Relatório Técnico

Theo Torminn Neto Nasser - 2022.1904.025-6

1. Introdução ao problema e justificativa do uso do Odd-Even Sort.

Este projeto tem como objetivo ordenar um grande volume de dados de sensores utilizando o algoritmo **Odd-Even Sort**, implementado em ambiente de memória distribuída com a biblioteca **MPI**, para ordenar grandes volumes de dados e compará-lo com a implementação sequencial do Odd-Even Sort, e com o algoritmo Quicksort, também na forma sequencial, visando avaliar ganhos de desempenho com a abordagem paralela..

A escolha pelo Odd-Even Sort se dá por sua eficiência e paralelizabilidade. Apesar de não ser o algoritmo de ordenação mais eficiente em termos de complexidade assintótica, ele possui um comportamento ideal para implementação em arquiteturas distribuídas usando MPI.

---

2. Descrição Teórica do Algoritmo Odd-Even Sort

2.1 Funcionamento Sequencial

O Odd-Even Sort é um algoritmo baseado em comparações de pares adjacentes. Em cada iteração, ele realiza duas fases:

1. Fase ímpar **:** compara e, se necessário, troca os elementos de índices ímpares com seus vizinhos da direita: (1,2), (3,4), (5,6), ...
2. Fase par **:** compara e, se necessário, troca os elementos de índices pares com seus vizinhos da direita: (0,1), (2,3), (4,5), ...

Essas duas fases são executadas **n** vezes para ordenar todo o vetor.

2.2 Estratégia de Paralelização com MPI

No contexto de MPI, o vetor global é dividido entre os processos, e cada processo ordena localmente seus dados com `qsort`. O Odd-Even Sort é aplicado de forma distribuída com troca de dados entre processos vizinhos, simulando os pares do algoritmo sequencial:

- Em cada fase, os processos trocam dados com seus vizinhos pares ou ímpares, dependendo da iteração.
- Cada processo realiza uma junção (merge) parcial-ordenada dos dados trocados com o vizinho, mantendo os menores ou maiores valores conforme sua posição.

A cada fase, todos os processos se sincronizam com uma barreira (`MPI_Barrier`), garantindo que as trocas estejam concluídas antes da próxima iteração.

---

3. Metodologia de Implementação

- A leitura dos dados do arquivo é feita apenas pelo processo 0, que armazena todos os registros lidos em um vetor.
- O total de registros é dividido de forma equilibrada entre os processos usando `MPI_Scatterv`. Quando o número total de registros não é divisível exatamente pelo número de processos, os primeiros processos recebem um elemento adicional.
- Para permitir o envio direto da `struct Sensor`, foi definido um tipo MPI personalizado (`MPI_Sensor`) usando o `MPI_Type_create_struct`.
- Em cada fase do Odd-Even Sort paralelo:
    - Um processo se comunica com o seu vizinho par ou ímpar usando `MPI_Sendrecv`, trocando seus dados ordenados.
    - Após a troca, cada processo executa um **merge** ordenado dos seus dados com os do vizinho, mantendo os menores valores se for o processo com rank menor.
- Todos os processos executam `MPI_Barrier` após cada fase, garantindo a sincronização global.
- O processo é repetido por número de fases igual ao número de processos, assegurando que todos os elementos atinjam suas posições finais ordenadas.

---

4. Resultados experimentais

A implementação do Odd-Even Sort com MPI demonstrou como algoritmos simples podem ser eficientemente paralelizados para lidar com grandes volumes de dados. A escolha do MPI permitiu uma abordagem escalável, facilitando a comunicação e sincronização entre os processos. O projeto alcançou o objetivo proposto, com sucesso na ordenação dos dados de sensores.

1. 100k de dados de entrada:

Tempo de execução do OddEven Sequencial: 64.407598 segundos

Tempo de execução do Quicksort Sequencial: 0.030725 segundos

1.1 Execução com 2 processos:

Tempo de execução do OddEven Paralelo: 0.013210 segundos

1.2 Execução com 4 processos:

Tempo de execução do OddEven Paralelo: 0.024872 segundos

1.3 Execução com 6 processos:

Tempo de execução do OddEven Paralelo: 0.033495 segundos

1.4 Execução com 8 processos:

Tempo de execução do OddEven Paralelo: 0.038246 segundos

1.5 Execução com 12 processos:

Tempo de execução do OddEven Paralelo: 0.066968 segundos

---

2. 1M de dados de entrada:

Tempo de execução do OddEven Sequencial =~ 6.4 . 10^2 segundos

Tempo de execução do Quicksort Sequencial: 0.347174 segundos

2.1 Execução com 2 processos:

Tempo de execução do OddEven Paralelo: 0.125101 segundos

2.2 Execução com 4 processos:

Tempo de execução do OddEven Paralelo: 0.296419 segundos

2.3 Execução com 6 processos:

Tempo de execução do OddEven Paralelo: 0.326709 segundos

2.4 Execução com 8 processos:

Tempo de execução do OddEven Paralelo: 0.378520 segundos

2.5 Execução com 12 processos:

Tempo de execução do OddEven Paralelo: 0.530382 segundos

---

3. 10M de dados de entrada:

Tempo de execução do OddEven Sequencial =~ 6.4 . 10^4 segundos

Tempo de execução do Quicksort Sequencial: 5.190561 segundos

3.1 Execução com 2 processos:

Tempo de execução do OddEven Paralelo: 1.238886 segundos

3.2 Execução com 4 processos:

Tempo de execução do OddEven Paralelo: 2.643378 segundos

3.3 Execução com 6 processos:

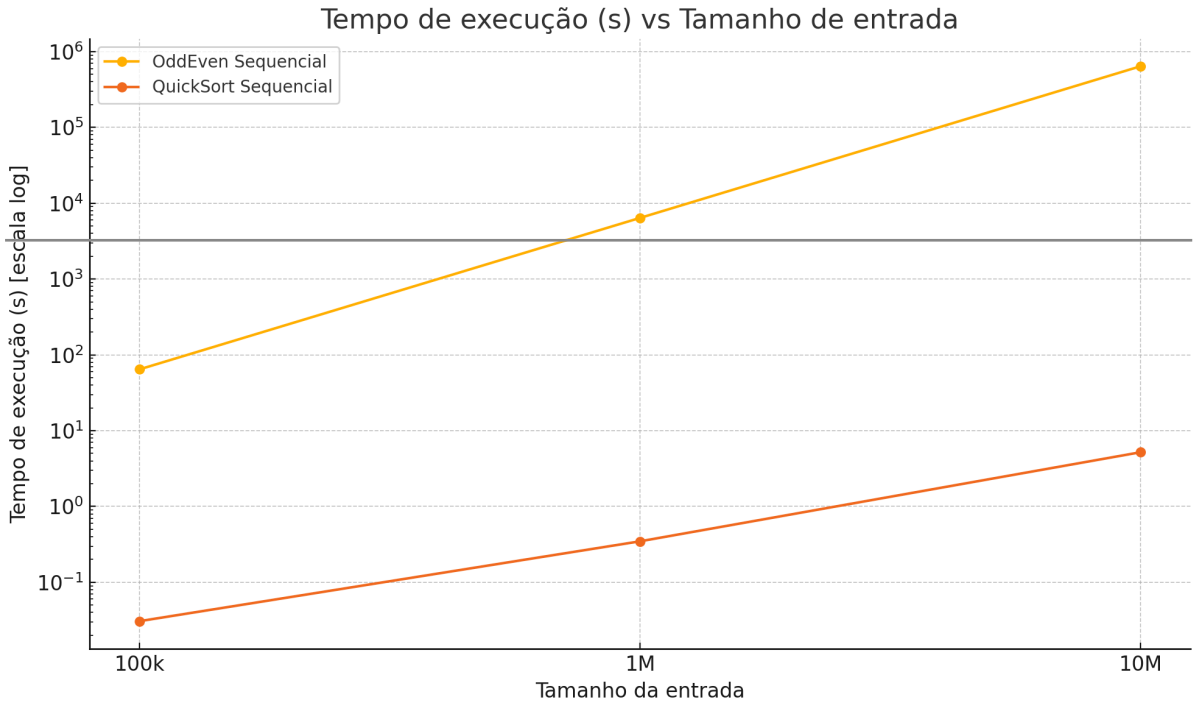
Tempo de execução do OddEven Paralelo: 3.227681 segundos

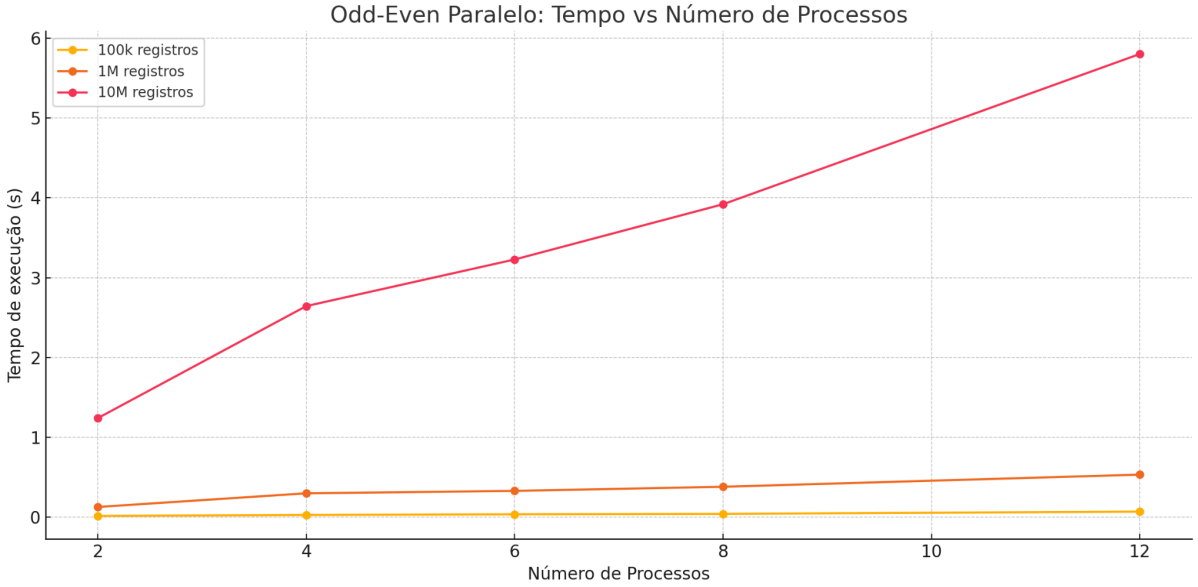
3.4 Execução com 8 processos:

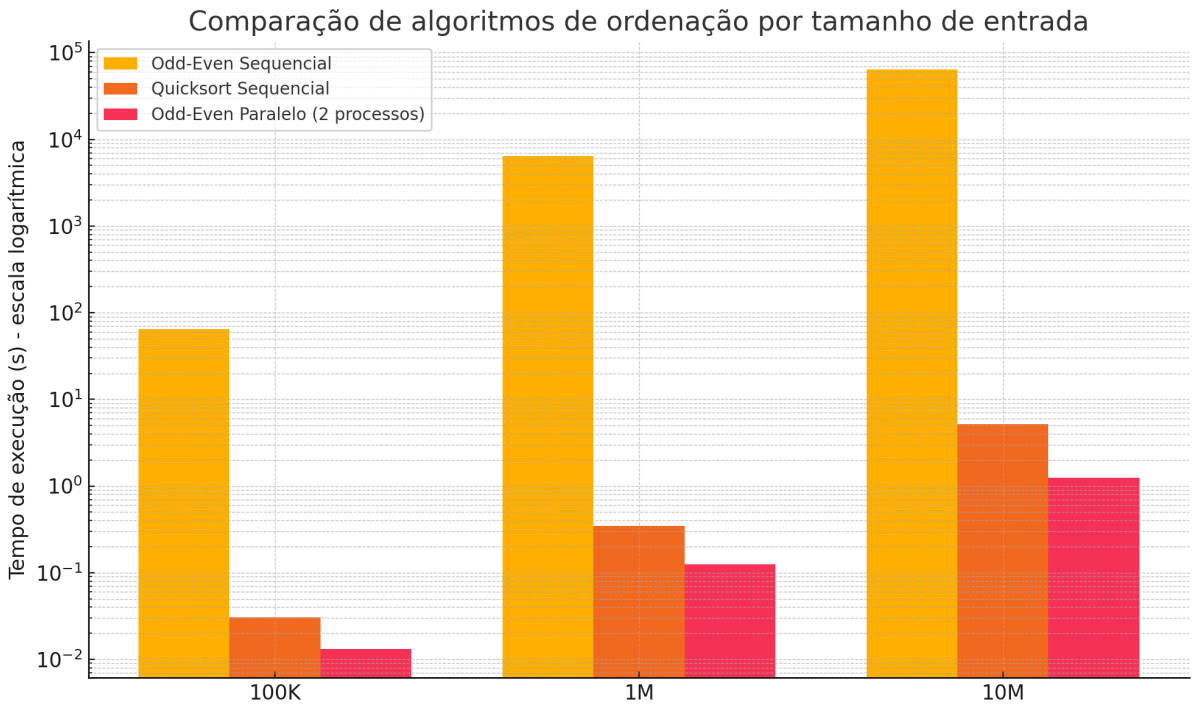
Tempo de execução do OddEven Paralelo: 3.920062 segundos

3.5 Execução com 12 processos:

Tempo de execução do OddEven Paralelo: 5.803620 segundos







5. Análise crítica dos resultados

1.Ganhos de Performance Observados

Quicksort Sequencial

- Apresenta desempenho excelente, mesmo com grandes volumes de dados.
- Para 10 milhões de registros, executou em aproximadamente 5,19 segundos, contra cerca de 640 segundos do Odd-Even Sort sequencial.
- A complexidade O(n log n) justifica sua escalabilidade e eficiência.

Odd-Even Sort Paralelo

- Apresenta ganhos significativos em relação à versão sequencial, especialmente com 2 a 4 processos.
- Para 10 milhões de registros, a versão paralela com 2 processos executou em cerca de 1,23 segundos (contra 640 segundos da versão sequencial).
- O paralelismo ajuda a compensar a ineficiência do algoritmo, até certo ponto.

2. Limitações do Método

Odd-Even Sort (Sequencial e Paralelo)

- Complexidade quadrática $O(n^2)$, tornando-o ineficiente para grandes volumes de dados.
- A versão paralela atinge rapidamente um ponto de saturação: aumentar o número de processos nem sempre resulta em melhor desempenho.

- Para 6, 8 e 12 processos, o tempo de execução do algoritmo paralelo voltou a crescer.

Overhead de Comunicação no Odd-Even Paralelo

- O custo de troca de mensagens e sincronização entre os processos é elevado.
- Em alguns casos, o tempo de comunicação supera os ganhos computacionais, tornando o paralelismo contraproducente.

3. Discussão sobre Escalabilidade

Quicksort

- Escala muito bem conforme o tamanho da entrada aumenta.
- Mesmo na versão sequencial, é eficiente para milhões de registros.

Odd-Even Sort Paralelo

- Ao aumentar os processos seu tempo de execução se eleva também.
- Melhor desempenho dentre as outras.

Conclusão (sugestão)

A implementação do Odd-Even Sort com MPI demonstrou como algoritmos simples podem ser eficientemente paralelizados para lidar com grandes volumes de dados. A escolha do MPI permitiu uma abordagem escalável, facilitando a comunicação e sincronização entre os processos. O projeto alcançou o objetivo proposto, com sucesso na ordenação dos dados de sensores.