Documentação do Projeto

Estruturação do projeto:

O projeto foi construido em três módulos, sendo dois deles totalmente independentes entre si, e o terceiro responsável pela "conexão" dos dois primeiros, formando um único programa. Segue abaixo uma breve descrição de cada um dos módulos.

- Módulo Sequencial:

Como o próprio nome diz, este é o módulo responsável pela solução sequencial do problema. Módulo simples, contém apenas duas funções: uma para alocação de um vetor de inteiros em memória, e outra que consiste na implementação do algoritmo sequencial para resolução do problema. A única estrutura de dados associada a este módulo é um vetor de inteiros de 3 posições, utilizado pela função *processaMatrizSeq()* para armazenar e retornar o resultado final para a função main (o primeiro índice do vetor armazena a soma dos elementos, o segundo índice o maior dos elementos e o terceiro índice o menor deles).

A função *processaMatrizSeq()* recebe como parametros da função *main()* - que pertence ao módulo principal do programa - uma matriz e sua dimensão. Seu algoritmo para calcular a soma e obter o maior e menor elemento é simples e intuitivo: através de dois loops encadeados percorre-se toda a matriz e, a cada iteração, são armazenados a soma dos elementos já percorridos, além do maior e menor dos elementos encontrados até o momento. Ao final do loop, o resultado é retornado à função *main()*.

- Módulo Concorrente:

Overview:

Este é o módulo responsável pela solução concorrente do problema, e é composto por seis funções: três para alocação dinâmica de vetores de diferentes tipos (inteiros, semáforos e threads); duas que são as responsáveis pela implementação do algoritmo concorrente para resolução do problema; e uma que é a função principal deste módulo, responsável por receber os parâmetros da função *main()* e coordenar a execução das outras funções de maneira a obter a solução do problema e retorná-la a *main()*.

No módulo concorrente, a estratégia adotada para a solução do problema consiste em "dividir" a matriz de entrada em quadrantes de dimensão 8x8, dado que cada thread será responsável por calcular a soma, o maior e menor elemento de um quadrante, e armazenar cada um destes resultados em vetores compartilhados pelas threads. Após todas as threads encerrarem o processamento de todos os quadrantes, inicia-se uma segunda etapa, que consiste no processamento iterativo dos resultados obtidos na etapa anterior até que se obtenha o resultado final.

A concorrência nesta solução se dá na 1ª etapa, quando o processamento dos quadrantes é executado de forma paralela por todas as threads, e na segunda etapa, onde o processamento dos vetores resultantes da etapa anterior é também executado de forma paralela, porém por um número menor de threads a cada iteração. Note-se que isto não

representa um problema ou deficiência do algoritmo, e sim uma característica do problema, que a cada iteração da segunda etapa tem a sua carga de processamento reduzida à metade.

Detalhes da implementação:

- *Identificação do quadrante a ser processado pela thread:*

Para identificar por qual quadrante cada thread será responsável, abstraímos a matriz original como uma matriz de quadrantes, onde cada elemento dessa matriz é formado por uma submatriz 8x8. O índice correspondente a linha do elemento dessa nova matriz é calculado como tid/(dim/8), onde tid é o id da thread e (dim/8) é a dimensão dessa nova matriz. Já o índice correspondente a coluna do elemento é calculado como tid%(dim/8). Com esses cálculos, conseguimos determinar o offset do primeiro elemento da submatriz a ser processada pela thread, tornando a tarefa de percorrê-la trivial.

- *Barreira:*

O mecanismo de barreira foi implementado através da utilização de um contador, inicializado com o número total de threads envolvidas no processamento da matriz na primeira etapa do algoritmo. Cada thread decrementa o contador após alcançar a barreira e, dependendo de qual for a thread, ela se bloqueia esperando o contador chegar a zero **ou** encerra a sua execução neste ponto. Esta estratégia foi adotada com o intuito de otimizar a solução, já que nem todas as threads seguem sendo utilizadas na segunda etapa do algoritmo, como já exposto anteriormente. Dessa forma, evitamos um consumo desnecessário de CPU por threads que não contribuem para a solução do problema. Finalmente, quando o contador chega a zero, a última thread que alcançou a barreira emite sinais de desbloqueio apenas para as threads que foram bloqueadas.

A identificação das threads a serem bloqueadas ou encerradas foi feita através do id da própria thread, que foi gerado de maneira sequencial começando de zero. Pela característica do problema, apenas as n/2 primeiras threads são utilizadas na iteração seguinte, considerando n como o número total de threads. Portanto, após o decremento do contador, verifica-se o id da thread para então bloqueá-la ou encerrá-la. Vale observar que já para a primeira iteração da segunda etapa o contador é reiniciado com metade do número de threads utilizadas no passo anterior. Este processo se repete até que o contador seja reiniciado com 1, e a última iteração seja executada.

- Módulo Principal:

Este é o módulo responsável pela interligação dos módulos anteriores, compondo uma única solução. Toda a entrada e saída de dados cabe a este módulo, que lê a matriz de entrada do arquivo, chama a execução do programa sequencial ou concorrente, de acordo com a escolha do usuário, e por fim apresenta os resultados finais na tela, além de efetuar e apresentar a tomada de tempo de execução da solução escolhida. A escolha da opção da programa a ser executada é feita através do parâmetro argv da main. Os parâmetros definidos são: “sequencial” para executar o módulo sequencial ou “concorrente” para execução do módulo concorrente.