基本算法介绍

并行从数据文件读取数据，将图数据以CSR格式存储，使用三边确定一个三角形来统计每条边的支撑度，将支撑度从小到大的边依次扫描出来，并将其剥离，更新对应三角形其他边的支撑度，直至图中没有边。最后扫描最大支撑度及其边的个数，输出结果。

并行化设计思路和方法

1.读取文件并行化

将数据文件根据线程数分割为多个bucket，每个线程读取一个bucket，将读取的数据分别存放在线程本地空间，利用 `gcc` 的内置原子指令，多线程并行构图。

2.三角形统计并行化

三角形统计最外层循环是节点数组，因为存储边是双向的，所以会导致同一条边统计两次，所以在扫描u节点的邻节点时，从大于u节点的邻节点开始，避免统计两次。内部操作没有依赖关系，所以可以并行化处理。使用gpu并行化时，两层for对应一个线程，内层采用二分搜索查找。

3.扫描边支撑度并行化

扫描每条边的支撑度，并将符合条件的边加入队列中，本身就可以并行化，但每次边加入队列需要原子操作，代价较高。所以每个线程设置一个buffer，将符合条件的边加入buffer中，当buffer满了再加入队列中，同步操作从O(E)降至O(E/n)，其中n为buffer的size。使用gpu并行化时，两层for对应一个线程，内层采用二分搜索查找。

4.剥离边并行化

当剥离每条边时，更新对应三角形另外边的支撑度时要原子操作。在并行化处理边时，可能会存在某条边被更新两次，导致该边支撑度低于当前level，需要有恢复操作，将其更新至当前level。

算法优化

1.因为支撑度为0的边不存在三角形，所以在剥离该边时，并不会出现更新其他边的情况，所以可以直接跳过支撑度为0的边。

2.当扫描到最后一层时，剩余的边不会再更新其他边，所以可以直接跳过最后一层。

3.构建边列表中，在查找邻节点时，采用线性搜索和二分搜索，在邻节点超过64个时，使用二分搜索可以减少搜索的时间。

4.通过k-core来将度低于lower_k的点对应边删除，然后level从lower_k开始剥离边，加快剥离边的速度。

5.计算可能构成k-truss的子图，使其作为初始upper_k,比例m的upper_k作为初始lower_k，然后删除k-core低于lower_k的边，计算k_truss,将lower_k更新为结果。在以新的lower_k作为level开始计算kmax-truss，避免从0开始计算，加快计算速度。

详细算法设计与实现

1.图存储结构

图采用CSR格式存储，`adj`存储每条边的邻节点，`num_edges`存储节点在`adj`的起始位置，`edge_id`存储每条边对应的`eid`。

```
class Graph {
public:
    uint32_t n;         //number of vertices
    uint32_t m;         //number of edges

    vid_t *adj;         //adjacency array
    eid_t *num_edges;   //starting postion of the vertice N in the adjacency
array and edgeId array
    eid_t *edge_id;     //edgeId array
}
```

2.三角形统计

首先找到大于节点u的起始邻节点，扫描每个小于u的邻节点v，当v的邻节点中通过二分搜索找到w，表示找到三条边并形成一个三角形，将每条边的支撑度加一。

```
__global__ void tc_kernel(eid_t *dev_num_edges, vid_t *dev_adj, eid_t
*dev_edge_id, eid_t *dev_start_edge, int *dev_edge_support,
                          bool *dev_processed, vid_t n, int mul) {

    uint32_t b_id = blockIdx.x / mul;
    uint32_t b_stride = gridDim.x / mul;
    uint32_t t_id = threadIdx.x + (blockIdx.x % mul) * blockDim.x;
    uint32_t t_stride = blockDim.x * mul;
    for (int u = b_id; u <= n; u += b_stride) {

        for (eid_t j = dev_num_edges[u] + t_id; j < dev_start_edge[u]; j +=
t_stride) {
            vid_t v = dev_adj[j];

            for (eid_t k = dev_num_edges[v + 1] - 1; k >= dev_start_edge[v]; k-
-) {
                vid_t w = dev_adj[k];
                if (w <= u) break;
                int inx = binary_search_v2(dev_adj, dev_start_edge[u],
dev_num_edges[u + 1], w);
                if (inx != -1) {
                    eid_t e1 = dev_edge_id[inx], e2 = dev_edge_id[j], e3 =
dev_edge_id[k];
                    atomicAdd(&dev_edge_support[e1], 1);
                    atomicAdd(&dev_edge_support[e2], 1);
                    atomicAdd(&dev_edge_support[e3], 1);
                }
            }
        }
    }
}
```

3.扫描边支撑度

在每层扫描时，将支撑度小于等于当前level且没有处理的边加入队列中，等待剥离并更新其他边。

```
// Size of cache line
const long BUFFER_SIZE_BYTES = 2048;
const long BUFFER_SIZE = BUFFER_SIZE_BYTES / sizeof(vid_t);

vid_t buff[BUFFER_SIZE];
long index = 0;

uint32_t tid = threadIdx.x + blockIdx.x * blockDim.x;
uint32_t stride = blockDim.x * gridDim.x;

for (long i = tid; i < num_edges; i += stride) {
    if (dev_edge_support[i] <= level && !dev_processed[i]) {
        buff[index] = i;
        dev_in_curr[i] = true;
        index++;

        if (index >= BUFFER_SIZE) {
            long tempIdx = atomicAdd(dev_curr_tail, BUFFER_SIZE);

            for (long j = 0; j < BUFFER_SIZE; j++) {
                dev_curr[tempIdx + j] = buff[j];
            }
            index = 0;
        }
    }
}

if (index > 0) {
    long tempIdx = atomicAdd(dev_curr_tail, index);

    for (long j = 0; j < index; j++) {
        dev_curr[tempIdx + j] = buff[j];
    }
}
```

4.剥离边

使用二分搜索查找另外两条边，在处理cur数组中的边时，如果另外两条边任意一条处理过，说明已经不能形成三角形，跳过该两条边。如果另外两条边都未在cur数组中，那么更新两条边的支撑度。

如果有一条在cur数组中，那么索引小的处理三角形。

```
uint32_t b_id = blockIdx.x;
uint32_t b_stride = gridDim.x;
uint32_t t_id = threadIdx.x;
uint32_t t_stride = blockDim.x;

for (auto i = b_id; i < *dev_curr_tail; i += b_stride) {
    //process edge <u,v>
    eid_t e1 = dev_curr[i];

    Edge edge = dev_id_to_edge[e1];
```

```
        vid_t u = edge.u;
        vid_t v = edge.v;

        eid_t u_start = dev_num_edges[u], u_end = dev_num_edges[u + 1];
        eid_t v_start = dev_num_edges[v], v_end = dev_num_edges[v + 1];
        if ((u_end - u_start) > (v_end - v_start)) {
            swap(u_start, v_start);
            swap(u_end, v_end);
        }

        for (int j = u_start + t_id; j < u_end; j += t_stride) {
            int w = dev_adj[j];
            int inx = binary_search_v2(dev_adj, v_start, v_end, w);
            if (inx == -1)
                continue;

            eid_t e2 = dev_edge_id[j];  //<v,w>
            eid_t e3 = dev_edge_id[inx];//<u,w>

            bool is_peel_e2 = !dev_in_curr[e2];
            bool is_peel_e3 = !dev_in_curr[e3];

            if (is_peel_e2 || is_peel_e3) {
                if ((!dev_processed[e2]) && (!dev_processed[e3])) {
                    if (is_peel_e2 && is_peel_e3) {
                        update_support(e2, level, dev_edge_support, dev_next,
dev_in_next, dev_next_tail);
                        update_support(e3, level, dev_edge_support, dev_next,
dev_in_next, dev_next_tail);
                    } else if (is_peel_e2) {
                        if (e1 < e3) {
                            update_support(e2, level, dev_edge_support,
dev_next, dev_in_next, dev_next_tail);
                        }
                    } else {
                        if (e1 < e2) {
                            update_support(e3, level, dev_edge_support,
dev_next, dev_in_next, dev_next_tail);
                        }
                    }
                }
            }
        }
    }
```

实验结果与分析

实验机器规格：

```
CPU：8核
内存：64GB
GPU：1*Tesla V100（显存16GB）
cuda:11.1
```

实验结果：

图	k max	edge	Time/s
`com-orkut`	78	6,859	10.415
`s23.e15.rmat.edgelist`	685	1,327,787	8.672

程序代码模块说明

```
src/main.cpp        ---程序入口
src/kt.cu           ---计算kmax-truss
src/log.cpp         ---日志
include/util.h      ---程序的输入输出
include/graph.h     ---存储图结构
include/edge.h      ---存储边
include/timer.h     ---记录时间
```

详细程序代码编译说明

```
在程序主目录下make all
```

详细代码运行使用说明

```
./kt -f 数据文件路径
如需设置线程数：
OMP_NUM_THREADS=线程数 ./kt -f 数据文件路径
```