Trabalho Prático 3: Plano de Dominação Global do professor WM. Jr
Universidade Federal de Minas Gerais

Aluno: João Lucas de Faria Gonçalves
Matrícula:2013048925

31 de janeiro de 2017

1 Introdução

O Professor WM. Jr quer se tornar o líder supremo do planeta e para isso ele tem um plano para dominar todas as cidades. Ele conta com a ajuda dos mestres do mal *CDFDR*, para executar o seu plano. O plano consiste em, os mestres do mal atacarem as cidades usando uma arma de manipulação mental e assim dominar a cidade transformando a população em seguidores do professor WM. Jr. Porém os ataques tem efeitos colaterais, toda vez que uma cidade a ser dominada é atacada algumas cidades próximas explodem. Você foi convidado a fazer uma solução computacional para o problema de escolher as cidades, visto que é necessário priorizar as escolha pois a destruição da arma não é tão intuitiva e é preciso maximizar o número de seguidores. Sabendo da grande quantidade de cidades(algo próximo de *9*10ˆ21*) e da disponibilidade de um laboratório com maquinas bem potentes com vários núcleos de processamento, o professor pediu para você usar paralelismo em seu programa para melhorar o seu tempo de execução devido ao grande número de cidades.

2 Modelagem

Os dados do problema foram modelados de acordo com a especificação, a representação das cidades como uma matriz *NxM*, onde cada elemento representa uma cidade e seu valor a população. Como no seguinte grid:

9	1	1	9	10
1	10	1	4	8
10	1	4	1	2
9	1	1	9	3
5	3	2	8	7

Na representação foi usado matriz de números inteiros alocada dinamicamente de acordo com as dimensões do grid de entrada do programa.

3 Solução do Problema

A solução apresentada foi o uso de programação dinâmica e paralelismo, como proposto na especificação. O problema foi modelado em subproblemas de soluções ótimas locais, encontrar a população máxima em cada linha de acordo com a politica de dano da arma. A segunda etapa da solução é a escolha das linhas de acordo com o valor máximo de população em cada linha já encontrado levando em consideração o dano causado pela arma. A escolha é feita de forma a tentar maximar a população final. Abaixo, temos um exemplo da politica de dano para a escolha da cidade que está no Grid[3][3] = 4.

Tabela 1: Politica de destruição da arma.

9	1	1	9	10
1	10	1	4	8
10	1	**4**	1	2
9	1	1	9	3
5	3	2	8	7

O problema foi divido em duas etapas, na primeira encontrar o máximo de cada linha do Grid e na segunda etapa escolher as linhas de acordo com os máximos de cada linha encontrados na primeira etapa.

A sub-solução ótima do problema pode ser vista como o máximo alcançado em cada linha, sendo assim, a programação dinâmica se baseia na seguinte relação de recorrência.

$$M[i] = \begin{cases} M[i] = max(M[i-1], Grid[l][i] + M[i-2]) & \text{se } i > 1; \\ \\ M[0] = Grid[l][0]; & \\ M[1] = max(Grid[l][0], Grid[l][1]); & \text{se } i \leq 1. \end{cases}$$

.

M[i...n] = vetor de tamanho n para memorizar os resultados já calculados que serão usados nas iterações seguintes do algoritmo.
l = linha ao qual desejamos calcular o máximo.
max() = calculo o máximo entre as duas posições.
Ao final do processo o máximo se encontra na posição M[i-1].

Apos calcular todos os maximos eles são salvos em um vetor maxlinhas de tamanho N onde cada posição *(maxlinhas[i])* esta associada ao máximo da linha (*Grid[i][]*) do Grid.

A segunda etapa da solução segue a ideia de solução já vista na primeira etapa. A solução depende apenas dos máximos obtidos na primeira parte da programação dinâmica, com isso aplicamos a relação de recorrência abaixo no vetor de sub-soluções ótimas maxlinhas.

$$M[i] = \begin{cases} M[i] = max(maxlinhas[i-1], maxlinhas[i] + M[i-2]) & \text{se } i > 1; \\ \\ M[0] = maxlinhas[0]; \\ M[1] = max(maxlinhas[0], maxlinhas[1]); & \text{se } i \leq 1. \end{cases}$$

.
Logo, o máximo possível no Grid está armazenado em M[N-1].

4 Análise de Complexidade:

A análise de complexidade foi realizada em dois aspectos distintos: tempo e espaço.

Análise de Tempo

Levando em consideração a não contagem do tempo para fazer a leitura da entra que é N x M, o algoritmo na primeira etapa faz (N x M) operações e na segunda etapa N operações, logo a complexidade de tempo é *O((N x M)+ N)*

Análise de Espaço

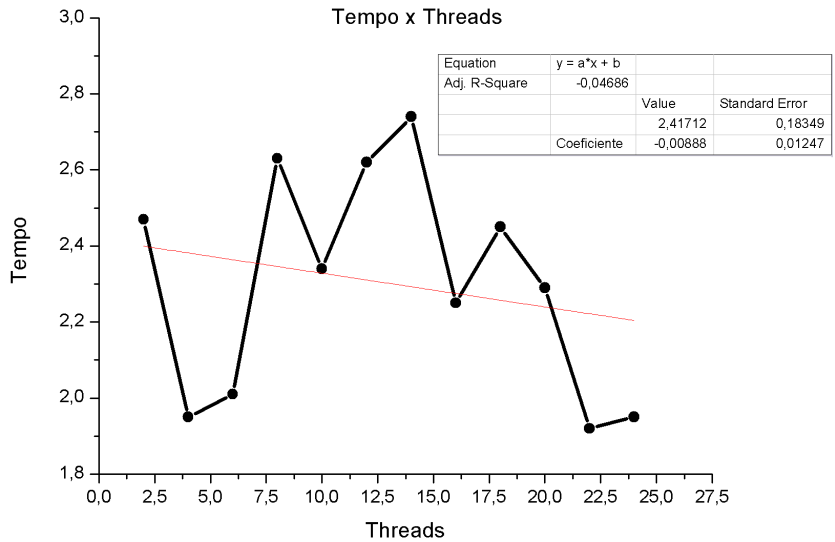
A complexidade espacial é calculada de acordo com o tamanho do Grid*(N x M)* mais o vetor de máximos de cada linha de tamanho (N), logo a complexidade de espaço é *O((N x M+1)*

5 Computação Paralela

A computação paralela foi usada para calcular os máximos das linhas , sendo assim cada Thread recebe um intervalo de linhas pare executar a tarefa de encontrar o máximo das linhas daquele intervalo. Para definir o tamanho dos intervalos foi dividido o número de linhas pelo o número de threads (*Linhas/Nthreads*) e o resto dessa divisão é distribuído entre as threads buscando deixar elas o mais balanceado possível.

6 Analise Experimental

Para medir o tempo de execução do código, foi utilizado a biblioteca (time.h). Os teste foram realizados em um Macbook Pro com processador 2,4 Ghz Intel Core 2 Duo e 4GB de memoria RAM.
Os testes foram realizados com uma entrada de dimensão 10000x10000 e as threads foram variadas de dois em dois até completar 24 threads. Abaixo temos um gráfico linearizado dos resultados obtidos.



7 Conclusão

De acordo com o gráfico e resultados obtidos, podemos concluir que o uso das múltiplas threads melhora de maneira significativa a complexidade do programa.