一种基于局部性的数据重组框架

付 雄[1,2] 王汝传[1]

(南京邮电大学计算机学院 南京 210003)[1] (南京邮电大学计算机技术研究所 南京 210003)[2]

摘 要 处理器和内存之间速度差距日益增大,使内存访问成为系统主要的性能瓶颈之一,Cache成为现代体系结构中用来解决这个问题的主要技术。利用数据重组优化程序自身的局部性,从而提高Cache性能成为一个值得研究的热点问题。提出了一种基于局部性的数据重组框架,该框架利用一种基于变量局部性特征的变量关系图来量化变量之间的关系,然后寻找变量之间的布局优化,通过数据重组和结构拆分两种常用的数据重组方法来提高Cache性能。针对SPEC CPU2000中的部分测试程序的实验表明,这种数据重组框架能够有效地减少Cache失效次数,提高程序性能。

关键词 复用距离,变量关系图,数据重组,数组重组,结构拆分

Locality-based Data Reorganization Framework

FU Xiong[1,2] WANG Ru-chuan[1]

(School of Computer, Nanjing University of Posts & Telecommunications, Nanjing 210003, China)[1]

(Institute of Computer Technology, Nanjing University of Posts & Telecommunications, Nanjing 210003, China)[2]

Abstract Cache is widely used to decrease the impact of the ever-increasing speed gap between processor and memory. Optimizing program locality by data reorganization to improve cache performance has become a hot area worthy of research. A locality-based data reorganization framework was introduced, which uses variable relation graph based on variables' locality to quantitate relation of variables, then searches an optimized layout for variables, optimizes program cache performances by two data reorganization techniques: array regrouping and structure splitting. Experiments for some benchmarks from SPEC CPU2000 show this framework is effective in decreasing cache miss and improving program performance.

Keywords Reuse distance, Variable relation graph, Data reorganization, Array regrouping, Structure splitting

1 引言

自上世纪70年代以来,计算机处理器速度的增长基本符合摩尔定律,每年增长约60%,但是内存速度每年增长不到10%,两者之间的速度差距越来越大,已经成为整个计算机系统性能的最主要瓶颈之一[1,2]。现代计算机体系结构中广泛采用Cache来缓解这个问题,使得Cache已经成为整个处理器性能、能耗和价格的决定性因素之一[1]。Cache功能的充分利用很大程度上取决于程序自身的局部性,特别是程序数据的局部性[2]。优化程序数据的布局,改善程序数据的局部性,从而提高数据Cache的性能,已经成为提高程序性能的一种重要方法[3-11]。

过去在优化程序数据布局,从而提高数据Cache的性能方面的工作主要可以分为两类:一类是代码变换(Code Transformation),如循环融合(Loop Fusion),循环分块(Loop Tiling)和循环交换(Loop Interchange)等[2,3],这类工作主要通过改变程序指令的执行顺序,从而优化程序访问数据的局部性,包括数据的时间局部性(Temporal Locality)和空间局部性(Spatial Locality),其难点是需要分析指令之间的依赖关系,且变换复杂,难以保持变换前后的执行语义不变;另一类是数据重组(Data Reorganization)变换,如数组重组(Array Regrouping)、结构拆分(Structure Splitting)等[5-10],这类工作只改变数据之间的关系,只优化程序数据的空间局部性,因此所需的变换也相对较简单,但是其难点在于分析数据之间的Cache行为关系。

近年来,随着复用距离(Reuse Distance)成为度量程序Cache行为的标准之一[2,8],出现了一些利用变量复用距离表示局部性,基于其特征来分析变量之间的关系,从而进行数据重组的方法[8,10]。本文根据这种思想提出了一个基于局部性的数据重组框架,该框架基于变量复用距离分布特征表现出来的局部性,设计了一种量化变量之间关系的变量关系图(VRG, Variable Relation Graph),在此基础上寻找变量之间的优化布局,通过数组重组和结构拆分两种数据重组方法来优化程序数据的布局,从而提高数据Cache性能。针对179.art,183.equake和188.ammp等SPEC CPU2000的部分标准测试程序的实验,获得了1.5601的平均加速比,这表明我们

到稿日期:2008-03-04 本课题得到国家自然科学基金(60573141和60773041),江苏省高技术研究计划(BG2006001),国家高科技863项目(2006AA01Z201,2007AA01Z404,2007AA01Z478),江苏省计算机信息处理技术重点实验室基金(kjs06006)资助。

付 雄(1979—),男,博士,讲师,研究领域为高性能计算、程序优化,网格计算、信息安全、计算机软件等;王汝传(1943—),男,教授,博士生导师,主要研究方向计算机软件、计算机网络和网格、信息安全、无线传感器网络、移动代理和虚拟现实技术等。

的数据重组框架能够有效地优化数据 Cache 性能，提高整个程序的性能。

本文组织如下：第 2 节介绍优化框架；第 3 节介绍基于变量关系图的变量关系分析；第 4 节介绍数组重组和结构拆分两种数据重组方法及变换算法主要过程；第 5 节是实验及结果分析；最后两节分别是相关工作和结束语及未来工作。

2 优化框架

图 1 给出了我们实现的基于局部性的数据重组框架的整体结构，整个框架主要分为两个模块。

一是变量关系分析模块。C 源代码经过源代码级插桩，得到插桩后的 C 源代码，经过普通的 C 编译器编译后运行。通过插桩加入的代码得到变量访问信息的序列，然后对序列分析变量的复用距离，基于复用距离分布的特征建立变量关系图，在此基础上寻找变量的优化布局，得到变量重组方案。

另一是数据重组模块。依据变量关系分析得到的数据重组方案，在对 C 源代码进行源代码到源代码的转换过程中进行数据重组，目前主要的实现是数组重组和结构拆分，得到优化后的 C 源代码。

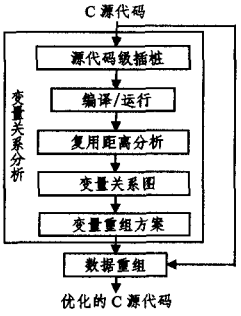


图 1 优化框架

3 变量关系分析

3.1 源代码级插桩

源代码级插桩(Source-Level Instrumentation)主要用来获取程序访问数据的序列和数据与变量的对应关系，以便进行变量之间的关系分析。源代码级插桩在对 C 源代码进行源代码到源代码的转换过程中加入记录访问变量信息的代码，这些代码在运行时将访问变量信息直接写入文件中，供后面进行变量关系分析。在重组框架的实现中，该源代码级插桩过程在基于 SUIF 编译框架[12]中实现。

实际上，访问变量的信息并不能完全代表程序访问数据的全部信息。这两者之间存在一些不同之处，主要可以分为两种：(1)由于寄存器文件的缓存作用，访问某些标量并不形成访存；(2)某些变量的一次访问可能会形成多次访存(如结构之间赋值)。由于框架中的变量关系分析的主要目的是分析变量之间的一种相对关系，寻找经常在一起访问的变量；并且(1)中出现的情况对所有其他变量的影响是相同的，(2)中情况出现的概率比较小，所以这些不同之处可以忽略，我们通过收集程序运行时变量的访问序列来表示程序访问数据的序列。

3.2 复用距离分析

3.2.1 复用距离概念

在 20 世纪 70 年代，Mattson 等人利用栈来研究虚拟内存中页面之间的复用关系，并基于这种方法评估页面置换算法的效率，提出了栈距离(Stack Distance)的概念[2]。为了用体系结构无关的一些特征来描述程序自身的局部性，Ding 等人在栈距离的基础上提出了复用距离的概念[2]。下面我们将给出复用距离的概念。

定义 1(数据元素，Data Element) 是指程序运行中被访问数据的标识，可以是内存地址，也可以是内存区域。通常用符号 a,b,c 来表示数据元素。

定义 2(复用距离，Reuse Distance) 指串行程序运行中前后两次访问同一数据元素之间所访问的不同数据元素的数目。

假设一个数据元素的访问序列为 *abcdeaedb*，则其相应的复用距离序列为$\infty\infty\infty\infty\infty\infty\infty$4124，其中$\infty$表示首次访问该数据元素，不存在复用关系。虽然两次对数据元素 b 访问之间的数据元素个数为 6，但是不同的个数为 4，所以第二次访问的复用距离为 4。

程序的复用距离通常指该程序所有访存的复用距离。由于单次访问的复用距离没有什么意义，所以一般所说的复用距离是指整个程序的复用距离。一般而言，数据的访问次数远远多于对指令的访问次数，通常研究复用距离只研究数据的复用距离[2,10]。复用距离一般用柱状图来表示，横坐标为表示复用距离大小的区间，纵坐标为复用距离在该区间的访问次数[2]。

复用距离能够用来表示程序的局部性。若局部性较好，则绝大部分访问的复用距离的值都较小；反之较大。当数据元素为 Cache 块所代表的内存区域时，两次对同一 Cache 块访问之间的复用距离 d 就可以用来判断第二次访问是否发生 Cache 失效(假设 Cache 为全关联，采用 LRU 替换策略)：如果 $d<n$，其中 n 为 Cache 块数目，则表示两次访问之间所访问其他 Cache 块数目小于总的 Cache 块数目，此时原来被访问的 Cache 块还没有替换出去，所以不会出现 Cache 失效；否则，即 $d\geqslant n$ 时，则会发生 Cache 失效[2]。

3.2.2 受限的复用距离

在分析复用距离时，我们发现了两个重要的特征：(1)除了少数访问，绝大部分访问的复用距离的值都比较小；(2)利用栈计算复用距离时，位于栈底的元素极少发生再次复用。这样给我们分析复用距离很大的启示：如果只分析复用距离在一定区间的访存，即对每次访存只搜索一定区间访存记录，判断这个区间是否存在复用关系，这将在很大程度上加快分析的速度；同时只需保存一定访存记录，所需空间也会减少。我们将这种复用距离分析方法称为受限的复用距离分析。

在具体的实现中，我们将可能需要的最大 Cache 大小作为搜索区间的限制长度，用 Cache 块数目来表示。下面给出我们设计的受限的复用距离分析算法。

算法 1 受限的复用距离分析

输入：访问数据序列 χ；

输出：复用距离序列 *Dist*；

对数据序列中的每个数据 $\chi_\tau(0\leqslant\tau<N)$重复下面的过程：

(1)查找。在栈上自顶向底搜索最近对 χ_τ 的访问，搜索的最大距离不超过最大 Cache 大小；

(2)计算。计算当前数据 χ_τ 的复用距离 $Dist_\tau$，如果在

(1)中能够找到对 χ_τ 的访问记录，则 $Dist_\tau$ 为栈中 χ_τ 到栈顶的元素个数。否则，$Dist_\tau$ 为∞。

(3)更新。如果(1)中查找时存在 χ_τ 的记录，则将其纪录从栈中删除。将当前的数据 χ_τ 压入栈顶，如果栈的大小超出最大 Cache 大小，删除栈底元素。

受限的复用距离分析有两个优点：一是确保用于存储数据访问记录的栈的大小不超出最大 Cache 大小；二是查找搜索区间的最大长度为最大 Cache 大小。这样既保证算法需要的空间不至于过大，又能加快查找速度。虽然这种复用距离分析方法丢失了少量访问数据的复用距离的精度，但是对总的分析结果影响很小，几乎可以忽略。

3.3 变量关系图

变量的复用距离是指对该变量对应内存区域的所有访问的复用距离，通常能够用来表示该变量的时间局部性特征。如果程序中两个变量经常在一起访问，则两者的复用距离分布会比较相似；反之，则两者的复用距离分布差异可能会较大。这样，我们可以利用变量之间复用距离分布的特征来分析变量之间空间局部性，寻找经常在一起访问的变量。我们设计的变量关系图(Variable Relation Graph)就是基于这种思想，下面将以测试程序 183. equake 中的部分变量为例介绍变量关系图。

变量的复用距离通常也用柱状图来表示，图 2(a)给出了 183. equake 之中部分变量的复用距离分布图，其中横坐标为 i 表示的区间为$[2^{i-1},2^i)$(复用距离为 0 表示连续访问同一变量，不能用来分析变量之间的空间局部性，所以图中没有给出；横坐标为∞表示复用距离为∞)，纵坐标表示该变量的复用距离的值在该区间的访问次数。

从图中可以看出，变量 $M23$，$V23$ 和 $C23$ 复用距离分布比较接近，表示经常在一起访问的概率较大。变量 C 和 M 除了横坐标为 2 的区间的差异较大外，其他的都非常接近，这表示除了部分访问外，其他访问经常在一起的概率很大。变量关系图中通过一些量化的性质来表示两个变量之间的关系，下面先给出一些相关的定义。

变量 i 和变量 j 的复用距离分布在横坐标为 k 的区间的相同部分 S_k 可以通过下面的式(1)计算得到：

$$S_k=\mathrm{Min}(N_k(i),N_k(j)) \tag{1}$$

其中 $N_k(i)$表示变量 i 在横坐标为 k 的区间的访问次数。变量 i 和变量 j 的复用距离不为 0 的访问次数 N_i 和 N_j 分别可以通过下面的式(2)和式(3)计算得到：

$$N_i=\sum_{k\in G}N_k(i) \tag{2}$$

$$N_j=\sum_{k\in G}N_k(j) \tag{3}$$

式(2)和式(3)中的 G 为变量复用距离分布图中表示复用距离大小的区间的集合。

变量 i 和变量 j 之间复用距离分布差异 R 可以通过式(4)来计算：

$$R=\sum_{k\in G}I_k\times\left(\frac{N_k(i)-S_k}{N_i}+\frac{N_k(j)-S_k}{N_j}\right) \tag{4}$$

其中 I_k 为复用距离的值对复用距离分布差异 R 的影响因子，这里的 I_k 用复用距离分布图中横坐标的值来表示。需要注意的是：在分布图中，横坐标有一个值为∞，此时 I_k 的值无法决定，由于 I_k 的意义在于体现复用距离值较大时的差异对 R 的值的影响也较大，因此我们将复用距离为∞时的 I_k 的值定义为 17，大于其他所有的值，即复用距离为∞时对 R 的值的影响最大。

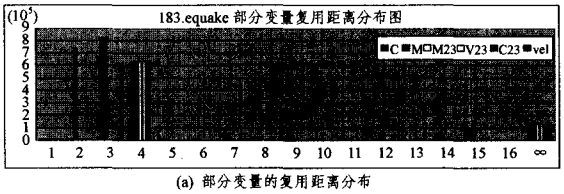
变量 i 和变量 j 之间的访问次数比 D 可以通过式(5)来计算：

$$D=\begin{cases}\dfrac{N_i}{N_j}, & N_i\geqslant N_j\\[2ex] \dfrac{N_j}{N_i}, & N_i<N_j\end{cases} \tag{5}$$

根据前面对变量复用距离分布特征的一些量化，我们就可以给出变量关系图的定义。

定义 3(变量关系图，VRG，Variable Relation Graph) 表示变量之间的关系，该图为一个无向加权图 $G=(V,E)$，其中顶点集合 V 表示程序中的变量，边集 E 中的边 e 表示该边连接的两个顶点所表示的变量之间的关系，其权值由两个变量之间的复用距离分布差异 R 和访问次数比 D 组成的序对〈R，D〉决定。

图 2(b)给出了图 2(a)中对应变量的变量关系图中的序对〈R,D〉值。可以看出，变量 $M23$，$V23$ 和 $C23$ 复用距离分布比较接近，则相互之间对应的 R 值也比较小，D 值比较接近于 1。



(a) 部分变量的复用距离分布

	C	M	M23	V23	C23	vel
C		0.2760, 0.7240	2.5504, 0.4060	2.5504, 0.4060	2.5504, 0.4060	2.9293, 2.9290
M	0.2760, 0.7240		2.1234, 0.2940	2.1234, 0.2940	2.1234, 0.2940	2.3968, 0.2836
M23	2.5504, 0.4060	2.1234, 0.2940		0.0004, 0.9999	0.0004, 0.9999	4.1830, 0.9646
V23	2.5504, 0.4060	2.1234, 0.2940	0.0004, 0.9999		0.0004, 0.9999	4.1830, 0.9646
C23	2.5504, 0.4060	2.1234, 0.2940	0.0004, 0.9999	0.0004, 0.9999		4.1830, 0.9646
vel	2.9293, 2.9290	2.3968, 0.2836	4.1830, 0.9646	4.1830, 0.9646	4.1830, 0.9646	

(b) 183.equake 中部分变量的变量关系图中的 R 和 D 值

图 2 变量关系量化

3.4 变量重组方案

根据变量关系图的定义可知，如果两个变量经常在一起访问，则在变量关系图中两者之间的 R 值会比较小。我们可以将其分为一组，通过数组重组或结构拆分充分利用两者之间的空间局部性来提高 Cache 性能。在实际的变量分组中，需要确保分在同一组内的变量之间存在的访问次数比较接近，这样确保数据重组的效果，我们可以充分利用变量之间的 D 值来确保。

在进行数据重组时，无论是数组重组还是结构拆分，都需要得到变量的分组关系。特别是结构拆分，还需要得到属于同组的变量的线性序。这样，我们就需要从变量关系图中寻找符合要求的变量的线性序。可以通过在变量关系图上进行一种改进的深度优先搜索实现此目的。下面给出该算法的描述：

算法 2 变量重组算法

输入：变量关系图 G；

R 值的最大标准 R_std；

D 值的最小标准 D_std。

1)如果变量关系图 G 中的变量集 $V(G)=\emptyset$，则完成算法退出。否则，继续执行；

2)从变量关系图 G 的变量集中取变量 s，并将其从变量集中删除 $V(G)=V(G)-\{s\}$，建立新的变量线性表 T，将 s 加入到 T 中；

3)从变量关系图 G 的变量集中获取变量 g，从变量线性表 T 的头部或尾部获取变量 t，使 g 和 t 之间的 R 值最小。如果 R 值小于 R_std，并且 D 大于 D_std，则执行 4)。否则执行 5)；

4)将变量 g 从变量关系图 G 的变量集中删除 $V(G)=V(G)-\{g\}$。如果 t 位于队列 T 头部，则将 g 插入 T 的头部之前；否则将 g 追加到队列 T 的队尾之后，执行 3)；

5)将变量线性表 T 中的变量按照线性序作为一组变量分组保存起来，得到需要进行变换的一组变量。执行 1)。

图 3 给出了图 2 中变量按照我们的重组算法进行分析后得到的重组方案。由于这些变量都是数组，所以进行的变换是数组重组，其中 C 和 M；$M23$，$C23$ 和 $V23$ 分别进行重组。

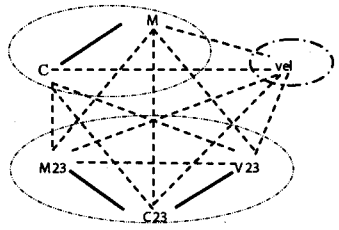


图 3　183. equake 中部分变量重组结果

4　数据重组

通过前面变量关系分析得到变量重组方案，就可以对这些变量进行数据重组，如数组重组、结构拆分等。使这些变量在内存中的位置临近，这样充分利用 Cache 提高整个程序的性能[8,10]。我们的数据重组框架为基于 SUIF 编译框架[12]实现的一个源代码到源代码的转换器，目前主要实现了数组重组和结构拆分两种数据重组方法。下面我们将分别介绍这两种重组方法和实现重组的变换算法的主要过程。

4.1　数组重组

在 C 程序中，数组的各个元素或者结构的各个属性域都被分配在相邻的内存区域。数组重组就是利用这个原理，将经常在一起访问的不同数组的元素重组为一个结构，这样访问多个数组的元素就变成了访问同一结构多个属性域，从而提高访问数据空间局部性，提高 Cache 性能。图 4 给出了我们数据重组框架中数组重组的示例，这是针对一种动态数组的重组示例。由于普通数组的重组更加简单，这里就不再给出相应的示例。

```
  //definition and initialization
double * * M23, * * C23, * * V23;
M23 = (double * * ) malloc(ARCHnodes * sizeof(double * ));
C23 = (double * * ) malloc(ARCHnodes * sizeof(double * ));
V23 = (double * * ) malloc(ARCHnodes * sizeof(double * ));
for (i = 0; i < ARCHnodes; i++) {
  M23[i] = (double * ) malloc(3 * sizeof(double));
  C23[i] = (double * ) malloc(3 * sizeof(double));
  V23[i] = (double * ) malloc(3 * sizeof(double));
}
  //reference
  V23[i][j] = 0.0;
```

(a) 重组前的定义和引用

```
//definition and initialization
struct MCV{
  double M23;
  double C23;
  double V23;
};
struct MCV * pmcv;
pmcv =
(struct MCV * )malloc(ARCHnodes * 3 * sizeof(struct MCV));
//reference
( * (pmcv+i * 3+j)). V23 = 0.0;
```

(b) 重组后的定义和引用

图 4　数组重组示例

4.2　结构拆分

结构拆分可以看作是数组重组的一种逆向变换。如果一个由结构组成的数组中，经常在一起访问的只有结构中少数属性域，将这些属性域分裂出来，重新构成一个结构，则组成数组后，能够较好地提高数据的空间局部性，从而提高 Cache 性能。图 5 给出了我们数据重组中结构拆分的示例，一个较大的结构根据属性域是否经常在一起访问被拆分成多个子结构。

```
Typedef struct {
  double * I;
  double W;
  double X;
  double V;
  double U;
  double P;
  double Q;
  double R;
} f1_neuron;
```

(a)结构拆分前

```
typedef struct {
  double W;
  double X;
} f1_neuronWX;
typedef struct {
  double U;
  double P;
} f1_neuronUP;
//double f1_neuronI
...
```

(b)结构拆分后

图 5　结构拆分示例

4.3　变换算法主要过程

我们的数据重组框架为一个基于 SUIF 编译框架[12]的源代码到源代码的转换器，在进行源代码的转换中进行数据重组，包括数组重组和结构拆分。整个变换算法通过在 SUIF 的中间格式进行变换，其变换过程中主要进行的变换包含下面几步：

(1)变换变量定义。解析 SUIF 中间格式表示的程序时，如果遇到需要重组的变量定义，如数组变量或结构变量的定义，需要更改其定义方式。如图 3 所示的数组重组示例中，需要将数组的变量 $M23$，$C23$ 和 $V23$ 的定义从下面的形式：

```
double * * M23, * * C23, * * V23;
```

变换为另一种的形式：

```
struct MCV{
  double M23;
  double C23;
```

```
    double V23;
};
struct MCV * pmcv;
```

(2) 变换变量初始化。被重组的变量的定义被改变后，这些变量相应的初始化过程也应该发生改变。需要注意的是，有些变量的定义和初始化过程是在一起的，需要同时进行两者的变换。

(3)更新变量的访问形式。变量被重组后，其访问方式也发生了改变。如在数组重组中，对数组元素的访问变为对结构属性域的访问；在结构拆分中，对拆分前结构属性域的访问变为了拆分后该属性域所属结构相应的属性域的访问。在对动态数组进行重组时，对重组后结构的访问变为基于指针的运算，这是其特殊之处。

此外，在进行变换的过程中还有一些特殊的情况需要处理，常见的如基于别名分析[13]寻找被变换变量的别名，对这些别名也要进行同样的变换；嵌套结构被拆分时，如果不是同一结构的循环嵌套，则只简单考虑最顶层的结构。由于我们数据重组框架只是一个初期的原型系统，所以对这些问题的处理都采用较为简单的方法，这里不再详叙。

5 实验结果及分析

在我们的数据重组框架中，源代码级插桩和数据重组框架都基于 SUIF 编译框架[12]实现。实验测试环境为 Red Hat Linux9.0 操作系统，CPU 为 Intel Celeron(R)2.0G(Cache 块大小为 64byte；L1 级数据 Cache 为 8k，4 路组关联；L2 级 Cache 为 128k，2 路组关联)。测试集由 SPEC CPU2000 中的 179.art，183.equake 和 188.ammp 组成，输入集分别为 test，train 和 ref，编译器为 GCC3.2，O3 优化选项。在数据重组框架中，我们利用 test 输入集进行变量关系分析，其他输入只用于测试数据重组前后的性能变化。

表 1 给出了数组重组、结构拆分和同时进行这两种优化时优化结果，包括 3 个测试程序优化前后的访问数据的次数(仅给 test 输入集的访问数据的次数)及各输入集时的加速比，其中访问数据的次数基于程序分析工具 Pin[14]收集。可以看出，3 个测试程序 10 组输入集在 3 种优化下性能都有了一定的提高，平均值分别为 1.0374，1.4778 和 1.5601。另外，除了 179.art 在结构拆分后访问数据次数增加外，其他的访问数据次数都有了一定减少。同时通过实验，我们发现在 O0 级编译 179.art 时，优化后访问数据次数减少，但是 O3 级编译 179.art 时，优化后访问数据次数增加，可见这是由于 GCC3.2 进行的优化导致了访问数据次数的增加。

为了验证我们数据重组优化框架对数据局部性的影响，我们比较了 test 输入集时 179.art 各种优化前后的数据 Cache 失效次数分布变化情况。我们利用程序分析工具 Pin[14]对优化前后的 179.art 的可执行程序进行插桩，收集程序运行时访问数据的地址序列，然后通过修改 SimpleScalar3.0 中的 Cheetah[15]来分析在 4 路组关联(由于实验平台的 CPU 的一级为 4 路组关联)下的 Cache 失效次数分布情况。图 6 给出了 179.art 优化前和 3 种优化后的 Cache 失效次数分布变化情况，相应的 Cache 块大小为 64byte，最大 Cache 大小为 4M，最小 Cache 大小为 1k。从图中我们可以看出：经过数组重组优化，其 Cache 失效次数有了一定的减少，但减小程度不如结构拆分优化和同时进行两种优化，这个优化后的程序加速比是对应的。

表 1 优化结果

Benchmark	Input	Memory Access	Array Regrouping		Structure Splitting		AR + SS	
		Standard	Memory Access	Speedup	Memory Access	Speedup	Memory Access	Speedup
179.art	test	1,392,146,709	1,389,023,415	1.0569	1,621,753,412	2.8148	1,604,623,842	2.9829
	train			1.0661		2.3431		2.6956
	ref1			1.0183		1.7669		1.7977
	ref2			1.0146		1.7252		1.7896
183.equake	test	542,890,080	539,682,079	1.0579	541,623,712	1.0214	539,6912,512	1.0752
	train			1.0737		1.0183		1.0865
	ref			1.0609		1.0242		1.0821
188.ammp	test	2,733,912,121	2,733,521,412	1.0083	2,731,324,935	1.0184	2,730,423,832	1.0295
	train			1.0109		1.0237		1.0316
	ref			1.0064		1.0218		1.0304
Average				1.0374		1.4778		1.5601

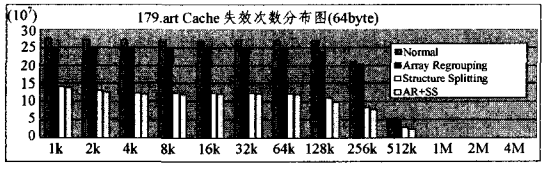


图 6 优化前后数据 Cache 失效次数比较

6 相关工作比较

T.Chilimbi 等人在文献[6]中建立了一种 Cache 敏感的数据结构定义方法，利用结构拆分和结构属性域位置调整(Structure Field Reordering)两种数据重组方法来提高 Cache 性能，其数据重组利用一个名叫属性域亲密图(Field Affinity Graph)模型来度量结构属性域之间的关系，然后根据度量结果来产生一个新的结构布局。文中的属性域亲密图是一个加权图，节点表示结构的属性域，连接节点的边利用特定时间内两节点相继访问的频率作为权值。

Y.Zhong 等人在文献[8]中提出了一种数组重组和结构拆分方法，该方法利用一种基于复用距离的访问亲密性(Reference Affinity)模型来度量变量之间的关系。文中的访问亲密性模型直接利用访问时变量之间的最大复用距离来描述变量之间关系。

Mostafa Hago 等人在文献[9]中提出了一种利用结构拆分来优化 Cache 性能的方法，该方法利用一种紧密接近图(Close Proximity Graph)模型来度量结构属性域之间的关系，并基于这种分析结果进行了结构拆分和属性域位置调整。文

中紧密接近图利用结构中属性域作为节点，连接节点的边利用两节点相继访问之间的“基本块”数目作为权值。

文献[10]介绍了我们在前期的工作中提出的一种简单的变量关系模型，用于支持数组重组这种数据重组方法。本文是在这方面的进一步工作，不同之处主要体现在下面的几个方面：(1)变量关系图相比于变量关系模型更加精确，增加了复用距离的值对差异程度的影响因子，引入了访问次数比等；(2)生成变量重组方案的更优算法，使之能更精确地对变量进行分组，并且可以用于支持更多的数据重组(如结构拆分、结构属性域位置调整等)；(3)在数据重组框架上实现了结构拆分，并进行了更多的实验。

结束语 本文介绍了一种基于局部性的数据重组框架，该框架中基于表示变量时间局部性的复用距离的分布特征，设计了一种变量关系图来量化变量之间的关系，在此基础上寻找变量的优化布局，利用数组重组和结构拆分两种数据重组方法来优化程序的 Cache 性能，从而提高整个程序的性能。针对来自 SPEC CPU2000 的部分测试程序的实验表明，这种数据重组框架能够有效地改善程序的数据 Cache 性能，从而提高整个程序的性能。

我们未来的工作主要包含两部分：一是在此基础上实现更多的数据重组方法，如结构属性域位置调整、全局数据和堆数据的交换等；二是将数据重组方法用于包括 GCC 在内的产品编译器，提高编译器的优化效率。

参 考 文 献

[1] Patterson D, Anderson T, Cardwell N, et al. A Case for Intelligent RAM. IEEE Micro, 1997: 34-44

[2] Beyls K. Software Methods to Improve Data Locality and Cache Behavior. PhD thesis. Ghent University, 2004

[3] McKinley K S, Carr S, Tseng C W. Improving Data Locality with Loop Transformations. ACM Transactions on Programming Languages and Systems, 1996, 18(4): 424-453

[4] Wu Youfeng. Efficient Discovery of Regular Stride Patterns in Irregular Programs and Its Use in Compiler Prefetching // Proceedings of PLDI'02. June 2002: 210-221

[5] Calder B, Krintz C, John S, et al. Cache-conscious data placement // Proceedings of ASPLOS'98. San Jose, October, 1998: 139-149

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

[7] Berg E, Hagersten E. StatCache: A Probabilistic Approach to Efficient and Accurate Data Locality Analysis // Proceedings of ISPASS'04. March 2004: 20-27

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

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

[10] Fu Xiong, Zhang Yu, Chen Yiyun. Data-layout Optimization Using Reuse Distance Distribution // The Proceedings of International Workshop on Embedded Software Optimization (ESO'06). Vol. 4097 of Lecture Notes in Computer Science. Seoul, Korea, August 2006: 858-867

[11] McIntosh N, Mannarswamy S, Hundt R. Whole-Program Optimization of Global Variable Layout // Proceedings of PACT'06. Washington, DC, September 2006

[12] Wilson B P, et al. SUIF: A Parallelizing and Optimizing Research Compiler. ACM SIGPLAN Notices, 1994, 29(12): 31-37

[13] Steensgaard B. Points-to Analysis in Almost Linear Time // Proceedings of POPL'96. St. Petersburg, Jan. 1996

[14] Luk Chi-Keung, Cohn R, et al. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation // Proceedings of PLDI'05. Chicago, Illinois, USA, June 2005: 190-200

[15] Sugumar R A, Abraham S G. Efficient simulation of multiple cache configurations using binomial trees. Technical Report CSE-TR-111-91. University of Michigan, 1991

(上接第 131 页)

参 考 文 献

[1] Burrows M, Abadi M, Needham M. A logic of authentication [J]. ACM Transactions on Computer Systems, 1990, 8(1): 18-36

[2] Kailar R. Accountability in electronic commerce protocols[J]. IEEE Transactions on Software Engineering, 1996, 22(5): 313-328

[3] Abadi M, Gordon D. A calculus for cryptographic protocols: The spi calculus[C] // Proceedings of the Fourth ACM Conference on Computer and Communications Security. 1997

[4] Abadi M, Gordon D. Reasoning about cryptographic protocols in the spi calculus[C]. Technical Report 414. University of Cambridge Computer Laboratory, 1997

[5] Lowe G. Breaking and fixing the Needham-Schroeder public-key protocol using FDR[C] // Tools and Algorithms for the Construction and Analysis of Systems, volume 1055 of Lecture Notes in Computer Science. Springer-Verlag, 1996: 147-166

[6] Fabrega F, Herzog J, Guttman J. Strand space: why is a security protocol correct [C] // Proceedings of the 1998 IEEE Symposium on Security and Privacy. California, USA: IEEE Computer Society Press, 1998: 160-171

[7] Fabrega F, Herzog J, Guttman J. Honest ideals on strand space [C] // Proceedings of the IEEE Computer Security Foundations Workshop XI. California, USA: IEEE Computer Society Press, 1998: 66-77

[8] 刘璟，祝世雄，周明天. Yahalom 协议的串空间模型及分析[J]. 小型微型计算机系统，2006，05：788-792

[9] Furqan Z, Muhammad S, Guha R. Formal Verification of 802.11i using Strand Space Formalism [C] // Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies. California, USA: IEEE Computer Society Press, 2006

[10] Miloslavskaya M, Tolstoy A, Ushakov D. Protocol of Secure Mutual Authentication [C] // Proceedings of the 2007 IEEE Workshop on Information Assurance. IEEE SMC, 2007: 43-48

[11] Li Yongjian, Pang Jun. Generalized Unsolicited Tests for Authentication Protocol Analysis [C] // Proceedings of the Seventh International Conference on Parallel and Distributed Computing, Applications and Technologies. California, USA: IEEE Computer Society Press, 2006: 509-514