文章编号:1001-9081(2007)05-1241-04

应用编译技术优化核外计算程序

李 森[1],张 建[1],张红艳[1,2],许桂艳[1,2],徐大庆[1,2],胡泽林[1],袁 媛[1]
(1. 中国科学院 合肥智能机械研究所,安徽 合肥 230031;
2. 中国科学技术大学 信息科学技术学院,安徽 合肥 230027)
(muyuzio@mail.ustc.edu.cn)

摘 要:阐述了一种适用于核外计算程序的变换技术,它通过联合使用循环变换和数据变换这两种编译优化技术来增强程序的局部性,提高数据存取效率。该方法不仅能优化单独一个嵌套循环,还能同时处理多个嵌套循环。实验结果表明了该方法能显著提高核外计算的性能。

关键词:循环变换;数据变换;局部性;核外计算
中图分类号:TP311.5 **文献标识码**:A

Application of compiler optimizing technology on out-of-core computations

LI Miao[1], ZHANG Jian[1], ZHANG Hong-yan[1,2], XU Gui-yan[1,2], XU Da-qing[1,2], HU Ze-lin[1], YUAN Yuan[1]
(1. *Institute of Intelligent Machines, Chinese Academy of Science, Hefei Anhui* 230031, *China*;
2. *School of Information Science and Technology, University of Science and Technology of China, Hefei Anhui* 230027, *China*)

Abstract: A transformation technique which is suitable for out-of-core program was described. Joint use of loop and data transformations help enhance the locality. This solution can not only optimize a single loop nest, but also handle a sequence of loop nests. The experimental results show that the method significantly improves the efficiency of out-of-core computation.

Key words: loop transformation; data transformation; locality; out-of-core computation

0 引言

越来越多的工程涉及到处理大量数据的问题,比如大规模网络病毒防范的数据库系统、信息检索、超文本和多媒体系统以及地震数据处理等等。在这些程序中,一次用到的数据量可能是1GB到4 000GB之多,即海量数据。内存无法同时容纳如此多的数据,待处理的数据只能存放在硬盘上,并通过某种策略实现内外存数据的交换。由于数据没有全部存放在内存中,我们称这样一种大规模数据计算为核外计算[1]。核外计算要处理的数据量可能超出内存容量,所以数据应该被划分为若干个块,当用到某个数据块时,将其从硬盘读到内存中,然后程序处理该数据块。当该数据块被处理完毕后,它将被重新存入硬盘。所以核外计算的速度很大程度上取决于数据在内外存之间的交换速度也即数据输入/输出速度。在研究这种应用程序时,我们考虑的重点应该放在"内存—硬盘"这一层次,而不是"cache—内存"这一层。为了提高效率,被读入内存的数据块应该尽量被重用。这要求我们必须增强程序的局部性,因为当一个程序具有好的局部性时,它的存取指令大多数数据访问的位置是相同的或者是很接近于刚执行过的存取指令的数据访问位置。

目前很多对核外计算的优化技术都是基于计算过程的重排序,而不考虑重新组织数据(驻留在硬盘上的核外数组)的存储布局。本文阐述了一种综合变换方法,它联合运用循环变换(或称为迭代空间变换)和数据变换(或称为文件布局变换)这两种编译优化技术来获得良好的程序局部性。本文采用的循环变换和数据变换矩阵都是非奇异方阵,并且本文的方法能同时优化多个嵌套循环。

1 解决方案

假设核外计算程序中有一系列嵌套循环,它们都访问程序中声明过的数组(核外数组)。我们的优化步骤如下:

(1) 通过循环分布,循环合并和循环展开等程序变换将这些循环转化为一系列独立的循环。

(2) 构造一个冲突图并在图上识别出连通子图。冲突图是一个二分图,记作(Vn,Va,E),其中Vn是所有代表循环的节点集合,Va是所有代表数组的节点集合,E是所有连接循环节点和数组节点的边的集合。若$v1 \in Vn$,$v2 \in Va$,当且仅当$v1$引用了$v2$时,才存在一条边e连接$v1$和$v2$,且$e \in E$。

(3) 对每一个连通子图:

(a) 对所有嵌套循环,根据*profile*信息计算出它的代价,然后根据这一代价来对这些循环进行排序。

(b) 首先仅用数据变换来优化代价最大的循环;然后对该循环进行迭代空间划分。

(c) 根据上面的排序,对连通子图中的其他每个循环:运用非奇异线性循环变换和数据变换(要考虑到当前已经访问过的数组的布局)来优化该循环,然后对其进行迭代空间划

收稿日期:2006-11-28;修订日期:2007-01-27

作者简介:李森(1955-),女,安徽庐江人,研究员,主要研究方向:人工智能、农业知识工程; 张建(1954-),男,陕西延安人,副研究员,主要研究方向:人工智能、农业知识工程; 张红艳(1982-),女,湖北钟祥人,硕士研究生,主要研究方向:模式识别、智能系统; 许桂艳(1983-),男,安徽安庆人,硕士研究生,主要研究方向:模式识别、智能系统; 徐大庆(1983-),安徽蚌埠人,硕士研究生,主要研究方向:模式识别、智能系统; 胡泽林(1977-),男,江西会昌人,助理研究员,主要研究方向:模式识别、智能系统; 袁媛(1981-),女,安徽肥东人,助理研究员,主要研究方向:智能农业信息技术.

分;将当前已经访问过的数组的布局传播到下一个将被处理的循环。

图 1 用一个例子演示了以上算法中步骤(1) 和步骤(2)的工作过程。左边是由两个嵌套循环组成的一个未经过优化的程序序列。U、V、W、X、Y 是循环过程中访问到的核外数组。第一步,编译器运用循环合并处理第一个嵌套循环,用循环分布来处理第二个嵌套循环;第二步,编译器构造出冲突图,并识别出其中的连通子图。根据冲突图,可以将程序重新划分为两段,这两段程序访问的数组构成的集合没有交集(第一个程序片段只访问 U、V、W 数组,第二个程序片段只访问 X、Y 数组)。由于各个连通子图涉及的数组集合的交集为空,所以步骤(3) 只需要对每一个连通子图对应的程序段分别进行处理。

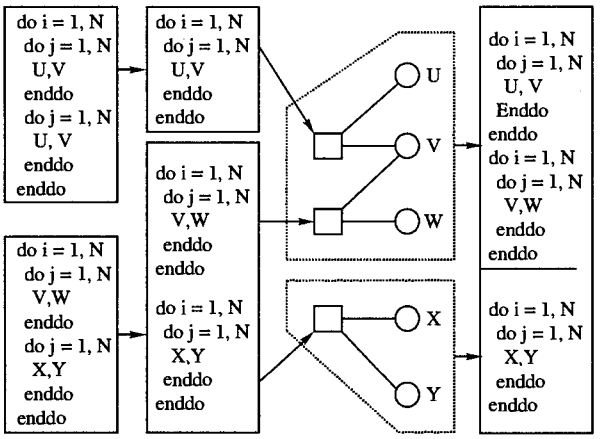

图 1 局部性优化算法实例

1.1 联合使用数据变换和循环变换的必要性

我们通过下面的例子来说明为什么必须联合使用数据变换和循环变换。下面的程序段对应于图 1 中的第一个连通子图;数组 U、V、W 都是驻留在硬盘上的核外数组,所有数组在硬盘上都是按列分布的。

```
do i = 1, N
   do j = 1, N
      U(i,j) = V(j,i) + 1.0
   end do
end do
do i = 1, N
   do j = 1, N
      V(i,j) = W(j,i) + 2.0
   end do
end do
```

如果仅仅使用循环变换,不可能同时对第一个嵌套循环中的两个数组的访问得到良好的空间局部性,因为这两个数组的重用轨迹是正交的;第二个嵌套循环中也存在这样的情况。如果仅仅使用数据变换,将 U 变换为按行分布,W 保持按列分布,但是前后两个嵌套循环对 V 的存储布局的需求产生冲突,不能同时按最优方式来访问。下文主要研究如何联合使用数据变换和循环变换使所有循环中对所有数据的访问都能获得良好的局部性。

1.2 技术细节

1.2.1 超平面和数组布局

一个 k 重嵌套循环对 m 维数组元素的访问轨迹可以用一个 $m \times k$ 访问矩阵 L 和 m 维偏移向量 $\bar{o}$ 来描述。例如在上例中的第一个嵌套循环里对数组元素 $V(i,j)$ 的访问可以用运算式 $L\bar{I} + \bar{o}$ 来表示,其中

$$L = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$\bar{I} = \begin{bmatrix} i \\ j \end{bmatrix}$$

$$\bar{o} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

在 m 维数据空间中,一个超平面可定义为 m 元组的集合:$\{(a_1, a_2, \cdots, a_m) \mid g_1 a_1 + g_2 a_2 + \cdots g_m a_m = c\}$

其中有理数 $g_1, g_2, \cdots, g_m$ 中至少有一不为 0,称它们为超平面系数;有理数 c 称为超平面常数。可以采用行向量 $g^{\mathrm{T}} = (g_1, g_2, \cdots, g_m)$ 来标记一个超平面族(c 取不同的值对应不同的平面)。为了简便起见,本文只讨论二维数组,但是本文的结论都适用于更高维数组。在二维数据空间里,超平面族可以用行向量 (g_1, g_2) 来标记,当 c 取不同值时,它实际上是一组平行线。当硬盘上的 2 个数据点(数组元素)(a,b) 和点 (c,d) 满足以下关系时,他们位于同一超平面上:

$$(g_1, g_2)\begin{bmatrix} a \\ b \end{bmatrix} = (g_1, g_2)\begin{bmatrix} c \\ d \end{bmatrix}$$

例如,对于超平面族(0,1),只要 2 个元素的列坐标相等(行坐标可以不同),则它们就同时属于超平面族(0,1) 中的某个超平面。我们可以用超平面族来标记数组的存储布局。例如对于一个二维数组,我们可以用向量(0,1) 来说明它的每一列(对应于具有特定 c 值的超平面) 的元素被连续存储在硬盘上(此时如果循环程序按列访问这个数组,则该数组将体现出最好的局部性)。可见,向量(0,1) 可用来表示数组按列分布的存储布局。同理,可得(1,0) 可表示按行分布,(1, -1) 可表示按对角分布,(1,1) 表示按反对角分布,等等。

1.2.2 循环变换

一个 k 重循环的迭代空间可以被看作是一个 k 维空间里的多面体,其每个点可以用 k 维向量(迭代向量)$\bar{I} = (i_1, i_2, \cdots, i_k)^{\mathrm{T}}$ 来表示。其中 i_n 代表循环索引变量,并且 i_1 代表最外层循环索引变量,i_k 代表最内层循环索引变量。循环变换可以用一个线性非奇异转换矩阵来表示。对于一个 k 重嵌套循环,其循环变换(迭代空间变换) 矩阵 T 是一个 $k \times k$ 方阵。迭代空间变换将原嵌套循环的迭代向量 $\bar{I}$ 变换成新的迭代向量 $\bar{I}'$,同时也就获得了新的嵌套循环。变换后,对数组元素的访问轨迹可用 $LT^{-1}\bar{I}' + \bar{o}$ 来表示。参考文献[2] 和[3] 讨论了如何选择合适的变换矩阵 T 来改善数据访问的局部性,同时确保不改变原先嵌套循环中存在的数据依赖关系。例如在一个 2 重嵌套循环中,循环变换可以由非奇异变换矩阵来表示:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

1.2.3 循环变换和数据变换的结合使用

下面的定理说明了在对嵌套循环中的数组进行访问时,为了使得最内层循环对数组的访问能获得最好的空间局部性,循环变换矩阵、超平面和访问矩阵之间应满足的关系,定理的证明见参考文献[4]。

定理 设 k 重嵌套对二维数组的访问轨迹为 $L\bar{I} + \bar{o}$,其中

$$L = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \end{bmatrix}$$

循环转换矩阵的转置矩阵为:

$$Q=\begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1k} \\ q_{21} & q_{22} & \cdots & q_{2k} \\ \vdots & \vdots & & \vdots \\ q_{k1} & q_{k2} & \cdots & q_{kk} \end{bmatrix}$$

为了使得最内层循环对该数组的访问能获得最好的空间局部性,代表数组存储布局的超平面(g_1,g_2)应满足关系式:

$$(g_1,g_2)L(q_{1k},q_{2k},\cdots,q_{kk})^{\mathrm{T}}=0 \qquad (完)$$

对于(g_1,g_2)和$(q_{1k},q_{2k},\cdots,q_{kk})^{\mathrm{T}}$,只要其中一个已知,很容易就能将另一个求出。如果矩阵Q的最后一列已知,则

$$(g_1,g_2)\in Ker\{L(q_{1k},q_{2k},\cdots,q_{kk})^{\mathrm{T}}\} \qquad (1)$$

同理,如果(g_1,g_2)已知,则

$$(q_{1k},q_{2k},\cdots,q_{kk})^{\mathrm{T}}\in Ker\{(g_1,g_2)L\} \qquad (2)$$

通常情况下求出的Ker集含有多个向量,我们选择其中一个向量,它的各个元素的最大公约数应该最小。再看 1.1 小节的那个例子,可以发现第一个嵌套循环的访问矩阵是

$$L_U=\begin{bmatrix}1&0\\0&1\end{bmatrix}$$

和

$$L_{V1}=\begin{bmatrix}0&1\\1&0\end{bmatrix}$$

第二个嵌套循环的访问矩阵是

$$L_{V2}=\begin{bmatrix}1&0\\0&1\end{bmatrix}$$

和

$$L_W=\begin{bmatrix}0&1\\1&0\end{bmatrix}$$

按前面所说的优化步骤,对第一个嵌套循环仅施加数据变换,已知$(q_{12},q_{22})^{\mathrm{T}}=(0,1)$。对数组$U$,运用上面给出的式(1),可得:

$$(g_1,g_2)\in Ker\left\{L_U\begin{bmatrix}0\\1\end{bmatrix}\right\}\Rightarrow(g_1,g_2)\in Ker\left\{\begin{bmatrix}0\\1\end{bmatrix}\right\}$$

其中符合条件的特解是$(g_1,g_2)=(1,0)$,也就是说数组U应当按行分布。同样,对数组V可得:

$$(g_1,g_2)\in Ker\left\{L_{V1}\begin{bmatrix}0\\1\end{bmatrix}\right\}\Rightarrow(g_1,g_2)\in Ker\left\{\begin{bmatrix}1\\0\end{bmatrix}\right\},$$

符合条件的特解是$(g_1,g_2)=(0,1)$,所以数组V应该按列分布。

确定好数组U和V的存储布局之后,我们再继续处理第二个嵌套循环。已经确定数组V的存储布局为$(g_1,g_2)=(0,1)$,运用上面的式(2),可得:

$$(q_{12},q_{22})^{\mathrm{T}}\in Ker\{(0,1)L_{w2}\}\Rightarrow(q_{12},q_{22})^{\mathrm{T}}\in Ker(0,1)$$

一个满足条件的特解是$(q_{12},q_{22})^{\mathrm{T}}=(1,0)^{\mathrm{T}}$。根据参考文献[5]中的方法,可以最终求得:

$$Q=\begin{bmatrix}0&1\\1&0\end{bmatrix}$$

Q^{T}即为循环转换矩阵。最后一步工作就是要确定数组W的最优存储布局。因为Q已知,所以对数组W使用式(1),可得:

$$(g_1,g_2)\in Ker\left\{L_w\begin{bmatrix}1\\0\end{bmatrix}\right\}\Rightarrow(g_1,g_2)\in Ker\left\{\begin{bmatrix}0\\1\end{bmatrix}\right\}$$

这意味着数组W应该按行分布。优化后的最终程序如下:

```
do u = 1, N
  do v = 1, N
    U(u,v) = V(v,u) + 1.0
  end do
end do
do u = 1, N
  do v = 1, NV(v,u) = W(u,v) + 2.0
  end do
end do
```

图2是上面的二维数组U、V、W从没有优化到优化后的核内不同访问形式:

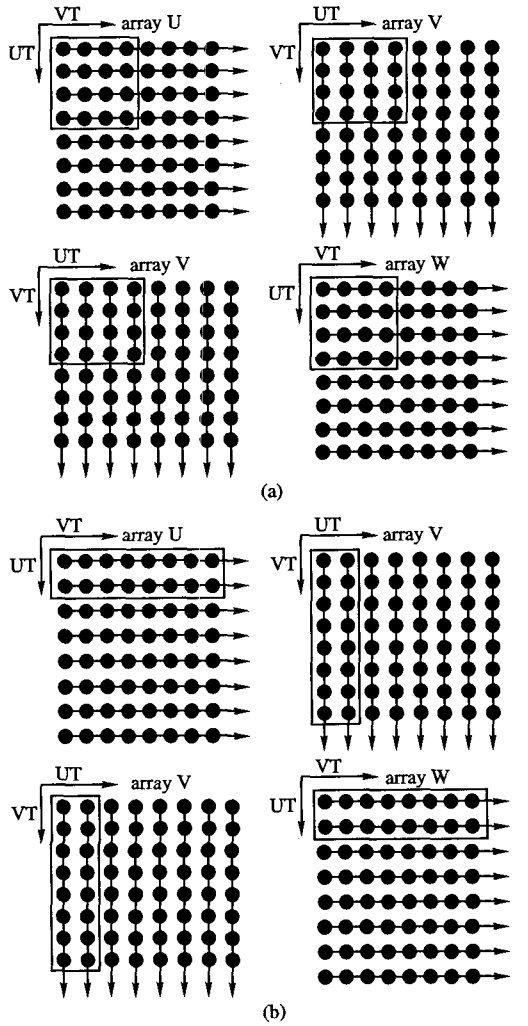

图2　不同的核内访问形式

2　实验结果

我们从几个基准测试程序库和数学计算程序库中选择 6 个测试程序,它们的特点如表 1 所示。我们对这六个程序运用各种变换技术进行优化,分别得到 5 个版本的优化程序:

- col:核外数组固定为按列分布
- row:核外数组固定为按行分布
- l-opt:仅用循环变换来优化程序
- d-opt:仅用数据变换来优化程序
- c-opt:联合使用循环变换和数据变换

其中 col 和 row 版本都是未经过优化的原始程序;l-opt 版本是运用参考文献[3]中算法产生的;d-opt 版本是运用参考

文献[6]中的算法得到的;c-opt 版本是用本文中的方法产生的。表格 2 中列出了将优化后的各版本的程序在具有 16 个 CPU 的 SMP 机器上运行的结果(单位 s)。从这些结果我们可以推断出:仅仅使用循环变换优化 l-opt 的经典局部优化策略效果可能不好,而基于数据变换 d-opt 的性能提高了许多,我们的 c-opt 方法相对于 col 方法减少了 42% 的执行时间,可以看出 c-opt 版本的程序运行效率最高。

表 1 实验中用到的程序

程序	程序来源	循环	数组个数和维数
mxm	Spec92	3 个	3,2-D
adi	livermore	5 个	3,2-D;3,1-D
emit	Spec92	2 个	3,3-D;10,1-D
Syr2k	BLAS	2 个	3,2-D
gfunp	Homepack	3 个	1,1-D;5,2-D
trans	Nwchem	3 个	2,2-D

表 2 实验结果

程序	col	row	l-opt	d-opt	c-opt
mxm	220.0	181.5	100.0	112.6	79.8
adi	144.1	134.9	22.8	45.6	22.8
emit	88.64	176.5	100.0	47.1	100.0
Syr2k	215.3	86.3	52.0	77.4	52.0
gfunp	86.05	128.4	73.3	68.0	46.9
trans	181.9	100.0	100.0	48.2	48.2
Average	155.6	134.6	74.68	66.48	58.28

3 结语

本文阐述了如何联合使用循环变换和数据变换两种编译优化技术来改善核外计算程序的性能。在此过程中运用了大量的线性代数知识,具有理论上的可靠性。实验结果证实了本文提出的方法的优点。

参考文献:

[1] 唐剑琪, 方滨兴, 胡铭曾. 面向工作站机群的核外计算模型及实现[J]. 高技术通讯, 2003, 13(7): 20 - 24.

[2] WOLF M, LAM M. A data locality optimizing algorithm[J]. Proceedings of the SIGPLAN 91 Conference on Programming Language Design and Implementation, Toronto, Canada, 1991. 30 - 44.

[3] LI W. Compiling for NUMA parallel machines[D]. Cornell University, Ithaca, New York, 1993.

[4] KANDEMIR M, CHOUDHARY A, RAMANUJAM J, *et al.* A matrix-based approach to the global locality optimization problem[A]. Proceeding PACT'98[C]. 1998.

[5] BIK A, WIJSHOFF H. On a completion method for unimodular matrices. 94 - 14[R]. Leiden University, 1994.

[6] O'BOYLE M, KNIJNENBURG P. Non - singular data transformations: Definition, validity, applications[A]. Proceedings of 6th Workshop on Compilers for Parallel Computers. Aachen Germany, 1996. 287 - 297.

(上接第 1234 页)

数分别为 6% 和 6。由前面的分析很容易得到,序列长度的增加,准确率会下降。而运行时间呈相反的趋势。这里仅给出运行时间的比较结果(图 5)。

3.2 近邻查询

实现前面的近邻算法,随机抽取 100 个序列进行实验后取平均得到图 6 的结果。

随着查询数目 m 的增加,所得到的阈值也会增加,但三者所得到的阈值相同。同时,查询数目的增加也会使时间增加。从图 6 看到,用 DCT 所花费的时间要少于 DWT,更少于 DFT 所花费的时间。

前面所有的实验都是使用现实生活中的数据。我们也对模拟数据进行了实验,结果与前面的相似,由于篇幅的原因,这里不再一一列出。

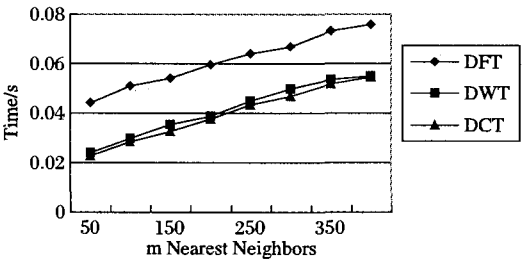

图 6 近邻查询的运行时间比较

4 结语

文中提出了一种更有效的时间序列相似性搜索方法,通过离散余弦变换进行维归约,并在其特征空间上建立空间索引结构来进行相似性搜索。给出了范围查询和动态更新 ε 的近邻查询的算法,并从理论上对算法进行了分析。我们用提出的方法与基于 DFT 和 DWT 的搜索方法进行了详细的比较,实验结果显示出该方法在准确率、时间上都具有很大的优越性。

下一步,我们将提出的方法推广到多变量的时间序列,并进一步应用于序列的分类、聚类等时序数据挖掘中。

参考文献:

[1] AGRAWAL R, FALOUTSOS C, SWAMI A. Efficient similarity search in sequence databases[A]. Proceedings of the 4th Int'l Conference on Foundations of Data Organization and Algorithms[C]. New York: Springer, 1993. 69 - 84.

[2] FALOUTSOS C, RANGANATHAN M, MANOLOPOULOS Y. Fast subsequence matching in time-series databases[A]. Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data[C]. Minneapolis: ACM press, 1994. 419 - 429.

[3] CHAN KP, FU AWC. Efficient time series matching by wavelets[A]. Proceedings of the 15th International Conference on Data Engineering[C]. IEEE, 1999. 126 - 133.

[4] POPIVANOV I, MILLER RJ. Similarity search over time-series data using wavelets[A]. Proceeding of the 18th International Conference on Data Engineering[C]. Washington DC: IEEE Computer Society, 2002. 212 - 221.

[5] VLACHOS M, LIN J, KEOGH E, *et al.* A wavelet-based anytime algorithm for k-means clustering of times series[A]. The 3 SIAM International Conference on Data Mining[C]. San Francisco, CA, 2003.

[6] WU YL, AGRAWAL D, ABBADI AE. A comparison of DFT and DWT based similarity search in time-series databases[A]. Proceedings of the 9th International Conference on Information and Knowledge Management[C]. McLean VA: ACM Press, 2000. 488 - 495.

[7] 吴乐南. 数据压缩[M]. 北京: 电子工业出版社, 2000.

[8] (加) HAN JW. 数据挖掘: 概念与技术(英文版)[M]. 第 2 版. 北京: 机械工业出版社, 2006.

[9] CHEUNG KL, FU A. Enhanced nearest neighbor search on the R-tree[J]. ACM SIGMOD Record, 1998, 27(3): 16 - 21.