Universidade Estadual de Maringá

Relatório

Relatório da implementação de uma solução com MPI do algoritmo de Aplicação de Alinhamento Global Needleman-Wunsch

Caio Vieira Arasaki (ra127513@uem.br)

Guilherme Frare Clemente (ra124349@uem.br)

Marcos Vinicius de Oliveira (ra124408@uem.br)

Professor Ronaldo Augusto de Lara Gonçalves

Programação Concorrente

1. Introdução

Este relatório descreve a implementação de um programa em C para o alinhamento de sequências genômicas utilizando o algoritmo de Needleman-Wunsch. A aplicação foi paralelizada utilizando MPI (Message Passing Interface), em conformidade com a especificação fornecida pelo professor. O objetivo é otimizar o cálculo da matriz de scores distribuindo a carga de trabalho entre múltiplos processos MPI, mantendo a robustez e a precisão da versão sequencial.

2. Especificações e Diretrizes Seguidas

O desenvolvimento seguiu estritamente as diretrizes fornecidas, incluindo:

Tamanho das Sequências: O usuário pode definir os tamanhos das sequências, que podem ter até milhares de bases.=

Geração das Sequências: As sequências podem ser geradas de forma aleatória, lidas de arquivos ou inseridas manualmente pelo teclado. Na geração aleatória, a segunda sequência é derivada da primeira, com um percentual de alterações fornecido pelo usuário.

Composição das Sequências: As sequências são compostas pelos caracteres que representam as bases (A, T, G e C).

Matriz de Pesos: A matriz de pesos é definida pelo usuário via digitação.

Penalidade de Gap: A penalidade para gaps é fornecida pelo usuário.

Alinhamento Global: O programa exibe o maior escore para o alinhamento global e até np possibilidades de alinhamento global.

Gravação da Matriz de Scores: A matriz de scores é salva em um arquivo de texto, de forma tabulada e autoexplicativa.

Paralelismo na Construção da Matriz de Scores: O paralelismo foi implementado na etapa de construção da matriz de scores, distribuindo o cálculo de forma equitativa entre np processos MPI, com transmissão de blocos de dados entre processos para garantir eficiência.

Execução do Traceback: O traceback é realizado unicamente pelo processo 0.

3. Funcionamento

3.1. Funcionalidades principais do algoritmo

O programa apresenta as seguintes funcionalidades:

- **Definição das Sequências:** O usuário pode definir as sequências de bases por meio de três opções: geração aleatória, leitura de arquivo ou inserção manual.

- **Configuração da Matriz de Pesos e Penalidade de Gap:** A matriz de pesos e a penalidade de gap são configuradas pelo usuário.

- **Geração e Preenchimento da Matriz de Scores:** O cálculo da matriz de scores é realizado de forma paralelizada usando MPI. Cada processo MPI é responsável por calcular linhas específicas da matriz, transmitindo blocos de dados conforme são gerados.

- **Execução do Traceback:** O processo 0 executa o traceback para identificar o melhor alinhamento global com base na matriz de scores gerada.

- **Visualização e Salvamento dos Resultados:** O usuário pode visualizar as sequências alinhadas, a matriz de scores e salvar esses dados em arquivos externos.

3.2. Componentes principais

Menu de Opções: Um menu de opções permite ao usuário interagir com o programa, definindo as sequências, configurando parâmetros, gerando a matriz de scores e visualizando os resultados.

Leitura de Sequências: As sequências podem ser lidas a partir de arquivos, geradas aleatoriamente ou inseridas manualmente.

Geração da Matriz de Scores com MPI: A matriz de scores é gerada utilizando processos MPI, onde cada processo calcula blocos de linhas e transmite para o processo 0.

Traceback: O processo 0 realiza o traceback para identificar o alinhamento global.

Visualização dos Resultados: O programa permite a visualização das sequências alinhadas, da matriz de scores e dos resultados dos alinhamentos.

4. Algoritmos Paralelizados

4.1. Algoritmo para Geração da Matriz de Scores

O algoritmo de geração da matriz de scores foi paralelizado utilizando MPI. Cada processo MPI calcula um subconjunto de linhas da matriz de scores, transmitindo os blocos calculados para o processo 0 e outros processos conforme necessário.

Inicialização da Matriz: O processo 0 inicializa a primeira linha e a primeira coluna da matriz de scores e as transmite para os demais processos.

```
// Processo 0 inicializa a primeira linha e a primeira coluna e envia para os outros p
if (rank == 0)
{
    for (col = 0; col <= tamSeqMaior; col++)
        matrizEscores[0][col] = -col * penalGap;

    for (lin = 1; lin <= tamSeqMenor; lin++)
        matrizEscores[lin][0] = -lin * penalGap;

    // Enviar a primeira linha e a primeira coluna para os outros processos
    for (int i = 1; i < np; i++)
    {
        MPI_Send(matrizEscores[0], tamSeqMaior + 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        for (lin = 1; lin <= tamSeqMenor; lin++)
        {
            MPI_Send(&matrizEscores[lin][0], 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
    }
}
```

Figura 1 - Inicialização da Matriz

Distribuição do Trabalho: Cada processo MPI calcula linhas equidistantes da matriz de scores. Os blocos de dados são transmitidos tanto para o processo 0 para a consolidação final quanto para os processos que necessitam dos dados da linha calculada para efetuarem o próprio cálculo. Como é feito o envio de blocos, os processos não precisam esperar a conclusão da linha anterior inteira para calcularem os seus próprios blocos. Como visto na imagem abaixo:

```
// Processos 1 a np-1 geram e enviam as linhas de forma paralela
if (rank > 0)
{
    for (lin = rank; lin ⩽ tamSeqMenor; lin += np - 1)
    {
        for (col = 1; col ⩽ tamSeqMaior; col++)
        {
            if (col % blockSize == 1 && lin ≠ 1 && np > 2)
            {
                int bloco = (col + blockSize ⩽ tamSeqMaior) ? blockSize : tamSeqMaior - col + 1;
                int proc = (rank == 1) ? np - 1 : rank - 1;
                MPI_Recv(&matrizEscores[lin - 1][col], bloco, MPI_INT, proc, 0, MPI_COMM_WORLD, &status);
            }

            peso = matrizPesos[seqMenor[lin - 1]][seqMaior[col - 1]];
            int escoreDiag = matrizEscores[lin - 1][col - 1] + peso;
            int escoreLin = matrizEscores[lin][col - 1] - penalGap;
            int escoreCol = matrizEscores[lin - 1][col] - penalGap;

            if (escoreDiag > escoreLin)
            {
                if (escoreDiag > escoreCol)
                {
                    matrizEscores[lin][col] = escoreDiag;
                }
                else
                {
                    matrizEscores[lin][col] = escoreCol;
                }
            }
            else
            {
                if (escoreLin > escoreCol)
                {
                    matrizEscores[lin][col] = escoreLin;
                }
                else
                {
                    matrizEscores[lin][col] = escoreCol;
                }
            }
```

Figura 2 - Recebimento dos blocos e cálculo dos Scores

```
            // Envia blocos de dados calculados para o processo 0
            if (col % blockSize == 0 || col == tamSeqMaior)
            {
                int bloco = (col % blockSize == 0) ? blockSize : (tamSeqMaior - col + blockSize - 1) % blockSize;

                // Enviar para o processo 0
                MPI_Send(&matrizEscores[lin][col - bloco + 1], bloco, MPI_INT, 0, 0, MPI_COMM_WORLD);

                // Enviar para o próximo processo, se ele existir
                if (rank < np - 1 && np > 2)
                {
                    MPI_Send(&matrizEscores[lin][col - bloco + 1], bloco, MPI_INT, rank + 1, 0, MPI_COMM_WORLD);
                }
                else if (rank == np - 1 && np > 2)
                {
                    // Se o último processo (rank == np - 1), enviar de volta para o processo 1
                    MPI_Send(&matrizEscores[lin][col - bloco + 1], bloco, MPI_INT, 1, 0, MPI_COMM_WORLD);
                }
            }
        }
    }
}
```

Figura 3 - Envio dos blocos

Recepção e Consolidação: O processo 0 recebe os blocos de dados dos demais processos e monta a matriz de scores completa. Ele também identifica o maior escore para o alinhamento global. Como visto na imagem seguinte:

```
// Processo 0 recebe os blocos de dados gerados pelos outros processos e determina o maior escore global
if (rank == 0)
{
    UMaior = -1;
    linUMaior = colUMaior = -1;

    for (int proc = 1; proc < np; proc++)
    {
        for (lin = proc; lin <= tamSeqMenor; lin += np - 1)
        {
            for (int b = 0; b < tamSeqMaior; b += blockSize)
            {
                int bloco = (b + blockSize <= tamSeqMaior) ? blockSize : tamSeqMaior - b;
                MPI_Recv(&matrizEscores[lin][b + 1], bloco, MPI_INT, proc, 0, MPI_COMM_WORLD, &status);
            }
        }
    }

    for (lin = 1; lin <= tamSeqMenor; lin++)
    {
        for (col = 1; col <= tamSeqMaior; col++)
        {
            if (matrizEscores[lin][col] >= UMaior)
            {
                UMaior = matrizEscores[lin][col];
                linUMaior = lin;
                colUMaior = col;
            }
        }
    }
    printf("\nMatriz de escores Gerada.");
    printf("\nUltimo Maior escore = %d na celula [%d,%d]", UMaior, linUMaior, colUMaior);
}
```

Figura 4 - Cálculo do maior Escore Global

Explicação:

Inicialização das Bordas da Matriz de Scores : A função começa inicializando a primeira linha e a primeira coluna da matriz de scores (`matrizEscores`). O processo 0 é responsável por preencher essas bordas com valores calculados a partir da penalidade de gap (`penalGap`). A primeira linha é preenchida com valores que aumentam linearmente pela multiplicação da penalidade de gap pelo índice da coluna. A primeira coluna é preenchida de maneira semelhante, utilizando o índice da linha. Após essa inicialização, o processo 0 envia a primeira linha e a primeira coluna para todos os outros processos MPI.

Distribuição e Cálculo Paralelo das Linhas: Os processos MPI, exceto o processo 0, calculam as linhas da matriz de scores de forma paralela. Cada processo é responsável por calcular linhas equidistantes da matriz, o que garante uma distribuição balanceada do trabalho. Para cada célula da matriz, o processo calcula três possíveis valores: `escoreDiag` (diagonal), `escoreLin` (esquerda), e `escoreCol` (cima), baseando-se na sequência menor, sequência maior, e penalidade de gap. O valor máximo entre esses três é selecionado como o valor da célula.

Transmissão de Blocos de Dados: Para evitar a sobrecarga no subsistema de comunicação, os processos MPI transmitem as linhas da matriz de scores em blocos. Assim que um bloco de células é calculado, ele é imediatamente enviado para o processo 0 e para qualquer outro processo que precise desses dados. Isso permite que a matriz de scores seja construída de maneira paralela e eficiente, sem que os processos fiquem ociosos esperando a conclusão de toda a linha.

Consolidação dos Resultados pelo Processo 0: O processo 0 recebe os blocos de dados de todos os processos MPI e monta a matriz de scores completa. Ele também é responsável por identificar o maior escore global, que representa o melhor alinhamento possível entre as sequências. Após a conclusão do cálculo, o processo 0 compara os maiores escores locais enviados pelos outros processos e determina o escore global máximo.

Vantagens da Paralelização com MPI:

Melhor Utilização de Recursos: Utilizando múltiplos processos MPI, o cálculo da matriz de scores é distribuído entre vários nós ou núcleos, melhorando a eficiência e reduzindo o tempo de execução.

Redução do Tempo de Execução: A paralelização permite que a matriz de scores seja preenchida muito mais rapidamente do que seria possível com um único processo.

Escalabilidade: A abordagem pode ser escalada para aproveitar arquiteturas de computação distribuída com muitos nós, o que é especialmente benéfico para sequências de grande tamanho.

Considerações sobre a Paralelização:

Balanceamento de Carga: É crucial que a distribuição das linhas entre os processos seja equilibrada para evitar que alguns processos fiquem ociosos enquanto outros ainda estão calculando.

Sincronização e Consistência: A sincronização e a comunicação entre os processos são feitas de maneira a minimizar a sobrecarga, utilizando transmissões de blocos. É importante garantir que as mensagens MPI sejam tratadas de forma eficiente para evitar congestionamento.

Inicialização e Conclusão: A criação de processos e a sincronização dos resultados adicionam uma sobrecarga, mas esta é justificada pela redução no tempo de execução dos cálculos. Além disso, a coleta dos maiores escores por parte do processo 0 garante a consistência dos resultados.

4.2. Algoritmo de Traceback

O traceback é realizado exclusivamente pelo processo 0. A partir do maior escore identificado na matriz de scores, o processo 0 reconstroi o alinhamento global das sequências. A função é equivalente a versão sequencial, porém, dentro da função `TrataOpcao()` chamamos o traceback() apenas para o processo 0.

```
case 10:
    if (rank == 0)
    {
        traceBack();

        // Envia sinal para outros processos
        for (int i = 1; i < size; i++)
        {
            MPI_Send(&resp, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
        }
    }
    else
    {
        // Recebe o sinal do processo mestre
        MPI_Recv(&resp, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &st);
    }
    break;
```

Figura 4 - TraceBack iniciado apenas pelo processo 0

Explicação:

Responsabilidade do Processo 0: A função `traceBack` é responsável por reconstruir o melhor alinhamento global entre as sequências, a partir da matriz de scores calculada previamente. No contexto do MPI, essa função é executada exclusivamente pelo processo 0, que possui a matriz de scores completa após a etapa de cálculo paralelizado.

Inicialização do Traceback: O processo 0 inicia o traceback a partir da célula que contém o maior escore global, identificada durante a construção da matriz de scores. As variáveis `tbLin` e `tbCol` são inicializadas para apontar para a linha e a coluna dessa célula na matriz de scores.

Execução do Traceback: A função percorre a matriz de scores "de trás para frente" (do maior escore até a célula [0,0]), reconstruindo o alinhamento baseado nas decisões feitas durante o preenchimento da matriz. Para cada posição, são considerados três possíveis movimentos:

1. **Diagonal (`escoreDiag`):** Alinhar as bases atuais de ambas as sequências. Esta é a escolha se as bases forem iguais ou se o movimento diagonal oferecer o melhor escore.
2. **Esquerda (`escoreLin`):** Introduzir um gap na sequência maior.
3. **Cima (`escoreCol`):** Introduzir um gap na sequência menor.

A função escolhe o movimento que maximiza o escore, e a posição correspondente é atualizada. Os caracteres das sequências alinhadas são armazenados nas variáveis `alinhaGMaior` e `alinhaGMenor`.

Tratamento de Gaps Restantes: Após sair do loop principal, caso restem bases não alinhadas em uma das sequências (quando `tbLin` ou `tbCol` não for zero), a função adiciona gaps correspondentes para completar o alinhamento. Isso garante que o alinhamento seja da mesma extensão em ambas as sequências..

Reversão do Alinhamento: Como o traceback constroi o alinhamento do final para o início, o alinhamento resultante é armazenado de trás para frente. Assim, a função inverte as sequências alinhadas antes de apresentá-las ao usuário. Esta inversão é feita manualmente, trocando as posições dos elementos nos arrays `alinhaGMaior` e `alinhaGMenor`.

Vantagens do Traceback no Processo 0:

Consistência: Como o traceback é executado por um único processo (processo 0), evita-se a necessidade de sincronização adicional entre processos, garantindo a consistência do alinhamento final.

Eficiência: A realização do traceback em um único processo reduz a complexidade de comunicação que seria necessária se múltiplos processos fossem envolvidos na tarefa.

Simplicidade: Delegar a responsabilidade do traceback a um único processo simplifica o código e o fluxo de execução, facilitando a manutenção e a depuração.

Considerações sobre o Traceback MPI:

Requisito de Memória: O processo 0 precisa armazenar toda a matriz de scores para realizar o traceback, o que pode ser exigente em termos de memória, especialmente para sequências muito grandes.

Eficiência: O uso do processo 0 para executar o traceback garante que o alinhamento global seja realizado de maneira eficiente, sem a necessidade de comunicação adicional entre processos após a construção da matriz de scores.

Completação do Alinhamento: A função assegura que todo o alinhamento, incluindo gaps restantes, seja corretamente gerado e armazenado antes de ser exibido ou salvo.

5. Erros Gerais

Leitura de Sequências: Podem ocorrer erros na leitura das sequências, especialmente na entrada manual, onde erros de digitação não são tratados de forma robusta.

Sincronização entre Processos: Falhas na sincronização ou na transmissão de dados entre processos podem resultar em inconsistências na matriz de scores.

Geração de Alinhamentos: A geração de alinhamentos pode não ser otimizada se os pontos de início do traceback não forem bem escolhidos.

6. Restrições e Limitações

Tamanho das Sequências: O tamanho das sequências é limitado à capacidade de memória do sistema, com um limite máximo definido no código.

Balanceamento de Carga: Embora o trabalho seja distribuído entre os processos MPI, o balanceamento de carga pode não ser perfeitamente igual, dependendo do número de processos e do tamanho das sequências.

Sincronização e Comunicação: A transmissão de blocos de dados entre processos é feita de forma a evitar a congestão do subsistema de mensagens, mas a eficiência pode variar dependendo da arquitetura do sistema.

7. Exemplos de Uso

Ao executar o programa, com os seguintes comandos:

$mpicc <nome do arquivo>.c -o <executável>

$mpirun - <número de processos> ./<executável>

O usuário verá um menu com opções para configurar o alinhamento:

```
Programa Needleman-Wunsch com MPI

Menu de Opcao:
<01> Ler Matriz de Pesos
<02> Mostrar Matriz de Pesos
<03> Ler Penalidade de Gap
<04> Mostrar Penalidade
<05> Definir Sequencias Genomicas
<06> Mostrar Sequencias
<07> Definir Tamanho do Bloco
<08> Gerar Matriz de Escores
<09> Mostrar Matriz de Escores
<10> Gerar Alinhamento Global
<11> Mostrar Alinhamento Global
<12> Sair
Digite a opcao =>
```

<0> Primeiro o usuário pode selecionar quantos processos serão necessários para resolver o problema em questão, neste exemplo será executado com 2 processos;

<1> Após aparecer o menu, o usuário pode selecionar digitar manualmente a matriz de pesos, caso o usuário não queira, a matriz será por padrão a matriz identidade, para esse exemplo vamos digitar a matriz {2, -1, -1, -1, -1, 2, -1, -1, -1, -1, 2, -1, -1, -,1 -,1, 2}, ou seja, todos os valores são -1 mas a diagonal principal é 2;

<2> Caso o usuário queira ver a matriz de pesos basta selecionar essa opção;

```
Matriz de Pesos Atual:
      A   T   G   C
   A -1   2  -1  -1
   T -1   2  -1  -1
   G -1  -1   2  -1
   C -1  -1  -1   2
```

<3> O usuário pode digitar qual a penalidade de Gap ele quer utilizar, para o nosso exemplo vamos utilizar um gap de 0 (Por padrão);

<4> O usuário pode observar a penalidade que foi aplicada;

```
Digite a opcao => 4

Penalidade = 0
```

<5> O usuário pode definir a sequência genômica de 3 formas (1. manual, 2. aleatória, 3. por arquivo), nesse exemplo usaremos manualmente, onde a sequência maior será: CGCTATAT, e a sequência menor: TATACTA;

<6> O usuário pode ter a opção de ver a sequência digitada, ou pelas outras opções também;

```
Sequencias Atuais:

Sequencia Maior, Tam = 8
CGCTATAT

Sequencia Menor, Tam = 7
TATACTA
```

<7> Nesse passo é necessariamente importante que o usuário digite o tamanho do bloco antes de gerar a matriz, se não pode ocorrer um erro de divisão por 0 e o algoritmo parar de rodar. Neste exemplo, usaremos um bloco de tamanho 2;

<8> O usuário gera a matriz de scores e salva em um arquivo de texto chamado matriz_escores.txt;

```
Matriz de escores Gerada.
Ultimo Maior escore = 10 na celula [7,8]
Matriz de escores gravada em matriz_escores.txt
```

<9> O usuário pode observar a matriz de scores que foi gerada;

```
Matriz de escores Atual:
          0   1   2   3   4   5   6   7   8
          -   C   G   C   T   A   T   A   T
  0   -   0   0   0   0   0   0   0   0   0
  1   T   0   0   0   0   2   2   2   2   2
  2   A   0   0   0   0   2   4   4   4   4
  3   T   0   0   0   0   2   4   6   6   6
  4   A   0   0   0   0   2   4   6   8   8
  5   C   0   2   2   2   2   4   6   8   8
  6   T   0   2   2   2   4   4   6   8  10
  7   A   0   2   2   2   4   6   6   8  10
```

<10> O usuário gera o último melhor alinhamento feito pelo traceback;

<11> O usuário consegue ver esse alinhamento feito pela matriz;

```
Alinhamento Obtido - Tamanho = 10:
CGCTATA-T-
---TATACTA
```

<12> O usuário finaliza o programa;

Isso define que com a sequência definida, a matriz gerada, com 2 processos e um bloco de tamanho 2, esse foi o melhor alinhamento obtido de todos os possíveis, de tamanho 10.