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.

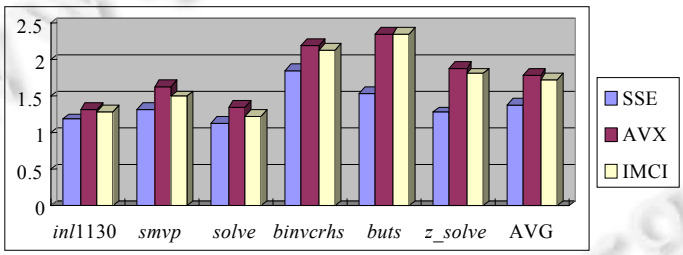


Fig.7 Performance result of low intra-iteration parallelism

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

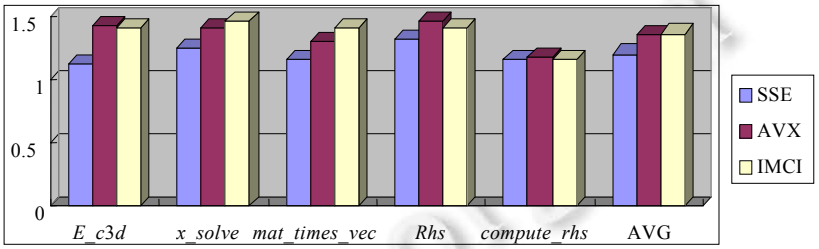


Fig.8 Performance result of low inter-iteration parallelism

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.

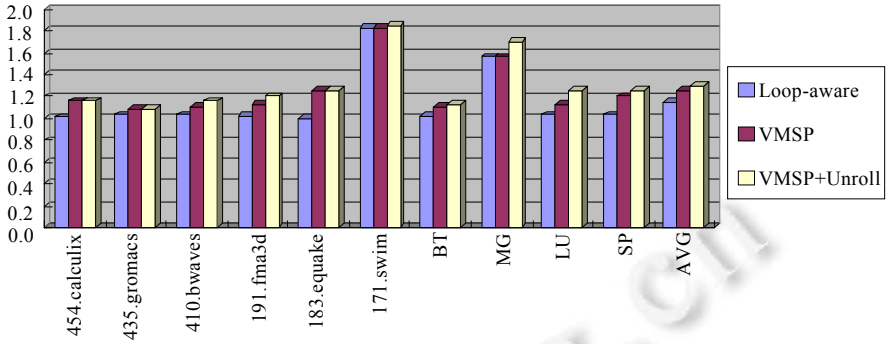


Fig.9 Performance result of full program

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.

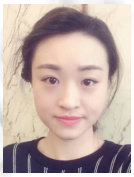
[43] 李玉祥,施慧,陈莉.面向向量化的局部数据重组.小型微型计算机系统,2009,30(8):1528−1534.

[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.

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.

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.



陈超然(1989−),女,硕士,主要研究领域为先进编译技术.

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