面向SIMD的数组重组和对齐优化

魏 帅 赵荣彩 姚 远 侯永生
(解放军信息工程大学信息工程学院 郑州 450002)

摘 要 随着多媒体应用的普及,越来越多的处理器集成了SIMD扩展,但是非连续或者非对齐访存会阻碍程序的向量化或者造成性能损失。针对实际应用中出现的数组引用不连续的情况,提出了一种数学模型,用以刻画数组的访存模式和数据重组方案,以判断这些数组引用是否可以通过数组转置的方法满足连续性要求;并采用过程间数组填充、循环剥离和基于SLP的向量化代码生成方法等进行对齐优化。最后基于SPEC2000测试集对该算法进行了测试,结果表明,该方法可以有效地提升向量化程序的执行效率。

关键词 SIMD,对齐分析,数据重组,多维数组填充

中图法分类号 TP311 **文献标识码** A

Data Regroup and Alignment Optimization Based on SIMD

WEI Shuai ZHAO Rong-cai YAO Yuan HOU Yong-sheng
(Information Engineering College, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract With the popularity of multimedia applications, more and more processors are integrated with SIMD extensions. However, incongruous or unaligned memory accesses incur performance degrade. This paper brought out the a mathematic model to describe array reference and array regroup, in order to do intra-procedural data regroup to avoid incongruous array reference, and it also adopted inter-procedural array padding, loop peeling and optimized code generation to avoid misalignment. At last the experiments implemented on SPEC2000 benchmark, show that combining with SLP, this algorithm can effectively improve the program performance.

Keywords SIMD, Alignment optimization, Data regroup, Multi-dimension array padding

1 引言

随着个人PC功能的不断增强,人们对PC的期望越来越高,已经不满足于简单的字符交互,而是期望其能提供丰富多彩的音频和动画,而多媒体应用大多具有计算密集工整、数据长度短、易于并行处理等特点,这就为SIMD扩展(单指令多数据)的出现奠定了应用基础。自从1996年Intel在奔腾处理器集成了MMX后,越来越多的通用处理器集成了SIMD(Single Instruction Multiple Data)扩展,例如AMD处理器提供了3DNOW/E3DNoW!扩展,PowerPC处理器提供了AltiVec扩展,Alpha处理器提供了MVI扩展[1]等。

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[2],gcc[3],open64[4]等都集成了自动向量化的功能。

与传统向量机不同的是,当前的SIMD扩展只支持连续的向量化装载或者存储指令,而且大多数只提供对齐的向量化装载或者存储指令,比如PowerPC处理器的AltiVec扩展,Alpha处理器的MVI扩展等;另外一些指令集如MAX-1/2,VIS,3DNow,VMX用软件实现非对齐访存,本文实验所用的平台SW-1600也是用软件实现的非对齐访存;Intel的MMX和SSE系列指令集则在硬件上提供了对非对齐内存访问的支持,但是对齐和非对齐的性能差异较大。而在许多应用程序中,循环内的数组引用常常是不连续的或者是不对齐的,这就会阻碍程序的向量化,即使进行了向量化也不能取得较好的收益。

本文针对SIMD体系结构的特点,提出了面向SIMD的数据重组和对齐分析技术。针对实际应用中出现的数组引用不连续的情况,提出了一种数学模型来刻画数组的访存模式和数据重组方案,以判断这些数组引用是否可以通过一定的重组方案(主要是数组转置)来满足连续性要求。针对多维数组低维长度不是向量化因子的整数倍而导致的数组引用不对齐的情况,结合数组填充的特点,提出了面向全局的多维数组

到稿日期:2011-03-15 返修日期:2011-06-20 本文受核高基国家科技重大专项(2009ZX01036)资助。

魏 帅(1984-),男,博士生,主要研究方向为先进编译技术,E-mail:weis0906@163.com;赵荣彩(1957-),男,博士,教授,博士生导师,CCF会员,主要研究方向为先进编译技术、软件逆向工程;姚 远(1972-),男,硕士,副教授,主要研究方向为先进编译技术;侯永生(1978-),男,博士生,主要研究方向为先进编译技术。

填充。针对数组访问的对齐性要求，采用多层次的对齐优化方法，对循环进行循环剥离和循环多版本，并在代码生成阶段结合 SLP(Superword Level Parallelism，超字并行)算法[5]进行优化的向量化代码生成。最后在基于 spec2000 程序的测试中表明了这些优化方法可以提升向量化程序的效率。

2 相关研究

2.1 数据重组

数据重组技术包括全局数据重组和局部数据重组，前者需要改变数据的全局声明和程序中的引用方式，需要在整个程序中保持别名一致[6]，包括多维指针展平、数组转置、数组压缩和数组合并等。结构体数组及指针在多媒体程序中应用广泛，面向数据局部性的基于复用距离[7]分析的结构拆分[6]、数据重组[8,9]等，可以提高 cache 命中率，提升程序性能。基于复用距离分析的结构属性域调整[10](Structure Field Reording)通过建立一种 cache 敏感的结构属性域亲密图(Structure Field Affinity Graph)，判断重组结构的属性域，然后进行结构体成员拆分和重组的方法来提高数据访问的局部性，提升 cache 的命中率。这些优化技术主要是为了增加程序的数据访问局部性，提升 cache 命中率，而并非以 SIMD 向量化为目标。

李玉祥[11]提出了一种面向向量化的局部数据重组，在循环之前对数据进行重新布局，等待向量化之后再对数据进行恢复，就但是这种局部数据重组代价比较大，往往难以取得收益，而且其实质也是为了提升程序的局部性，只是在重组的时候加入了对数组引用对齐的考虑。采用全局数据重组不需要对数组进行重新布局和恢复，可以取得更大的收益，但是需要进行过程间分析，而且通常需要将程序中对这些数据的引用方式进行修改。所以如果数据重组需要复杂的收益分析和代码修改，就不适宜在全局进行，否则会因为代价太大而难以实现。

2.2 对齐分析

数组填充和数据重组通过改变数据布局来增加对齐连续访问，而循环分裂和循环多版本则通过循环变换来避免非对齐访问造成的性能损失。

Larsen 等人[12]讨论了如何在过程间对数组引用进行连续性分析，并综合循环剥离、循环多版本、数组填充等方法使得能够对更多的数组引用进行对齐访问。文献[13-15]采用动态剥离的方法分析指针引用的对齐信息，并应用在 Intel 的 C++编译器中。Fridman 等人[16]用冗余的系数存储来解决常数数组引用的对齐访问问题，Shahbahrami 等人[17]综合这些方法在 Intel 平台上进行了实验，分析了非对齐访存可能造成的性能损失。

文献[18]基于传统向量化方法提出了在复杂情况下如何利用移位操作和对齐访存来实现对数组的不对齐访问，首先将循环中的语句抽象为数据重组图(Data Reorganization Graph)，然后基于非对齐访存需要满足的条件提出了 4 种移位策略，并通过实验分析了各种移位策略的优劣。文献[19,20]对其进行了扩展，分析了如何解决动态的非对齐访存以及如何在特定条件下实现代价最小的非对齐访存。Nuzman 等人[21]研究了如何利用两种特定的交织和收集操作来实现对固定跨幅数组引用(主要是访问跨幅为 2/4/8 的数组引用)的向量化。钱兴隆等人[22]基于 Intel 平台借助插入指令和移位指令来避免非对齐访存，提高寄存器的重用率，提升向量化效率。

3 数组重组

向量化一般都是对数组进行的，阻碍向量化的两个主要因素：一个是数组的连续性，一个是数组的对齐性。理论上所有的数组对齐和连续性问题都可以通过数组重组的方法来解决，但是有时候程序中对数组的引用方式差别很大，难以用一种统一的重组方式来解决，如果针对每一种数组访问模式都构造一个新的数组来满足对齐和连续性要求，重组的代价往往会大于向量化取得的收益。本文从实际应用中经常遇到的数组访问不连续和多维数组不对齐问题，结合数组填充的特点，提出了过程内数组转置和过程间数组填充，使得数组经过这些变换之后能满足向量化所需的对齐连续要求。

3.1 过程内数组转置

由于进行全局的数组重组需要进行过程间分析，代价太大；而面向循环的局部数据重组由于重组的开销通常会大于重组带来的收益，难以取得效果，因此本文采取了折中的办法，即采用面向过程的数据重组，亦即对函数内所有循环的数据访问方式进行统计分析，然后找出一种适合进行向量化的数据布局，在过程开始时对数据进行重新布局，在过程结束时对数据进行恢复。在科学计算应用中，许多应用的核心函数运行时间超过整体运行时间的一半以上，里面存在一些核心循环对数组进行重复访问，在过程内只进行一次数据重组和恢复，这些开销相对于重组带来的收益相对较小，所以易于取得收益。而在过程内进行分析又无需进行过程间分析，使得这种重组方法易于实现。

数组重组的难点在于如何对数组的访问模式进行统计，并得到一种重组方案，使得经过重组之后适合进行向量化的数组引用数量最多。本文提出了一种数学模型来刻画数组的访存模式和数据重组方案，以判断这些数组引用是否可以通过一定的重组方案来满足连续性要求。理论上这些重组模型可以包括数组转置、数组填充、数组压紧、数组拆分等，但是由于数组填充通常用于对齐性分析且比较简单，其他的重组方式不是经常使用，因此本节讨论的数组重组方式主要是指数组转置。图 1 给出数组转置实例。

```
do 20 j=1,napx                          do 20 j=1,napx
    xv(1,j)=xv(1,j)+stracki*yv(1,j)         xv'(j,1)=xv'(j,1)+stracki*yv'(1,j)
    xv(2,j)=xv(2,j)+stracki*yv(2,j)         xv'(j,2)=xv'(j,2)+stracki*yv'(j,2)
    sigmv(j)=sigmv(j)+stracki*(c1e3-rvv(j)*     sigmv(j)=sigmv(j)+stracki*(c1e3-rvv(j)*
            (c1e3+(yv(1,j)                          (c1e3+(yv'(j,1)
  +  *yv(1,j)+yv(2,j)*yv(2,j))*c5m4))     +  *yv'(j,1)+yv'(j,2)*yv'(j,2))*c5m4))
20  continue                            20  continue
        (a)                                     (b)
```

图 1 数组转置实例

1. 数组访存模式

大部分的数组引用可以用仿射函数表示，如下所示：

$$\vec{a}=A\vec{i}$$

式中，向量 $\vec{i}$ 代表数组引用所在的循环各层的索引，以及附加的一个常量 1。如果数组引用所在的循环是 N 层嵌套循环，则 $\vec{i}$ 就有 $N+1$ 维。$\vec{a}$ 代表该数组引用，其维数与该数组定义的维数一致，假设有 M 维。A 是数组引用相对于循环索引的映射矩阵，规模为 $M*(N+1)$。

比如在图 1(a)所示的循环中，$xv(1,j)$(Fortran 的数组表达式，采用行优先的存储方式，如果用 C 语言来表示就是 xv

[j][1])是一个2维的数组引用，其所在的循环也是1维的，则 $\vec{i}$ 就是一个2维向量$(j,1)^T$，A 就是一个 $2*2$ 的仿射矩阵 $\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix}$。

2. 数组访存变换函数

数据访存变换可以包括数组转置、数组填充、数组压紧、数组拆分等，不同的数组变换方式对应不同的数组访存变换函数，但是都可以用统一的数学模型进行表示。

变换函数由矩阵 F 定义，F 是一个 $T*M$ 维的矩阵，T 是进行转换之后新数组的维数，如果是数组转置，T 就是一个 $M*M$ 维的幺模矩阵，进行数组变换之后，就会产生一个新的数组引用：

$$\vec{a}'=A\vec{i}=FA\vec{i}$$

比如在图1(a)所示的循环中，如果需要对数组引用 $xv(j,1)$ 进行转置，产生一个新的数组 $xv'(j,1)$，则循环转换矩阵 F 就为 $\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$，$FA$ 相乘之后产生的转换矩阵就为 $\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$。

3. 重组方案选择

重组方案可能根据采用的向量化方法不同而有所不同。如果不考虑外层循环的向量化，只对最内层循环进行向量化，数组重组的目的就是使得该数组引用相对于最内层循环索引连续。如果数组引用对最内层循环索引连续，也可以提高程序的局部性，提升 cache 命中率。

如果该数组引用适于在最内层循环向量化，则该数组引用相对于最内层循环索引不变或者连续。从数组引用矩阵的角度来看，就是映射矩阵 A 的倒数第二列(倒数第一列代表的是常数偏移)对应的向量是零向量或者$(0,\cdots,0,1)$。零向量代表该数组引用相对于最内层循环索引不变，$(0,\cdots,0,1)$ 代表该数组引用相对于最内层循环索引是连续的。如果映射矩阵 A 的倒数第二列不是$(0,\cdots,0,1)$，而是$(0,\cdots,1,\cdots,0)$(其中只存在一个元素为1)，就可以通过数组转置的方法将该数组转置为一个新的数组，使得这个新数组相对于最内层循环索引连续。

判断过程是否适合通过循环转置的方法进行向量化，就是分析过程内多数的数组引用是否能通过统一的数组转置使其适合进行向量化。具体步骤如下：

1. 收集循环中对该数组的引用，建立数组引用矩阵。

2. 对所有关于该数组的数组引用进行分类，将那些可以用相同的转置矩阵 F 变换成连续数组引用的数组引用归为一类。

3. 如果某一类中的数组引用数目占整个数组引用的百分比大于一定的阈值，则判定可以按照该数据转置方案对数组进行重组。

4. 如果需要对数组进行转置，则在过程开始时对数组进行转置，在过程结束时对数组进行恢复，并对过程内的数组引用进行修改。

在 spec2000 的 sixtrack 应用中，核心函数为 thin6d，其运行时间占到整体运行时间的98%。其中一个典型的循环即如图1(a)所示，其中存在大量的数组引用诸如 $xv(1,j)$ 或 $yv(2,j)$，相对于内层循环索引 j 数组引用不连续，导致主流的向量化编译器诸如 icc，gcc 等不对其进行向量化。而采用本节所述的方法，以数组 xv 为例，循环中存在相应的数组引用 $xv(1,j)$ 和 $xv(2,j)$，经过分析发现可以通过相同的数组转置矩阵使其满足连续性要求，则对其进行循环转置，并改变程序中对该数组的引用。以同样的方法也可以对 yv 进行数组转置，转置之后的循环如图1(b)所示，这样就可以对循环进行向量化，并且提高了数组的局部性，提升了 cache 效率。

3.2 过程间数组填充

数组填充主要是当数组低维长度不是向量化因子的整数倍时，增加数组低维的长度，使其在进行向量化的时候能够统一地按照对齐的方法进行向量化装载或者存储。在一些循环中，数组的低维长度不是向量长度的整数倍，在进行向量化的时候难以判断此次访存是对齐访问还是非对齐访问，为了避免频繁地进行判断而保守地采用非对齐访问。如图2(a)中的循环所示：假定向量寄存器长度为256bit，则每个向量寄存器可以装载4个数组元素，而 a 数组的最低维长度为1335，不是4的整数倍，所以相对于循环索引 i 的每次迭代难以判断数组引用 $a[i][0]$ 是否对齐，比如 $a[0][0]$ 是对齐的，而 $a[1][0]$、$a[2][0]$ 和 $a[3][0]$ 是不对齐的，即当 i 能被4整除的时候数组 $a[i][j]$ 是对齐的，否则就是不对齐的。如果在进行循环向量化的时候附加一些判断语句和优化措施则会降低向量化的性能，并且会造成代码膨胀。为了解决这个问题，可以进行数组填充，增加数组的存储空间，使得数组的最低维长度是该数据类型的整数倍，这样就容易统一按照对齐方式对其进行向量化。对图2(a)中的循环来说，数组的最低维长度为1335，不是4的整数倍，如果将其最低维长度增加1，则其长度就变为1336，就能被4整除，虽然存储空间有少许增加，但是都能用统一的对齐的向量化指令进行代码生成，进行数组填充之后的循环如图2(b)所示。

```
double a[1335][1335]
for(i=0; i<1335;i++)
for(j=0; j<1335;j++)
a[i][j] = a[i][j] * 2.0;
```
(a)

```
double a[1335][1336]
for(i=0; i<1335;i++)
for(j=0; j<1335;j++)
a[i][j] = a[i][j] * 2.0;
```
(b)

```
double a[1335][1335];
double* p = &(a[0][0]);
for(i=0; i<1335*1335;i++){
p[i] = p[i] * 2.0;}
```
(c)

```
double a[1335][1336];
double* p = &(a[0][0]);
for(i=0; i<1335*1336;i++){
p[i] = p[i] * 2.0;}
```
(d)

图2 多维数组填充实例

数组填充的另一个优点就是进行代码填充之后，如果程序中不存在该数组的别名(对该数组指针方式的引用)，则不需要对这些引用进行修改，因为数组填充只是对数组的存储空间进行扩充，不影响数据的存储顺序。

虽然数组填充也属于数据重组，但是因为向量化因子一般是2、4、8、16，所以对数组低维长度增加的幅度不会很大，进行数组填充后数组增加的存储空间可以忽略，而进行数组填充后会增加向量化的效率，所以不需要像普通的数据重组一样对程序中数组的引用方式进行收益分析，判断其是否能够取得收益。多维数组填充只需在判断数组定义时，如果发现其不为向量化因子的整数倍，直接对其进行填充即可。由于对数组填充无需进行收益分析且不需要对程序内的数组引用进行复杂的修改，因此可以在全局范围进行多维数组填充。

数组填充的难点在于如何保证程序的合法性。假设数组为 a，则当对数组进行正常引用(比如 $a[2][3]$)时，数组填充不影响程序的正确性。但是当对数组用指针方式进行引用

时，就需要进行一定的变换才能保证程序的正确性。如图 2(c)所示，数组 a 存在别名 p，对数组 $p[i]$ 的修改即是对数组 a 内容的修改。如果只是对数组 a 进行填充，而不对程序进行修改就会造成程序运行错误。在图 2(c)中，如果需要对数组进行填充，对循环的终止条件也需进行相应的修改，这样才能保证循环是正确的。修改之后的循环如图 2(d)所示。

由于在 Fortran 程序里，对指针的使用较少，因此适于进行数组填充。但是在一些 C 语言中，由于指针使用比较灵活，使得别名分析难以进行，或者程序中的一些参数运行时才能确定，这时就无法保证数组填充的正确性，从而不适宜进行数组填充。

4 对齐优化

4.1 对齐预优化

本文主要采用循环剥离和循环多版本对循环内的数组引用进行对齐优化。循环剥离是对齐优化的一个重要手段。以图 3(a)中的循环为例，如果直接对其进行向量化，则向量化之后的程序中会出现两个非对齐 load 操作和一个非对齐的 store 语句，这种非对齐的向量化访存指令会拖累向量化的效率。

```
for(i=0;i<N;i++)
a[i+2] = b[i+2] + c[i+2]
```
(a)

```
for(i=0;i<2;i++)
  a[i+2] = b[i+2] + c[i+2]
for(i=2;i<N;i++)
  a[i+2] = b[i+2] + c[i+2]
```
(b)

```
for(i=0;i<N;i++)
 p[i] = B[i];
```
(c)

```
for(i=0;i<N;i++){
if (p%32 == 0)
p[i] = b[i];  //向量化之后采用对齐的数组装载指令
else
p[i] = b[i];  //向量化之后采用非对齐的数组装载指令
}
```
(d)

图 3 循环剥离和循环多版本实例

针对这个问题，可以进行循环剥离，剥离后的循环如图 3(b)所示：在循环剥离之后的循环中，所有的数组引用是对齐的，向量化之后的效率就会大大增加。进行循环剥离首先要收集循环中所有数组引用的对齐信息，然后找到一种可能满足大多数数组引用对齐要求的循环剥离方案，使得进行循环剥离后循环中对齐的数组引用数目达到最大。当循环中数组引用的对齐方式各不相同时，单纯依靠循环剥离无法使循环内所有的数组引用都变为对齐访问。

有时候数组的对齐信息未知或者在运行时才能确定，这时就需要进行循环多版本，如图 3(c)中所示的循环中数组 p 的对齐信息未知，为了对齐优化就需要进行循环多版本，经过循环多版本之后就如图 3(d)所示。循环多版本需要增加一些判定条件，以判定在什么条件下数组引用是对齐的，什么情况下是不对齐的，然后针对不同的情况对循环进行处理。有时候数组引用不同的偏移对应不同的优化方案，这时需要判定其对齐方式是何种形式，假设循环中有 m 条数组引用，每个数组引用需要判定 n 种对齐方式，则总共需要进行 n^m 次判定。如果只是判定对齐或者不对齐，则只需要判定 2 种对齐方式，多版本也需要进行 2^m 次判定。当循环中的语句数目较多时，这种判定语句数目过于庞大反而会造成性能降低，所以多版本只适于循环中数组引用较少的情况。

本文采用的对齐预优化框架如图 4 所示，首先收集循环内各个数组引用的对齐信息，如果有一些数组引用的对齐信息不确定并且数组引用较少，则考虑用循环多版本进行优化。如果所有数组引用的对齐信息确定，则判断这些数组引用的对齐信息是否一致，即判定这些数组引用相对于向量化因子的偏移是否相等，如果相等，则直接进行循环剥离，剥离之后循环内所有的数组引用都是对齐的，当然这种情况比较少见。大多数情况下这种对齐信息不一致，这时按照各种对齐信息对数组引用进行归类，计算各个对齐信息对应的数组引用数目，然后按照数目最大的偏移进行循环剥离。进行循环剥离之后就可以满足大多数数组引用的对齐要求，但是可能还存在一些非对齐访存，这些非对齐访存可以在向量化代码生成阶段进一步进行优化。

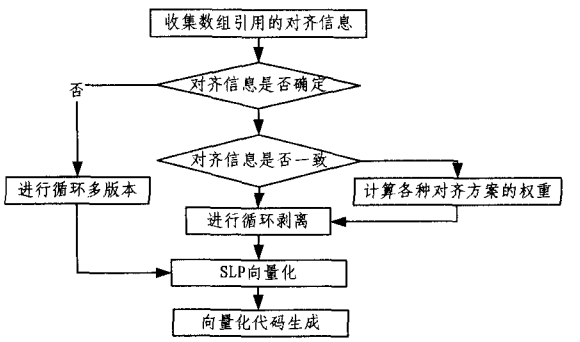

图 4 面向 SLP 的循环剥离和循环多版本的算法框架

4.2 结合 SLP 的向量化代码生成

本文采用 SLP 方法[5]对程序进行向量化。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包(pack)，识别过程即为 pack 生成过程。SLP 算法的主要步骤如下：

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越短，识别过程就越简单，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。

如果 load pack 中的语句不对齐或者不连续，可以首先进行对齐的向量化装载，将需要的数组元素装载进向量寄存器中，然后通过向量寄存器之间的重组操作来实现所需的向量寄存器。比如对 load pack：$\{s1=a[4*i+1], s2=a[4*i+2], s3=a[4*i+3], s4=a[4*i+4]\}$，可以产生对应的对齐的向量化代码 $load(v1, \&(a[4*i]))$ 以及 $load(v2, \&(a[4*i+4]))$，则向量寄存器 $v1$ 中对应的标量为 $\{a[4*i], a[4*i+1], a[4*i+2], a[4*i+3]\}$，向量寄存器 $v2$ 中对应的标量为 $\{a[4*i+4], a[4*i+5], a[4*i+6], a[4*i+7]\}$，然后通过向量化重组指令提取 $v1$ 和 $v2$ 中的元素组成向量 $\{a[4*i+1], a[4*i+2], a[4*i+3], a[4*i+4]\}$，具体重组过程如图 5 所示。其中 $v1$ 和 $v2$ 还可以通过软件流水的方法进一

步地进行优化，即每次都保存上一次进行对齐装载的向量寄存器值，然后与当前的向量寄存器值重组生成所需的向量寄存器，这样可以减少冗余访存，提升程序效率。

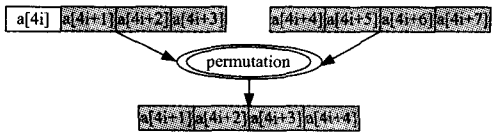

图 5　非对齐的向量化装载

为了使 SLP 能够对程序更好地进行向量化，一般都需要在 pack 生成之前进行三地址化。三地址化就是将语句中出现的所有数组元素都用一个标量相关联，也就是所有数组元素只可能出现在两类运算中，即 $t=a[i]$ 和 $a[i]=t$（t 为标量），将这两类语句分别称为 load 语句和 store 语句，由其组成的 pack 称为 load pack 和 store pack。其他语句按照正常的三地址化进行分裂，分裂后每个语句中最多只能有两个原子操作数，因为所有的数组元素都用标量关联，所以这种语句中的源操作数和目的操作数都只能为标量，将这种类型的语句称为 computation 语句，将其形成的 pack 称为 computation pack。这也对应于向量化运算通常需要的 3 个步骤，即将数据装载到向量中，进行向量运算，将向量运算的结果再存入内存中。如果 load pack 或者 store pack 中所包含的标量存储语句是非对齐连续的，就需要采用一定的方式进行优化，以避免直接采用非对齐存储指令对向量化性能产生影响。

相对于 load pack，如果非对齐连续的 store pack 用对齐的向量化存储指令进行存储，则需要进行一些循环剥离工作。比如对 store pack：$\{a[4*i+1]=R1, a[4*i+2]=R2, a[4*i+3]=R3, a[4*i+4]=R4\}$，其中 $\{R1,R2,R3,R4\}$ 存放在一个向量寄存器 $v1$ 中，就不能简单利用移位的方法产生向量化代码 store($v1$, &($a[4*i]$))或者 store($v1$, &($a[4*i+4]$))；因为 $v1$ 中所有的数组元素必须要写到这些内存地址中，否则就不能保证程序的正确性。一种类似于 load pack 的处理方法是将相邻的 store pack 用向量重组指令产生对齐的向量化 pack，将其余的装载按照标量执行，具体过程如图 6 所示，可以看出，非对齐的 store pack 在向量化代码生成的时候需要在循环开始和结束时进行一些特殊处理。同样地，store pack 也可以用软件流水进行优化。

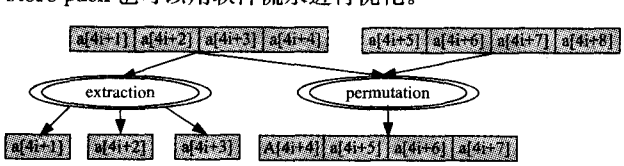

图 6　非对齐的向量化存储

5　实验结果和分析

本文中使用的自动向量化编译系统是基于开源编译器 Open64[4] 框架实现的 SW-VEC，编译环境为 Linux 操作系统，版本为 Readhat Enterprise 5。实验平台 CPU 主频为 2.0GHz，内存为 2GB，L1 数据 cache 为 32kB，L2 cache 为 256kB，基本页面为 8kB，向量寄存器的宽度为 256 位，可以同时处理 4 个浮点型数据或者 8 个整形数据。

实验时，首先用 SW-VEC 将源程序转化为向量化程序，然后再用基础编译器编译成二进制代码并在国产 CPU SW-1600 上运行，最后用串行程序的运行时间除以向量化程序的运行时间便得到向量化的加速比。同样采用本文所提出的优化方法结合 SLP 对源程序进行向量化就可以得到对齐优化的加速比。

过程内数组转置测试所用的核心函数片段在 SPEC CPU2000 标准浮点测试集的 sixtrack 中提取，即其核心函数为 thin6d，其中存在着大量的非对齐数组引用，其中一个典型的循环如图 1 所示。针对其中存在的相对于内层循环索引 j 不连续的数组引用 $xv(1,j)$ 或 $yv(1,j)$ 进行重组，在函数开始时对这些数组进行转置，即将其行列顺序互换，得到新的重组后的数组 xv' 和 yv'，则相应的新的数组引用即为 $xv'(j,1)$ 或 $yv'(j,1)$。不经过重组向量化和经过重组之后向量化得到的加速比如图 7 所示。由于本文采用的是 SLP 向量化方法，如果未进行重组循环只存在少量连续的数组引用，因此不能对其进行向量化，加速比为 1。进行重组之后，大部分循环都能进行向量化，并且提升了局部性，向量化效率可以得到明显提升。

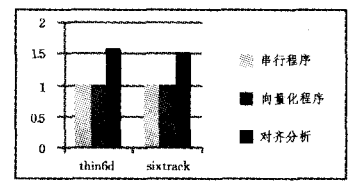

图 7　过程内数组转置收益对比图

综合测试采用 SPEC2000 中的部分核心函数和整体应用作为测试集，分别就串行程序、向量化程序、经过对齐优化 3 个版本进行了测试，最后给出了各个版本的加速比，如图 8 和图 9 所示。从图中可以看出，本文所提出的数组重组和对齐分析优化方法可以明显地提升向量化效率。其中 swim 主要是由于数组扩展，使得程序中大部分数组引用都能实现对齐方式的向量化，因此性能得到了提升。mgrid，applu，galgel，apsi 中存在着大量的非对齐数组引用，通过循环剥离和多版本，可以增加程序中的对齐访问。facerec 和 equake 存在一些不连续访存和非对齐的数组引用，通过数据重组和对齐优化，也可以在一定程度上提高向量化的性能。

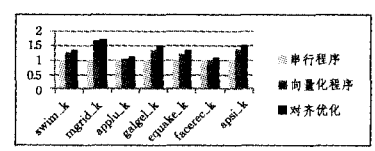

图 8　不同向量化方法对 SPEC2000 核心循环的加速比

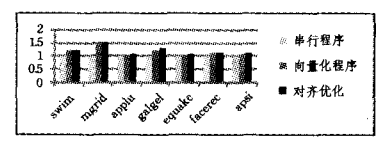

图 9　不同向量化方法对 SPEC2000 应用程序的加速比

结束语　由于 SLP 算法能更好地对科学计算领域的应用进行向量化，而现有的对齐分析方法大多针对传统向量化方法，因此本文在对现有一些数组重组和对齐优化技术分析的基础上，提出了面向 SIMD 的数据重组和对齐分析优化技术，主要包括数组转置、数组填充、对齐预优化以及结合 SLP 的向量化代码生成。最后基于 SPEC2000 测试集对该优化方法进行了测试，实验结果表明该方法结合 SLP，可以提升向量

化程序的执行效率。

参 考 文 献

[1] Stewart J. An Investigation of SIMD instruction sets[M]. University of Ballarat School of Information Technology and Mathematical Sciences, 2005

[2] icc[OL]. http://icc. gnu. org

[3] Free Software Foudation, GCC[OL]. http://gcc. gnu. org

[4] Open64[OL]. http://open64. sourceforge. net

[5] 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. June 2000: 145-156

[6] Hagog M, Tice C. Cache Aware Data Layout Reorganization Optimization in GCC[C]// Proceedings of the GCC Developers' Summit. June 2005: 69-92

[7] Beyls K. Software Methods to Impmve Data Locality and Cache Behavior[D]. Ghent Universitv, 2004

[8] Zhong Y, Orlovich M, Shen X, et al. Array regrouping and structure splitting using wholeprogram reference affinity[C]// Proceedings of PLDI'04. June 2004: 255-266

[9] Fu Xiong, Wang Ru-chuan. Locality-based Data Reorganization Framework [J]. Computer Science, 2009, 36(2)

[10] Chilimbi T M, Davidson B, Larus J R. Cache-conscious Structure Definition[C]// Proceedings of PLDI. May 1999: 13-24

[11] 李玉祥，施慧，陈莉. 面向非多媒体程序的 SIMD 向量化算法的研究及改进[J]. 小型微型计算机系统, 2009, 30(10): 1927-1935

[12] Larsen S, Witchel E, Amarasinghe S. Techniques for Increasing and Detecting Memory Alignment [R]. LCS-TM-621, MIT/LCS. November 2001

[13] Bik A J C. The Software Vectorization Handbook: Applying Multimedia Extensions for Maximum Performance[M]. Hillsboro, Intel Press, 2004

[14] Bik A J C, Girkar M, Grey P M, et al. Automatic Intra-Register Vectorization for the Intel Architecture[J]. International Journal of Parallel Programming, 2002, 30(2): 65-98

[15] Larsen S, Witchel E, Amarasinghe S. Increasing and Detecting Memory Address Congruence[C]// Proc. 11th Int. Conf. on Parallel Architectures and Compilation Techniques. September 2002: 18-29

[16] Fridman J. Data Alignment for Sub-Word Parallelism in DSP[C]// Proc. IEEE Workshop on Signal Processing Systems. October 1999: 251-260

[17] Shahbahrami A, Juurlink B, Vassiliadis S. Performance Impact of Misaligned Accessesin SIMD Extensions

[18] Eichenberger A E, Wu P, O'Brien K. Vectorization for SIMD Architectures with Alignment Constraints[C]// Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation. June 2004

[19] Wu P, Eichenberger A E, Wang A. Efficient simd code generation for runtime alignment and length conversion[C]// Proceedings of CGO. Washington, DC, USA, 2005: 153-164

[20] Fireman L, Petrank E, Zaks A. New Algorithms for SIMD Alignment

[21] Nuzman D, Rosen I, Zaks A. Auto-Vectorization of Interleaved Data for SIMD[C]// PLDI'06. June 2006: 132-143

[22] 钱兴隆，臧斌宇，朱传琪. 一种 SIMD 优化中的向量寄存器部分重用方法[J]. 计算机工程与科学, 2007: 29(5)

(上接第 279 页)

[4] Niu Y, Shen L. The Optimal Multi-objective Optimization Using PSO in Blind Color Image Fusion[C]// International Conference on Multimedia and Ubiquitous Engineering. 2007: 970-975

[5] Nebro A J, Luna F, Alba E, et al. AbYSS: Adapting scatter search to multiobjective optimization[J]. IEEE Transactions on Evolutionary Computation, 2008, 12(4): 439-457

[6] Brockhoff D, Zitzler E. Are all objectives necessary? On dimensionality reduction in evolutionary multiobjective optimization [M]. Parallel Problem Solving from Nature-PPSN IX, 2006: 533-542

[7] Deb K, Saxena D K. Searching for Pareto-optimal Solutions Through Dimensionality Reduction for Certain Large-dimensional Multi-objective Optimization Problems[C]// IEEE Congress on Evolutionary Computation. 2006: 3353-3360

[8] Saxena D, Deb K. Non-linear dimensionality reduction procedures for certain large-dimensional multi-objective optimization problems: Employing correntropy and a novel maximum variance unfolding[C]. New York, 2007: 772-787

[9] Saxena D, Deb K. Dimensionality Reduction of Objectives and Constraints in Multi-objective Optimization Problems, A System Design Perspective[C]// IEEE Congress on Evolutionary Computation. Hong Kong, 2008: 3352-3360

[10] Drechsler D, Drechsler R, Becher B. Multi-objective Optimization Based on Rdlation Favour [C] // Proc 1st Evolutionary Multi-Criterion Optimization. Berlin, 2001: 154-166

[11] Di Pierro F, Djordjevic S, Khu S-T. Automatic Calibration of Urban Drainage Model Using a Novel Multi-objective GA[J]. Water Science and Technology, 2004, 52(5): 41-52

[12] Fleming P J. Many-objective optimization: An engineering design perspective[J]. Lecture Notes in Computer Science, 2005, 34 (10): 14-32

[13] Hernandez-Diaz A, Santana-Quintero L, Coello-Coello C, et al. Pareto-adaptive ε-dominance [J]. Evolutionary Computation, 2007, 15(4): 493-517

[14] Kang Z, Kang L, Zou X, et al. A new evolutionary decision theory for many-objective optimization problems[M]. Advances in Computation and Intelligence, 2007: 1-11

[15] 杨咚咚，焦李成，公茂果，等. 求解偏好多目标优化的克隆选择算法[J]. 软件学报, 2010, 21(1): 14-33

[16] Leung Y, Wang Y. An orthogonal genetic algorithm with quantization for global numerical optimization[J]. IEEE Transactions on Evolutionary Computation, 2002, 5(1): 41-53

[17] 曾三友，魏巍，康立山，等. 基于正交设计的多目标演化算法[J]. 计算机学报, 2005, 28(7): 1153-1162

[18] Huband S, Hingston P, Barone L, et al. A review of multiobjective test problems and a scalable test problem toolkit[J]. IEEE Transactions on Evolutionary Computation, 2006, 10(5): 477-506

[19] Deb K, Laumanns T L M, Zitzler E, et al. Scalable multi-objective optimization test problems[J]. Proceedings of the 2002 Congress on Evolutionary Computation(Cec'02), 2002(1/2): 825-830