Ordenação com MPI: Bubble Sort Distribuído

Luis Vinicius

5 de agosto de 2025

Bubble Sort Tradicional em C

Objetivo: Ordenar um vetor comparando pares adjacentes e "empurrando"o maior para o fim a cada passagem.

Algoritmo:

- Dois laços `for`: o externo controla o número de passagens.
- O laço interno compara e troca elementos adjacentes.
- Após cada iteração do laço externo, o maior valor restante "borbulha"para o final.

Complexidade: $\mathcal{O}(n^2)$ no pior caso.

Aplicação: Simples, porém ineficiente para grandes volumes de dados — útil para introdução à lógica de ordenação.

Teste de Mesa: Bubble Sort Tradicional

Vetor inicial: `{5, 3, 8, 4, 2}`

Passo a passo:

- **1ª passagem:** compara e troca
  - $5 > 3 \rightarrow$ troca $\rightarrow$ `{3, 5, 8, 4, 2}`
  - $5 < 8 \rightarrow$ mantém
  - $8 > 4 \rightarrow$ troca $\rightarrow$ `{3, 5, 4, 8, 2}`
  - $8 > 2 \rightarrow$ troca $\rightarrow$ `{3, 5, 4, 2, 8}`
- **2ª passagem:**
  - $3 < 5 \rightarrow$ mantém
  - $5 > 4 \rightarrow$ troca $\rightarrow$ `{3, 4, 5, 2, 8}`
  - $5 > 2 \rightarrow$ troca $\rightarrow$ `{3, 4, 2, 5, 8}`
- **3ª passagem:**
  - $3 < 4 \rightarrow$ mantém
  - $4 > 2 \rightarrow$ troca $\rightarrow$ `{3, 2, 4, 5, 8}`
- **4ª passagem:**
  - $3 > 2 \rightarrow$ troca $\rightarrow$ `{2, 3, 4, 5, 8}`

Resultado final: `{2, 3, 4, 5, 8}`

Bubble Sort com MPI: Explicação + Teste de Mesa

Objetivo: Ordenar um vetor distribuindo partes entre processos MPI.

Etapas do código:

1. **Inicialização:** MPI inicia, cada processo descobre seu `rank`.
2. **Divisão:** Processo 0 gera vetor e distribui com `MPI_Scatter`.
3. **Ordenação local:** Cada processo aplica Bubble Sort em sua parte.
4. **Coleta:** Partes ordenadas retornam com `MPI_Gather`.
5. **Ordenação final:** Processo 0 faz Bubble Sort no vetor completo.

Teste de Mesa (4 processos, vetor com 8 elementos):

- Vetor original (gerado no processo 0): `{8, 3, 7, 4, 2, 9, 1, 5}`
- Divisão por processo:
  - P0: `8, 3` $\rightarrow$ `3, 8`
  - P1: `7, 4` $\rightarrow$ `4, 7`
  - P2: `2, 9` $\rightarrow$ `2, 9`
  - P3: `1, 5` $\rightarrow$ `1, 5`
- Reunião em P0: `{3, 8, 4, 7, 2, 9, 1, 5}`
- Ordenação final em P0: `{1, 2, 3, 4, 5, 7, 8, 9}`

Explicação do Código: Bubble Sort com MPI

Objetivo: Ordenar um vetor usando múltiplos processos com MPI.

Etapas principais do código:

1. **Inicialização do MPI:**
   - Cada processo descobre seu `rank` e o total de processos com `MPI_Comm_rank` e `MPI_Comm_size`.
2. **Distribuição do vetor:**
   - O processo 0 cria um vetor com números inteiros.
   - O vetor é dividido igualmente entre os processos usando `MPI_Scatter`.
3. **Ordenação local:**
   - Cada processo aplica Bubble Sort na sua parte do vetor.
4. **Coleta das partes ordenadas:**
   - As partes locais são enviadas de volta ao processo 0 com `MPI_Gather`.
5. **Ordenação final (processo 0):**
   - O processo 0 aplica Bubble Sort no vetor completo para garantir ordenação global.

Observação: A ordenação local é paralela, mas a fusão final ainda é

Bubble Sort com MPI: Explicação + Teste de Mesa

Objetivo: Ordenar um vetor distribuindo partes entre processos MPI.

Etapas do código:

1. **Inicialização:** MPI inicia, cada processo descobre seu `rank`.
2. **Divisão:** Processo 0 gera vetor e distribui com `MPI_Scatter`.
3. **Ordenação local:** Cada processo aplica Bubble Sort em sua parte.
4. **Coleta:** Partes ordenadas retornam com `MPI_Gather`.
5. **Ordenação final:** Processo 0 faz Bubble Sort no vetor completo.

Teste de Mesa (4 processos, vetor com 8 elementos):

- Vetor original (gerado no processo 0): `{8, 3, 7, 4, 2, 9, 1, 5}`
- Divisão por processo:
  - P0: `8, 3` $\rightarrow$ `3, 8`
  - P1: `7, 4` $\rightarrow$ `4, 7`
  - P2: `2, 9` $\rightarrow$ `2, 9`
  - P3: `1, 5` $\rightarrow$ `1, 5`
- Reunião em P0: `{3, 8, 4, 7, 2, 9, 1, 5}`
- Ordenação final em P0: `{1, 2, 3, 4, 5, 7, 8, 9}`

Atividade Prática: Bubble Sort Paralelo com MPI

Objetivo: Experimentar paralelismo com MPI no algoritmo Bubble Sort e analisar desempenho.

Tarefas:

1. Compile e execute o código Bubble Sort com MPI fornecido.
2. Modifique o vetor para os tamanhos: 32, 64, 128 e 256 elementos.
3. Execute com 2, 4 e 8 processos, e registre o tempo de execução.
4. Faça cada processo imprimir seu vetor local antes e depois da ordenação.
5. (Opcional) Implemente uma fusão (merge) no processo 0 para substituir a ordenação final.
6. Responda:
   - Qual o impacto do número de processos no tempo?
   - Em quais situações houve ganho real?
   - Quais limitações você identificou?
   - Como melhorar a eficiência do algoritmo?

Dica: Use `MPI_Wtime()` para medir tempo.