基于谓词代码的编译优化技术研究

田祖伟[1] 孙 光[2]

(湖南第一师范学院信息科学与工程系 长沙 410205)[1] (湖南财政经济学院信息管理系 长沙 410205)[2]

摘 要 程序中大量分支指令的存在,严重制约了体系结构和编译器开发并行性的能力。有效发掘指令级并行性的一个主要挑战是要克服分支指令带来的限制。利用谓词执行可有效地删除分支,将分支指令转换为谓词代码,从而扩大了指令调度的范围并且删除了分支误测带来的性能损失。阐述了基于谓词代码的指令调度、软件流水、寄存器分配、指令归并等编译优化技术。设计并实现了一个基于谓词代码的指令调度算法。实验表明,对谓词代码进行编译优化,能有效提高指令并行度,缩短代码执行时间,提高程序性能。

关键词 编译优化,指令级并行,谓词代码,指令调度

中图法分类号 TP31 **文献标识码** A

Research of Compiler Optimization Technology Based on Predicated Code

TIAN Zu-wei[1] SUN Guang[1]

(Department of Information Science & Engineering, Hunan First Normal College, Changsha 410205, China)[1]

(Information Management Department, Hunan Financial and Economic College, Changsha 410205, China)[2]

Abstract A lot of branch instructions in program severely restrict the exploiting of parallelism of the architecture and the compiler. One of the major challenges to exploit effectively instruction level parallelism is overcoming the limitations imposed by branch instructions. Predicated execution can effectively delete branch instructions, and convert branch instructions to predicated code, which enlarges instruction scheduling scope and removes branch misprediction penalties. This paper described compiler optimization technology based on predicated code, such as instruction scheduling, software pipeline, register allocation, instruction merging. An instruction scheduling algorithm based on predicated code was designed and implemented. The experimental results show that compiler optimization based on predicated code can improve instruction parallelism degree, shorten code execution time effectively, and improve greatly the program's performance.

Keywords Compiler optimization, Instruction level parallelism, Predicated code, Instruction scheduling

1 引言

指令级并行(ILP, Instruction Level Parallelism)是实现高性能微处理器的主要手段,也是编译领域的一个研究热点。对于任何具有指令级并行能力的微处理器而言,使得CPU能够连续不间断地流出多条指令,即保持较高的指令流出率(IPC, Instructions Per Cycle)是充分发挥其性能的关键。但是无论在什么体系结构上,分支指令都是影响这种连续性的一个重要因素。特别是对于采用宽流出、深流水体系结构的现代高性能微处理器而言,分支指令更是一种"昂贵"的指令[1]。分支导致的控制流转移中断了指令流水线的连续性,导致指令Cache的缺失和程序执行的停顿。流水站数越深,分支的延迟就越长,其影响也越大。据统计[2],如果程序的分支频率为20%,对于分支延迟为2拍的微处理器而言,隐含着有40%的机器时间浪费在分支延迟上,而3拍的延迟则浪费了60%的机器时间。更重要的是,据统计[3],程序中分支指令约占整个程序的10%~30%。另一方面,分支指令导致程序被分割成许多基本块,使得指令调度无法并行一些原本能够并行执行的指令。为了减少分支指令的执行代价,谓词执行(Predicated Execution)技术被提出,采用谓词执行技术可以优化程序中的分支条件以减少分支指令的出现数量,改变分支条件与转移目标以提高分支预测的命中率,从而减少分支造成的负面影响,特别是减少那种难以预测的分支指令的开销。

编译器通过IF转换(If- conversion)算法来产生谓词代码,IF转换是一种能有效地支持谓词执行的编译技术。在IF转换生成的谓词代码中,每条指令都被加上一个限定谓词,谓词的值为真,则指令正常执行,否则该指令被作废(被视为一条空指令处理)。体系结构对谓词执行的支持给编译优化带来了许多好处和便利,这使编译器能够充分利用谓词表示,执

到稿日期:2009-07-15 返修日期:2009-09-07 本文受湖南省教育厅优秀青年基金项目(08B014),湖南省科技厅科技计划项目(2008GK3134)资助。

田祖伟(1973—),男,博士生,副教授,主要研究方向为编译优化,E-mail:tianzuwei@126.com;孙 光(1972—),男,博士生,主要研究方向为计算机软件与理论。

行更多的基于谓词的 ILP 优化和控制流变换，如基于谓词代码的编译优化：指令调度、软件流水、指令提升、寄存器重命名、寄存器分配和指令归并等，能够有效地提高其它代码变换技术（例如软流水和模调度）的性能，使得原来受条件分支限制而不得不采取保守策略的编译优化能够以更为激进的方式进行，可有效提高指令并行度，缩短代码执行时间，提高程序性能。

2 谓词代码的产生

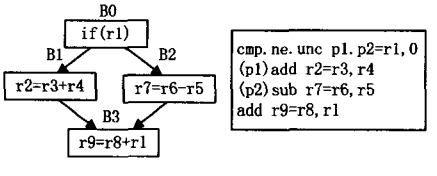
分支指令带来了控制依赖，由于需要根据分支指令的执行结果来确定其后续指令的执行顺序，控制依赖很容易造成流水线停顿，而且会严重限制指令调度范围，同时不确定的分支结果强迫编译器和硬件调度器作出保守的调度决策，从而限制了 ILP 的提高。传统体系结构通常采用以下两种方法来解决这一瓶颈。

(1)利用循环展开技术，减少一部分分支指令，从而消除控制依赖；

(2)通过分支预测技术，预测分支的目标地址。

但是这些技术都有各自的局限性：首先，循环展开会增加寄存器的压力，在资源有限的情况下，过分地展开会使得寄存器需求快速增长，以致超过可用寄存器数目；其次分支预测需要分支目标缓冲器、分支预测表和相应的控制逻辑等硬件资源，并且分支预测不可能 100% 准确，预测失败（Misprediction）引起的开销可能很大。可见传统体系结构对于分支指令对性能的影响没有得到很好的解决，尚未充分发掘出指令间的并行性。分支指令是阻止我们发掘指令级并行性的一大障碍[4]。近期的研究表明，谓词执行可以有效地删除指令流中的分支指令[5,6]，并且帮助消除循环中的条件分支，增加软件流水优化的机会[7]。

谓词执行是 IA-64 等体系结构支持的一种执行方式，它提供了除分支指令之外的、支持条件执行指令的一种新方法，即谓词执行为每条指令增加一个布尔类型的源操作数作为指令执行的条件，这个操作数亦即谓词。运行时指令对应限定谓词的值决定了指令的执行方式，当限定谓词为真时指令正常执行，否则指令被转换为空操作，并且不允许它修改处理器的状态[4]。图 1 是谓词执行的一个实例，其中 p1，p2 为谓词，第一条指令为谓词定义指令。当条件为真时谓词 p1 被置 1，条件为假时 p1 置 0，p2 总是 p1 的补。分支中的指令 r2＝r3＋r4 与 r7＝r6－r5 的控制依赖现在被转化成了数据依赖，通过 cmp 指令设置 p1 与 p2，分支被删除了，原来的 4 个基本块被合并为一个基本块，扩大了指令调度的范围。



(a)原控制流图　　(b)谓词代码

图 1　IA-64 谓词代码实例

编译器通过 IF 转换（If- conversion）算法来产生谓词代码，谓词执行为编译优化和调度提供了一种有用且有效的模型。使用谓词去监视指令的执行能够减少甚至完全删除分支，分支能够通过 IF 转换过程被删除，转换后分支指令已被谓词计算指令所代替，这样分支和其它指令之间的控制依赖被转换成谓词指令和新的谓词定义指令之间的数据依赖。

3 基于谓词代码的编译优化

通过 IF 转换生成谓词代码之后，给编译优化带来了许多好处和便利，能够使编译器充分利用谓词表示，执行更多的基于谓词的 ILP 优化和控制流变换，如基于谓词代码的编译优化：指令调度、软件流水、指令提升、寄存器重命名、寄存器分配和指令归并等，能够有效地提高其它代码变换技术（例如软流水和模调度）的性能，使得原受条件分支限制而不得不采取保守策略的编译优化得以以更为激进的方式进行[9]。

3.1 基于谓词代码的指令调度技术

指令调度（Instruction Scheduling）的任务是在满足依赖关系、资源约束及硬件施加的其它约束条件的前提下，重新排定指令执行的顺序，提高资源利用率，使多个操作能够并行执行，同时减少因相关引起的处理器停顿，以缩短程序的执行时间。很显然，正确的指令调度是一个语义等价的程序变换。

分支指令严重限制了基本块的体积，许多的研究表明程序中的基本块通常都非常小，平均每个基本块中仅含有 5～9 条指令。在这样大小的基本块中显然指令并行度低。因此如何扩大指令调度范围是指令调度的关键。IF 转换是扩大指令调度范围的一种主要方法。IF 转换把程序的控制流分支转换成带谓词的指令，这样就把控制依赖转化成对谓词的数据依赖，形成更大的基本块，使编译器有了更大的调度空间，同时也扩大了数据和控制投机的范围，从而可以提高指令的并行度[10]。

一方面，IF 转换完成之后，通过在数据流分析过程中收集谓词之间的关系并进行分析，可识别并消除表面存在的数据依赖，更进一步地发掘指令级并行。在传统的指令级并行中，指令调度只是把串行的指令按照它们执行的串行顺序在保守的原则下进行并行执行，这种指令级的并行性留下了一个时钟周期的很多空槽，并没有充分利用处理器的高效性能。而在 IA-64 体系结构中，通过引入谓词执行，指令级的并行性可以大大提高，通过查询谓词之间的关系，确定指令之间的数据依赖关系，没有依赖的指令可以并行执行，从而填补了时钟周期中的可能的空槽，指令级的并行性加强了，处理器的性能也得到了高效的发挥。例如有以下指令：

①cmp. le p,q＝y,0

②(p) x＝x＋y＋1；

③ (q) x＝x－y＋1；

在传统的指令调度中，发现这两条指令对变量 x 定值，它们之间有数据依赖关系，所以不能被调度到一个时钟周期中并行执行，即使在上一个时钟周期存在多个指令空槽，且处理器也是空闲的。但是这样的两条指令在调度的过程中显然是可以被调度到一个时钟周期并行执行的，因为经过谓词分析得出，谓词 p 和谓词 q 之间的关系是互斥的，即谓词 p 为真，则 q 为假，反之亦然，所以它们在执行的过程中只会有一条指令会产生执行结果，因此可以并行执行。IF 转换后带来的好处就是不会因为指令之间表面存在数据依赖关系而降低了指令级的并行性。

另一方面，分支带来的控制依赖限制了编译器移动改变存储器状态的指令的能力，否则会导致异常。为了减少指令

执行在关键路径上的延迟时间，IA-64 体系结构增加了对控制投机和数据投机的硬件支持。编译器能够依靠投机来使操作提前发射，以减少这些操作可能在关键路径上引起的延迟，使编译程序能够提高指令级并行度，最大限度地减少内存延迟的影响。投机是编译器中能够扩大指令级并行，从而更大限度地容忍指令延迟的基本机制之一。控制投机是 IA-64 编译器中实现全局代码移动的主要机制，然而，当一条指令可能会破坏存储器状态时，传统的控制投机就无能为力了。在这种情况下，可给指令加上谓词，并在调度时向上移动或向下移动以减少相关深度，改善指令调度的性能。

给定 IA-64 的汇编序列如下，其中的存储指令不能移到条件指令之上，因为在分支走向不确定的情况下有可能会导致地址错误或其它异常。利用 IF 转换产生的谓词执行则可避免这种不确定性。

```
cmp. ne p1,p2=r1,r2
(p1)br. cond some_lable
st4 [r34] = r23
ld4 r5 = [r56]
ld4 r6 = [r57]
```

但是如果能将 stores 指令向上移动到分支指令之前则能释放一个 M-unit slot，这样可允许其下的 load 指令上移。通过使用另外一个谓词 p2 作为 p1 的补充，重写上述代码使得 stores 移于分支前：

```
cmp. ne p1,p2=r1,r2
(p2)st4 [r34] = r23
(p2)ld4 r5 = [r56]
(p1) br. cond some_lable
ld4 r6 = [r57]
```

指令提升(Instruction Promotion)常用于在相关的谓词值知道前投机执行指令。谓词指令的提升将删除谓词指令和谓词定义指令之间的流依赖。通过删除谓词相关的指令可以实现指令的投机执行。有较长延迟的指令可以通过这种技术提前调度。通过提早发射长延迟的指令，其它依赖于此指令的执行结果的指令也可以提早发射。

图 2(b)显示了一个指令提升优化的例子。被提升的指令用黑体显示，投机执行这条被提升的指令将允许这条指令在谓词定义指令之前调度。通过提早调度这条 LOAD 指令，其它使用 LOAD 指令结果的指令也能提早调度，这将产生更高性能的紧凑代码。

由于指令的目的操作数依赖于选择的分支路径，许多指令不能被提升和投机执行。一个解决方法是重命名指令的目的寄存器，并且重命名后面依赖于这条指令定值的所有源寄存器。如图 2(c)所示，被提升指令的目的寄存器 R4 被重命名为 R5，随后使用 R4 的指令也用 R5 来代替。寄存器重命名和指令提升相结合可以用来投机执行更多的指令。

```
ld r1, (a)
cmp. le p1, p2=r1, 0
(p2)add r2=, r1, 1
(p2)ld r3, (B)
(p2)ld r4, (r6)
(p2)add r4=r4, r2
(p1)add r4=r4, 1
(p1)add r2=r1, 1
br. gt r4, 015
sub r4=r4, 1
st(r4), r2
```

(a)

```
ld r1, (a)
cmp. le p1, p2=r1, 0
(p2)add r2=, r1, 1
ld r3, (B)
(p2)ld r4, (r6)
(p2)add r4=r4, r2
(p1)add r4=r4, 1
(p1)add r2=r1, 1
br. gt r4, 015
sub r4=r4, 1
st(r4), r2
```

(b)

```
ld r1, (a)
cmp. le p1, p2=r1, 0
(p2)add r2=, r1, 1
ld r3, (B)
ld r5, (r6)
(p2)add r4=r5, r2
(p1)add r4=r5, 1
(p1)add r2=r1, 1
br. gt r4, 015
sub r4=r4, 1
st(r4), r2
```

(c)

图 2　指令提升与寄存器重命名

3.2　基于谓词代码的软件流水技术

软件流水(SoftWare Pipelining)是一种能够改善支持指令级并行的循环执行性能的编译优化技术，它通过并行执行来自多个循环体的操作来达到类似于将循环完全展开的效果，使得在一个循环迭代执行完毕以前，下一个循环迭代就开始执行。这种机制跟处理器功能部件的硬件流水线十分相似，因此被称为软件流水。启动相邻循环迭代相隔的时钟周期数被称作启动间距(II, Initiation Interval)。软件流水就是要为核心找到一个最小的启动间距 MII(Minimal Initiation Interval)，使得相继的循环体以 II 顺序启动时满足以下两个限制条件：

(1)不发生资源冲突；

(2)不发生相关冲突，包括循环体内的相关关系和循环体间的相关关系。

模调度是一类重要的软件流水调度算法。大量关于软件流水的研究工作是在模调度的框架下进行的，Rau 等在 20 世纪 80 年代初提出的单遍模调度算法只能处理循环界确定的 DO 循环，且循环体只能包含单个基本块，资源预留表必须是简单型的，这些过于苛刻的条件限制了模调度的应用。循环中的条件分支引起在调度时要考虑多条可能的执行路径的问题，增加了软件流水算法的复杂性，同时也降低了其有效性。循环中的条件分支是影响软件流水算法的一个主要障碍。随着支持谓词执行的体系结构的出现，利用 IF 转换消除循环中的条件分支，将控制依赖转换为数据依赖，同时段谓词允许循环内核不使用显式的 prologs 和 epilogs，可提高软件流水算法的性能。Warter 等人的实验表明，使用谓词的循环软流水平均速度比不使用谓词的快 34%[7]。特别是 IA-64 体系结构支持谓词寄存器旋转，更进一步提高了软流水的有效性。

3.3　基于谓词代码的寄存器分配技术

寄存器分配(Register Allocation)和指令调度是编译器提高指令级并行的两个最关键的优化技术。寄存器分配的任务是确定哪些变量的值存放在寄存器中和放在哪个寄存器中，并尽量减少溢出代码。谓词执行为发掘指令级并行提供了新的途径，同时给编译过程中流分析和优化带来新的挑战。对于非谓词代码，控制流图指明了每条指令的读取条件(Fetch Condition)，在不考虑投机执行的情况下，指令读入就一定会执行，读取条件等价于执行条件(Execution Condition)。为保证指令调度和数据流分析结果的正确性，必须考虑指令在控制流图中与其它指令的相对位置。这样在整个编译过程的关键阶段指令调度和寄存器分配都产生了过于保守的代码。而在支持谓词执行体系结构中，指令只有被读出且其限定谓词为真时才会执行。而在控制流图中仅指明了指令的读取条件，这样，为了给支持谓词执行的芯片生成高质量的代码，编译器必须同时考虑指令的读取条件及限定谓词的值，通过在数据流分析过程中收集谓词之间的关系来分析活跃变量，消除假数据依赖，构造更加精确的干涉图，提高寄存器分配的有效性。

在处理器执行指令的时候，其中很大一部分时间开销就是访问存储器的延迟，怎样使寄存器能够得到有效的分配，最大限度地减少处理器的访存次数，一直以来在编译技术领域是一个重要的研究方向。在 IA-64 体系结构中，应用谓词分析技术并且把谓词分析得到的结果应用到寄存器分配当中，

使有限的寄存器得到更加有效的分配，充分利用系统资源，提高整个系统的处理性能。比如有这样一段代码：

```
S1：p,q=cmp. un. uc(a<b)   if true
S2：x=…                    if p
S3：y=…                    if q
S4：…=x                    if p
S5：…=y                    if q
```

指令 cmp 对谓词 p,q 进行初始化，变量 x 在限定谓词 p 的控制下被定义和引用，变量 y 在限定谓词 q 的控制下被定义和引用。根据传统的活跃变量分析，编译器会发现变量 x 的活跃区间是从 S2 到 S43 而变量 y 的活跃区间是从 S3 到 S5。基于这种活跃变量分析，传统的寄存器分配方案得出的结论是变量 x 和变量 y 是互相干涉的，所以给它们分配不同的物理寄存器。但是经过谓词分析之后，因为谓词 p 和谓词 q 是互补的，变量 x 和变量 y 不会同时有效，所以它们可以共享一个物理寄存器。通过 IF 转换并应用谓词分析得到的结果，可以更有效地分配寄存器，充分利用系统的资源。

3.4 基于谓词代码的指令归并技术

基于谓词的指令归并(Instruction Merging)可用于删除冗余的操作，用于将 Hyperblock 中的具有互补谓词的指令复合成一个单一的指令，这样不管谓词的值是 TRUE 还是 FALSE 都将被执行。如果两条相同的指令中源寄存器的定值相同且具有互补的限定谓词，这样的两条指令可以被合并。这可以通过提升一条指令并删除另一条指令来实现。指令归并与删除部分冗余的方法是相似的。图 2(c)的两条 add r2,=r1,1 指令可以被归并，如图 3 所示。

```
ld r1, (a)
cmp. le p1, p2=r1, 0
add r2=, r1, 1
ld r3, (B)
ld r5, (r6)
(p2)add r4=r5, r2
(p1)add r4=r4, 1
br. gt r4, 015
sub r4=r4, 1
st(r4), r2
```

图 3 指令归并的例子

4 算法实现及实验结果

根据谓词代码的特点和以上的分析，设计并实现了一个基于谓词代码的指令调度算法，描述如下：

```
Scheduler_predicate_based( )
{ Init();
Create_region() //创建调度区域
Create_predicate_relation_database();//建立谓词关系数据库
Build_DAG(); // 对每条谓词指令，查询谓词关系数据库，/确定
                限定谓词之间的关系，剔除假依赖；
Scheduler(); //区域调度算法
Instruction_promotion(); //基于谓词的提升算法；
Register_rename(); //基于谓词的寄存器重命名；
}
```

测试平台是在一个配置有两个 1.6GHz 主频、双 Itanium 2 处理器的服务器上进行的，每个处理器有 3 级 Cache，其中第一级 Cache 是分离的指令 Cache 与数据 Cache，两者都是 156k，第二级和第三级 Cache 不区分指令 Cache 与数据 Cache，第二级 Cache 为 256k，第三级 Cache 为 6M。安装操作系统为 Linux version 2.4.18-e.12 SMP，测试对象是嵌入了谓词代码优化算法的 GCC 3.4 编译器和未嵌入谓词代码优化算法的 GCC3.4 编译器，测试工具是 SPEC 2000 中的测试包。

bzip2 是一个文件压缩工具(见表 1)，程序由两个.c 文件和一个.h 文件构成。

表 1 bzip2 在两种情况下运行时的性能比较

算法	编译时间	源文件大小	目标文件大小	运行时间
嵌入谓词代码化算法	1.9379s	131882B	49265B	27.238s
未嵌入谓词代码化算法	1.541s	131882B	56132B	30.862s

对测试数据的分析还发现，嵌入谓词代码优化算法后不仅有助于提升运行时性能，而且得到的可执行文件的长度比未嵌入改进算法编译时得到的要小。主要原因是通过指令提升和指令归并后可删除冗余操作，生成更紧凑的代码。

结束语 现代处理器的性能越来越依赖于它每拍执行多条指令的能力。为了发掘处理器性能的潜能，需要提高程序中的指令级并行性，谓词执行技术能够有效提高 ILP 开发效果。研究表明，合并多个控制流能够将指令级并行增加到原来的 3 倍，能够有效提高其它代码变换技术(例如软件流水和模调度)的性能，而且谓词执行技术能够平均消除 27%的分支以及 56%的分支预测错误[1,8]。可以看出，谓词执行技术是一种有效解决分支瓶颈的技术。

本文针对 IF 转换后生成的谓词代码，系统地研究了与其相关的编译优化问题。研究表明可对谓词代码进行指令调度、软件流水、寄存器分配、指令提升等一系列的编译优化。实验表明，对谓词代码进行编译优化，提高了指令并行度，缩短了代码执行时间，提高了程序的性能。

参 考 文 献

[1] Monica S L, Robert P W. Limits of control flow on parallelism [C]// roceedings of the 19th Annual International Symposium on Computer Architecture. Gold Coast, Australia, 1992

[2] August D I. Systematic compilation for predicated execution [M]. Urbana-Champaign: University of Illinois at Urbana-Champaign, 2000

[3] Bringmann R A, Mahlke S A, Hank R E, et al. Speculative execution exception recovery using write-back suppression [C]// Proceedings 26th Annual International Symposium on Microarchitecture. Dec. 1993

[4] Park J C, Schlansker M S. On predicated execution[R]. HPL-91-58. Hewlett Packard Laboratories. 1991

[5] Hsu P Y, Davidson E S. Highly concurrent scalar processing [C]// Proceeding of the 13th International Symposium on Computer Architecture. Arizona, USA, 1986

[6] Rau B R. Cydra5 Directed Dataflow architecture [C]// Proc. COMPCON'88. San Francisco, USA, 1988

[7] Waiter N J, Lavery D M, Hwu W W. The benefit of Predicated Execution for software pipelining[C]// Proceedings of the 23rd Hawaii International Conference on System Sciences. Hawaii, USA, 1993

(下转第 138 页)

试验任务结束后，每个用户回答一组测试问题。所有的用户认为 Spreadsheet 具有更强的可用性，其数据即时操作的特点比较适合于无编程经验的用户。所有的用户均认为 BM4 非常复杂，以至于在 MapForce 中难以分清模式间的连线，而在原型系统中以 Spreadsheet 非常直观。用户提供的负面反馈是在完成测试任务过程中原型系统有时会产生映射验证错误。原型系统不是很稳定，在映射验证上需要进一步提高。在下一步工作中从更多的方面使用更多的测试任务来比较、评价原型系统。

5 相关研究工作

Spreadsheet 已经被证明是用户友好的数据处理界面之一，微软的 Excel 被广泛采用也可以作为例证。Tableau[6] 建立在 VizQL[7] 之上，特点是交互的数据可视化，但查询能力有限。Spreadsheet 也用于数据清洗[8]、可视化探索[9]、图像管理[10]等。Witkowski 等人提出扩展 SQL，以使关系数据库管理系统支持 Spreadsheet 方式计算。

有很多在线数据库查询和管理工具也使用 Spreadsheet 方式的界面。Zoho[12] 数据库允许用户导入、创建、查询以及可视化在线数据库。但其查询能力相对有限，只支持简单查询和排序。Dabble[13] 数据库与 Zoho 数据库相似，增加了分组功能。

在关系数据库的可视化查询界面领域也有许多工作。文献[14]采用本体帮助用户构造查询语句。VisTrails[15] 为用户提供可视化界面，增量地修改一种工作流，以操纵数据库。文献[16]提出一种可视化查询语言以支持用户通过操纵模式树查询含有复杂值的关系数据库，其模式树与树模型很相似，但模式树上的操作没有考虑层次模型上的语义。

文献[3]提出一种用于查询关系数据库的 Spreadsheet 方式的操作代数。文献[4,5]提出使用 Spreadsheet 方式构造面向数据操作与聚合的 Mashup。

结束语 采用 Spreadsheet 结构提供直接数据操作能力，界面友好，适合无编程经验的用户构造查询。为解决 Spreadsheet 中处理复杂 XML 数据及查询表示问题，提出一种类 Spreadsheet 结构的基于 XML 模式的信息汇聚框架，以支持可视化显示，访问并操作多 XML 数据源。结构具有以下特点：①使用基于 XML 模式的树模型作为 Spreadsheet 结构与 XML 数据源间的中间模型。在 Spreadsheet 中以嵌套表格形式显示树模型，用户在嵌套表格上的数据操作转换为树模型上的 XQuery 查询，执行 XQuery 查询得到结果，再以嵌套表格形式显示给用户。②以 XML 模式为基础，提出一组简单的 Spreadsheet 操作表示方法，用于表示复杂的 XQuery 查询。③实现了一个 Spreadsheet 模式的 XML 数据源的显示、查询和操作的界面。无编程经验的用户通过界面可以使用简单的复制、粘贴、移动等操作构造复杂的 XML 数据源查询。

Spreadsheet 界面中用户可以直接修改数据源的数据并更新至数据源，但经过多个操作叠加产生的数据并不一定能直接映射到原始数据源，因此将进一步研究动态更新机制。操作的可交互性也是下一步研究的内容。

参 考 文 献

[1] Shneiderman B. The future of interactive systems and the emergence of direct manipulation[J]. Behaviour & Information Technology, 1982, 1(3): 237-256

[2] Shneiderman B. Direct manipulation: a step beyond programming languages[J]. IEEE Computer, 1983, 16(8): 57-69

[3] Liu B, Jagadish H V. A Spreadsheet Algebra for a Direct Data Manipulation Query Interface[C]// ICDE09. 2009

[4] Kongdenfha W, Eenatallah B, Vayssiere J, et al. Rapid Development of Spreadsheet-based Web Mashups[C]// WWW09. 2009

[5] Wang G, Yang S, Han Y. A Spreadsheet-like Construct for Streamlining and Reusing Mashups[C]// ICYCS08. 2008

[6] Tableau software[EB/OL]. http://www.tableausoftware.com/

[7] Hanrahan P. Vizql: a language for query, analysis and visualization[C]// SIGMOD. 2006: 721

[8] Raman V, Hellerstein J M. Potter's wheel: An interactive data cleaning system[C]// VLDB. 2001: 381-390

[9] Jankun-Kelly T J, Ma K-L. A spreadsheet interface for visualization exploration[C]// IEEE Visualization. 2000: 69-76

[10] Kandel S, Paepcke A, Theobald M, et al. The photospread query language[R]. Stanford Univ., 2007

[11] Witkowski A, Bellamkonda S, Bozkaya T, et al. Spreadsheets in rdbms for olap[C]// SIGMOD. 2003

[12] Zoho db & reports[EB/OL]. http://db.zoho.com/

[13] Dabble db-online database[R]. http://dabbledb.com/

[14] Catarci T, Dongilli P, Mascio T D, et al. An ontology based visual tool for query formulation support[C]// ECAI. 2004: 308-312

[15] Scheidegger C E, Vo H T, Koop D, et al. Querying and re-using workflows with vistrails[C]// SIGMOD. 2008

[16] Koch C. A visual query language for complex-value databases [J]. ArXiv Computer Science e-prints, 2006

[17] Dang-Ngoc T, Gardarin G. Federating heterogeneous data sources with XML[C]// Proceeding of IASTED IKS Conf. 2003

[18] 王桂玲. 用户主导的互联网虚拟应用构造原理与方法研究[R]. 中国科学院计算技术研究所, 2009

[19] Altova MapForce[EB/OL]. http://www.Altova.com/MapForce

[20] Stylus Studio[EB/OL]. http://www.stylusstudio.com

(上接第 133 页)

[8] Mahlke S A, Lin D C, Chen W Y, et al. Effective compiler support for predicated execution using the hyperblock[J]. SIGMICRON ew sl., 1992, 23 (122): 45-54

[9] Quinones E, Parcerisa J M, González A, et al. Improving branch prediction and predicated execution in out-of-order processors [C]// Proceedings of International Symposium on High-Performance Computer Architecture, 2007 IEEE 13th Annual International Symposium on High Performance Computer Architecture, HPCA-13. 2007

[10] Faraboschi P, Fisher J A, Young C, et al. Instruction scheduling for instruction level parallel processors [J]. Proceedings of the IEEE, 2001, 89 (11): 1638-1659

[11] 朱朝霞，周云才. 语法分析方法实践教学模式[J]. 重庆工学院学报：自然科学版，2008，22(6)：176-180