Universidade de São Paulo
Instituto de Física de São Carlos

SFI5822 - Introdução à Programação Paralela (2024)
Prof. Gonzalo Travieso

Trabalho B - Paralelo (Grafos)

Lucas Constante Mosquini - **NºUSP** 11858258

December 23, 2024

Contents

1 Introdução **2**

2 Proposta de paralelização **2**

3 Desempenho **3**

4 Resultados **3**

1 Introdução

Como descrito no projeto, receberemos um grafo com direção e sem peso. Também receberemos dados sobre ele, como número de arestas e vértices. Assim, dado a literatura, temos que calcular a eficiência do mesmo, e imrprimi-la, junto ao tempo de cálculo.

Como diretrizes principais: respeitar boas práticas, como visto em aula, ou seja, evitar usar estruturas complexas como árvores de armazenamento, mesmo que elas tenham um desempenho superior. Outro ponto importante, é que nos foi dito que todos os grafos usados seriam esparsos, o que influenciou a não adesão a vários métodos para gráfos densos.

Por fim, esse pdf se resume em justificar as escolhas visando o desempenho, assim como explicar o funcionamento e ideias bases por trás do código.

2 Proposta de paralelização

Nesta secção, vamos explorar as principais diferenças entre os algortimos sequencial x paralelo, assim como a proposta teórica de paralelização.

Primeiramente, como apontado na correção do trabalho anterior, houve a mudança envolvendo o uso de listas ligadas para a representação de adjacências, embora eficiente em termos de memória para grafos esparsos, tem impacto negativo no desempenho devido à baixa localidade de referência e ao mau uso da memória cache. Assim, a proposta para corrigir isso no código paralelo foi usar uma estrutura baseada em arrays, como uma combinação de arrays de início e de arestas (CSR - Compressed Sparse Row). Essa estrutura reduz a necessidade de alocação dinâmica, elimina o uso de ponteiros e melhora a localidade de dados, aumentando o desempenho.

No caso sequencial, um único processo percorre todos os vértices do grafo, realizando buscas em largura (BFS) de forma iterativa para calcular as distâncias e a eficiência. Já no caso paralelo o conjunto de vértices é dividido entre os processos MPI. Cada processo é responsável por calcular uma parte da eficiência, correspondente a um subconjunto de vértices. A função computePartialEfficiency realiza os cálculos localmente em cada processo. A operação $MPIReduce$ é utilizada para agregar os resultados parciais em um processo mestre (rank 0).

Para a distribuição de dados entre os processos, o processo mestre (rank 0) carrega os dados do grafo e os distribui para os outros processos usando a função MPIBcast. As estruturas adjacencyOffsets e adjacencyEdges são transmitidas em broadcast para garantir que todos os processos tenham acesso às mesmas informações do grafo.

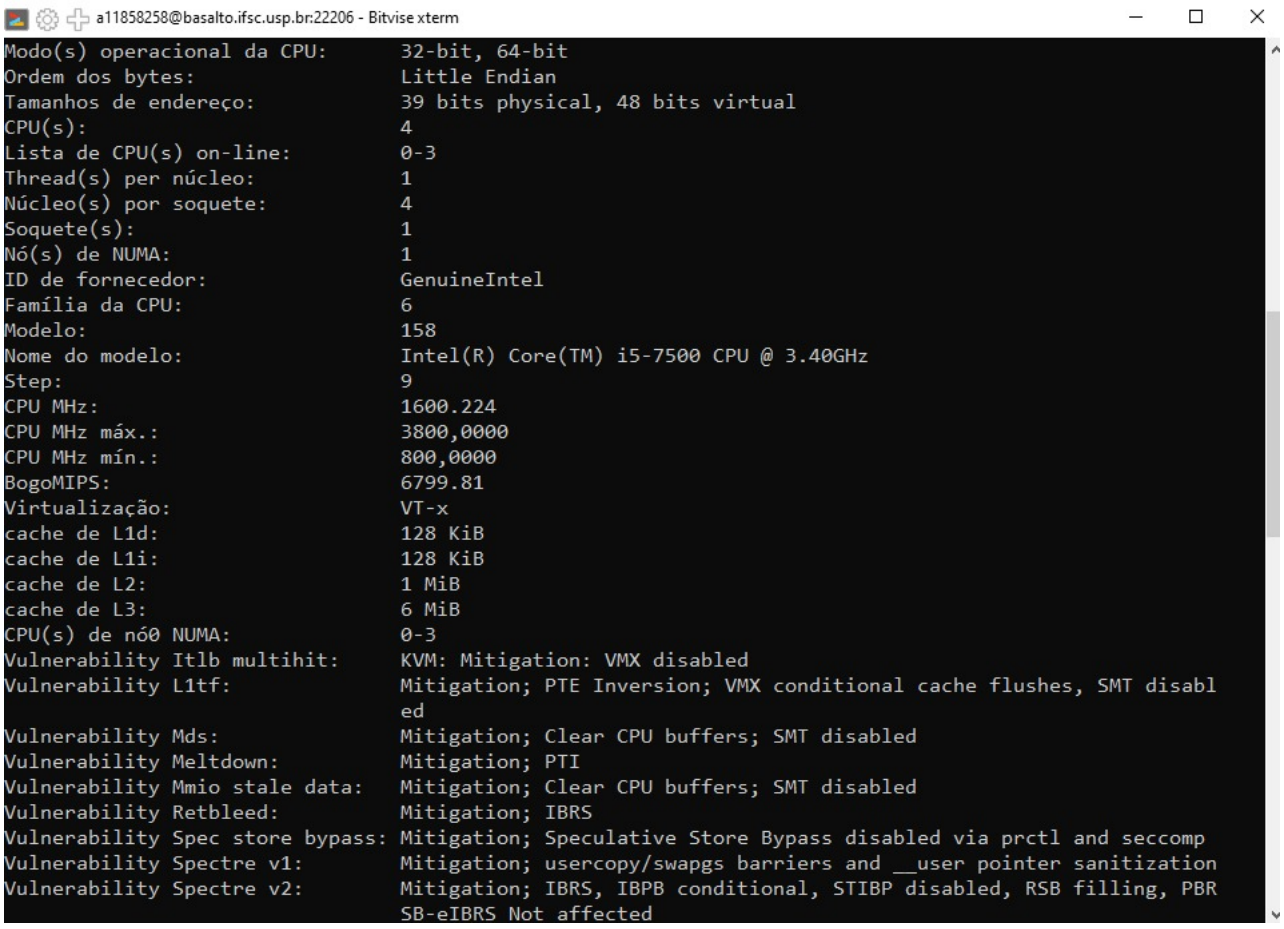
Para a distribuição estática de carga de tarefas, a carga é dividida igualmente (sempre que possível) entre os processos. Cada processo calcula a eficiência para um subconjunto de vértices, determinado pelo chunk size:

$$chunk_{size} = \frac{vertexCounter}{size} \tag{1}$$

Por fim, teoricamente o caso sequencial adequado para grafos pequenos ou um ambiente de execução único. Torna-se ineficiente para grafos grandes devido à complexidade O(V^2+VE), sendo V o número de vértices e E o número de arestas. Já o caso paralelo, reduz o tempo total de execução ao dividir a carga de trabalho, mas introduz sobrecarga de comunicação entre os processos (particularmente no broadcast e redução), o que pode afetar a eficiência para grafos pequenos ou em sistemas com alta latência de comunicação.

3 Desempenho

Para os cálculos, foi usado o processador abaixo:



Vale notar que na linha de saida, o tempo de cálculo sempre varia. Isso acontece por diversos fatores, como visto em aula: diferente tempos de latência entre outros hardwares com a CPU, localidades (principalmente temporal para as memórias na escrita e leitura), aquecimento e variação de frequência da CPU, distribuição de uso de outras aplicações e etc.

Assim, inicialmente é feito um "aquecimento" do código, rodando ele algumas vezes, com o intuito de minimizar esses efeitos. Em seguida, é tirado os valores da linha de saída 5 vezes.

4 Resultados

Como pedido no projeto, foram feitas as análises do tempo de cálculo em função de n e m.

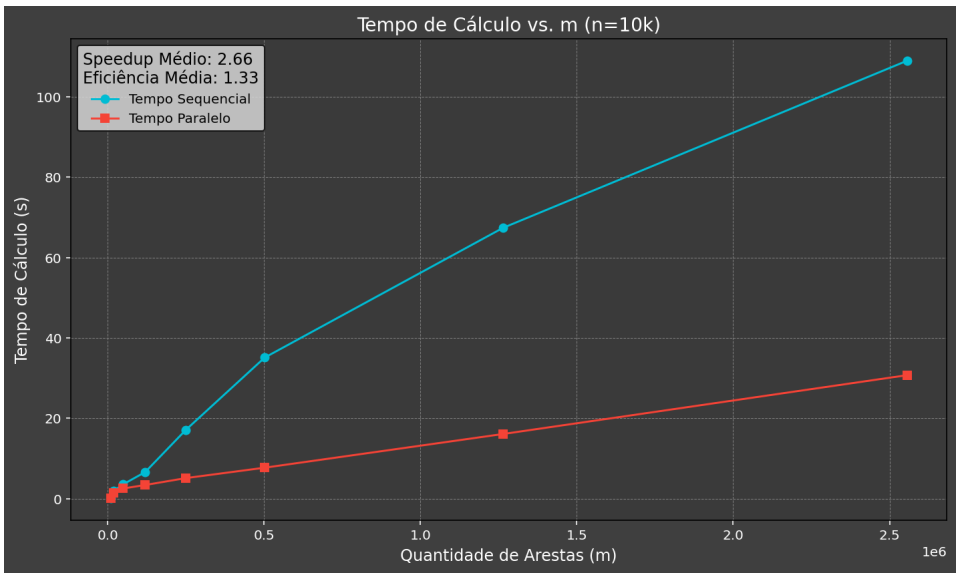


Figure 1: Variação do tempo de cálculo com m, mantendo n constante = 10k

No gráfico em que m (número de arestas) varia e n (número de vértices) é constante, o tempo de cálculo aumenta quase linearmente. Isso ocorre porque o algoritmo precisa percorrer todas as arestas em cada iteração da busca em largura (BFS). Para grafos esparsos, onde E $\sim$ O(V), a complexidade é dominada pelo número de arestas E, gerando um crescimento quase linear.

No caso paralelo, vemos uma redução significativa no tempo de cálculo. Essa redução ocorre porque a carga de trabalho (número de arestas a serem processadas) é dividida entre os processos MPI, aproveitando a capacidade de múltiplos núcleos. No entanto, como a quantidade de arestas aumenta, a sobrecarga de comunicação entre os processos torna-se menos relevante comparada ao tempo de computação. Para grafos muito grandes (valores altos de m), o tempo do algoritmo paralelo tende a ser limitado pela velocidade de comunicação e sincronização entre os processos MPI, mas a eficiência geral permanece alta para o intervalo apresentado.

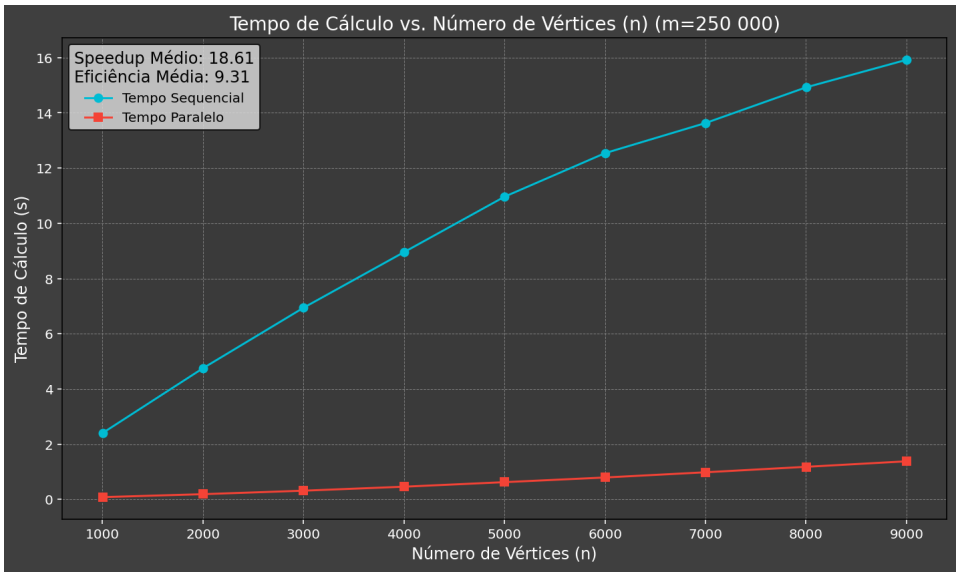


Figure 2: Variação do tempo de cálculo com n, mantendo m constante = 250 000.

n	Tempo Sequencial (s)	Tempo Paralelo (s)	Speedup	Eficiência
1001	2.402411	0.082006	29.27	14.64
2001	4.752294	0.188533	25.23	12.62
3001	6.941398	0.314826	22.06	11.03
4001	8.955147	0.459367	19.50	9.75
5001	10.962875	0.623737	17.59	8.80
6001	12.540526	0.792757	15.84	7.92
7001	13.631049	0.981457	13.90	6.95
8001	14.923049	1.179180	12.66	6.33
9001	15.921209	1.378570	11.55	5.77

Table 1: Resultados obtidos para diferentes valores de n, comparando os tempos de execução sequencial e paralelo, junto ao cálculo de Speedup e Eficiência. m constante = 250k

Quando n varia e m é constante, o tempo de cálculo aumenta proporcionalmente ao número de vértices. Isso é devido ao aumento de chamadas BFS necessárias, uma vez que cada vértice precisa ser visitado para calcular a eficiência global.

No caso paralelo, a redução do tempo é notória, especialmente para valores maiores de n. Isso é porque a carga é balanceada entre os processos, permitindo que mais vértices sejam processados simultaneamente. Em contraste, para valores pequenos de n, o tempo paralelo se aproxima do sequencial, devido à sobrecarga de comunicação e sincronização.

A eficiência e o speedup tendem a decair levemente para números maiores de n por conta do aumento proporcional do tempo necessário para operações de comunicação, como broadcast e reduce. O paralelismo supera o sequencial em todas as simulações. A introdução de estruturas de dados eficientes (como CSR no código paralelo) elimina o impacto de alocação dinâmica que prejudicava o desempenho do código sequencial. Assim, apesar da comunicação em paralelo ser uma limitação em grafos pequenos, para grafos maiores o paralelismo mostra sua força, compensando qualquer overhead inicial.