Short Survey

郭宇祺 1700012785

在本篇short survey中，我简要描述一下此次任务中节点分配算法思路的由来和一些尝试的过程，以及描述一下并行化的思路。

在处理算法的过程中，我第一步关注的问题就是节点在进程之间的分配方案。在本次任务中，由于我使用了MPI框架，考虑到采用根据输入数据动态划分节点的做法可能给进程间通讯带来困难，本次任务中我只打算采用静态划分节点的做法。我尝试使用了三种分配方案。第一种分配方案是块状分配，即首先把所有节点按编号从小到大排序，然后平均地分割成连续的p段，每一段分配给一个进程处理。第二种分配方法是循环分配，即将编号为i的节点分配给i%p号进程处理。第三种分配介于第一种和第二种分配方案之间，首先设定一个group_size，每连续的group_size个节点被编为一个group，并赋予group编号，从0开始按顺序递增，然后将编号为i的group的所有节点都分配给i%p号进程处理。容易看出，这三种分配方案对应不同的分割粒度，第一种分配方案粒度最粗，第二种分配方案粒度最细，第三种分配方案处于二者之间，而且可以通过调整group_size的大小进一步调整分割粒度。

在这个任务中，树上每一个节点所做的计算量是少量而且固定的，因此影响程序并行性能的关键是进程之间的通讯量。程序并行效果的好坏很大程度上取决于划分算法能否将整棵树划分成多个相互独立的子树，并平均地分配给每一个处理单元。最理想情况下，划分算法将所有节点平均分配给p个处理单元，每个处理单元负责处理一棵或多颗完整的子树，相互之间没有依赖，可以完美并行。然而，这种理想的分配建立在对树结构复杂分析的基础之上，这种复杂分析的耗时很可能比串行处理的耗时更多，那样这种分割算法就没有任何意义。因此，我设计分割算法的原则就是简单，用最简单的办法、最少量的计算，完成节点在处理单元之间的划分。基于这样的原则，我提出了上面三种分割算法。由于缺乏相应的经验，我无法直接判断这三种算法孰优孰劣，因此决定分别编写三个版本的代码，并根据实验结果决定采用哪一种方案。

实验过程在此略去。最后的实验结果是，第一种方案和第三种方案效果相差无几，难分优劣，而第二种方案效果远远差于其他两种。第二种方案效果很差的主要原因是，这种分配方案使得树上的节点被零碎地分布到每一个处理单元之中，因此父子节点通常会被分配到不同的进程之中，这大大增加了通讯量，拖慢了程序整体性能，因此第二种方案被首先弃用。在衡量和第一种方案和第三种方案后，我最终决定采用第一种方案，一是因为第一种方案实现更简便，二是因为第三种方案获得好的性能非常依赖group_size参数的选取，而这个参数需要人为设定，不利于程序的自动化运行。综合考虑，我选择使用最简单、效果也最稳定的第一种方案进行节点的分配。

随后进入代码思路设计的环节。我的代码思路主要来源于垃圾回收算法中的引用计数算法。在串行版本的算法中，节点需要严格按编号顺序访问。但实际上，只有父子节点之间存在数据依赖关系，只要满足这种关系，就可以不必严格遵循按编号顺序访问的限制。为了最大程度地挖掘算法并行性，我需要设计一种数据结构代替编码顺序反应节点之间的依赖关系。一种显然的做法是构建依赖图，但是直接构建完整的依赖图开销太大，而且也不利于分布式存储。在思考中，我想到了垃圾回收算法。将每个节点视为一片分配的内存区域，一个依赖关系视为一条引用，而节点的访问视为释放对应的内存区域，这样，对树自顶向下和自底向上遍历的顺序就可视为对所有内存空间安全释放的顺序。那这个任务就变成了一个垃圾回收的任务。垃圾回收算法中的引用计数算法，由于其具有数据独立存储、无停顿的特性，完美契合分布式内存编程的需求。因此，在本次的project中，我大量使用了这种引用计数的算法，取得了不错的效果。具体实现详见实验报告。

其余部分则是模式化的代码优化，没有太多可供陈述的内容。

以上就是本次**short survey**的全部内容。