PCAM Menor valor maior que Vet[k]

- Daniel Sá Barretto Prado Garcia 103734344
- Felipe Guilermmo Santuche Moleiro 10724010
- Laura Alves de Jesus 10801180
- Tiago Marino Silva 10734748

Particionamento

Após a leitura do vetor Vet, ele é particionado em partes iguais, sendo a menor parte possível uma posição única do vetor e a maior possível o próprio vetor. Assim, são geradas N tarefas. Cada tarefa é responsável por encontrar o menor valor maior que Vet[k] dentro de sua parte do vetor.

Comunicação

A comunicação entre as tarefas é feita através de uma região de memória compartilhada, correspondente a 3 valores inteiros de 32 bits, que armazenam o menor valor maior que Vet[k] encontrado, o tamanho do vetor e o valor de Vet[k], respectivamente. Além disso, cada tarefa armazena localmente o valor calculado do menor valor maior que Vet[k] em sua parte. Cada tarefa, após o cálculo deste valor, entra em uma região crítica para ler e escrever nessa região de memória compartilhada. Nesta região crítica, ela verifica se o valor local calculado é menor que o valor global e diferente de -1, caso seja, ela atualiza o valor global para o valor calculado. Ao acessar a região crítica, a tarefa impede que outras tarefas entrem nesta mesma região simultaneamente.

Aglomeração

A aglomeração das N tarefas é feita em T threads, cada uma é responsável por encontrar o menor valor maior que Vet[k] dentro de sua parte do vetor.

Após a leitura da entrada, teremos um vetor com P posições. Iremos dividir essas P posições em T blocos, onde T é o número de threads sendo executadas no sistema. Por exemplo, se o vetor tem 20 posições e temos 4 threads, cada thread achará um mínimo maior que o valor vet[k] em um bloco de (20/4) posições, ou seja de 5 posições consecutivas do vetor. Após isso, cada thread irá atualizar o valor do menor global, pertencente a uma variável compartilhada, caso seu valor encontrado seja menor que ele e seja diferente de -1.

Mapeamento

O mapeamento das Threads em cada núcleo de processamento lógico é feito pelo SO e não depende da implementação. Entretanto, ao criar um número de threads iguais ao número de núcleos de processamento lógico, espera-se que com sorte o SO escalone cada thread para um núcleo, utilizando toda a capacidade do processador de uma só vez.