

## Objetivo

O objetivo deste trabalho prático é permitir o projeto e implementação de um algoritmo paralelo em uma arquitetura de memória compartilhada convencional (CPU multicore) utilizando o modelo de programação OpenMP, e comparar o tempo de execução dessa implementação com os tempos obtidos na implementação do mesmo algoritmo utilizando o modelo de programação Pthread.

## Descrição do problema

Neste trabalho vocês projetarão e implementarão algoritmos que solucionem o problema da **contagem de cliques em grafos não direcionados**. Por definição, um grafo  $G$  pode ser representado por um conjunto de vértices denotado  $V(G)$ , e por um conjunto de arestas  $E(G)$  que conecta vértices em  $V(G)$ . No nosso contexto, cada vértice de um grafo é identificado unicamente utilizando um inteiro no intervalo  $[0 \dots V(G) - 1]$ , e as arestas do grafo não possuem direção. A Figura 1 apresenta um grafo  $G$  conexo e não direcionado contendo 8 vértices e 11 arestas.

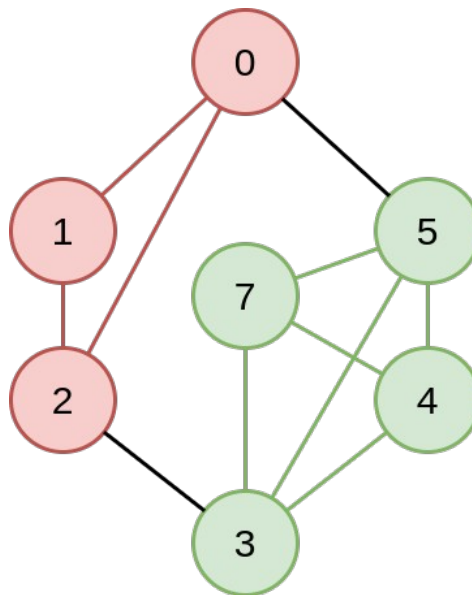


Figura 1. Exemplo de grafo conexo não direcionado com 2 cliques.

O grafo da Figura 1 é dito *conexo* pois, dado um vértice  $v$  qualquer do grafo, existe um caminho que conecta  $v$  a qualquer outro vértice do grafo. Os subgrafos da Figura que estão realçados em vermelho e verde são chamados de cliques, pois todos os vértices pertencentes a essas subgrafos estão conectados entre si. Uma  $k$ -clique é uma clique contendo  $k$  vértices. Ou seja, a clique em vermelho é uma 3-clique, enquanto a clique em verde é uma 4-clique.

Dado um grafo de entrada  $G$  conexo e não direcionado, você construirá algoritmos para contar a quantidade total de  $k$ -cliques em  $G$ . O pseudocódigo abaixo (estilo Python) apresenta um algoritmo genérico para a contagem de  $k$ -cliques em um grafo  $G$  conexo e não direcionado, utilizando a abordagem BFS (*Breadth-First Search*).

```

1. def contagem_de_cliques_serial(g: Grafo, k: int):
2.     cliques = []
3.     for each v in g.V(G):
4.         cliques.append([v])
5.
6.     contador = 0
7.     while not empty cliques:
8.         clique = cliques.pop()
9.
10.        // A clique atual já tem k vértices
11.        if (len(clique) == k):
12.            contador+=1
13.            continue
14.
15.        ultimo_vertice = clique[len(clique)-1]
16.        for each vertice in clique:
17.            for each vizinho in (adjacência de vertice):
18.                if (vizinho not in clique) and
19.                (vizinho se conecta a todos os vértices de clique) and
20.                (vizinho > ultimo_vertice):
21.                    nova_clique = clique.clone()
22.                    nova_clique.append(vizinho)
23.                    cliques.push(nova_clique)
24.
25.    return contador

```

Algoritmo 1. Contagem de cliques serial.

As linhas 2-4 constroem o conjunto de cliques visitadas pelo algoritmo, que é composto inicialmente por todos os vértices do grafo (dado que um subgrafo composto por apenas um vértice é uma clique). Enquanto o conjunto de cliques visitadas não for vazio (linha 7), uma clique é extraída do conjunto (linha 8). Caso essa clique extraída já tenha a quantidade de vértices  $k$ , a contagem de cliques é incrementada (linha 12) e o algoritmo continua a busca por uma próxima clique. Caso essa clique tenha menos que  $k$  vértices, o algoritmo explora as vizinhanças dos vértices da clique atual (linhas 16 e 17) e tenta criar uma clique maior (linhas 18-23). Caso haja um vértice novo que não faça parte da clique atual (linha 18) e que, ao ser inserido na clique atual gera uma clique maior (linha 19), o algoritmo aumenta a clique atual (linhas 21-22) e insere essa nova clique no conjunto de cliques a serem visitadas (linha 23). A linha 20 é uma condição necessária para que não sejam geradas cliques repetitivas durante a visitação.

## Metodologia

Neste trabalho você implementará duas versões do algoritmo de contagem de cliques. Todas as versões receberão um grafo de entrada  $G$  conexo não direcionado e um inteiro  $k$  que representa o tamanho da clique buscada. A primeira versão é uma versão paralela que distribui o conjunto inicial de cliques (linhas 2-4 do Algoritmo 1) utilizando schedules *static*, *dynamic* e *guided*. Você pode deve testar variações dos parâmetros associados a esses escalonamentos.

A segunda versão paralela não utilizará mecanismos de escalonamento do OpenMP. Você deve projetar tarefas paralelas independentes da mesma forma que você projetou a terceira estratégia paralela para a submissão anterior. Ou seja, você deve projetar uma distribuição de trabalhos inicial que seja fixa entre as threads, e quando uma thread fica sem trabalho a realizar, ela acessa o conjunto de trabalhos de uma outra thread e “rouba” um conjunto de  $r$  trabalhos. Os critérios utilizados para escolher qual(is) thread(s) terá(ão) seus trabalhos “roubados” ficam à sua escolha. Não se esqueça de utilizar um mecanismo de sincronização/seção crítica durante o acesso às tarefas de outra thread.

## Requisitos de Entrega

Neste trabalho você deve implementar e entregar os seguintes artefatos:

1. Implementação das duas versões paralelas propostas;
2. Conferência de todos os resultados produzidos pelas implementações dos algoritmos paralelos propostos em relação à corretude da quantidade de cliques geradas considerando os seguintes datasets:

<b>Grafo</b>	<b>k</b>	<b># Cliques</b>
citeseer	3	1166
	4	255
	5	46
	6	4
ca_astroph	3	1351441
	4	9580415
	5	64997961
	6	400401488
	7	2218947958
dblp	3	2224385
	4	16713192
	5	262663639
	6	4221802226

3. Descrição textual da estratégia adotada para escolher qual a thread que terá suas tarefas “roubadas”;
4. Análise dos tempos de execução dos algoritmos paralelos propostos nessa entrega, bem como a comparação desses resultados com as implementações propostas na entrega anterior (Pthread). Nessa análise você deve medir apenas o tempo de execução gasto para a geração das contagens de cliques, e não o tempo total de execução do arquivo executável

correspondente ao Algoritmo. Você deve variar o tamanho das cliques buscadas até que o programa chegue à um **tempo de execução máximo de 4 horas**.

5. Em relação à sua implementação da primeira versão paralela, você deve também prover uma análise experimental da variação dos parâmetros de escalonamento do OpenMP, bem como seus impactos nos tempos de execução finais;
6. Em relação à sua implementação da segunda versão paralela, você deve também prover uma análise experimental variando o valor de  $r$ . Nesse experimento você deve tentar encontrar o ponto em que o valor de  $r$  deixa de prover ganhos de desempenho;
7. Arquivo .zip contendo todos os códigos implementados;
8. Relatório em PDF contemplando os itens 3, 4, 5, e 6.

## ***Entrega***

Todos os artefatos descritos no item anterior devem ser entregues até o dia 04 de Novembro de 2024, via AVA.