一种面向SIMD扩展部件的向量化统一架构

刘 鹏 赵荣彩 赵 博 高 伟
(信息工程大学 郑州 450001) (数学工程与先进计算国家重点实验室 郑州 450001)

摘 要 随着多媒体应用的普及和高性能计算的需求,越来越多的处理器集成了SIMD扩展。为了针对不同SIMD扩展部件自动生成高效的向量化代码,设计了一套虚拟向量指令集,在此基础上构建了一种面向SIMD扩展部件的向量化统一架构。将输入程序通过向量识别等阶段转变为虚拟向量指令的中间表示,而后通过向量长度解虚拟化和指令集解虚拟化,将其转变为特定SIMD部件的向量指令集。在申威1600、DSP和Alpha上的实验结果表明:统一架构能够针对3种平台自动变换出高效的向量化代码,在DSP上的加速比要明显优于其它两种平台。

关键词 向量化,单指令多数据,解虚拟化,基本块,循环展开
中图法分类号 TP312 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.09.004

Unified Vectorization Framework for SIMD Extensions

LIU Peng ZHAO Rong-cai ZHAO Bo GAO Wei
(Information Engineering University, Zhengzhou 450001, China)
(State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China)

Abstract With the popularization of multimedia applications and the requirement of high performance computing, more and more processors are integrated with SIMD extensions. A virtual simd instruction set was designed for generating efficient simd code for different SIMD extension units, and a unification framework facing isomeric SIMD extension units was built. Program input can be transformed into intermediate representation of virtual simd instruction and then translated into simd instruction set of specific SIMD extensions through vector length and instruction set devirtualization solutions. The experimental results on SW1600, DSP and Alpha show that efficient simd code in three platforms is generated automatically on this framework and DSP has the better speedup than the other two platform.

Keywords Vectorization, SIMD, Devirtualization, Basic block, Loop unrolling

1 引言

人类对计算能力的需求不断增加,使得并行计算技术越来越受到人们的重视,总体上并行硬件技术的发展要远远快于并行软件技术的发展。近年来SIMD扩展部件为提升程序性能提供了硬件支持,并广泛应用于通用处理器、DSP等平台。典型的指令集包括通用处理器Intel的MMX系列[1,2]、AMD的3DNow![3]、PowePC的AltiVec[4],以及DSP中StarCore[5]、TigerSharc[6]、TI C64x[7]的SIMD扩展指令。为充分发挥SIMD扩展的性能,需要充分发掘程序中的并行性,开发具有良好可扩展性的向量化程序。

如今的计算问题规模庞大、计算量大,手工编写向量化程序难度较大,自动向量化技术通过分析程序中语句操作和数据的特征,识别出串行程序中可以向量化的代码部分,不需要程序员对程序进行修改就可以在编译器下进行向量化编译,使程序员摆脱繁琐及手工易错的向量化代码编写工作。当前有以下3种典型的向量化方法:传统向量化方法由Allen和Kennedy提出,其基本思想是按照循环内的数据依赖构造相应的语句依赖图,语句依赖中不在强连通分量的语句就是可以向量执行的语句[8];2000年Larsen和Amarasinghe提出了超字级并行(Superword Level Parallelim, SLP)向量化方法[9],其思想来源于指令级并行,以基本块为单位识别出相邻且连续的访问语句,对其中的同构语句进行打包,然后根据定义使用关系进行包扩展,最后生成比传统向量化更有效的打包方案;模式匹配的向量化方法[10]需要视目标程序的特征来确定匹配的模式,首先将循环内的指令组进行划分,以数据存取指令为起始节点构建树形结构,然后识别基本块内的公共子表达式,最后采用数据重组算法对其中的公共子表达式进行优化。

以上3种典型向量化方法中,传统向量化对循环进行逐层分析,当依赖为内层循环携带依赖环时无法向量化;超字并行向量化在包生成中有一定的随机性,可能导致最终的向量化策略与理想的结果不一致;而模式匹配向量化则是上述两种方法的补充。在上述3种典型方法的基础上,存在大量

到稿日期:2013-11-04 返修日期:2014-01-30 本文受核高基国家科技重大专项(2009ZX01036)资助。
刘 鹏(1981-),男,博士生,主要研究方向为高性能计算和先进编译技术,E-mail:magiceelp@gmail.com;赵荣彩(1957-),男,博士,教授,博士生导师,主要研究方向为先进编译技术、软件逆向工程等;赵 博(1989-),男,硕士生,主要研究方向为高性能计算和先进编译技术;高 伟(1989-),男,硕士生,主要研究方向为高性能计算和先进编译技术。

SIMD 自动向量化的研究工作，包括：对界问题[11,12]、数据重组问题[13]、控制流问题[14,15]和跨步访问问题[16]。当前的主流商业编译器和开源编译器工具大多仅能对具有单一的向量长度的一种 SIMD 指令集生成向量化代码，然而不同的 SIMD 部件在向量长度、支持的数据类型、支持的指令集等方面差异很大，为一种 SIMD 部件开发的向量化工具很难克服不同 SIMD 部件的差异，以应用于另一种 SIMD 部件，向量化工具的代码不能很好地重用。为满足不同 SIMD 部件的自动向量化需求，本文针对 SIMD 部件抽象出一套虚拟向量指令集，由此建立的向量化统一架构能够面向 SIMD 部件自动生成向量化代码，提出的虚拟指令集能够较好地支持非对界、非连续和控制流等特征的向量化，弥补既有方法的局限性。

本文第 2 节对向量化统一架构进行了整体介绍；第 3 节给出了一种虚拟向量指令集；第 4 节对架构中的各组成部分进行了分析；第 5 节通过实验对该架构进行了验证；最后是结论。

2 向量化统一架构概述

为克服现有向量化方法存在的不足，针对目前 SIMD 扩展部件具有不同向量长度和不同指令集的特点，本文设计了一种具有可扩展性、可移植性且灵活、高效的面向 SIMD 扩展部件的统一向量化架构，整体架构如图 1 所示。

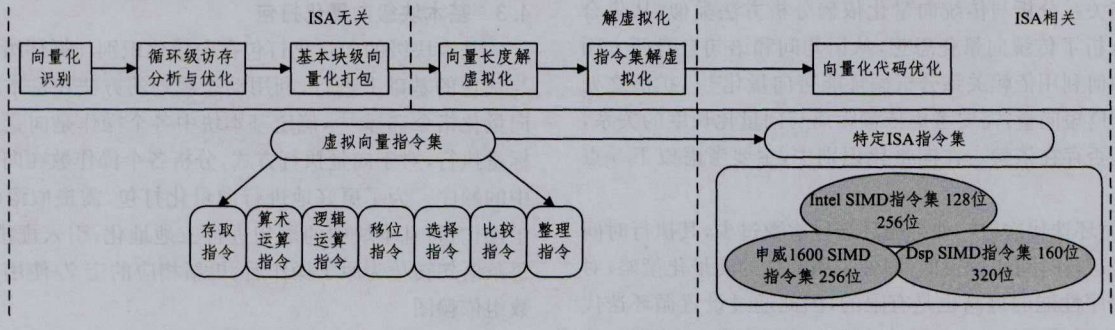

图 1 面向 SIMD 扩展部件的统一向量化架构

该向量化架构包含与指令集架构(ISA)无关、解虚拟化、与 ISA 相关 3 部分，与 ISA 无关的部分包括向量化识别、循环级访存分析与优化、基本块级向量化打包 3 部分，这些阶段通过向量识别和并行性发掘，变换为包含虚拟向量指令集信息的中间表示，该部分与特定 ISA 的向量指令集、程序分析和变换均基于具有虚拟向量长度的虚拟向量指令集；解虚拟化部分包括向量长度解虚拟化和指令集解虚拟化，两者在根据虚拟向量指令集中的解虚拟化属性进行虚实转换，将虚拟向量指令集转换为 DSP、Intel、申威等特定 ISA 向量指令集；与 ISA 相关的部分为向量化代码优化，其主要进行跨基本块冗余删除等常规优化以及针对特定平台进行代码优化。

3 虚拟向量指令集

本文所提出的向量化统一架构建立在虚拟指令集的基础上，该指令集适用于不同向量长度、不同向量指令集的 SIMD 扩展部件，是一种精简指令集，对于复杂指令可通过多条基本指令的组合实现。具体的虚拟指令集见表 1，基本指令包括存取指令、算数运算指令、逻辑运算指令、移位指令、选择指令、比较指令和整理指令 7 类，它们与指令集、向量长度以及数据类型均无关，解虚拟化的过程是由虚拟指令向实际指令变换的过程，解虚拟化属性规定了从虚拟指令向实际指令如何映射，根据这些解虚拟化属性能够将虚拟指令映射到不同平台的向量指令集。

虚拟向量长度值为 $Len_v = 2^{\lceil \log_2^{MAX(Len_1,\cdots,Len_i,\cdots)} \rceil}$，其中 Len_i 为不同平台的向量长度。实际指令集中具有 128 位、160 位、256 位、320 位和 512 位等多种不同的向量长度。由于主流 SIMD 指令集中向量长度非 2^n 时包含了符号位扩展，因此 Len_v 的值为实际 ISA 向量长度按 2^n 取整的最大值。由于当前大多 SIMD 硬件或向量化方法支持跨步(stride)访问的向量操作，当 stride 为 2^n 时，它们具有一定的向量化收益。simd_gather 和 simd_scatter 两种指令用来实现 stride 内存访问的虚拟操作，整理指令中 simd_shuffle 通过掩码能够将两向量任意位置的元素进行重组。

表 1 虚拟向量指令集

指令类别	指令名	指令描述	解虚拟化属性
存取指令	simd_load simd_store simd_gather simd_scatter	连续访存及跨步访存指令	align/unalign 是否对齐 cyclic/uncyclic 是否循环寻址 return/param 向量寄存器为返回值或第一个参数 base + offset/base，offse 是否将基地址和偏移拆分为两个参数 vlen 向量长度 type 数据类型
算术运算指令	simd_add simd_sub simd_mul simd_div	基本的加、减、乘、除、乘方等算术运算指令	sat/unsat 是否饱和 vlen 向量长度 type 数据类型
逻辑运算指令	simd_and simd_or simd_not	基本的与、或、非逻辑运算指令	vlen 向量长度
移位指令	simd_shl simd_shr	左移、右移指令	arithmetic/logic/cycle 移位类型(算术移位，逻辑移位，循环移位) vlen 向量长度 type 数据类型
比较指令	simd_cmpgr simd_cmpeq simd_cmplt	进行大于、等于、小于 3 种基本比较，并将结果返回	vlen 向量长度 type 数据类型
选择指令	simd_selectgr simd_selecteq simd_selectlt simd_ifthen simd_ifelse simd_return	根据大于、等于、小于 3 种比较结果选择不同的向量	vlen 向量长度 type 数据类型
整理指令	simd_shuffle simd_insert simd_extract simd_set	混洗指令、插入提取指令和赋值指令	vlen 向量长度 type 数据类型

为支持控制流，虚拟指令中设计了 simd_ifthen，simd_ifelse，simd_ifreturn 指令，这些指令实现了将控制流中的条件、

分支和结果等参数打包为向量，有效消除了循环中的控制流，进而通过识别指令序列生成 simd_selectgr 等选择指令。

4 面向 SIMD 部件的向量化统一架构

4.1 向量化识别

对循环向量化时会耗费一定的编译时间用于程序分析和代码生成，如基本块向量化指令的打包策略需要在全空间搜索同构指令的组成策略。因此并非所有的循环都适合进行向量化变换，在对循环作基本块向量化分析和变换之前进行循环的向量化识别，可以减少编译时间，避免盲目的变换和优化。

依赖关系分析与传统向量化依赖分析方法类似，从集合的角度分析了传统向量化思想，从语句间和语句自身两方面阐述了如何利用依赖关系分析循环能否向量化[17]，扩展之处在于 SIMD 短向量，需要考虑依赖距离与向量化长度的关系，并确定是否存在依赖。在向量化识别中，主要考虑以下 5 点准则：

(1)循环迭代次数。如果循环迭代次数过少，其执行时间所占程序运行时间比例很低，即使采用最佳的向量化策略，对于整个程序性能的提高也是有限的，因此通过设置循环迭代次数阈值，可以避免对执行迭代次数少的循环进行向量化，从而降低程序编译时间。

(2)语句中可向量化指令个数与语句中指令总数的比例。一条语句中包含多个原子指令操作，若这些原子操作中可向量化操作比例很低，即使向量化后可能也无性能收益。当比例大于某个阈值时，就认为该语句值得向量化。

(3)循环中可向量化语句占循环中语句总数的比例。通过第二条标准可以得到循环中可向量化语句占循环中语句总数的比例，当该值大于某个阈值时，可以预测出该循环向量化后收益的大致趋势。

(4)循环中可向量化的操作个数占所有操作个数的比例。该阈值是从总体上判断循环中可向量化操作的数目，采用对不同可向量化操作赋于不同权重的方法，通过整体分析，考虑向量化的可行性。

(5)循环中访存操作个数占循环中所有操作个数的比例。该阈值可以从总体上判断循环访存的操作数目，分析是否存在足够的连续的访存，以保证向量化产生收益。

4.2 循环级访存分析与优化

在循环级对数组访存进行分析，主要考虑存储或装载的内存地址是否对齐与连续，并根据连续和对齐信息进行优化。计算出循环中的每个访存首地址相对于向量因子的数据偏移，根据循环中数组访问的起始地址是否对齐来判断该循环是否需要进行多版本变换，从而可以产生更为高效的向量化代码。其主要目的是通过循环展开，发掘更大力度的并行性。其流程包含以下 5 个步骤：

(1)数组访存点对齐分析。为基本块向量化打包建立数组访存首地址的对齐信息，并且建立访存点到对齐信息的映射。

(2)确定循环展开因子。在循环内通过相邻地址分析最大限度地发掘相邻的地址引用，在收集所有在迭代间连续的地址偏移的基础上，通过虚拟向量长度确定展开因子：

$$unroll\_factor=\frac{Len_v}{GCD(Len_v,offset_1,\cdots,offset_i,\cdots)}$$

其中，$offset_i$ 为不同引用点的连续地址偏移。

(3)循环剥离。确定循环剥离因子，将循环头部的若干次迭代进行循环剥离变换，以对齐访存的中间表示。

(4)循环展开。按循环展开因子进行循环展开变换，如果待展开循环中有归约操作且归约语句与循环内其它语句无依赖时，对归约变量进行重命名，在该循环前加上归约初始化部分，后面加上归约结束处理。

(5)多版本优化。对于数组访存首地址不可知、数组某一维不可知或某一维线性下标中有符号量的情况，通过多版本优化来确定其对齐信息，以便生成包含控制流的高效的向量化代码。

4.3 基本块级向量化打包

基本块级的向量化打包在向量化识别与循环级访存分析与优化的基础上进行，利用动态规划的方法在基本块内进行向量化指令选择[18]，确定基本块中各个操作是向量执行还是标量执行，对于向量执行方式，分析各个操作数在向量寄存器中的顺序。为了更好地进行向量化打包，需提取语句中的原子操作，将基本块中的语句进行三地址化，引入虚拟寄存器，每条语句转化为原子操作后，更新相应的定义-使用关系图和数组依赖图。

基本块级向量化打包时，包大小为虚拟向量长度 Len_v，先根据相邻地址访存建立初始可向量化 pack 集合，然后沿依赖图的遍历顺序通过使用定义链(U-D chain)实现包扩展，扩展中采用搜索树的方式，以 SIMD 向量化收益模型为基础进行启发式的搜索和扩展，最后选择收益最大的包生成方法，从而确定一条完整的最优路径，并在包生成之后删除冗余的装载包和对三地址化的语句进行恢复。在向量化发掘的后续优化中，根据数据使用的上下文对向量化发掘结果进行调整，如将某些标量执行语句转换为向量执行语句，以减少重组操作。

进行访存语句打包时，一条标量 load 或 store 可以映射到多个 pack，以防止前期的冗余访存语句消除中带来的负面影响，也可以防止打包后出现过多的标量访存操作。

4.4 向量长度解虚拟化

进行基本块级向量化打包后得到了具有虚拟向量长度的虚拟向量指令，为将虚拟向量长度能够转化特定 SIMD 架构的向量长度，需要进行向量长度的解虚拟化，其流程如图 2 所示。

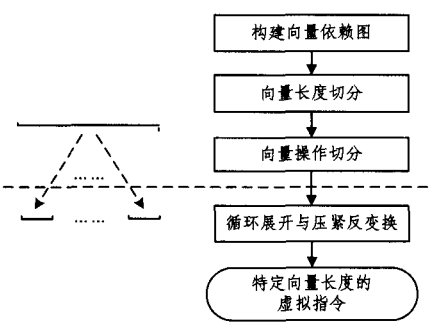

图 2 向量长度解虚拟化流程

(1)依据向量间、标量间和向量与标量间的依赖关系构建向量依赖图；

(2)根据实际向量长度和虚拟向量长度对向量进行切分；

(3)在依赖图的基础上按照拓扑顺序对向量操作进行切分，虚拟向量长度为 Lenv，实际向量长度为 Lens，两者之间存

在关系 Lenv=2n * Lens，例如：一条连续的 SIMD 装载操作将被切分为 Lenv/Lens 条长度为 Lens 的连续的 SIMD 装载操作；

(4)在对基本块内的所有 SIMD 向量操作进行切分后，对其进行循环展开与压紧反变换；

(5)变换后得到具有特定向量长度 Lens 的 SIMD 虚拟指令。

4.5 指令集解虚拟化

指令集解虚拟化是由虚拟指令向特定平台指令集进行映射，根据解虚拟化属性的不同，其映射方式存在一定差异，例如：simd_gather 将映射为特定平台的向量装载和整理指令，若平台中不存在对应的指令，则映射为多个标量装载指令。具体流程(见图 3)如下：

(1)在依赖关系图的基础上依次分析每条虚拟指令，并提取该操作对应的解虚拟化属性；

(2)若能进行一对一向量指令映射，则直接将虚拟向量指令转化为实际向量指令，转(1)；若不能进行一对一向量指令映射，则转(3)；

(3)若能进行多对一向量指令映射，则直接将虚拟向量指令转化为实际向量指令，转(1)；若不能进行多对一向量指令映射，则转(4)；

(4)若能进行一对多向量指令映射，则直接将虚拟向量指令转化为实际向量指令，转(1)；若不能进行一对多向量指令映射，则转(5)；

(5)进行一对多标量指令映射和转换，转(1)。

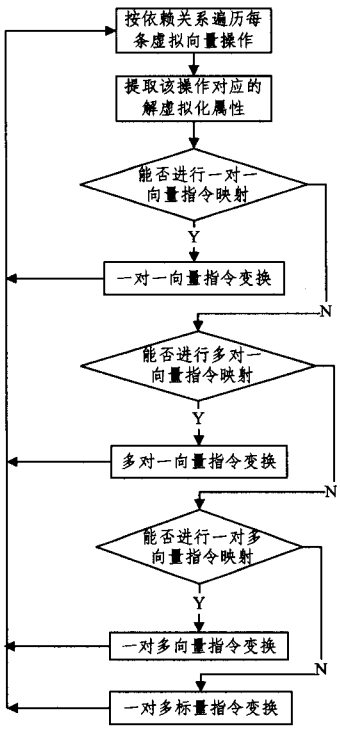

图 3 指令集解虚拟化流程

对依赖关系图中的所有语句进行遍历后，生成在特定向量 ISA 的实际向量指令。

4.6 向量化代码优化

生成特定平台向量指令后，在基本块边界可能产生大量冗余操作，其主要原因是超字并行的向量化发掘针对基本块展开，未对跨基本块间的打包、拆包的冗余操作进行分析。针对基本块间的冗余操作，以基本块为单位构建控制流图和数据流图，发掘基本块间的数据依赖，在基本块间建立各个变量打包、拆包的收益模型，在相邻基本块间进行向量化代码优化，避免生成低效、冗余的向量化代码。同时，对于一些特定平台的特殊性，如：分支预测、流水等，为增强向量化代码的功能，可以进行有针对性的可选的代码优化工作。

5 实验与分析

5.1 实验环境与测试集

向量化统一架构基于开源编译器 Open64 实现，核心部分位于循环嵌套优化，特定 SIMD 部件可通过对指令集、操作数、操作码、数据类型和向量长度的等上层接口进行修改，动态生成指令集文件 intrn_entry. def、操作数和操作码文件 opcode_gen_core. h/. cxx、支持的数据类型文件 mtypes. h/. cxx 等等。解虚拟化属性在 Open64 中既有的参数配置文件 lno_config. h/. cxx 中进行配置(例如 load 指令基地址与偏移拆分的配置信息为：# define UVI_Load_Addr_Split Current_LNO —> Load_Addr_Split)，并在向量长度解虚拟化和指令集解虚拟化两部分中对各种参数均有接口实现。

编译环境为 IBM 3850 服务器，处理器频率 2. 0GHz，内存 4GB，L1 数据缓存为 32KB，L2 数据缓存为 256KB，基本页面大小为 8kB。操作系统内核为 Linux 2. 6. 18，版本为 Redhat Enterprise AS 5. 0。

面向 SIMD 部件包括国产 CPU 申威 1600、华为某 DSP 和 Alpha 的自动向量化工作，其中申威 1600 的向量长度分别为 256 位和 128 位，华为某 DSP 支持 160 位和 320 位两种向量长度。运行环境分别为神威蓝光服务器、华为某 DSP 模拟器和 Alpha 服务器。实验中综合考虑 DSP 和通用 CPU 的 SIMD 加速部件，采用 C 语言测试集，测试集及描述如表 2 所列。

表 2 测试集列表

序号	名称	描述
1	sum_i16	Short 类型归约加
2	sum_i32	int 类型归约加
3	convolution	二维卷积运算
4	MMM_f32	float 类型矩阵乘
5	MMM_f64	double 类型矩阵乘

5.2 实验结果与分析

实验面向申威 1600、华为某 DSP 和 Alpha 3 种平台的 SIMD 部件进行测试，主要测试 3 种平台的向量化加速比，验证向量化工具的有效性，并对比 SIMD 部件的加速效果。实现架构为源源变换，对向量化后的代码膨胀率进行测试，分析不同的指令集特征对代码膨胀率的影响。各平台 SIMD 部件加速比和膨胀率分别如图 4 和图 5 所示。

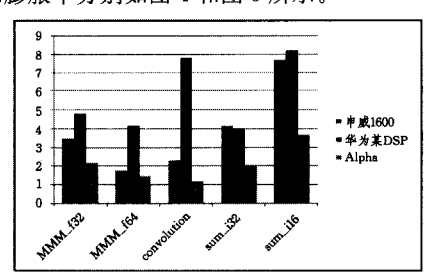

图 4 各平台 SIMD 部件向量化加速比

(下转第 44 页)

[10] Fan Y, He H. The Research of 2-Tier and 3-Tier Structure Based on the Client/Server Architecture [J]. Application Research of Computers, 2001, 12: 23-24

[11] De Clercq J. Single sign-on architectures[M]. Infrastructure Security. Springer Berlin Heidelberg, 2002: 40-58

[12] Srivastava B, Koehler J. Web service composition-current solutions and open problems[C]// ICAPS 2003 workshop on Planning for Web Services. 2003, 35: 28-35

[13] Using PHP from the command line [EB/OL]. http://www.php.net/manual/en/features.commandline.php

(上接第 31 页)

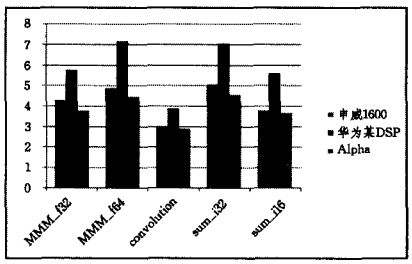

图 5 各平台 SIMD 部件向量化膨胀率

以 int 型为基准，申威 1600、DSP 和 Alpha 的向量化因子分别为 8、4 和 4，由于 DSP 的应用为大量密集的数字计算，其指令集相对复杂，一些指令的设计更为高效，这种高效会带来更高的代码膨胀率。以卷积为例进行分析，convolution 进行外层向量化后，DPS 生成的循环寻址指令能够高效地获取下一个操作数，并通过软件流水获得了时间上的并行性，具有明显的加速效果。

通过实验结果可以看出：

(1)该架构能够对 SIMD 部件的不同指令集自动变换出高效的向量化代码；

(2)指令集越复杂，向量化后的加速效果越好，代码膨胀率越高；

(3)DSP 具有较高的加速比，加速性能高于申威 1600 平均 48.7%，高于 Alpha 平均 173.6%；

(4) 向量化后 Alpha 的膨胀率较小，可读性较好，膨胀率优于申威 1600 平均 8.8%，优于 DSP 平均 51.3%。

结束语 多媒体应用的普及和高性能应用的需求使得越来越多的处理器集成了 SIMD 扩展，本文针对 SIMD 部件的自动向量化提出了一种统一架构，其主要贡献有以下 3 点：

(1) 统一架构能够自动变换为面向 SIMD 扩展部件的向量化代码，让程序员从繁冗复杂的手工向量化编码中解脱出来；

(2) 统一架构适用于不同向量长度和不同向量指令集的 SIMD 扩展部件，避免了针对不同 SIMD 扩展开发不同向量化工具的尴尬局面，有效提高了向量化编译器的开发效率；

(3) 设计的虚拟向量指令集能够支持非对齐访存、控制流、跨步访问、数据重组等向量化中的难点问题，对多媒体应用的自动向量化具有一定的普适性。

当前的虚拟向量指令集仅为特定 SIMD 指令集的抽象和精简，由于诸如 Intel 等平台向量指令集的复杂性，统一架构很难保证面向多种平台均能生成高效的向量化代码。下一步工作中，我们计划将更多的向量化指令特征融入虚拟向量指令集，以提高架构的向量化能力和加速效果。

参 考 文 献

[1] Peleg A, Weiser U. MMX Technology Extension to the Intel Architecture[J]. IEEE/ACM International Symposium on Microarchitecture, 1996, 16(4): 42-50

[2] Intel 64-ia-32-architectures-software-developer-manual [EB/OL]. http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf, September 2013

[3] Stewart J. An Investigation of SIMD instruction sets. School of Information Technology and Mathematical Sciences[R]. University of Ballarat, 2005

[4] Franchetti F, Kral S, Lorenz J, et al. Efficient utilization of SIMD extensions[J]. Proceedings of the IEEE, 2005, 93(2): 409-425

[5] SC140 DSP Core Reference Manual. Freescale Semiconductor [EB/OL]. http:// catch.freescale.com/files/dsp/doc/ref_manual/MNSC140CORE.pdf, 2004

[6] Fridman J, Greenfield Z. The Tiger SHARC DSP Architecture [J]. IEEE Micro, 2000, 20(1): 66-76

[7] TMS320C6000 CPU and Instruction Set Reference Guild(Rev. F)[R]. Texas Instruments Inc. 2000

[8] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures——A Dependence-based Approach[M]. US: Morgan Kaufmann Publishers, 2001

[9] Larsen S, Amarasinghe S. Exploiting superword level parallelism with multimedia instruction sets[C]// Proc of the ACM SIGPLAN Conference on Programming Language Design and Implementation. June 2000: 145-156

[10] Kudriavtsev A, Kogge P. Generation of Permutations for SIMD Processors[C]//PLDI, 2006. Ottawa, Canada, 2006

[11] Eichenberger A E, Wu Peng, O'brein K. Vectorization for simd architectures with alignment constraints[C]//PLDI. June 2004

[12] Wu Peng, Eichenberger A E, Wang A. Efficient simd code generation for runtime alignment[C]//CGO. March 2005

[13] Hiroaki T, Chi Y T, Sakanushi K, et al. Pack Instruction Generation for Media Processors Using Multi-valued Decision Diagram [C]//CODES+ISSS. Seoul, Korea, October 2006: 154-159

[14] Karrenberg R. Whole Function Vectorization[C]//2011 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization. 2011: 141-150

[15] Zhu Jia-feng, Zhao Rong-cai, Han Lin, et al. A Vectorization Method of Export Branch for SIMD Extension [C] // 2011 IEEE/ACIS 10th International Conference on Computer and Information Science (ICIS). 2011: 265-269

[16] Nuzman D, Rosen I, Zaks A. Auto-Vectorization of Interleaved Data for SIMD[C]//PLDI. June 2006: 132-143

[17] 魏帅，魏然，侯永生. 面向科学计算程序的向量化[J]. 信息工程大学学报，2011(6): 759-763, 768

[18] Barik R, Zhao Ji-sheng, Sarkar V. Efficient Selection of Vector Instructions using Dynamic Programming[C]//2010 43 rd Annual IEEE/ACM International Symposium on Microarchitectures. 2010