基于丛生树的多流水线并行 Hash 连接的处理机分配算法*

昌月楼

(国防科技大学计算机系 长沙 410073)

摘 要 本文介绍了并行数据库中实现多流水线 Hash 连接的处理机分配算法，该算法对于执行 Hash 连接的丛生查询树可同时实现流水线内并行 (Intrapipeline Parallel) 和多流水线间的并行 (Intrapipeline Parallel)

关键词 并行数据库，Hash 连接，流水线，丛生树
分类号 TP392

An Algorithm to Allocate Processors among Bush Tree to Get Multiple Pipelined Parallel Execution on Hash Join

Chang Yuelou

(Department of Comoputer Science, NUDT Changsha, 410073)

Abstract An algorithm on how to allocate processors for bush query tree to get multiple pipelined parallel execution with hash join is presented in this paper. An example is given for further description.

Key words parallel data base, hash join, pipeline, bush tree

并行处理多年来一直是计算机领域中一个重要研究课题。随着硬件技术的不断发展和及其成本的不断降低，人们在追求高性能计算的时候，把目标转向了并行体系结构、并行算法和并行软件的研究。近几年来值得注意的是，并行数据库已由过去的原型研究走向商业应用[1][2][3]如 Oracle、Sybase、Informix、DB2 等都提供了并和行功能。并行数据库的研究包括并行装载、并行索引和并行查询，而其中最受关注的是并行查询，因为它是提高数据库性能的关键。并行查询可分为划分并行 (Partitioned Parallel) 和流水线并行 (Pipelined Parallel) 两大类，按并行方式又可分为分类合并 (Sort-Merge)、嵌套循环 (Nested-Loop) 和散列 (Hash)[4]。大粒度的并行是查询间的并行，即同时执行多个查询；小粒度的并行是查询内的并行，即在一个查询内涉及到多个连续的并行。目前研究得最多的，也是难度最大的是后者，本文是针对这一情况而言。

一个查询要求往往被编译成一棵查询树，其叶结点是输入的数据库关系 (表)，中间结点是操作算子。查询树分三类：左深树、右深树和丛生树，前二者被称为线性树。由于 Hash 连接提供了流水线并行的机会，所以在开发并行连接时人们往往对它感兴趣。流水线 Hash 连接有很多步，每一步都可执行连接运算，每一步的内关系 (Inner Relation) 是用来建立 Hash 表的。然而，怎样建立一个有效的查询树以便发挥 Hash 连接的流水线并行功能是一个难题。过去的静态右深树调度算法、动态自底向上调度算法和分段右深树算法[5]是采用简单的启发式算法来将处理机分配给各流水站的。尽管对于分类合并算法来讲，丛生树优于线性树，特别是在一个查询中涉及的关系很多时尤其这样；但是对于 Hash 连接而言，要对丛生树进行处理机调度是很难的，这是因为很难获得为充分利用流水线而需要的同步。正因为这样，过去在流水线 Hash 连接的研究方面往往侧重在右深树，下面的算法就是针对上述问题的。

* 1998 年 3 月 19 收稿
国家 863 计划项目
昌月楼，1945 年生，副教授

1 基于丛生树的多流水线并行 Hash 连接处理机分配算法

1.1 基本思想

为了开发丛生树中Hash 连接的流水线并行, 首先必须能够识别其中的流水线特征, 然后用流水线的方法执行 Hash 连接。然而, 丛生树中的执行依赖使得重组连接工作变得相当微妙和难以处理。本文的思路是首先将丛生树进行变换, 变成另一个规模更小的树, 使得该树中的每一个结点代表一条流水线, 该树可称为分配树, 因为可以利用它进行处理机分配。然后, 利用同步执行时间的概念, 将处理机分配到分配树的各结点上, 分配的原则是要做到流水线内部的各内关系都同时获得, 这样能尽量减少处理机等待时间。执行时间包括 CPU 时间和I/O 时间, CPU 时间由关系中的元组总数乘以处理每个元组所需的 CPU 指令条数来定, 另一个关于CPU 速度的参数MPS 用来计算实际的CPU 时间: $T_{cpu}=(n_t^i)/s$, 其中 n_t 表示元组总数, i 表示处理每个元组所需的 CPU 指令数, s 表示CPU 速度, 即每秒运算指令数。I/O 时间表示为处理一个连接所需的换页次数乘以处理每一页所需的时间。上述两者之和即为执行一个连接所需的时间。

1.2 算法描述

丛生树与右深树的差别体现在: 右深树的左支只含叶结点, 而丛生树的左支至少有一棵子树, 如图 1 所示。

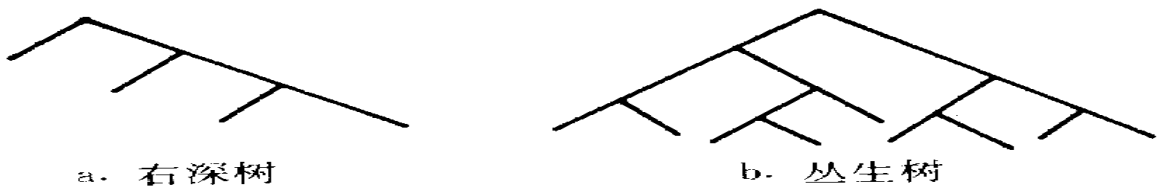

图 1 右深树和丛生树

右深树已成功地用于 Hash 连接的流水线并行中[4], 受这一启发, 我们可将丛生树中有流水线并行潜力的结点加以组合, 如图 2 所示

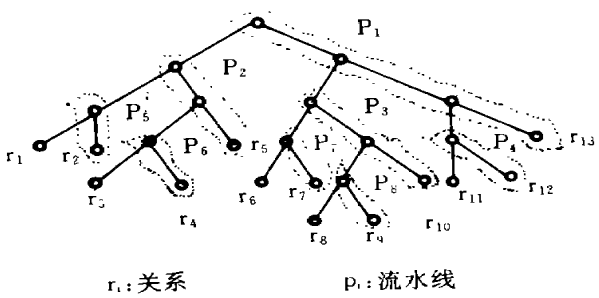

图 2 丛生树的流水线识别

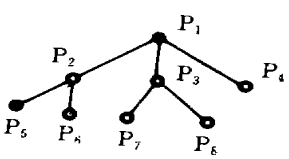

图 3 由图 2 的丛生树转换来的分配树

从此看出, 这里有 13 个关系参加连接, 可识别出 8 条流水线。将每条流水线当一个结点以后, 可得出一棵由丛生树转换来的分配树, 如图 3 所示。

接着要做的是按图 3 的分配树对将处理机分配到各流水线 P_i 上。分配是自顶向下进行的。对图 3 来讲, 因为实际执行连接时是自底向上进行的, 所以 P_1 是执行的最后一步, 这时自然将所有的处理机分配给它。下一步是将 P_1 的所有处理机分配给 P_2、P_3 和 P_4, 分配的原则是使得这三条流水线的操作几乎同时完成, 从而为上一级 (P_1) 的操作提供近视同步输入, 使用这一原则, 可将 P_2 使用的处理机分配给 P_5 和 P_6, 将 P_3 使用的处理机分配给 P_7 和 P_8。下面给出同步执行时间。

定义流水线 P_i 的累计执行时间 CT (P_i) 为该流水线本身的执行时间 E (P_i) 和它的各子树的累计执行时间之和, 即

$$CT(P_i) = E(P_i) + \sum CT(P_j) \qquad \forall P_j \in S(P_i) \tag{1}$$

其中S(Pi)是Pi的子树的集合。执行时间是CPU时间和I/O时间之和，如§1.1所述。累计执行时间的获得是把丛生树进行变换时自底向上计算得到的。分配树的每个结点的累计执行时间也称为同步执行时间，因为正是根据这一时间进行处理机分配从而保征向上一级结点提供近视同步输入。下面讨论处理机分配。

设流水线Px所得处理机数为N(Px)，并设Px是Pi的子结点，则Px应得处理数为:

$$N(P_x) = \left[N(P_i) * \frac{CT(P_x)}{\sum CT(P_j)} \right] \qquad \forall P(j) \in S(P_i) \tag{2}$$

其中S(Pi)是Pi的子结点的集合。这里考虑的原则是分配树中每个结点应得处理机数和它的同步执行时间成正比。

综上所述，上述算法可形式地描述如下:

算法PA: 将处理机分配到使用Hash连接和流水线并行的丛生树。

第一步: 采用连接顺序试探法建立丛生树Tb;

第二步: 从上述丛生树Tb，将可流水线并行部分合并，产生分配树Ta;

第三步: 按照公式(1)，采用自底向上的方法计算Ta中各结点的同步执行时间;

第四步: 按照公式(2)，采用自顶向下的方法计算Ta中各结点的处理机数。

1.3 举例

设有分配树及其各结点的执行时间如图4所示。各结点的同步执行时间如图5所示。设有20个处理机，则各流水线结点所得处理机数如图6所示。

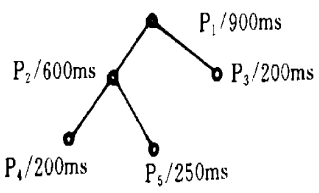

图4 分配树及各结点的执行时间

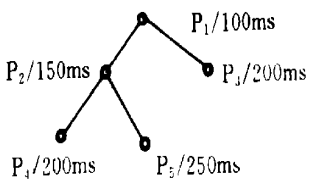

图5 分配树及各结点的同步执行时间

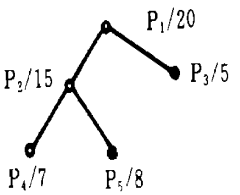

图6 分配树及各结点所得处理机数

2 结束语

多个关系和多个连接的查询要求在实际应用中屡见不鲜，而由这种查询产生的查询树多为丛生树。为了开发并行连接，将丛生树变为右深树，然后执行单一的流水线并行未尝不可。本文提出的这一算法减少了查询树的深度，将流水线并行和各流水线之间的并行结合起来，更能加快查询速度。

为了提高查询速度，Hash连接的内关系要尽可能小，以便能将它尽可能放入内存中。Hui-I Hsiao等提出的Hash过虑算法能减小内关系的大小，进一步提高连接速度[6]。本算法的第一步，即采用连接顺序试深法建立丛生树，也是重要的一步，它影响后面各步，关系到最终执行效率。如何建立一个好的丛生树，这是一个值得研究的问题。此外，本算法中各流水线内处理机的有效分配，也是另一个值得研究的问题。

参考文献

1 DeWitt. D The Gamma DataBase Machine Project. IEEE transactions on Knowledge And Data Engineering, 1990, (2)

2 Oracle Corp. nCUBE for ORACLE-Technology Overview, 1992

3 Stonebrake. M The Case for Shared-nothing. IEEE transactions on Knowledge And Data Engineering, 1986, (9)

4 杨利，李霖，周兴铭. 基于散列技术的并行流水线Join算法的研究与评价. 电子学报 Feb 1996, 24

5 Chen M. S. Lo. M. L, Yu. P. S. and Young. H. C. Using Segmented Right-DeepTree for the Execution of Pipelined Hash Joins. Proceedings of the 18thInternational Conference on Very Large Data Base, Aug. 1992, 15～26

6 Hsiao Hui-I, Chen Ming-Syan and Yu Philip S. On Parallel Execution of Multiple Pipelined Hash Join. SIGMOD 94, May 1994