Discentes: Gabriel Meireles Dias Narciso e Thayná Andrade Gimenez
RGAs: 2022.1904.038-8 e 2023.1904.038-0
Docente: Marco Aurélio Stefanes

Relatório Técnico

1. Algoritmo de Ordenação Paralela

O algoritmo escolhido para ser desenvolvido paralelamente foi o Sample Sort. Um algoritmo de ordenação baseado em *Divisão e Conquista*, que distribui as n sequências em p processadores de tamanho n/p, ordenando-as localmente em seus respectivos blocos. Em cada bloco é escolhido $m - 1$ elementos igualmente espaçados, totalizando $m(m - 1)$ elementos selecionados de todos os blocos. Após essa escolha, os elementos selecionados são ordenados localmente para que sejam escolhidos $p - 1$ separadores locais, também chamados de splitters, que são compartilhados através do broadcast all-to-all.

Com os separadores definidos, realiza-se a separação por intervalos com base em todos os elementos, ordenando o arranjo global.

2. Implementação e Distribuição dos DADOS

A implementação desenvolvida utiliza o algoritmo Sample Sort para ordenação distribuída de sequências de DNA. A estratégia adotada para distribuição dos dados baseia-se na função distribuir_sequencias, que emprega a operação MPI_Scatterv do MPI para distribuir as sequências de forma balanceada entre todos os processos participantes. Esta abordagem garante que cada processo receba aproximadamente o mesmo número de sequências, com os primeiros processos recebendo uma sequência adicional quando a divisão não é exata, assegurando assim um balanceamento inicial adequado da carga de trabalho.

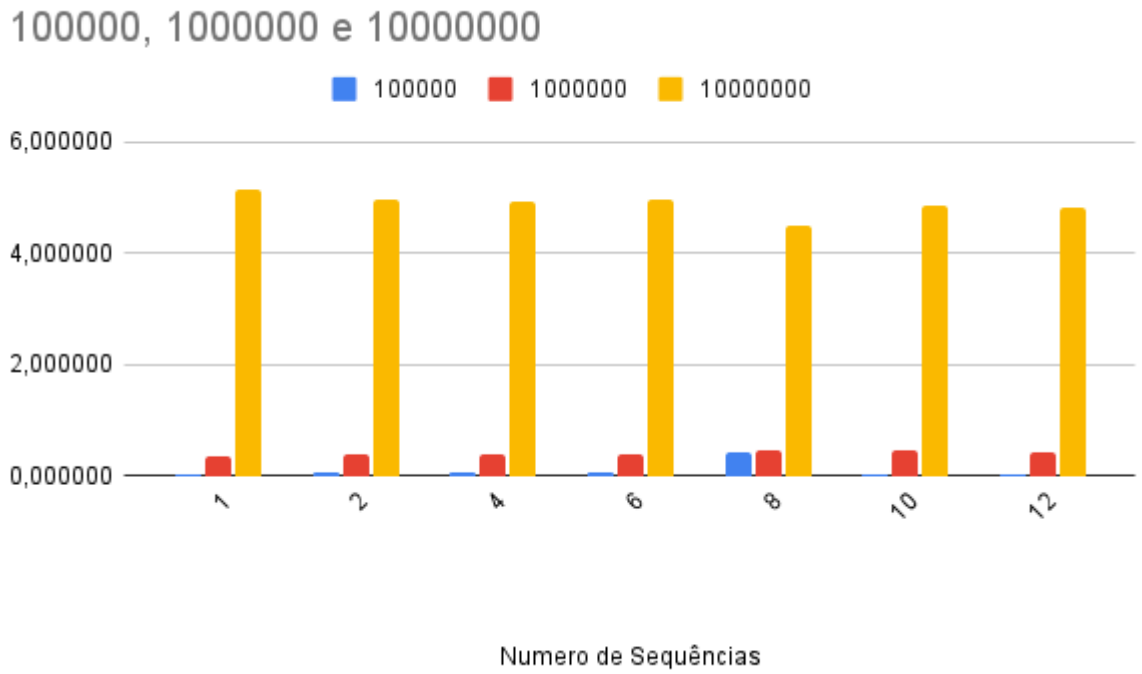
Inicialmente, os dados são distribuídos de maneira balanceada entre os processos. Em seguida, cada processo realiza uma ordenação local do seu subconjunto de dados utilizando a função qsort da biblioteca padrão do C. Na

terceira etapa, são selecionadas amostras locais igualmente espaçadas de cada processo, que servirão para determinar os pivôs globais. O processo raiz então coleta e ordena todas essas amostras, selecionando os pivôs que definirão os intervalos para o particionamento.

Na fase de particionamento, cada processo classifica seus dados nos buckets definidos pelos pivôs globais. A sexta etapa envolve uma operação crítica de troca All-to-All utilizando MPI_Alltoallv, onde cada processo envia os dados classificados para os processos responsáveis por cada bucket. Cada processo recebe então todos os elementos do seu bucket designado e realiza uma ordenação final local. Por fim, o processo raiz coleta todos os resultados ordenados para gerar a saída final.

3. Resultados Experimentais

Os testes foram conduzidos em um ambiente com processador AMD Ryzen 5 5600G com 12 cores, 16GB de memória DDR4, sistema operacional Ubuntu 20.04 LTS, utilizando OpenMPI e compilador GCC. Foram avaliados três conjuntos de dados de diferentes magnitudes: 100 mil sequências , 1 milhão de sequências e 10 milhões de sequências.



4. Análise Comparativa

A comparação entre as versões sequencial e paralela evidencia que os benefícios do paralelismo são mais acentuados conforme aumenta o volume de dados processados. O conjunto de 10 milhões de sequências alcançando speedup de 1,15x, demonstrando boa eficiência paralela.

A eficiência do paralelismo diminui progressivamente com o aumento do número de processos, fenômeno atribuído principalmente a três fatores: o overhead das operações coletivas de comunicação como Scatterv, Gatherv e Alltoallv; o desequilíbrio de carga resultante do particionamento; e as operações sequenciais que precisam ser executadas no processo raiz, como a amostragem e seleção de pivôs globais.

5. Discussão sobre Ganhos e Limitações

Os ganhos de desempenho observados confirmam a escalabilidade da solução para conjuntos volumosos de dados. O balanceamento inicial mostrou-se eficaz na distribuição equitativa da carga de trabalho, enquanto a utilização de buffers contíguos nas operações de comunicação contribuiu para reduzir o overhead. A estratégia de amostragem para seleção de pivôs demonstrou ser eficiente na criação de partições relativamente balanceadas.

Entre as limitações identificadas, destaca-se o overhead de comunicação que se torna predominante para conjuntos pequenos de dados, onde o custo das operações coletivas supera os benefícios do paralelismo. A complexidade em relação ao gerenciamento manual de memória e buffers também representa um desafio na implementação.

6. Conclusão

A implementação do Sample Sort Paralelo utilizando MPI demonstrou ser uma solução eficaz para a ordenação de grandes volumes de dados genômicos. O algoritmo apresentou escalabilidade adequada, particularmente para conjuntos de dados extensos onde o tempo de computação supera significativamente o overhead de comunicação.

As limitações relacionadas ao overhead de comunicação e desequilíbrio de carga tornam-se mais relevantes com o aumento do número de processos, porém para aplicações práticas que envolvem grandes volumes de dados genômicos, a solução paralela desenvolvida oferece benefícios substanciais em desempenho quando comparada à abordagem sequencial tradicional. O trabalho atingiu os objetivos propostos, fornecendo uma base sólida para futuras otimizações e expansões do algoritmo.