Instituto Tecnológico de Aeronáutica

CE-265: Processamento Paralelo

Exercício 12: Contar triângulos

Bruno Benjamim Bertucci - Turma 23.2

1 Contagem de triângulos com *GraphBLAS*

A implementação do algoritmo de contagem de triângulos em grafos, feita usando a API *GraphBLAS* para a linguagem C, envolve uma operação entre uma matriz triangular inferior L e uma matriz A, constituída pela soma de L e de sua transposta, tal que $A = L + L^T$, de tal forma que A é simétrica, representando assim um grafo não-orientado.

Essa operação é representada como $C = A + .* L$, onde C é a matriz que armazenará o resultado dela. Além disso, a operação passa por uma *mask* dada pela matriz A, ou seja, ela é aplicada somente nas posições de C em que há um elemento da matriz A nessas mesmas posições.

Finalmente, é criado um vetor de mesmo tamanho que a dimensão da matriz C, e armazena-se nele a soma dos elementos de cada linha de C. A soma dos elementos desse vetor, por sua vez, resulta no número de triângulos desejado. O código referente a essa implementação encontra-se no Anexo A.

Uma outra abordagem para essa contagem é a realização da operação $D = A + .* A + .* A$, na qual o mesmo vetor construído na abordagem anterior é diretamente dado pela diagonal dessa matriz D. O método implementado apresenta vantagens com relação a essa outra forma. Percebe-se que, devido ao uso da *mask*, a matriz final não deixa de ser esparsa, e portanto a quantidade de operações necessárias para obtê-la se mantém reduzida.

Enquanto isso, nessa outra abordagem citada, a matriz final não é esparsa, possivelmente tendo todas as suas posições preenchidas, e ao final, somente a diagonal dela é usada, o que implica que a maioria dos cálculos realizados não importa para o resultado que desejamos.

Está anexo abaixo a saída da implementação feita usando *GraphBLAS*, detalhando todas as matrizes calculadas durante a computação, bem como o vetor e o número de triângulos calculado.

```
Matrix: L =
[  -,   -,   -,   -,   -,   -,   -]
[  1,   -,   -,   -,   -,   -,   -]
[  -,   -,   -,   -,   -,   -,   -]
[  1,   1,   1,   -,   -,   -,   -]
[  -,   1,   -,   -,   -,   -,   -]
[  -,   -,   1,   1,   1,   -,   -]
[  -,   1,   1,   1,   1,   -,   -]
Matrix: A =
[  -,   1,   -,   1,   -,   -,   -]
[  1,   -,   -,   1,   1,   -,   1]
[  -,   -,   -,   1,   -,   1,   1]
[  1,   1,   1,   -,   -,   1,   1]
[  -,   1,   -,   -,   -,   1,   1]
[  -,   -,   1,   1,   1,   -,   -]
[  -,   1,   1,   1,   1,   -,   -]
Matrix: C =
[  -,   1,   -,   -,   -,   -,   -]
[  1,   -,   -,   1,   1,   -,   -]
[  -,   -,   -,   2,   -,   -,   -]
[  1,   1,   2,   -,   -,   -,   -]
```

```
[  -,   1,   -,   -,   -,   -,   -]
[  -,   -,   1,   -,   -,   -,   -]
[  -,   2,   1,   -,   -,   -,   -]
Vector: tri =
[  1,   3,   2,   4,   1,   1,   3]
Numero de triangulos = 15
```

2 Usando a matriz L como *mask*

Usando a matriz L como *mask* no lugar da matriz A, a operação será aplicada somente para os elementos abaixo da diagonal da matriz C. Com isso, ela será triangular inferior. Porém, como pode ser visto na saída acima, a matriz C correta que gera o vetor *tri* não é simétrica.

Isso implica que somente a sua componente triangular inferior não é capaz de gerar o mesmo vetor, ou seja, essa componente é incompleta. Dessa forma, não é possível usar a matriz L como *mask* para a contagem de triângulos.

Anexo A – Código fonte do programa

```
/*
 * This file is part of the GraphBLAS Tutorial materials,
 * Copyright (c) 2018 Carnegie Mellon University and Intel Corporation.
 * All Rights Reserved
 *
 * THIS SOFTWARE IS PROVIDED "AS IS," WITH NO WARRANTIES WHATSOEVER. CARNEGIE
 * MELLON UNIVERSITY AND INTEL CORPORATION EXPRESSLY DISCLAIMS TO THE FULLEST
 * EXTENT PERMITTED BY LAW ALL EXPRESS, IMPLIED, AND STATUTORY WARRANTIES,
 * INCLUDING, WITHOUT LIMITATION, THE WARRANTIES OF MERCHANTABILITY, FITNESS
 * FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY RIGHTS.
 *
 * Released under a BSD (SEI)-style license, please see LICENSE.txt for
 * full terms.
 *
 * DM18-xxx
 *
 * Authors: Scott McMillan, Timothy G. Mattson
 */

/**
 * @file triangle_count.c
 *
 * @brief A triangle counting implementation using GraphBLAS C API.
 */

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <GraphBLAS.h>
#include "tutorial_utils.h"

//****************************************************************************
/*
 * Dado L, a parte inferior triangular da matriz de adjacencia de um grafo nao
 * orientado, compute o numero de triangulos do grafo para cada vertice
 */
uint64_t triangle_count(GrB_Matrix L)      // M: NxN, triangular inferior, bool
```

```
{
  GrB_Index n;
  GrB_Matrix_nrows(&n, L);                    // n = # de vertices



  // Crie um monoide + (GrB_PLUS_T) para o dominio dos inteiros positivos
  // de 64-bit (UINT64)
  GrB_Monoid UInt64Plus;
  // Para insto, basta preencher os '...' abaixo com o operador adequado
  GrB_Monoid_new(&UInt64Plus, GrB_PLUS_UINT64, (uint64_t)0ul);

  // Crie um semi-anel para realizar os saltos
  GrB_Semiring UInt64Arithmetic;
  // Para isto, basta preencher os dois
  // campos '...' abaixo, respectivamente, com:
  // monoide (+) criado anterioremente
  // operador (*) para o tipo UINT64, respectivamente
  GrB_Semiring_new(&UInt64Arithmetic, UInt64Plus, GrB_TIMES_UINT64);



  // Modelo de descritor para transpor o
  // argumento 1 (segundo argumento) de uma operacao
  GrB_Descriptor desc_tb;
  GrB_Descriptor_new(&desc_tb);
  GrB_Descriptor_set(desc_tb, GrB_INP1, GrB_TRAN);
  // (GrB_TRAN) transpoe; (GrB_INP1) entrada 1

  // Construa a matriz A = L .+ L'
  // Para isto, descubra na documentacao,
  // como realizar a operacao elemento a elemento
  GrB_Matrix A;
  GrB_Matrix_new(&A, GrB_UINT64, n, n);
  GrB_eWiseAdd(A, GrB_NULL, GrB_NULL, UInt64Plus, L, L, desc_tb);
  pretty_print_matrix_UINT64(A, "A");



  // A computacao devera ser como sugerido no final dos slides sobre TC:
  // C<A> = A +.* L
  // Defina C como uma matriz com n x n elementos do tipo GrB_UINT64
  GrB_Matrix C;
  GrB_Matrix_new(&C, GrB_UINT64, n, n);

  // Realize a computacao
  GrB_mxm(C, A, GrB_NULL, UInt64Arithmetic, A, L, GrB_NULL);
  pretty_print_matrix_UINT64(C, "C");

  // Reducao para vetor
  GrB_Vector tri;
  GrB_Vector_new(&tri, GrB_UINT64, n);
  GrB_reduce(tri, GrB_NULL, GrB_NULL, UInt64Plus, C, GrB_NULL);
  pretty_print_vector_UINT64(tri, "tri");

  // Reducao para escalar
  // Computa a norma 1 da matriz resultante (# total de triangulos)
```

```
  uint64_t count;
  GrB_reduce(&count, GrB_NULL, UInt64Plus, C, GrB_NULL);

  GrB_free(&C);                          // C matrix no longer needed
  GrB_free(&UInt64Arithmetic);           // Semiring no longer needed
  GrB_free(&UInt64Plus);                 // Monoid no longer needed
  GrB_free(&desc_tb);                    // descriptor no longer needed
  // Libere os outros objetos opacos que faltam
  GrB_free(&A);
  GrB_free(&tri);

  return count;
}



int main(int argc, char **argv)
{

  // Grafo apresentando nos slides
  GrB_Index const NUM_NODES = 7;
  GrB_Index const NUM_EDGES = 12;
  GrB_Index row_indices[] = {0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 4, 4};
  GrB_Index col_indices[] = {1, 3, 3, 4, 6, 3, 5, 6, 5, 6, 5, 6};
  uint64_t num_triangles = 0;

  // Inicializa o contexto do GraphBLAS no modo bloqueante
  GrB_init(GrB_BLOCKING);

  // Defina L como uma matriz com NUM_NODES x NUM_NODES elementos
  // do tipo GrB_BOOL
  GrB_Matrix L;
  GrB_Matrix_new(&L, GrB_UINT64, NUM_NODES, NUM_NODES);

  // O algoritmo funciona apenas com grafos nao orientados.
  // Assim, o trecho abaixo representa apenas a parte inferior
  // triangular de uma matriz simetrica.
  for (GrB_Index ix = 0; ix < NUM_EDGES; ++ix)
  {
    if (row_indices[ix] > col_indices[ix])
      GrB_Matrix_setElement(L, true, row_indices[ix], col_indices[ix]);
    else
      GrB_Matrix_setElement(L, true, col_indices[ix], row_indices[ix]);
  }
  pretty_print_matrix_UINT64(L, "L");

  // Invocacao da funcao
  num_triangles = triangle_count(L);
  fprintf(stdout, "Numero de triangulos = %ld\n", (long int)num_triangles);

  if (num_triangles != 15)
    fprintf(stderr, "Valor errado!\n");

  // Abaixo, invoque as funcoes para finalizar o contexto
  // e liberar a memoria da matriz instanciada
  // ...
  // ...
```

```
  GrB_free(&L);
  GrB_finalize();

  return 0;
}
```