作业九　FOX 并行算法和 PageRank 算法

150*******　李师尧

1、 分析在分布存储系统上采用 5.1.4 中 FOX 算法实现数组 A*B 时，每颗处理器上的存储空间开销和数据交换量。假设有 P*P 颗处理器，数组 A 的规模为 N*K、B 的规模为 K*M。

2、 请设计一个适合 NUMA 存储结构的并行 PageRank 算法。

1 Fox 算法

采用 FOX 算法计算矩阵相乘

$$C_{N\times M} = A_{N\times K}B_{K\times M} \tag{1}$$

假设有 P^2 个处理器。根据 5.1.4 节的算法设计，每个处理器上的数据存储量由三部分组成，分别存储 A、B、C 三个矩阵的子矩阵，由于 A、C 的子矩阵固定存储在一个固定的处理器上，而 B 的分布则是动态的。所以单个处理器的数据存储量：

$$\begin{aligned} Mem &= Mem_A + Mem_B + Mem_C \\ &= \frac{N}{P}\frac{K}{P} + \frac{K}{P}\frac{M}{P} + \frac{N}{P}\frac{M}{P} \end{aligned} \tag{2}$$

数据交换量包括：在第 k 个超级时间步前，将 A 广播给每个处理器；在超级时间步结束后，每个处理器均要完成将自己存储的 B 矩阵传递给“上邻居”，接受“下邻居”传来的 B 子矩阵。因此数据交换量为：

$$Swap = \left[\frac{N}{P}\frac{K}{P} + \frac{K}{P}\frac{M}{P}\times 2\right](P-1) \tag{3}$$

2 PageRank 算法

根据 5.2.3 节所述，PageRank 排名抽象为计算有向图 G=（V,E,Value,Weight）的各个顶点的状态值，该有向图的每条边的权值均设为常量 1。初始时令，每个顶点的状态值为 1/N，然后不断迭代求解得到一个稳态解。迭代式为：

$$PR(p_i,T) = \frac{1-d}{N} + d\times\sum_{p_j\in M(p_i)}\frac{PR(p_j,T-1)}{L(p_j)} \tag{4}$$

考虑 NUMA 存储结构的特点。NUMA（Non Uniform Memory Access Achitecture）是非均匀的共享式存储结构，与 UMA 相比，主要的区别在于通过软件和硬件协作，能够实现存储空间的统一寻址，而不同存储单元的访问效率不同。

鉴于 PageRank 算法极容易出现的负载不均衡性和数据访问性的特点，在基于 NUMA 存储结构的计算平台上，MPI 的消息传递模型并不适合此问题，而应该选取 openMP、pthread、CUDA 等并行编程模型。pthread 线程轻量化，极适合此问题，进一步思考也表面，openmp 和 cuda 也很难应对此问题。因此，在编程模型方面，优选 pthread。

并行算法设计如下：假设有 p 个线程，将总的 N 个顶点划分为（n*p）个子任务（n 是一个可以调节的参数，暂定为 20），每个线程每次领取一个子任务，处理一部分的顶点，并将结果返回给主进程；为了提高数据的局部访问性，应该在将网站的诸多页面信息抽象成一

张有向图之前，进行一些必要的预处理工作，比如将访问关系比较接近或者集中的页面的顶点索引尽量接近，以便于后面划分任务时根据顶点的索引划分，能够尽量使得线程领取到子任务后，所要处理的顶点的连入数组尽量在子任务范围内；为了提高负载均衡性，有必要对连入数目较多的页面，进行一些标志，在任务划分的过程中，将这些连入数目较多的顶点比较均匀地布散开，使得每个线程的计算量尽量接近。因此，对本并行算法来说，进入并行阶段的预处理是非常重要的，关系到整个算法的负载均衡和数据访问局部性。

事实上，针对 PageRank 问题，虽然真实场景下面的网站、 网页都很多，但在一段时间内，有向图的内部拓扑结构并不会发生巨大的、本质性的变化，因此可以隔一段时间进行一次前处理，而前处理结果可以作为这一段时间内任意时刻的 PageRank 并行算法的任务划分的依据。

最后写出该并行算法的伪代码。

```
void main(){
    getG();   //获得有向图数据，该函数可以从一定时间段更新的后台进程中获得信息
    taskSize=taskSplit();   //任务划分
    totalThread = 0;
    set_mtx();      //进行设置互斥锁操作
    if ( thread_num>Max_Thread_Num ) thread_num = Max_Thread_Num;
    for(i=0; i<thread_num; i++) pthread_create(&(threads[i].id), NULL, threadfun, NULL);
//调用线程进行处理



}

void *threadfun(){
    //线程函数，根据 tasksize 去计算所负责的部分任务的顶点的状态值
    while （TRUE）{
        calculateTask();    //明确自己的任务范围
        process(lb,ub);      //处理 lb~ub 之间的顶点的，计算其状态值
        if (setbreak()==true) break;          //查看主进程是否发来计算结束的信号量，如果查到，则计算结束
    }
}
```