**数据库大作业感想**

这次的数据库大作业任务是：把串行批量构建的B+树更改为并行批量构建。为完成本次大作业，我首先需要去理解助教给出的代码bulkloading的过程，然后再去学习并行开发的设计，提出并行方案，并实现。理解bulkloading的过程花了一两天的时间，一开始因为给的代码太多，看的有点疲惫，看完这个文件就忘记另一个文件写了啥。为了便于我自己理解，我使用了做笔记整理的方法，通过把关键信息记录下来的方法，

bulkloading构建B+树的过程跟以往学习的构建B+树的过程不同，是自底向上构建的，先完成底层叶子结点的构建，连接成双向链表。再构建索引节点，依次向上，最后完成根节点的构建。完成bulkloading过程的理解，就可以开始完成并行的批量构建。

我们选用的并行设计方法是多线程运行，多线程在操作系统课学习过，因此学习起来比较快。我们想出的并行优化办法是，额外搭建一个小顶堆，然后把指定数量的叶子结点（经过计算的数量）分发给workerThread，并让workThread分别按照各自的下标顺序进行构建。最后通过小顶堆传给consumerThread/composerThread，把构建完成的节点一次性写入磁盘来完成搭建。

经过这次数据库，我学会了很多新的知识，对并行开发也有了进一步的理解，收获满满。

在本次数据库的课程设计中，我理解了 B+ 树如何批量地载入数据，即 Bulk Load 的过程以及关于 B+ 树的数据结构和磁盘交互是如何实现的。并且在理解源码时，同时阅读了 QALSH 的相关源码以及论文，学习到了 B+树、哈希、索引在关系型数据库中的作用。并且在也学习了如何通过 c++11 中提供的多线程进行并发操作，以实现并行构建叶子、索引结点从而构建 B+ 树实现并行 Bulk Load。

（1）实验内容：在本次实验中，我负责编写并行设计思路以及具体的代码实现。

（2）实验过程

在实验的开始，我仔细阅读了实验提供的代码，对 B 树 bulkloading 构建的概念有了一定了解。然后仔细阅读了需要并行化处理的串行代码，结合暑假时阅读的《C++ Concurrency in Action》，很快找到了突破口。其实并行化处理的思想比较简单，就是将能并行化的代码都并行化，不能并行的就尽量缩短串行执行时间。这个思想也是阿姆达尔定律（Amdahl's Law）的体现，该定律应用到并行计算领域时可以向人们揭示一个道理：即使通过无数个 CPU 核心运行那些并行化处理的代码，最终得到的加速比仍会受制于串行代码的运行时间。如何识别、划分能与不能并行的代码，就是这次实验的主题。

能够并行化的代码，在本次实验中在我看来应该是对新的节点进行 `add\_child`、连接节点、以及调用 `write\_to\_buffer` 的过程。而不能并行化的代码则是将 block 写入硬盘的过程，因为在机械硬盘甚至是 SSD 上，像写入一个 512 byte 的 block 这样的小规模随机读写的性能（在一些性能测试指标上被称为 4K 随机读写）是非常糟糕的。一个很直接却有效的办法就是人为地做 buffer，即将多次随机写入转换成一次大规模的顺序写入。尽管操作系统已经会自动做这层优化，但针对如此大规模的 bulkloading 过程，操作系统提供的 buffer 一般是不足的。

不过在实验最后，我还是将连接节点和调用 `write\_to\_buffer` 的工作交给了一个单独的线程去串行处理，因为我认为这两个需要花费的时间可以等同于其他线程并行 `add\_child` 生产加载后的节点的时间。当然 CPU 核心再多一点可能就不是这样了，不过我的虚拟机只有 4 个核心。并且也得到了一个尚可接受的结果：2 以上的加速比。

（3）遇到的困难

本次实验中我遇到的困难主要是在调试程序的部分。在我确定好并行设计的思路以及编写好初版的代码时还算顺利，但之后的调试时间却花费了更多时间。

在调试过程中，为了让程序的行为正确，我修复了在并行设计思路中没有仔细思考好的细节，例如 `block` 号的计算。总之最后程序能够如期运行。