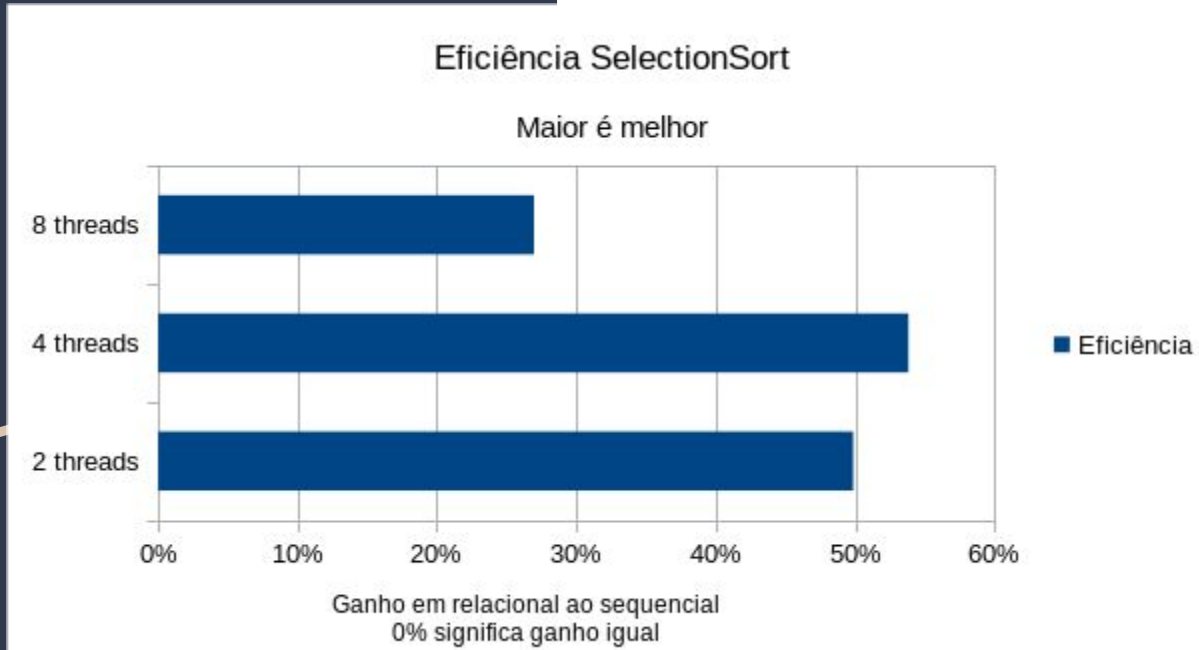
Speed Up


SpeedUp ShellSort
Maior é melhor
12 threads
8 threads
4 threads
2 threads
0
0,5
1
1,5
2
2,5
3
3,5
4
SpeedUp
Ganho em relação ao sequencial
1 é igual ao sequencial

Ganho


Ganho ShellSort
Maior é melhor
12 threads
8 threads
4 threads
2 threads
21,00% 21,50% 22,00% 22,50% 23,00% 23,50% 24,00% 24,50% 25,00%
Ganho em relação ao sequencial
0% é igual ao sequencial
Ganho

Eficiência


Eficiência SelectionSort
Maior é melhor
8 threads
4 threads
2 threads
0%
10%
20%
30%
40%
50%
60%
Eficiência
Ganho em relacional ao sequencial
0% significa ganho igual

Hardware utilizado

Processador: Ryzen 5 3600XT - 6 cores /12 threads

RAM: 16GB - 2400mhz

OS: Arch linux | versão: X86_64