Módulo 2 - Laboratório 5
Problemas clássicos de concorrência usando locks e variáveis de condição (barreira)

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ

Introdução

O objetivo deste Laboratório é praticar a implementação de problemas clássicos de concorrência usando locks e variáveis de condição. Nessa semana, vamos abordar o problema de **soma de prefixo** (do termo em inglês *prefix sum*), fazendo uso de **barreira** (ou **sincronização coletiva**).

Atividade 1

Descrição: Considere o seguinte problema. Dado um vetor de inteiros, precisamos calcular as somas parciais em cada posição desse vetor (somatório de todos os elementos que antecedem essas posições, incluindo o elemento da própria posição). Por exemplo, dado o vetor $[1, 4, -1, 7]$ como entrada, o vetor resultante com as somas parciais em todas as posições é: $[1, 5, 4, 11]$. O primeiro elemento se mantém, o segundo elemento é a soma de $1 + 4$, o terceiro elemento é a soma de $1 + 4 + (-1)$, e o quarto elemento é a soma de $1 + 4 + (-1) + 7$.

O algoritmo sequencial para resolver esse problema é bastante simples:

```
for(int i=1; i<n; i++)
   vetor[i] = vetor[i] + vetor[i-1];
```

Repare que o próprio vetor de entrada é modificado, isto é, usamos apenas um vetor que é alterado como saída do programa.

Para resolver esse problema de forma concorrente, vamos considerar que o número de elementos do vetor n é sempre uma potência de 2. E não vamos nos preocupar com ganho de desempenho nessa atividade, apenas com o "raciocínio paralelo".

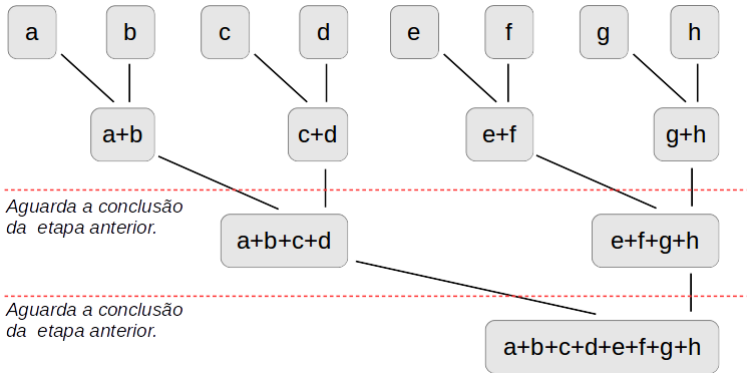
Criaremos n threads, com índices de 0 a $n - 1$. (O número máximo de elementos do vetor nesse caso ficará limitado ao número máximo de threads que podemos criar.) Em cada iteração do algoritmo, as threads lêem o elemento que está na posição $(id - i)$ (caso ele exista, id é o identificador da thread) e guardam esse valor em uma variável auxiliar; em seguida, somam esse valor com o valor da posição id e escrevem o resultado na posição id.

Repare a necessidade de **sincronização coletiva** entre a leitura e armazenamento na variável auxiliar e o passo seguinte que faz a soma de fato. Isso porque precisamos garantir que todas as threads leram o valor atual, antes das somas começarem.

O valor de i inicia e cresce em potência de 2: $2^0, 2^1, 2^2, 2^4...$ até o limite do tamanho do vetor. Note que as threads vão terminando sua tarefa em momentos diferentes, sendo que apenas a última thread executa até a última iteração (e a primeira thread

não executa nenhuma iteração, uma vez que o primeiro elemento do vetor se mantém na saída).

A figura abaixo ilustra essa ideia, olhando da perspectiva da última thread. A linha horizontal tracejada destaca a necessidade de uma segunda **sincronização coletiva**, para aguardar a conclusão de todas as somas de pares de valores da iteração atual antes de avançar para a iteração seguinte, uma vez que os resultados dessas somas são usados como valores de entrada para as somas realizadas na próxima iteração.



Tarefa:

1. Compreenda primeiro o problema e a solução concorrente proposta, com os pontos de sincronização coletiva.
2. Implemente o algoritmo concorrente proposto acima, seguindo as restrições e considerações feitas. Permita que o número de elementos do vetor possa variar de uma execução para outra (sempre mantendo a condição de que seu tamanho seja potência de 2). Lembre-se de usar um único vetor, para entrada e saída.
3. Teste sua solução (de forma automática) para garantir que o vetor resultande está correto.

Entrega: Disponibilize o código implementado em um ambiente de acesso remoto (GitHub ou GitLab). **Use o formulário de entrega desse laboratório para enviar o link do repositório do código implementado e responder às questões propostas.**