大规模图数据中的 kmax-truss 问题的求解和算法优化

自由之翼

蔡晓帆
计算机技术&研三
中国科学院信息工程研究所
中国-北京
caixiaofan@iie.ac.cn

团队简介

队长：蔡晓帆，毕业于北京理工大学计算机学院，现就读于中国科学院信息工程研究所，研究方向为计算机技术。曾获得的奖项有国际大学生程序设计竞赛（ICPC）区域赛的金牌、以及中国大学生程序设计竞赛（CCPC）区域赛的金牌、蓝桥杯 C/C++ A 组全国总决赛一等奖（第六名）。

本团队指导老师为来自中国科学院信息工程国家重点实验室的孙瑶研究员。

摘要

本赛题的任务是利用给定的 GPU 计算资源，设计高效的大规模图数据中 kmax-truss 问题的求解算法。我们团队对图的数据进行分析，提出了一套高效的两阶段求解算法，包含预测与搜索阶段，并且使用了并行去冗余算法大幅度提高了三角形计数性能、GPU 求交集的核心算法优化、并基于数据分布使用了大中小步的搜索策略等优化策略想结合，最终在复赛性能测试中排名第一。通过测试，我们的单数据集运行速度相比其他队伍至少提高 19%，并且在更大的数据集测试中表现出良好的线性复杂度，能够在 40s（不含读取时间）内处理总量达 11 亿的两个数据集，算法具有良好的泛化性和稳定性。

关键词

图数据处理，kmax-truss，三角形计数，并行计算，CUDA 编程

1 引言

本赛题任务为在给定 GPU 计算资源中，设计高效的 kmax-truss 求解算法，我们使用了一个两阶段求解算法，包含预测以及搜索阶段。预测阶段以一个较小的复杂度得到一个较为准确的 k 作为下界。

相比与传统的 truss 分解法，我们的算法能够有效的避免缓慢的逐步尝试操作。

我们认为我们的算法主要有以下三方面的创新点：

1) 预测：在预测阶段使用了 kmax-core 这个子图（小图），较为准确的预测了原图（大图）上的 kmax-truss 的大小。减少 95%以上的计算时间。
2) 去冗余：通过并行方法高效避免了三角形的冗余计数。减少近 85%的计算时间。
3) 快速搜索策略：利用数据集的数据分布特点，提出了大中小步相结合的 kmax-truss 搜索方法。计算时间相比二分减少 22.5%

	正确	性能	文档	总分
自由之翼	20	70	10	100
第二名团队	20	44.84897	9	73.84897
第三名团队	20	43.06085	10	73.06085
第四名团队	20	38.32793	9	67.32793
第五名团队	20	19.78923	10	49.78923

表 1：kmax-truss 复赛结果

官方的复赛结果如表 1 所示，我们的算法时间在各个单数据上性能均为第一名，并获得了 100 分的总分。

在之后的第二章节中我们将首先介绍符号与基础知识。在第三章介绍我们的算法，以及 3 个创新点，第四章节进行试验与分析，第五章节进行总结。

2 符号与基础知识

本章节主要介绍一些符号以及概念。包括 k-truss 以及 k-core 的概念，经典的计算 kmax-truss 的 truss 分解法。

2.1 符号与概念

K-truss 是由 Jonathan Cohen 于 2008 年提出的一种稠密子图结构，其要求每条边至少属于(k-2)个三角形。

K-core 是一种稠密子图结构，要求每个节点至少有 k 个邻居，k-core 是比 k-truss 更易于获取的一种子图，容易证明(k-1)-core 是 k-truss 的必要条件。

设点集合为V，有向边集合为E，顶点的数量为N，在并行三角形计数过程中，还未进行三角形计数的边集为E'。

符号$\Delta * v_m v_n$ 表示三角形集合$\{\Delta v_k v_m v_n | \forall v_k \in V, v_k < v_m < v_n, \overrightarrow{v_k v_m}, \overrightarrow{v_k v_n} \in E\}$。

符号$\Delta' * v_m v_n$ 表示已经进行计数的三角形集合$\{\Delta v_k v_m v_n | \forall v_k \in V, v_k < v_m < v_n, \overrightarrow{v_k v_m} \notin E', \overrightarrow{v_k v_n} \in E\}$。

符号$L' * v_n$ 表示集合 $\{\overrightarrow{v_k v_n} | \forall v_k \in V, \overrightarrow{v_k v_n} \in E'\}$，即以 v_n 为终点的还未进行三角形计数的边的集合。

符号$N(n)$表示集合，$\{v_k | \forall v_k \in V, \overrightarrow{v_n v_k} \in E\}$即以 v_n为起始的边的终点集合，即 v_n在有向图中的邻居。

2.2 Truss 分解法

根据定义 k-truss 的定义，我们只需要删除所有图中三角形数量不满足(k-2)的边便可以得到 k-truss。

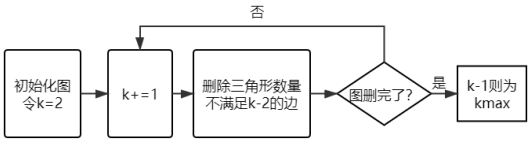

图 1：Truss 分解法求解 kmax-truss

因此如图 1 所示，我们只需要令 k 不断的递增，并且在递增过程中删除图中所有三角形数量不满足(k-2)的边，那么当图中的边都被删除时，上一轮的图就是 kmax-truss。这个算法被称作 truss 分解法。这个算法计算效率不高，主要原因是每次 k 值的步长仅加 1，需要缓慢的逐步递增以及依次删边。

3 两阶段的 kmax-truss 求解算法

3.1 算法动机与算法概述

Truss 分解法因为 k 的缓慢递增，算法性能缓慢，由 CPU 的分支预测功能所带来的灵感，如果能够利用较少的计算得到一个接近 kmax 的 k 值，便可以大大提高算法的性能。因此我们设计了两阶段的 kmax-truss 求解算法。

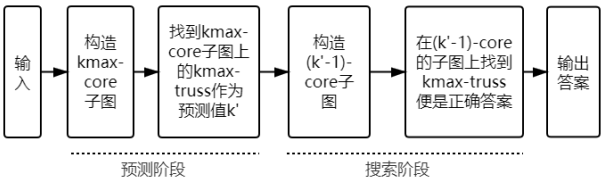

图 2：两阶段的 kmax-truss 求解算法

如图 2 所示，我们的算法分为预测和搜索两个阶段，预测阶段利用了 kmax-core 子图上的 kmax-truss 对于原图上的 kmax-truss 进行预测，避免了 truss 分解法总缓慢的逐步尝试，搜索阶段利用预测阶段搜索出来的 k 值作为下界，去除冗余的边之后，在原图上搜索出真正的 kmax-truss。

算法包含着快速的预测，三角形计数过程的去冗余计算以及根据利用数据集的数据分布特点提出的快速搜索策略这三大亮点。将在以下的子章节逐一介绍。

3.2 预测

预测阶段的目的是为了以一个较小的开销，较为准确的估计原图上 kmax-truss 的 k 值，以提高搜索阶段的速度。

我们发现了 kmax-core 和 kmax-truss 有紧密的联系。因此我们在多个数据集上对 kmax-core 上的 kmax-truss 以及原图上的 kmax-truss 进行比较。

数据集	kmax-core 上的 kmax-truss(点|边)	原图上的 kmax-truss(点|边)	差距 (点|边)
cit-Patents	36 | 2464	36 | 2625	0 | 161
soc-LiveJournal	362 | 72913	362 | 72913	0 | 0
s18.e16.rmat.edgelist	164 | 222935	164 | 225529	0 | 2594
s19.e16.rmat.edgelist	223 | 332566	223 | 334934	0 | 2368
s24.kron.edgelist	935 | 1856983	935 | 1856983	0 | 0
com-orkut	61 | 26620	78 | 6859	17
s23.e15.rmat.edgelist	685 | 1327787	685 | 1327787	0 | 0
graph500-scale25-ef16	996 | 1956901	996 | 1956901	0 | 0
Kron s25	1388 | 2654606	1388 | 2654606	0 | 0

表 2：kmax-core 上的 kmax-truss 与原图上的区别

由表 2 可见，在绝大多数数据集上，kmax-core 上的 kmax-truss 和原图上的 kmax-truss 的 k 值是相同的，仅有 com-orkut 数据集差别为 17，此差别并不大，而且综合多个数据集进行统计，kmax-core 仅占原图的大小的 3.4%。因此利用其对 kmax-truss 进行预测的时间开销远比在原图上直接计算开销低。

3.3 三角形计数-去冗余

定义边$\overrightarrow{v_m v_n} \in V$相关的三角形个数$Num(\overrightarrow{v_m v_n})$为：

$$Num(\overrightarrow{v_m v_n}) = |\Delta * v_m v_n| + |\Delta v_m * v_n| + |\Delta v_m v_n *|. \quad (1)$$

通过了预测阶段，算法已经有了一个较为接近 kmax-truss 的 k 值。如果已知 k，则需要删除所有三角形数量不满足 k-2 的边。综合多个数据集进行统计和分析，在搜索阶段将有 90%的边最终会被删除。如果其全部进行三角形计数，会引入大量的冗余计算。

如果能够在计数阶段，快速判断$Num(\overrightarrow{v_m v_n})$无法达到 k-2，则可以避免大量的冗余计算。

等式(1)分为$|\Delta * v_m v_n|$、$|\Delta v_m * v_n|$、$|\Delta v_m v_n *|$这 3 个部分之和，在下面将逐一介绍对其的估计方法。

其中$|\Delta v_m v_n *|$等价于$N(m)$与$N(n)$的交集，即：

$$|\Delta v_m v_n *| = |N(m) \cap N(n)|. \quad (2)$$

在利用 GPU 求等式（2）时，会将两个有序集合$N(a)$和$N(b)$进行求交集，在求交集的过程中，容易判断已经求得的交集大小以及剩余还未进行交集部分集合的大小，这部分的上界容易判断。

在三角形计数过程中，算法会将起始点相同的边调度给一个线程束，并且线程束会按照定点结尾顺序对这些边以此进行求交集运算，则在统计$\overrightarrow{v_m v_n}$的三角形时，$|\Delta v_m * v_n|$将已经计算完毕，此部分不需要估计上界。

根据定义得出

$$|\Delta * v_m v_n| <= |\Delta' * v_m v_n| + |L' * v_m| \quad (3)$$

其中$|\Delta' * v_m v_n|$是已经完成计数的三角形数量，不需要估计，而$|L' * v_m|$实际上是以v_m为终点的边的数量，并且还未以其为底，进行三角形计数的边数量。这个量只需要开N个全局计数器进行维护即可。

综上所述，我们能够利用调度以及使用N个全局计数器使得在并行统计三角形数量的过程中，能够快速的判断每条边的三角形的数量上界，从而可以避免大量的冗余计算。经过去冗余后，在多个数据集上进行试验三角形计数时间变为原来的 25%

不仅如此，去冗余三角形计数算法还可以使得在计数完成后，便会有大量的边的三角形数量不满足 k-2，进而能够在第一轮删除中同时将其进行删除，从而提高了删边速度。删边过程所需时间变为去冗余前的 27%。

3.3.1 三角形计数求交集算法的改进

为了适应去冗余操作，在求解等式(2)时，会将求交集的任务分配给一个线程处理。实际上等价于在一个线程束中计算两个有序集合的交集。而经典的 GPU 交集匹配算法是使用全连接来求交集的，这是为了适应 GPU 结构大量的冗余计算算法，完成一个长度为 32 的集合匹配需要 32 次读操作。

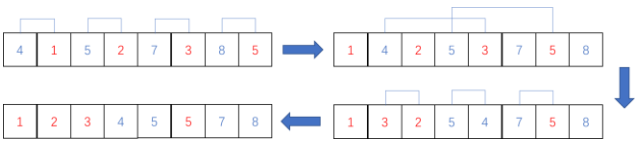

图 3：基于 Batcher 方法的合并交换排序[1]

我们的算法如图 3，我们将两个有序集合分别放在奇数位以及偶数位，之后使用了基于 Batcher 方法的合并交换排序来进行交集的求解，完成一个长度为 32 的集合匹配操作只需要 8 次读和 7 次写入，相比全连接的方法，时间开销变为原来的 78.8%。

3.4 快速搜索策略

在预测阶段，我们需要快速的找到 kmax，我们发现，在预测过程中，每当 k>kmax 时，处理较为耗费时间，因为此时需要将整个图的所有边都删掉，而且需要进行存档恢复操作，针对这样的数据分布特点，我们创新的提出了大中小步相结合的快速定位 kmax-truss 方法。

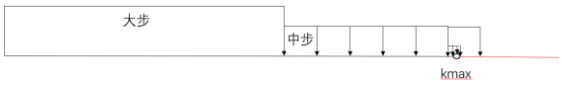

图 4：大中小步相结合的 kmax 搜索策略

如图 4 所示，详细算法为，大步走一次，之后走中步，中步走到 k>kmax 后，还原回 k>kmax 之前状态后走步长为 1 的小步。在大步不会导致 k>kmax 的情况下，大中小步法仅会固定发生 k>kmax 状态 2 次，比二分法要少，在多个数据集上进行统计，相比二分法可以节约 22.5%的计算时间。

4. 实验

4.1 测试环境

CPU：Indel Xeon E5-2650 v3（20 逻辑核心）、内存：64G、GPU： NVIDIA GeForce GTX 1080Ti （11G 显存）、操作系统：Ubuntu 20.04 LTS

4.2 正式比赛数据集测试

数据名称	s23.e15.rmat.edgelist	com-orkut
顶点数	8,388,608	3,072,626
边数	121,193,913	117,185,083
kmax-truss k 值	685	78
kmax-truss 边数	1,327,787	6,859
读取时间(s)	1.03	0.92
算法执行时间(s)	3.68	3.79
总时间(s)	4.71	4.39

表 3：复赛数据集的测试结果

4.3 更大规模的数据集测试

数据名称	s24	s25.ef16	Kron s25
顶点数	16,777,215	17,043,780	52,428,798
边数	260,379,850	523,467,448	611,860,695
kmax-truss k 值	935	996	1388
kmax-truss 边数	1,856,983	1,956,901	2,654,606
读取时间(s)	1.96	3.82	4.49
算法执行时间(s)	9.14	19.06	20.39
总时间(s)	11.10	22.88	24.88
边数较 com-orkut 数据集倍数	2.22	4.46	5.22
总时间较 com-orkut 数据集倍数	2.53	5.21	5.67

表 4：在更大的数据集上的测试结果

由表 4 可以发现，我们的算法能够随着数据集的增大，时间的增大是相对线性的，算法具有良好的稳定性以及泛化性。

5. 总结

我们的算法创新的采用了先预测，后搜索的两阶段算法，包含了预测，三角形去冗余计算以及快速搜索策略这 3 个亮点。最终在复赛的性能评测中排名第一。

除了之前讲过的创新点，算法还包括将大量函数迁移至 GPU，利用 GPU 高速的 share memory 存储临时结果，多线程读取以及 kmax-core 同样采取预测结构进行计算等实现优化。

我们的算法还包含了关于预测失败的处理，保证了算法的正确性。当然此算法还有许多的改进空间，还未使用图的连通性相关的性质。未来还有改机的机会。

致谢

感谢平台和主办方给了我们这次锻炼机会，感谢指导老师孙瑶研究员给予的指导和帮助。感谢主办方提供包含 Tesla V100 的开发测试环境。感谢陈彦棋和杨润河同学提供提供的符号化指导以及帮助。

参考

[1] Donald E.Knuth. 计算机程序设计艺术[M]. 国防工业出版社, 2001.