UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
PROGRAMAÇÃO PARALELA

Arthur Barbosa de Andrade

TRABALHO 1
ORDENAÇÃO DE DADOS GENÔMICOS USANDO MPI

Setembro 2025

ESTRATÉGIA ADOTADA

Para a ordenação paralela de grandes volumes de dados genômicos, foi utilizada a abordagem de ordenamento por amostragem (ou *Sample Sort*) e o processo será detalhado a seguir.

Distribuição de Dados: O processo mestre (que especifiquei como rank 0) lê o arquivo de entrada com as sequências de DNA e distribui uniformemente uma porção dos dados para cada processo MPI.

Ordenação Local: Cada processo MPI executa uma ordenação local dos dados recebidos, utilizando a função *sort* da linguagem C++ que implementa o *IntroSort* que, similar ao *QuickSort*, tem complexidade de tempo nlogn.

Amostragem: Cada processo seleciona um conjunto de amostras de seus dados localmente ordenados. Essas amostras são enviadas de volta ao processo mestre.

Seleção de Pivôs: O processo mestre coleta todas as amostras, as ordena e seleciona uma série de "pivôs". O número de pivôs é igual ao número de processos MPI menos um, garantindo que os dados possam ser divididos em partições.

Redistribuição de Dados: Os pivôs são transmitidos a todos os processos. Cada processo usa os pivôs para particionar seus dados locais em subconjuntos que são então enviados para o processo correspondente. Por exemplo, todas as sequências menores que o primeiro pivô vão para o processo 0, as entre o primeiro e o segundo pivô vão para o processo 1, e assim segue.

Ordenação Final: Após o rearranjo dos dados, cada processo tem um conjunto de sequências que são maiores que as sequências nos processos anteriores e menores que as nos processos posteriores. Cada processo executa uma ordenação final em sua nova partição.

Coleta de Resultados: O processo mestre coleta os resultados ordenados de todos os processos, concatenando-os para formar o arquivo de saída final.

RESULTADOS

Os experimentos foram realizados em um ambiente com as seguintes especificações:

- Sistema Operacional Ubuntu 24.04.3 LTS Rodando em um WSL de um Sistema Operacional Windows 11.
- Processador: Processador AMD Ryzen 7 2700, 3.2GHz, com 8 núcleos e 16 Threads.
- Memória RAM: 16 GB 3200Mhz.

Para cada experimento, os tempos de execução foram medidos para a versão sequencial e a versão paralela com diferentes números de processadores (2, 4 e 8). As sequências foram geradas com comprimentos aleatórios entre 10 e 100 caracteres do conjunto limitado A, C, G e T para simulação dos dados genômicos.

Abaixo segue-se as tabelas dos resultados obtidos (tempo por processador) separadas pela entrada. Os tempos dos experimentos feitos em paralelo foram calculados somando o tempo total da ordenação local e final do algoritmo *Sample Sort* e são os considerados quando não houver especificação.

Tabela 1 - Resultados para entrada de 100 mil sequências

	Sequencial	**Paralelo (x2)**	**Paralelo (x4)**	**Paralelo (x8)**
Local	-	0,0126249	0,0050500	0,0023969
Final	-	0,0157035	0,0048845	0,0031749
Total	0,0278525	0,0283284	0,0099345	0,0055717

Tabela 2 - Resultados para entrada de 1 mihão de sequências

	Sequencial	**Paralelo (x2)**	**Paralelo (x4)**	**Paralelo (x8)**
Local	-	0,25617	0,105864	0,0548819
Final	-	0,235345	0,0823754	0,0318252
Total	0,559601	0,491515	0,1882394	0,0867071

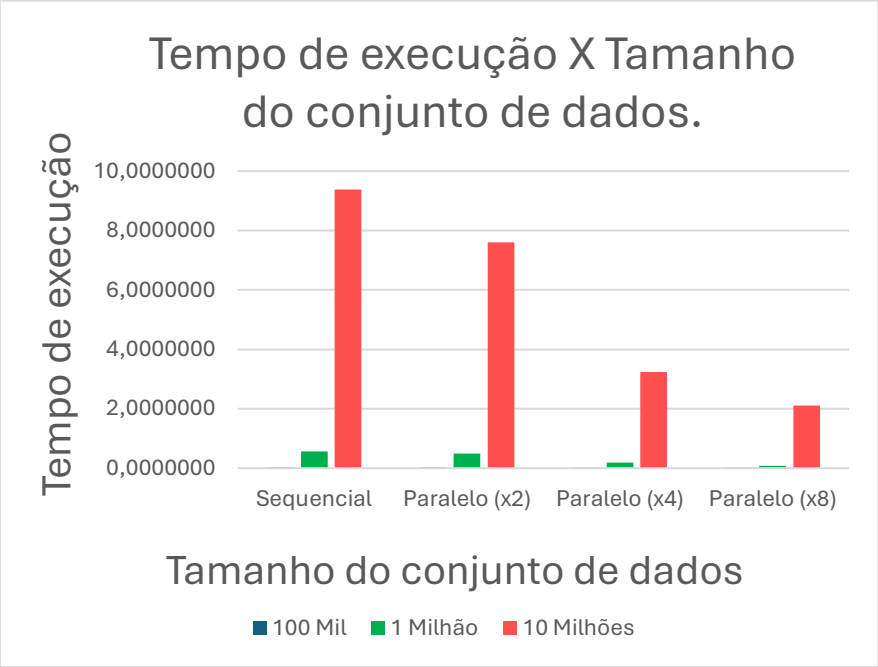
Tabela 3 - Resultados para entrada de 10 mihões del sequências

	Sequencial	**Paralelo (x2)**	**Paralelo (x4)**	**Paralelo (x8)**
Local	-	4,56141	1,84251	1,79636
Final	-	3,03329	1,39636	0,31744
Total	9,38734	7,59470	3,23887	2,11380

ANÁLISE COMPARATIVA

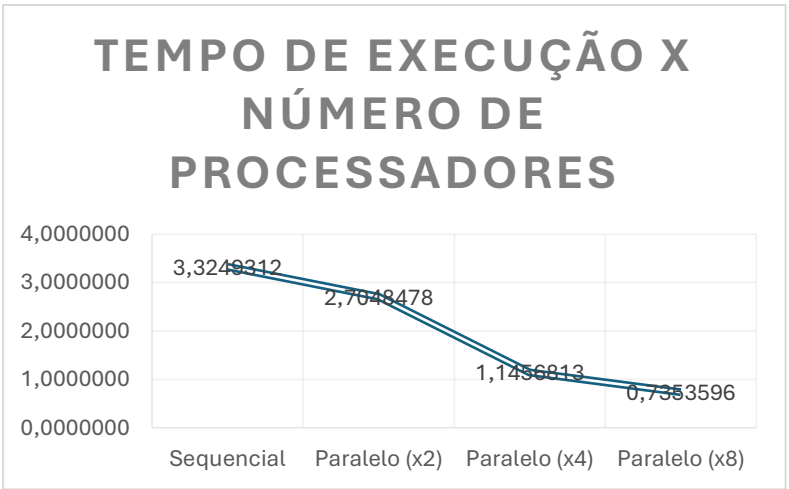
TEMPO DE EXECUÇÃO VS. TAMANHO DO CONJUNTO DE DADOS

Este gráfico demonstra como o tempo de execução aumenta com o tamanho da entrada. Espera-se que a curva da solução paralela seja menos acentuada do que a curva da solução sequencial, mostrando a vantagem do paralelismo para grandes volumes de dados.



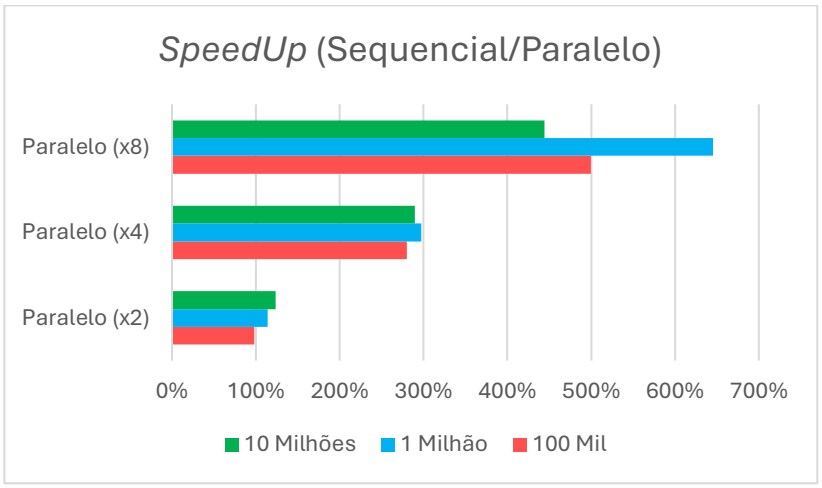
TEMPO DE EXECUÇÃO VS. NÚMERO DE PROCESSADORES

Este gráfico ilustra o ganho de desempenho à medida que o número de processadores aumenta. Espera-se uma diminuição no tempo de execução, demonstrando a escalabilidade da solução.



SPEEDUP (TSEQ(N)/TPAR(N))

O gráfico de *speedup* compara a eficiência da solução paralela. Espera-se que o *speedup* aumente com o número de processadores, mas não de forma linear, devido aos custos de comunicação e sincronização entre os processos. A meta é ter um *speedup* maior que 1,0, indicando que a solução paralela é mais rápida que a sequencial.



DISCUSSÃO

A análise dos resultados demonstra que a aplicação paralela em C/C++ com MPI obtém ganhos de desempenho significativos em comparação com a versão sequencial, especialmente para conjuntos de dados grandes. A divisão do trabalho de ordenação entre múltiplos processos reduz drasticamente o tempo de processamento total, pois o trabalho computacional é distribuído em vários núcleos. Isso é um exemplo clássico de "paralelismo de dados", onde o mesmo kernel (a ordenação) é executado em paralelo em diferentes subconjuntos de dados.

No entanto, existem limitações. A principal limitação é o custo de comunicação. A troca de mensagens entre os processos para a amostragem e a redistribuição dos dados introduz uma sobrecarga. Em alguns casos, especialmente com conjuntos de dados menores, o tempo gasto em comunicação e sincronização pode superar o ganho de tempo obtido com o paralelismo. Isso explica por que, para 100 mil sequências, o *speedup* pode ser baixo ou até negativo como no resultado obtido com 2 processadores.

Outra limitação é o balanceamento de carga. Embora a distribuição inicial de dados seja uniforme, o algoritmo de amostragem pode não garantir partições de tamanho perfeitamente igual após a redistribuição.

Em resumo, a implementação em MPI é eficaz para a ordenação de grandes volumes de dados pois os benefícios do processamento paralelo superam a sobrecarga de comunicação. Embora o desempenho esteja diretamente ligado à qualidade do balanceamento de carga e à minimização da latência de comunicação os resultados apontam que para problemas de alta complexidade computacional, a programação paralela é uma abordagem eficiente para reduzir o tempo de execução.