Virgínia Baroni e Clauber Stipkovic
Professor Calebe de Paula Bianchini
6 May 2018

Relatório
Maratona de Programação Paralela

Índice

1. 4th Marathon of Parallel Programming – SBAC'2009

Problem B - DNA Subsequences 2

Descrição do Problema

O formato FASTA é um formato de texto para representação de sequências nucleotídeas ou sequências peptídeas, onde nucleotídeos ou amino-ácidos são representados usando códigos de letras únicas. O código é normalmente formado por um header e a sequência, sendo que o header pode possuir certos identificadores dependendo da database da NCBI (National Center for Biotechnology Information).

Foi pedido dos participantes que escrevessem um código paralelo baseado em um sequencial, para encontrar subsequências de DNA (uma busca na sequência completa) em uma database com formato FASTA. Caso essa busca retornasse com mais do que um resultado, que fosse ser listado na ordem em que foi encontrado.

Solução Sequencial

Feito obrigatório para todos os participantes que o input do programa seria ler de dois arquivos diferentes, um da database e outro da query, ambos em formato FASTA.

Como já explicado, o formato FASTA possui um header que inicia com '>' e conteúdo de caracteres de DNA (A, T, C, G) com um limite de 1 milhão de caracteres.

Obs. Não há solução sequencial no pdf de problemset dessa maratona.

Baseado nos input/output da explicação do desafio, eu assumo que a parte principal do algoritmo foi uma função que recebia dois arquivos em formato FASTA, o database e query como abaixo:

EX database: >ex1
AGTTTAGATGACCCTGAGTCTG (1)

EX query: >query1
GTT (2)
>query2
CTG (3)

Essa função então percorria a linha seguinte à que iniciava com '>' e buscava sequencialmente pela primeira ocorrência inteiriça de (2), (3), (n) em (1) com um contador de ocorrências. Assim sucessivamente até o fim de (1). E retornava:

(2) -1 linhas: >query1
(1) -1 linhas = '>ex1'
1 = quantas vezes (2) apareceu em (1).

(3) - 1 linhas: >query2

(1) - 1 linhas = ‘ex1'

2 = quantas vezes (3) apareceu em (1).

2. 7th Marathon of Parallel Programming – WSCAD-SSC/SBAC-PAD'2012

Problem A - Bucket Sort

Descrição do Problema

Bucket Sort é um algoritmo que separa e ordena elementos de uma lista em um numero de “baldes” finitos, delimitados por range de números os quais vão receber os elementos da lista. Seu método de atribuição é toma como base a idéia do algoritmo de divisão e conquista, onde ao receber um vetor não-ordenado, ele organiza esses elementos por n pedaços, que possuem uma regra delimitadora.

Assim que todos os elementos foram inseridos dentro do seu respectivo Bucket, esses elementos são ordenados e devolvidos para a lista original.

O processo do Bucket Sort pode ser aplicado também dentro de cada Bucket enquanto for possível separar a lista pedaços menores.

Esse algoritmo de ordenação tem complexidade linear de O(n), quando o vetor a ser ordenado contém valores que são uniformemente distribuídos.

Solução Sequencial

Dada uma lista de elementos não ordenados, é inicializado um vetor de Bucket vazios.

O algoritmo percorre o vetor copiando a posição em memória de cada um elemento para o seu respectivo Bucket.

É executada a ordenança em cada Bucket, que é devolvido com seus elementos já ordenados.

3. 9th Marathon of Parallel Programming – WSCAD'2014

Problem F - Dijkstra

Descrição do Problema

Um dos códigos mais conhecidos para a solução de caminho de custo mínimo em Grafos é definitivamente o algoritmo de Dijkstra.

Existem variações do algoritmo original, que propunha sempre o menor caminho entre dois vértices, e que hoje encontram um caminho completo entre vários vértices, de um ponto A à B.

Gramaticalmente falando, o algoritmo funciona dessa forma:

Um ponto de partida inicialmente marca uma distância à qualquer outra intersecção de outros pontos do mapa até seu ponto final, de maneira desordenada. Ou seja, neste grafo que

há uma determinação do ponto inicial e final, todos seus meios possuem pesos. Esses pesos serão a distância entre pontos, calculados supostamente de maneira randômica.

Ao supor que nenhum dos outros pontos já tenha sido visitado, se parte do ponto principal e à cada interação, é selecionado um ponto de partida e um destino, entre o ponto inicial e final, a serem chamados de intersecção. Essa distância inicial começa com valor nulo, pois ainda não houve uma movimentação de nenhum ponto à outro. Da intersecção inicial, é atualizada a distância para qualquer intersecção não-visitada que está diretamente conectada ao ponto inicial com esse valor da somatória, se for menor que a mesma.

Para facilitar a identificação de quais caminhos foram escolhidos, cada caminho é imposto como visitado, e assim por diante, até ser encontrado o primeiro menor caminho temporário. Pontos marcados como "menor" serão então o caminho mais viável da intersecção e não se voltará a eles.

Esse processo de atualização é realizado até que a somatória de cada intersecção seja calculada como menor até chegar no ponto de destino de todo o grafo, resultando no menor caminho entre dois pontos.

É valido pensar que esse algoritmo apenas considera a próxima intersecção a partir de um ponto inicial, ao invés de englobar todo o problema de uma vez só, o que o torna lento dependendo do tamanho do grafo, e impossível em certos casos, de obter sucesso caso um peso de um caminho já desconsiderado seja negativo.

Solução Sequencial

A solução inicia sem um grafo pré-formado, recebendo para tal três inteiros que indicam quantos nodes, caminhos por nodes e um número random que indica o peso do node.

Com o grafo pronto, a função principal recebe apenas o grafo e o ponto inicial, e controla um vetor de pontos visitados e outro vetor da incrementação da somatória da distância para no fim saber o menor caminho, e poderia ter sido substituído por uma estrutura de Fila.