Faculdade de Tecnologia
Universidade Estadual de Campinas

Relatorio Sistemas Operacionais
Quick Sort

Davi Lopes Mezencio - 169759
Fabio do Prado Junior - 170796
William Maiko Balzanello - 188678

Algoritmo em Alto Nível

Utilizamos o algoritmo de ordenação Quick Sort, que consiste em selecionar um pivô (no vetor de números), olhamos o vetor e mudamos todas as posições maiores que o pivô para trás dele e as menores para frente (partition). Vamos fazendo isso recursivamente ordenando a sub lista dos elementos menores e a sub lista dos elementos maiores
Em C (linguagem de alto nível), sua implementação é a seguinte:

```
int partition(float vetor[], int esq, int dir){

   float aux;//trocar a posição
   int contador = esq;

   for(int i=esq+1; i <= dir; i++){
      if(vetor[i] < vetor[esq]){
         contador++;

         aux = vetor[i];
         vetor[i] = vetor[contador];
         vetor[contador] = aux;
      }
   }

   aux = vetor[esq];
   vetor[esq] = vetor[contador];
   vetor[contador] = aux;

   return contador;
}


void quickSort(float vetor[], int esq, int dir){

   int posDividido;

   if(esq < dir){
      posDividido = partition(vetor, esq, dir);
      quickSort(vetor, esq, posDividido-1);
      quickSort(vetor, posDividido+1, dir);
   }

}
```

Na main do programa, devem ser passados somente o Vetor para ser ordenado, a primeira posição(0) e a última posição (n-1) do mesmo.

Video e GitHub

GitHub com todo o codigo fonte:
https://github.com/Davilopesm/MultithreadQuickSort

Vídeo do programa em execução pode ser encontrado em:
https://drive.google.com/open?id=17pqCUqx4Ueev7MG97Gl9yySUAEpPx65J

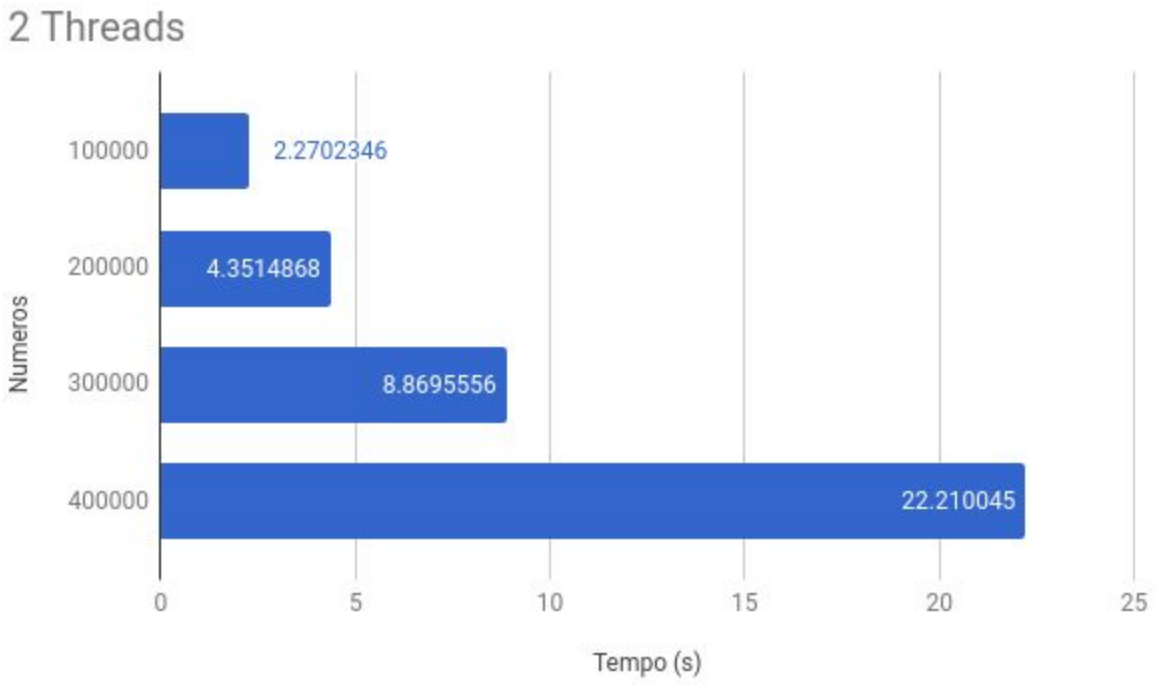
Instruções para compilar

Para compilar o programa tudo que deve-se fazer é:

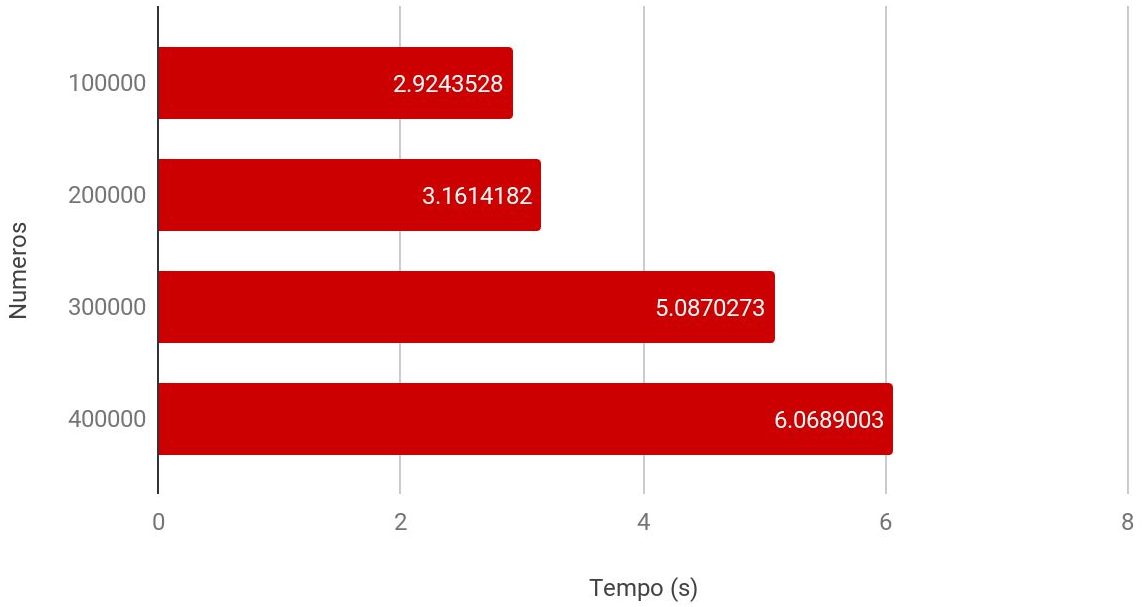
1 - Acessar o github https://github.com/Davilopesm/MultithreadQuickSort
2 - Fazer download do repositorio
3 - Navegar pelo terminal até a página onde estão os arquivos baixados
4 - Dar um “make” no terminal
5 - Digitar “./quick” no terminal
6 - Seguir os passos do programa

Gráficos de tempo de execução

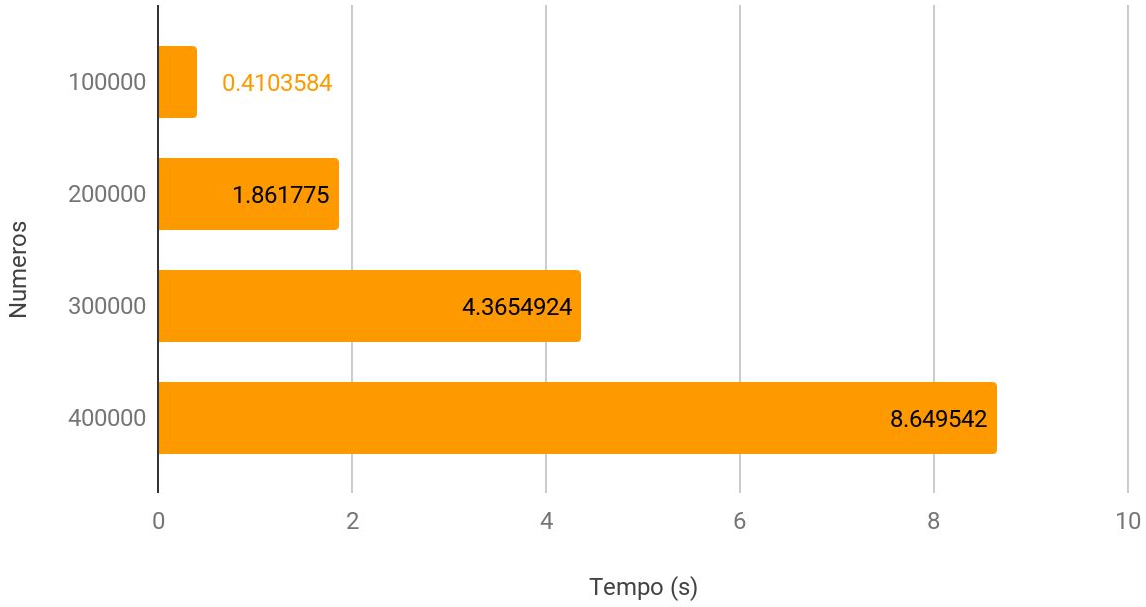
Para fazer os gráficos rodamos o algoritmo 10 vezes para cada número de threads e para cada tamanho de dados. Por exemplo com 100.000 dados, rodamos 10 vezes com 2 threads e fizemos a média do tempo gasto para colocar no gráfico.

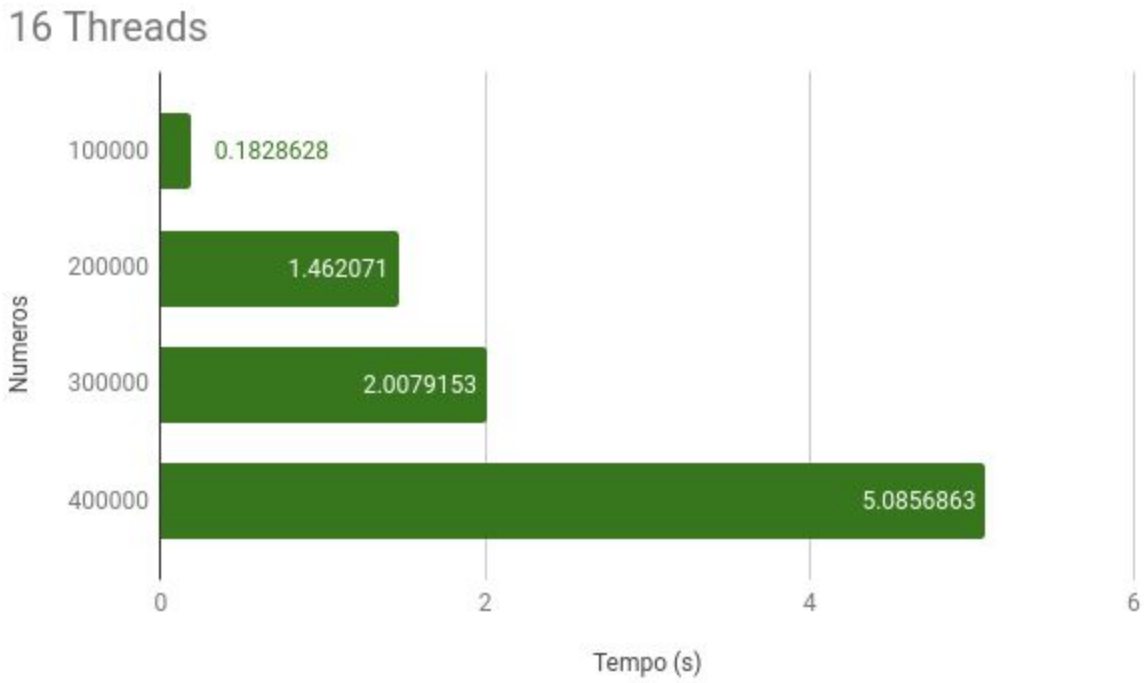


4 Threads

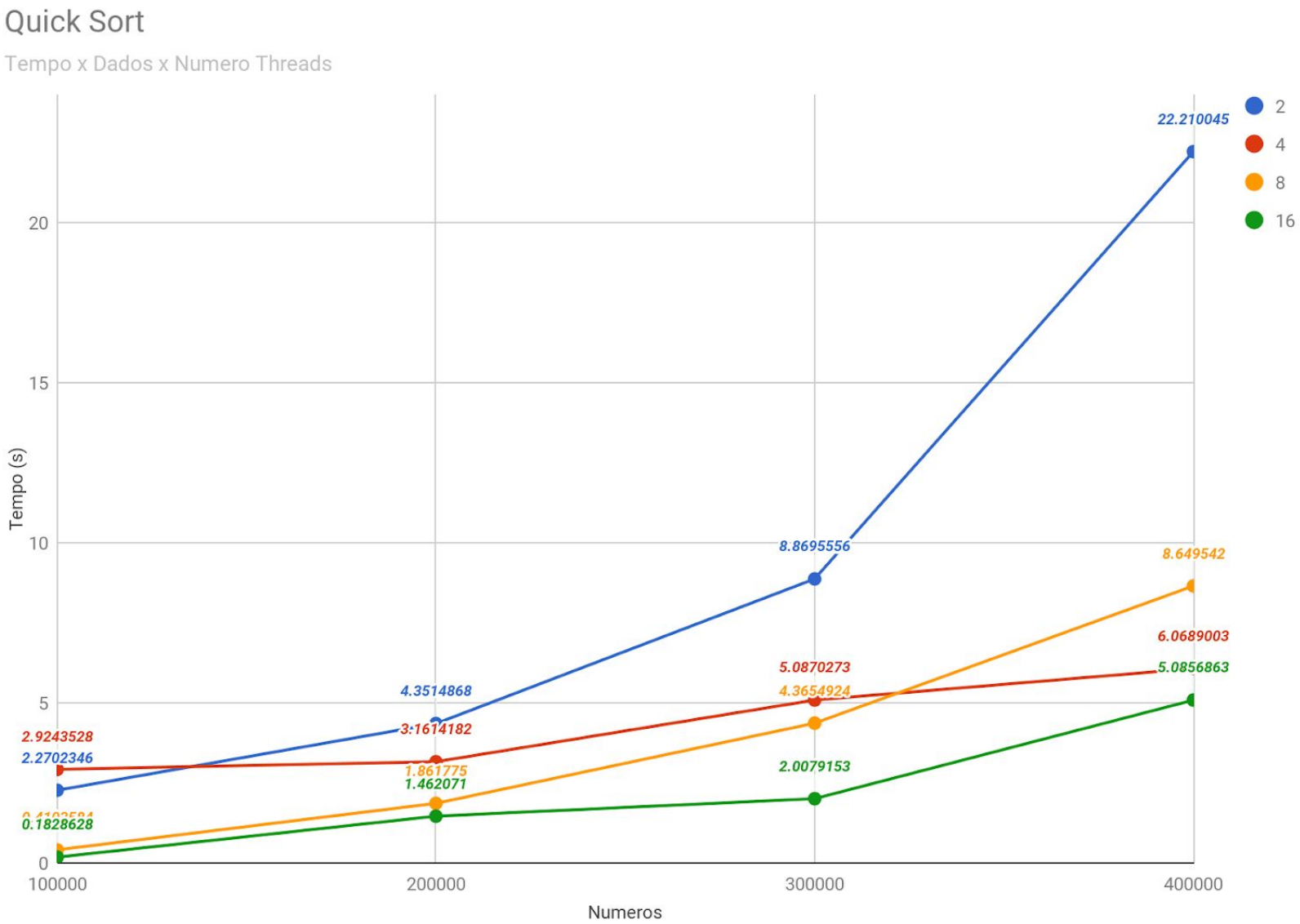


8 Threads




16 Threads
100000
0.1828628
200000
1.462071
300000
2.0079153
400000
5.0856863
Numeros
0
2
4
6
Tempo (s)

Relacionamento de todas as médias dos tempos de execução com número de dados e número de threads:



Conclusões

Podemos concluir com as observações dos números de threads e elementos no vetor, que conforme aumentamos o número de threads disponíveis para a ordenação do vetor, mais rápida será esta ordenação, dependendo do número de dados no vetor. Por exemplo com 2 threads e 100.000 dados conseguimos organizar o vetor mais rapidamente que com 4 threads, já se mudarmos o tamanho de elementos do vetor para 200.000, com 4 threads se torna mais rápido que com 2. Já com 16 threads conseguimos ser mais rápidos que qualquer outro número de threads testados, sem importar o número de dados. Isto tudo utilizando o Quick Sort e o número de threads de 2, 4, 8 ou 16.

Referencias Bibliográficas

1 - https://www.geeksforgeeks.org/quick-sort/
2 - https://pt.wikipedia.org/wiki/Quicksort
3 - https://www.youtube.com/watch?v=kUon6854joI&t=63s
4 - https://www.gnu.org/software/make/manual/html_node/Simple-Makefile.html
5 - https://www.youtube.com/watch?v=nVESQQg-Oiw&t=221s