Trabalho – Ordenação de Números Inteiros em Paralelo

INE5410 - Programação Concorrente

2025/1

1 Definição

Você deverá desenvolver uma solução paralela em C para o problema da ordenação de números inteiros sem sinal (`unsigned int`). A solução deverá ser implementada com uso de POSIX threads (não é permitido o uso de OpenMP).

O programa deverá funcionar da seguinte forma:

- O programa deverá receber 4 parâmetros de entrada via linha de comando na seguinte ordem:
  - `input`: um arquivo contendo números inteiros sem sinal desordenados e separados por espaço;
  - `nnumbers`: a quantidade de números armazenados no arquivo de entrada;
  - `ntasks`: o número de tarefas a serem executadas; e
  - `nthreads`: o número de threads que executarão as tarefas em paralelo.
- O vetor de entrada sempre conterá valores entre 0 e `nnumbers`-1. Por exemplo, um vetor de entrada de tamanho 10 (`nnumbers=10`) poderá conter somente números dentro da faixa $[0, 10)$ enquanto um vetor de entrada de tamanho 30 (`nnumbers=30`) poderá conter somente números dentro da faixa $[0, 30)$;
- O programa deverá ler os números do arquivo e armazenar em um vetor na memória de tamanho `nnumbers` alocado dinamicamente (`malloc`);
- A versão paralela deverá ter 3 etapas, descritas a seguir:
  - **Etapa 1**: O vetor contendo os números desordenados deverá ser dividido em um número específico de tarefas (`ntasks`), identificadas de 0 até `ntasks`-1. Cada tarefa deverá conter números a serem ordenados pertencentes a uma determinada faixa de valores. O tamanho mínimo dos intervalos de números em cada tarefa deverá ser obtido da seguinte forma: `intervalo` = `nnumbers`/`ntasks`. Então, cada tarefa i deverá cobrir um intervalo $[i \cdot \text{intervalo},\ (i+1) \cdot \text{intervalo})$, onde i é a identificação da tarefa. Um tratamento especial deverá ser dado para os casos em que houver resto da divisão entre `nnumbers` e `ntasks`. Nestes casos, as faixas de números permitidos em cada tarefa deverão ter **uma diferença de tamanho de no**

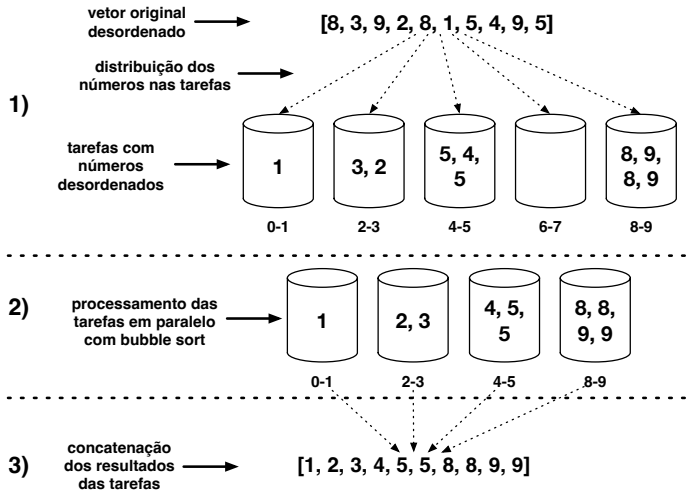

Figura 1: Exemplo com `nnumbers=10` elementos e `ntasks=5`.

máximo 1 número. Por exemplo, para `nnumbers=20` e `ntasks=8` teríamos $20/8 = 2$ com resto igual a 4. Nesse caso, 4 tarefas teriam intervalos de 3 elementos e 4 tarefas teriam intervalos de 2 elementos. Um exemplo de intervalos para esse caso seria: $[0, 3)$, $[3, 6)$, $[6, 9)$, $[9, 12)$, $[12, 14)$, $[14, 16)$, $[16, 18)$ e $[18, 20)$.

- **Etapa 2**: As threads deverão computar as tarefas existentes em paralelo. A computação de uma tarefa é a ordenação dos elementos presentes na tarefa usando-se o algoritmo `bubble_sort` fornecido;
- **Etapa 3**: Uma vez finalizada a computação das tarefas, o programa deverá concatenar os resultados ordenados em cada tarefa em um único vetor na memória. Esta concatenação fará com que o vetor final esteja ordenado.

No exemplo da Figura 1 considerou-se `nnumbers=10`. Logo, o vetor de entrada contém obrigatoriamente números entre $[0, 10)$. Estes números foram divididos em 5 tarefas (Etapa 1), cada um armazenando números em uma faixa contendo 2 valores possíveis: $[0, 2), [2, 4), ..., [8, 10)$. Note que uma das tarefas ficou vazia, pois não havia nenhum número pertencente ao intervalo $[6, 8)$. Logo, essa tarefa poderá ser simplesmente descartada na Etapa 2. Os números em cada tarefa foram ordenados em paralelo (Etapa 2). Por fim, na Etapa 3, os números ordenados em cada tarefa foram concatenados, gerando o resultado final.

O programa deverá funcionar em todos os casos, independentemente do tamanho do vetor, número de tarefas ou threads, exceto nos seguintes casos: quando o número de threads for menor que 1 ou quando o número de tarefas for maior que o tamanho do vetor.

1.1 Saída

Ao final da execução, o seu programa deverá mostrar **obrigatoriamente** uma saída no seguinte formato: vetor desordenado, atribuição de *threads/tarefas* (uma por linha) e vetor ordenado. Um exemplo de saída válido é mostrado abaixo com `nnumbers=100`, `ntasks=20` e `nthreads=4`:

```
10 48 39 84 3 45 20 48 47 73 4 19 31 45 30 71 10 72 84 10 64 22 20 11 63 17 11 84 73 2 21 8 24 50 54 29 87 53 33 86
68 88 1 26 38 15 45 96 30 13 17 62 60 42 76 75 92 74 73 11 86 79 92 75 18 0 52 46 34 88 58 50 95 7 29 36 77 82 57 71
23 55 12 14 21 0 50 78 55 45 83 51 95 62 50 71 11 56 37 46
Thread 0 processando tarefa 0
Thread 1 processando tarefa 1
Thread 0 processando tarefa 2
Thread 2 processando tarefa 3
Thread 3 processando tarefa 4
Thread 1 processando tarefa 5
Thread 0 processando tarefa 6
Thread 2 processando tarefa 7
Thread 3 processando tarefa 8
Thread 1 processando tarefa 9
Thread 0 processando tarefa 10
Thread 2 processando tarefa 11
Thread 3 processando tarefa 12
Thread 1 processando tarefa 13
Thread 0 processando tarefa 14
Thread 2 processando tarefa 15
Thread 3 processando tarefa 16
Thread 1 processando tarefa 17
Thread 0 processando tarefa 18
Thread 2 processando tarefa 19
0 0 1 2 3 4 7 8 10 10 10 11 11 11 11 12 13 14 15 17 17 18 19 20 20 21 21 22 23 24 26 29 29 30 30 31 33 34 36 37 38 39
42 45 45 45 45 46 46 47 48 48 50 50 50 50 51 52 53 54 55 55 56 57 58 60 62 62 63 64 68 71 71 71 72 73 73 73 74 75 75
76 77 78 79 82 83 84 84 84 86 86 87 88 88 92 92 95 95 96
```

2 Grupos, Avaliação e Entrega

O trabalho deverá ser realizado em grupos de **3 alunos**. Os alunos deverão apresentar o trabalho ao professor assim como mostrar sua solução em funcionamento. As apresentações serão feitas durante as aulas conforme cronograma da disciplina.

Pelo menos um dos integrantes de cada grupo deverá enviar através do Moodle um arquivo contendo o código fonte em C da solução para o trabalho. A data/hora limite para o envio dos trabalhos está estipulada no cronograma da disciplina.

O professor irá avaliar não somente a **corretude** mas também o **desempenho** e a **clareza** da solução. Durante a apresentação, o professor irá avaliar o **conhecimento individual dos alunos sobre os conteúdos teóricos e práticos vistos em aula e sobre a solução adotada no trabalho**. A nota atribuída à cada aluno i no trabalho ($NotaTrabalho_i$) será calculada da seguinte forma, onde A_i é a nota referente à apresentação do aluno i e S é a nota atribuída à solução do trabalho:

$$NotaTrabalho_i = \frac{A_i * S}{10} \tag{1}$$

ATENÇÃO: como indicado pela fórmula mostrada acima, **a nota atribuída à solução adotada será ponderada pelo desempenho do aluno durante a apresentação do trabalho**. Por exemplo, se o professor atribuir nota 10 para a solução adotada pelo grupo mas o aluno receber nota 5 pela apresentação – devido ao desconhecimento dos conteúdos teóricos, práticos e/ou da solução do trabalho – a sua nota final do trabalho será 5. A ausência no dia da apresentação ou recusa de realização da apresentação do trabalho implicará em nota zero na apresentação, fazendo com que a nota atribuída ao aluno também seja zero.

3 Bubble Sort

Os grupos deverão utilizar a implementação do algoritmo `bubble_sort` mostrada a seguir para ordenar os elementos dentro dos tarefas. Os parâmetros `v` e `tam` correspondem ao vetor a ser ordenado e o seu tamanho, respectivamente.

```
1  void bubble_sort(int *v, int tam){
2      int i, j, temp, trocou;
3      for(j = 0; j < tam - 1; j++){
4          trocou = 0;
5          for(i = 0; i < tam - 1; i++){
6              if(v[i + 1] < v[i]){
7                  temp = v[i];
8                  v[i] = v[i + 1];
9                  v[i + 1] = temp;
10                 trocou = 1;
11             }
12         }
13         if(!trocou) break;
14     }
15 }
```