Modelo PCAM

João Vitor Silva Ramos, 10734769
Giovani Decico Lucafó, 10288779
Marcelo Isaias de Moraes Junior, 10550218
Victor Giovannoni Vernalha, 10786159
Vitor Santana Cordeiro, 10734345

1. Particionamento

Para particionar esse problema, pensando na menor granularidade possível, iremos gerar uma tarefa para cada caractere da string original. Ou seja, considerando uma string de tamanho N, serão geradas **N tarefas**.

Se o caractere está na posição i, i >= 0, então a correspondente tarefa será responsável por lidar com a substring que começa naquela posição e termina quando achar um espaço (“ ”) ou o final da string. O objetivo de uma tarefa individual é retornar o número de caracteres lidos até o próximo espaço (sem contar este), sem se preocupar em “passar por cima” de caracteres que serão analisados por outras tarefas.

2. Comunicação

O primeiro passo é a atribuição de dados para cada tarefa. Para tanto, cada uma receberá uma posição da string original, indo de 0 até N-1.

Após cada tarefa realizar o seu trabalho de computar o número de caracteres até o espaço, cada uma retornará o seu “candidato” ao tamanho da maior palavra, considerando seu contexto (ínicio e fim). Após esse processo, uma operação de redução será realizada em $\log_2(n)$ passos para determinar o máximo entre os retornos de todas as tarefas, necessitando de uma comunicação global entre “tarefas adjacentes”: a tarefa j irá se comunicar com a tarefa j + 1, a tarefa j + 2 irá se comunicar com a tarefa j + 3 e assim por diante.

Um detalhe de comunicação importante é que o tamanho da string original (N) deverá ser compartilhado por todas as tarefas globalmente, através de uma variável *read-only*. Isto porque as tarefas responsáveis por caracteres que caem em palavras que são a última da string não terão um

espaço para usar de critério de parada. Portanto, a comparação da posição atual contra o tamanho da string se faz necessária para não estourar o tamanho da string.

3. Aglomeração

A aglomeração das N tarefas é feita em P processos, sendo P o número de elementos de processamento. Assim, conseguimos aumentar a granularidade de cada processo.

Cada processo receberá uma posição inicial *i* que será um múltiplo de teto(N/P), 0 <= *i* <= N. Desta forma, evita-se de gerar um processo para cada um dos N/P - 1 caracteres adjacentes a este. No caso de a divisão não ser exata, a "última tarefa" (que fica responsável pelo maior índice) irá computar até o final do tamanho da string, sem maiores problemas (conforme descrito na seção de Comunicação).

Por exemplo, caso N=17 e P=4, teto(N/P) = teto(4.25) = 5. Então, o primeiro processo começará sua computação a partir da posição 0, o segundo a partir da posição 5, o terceiro da posição 10 e o quarto da 15, parando na posição 10 por contato da checagem de tamanho da string (para não o estourar). A grande diferença da computação inicialmente pensada é que agora cada processo irá ignorar o primeiro espaço que encontrar **caso ele esteja dentro do seu *range* de computação**. Utilizando do exemplo acima e supondo que a string original seja "o pai do meu gato", o primeiro processo, que estará responsável pela substring que começa da posição 0, irá incrementar em 1 a sua variável local a cada caractere lido sequencialmente (resultando em max_local = 1) e, quando achar o espaço na posição 1, ele irá ignorar esse espaço como critério de parada já que ele ainda não ultrapassou o espaço que estava sob sua responsabilidade (que era de 0 a 4). Então, esse processo continuará incrementando sequencialmente uma outra variável local auxiliar até, aí sim, atingir um espaço (" ") que esteja numa posição fora de sua "jurisdição". Somente neste momento que o processo parará.

Note que um processo não se importa em computar em cima de caracteres que outros processos já computaram. Portanto, um processo pode ler mais caracteres do que originalmente designados a ele, porém **nunca**

menos. Após a finalização dessa computação, o processo segue igual: faz-se uma redução para achar o máximo dentre os valores retornados por cada processo.

4. Mapeamento

Considerando a execução deste algoritmo em um cluster de computadores, e considerando que os nós desse cluster tem um desempenho homogêneo, o mapeamento desses P processos será dado por meio de uma fila circular Round-Robin. No caso de termos P = PROC elementos de processamentos, um processo será atribuído para cada elemento de processamento. Caso o desempenho dos nós do cluster seja diferente, o mapeamento dos nós deve considerar a diferença de desempenho entre os elementos de processamento, o que pode ser feito estaticamente ou dinamicamente (considerando a atribuição de mais carga de trabalho para os nós que estejam menos ocupados).