TP3: Plano de Dominação Global do Professor W.M. Jr.

Rúbia Reis Guerra
rubia-rg@ufmg.br

2016/02

1 Introdução

O problema proposto exige o cálculo da maior população possível a ser dominadas a partir de um grid de cidades, com a condição de que quando se toma uma cidade, todas as outras das linhas superior e/ou inferior e posicionadas à esquerda e/ou à direita são eliminadas.

71	91	63	79	31	01	92	21	43	69
12	80	65	74	36	84	54	89	52	11
73	93	90	81	33	03	44	23	45	68
04	34	67	70	22	48	56	02	32	13
75	95	46	83	35	05	82	25	47	57
94	06	98	24	14	64	58	53	72	15
77	97	26	85	37	07	16	27	49	61
28	50	00	30	20	62	60	55	66	17
10	40	59	87	39	09	96	29	51	41
08	18	42	38	76	99	88	78	86	19

Figura 1: Exemplo de grid.

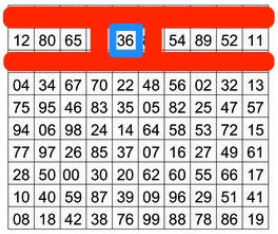

Figura 2: Exemplo de uso da arma de dominação.

No presente trabalho, propôs-se a solução do problema envolvendo programação dinâmica e técnicas de paralelização por *pthreads*.

2 Análise da Solução

2.1 Solução por Programação Dinâmica

Inicialmente, é importante observar as seguintes propriedades:

- Se dominar uma cidade C1 ocasiona na destruição de C2, logo dominar a cidade C2 ocasiona na destruição da cidade C1.
- Se dominar uma cidade C1 não ocasiona na destruição de C2, então dominar a cidade C2 não ocasiona na destruição da cidade C1.

Essas propriedades são importantes pois indicam que a ordem com que as cidades são dominadas não contribui para o resultado final.

Observa-se, ainda, que encontrar a solução para o problema constitui em resolver subproblemas mais simples e, em seguida, expandir a solução: dada uma sequência de cidades, calcula-se o número máximo de pessoas que podem ser dominadas, sabendo-se que dominar uma cidade destroi as cidades adjacentes. Isto é equivalente a otimização de uma linha no problema original.

Para encontrar o maior número possível de seguidores em uma fileira do grid, pode-se assumir que a dominação das cidades ocorre da esquerda para a direita, visto que a ordem não contribui para a solução final.

Utilizando programação dinâmica, seja **n** o número de células na linha, P_i o número máximo de seguidores que podem ser dominados se somente as cidades de i a $n-1$ estão disponíveis, e C_i a população em uma cidade i . Pode-se determinar P_i recursivamente da seguinte forma:

- O primeiro caso ($i \geq n$) é o caso-base, o que acontece quando se tenta tomar o máximo de cidades em uma linha vazia. Como não há nenhuma cidade para dominar, o valor de retorno deve ser 0.
- O segundo caso ($i < n$) é o único caso recursivo, que ocorre quando há ainda cidades para tomar. Há duas opções: dominar a cidade e passar para a próxima, ou não dominar a cidade e avançar para a próxima caixa. Entre estas duas opções, deseja-se escolher a que permita a maior quantidade de seguidores.

Uma vez que o problema tenha sido resolvido para uma linha, basta estender a solução para toda a tabela. É importante notar uma vez tomou uma cidade em uma fileira, a solução ótima consiste em maximizar a quantidade de pessoas dominadas dessa linha. Esta propriedade permite a criação de uma solução mais simples: em primeiro lugar, o número máximo de pessoas que podem ser dominadas em cada linha é calculado e, em seguida, (usando o mesmo algoritmo) o melhor conjunto de linhas a dominar é calculado. Assim:

- Inicializar os dois casos de base com o valor de retorno 0.
- Processar os estados a partir do menor para o maior, ou seja: a partir de $n-1$ a 0.
- Para cada estado, calcular o valor de P_i usando o caso recursivo. Os valores de P_{i+1} e P_{i+2} pode ser diretamente tomandos do arranjo, uma vez que os índices superiores já foram calculados.

- Depois de processar todos os casos, retornar o resultado de otimizar ao longo de toda a linha, ou seja: F_0.

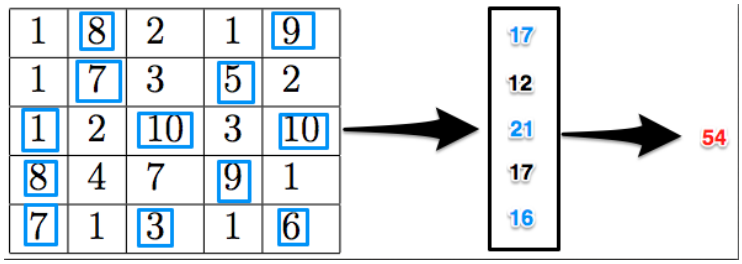

Figura 3: Exemplo de grid resolvida.

2.2 Análise de Complexidade

Como cada linha é iterado apenas uma vez e todas as colunas são iteradas, a complexidade final do algoritmo é $O(m * n)$ (sendo n o número de colunas e m, o número de linhas).

A complexidade de espaço necessária $O(n)$, sendo n o maior valor entre o número de linhas e de colunas.

2.3 Paralelização

A paralelização de dados foi implementada, conforme requisitado na documentação fornecida, utilizando-se da biblioteca *pthread.h*. A base escolhida para a solução foi o modelo *fork-join* com um *mutex* para sincronizar o acesso de leitura de cada thread. Foram paralelizadas as tarefas de leitura de dados e o cálculo das somas máximas por linha do grid. Por facilidade e controle da implementação, definiu-se que o número de threads criadas pelo programa nunca é superior ao número de linhas (N) do grid.

3 Testes de desempenho

Para verificar a validade das análises de complexidade, foram realizadas três baterias de testes alternando as dimensões da matriz (N e M) e a quantidade de threads a serem executadas. Os testes foram executados sobre matrizes quadradas de tamanhos 10x10, 20x20, 100x100, 500x500 e 1000x1000, respectivamente.

Observou-se, a partir dos tempos de execução, o comportamento esperado a partir da análise de complexidade do algoritmo. Porém, nota-se que com o aumento de threads, não há mudança considerável no tempo de execução. Isso se deve, provavelmente, à implementação ineficiente da técnica de paralelização e aos trechos seriais do algoritmo.

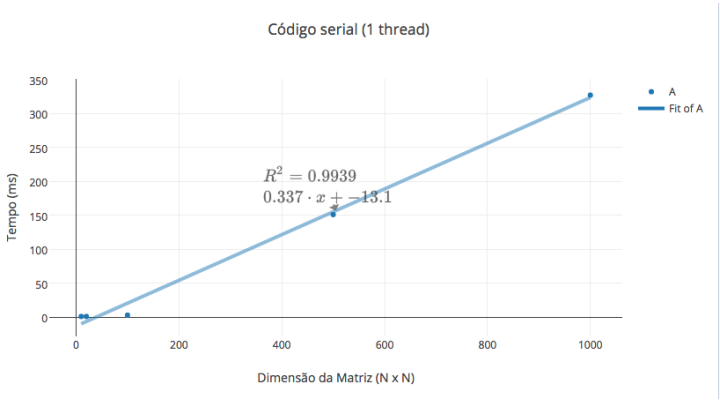

Figura 4: Tempo de execução para 1 thread.

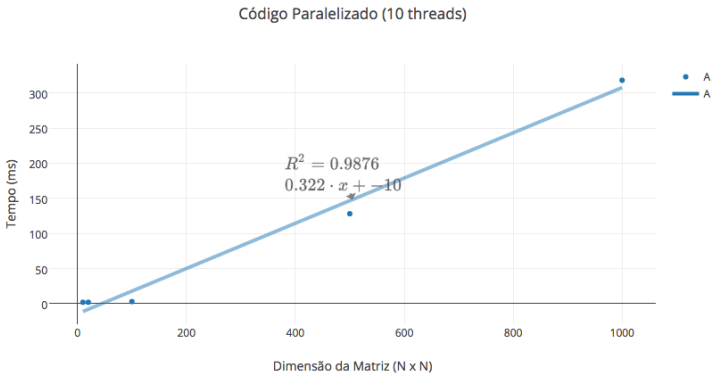

Figura 5: Tempo de execução para 10 threads.

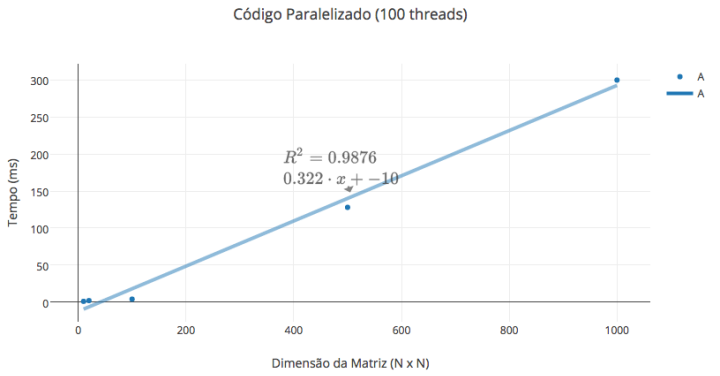

Figura 6: Tempo de execução para 100 threads.

4 Conclusão

Neste trabalho, foi proposto o estudo da solução de problemas por meio de paradigmas de programação. O problema foi resolvido adotando-se solução em programação dinâmica contendo um caso base e um caso recursivo. Ainda, foram aplicadas técnicas de paralelização utilizando a biblioteca *pthread.h*. Ao final, a análise das complexidades teóricas dos procedimentos do projeto foram confirmadas por experimentos utilizando parâmetros adequadamente dimensionados para permitir a observação do comportamento assintótico.

Referências

[1] R. Sedgewick and K. Wayne. *Algorithms*. 4th Edition, Addison-Wesley, 2011.

[2] U. Mamber. *Introduction to Algorithms*. 1st Edition, Addison-Wesley, 1989.

[3] IEEE. Pthreads.
$http://pubs.opengroup.org/onlinepubs/009695399/functions/pthread_join.html$

[4] Randu. Threads.
`https://randu.org/tutorials/threads/`