三角形图计算算法设计及性能优化

冀数所

王志强
副研究员
河北省科学院应用数学研究所
中国-石家庄
wzhq77@dingtalk.com

赵会民
助理研究员
河北省科学院应用数学研究所
中国-石家庄
zhm_1213@163.com

团队简介

冀数所队是由两位算法爱好者组成的队伍，成员都来自位于石家庄市的河北省科学院应用数学研究所，我们日常工作是进行数据挖掘相关的研究工作，平时喜爱钻研一些基本算法技术和编程技巧。我们相信“算法成就未来”。

队长：王志强，博士，毕业于浙江大学控制系。当前的主要研究方向为非线性优化、大数据分析等。在本次比赛中主要负责整体方案设计，CPU 端算法实现与优化。

队员：赵会民，硕士，毕业于河北工业大学控制科学与工程学院。当前主要研究方向为机器学习、大数据分析。在本次比赛中主要负责 GPU 端算法的实现与优化。

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三角形计数方案。我们团队通过对图数据的邻接矩阵计算原理进行分析，采用了按顶点度次序重载数据、建立行区间索引表、并行排序等策略对原数据进行深度预处理，并充分利用映射内存空间和 GPU 内存空间、数据分块计数操作等编程思路对方案进行充分优化，实现了对大规模图数据的正确计数。通过测试，我们方案能够在 10 分钟之内完成对 40G 大小的图数据进行三角形计算运算，并且具备很好的扩展性，即使数据量再增加 30%也能够保证计算效率。

关键词

图数据处理，三角形计数，邻接矩阵，并行计算，CUDA 编程

1 赛题简介及理解

本赛题的任务是根据给定的并行计算资源，设计超大型图数据中三角形个数的计算方案，并通过优化和调试来获得最高的计算效率。通过我们对赛题和数据的分析，本题需要解决以下几个方面的问题：

1) 图数据中进行准确三角形计数的问题；

2) 在有限计算资源下进行超大规模数据处理的问题；

3) 高效利用多处理器与 GPU 等并行资源的问题；

4) 根据编程语言特点进行算法方案优化的问题。

2 基本原理

将待分析的关联图数据处理成由低序号顶点向高序号顶点传播的有向图，则该数据即可用上三角形式的邻接矩阵 $\mathbf{A}$ 表示。该图数据中的三角形个数可用矩阵求解公式(1)表示。

$$n_{\mathrm{TC}} = \sum(\mathbf{A}^2 \odot \mathbf{A}) \tag{1}$$

式中，$\odot$ 为前后的矩阵按元素求积，Σ 为将矩阵中所有元素累加。

为了更直观地表示邻接矩阵与图数据的对应关系，如图 1 所示，把图 1 中左边的图转化为右边的上三角矩阵表示，可见邻接矩阵的非零元素即对应图的一条边。所有共边 (a,b) 的三角形个数，即为 a 和 b 所在行的元素位置重合的数目。那么，对式(1)的计算过程可理解为对邻接矩阵 $\mathbf{A}$ 的所有非零元，先找到其代表的边所对应的两个顶点，然后找到两个顶

点序号所对应的两行，对这两行非零元的列标号求交集，即为共一条边的三角形个数，遍历所有边即可求出总的三角形个数。

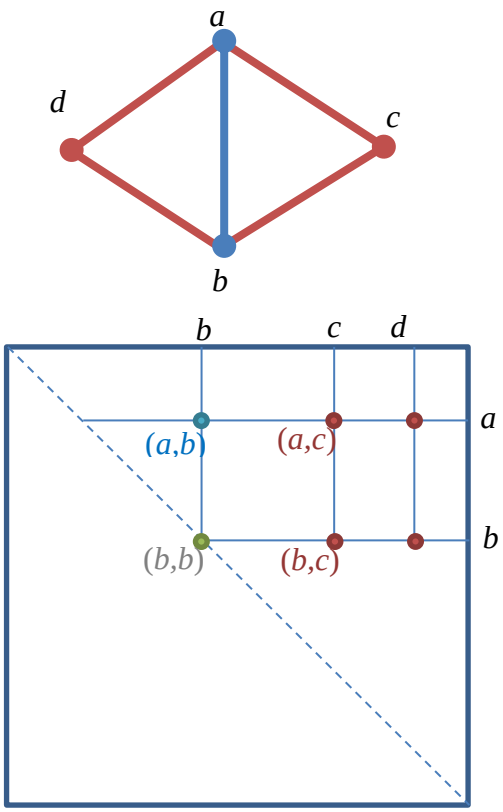


图 1 图表示与对应其邻接矩阵

3 基本算法过程

图数据三角形计算的基本算法分三步：

Step 1. 读入图数据文件；

Step 2. 对数据进行排序、去重、去自环边，形成上三角邻接稀疏矩阵；

Step 3. 利用求解原理计算三角形个数。

4 大规模计算面临的问题及对策

本次赛题所出数据集非常巨大，最大数据集的文件大小就达到了 40G，而所给计算资源仅有 60G 物理内存可以使用。无论在 CPU 还是 GPU 上很多计算均无法一次计算完成。因此，我们给出了以下策略来解决大规模计算所面临的计算资源受限的问题。

4.1 映射内存空间利用最大化

大文件必定占用大内存，我们早期的方案是将文件映射到内存，再拷贝到另一个数组中再进行后续操作，仅仅变量的声明、初始化、数据交换等步骤就需要大量的计算开销。而实际上数据交换、排序、去重等步骤可以在映射的数据内容上进行操作，并且在后续处理压缩后，还可以借用内存空间完成其它变量的存储，这样即可以节省大量的内存空间开销，在计算过程中，还因为在同一地址块内操作，大幅提高计算效率。

4.2 GPU 内存空间利用最大化

受映射空间利用的启发， GPU 计算卡同样具有一大块内存空间，而我们的 GPU 方案大部分的计算又是在 GPU 上完成的，所以完全可以把能够完成的计算工作在 GPU 中进行完成，只用输出最终的计算结果。经过合理的设计后，同样可以大幅减小 CPU 与 GPU 之间的数据通讯开销，从而提高计算效率。

4.3 根据 GPU 内存大小实现数据动态分块操作

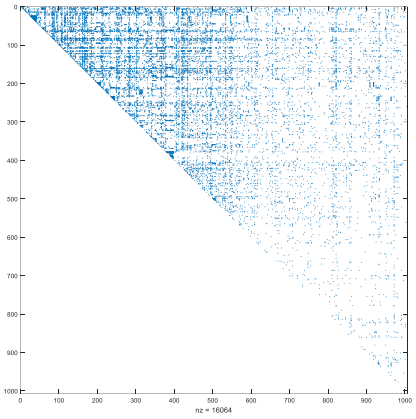
由于数据量巨大，而 GPU 内存有限，不能实现一个批次完成数据处理与计算。因此根据 GPU 的计算的特点将 CPU 中的按行求交集操作升级为按“块”求交集操作，同时根据 GPU 内存容量来动态分配块的大小，尽量减小数据调度开销。

5 方案优化

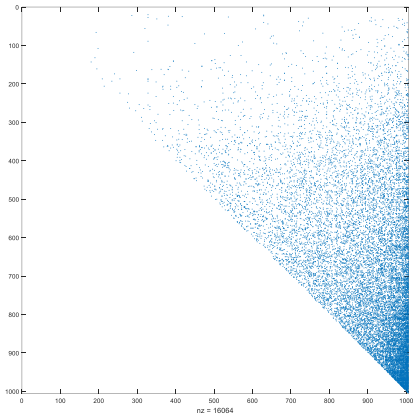
本方案最大的优化策略之一就是根据顶点度数进行了重新编号，即对顶点的总度数（出度+入度）进行统计，然后根据度数大小次序进行重新编号，并对原数据进行统一替换。增加该步骤会大幅增加预处理时间，但同时会更大幅度降低后期求交集的计算量，总体来看这个代价是非常划算的。

以 email-Eu-core（snap.stanford.edu）数据集为例，顶点 1005，边 25571，三角形 105461）。其邻接矩阵的非零元分布如图 2 所示。从图中可以看出，重新编号后非零元都往右下方集中，这样有效消除了长数据集合，大大减少了求交集的计算量。我们用 CPU 方案对该数据集进行了测试，

原图计算比较次数为 745528，重新编号后的比较次数为 370279，不到原来的一半。我们同样测试了预赛数据集 soc-LiveJournal1，原数据比较次数为 5,415,185,333，而重排后次数为 1,555,842,838，降幅更是高达 71.3%。



（a）重编号前



（b）重编号后

图 2 连接矩阵非零元分布图

6 详细算法设计

6.1 方案一（CPU 方案）

Step 1. 利用 mmap()将图数据映射为数组

Step 2. 遍历数据找到最大值，即为顶点个数

Step 3. 统计顶点总度数，并按度数大小排序后生成新的编号，对原数据进行替换（没有该步骤，不影响结果正确性，但会大幅增加计算时间）

Step 4. 整理数据，对数据进行排序，将数据规整为上三角矩阵

Step 5. 对数据进行去重、归并作，建立行区间索引表

Step 6. 根据计算原理与行区间索引表对排序后的图数据进行计算，即得到三角形计数

对该方案的排序和三角形计数的模块替换为 GPU 算法，即可得到初步的 GPU 方案。

6.2 方案二（GPU 方案）

在方案一中，预处理大多数情况下是用 CPU 完成的，但随着数据集增大，由于文件镜像的增大，计算内存越来越少，因此我们将方案升级为由 GPU 完成几乎所有的计算任务。

Step 1. 对原文件进行内存映射（实现快速读取）

Step 2. 查找最大顶点号，统计各顶点的总度数，排序生成新旧编号对照表（以最大顶点来确定邻接矩阵维度）

Step 3. 按新顶点序，遍历数据，统计小端顶点度，建立排序后的行区间索引表（这里与方案一略有不同，重编号后没有对原数据进行实际替换，只是根据对照表来计算出度数，再计算出排序后区间索引表）

Step 4. 根据编号对照表与行区间索引表，分段打开镜像，生成只占一半内存空间的列序号数组（列序号数组与行区间索引表即将图数据表示为 CSR 格式的稀疏矩阵）

Step 5. 对新生成的列序号数组进行排序与去重，即将数据表示为严格的上三角形式的邻接矩阵（该步骤是按行区间进行多线程 CPU 并行排序的，就目前的方案来看比 GPU 排序的效率要高）

Step 6. 计算三角形个数（以各边端点索引到桶，然后对两桶数据求交集，相交元素个数即共享该边的三角形数。利用 GPU 与上三角阵的特点，是每次载入两块数据，前面的一块与后面的块数求交集计算依次完成后，不用再次载入）

7 运行环境及测评分析

7.1 正式比赛的运行测试环境参数

CPU：Intel Xeon E5-2686 v4 (8 逻辑核心)

内存：60G　　硬盘：500G

GPU：NVIDIA TESLA V100 (16G GPU 内存)

操作系统： Ubuntu 16.04.06 LTS

7.2 正式比赛数据集测试数据

由于数据过大，程序运行过程中，首次计算一个数据集时往往会有很长的数据载入内存的时间，而如果连续重复计算一次，速度会快很多。因此我们每个数据集连续计算两次，相应的时间记录请参见表 1。首次计算的时间会有较大的波动幅度，而后续计算时间十分稳定，我们计划在将来的研究中对成因进行深入分析并加以改进。

表 1 正式比赛数据集测试数据

数据名称	**s28.e15.kron.edgelist**		**s29.e10.kron.edgelist**	
顶点数	268,435,456		536,870,909	
边数	4,026,531,840		5,368,709,120	
三角形数	199,196,078,202		164,015,236,714	
预处理批次	首次	第二次	首次	第二次
预处理(s)	283.90	126.25	415.87	279.68
计数(s)	146.18	146.10	179.32	179.41
总时间(s)	430.84	**273.87**	571.94	**459.85**

7.3 超大规模的极限测试数据

由于我们特别针对大规模数据进行设计优化，可以胜任特大数据集的计算。我们按 Graph500(graph500.org)给出的 Kronecker Graph 生成算法生成了三个特大数据集进行了测试。

表 2 特大数据集测试数据

数据名称	**Kron 数据一**	**Kron 数据二**	**Kron 数据三**
大小(M)	45,056	49,152	**56,320**
顶点数	536,870,911	**1,073,741,817**	738,197,499
边数	5,905,580,032	6,442,450,944	7,381,975,040
三角形数	**206018770735**	**100280521397**	**349953480955**
预处理(s)	652.95	835.19	856.38
计数(s)	207.40	196.65	313.30
总时间(s)	**860.25**	**1,032.62**	**1,170.50**

上述结果均为在官方给定环境中测出。我们另有改进方案，通过分片映射的策略，可以实现数据文件大于内存容量的计算。

8 结语与展望

通过本次比赛，对三角形的并行计算方法有了更加深刻的了解，在算法设计方面有如下心得：

1）合理复用已经分配的内存空间，尽量避免过多的中间变量生成。

2）多核 CPU 计算方面，尽量避免内存写入冲突，尽量避免原子操作。

3）尽量减少 CPU 与 GPU 的数据交互频次，将 GPU 的内存容量利用到极致。

由于团队成员对 LINUX 下的 GCC 编程环境和 CUDA 编程均处于学习摸索阶段，因此方案还有很大的提升空间。保守估计，我们方案的计算效率还有 20%-50%的优化提升空间。

9 致谢

感谢平台和主办方给了我们这次锻炼机会。感谢指导老师刘利军研究员在 GPU 编程上给予的指导和帮助。感谢我们单位保障了 Tesla V100 计算资源，让我们能够 24 小时地拥有跟官方一致的开发测试环境。感谢去年冠军队“欧拉的核弹”的成果分享，为我们提供了宝贵的程序模板和努力基线。