Quicksort

O que é?

O quicksort, desenvolvido em 1960 por Tony Hoare, é um dos algoritmos de ordenação mais eficiente e elegante.
Utiliza como estratégia "divide and conquer":

- Escolher um valor(pivot)
- Partir a sequência em duas partes
  - valores>pivot
  - pivot<valores
- Recursivamente ordenamos os dois arrays resultantes

Conjunto de Dados

Quatro inputs de diferentes tamanhos:

- 500,000 Integers
- 250,000 Integers
- 100,000 Integers
- 50,000 Integers

Código Base

```
1  void quicksort(int *array, int lo, int hi){
2          int i=lo,j=hi,h;
3          int x=array[(lo+hi)/2];
4      //partition
5      do  {
6          while(array[i]<x) i++;
7          while(array[j]>x) j--;
8          if(i<=j){
9          h=array[i]; array[i]=array[j]; array[j]=h;
10         i++; j--;
11         }
12     }while(i<=j);
13     //recursion
14     if(lo<j) quicksort(array,lo,j);
15     if(i<hi) quicksort(array,i,hi);
16 }
```

openMP

Main

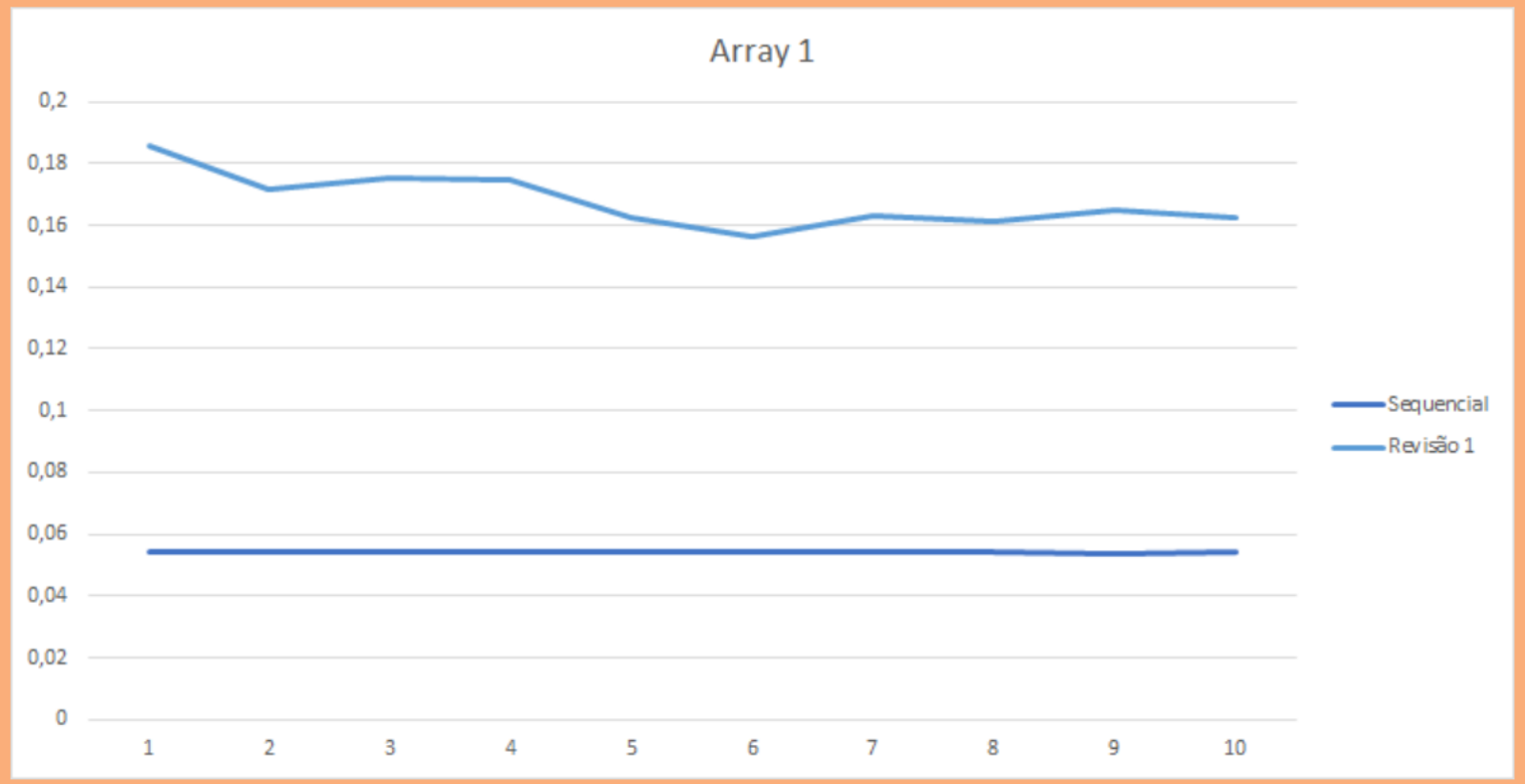
```
int main(int argc, char* argv[])
{
    //Open the file where the array is at
    FILE *fp = fopen(argv[1],"r");
    int arraySize = atoi(argv[2]);
        int arr[ARRAY_MAX_SIZE];
    //Read the values from file do array arr
    readFile(fp,arr,arraySize);
    double start_time, run_time;
    // See Beginning time
    start_time = omp_get_wtime();
    //Make QuickSort
    #pragma omp parallel
        #pragma omp single
        {
        quicksort(arr, 0, arraySize-1);
        }
    //Beginning time - Final time
    run_time = omp_get_wtime() - start_time;
        printf("Run time was %lf seconds\n ",run_time);
        return 0;
}
```

Revisão 1

```
1  int main(int argc, char* argv[])
2  {
3      //Open the file where the array is at
4      FILE *fp = fopen(argv[1],"r");
5      int arraySize = atoi(argv[2]);
6          int arr[ARRAY_MAX_SIZE];
7      //Read the values from file do array arr
8      readFile(fp,arr,arraySize);
9      double start_time, run_time;
10     // See Beginning time
11     start_time = omp_get_wtime();
12     //Make QuickSort
13     #pragma omp parallel
14         quicksort(arr, 0, arraySize-1);
15     //Beginning time - Final time
16     run_time = omp_get_wtime() - start_time;
17         printf("Run time was %lf seconds\n ",run_time);
18         return 0;
19 }
```

```
1  void quicksort(int * array, int lo, int hi){
2          int pivotElement;
3      //Se o array for menor que 10000, o custo de atribuir threads não compensa,
4          if((hi - lo + 1) < 10000) {
5                  serial_quicksort(array, lo, hi);
6          } else {
7                  int i=lo,j=hi,h;
8                  int x=array[(lo+hi)/2];
9                  do{
10                 while(array[i]<x) i++;
11                 while(array[j]>x) j--;
12                 if(i<=j){
13                         h=array[i]; array[i]=array[j]; array[j]=h;
14                         i++; j--;
15                 }
16         }while(i<=j);
17                 #pragma omp task
18                 {
19                         quicksort(array,lo,j);
20                 }
21                 #pragma omp task
22                 {
23                         quicksort(array,i,hi);
24                 }
25         }
26 }
```

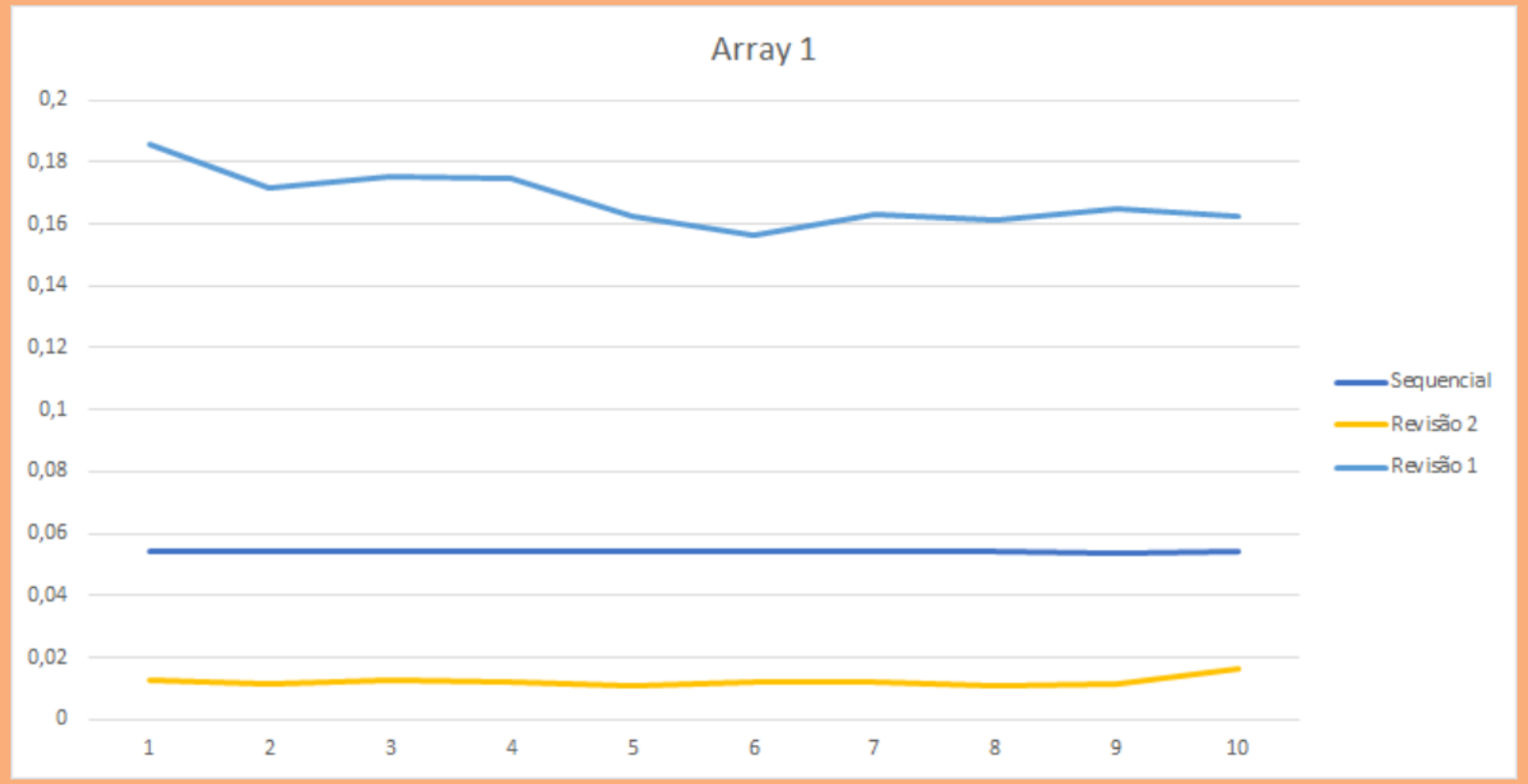
Resultados


Array 1
0,2
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
1
2
3
4
5
6
7
8
9
10
Sequencial
Revisão 1

Revisão 2

```
1  int main(int argc, char* argv[])
2  {
3      //Open the file where the array is at
4      FILE *fp = fopen(argv[1],"r");
5      int arraySize = atoi(argv[2]);
6          int arr[ARRAY_MAX_SIZE];
7      //Read the values from file do array arr
8      readFile(fp,arr,arraySize);
9      double start_time, run_time;
10     // See Beginning time
11     start_time = omp_get_wtime();
12     //Make QuickSort
13     #pragma omp parallel
14     #pragma omp single
15     {
16                 quicksort(arr, 0, arraySize-1);
17     }
18     //Beginning time - Final time
19     run_time = omp_get_wtime() - start_time;
20         printf("Run time was %lf seconds\n ",run_time);
21         return 0;
22 }
```

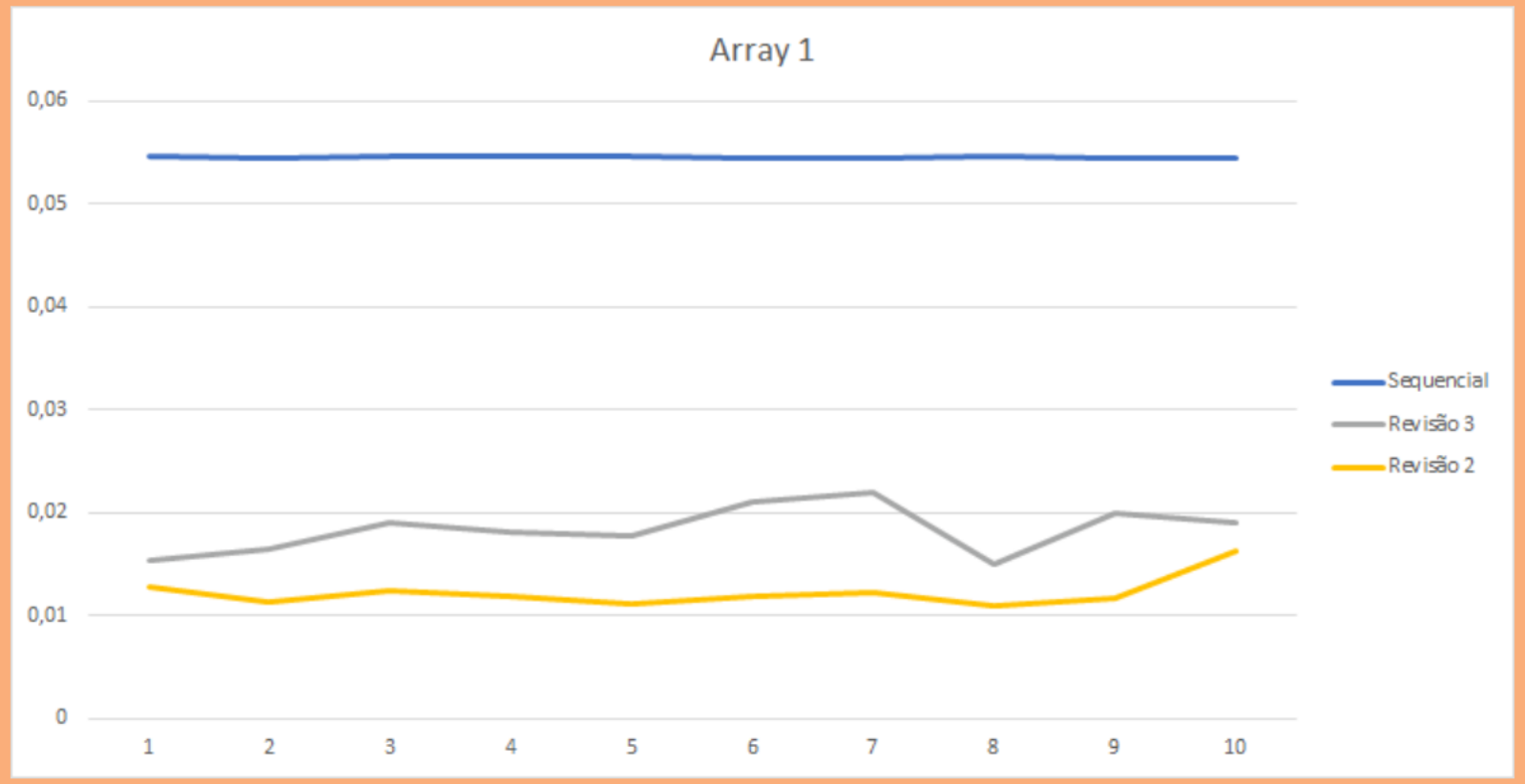
Resultados


Array 1
0,2
0,18
0,16
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
1
2
3
4
5
6
7
8
9
10
Sequencial
Revisão 2
Revisão 1

Revisão 3

```
1  void quicksort(int * array, int lo, int hi){
2          int pivotElement;
3      //Se o array for menor que 10000, o custo de atribuir threads não compens
4          if((hi - lo + 1) < 10000) {
5                  serial_quicksort(array, lo, hi);
6          }
7      else {
8                  int i=lo,j=hi,h;
9                  int x=array[(lo+hi)/2];
10                 do{
11                 while(array[i]<x) i++;
12                 while(array[j]>x) j--;
13                 if(i<=j){
14                         h=array[i]; array[i]=array[j]; array[j]=h;
15                         i++; j--;
16                 }
17         }while(i<=j);
18                 #pragma omp task
19                 {
20                         quicksort(array,lo,j);
21                 }
22                 #pragma omp task
23                 {
24                         quicksort(array,i,hi);
25                 }
26                 #pragma omp taskwait
27         }
28 }
```

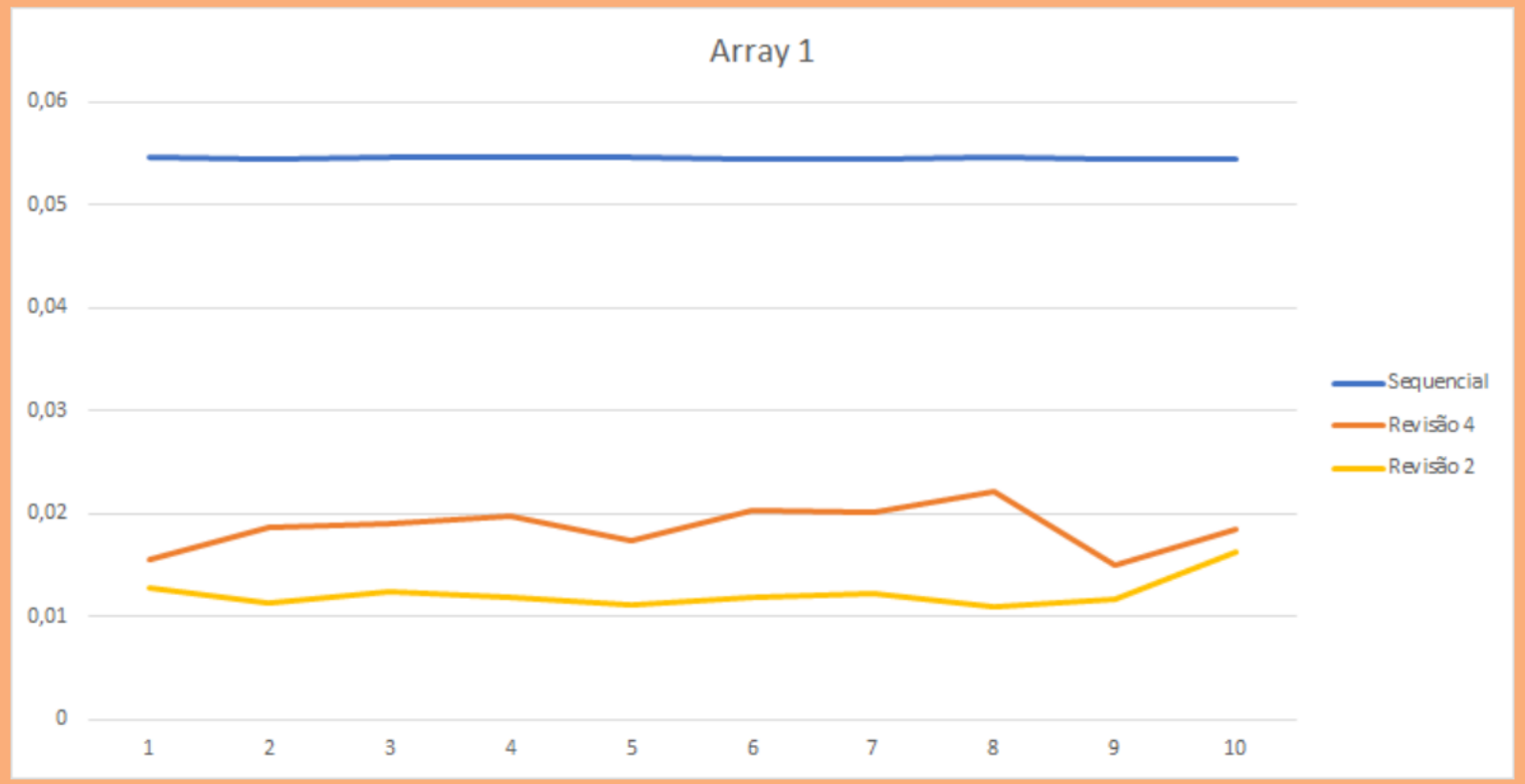
Resultados


Array 1
0,06
0,05
0,04
0,03
0,02
0,01
0
1
2
3
4
5
6
7
8
9
10
Sequencial
Revisão 3
Revisão 2

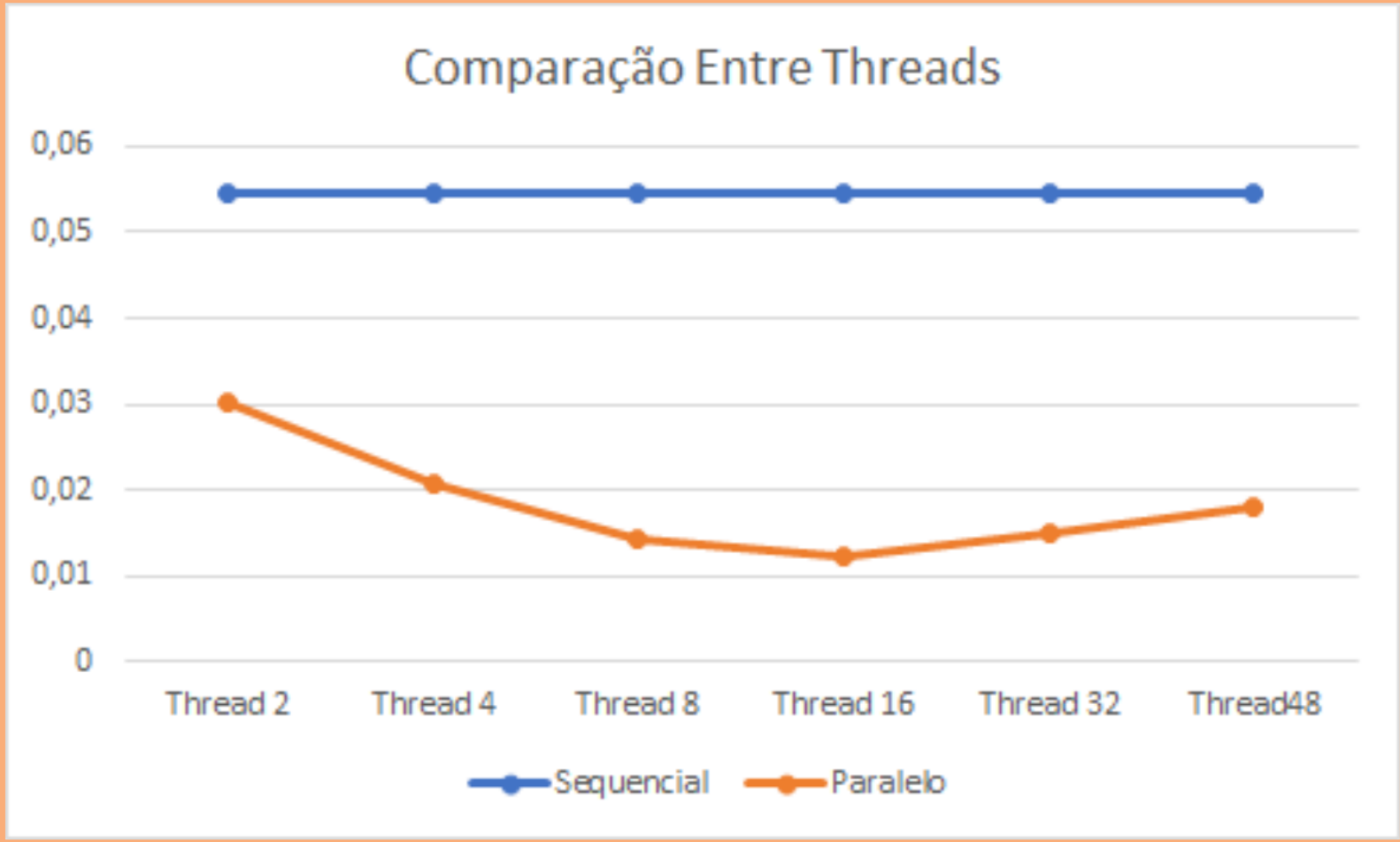
Revisão 4

```
1  void quicksort(int * array, int lo, int hi){
2          int pivotElement;
3      //Se o array for menor que 10000, o custo de atribuir threads não compensa
4          if((hi - lo + 1) < 10000) {
5                  serial_quicksort(array, lo, hi);
6          }
7      else {
8                  int i=lo,j=hi,h;
9                  int x=array[(lo+hi)/2];
10                 do{
11                 while(array[i]<x) i++;
12                 while(array[j]>x) j--;
13                 if(i<=j){
14                         h=array[i]; array[i]=array[j]; array[j]=h;
15                         i++; j--;
16                 }
17         }while(i<=j);
18                 #pragma omp task default(shared)
19                 {
20                         quicksort(array,lo,j);
21                 }
22                 #pragma omp task default(shared)
23                 {
24                         quicksort(array,i,hi);
25                 }
26         }
27 }
```

Resultados


Array 1
0,06
0,05
0,04
0,03
0,02
0,01
0
1
2
3
4
5
6
7
8
9
10
Sequencial
Revisão 4
Revisão 2

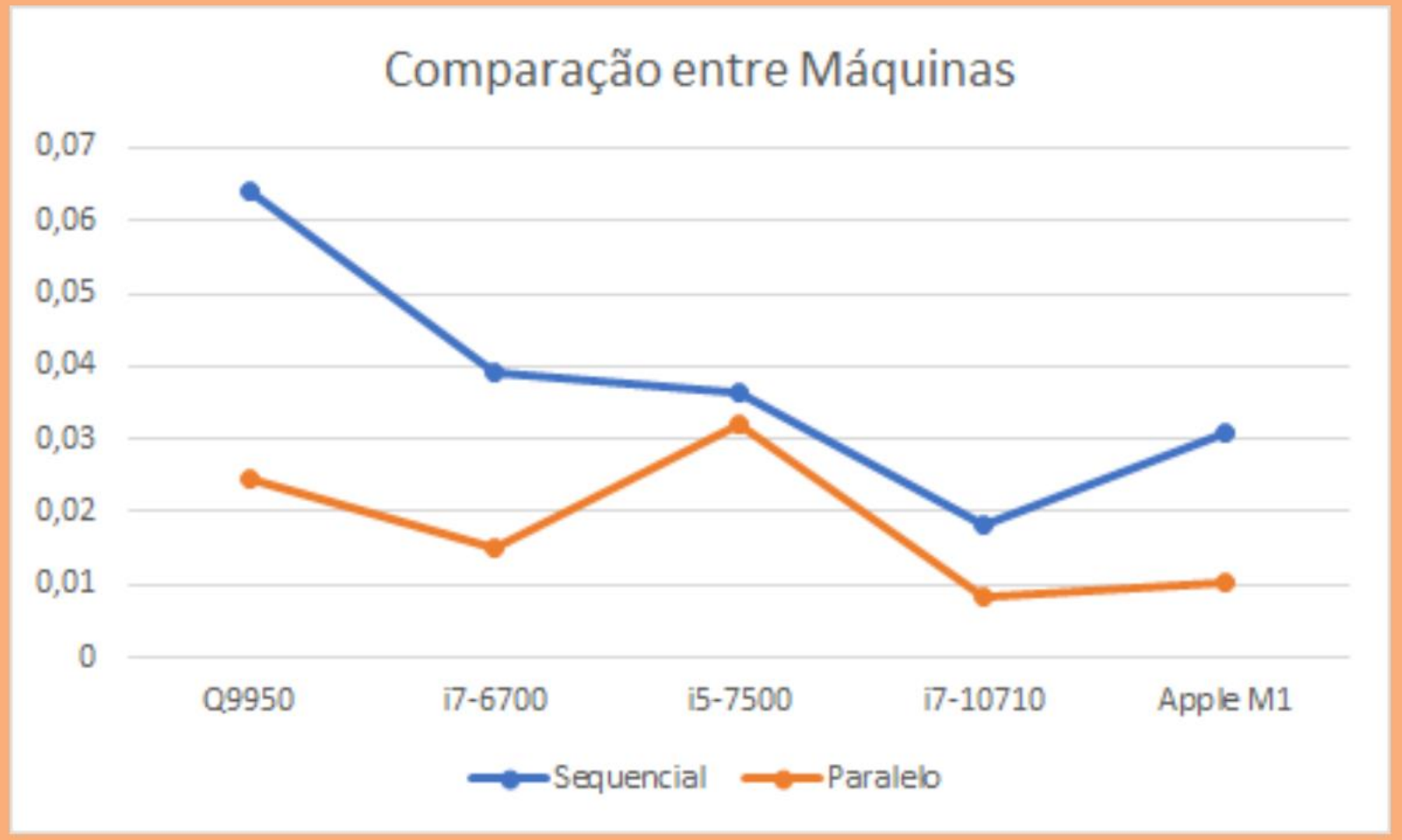
Comparação de Threads



Comparação de Máquinas

Máquina	CPU	GPU	RAM
GL551	i7-6700HQ	GTX960M	16GB
L-Fixo	Core2Quad Q9550	GT710M	6GB
MSI	i7-10710U	GTX 1650	16GB
D-Fixo	i5-7500HQ	GTX1080	16GB
Macbook	Apple M1	Apple M1	16GB
Cluster	———	———	—-

Comparação de Máquinas



Fim