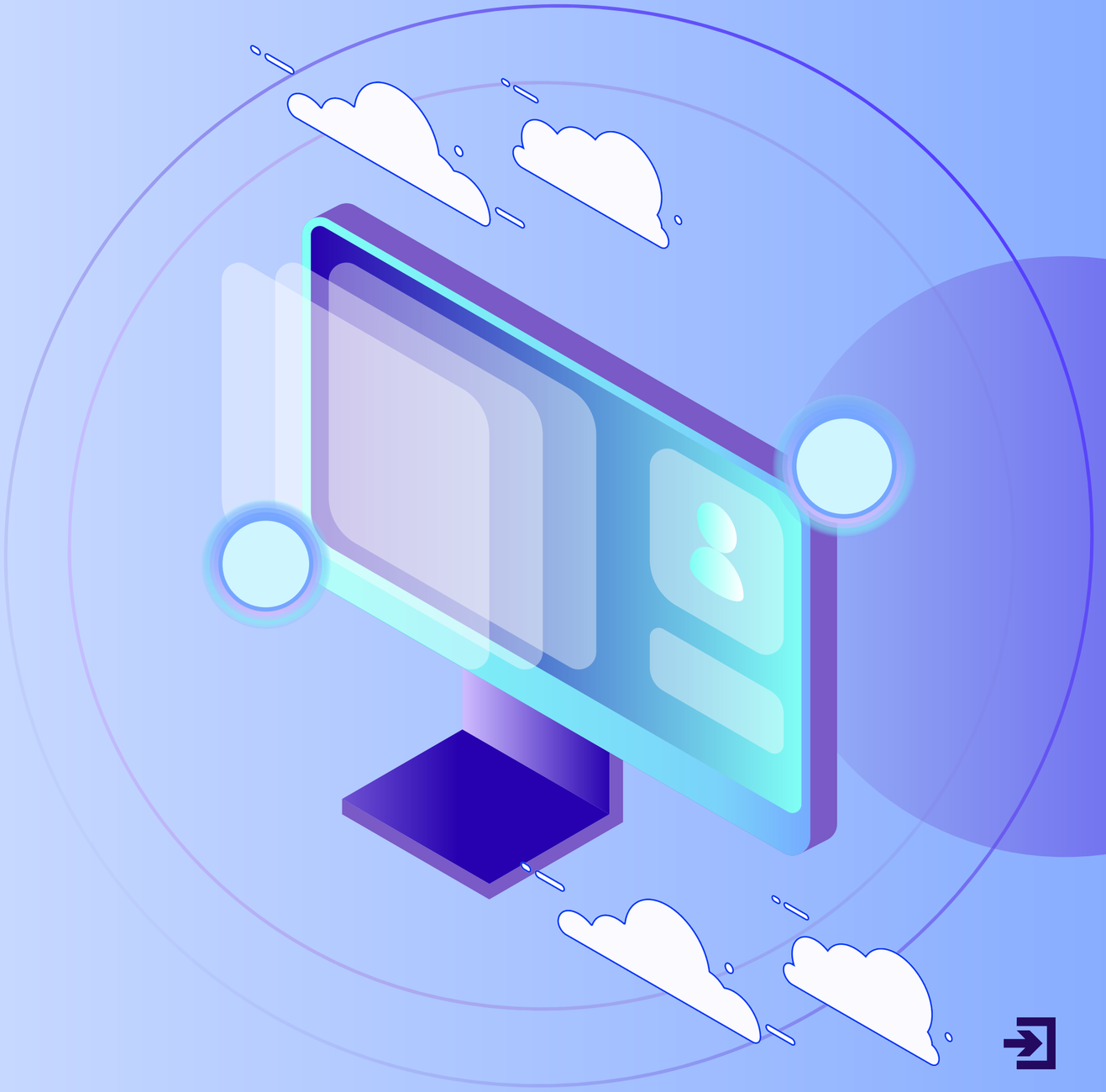
MERGE SORT THREADS



Universidade do Vale do Itajaí
Professor: Douglas Rossi de Melo
Aluno(a)s: Ana Clara Kniss, Erick Lemos Barreto, Mariana Ferreira, Melissa Moreira de Oliveira e Vinicius Mateus Colet

INTRODUÇÃO AO MERGE SORT

O **Merge Sort** é um algoritmo de ordenação eficiente, baseado na divisão e conquista. Ele divide o problema em subproblemas menores, resolvendo-os recursivamente e, por fim, combina as soluções para obter o resultado final.

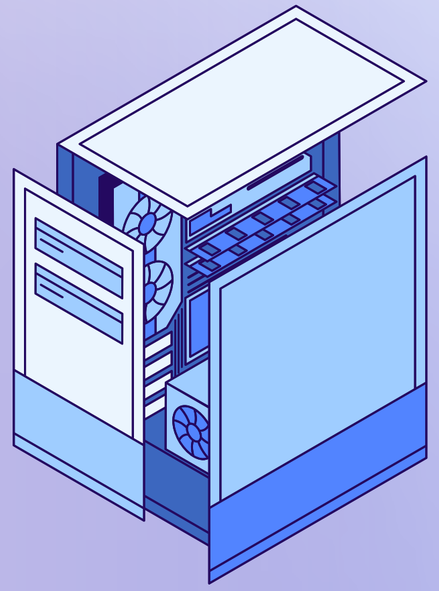
Apesar de sua eficiência com complexidade de tempo, sua implementação sequencial pode enfrentar limitações de desempenho em conjunto de dados muito grandes, principalmente em sistemas com múltiplos núcleos de processamento.

ESPECIFICAÇÃO DE SW E HW UTILIZADO

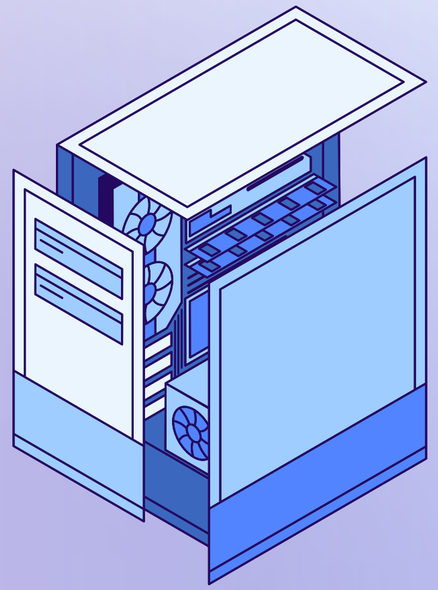
HARDWARE

- **Processador:** AMD Ryzen 7 3750H com Radeon Vega Mobile Graphics, com 8 núcleos e clock base de 2,3 GHz.

- **Memória RAM:** 32 GB, garantindo uma alta capacidade para multitarefa e execução de aplicações que demandam maior processamento.



ESPECIFICAÇÃO DE SW E HW UTILIZADO



HARDWARE

Placas de vídeo:

- **Integrada:** AMD Radeon(TM) RX Vega 10 Graphics, com 2 GB de memória dedicada e 15 GB de memória compartilhada.
- **Dedicada:** NVIDIA GeForce GTX 1650, com 4 GB de memória dedicada e 15 GB de memória compartilhada.

Display:

- Suporte para DirectX 12, porém sem habilitação para o DirectX 12 Ultimate.

ESPECIFICAÇÃO DE SW E HW UTILIZADO

SOFTWARE

- **Sistema Operacional:** Windows 11 Pro 64-bit (Build 22631), com configuração regional em português.

- **Versão do BIOS:** V1.10.

Driver da GPU AMD

- **Versão:** 27.20.1028.1
- **Data:** 13 de julho de 2020
- **Modelo do driver:** WDDM 2.7

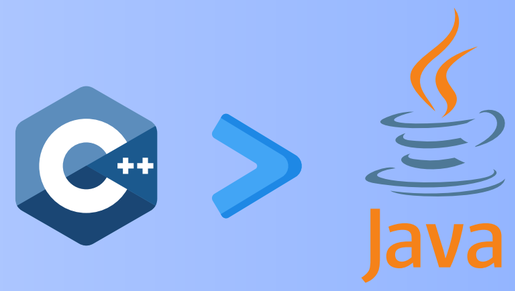
Driver da GPU NVIDIA

- **Versão:** 31.0.15.5222
- **Data:** 10 de abril de 2024
- **Modelo do driver:** WDDM 3.1

DETALHAMENTO DA LINGUAGEM E BIBLIOTECAS

Por que C++?

- **C++ é frequentemente escolhido para aplicações onde o desempenho é crucial, pois o código em C++ é compilado diretamente para linguagem de máquina, o que resulta em maior velocidade de execução comparado a linguagens como Python ou Java.**



DETALHAMENTO DA LINGUAGEM E BIBLIOTECAS

Algumas qualidades do C++:

- **Controle de recursos: C++ permite controle direto sobre alocação e gerenciamento de memória, dando a possibilidade de evitar um possível overhead de uma atuação automática.**
- **Multithreading nativo: C++ possui suporte para paralelismo, utilizando bibliotecas, o que permite maximizar o uso do hardware moderno que tem múltiplos núcleos.**
- **Portabilidade: C++ é bem adaptativo e portável, o que significa que ele pode ser executado em diversas plataformas.**

DETALHAMENTO DA LINGUAGEM E BIBLIOTECAS

Biblioteca Time.h:

Essa Biblioteca está nos oferecendo a função de medição de tempo no código.

O Clock(): Retorna o tempo de CPU usado pelo programa desde o início de sua execução. É usado para medir a performance de blocos de código.

A função Clock( ) retorna o tempo em ticks, que é a unidade específica do sistema.

Ao dividir CLOCKS_PER_SEC estamos apenas convertendo o valor para segundos.

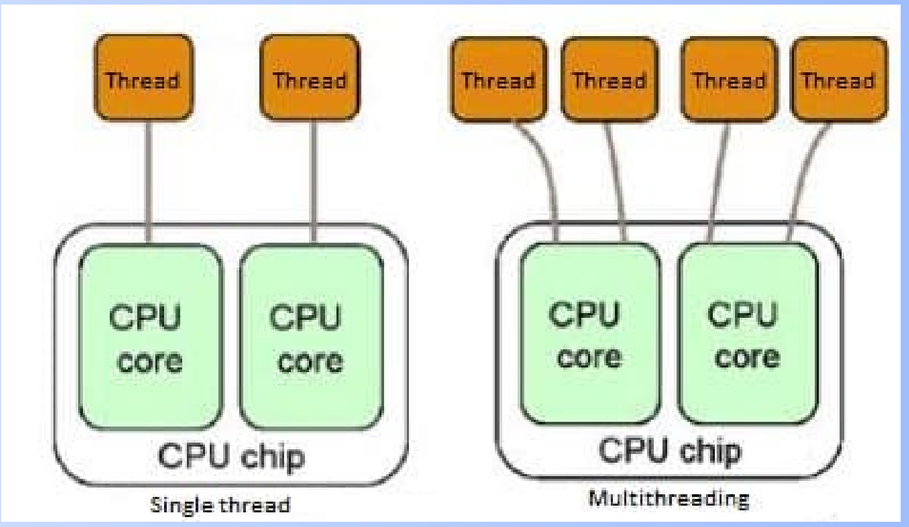
```
cout << "COM PARALELISMO" << endl;
double start_time = (double)clock() / CLOCKS_PER_SEC;
merge_sort_parallel(A, left: 0, right: n - 1);
double end_time = (double)clock() / CLOCKS_PER_SEC;
time_parallel = end_time - start_time;
cout << "TIME: " << time_parallel << " segundos" << endl;
```

DETALHAMENTO DA LINGUAGEM E BIBLIOTECAS

Biblioteca Thread:

Ela possibilita criar threads do SO, ao instanciar um thread, é criado um thread no nível de sistema operacional, que é gerenciado diretamente pelo Kernel.

Cada thread tem sua própria pilha de execução, mas compartilha o mesmo espaço de memória no processo principal



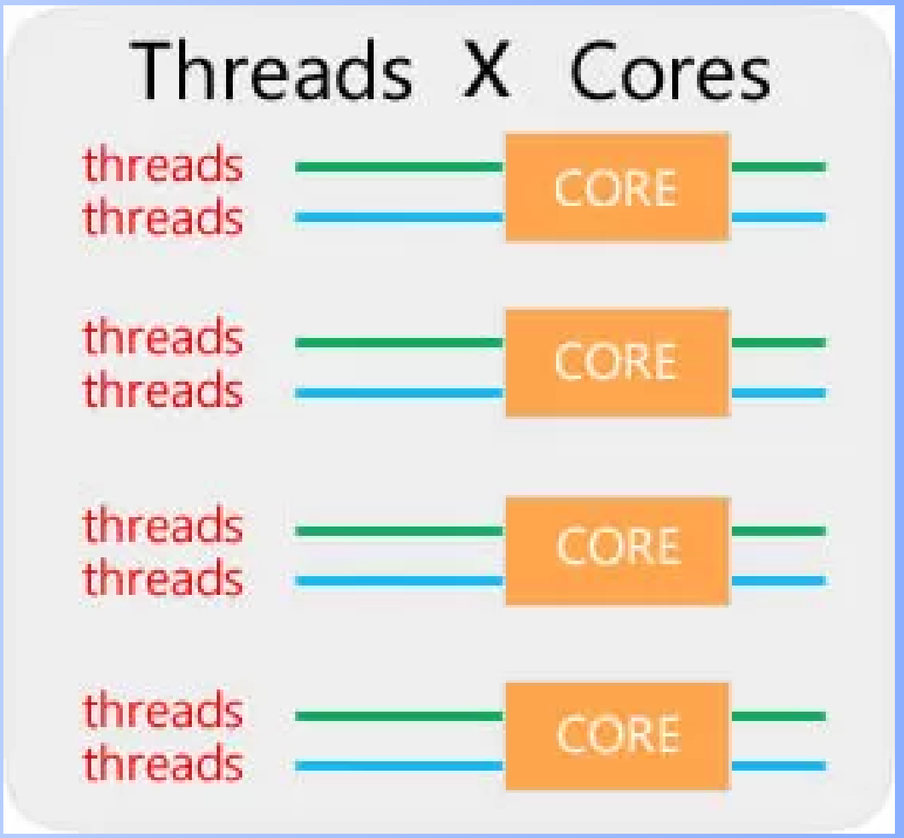
DETALHAMENTO DA LINGUAGEM E BIBLIOTECAS

Biblioteca Thread:

Com ela podemos gerenciar sincronizando threads com o Join() e também nos ajuda a explorar o máximo desempenho dos núcleos físicos e lógicos da CPU.

Ao utilziar tecnologias como o Hyper-Threading, um núcleo do processador pode executar dois threads lógicos ao mesmo tempo.

Após os threads terminarem as tarefas, os resultados podem ser combinados para resolver o problema original.

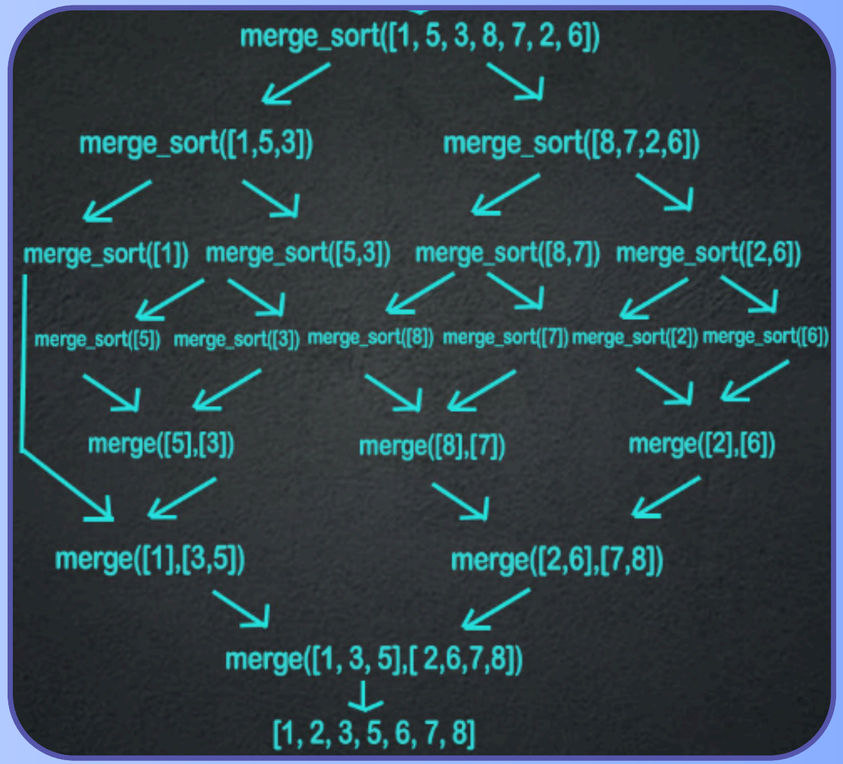


MERGE SORT

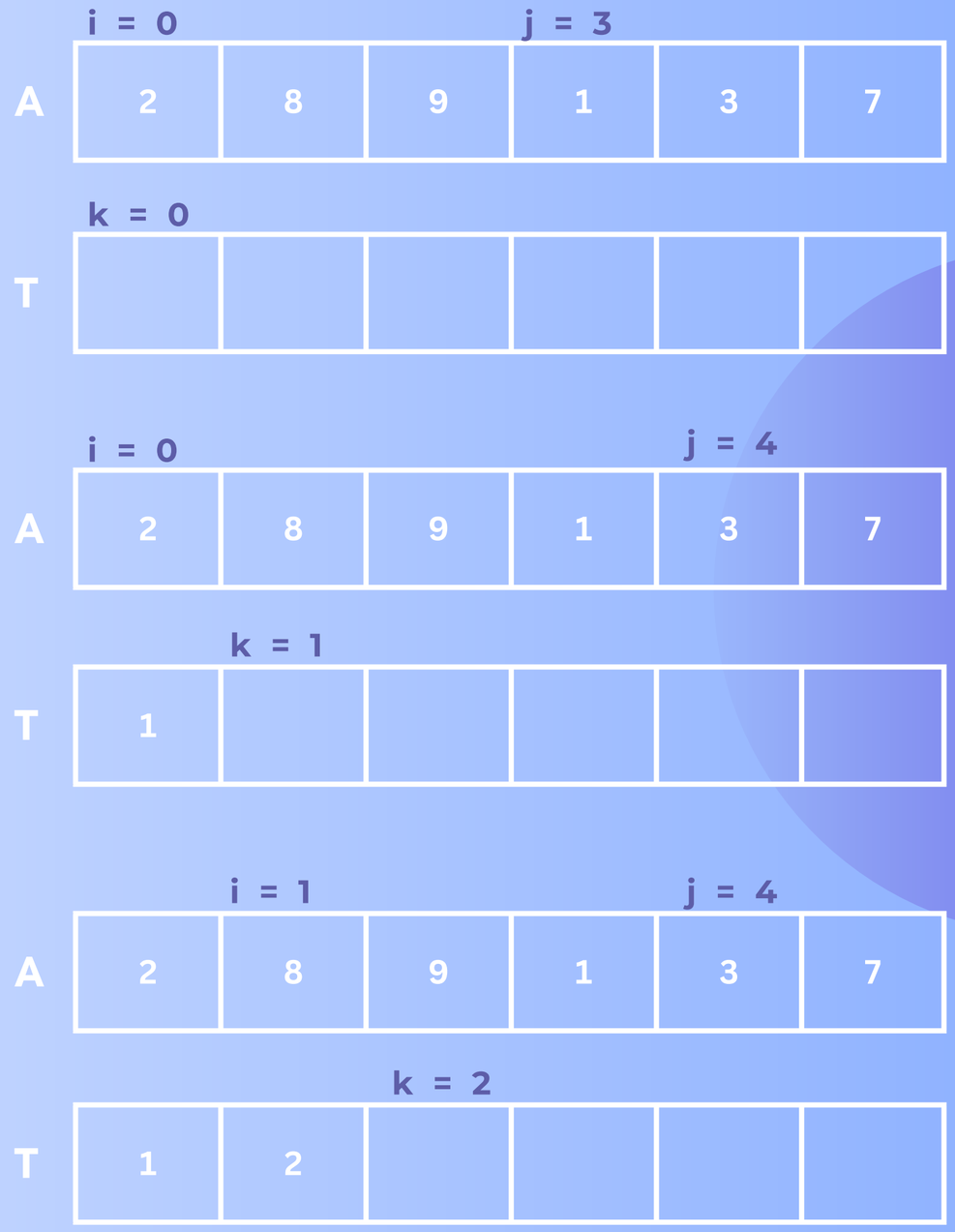
```
void merge_sort(int A[], int left, int right) {
    if (left < right) {
        int mid = (left + right) / 2;

        merge_sort(A, left, mid);
        merge_sort(A, mid + 1, right);

        merge(A, left, right, mid);
    }
}
```



```c
void merge(int A[], int left, int right, int mid) {
    int i, j, k, t[TMAX];
    i = left; k = left; j = mid + 1;
    while (i <= mid && j <= right) {
        if (A[i] < A[j]) {
            t[k] = A[i];
            i++;
        }
        else {
            t[k] = A[j];
            j++;
        }
        k++;
    }
    while (i <= mid) {
        t[k] = A[i];
        k++;
        i++;
    }
    while (j <= right) {
        t[k] = A[j];
        k++;
        j++;
    }
    for (i = left; i < k; i++) {
        A[i] = t[i];
    }
}
```



```
void merge_sort_parallel(int A[], int left, int right, int depth = 0) {

    if (left < right) {
        int mid = (left + right) / 2;

        if (depth <= 1) {
            thread t1(merge_sort_parallel, A, left, mid, depth + 1);
            thread t2(merge_sort_parallel, A, mid + 1, right, depth + 1);

            t1.join();
            t2.join();
        }
        else {
            merge_sort(A, left, mid);
            merge_sort(A, mid + 1, right);
        }


        merge(A, left, right, mid);

    }
}
```

```cpp
int main() {
    srand(Seed:time(NULL));
    int A[TMAX], B[TMAX], n = 80000;
    double time_parallel, time_non_parallel;

    for (int i = 0; i < n; i++) {
        A[i] = rand() % 100 + 1;
    }

    for (int i = 0; i < n; i++) {
        B[i] = A[i];
    }

    cout << "COM PARALELISMO" << endl;
    double start_time = (double)clock() / CLOCKS_PER_SEC;
    merge_sort_parallel(A, left:0, right:n - 1);
    double end_time = (double)clock() / CLOCKS_PER_SEC;
    time_parallel = end_time - start_time;
    cout << "TIME: " << time_parallel << " segundos" << endl;

    cout << "SEM PARALELISMO" << endl;
    start_time = (double)clock() / CLOCKS_PER_SEC;
    merge_sort(B, left:0, right:n - 1);
    end_time = (double)clock() / CLOCKS_PER_SEC;
    time_non_parallel = end_time - start_time;
    cout << "TIME: " << time_non_parallel << " segundos" << endl;

    cout << "Aceleracao: " << time_non_parallel / time_parallel;

    return 0;
}
```

MAIN

1. Inicializar as variáveis.
2. Preencher os vetores.
3. Chamar a função sort, e mostrar o tempo decorrido.
4. Mostrar a aceleração calculada com base no tempo decorrido com e sem paralelismo.

Análise de Desempenho e Aceleração Obtida

```
COM PARALELISMO
TIME: 0.01 segundos
SEM PARALELISMO
TIME: 0.035 segundos
Aceleracao: 3.5
```

```
COM PARALELISMO
TIME: 0.013 segundos
SEM PARALELISMO
TIME: 0.034 segundos
Aceleracao: 2.61538
```

A implementação com paralelismo foi de 61,76% a 71,4% mais eficiente em relação à sem paralelismo utilizando 100 mil posições no vetor

Referências

https://petbcc.ufscar.br/time/

https://en.cppreference.com/w/cpp/thread

https://www.tutorialspoint.com/cpp_standard_library/thread.htm

https://dicasdedevelop.wordpress.com/2015/06/15/a-utilizacao-de-threads-em-c/

https://www.servti.com/2018/01/31/entenda-a-diferenca-entre-nucleo-e-threads/

https://www.mobilebit.com.br/tecnologia/2020/12/15/threads-o-que-sao-para-que-servem-em-um-processador/

THANK YOU!