软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(4):925−939 [doi: 10.13328/j.cnki.jos.005029]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

向量并行度指导的循环 SIMD 向量化方法*

高 伟[1], 韩 林[1], 赵荣彩[1], 徐金龙[1], 陈超然[2]

[1](数学工程与先进计算国家重点实验室(解放军信息工程大学),河南 郑州 450000)

[2](防空兵指挥学院,河南 郑州 450000)

通讯作者: 高伟, E-mail: yongwu22@126.com

摘 要: SIMD 扩展部件是集成到通用处理器中的加速部件,旨在发掘多媒体和科学计算等领域程序的数据级并行.当前,两种基本的向量发掘方法分别是发掘迭代间并行的 Loop-based 方法和发掘迭代内并行的 SLP 方法.Loop-aware 方法是对 SLP 方法的改进,其思想是:首先,通过循环展开将迭代间并行转换为迭代内并行,使循环体内的同构语句条数足够多;再利用 SLP 方法进行向量发掘.但当循环展开不合法或者并行度低于向量化因子时,Loop-aware 方法无法实现程序向量并行性的发掘.因此提出了向量并行度指导的循环向量化方法,依据迭代间并行度、迭代内并行度和向量化因子构建循环向量化方法选择方案,同时提出了不充分向量化方法发掘并行度低于向量化因子的循环向量并行性,最后,依据向量并行度对生成的向量循环进行展开.经过标准测试集测试,向量并行度指导的循环 SIMD 向量化方法比 Loop-aware 方法的识别率提升了 107.5%,性能提升了 12.1%.

关键词: SIMD 扩展部件;向量并行度;Loop-aware;循环展开

中图法分类号: TP314

中文引用格式: 高伟,韩林,赵荣彩,徐金龙,陈超然.向量并行度指导的循环 SIMD 向量化方法.软件学报,2017,28(4):925−939. http://www.jos.org.cn/1000-9825/5029.htm

英文引用格式: Gao W, Han L, Zhao RC, Xu JL, Chen CR. Loop vectorization method guided by SIMD parallelism. Ruan Jian Xue Bao/Journal of Software, 2017,28(4):925−939 (in Chinese). http://www.jos.org.cn/1000-9825/5029.htm

Loop Vectorization Method Guided by SIMD Parallelism

GAO Wei[1], HAN Lin[1], ZHAO Rong-Cai[1], XU Jin-Long[1], CHEN Chao-Ran[2]

[1](State Key Laboratory of Mathematical Engineering and Advanced Computing (PLA Information Engineering University), Zhengzhou 450000, China)

[2](PLA Air Defense Forces Command College, Zhengzhou 450000, China)

Abstract: SIMD extension is an acceleration component integrated into the general processor, aiming at exploiting data level parallelism in multimedia and scientific computation programs. Two of the mainstream vectorization methods are loop-based method oriented to inter-iteration and SLP method oriented to intra-iteration. Derived from SLP, loop-aware method transforms inter-iteration to intra-iteration through loop unrolling, so as to obtain enough isomorphic statements and then uses SLP to explore vectorization. However, when loop unrolling is illegal or SIMD parallelism is lower than the vector factor, loop-aware method cannot exploit SIMD parallelism of programs. To address this drawback, a vectorization method guided by SIMD parallelism for loops is proposed. Alternative scheme for loop vectorization is constructed in view of inter-iteration parallelism, intra-iteration parallelism and vector factor. Simultaneously, insufficient vectorization is proposed to vectorize loops whose parallelism is lower than the vector factor. Lastly, vectorized loop is unrolled according to SIMD parallelism. Test results by benchmarks show that vectorization method guided by SIMD parallelism outperforms loop-aware method by 107.5%. Moreover, the performance is improved by 12.1% compared with loop-aware method.

* 基金项目: “核高基”国家科技重大专项(2009ZX01036)
Foundation item: CHB National Major Science and Technology Project Foundation of China under Grant (2009ZX01036)
收稿时间: 2015-04-12; 修改时间: 2015-07-31; 采用时间: 2015-12-29; jos 在线出版时间: 2016-03-16
CNKI 网络优先出版: 2016-03-17 09:57:12, http://www.cnki.net/kcms/detail/11.2560.TP.20160317.0957.001.html

Key words: SIMD extension; degree of SIMD parallelism; Loop-aware; loop unrolling

伴随着多媒体产业的迅猛发展,20 世纪 90 年代中期,各大厂商在处理器中集成了一套专用的多媒体扩展指令集.该指令集采用单指令多数据(single instruction multiple data,简称 SIMD)扩展技术,可同时对多个数据进行相同的操作,称之为 SIMD 扩展部件.1996 年,Intel 在其奔腾处理器上集成了 SIMD 扩展部件 MMX,后来又相继推出了 SSE,AVX,IMCI 和 AVX-512.其他 SIMD 扩展部件还包括摩托罗拉 PowerPC 处理器的 AltiVec、Sun 公司 SPARC 处理器中的 VIS、HP 公司 PA-RISC 处理器中的 MAX、DEC 公司 Alpha 处理器中的 MVI-2、MIPS 公司 V 处理器中的 MDMX[1]等.SIMD 扩展部件最初仅用于多媒体领域和数字信号处理器中,后来,研究人员将 SIMD 扩展部件应用到高性能计算机中,如,IBM 的超级计算机 BlueGene/L 和国产的神威蓝光超级计算机中都集成短向量扩展部件.国产处理器中,龙芯[2]、迈创[3]以及魂芯一号[4]都含有 SIMD 扩展部件.

SIMD 扩展指令能够将原来需要多次装载的内存中地址连续的数据一次性装载到向量寄存器中,通过一条 SIMD 扩展指令,实现对 SIMD 向量寄存器中所有数据元素的并行处理.手工向量化程序不仅容易出错而且程序员需要了解 SIMD 指令集,所以自动向量化成为发掘 SIMD 并行的首要选择[5].如今,主流编译器,如 Open64,ICC 和 GCC 等都集成了自动向量化功能.当前,两种基本的向量发掘方法分别是 Loop-based 方法和超字并行(superword level parallelism,简称 SLP)方法.面向 SIMD 向量化的 Loop-based 方法是从传统向量化方法引入的,其可发掘循环的迭代间并行.SLP 方法旨在发掘基本块内的向量并行性.Loop-aware 方法是对 SLP 方法的改进,其思想是:首先,通过循环展开将迭代间并行转换为迭代内并行,使循环体内的同构语句条数足够多;再利用 SLP 方法进行向量发掘.当循环展开次数为 1 时,Loop-aware 方法相当于 SLP 方法,当循环展开次数为向量化因子(vector factor,简称 VF)时,将同一条语句展开后的多条语句打包成向量则为 Loop-based 方法.然而,当循环展开不合法或者并行度低于向量化因子时,Loop-aware 方法无法实现程序向量并行性的发掘,如在 SPEC 2000 中,183.equake 的核心函数 *smvp* 代码段如图 1 所示,其核心比重为 72.02%,循环体内同构语句条数为 3.由于循环上下界不确定,因此不能通过循环展开将迭代间并行转为迭代内并行,导致 Loop-aware 方法不能对其向量化.

```
while (Anext<Alast) {
  col=Acol[Anext];
  sum0+=A[Anext][0][0]*v[col][0]+A[Anext][0][1]*v[col][1]+A[Anext][0][2]*v[col][2];
  sum1+=A[Anext][1][0]*v[col][0]+A[Anext][1][1]*v[col][1]+A[Anext][1][2]*v[col][2];
  sum2+=A[Anext][2][0]*v[col][0]+A[Anext][2][1]*v[col][1]+A[Anext][2][2]*v[col][2];

  w[col][0]+=A[Anext][0][0]*v[i][0]+A[Anext][1][0]*v[i][1]+A[Anext][2][0]*v[i][2];
  w[col][1]+=A[Anext][0][1]*v[i][0]+A[Anext][1][1]*v[i][1]+A[Anext][2][1]*v[i][2];
  w[col][2]+=A[Anext][0][2]*v[i][0]+A[Anext][1][2]*v[i][1]+A[Anext][2][2]*v[i][2];
  Anext++;
}
```

Fig.1 The kernel of 183.equake in SPEC 2000

图 1 SPEC 2000 中 183.equake 的核心循环

此外,当循环的迭代次数较小于向量化因子时,Loop-aware 方法也不能成功发掘.如 SPEC 2006 中,454.calculix 的核心函数 *E_c3d* 代码段如图 2 所示,其核心比重为 69.12%,最适合向量化的 i_1 层循环的迭代次数为 3,因此在 AVX 等向量化因子大于 3 的平台,Loop-aware 方法不能成功发掘.

```
do i1=1,3
    do j1=1,3
      vo(i1,j1)=0.d0
      do k1=1, nope
        vo(i1,j1)=vo(i1,j1)+shp(j1,k1)*voldl(i1,k1)
      enddo
    enddo
  enddo
```

Fig.2 The kernel of 454.calculix in SPEC 2006

图 2 SPEC 2006 中 454.calculix 的核心循环

SIMD 扩展部件的向量寄存器长度在不断增长,以 Intel 为例,最初的 MMX 为 64 位,后来出现 SSE 的 128 位到 AVX 的 256 位,再到目前 IMCI 的 512 位.即将在下一代 Xeon phi 中支持的 AVX-512 也为 512 位,并且兼容 AVX 和 SSE[6,7].向量寄存器长度的不断增长,导致向量化因子在不断增大,如 IMCI 的向量寄存器长度为 512 位,它能同时处理 8 个 double 数据,然后,程序内蕴含的向量并行性是固有的,因此,本文首先提出了不充分向量化方法,即,部分利用向量寄存器的方法;然后引入了向量并行度的概念,从迭代内并行度和迭代间并行度两个方面表达循环的向量并行度,并依据迭代间并行度、迭代内并行度和向量化因子,构建循环向量化方法选择方案;最后,再对并行性充足的向量循环进行展开,以提高向量程序的指令级并行.本文提出了向量并行度指导的循环 SIMD 向量化方法,主要贡献有以下 3 个方面:一是提出了不充分向量化方法,以发掘向量并行度低的循环向量并行性;二是引入了程序向量并行度的概念,给出了迭代内向量并行度和迭代间向量并行度的计算方法,并用它指导向量化方法的选择;三是提出了面向向量循环的展开技术,进一步提高向量程序的性能.

本文第 1 节介绍不充分向量化方法.第 2 节介绍向量并行度指导的循环 SIMD 向量化方法.第 3 节是实验分析.第 4 节是相关研究.最后总结全文.

1 不充分向量化

向量寄存器可同时装载多个数据,当前,绝大多数向量寄存器被设计为不可拆分的整体进行使用.不可拆分的整体是指向量寄存器中的每个数据都是有效的,从向量装载到向量计算再到向量存储,一系列的操作都要保证寄存器中所有数据的有效性.然而,当程序的数据并行性不足时,需要向量寄存器的部分使用.部分使用是指向量寄存器中的某些槽位为有效数据,其他槽位为无效数据.向量寄存器的 4 种使用方式如图 3 所示:图 3(a)所示为整体使用,图 3(b)所示为一端无效的部分使用,图 3(c)所示为两端无效的部分使用,图 3(d)所示为不连续的部分使用.

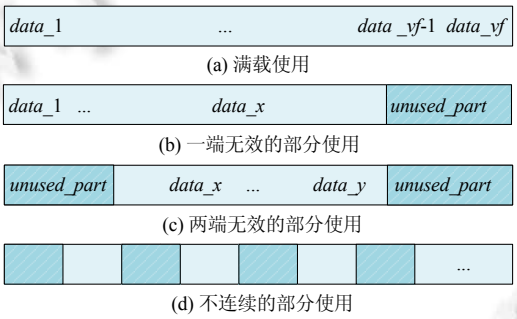


Fig.3 The use manner of vector register

图 3 向量寄存器的使用方式示意图

不充分向量化就是向量寄存器内不是每个槽位的数据都是有效的.以计算语句 S_1:$c[2i]=a[2i]+b[2i]$为例,如图 4 所示:在向量寄存器 V_a 和 V_b 中,偶数位是需要参与运算的槽位,而奇数位则不是需要运算的槽位.由于 load 指令从内存中连续地加载数据到向量寄存器中,不充分向量化就是直接对 V_a 和 V_b 的偶数位进行运算并将结果保存到 V_c 中的偶数位,而不考虑奇数位的数据.

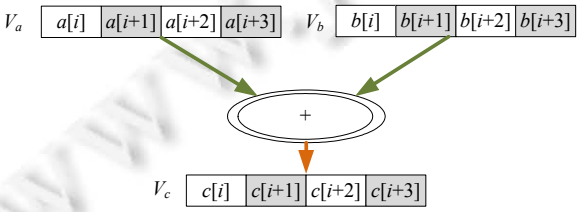


Fig.4 An example of insufficient vectorization

图 4 不充分向量化实例

充分向量化方法都是将不连续的 4 个偶数位拼到一个向量寄存器中,使得向量寄存器内都是需要参与运算的数据,这个过程需要多次的向量加载,然后利用拼接指令实现.为便于下文描述,我们将向量寄存器中是计算时需要数据的槽位称为有效槽位,而不是计算时需要的槽位称为无效槽位.利用充分向量化计算语句 S_1 的理论上最大加速比为 4,而不充分向量化的最大加速比为 2.虽然不充分向量化并没有充分发挥向量寄存器的作用,但是很多情况下其仍具有独特的优势.

1.1 适用场景

由图 3 的向量寄存器使用方式可以看出,不充分向量化方法适合于以下两种场景:一是并行度低的程序向量化;二是不连续访存程序的向量化.

1.1.1 并行度低

(1) 当迭代内并行度较低时.

如图 1 所示为在 SPEC 2000 中 183.equake 的核心函数,此外,还以 SPEC 2006 中 435.gromacs 为例,其核心循环 1130 如图 5 所示,核心中存在间接数组访问,发掘迭代间并行不能保证向量化的正确性.而在 straight-line 内同构语句的条数就是 3.将同构的 3 条语句发掘成向量化操作是最合适的.目标平台的向量化因子一般都为 2 的指数倍,当目标平台的向量化因子为 2 时,则不能充分发掘程序向量化特性;当目标平台的向量化因子为 4 或者 8 时,则需要利用不充分向量化.

```
1 for (k=nj0; k<nj1; k++) {
2 j=3*jjnr[k];
3
4 jx=pos[j];
5 jy=pos[j+1];
6 jz=pos[j+2];
7
8 dx=ix-jx;
9 dy=iy-jy;
10 dz=iz-jz;
11 rsq=dx*dx+dy*dy+dz*dz;
12
13 rinv=1.0/sqrt(rsq);
14
15 rinvsq=rinv*rinv;
16 rinvsix=rinvsq*rinvsq*rinvsq;
17 vnb6=c6*rinvsix;
18 vnb12=c12*rinvsix*rinvsix;
19 fs=(vnb12-vnb6+vcoul)*rinvsq;
20
21 fac[j]-=dx*fs;
22 fac[j+1]-=dy*fs;
23 fac[j+2]-=dz*fs;
24 }
```

Fig.5 The kernel of 435.gromacs in SPEC 2006

图 5 SPEC 2006 中 435.gromacs 的核心循环

(2) 当迭代间并行度较低时.

如图 2 所示的 SPEC 2006 中 454.calculix 的核心函数 *E_c3d*,最适合向量化的 i_1 层循环的迭代次数为 3,此外,NPB 中的 BT,SP 等很多核心循环的最内层循环迭代次数都为 5 次,因此当向量化因子大于 5 时,则必须利用不充分向量化方法发掘.

此外还包括一些其他循环并行度低的情况,如循环剥离后的尾循环或者循环体的真依赖距离小于向量化因子等,它们也是由于并行度低而必须利用不充分向量化方法发掘.

1.1.2 不连续访存

(1) 当平台没有向量重组指令或者向量重组指令的功能较弱时.

向量重组指令可以通过拼凑操作处理不连续访存的数据,然而,不同平台提供的向量重组指令能力不相同,在向量重组指令能力较弱或者根本不支持向量重组指令时,如强制将不连续的访存数据组成向量的开销是大的,甚至导致向量化没有收益.而不充分向量化无需考虑平台是否支持向量重组指令,同样可以生成向量程序.

(2) 当充分向量化效果低于不充分向量化效果时.

由于向量重组指令的开销较大,而且在充分向量化时为了组成向量,可能需要多次内存读写操作以保证结果的正确性,这些开销会大大抵消获得的向量化收益.此外,计算出最优的重组模式是 NP 难问题,因此,利用向量重组指令向量化不连续访存的程序不仅发掘方法复杂,而且有时由于向量重组代价过大而导致向量化没有效果.不充分向量化不需要重组,不存在上述几个问题.

1.2 代码生成

不论是 Loop-based 方法还是 SLP 方法,都需要通过预分析、依赖关系和代码生成等阶段.不充分向量化与充分向量化在预分析和依赖分析阶段区别不大,只需要在预分析和依赖分析等阶段添加相应的修改,使其最终能够生成不充分向量化代码.最大的挑战来自代码生成阶段,如何保证正确性是不充分向量化代码生成要解决的重要问题.不充分向量化的正确性需要从以下 3 个方面进行考虑:一是向量读内存时需要记录哪些是有效槽位、哪些是无效槽位;二是向量运算时如何保证两个向量寄存器的有效槽位是对应的;三是向量写内存时需要将有效槽位的数据写入内存,同时避免无效槽位的数据写入内存.下面对基于 SLP 和 Loop-based 的不充分向量化方法进行分析.

SLP 方法的步骤首先是按照地址相邻的原则对基本块中的语句生成初始的同构语句包(pack),然后根据 DU 和 UD 链对 pack 进行扩展,最后按照依赖关系对 pack 进行调度.首先,pack 内的数据都是有效的,在对 pack 生成对应 load 指令时,从向量寄存器的最低位开始使用,必要时可生成不对齐的内存访问以实现有效槽位的对应.由于 SLP 要求语句是同构的,因此运算时两个向量寄存器的有效槽位是对应的.计算时有效槽位和无效槽位都参与运算,因此不能将计算结果直接存入内存,利用提取指令将有效槽位的值从运算结果的向量中提取出来,并写回内存.

Loop-based 方法先是通过预分析和依赖分析,然后进行类型转换,即,将语句的数据类型由标量直接转化为向量,最后生成向量代码.通过待向量化数组的跨幅和偏移,就能够算出在向量寄存器中哪些槽位是有效槽位.下面考虑如何保证有效槽位的对应问题.首先考虑语句内参与计算的数组跨幅一致的情况,跨幅一致时,能够很好地保证槽位的对应,只要保证第 1 个有效槽位的位置相同,后面的有效槽位也是相对应的.多种跨幅类型需要引入插入提取指令甚至混洗指令,才能保证有效槽位的对应,而两种跨幅类型时引入辅助指令较少,因此,本文仅考虑两种跨幅的代码生成.跨步较大的数组有效槽位确定后,利用插入提取指令调整跨幅较小数组的有效槽位,使其与跨幅较大数组的有效槽位对应.

2 并行度指导的向量化方法

2.1 程序向量并行度的概念

基于前人对程序并行性的研究[8−13],本文认为,向量并行性作为一种并行形式同样蕴含在程序内.向量并行性可以从向量并行特征与向量并行度两个方面描述,其中,向量并行特征是指程序中蕴含的单指令多数据模式所具有的向量运算和访存等特征,而向量并行度描述了符合这种模式的向量宽度.程序存在着垂直与水平两个方向的向量化机会,垂直向量并行化是指程序的单条执行路径上先后执行的多个标量处理单元被变换为向量执行的过程,水平向量化是指程序的多条并行执行路径被变换为向量执行的过程.垂直向量化如图 6(a)所示,水平向量化如图 6(b)所示.与程序的向量化机会相对应,程序的向量并行度也可以从水平和垂直两个方面进行计算.水平向量并行度即能够被变换为向量执行的最大并行执行路径数,垂直向量并行度即一条执行路径上的可向量执行的最大单元数.

程序的向量并行性发掘可以从基本块、循环和函数这 3 个粒度进行发掘,从基本块区域内蕴含的指令级并行寻找数据级并行属于挖掘垂直向量并行度,这包括直线型代码和循环体内的基本块等.从循环区域内蕴含的

线程级并行寻找数据级并行属于挖掘水平向量并行度,从函数体区域内蕴含的任务级并行寻找数据级并行也属于挖掘水平向量并行度.为便于描述,本文中的并行度特指向量并行度.

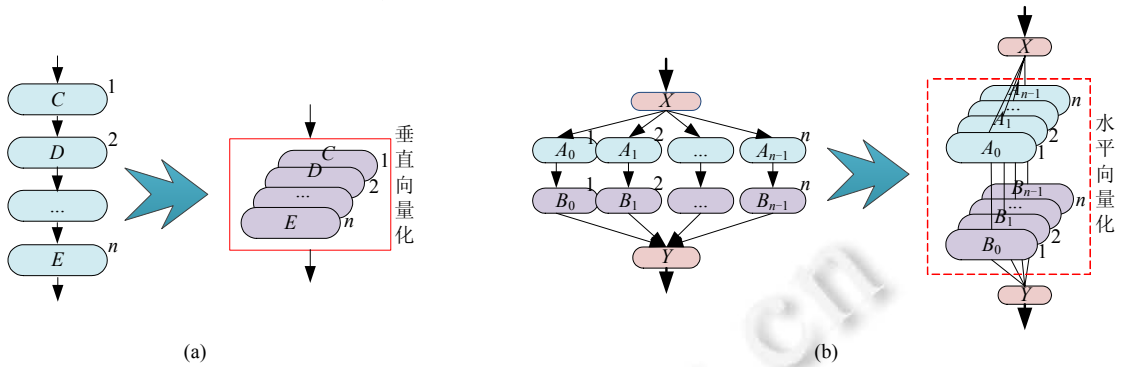


Fig.6 Illustration of vertical and horizontal vectorization method

图 6 垂直向量化和水平向量化示意图

2.2 循环向量并行度计算方法

由于循环既可以发掘迭代内向量并行性,又可以发掘迭代间向量并行性,因此,循环向量并行度的计算也需要从迭代内和迭代间两个方面进行计算.

2.2.1 迭代内并行度计算方法

由于迭代内向量并行度是程序的固有属性,因此其为定值.但是由于发掘方法的限制,不同的发掘方法计算出的并行度可能不同.本文选用 SLP 算法来发掘迭代内的并行性,因此,迭代内并行度(intra-iteration parallelism,简称 IAP)的计算也需要结合 SLP 算法.SLP 打包算法首先要求待打包语句必须是同构的.SLP 首先利用相邻地址的内存访问作为打包的种子,然后通过定义-使用链和使用-定义链启发式地扩展包,最后利用依赖关系调度包.地址相邻作为打包的种子,可使包内数据经过一次向量读从内存中取出,按照定义-使用链和使用-定义链扩展包,可以减少拆包的机会.这些启发式条件可以大大提高生成向量代码的性能.因此基于以上分析,地址连续是要满足的一个条件.此外,地址连续引用的语句之间必须没有依赖,否则,向量化后结果不正确.综合以上分析,给出了迭代内并行度的计算条件:一是语句是同构的;二是存在相邻地址引用;三是相邻地址引用所在语句间无依赖.满足以上 3 个条件的语句条数即为循环的迭代内并行度 IAP.

2.2.2 迭代间并行度计算方法

循环的迭代间并行度(inter-iteration parallelism,简称 IEP)是指循环体内某一条语句的连续多少次迭代可以向量并行执行.并行执行的合法性也是语句重排的合法性,检测重排序合法性就是对循环进行依赖关系分析,若循环内不存在影响迭代间语句重排的依赖,就认为循环体内存在迭代间并行性.传统向量机中认为硬件支持的向量长度足够长,循环内的所有迭代可以一次执行完毕,因此,待被向量化的循环不能含有循环携带依赖,仅能含有循环无关依赖.SIMD 扩展部件向量寄存器宽度是固定的,因此其对应的并行方式通常为将迭代空间分组,组内的几次迭代并行,而组间保持原执行顺序.因此,迭代间向量并行度受两个因素影响,分别是循环的依赖距离 *dep* 和循环迭代次数 *iter*.当为循环无关依赖,即依赖距离等于 0 时,迭代间向量并行度 *IEP* 等于循环迭代次数 *iter*;当依赖距离大于 0,即循环携带依赖时,迭代间向量并行度 *IEP* 为 min(*dep*,*iter*).

2.3 向量化方法选择

Loop-aware 向量化方法的实质就是当迭代内并行度较低时,通过循环展开将迭代间并行转换为迭代内并行,其要求循环的迭代间并行度较高.但是 Loop-aware 不能对以下两种循环进行发掘:一是迭代内并行度较低却又不能通过循环展开将迭代间并行转换为迭代内并行(如 while-do 或间接数组访问等);二是迭代间并行度低(迭代次数小于向量化因子 *VF*),向量寄存器长度的不断增长导致迭代间并行度低的循环越来越多.因此,采用并

行度指导的循环向量化方法去发掘这两种循环的向量并行性,将本文提出的并行度指导的循环SIMD向量化方法命名为 VMSP(vectorization method for loops guided by SIMD parallelism)方法,其核心为:根据迭代间并行度 IEP、迭代内并行度 *IAP* 和向量化因子 *VF* 来决定采用何种发掘方法,见表 1.

Table 1 Vectorization method for loops guided by SIMD parallelism

表 1 并行度指导的循环向量化方案

$IAP \geqslant VF$	$1<IAP<VF$		$IAP=1$		
直接采用 SLP 算法	$IEP \geqslant UF$	$1 \leqslant IEP<UF$	$IEP \geqslant VF$	$1<IEP<VF$	$IEP=1$
	Loop-aware 方法	基于 SLP 的不充分向量化	Loop-based 方法	基于 Loop-based 的不充分向量化	标量执行

- 当迭代内并行度 *IAP* 大于等于向量化因子 *VF*,即 $IAP \geqslant VF$ 时,说明迭代内并行度较高,直接采用 SLP 算法;
- 当迭代内并行度 *IAP* 大于 1 但是小于向量化因子 *VF*,即 $1<IAP<VF$ 时,需要循环展开才能满足充分向量化的要求,展开的次数为 $UF=\left\lceil \frac{VF}{IAP} \right\rceil$.如果 $IEP \geqslant UF$,说明迭代间并行度较大,展开 *UF* 次后可以直接打包,此时采用 Loop-aware 算法;如果 $1 \leqslant IEP<UF$,说明迭代间的并行度也较低,此时展开 *IEP* 次将所有迭代间并行都转换到迭代内以满足一次向量计算的要求,采用的方法为基于 SLP 的不充分向量化;
- 当迭代内并行度 *IAP* 等于 1,即 $IAP=1$ 时,说明迭代内不存在并行性,只能发掘迭代间并行.当 $IEP \geqslant VF$ 时,采用 Loop-based 方法;当 $1<IEP<VF$ 时,采用基于 Loop-based 的不充分向量化方法;当 $IEP=1$ 时,循环不能被向量化,只能标量执行.

在上述的各种情形中,如果利用 SLP 算法发掘结束后,循环体内仍有标量语句,可以继续利用 Loop-based 方法发掘迭代间并行,但是需要迭代间同样含有足够的并行性.此时,标量语句被转为向量形式,而原来的向量语句直接循环展开.

2.4 面向向量循环的展开

循环展开不仅可以提高指令级并行,还可以提高寄存器重用.面向向量循环的展开与标量循环的展开不同之处表现在以下 3 个方面:一是在计算标量寄存器个数时,不仅要考虑循环体内的运算和访存用的寄存器,还要考虑循环索引变量、基址寄存器、栈顶指针寄存器等,而向量寄存器仅需考虑用于循环体内的运算和访存操作即可;二是经过 if 转换后生成的向量程序中已经没有分支语句,因此向量程序展开不需要考虑分支语句的开销;三是标量程序有一些寄存器已被全局变量所有,但是向量化后的循环体内很少有全局变量.过度的循环展开会导致寄存器甚至指令缓存区溢出,因此,循环展开的关键问题是如何确定展开因子.由于向量程序在循环展开时不需要考虑循环是否包含分支和自身的寄存器需求,因此只需考虑循环体中的操作个数即可.根据循环内操作的个数 *N_sum* 和平台设定的操作上限 *SL*,计算展开因子的上限 $UFU=SL/N\_sum$.

向量化时循环的迭代次数会变为原来的 $1/VF$,因此当展开因子很大时,向量循环适合完全展开.完全展开可以消除循环开销,这对于循环嵌套的收益是很大的.然后,过度的完全展开会导致代码膨胀,同样影响程序的执行效率.如果循环迭代的次数编译时已知,只有当迭代次数 $simd\_ite \leqslant unroll\_times+1$ 时才进行完全展开,其中,*unroll_times* 表示展开次数.因为当 $simd\_ite=unroll\_times+1$ 时,循环会以 *unroll_times* 作为展开因子进行展开,且会生成一个迭代次数为 1 的向量化尾循环,与完全展开的代码膨胀程度相同;当 $simd\_ite \leqslant unroll\_times$ 时,展开后的循环中实际上只有一次迭代,同样可以完全展开.添加编译选项 *vec_unroll* 来控制循环展开,向量循环的展开流程如下.

(1) 当 $vec\_unroll=0$ 时转到步骤(4);当 $vec\_unroll=1$ 时转到步骤(2);当 $vec\_unroll>1$ 时,$unroll\_times=vec\_unroll$ 转到步骤(3);

(2) 计算出展开因子的上限 $UFU=SL/N\_sum$.为了尽量减少冗余预取的个数,需要把展开因子调整为 2 的整数次幂,$unroll\_times=2^{\lfloor \log_2^{UFU} \rfloor}$;

(3) 根据 *unroll_times* 进行循环展开.如果迭代次数 *simd_ite* 编译时已知并且 *simd_ite*≤*unroll_times*+1,对向量循环完全展开;

(4) 结束流程.

3 实验与分析

测试实验包括了识别率测试、核心函数测试和程序测试这 3 部分.将本文提出的方法在开源编译器 GCC 中实现,编译环境为 Linux 操作系统,版本为 Redhat Enterprise 5.测试平台分别为 E5-2680 和 KNC,在 E5-2680 平台上测试 AVX 和 SSE 目标码的性能,平台 CPU 主频 2.7G;在 KNC 平台上测试 IMCI 目标码的性能,平台 CPU 主频 1.3G.IMCI 的向量寄存器长度为 512 位,它能同时处理 8 个 double 数据或者 16 个 float 数据.AVX 的向量寄存器长度为 256 位,它能同时处理 4 个 double 数据或者 8 个 float 数据.SSE 的向量寄存器长度为 128 位,它能同时处理 2 个 double 数据或者 4 个 float 数据.编译器加选项-ftree-vectorize 编译运行后获得向量化时间,加选项-ftree-no-vectorize 编译运行后得到未向量化时间,用未向量化时间除以向量化时间即得到向量化的加速比.

3.1 识别率测试

我们选择 SPEC 2000 和 NPB 3.3 测试集中向量执行效果好的程序作为测试用例,选择 KNC 作为识别率的比较平台,比较本文提出的 VMSP 方法和 Loop-aware 方法的识别率.表 2 列出了两种方法在 SPEC 2000 部分程序的比较结果,表 3 列出了两种方法在 NPB 部分程序的比较结果.

Table 2 The comparision of recognition rate in SPEC 2000

表 2 两种方法对 SPEC 2000 识别率比较结果

测试集	循环总数	VMSP	Loop-aware	提升率(%)
171.swim	29	11	10	10
173.applu	26	5	5	0
183.equake	50	26	5	420
187.facerec	30	6	6	0
191.fma3d	190	21	5	320
301.apsi	20	5	5	0
平均				125

Table 3 The comparision of recognition rate in NPB

表 3 两种方法对 NPB 识别率比较结果

测试集	循环总数	VMSP	Loop-aware	提升率(%)
BT	23	15	5	200
FT	19	9	6	50
MG	62	15	15	0
LU	26	14	7	100
SP	30	12	6	100
平均				90

我们选择 SPEC 2000 中 6 个程序作为测试用例,在 171.swim,173.applu,187.facerec 和 301.apsi 这 4 个程序,本文提出的 VMSP 方法与现有的 Loop-aware 方法的识别率基本相同.由于程序 183.equake 的核心循环迭代内并行度为 3 并且为 while 循环,不能通过循环展开将迭代间并行转为迭代内并行,因此 Loop-aware 方法不能识别,而 VMSP 方法可以利用基于 SLP 的不充分向量化发掘.程序 191.fma3d 的核心循环 solve 内迭代内并行度也为 3,如下所示.

```
DO N=1, NUMRT
   MOTION(N)%Vx=MOTION(N)%Vx+DTaver*MOTION(N)%Ax
   MOTION(N)%Vy=MOTION(N)%Vy+DTaver*MOTION(N)%Ay
   MOTION(N)%Vz=MOTION(N)%Vz+DTaver*MOTION(N)%Az
ENDDO
```

A_x,A_Y,A_z 是结构体 *MOTION* 内的 3 个成员变量,由于结构体 *MOTION* 内还有许多其他的成员变量,因此即

便循环展开,相邻两次迭代的语句也不连续,导致 Loop-aware 方法不能识别,而 VMSP 方法可直接利用基于 SLP 的不充分向量化方法进行识别.以 183.equake 为例说明提升率的计算方法:由于 VMSP 成功识别了 26 个循环,而 Loop-aware 成功识别了 5 个循环,因此提升率为(26−5)/5=420%.在 SPEC 2000 中,VMSP 方法相对于 Loop-aware 方法平均提升 125%.

我们选择 NPB 中的 5 个程序作为测试用例.在 MG 和 FT 这两个程序中,本文提出的 VMSP 方法与现有的 Loop-aware 方法的识别率基本相同.BT 的第 1 核心函数 *binvcrhs* 经过前向替换和语句重排以后,核心函数存在的并行度可分为如下形式.

- 并行度为 4 的语句块.
  - *lhs*(2,2)=*lhs*(2,2)−*lhs*(2,1)**lhs*(1,2);
  - *lhs*(3,2)=*lhs*(3,2)−*lhs*(3,1)**lhs*(1,2);
  - *lhs*(4,2)=*lhs*(4,2)−*lhs*(4,1)**lhs*(1,2);
  - *lhs*(5,2)=*lhs*(5,2)−*lhs*(5,1)**lhs*(1,2);
- 并行度为 3 的语句块.
  - *lhs*(3,3)=*lhs*(3,3)−*lhs*(3,2)**lhs*(2,3);
  - *lhs*(4,3)=*lhs*(4,3)−*lhs*(4,2)**lhs*(2,3);
  - *lhs*(5,3)=*lhs*(5,3)−*lhs*(5,2)**lhs*(2,3);
- 并行度为 2 的语句块.
  - *lhs*(4,4)=*lhs*(4,4)−*lhs*(4,3)**lhs*(3,4);
  - *lhs*(5,4)=*lhs*(5,4)−*lhs*(5,3)**lhs*(3,4).

而 NPB 中 BT 的第 2 个核心循环 *compute_rhs* 如下.

```
do j=1, grid_points(2)−2
  do i=3, grid_points(1)−4
    do m=1, 5
      rhs(m,i,j,k)=rhs(m,i,j,k)−dssp*
>             (u(m,i−2,j,k)−4.0d0*u(m,i−1,j,k)+
>             6.0*u(m,i,j,k)−4.0d0*u(m,i+1,j,k)+
>             u(m,i+2,j,k))
    enddo
  enddo
enddo
```

第 1 核心函数 *binvcrhs* 内的并行度为 2,3 和 4,而第 2 核心函数适合向量化的最内层循环的迭代间并行度为 5,它们都小于 KNC 平台的向量化因子 8.LU 和 SP 的核心函数与 BT 类似,也都存在向量并行度低的情况.因此,Loop-aware 方法对 BT,SP 和 LU 的向量识别率低于本文提出的 VMSP 方法.在 NPB 中,VMSP 方法相对于 Loop-aware 方法,平均提升 90%.

由于在 SPEC 2000 中,VMSP 方法相对于 Loop-aware 方法平均提升 125%,在 NPB 中,VMSP 方法相对于 Loop-aware 方法平均提升 90%,因此,本文提出的 VMSP 方法相对于 Loop-aware 方法在选定的测试程序中平均提升了 107.5%.

3.2 核心函数测试

本节对上节 VMSP 识别成功而 Loop-aware 没有识别成功的核心函数进行测试,分为迭代内并行度低的核心函数测试和迭代间并行度低的核心函数测试,测试在 SSE,AVX 和 IMCI 这 3 个平台上进行,同时比较不同向量寄存器长度对程序向量执行性能的影响.

3.2.1 迭代内并行度低的核心函数测试

迭代内并行度低的循环在实际应用中广泛存在.我们从 SPEC 2006,SPEC 2000 和 NPB 3.3 这 3 个标准测试集中选择比重较高的核心函数作为测试用例,表 4 列出了它们的比重、并行度以及仅能发掘迭代内并行的原因.

虽然迭代间并行可以通过循环展开转换为迭代内并行,但是由于不是所有的循环展开都合法,因此有些循环仅能直接发掘迭代内并行.表 4 列出了不能利用 Loop-aware 方法发掘充分向量化的原因,包括以下 4 个方面:一是非循环结构,即,语句在基本块内;二是语句在 while-do 的非标准循环结构,不能展开;三是循环内含有间接数组,导致循环展开的正确性不能保证;四是循环展开后,相邻两次迭代的语句内存引用并不连续.

Table 4 Kernels of low intra-iteration parallelism

表 4 迭代内并行度低的核心函数测试用例

函数名称	程序	比重(%)	并行度	仅能发掘迭代内并行的原因
*inl*1130	435.gromacs	75	3	间接数组访问,不能保证循环展开的正确性
smvp	183.equake	72	3	非标准循环结构,不能循环展开
solve	191.fma3d	33	3	相邻两次迭代的语句内存引用不连续
binvcrhs	BT	23	2,3,4	非循环结构,不能展开
buts	LU	19	2,3,4	非循环结构,不能展开
z_solve	SP	18	2,3,4	非循环结构,不能展开

我们选择 SSE,AVX 和 IMCI 这 3 个平台进行测试比较,由于选择的测试用例都是双精度浮点,因此这 3 个平台的向量化因子分别为 2,4 和 8,迭代内并行度低的核心函数测试结果如图 7 所示.

- 在 SSE 平台上,所有的程序都可以利用充分向量化方法挖掘,*inl*1130,*smvp* 和 *solve* 的加速比分别为 1.19,1.34 和 1.14,*binvcrhs*,*buts* 和 *z_solve* 的加速比分别为 1.86,1.54 和 1.28;
- 在 AVX 平台,*inl*1130,*smvp* 和 *solve* 这 3 个程序需要利用不充分向量化发掘,加速比分别为 1.33,1.64 和 1.37.*binvcrhs*,*buts* 和 *z_solve* 这 3 个程序同时需要利用充分向量化方法和不充分向量化方法,其加速比分别为 2.203,2.36 和 1.88;
- 在 IMCI 平台,所有程序都需要不充分向量化发掘,*inl*1130,*smvp* 和 *solve* 的加速比分别为 1.29,1.50 和 1.24.*binvcrhs*,*buts* 和 *z_solve* 的加速比分别为 2.15,2.36 和 1.81.

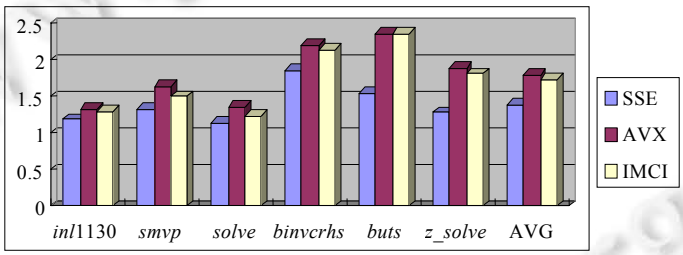


Fig.7 Performance result of low intra-iteration parallelism

图 7 迭代内并行度低的核心函数测试结果

图 7 中最后一栏 AVG 表示选定测试用例在 3 个平台的平均加速比,SSE 平台的平均加速比为 1.39,AVX 平台的平均加速比为 1.80,IMCI 平台的平均加速比为 1.73.Loop-aware 方法仅能发掘充分向量化,因此,其在这 3 个平台上的最优加速比为 SSE 平台的 1.20;而 VMSP 方法在这 3 个平台的最优加速比为 AVX 的 1.80.

3.2.2 迭代间并行度低的核心函数测试

迭代间并行度低的循环同样广泛存在于实际应用中.我们从 SPEC 2006,SPEC 2000 和 NPB 3.3 这 3 个标准测试集中选择比重较高的核心函数作为测试用例,见表 5.表中列出了核心函数名称、来自哪个程序,并列出了函数中占的比重和并行度.这些都是排名第 1 位或者第 2 位的核心函数,因此,如能将它们向量化,则能有效提升程序运行效率.由于依赖距离导致向量并行度低的例子在实际应用中并非广泛存在,因此迭代间并行度低的原因更多是因为循环的迭代次数较少所致.

我们分别在 SSE,AVX 和 IMCI 这 3 个平台上测试.由于选择的测试用例都是双精度浮点,因此,这 3 个平台的向量化因子分别为 2,4 和 8.所有选定的程序在 IMCI 平台上都需要利用不充分向量化技术,而 E_c3d 和

x_solve 在 AVX 平台需要发掘不充分向量化,其他程序都可以发掘充分向量化,所有程序在 SSE 平台都发掘充分向量化,迭代间并行度低的测试结果如图 8 所示.

Table 5 Kernels of low inter-iteration parallelism

表 5 迭代间并行度低的核心函数测试用例

函数	程序	比重(%)	并行度
E_c3d	454.calculix	69	3
x_solve	SP	17	3
mat_times_vec	410.bwaves	30	5
compute_rhs	BT	16	5
Rhs	LU	24	5

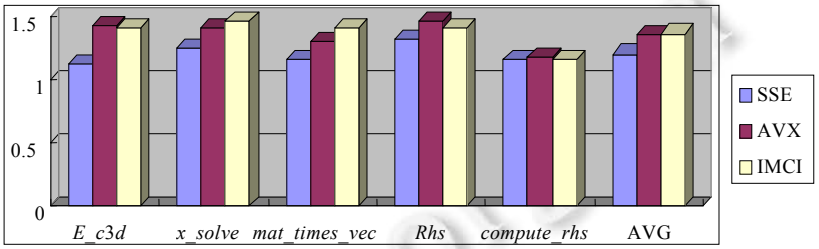


Fig.8 Performance result of low inter-iteration parallelism

图 8 迭代间并行度低的核心函数测试结果

由于 *E_c3d* 和 *x_solve* 的并行度为 3,因此在 AVX 和 IMCI 必须利用不充分向量化,*E_c3d* 在两个平台的加速比分别为 1.42 和 1.40,而 *x_solve* 在 IMCI 平台的加速比分别为 1.40 和 1.46.SSE 平台可以实现充分的向量化发掘,两个程序的加速比分别为 1.12 和 1.24.

mat_times_vec,*compute_rhs* 和 *Rhs* 在 SSE 和 AVX 平台上都可以发掘充分向量化.3 个核心函数在 AVX 平台上的加速比分别为 1.30,1.18 和 1.45,在 SSE 平台上的加速比分别为 1.15,1.15 和 1.32.IMCI 平台上由于不能发掘充分向量化,因此采用掩码存取实现不充分向量化,3 个核心函数在 IMCI 平台的加速比分别为 1.40,1.15 和 1.40.

图 8 中,最后一栏 AVG 表示选定测试用例在 3 个平台的平均加速比,SSE 平台的平均加速比为 1.20,AVX 平台的平均加速比为 1.35,IMCI 平台的平均加速比为 1.36.Loop-aware 方法仅能发掘充分向量化,因此其在这 3 个平台上的最优加速比为 SSE 平台的 1.20,而 VMSP 方法在这 3 个平台的最优加速比为 IMCI 的 1.36.

AVX 平台的理论加速比为 SSE 平台的 2 倍,并且 AVX 兼容 SSE,即:当 AVX 利用不充分向量化的效果低于 SSE 时,可直接利用 AVX 的低 128 位向量寄存器实现.所以,AVX 平台的向量执行效率不会劣于 SSE 平台.SSE 虽然利用充分向量化技术,也不会高于 AVX 平台的加速比.这说明,当向量化因子小于并行度时,向量寄存器长度是制约程序向量执行效果的关键因素.从 AVX 和 IMCI 的平台测试结果可以看出:AVX 虽然有时不能充分挖掘其并行度,而 IMCI 平台可以实现充分发掘,但是 IMCI 平台的加速比低于 AVX.这是由于 IMCI 不兼容 AVX,并且 IMCI 没有支持高效灵活的向量寄存器部分利用的功能,因此对于 512 位甚至更长的向量寄存器来说,支持高效的部分利用向量寄存器是不充分向量化的关键.

3.3 程序测试

本节对上面的程序进行整体测试,比较本文提出的 VMSP 方法和 Loop-aware 方法的性能.测试程序从 SPEC 2006,SPEC 2000 和 NPB 中选择.SPEC 2006 和 SPEC 2000 采用 reference 规模输入,NPB 采用 B 规模输入,在 KNC 平台上进行测试,测试结果如图 9 所示,分别比较 Loop-aware,VMSP 和打开面向向量循环的开关后的 VMSP 方法的性能,循环展开选项是自动计算循环展开因子.

Loop-aware 方法对程序 454.calculix,435.gromacs 和 410.bwaves 的识别率比较低,因此,程序的向量程序执行的加速比也不高,分别是 1.01,1.03 和 1.03.而 VMSP 方法能够成功识别这 3 个程序的核心函数,因此,VMSP

方法对这 3 个程序的加速比高于 Loop-aware 方法,分别是 1.16,1.08 和 1.10.打开 Unroll 选项后,410.bwaves 的向量性能得到进一步提升,提升到了 1.16.这是由于 410.bwaves 的核心 mat_times_vec 是一个 5 层嵌套循环,通过完全展开后,不仅提高了指令级并行,而且消除了最内层循环的开销.

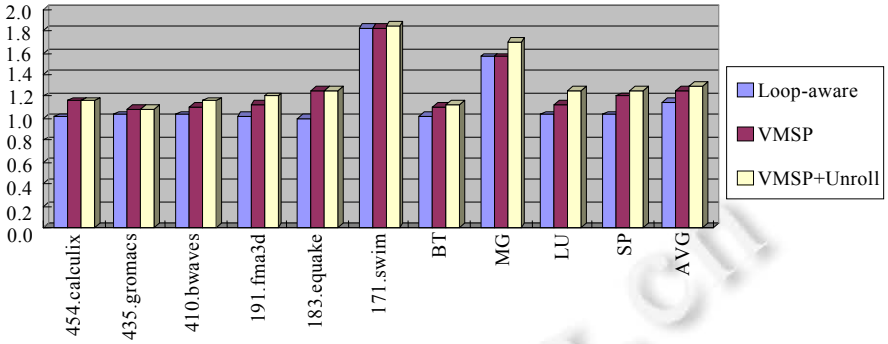


Fig.9 Performance result of full program

图 9 整体程序测试结果

由于程序 191.fma3d 和 183.equake 的核心函数的迭代内并行度都低于向量化因子,并且不能通过循环展开将迭代内并行转换为迭代间并行,因此,Loop-aware 方法不能成功地将这两个程序的核心函数向量化,导致 Loop-aware 方法的加速比较低,分别为 1.02 和 1.0.而 VMSP 方法可以成功地将这两个循环向量化,因此,获得的加速比高于 Loop-aware 方法,分别是 1.13 和 1.25.但是由于 183.equake 的核心为 while 循环,因此,打开 Unroll 选项后对其无效;而 191.fam3d 打开 Unroll 选项后有一点点性能提升,为 1.20.

BT,LU 和 SP 这 3 个程序的核心循环的迭代间并行度为 5,迭代内并行度也比较低.因此,利用 Loop-aware 方法识别率较低,而 VMSP 方法可以将这些循环向量化.Loop-aware 方法对这 3 个程序的加速比分别为 1.02,1.03 和 1.03,而 VMSP 方法对这 3 个程序的加速比分别为 1.10,1,13 和 1.20.由于这 3 个程序的核心循环都为 3 层循环嵌套,因此虽然最内层完全展开,但是性能提升不高,分别为 1.12,1.25 和 1.25.

VMSP 方法和 Loop-aware 方法对 171.swim 和 MG 的识别率相同,这是因为这两个程序都可以发掘充分的向量化,因此两种方法获得的加速比也相同,都为 1.83 和 1.56.由于循环的向量并行性充足,因此打开 Unroll 选项后,程序的性能得到进一步提升,分别为 1.85 和 1.70.

最后一栏 AVG 表示 3 种方法加速比的平均值.Loop-aware 方法的平均加速比为 1.16,VMSP 的平均加速比为 1.26,VMSP+Unroll 方法的平均加速比为 1.30.因此,本文提出的向量并行度指导的方法比现有的 Loop- aware 方法在性能上提升了 12.1%.

4 相关研究

目前,自动向量化研究主要集中在发掘和优化两个方面.

在发掘方法方面,面向循环的 Loop-based 向量化方法与面向向量机的传统向量化方法相同[14,15],都是在迭代间并行.随后提出了面向 SIMD 的外层循环向量化[16]、多面体指导的多重循环向量化[17,18].SLP 是第一个发掘迭代内并行的算法,后面又对其进行了改进[19−23].此外,还有函数级向量化[24],但是由于函数级向量化涉及到过程间分析和别名分析,使得发掘难度比较大.投机并行的 SIMD 向量化是未来研究的重要方向[25−27].由于程序中大量控制流语句的存在,控制依赖成为发掘 SIMD 向量化的一道难题.当前,向量化控制依赖有两种方法:一是直接处理控制依赖,通过首先向量化各个基本块,然后在每个基本块内插入赋值语句,以保证向量化的正确性[28];二是采用 if 转换,将控制依赖转换为数据依赖[29−31].

向量优化是指对编译器构造的向量操作进行优化的过程.由于在向量操作生成时,为了解决不对齐和不连续的内存访问,引入了许多辅助指令,从而导致产生额外的开销,优化阶段就是要减小辅助指令的开销[32].由于对齐访存要比非对齐访存快得多,而应用程序中很多程序都不是对齐的,因此,处理好非对齐访存是提高向量化

代码执行效率的重要途径[33].解决非对齐访存的方法可分为 3 类:一是通过多次对齐访存,然后移位[34]或者重组[35];二是通过循环变换[36];三是硬件支持非对齐访存[37,38].由于访存指令只能从内存中连续地加载数据到向量寄存器中,因此,不连续访存的程序就需要额外的操作使其在向量寄存器内连续.解决方法可分为 3 类:一是硬件支持[39];二是向量重组[40−42];三是程序变换[43].

5 结束语

本文利用向量并行度指导循环的 SIMD 向量化方法选择,以充分发掘循环的向量并行性,提高程序的向量执行效率.首先提出了程序向量并行度的概念,并给出了循环迭代内并行度和迭代间并行度的计算方法;其次,提出了部分利用向量寄存器的不充分向量化方法;然后,利用循环的向量并行度指导选择应用哪种向量化方法,以充分挖掘该循环的向量并行性;最后提出了面向向量循环的展开技术,进一步提高程序的向量执行效率.在 KNC 平台的实验结果表明:与目前广泛应用于编译器的 Loop-aware 方法相比,本文提出的方法识别率提高了 107.5%,性能提高了 12.1%.

程序的向量并行性可以从循环、基本块和函数等多个粒度进行挖掘,本文提出了并行度指导的循环向量化方法.如何发掘函数级向量化,以及针对具体的应用程序如何依据其向量并行性合理地选择向量化方法编译组合策略以获得最优的向量并行收益,都是亟待解决的问题[44].

致谢 在此,向对本文研究工作提供基金支持的单位和评阅本文的审稿专家表示衷心的感谢,向为本文研究工作提供基础和研究平台的前辈致敬.

References:

[1] Zhang WH, Zang BY. SIMD compiling technology review. Communication of the CCF, 2007,3(2):27−36 (in Chinese with English abstract).

[2] Peng F, Gu NJ. SIMD compiler optimization and analysis based on Godson-3B processor. Journal of Chinese Computer Systems, 2012,33(12):2733−2737 (in Chinese with English abstract).

[3] Xin NJ, Chen XC. Extending the vector instruction set for high-performance DSP matrix based on GCC. Computer Engineering & Science, 2012,34(1):57−63 (in Chinese with English abstract).

[4] Xu HY, Zheng QL, Ding CF, Xu DP. Vectorization algorithm for multi-cluster and VLIW DSP. Computer System and Application, 2013,22(12):140−143 (in Chinese with English abstract).

[5] Wald I, Leißa R, Hack S. Extending a C-like language for portable SIMD programming. In: Proc. of the 17th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PPoPP). New Orleans, 2012. [doi: 10.1145/2145816.2145825]

[6] Huo X, Ren B, Agrawal G. A programming system for Xeon Phis with runtime SIMD parallelization. In: Proc. of the ICS. 2014. [doi: 10.1145/2597652.2597682]

[7] Ramachandran A, Vienne J, Wijngaart RVD, Koesterke L. Performance evaluation of NAS parallel benchmarks on Intel Xeon Phi. In: Proc. of the 42nd Int'l Conf. on Parallel Processing (ICPP). 2013. 736−743. [doi: 10.1109/ICPP.2013.87]

[8] Jeon D, Garcia S, Louie C, Taylor MB. Kismet: Parallel speedup estimates for serial programs. In: Proc. of the 26th Annual ACM SIGPLAN Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). 2011. [doi: 10.1145/2048066.2048108]

[9] Jeon D, Garcia S, Louie C. Parkour: Parallel speedup estimates for serial programs. In: Proc. of the 3rd USENIX Conf. on Hot topic in Paralelism. Berkeley: USENIX Association, 2011. 519−536.

[10] Sah S, Vaidya VG. A review of parallelization tools and introduction to easypar. Int'l Journal of Computer Applications, 2012, 56(12):30−34. [doi: 10.5120/8944-3108]

[11] Peternier A, Binder W, Chen L. Parallelism profiling and wall-time prediction for multi-threaded applications. In: Proc. of the ICPE 2013. Prague, 2013. [doi: 10.1145/2479871.2479901]

[12] Kim MJ, Kumar P, Kim HS, Brett B. Predicting potential speedup of serial code via lightweight profiling and emulations with memory performance model. In: Proc. of the Int'l Conf. on Parallel and Distributed Processing Symp. (IPDPS). 2012. 1318−1329. [doi: 10.1109/IPDPS.2012.128]

[13] Li Z, Jannesari A, Wolf F. Discovery of potential parallelism in sequential programs. In: Proc. of the 42nd Int'l Conf. on Parallel Processing (ICPP). 2013. [doi: 10.1109/ICPP.2013.119]

[14] Smith JE, Faanes G, Sugumar R. Vector instruction set support for conditional operations. In: Proc. of the 27th Annual Int'l Symp. on Computer Architecture. Vancouver, 2000. 260−269. [doi: 10.1145/339647.339693]

[15] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures. San Francisco: Morgan Kaufmann Publishers, 2001.

[16] Nuzman D, Zaks A. Outer-Loop vectorization-revisited for short SIMD architectures. In: Proc. of the 2008 Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2008. [doi: 10.1145/1454115.1454119]

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

[18] Kong M, Veras R, Stock K. When polyhedral transformations meet SIMD code generation. In: Proc. of the 2013 Conf. on Programming Language Design and Implementation (PLDI). 2013. [doi: 10.1145/2491956.2462187]

[19] 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. 2000. 145−156. [doi: 10.1145/349299.349320]

[20] Barik R, Zhao JS, Sarkar V. Efficient selection of vector instructions using dynamic programming. In: Proc. of the 43rd Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). 2010. [doi: 10.1109/MICRO.2010.38]

[21] Liu J, Zhang YR, Kandemir M. A compiler framework for extracting superword level parallelism. In: Proc. of the 2012 Conf. on Programming Language Design and Implementation (PLDI). 2012. [doi: 10.1145/2254064.2254106]

[22] Porpodas V, Magni A, Timothy M. PSLP: Padded SLP automatic vectorization. In: Proc. of the 2015 Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). 2015. [doi: 10.1109/CGO.2015.7054199]

[23] Kim T, Hoskote Y. Automatic generation of custom SIMD instructions for superword level parallelism. In: Proc. of the DATE 2014 Conf. on Design, Automation & Test in Europe. 2014. [doi: 10.7873/DATE.2014.375]

[24] Karrenberg H, Hack S. Whole-Function vectorization. In: Proc. of the 9th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). 2011. 141−150. [doi: 10.1109/CGO.2011.5764682]

[25] Kumar1 R, Martínez2 A. Speculative dynamic vectorization for HW/SW codesigned processors. In: Proc. of the 2012 Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2012. [doi: 10.1145/2370816.2370895]

[26] Haque M, Yi Q. Past dependent branches through speculation. In: Proc. of the 22nd Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT). Washington: IEEE Computer Society, 2013. [doi: 10.1109/PACT.2013.6618831]

[27] Yi Q, Wang Q, Cui HM. Specializing compiler optimizations through programmable composition for dense matrix computations. In: Proc. of the 47th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). 2014. [doi: 10.1109/MICRO.2014.14]

[28] Tanaka H, Ota Y. A new compilation technique for SIMD code generation across Basic block boundaries. In: Proc. of the 15th Asia and South Pacific Design Automation Conf. (ASP-DAC). 2010. 101−106. [doi: 10.1109/ASPDAC.2010.5419911]

[29] Zhu JF, Zhao RC. A vectorization method of export branch for SIMD extension. In: Proc. of the 10th Conf. IEEE/ACIS Int'l Conf. on Computer and Information Science (ICIS). 2011. [doi: 10.1109/ICIS.2011.49]

[30] Allen J, Kennedy K, Porterfield C, Warren J. Conversion of control dependence to data dependence. In: Proc. of the 20th Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages (POPL). New York: ACM Press, 1983. 177−189. [doi: 10.1145/567067.567085]

[31] Smith JE, Faanes G, Sugumar R. Vector instruction set support for conditional operations. In: Proc. of the 27th Annual Int'l Symp. on Computer Architecture. Vancouver, 2000. 260−269. [doi: 10.1145/339647.339693]

[32] Stock K, Kong M, Grosser T, Pouchet L-N, Rastello F, Ramanujam J, Sadayappan P. A framework for enhancing data reuse via associative reordering. ACM SIGPLAN Notices, 2014,49(6):65−76. [doi: 10.1145/2594291.2594342]

[33] Shahbahrami A, Juurlink B, Vassiliadis S. Performance impact of misaligned accesses in SIMD extensions. In: Proc. of the 17th Annual Workshop on Circuits, Systems and Signal Processing. 2006. 334−342.

[34] Eichenberger AE, Wu P, O'Brien K. Vectorization for SIMD architectures with alignment constraints. In: Proc. of the ACM SIGPLAN 2004 Conf. on Programming Language Design and Implementation (PLDI). New York: ACM Press, 2004. 82−93. [doi: 10.1145/996841.996853]

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

[36] Larsen S, Witchel E, Amarasin SP. Increasing and detecting memory address congruence. In: Proc. of the 2002 Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2002. 18−29. [doi: 10.1109/PACT.2002.1105970]

[37] Chang H, Sung W. Efficient vectorization of SIMD programs with non-aligned and irregular data access hardware. In: Proc. of the 2008 Int'l Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES). 2008. 167−176. [doi: 10.1145/1450095.1450121]

[38] Jie SJ, Kapre N. Comparing soft and hard vector processing in FPGA-based embedded systems. In: Proc. of the 24th Field Programmable Logic and Applications (FPL). 2014. [doi: 10.1109/FPL.2014.6927467]

[39] Chang H, Sung W. Efficient vectorization of SIMD programs with non-aligned and irregular data access hardware. In: Proc. of the 2008 Int'l Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES). 2008. 167−176. [doi: 10.1145/1450095.1450121]

[40] Ren G, Wu P, Padua DA. Optimizing data permutations for SIMD devices. In: Proc. of the 2006 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). 2006. 118−131. [doi: 10.1145/1133981.1133996]

[41] Huang LB, Shen L, Wang ZY, Shi W, Xiao N, Ma S. SIF: Overcoming the limitations of SIMD devices via implicit permutation. In: Proc. of the 16th Int'l Symp. on High-Performance Computer Architecture (HPCA). 2010. 355−366. [doi: 10.1109/HPCA.2010.5416631]

[42] Nuzman D, Rosen I, Zaks A. Auto-Vectorization of interleaved data for SIMD. In: Proc. of the ACM SIGPLAN 2006 Conf. on Programming Language Design and Implementation (PLDI). 2006. 132−143. [doi: 10.1145/1133981.1133997]

[43] Li YX, Shi H, Chen L. Vectorization-Oriented local data regrouping. Computer System, 2009,30(8):1529−1534 (in Chinese with English abstract).

[44] Gao W, Zhao RC, Han L, Pang JM, Ding R. Research on SIMD auto-vectorization compiling optimization. Ruan Jian Xue Bao/Journal of Software, 2015,26(6):1265−1284 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4811.htm [doi: 10.13328/j.cnki.jos.004811]

附中文参考文献:

[1] 张为华,臧斌宇.SIMD 编译优化技术研究概述.中国计算机学会通讯,2007,3(2):27−36.

[2] 彭飞,顾乃杰.龙芯 3B 的 SIMD 编译优化及分析.小型微型计算机系统,2012,33(12):2733−2737.

[3] 辛乃军,陈旭灿.基于 GCC 的高性能 DSP Matrix 向量指令集扩展.计算机工程与科学,2012,34(1):57−63.

[4] 徐华叶,郑启龙,丁陈飞,徐东鹏.面向多簇超长指令字 DSP 的向量化优化算法.计算机系统应用,2013,22(12):140−143.

[43] 李玉祥,施慧,陈莉.面向向量化的局部数据重组.小型微型计算机系统,2009,30(8):1528−1534.

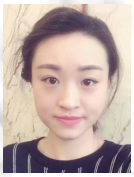
[44] 高伟,赵荣彩,韩林,庞建明,丁锐.SIMD 自动向量化编译优化概述.软件学报,2015,26(6):1265−1284. http://www.jos.org.cn/1000-9825/4811.htm [doi: 10.13328/j.cnki.jos.004811]

高伟(1988－),男,黑龙江齐齐哈尔人,博士生,主要研究领域为先进编译技术.

韩林(1978－),男,博士,副教授,CCF 专业会员,主要研究领域为高性能计算,先进编译技术.

赵荣彩(1957－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为高性能计算,先进编译技术.

徐金龙(1985－),男,博士,讲师,主要研究领域为高性能计算,先进编译技术.



陈超然(1989－),女,硕士,主要研究领域为先进编译技术.