面向自动向量化的结构体优化

于海宁 韩 林 李鹏远
(解放军信息工程大学数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 结构体广泛应用在科学计算等应用程序中,向量化结构体数组存在的非连续和非对齐访存会严重影响程序的向量化效果。为减少结构体数组SIMD向量化过程中的非连续和非对齐数据访问,提出了基于域访问亲和度与域数据类型相结合的结构体拆分模型,以消除域存储间的内存"间隙";同时利用结构体数组到二维数组的地址映射方式来满足结构体数组向量化时的访存连续和对齐要求,以降低Cache的失效率,从而提升应用程序性能。在自动向量化系统SW-VEC上,选取gcc-vec、spec2000和spec2006标准测试集中部分相关的测试用例,测试结果表明:与相应的串行程序相比,采用该方法后,测试用例程序性能加速比提高了8%以上。

关键词 访问亲和度,结构体拆分,地址映射,SIMD向量化
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.2.045

Structure Optimization for Automatic Vectorization

YU Hai-ning HAN Lin LI Peng-yuan
(State Key Laboratory of Mathematical Engineering and Advanced Computing, PLA Information Engineering University, Zhengzhou 450001, China)

Abstract Sturcture is used more extensively to promote the performance of application program such as scientific computing. The noncontinuity and the nonalignment of its non-array memory address have a dramatic influence on the efficiency of program's vectorization. To reduce the access to these addresses during the SIMD's vectorization, this paper applied a structure peeling model based on the structure which combines field access affinity with type to eliminate the "clearance" of memory between field storage, and proposed an address conversion method of structure array one by one mapping to the two dimensional array to meet the request of the continuity and the alignment of its non-array memory address, further reducing the failure rate of Cache, so as to improve application performance. By using the test suites of gcc-vec, spec2000 and spec2006, the experimental results on the compiler of automatic vector show that using the method, the performance of optimized programs can be improved by more than eight percent.

Keywords Access affinity, Structure peeling, Address mapping, SIMD vectorization

1 引言

在科学计算程序中存在大量的结构体数据类型,结构体数组是一种常见的数据结构,结构体一般为多种数据类型的组合。存储时,为满足域存储地址对齐和结构体对齐的要求,其存储地址通常是非连续的,可能存在内存"间隙"。结构体数组可借助多维数组实现结构体数组的迭代变化。在C、C++程序中,结构体的各个域都被分配到相邻的内存区域,这样访问结构体数组中所有结构体中特定的域就变成了访问多维数组中特定行或者列中的元素。

目前,对于图形处理、非多媒体、科学计算等大型应用程序,影响其SIMD向量化效果的因素主要有两个方面[1]:1)大量图形处理、科学计算等程序中使用大量的指针,指针别名分析和保守的依赖分析严重影响向量化效果;2)程序中存在非对齐和非连续的数据引用,阻碍程序向量化的发掘。

本文主要研究如何减少程序中结构体数组的非对齐和非连续数据引用,充分发掘结构体向量化。结构体的域通常包括指针、数组、非数组等。

对结构体中的数组域进行SIMD向量化,首先分析数组首地址是否对齐,对非对齐的结构体中的数组访存进行对齐优化,但与此同时,会增加额外的开销,影响程序SIMD向量化后的性能提升。通过对齐优化来保证每次访问数组域时其首地址都是对齐的,尽可能降低结构体进行SIMD向量化引入的对齐优化和连续优化开销,提高程序SIMD向量化后的性能。因此对于结构体的数组域,本文不作考虑。结构体非数组域的引用在循环迭代内通常是非连续的,在循环迭代间也是非连续的,非数组域引用地址间也是非连续的,存在为满足地址对齐化要求而进行数据填充的内存"间隙",只有对其进行大量的向量拼接操作才能实现SIMD向量化,额外的操作开销可能严重降低甚至抵消结构体SIMD向量化的收益。

到稿日期:2014-12-03 返修日期:2015-04-27 本文受"核高基"国家科技重大专项(2009ZX01036)资助。
于海宁(1989-),男,硕士生,主要研究方向为先进编译技术、高性能计算,E-mail:yuhaining0111@163.com;韩 林(1978-),男,博士,副教授,主要研究方向为先进编译、高性能计算;李鹏远(1989-),男,硕士生,主要研究方向为高性能计算、先进编译技术。

2 相关研究

结构体数组及结构体指针(指针可视为隐式数组)在科学计算程序中应用广泛,运行时程序访问的大部分结构体数据集中在结构体的少数属性域,通过与数组重组局域相逆思想的数据变换——结构体拆分[3-5],使其在内存的存储地址满足对齐和连续要求,再利用 SIMD 向量化技术来提升程序性能。

目前,对结构体程序分析和优化变换的技术主要包括基于复用距离[2]分析的结构属性域调整[3]和结构拆分[4]、数据重组[5,6]等,通过优化数据的局部性来提高 Cache 命中率,减少程序中非对齐或非连续数据访问。

如何量化结构体属性域之间的关系,寻求一起访问的属性域,从而获得子结构体的拆分方法,一直是进行结构体拆分的难点。结构体拆分可以将引用频繁的数组域拆分成一个单独的结构体,使其首地址满足地址对齐要求,同时对此数组域所在的结构体进行数据填充,使得每次访问都是地址对齐的,避免了非对齐访问带来的额外开销,但结构体拆分优化改变的是结构体的全局声明,需要在整个程序中保持别名一致[4]。

结构体数组存储访问的地址连续优化[5]方法与局部数据重组的方案大致类似,包括多维指针展平、数组转置、数组压缩和数组合并等,即在循环开始前申请一块容量为循环中访问需要重组的数据大小的用于重组的内存空间。在局部数据引用之前定义临时数组进行结构体数组到临时数组的数据拷贝,通过临时数组进行 SIMD 向量化运算,计算完成后进行临时数组到结构体数组的写回操作。此优化方法需要耗费较大的空间和数据拷贝的时间,程序的性能提升程度受计算复杂度的影响,可能达不到最优性能,经过代价分析,可能会认为此循环进行向量化后无收益,因而不进行向量化。

Chilimbi[7]和 Hundt[8]等通过在结构各域组间插入指针以保持域组间的可寻址,额外引入指针以及指针的间接引用增加了程序对于缓存以及内存传输宽带的需求和访问的开销。Curial 等人[9]为每个域组建立内存池,内存池长度正比于相应的域组长度。同一结构体各域组的内存池在内存空间连续存放。各域组同步地分配和释放内存以保证同一结构体实例的不同域组间内存地址偏移可计算,域地址在程序运行时动态计算。YAN 等人[10]同样应用内存池存放域组,完全按数据最优布局进行结构体数据重布局。利用静态的内存池基地址结合域组内偏移来计算域地址,避免了运行时动态计算内存池的基址,同时循环中域地址计算操作可被编译器优化。Linux 系统中使用 mremap 机制来降低数据移动开销,该机制由于需要 Linux 内核支持,因此几乎无可移植性。

文献[3]建立了一种基于复用距离分析的结构属性域调整(Structure Field Reording)的 Cache 敏感的结构属性域亲密图(Structure Field Affinity Graph)来度量结构体属性域之间的关系,依据属性域关系模型来调整结构体属性域位置顺序,从而优化程序的数据布局,该优化技术并非以 SIMD 向量化为目标,而是为了提高数据局部性,提升 Cache 命中率。同时,文献[5]在复用距离的基础上将结构体拆分和属性域顺序调整相结合,定义了一种访问亲密性(Reference Affinity)模型来量化结构属性域之间的关系。利用属性域访问的最大复用距离来描述属性域之间的线性关系,并基于分析结果进行结构体拆分。借鉴其思想,可以进行以 SIMD 向量化为目标的结构体属性域的拆分和位置调整,使得结构体数组中非数组域的引用在内存中连续存放,满足 SIMD 向量化的地址连续要求,同时进行对齐分析和优化,满足地址对齐要求。

对结构体数组的非数组域的操作多为非连续和非对齐访问,导致循环内结构体的 SIMD 向量化效果不佳。本文利用结构体域亲和度和域数据类型相结合的结构体拆分模型,对所引用的结构体进行拆分,为满足地址对齐和连续的要求,其将所引用的结构体数组的非数组域一一映射到二维数组,以达到访问非数组域地址对齐且连续的存储优化,充分发掘结构体的 SIMD 向量化,进而提升程序的 SIMD 向量化性能。

3 结构体向量化

3.1 结构体向量化框架

为解决结构体数组中非数组域访问的非连续问题,本文提出了一种结构体 SIMD 向量化的框架,如图 1 所示。该框架主要包括可向量化候选区域的确定、结构体数组数据引用等信息收集、结构体拆分、结构体数组地址映射和向量化代码生成等。

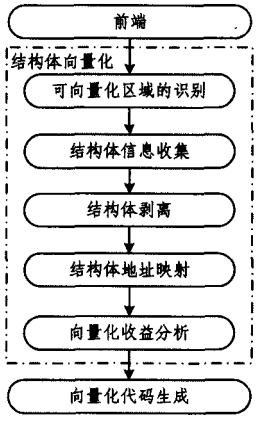

图 1 结构体向量化框架

(1)候选循环的判断主要是通过向量化预分析模块,从传统向量化算法的角度来分析循环特征,主要是排除不可向量化的循环,向量化候选区域主要满足如下条件:

- 当前循环只有唯一的入口,没有 GOTO 语句跳入循环体
- 循环中没有 GOTO、if 等控制流以及函数调用
- 循环的迭代次数在循环前已知且不能太少
- 循环中不能存在不完整的 DU 链,若存在,那么其引入的依赖会导致整个循环能被向量化
- 循环中存在可进行结构体 SIMD 向量化的操作

(2)结构体数组数据引用信息收集阶段,收集候选区域内的结构体数组引用点的信息,如非数组域的引用访问、内存分配、域数据类型等相关信息,并进行合法性检查。

(3)结构体拆分阶段,利用结构体域的访问亲和度与域数据类型相结合的结构体拆分模型,对当前向量化区域内所引用的结构体进行拆分,依据结构体信息收集阶段所采集到的相应的信息,对分裂产生的子结构体进行定义-使用链、数据引用访问等信息的更新。

(4)结构体数组地址映射阶段,为当前向量化区域内所引用的结构体数组动态申请固定大小的内存存储空间,要保证申请的内存空间的起始地址满足对齐方式的要求。将当前向量化区域内所引用的结构体数组映射为二维数组,当前引用的所有结构体数组元素的非数组域与二维数组中的元素一一映射,使其数据访问满足连续和对齐要求,以减少循环中SIMD向量化过程中的非连续访存和非对齐访存。

(5)向量化收益分析阶段,对结构体存储优化模型的合法性和收益进行分析,确保优化的正确性并取得较好的优化性能。

(6)向量化代码生成,将在3.4节中对其做详细介绍。

3.2 结构体拆分

结构体一般为多种数据类型的组合,在内存中存储时,要满足域地址对齐和结构体对齐的要求,在内存中通常是非连续的,可能存在内存"间隙"。结构体拆分是一种重要的数组重组方法,是按照引用点信息对结构体的域进行拆分的,将局部引用密集的数据域封装构建成新的子结构体[5]。通常采用访问亲和度模型[5](Reference Affinity)和紧密接近图[4](Close Proximity Graph)来度量结构属性域之间的关系,进行结构体拆分,以降低Cache的失效率,减少SIMD向量化过程中的非对齐和非连续访问。

本文利用基于访问亲和度[5]与域数据类型相结合的结构体拆分模型进行结构体拆分,生成域数据类型相同的子结构体,特别是只含数据类型相同的非数组域的子结构体,消除了为满足域地址对齐和结构体对齐而进行数据填充的内存"间隙"。

将结构体dd按要求拆分成4个新的子结构体,如图2所示。

```
//结构体数组的定义
typedef struct
{
double a[100];
double b[100];
double c[100];
float f;
double d;
int g;
double e;
short h;
}dd;
dd s[N];
//结构体数组成员的密集引用
for(i=0;i<100;i++){
dd.a[i]=dd.b[i]+dd.c[i];}
for(i=0;i<N;i++){
 s[i].e = s[i].b+5;}
```

结构体分裂

```
//结构体拆分后的定义
typedef struct
{
int g;
float f;
}dd;
typedef struct
{
short h;
}dd_0;
typedef struct
{
double a[100];
double b[100];
double c[100];
}dd1;
//结构体数组成员的密集引用
typedef struct
{
double d;
double e;
}dd2;
//成员引用密集的新结构体
for(i=0;i<100;i++){
dd1.a[i]=dd1.b[i]+dd1.c[i];}
for(i=0;i<N;i++){
 s[i].e =s[i].b+5;}
```

图2 结构体的分裂及优化

结构体在内存中存储时,要满足域地址对齐和结构体对齐的要求,在内存中难以连续存放。图2中所示的结构体以及分裂生成的子结构体在内存的存储形式如图3所示,图2中的原结构体数组中域a的首地址是对齐的,存储域d、e时都需要先填充4个字节,h存储完成后需要再次填充6个字节才能满足其存储对齐要求。

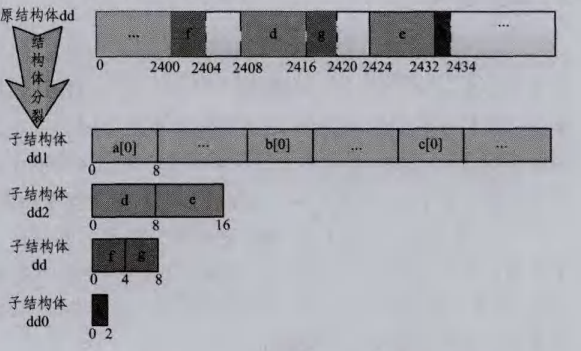

图3 原结构体以及子结构体在内存中的存储形式

域访问亲和度的度量主要借鉴访问亲和度模型[5](Reference Affinity)中的方法。下面是结构体拆分算法的描述:

1)收集程序中可向量化的结构体类型,并进行合法性检查。满足下列条件即为非法:域地址使用、类型逃逸、类型转换。

2)收集程序中的内存分配点,包括malloc、realloc。

3)结构体访问信息收集,包括域的数据类型、域的访问次数、读/写等。在循环级别针对每一个结构体记录域的访问次数。

4)基于结构体各域的访问次数确定域访问亲和度。

5)基于域访问的亲和度和数据类型来拆分结构体,将具有高亲和度的域组合成新的结构体。

6)基于域访问信息表,合并具有相同域访问模式的循环中的数据,形成亲和组。对于域集合相交的亲和组,若亲和组的热度超过既定的阈值,则进行亲和组合并。

3.3 结构体数组地址映射

本文假设编译器的缺省存储布局按行优先存储。为了达到提升数据局部性和增加可向量化循环的个数的目的,首先利用上文描述的结构体拆分模型对目标结构体进行结构体拆分优化,生成只含单一数据类型的子结构体,即子结构体中所有域的数据类型相同。

通常结构体数组的非数组域访问地址是非连续的,利用上述结构体拆分模型进行拆分后,采取只含单一数据类型非数组域的结构体数组到二维数组映射的方式,将当前分析的区域内的所引用的结构体数组的非数组域一一映射到相应的二维数组连续的每个数组元素当中,相当于动态地为当前区域内的结构体数组开辟一个固定大小的连续存储空间,该映射过程必须满足以下要求:

(1)动态分配内存空间

对于当前只含单一数据类型非数组域的结构体数组,结合收集的结构体数组的访问信息以及内存分配等信息,动态地为当前区域内所引用的结构体数组申请固定大小的连续内存存储空间,其大小为:$n \times N \times sizeof(type)$,其中,$n$是当前区域内结构体中非数组域的个数,$N$为当前区域内所引用的结构体数组的长度,$type$表示结构体中非数组域的数据类型。申请的内存空间起始地址要满足对齐要求。

(2)数据排列

当前结构体数组所引用的非数组域的序列与二维数组的

行序一一对应，二维数组的行数等于当前所引用的结构体中非数组域的个数，列数等于结构体数组的元素个数，行首地址要满足对齐方式的要求，以保证当前结构体数组的起始存储是对齐的。结构体数组中所有元素的第一个非数组域映射为二维数组的第一行，所有元素的第二个非数组域对应着二维数组的第二行，以此类推。

结构体数组 $S[N]$ 如图 4 所示，当前区域内所引用的结构体数组的长度为 N，结构体中含有两个非数组域，则预申请的内存空间大小为 $2\times N\times sizeof(\text{double})$。当前结构体数组所映射到的二维数组为 $S[2][N]$，即 2 行×N 列。所引用的结构体数组第一个元素的第一个非数组域（$S[0].a$）的起始地址（S_{start}）对应申请的内存空间的起始地址，即二维数组第 0 行的行起始地址（$s[0]$）。所引用的结构体数组第一个元素的第二个非数组域（$S[0].b$）的起始地址对应的开辟的内存空间的地址为 $s_{start}+N\times sizeof(type)$，也就是二维数组第 1 行的行起始地址（$s[1]$）。所有结构体数组元素的第一个非数组域（$S[0].a, S[1].a, \cdots, S[N].a$）一一映射为二维数组的第一行。所有结构体数组元素的第二个非数组域映射为二维数组的第二行。当前区域内所引用的结构体数组含有多个非数组域时，以此类推，即可得到相应的二维数组。

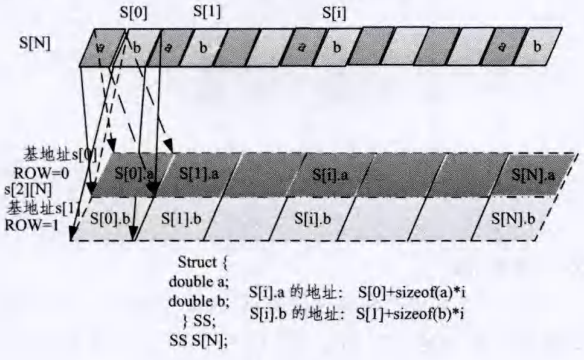

图 4 结构体地址映射

3.4 向量化代码生成

（1）本文的预分析模块沿用了自动向量化编译系统 SW-VEC 中的预分析模块，只是添加了部分功能：

a）合法性分析

在确定向量化候选区域和向量化区域内的所引用的结构体的相关信息收集阶段都添加了相应的合法性分析。

b）依赖分析

结构体拆分和结构体数组映射都没有改变程序的结构，因此不存在优化后的别名问题和依赖关系的变化问题。

c）收益分析

文献[12]对不同情况下进行 SIMD 向量化的向量化收益进行了建模和详细的分析，本文借鉴其不连续 SIMD 向量化收益评估模型并做修改，进行结构体数组 SIMD 向量化收益分析，下文将做详细的介绍。

（2）SIMD 指令生成

向量化代码生成指的是将可以向量化的串行程序经过向量化发掘后变换为目标系统 SIMD 扩展部件可执行的向量化程序，从而提升程序的执行效率。

关于向量化代码生成，文献[11]按照语句操作的类型不同，将语句大致分为 3 类：装载操作、计算操作、存储操作。分类对其进行向量化代码生成。向量化代码生成过程中一个非常重要的问题是：新生成的向量如何维持原程序中标量所具有的依赖关系。本文借助一个数据结构来记录向量中位置与存放变量的映射关系以解决这个问题。以图 4 所示的结构体数组 $S[N]$ 和映射得到的二维数组 $s[2][N]$ 为例：

（1）操作类型为 *load* 的语句即为数组的装载操作，向量化代码生成时转换为相应的向量化装载语句。如 $x=S[i].a$ 转换为相应的 SIMD 向量化装载语句为 $simd\_load(v\_x, \&s[0][i])$，其中 v_x 是向量寄存器。此装载操作的含义是从 $s[0][i]$ 的起始地址起，连续存储一个向量长度的数据到向量寄存器 v_x，假设当前 $i=0$（满足对齐要求），向量长度为 256，数据类型为双浮点精度（double），则 v_x 存储的数据为 $\{s[0][0], s[0][1], s[0][2], s[0][3]\}$。

（2）除操作类型为 *load* 或 *store* 的所有操作统称为计算操作。比如 $y=S[i].a+S[i].b$ 转换为相应的向量化计算语句为 $v\_y=simd\_add(v\_a, v\_b)$。该向量化语句将 v_a 和 v_b 中装载的向量长度的数据对应的槽位数据进行相加，将结果存储在 v_y 向量寄存器中。

（3）操作符类型为 *store* 的语句即为数组的存储操作，它在向量化代码生成时转换为相应的向量存储语句。比如 $S[i].b=x$ 转换为相应的向量化存储指令，即 $simd\_store(v\_x, \&s[1][i])$。该向量化语句将 v_x 中装载的一个向量长度的数据存储到从当前数组 $s[1][i]$ 起始地址算起的一个向量长度的连续数组中。假设 $i=0$（满足对齐要求），向量长度为 256，数组类型为双浮点精度（double），v_x 中存储的数据为 $\{s[0][0], s[0][1], s[0][2], s[0][3]\}$，则该语句一次实现 $s[1][i]=s[0][i], i=0,1,2,3$，即 $S[i].b=S[i].a(i=0,1,2,3)$。

4 SIMD 向量化收益分析

进行 SIMD 收益分析时，当前循环进行向量化时数据内存访问是否连续、是否对齐，是影响程序向量化性能提升的重要因素。连续与对齐是紧密联系在一起的，对内存的访问方式分为连续对齐式、连续不对齐式和不连续式，有些目标平台对于不连续、不对齐的访问提供了特殊操作，可实现向量化，尽管不同的体系结构下特殊操作的实现方式有所不同，但都会引入额外操作开销。

结构体数组 SIMD 向量化的收益分析主要考虑对结构体数组存储地址的连续优化和对齐优化是否有收益，在对结构体数组的存储优化中，结构体拆分生成多个子结构体，增加了新的变量的定义、声明的开销。同时，引用的结构体数组映射到二维数组，需要预申请一个固定大小的连续存储空间，且所有结构体数组元素的每个非数组域一一映射到二维数组中的一个元素，伴随着产生了额外的内存存储空间的申请和地址转换的代价，极大地减少了结构体 SIMD 向量化的非连续访问和非对齐访问。

结合上面影响结构体 SIMD 向量化收益因素的分析，得到 SIMD 代价评估函数如下：

$$c(C_{iters}, M) = C_{iters}(\sum C_{vec\_instr}) + \sum\{C_{iters} \times (C_{vec\_load} + C_{vec\_store})\} + C_{split} + C_{map}$$

其中，C_{iters} 表示向量化后循环的新迭代次数；C_{split} 为结构体拆分的开销；C_{map} 为地址映射的开销；C_{vec_load}、C_{vec_store} 分别表示向量化读、写操作的指令延迟；C_{vec_instr} 为除向量读写指令外的其它 SIMD 指令延迟之和，M 为可 SIMD 向量化的结构体个数。

此外，影响结构体的 SIMD 向量化收益的因素可能还包括预申请内存空间的开销、存储空间扩大后可能导致的 Cache 失效的时间开销等。综合上述可能影响收益的因素，若整体收益为正，则进行结构体数组的 SIMD 向量化。

5 实验与分析

5.1 实验设置

本文使用的自动向量化编译系统是基于开源编译器 Open64 框架实现的 SW-VEC，具有专用的 SIMD 扩展部件的 64 位实验平台。

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，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 申威-1600 上运行。

INTEL 编译器分别采用 11.0 和 13.0 版本，向量化编译选项采用-vec-report3 选项查看其向量化过程中的信息。INTEL 编译器的编译环境为 Linux 操作系统，版本为 Readhat Enterprise 5。实验平台为 Intel(R) Xeon(R) CPU，主频为 2.0GHz，内存为 2GB。

5.2 向量化识别能力测试

选取 gcc-vec 测试集中含有结构体的 58 个测试用例，来测试对结构体 SIMD 向量化的识别能力。对部分测试用例程序的分析如表 1 所列。

表 1 gcc-vec 测试集中关于结构体的测试用例分析

gcc-vec 用例	程序特点
vect-100.c	数组直接赋值给结构体数组域
vect-115.c	数组直接赋值给结构体数组域
vect-31.c	结构体数组引用
vect-32.c	数组直接赋值给结构体数组域
vect-34.c	数组直接赋值给结构体数组域
vect-36.c	数组直接赋值给结构体数组域
vect-68.c	赋值给结构体数组域
vect-70.c	赋值给结构体数组域
vect-89.c	赋值给结构体指针
vect-96.c	赋值给结构体数组域
vect-97.c	赋值给结构体数组域
pr21591.c	结构体指针
pr24059.c	结构体引用
pr31041.c	结构体指针
pr31343.c	结构体引用
pr32230.c	Char 数组类型，结构体指针，结构体引用
pr33369.c	结构体引用，移位
pr33833.c	结构体指针，结构体引用
section-anthor-vect-69.c	结构体数组成员赋值
O3-pr36098.c	结构体指针形参，结构体
O3-vect-pr32243.c	比较复杂的结构体指针引用，结构体指针形参
fast-math-pr35982.c	结构体指针参数、结构体引用、强制类型转换
fast-math-vect-pow-2.c	结构体引用

从 gcc-vec 标准测试集中选取含结构体的 56 个测试用例，测试结果如图 5 所示，结果表明 SW-VEC 对循环内可向量化的结构体的识别能力和向量化能力与 INTEL 的 icc 11.0 编译器基本相当，两者都未能对含结构体指针或结构体引用的循环做出正确的识别和向量化，56 个含结构体的测试用例中 SW-VEC 和 icc 11.0 都分别识别和向量化了 41 个和 40 个，剩余未能识别和向量化的主要原因是循环中含有结构体指针、复杂控制流、非连续或非对齐的数据访问等。SW-VEC-Str 识别并向量化了 46 个测试用例，10 个未识别和向量化的主要原因是复杂的结构体指针、控制流、数据依赖分析等。而 INTEL 的 icc 13.0 对结构体的识别能力和向量化能力要远远高于 icc 11.0 和 SW-VEC。56 个含结构体的测试用例中只有 4 个测试用例未能正确识别和向量化，通过对未识别和向量化的测试用例进行分析，得出其主要原因是程序中含有复杂控制流的循环结构、复杂数据依赖关系分析、结构体指针别名分析等。其中，icc 11.0、icc 13.0 分别表示版本为 11.0、13.0 的 INTEL 的编译器；SW-VEC 表示未添加结构体向量化的编译器；SW-VEC-Str 表示添加结构体向量化的编译器。

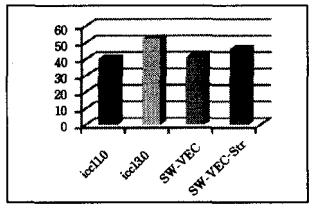

图 5 gcc-vec 中测试用例识别和向量化情况

5.3 性能测试

关于本文所提方法的性能测试，选取了 SPEC CPU2000 标准测试集中的 179.art、188.ammp、191.fma3d 和 SPEC 2006 测试集中的 429.mcf、462.libquantum 作为测试用例。关于 5 个测试用例的功能、核心函数及核心函数运行时间占总运行时间的百分比、可向量化循环特征的分析如表 2 所列。其中，核心函数执行时间占用百分比是编译器的 profiling 信息，编译选项使用-pg，测试输入规模为 ref。

表 2 测试用例分析

测试用例	功能	核心函数	循环特点	执行时间百分比%
179.art	图像处理	match	非数组成员引用、数组成员	83.36
	图像处理	train_match		12.31
188.ammp	计算化学	mm_fv_update_nonbon	非数组成员引用	75.65
191.fma3d	有限元碰撞模拟	solve	非数组成员引用	32.87
		scatter_element_nodal_forces		10.90
		khplq_gradient_operator		10.46
429.mcf	组合优化	primal_bea_mpp	非数组成员引用	51.58
462.libquantum	量子计算	quantum_toffoli	非数组成员引用	60.02

图 6 是应用 INTEL 的 icc 11.0、icc 13.0 和常规的 SW-

VEC、结合结构体映射的 SW-VEC-Str 对 5 个测试用例的测试结果，除 SPEC 2006 中的 462. libquantum 外，常规的 SW-VEC 与 INTEL 的 icc 11.0 对应用程序的向量化效果不相上下，结合本文的方法之后，5 个测试用例的向量化效果在常规 SW-VEC 的基础上都有不同程度的提升。179. art 程序性能提升了约 10%，429. mcf 性能提升了 11%，剩余 3 个测试用例性能提升幅度较小，大约在 8%。

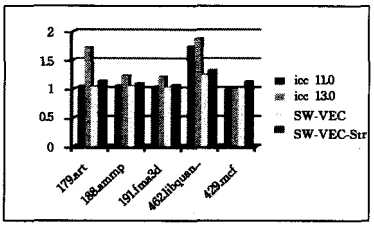

图 6　测试用例的向量化效果

下面主要是对选自 SPEC 2000 测试集的测试用例向量化效果的原因作简要的分析：

179. art 的核心函数的核心循环中主要是对非数组域的引用，并且所有域的数据类型都相同，只需将数组域与非数组域分裂。而且无论是数组域还是非数组域的存储都已经满足地址的连续和对齐要求，只要将结构体数组映射到二维数组即可，不会带来太多额外的开销。程序中存在大量的对结构体数组非数组成员的初始化操作，进行结构体数组的 SIMD 向量化会使程序有较好的性能提升。

188. ammp 的核心函数中虽然存在结构体的非数组成员引用，但由于所占比例较小，进行结构体 SIMD 向量化后程序性能提升不大。

191. fma3d 的核心函数的循环中结构体的非数组成员引用所占比例较大，且结构体的非数组域基本为相同数据类型，进行结构体拆分和结构体映射到二维数组后，使循环中对非数组域的引用访问满足存储地址连续和对齐要求，虽然带来了结构体拆分、预申请空间、结构体映射等额外的开销，但在一定程度上抵消了程序 SIMD 向量化的部分收益。进行结构体 SIMD 向量化后，程序有一定的性能提升。

综合对测试结果以及用例程序的分析，得出影响结构体的 SIMD 向量化对程序性能提升的因素包括以下几个方面：

- 循环中存在数据依赖、不支持的数据类型、结构体指针、函数调用、控制流等影响向量化的因素，导致循环不可向量化；
- 优化过程不在核心函数中或在核心函数中所占比例较小，对程序整体性能提升不明显；
- 循环中结构体太复杂，为满足地址连续化和对齐化要求，带来的额外开销太大，导致结构体 SIMD 向量化收益甚微。

结束语　本文提出了含非数组域的结构体数组的向量化框架，利用基于域访问亲和度与域数据类型相结合的结构体拆分模型进行结构体拆分，将只含数据类型相同非数组域的结构体数组地址映射到二维数组，实现结构体数组非数组域引用的地址连续和对齐。自动向量化编译系统 SW-VEC 虽然日趋完善，已给部分应用程序带来了较好的性能提升，但在诸多方面还存在缺陷，如指针别名分析、控制流、复杂依赖关系分析等。且硬件平台支持也存在不足，比如部分数据类型不支持向量化；由于限制条件太多，具有很大局限性。今后我们将对指针别名分析、控制流、依赖关系分析等方面做深入的研究。

参 考 文 献

[1] Li Yu-xiang, Shi Hui, Chen Li. Vectorization-oriented Local Data Regrouping[J]. Journal of Chinese Computer Systems, 2009, 30(8): 1528-1534(in Chinese)
李玉祥，施慧，陈莉. 面向向量化的局部数据重组[J]. 小型微型计算机系统，2009，30(8)：1528-1534

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

[3] Chilimbi T M, Davidson B, Larus J R. Cache-conscious Structure Definition[C]// Proceedings of PLDI'99. 1999: 13-24

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

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

[6] Fu Xiong, Wang Ru-chuan. Locality-based Data Reorganization Framework[J]. Computer Science, 2009, 36(2): 146-151

[7] Chilimbi T M, Davidson B, Larus J R. Cache conscious structure definition[C]// Proceedings of the ACM SIGPLAN 1999 Conference on Programming Language Design and Implememntation. New York, USA: ACM, 1999: 13-24

[8] Hundt R, Mananrswamy S, Chakrabarti D R. Practical structure layout optimization and advice[C]// Proceedings of the Fourth IEEE/ACM International Symposium on Code Generation and Optimization. Washington DC, USA: IEEE, 2006: 233-244

[9] Curial S, Zhao Peng, Amaral J N, et al. MPADS: Memory-pooling-assisted data splitting[C]// Proceedings of the 7th international sysposium on Memory management. New York, USA: ACM, 2008: 101-110

[10] Yan Jia-nian, He Jiang-zhou, Chen Wen-guang, et al. ASLOP: A field-access affinity based structure data layout optimizer [J]. Science in China Series F: Information Science, 2011, 54(9): 1769-1783

[11] Tanaka H, Ota Y, Matsumoto N, et al. A new compilation technique for SIMD code generation across basic block boundaries [C]// Proceedings of the 2010 Asia and South Pacific Design Automation Conference. IEEE Press, 2010: 101-106

[12] Zhang Yuan-yuan, Zhao Rong-cai, Han Lin. Vectorization Benefit Evaluation Method Based on Polyhedron Representation[J]. Computer Engineering, 2012, 38(7): 266-268(in Chinese)
张媛媛，赵荣彩，韩林. 基于多面体表示的向量化收益评估方法[J]. 计算机工程，2012，38(7)：266-268