自动向量化中基于数据依赖分析的循环分布算法

黄 磊 姚 远 侯永生 杨 明
(解放军信息工程大学信息工程学院 郑州 450002)

摘 要 循环分布是开发向量化程序的一个有效的方法。但是由于程序中的数据相关性,当前的自动向量化编译器实现完全的循环分布非常困难。因此,当前的自动向量化编译器一般采用简单的循环分布方法。以数据依赖关系分析为基础,从有无依赖环的角度分析了程序中语句的向量化能力,提出了基于语句向量化识别的循环分布算法,并在自动向量化中加以实现。通过此方法,可以充分地分析语句或依赖环的向量化能力,最终采用循环分布,将可向量化的语句与不可向量化的语句分布在不同的循环中。该方法可以处理当前的自动向量化编译器无法向量化的循环,对一些语句间有依赖关系的循环可达到较好的效果。

关键词 自动向量化,SIMD,依赖关系分析,循环分布
中图法分类号 TP311 **文献标识码** A

Loop Distribution Algorithm Based on Data Dependence Analysis in Auto-vectorization

HUANG Lei YAO Yuan HOU Yong-sheng YANG Ming
(School of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract Loop distribution is a useful method to vectorization programs, but because of the data dependence, it's very hard to completely achieve loop distribution in auto-vectorization. So, it's usually used easily loop distribution in current auto-vectorization compiler. Here, discussed a new loop distribution method based on identify the statement vectorization, from the data dependence view, and achieved in current auto-vectorization compiler. By this method, we can completely analyse which statement can vectorize, which dependence cycle can vectorize, finally using loop distribution, the vectorization statements can and no-vector statements be distributed in different loops. This method can handle these loops which can't be vectorized by other auto-vectorization compilers, and have good effect for some loops which have complex dependence.

Keywords Auto-vectorization, SIMD, Data dependence analysis, Loop distribution

1 引言

20世纪末,家用电脑得到充分的发展,而多媒体应用是家用电脑的主要用途之一。多媒体程序由于对大量数据采用相同的操作,因此适合进行向量化。自从1996年Intel在奔腾处理器上集成了MMX后,越来越多的通用处理器上集成了SIMD(Single Instruction Multiple Data,单指令多数据)扩展,很多的厂商在处理器中都集成了这一多媒体扩展应用。如今数据的处理越来越庞大,如游戏机芯片、大规模并行处理机等,因此SIMD技术不仅用在多媒体处理方面,而且用在各种有大量数据存取的应用中,为当前的海量数据存取以及高性能计算带来了新的解决问题的方法。

SIMD扩展技术的广泛应用,不仅为大量数据的存取带来了新的解决问题、提升性能的途径,也带来了一系列新的问题。传统向量化技术主要基于Randy Allen和Ken Kennedy提出的向量化算法[1],在依赖关系分析的基础上进行循环分布[1,9],从而得到可向量化的循环。传统的向量化方法针对向量计算机提出,不用考虑向量化长度,与SIMD向量化有所不同。曾扬[8]提出了一种类似于π图的用于循环分布的H图算法,并提出了破除依赖关系优化算法,但这主要是针对多层循环提出的方法。Kennedy等[4,5]研究了如何使用循环分布来提高程序并行和向量化的能力,并联合使用循环合并来增强数据的局部性。2000年Larsen等提出了SLP算法[2],它根据连续的内存访问和定义使用链发掘基本块内的向量化,没有用到传统的向量化方法,也存在着一些不足之处。在向量化方面的理论研究很深入,但这些理论实际应用很困难。当前的大多数商用向量化编译器都是采用传统的向量化方法,循环分布也是采用简单的分布算法。因此,本文在自动向量化中,从数据依赖分析的角度,提出了基于语句向量化识别的循环分布算法,以解决实际应用的不足之处。

2 SIMD中的依赖关系分析

语句的依赖关系直接关系到语句能否向量化,也影响到循环能否做分布。由于SIMD中的依赖关系分析与传统向量

到稿日期:2010-10-06 返修日期:2011-01-02 本文受核高基重大专项"支持国产CPU的编译系统及工具链"(2009ZX01036-001-001-2)资助。
黄 磊(1984—),男,硕士生,主要研究领域为先进编译技术,E-mail:huangl18@163.com;姚 远(1974—),男,副教授,硕士生导师,主要研究领域为先进编译技术和高性能计算等;侯永生(1979—),男,博士生,主要研究领域为先进编译技术;杨 明(1984—),男,硕士生,主要研究领域为先进编译技术。

化的依赖关系分析的结果略有不同，因此我们在讨论 SIMD 循环分布之前，先讨论 SIMD 的依赖关系分析。

当程序员用程序设计语言编写应用程序时，由于人的思维习惯，他希望程序的计算结果，首先是从第一个语句得到，然后是第二条等等。其中，分支语句和循环控制语句是例外，但这也是程序员期望的计算机的具体执行顺序。当然，直接从程序员编写的程序顺序来向量化是不可能的，因为向量操作会改变操作的顺序。如下面的例子：

例 1　for(i=0;i<2;i++){

$S1(i)$:$a[i]$=$b[i]$+$c[i]$;

$S2(i)$:$b[i+1]$=$d[i]$;}

$S1$,$S2$ 代表循环中的两条语句，i 是循环索引。若串行执行，$S2(0)$对 $b[1]$有一个写操作，$S1(1)$对 $b[1]$有一个读操作，是先写后读的流依赖，执行顺序不可改变；若向量执行，$S1(1)$先对 $b[1]$有一个读操作，$S2(0)$再对 $b[1]$有一个写操作，这改变了程序的语义。下面可以从有依赖环和无依赖环的角度来讨论依赖关系。

2.1　无环依赖分析

2.1.1　依赖方向为正向的依赖

依赖方向为正向的依赖，即语句的逻辑执行顺序与实际执行顺序一致。这种情况下，进行 SIMD 的向量化操作不会改变程序的逻辑执行顺序，可直接向量化。

例 2　for(i=0;i<2;i++){

$S1(i)$:$a[i+1]$=$d[i]$+$c[i]$;

$S2(i)$:$b[i+1]$=$a[i]$;}

2.1.2　依赖方向为反向的依赖

依赖方向为反向的依赖，即语句的逻辑执行顺序与实际执行顺序不一致。此时，进行 SIMD 的向量化操作会改变程序的执行顺序，导致不正确的向量化结果，如例 1。

例 1　若要向量化，可调换 S1 与 S2 的语句顺序，将其反向的依赖变为正向依赖即可向量化，如下所示：

例 1 变换后：for(i=0;i<256;i++){

$S2(i)$:$b[i+1]$=$d[i]$;

$S1(i)$:$a[i]$=$b[i]$+$c[i]$;}

这样虽然改变了语句的执行顺序，但语义并没有改变，且这样的正向依赖可直接向量化，向量化后不会对结果产生影响。一般对后向依赖的向量化方法就是对语句拓扑排序后，直接进行向量化。

2.2　有环依赖分析

依赖环是由于语句之间的依赖关系形成了一个回路而造成的。有依赖环的情况下，仅仅通过调整语句顺序一般是不可向量化的。下面我们从在语句分裂后可否向量化的角度来讨论依赖环，前提是不考虑其它影响向量化的因素，单从依赖的角度来考虑。

2.2.1　不可向量化的依赖环

不可向量化的依赖环就是无论怎样改变语句的执行顺序或将语句进行分裂操作都不能将这个环向量化。关键依赖边是依赖环中的一条依赖边，且去除关键依赖边后，依赖环将会被解开，依赖环也不存在。不可向量化的依赖环中所有的关键依赖边必定是流依赖(先写后读)。由于流依赖是程序必须保持的执行顺序，因此不论如何变换都不能将流依赖关系消除，于是依赖环一直都存在，故不可向量化。

下面的例子就是不可向量化的依赖环：

例 3

for(i=0;i<256;i++){

$S1$:$a[i+1]$=$c[i]$+$d[i]$;

$S2$:$b[i+1]$=$a[i]$+$d[i]$;

$S3$:$c[i+1]$=$b[i]$+$d[i]$;}

例 3 的依赖关系如图 1 所示。语句 $S1$、$S2$、$S3$ 之间形成了一个流依赖的依赖环，因此这 3 条语句均不可向量化。在 SIMD 的向量化中，流依赖是不可向量化的一个必要条件，但不是充分条件，因为 SIMD 的向量化有向量化长度的概念。一次向量化操作同时执行的指令的条数是有限的，这要根据寄存器的长度和数据类型来判断。因此不可向量化的条件中还有一条，就是关键依赖边的依赖距离必须小于向量长度。

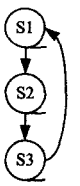

图 1　例 3、6、7 的语句依赖图

不可向量化的依赖环中还有一种情况，即自身真依赖的情况。虽然是单条语句，但其也组成了一个依赖环，如下面的例子。

例 4　for(i=0;i<256;i++)

$a[i+1]$=$a[i]$+2;

进行变换后，相当于如下的例子：

例 5　for(i=0;i<256;i++){

$S1$:$tmp[i]$=$a[i]$+2;

$S2$:$a[i+1]$=$tmp[i]$;}

这样 $S1$ 与 $S2$ 就组成了一个流依赖的依赖环，因此不可向量化。例 4 与例 5 在语义上是等价的，结果上也没有什么区别。例 4 是例 5 的一个特殊情况，因此自身真依赖可看作是流依赖环的一个简化版，因此其不可向量化。

2.2.2　可向量化的依赖环

若通过依赖分析，依赖环中可去除一条关键依赖边，此依赖环将会被解开，依赖环也就不存在了。这样，依赖环的语句也可以向量化了。

类似于不可向量化的依赖环中的判断，可向量化的依赖环中的关键依赖边能否去除，主要从两个方面来考虑：一是依赖距离是否不小于向量长度，若依赖距离足够大，在 SIMD 的向量化后，数据的存取并不会占用相同的地址空间，因此依赖也就不存在，如下例：

例 6　for(i=0;i<256;i++){

$S1$:$a[i+4]$=$c[i]$+2;

$S2$:$b[i+1]$=$a[i]$+3;

$S3$:$c[i+1]$=$b[i]$+5;}

例 6 的依赖关系如图 1 所示，3 条语句之间构成了一个依赖环。假设向量化寄存器的长度为 128 位，数组的数据类型在整形 32 位，则向量长度为 4。尽管 3 条语句构成了一个依赖环，但是 $S1$ 中的数组 a 与 $S2$ 中的数组 a 之间的依赖距离为 4，已经不小于向量长度，数组 a 之间由于占用相同的地址空间而造成的依赖也就不存在。因此，对这个循环做向量化不会影响程序的执行结果。但需要调整语序来执行，向量化的执行顺序应该是 $S2$、$S3$、$S1$。

另一种情况，向量化的依赖环环中至少有一条关键依赖

边为反依赖，即为先读后写，此种依赖关系边可通过语句分裂的方式消除依赖，环就不存在了，故也可以向量化。如下例：

例 7　for(i=0;i<256;i++){
$S1$:$a[i+1]=c[i]+2$;
$S2$:$b[i+1]=a[i]+3$;
$S3$:$d[i+1]=b[i]+a[i+2]+5$;}

例 8　for(i=0;i<256;i++){
$S4$:$tmp[i]=a[i+2]$;
$S1$:$a[i+1]=c[i]+2$;
$S2$:$b[i+1]=a[i]+3$;
$S3$:$d[i+1]=b[i]+tmp[i]+5$;}

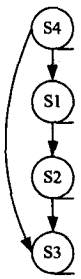

图 2　例 8 的语句依赖图

例 7 的语句依赖关系如图 1 所示，$S1$、$S2$、$S3$ 之间形成了一个依赖环。但 $S3$ 到 $S1$ 的依赖边为反依赖，而反依赖通过循环变换可将依赖关系去除，可实现向量化。对例 7 中的语句 $S3$ 进行语句分裂，分裂后如例 8 中的 $S4$ 和 $S3$，且 $S4$ 中的 $a[i+2]$要比 $S1$ 中的 $a[i+1]$先执行，因此若要进行 SIMD 向量化，则 $S4$ 要放在 $S1$ 之前。例 8 与例 7 在程序的语义及执行结果上都是等价的，例 8 的依赖关系如图 2 所示。从图 2 中我们可以看出，这 4 条语句之间并没有构成依赖环，因此例 8 可以进行完全向量化，也就是破除了例 7 的一条关键依赖边，使其可以向量化。

3　基于向量化识别的循环分布算法

在 SIMD 的自动向量化中，需要将循环自动向量化。但若循环中存在不可向量化的依赖关系，就没有办法进行自动向量化。而循环中又有可向量化的语句，要开发这些语句的向量化特性，我们就需要进行循环分布。循环分布将一个循环分解为多个循环，每个循环都有与原循环相同的迭代空间，但只包含原循环的语句子集。之后按凝聚图（其结点为强连通子图）确定的偏序关系来执行分布后的各个子循环，通常将其用于分布出可向量化的循环。

3.1　当前自动向量化中循环分布算法的不足

当前的自动向量化中使用传统的循环分布方法，以循环的语句依赖图为依据，通过 tarjan 算法[6,7]找出语句依赖图中的强连通子图，再将语句依赖图按强连通子图进行分解，然后按凝聚图（其结点为强连通子图）确定的偏序关系来执行分解后的各个子循环。下面举例说明传统的循环分布是如何实现的。

例 9　for(i=1;i<256;i++){
$S1$:$a[i]=a[i+1]+2$;
$S2$:$b[i+1]=c[i]+3$;
$S3$:$c[i+1]=a[i+1]+a[i-1]$;
$S4$:$d[i+1]=d[i]+c[i]$;}

例 10　$L1$:for(i=0;i<256;i++){
$S1$:$a[i]=a[i+1]+2$;
$S3$:$c[i+1]=a[i+1]+a[i-1]$;}
$L2$:for(i=0;i<256;i++){
S2:b[i+1]=c[i]+3;
S4:d[i+1]=d[i]+c[i];}

例 9 中的语句依赖图如图 3(a)所示。通过较简单的依赖关系分析，语句 $S1$ 与 $S3$ 由于数组 a 的依赖构成了一个依赖环，$S3$ 由于数组 c 的依赖，分别有指向 $S2$ 和 $S4$ 的依赖。循环分布时，通过 tarjan 算法可找出依赖图中的强连通分量，将强连通分量凝聚成无环凝聚图，再通过凝聚图的拓扑排序。本例中将 $S1$ 与 $S3$ 作为一个整体，拓扑排序时，要将这个环提至 $S2$ 之前执行。如图 3(b)所示，为满足排序之后的依赖关系，分布的结果如例 10 所示。

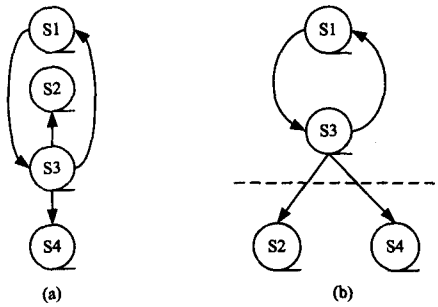

图 3　(a)例 9 的依赖图；(b)例 9 重排后的依赖图，再按图中虚线分裂成例 10

上例为传统的循环分布实现过程。通过分析上例的向量化特性，可知上述分布过程并不能达到理想的结果。通过上一节的分析可知 $S1$ 与 $S3$ 组成的依赖环，有一条依赖边为反依赖，而且又是关键依赖边，因此 $S1$ 与 $S3$ 组成的环可向量化。向量化时只需要对 $S3$ 进行分解，分成两条语句；而 $S4$ 与上一节的例 4 类似，是自身真依赖的情况，不可向量化，如图 4(a)所示。分解时应该将可向量化语句 $S1$、$S2$、$S3$ 分成一个循环，不可向量化语句 $S4$ 分成一个循环，如例 11 所示，依赖图如图 4(b)所示。

例 11　$L1$:for(i=1;i<256;i++){
$S5$:$tmp[i]=a[i+1]$;
$S1$:$a[i]=a[i+1]+2$;
$S3$:$c[i+1]=tmp[i]+a[i-1]$;
$S2$:$b[i+1]=c[i]+3$;}
$L2$:for(i=1;i<256;i++){
$S4$:$d[i+1]=d[i]+c[i]$;}

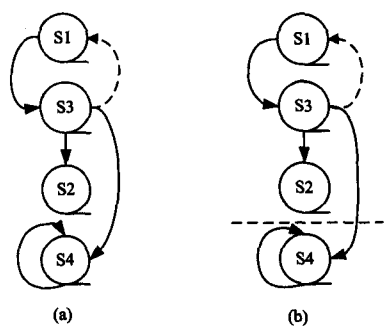

图 4　(a)例 9 的依赖图，虚线表示反依赖，S4 有自身真依赖；(b)按图中水平虚线分布可得例 11

按传统的循环分布布到的结果例 10 与实际应该分得的结果例 11 不同，主要是传统的循环分布分析语句的向量化特性并不充分。传统循环分布在依赖性方面有以下几个方面考虑不足：

(1) 未将依赖距离考虑在循环分布中去；

(2) 未区分语句是否有自身真依赖，导致不可向量化；

(3) 未区分可向量化的依赖环和不可向量化的依赖环。

下面主要针对当前的自动向量化中循环分布对数据依赖性分析不足的问题，来讨论一种改进的循环分布方法。

3.2 基于依赖关系的语句向量化能力分析

SIMD 中语句的依赖关系分析和传统的向量化不同，传统的向量化只从依赖环的角度来分析可否向量化。一般认为依赖环内的语句不可向量化，其它语句可向量化。但是通过第 2 节的分析可以知道，有些依赖环通过循环变换可以去除一些依赖边，从而可向量化，有些单个语句由于自身真依赖的存在而不可向量化。因此，我们根据 SIMD 向量化的特点，对语句能否向量化从新的角度来分析。另外，我们的分析只从数据依赖的角度出发，分析最内层循环的数据相关性以进行循环分布，暂不考虑其数据类型、操作类型、连续和对齐等其它影响向量化的因素。

图 5 给出了一个通过对循环内语句依赖关系分析得到的语句可否向量化的算法。此算法是基于语句依赖图，通过 tarjian 算法找强连通分量，再判定强连通分量中的语句是否可向量化。

```
procedure statement_vectoriable(R,k,D)
    //R 是要分析的循环区域；
    //k 是通过数组下标分析得到的语句的依赖距离；
    //D 是 R 中的语句依赖图；
    去除依赖图 D 中依赖距离大于向量长度的边
    从依赖图 D 中找到 R 的最大强连通分量集合[S1,S2,…,Sm](使用 tarjian 算法)
    把每个 Si 约简为一个节点，从 R 构造 R*，相应地从 D 构造 D*
    [U1,U2,…,Um]为 R* 中对应于 D* 的节点
    for i=1 to m do
      if Ui 中只有一条语句 then
        if 此语句为自身真依赖
          then 标记此语句不可向量化
          else 标记此语句可向量化
      else begin
          清除 Ui 中可删除的边(反依赖边和输出依赖边)
            找到依赖图 D 中关于 Ui 的最大强连通分量集合[US1,US2,…,USn](使用 tarjian 算法)
          for i=1 to n do
              if USi 中语句数大于 1
              then 标记 USi 中的所有语句不可向量化
              else if USi 中有语句为自身真依赖
              then 标记 USi 中的此语句不可向量化
              else 标记 USi 中的语句可向量化
          end
    end
      end statement_vectoriable
```

图 5 标记语句可否向量化的识别框架

由于 SIMD 向量长度的限制，可以将循环看作是由许多个循环迭代次数等于向量长度(向量长度＝向量寄存器的位数/数据类型的位数)的子循环组成，对每个子循环按照传统向量化的方法进行分析。如果依赖距离大于向量长度，则两条语句分属于不同的子循环，循环间的依赖次序就能得到保持，就可以不予考虑。这样，传统的向量化分析方法也可以用于针对 SIMD 体系结构的向量化分析。例如现在的 Intel 的 SSE 系列指令集，其向量寄存器的宽度为 128 位，其整型的向量长度为 4。依赖距离不小于向量长度时，例如语句 $a[i+4]=a[i]+2$，依赖距离为 4，一次执行 4 条指令后，语句执行顺序的改变并不会导致程序结果的改变。此算法中首先进行依赖距离的检测，与传统的向量化不同，其结合了 SIMD 结构的特点，去除了一些不必要的依赖。

若强连通分量中只有一条语句，判断语句是否为自身真依赖，就是判断自身流依赖。判断的方法是：首先查看语句有没有指向其自身的依赖边，再判断此依赖边是否为流依赖，也就是判断其依赖关系是否为先写后读。另外，我们没有讨论自身读依赖的情况，因为此种情况先读后写，通过第 1 节的分析知道它可以直接向量化，不用考虑其依赖性，故在此不讨论。

在有多个语句的依赖环中，分裂所有反依赖边的源点语句。因为反依赖边为先读后写，其源点语句就是读操作。向量化的过程中，可能会将一个地址空间先读变为后读，从而导致语义不正确。因此，可将源点语句的读操作分裂出来，拓扑排序时会将分裂得到的临时语句排在前面，不会将先读变为后读，环中的依赖关系也会被去掉。

语句分裂的方法：将依赖环中所有的反依赖的源点分裂成两个新的语句，即将依赖的数组值赋给一个临时数组，原语句中的数组也用临时数组代替，并同时更新依赖图。例如，语句 $a[i]=b[i+2]+2$，若 $b[i+2]$是反依赖的源点，则分裂为 $tmp[i]=b[i+2]$和 $a[i]=tmp[i]+2$ 两条语句。之后，去掉依赖图中所有的反依赖边，依赖图中剩余的边均为流依赖。再更新语句依赖图，通过 tarjian 算法重新找强连通分量。

当一个 SCC(强连通分量)中有多条语句时，由于依赖图中剩余的边均为流依赖，多条语句构成的环也就是由流依赖构成的环，故环中的语句均不可向量化。

通过上述的分析过程，可以发掘出语句基于依赖关系的向量化特性。知道了语句的向量化特性，再根据语句的拓扑排序，就可以实现基于依赖分析的 SIMD 循环分布。下面，再对循环的分布过程加以讨论，提出其改进策略。

3.3 基于语句向量化识别的循环分布

利用对语句向量化能力的分析，我们可以在此基础上实现基于依赖的循环分布，最终将循环中可向量化的语句与不可向量化的语句分布在不同的循环中，以充分开发循环的向量化能力。在循环分布的过程中，尽量将可向量化的语句放入同一个循环，将不可向量化的语句放入一个循环。这样，可以减少循环的生成，提高数据的局部性，也避免了再次使用循环合并。

```
procedure loop_distribution(R,k,D)
    //R 是要分析的循环区域；
```

```
//k 是通过数组下标分析得到的语句的依赖距离;
//D 是 R 中的语句依赖图;
去除依赖图 D 中依赖距离大于向量化因子的边
从依赖图 D 中找到 R 的最大强连通分量集合[S1,S2,…,Sm](使用 tarjian 算法)
构造 SIMD 队列保存可向量化的语句和 NONSIMD 队列保存不可向量化的语句
statement_vectoriable(R,k,D);
  for i=1 to m do
    if Si 中的语句均可向量化
    then 将 Si 中的所有语句加入到 SIMD 队列中
    else 将 Si 中的所有语句加入到 NONSIMD 队列中
  end
  while(SIMD 队列不为空 or NONSIMD 队列不为空) do
    while(SIMD 队列头 Sp 无前驱结点) do
      将此 Sp 中的所有语句放入循环 loopi 中
      从 SIMD 队列中去除此 Sp 并去除其所有出边
    end
      if(SIMD 队列中头 Sp 中的语句有前驱结点在 NONSIMD 队列中)
      then 分布循环 loopi
      while(NONSIMD 队列头 Sq 无前驱结点) do
        将此 Sq 中的所有语句放入循环 loopj 中
        从 NONSIMD 队列中去除此 Sq 并去除其所有出边
      end
      if(NONSIMD 队列中头 Sq 中的语句有前驱结点在 SIMD 队列中)
      then 分布循环 loopj
    end
  end loop_distribution
```

图 6　从语句可否向量化的角度循环分布算法

上述算法，单从一个队列进行循环分布并不能得到最少的循环个数。下面以一个例子来加以说明：

例 12　for(i=1;i<256;i++){

$S1$:$a[i]$=$a[i+1]$+2;

$S2$:$b[i+1]$=$b[i]$+$c[i+1]$+3;

$S3$:$c[i]$=$a[i]$+5;}

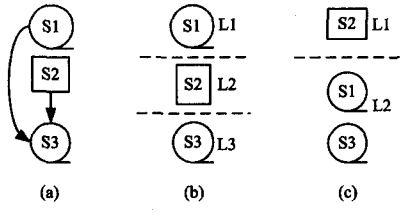

图 7　(a)例 12 的依赖；(b) SIMD 队列开始分布结果；(c) NONSIMD 分布结果

例 12 的依赖图如图 7(a)所示。若按照 AMD Open64 中的分布方法，从 SIMD 队列开始找无前驱结点的语句，可找到 $S1$，再找到 $S3$。但 $S3$ 依赖于 $S2$，于是队列要跳转到 NONSIMD，需要进行分布，得到循环 $L1$，只包含语句 $S1$；再从 NONSIMD 队列遍历，分布 $S2$ 得到循环 $L2$；最后回到 SIMD 队列，分布 $S3$ 得到循环 $L3$。通过此方法分布最后生成了 3 个循环，结果如图 7(b)所示。若从 NONSIMD 队列开始遍历，我们首先找到无前驱的结点 $S2$，只有一个结点，需要跳转到另一个队列，分布得到 $L1$；再遍历 SIMD 队列，$S1$ 与 $S3$ 分布得到循环 $L2$，分布结果如图 7(c)所示，最后只有两个循环。图 7(b)与(c)的结果不同，但就此例的语义，这两种分布方法的语义是相同的，从 NONSIMD 分布得到的循环数目要少。因此，本算法从两种角度均做一次分布分析，最后按产生的循环个数少的结果来进行最终的循环分布。

我们用此分布方法对上例 9 进行分布，在此处记为 9-0，根据语句可否向量化的分析方法，可知 S1 与 S3 组成的环中有一条反依赖边，将其分裂出 S5，结果如例 9-1，更新依赖图，再用 tarjian 算法找 SCC 时，$S1$、$S5$、$S3$ 就不是一个环了，拓扑排序后就如例 9-2。再标记 $S1$、$S2$、$S3$、$S5$ 可向量化，$S4$ 不可向量化。上述方法分布得例 9-3，对比例 9-3 与例 11 可知与我们需要的结果一致。分布过程如图 8 所示。

例 9-0：

```
for(i=1;i<256;i++){
S1:a[i]=a[i+1]+2;
S2:b[i+1]=c[i]+3;
S3:c[i+1]=a[i+1]+a[i--1];
S4:d[i+1]=d[i]+c[i];}
```

例 9-1：

```
for(i=1;i<256;i++){
S1:a[i]=a[i+1]+2;
S2:b[i+1]=c[i]+3;
S5:tmp[i]=a[i+1];
S3:c[i+1]=tmp[i]+a[i-1];
S4:d[i+1]=d[i]+c[i];}
```

例 9-2：

```
for(i=1;i<256;i++){
S5:tmp[i]=a[i+1];
S1:a[i]=a[i+1]+2;
S3:c[i+1]=tmp[i]+a[i-1];
S2:b[i+1]=c[i]+3;
S4:d[i+1]=d[i]+c[i];}
```

例 9-3：

```
L1:for(i=1;i<256;i++){
S5:tmp[i]=a[i+1];
S1:a[i]=a[i+1]+2;
S3:c[i+1]=tmp[i]+a[i-1];
S2:b[i+1]=c[i]+3;}
L2:for(i=1;i<256;i++){
S4:d[i+1]=d[i]+c[i];}
```

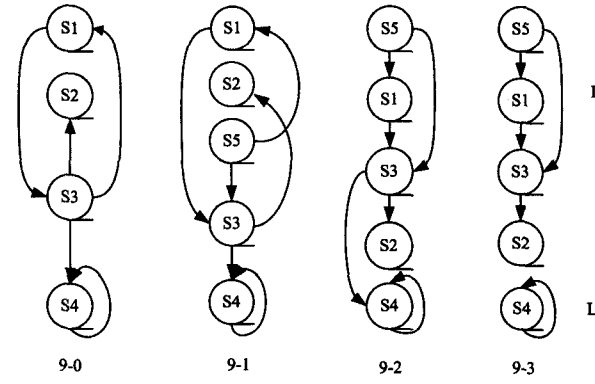

图 8　例 9 的循环分布过程

经上述分布后，$L1$ 可完全向量化，$L2$ 不可向量化串行执行。上例分布过程中对语句 $S3$ 采用了语句分裂的方法，多出了一条语句 $S5$，看起来是多了执行的语句数。但在我们当前的向量化执行中，执行 $S5$ 会 *load* 数组 $a[i+1]$，并保存在 SIMD 寄存器中，$S3$ 中需要用到 $tmp[i]$ 时，就直接从寄存器中去寻找数组 $a[i+1]$，而 $tmp[i]$ 只是一个临时变量，并不会去执行。但是对更为复杂的依赖关系，为了开发更多的 SIMD 特性，按此分裂方法可能会多出语句数，并且不能通过私有化消去。这种情况很少存在，而且我们可以对此分布过程进行收益分析，以期找到最佳的分布结果，在本文中不再做

深入讨论。

4 实验结果

采用本文的循环分布算法，可以充分地将循环内可向量化与不可向量化的语句放在不同的循环中。而当前主流的编译器都做不到这一点，也就无法开发这种部分语句可向量化的循环。上一节讨论的例 9，在 alpha 处理器上(256bit)进行试验，采用本文的分布方法与采用传统的分布算法，加速比可达到 1.56。

图 9 是基于 open64 实现的传统循环分布和基于语句依赖的循环分布在 alpha 机器上的测试结果，显示了循环分布算法改进前后的性能对比。改进后程序平均加速比为 1.11，相对于原来的向量化方法对测试集的性能提高平均达到 4%。从测试可以看出，使用本文的方法通过充分分析程序中语句的数据依赖性，可分布出更多可向量化的循环，生成向量化代码，可更充分地利用向量化功能部件。但是在分布时可能会引入一些额外开销，造成部分测试用例效果不明显。

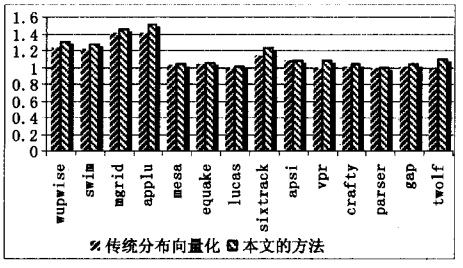

图 9 传统分布算法和基于语句的分布算法对 spec2000 程序的加速比

结束语 本文分析了传统向量化中循环分布方法的不足之处，从依赖关系角度来分析语句可否向量化，又从语句能否向量化的角度来进行循环分布。此循环分布方法主要是对传统的循环分布方法的改进，增加了对单个语句可否向量化和依赖环语句可否向量化的分析，对可向量化的依赖环采用语句分裂的方法来实现，这一改进可以从依赖关系的角度进行彻底的循环分布。虽然此循环分布方法是对传统的循环分布的改进，但还存在一些问题值得研究，主要是以下两个方面：

1)通过本文的循环分布方法对可向量化的依赖环进行分析时，为了去除反依赖边，需要对反依赖边的源点语句进行语句分裂，此过程会增加语句的数目。若整个依赖环内的全部语句均可向量化，增加的语句在进行 SIMD 时会消除掉，不会带来附加的操作。但是，若依赖环中有可向量化的语句又有不可向量化的语句，进行循环分布时，那些为了消除反依赖边进行语句分裂而产生的临时语句会被分布在不同的循环中，这样就无法不执行这些临时操作，会产生额外的开销，这也是有些例子测试效果不好的原因。而如果产生的开销大于 SIMD 向量化的收益，循环分布就不应该做，因此需要对此进行循环分布的收益分析。

2)此循环分布方法目前只是从依赖关系的角度来考虑的，在实际应用中，循环能否向量化的影响因素有很多，比如循环内有无函数调用、指令跳转、循环内语句是否连续、对齐等，这些方面均可影响循环的向量化能力。循环分布应该从多个方面和角度出发，且在实际应用中自动分析这些情况均很复杂，自动进行循环分布可能并不能达到我们预期的目标，因此可以考虑采用交互式的循环分布方法，目前该方法还在进一步的研究中。

参 考 文 献

[1] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures——A Dependence-Based Approach[M]. US: Morgan Kaufmann Publishers, 2001

[2] Larsen S, Amarasinghe S. Exploiting superword level parallelism with multimedia instruction sets[C]// Proc of the ACM SIGPLAN Conference on Programming Language Design and Implementation. 2000: 145-156

[3] Stewart J. An investigation of SIMD instruction sets [R]. University of Ballarat School of Information Technology and Mathematical Sciences. 2005

[4] Kennedy K, Mckinley K S. Loop distribution with arbitrary control flow[C]// Proceedings of the 1990 Conference on Supercomputing. 1990: 407-416

[5] Kennedy K, Mckinley K S. Loop distribution with multiple exits [C]// Proceedings of the 1992 ACM/IEEE Conference on Supercomputing. Minneapolis, Minnesota, United States, 1992: 204-213

[6] Lengauer T, Tarjian R E. A fast algorithm for finding dominators in a flowgraph[J]. ACM Transactions on Programming Languages and Systems, 1979: 1(1): 121-141

[7] Tarjian R E. Depth first search and liner graph algorithms[J]. SIAM Journal of Computing, 1972, 1(2): 146-160

[8] 曾扬. 循环分布及依赖关系破除的优化问题[J]. 计算机学报, 1993, 16(6)

[9] 沈志宇, 胡子昂, 等. 并行编译方法[M]. 北京: 国防工业出版社, 2000

(上接第 274 页)

[9] Gllavat J, Ewerth R, Freislebe B. Text detection in images based on unsupervised classification of high-frequency wavelet coefficients[C]// Proceedings of the 17th International Conference on Pattern Recognition. 2004(1): 425-428

[10] Hontani H, Koga T. Character extraction method without prior knowledge on size and position information[A]// Proceedings of the IEEE International Vehicle Electronics Conference [C]. 2001: 67-72

[11] Guo Ge, Jin Jin, Ping Xi-jian, et al. Automatic Video Text Localization and Recognition[A]// Proceedings of Fourth International Conference on Image and Graphics (ICIG 2007)[C]. Chengdu, China, August 2007: 484-489

[12] Qian Xue-ming, Liu Gui-zhong, Wang Huan, et al. Text detection, localization, and tracking in compressed video[J]. Signal Processing: Image Communicaion, 2007, 22(9): 752-768

[13] 胡学龙, 许开宇. 数字图像处理[M]. 北京: 电子工业出版社, 2006: 169-179