Nome	Nº USP
Abner Eduardo Silveira Santos	10692012
Gyovana Mayara Moriyama	10734387
Henrique Matarazo Camillo	10294943
Vitor Augusto de Oliveira	9360815

Descrição do Problema

O problema a ser resolvido determinava que seria necessário calcular o maior número de uma array seguindo um conjunto de orientações pré-definidas: todos os índices do vetor deveriam ser preenchidos com o valor 1, com exceção do elemento central, que deveria apresentar o valor de tamanho do vetor; em seguida, calcula-se os máximos locais; e por fim, determina-se o máximo global através da avaliação de máximos locais. Foi desenvolvido um algoritmo paralelo que segue a metodologia PCAM, detalhada abaixo.

Particionamento

Sejam $size$ o número de elementos do vetor e $nThreads$ o número de threads fornecidas pela biblioteca OpenMP, define-se o valor inteiro $partitionSize$ que é calculado através da fórmula $\frac{size}{nThreads}$ e que corresponde ao tamanho da partição que cada thread será encarregada de realizar as operações de preenchimento de vetor e cálculo de máximo local. No caso em que a divisão apresentada anteriormente não fornece um resultado exato, atribui-se à thread de $id = nThreads - 1$ o tamanho devidamente ajustado, garantindo a coerência do algoritmo.

Comunicação

No que diz respeito a comunicação, a abordagem local vs global foi empregada para resolver o problema. As variáveis num, $threads$, $size$ e max que correspondem, respectivamente, ao vetor com todos os valores, o número de threads sendo utilizadas, o tamanho do vetor e a variável que guarda o maior valor do vetor, encontram-se na região compartilhada de memória e são acessados por cada thread durante a etapa paralela. Já a variável *max_parcial*, que corresponde ao inteiro que armazena os máximo locais, é uma variável privada, ou seja, cada thread possui sua própria instância dessa variável.

A execução paralela do algoritmo é dividida em duas etapas: preenchimento do vetor e cálculo dos máximos locais. Em ambas as etapas, cada thread age respeitando sempre sua região de particionamento, sendo que a diretiva *#pragma omp barrier* as divide garantindo a sincronização do preenchimento do vetor antes do cálculo de máximos.

Além disso, utilizamos o *lock* para encontrar o maior valor do vetor, para que, não haja confusão na hora de comparar o maior valor, assim, cada thread precisa esperar a

anterior acabar a comparação para fazer a sua comparação do maior valor, evitando assim que tenhamos várias threads escrevendo na mesma variável assincronamente.

Aglomeração

O tamanho das regiões de partição (*partitionSize*) foram definidas através da relação entre o número de elementos do vetor (*size*) e a quantidade de threads fornecidas (*nThreads*) e correspondem a blocos contínuos dentro da array.

A regra que rege os limites de extremidade da cada região é a seguinte:

1. As variáveis *start* e *end* representam, respectivamente, o limite à esquerda e à direita de cada região.
2. $start = partitionSize * id$, onde id é o número da thread sendo executada.
3. se $end \neq nThreads - 1$:

   $end = start + partitionSize$

   senão:

   $end = size$

Desta forma, garante-se que cada thread respeita os limites de cada região e opera corretamente sobre os vetores nas etapas de preenchimento e definição de máximos, independente dos valores de tamanho de vetor e número de threads.

Mapeamento

Como nesse momento só temos acesso a um dos nós do cluster por vez com o OpenMP e ele considera que todos os núcleos de processamento são homogêneos em sua distribuição de threads, nosso mapeamento é um mapeamento simples em que cada elemento de processamento receberá um processo.