并行面向对象数据库中的查询优化*

王意洁 胡守仁

(国防科技大学计算机系 长沙 410073)

摘 要 为了在开发并行性的同时, 进一步提高查询处理的效率, 本文针对并行面向对象数据库的异步并行查询执行模型的特点, 提出了三种并行查询优化策略: (1) 数据子集预选策略; (2) 信息流延迟策略; (3) 信息流消减策略。它们既适用于单查询处理, 又适用于多查询处理, 测试结果表明它们是实用有效的并行查询优化策略。

关键词 并行面向对象数据库, 异步, 并行查询执行, 查询优化

分类号 TP311 13

Query Optimization in Parallel Object- Oriented Databases

Wang Yijie Hu Shouren

(Department of Computer, NUDT, Changsha, 410073)

Abstract In order to improve efficiency of query processing while exploring the parallelism, according to the features of asynchronous parallel query processing model in parallel object- oriented databases, three parallel query optimization strategies are proposed. They are: (1) the data subset preselection strategy; (2) the information flow delayed strategy; (3) the information flow reduced strategy. They are suitable for not only the single query but also the multiple queries. The simulation results show that they are efficient practical parallel query optimization strategies.

Key words parallel object- oriented database, asynchronous parallel query processing, query optimization

目前, 许多处于研制阶段的并行面向对象数据库 (POODB) 系统的共同特点是: 借用并行关系库的思想, 采用同步并行查询执行模型。同步并行查询执行模型适于开发对规格化元组进行规范化操作的关系数据库的并行性, 利用它来开发对非规格化对象进行非规范化操作的 OODB的并行性, 将会因系统开销过大而抵消了由并行性所带来的性能提高, 因此, 同步并行查询执行模型不是并行面向对象数据库的最佳选择。AGM[1,2]是一个数据流数据库机, 它把数据库表示成由实体和关系构成的网络, 采用异步方式进行查询处理, 利用处理的异步特性通过消除对每一执行步的集中控制来提高并行度。以 ODMG- 93标准[3]为基础, 借鉴 AGM 的异步处理思想, 我们提出了适于 POODB的异步并行查询执行模型。为了在开发并行性的同时, 更进一步地提高异步并行查询执行模型的执行效率, 从而使 POODB 的性能更理想, 我们提出了相应的查询优化策略。

1 异步并行查询执行模型

异步并行查询执行模型由三个部分组成: 分阶段执行策略、异步并行执行算法和基于对象类的混合式数据放置策略。

* 国防科技预研项目
1997年 9月 5日收稿
第一作者: 王意洁, 女, 1971年生, 博士生

1.1 分阶段执行策略

为了避免访问和处理大量不必要的数据，减少查询处理的时间开销，我们提出了面向对象查询的分阶段执行策略：首先，通过在对象类之间处理和传播 OId，以 OId的形式确定“合格”的对象，也就是查询的前期结果；然后，将系统定义的或用户定义的函数作用于第一步得到的前期结果，从而产生最终结果。为了减少处理和存储中间结果所需的时间和空间，通过对数据库中符合条件的对象实例进行标记来确定“合格”的对象而避免产生大量的中间结果。

1.2 异步并行执行算法

针对面向对象查询的本质特点，我们提出了一种有效的面向对象查询异步并行执行算法[6]，它克服了传统的基于树的并行查询处理方法的局限性和不足，能够较充分地开发各种类型的并行性。查询图中包含的对象类可以分为两种类型：孤立类和非孤立类。孤立类在查询图中只有一条边与其相连，非孤立类在查询图中有多条边与其相连。面向对象查询的异步并行执行算法从所有孤立类开始，它的基本思想是：（设 C为某对象类）

· 预处理：如果查询中包含有关于 C的查询条件，则根据查询条件对类 C中的对象进行处理，对符合查询条件的对象进行标记。

· 如果 C是孤立类，那么根据预处理得到的合格对象和不合格对象的数目多少，向 C的唯一相邻类传送合格对象的 OId流或不合格对象的 OId流以及有关控制信息。如果 C接收到由 C唯一相邻类传送给 C的 OId流和控制信息，那么根据得到的消息以及连接 C与其相邻类的边上的关联符号（“+”或“-”）对 C进行适当处理，从而得到所需的对象，然后结束处理过程。

· 如果 C是非孤立类，

（1）若 C接收到 C的相邻类中除一个以外其它所有类传送给 C的 OId流和控制信息，那么根据接收到的消息以及 C的分支情况（“与”或“或”）和 C与相邻类之间的边上的关联符号，对已经标记的合格对象进行处理并重新标记。然后，根据处理结果将合格的的 OId流或不合格对象的 OId流以及相应的控制信息，传送给那个唯一没有消息传递的相邻类。

（2）若 C接收到来自 C的最后一个相邻类 D的 OId流和控制信息，那么根据接收到的消息以及 C的分支情况和 C与相邻类 D之间的边上的关联符号，对已经标记的合格对象进行处理，从而得到类 C的最终查询结果。然后将相应的 OId流和控制信息传送给除 D之外的 C的其它相邻类。

（3）否则，C根据接收到的 OId流和控制信息以及 C的分支情况和 C与相邻类之间的边上的关联符号，对已经标记的合格对象进行处理并重新标记，但是不发送任何消息。

1.3 基于对象类的混合式数据放置策略

基于对象类的混合式数据放置策略包括两部分：混合式数据划分策略和基于对象类的数据分配策略。

混合式数据划分策略的具体思想是：首先，对数据库中的对象进行垂直划分；然后，在此基础上，再对数据库中的对象进行水平划分。对数据库进行垂直划分之后，产生了两种数据子集：一种是由二元组（OId，属性值）构成的属性值集；另一种是由二元组（OId，OId）构成的关联信息集，它们分别存储对象的属性值和对象与其它对象之间的关联信息。对这两种数据子集基于 OId进行水平划分，便得到了混合式数据子集。基于对象类的数据分配策略的具体思想是：首先，根据各类的工作量和各相邻类之间的通信量，以及类的数目与处理机数目之间的关系，对类进行适当的合并或分离，从而形成若干个类集合，类集合的工作量趋于均衡，类集合之间的通信开销趋于最小，并且类集合的数目与处理机的数目相等；然后，将类集合分配到处理机上，使处理机之间的通信开销趋于最小。

2 并行查询优化

与传统的基于树的并行查询执行算法相比，面向对象查询的异步并行执行算法能获得更高的并行度。然而，高并行度并不一定意味着高效性，因为处理机有可能忙于无意义的工作。而且，在多查询的情况下，需要同时处理大量的查询，与其将所有处理机用于处理一个查询且其中某些处理机为无意义

的工作而繁忙，不如分配一些处理机去处理其它的查询。因此，有必要进行适当的并行查询优化，尽量避免处理机从事无意义的工作，并且尽量减少处理机之间的通信开销，从而提高异步并行执行算法的效率，进一步提高 POODB 的系统性能。

针对异步并行查询执行模型的特点，我们提出了三种并行查询优化策略：(1) 数据子集预选策略；(2) 信息流延迟策略；(3) 信息流消减策略。

2.1 数据子集预选策略

在基于对象类的混合式数据放置策略中，一个类有可能被划分为若干个混合式数据子集并分别存放在不同的处理机上。如果某类的某混合式数据子集不满足该类的查询条件，那么该混合式数据子集所在的处理机与其它处理机之间的 Oid流是没有意义的，而且该处理机对 Oid流的处理也是毫无意义的。查询处理过程中应该避免这些无意义的通信开销和工作量的出现，从而既减少通信开销，又可以使部分处理机从无意义的工作中解脱出来去参与其它查询的处理工作。

基于上述考虑，我们提出了数据子集预选策略，它的基本思想是：把数据分布情况存储于数据库字典中；在查询处理开始之前，将查询条件作用于数据分布情况，从而预选出查询条件范围内的各类的混合式数据子集及其所在处理机并进行适当标记；未标记的混合式数据子集及其所在处理机不参与查询处理。

2.2 信息流延迟策略

定义　类的发送信息量是指在查询图中类向其各相邻类发送的 Oid流的大小的平均值，它是根据类的大小和类与其相邻类之间的关联基数估算的。(设类 C_i 与相邻类 C_j 之间的关联基数为 m_{ij}，m_{ji})

$$Send\ Info(C_i) = \sum_j Size_i \times m_{ij}/N$$

其中：$Send\ Info(C_i)$ 表示类 C_i 的发送信息量；

$Size_i$ 表示类 C_i 所含对象的数目；

N 表示在查询图中类 C_i 的相邻类的数目。

在异步并行执行算法中，根据合格对象和不合格对象的数目多少，选择数目少的形成 Oid流向外发送，这在一定程度上减少了处理机之间的通信开销和处理机的工作量，但没有从根本上最大限度地解决问题。实际上，某类接收到的消息越多，它的合格对象的数目就可能越少，它向外发送的消息也就可能越小(即所含 Oid的数目越少)，对于发送信息量大的类，这种效果更明显。因此，应该将发送信息量大的类的消息发送推迟，这样既可以从根本上减少处理机之间的通信开销，又可以从根本上减少处理机的工作量，将处理机从无意义的工作中解脱出来，使其能够参与其它查询的处理工作，从而提高查询处理的效率。

基于上述考虑，我们提出了信息流延迟策略，它的具体思想是：(1) 对查询图中的每个类估算其发送信息量；(2) 若发送信息量最大的类与发送信息量最小的类之间的发送信息量差额超过给定的阈值，那么将发送信息量最大的类设定为被动类；否则，不作任何设定；(3) 若已经设定了被动类，则将其余的孤立类设定为主动类，将其余的非孤立类设定为弱被动类。显然，孤立类可以是主动类或被动类，非孤立类可以是弱被动类或被动类。

如果信息流延迟策略没有对查询图中的类作任何设定，则按 1.3节中给出的异步并行执行算法进行查询处理；否则，按改进的异步并行执行算法 A 进行查询处理，其中，主动类和弱被动类的处理分别与原异步并行执行算法中孤立类和非孤立类的处理相同。改进的异步并行执行算法 A 的具体思想是：(设 C 为某对象类)

· 预处理：如果查询中包含有关于 C 的查询条件，则根据查询条件对类 C 中的对象进行处理，对符合查询条件的对象进行标记。

· 如果 C 是主动类，那么根据预处理得到的合格对象和不合格对象的数目多少向 C 的唯一相邻类传送合格对象的 Oid流或不合格对象的 Oid流以及有关控制信息。如果 C 接收到由 C 唯一相邻类传送给 C 的 Oid流和控制信息，那么根据得到的消息以及连接 C 与其相邻类的边上的关联符号(“+”或

“-”) 对 C进行适当处理, 从而得到所需的对象, 然后结束处理过程.

· 如果 C是弱被动类,

(1) 若 C接收到 C的相邻类中除一个以外其它所有类传送给 C的 O id流和控制信息, 那么根据接收到的消息以及 C的分支情况 (“与”或“或”) 和 C与相邻类之间的边上的关联符号, 对已经标记的合格对象进行处理并重新标记. 然后, 根据处理结果将合格的的 O id流或不合格对象的 O id流以及相应的控制信息传送给那个唯一没有消息传递的相邻类.

(2) 若 C接收到来自 C的最后一个相邻类 D的 O id流和控制信息, 那么根据接收到的消息以及 C的分支情况和 C与相邻类 D之间的边上的关联符号, 对已经标记的合格对象进行处理, 从而得到类 C的最终查询结果. 然后将相应的 O id流和控制信息传送给除 D之外的 C的其它相邻类.

(3) 否则, C根据接收到的 O id流和控制信息以及 C的分支情况和 C与相邻类之间的边上的关联符号, 对已经标记的合格对象进行处理并重新标记, 但是不发送任何消息.

· 如果 C是被动类, 若 C接收到来自 C的所有相邻类的 O id流和控制信息, 那么根据接收到的消息以及 C的分支情况和 C与相邻类之间的边上的关联符号, 对已经标记的合格对象进行处理, 从而得到类 C的最终查询结果. 然后将相应的 O id流和控制信息传送给 C的所有相邻类.

信息流延迟策略通过改变 O id流的传输次序, 间接地减小了 O id流的信息量, 从而既减少了通信开销, 又减少了处理机对 O id流进行处理所需的时间, 使其有时间去处理其它的查询.

2.3 信息流消减策略

通常, 一个面向对象查询涉及若干个类, 但是只需要检索其中少数几个类的描述信息来构成最终的查询结果, 其余的类是用于判定查询中所涉及的对象间的关联情况. 例如, 查询 1: “找出威斯康辛大学家住纽约第 3号街的所有研究生的姓名”涉及 Department, Graduate, Address, City和 Street五个类, 查询结果仅由 Graduate类的 name信息构成. 那么, 一定要找出 Graduate类的最终合格对象, 而不一定要找出其余四个类的最终合格对象, 只要它们能起到对查询中涉及的对象间关联情况的判定作用即可.

基于上述考虑, 我们提出了信息流消减策略, 它的具体思想是: (1) 若某类的描述信息将用于构成最终的查询结果, 则将该类的状态置为 1; (2) 在查询图中, 若某类与两个状态为 1的类相邻, 则将其状态置为 2; (3) 对其余的类根据它们与状态 1和状态 2的类的距离置状态, 若某类与状态 1或状态 2的类相邻, 则置其状态为 3; 以此类推. 仍以查询 1为例, 状态设置后的查询图如图 1所示.

对查询图中的所有类进行状态设置后, 按改进的异步并行执行算法 B进行查询处理. 改进的异步并行执行算法 B的具体思想是: (设 C为某对象类)

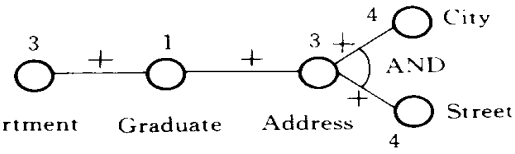

图 1 状态设置后的查询

· 预处理: 如果查询中包含有关于 C的查询条件, 则根据查询条件对类 C中的对象进行处理, 对符合查询条件的对象进行标记.

· 如果 C是孤立类,

(1) 若 C的状态高于 C的唯一相邻类 E的状态或 C与 E的状态均不高于 2, 那么根据预处理得到的合格对象和不合格对象的数目多少向 E传送合格对象的 O id流或不合格对象的 O id流以及有关控制信息; 否则, 只向 E传送控制信息.

(2) 如果 C接收到来自唯一相邻类 E的消息, 若消息中包含 O id流, 那么根据得到的消息以及连接 C与其相邻类的边上的关联符号 (“+”或“-”), 对 C进行适当处理从而得到所需的对象; 否则, 不进行任何处理.

(3) 结束处理过程.

· 如果 C是非孤立类,

(1) 若 C接收到 C的相邻类中除一个以外其它所有类传送给 C的消息, 那么根据接收到的消息以及 C的分支情况 (“与”或“或”) 和 C与相邻类之间的边上的关联符号, 对已经标记的合格对象进行

处理并重新标记. 若 C的状态高于 C的那个唯一没有消息传递的相邻类 D的状态或 C与 D的状态均不高于 2, 那么根据处理结果将合格的的 Oid流或不合格对象的 Oid流以及相应的控制信息传送给那个唯一没有消息传递的相邻类 D.

(2) 若 C接收到来自 C的最后一个相邻类 D的消息, 若消息中包含 Oid流, 那么根据接收到的消息以及 C的分支情况和 C与相邻类 D之间的边上的关联符号, 对已经标记的合格对象进行处理, 从而得到类 C的最终查询结果; 否则, 不进行任何处理, 将已经标记的合格对象作为最终查询结果. 根据 C和 C的除 D之外的其它相邻类的状态, 将相应的 Oid流和控制信息或只有控制信息传送给除 D之外的其它相邻类.

(3) 否则, C根据接收到的消息以及 C的分支情况和 C与相邻类之间的边上的关联符号, 对已经标记的合格对象进行处理并重新标记, 但是不发送任何消息.

显然, 改进的异步并行执行算法 B中只允许状态 1类和状态 2类之间的 Oid流以及从高状态类到低状态类的 Oid流. 信息流消减策略直接消减了不必要的 Oid流, 从而一方面减少了通信开销, 另一方面使处理机免于处理不必要的 Oid流, 而可以腾出时间去处理其它查询.

总之, 三种并行查询优化策略都能起到减少通信开销, 提高查询处理速度的作用. 它们不仅适用于单查询, 也适应于多查询, 而且所带来的时间开销很小. 因此, 它们是实用有效的并行查询优化策略.

3 性能评价

我们在工作站机群系统上基于 PVM 软件平台对异步并行查询执行模型及其并行查询优化进行了模拟实现. 采用的工作站机群系统是由 9台 SGI Indy工作站 (主频: 100MHz, 内存: 32MB, 硬盘: 530MB), 通过 10MB/s Ethernet互连而成. 我们利用文件系统对 POODB的存储系统进行模拟. 选择 0号工作站用于控制算法执行和负责回收结果, 其余 8台工作站用于并行查询处理. 我们对比进行并行查询优化和不进行并行查询优化两种情况下的查询处理效率.

3.1 数据子集预选策略

我们选用的测试数据库 (如图 2) 由五个类构成, A, D, E三个类均含有 5000个对象, B, C两类均含有 2500个对象, 各类的访问概率相同, 各相邻类之间的关联基数相同. 数据放置的结果是: A, D, E三个类各等分为两个大小相同的混合式数据子集, B, C两类和六个混合式数据子集分别位于八个不同的工作站上. 我们选用的查询 (如图 3)涉及五个类. 通过改变查询条件来改变预选的数据子集的数目, 从而观察查询处理时间的变化.

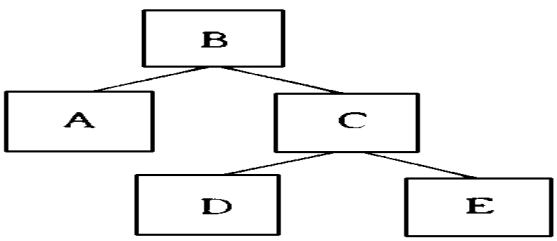

图 2 测试数据库 1

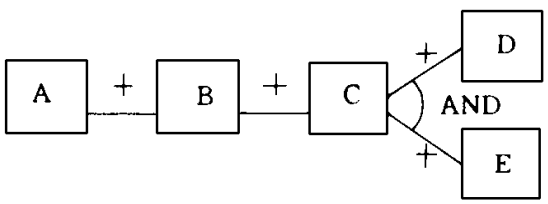

图 3 测试查询 1

图 4反映了两种情况下预选数据子集数目对查询处理时间的影响情况. 测试结果表明, 预选的数据子集的数目越少, 数据子集预选策略的优化效果越明显, 这是因为减少的通信开销和工作量都越多.

3.2 信息流延迟策略

我们选用的数据库 (如图 5) 由六个类构成, A, B, C, D, E五个类均含有 4000个对象, F类所含对象的数目从 500个变化到 4000个. 我们选用的查询 (如图 6) 涉及六个类. 根据信息流延迟策略, A, B, D, E四个类中的任何一个类都可以被设定为被动类. 这里, 我们选择 A作为被动类, 随着 F类所含对象数目的变化, A与 F之间所含对象数目的差额也在变化, 从而进一步观察查询处理时间的变化情况.

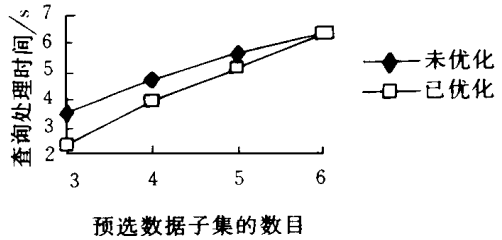

图 4 测试结果 1

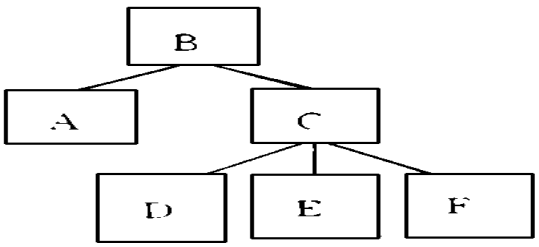

图 5 测试数据库 2

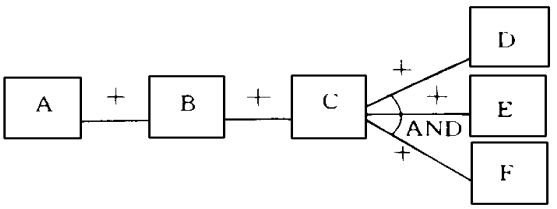

图 6 测试查询 2

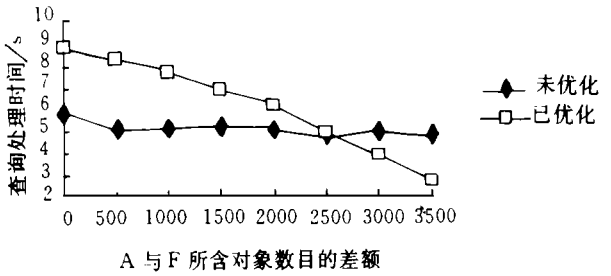

图 7 测试结果 2

图 7反映了两种情况下 A 与 F 之间所含对象数目的差额对查询处理时间的影响情况. 从测试结果可以看出, 当 A 与 F 之间所含对象数的差额较大 (大于 2500) 时, 信息流延迟策略能够较好地提高查询处理速度; 当 A 与 F 之间所含对象数的差额不大时, 由于异步并行查询处理是从所含对象数目较少的主动类开始的, 因而降低了并行度, 从而使单个查询的处理时间变长. 如果只考虑单查询的优化, 从图 7中可以看出, 所含对象数目差额的阈值大约是 3000. 然而, 如果我们考虑多查询的优化, 那么单个查询的处理时间不是衡量并行查询优化策略性能的唯一标准, 不仅要考虑单个查询所占用的处理时间, 而且要考虑它留给其它查询多少处理时间. 因此, 并行多查询处理的目标是寻求单查询的处理时间和系统开销之间的平衡, 也就是寻求并行性与有效性之间的平衡, 从而减少多查询的处理时间. 基于以上目标, 由图 7可知, 所含对象数目差额的阈值应该是 0与 2500之间的某个值.

3.3 信息流消减策略

我们选用的数据库 (如图 8) 由 14个类构成, 每类均含有 3000个对象. 我们选用的查询 (如图 9) 涉及 14个类. 从 14个类中任意选择需检索描述信息的类 (简称检索信息类), 通过改变检索信息类的数目, 从而观察查询处理时间的变化.

图 10反映了两种情况下检索信息类的数目对查询处理时间的影响情况. 测试结果表明, 检索信息类的数目越少, 信息流消减策略的优化效果越明显.

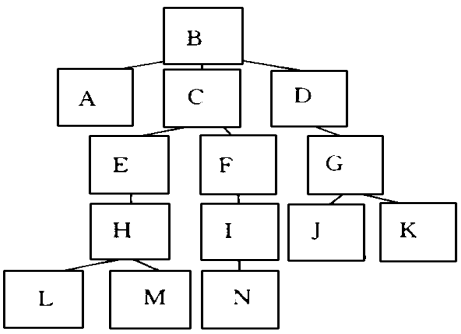

图 8 测试数据库 3

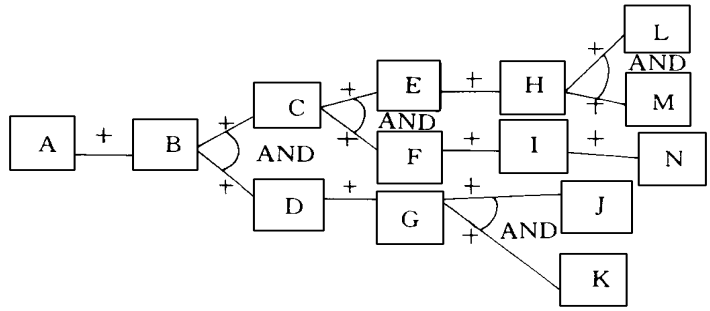

图 9 测试查询 3

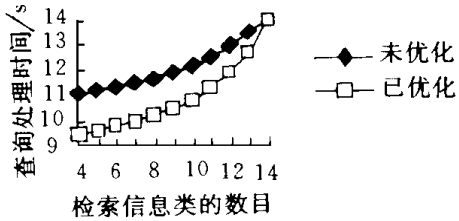

图 10 测试结果 3

4 结论

并行面向对象数据库的异步并行查询执行模型克服了传统的基于树的并行查询处理策略的不足，能够充分地开发各种并行性。为了在开发并行性的同时，进一步提高查询处理的效率，我们针对异步并行查询执行模型的特点，提出了三种实用有效的并行查询优化策略：（1）数据子集预选策略；（2）信息流延迟策略；（3）信息流消减策略。它们不仅适用于单查询处理，而且适用于多查询处理。

参考文献

1 Bic L, Hartmann L R. Simulated performance of a data-driven database machine. J. Parallel and Distributed Computing, 1986, 3 (1): 1~22

2 Bic L, Hartmann L R. AGM: A dataflow database machine. ACM Trans. Database Systems, 1989, 14 (1). 114~146

3 Cattell R. et al. The Object Database Standard: ODMG-93. Morgan Kanfmann, 1994

4 Shahram Ghandeharizadeh et al. Object Placement in Parallel Object-Oriented Database Systems. Proc. of the 10th Int'l Conf. on Data Engineering, Houston, Texas, 1994. 253~262

5 Haddleton R. F, Pfaltz L. Parallelism in Scientific Database Queries. Proceedings of the Eighth International Working Conference on Scienfic and Statistical Database Management, Stockholn, 1996

6 王意洁，胡守仁. 面向对象数据库中的并行查询执行. 计算机学报，1997年增刊