Trabalho Prático Aeds III

Pedro Nascimento Costa

TP3

Introdução:

A proposta do trabalho envolve resolver um problema através do uso de programação paralela pelo uso de *pthreads* em C. O problema consiste em determinar o resultado de maior valor em uma situação em que cada valor escolhido numa matriz resulta em outros valores desaparecendo, no caso, os valores aos lados esquerdo e direito e todos da linha acima e abaixo.

Encontramos a solução ao analisar a matriz de dimensões n x m, que por sua vez é composta por números, cada um indicando a população de uma cidade, o objetivo é escolher um conjunto de cidades cujos valores populacionais somados é o maior possível no contexto do problema.

A solução proposta consiste em entender primeiro que linhas logo acima ou logo abaixo uma da outra são independentes e valores ao lado um do outro na mesma linha também, isso ocorre devido a característica de que ao escolher um valor numa linha, os valores logo ao lado viram zero e os contidos nas linhas logo abaixo e acima também.

Com isso em vista, temos então a solução proposta, que consiste em achar primeiro o valor máximo de cada linha e depois o valor máximo da matriz utilizando o máximo de cada linha. São então utilizadas *threads* que paralelamente acham o máximo para as linhas e depois, pelo mesmo processo que determina o máximo de cada linha, determina-se o máximo da matriz.

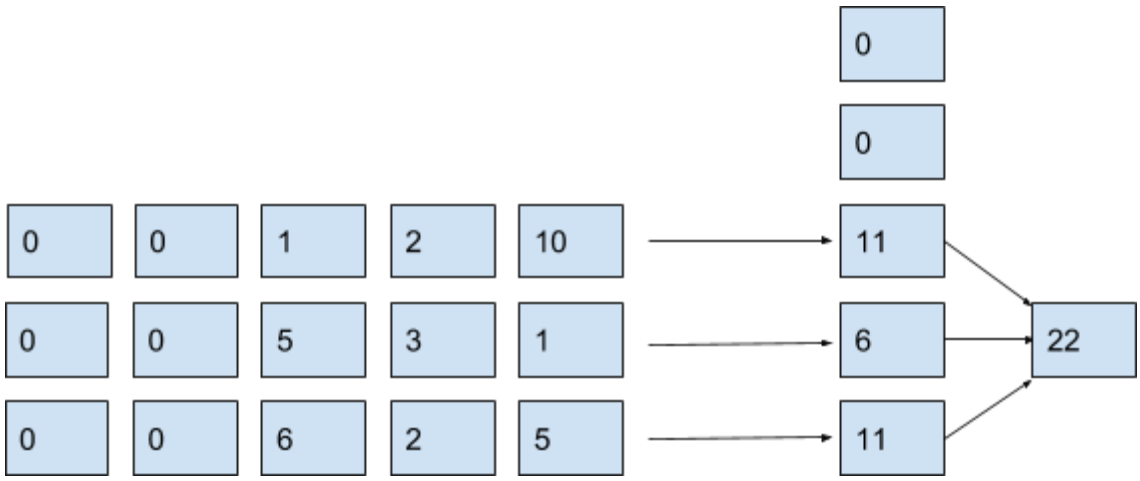

Figura I

A Figura I representa a solução do problema, temos a matriz construída, obtemos um vetor com os máximos de cada linha e por fim, o máximo da matriz por esse vetor. Observe os 0s

extras na matriz e no vetor, isso se deve a uma facilidade na visualização e execução da solução.

Desenvolvimento:

Na implementação foram utilizados dois TADs em junção com a main, um TAD foca na organização e realização da parte paralela do código que envolve as *pthreads*, a outra TAD, foca na manutenção dos parâmetros de entrada e faz a montagem do problema para a resolução. Além disso, foi criado uma estrutura de dados auxiliar, um pacote no formato de *struct,* que agrupa as entradas necessárias para o parâmetro de entrada que as *pthreads* utilizam.

Para o desenvolvimento, há três módulos, o da main, o de modelagem e entrada e o do paralelismo e solução. O da main é o administrador, ele faz a hierarquia das chamadas e coordena os demais, o de modelagem e entrada é o que pega os dados iniciais da entrada lidos na main, lê os demais necessários e faz a modelagem da matriz a ser utilizada para resolver o problema, por fim, o de paralelismo e solução utiliza a modelagem já feita e trabalha em cima dela usando *pthreads* para achar o máximo de cada linha e no final, o máximo da matriz, solução do problema.

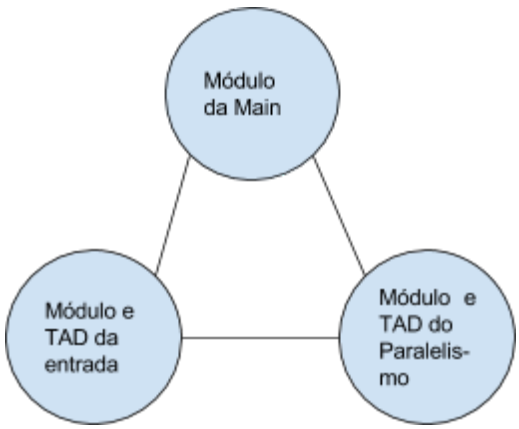

Figura II

A Figura II representa a relação de dependência entre os módulos do programa, note que nenhum dos módulos é independente, por exemplo, o módulo do paralelismo depende da criação e modelagem do problema em uma matriz, algo feito pelo da entrada.

A entrada do programa consiste em valores inteiros, em particular, deve-se ressaltar o uso do *argv*, pois através dele se obtém a informação do número de *threads* que podemos utilizar, além disso os inteiros na enteada consistem em dois, m e n, linhas e colunas, respectivamente, da dimensão da matriz, e em seguida, temos m linhas com n inteiros, correspondentes a população de cada cidade presente na matriz.

A saída do programa consiste em um único inteiro, que corresponde a população máxima que se pode obter na matriz dadas as restrições do problema. Ambas saída e entrada estão no padrão 'stdin' e 'stdout' do C.

O encontro do máximo de cada linha e do máximo da matriz se baseia através de um cálculo de máximo para cada valor da linha, pega-se dois valores anteriores soma-se ao valor que está sendo olhado e compara-se com o valor anterior, o maior entre eles se torna o valor que estava sendo olhado, em forma de equação:

v[i]=máximo(v[i]+v[i-2], v[i-1])

Ao final de fazermos isso, o máximo vai se encontrar na última posição da linha.

Para o auxílio dessa execução, a matriz montada na entrada possui duas colunas extras preenchidas com 0s, essa escolha foi feita, pois auxilia na visualização e execução do cálculo dos máximos.

Para se executar corretamente o programa, primeiro utilize o comando make, assim fazendo uso do makefile para compilar, em seguida, para executar, deve se digitar no terminal apenas: ./exec #

em seguida entre com os dados de entrada, de acordo com o especificado acima.

obs: Substitua o # por um número, ele indicará o número de *threads* que podem ser utilizadas.

Analisando a Complexidade:

Tempo:

A criação da matriz de entrada é da ordem de n x m, O(n*m), sendo m e n as dimensões da matriz, a execução da função de cada uma das *pthreads* é da ordem de O(n), sendo n o número de colunas da matriz, ou seja de elementos em cada linha, a função que cria as *pthreads* é da ordem de O(m*(n+t)), m sendo o número de linhas e n o de colunas, ou seja, chama até m *pthreads* que executam em O(n) e t sendo o número de *pthreads* atual que passam pelo fechamento(*join)*.

No escopo dessa análise, note que a execução fica mais rápida a medida em que temos mais *threads* possíveis para usar, entretanto ela atinge um limite ao chegarmos em exatamente uma *thread* para cada linha visto que qualquer número que exceda esse, é irrelevante, pois dispõe de *threads* extras que nem serão usadas.

Gráfico I

No Gráfico I, observamos o aumento de tempo de execução em função do programa todo(leitura+modelagem+resolução) e em função apenas da parte de resolução. Observa-se que a parte de leitura e modelagem da matriz é significativamente responsável pelo aumento no tempo de execução do programa. Neste gráfico, o número máximo de *threads* foi mantido constante em dois.

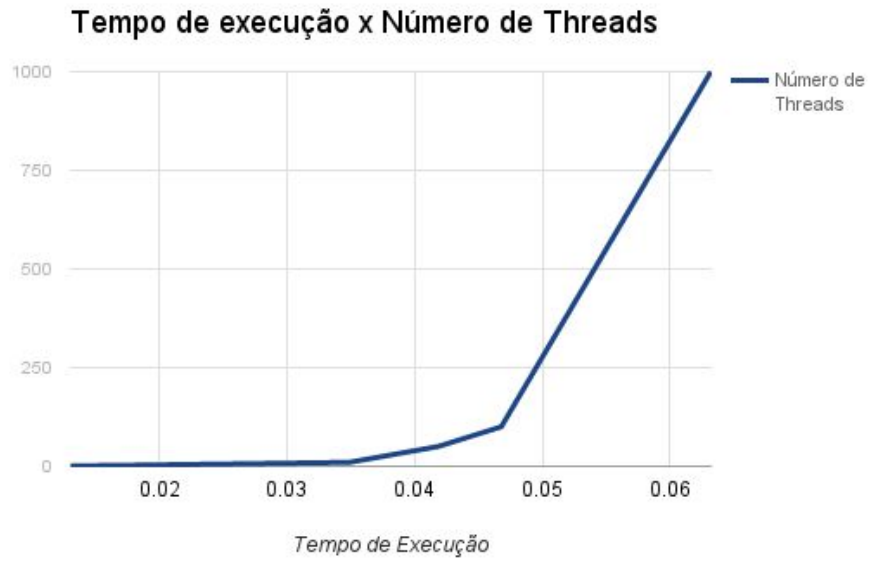

Gráfico II

No Gráfico II observamos um comportamento anormal, com o aumento do número de *threads* o tempo de execução aumenta, entretanto isso se deve ao computador onde foram realizados os testes, ao observar o processo em execução apenas uma thread por vez era executada, o que fazia ocorrer tempo de espera entre *threads*. Nesse gráfico, as dimensões foram mantidas constantes em 1000x1000 para a matriz de entrada.

Todos os testes e gráficos de tempo de execução foram realizados e montados utilizando a biblioteca Time.h do C e suas respectivas funções.

Espaço:

O espaço depende de primeiramente o tamanho da matriz utilizada O(n*m) e segundamente do número de *threads* O(t), sendo t o limite de *threads* que pode-se ter . Portanto é da ordem de O(n*m + t).

Conclusão:

O uso do paralelismo é uma ferramenta forte na execução de programas, entretanto seu uso não pode ser excessivo ou aleatório, deve-se notar que dependendo de excesso de paralelismo e do próprio computador no qual se executa os testes, o uso de *threads* pode acarretar em um tempo de execução maior, por exemplo, ter partes desnecessárias integradas no paralelismo pode causar *slow down* e o caso em que o computador execute apenas um dos processos paralelos por vez, como foi o caso no Gráfico II, devido a espera entre *threads*.

Referências:

https://www.tutorialspoint.com/c_standard_library/c_function_clock.htm
http://timmurphy.org/2010/05/04/pthreads-in-c-a-minimal-working-example/
https://virtual.ufmg.br/20162/pluginfile.php/251291/mod_resource/content/3/pthreads.pdf