软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(7):1717−1728 [doi: 10.3724/SP.J.1001.2012.04106]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

面向 SLP 的多重循环向量化*

魏 帅+, 赵荣彩, 姚 远

(解放军信息工程大学 信息工程学院,河南 郑州 450002)

Loop-Nest Auto-Vectorization Based on SLP

WEI Shuai+, ZHAO Rong-Cai, YAO Yuan

(Information Engineering Colledge, PLA Information Engineering University, Zhengzhou 450002, China)

+ Corresponding author: E-mail: weis0906@163.com

Wei S, Zhao RC, Yao Y. Loop-Nest auto-vectorization based on SLP. *Journal of Software*, 2012,23(7):1717−1728 (in Chinese). http://www.jos.org.cn/1000-9825/4106.htm

Abstract: Nowadays, more and more processors are integrated with SIMD (single instruction multiple data) extensions, and most of the compilers have applied automatic vectorization, but the vectorization usually targets the innermost loop, there have been no easy vectorization approaches that deal with the loop nest. This paper brings out an automatic vectorization approach to vectorize nested loops form outer to inner. The paper first analyzes whether the loop can do direct unroll-and-jam through dependency analysis. Next, this study collects the values about the loop that will influence vectorization performance, including whether it can do direct unroll-and-jam, the number of array references that are continuous for this loop index and the loop region. Moreover, the study also presents an aggressive algorithm that will be used to decide which loops need to do unroll-and-jam at last generate SIMD code using SLP (superword level parallelism) algorithm. The test results on Intel platform show that the average speedup factor of some numerical/video/communication kernels achieved by this approach is 2.13/1.41, better than the innermost loop vectorization and simple outer-loop vectorization, the speedup factor of some common kernels can reach 5.3.

Key words: SIMD (single instruction multiple data); vectorization; data dependence analysis; nested loop; SLP (superword level parallelism)

摘 要: 如今,越来越多的处理器集成了 SIMD(single instruction multiple data)扩展,现有的编译器大多也实现了自动向量化的功能,但是一般都只针对最内层循环进行向量化,对于多重循环缺少一种通用、易行的向量化方法.为此,提出了一种面向 SLP(superword level parallelism)的多重循环向量化方法,从外至内依次对各个循环层次进行分析,收集各层循环对应的一些影响向量化效果的属性值,主要包括能否对该循环进行直接循环展开和压紧、有多少数组引用相对于该循环索引连续以及该循环所包含的区域等,然后根据这些属性值决定在哪些循环层次进行直接循环展开和压紧,最后通过SLP对循环中的语句进行向量化.实验结果表明,该算法相对于内层循环向量化和简单的外层循环向量化平均加速比提升了 2.13 和 1.41,对于一些常用的核心循环可以得到高达 5.3 的加速比.

关键词: SIMD;向量化;依赖关系分析;多重循环;超字并行

中图法分类号: TP311 **文献标识码**: A

* 基金项目: 国家高技术研究发展计划(863)(2009AA012201); “核高基”国家科技重大专项(2009ZX01036)
收稿时间: 2011-04-19; 修改时间: 2011-07-21; 定稿时间: 2011-09-01

© 中国科学院软件研究所 http://www.jos.org.cn

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)扩展.1997 年,摩托罗拉在 G3 PowerPC 上引入了 Altivec 指令集;2000 年,AMD 在 Athlon 处理器上集成了类似于 SSE 的 3DNow!指令集[1].通过并行执行一条指令的多个数据实现,可以使应用程序实现并行处理[2].

SIMD 功能部件最初只是为多媒体应用设计的,向量寄存器长度较短.随着人们对 SIMD 认识的加深和向量寄存器长度的增加,SIMD 扩展越来越多地应用于高性能计算中,从语音通信到信息安全传输,从视频处理到天气预报,处处可以见到 SIMD 的应用.向量化属于细粒度并行,指令种类繁多,不同平台的指令差异较大,没有统一的接口.如果手工对程序进行向量化,工作量太大,难以实现,所以自动向量化成为实现 SIMD 向量化的必然趋势.如今主流编译器,诸如 icc,gcc,open64 等都集成了自动向量化的功能.向量化算法按照目标系统的不同可以分为如下两类:

(1) 针对向量机的传统向量化算法

针对向量机[3]的传统向量化算法[4]由 Allen-Kennedy 提出,可以对非完美嵌套循环从外到内逐层进行向量化.主要方法是,通过求解强连通图[5]将循环体中那些不构成依赖环的语句向量化,因为针对向量机的向量化对向量长度没有限制,所以这种对语句的向量化实质上等于将这些语句从循环中分裂出去.而 SIMD 扩展只支持长度有限的向量寄存器,所以这种向量化策略不适用于针对 SIMD 扩展的向量化.此外,传统向量化处理内层循环时假设外层循环串行执行,以消除外层循环引起的依赖.但当依赖环为内层循环所携带而与外层循环无关时,按照传统向量化算法就不能对其进行向量化,只有经过一些循环变换,比如循环交换才能对其进行向量化.

(2) 针对 SIMD 扩展的向量化算法

现在,处理器上集成的 SIMD 扩展在向量化上增加了一些限制,比如有限的向量寄存器长度,只支持对连续的数组访问进行向量装载和存储等.

A. 内层循环向量化

由于在一些常见的循环中,数组相对于最内层循环索引是连续的,而且最内层循环相对于外层循环必然有更多的迭代次数,没有复杂的控制结构,所以现在的编译器大部分只对最内层循环进行向量化,这种向量化易于实现,代价较小.但是,有时候由于最内层循环存在依赖环或者归约等情况,就不能对其进行向量化或者进行向量化代价比较大;或者循环中的数组引用相对于最内层循环索引不连续,进行最内层循环向量化就不能取得收益或者收益较小.

B. 针对某个外层循环的向量化

循环交换通过将某个外层循环交换至最内层进行向量化[6−8],但是循环交换只能针对完美嵌套循环,如果是非完美嵌套循环,则需要进行一些循环变换将其转换为完美嵌套循环;通过将内层循环识别为特殊函数或者进行循环压缩也可以实现外层循环向量化[9,10];还有一些向量化方法只是针对两层循环进行外层循环向量化[11].

C. 复杂的外层循环向量化

固定若干种向量化选择,比如在 N 个循环层次对 N 个循环进行向量化,然后通过收益分析计算各种向量化选择对应的收益,从中挑选出收益最大的向量化方案进行向量化[12].但是这种向量化方法需要多面体模型的支持,涉及到一些复杂的程序变换和依赖关系分析,代价太大,难以实现.

可以看出,与传统向量化相比,针对 SIMD 扩展的向量化算法缺少一种简单、易行的向量化方法,使其能对非完美嵌套循环的各个循环层次进行分析,然后找出一种优化的向量化方案.为此,本文提出了一种面向 SLP[13](superword level parallelism,超字并行)的多重循环向量化算法,从外至内依次对各个循环层次进行分析,收集各层循环对应的一些影响向量化效果的属性值,主要包括能否对该层循环进行直接循环展开和压紧、有多少数组引用相对于该循环索引连续以及该循环所包含的区域等,然后根据这些属性值决定在哪些循环层次进行直接循环展开和压紧,最后通过 SLP 对循环中的语句进行向量化.

例 1 是一个典型的多重循环示例——矩阵乘,现在的编译器,如 open64 等只对最内层循环 k 进行向量化(如

© 中国科学院软件研究所 http://www.jos.org.cn

图 1(a)所示),而外层循环向量化可以对循环 j 进行向量化以实现更高的向量化效率(如图 1(b)所示).而按照本文所提出的向量化算法,对循环 k 和 j 进行直接循环展开和压紧,然后用 SLP 进行向量化,可以得到更高的效率(如图 1(c)所示).在 Intel 平台上对这 3 种向量化方法进行测试发现,3 种向量化方法得到的加速比分别为 1.14,1.9,2.25; 当向量化因子为 4 时,3 种向量化方法得到的加速比分别为 1.54,3.74,5.3.第 3 种向量化方法之所以能够取得较高的加速比,是因为在 j 层进行向量化时没有依赖关系的阻碍,但是循环中的数组引用 $a[i][k]$对循环索引 j 不连续,而在 k 层进行向量化时由于依赖关系的阻碍,需要对归约进行特殊处理才能进行向量化,而对归约的处理会增加冗余操作,从而带来性能损失.所以单独对循环索引 k 或者 j 进行向量化都不能达到最优的性能,而通过对循环 j 和循环 k 进行直接循环展开和压紧,然后用 SLP 进行向量发掘,可以取得更好的性能.

例 1:
```
for (i=0; i<M; i++)
    for (j=0; j<M; j++){
        c[i][j]=0;
        for (k=0; k<M; k++)
        c[i][j]=c[i][j]+a[i][k]*b[k][j];}
```

```
for (i=0; i<M; i++){
for (j=0; j<M; j++){
    vsum={0,0};
    for (k=0; k<M; k+=2){
    v1=vload(&a[i][k]);
    v2=vset(b[k][j],b[k+1][j]);
    vsum+=v1*v2;}
    vstore(vsum,sum[0]);
    c[i][j]=sum[0]+sum[1];}
```

(a) 内层循环向量化

```
for (i=0; i<M; i++){
for (j=0; j<M; j+=2){
   vsum={0,0};
   for (k=0; k<M; k++){
   v1=vload(&b[k][j]);
   v2=vset(a[i][k],a[i][k]);
   vsum+=v1*v2;}
   vstore(vsum,c[i][j]);}
```

(b) 外层循环向量化

```
for (i=0; i<M; i++){
for (j=0; j<M; j+=2){
   vsum={0,0};
   for (k=0; k<M; k+=2){
   va=vload(&b[k][j]);
   v0=vload(&a[i][k]);
   v1=vextend(v0,0);
   vsum+=v1*va;
   v2=vextend(v0,1);
   vsum+=v2*va;}
   vstore(vsum,c[i][j]);}
```

(c) 混合向量化

Fig.1 Three vectorization scheme

图 1 3 种循环向量化方案

本文的贡献主要表现在以下 4 个方面:

- 总结了影响循环向量化效果的 3 个因素:能否对该层循环进行直接循环展开和压紧、有多少数组相对于该循环索引连续以及该循环所包含的区域;
- 提出了用直接循环展开和压紧来进行循环变换,这种循环变换可以保持内层循环的结构不变,这样就可以用同一种方法递归地处理多层嵌套循环,并且循环变换之后便于 SLP 进行向量化;
- 分析了循环能够进行直接循环展开和压紧的充分条件;
- 综合影响多重循环向量化效果的 3 个因素,提出了一种激进的向量化策略.

本文第 1 节介绍基于 open64 的向量化编译框架.第 2 节介绍预优化阶段采用的候选循环识别和循环不变数组引用外提技术.第 3 节详细介绍面向 SLP 的多重循环向量化技术.第 4 节是实验结果和分析.第 5 节介绍相关研究.最后是全文的总结.

1 编译框架

整个 open64 的编译框架如图 2 所示,前端采用 gcc 的前端,将源程序转化成 whirl 结构的中间表示,接着进行过程间的分析和优化、循环级的分析和优化以及全局优化.这些优化不一定严格地按照顺序进行,可能有一些交叉,比如在循环级的优化中需要借助全局优化中的一些优化方法.后端采用 whir2c 将中间表示转换成 C 程序,向量化主要通过将源程序中可向量化的语句替换成 intrinsic 向量函数调用的形式实现.向量化工作主要在循环级的优化即 LNO 中进行,首先进行预优化,对循环进行筛选,选取那些适于进行向量化识别和变换的循环,

© 中国科学院软件研究所 http://www.jos.org.cn

进行循环不变数组引用外提等;然后从外至内依次对各个循环进行分析,通过依赖关系分析能否在各个循环层次上进行向量化,即能否进行直接循环展开和压紧,并通过一定的向量化策略决定在哪些循环层次进行直接循环展开和压紧;最后按照 SLP 算法进行向量发掘,并通过 whir2c 产生最终的向量化代码.

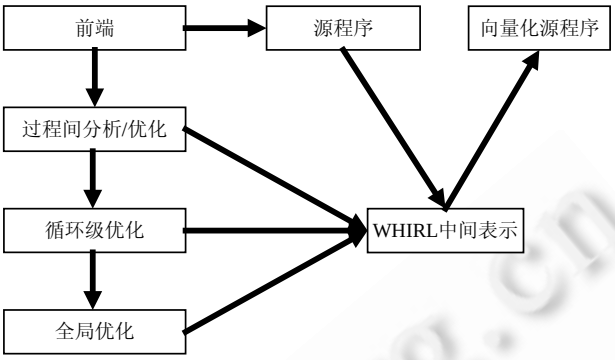

Fig.2 Vectorization compiler framework based on open64

图 2 基于 open64 的向量化编译框架

2 预优化

2.1 候选循环识别

候选循环识别主要是为了剔除那些不适于向量化的循环,或者是按照当前的向量化方法还不能处理的循环.本文采用的向量化方法要求循环具有以下特征:

(1) 循环具有明确的上下界和步长;

(2) 循环内没有函数调用和 goto,if 等控制流语句,只有唯一的入口,没有 goto 语句跳入循环体;

(3) 循环的迭代次数不能太低并且存在足够的向量化操作,否则不值得进行向量化;

(4) 循环中标量的定义使用关系完备;

(5) 循环层次也不能太多,循环内语句数目不能太多,本文所实现的算法将循环层数限制在 4 层以内.

当一个循环为嵌套循环时,如果它满足向量化的条件,则其所有的内层嵌套循环都满足这些条件.所以,在进行候选循环识别时,对于嵌套循环需要从最内层循环开始从内至外地进行分析,直至遇到一个不满足条件的外层循环为止,然后标志最外层适合向量化的循环为候选循环.在下面对循环进行向量化分析时,则从候选循环开始,由外至内地对各个循环层次进行分析.

2.2 循环不变数组引用外提

本模块主要是识别循环中一些相对于循环索引不变的数组引用,并将这些数组引用提至循环之外,其在循环中的定义和使用均用标量代替.比如在矩阵乘 MMM 中,因为 $c[i][j]$相对于内层循环嵌套是不变的,所以可以将其提取出来,则循环就转化为:

例 2:
```
for (i=0; i<M; i++)
    for (j=0; j<M; j++){
        s=0;
        for (k=0; k<N; k++)
        s+=a[i][k]*b[k][j];
        c[i][j]=s;}
```

这样做可以减小内存访问次数,增加寄存器利用率.此外,在外层向量化的时候,外层循环中定义的由标量组合的向量还可以作为内层循环向量化中的一些初始 pack,并以此作为种子进行 pack 扩展,从而在向量化之后

© 中国科学院软件研究所 http://www.jos.org.cn

可以提高向量寄存器的利用率,提升程序的效率.循环不变数组引用外提可以分为如下两个步骤:

- 分析数组引用是否为循环不变量;
- 如果确定该数组引用是循环不变量,并且在循环中存在对该数组引用的读操作,则在循环前将数组引用的初始定义值赋给一个标量 s,将其在循环内的所有引用都用 *s* 代替,如果在循环中存在对该数组引用的写操作,则在循环后将 *s* 赋值给该数组引用.

3 基于 SLP 的多重循环向量化

向量化操作就是将若干同构语句并行执行,1 条操作只能被向量化 1 次,所以一般的向量化研究都是分析在哪一层循环上进行向量化可以得到最大的收益.而事实上,影响向量化效果的因素有多种,比如在一些程序中,外层循环虽然能够向量化,但是数组引用却相对于内层循环连续,而内层循环却由于依赖关系的阻碍而不能向量化,如果单独地对哪一层循环进行向量化,都不能取得很好的收益;而同时对外层循环和内层循环进行直接循环展开和压紧,然后用 SLP 进行有效的向量化发掘,往往能取得更好的效果.

我们首先在第 3.1 节中介绍了直接循环展开和压紧这种循环变换方法,如果能够进行这种变换,即代表可以在不影响循环结构的情况下对该循环进行向量化;接着在第 3.2 节中分析了影响向量化效果的主要因素;在第 3.3 节中提出了如何利用这些因素决定在哪些循环进行直接循环展开和压紧;最后在第 3.4 节中描述了如何利用 SLP 算法对变换后的循环进行向量化发掘及代码生成.

3.1 直接循环展开和压紧

为了避免复杂的循环变换,直接对程序进行向量化,本文采用直接的循环展开和压紧.与通常所讲的循环展开和压紧不同,经过直接循环展开和压紧变换之后,由同一条源语句展开而来的子语句彼此相邻.这种变换相当于将循环中的每条语句在原位置复制若干次,并修改相应的循环索引(循环可变量),进行重命名(标量),如例 2 中的循环在 *j* 层进行直接循环展开和压紧后如例 3 所示.事实上,这种循环变换融合了循环展开压紧和语句顺序调整,变换后的循环满足如下 3 个条件:

(1) 内层循环的结构不变;

(2) 语句的相对顺序不变;

(3) 由同一个源语句展开后的语句处于相邻的位置.

容易看出,如果可以对某层循环进行直接循环展开和压紧,则该循环就可以直接进行向量化.但是这种循环变换可以保持内层循环的结构不变,这样就可以用同一种方法递归地处理多层嵌套循环,并且循环变换之后便于 SLP 进行向量化.因此,本文采用直接循环展开和压紧作为向量化的基本程序变换方法.

值得注意的是,循环展开和压紧是针对外层循环的,内层循环只需要进行循环展开,不需要压紧.本文所述的直接循环展开和压紧实际上是进行了循环展开压紧和语句顺序调整,对应于内层循环就是循环展开和语句顺序调整,为了叙述上的统一性,统称为直接循环展开和压紧.

事实上,如果经过循环变换之后,语句满足条件(3)即可立即对该语句进行向量化.而如果语句在该层循环上没有构成依赖环,理论上就可以通过循环展开和语句顺序调整等方法使展开后的语句满足条件(3),但是可能满足这个条件的循环变换太多,太复杂,从而难以通过简易的步骤来实现.传统向量化每次从循环中分裂出不在依赖环中的语句进行向量化,对于内层循环没有影响,所以其变换满足条件(3)和条件(1),但其通过依赖关系图对语句进行了重排,可能会进行语句顺序的调整,所以不满足条件(2).传统向量化也可以用同一种方法递归地对各层循环进行处理,但是由于 SIMD 向量长度的限制和访存连续性要求使得传统向量化方法不适于面向 SIMD 的向量化.

例 3:
```
for (i=0; i<M; i++)
    for (j=0; j<M; j+=2){
        s1=0;
        s2=0;
```

© 中国科学院软件研究所 http://www.jos.org.cn

```
for (k=0; k<N; k+=1){
s1+=a[i][k]*b[k][j];
s2+=a[i][k]*b[k][j+1];}
c[i][j]=s1;
c[i][j+1]=s2;}
```

3.2 影响向量化的主要因素

影响循环向量化效果的主要因素包括:能否对该层循环进行直接循环展开和压紧,即用依赖关系判定能否在不影响循环结构的情况下对循环进行向量化;相对于循环索引连续的数组引用数量;以及该循环所包含的区域.下面分别就这 3 种因素分别加以讨论.

3.2.1 针对直接循环展开和压紧的依赖关系分析

进行依赖关系分析是为了确保程序变换的正确性,使得变换之后的程序和原来的程序语义相同.对循环进行直接循环展开和压紧,实质上就是在不影响循环结构的情况下对循环进行向量化,向量化也可以看作是一种并行.因此,可以借助于并行化的依赖关系判定[4]来分析能否进行直接循环展开和压紧.并行化本意上是将循环的每一个独立的迭代转换为一个并行线程,如果某个循环满足循环并行化的要求,显然它可以安全地进行直接循环展开和压紧.因此,可以将循环并行化的充要条件——循环不携带依赖,作为循环可以进行直接循环展开和压紧的充分条件.面向 SIMD 的向量化可以看作是一种有限长度的并行,对应的循环展开次数一般情况下等于向量化因子,所以,当外层循环所携带的依赖距离大于向量化因子时不影响向量化,也认为其不携带依赖.

由于标量的依赖具有跨迭代的性质,所以如果在循环中存在对标量的读写,就会造成跨迭代依赖,使得不能对循环进行直接循环展开和压紧.消除标量依赖的影响一般有两种方法:一种是标量扩展,另一种是标量私有化.标量扩展总是安全的,可以用于任何嵌套循环,经过扩展之后就可以消除标量造成的循环携带假依赖.但是标量扩展有一个明显的缺点,即增加了内存需求和内存访问量,从而拖累了向量化效率.标量的私有化主要用于进行并行性分析,不会增加额外的代价,所以本文在对外层循环进行依赖关系分析时采用私有化的方法来处理标量所携带的跨迭代依赖,如果私有化之后循环不存在循环携带依赖,则认为循环可以进行直接循环展开和压紧.如例 2 所示,s 相对于循环索引 i 和 j 来说是可以私有化的,私有化之后就没有循环携带依赖,所以可以认为 i 层循环和 j 层循环都能够进行直接循环展开和压紧.

由于 SLP 采用的是将同构语句组成 pack 的方式进行向量化,所以在实际进行循环变换时,可以采用标量重命名的方法对能够私有化的变量进行重命名,以保证对外层循环进行直接循环展开和压紧的正确性.标量重命名就是确保每个标量都只有一次定义,如果存在对标量的重复定义,就将挑选一个标量将其定义和引用都进行重命名.

3.2.2 相对于循环索引连续的数组引用

由于 SIMD 扩展只能对连续的数组引用进行向量化,所以影响向量化效果的因素除了运算的向量化之外,还有数组引用的连续性.前者通过并行执行同样的操作提升性能,后者可以避免跨幅访存带来的性能损失.对前者来说,如果不存在依赖的阻碍,则可以对语句中的操作进行向量化;后者则需要通过对齐分析来决定,如果数组引用对于循环索引的跨幅为 1,则说明数组引用对该循环索引是连续的,否则就是不连续的.此外,运算的向量化只能进行 1 次,而数组引用如果相对于其他的循环索引连续,则可以通过进一步的展开进行优化.例 2 中最内层循环中的语句为 $s+=a[i][k]*b[k][j]$,对最外层循环索引 i 来说,没有语句携带依赖,所以这条语句可以在 i 层进行向量化,$v_s+=(a[i][k],a[i+1][k],\ldots,a[i+VF-1][k])*(b[k][j],b[k][j],\ldots,b[k][j])$.但是,因为 $a[i][k]$对 i 不连续,所以如果要将$\{a[i][k],a[i+1][k],\ldots,a[i+VF-1][k]\}$装载到一个向量寄存器中,则可能需要耗费较长的时间.因此可以对循环 k 进行直接循环展开和压紧,然后进行向量化装载,将$\{a[i][k],a[i][k+1],\ldots,a[i][k+VF-1]\},\ldots,\{a[i+VF-1][k],a[i+VF-1][k+1],\ldots,a[i+VF-1][k+VF-1]\}$等分别装载到向量寄存器中,最后再通过向量化重组指令得到向量$\{a[i][k],a[i+1][k],\ldots,a[i+VF-1][k]\}$等.假设向量化因子 $VF=2$,则这种多重循环向量化的例子如图 3 所示.图 3(a)所示的语句 S_1,S_2 对应的标量 t_1,t_2 需要组合在一个向量中,但是因为 S_1,S_2 对应的数组引用不连续,所以可以对循环索引 k

© 中国科学院软件研究所 http://www.jos.org.cn

进行直接循环展开和压紧,如图 3(b)所示;展开之后,按照内存连续的原则将 S_1,S_3 以及 S_2,S_4 组合起来(如图 3(c)所示),按照对齐装载的顺序装载至向量 V_1,V_2 中,然后再通过重组得到需要的向量 V_3,V_4(如图 3(d)所示).

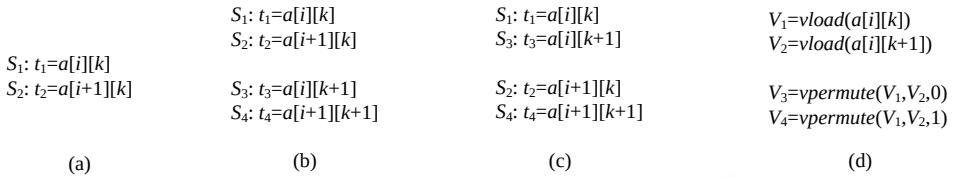

Fig.3 Vectorization at multi loop levels

图 3 多重循环向量化示意图

3.2.3 循环区域

与并行化追求粗粒度相似,如果有更多的语句能够向量化,则最后向量化的效果就可能更好.所以在一般情况下,如果外层循环没有依赖关系阻碍向量化,则优先选择外层循环进行向量化.但是这种选择方法不一定是最优的,如例 2 所示,循环 i 和 j 都可以进行向量化,但是在循环中没有对 i 连续的数组引用,并且循环层次 i 和 j 之间没有语句,即循环 i 和循环 j 的向量化区域是一样的,所以只需对循环 j 进行向量化即可,对循环 i 进行直接循环展开和压紧不仅不会增加向量化的效果,反而会由于基本块中语句数目的增加影响 SLP 向量发掘的效率.但是也不能简单地认为,如果没有数组引用对循环索引连续就不对其进行直接循环展开和压紧,如例 4 所示,循环中所有的数组都不对 i 连续,都只针对 j 连续,但是循环 j 由于携带依赖而不能向量化,如果不对循环 i 进行直接循环展开和压紧,循环中的语句就不能被向量化.为了解决这些问题,本文引入组的概念来衡量循环所包含的区域,如果循环索引对应的循环层次之间没有语句,则认为其处于相同的组中,在同一组中优先挑选没有依赖关系阻碍且有数组引用连续的循环进行循环展开和压紧,如果没有,则选择组内可以向量化的循环进行循环变换.

例 4:
```
for (i=0; i<N; i++)
    for (j=0; j<N–3; j++)
        A[i,j+1]=a*A[i,j]+b*A[i,j+2];
```

3.3 决定循环变换方案

本文针对影响外层循环向量化的 3 个因素,从整体上寻找一种优化的向量化方案.为此,从外至内对各个循环层次进行分析,为每个循环附加 3 个值域(*vectorable*,*narray*,*veclevel*),其中,*vectorable* 是一个布尔变量,为 1 代表可以在该循环层次进行直接循环展开和压紧,否则就代表有依赖关系阻碍在该层次进行循环变换;*narray* 是一个整形变量,代表循环内对该循环索引连续的数组引用数量;*veclevel* 是一个整形变量,代表循环所在的组,假定最外层循环所在的组为 1,从外至内依次递增.如例 2 所示,循环 i 和 j 之间没有语句,所以认为它们处于同一组中,其对应的 *veclevel* 值为 1,循环 j 和 k 之间存在语句,所以 k 对应的 *veclevel* 有不同的值,值为 2.例 2 中各层循环对应的属性值见表 1.

Table 1 Attribute values for the loops illustrated in example 2

表 1 例 2 中各层循环对应的属性值

	Vectorable	*Narray*	*Veclevel*
i	1	0	1
j	1	2	1
k	0	1	2

根据每层循环所附带的这些信息,就可以根据不同的向量化策略得到不同的循环变换方案,下面给出一个激进的向量化策略,对所有可能取得收益的循环都进行直接循环展开和压紧.

1. 按照 *veclevel* 的值对循环进行排序,将具有相同 *veclevel* 值的循环分到一组中;
2. 从外至内对各组循环(非最内层循环)进行分析,如果该组中存在 *vectorable* 为 1 且 *narray* 不为 0 的循环,则对所有 *vectorable* 为 1 且 *narray* 不为 0 的循环进行直接循环展开和压紧;如果该组中所有 *vectorable*

© 中国科学院软件研究所 http://www.jos.org.cn

为 1 的循环其 *narray* 都为 0 并且还没有外层循环进行过直接循环展开和压紧,则选择最内层的 *vectorable* 为 1 的循环进行直接循环展开压紧;

3. 如果最内层循环对应的 *vectorable* 和 *narray* 均不为 0,则对其进行循环展开和语句顺序调整;否则,如果 *narray* 不为 0 并且已有外层循环进行过直接循环展开和压紧,则对其进行循环展开.

根据该算法对例 2 进行分析,首先对 *veclevel* 为 1 的组进行分析,该组中存在 *vectorable* 为 1 且 *narray* 不为 0 的循环索引 *j*,所以对 *j* 层进行直接循环展开和压紧,*i* 层循环保持不变.接着分析 *k* 层循环,因为它是最内层循环且循环中存在数组对 *k* 连续,所以对其进行循环展开.按照这种变换策略对例 2 中的循环进行循环变换后,如例 5 所示(假设向量化因子为 2).

例 5:
```
for (i=0; i<M; i++)
    for (j=0; j<M; j+=2){
        s1=0;
        s2=0;
        for (k=0; k<N; k+=2){
        s1+=a[i][k]*b[k][j];
        s2+=a[i][k]*b[k][j+1];
        s1+=a[i][k+1]*b[k+1][j];
        s2+=a[i][k+1]*b[k+1][j+1];}
        c[i][j]=s1;
        c[i][j+1]=s2;}
```

这种向量化策略采用比较激进的方法来确定循环变换方案,适用于循环内语句比较少、循环层次也不多的情况,否则会造成寄存器压力过大反而影响向量化的效率.事实上,可以根据目标系统的特点选取不同的向量化策略,以充分发挥目标系统的性能.

3.4 用SLP进行向量发掘和代码生成

2000 年,Larsen 和 Amarasinghe 提出了 SLP 的概念,这是一种针对 SIMD 体系结构的向量化算法.SLP 向量化的对象是基本块,通过识别基本块中的同构语句来识别可向量化语句,这些同构语句构成的集合称为 pack,识别过程即为 pack 生成过程[13].SLP 一般需要经过以下几个步骤:

(1) 将基本块按照一定的因子进行展开.因子越大,展开后的代码长度就越长,可能生成 pack 的候选语句就越多,但是识别过程就越复杂;反之,因子越小,展开后的代码长度就比较短,识别过程就比较简单,展开因子一般等于 *VF*;

(2) 对齐分析,主要是分析含有数组语句的对齐信息;

(3) 进行预优化操作,比如三地址化、常量传播、冗余 load/store 删除、死代码消除等优化;

(4) 按照地址相邻的原则对基本块中的语句生成初始的 pack;

(5) 根据 DU 和 UD 链对 pack 进行扩展;

(6) 合并有共同语句的 pack;

(7) 按照语句依赖关系对 pack 进行调度,如果有依赖关系阻碍数据 pack 的调度,则需要串行执行.

由于需要对多个基本块进行向量化,所以本文采用的向量化方法与 Larsen 和 Amarasinghe 提出的 SLP 算法略有不同,主要表现在以下两个方面:

(1) SLP 的向量化对象是基本块,所以对于多重循环的向量化只能将循环体按照基本块的定义划分为若干个基本块,然后依次对每个基本块进行向量化.这些基本块之间可能有一些标量的定义使用关系,在进行下一个基本块的向量化时,需要以前一个基本块中已经打包的标量作为初始的 pack,并以此为种子按照定义使用链进行扩展;

(2) 由于在决定循环变换方案时已经进行了循环展开压紧或者循环展开,所以就不用再进行循环展开,直

© 中国科学院软件研究所 http://www.jos.org.cn

接进行 pack 生成即可,在进行向量发掘时,直接将 pack 长度定义为向量化因子.

不对齐访存会对向量化程序的效率有较大影响,而一些替代方法,比如对齐访存和移位操作可以提供更高的性能,所以在向量化中尽量采用一些方法避免直接使用不对齐的向量访存,比如循环剥离、循环多版本等.但是这些方法并不能彻底解决不对齐访存的问题,因为语句中的数组引用可能有着不同的偏移.此外,外层循环向量化的对齐分析和内层循环向量化还有一些区别,如在例 6 所示的 convolve 循环中,如果只对循环索引 h 进行展开,则对数组 $image[v+i][h+j]$来说虽然可以保证 h 是向量化因子的整数倍,但是 j 每次迭代步长都增加 1,所以对整个数组引用来说其地址是不对齐的.由于本文提出的激进向量化方案对所有可能取得收益的循环都进行展开,经过 SLP 的冗余 load/store 删除,可以删除许多冗余的数组访问,最后只对对齐的数组引用进行向量装载,其他的不对齐访存由这些对齐的访存再加上移位和加法操作得到.

例 6:
```
for (v=0; v<N; v++)
    for (h=0; h<N; h++) {
        s=0;
        for (i=0; i<K; i++)
        for (j=0; j<K; j++)
            s+=image[v+i][h+j]*filter[i][j];
            out[v][h]=s>>factor;}
```

由于目标系统可能在处理某些操作时效率较高,而处理另一些操作时则性能较低,为此可以根据目标系统的特点在向量化代码生成时进行一些变换,以提升系统的效率.比如,在某些处理器上将标量组装成向量时代价比较大,所以可以在代码生成阶段尽量避免将标量组装成向量,而用其他可替代的操作来组合实现.比如现在需要将常量(0,0,0,0,0,0,0,1)组装成一个向量,为了避免使用 *simd_set*(0,0,0,0,0,0,0,1)这种比较耗时的操作,可以先将向量 V 中的值全赋为 0,$V_1=V_1\hat{}V_1$,然后再将 1 插入至相应的位置 *simd_ins0*(V_1,1).

4 实验结果及分析

基于 open64 实现了这种针对多重循环的向量化算法,在 IBM x3650 上进行实验,实验平台采用 Intel 至强处理器 5500,内存 4G.处理器的向量化寄存器长度为 128bit,可以同时处理 4 个 int 型数据或者 2 个 double 型数据.后端基础编译器采用 Intel C/C++ compiler (v11.0),源程序首先经过基于 open64 的向量化编译器将源程序转化为向量化程序,然后再用基础编译器编译后在实验平台上运行.

表 2 简要描述了实验所用到的程序核心片段,其中 2 个来自多媒体和信号处理领域:一个是有限脉冲反应滤波器(Fir),另一个为 2D 卷积运算(convolve);2 个是常用的矩阵乘运算 MMM 和 MMM_double,规模均为 64×64,区别是 MMM 为 float 类型,MMM_double 为 double 类型;3 个来自 spec2000 测试集:173.applu 中核心函数 buts 中的循环(buts),187.galgel 核心函数 sysnsn 中的循环(sysnsn),200.sixtrack 核心函数 thin6d 中的循环(thin6d);1 个来自 NPB3.2SER 测试集:UA 热点函数 transfb_nc2 中的循环(transfb).这些程序具有广泛的代表性,包含了向量化所具有的一些特征,诸如多种数据类型、归约、特殊的处理方式以及跨幅访问.

Table 2 Benchmarks

表 2 测试集

程序名	N_1	N_2	N_3	N_4	δ_1	δ_2	δ_3	δ_4	数据类型
Fir	128	16			0, 1	1, 1			short-int
MMM	64	64	64		64, 0	0, 1	1, 64		float
MMM_double	64	64	64		64, 0	0, 1	1, 64		double
convolve	128	128	16	16	144, 128, 0	1, 0	144, 16	1	short-int
buts	5	5	5		1, 0	5, 1			double
sysnsn	128	16	128	16	16, 1, 0	1, 0	32 766, 128, 0	2048, 16, 0	double
transfb	5	5	5		1, 0	5, 0	5, 0		double
sixtrack	64	16			64, 0	1			float

© 中国科学院软件研究所 http://www.jos.org.cn

表格的第 2 列~第 5 列由外至内(即从 N_1 到 N_4)显示了各层循环的迭代次数,多个数据代表有多重循环,如果循环层次少于 4,则其中有一些空格.比如,Fir 有两重循环,外层循环迭代次数为 128,内层循环迭代次数为 16.表格的第 6 列~第 9 列(即从 δ_1 到 δ_4)由外至内显示了循环中的数组引用相对于各个循环索引的访问跨幅,不同的数据表示循环内不同的数组引用相对于循环索引有不同的访问跨幅,0 代表有数组引用,相对于循环索引是循环不变的,1 代表有数组引用,相对于循环索引是连续的.比如,convolve 循环内有数组引用 *out*[*v*][*h*],*image*[*v*+*i*][*h*+*j*]和 *filter*[*i*][*j*],则其相对于最外层循环索引 *v* 的访问跨幅就是 128,144 和 1.

对这些程序的核心片断分别按照源程序、内层循环向量化、外层循环向量化以及本文提出的混合向量化在目标系统上进行测试,然后求出 3 种向量化方法相对于源程序的加速比,测试结果如图 4 所示.inner 代表内层循环向量化的加速比,outer 代表最好的外层循环向量化的加速比,complex 代表按照本文提出的激进的多重循环向量化策略向量化之后得到的加速比.

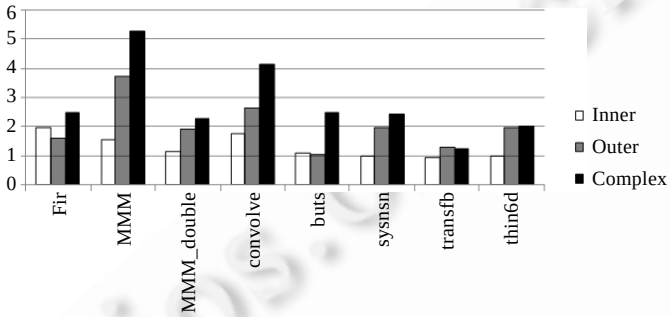

Fig.4 Speedups for vectorization alternatives

图 4 不同的向量化方法得到的加速比

从向量化结果来看,混合向量化都取得了较高的加速比,convolve 中有 4 重循环,循环中的数组引用相对于 *j* 和 *h* 连续,其中,*j* 是最内层循环,*h* 是外层循环.在所有的外层循环中,对 *h* 进行向量化可以得到最好的效果,所以挑选 *h* 层作为外层循环进行测试.循环层次 *h* 和 *i* 之间存在语句,所以循环分为两组,混合向量化对 *h* 层进行直接循环展开和压紧,对 *j* 层循环进行展开,3 种向量化得到的加速比为 1.77,2.62 和 4.14.因为循环中的数组引用对 *h* 层连续,并且外层向量化没有做归约操作向量化所需要的一些附加操作,所以可以取得比内层循环向量化更高的加速比.sixtrack 中的核心片断 thin6d 由于在内层循环存在依赖环而无法向量化,所以对内层循环进行向量化得到的加速比为 1.虽然循环中没有对外层循环索引连续的数组引用,但是对外层循环进行向量化没有依赖关系的阻碍,并且其中存在许多标量,也可以认为其对外层循环索引连续,外层循环向量化的加速比可以达到 1.94.循环体内存在对最内层循环索引连续的数组引用,所以混合向量化对其进行展开,然后向量化,得到了更高的加速比 2.03.Fir 核心循环中的数组引用相对于最内层循环索引连续,但是存在归约操作,所以需要附加一些操作才能向量化;没有数组引用相对于外层循环连续,但是外层循环可以向量化且不需要进行归约变换,3 种向量化方法得到的加速比为 1.98,1.61 和 2.5.buts 的情况与 Fir 类似.sysnsn 中的循环是一个紧嵌套循环,只有最内层循环中存在语句,并且对各层循环来说都没有依赖关系的阻碍,循环中的数组引用有 4 个相对于最外层循环索引 *i* 连续,5 个相对于次外层循环索引 *j* 连续,内层循环向量化得到的加速比为 1,选取 *j* 进行外层循环向量化,得到的加速比为 1.98;混合向量化对 *i* 和 *j* 都进行直接循环展开和压紧,最终得到加速比为 2.41.transfb 中有 6 个数组引用相对于最外层循环索引 col 连续,1 个数组引用相对于次外层循环索引 *i* 连续,对各层循环来说都没有依赖关系阻碍向量化,3 种向量化得到的加速比为 0.95,1.29 和 1.24.混合向量化的效果略低于外层循环向量化,原因是只有 1 个数组引用相对于循环索引 *i* 连续且不对齐.

图 5 显示了混合向量化相对于其他两种向量化方法得到的性能提升,从整体上来看,除了 transfb 以外,混合向量化都取得了高于内层循环向量化和外层循环向量化的加速比.相对于内层循环向量化,加速比平均提升了 2.13;相对于外层循环向量化,加速比平均提升了 1.41.所以,本文提出的面向 SLP 的多重循环向量化方法是可行

© 中国科学院软件研究所 http://www.jos.org.cn

且有效的.

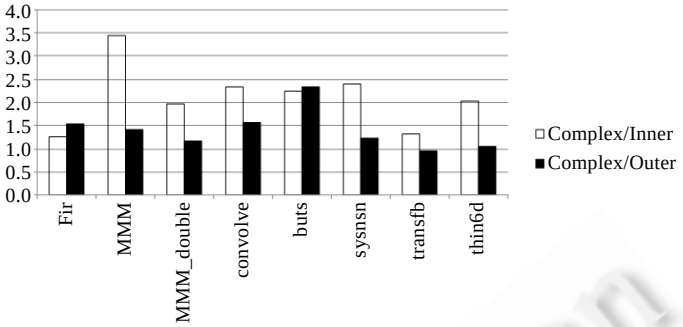

Fig.5 Speedup improvements for complex vectorization versus inner/outer vectorization

图 5 混合向量化相对于内层/外层循环向量化的性能提升

5 相关研究

Allen-Kennedy 提出的针对向量机[3]的传统向量化[4]方法,可以逐层地对循环进行向量化,通过求解强连通图[5]将那些不构成依赖环的语句向量化,并且在分析内层循环时假设外层循环串行执行,以消除外层循环引起的依赖.但当依赖环为内层循环所携带而与外层循环无关时,按照传统向量化算法就不能对其进行向量化,只有经过一些循环变换,比如循环交换才能对其进行向量化,而且这种方法不适于 SIMD 扩展.

文献[6,7]通过将外层循环移至最内层并进行向量化,外层循环可以不作任何变换而直接被向量化,这种向量化方法已经被实现在一个面向 SIMdD DSP 的自动向量化编译框架中[8].文献[9]通过循环分割和向量替换的方法可以直接对外层循环进行向量化,Wu 提出了一种基于虚拟寄存器的向量化编译框架[10],将迭代次数较少的内层循环看作一个特殊函数,从而实现对外层循环的向量化.这些向量化方法都是为了能在某个外层循环进行向量化,以实现更好的向量化效果.

文献[14]通过对紧嵌套循环进行外层循环展开压紧和内层循环展开来提升向量寄存器的利用率,减少访存次数,结合 SLP 算法可以得到更好的加速比;文献[11]也通过外层循环展开压紧并对 SLP 算法进行修改以实现对外层循环的向量化,但其研究对象是两层的紧嵌套循环.文献[12]基于多面体模型提出了可以在各个循环层次上对嵌套循环进行向量化,并提供了一种收益分析模型对各种向量化方案进行分析,得到收益最大的向量化方案.但是实施起来难度较大,一种向量化方案可能需要进行若干种循环变换和复杂的依赖关系分析,并且循环变换还可能影响程序的局部性,从而对向量化的效率产生影响.

6 总 结

本文针对多重循环向量化问题,根据影响向量化性能的 3 个方面——能否进行直接循环展开和压紧、相对于该循环索引连续的数组引用数量以及该循环所包含的区域,提出了针对不同循环层次进行直接循环展开和压紧来寻找更优的向量化策略.首先分析了满足什么样的依赖关系才能进行直接循环展开和压紧,然后通过收集各个循环层次对应的属性值并通过一定的向量化策略对循环进行直接循环展开和压紧,最后通过 SLP 算法对循环中的基本块进行向量化.

本文在第 3.3 节中提出了一种激进的多重向量化策略,但是这种向量化策略还比较简单.事实上,当循环中语句较多时,这种激进的向量化策略会导致寄存器压力过大从而影响向量化的性能;其次,如果循环中的数组引用相对于外层循环索引的跨幅较大,进行循环展开就有可能影响 cache 命中率.如何将这些因素引入向量化策略中,是值得进一步研究的课题.

致谢 在此,谨向对本文提出宝贵建议的审稿专家以及参与本文内容讨论的所有老师和同学表示衷心的感谢.

© 中国科学院软件研究所 http://www.jos.org.cn

References:

[1] Stewart J. An investigation of SIMD instruction sets. University of Ballarat School of Information Technology and Mathematical Sciences, 2005. http://noisymime.org/blogimages/SIMD.pdf

[2] Nuzman D, Rosen I, Zaks A. Auto-Vectorization of interleaved data for SIMD. In: Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation. Ottawa: ACM Press, 2006. 132−143. [doi: 10.1145/1133981.1133996]

[3] Zheng WM, Tang ZZ. Compiler Archtecture. Beijing: Tsinghua University Press, 1998 (in Chinese).

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

[5] Shen ZY, Hu ZA, Liao XK, Wu HP, Zhao KJ, Lu YT. Methods of Parallel Compilation. Beijing: National Defence Industry Press, 2000 (in Chinese).

[6] Bik AJC. The Software Vectorization Handbook—Applying Multimedia Extensions for Maximum Performance. Intel Press, 2004.

[7] Hampton M, Asanovic K. Compiling for vector-thread architectures. In: Proc. of the 6th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization. Boston: ACM Press, 2008. 205−215. [doi: 10.1145/1356058.1356085]

[8] Naishlos D, Biberstein M, Ben-David S, Zaks A. Vectorizing for a SIMdD DSP architecture. In: Proc. of the 2003 Int'l Conf. on Compilers, Architecture and Synthesis for Embedded Systems. San Jose: ACM Press, 2003. 2−11. [doi: 10.1145/951710.951714]

[9] Bik AJC, GirKar M, Grey PM, Tian XM. Automatic intra-register vectorization for the Intel architecture. Int'l Journal of Parallel Programming, 2002,30(2):65−98. [doi: 10.1023/A:1014230429447]

[10] Wu P, Eichenberger AE, Wang A, Zhao P. An integrated simdization framework using virtual vectors. In: Proc. of the 19th Annual Int'l Conf. on Supercomputing. Cambridge: ACM Press, 2005. 169−178. [doi: 10.1145/1088149.1088172]

[11] Tenllado C, Piñuel L, Prieto M, Catthoor F. Improving superword level parallelism support in modern compilers. In: Proc. of the 3rd IEEE/ACM/IFIP Int'l Conf. on Hardware/Software Codesign and System Synthesis. Jersey City: ACM Press, 2005. 303−308. [doi: 10.1145/1084834.1084909]

[12] Trifunovic K, Nuzman D, Cohen A, Zaks A, Rosen I. Polyhedral-Model guided loop-nest auto-vectorization. In: Proc. of the 18th Int'l Conf. on Parallel Architectures and Compilation Techniques. Raleigh: IEEE Computer Society, 2009. 327−337. [doi: 10.1109/PACT.2009.18]

[13] Larsen S, Amarasinghe S. Exploiting superword level parallelism with multimedia instruction sets. In: Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation. Vancouver: ACM Press, 2000. 145−156. [doi: 10.1145/349299.349320]

[14] Shin J, Chame J, Hall MW. Compiler-Controlled caching in superword register files for multimedia extension architectures. In: Proc. of the 2002 Int'l Conf. on Parallel Architectures and Compilation Techniques. Charlottesville: IEEE Computer Society, 2002. 45−55. [doi: 10.1109/PACT.2002.1106003]

附中文参考文献:

[3] 郑伟民,汤志忠.计算机系统结构.北京:清华大学出版社,1998.

[5] 沈志宇,胡子昂,廖湘科,吴海平,赵克佳,卢宇彤.并行编译方法.北京:国防工业出版社,2000.

魏帅(1984－),男,河南邓州人,博士生,主要研究领域为先进编译技术.

姚远(1972－),男,副教授,主要研究领域为先进编译技术.

赵荣彩(1957－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为先进编译技术,软件逆向工程.

© 中国科学院软件研究所 http://www.jos.org.cn