并行期末大作业报告

江振升 1400012781
何卓论 1300012885

2017 年 6 月 28 日

1 项目概述

1.1 选题

本组所选的题目为问题一"Parallel DAG Traversal"。

1.2 问题描述

Parallel DAG Traversal 给定一个有向无环图 (DAG)$G = (V, E)$，其中 $V = I \cup M \cup O$。M 是内部点集，每个内部点 $m \in M$ 都有一个权重；I 是源点集，O 是汇点集，每一对源点和汇点之间都有一个权重。

对于每一个汇点 $o \in O$，找出所有从任意源点 $i \in I$ 出发的到达 o 的路径中权重前十大的路径。一个路径的权重定义为源点和汇点的对权重加上经过的内部点的权重之和。

2 算法实现

2.1 串行部分算法

2.1.1 总体思路

首先，如果不考虑源点和汇点的对权重，对于内部点 $m \in M$ 以及汇点 $o \in O$ 考虑原问题，可以定义 $Top(m)$ 为由任意源点出发到达 m 的路径中权重前十大的路径的权重。那么则有：

$$Top(m) = max10\{\bigcup_{t \in par(m)} Top(t) + w(m)\}$$

其中，$max10$ 指取集合中的前 10 大元素；$par(m)$ 指节点 m 在 DAG 中的父节点的集合；对于源点 $i \in I$ 而言，$Top(i) = \{0\}$；对于源点与汇点而言，其单点权重都为 0。

如果需要考虑源点和汇点的对权重，那么就要对上式进行修改：

$$Top(m, i) = max10\{\bigcup_{t \in par(m)} Top(t, i) + w(m)\}$$

其中，$Top(m, i)$ 表示从源点 i 出发，到达 m 的路径中前十大的路径的权重；对于源点 $i \in I$ 而言，$Top(i, i) = 0, Top(i, j) = \emptyset (i \neq j)$。

那么，原问题的结果，即 $Top(o)$，就可以表示为：

$$Top(o) = max10\{\bigcup_{i \in I} Top(o, i)\}$$

观察上述各式，主要要解决的问题有：

1. 如何递推的计算 $Top(m, i)$，即以什么顺序遍历 DAG；
2. 如何计算 $max10$，其中它的参数来自若干个长度最多为 10 的降序排列的数组。

2.1.2 递推计算

观察 $Top(m, i)$ 的计算式我们可以知道，它的依赖关系是它的父节点，即在计算 $Top(m, i)$ 之前，$\forall j \in par(i), Top(j, i)$ 都要已经计算出来。因此，我们可以通过拓扑排序来确定正确的递推计算顺序。

2.1.3 $max10$ 的计算

分析 $max10$ 的参数的特点：

- 来自若干个数组；
- 这些数组是降序排列的。

可见，这其实是一个多路归并的问题，而且只要求归并后的前 10 个结果，可以通过败者树来实现。

2.2 并行部分算法

2.2.1 $Top(m, i)$ 的并行性

根据之前的分析，$Top(m, i)$ 的计算只依赖于它的父节点，即对于 $\forall i, j, i \neq j$，则有 $Top(\cdot, i)$ 与 $Top(\cdot, j)$ 的计算是相互独立的。因此可以对于每一个源点进行并行计算。

2.2.2 等深度的并行性

对于 $v \in V$，定义 $d(v) = \max\limits_{t \in par(v)} d(t) + 1$，且对于 $i \in I$，$d(i) = 0$，称 $d(v)$ 为 v 的深度。在同一个源点的计算中，$Top(m, i)$ 的计算只依赖于它的

父节点，即等深度的节点的计算是相互独立的。因此可以先通过拓扑排序初始化确定每一个节点的深度，然后再根据深度进行并行计算。

3 代码实现与数据构造

3.1 串行部分的代码实现

3.1.1 图的数据结构

使用邻接表实现，而且对于每条边都要存正向边和反向边。正向边是用于图的遍历，反向边是用于递推计算。

关于图的数据结构在 *config.h* 中，其中 *Node* 是节点的类，*Edge* 是边的类，*Graph* 是图的类。

3.1.2 败者树

根据问题的特点，败者树实现为：

- 初始化时接受一个最大的路数，便于在不同计算中复用；
- 排序时只取最多前 10 大的结果。

关于败者树的数据结构在 *LoserTree.cpp* 以及 *LoserTree.h* 中。

3.1.3 图的遍历

在图的初始化时同时计算每个节点的入边数 fa_num，在运行时维护一个队列，一开始在队列中的是源点，每处理完一个节点，检查一次它的子节点，并将 fa_num 减 1。如果减 1 后为 0，则将该节点入队。

3.2 并行部分的代码实现

根据之前的分析，我们只要在每个节点计算 $Top(m, i)$ 是对于每个 i 进行并行化即可。

3.3 数据构造

3.3.1 无权 DAG 的生成

我们采用了现成的开源代码进行生成。该生成器[1]是由 DePaul University 的 Richard Johnsonbaugh 与 Martin Kalin 所编写的，可以根据节点数与边数生成随机的 DAG。但由于图的数据结构是邻接矩阵，因此无法生成规模太大的图。

3.3.2 点权重与对权重的生成

在生成了 DAG 后，我们自己编写了一个权重生成器，在 *competor.cpp* 中，用于为已有的 DAG 随机生成范围在 $0 \sim 127$ 的点权重与对权重。

4 项目结构与编译向导

4.1 项目结构

- *CMakeLists.txt*: cmake 文件；
- *data*: 数据文件目录
  - *__graph__10000__5000000.dat*: 节点数为 10000，边数为 5000000 的图；
  - *__graph__40000__10000000.dat*: 节点数为 40000，边数为 10000000 的图。该图的串行版本运行时间超过 1 分钟；
  - *tiny__sample1.dat*: 与所给的样例一致的图，用于检测正确性。
- *doc*: 文档文件目录，包括本文件；
- *tools*: 工具目录
  - *graph__ge.c*: 图随机生成器；
  - *stat.py*: 测试统计代码，请使用 python3 运行。
- *src*:
  - *buildDAG.cpp*, *buildDAG.h*: 读入数据相关；

[1] `http://condor.depaul.edu/rjohnson/source/graph_ge.c`

- *competor.cpp*: 生成点权重与对权重；
- *config.h*: 图的数据结构；
- *LoserTree.cpp*, *LoserTree.h*: 败者树实现相关；
- *parallem_main.cpp*: 并行实现的主函数；
- *parameters.h*: 一些宏定义；
- *serial_main.cpp*: 串行实现的主函数。

4.2 数据位置

由于数据过大，我将生成好的数据放在了课程服务器上，位置是：
/home/parallel_class/s1400012781/project/data

4.3 如何编译

本项目通过 cmake 组织的，可以按照以下步骤进行编译：

1. 在项目根目录建立 *build* 目录，并进入 *build* 目录；
2. *cmake ..*
3. *make*

另外，如果默认的编译器不支持 openmp，假设另外一个编译器为 clang 与 clang++，请将 *cmake ..* 改为 *CC=clang CXX=clang++ cmake ..*。

4.4 如何运行

串行版本： ./serial <data_path>
并行版本： ./parallel <data_path> <threads_num>

5 实验结果

5.1 实验方法

设备硬件环境：**CPU:** Intel Core i7-6820HQ @ 2.70GHz, **RAM:** 16GB
操作系统： **OS:** 64bit Mac OS X 10.12.5 16F73, **Kernel:** x86_64 Darwin 16.6.0

表 1: 实验结果

程序类型	I/O 时间	计算时间	speedup	efficiency
串行	3.23s	63.2s	/	/
1 线程	2.94s	85.5s	75.09%	75.09%
2 线程	3.02s	45.4s	137.19%	68.59%
3 线程	2.92s	31.6s	192.36%	64.12%
4 线程	2.94s	25.4s	234.08%	58.52%
5 线程	3.13s	23.9s	245.90%	49.18%
6 线程	2.96s	22.1s	264.74%	44.12%
7 线程	3.01s	21.4s	271.70%	38.81%
8 线程	2.99s	22.2s	264.06%	33.01%

编译器版本：g++-6 (Homebrew GCC 6.3.0_1 –without-multilib) 6.3.0

测试数据：*_graph_40000_10000000.dat*,包含 40000 个节点与 10000000 条边

测试方法： 对于每个指标运行三次，取平均值。测量 *I/O* 时间以及计算时间。时间测量使用 OpenMP 提供的 *omp_get_wtime* 函数

5.2 实验结果

数据见表 1。其中，*speedup* 以及 *efficiency* 都是用总时间计算的。