

# Homework

OpenMP编程

Name: Li Shibo Student ID: 119033910046 Email: ShiboLi@sjtu.edu.cn

**Monte Carlo algorithm** 蒙特卡洛方法，也称为统计模拟方法，是二十世纪四十年代中期由于科学技术的发展和电子计算机的发明，而被提出的一种以概率统计理论为指导的一类非常重要的数值计算方法。是指使用随机数（或更常见的伪随机数）来解决很多计算问题的方法。

蒙特卡洛是一种随机算法，在很多领域内应用较为广泛。本次实验将采用蒙特卡洛方法实现圆周率 $\pi$ 的近似计算。

$\pi$ 的近似计算：构造一个单位正方形和一个单位圆的 $1/4$ ，往整个区域内随机投入点，根据点到原点的距离判断点是落在 $1/4$ 的圆内还是在圆外，从而根据落在两个不同区域的点的数目，求出两个区域的比值。如此一来，就可以求出 $1/4$ 单位圆的面积，从而求出圆周率 $\pi$ 。方法如下图所示。

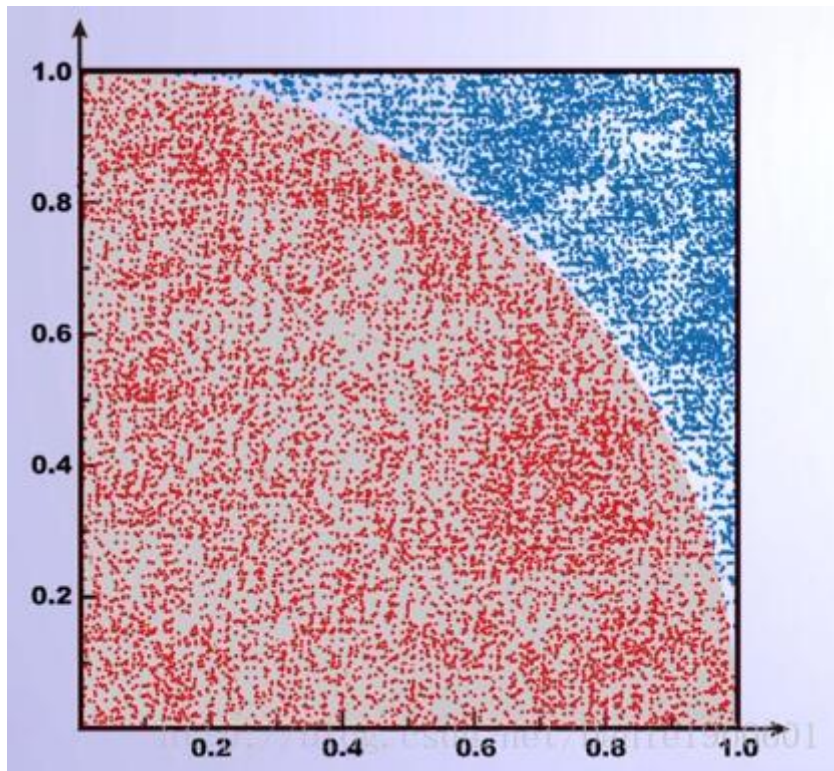


图 1: 使用蒙特卡洛方法来计算圆周率的值。图中红点表示落在单位圆内的点，而所有的点都落在单位正方形内。

用OpenMP求解 $\pi$ 值，本次实验中利用的是生成随机点落在单位圆的个数和整个落入单位正方形中点的个数的比值来表示单位圆和单位正方形的面积之比。

由于要生成随机点，本次实验采用循环生成 $x$ 和 $y$ 坐标分别在 $[0, 1]$ 内的点。每次产生两个 $[0, 1]$ 内的随机数 $x$ 和 $y$ ，利用 $x^2 + y^2$ 和1的大小来判断该点是不是落在单位圆中。由于想要近似的比较准确，

要生成的点比较多，因此循环的次数也比较多。本次实验过程中使用OpenMP来对循环进行并行化处理。实验结果如下图所示。

```
root@ecs-00:~/OMPPProgram# ./montecarloalgforPI
Input thread_num:
1
Input point_num:
10000000000
the estimate value of pi is 3.141595
the absolute error is 0.000001
elapsedtime: 409.747594
```

图 2: 使用蒙特卡洛方法来计算圆周率的值。图中为采用1个线程循环10000000000次的结果。

```
root@ecs-00:~/OMPPProgram# ./montecarloalgforPI
Input thread_num:
64
Input point_num:
10000000000
the estimate value of pi is 3.141614
the absolute error is 0.000007
elapsedtime: 900.323267
```

图 3: 使用蒙特卡洛方法来计算圆周率的值。图中为采用64个线程循环10000000000次的结果。

从图2中可以看出，循环所花费的时间较长，约为410秒。图3中为采用64个线程循环同样多次数所获得解结果，从图中可以看出，所花费时间随着线程数量得增加而增加。原因可能是由于服务器只有两个核，而较多的线程并没有得到比较好的并行效果，反而线程之间对共享内存的处理消耗较多的时间。

**Quick Sort** 快速排序的基本思想是通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

在本次实验中使用OpenMP对快速排序的每一部分进行并行处理。实验中对1000000个随机数组进行排序，因此没有打印排序结果结果，实验过程中所消耗的时间如下图所示。

```
root@ecs-00:~/OMPPProgram# ./qsortwithomp
elapsedtime with omp: 0.362021
elapsedtime without omp: 0.550369
```

图 4: 使用OMP进行并行处理的快速排序所消耗的时间和不适用OMP进行并行处理所花费的时间对比。

从图4中可以看出，使用OMP进行并行处理所消耗的时间比普通的快速排序所花费的时间要少的多。

**PageRank** PageRank通过网络中的超链接关系来确定一个页面的等级，每一个页面都有一个PR值，用来表示网页的等级。在PageRank算法中，将网页看作是顶点而将网页的链接看作是有向边，从而整个网络可以构造成一个有向图。这样计算网页的PR值就转化为计算有向图中节点的PR值。PageRank算法就是根据有向图来计算每个节点的PR值。以下图为例。

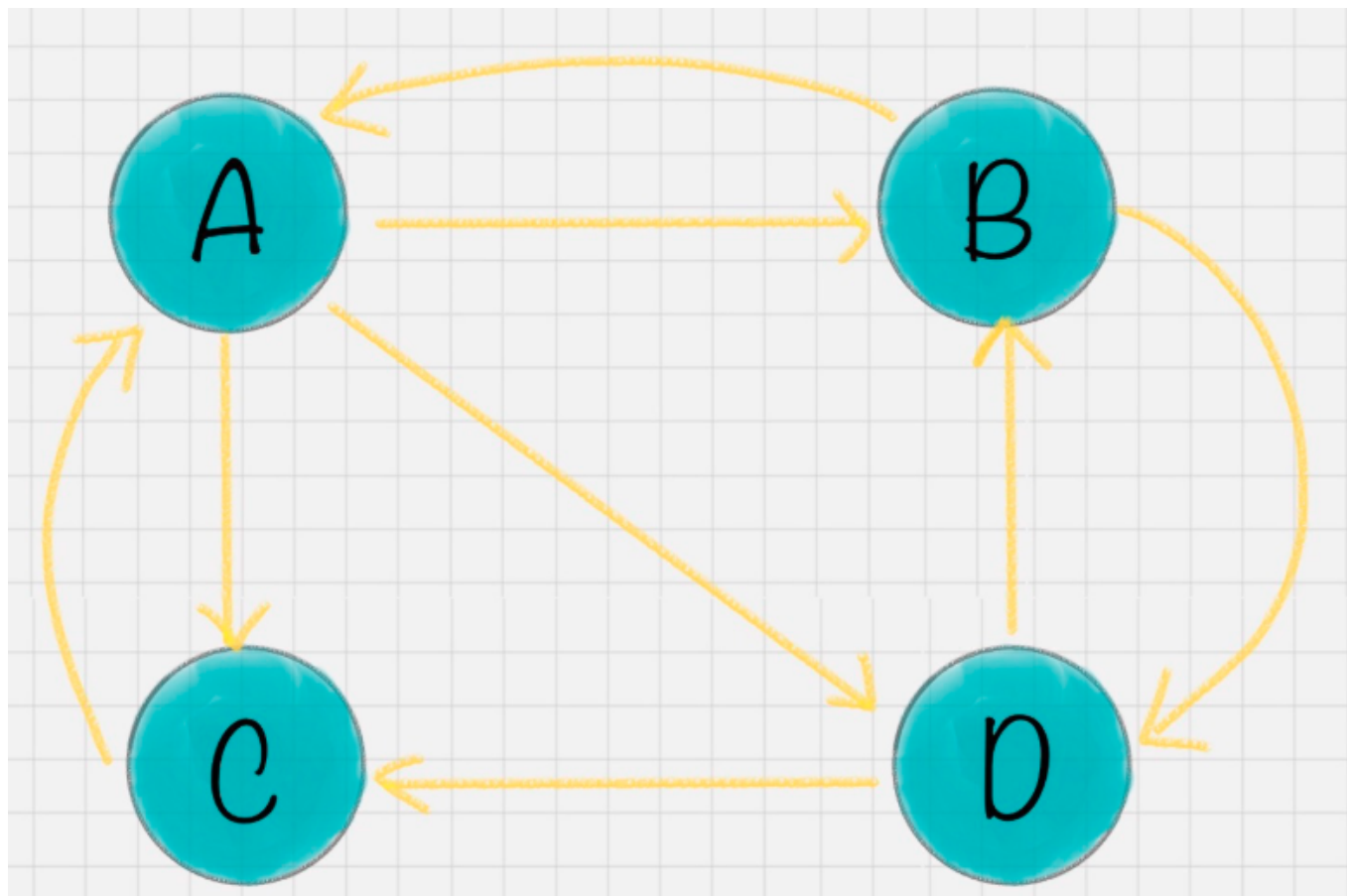


图 5: pagerank的图例。

现假设一共有 4 个网页 A、B、C、D。它们之间的链接信息如图5所示。图中出链指的是链接出去的链接。入链指的是链接进来的链接。比如图中 A 有 2 个入链，3 个出链。一个网页的影响力 = 所有入链集合的页面的加权影响力之和，用公式表示为：

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)} \quad (1)$$

其中  $u$  为待评估的页面， $B_u$  为页面  $u$  的入链集合。针对入链集合中的任意页面  $v$ ，它能给  $u$  带来的影响力是其自身的影响力  $PR(v)$  除以  $v$  页面的出链数量，即页面  $v$  把影响力  $PR(v)$  平均分配给了它的出链，这样统计所有能给  $u$  带来链接的页面  $v$ ，得到的总和就是网页  $u$  的影响力，即为  $PR(u)$ 。出链会给被链接的页面赋予影响力，当我们统计了一个网页链出去的数量，也就是统

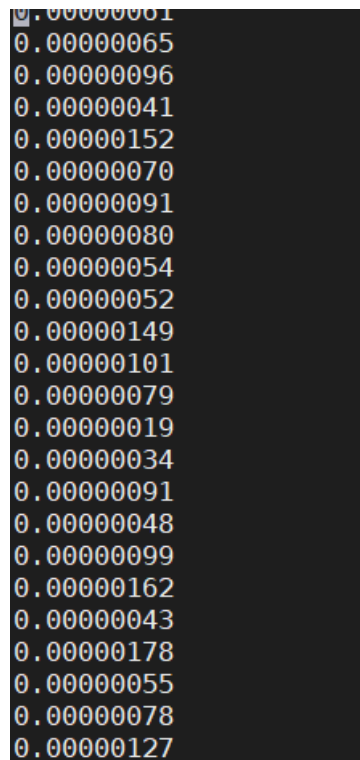
计了这个网页的跳转概率。在这个例子中，你能看到 A 有三个出链分别链接到了 B、C、D 上。那么当用户访问 A 的时候，就有跳转到 B、C 或者 D 的可能性，跳转概率均为 1/3。

由于会出现网页没有出链的情况，该网页会吸收其他网页的影响力而不进行释放，最终会导致其他网页的PR值为0。网页也可能出现只有出链没有入链的情况。这样计算过程迭代下来会导致该网页的PR值为0.针对这两种情况，对(1)式进行了更新：

$$PR(u) = \frac{1-d}{N} + \sum_{v \in B_u} \frac{PR(v)}{L(v)} \quad (2)$$

其中 N 为网页总数。

因此需要迭代的计算每个网页的PR值，本次实验对1,024,000个节点，每个节点有随机的1到10个入链的有向图迭代100次进行计算每个节点的PR值，因此需要两层循环，外层循环为100次迭代，而内层循环为更新节点PR值，从2式可以很容易的知道外层循环每次迭代都依赖于上次迭代的结果所以无法使用OpenMP进行并行处理。本次实验过程中只针对内层循环进行并行化处理。得到的部分结果如下图所示。



```
0.00000081
0.00000065
0.00000096
0.00000041
0.00000152
0.00000070
0.00000091
0.00000080
0.00000054
0.00000052
0.00000149
0.00000101
0.00000079
0.00000019
0.00000034
0.00000091
0.00000048
0.00000099
0.00000162
0.00000043
0.00000178
0.00000055
0.00000078
0.00000127
```

图 6: pagerank的实验结果。

从图中可以看出，由于网络过于稀疏，所以最后每个节点的PR值都比较小。实验过程所消耗的时间如下图所示。

```
root@ecs-00:~/OMPPProgram# ./pagerank
Input thread_num:
64
elapsedtime: 20.895130
The result PR is in flie result.txt!root@ecs-00:~/OMPPProgram#
```

图 7: pagerank在64个线程的情况下所消耗的时间。

在实验中分别对，4，16，64个线程进行了实验，所消耗的时间都相差不大，因此在这里只显示64个线程的结果。原因与上面实验一样，由于服务器只有两个核，因此线程数量对实验所花费的时间影响较小。

**实验总结** OpenMP是主要基于共享内存的并行方式，主要作用在程序的for循环处，或程序之间的各个部分。由添加omp注释的方式实现。实现方法较为简单，但不宜与处理线程之间通信过多或数据过于依赖的情况比如PageRank中的外层循环。但是对于普通的循环或数据依赖关系不强的程序块使用OpenMP进行并行化处理较为方便。