

基于 MPI 的精确非参数检验之 Fisher 网络算法的并行算法研究

刘 晨¹, 胡永宏², 迟学斌¹

(1 中国科学院 计算机网络信息中心, 北京 100190; 2 中央财经大学 统计学院, 北京 100081)

摘 要: 非参数检验是样本为非正态分布或未知分布情况下进行的假设检验, 适应性强, 稳定性好, 在医学、工程、金融等各种统计场合都有广泛应用. 精确非参数检验提高了计算的精度, 但是直接计算精确很费时间. 文中针对精确非参数检验主要使用的 Fisher 网络算法, 基于 MPI 提出并行算法并予以实现. 实验结果表明并行效率良好, 能够大大加快计算的速度.

关键词: 非参数检验; 精确 P 值; Fisher 算法; 网络算法; MPI; 并行化

中图分类号: TP391

文献标识码: A

文章编号: 1000-7180(2010)03-0158-05

Parallel Study of Fisher Network Algorithm for Exact Nonparametric Test Based on MPI

LIU Chen¹, HU Yong-hong², CHI Xue-bin¹

(1 Computer Network Information Center, Chinese Academy of Sciences, Beijing 100080, China;

2 School of Statistics, Central University of Finance & Economics, Beijing 100081, China)

Abstract: Non-parameter test is used for non-normal distribution or unknown distribution samples. It has strong adaptability, good stability, and is widely used in medicine, engineering, finance field etc. Exact non-parameter test improves the calculation precision but takes a lot of time. This article proposes and implements parallel algorithms of exact p-value based on Fisher network algorithm.

Key words: nonparametric test; exact p-value; Fisher's exact test; network algorithm; MPI; parallelism

1 引言

非参数检验对样本的分布形态没有特殊的要求, 可对样本数据的符号、等级程度、大小顺序等进行比较, 其方法简便, 易于掌握, 因而应用及其广泛^[1].

精确非参数检验是非参数检验的精确算法, Fisher 精确算法是其中主要的一种, 适用于小样本, 能大大提高计算精度, 但非常费时^[2-3].

Mehta 和 Patel 在 Fisher 基础上, 提出较为高效的网络算法. 下文的 Fisher 网络算法, 指的就是 Mehta 和 Patel 基于 Fisher 算法基础上的网络算

法^[3-4].

文中基于 MPI, 提并实现出 Fisher 网络算法的并行算法. 本算法通过主从模式针对原有算法的特点, 选择合适的计算粒度进行并行并过去重压缩减少网络传输的数据量以提高并行效率. 实验表明, 平均加速效率超过 70%, 加速效果十分明显.

2 Fisher 网络算法

2.1 Fisher 算法

枚举所有相同行和及列和的矩阵集(服从超几何分布), 所有小于样本出现概率的矩阵的概率之和即为 p 值^[4-6], 形式化定义如下:

收稿日期: 2009-01-20; 修回日期: 2009-04-28

基金项目: 国家“八六三”计划项目(2006AA01A116)

给定一个 $r \cdot c$ 的矩阵 X , 定义: i 行 j 列的元素为 x_{ij} , 第 i 行和 $R_i = \sum_{j=1}^c x_{ij}$, 第 j 列和 $C_j = \sum_{i=1}^r x_{ij}$, 具有同样行和列和的矩阵集合为

$$\Gamma = \{Y : \sum_{j=1}^c y_{ij} = R_i, \sum_{i=1}^r y_{ij} = C_j\}$$

原假设 H_0 : 行与行之间都是互相独立的

Γ 中任一矩阵 Y 出现的概率表示为:

$$P(Y) = \left(\prod_{j=1}^c \frac{C_j!}{y_{1j}! y_{2j}! \cdots y_{rj}!} \right) / \frac{T!}{R_1! R_2! \cdots R_r!},$$

$$T = \sum_{i=1}^r R_i, \text{ 定义 } D = T! / R_1! R_2! \cdots R_r!$$

矩阵 X 的精确 p 值可以定义为: Γ 中所有概率小于样本矩阵 X 概率的矩阵出现的概率总和, 即

$$p = \sum P(Y), \text{ 其中 } P(Y) \leq P(X)$$

2.2 Fisher 网络算法的表述

Fisher 网络算法将枚举矩阵集转化为构造图, 利用广度优先和剪枝的思想构建图, 减少重复的计算提高速度^[4].

按照列的顺序依次确定每列的元素, 共分为 $c+1$ 阶段 $c, c-1, \dots, 0$, 每个阶段确定一列. 每列元素的一种取值记录为一个节点. 在第 k 阶段, 节点均被标示为 $(k, R_{1k}, \dots, R_{rk})$, 其中 $R_{ik} = \sum_{j=1}^k x_{ij}$ 表示剩余的行和.

k 阶段和 $k-1$ 阶段的任一两节点之间的差表示矩阵第 k 列的一种取值, 即 $R_{ik} - R_{i,k-1} = y_{ik}$.

核心思想一是宽度优先, 从 $(k, R_{1k}, \dots, R_{rk})$ 生成所有 $(k-1, R_{1,k-1}, \dots, R_{r,k-1})$ 节点, 并且递归下去直到最后的终点 $(0, 0, \dots, 0)$, 这样完成了整个矩阵集的构造过程.

定义 $C_k! / y_{1k}! \cdots y_{rk}! = C_k! (R_{1k} - R_{1,k-1})! \cdots (R_{rk} - R_{r,k-1})!$ 为节点 $(k, R_{1k}, \dots, R_{rk})$ 到 $(k-1, R_{1,k-1}, \dots, R_{r,k-1})$ 的边长度, 定义从起点到终点的任一路径的路径长度为首尾相连的各边长度乘积, 这样该矩阵的整个路径长度即为 $D \cdot P(Y)$, 这样就将问题从之前 $P(Y) \leq P(X)$ 转化为 $D \cdot P(Y) \leq D \cdot P(X)$, 目标就是找到那些长度不超过 $D \cdot P(X)$ 的路径, 并求出这些路径长度之和(最后除以 D 得到 p 值).

核心思想二是剪枝, 减少不必要的计算量.

首先定义两个变量如下:

$SP(k, R_{1k}, \dots, R_{rk})$ 为点 $(k, R_{1k}, \dots, R_{rk})$ 到终点的所有路径中的最短长度

$LP(k, R_{1k}, \dots, R_{rk})$ 为点 $(k, R_{1k}, \dots, R_{rk})$ 到终点的所有路径中的最长长度

在整个图的所有路径当中, 有这样的一部分路径, 它们在 $c, c-1, \dots, k$ 阶段都经过同样的点, 在 $(k, R_{1k}, \dots, R_{rk})$ 之后才不同, 定义这段相同路径的长度为

$$PAST = \prod_{j=k+1}^c \{C_j! / (R_{1j} - R_{1,j-1})! \cdots (R_{rj} - R_{r,j-1})!\}$$

如果能够满足下面两个任一不等式, 则不用枚举 $(k, R_{1k}, \dots, R_{rk})$ 点之后的所有路径:

$$PAST \cdot LP(k, R_{1k}, \dots, R_{rk}) \leq D \cdot P(X)$$

(1)

$$PAST \cdot SP(k, R_{1k}, \dots, R_{rk}) \geq D \cdot P(X)$$

(2)

如果满足式(1), 则该矩阵子集中所有矩阵的路径长度均不会超过 $D \cdot P(X)$, Mehta 和 Patel 在文献 [4] 中证明, 这些路径长度的和为 $PAST \cdot (S_k! / R_{1k} \cdots R_{rk}!)$.

如果满足式(2), 可以判断路径长度将超过 $D \cdot P(X)$, 对 p 值没有影响.

当均不满足时候, 仅仅保存该点.

2.3 Fisher 网络算法的实现

定义当前阶段节点为 mother, 下一阶段节点为 daughter. 由串行程序流程看出(图1), 每阶段有若干 mother, 每个 mother 下又有若干 daughter, 对于每个 daughter, 又有三种情况, 满足式(1)和式(2)子进行剪枝, 不满足保存该 daughter, 在下一阶段取出将成为 mother. 核心伪代码如下:

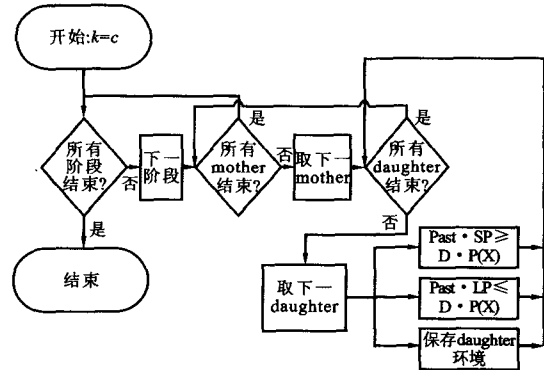


图1 串行程序流程图

for $i=c, i>0; i--$

while(取出下一 mother 成功)

while(生成下一 daughter 成功)

if 满足式(1)

```

sum += PAST * (Sk! / R1k! ... Rrk!);
else if 满足式(2)
    continue;
else
    保存 daughter 状态
end if
continue;
end while
end while
end for

```

3 并行算法设计与实现

3.1 并行算法设计

并程序一般有三种设计模型:数据并行模型、消息传递模型、共享变量模型^[7-9],这里采用应用比较广泛的消息传递模型,并使用消息传递函数库 MPI 进行主从模式编程^[9].首先需要对原有算法进行分析,然后进行设计.原有算法在实际运行中有 4 个特点,下面分析这 4 个特点对并行化的影响:

(1) 在三重循环中:第一重循环,阶段循环的前后依赖严重,在当前阶段的处理过程中,下一阶段的任务是未知的,只有当前阶段完全处理完毕,才可以产生下一阶段的任务.根据这一特点,对阶段循环不采用并行,而是采用顺序执行的方式.

(2) 第二重循环, mother 与 mother 之间可以独立分开,第三重循环同第二重循环一样, daughter 实际上也是互相独立的.根据这个特点,可以在循环的时候将任务分散到各个计算节点中去.又由于为了节省网络时延,加大计算粒度,经过实验测试,仅仅在 mother 阶段进行任务的并行,而 daughter 阶段不进行并行.

(3) 不同的 mother 生成的 daughter 又有很多是重复的,这个特点决定,如果仅仅在 $k = c - 1$ 阶段将 mother 任务分开,之后的阶段每个从节点顺序全部执行完的话,整体的计算量将扩大接近 N 倍,反倒抵消了并行的优势.这个特点决定并行程序必须每个阶段都要汇总以减少计算量.

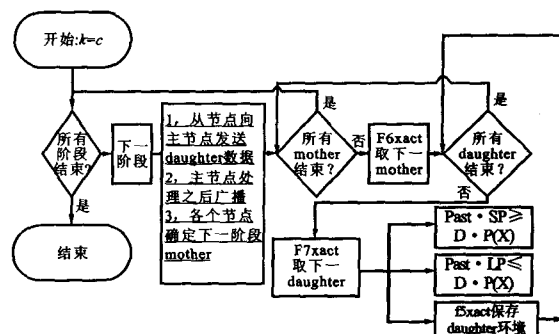
(4) 每个 daughter 处理过程,遇到不满足式(1)和式(2)的时候,都必须将该 daughter 节点及其状态保存起来,在实际运行中的这类保存操作非常多,因此,每个从节点处理 daughter 时候,先将 daughter 及状态保存在自己的节点中,所有任务处理完最后向主节点汇总一次可以减少网络交互的时延.

另外,这里采用压缩去重的办法:在保存 daughter 节点及其状态时,因为相同的情况非常多,

所以进行去重以减少大量的数据.当从节点的 mother 循环阶段完成之后,每个从节点将去重后的数据传递给主节点进行处理.由于不同从节点传递过来的数据仍然有重复的,所以主节点要进行去重处理,然后广播处理后的结果,这样可以将需要传递的数据量大大减少,进而提高了并行的效率.这里的去重是只记录保留一份 daughter,并记录出现的次数.

3.2 并行算法实现

根据串行方法设计并行流程如图 2 所示.



注:下划线字体为并行部分代码

图 2 并行程序流程图

每个阶段完成后进行以下三步处理:

- (1) 从节点将去重后的 daughter 传给主节点,主节点接收之后进行去重处理.
- (2) 主节点再次广播去重后的 daughter 数据.
- (3) 各个节点通过搜索下阶段所有 mother,确定自己要处理的 mother 任务.

核心代码如下(下划线为新增并行部分):

```

for i = c, i > 0; i --
    while(取出下一 mother 成功)
        if(从节点)
            sum = 0;
            while(根据 mother 生成下一 daughter)
                if 满足式(1)
                    sum += PAST * (Sk! / R1k! ... Rrk!);
                else 满足式(2)
                    continue;
                else
                    保存 daughter, 去重压缩保存
                end if
            end while
        if(主节点)
            收集从节点计算结果 sum
            收集从节点 daughter 数组, 去重压缩保存
            向所有从节点广播 daughter 数组
        else(从节点)

```

```
向主节点发送计算结果 sum
向主节点发送 daughter 数组
接收主节点 daughter 数组
end if
根据节点 ID 和 daughter 确定下一阶段 mother
end while
end for
```

4 实验结果及说明

4.1 实验结果

并行的目的是为了加快计算速度,比较并行计算速度的两个重要指标是加速比和效率.其中加速比用来衡量多节点并行平台和单节点计算平台之间相对性能的标准,定义为:

$S(x) = t_s / t_p,$

其中, t_s 为单节点计算平台串行执行的时间,是多节点并行平台并行执行的时间.

效率定义了处理器用于计算的时间占有所有时间(包括计算时间和通信时间)的比例,定义为:

$E = t_s / (t_p * n) = S(n) / n$

这里选取 7×7 矩阵,利用并行化的网络算法求 p 值,分别对串行和并行方法应用了两套并行平台进行了测试.

首先是在已成主流的双核 CPU 系统的笔记本电脑(酷睿 2 双核 T7250, 2GHz, 2G 内存) MPI 环境,测得具体数据如表 1 所示.

表 1 测试结果

CPU 个数	时间(s)	加速比	效率
1	184.73	1	100%
2	104.73	1.764	88.2%

结果显示,在现有双核环境下,使用 2 个 CPU 的时候,整体时间明显减少接近一半,加速效果很明显.

其次是在深腾 6800 超级计算机的 MPI 环境下扩展到四核和八核,测得结果如表 2 所示.

表 2 测试结果

CPU 个数	时间(s)	加速比	效率
1	135.1	1	100%
2	69.0	1.954	97.7%
4	40.6	3.427	85.7%
8	23.8	5.676	70.9%

结果显示,随着 CPU 的增多,整体时间在不断的减少,当使用 4 个或者 8 个 CPU 的时候,整体的加速比可以达到 3.4 和 5.7 倍,加速效果是非常明显的.由此可以相信,在将来多核 CPU 登陆桌面系统之后,

并行化将能够得到更多的应用.

4.2 关于并行粒度的经验说明

在实现并行化的过程中,要注意并行的粒度问题,选取适合自己的粒度.

计算粒度较粗时候,可以降低网络传输在整体计算中的比例,从而更好的提高效率.在本算法中,尽可能的使得计算粒度变的较粗,同时尽可能减少网络传输,这主要体现在以下几个方面:

(1) 对 mother 循环进行并行而非对 daughter 循环.由于 daughter 循环次数太多,而且 daughter 的计算量比较小,计算粒度过细,使得网络传输占整体计算的比例过大,降低了并行的效率.所以我们这里采用对 mother 循环进行并行化.

(2) 从节点遇到不满足式(1)和式(2)的 daughter 时进行去重操作后保存在本地,阶段完成后再统一发送给主节点,而不是每遇到一次就发送一次.这是由于每个阶段过程中,不满足式(1)和式(2)子的 daughter 非常多,频繁的传输数据,一方面由于网络初始化和结束需要一定时间,多次传输将增大这部分的时延,另一方面,频繁和主节点网络通信导致其他节点的等待,也降低了网络的效率,总体看将会在网络传输上占用过多的时间.所以这里采用先收集,最后一次发送的方法.

(3) 从节点进行去重以减少数据传输量,而非直接把原始数据传输.这里也是为了减少网络传输的时延在整体计算中所占的比例.

以上都是加大计算粒度的方法,然而需要注意的是,计算粒度过大有时候也会产生不利的影响.例如在本算法中,还有一种方法,可以使得粒度达到最粗化:在处理 $c-1$ 阶段的时候就根据 mother 将任务分开(c 阶段只有一个 mother,因此无法将任务并行),由各个 CPU 进行独立处理,从 $c-2$ 直到 0 阶段,各个计算节点没有任何交互,都执行原有的串行代码,直到处理完才将结果汇总收集起来.也就是说,这种粒度最粗化的并行方法在第 $c-1$ 阶段把任务分开之后,各个节点就可以独立执行之后的任务,直到最后汇总结果.

这种方法在处理 $r \times 3$ 矩阵的时候非常有效,因为 $r \times 3$ 矩阵在处理完 $c-1$ (就是 2) 阶段之后就结束了, $c-2$ 阶段没有 mother 生成.

但是在处理超过 3 列矩阵的时候,因为不同的 mother 有可能生成相同的 daughter,每个节点处理完 $c-1$ 阶段之后又在整体上加大了计算量,效率反而大大降低,甚至不如串行的效果好.

所以在并行计算中,把握计算粒度的粗细非常重要.任何一个程序,都有许多潜在的并行点,根据这些并行点可以划分不同粗细的计算粒度,只有选择合适的并行粒度才能获得最佳的并行效果.

5 结束语

文中针对 Fisher 网络算法求解非参数检验精确 p 值速度较慢的问题,基于 MPI,设计和实现并行算法,从各种计算粒度中选取了合适的粒度,在并行机上运用 2 到 8 个 CPU 进行数值实验和效率分析,结果表明随着 CPU 的增多,整体的计算时间在不断减少,加速效果非常明显,说明并行化取得了良好的效果.随着小型并行计算平台的使用越来越广泛,在小型工作站基础之上的并行编程必将发挥越来越重要的作用.研究如何利用并行优化技术提升类似于 Fisher 网络算法问题的计算性能具有重要意义.

参考文献:

- [1] 宋昕,蔡泳,徐刚,等.非参数检验方法概述[J].上海口腔医学,2004,13(6):561-563.
- [2] 朱赛楠,马林茂.应用 SAS 软件进行精确非参数检验[J].数理医药学杂志,2003,16(6):41-43.
- [3] SAS Institute Inc., SAS/STAT user's guide[M]. Gary,

NC: SAS Institute Inc., 1999:2532.

- [4] Mehta and Patel. A network algorithm for performing Fisher's exact test in $r \times c$ contingency tables[J]. J. AM. STAT. ASSOC., 1983(78):427-434.
- [5] Mehta, Patel. Algorithm 643 fexact: a fortran subroutine for fisher's exact test on unordered $r \times c$ contingency tables[J]. ACM Transactions on Mathematical Software, 1986, 12(2): 154-161.
- [6] Alan A. A survey of exact inference for contingency tables[J]. Statistical Science, 1992, 7(1):131-177.
- [7] 陈国良.并行计算——结构·算法·编程[M].北京:高等教育出版社,2001.
- [8] 刘玮.一种 SMP 机群下程序并行优化模式的设计与应用[J].微电子学与计算机,2005,22(11):110.
- [9] 赵永华,迟学斌.基于 SMP 机群的 MPI+OpenMP 混合编程模型及有效实现[J].微电子学与计算机, 2005, 22(10):7.

作者简介:

刘 晨 男,(1983-),硕士研究生.研究方向为并行计算、计算金融.

胡永宏 男,(1966-),博士,副教授.研究方向为金融时间序列分析和综合评价方法.

迟学斌 男,(1963-),研究员,博士生导师.研究方向为高性能计算、并行处理.

(上接第 157 页)

- [5] Ge T, Chang J S, Shu W, et al. Modeling an analysis of PSRR in PWM class D amplifiers[C]// ISCAS '06. Kos, Greece, 2006:1386-1389.
- [6] Bruno Putzeys. Simple selfoscillating class D amplifier with full output filter control[C]// 118th AES Convention. Spain: Barcelona, 2005.
- [7] Steven R Norsworthy, Richard Schreier, Gabor C Temes. Delta-sigma data converters: theory, design, and simulation [M]. New York: IEEE Press, 1996:21-38.
- [8] Poulsen S, Andersen M A E. Self oscillating PWM modulators,

a topological comparison[C]// Power Modulator Symposium, 2004 and 2004 High-Voltage Workshop. Conference Record of the Twenty-Sixth International. Denmark, Lyngby, 2004:403-407.

- [9] Behzad Razavi. Design of analog CMOS integrated circuits[M]. Boston, MA: McGraw-Hill, 2001:209-218.

作者简介:

孙煜晴 女,(1981-),硕士研究生,助理讲师.研究方向为数模混合专用集成电路设计.