软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(6):1265−1284 [doi: 10.13328/j.cnki.jos.004811]

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

SIMD 自动向量化编译优化概述*

高 伟, 赵荣彩, 韩 林, 庞建民, 丁 锐

(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 450001)

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,旨在发掘多媒体程序和科学计算程序的数据级并行.首先介绍 SIMD 扩展部件的背景和研究现状,然后从发掘方法、数据布局、多平台向量化这 3 个角度介绍了 SIMD 自动向量化的研究问题、困难和最新研究成果,最后展望了 SIMD 编译优化未来的研究方向.

关键词: SIMD 扩展部件;自动向量化;数据级并行;编译优化

中图法分类号: TP312 **文献标识码**: A

中文引用格式: 高伟,赵荣彩,韩林,庞建民,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

英文引用格式: 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). http://www.jos.org.cn/1000-9825/4811.htm

Research on SIMD Auto-Vectorization Compiling Optimization

GAO Wei, ZHAO Rong-Cai, HAN Lin, PANG Jian-Min, DING Rui

(State Key Laboratory of Mathematical Engineering and Advanced Computing (PLA Information Engineering University), Zhengzhou 450001, China)

Abstract: SIMD extension is an acceleration component integrated into the general processor for developing data level parallelism in multimedia and scientific computing applications. Firstly, in this study the background and research status of SIMD extension are introduced. Next, challenges and latest research achievements in SIMD auto-vectorization are discussed from three perspectives: development method, data layout and vectorization in multi-platform. Finally, some future trends in the SIMD compiling optimization are addressed.

Key words: SIMD extension; auto-vectorization; data level parallelism; compiling optimization

伴随着多媒体产业的发展,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,称为 SIMD 扩展部件.SIMD 扩展部件能够对多媒体程序中的数据进行并行处理,提升了多媒体程序的运行速度.在特定的微处理器体系结构上,SIMD 扩展指令允许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.这种执行方式非常适合于处理计算密集、数据相关性少的音视频解码等多媒体程序.

1996 年, Intel 率先在处理器中集成 SIMD 扩展部件,此后一直都在改进 SIMD 扩展部件,包括向量寄存器长度和指令集等,先后出现了 MMX,SSE,AVX 和 IMCI.面向 SIMD 扩展部件编译技术的发展与其体系结构和应用程序的发展息息相关.图 1 以 Intel 的 SIMD 扩展部件体系结构的发展说明应用程序和编译技术在过去十几年的发展变化情况.1996 年,Intel 在其奔腾处理器上集成了 MMX,用于加速视频、音频和图像处理等多媒体程序.

* 基金项目: “核高基”国家科技重大专项(2009ZX01036-001-001-2)
收稿时间: 2014-04-08; 修改时间: 2014-08-12, 2014-11-17, 2014-12-09; 定稿时间: 2014-12-22; jos 在线出版时间: 2015-02-02
CNKI 网络优先出版: 2015-02-02 15:14, http://www.cnki.net/kcms/detail/11.2560.TP.20150202.1514.002.html

1996 1997 1998	MMX 率先在 Pentium 支持,包括算术、移位、逻辑、比较和置位等指令			
1999 2000		SSE 率先在 Pentium 3 中支持,向量寄存器长度由 MMX 的 64 位扩展到 128 位		
2001				
2002 2003		SSE2 率先在 Pentium 4 中支持,包括 SIMD 浮点和整型指令以及浮点和整型 SIMD 数据之间的转换指令		
2004 2005		SSE 3 率先在 Prescott 核心的 Pentium 4 中出现,支持不对齐访存,处理虚数运算的复杂算术指令和水平加、减操运算指令		
2006 2007 2008		SSE 4.1 率先在 Penryn 中出现,改进了插入、提取、寻找、离散、跨步存取等操作 SSE 4.2 率先在 Nehalem 中出现,加入了处理字符串文本和面向应用的优化指令		
2009 2010 2011 2012 2013			AVX 率先在 SNB 中支持,向量寄存器长度由 SSE 的 128 位扩展到 256 位,增强了数据重排和灵活的不对齐内存地址访问 AVX 2 率先在 Haswell 中支持,添加了 256 位整数矢量操作、融合乘加、256 位跨通道数据重排、gather 指令和寄存器间的广播指令	IMCI 率先出现在 KnightsCorner 中,向量寄存器长度扩展到 512 位,支持 gather/scatter,添加了写掩码寄存器,通过 swizzels 重新排序向量中的数据
2014				AVX-512 预计在 Knights Landing 中支持
	64-bit vector	128-bit vector	256-bit vector	512-bit vector
编译技术	(1) 利用传统向量机中循环级向量化的思想生成 SIMD 向量程序	(1) 面向基本块的 SLP 向量化方法 (2) 利用移位指令处理非对齐访存 (3) 利用重组指令解决不连续访存 (4) 控制流的向量化 (5) 利用硬件支持的跨幅访存指令生成向量化程序	(1) 外层循环向量化 (2) 多重循环向量化 (3) 面向多平台的向量化 (4) 更多的硬件支持不对齐和不连续	(1) 含有间接数组程序的向量化 (2) 如何利用掩码寄存器 (3) 利用 gather 和 scatter 指令生成高效的向量程序 (4) 根据程序的向量并行度决定生成 SSE,AVX 或者 AVX- 512 的向量程序
应用程序	(1) 视频、音频和图像处理等多媒体程序	(1) 绘图、3D 游戏、视频等多媒体程序 (2) XML 文档和字符串文本处理程序	(1) 视频和 3D 动画等多媒体程序 (2) 高性能计算	(1) 高性能计算 (2) 多媒体程序

Fig.1 The development of SIMD extension, compiling technology and application

图 1 SIMD 扩展部件结构及编译技术和应用程序的发展

1999 年,Intel 在 Pentium 3 处理器上采用了 SSE.2002 年,Intel 发布了包含 SSE 2 指令的 Pentium 4 处理器,2004 年,Intel 在其最新发布的奔腾 4 处理器上集成了 SSE 3.SSE 3 支持不对齐访存,处理虚数运算的复杂算术指令和水平加、减操运算指令.2006 年,Intel 发布了新一代的 SSE 4 指令集,总共新增了 54 条指令,其中 SSE 4.1 包括 47 条指令,SSE 4.2 包括另外 7 条指令.SSE 4.1 率先在 Penryn 中出现,改进了插入、提取、寻找、离散、跨步存取等操作,用于加速绘图、3D 游戏、视频等多媒体程序.SSE 4.2 率先在 Nehalem 中出现,加入了处理字符串文本和面向应用的优化指令.此阶段 SIMD 自动向量化编译主要利用移位和重组指令处理非对齐和非连续访存的代码生成以及控制流向量化等.2008 年,Intel 在 Sandy Bridge 中支持 AVX(advanced vector extensions),向量寄存器长度由 SSE 的 128 位扩展到 256 位,AVX 增强了数据重排和灵活的不对齐内存地址访问.2011 年又提出 AVX 2 并在 Haswell 中支持,AVX 2 融合了乘加操作、256 位跨通道数据重排、gather 指令和寄存器间的广播指令.AVX 2 指令集引入了对 256 位整数向量指令,极大地提高了 Haswell 处理器的整数运算能力.2011 年,Intel 在 Knights Corner 中提出了 IMCI(initial many core instructions),向量寄存器长度扩展到 512 位,IMCI 支持 gather/scatter 操作,添加了写掩码寄存器,通过 swizzels 重新排序向量中的数据.当前 Xeon Phi 协处理器 Knights Corner 更多的应用在天河等高性能计算机中.2014 年 2 月,Intel 推出 512 位 SIMD 指令集 AVX-512,并准备应用在下一代 Xeon Phi 协处理器.目前 SIMD 自动向量化编译的主要研究问题为如何利用掩码寄存器,如何利用 gather 和 scatter 指令生成高效的向量程序以及如何根据程序的并行度面向 SSE,AVX 还是 AVX-512 生成向量程序.

不仅 Intel 在其处理器中支持 SIMD 扩展部件,很多其他厂商也在研发的处理器中添加 SIMD 扩展部件.1997 年,摩托罗拉在 G3 PowerPC 处理器上引入了 AltiVec 指令集.2000 年,AMD 在 Athlon 处理器上集成立类似于 SSE 的 3DNow!指令集.此外还有 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/MIPS-3D.SIMD 扩展部件最初仅用于多媒体领域,后来人们将 SIMD 扩展部件应用在数字信号处理器(digital signal processor,简称 DSP)和高性能计算机中.IBM 的高性能计算机 BlueGene/L 中含有 SIMD 扩展部件.国产神威蓝光超级计算机中的申威处理器也含有 SIMD 扩展部件[72].国产处理器龙芯[100]以及国产数字信号处理器 Matrix[101]和 BWDSP[102]也都含有 SIMD 扩展部件.表 1 列出了部分主流处理器厂商的处理器型号及 SIMD 扩展指令集名称和特征.

Table 1 Processors with SIMD extension

表1 带有 SIMD 扩展部件的处理器

Vendor	Motorola	DEC	SGI	Intel					Sony
Processor	G4	Alpha	MIPS V	Pentium		Core			Cell
Extension	AltiVec	MVI	MDMX	MMX	SSE	AVX	IMCI		AltiVec
Reg. size	128bits	64bits	64bits	64bits	128bits	256bits	512bits		128bits

Vendor	Sun	HP	AMD	ARM		IBM		国防科学技术大学	中国科学院
Processor	SPARC v9	PA-RISC	Athlon	ARMv6	PPC970	P6	BG/L	Matrix	Godson
Extension	VIS	MAX-2	3DNow!	NEON	VMX	VMX			
Reg. size	64bits	64bits	128bits	128bits	128bits	128bits	256bits	1 024bits	256bits

如今 SIMD 扩展部件不仅用于加速多媒体程序,也可用于加速数字信号处理程序和科学计算程序[1−7].基于 SIMD 扩展部件的向量化已成为程序并行的重要手段之一.SIMD 自动编译技术的发展和硬件结构之间的发展变化息息相关.随着应用领域的变化,SIMD 扩展部件的结构也在不断调整变化.这些变化给自动编译造成了新的挑战.此外,一些自动编译难以解决的问题也需要硬件来支持.SIMD 扩展部件的向量长度由最初 SSE 的 128 位到现在的 Matrix 的 1 024 位,支持的数据类型包括整型、浮点和按位的逻辑运算,指令集也在不断丰富.与适合 SIMD 多媒体扩展的多媒体应用相比,科学计算程序等其他应用领域的程序具有如下特性:计算规模大,需要进行过程间的向量化性能优化;数据访问模式不规则且可能存在大量的边界访问,所造成的数据访问不连续与不对齐问题极大地影响了向量化代码性能;存在复杂的数据依赖和控制依赖,使得向量化时依赖关系的判断变得非常复杂,很多情况下也阻碍了向量化发掘;访存带宽的需求增加,对向量寄存器文件产生了更大的压力;以上这些问题不仅给 SIMD 自动向量化编译带来新的挑战,也在改变着 SIMD 功能单元的结构.因此,为了使得

SIMD 扩展部件能够在科学计算等领域发挥作用,需要更深入地研究 SIMD 自动向量化编译.

本文第 1 节比较 SIMD 扩展部件与传统向量机以及超标量和超长指令字体系结构的特点,说明 SIMD 扩展部件的优点,列举面向 SIMD 扩展部件的手工向量化和自动向量化,给出 SIMD 自动向量化编译优化研究的内容.第 2 节~第 4 节详细介绍发掘方法、数据布局以及多平台 SIMD 扩展部件编译优化的研究进展和相关优化算法.第 5 节给出 SIMD 编译优化未来的发展趋势.第 6 节总结全文.

1 引 言

1.1 体系结构对比

虽然传统向量机和 SIMD 扩展部件都属于单指令多数据结构,但是在体系结构上存在着巨大差异.传统的向量机运算多使用很深的向量流水线来实现,而 SIMD 扩展部件采用短向量并行.传统的向量机在存储系统和互联结构上都针对向量处理进行了专门的设计,而 SIMD 扩展部件只是在通用处理器上附加了一个功能部件.SIMD 扩展部件访存需要先经过数据 Cache,而传统向量机可直接访问内存,寄存器和内存之间没有数据 Cache.SIMD 扩展部件只能将对齐并且连续的数据加载到向量寄存器,所以针对传统向量机的向量化算法并不能简单地应用于 SIMD 扩展.向量机编译时需要创建向量操作掩码等复杂问题,这些问题在后来的超标量和超字长结构(very long instruction word,简称 VLIW)体系结构中得到解决.VLIW 处理器可以在一个周期发射多条指令,每条指令可以对应不同功能部件上的操作.虽然超标量和 VLIW 结构可以达到向量执行的速度,但是其对指令 Cache 带宽和数据 Cache 带宽的要求都非常高.与超标量和超字长指令结构相比,SIMD 扩展部件的优势包括:一是并行成本相对较低;二是多媒体扩展指令不仅减轻了指令预取部件的压力,还减轻了指令 Cache 的压力;三是多媒体扩展的访存方式更加高效;四是控制逻辑更为简单而且功耗低.此外,与共享内存和消息传递并行结构相比,SIMD 扩展部件不需要增加同步和通信的开销.但 SIMD 扩展部件也有不足之处:一是目前存在的扩展部件的向量寄存器长度都是固定的.如果向量寄存器长度过长而循环迭代次数或基本块内同构语句条数较少,则程序不能被向量化.如果向量寄存器长度过短而程序有很好的向量并行性,则程序不能获得更大的加速效果;二是当程序没有向量并行性时,SIMD 扩展部件就处于空闲状态,造成硬件资源的浪费.

1.2 使用途径

SIMD 扩展指令发掘方法分为两种:一是手工向量化;二是自动向量化.手工向量化包括使用嵌入式汇编,Intrinsic 函数,C++类库扩展,SIMD 库函数等[8−10].手工向量化都是通过内嵌手写的汇编代码或编译器提供的内函数来添加 SIMD 指令.自动向量化即利用编译器分析串行程序中控制流和数据流的特征,识别程序中可以向量执行的部分,将标量语句自动转换为相应的 SIMD 向量语句.手工向量化出现在 SIMD 扩展部件的早期,针对指令集较小核心代码不复杂的情况.但是,随着 SIMD 扩展指令集的复杂化,所面向的应用也远远超出了原来的多媒体范畴,理论上虽然手工向量化能够实现最高程度的向量化,但由于不同处理器提供的 SIMD 扩展指令集各不相同,不同应用程序特征也复杂多变,程序员不仅要掌握程序的结构特征和数据布局,还要熟悉目标处理器底层结构和指令集的特点,所有这些都增加了程序员的编程难度和工作量,同时也使得 SIMD 向量化代码的可读性较差,可移植程度较低,并难以进行继续优化.但人们还是一直开发在一种程序员不需要考虑硬件结构的 SIMD 编程语言[11,12].与手工向量化相比,自动向量化不需要程序员考虑过多细节,通过自动向量化工具来对发掘程序中可向量执行的代码,并自动生成针对目标处理器 SIMD 扩展的向量程序,减轻了程序员的工作负担,因此自动向量化逐步成为当前 SIMD 向量化程序的主要方式.

OpenMP 从版本 4.0 开始提供了 SIMD 的编译指示,而 OpenCL 和 OpenACC 从最初标准就支持 SIMD.Intel 的 cilk+也支持 SIMD.OpenMP 和 OpenACC 是通过添加编译指示的方式发掘程序的 SIMD 并行性.在利用 cilk+和 OpenCL 编程时,程序员无需关心底层提供的 SIMD 扩展部件结构和指令集等问题,只需要把程序中存在的并行性表达清楚.这几种使用 SIMD 扩展部件的共同特点都是对编程人员透明.优点就是编程人员不需要考虑 SIMD 扩展部件提供的指令,便于发掘.缺点就是不能从指令的角度发掘程序存在的向量并行性,导致不能充分

发掘程序的 SIMD 特性.此外,OpenMP,OpenCL 和 OpenACC 等都是在发掘程序的任务级并行.与 SIMD 并行相比,它们都属于粗粒度并行.OpenMP 等提供的 SIMD 编译指示能够指导编译器在实现线程级并行的同时发掘程序的 SIMD 并行.

1.3 相关研究

很多科研机构和大学开发了 SIMD 向量化编译器,并对其中的关键技术进行了研究.Cheong 等人在 SUIF 上开发了针对 SUN 公司的 VIS 指令集的编译优化器,Krall 和 Lelait 也在 SUIF 上开发了面向 VIS 多媒体指令集的自动向量化工具,Sreraman 和 Govindarajan 也基于 SUIF 开发了一个针对 Intel MMX 指令集的向量化编译器[13].Jiang 和 Zhu 分别基于 Agassiz(Minnesota 大学,Purdue 大学和复旦大学联合开发的 C/Fortran 高性能编译器)开发了针对 Intel MMX/SSE 指令集的向量化编译器.赵荣彩等人面向申威处理器开发了自动向量化工具 SW-VEC[72].GNU 的 GCC 和 Open64 编译器以及 Path EKO 的 Pathcc 编译器也都增加了自动向量化功能.

许多提供 SIMD 扩展部件的处理器厂商都开发对应的自动向量化编译器或者在编译器中集成了自动向量化功能.Crescent Bay Software 扩展了 VAST 编译器以支持自动生成 AltiVec 扩展指令集的代码.Portland Group 在 PGI Workstation Fortran/C/C++编译器中提供了对 SSE/SSE2 指令集的支持.Codeplay 发布了能够支持所有 X86 扩展指令的 VectorC 编译器.Intel 在 C/C++/Fortran 编译器中也针对 MMX/SSE/AVX 提供了自动向量化支持.Alexandre 等人开发了 CELL 的优化编译器,CELL 处理器中有 128 位的 SIMD 数据支持,支持浮点和整型的 SIMD 操作.IBM 公司为 BlueGene/L 开发了 xlc 编译器.国防科学技术大学基于 GCC 开发了面向 Matrix 的自动向量化编译器[101].中国科学院计算技术研究所对龙芯、申威等国产处理器也做过 SIMD 编译优化方面的工作.BWCC 是基于 Openimpact 的面向 BWDSP 的编译器,也含有自动向量化功能[102].

虽然经过十几年的发展,但是面向 SIMD 扩展部件的自动向量化对程序的加速效果并不理想.其原因包括 3 个方面:一是硬件原因,以 Intel 公司的 SIMD 扩展指令为例,实现 1 个 16 位运算的向量化,所需的执行时间是 8 个 16 位运算的时间的 1/2,而不是 1/8[14];二是程序结构的原因.首先程序本身计算访存比的约束,其次程序分为可向量化部分和不可向量化部分.受阿姆达尔法则(Amdahl's law)的约束,加速比还受限于应用程序中可向量化代码的比例.当程序的不可向量化部分过多时加速效果依然较差;三是向量发掘方法的限制,由于程序分析和优化技术能力的限制,并非程序中所有可向量化的部分都被发掘出来.

SIMD 自动向量化编译流程大致分为 3 部分,分别是发掘、优化和代码生成,如图 2 所示.发掘和优化都位于编译的 LNO 阶段.发掘就是如何识别生成出 SIMD 指令,包括面向循环、基本块和函数等方法,同时解决控制依赖对发掘的影响.为了解决不对齐和不连续的内存访问引入许多辅助指令,导致额外开销.优化阶段首先减小辅助指令的开销,同时考虑数据局部性、寄存器重用等工作.代码生成位于编译的 CG 阶段.代码生成时需要考虑平台支持哪些数据类型和向量运算.面向特定体系结构的向量化代码存在可移植性差等不足,因此面向多平台的向量化也是研究的一个重要方面.

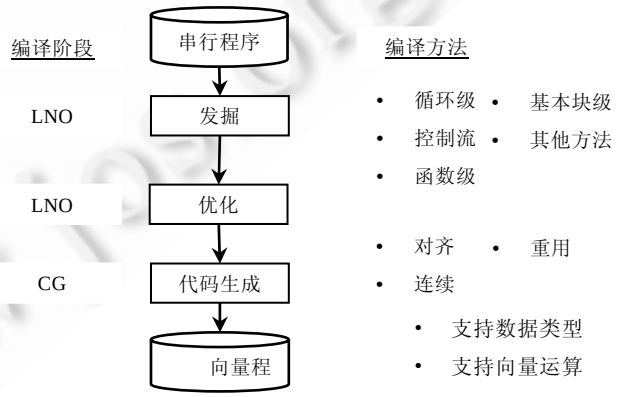


Fig.2 Compiling flow of auto-simdization

图 2 SIMD 自动向量化编译流程

2 发掘方法

SIMD 扩展部件可在不同的粒度向量化,包括面向基本块内向量化、面向最内层循环或者循环嵌套的向量化以及面向函数级别的向量化.最初面向 SIMD 向量化的循环级方法是从传统向量化方法引入的,它们都是求

解依赖图的强连通分量确定向量化语句的执行顺序.Larsen 等人首先提出了超字并行向量化方法,该方法的发掘对象为基本块而不是循环.由于科学计算程序中大量控制流语句的存在,使得 SIMD 扩展部件引入了选择指令 select.发掘 select 指令的方法同样采用传统向量化中的 if 转换思想.当编译阶段的依赖关系等信息不能判断向量化的正确性时可以借助交互方式.此外还有指令级并行、模式匹配以及多粒度向量化等方法.

2.1 循环级

面向 SIMD 扩展部件的循环级向量化方法来自传统向量机,包括预分析、依赖分析、类型转换和代码生成这 4 部分.预分析就是排除不能向量化的循环形式,如循环内含有函数调用、跳转语句等.依赖分析[16]就是根据循环内的数据依赖关系构造语句依赖图,在语句依赖图上求解强连通分量,不在强连通分量的语句就是可以向量执行的语句.由于只有真依赖影响向量化的执行正确性,而反依赖和输出依赖都是可以被消除的依赖.因此面向 SIMD 扩展部件的向量化中,只有语句依赖图上所有真依赖环上的依赖距离之和小于或等于向量化因子不可向量化,其中向量化因子表示一次向量操作同时运算标量数据的个数.其他的依赖形式都可以通过循环分布、节点分裂、标量扩展等程序变换技术破除其对向量化的影响.类型转换就是将循环体内的语句由标量类型转换为向量类型.代码生成时首先将向量类型语句三地址,转换为含有 SIMD 指令的向量化代码.代码生成时需要考虑目标平台的 SIMD 扩展指令.

最内层循环向量化实现直观,代价小.但当最内层循环存在依赖环、归约或者数组引用相对于最内层循环索引不连续时,内层向量化代价较大或者无法向量化.循环交换通过将某个外层循环交换至最内层进行向量化[17,18],但是循环交换只能针对完美嵌套循环.针对含有复杂的数据依赖而不能实施循环交换或者分裂的循环嵌套,文献[19]提出了沿着内层循环索引的垂直修剪和沿着外层循环索引的水平修剪,成功向量化了起泡排序算法.面向 SIMD 扩展部件的外层循环向量化方法能够很好地发掘非完美嵌套的向量并行性[20].考虑到循环交换、循环剥离,循环分布等循环变换因素以及影响向量化的对齐、连续、类型转换、迭代次数等因素,多重循环的最大向量化收益研究很多[21−25].多面体模型利用矩阵表示多重循环的迭代空间,统一了各种循环变换的数学表达,是发掘多重循环向量化的有效方法.文献[21]建立一种考虑对齐、连续、类型转换、迭代次数等影响向量化的收益模型,利用收益分析模型计算每层循环向量化的收益,并选择收益最大的向量化方案.文献[22]利用多面体模型全局考虑了多重循环的并行化、向量化、数据重用等问题.文献[23]利用机器学习指导 SIMD 代码生成的循环变换.文献[24,25]考虑连续性和循环嵌套在 R-Stream 编译器中的实现,但是并没有考虑后端的代码生成.

面向最内层循环的传统向量化方法技术成熟,不论是 ICC 等商用编译器以及 GCC,Open64 等开源编译器一直都保留传统向量化模块.多面体模型是解决多重循环向量化的重要手段,但是多面体模型要求循环嵌套满足一定的条件,如不能含有复杂控制流,函数调用等.因此,含有复杂控制流和函数调用的多重循环向量化方法是未来要解决的重要问题之一.此外,外层循环向量化会影响到 cache 的命中率,特别是数组访存不连续时,如何解决好外层循环向量化时的 cache 命中率问题也需考虑.

2.2 基本块级

面向基本块的向量化常常提到打包、解包操作.包是一个同构语句的集合.将多个同构语句组成包的过程叫做打包,相反则称为拆包.Larsen 等人首先提出了面向基本块的超字并行(superword level parallelism,简称 SLP)向量化方法[26].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.对于图 3(a)中的串行程序,SLP 首先利用相邻地址访存的原则,生成{b=a[i+0];e=a[i+1]}和{e=a[i+1];h=a[i+2]}两个初始的 *pack* 如图 3(b)所示.接着按照 *DU* 链扩展 *pack* 得到图 3(c)所示的结果,{b=a[i+0];e=a[i+1]}定义的标量{b,e}在{d=b+c;g=e+f}中被使用,{e=a[i+1];h=a[i+2]}定义的标量{e,h}在{g=e+f;k=h+j}中被使用,所以根据 *DU* 链扩展出来两个新 *pack* 为{d=b+c;g=e+f}和{g=e+f;k=h+j}.然后按照 *UD* 链扩展 *pack*,结果如图 3(d)所示.{d=b+c;g=e+f}使用的标量{c,f}在{c=5;f=6}中被定义,{g=e+f;k=h+j}使用的标量{f,j}在{f=6;j=7}中被定义,所以根据 *UD* 链扩展出来两个新 *pack*{c=5;f=6}和{f=6;j=7}.接下来是 *pack* 组合.

组合的方法是如果 *pack A* 中的右语句和 *pack B* 中的左语句是同一个语句时,就将 *pack A* 和 *pack B* 组合成一个 *pack*,一条语句可以出现在两个 *pack* 中,但不能同时为左语句或者同时为右语句.{*b*=*a*[*i*+0];*e*=*a*[*i*+1]}中的右语句和{*e*=*a*[*i*+1];*h*=*a*[*i*+2]}中的左语句相同,组合后形成 *pack*{*b*=*a*[*i*+0];*e*=*a*[*i*+1];*h*=*a*[*i*+2]}.{*d*=*b*+*c*;*g*=*e*+*f*}中的右语句和{*g*=*e*+*f*;*k*=*h*+*j*}中的左语句相同,组合后形成 *pack*{*d*=*b*+*c*;*g*=*e*+*f*;*k*=*h*+*j*};同样,{*c*=5;*f*=6}中的右语句和{*f*=6;*j*=7}中的左语句相同,组合后形成 *pack*{*c*=5;*f*=6;*j*=7},组合结果如图 3(e)所示.最后按照依赖关系调度包,形成发掘结果如图 3(f)所示.

为使循环体内含有足够多的地址连续的同构语句,SLP 在打包前先做循环展开.包转置可以解决内层循环携带依赖,外层循环可以向量化但是数组引用相对于外层循环索引不连续的问题[27].SLP 是一种启发式算法,并不能获得最优的向量化方案.BS 向量化方法去除了 SLP 中按照定义-使用链和使用-定义链扩展的限制,采用动态规划寻求基本块内最优的向量化方案[28].代价模型也是指导 SIMD 指令选择的方法[29].打包冲突图提供了更多包重用的机会,在包调度时决定包的执行顺序和包内语句的顺序,以减少拆包的次数[30].采用多元决策图可以有效的表示包内的数据,减少拆包的次数[31].

面向基本块的向量化又叫直线型(straight-line)向量化,要求基本块内有足够的并行性,否则会有很多向量和标量之间的转换影响向量化效果.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.SLP 首先提出了面向基本块向量化,其后出现的面向基本块的向量化方法都是对 SLP 的改进.目前面向基本块的 SIMD 向量化方法已经研究得很深入.

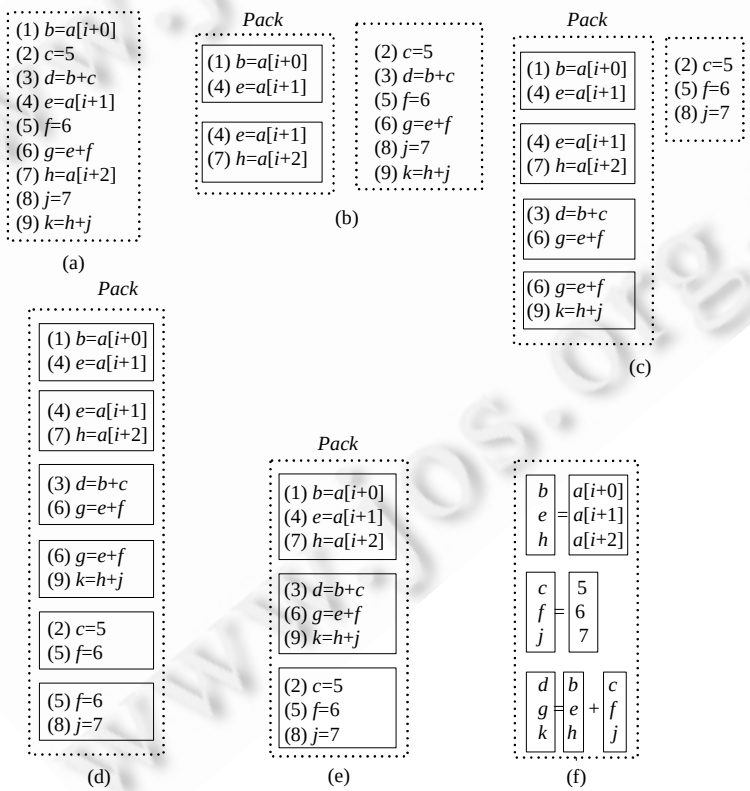


Fig.3 Various phases of SLP

图3 SLP发掘向量代码的各个阶段

与发掘迭代间并行性的循环级向量化方法相比,面向基本块的向量化方法是在发掘迭代内的并行性,而基于循环展开的 *loop-aware* 向量化则既发掘迭代内并行又发掘迭代间并行[32].设循环展开因子为 *uf*,向量化因子

为 *vf*.当展开因子等于向量化因子即 *uf*=*vf* 时是循环级并行,当 1<*uf*<*vf* 时为 *loop-aware* 向量化,当 *uf*=1 即不用循环展开时则为基本块内向量化.以图 4(a)中循环为例详细说明基本块级、循环级和 *loop-aware* 这 3 种向量化方法.假设向量化因子 *vf*=4.图 4(b)中将数组 *b*[6*i*+0]到 *b*[6*i*+3]直接组成一个向量运算基本块级向量化,此时 *unroll*=1.图 4(c)中 *unroll*=2,展开 2 次循环体内有 12 条语句后再利用基本块向量化,将数组 *b*[6*i*+0]到 *b*[6*i*+3]组成一个向量,将数组 *b*[6*i*+4]到 *b*[6*i*+7]组成一个向量,将数组 *b*[6*i*+8]到 *b*[6*i*+11]组成一个向量,此时为 *loop-aware* 向量化.图 4(d)中 *unroll*=4,展开 4 次后循环体内有 24 条语句,如将同一条语句展开的各个语句组成向量则为面向循环的传统向量化方法,但也可以像图 4(d)按照地址相邻的原则打包,图 4(d)中的打包向量方法效率更高.选择哪种向量化方法主要根据程序的特点.

```
for (i=0; i<n; i++){
b[6i+0]=b0;
b[6i+1]=b1;
b[6i+2]=b2;
b[6i+3]=b3;
b[6i+4]=b4;
b[6i+5]=b5;
}
```

(a) 标量

```
vb0={b0,b1,b2,b3};
for (i=0; i<n; i++){
b[6i+0:3]=vb0;
b[6*i+4]=b4;
b[6*i+5]=b5;
}
```

(b) 基本块级 *unroll*=1:

```
vb0={b0,b1,b2,b3};
vb1={b4,b5,b0,b1};
vb2={b2,b3,b4,b5};
for (i=0; i<n; i+=2){
b[12*i+0:3]=vb0;
b[12*i+4:7]=vb1;
b[12*i+8:11]=vb2;
}
```

(c) *loop-aware*级 *unroll*=2:

```
vb0={b0,b1,b2,b3};
vb1={b4,b5,b0,b1};
vb2={b2,b3,b4,b5};
for (i=0; i<n; i+=4){
b[24*i+0:3]=vb0;
b[24*i+4:7]=vb1;
b[24*i+8:11]=vb2;
b[24*i+12:15]=vb0;
b[24*i+16:19]=vb1;
b[24*i+20:23]=vb2;
```

(d) 循环级 *unroll*=4:

Fig.4 Comparison of the different vectorization approaches

图 4 各种向量选择方案

2.3 函数级

不论是面向循环还是面向基本块的向量化都是在标量过程内发掘,即该函数的参数为标量.函数级向量化即函数的参数为向量,返回值也为向量.它需要将多次标量函数调用转换为一次向量函数调用,连续多次函数调用一般仅出现在循环体内.如图 5 所示,图 5(a)是标量函数,图 5(b)是向量函数.函数级向量化涉及到过程间分析等复杂编译问题,当前仅能向量化简单函数[33].可向量化的函数必须满足 3 个条件:一是函数体内没有循环;二是数组下标不允许为非常数;三是函数的形式参数只允许为标量形式.首先将多出口函数改为单出口函数,然后将标量类型改为向量类型,向量化函数体内的控制流,函数体内可扩展的函数如不能向量化则复制为 *vf*(*vf* 为向量化长度)个拷贝.另一向量化函数的方法是利用静态单赋值 SSA[34],基于 SSA 形式将函数生成有效的标量和向量混合的代码.

函数级的向量化可以取得类似任务级并行的加速效果,具有很好的发展前景.但目前仅能向量化非常简单的函数,复杂函数的向量化研究得不够深入.主要是因为函数级向量化涉及到过程间分析和别名分析,而过程间分析和别名分析是编译中的难题,因此当函数体内存在函数调用时,如何保证向量化结果的正确性是困难的.此外函数级向量化还需要处理复杂的控制流.

```
float f(float a, float b) {
float r;
if (a>b)r=a+1;
else r=a  1;
return r;
}
```

(a)

```
__m128 f_sse(__m128 a,__m128 b){
__m128 mask=_mm_cmpgt_ps(a,b);
__m128 s=_mm_add_ps(a,_mm_one);
__m128 t=_mm_sub_ps(a,_mm_one);
__m128 r=_mm_blendv_ps(mask,s,t);
return r;
}
```

(b)

Fig.5 Whole-function vectorization

图 5 函数级向量化

2.4 控制依赖

控制依赖是发掘 SIMD 向量化的一道难题.向量化控制依赖有两种方法:一是直接处理控制依赖;二是将控

制依赖转换为数据依赖.直接向量化控制流是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[35].控制依赖转换为数据依赖就是采用 if 转换.If 转换最先应用于向量机中[36−38].控制流语句在科学计算程序中大量存在.为了更好地向量化科学计算程序,SIMD 扩展部件引入了处理控制流的 select 指令[39].目前支持向量条件操作的 SIMD 扩展部件体系结构有 AltiVec,DIVA,SSE 等[40].超字条件分支(branches-on- superword-condition-codes,简称 BOSCCs)指令可绕过具有相同预测谓词的语句,当含有复杂控制流的 if 嵌套,BOSCCs 指令能够生成一长串没有 if 嵌套的代码,提高 SIMD 代码的性能[41,42].图 6(a)所示为串行程序,利用 if 转换和 select 指令可以转换为图 6(b)所示的向量化程序,但当 *v_pT* 为假时仍然会执行 select 大大降低了程序的执行效率,因此在图 6(c)中插入 BOSCCs 指令当 *v_pT* 为假时直接绕过 select 指令.程序结构等价变换可以向量化含有出口分支的循环[43].

利用 if 转换不能很好的向量化 if 嵌套,如何高效地向量化 if 嵌套有待研究.利用静态单赋值(static single assignment,简称 SSA)解决控制流问题是一个值得研究的问题,特别是当循环嵌套内含有控制流,如何利用多面体模型和 SSA 共同解决其 SIMD 发掘是一个研究问题.如何解决因为控制依赖构成的多个基本块间的数据重组问题也是一个值得研究的问题.

```
for (i=0;i<1024;i++)                for (i=0;i<1024;i+=4){                for (i=0;i<1024;i+=4){
  if (fore[i]!=255)                   v255=(255,255,255,255);             v255=(255,255,255,255);
    back[i]=fore[i];                  v_pT=fore[i:i+3]!=v255;             v_pT=fore[i:i+3]!=v255;
                                      back[i:i+3]=select(back[i:i+3],     branch-on-none(v_pT) L1;
                                                         fore[i:i+3],v_pT);back[i:i+3]=select(back[i:i+3],
                                                                                     fore[i:i+3],v_pT);
                                                                          L1;        }
```

(a) 串行　　(b) 利用select指令　　(c) 利用select指令和BOSCC指令

Fig.6 Vectorization in the presence of control flow

图 6 控制流向量化

2.5 交互向量化

为保证正确性,编译器往往采取相对保守的优化手段,并不能获得最优的向量化程序.当前的 ICC,GCC 等编译器仅报出源程序向量化成功否,对于各种阻碍向量化的因素并没有做过多的解释分析,也没有提示用户如何解决这一问题,这给程序员调整代码带来了困难.交互工具为用户提供程序的控制流、调用关系、依赖关系等信息[44,45],用户将编译信息与编程经验结合提高程序的性能[46−48].交互分为静态交互和动态交互.

静态交互就是通过编译指示弥补编译器分析和优化的不足,帮助编译器排除影响自动向量化的因素以获得更高效的向量化代码.向量化编译指示分为以下几类:一是强制性编译指示,包括强制向量化与强制不向量化编译指示;二是数据访问的对齐性编译指示,包括不对齐和对齐编译指示;三是消除数据依赖关系编译指示,包括消除数据依赖关系图的冗余边和冗余回边编译指示;四是各种向量化相关的编译优化指示,包括函数内联、设定循环展开因子、循环不变量外提等优化编译指示.

动态交互就是通过插桩方式获得程序的剖面信息并利用剖面信息指导调优[49−51].向量化的动态交互如图 7 所示,首先是插桩后编译运行,收集对齐、连续、依赖等影响向量化的编译信息;二是利用反馈信息调整程序结构或插入向量化编译指示指导编译器向量化.

目前,交互向量化调优的研究还不够深入.首先需要引入更多的编译指示,并像 OpenMP 编译指示那样规范向量化编译指示使其满足各个平台.同时需要给出更多的反馈信息指导向量化,如给出条件分支语句中 false 执行的比重,然后即可判断是否生成 BOSCC 指令.最后,将静态的编译指示和动态的反馈信息结合起来能够更好的向量化程序.

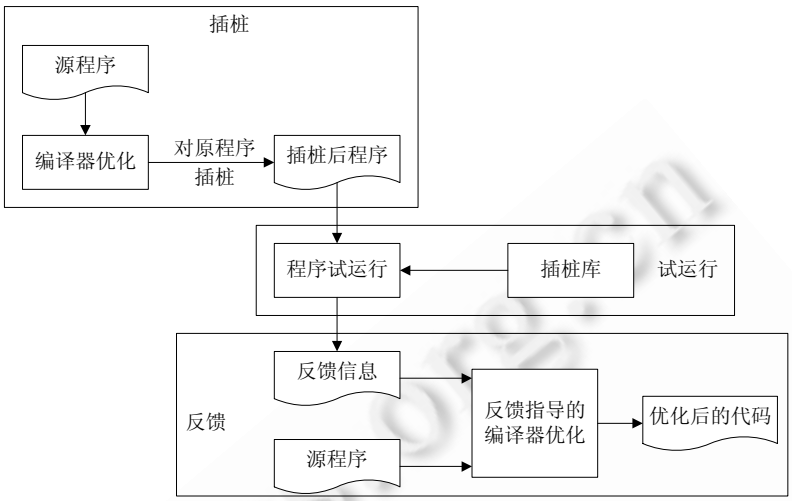


Fig.7 Interactive program analysis based on profiling

图 7 基于反馈的程序调优

2.6 其他方法

模式匹配向量化方法[52]主要针对一些应用程序中的特殊运算操作,如归约操作、饱和运算、乘加运算、条件运算等,通过模式匹配的方法来生成与设定的特殊操作相对应的扩展指令.

多层次 SIMD 向量化更能充分发掘程序的并行性.基于流图和代价模型的向量化方法 MacroSS 分为 actor 内,水平 actor 间和垂直 actor 间 3 种,其中 actor 是一个自治、独立的计算单元[53].Actor 内的向量化将连续串行执行的程序转换为数据级并行执行.垂直向量化融化多个可向量化 actor 建立一个更大可向量化 actor,减少标量到向量和向量到标量的转换.水平向量化要求 actor 是同构的,并且在启动和同步之间不能将它们合并为垂直 actor.基于代价模型和流图结构选择哪个层次 SIMD 向量化.

投机向量化方法能够消除静态编译存在的不确定因素[54].投机执行前拷贝指令保存在一个影子寄存器中并对每个访存指令标号.投机执行时将结果保存到缓冲区中,如果硬件检测到序号高的指令执行在序号低的指令之前并且它们访存迭交,则将影子寄存器中的值拷贝到工作集中,否则将执行结果写入内存.

向量化程序运行时,标量部件处于空闲状态.选择性的生成向量指令充分的利用标量和向量部件提高程序性能[55].SIMD 部件在发掘程序的数据级并行性失败时,可重构的 SIMD 扩展部件可以充分发挥指令级并行[56].并行基本块内有相同操作和数据流的子图,可将指令级并行转换为数据级并行[57].

SIMD 扩展部件都是静态同构的.动态配置可使槽位之间共享昂贵的硬件资源,异构可使不同的槽位有不同的运算功能.动态重构部件不仅像传统 SIMD 部件也可像一个 VLIW 部件.Libra 是动态异构的 SIMD 加速部件[58].多个 SIMD 扩展部件组成的 MSMD 硬件结构可动态或者静态划分多控制流,不同分支的 SIMD 片段可以并行执行获得类似 MIMD 的性能[59].向量化线程体系结构能够获得传统向量处理器或者 VLIW 的代码等多种并行形式[60].

与串行执行相比,SIMD 扩展部件只是将多个操作并行执行,能耗似乎没有降低.但是利用一套整合到编译器和模拟器中的指令级能耗代码模型表明 SIMD 扩展部件能够有效地降低功耗节约能源[61,62].设计出低功耗 SIMD 扩展部件[63]和面向低功耗的 SIMD 代码生成[64]都是研究的重要方面.

3 数据布局

与标量部件相比,SIMD 扩展部件的收益来自于向量运算和向量访存.因此利用更少的时钟周期组成向量是提高 SIMD 整体效益的关键.但是由于早期的 SIMD 向量存取指令只能从内存中连续对齐的存取数据,因此当

程序中存在不对齐或不连续内存引用时需要移位或者重组等辅助指令才能组成向量.减少辅助指令的数量和提高辅助指令的效率是增加程序 SIMD 向量化收益的关键问题.此外,cache 重用和向量寄存器重用等数据布局也是影响向量化效果的重要因素.

3.1 对 齐

多媒体扩展指令集所引起的一个重要问题是数据的对齐.内存访问是对齐的当且仅当 $A \bmod n=0$,其中,A 内存地址,n 为访存数据的字节数.向量数据访问的对齐实质是指被访问数据与向量寄存器(或类似硬件部件)的对齐.多媒体扩展指令集希望这些数据是对齐的,这样它们就可以快速读取这些数据.不对齐的数据访问因需要额外的开销而抵消部分向量化的收益.处理好程序在向量化时不对齐访存问题是提升向量程序性能的途径之一[65].解决不对齐访存问题有 3 种方法:一是通过多次对齐访存,然后移位或者重组;二是通过循环变换;三是硬件支持.

移位或重组实现不对齐访存首先需要 2 次对齐读,将数据加载到向量寄存器中,然后向量寄存器之间拼接获得目标向量.图 8 所示为利用 shift 指令实现重组的流程.文献[66]利用移位指令解决非对齐内存访问,并提出了 zero-shift 策略、eager-shift 策略、lazy-shift 策略、dominant-shift 策略这 4 种移位策略.Zero-shift 策略表示立即移到对齐位置.eager-shift 策略移位每个 load 移位到 store 的对齐偏移.Lazy-shift 策略是尽可能晚的插入移位操作.dominant-shift 策略降低移位的次数通过移动寄存器至主导的数据偏移.文献[67]利用重组指令 shuffle 实现不对齐访存.文献[73]在 cache 中添加了“选择-移位”部件,使得重组在 cache 内而不是寄存器内实现.

在循环变换解决不对齐访存方面,文献[71]利用循环剥离可以将不对齐数据剥离出去,文献[72]利用数组填充解决多维数组对齐的问题.MMX 指令集中首先提供不对齐的访存指令,目前几乎所有 SIMD 平台都有不对齐访存指令.针对不对齐访存还有一些其他的研究,如编译时认为不对齐但运行时对齐的问题在文献[68]中解决,文献[69]研究了函数调用指针值传播对齐信息指导 SIMD 代码生成.文献[70]将输入表达式仅有两个不同对齐要求的问题转换为最小节点 S-T 分解问题,利用动态规划方法处理输入表达式中每个数组仅出现一次的对齐问题.

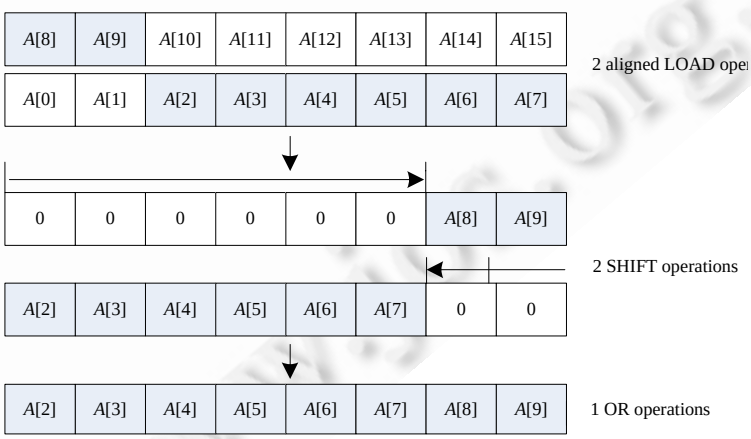


Fig.8 Implement unaligned load using shift

图 8 利用 shift 指令处理不对齐访存

目前静态的对齐分析已经研究的很深入,但是动态和过程间的对齐分析仍是需要重点解决的难题.虽然硬件实现的不对齐访存降低了 SIMD 编译优化的难度,处理好对齐访存仍是提高 SIMD 向量化性能的重要途径.

3.2 连 续

连续的向量访存不仅可以提高向量访存指令的效率,也可以提高向量寄存器中有效数据的比率.目前虽然一些平台提供的掩码寄存器来避免向量寄存器中无效或者非法的数据,但是处理好向量化时不连续访存问题

是提升向量程序性能的途径之一.与向量处理器不同,不是所有的 SIMD 扩展部件的内存访问机制中都存在聚集/分布(gather/scatter)操作,这使有些不连续内存访问必须经过编译提供的数据重组才能完成向量化的操作.数据重组操作通常会带来额外的开销,降低 SIMD 优化的效果.因此,如何有效地重组解决数据连续性的问题,对于 SIMD 编译优化来说显得尤为重要.不连续的访存处理方法有 3 种:向量重组、硬件支持和程序变换.

向量重组是指当目标向量中的所有元素不在同一个向量中时,需要通过两个或者多个向量之间组合得到目标向量,如图 9 所示.文献[80]首先利用重组处理不连续的内存访问.文献[81,82]都利用数据流图减少重组次数,以减少重组指令的数量.文献[83]统一表示程序中所有类型重组减少重组次数,以减少重组指令的数量.隐含数据拼凑能够合并重组和其他 SIMD 操作,提供了更多的优化空间[84].文献[72]用贪心法指导插入、提取的生成指令,实现向量重组.

不连续访存可通过硬件指令实现.SSE 平台提供存取内存中奇数位和偶数位的指令,提出了一种发掘步长为 2、4、8 等 2 的指数次循环的向量化方法[74].多索引寻址部件可同时从内存多个不连续的地址取数据构成向量[73].IBM 研发的数字信号处理器 eLite 硬件上支持不连续访存[75].文献[76]定义了新的拼凑指令以获得所有分层二维块的重组.文献[77]讨论了按位重组优化.对角寄存器[78]和矩阵寄存器[79]都可处理矩阵重组.

局部数据重组是指将不连续的数据拷贝到连续的临时数组中向量化后再复制回去,该过程代价较大[85].文献[72]利用过程内的数组转置解决连续性问题.间接数组访问导致连续、对齐和依赖关系等编译信息不确定,文献[86]同时发掘迭代内和迭代间的并行,提出了一种利用数据流图发掘间接数组访问的 SIMD 向量化方法.树和图的遍历没有并行性,文献[87]将多个遍历任务放到一起实现了树和图等不规则数据结构的 SIMD 向量化.

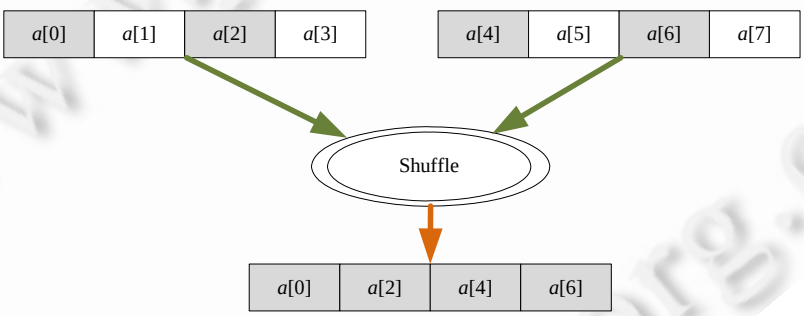


Fig.9 Implement stride memory access using shuffle

图 9 利用 shuffle 指令处理不连续访存

3.3 重 用

Shin 将向量寄存器视为编译器可见的 Cache 以增加向量寄存器的重用,不仅可以考虑时间局部性,还考虑了空间局部性[88,89].针对 iVMX 体系结构含有 4 096 个向量寄存器,文献[90]研究了向量寄存器的利用率问题.

如果循环中的数组引用相对于外层循环索引的跨幅较大,循环展开就有可能影响 cache 命中率.多媒体程序的特点之一就是大数据量,而且这些数据在时序上非常有规律,使得数据读取的可预测性很强.在进行缓存优化时应该对数据进行预取,以保证在需要的时候可以快速读取,减少延迟.很多多媒体指令集都提供了这类优化功能.所以,SIMD 编译优化的一个重要方面就是针对这个方面进行缓存优化.

4 多平台向量化

直接面向特定平台的 SIMD 向量化代码生成存在许多不足,如有新指令支持代码的调整难度大、可移植性差等.分阶段并行编译优化和虚拟向量是解决面向多平台向量化的两个方法[91].

分阶段并行编译优化推迟面向平台的代码生成和优化.为解决不同平台的对齐和归约等问题,Nuzman 等人 2006 年在 GCC 中间表示 GIMPLE 加入新的操作符和描述符,使其能够面向多平台生成自动向量代码[92].

Nuzman 等人 2011 年提出了分阶段自动向量化方法[93].该方法首先利用 gcc4cli CLI 后端[94]将 C 程序转换为含有的向量化操作的 CLI 字节码,然后利用扩展的 JIT 编译器 Mono 将 CLI 字节码映射到 SSE 和 AltiVec 等平台.当目标平台没有向量化部件时将 CLI 字节码转换回标量代码,整个流程如图 10 所示.基于 Jikes RVM 的面向多平台的动态向量化方法可运行在 Java 和.net 平台,首先将输入的字节码降级并做循环展开等各种优化,然后代码生成工具 BURS 利用模式匹配技术向量化中间表示并生成目标代码[95].体系结构描述语言 ADL 表达处理器 SIMD 扩展部件的属性,利用编译器可知函数 CFK 表达 SIMD 指令,开发一个 ADL 和编译器 CoSY 之间的接口.利用编译器 CoSY 实现了多平台的代码优化方法[96].

虚拟向量的引入可以使编译的前期阶段少受制于机器特征的限制,在编译的后期逐步将虚拟向量转换为实际向量.虚拟向量可解决对齐、不同数据长度和不同粒度的向量化开发[97].向量虚拟指令集一般包括内存操作、算术、逻辑、比较、选择操作以及重组操作[98].指令集覆盖了向量部件的所有静态信息,提供任意的向量长度和固定的向量长度,实现从 vector LLVA 到 RSVP,Altivec 和 SSE 体系结构的转换.

建立一个面向多平台的实现不同粒度的健壮向量化编译工具是未来的目标.该工具综合考虑不同向量长度、不同程序特征、各种各样的指令.

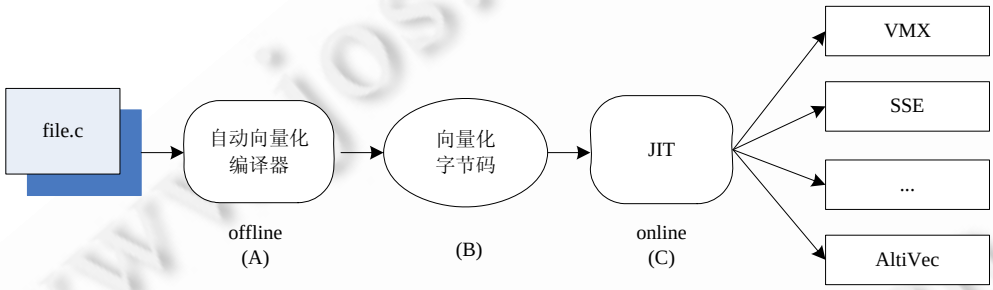


Fig.10 Split compilation flows

图 10 分阶段编译流程图

5 未来的挑战

目前面向 SIMD 自动向量化的循环变换、简单控制依赖、对齐和重组等很多规则问题都得到了很好的解决,这给 SIMD 向量化发掘提供了坚实的基础.针对单层循环的向量化方法也研究得很成熟.面向循环嵌套解决循环体内数据的对齐和连续问题、面向基本块的向量化方法、SIMD 指令和标量的指令级的并行、面向后端指令选择的方法都研究得很多.但在 Intel 至强处理器 5500 采用 Intel 的编译器 ICCV14.0 对 SPEC2006 中 30 个程序也仅 8 个有效果,最高加速比为 1.930,平均加速比为 1.151.加速效果不理想的原因包括 3 个方面:一是发掘方法的限制,当前向量化方法对含有复杂依赖关系和过程间调用的程序不能向量化,对含有不规则内存访问的程序也不能很好的向量化,这些原因导致程序 437.leslie3d,416.gamess, 410.bwaves,435.gromacs 等程序没有很好的向量化效果;二是需要更丰富的硬件支持,包括处理控制流、归约和类型转换等指令,如果 SIMD 扩展部件支持复数类型运算可大大提高 462.libquantum 的向量化效果;三是非计算密集型程序不可能有向量化效果,如 400.perlbench,403.gcc 等,预测程序的最大向量化加速比是值得研究的问题.因此我们认为未来 SIMD 扩展部件还需要在发掘方法,SIMD 硬件支持以及性能预测等方面值得深入研究.

5.1 多粒度向量化

当前向量化分为 3 个粒度:函数级、循环级和基本块级.面向循环和基本块的向量化方法已经很多,当前自动向量化编译器中的发掘方法主要是循环级和基本块级.函数级向量化涉及到过程间分析、复杂控制流等问题,因此函数级向量化难度很大.函数级向量化的第 1 个关键问题就是如何确定候选函数.首先判断函数被调用次数是否足够多.其次是如何确定函数级向量化的先后顺序.第 2 个关键问题就是如何对选定的函数向量化.函数向量化就是将函数参数和函数体由标量类型转换为向量类型的过程.当一个函数不仅需要有向量参数的过

程,同时还要存在标量参数的过程时,就需要通过过程克隆技术生成标量和向量两个版本的过程.过程克隆有两种应用情况,一是当函数有多个被调用点,但不是每个被调用点都能转为向量形式时,二是函数在被调用点存在尾循环,不能都转为向量形式,还存在标量标量形式.

针对一个程序,如何发掘不同粒度的向量化也是一直值得研究的问题.

5.2 不规则内存访问

不规则内存访问经常出现在程序中,如 SPEC2006 的 435.gromacs,444.namd,416.gamess 以及 CHAOS 中的 Moldyn 都含有不规则内存访问.投机并行是解决不规则内存访程序向量化的方法.投机并行通过运行时程序执行动态信息来消除静态编译存在的不确定因素,从而更精确地反映程序中的依赖关系,为实施重排序变换创造条件.当前投机并行在面向分布存储结构和共享内存结构以及指令级并行等方面的研究比较多,而面向 SIMD 扩展部件的投机并行研究很少.利用投机执行技术解决不规则程序的自动向量化是值得研究的一个重要方面.

5.3 更丰富的硬件支持

编译技术的发展和硬件结构以及应用领域的变化有着密切的关系.应用程序的需求催生出新的硬件结构,新的硬件结构的出现给编译技术带来了挑战.编译技术的不断发展又给硬件结构提出了更高的要求.如多媒体程序的需求产生了 SIMD 扩展部件,SIMD 扩展部件需要编译解决好不连续的访存问题,如利用数据重组解决不连续的内存访问.当利用编译技术或者现有硬件支持生成的程序效率较低或者不能实现时,则需要 SIMD 扩展部件提供不连续访存方面的指令支持,如 SSE 中提供的 extract_odd/extract_even 以及 KNC 中支持的 gather/scatter 等.当前由于 KNC 中的 gather/scatter 指令开销较大,又需要编译技术指导产生高效的 SIMD 向量代码.因此,编译技术的发展和硬件结构息息相关.目前从 SIMD 自动编译的角度出发,认为硬件应该给予以下支持:

(1) 当前所有的 SIMD 指令集都包括访存指令以及算术、逻辑、比较运算指令.但不是所有的指令集都支持不对齐、不连续的访存指令以及向量重组指令.指令集以及指令节拍和向量长度的不同都是导致不同平台对同一程序性能提升不同的原因.更丰富的硬件支持不仅可以提高程序向量化的程度而且使自动向量化发掘方法更为简单.有些平台支持向量化控制流、归约和类型转换等指令.硬件支持不对齐不连续的访存可以大大提高向量化程序的效率.

(2) 指令周期短、低功耗的 SIMD 指令是未来研究的一个重要方面.当前 SIMD 扩展部件都是静态同构的,这导致程序不能向量化时的硬件处于空闲状态.如将 SIMD 部件做成可重构的硬件,那么当程序适合向量执行时可加长 SIMD 部件长度,而不合适向量执行时可减小 SIMD 部件长度甚至为标量部件.单指令多操作多数据(single instruction mulptily operator multiply data,简称 SIMOMD)部件为向量中元素提供不同的操作,能够提高 SIMD 扩展部件的处理能力.

5.4 其他方面

预测程序向量执行的最大加速比是值得研究的问题.Kismet 是一个利用分层的关键路径收集并行数据的信息,然后根据目标平台和系统预测串行程序的并行加速比预测工具[99].如何利用 Kismet 工具预测串行程序的最大 SIMD 向量化加速比也是尚待解决的问题.

目前各大处理器厂商都在研制长度更大的向量寄存器,如现在寄存器长度由 SSE 的 128 位到 Matrix 的 1 024 位.但是加速效果并没有随着向量寄存器长度的增加而增大,有时甚至更低.NPB 测试集中 BT 在向量长度为 256 位的 AVX 平台加速比为 1.239,而在向量长度为 512 位的 KNC 平台加速比为 0.798.SP 在 AVX 平台的加速比为 1.182,在 KNC 平台的加速比为 0.980[104].出现这样结果的原因是多方面的.首先是硬件平台指令支持的差异有关.向量寄存器长度越长虽然理论加速比很大但是组成向量越难,特别是对不连续访存的程序来说,此外 KNC 中不支持向量除法等指令.因此增长向量寄存器长度的同时还应该增强向量重组和计算能力,这样才能更好地发挥长向量寄存器的优势.其次与程序固有的 SIMD 并行度有关.当程序的并行度小于向量化因子时,继续增长向量寄存器并不能提升性能,带有掩码的向量寄存器是解决这个问题的途径之一.或许利用 FPGA 实现长度可变的向量寄存器能够更好地发掘程序的 SIMD 并行性.此外,不同平台的向量寄存器数量也不相同.如

CELL 的 SPU 有 128 向量寄存器,Altivec 有 32 个向量寄存器.向量寄存器数量过少会导致寄存器压力过大从而影响向量化的性能,而向量寄存器过多又影响芯片的布局,因此向量寄存器的数量也是要研究的问题.

6 结 论

本文首先描述了 SIMD 扩展部件出现的背景和处理器厂商对 SIMD 扩展部件支持情况,介绍了 SIMD 扩展部件体系结构特点并比较了 SIMD 扩展部件与传统向量机,超标量和超字长结构等优越性,比较了手工向量化和自动向量化的优缺点,说明了为什么自动向量化为 SIMD 扩展部件发掘的主流,介绍了科研机构和大学以及处理器公司对自动向量化的支持.第 2 节~第 4 节介绍了 SIMD 自动向量化研究内容.总体来说,SIMD 编译优化要解决的主要问题是如何有效地把各种用高级语言编写的多种形式的代码转换成对应的 SIMD 指令和如何使生成的 SIMD 代码尽可能高效.从发掘方法、数据布局以及多平台向量化 3 个方面展开,详细介绍了面向 SIMD 扩展部件编译优化的研究进展和相关优化算法.

当前 SIMD 编译优化对数据对齐和重组研究很深入,面向循环和基本块的向量化方法也很成熟.目前,针对 SIMD 扩展部件编译优化的关注热点是生成高性能低功耗的代码.未来面向 SIMD 编译优化要解决以下几个问题:一是利用过程间分析和优化实现面向函数级的 SIMD 向量化,从函数、循环和基本块这 3 个粒度综合考虑 SIMD 向量化.二是更好的向量化不规则内存访问的程序,将投机并行应用到向量化中是未来的一个研究热点,如何向量化结构体、树、图等不规则数据结构也是重要的方向之一.三是改进 SIMD 扩展部件的体系结构和指令集.四是建立一个面向多平台的实现不同粒度的健壮向量化编译工具是未来的目标.该工具综合考虑不同向量长度、不同程序特征、各种各样的指令.此外,从自动编译的角度 SIMD 硬件结构还需提供更多的支持.具体包括支持归约、控制流、类型转换等指令,更高效的灵活的跨幅访存指令以及长度可变的向量寄存器.

经过 10 余年的发展,SIMD 扩展部件的应用范围越来越广,在多媒体领域,数字信号处理和高性能计算中都起着关键作用.很多厂商都在处理器内集成 SIMD 扩展部件,很多应用程序都在寻求 SIMD 并行.SIMD 扩展部件结构简单、功耗低、加速效果明显,不需要增加通信以及 cache 和内存的开销.因此即使在多核时代,SIMD 扩展部件仍然是程序加速的重要手段之一[85,102].

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] Furtak T, Amaral JN, Niewiadomski R. Using SIMD registers and instructions to enable instruction-level parallelism in sorting algorithms.In: Proc. of 19th Annual ACM Symp. on Paralleism in Algorithms and Architectures (SPAA). 2007. [doi: 10.1145/1248377.1248436]

[2] Schlegel B, Gemulla R, Lehner W. Fast integer compression using SIMD instructions. In: Proc. of the 6th Int'l Workshop on Data Management on New Hardware (DaMoN). 2010. [doi: 10.1145/1869389.1869394]

[3] Hayes T, Palomar O, Unsal O, Cristal A, Valero M. Vector extensions for decision support DBMS acceleration. In: Proc. of the 45th Annual Int'l Symp. on Microarchitecture. 2012. [doi: 10.1109/MICRO.2012.24]

[4] Shou YX, van Engelen RA. Automatic SIMD vectorization of chains of recurrences. In: Proc. of the 22nd Int'l Conf. on Supercomputing (ICS). 2008. [doi: 10.1145/1375527.1375564]

[5] Stepanov AA, Gangolli AR, Rose DE, Ernst RJ, Oberoi PS. SIMD-Based decoding of posting lists. In: Proc. of the 20th ACM Int'l Conf. on Information and Knowledge Management (CIKM). Glasgow, 2011. 317–326. [doi: 10.1145/2063576.2063627]

[6] Sompolski1 J, Zukowski M, Boncz P. Vectorization vs. compilation in query execution. In: Proc. of the 7th Int'l Workshop on Data Management on New Hardware (DaMoN). Athens, 2011. [doi: 10.1145/1995441.1995446]

[7] Rohou E, Williams K, Yuste D. Vectorization technology to improve interpreter performance. ACM Trans. on Architecture and Code Optimization (TACO), 2013,9(4):Article 26. [doi: 10.1145/2400682.2400685]

[8] Esterie P, Gaunard M, Falcou J, Laprest JT, Rozoy B. Boost. SIMD: Generic programming for portable SIMDization. In: Proc. of the 2012 Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2012. [doi: 10.1145/2370816.2370881]

[9] Wang H, Andrade H, Gedik B, Wu KL. A code generation approach for auto-vectorization in the spade compiler. In: Proc. of the 21st Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). Berlin: Springer-Verilog, 2009. 383−390. [doi: 10.1007/978-3-642-13374-9_26]

[10] Kretz M, Lindenstruth V. Vc: A C++ Library For Explicit Vectorization. Software: Practice and Experience, 2011.

[11] Leißa R, Hack S, Wald I,. 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]

[12] Lokhmotov A, Gaster BR, Mycroft A, Hickey N, Stuttard D. Revisiting SIMD programming. In: Proc. of the 19th Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). Berlin: Springer-Verilog, 2007. 32−46. [doi: 10.1007/978-3-540-85261-2_3]

[13] Sreraman N, Govindarajan R. A vectorizing compiler for multimedia extensions. Int'l Journal of Parallel Programming, 2000,28(4): 363−400. [doi: 10.1023/A:1007559022013]

[14] Zhang WH, Zang BY. SIMD compiling technology review. Communications of the CCF, 2007,3(2):27−36 (in Chinese).

[15] Ren G, Wu P, Padua D. A preliminary study on the vectorization of multimedia applications for multimedia extensions. In: Proc. of the 16th Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). Berlin: Springer-Verilog, 2004. 420−435. [doi: 10.1007/978-3-540-24644-2_27]

[16] Allen R, Kennedy K. Automatic translation of Fortran programs to vector form. ACM Trans.on Programming Languages and Systems, 1987,9(4):491−542. [doi: 10.1145/29873.29875]

[17] Bik AJC. Applying multimedia extensions for maximum performance. In: The Software Vectorization Handbook. Intel Press, 2004.

[18] Hampton M, Asanovic K. Compiling for vector-thread architectures. In: Proc. of the 6th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). 2008. [doi: 10.1145/1356058.1356085]

[19] Iwasawa K, Mycroft A. Choosing method of the most effective nested loop shearing for parallelism. In: Proc. of the 8th Int'l Conf. on Parallel and Distributed Comuputing, Applications and Technologies (PDCAT). 2007. 267−276. [doi: 10.1109/PDCAT.2007.44]

[20] 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]

[21] 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. [doi: 10.1109/PACT.2009.18]

[22] Kong M, Veras R, Stock K, Franchetti F, Poucher LN, Sadayappan P. 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]

[23] Stock K, Poudhet LN, Sadayappan P. Using machine learning to improve automatic vectorization. ACM Trans. on Architecture and Code Optimization (TACO), 2012,8(4):Article 50. [doi: 10.1145/2086696.2086729]

[24] Vasilache N, Meister B, Baskaran M, Lethin R. Joint scheduling and layout optimization to enable multi-level vectorization. In: Proc. of the Int'l Workshop on Polyhedral Compilation Techniques (IMPACT). 2012.

[25] Kong M, Pouchet LN, Sadayappan P. Abstract vector SIMD code generation using the polyhedral model. Technical Report, 4/13-TR08, Ohio State University, 2013.

[26] Larsen S, Amarasinghe S. Exploiting superword level parallelism with multimedia instruction sets. In: Proc. of the Conf. on Programming Language Design and Implementation (PLDI). 2000. 145−156. [doi: 10.1145/349299.349320]

[27] Tenllado C, Piuel L, Prieto M, Tirado F, Catthoor F. Improving superword level parallelism support in modern compilers. In: Proc. of the 3rd IEEE/ACM/IFIP Int'l Conf. on Hardware/Software Codesign and System Synthesis. 2005. [doi: 10.1145/1084834.1084909]

[28] 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]

[29] Leupers R. Code selection for media processors with SIMD instructions. In: Proc. of the Conf. on Design, Automation and Test in Europe (DATE). 2000. 4−8. [doi: 10.1109/DATE.2000.840007]

[30] 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]

[31] Hiroaki T, Takeuchi Y, Sakanushi K, Imai M, Ota Y, Matsumoto N, Nakagawa M. Pack instruction generation for media processors using multi-valued decision diagram. In: Proc. of the 4th Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS). 2006. 154−159. [doi: 10.1145/1176254.1176292]

[32] Rosen I, Nuzman D, Zaks A. Loop-Aware SLP in GCC. In: Proc. of the GCC Developers' Summit. 2007. 131−142.

[33] Shin J. SIMD programming by expansion. In: Proc. of the 21st Int'l Workshop on Languages and Compilers for Parallel Computing (LCPC). 2009.

[34] Karrenberg R, Hack S. Hack whole-function vectorization. In: Proc. of the 9th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). 2011. [doi: 10.1109/CGO.2011.5764682]

[35] Tanaka H, Ota Y, Matsumoto N, Hieda T, Takeuchi Y, Imai M. 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]

[36] Allen JR, 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]

[37] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures: A Dependence-Based Approach. Burlington: Morgan Kaufmann Publishers, 2001.

[38] Park J, Schlansker M. On predicated execution. Technical Report, HPL-91-58, Software and Systems Laboratory, 1991.

[39] Shin J, Hall M, Chame J. Superword-Level parallelism in the presence of control flow. In: Proc. of the 3rd Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). New York: ACM Press, 2005. 165−175. [doi: 10.1109/CGO.2005.33]

[40] 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]

[41] Shin J. Introducing control flow into vectorized code. In: Proc. of the 16th Int'l Conf. on Parallel Architecture and Compilation Techniques (PACT). Washington: IEEE Computer Society, 2007. 280−291. [doi: 10.1109/PACT.2007.4336219]

[42] Shin J, Hall M, Chame J. Evaluating compiler technology for control-flow optimizations for multimedia extension architectures. Microproesrs & Mircrosystems, 2009,33(4):235−243. [doi: 10.1016/j.micpro.2009.02.002]

[43] 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]

[44] von Dincklage D, Diwan A. Explaining failures of program analyses. In: Proc. of the 2008 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). 2008. [doi: 10.1145/1375581.1375614]

[45] von Dincklage D, Diwan A. Optimizing programs with intended semantics. In: Proc. of the 24th Annual ACM SIGPLAN Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA). 2009. [doi: 10.1145/1640089.1640120]

[46] Chapman B, Hernandez O. Dragon: An Open64-based interactive program analysis tool for large application. In: Proc. of the 4th Int'l Conf. on Parallel and Distributed Computing, Applications and Technologies (PDCAT). 2003.

[47] Intel Parallel Studio Guide. Beijing: Tsinghua University Press, 2010 (in Chinese).

[48] Hernandez O, Song F, Chapman B, Dongarra J, Mohr B, Moore S, Wolf F. Performance instrumentation and compiler optimizations for MPI/OpenMP applications. In: Proc. of the IWOMP 2006. 2006. [doi: 10.1007/978-3-540-68555-5_22]

[49] Whaley J, Kozyrakis C. Heuristics for profile-driven method-level speculative parallelization. In: Proc. of the 2005 Int'l Conf. on Parallel Processing (ICPP). 2005. 147−156. [doi: 10.1109/ICPP.2005.44]

[50] Tournavitis G, Wang Z, Franke B, O'Boyle MFP. Towards a holistic approach to auto-parallelization integrating profile-driven parallelism detection and machine-learning based mapping. In: Proc. of the 30th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). 2009. 147−156. [doi: 10.1145/1542476.1542496]

[51] von Dincklage D, Diwan A. Explaining failures of program analyses. In: Proc. of the 29th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI). 2008. [doi: 10.1145/1375581.1375614]

[52] Boekhold M, Karkowski I, Corparaal H. Transforming and parallelizing ANSI C programs using pattern recognition. In: Lecture Notes in Computer Science 1593, 1999. 673. [doi: 10.1007/BFb0100628]

[53] Hormati1 AH, Choi Y, Who M, Kudlur M, Rabbah R, Mudge T, Mahlke S. MacroSS: Macro-SIMDization of streaming applications. In: Proc. of the 2010 Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2010. [doi: 10.1145/1736020.1736053]

[54] Kumar1 R, Martínez A, Gonzalez 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]

[55] Larsen S, Rabbah R, Amarasinghe S. Exploiting vector parallelism in software pipelined loops. In: Proc. of the 38th Annual Int'l Symp. on Microarchitecture. 2005. 119−129. [doi: 10.1109/MICRO.2005.20]

[56] Dasika G, Woh M, Seo S, Clark N, Mudge T, Mahlke S. Mighty-Morphing power-SIMD. In: Proc. of the 2010 Int'l Conf. on Compilers, Architectures and Synthesis for Embedded Systems (CASES). 2010. [doi: 10.1145/1878921.1878934]

[57] Park Y, Seo S, Park H, Cho HK, Mahlke S. SIMD defragmenter: Efficient ILP realization on data-parallel architectures. In: Proc. of the 17th Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS). 2012. [doi: 10.1145/2150976.2151014]

[58] Park YJ, Park JJK, Park H, Mahlke S. Tailoring SIMD execution using heterogeneous hardware and dynamic configurability. In Proc. of the 45th Annual IEEE/ACM Int'l Symp. on Microarchitecture (MICRO). 2012. [doi: 10.1109/MICRO.2012.17]

[59] Wang YH, Chen SM, Wan JH, Meng JY, Zhang K, Liu W, Ning X. A multiple SIMD, multiple data (MSMD) architecture: parallel execution of dynamic and static SIMD fragments. In: Proc. of the 2013 Int'l Symp. on High-Performance Computer Architecture (HPCA). 2013. [doi: 10.1109/HPCA.2013.6522353]

[60] Hampton M, Asanovic K. Compiling for vector-thread architectures. In: Proc. of the 6th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). 2008. [doi: 10.1145/1356058.1356085]

[61] Lorenz M, Marwedel P, Drager T, Fettweis G, Leupers R. Compiler based exploration of DSP energy savings by SIMD operations. In: Proc. of the 15th Asia and South Pacific Design Automation Conf. (ASP-DAC). 2004. 839−842. [doi: 10.1109/ASPDAC.2004.1337711]

[62] Ibrahim MEA, Rupp M, Fahmy HAH. Code transformations and SIMD impact on embedded software energy/power consumption. In: Proc. of the 2009 Int'l Conf. on Computer Engineering and Systems (ICCES). 2009. 27−32. [doi: 10.1109/ICCES.2009.5383317]

[63] Paci G, Marchal P, Benini L. Exploration of low power adders for a SIMD data path. In: Proc. of the 15th Asia and South Pacific Design Automation Conf. (ASP-DAC). 2007. [doi: 10.1109/ASPDAC.2007.358106]

[64] Lorenz M, Wehmeyer L, Dräger T. Energy aware compilation for DSPs with SIMD instructions. In: Proc. of the 2002 ACM Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES). 2002. 94−101. [doi: 10.1145/513829.513847]

[65] 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.

[66] 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]

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

[68] Wu P, Eichenberger AE, Wang A. Efficient simd code generation for runtime alignment and length conversion. In: Proc. of the 3rd Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). New York: ACM Press, 2005. 153−164. [doi: 10.1109/CGO.2005.18]

[69] Pryanishnikov I, Krall A, Horspool N. Pointer alignment analysis for processors with SIMD instructions. In: Proc. of the 5th Workshop on Media and Streaming Processors. 2003. 50−57.

[70] Fireman L, Petrank E, Zakas A. New algorithms for SIMD alignment. In: Proc. of the 16th Int'l Conf. on Compiler Construction. 2007. 26−30. [doi: 10.1007/978-3-540-71229-9_1]

[71] 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]

[72] Wei S. Research of SIMD vectorization algorithm and regroup technology [Ph.D. Thesis]. Zhengzhou: Information Engineering University, 2012 (in Chinese with English abstract).

[73] 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]

[74] 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]

[75] Naishlos D, Biberstein M, Ben-David S, Zaks A. Vectorizing for a SIMdD DSP architecture. In: Proc. of the 2003 Int'l Conf. on Compilers, Architecture and Synthesis for Embedded Systems (CASES). 2003. [doi: 10.1145/951710.951714]

[76] Lee RB. Subword permutation instructions for two-dimensional multimedia processing in micro SIMD architectures. In: Proc. of the 2000 Conf. on Application-Specific Systems, Architectures, and Processors (ASAP). 2000. 3–14.

[77] McGregor JP, Lee RB. Architectural techniques for accelerating subword permutations with repetitions. In: IEEE Trans. on Very Large Scale Integration (VLSI) Systems. 2003. 325−335. [doi: 10.1109/TVLSI.2003.812318]

[78] Hanounik B, Hu XB. Linear-Time matrix transpose algorithms using vector register file with diagonal registers. In: Proc. of the Int'l Conf. on Parallel and Distributed Processing Symposium (IPDPS). 2001. 23−27. [doi: 10.1109/IPDPS.2001.924973]

[79] Shahbarami A, Juurlink B, Vassiliadis S. Matrix register file and extended subwords: Two techniques for embedded media processors. In: Proc. of the Conf. Computing Frontiers. 2005. 171−179. [doi: 10.1145/1062261.1062291]

[80] Kudriavtsev A, Kogge P. Generation of permutations for SIMD processors. In: Proc. of the 2005 ACM Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES). 2005. 147−156. [doi: 10.1145/1065910.1065931]

[81] Franchetti F, Püschel M. Generating SIMD vectorized permutations. In: Proc. of the Joint European Conf. on Theory and Practice of Software 17th Int'l Conf. on Compiler Construction. Budapest, 2008. [doi: 10.1007/978-3-540-78791-4_8]

[82] 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]

[83] Huang L, 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]

[84] Shen L, Huang L, Xiao N, Wang Z. Implicit data permutation for simd devices. In: Proc. of the 4th Int'l Conf. on Embedded and Multimedia Computing. 2009. 1−6. [doi: 10.1109/EM-COM.2009.5403000]

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

[86] Kim S, Han H. Efficient SIMD code generation for irregular kernels. In: Proc. of the 17th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming (PPoPP). 2012. [doi: 10.1145/2145816.2145824]

[87] Ren B, Agrawal G, Mytkowicz T, Poutanen T. Wolfram schulte fine-grained parallel traversals of irregular data structures. In: Proc. of the 2012 Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2012. [doi: 10.1145/2370816.2370896]

[88] Shin J, Chame J, Hall MW. Compiler-Controlled caching in superword register files for multimedia extension architectures. In Proc. of the 11th Int'l Conf. on Parallel Architectures and Compilation Techniques (PACT). 2002. 45−55. [doi: 10.1109/PACT.2002.1106003]

[89] Shin J, Chame J, Hall MW. Exploiting superword-level locality in multimedia extension architectures. JILP, 2003.

[90] Nuzman D, Namolaru M, Zaks A, Derby JH. Compiling for an indirect vector register architecture. In: Proc. of the CF. 2008. [doi: 10.1145/1366230.1366266]

[91] Lesnicki P, Cohen A, Fursin G, Cornero M, Ornstein A, Rohou E. Split compilation: An application to just-in time vectorization. In: Proc. of the Workshop on GCC for Research in Embedded and Parallel Systems (GREPS). Brasov, 2007.

[92] Nuzman D, Henderson R. Multi-Platform auto-vectorization. In: Proc. of the 4th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). 2006. 281−294. [doi: 10.1109/CGO.2006.25]

[93] Nuzman D, Dyshel S, Rohou E, Rosen I. Vapor SIMD: Auto-Vectorize once, run everywhere. In: Proc. of the 9th Annual IEEE/ACM Int'l Symp. on Code Generation and Optimization (CGO). 2011. [doi: 10.1109/CGO.2011.5764683]

[94] Costa R, Ornstein AC, Rohou E. CLI back-end in GCC. In: Proc. of the GCC Developers' Summit. Ottawa, 2007. 111−116.

[95] El-Shobaky S, El-Mahdy A, El-Nahas A. Automatic vectorization using dynamic compilation and tree pattern matching technique in Jikes RVM. In: Proc. of the 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems (ICOOOLPS). 2009. 63−69. [doi: 10.1145/1565824.1565833]

[96] Hohenauer M, Schumacher C, Leupers R, Ascheid G, Meyr H, van Someren H. Retargetable code optimization with SIMD instructions. In: Proc. of the 4th Int'l Conf. on Hardware/Software Codesign and System Synthesis (CODES+ISSS). 2006. [doi: 10.1145/1176254.1176291]

[97] Wu P, Eichenberger AE, Wang A, Zhao P. An integrated simdization framework using virtual vectors. In: Proc. of the 19th Annual Int'l Conf. on Supercomputing. 2005. [doi: 10.1145/1088149.1088172]

[98] Jr. Bocchino RL, AdveVector VS. Vector LLVA: A virtual vector instruction set for media processing. In: Proc. of the 2nd Int'l Conf. on Virtual Eexecution Environments (VEE). 2006.

[99] 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]

[100] Peng F, Gu NJ, Gao X, Sun MM. 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).

[101] Xin NJ, Chen XC, Sun HY, Yang L, Luo J, Dan XQ, Wang J. 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).

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

[103] Hassaballah M, Omran S, Mahdy YB. A review of SIMD multimedia extensions and their usage in scientific and engineering applications. The Computer Journal, 2008,51(6):630−650. [doi: 10.1093/comjnl/bxm099]

[104] Ramachandran A, Vienne J, van der Wijngaart R, Koesterke L, Sharapov I. Performance evaluation of NAS parallel benchmarks on Intel Xeon Phi. In: Proc. of the 42nd Int'l Conf. on Parallel Processing (ICPP). 2013. [doi: 10.1109/ICPP.2013.87]

附中文参考文献:

[14] 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.

[47] 英特尔 Parallel Studio 并行开发指南.北京:清华大学出版社,2010.

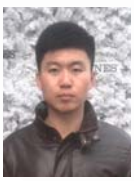
[72] 魏帅.面向 SIMD 的向量化算法及重组技术研究[博士学位论文].郑州:信息工程大学,2012.

[85] 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.

[100] 彭飞,顾乃杰,高翔,孙明明.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.

[101] 辛乃军,陈旭灿,孙海燕,阳柳,罗杰,淡孝强,王霁.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.

[102] 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.计算机应用系统,2013,22(12):140−143.



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.

庞建民(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算,信息安全.

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

丁锐(1984－),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 会员,主要研究领域为高性能计算,先进编译技术.