实验报告标题

张琴

苏州大学计算机科学与技术学院

Email: 20124527022@suda.edu.cn

摘要：skyline查询是近年来数据库领域的一个研究重点和热点，这主要是因为skyline查询在许多领域有着广泛的应用，现有的工作大部分集中于单处理机环境，然而，由于skyline查询时CPU敏感，在实际的应用中，现有的方法具有很大局限性。基于此，在本文中我们使用了openmp对skyline实现算法的并行，有效提高并行处理skyline查询的效率。实验结果表明，该算法具有有效性和实用性。

关键词：并行算法，openmp，skyline

1 问题描述

近年来，由于skyline算法在多目标决策支持系统、数据库系统、导航系统、数据挖掘等领域的广泛应用，已逐渐成为研究热点。当前，由于功耗、散热等问题，单核CPU性能的提升已到达极限，直接影响到串行算法的执行效率，面对日益增多的数据，为保证skyline计算的高效性，并行计算就成为自然的选择。

1.1 关于openmp

openmp是一种面向共享内存以及分布式共享内存的多处理器多线程并行编程语言，是一种能够被用于显式制导多线程、共享内存并行的应用程序编程接口，其具有良好的可移植性，支持多种编程语言。

openmp的执行模型采用fork-join的形式,如下图1-1：

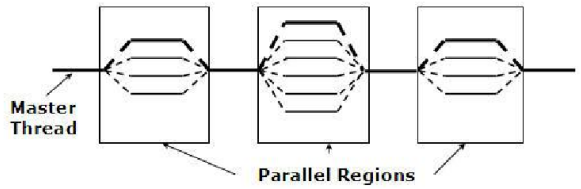

Figure 1-1 openmp执行模型

编写openmp程序的必要步骤有以下六点：

1. 生成项目；
2. 配置项目，使之支持openmp；
3. 编写代码，加入include“omp.h”；

4. 编写源程序；

5. 配置环境变量，确定线程的数目；

6. 执行程序。

1.2 关于skyline

skyline查询问题也称为Pareto最优或极大向量问题，它是指从给定的一个N-维空间的对象集合S中选择一个子集，该子集中的点都不能被S中的任意一个其他点所控制，满足这个条件的点称为SP(skyline point)。这里的控制关系是指给定一个N-维空间中的多个对象（对象集S），若存在这样两个对象$P=\{p_1,p_2\cdots,p_n\}$和$Q=\{q_1,q_2\cdots,q_n\}$,对象P在所有维上的属性值都不比对象Q差，且至少在一维上的属性值优于对象Q，则称P控制Q。

1.2.1 skyline

给定一个d维空间上的数据集$D=\{p_1,p_2\cdots,p_n\},|D|=n,p_i(i=1,2,\cdots,n)$为D中的元组，$p_i[k](k\in[1,d])$表示第i个元组第k维的值，符号“$\lhd$”的含义为“不差于”。“$\lhd$”的含义为“好于”。

定义 1 支配。假定$p_i,p_j\in D$,当且仅当$\forall k\in[1,d],p_i[k]\lhd p_j[k]\wedge\exists t\in[1,d].p_i[t]\in p_j[t]$时，$p_i$支配$p_j$，记作$p_i\prec p_j$。

当上述条件不满足时，称p_i不支配p_j，记作$p_i\not\prec p_j$。若$p_i\not\prec p_j\wedge p_j\not\prec p_i$，则称$p_i$和$p_j$互不支配，记作$p_i\prec\succ p_j$。

定义 2 *skyline*集合。D的*skyline*集合记作$S_{ky}(D)$，由D中不被支配的元组构成，即$S_{ky}(D)=\{p_i\in D|p_i\prec p_j,p_i\in D\}$

性质1 传递性。设$\mathrm{p}_i,p_j,p_k\in D\mathrm{e}\mathrm{p}_i\prec p_j\wedge p_j\prec p_r$,则有$p_i\prec p_r$。

性质2 分配性。设D_1和D_2为2个数据集，则$S_{ky}(D_1\cup D_2)=S_{ky}(D_1)\cup S_{ky}(D_2)$。

性质2说明：要计算$S_{ky}(D_1\cup D_2)$,可以先分别计算$S_{ky}(D_1)$和$S_{ky}(D_2)$，然后对其进行合并。

2 skyline串行算法

skyline查询时选择数据集中不被其他数据所支配的数据，求解skyline结果的算法有DC，BNL算法，BNL算法是对待测元组建立临时表T，T由一个个的临时表T_i组成，要读取的第1组输入放在临时表T_0中，然后再主存中维持一个窗口队列，以收集相互间不存在控制关系的skyline元组，窗口队列初始化为空。算法开始时，第一个读取的元组自然地被放进窗口队列中，然后，每当从当前临时表队列T_i中读入一个元组p时，就用该元组和窗口队列中已有的所有元组一次进行比较。

下图2-2是测试函数所参照的求解支配关系的对比表，GT即（greater-than），GE即（greater-than-or-equal）

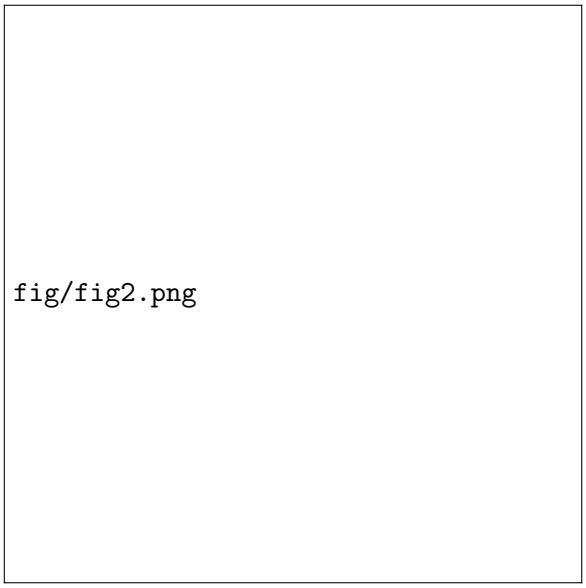

Figure 2-2 求解支配关系的对比表

```
void dominanceTest(int sub[], int* check, int p, int q)
{
    bool GT = false;
    bool GE = true;
    for(int i = 0; i < DIM; i++)
    {
        if(sub[i] > 0)
        {
            GT = true;
            break;
        }
    }
    for(int i = 0; i < DIM; i++)
    {
        if(sub[i] < 0)
        {
            GE = false;
            break;
        }
    }

    if( !GT && !GE )          //right
        *(check + p) = 0;
    if( GT && GE )          //left
        *(check + q) = 0;
}
```

3 skyline并行算法

For命令在多个线程间分割并行迭代空间。for循环的每次迭代都是独立的，因此可以并发地执行。利用skyline查询的支配性质，可以得到这样的一个定理：子数据集中的非skyline点肯定不是全局数据集合中的skyline点，只有子数据集中的skyline点才有可能成为全局数据集合的skyline点。于是我们将要查询的数据集N等分，然后让N个处理器分别处理这N个数据集,找出每个子数据集中的skyline,然后由其中一个处理器负责收集其他N-1个处理器查询的结果，然后对各个处理器获得的结果再次进行一次skyline查询求解，获得最后的查询结果。

并行部分如下所示：

```
1
2 #pragma omp parallel for num_threads(NUM)
3     for(int id = 0; id < NUM; id++)
4     {
5         for(int i = id*EVERYLEN; i < (id+1)*EVERYLEN-1; i++)
6         {
7             if(check[i] == 0)
8                 continue;
9
10            for(int j = i+1; j < (id+1)*EVERYLEN; j++)
11            {
12                int sub[DIM] = {0};
13                for(int k = 0; k < DIM; k++)
14                    sub[k] = data[i*DIM+k] - data[j*DIM+k];
15                dominanceTest(sub,check,i,j);
16                if(check[i] == 0)
17                    break;
18            }
19        }
20    }
```

4 实验结果及分析

通过以下图表??信息我们可以看出openmp对于算法有相当不错的加速效果。分配的线程数分别为2、4、8、16、32，表??清楚的表示了随着分配的线程数增加，算法的计算时间减少，从实验结果中大概可知随着线程数增加一倍，计算时间减少一半，最终趋于稳定；随着线程数的增加，算法的加速比增加，从实验结果中大概可知随着线程数增加一倍，加速比也也相应增加一倍，最终也趋于稳定，不再有大的变化。图??折线统计图和图??柱状表现的更为明显。

Table 4-1 分配不同线程数所对应的计算时间以及加速比

线程数	时间/s	加速比
1	17.926866	1
2	10.345988	1.73273
4	5.025060	3.56749
8	2.747738	6.52423
16	2.322764	7.71790
32	2.169450	8.26332

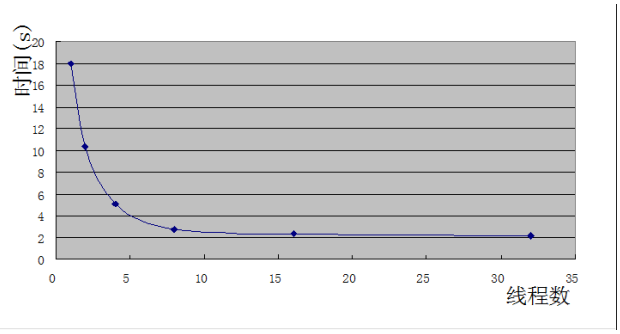

Figure 4-3 分配不同线程数计算时间的折线统计图

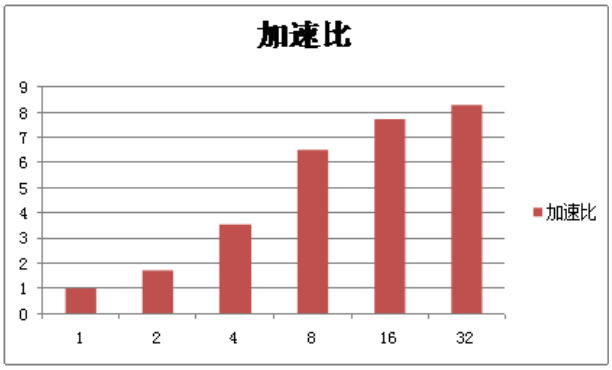

Figure 4-4 分配不同线程数所产生的加速比柱状图

5 总结

为了使openmp程序得到优化，我们通常应注意如下问题：

1. 负载均衡–要非常注意使得线程之间的负载大致均衡，能够让多个线程在大致相同的时间内完成工作，从而能够提高程序运行的效率。

2. 局部性–在编写程序的时候，可以考虑到高速缓存的作用，有意地运用这种局部性带来高速缓存的效率提高。

3. 线程同步带来的开销–在使用多线程进行开发时需要考虑同步的必要性，消除不必要的同步，或者调整同步的顺序，就有可能带来性能上的提升。

References

The DBLP Computer Science Bibliography. `http://www.informatik.uni-trier.de/~ley/db/index.html`, 2011.

Guojie Li. *Parallel Processing of Combinatorial Search Problems.* Ph.D. Thesis, School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN, Dec. 1985.

Paul Johnson and Neeraj Mittal. A Distributed Termination Detection Algorithm for Dynamic Asynchronous Systems. In *ICDCS 2009: Proceedings of the 29th IEEE International Conference on Distributed Computing Systems*, pages 343–351, Montreal, Québec, Canada, Jun. 2009.

王宇，方滨兴，吴博，宋林海，郭岩. 结合属性分布特征的模式匹配算法. 中文信息学报, 24(3):89–96, 2010.