



## 面向国产平台的LLVM自动向量化移植与优化

李嘉楠<sup>1</sup>, 韩林<sup>2</sup>, 柴贻达<sup>1</sup>

(1. 郑州大学 信息工程学院, 郑州 450000; 2. 国家超级计算郑州中心, 郑州 450000)

**摘要:** 作为SIMD扩展部件向量化的重要手段, 自动向量化已在LLVM编译器中得到实现, 但向量长度以及指令集功能的差异, 导致国产平台在自动向量化过程中容易错失向量化机会以及向量化后产生倒加速的问题。为使SIMD得到充分应用, 结合国产平台的指令集特征完善指令代价信息以提高收益分析精准度, 使其在自动向量化后生成后端支持且简洁高效的向量指令。在此基础上, 提出一种改进的控制流向量化方法, 通过添加指令代价信息提高自动向量化的适配能力, 从而形成一套面向国产平台的LLVM自动向量化系统。实验结果表明, 相比自动向量化移植前, 通过该方法进行移植优化后, SPEC测试的整体性能提升10.8%, TSVC测试集中的加速比提升16%, 精准代价指导下的加速比提升42%, 控制流向量化下的加速比提升51%。

**关键词:** 自动向量化; 向量化收益; 移植; LLVM编译器; 国产平台

开放科学(资源服务)标志码(OSID):



**中文引用格式:** 李嘉楠, 韩林, 柴贻达. 面向国产平台的LLVM自动向量化移植与优化[J]. 计算机工程, 2022, 48(1): 142-148.

**英文引用格式:** LI J N, HAN L, CHAI Y D. Automatic vectorization transplant and optimization of LLVM for domestic processors[J]. Computer Engineering, 2022, 48(1): 142-148.

## Automatic Vectorization Transplant and Optimization of LLVM for Domestic Processors

LI Jia'nan<sup>1</sup>, HAN Lin<sup>2</sup>, CHAI Yunda<sup>1</sup>

(1. School of Information Engineering, Zhengzhou University, Zhengzhou 450000, China;

2. National Supercomputing Center in Zhengzhou, Zhengzhou 450000, China)

**[Abstract]** Automatic vectorization is essential in SIMD extension vectorization, and has been implemented in the LLVM compiler. However, the difference of vector length and instruction set functions can cause the domestic processors to lose the opportunity of vectorization in the process of automatic vectorization, or produce negative acceleration after vectorization. To make full use of SIMD, this paper discusses how to improve instruction cost information according to the instruction set features of domestic processors, so the accuracy of benefit analysis is increased. On this basis, precise and efficient vector instructions supported by the back end are generated after automatic vectorization. Furthermore, this paper proposes a vectorization method with improved control flows. By adding instruction cost information, the adaptability of automatic vectorization is improved. Finally a LLVM-based automatic vectorization system for domestic platforms is formed. The experimental results show that for the platforms having received automatic vectorization transplant, the proposed method provides a 10.8% overall performance improvement in SPEC tests, 16% acceleration ratio improvement on the TSVC test, 42% acceleration ratio improvement under the guidance of precision cost, and 51% acceleration ratio improvement under the control flow vectorization.

**[Key words]** automatic vectorization; vectorization cost; transplant; LLVM compiler; domestic processor

DOI: 10.19678/j.issn.1000-3428.0060240

### 0 概述

高性能计算可用于开发解决全球性问题的科学应用程序, 如疫苗研制、气候建模等。如今, 提高处

理器的性能变得越来越重要, 而电子器件的更新已不能满足日益增长的计算需求, 在此情况下, 微型的向量并行部件SIMD(Single Instruction Multiple Data)扩展得到迅速发展, 并被广泛应用于各种科学计算

**基金项目:** 国家重点研发计划“全球对地观测成果管理及共享服务系统关键技术研究”(2018YFB0505000)。

**作者简介:** 李嘉楠(1994—), 女, 硕士研究生, 主研方向为先进编译技术、高性能计算; 韩林, 副教授; 柴贻达, 硕士研究生。

**收稿日期:** 2020-12-09 **修回日期:** 2021-01-19 **E-mail:** lijianan713@163.com

类程序的加速优化任务。

基于SIMD扩展部件的向量化已成为程序并行的的重要手段,其向量寄存器和SIMD指令是程序员以及处理器厂商的研究重点<sup>[1]</sup>。目前,ICC、GCC、LLVM(Low Level Virtual Machine)等编译器已相继支持了SIMD的自动向量化编译,面向SIMD扩展部件的自动向量化编译已逐渐成为程序向量化变换的主要方式。

LLVM是对任意编程语言提供一种基于SSA(Static Single Assignment)<sup>[2]</sup>的静态与动态编译的现代编译技术,与其他主流编译器相比,LLVM具有如下优势:统一的IR(Intermediate Representation)与模块化<sup>[3]</sup>,能够以库的形式抽取其组件并用于其他领域;编译器中的优化和分析被组织成遍(Pass)结构,通过不同遍完成不同的优化算法<sup>[4]</sup>;开源License的优势使其被各大公司广泛采用。在LLVM编译器中,已实现了循环级与基本块级的自动向量化方法<sup>[5]</sup>。

申威系列处理器是我国自主研发、面向高性能计算的通用微处理器,该处理器主要针对计算密集型程序,如科学计算、数字信号分析、多媒体处理等。申威系列处理器包含丰富的向量运算指令和完善的向量重组指令,为程序的向量化提供了良好的硬件基础。

LLVM编译器中自动向量化部分主要由Intel团队开发推进,算法以及向量指令的应用更适用于X86平台<sup>[6]</sup>。目前,LLVM编译器还未支持面向国产平台的自动向量化。由于指令集的差别,申威处理器与X86处理器的自动向量化实现方法有所不同,如AVX指令集兼容256位与128位的向量长度,而国产处理器一般只支持单一的向量长度,导致向量化后生成不支持的向量类型,从而产生段错误、结果错误等问题。

本文针对申威1621处理器平台进行自动向量化移植研究,从循环级与基本块级2个方面提高自动向量化适配能力,以完善向量化所需的指令代价信息,并在精准代价模型的指导下生成后端支持的向量指令。同时,针对循环级向量化中控制流向量化进行算法改进,以解决后端不支持掩码访存指令的问题。

## 1 LLVM自动向量化框架

### 1.1 循环级向量化

循环级向量化主要是在迭代间寻找并行机会以进行向量化,其主要流程如图1所示。

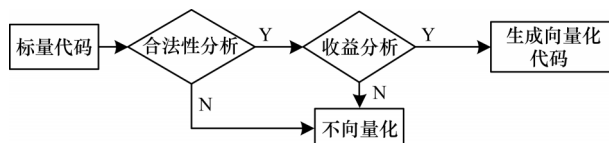


图1 循环级向量化流程

Fig.1 Procedure of cyclic vectorization

合法性分析步骤为:首先检查是否为嵌套循环,排除不能向量化的循环形式,如多出口多回边结构;

然后对包含控制流的循环进行分析,收集分支掩码以用于后续向量代码生成<sup>[7]</sup>;接着对phi指令以及调用指令进行分析,判断其是否符合向量化格式要求;最后调用循环访存信息,分析访存指令是否具有阻止向量化的依赖关系。依赖分析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,不具有强连通分量的语句就是可以进行向量执行的语句<sup>[8-9]</sup>。

通过调用代价模型,首先对基本块中访存指令进行分析,连续访存可直接获取指令代价,非连续访存需要对比不同策略下的收益,选出访存指令向量化的最佳执行方案<sup>[10-11]</sup>;然后将该指令与基本块内其他指令一起,以2的整数幂在 $[2, I_{\text{MaxVF}}]$ 范围内进行收益比较;最终将其与标量代价进行对比,选出最优的向量化因子。

向量代码生成是在原始标量循环结构之前创建一个新的循环,使其成为向量指令的载体。标量循环中不同类型的指令分别调用不同的转换函数以进行向量指令生成,最后将生成的向量指令逐一添加到新的循环中,更新支配关系,原始的标量循环将作为新向量循环的“标量尾循环”处理<sup>[12]</sup>。

### 1.2 基本块级向量化

紧跟在循环级向量化后的是基本块级向量化,其主要在基本块内寻找同构语句以发掘并行机会。基本块级向量化流程如图2所示。

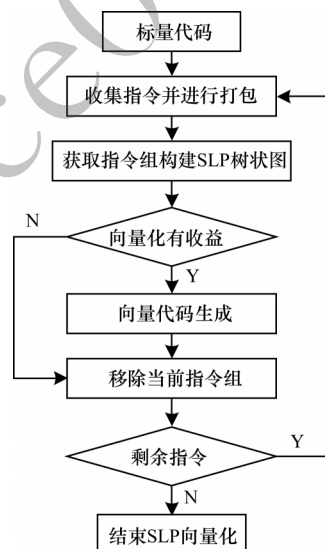


图2 基本块级向量化流程

Fig.2 Procedure of basic-block level vectorization

在基本块级向量化中,首先找到函数中所有的内存引用,收集指令并进行打包<sup>[13]</sup>,包是一个同构语句的集合,将多个同构语句组成包的过程称作打包,相反则称为拆包<sup>[14]</sup>;然后SLP利用相邻地址的存储指令作为种子进行打包,通过“定义-使用链”和“使用-定义链”启发式地扩展包<sup>[15]</sup>,若循环中的每一个操作都可以被目标平台以向量形式支持,则进行语法树的代价分析,在有收益的情况下构建向量化树,从上到下扫描基本块的所有语句,在需要向量化的

标量语句前插入向量语句,以完成向量代码生成<sup>[16]</sup>;最后移至下一组指令重复以上分析,直至完成基本块内所有指令的向量发掘。

## 2 自动向量化移植

近年来,作为编译优化领域的研究热点,SIMD扩展部件不断发展。申威系列处理器的SIMD向量长度在持续增长,指令集功能也越来越丰富,因此,需要针对每一种处理器实现其自动向量化功能移植。移植主要包含自动向量化优化遍、后端相关转换信息2个方面。自动向量化可分为识别、优化、指令生成3个部分。识别方法在循环级向量化与基本块级向量化中通用,最重要的是考虑SIMD部件差异与指令集特征,其中,寄存器信息、跨幅因子、基本指令代价的精确描述是自动向量化的基础条件。另外,本文提出一种掩码指令转换方法,使得申威平台支持包含控制流结构的向量化。

### 2.1 寄存器信息

完善SIMD向量化的寄存器基础信息包含RegisterInfo文件中向量寄存器数量、特征信息以及TargetTransformInfo文件中寄存器宽度描述,必要时数据类型长度也需要根据指令集信息进行修改<sup>[17]</sup>,否则会生成后端不支持的向量长度或向量类型,增加后端指令降级工作从而导致向量代码生成效率降低。

### 2.2 跨幅因子

跨幅因子即循环级向量化中的“Interleave Count”,为基本块中单条语句的展开数,默认值为1。结合向量指令特征,将TargetTransformInfo中最大跨幅因子设置为4,向量化阶段调用代价模型从[1,4]范围内分析出最佳跨幅因子,与原始默认值1相比,提升了向量化性能。以TSVC(Test Suit for Vectorizing Compilers)测试集中的vpvts函数为例,移植后进行收益分析,选择出最佳跨幅因子为4,以此进行向量代码的局部展开,相比原始向量化其性能提升了70%。

### 2.3 基本指令延迟信息

基本指令包含逻辑运算指令、类型转换指令、比较指令、内建函数指令、访存指令等。根据硬件提供的指令延迟表,在后端TargetTransformInfo文件中对指令代价进行精确描述,包含数据类型、操作码识别、指令延迟数补充。将后端不支持的向量指令代价调高,防止向量化后产生倒加速问题。对于复杂指令如混洗指令,结合后端指令降级中自定义的指令拆分组合情况进行精确描述。

X86平台支持128位的向量寄存器<sup>[18]</sup>,在自动向量化中最小向量化因子限制为2,但这在申威平台不适用,原因是会造成数据处理过程中读取错误信息,在程序运行时引发段错误。基于上述原因,本文分别在循环级向量化与基本块级向量化中修改最小向量化因子。

### 2.4 掩码内建指令

掩码内建函数是对LLVM基本指令集的补充,

由特定目标平台的特殊指令组合而成,但并不是所有目标架构指令集都具备全面的掩码指令<sup>[18-19]</sup>。在申威平台下进行SIMD编译时,自动向量化并不支持掩码访存指令,导致大量包含控制流的循环错失向量化机会。在循环级向量化时将掩码访存指令替换为select向量指令,可解决目标平台不支持掩码访存指令的问题,核心转换算法描述如算法1所示。

#### 算法1 掩码指令转换算法

输入 最内层循环体IF结构

输出 由IF结构中的掩码访存指令转换成的select指令

```
1.procedure TranslateIfToSelect()
2.get(cmp->getCmp)//获取分支条件cmp;
3.mask=get(cmp);//获取掩码
4.if mask存在
5.//将掩码存储指令替换成以下的select指令
6.//获取参数进行指令构建
7.LI=CreateLoad(DataTy, VecPtr);
//创建load语句
8.SelectI=CreateSelect(mask, StVal, LI);
//创建select语句
9.NewSI=CreateStore(SelectI, VecPtr);
//构建新的store语句
10.end if
11.NewSI(store);//指令替换
12.return;
13.//更新NewSI指令代价计算;
14.CostNewSI+=CostLoad+CostSel;
15.}
```

## 3 收益评估模型

循环或函数被向量化后的基本收益就是基本块中减少的指令周期数,收益评估用于衡量向量化是否有利于提高程序效率<sup>[7]</sup>。自动向量化移植使得收益评估更加精准与完善,可以在收益评估的指导下判断是否需要向量化以及如何向量化,从而生成最符合申威后端需求的向量或标量代码。

### 3.1 面向循环级向量化的收益评估

循环级向量化收益分析流程如图3所示。

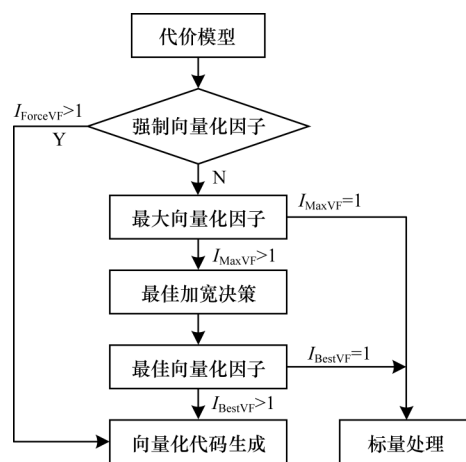


图3 循环级向量化收益分析流程

Fig.3 Procedure of cost analysis in cyclic vectorization



收益分析主要集中在2个部分: 针对访存指令的最佳加宽决策; 针对循环基本块的最佳向量化因子选择。具体步骤如下:

1) 计算可行的最大向量化因子  $I_{\text{MaxVF}}$ , 获取后端向量寄存器长度  $D_{\text{WidthRegister}}$  以及数据宽度  $D_{\text{WidthType}}$  信息, 对于只支持单一向量长度的后端:

$$I_{\text{MaxVF}} = D_{\text{WidthRegister}} / D_{\text{WidthType}} \quad (1)$$

2) 访存指令对程序的向量化或性能提升起决定性作用, 对于访存指令, 循环级向量化具有专属的决策分析以求得最大收益。对于连续访存代价, 直接从后端指令延迟表获取; 对于不连续访存, 比较跨幅访存、聚合访存、标量化访存3种方案下的收益以选取最佳方案进行执行。

(1) 跨幅访存代价首先获取访存指令的代价  $C_{\text{MemoryOp}}^{\text{Cost}}$ , 在此基础上累加跨幅访存中使用到的额外指令代价  $C_{\text{ExtraInsert}}^{\text{Cost}}$ , 包含插入与抽取指令。load与store指令的计算方式稍有区别, 跨幅load指令从源向量中提取子向量的元素, 然后将它们插入子向量中, 例如:

```
%vec = load <8 x i32>, <8 x i32>* %ptr
%v0 = shuffle %vec, undef, <0, 2, 4, 6>
```

额外代价来源于从<8 x i32>向量中的0、2、4、6位抽取数据并插入到<4 x i32>向量过程中的插入抽取指令, 设  $G_{\text{GroupNums}}$  为跨幅访存组指令数量, 其额外代价为:

$$C_{\text{ExtraInsert}}^{\text{Cost}} = C_{\text{Extra}}^{\text{Cost}} + G_{\text{GroupNums}} \cdot C_{\text{Insert}}^{\text{Cost}} \quad (2)$$

跨幅store指令是从子向量中提取所有元素, 并将它们插入到目的向量中, 例如:

```
%v3 = shuffle %v0, %v1, <0, 4, 1, 5, 2, 6, 3, 7>
store <8 x i32> %v3, <8 x i32>* %ptr
```

额外代价来源于从2个<4 x i32>向量中抽取8个元素插入到<8 x i32>向量过程中的插入抽取指令代价, 设跨步为  $S_{\text{Steps}}$ , 其额外代价为:

$$C_{\text{ExtraInsert}}^{\text{Cost}} = C_{\text{Extra}}^{\text{Cost}} \cdot S_{\text{Steps}} + C_{\text{Insert}}^{\text{Cost}} \quad (3)$$

(2) 聚合访存代价的计算方式较为简单, 设  $C_{\text{address}}^{\text{Cost}}$  为地址计算代价,  $C_{\text{gatherscatter}}^{\text{Cost}}$  为聚合指令代价, 则聚合访存代价为:

$$C^{\text{Cost}} = (C_{\text{address}}^{\text{Cost}} + C_{\text{gatherscatter}}^{\text{Cost}}) \cdot G_{\text{GroupNums}} \quad (4)$$

对于不支持聚合指令的后端,  $C_{\text{gatherscatter}}^{\text{Cost}}$  可手动调整到较大值, 以防止产生不必要的指令。

(3) 标量化访存代价首先获取标量存储指令和地址计算:

$$C^{\text{Cost}} = V_{\text{VF}} \cdot C_{\text{address}}^{\text{Cost}} + V_{\text{VF}} \cdot C_{\text{MemoryOp}}^{\text{Cost}} \quad (5)$$

包含控制流的循环属于不连续访存, 掩码指令转换算法将包含控制流的循环进行select指令转换, 改变包含控制流访存收益分析的计算方式, 如下:

$$C^{\text{Cost}} = C_{\text{Load}}^{\text{Cost}} + C_{\text{Select}}^{\text{Cost}} + C_{\text{Store}}^{\text{Cost}} \quad (6)$$

在更新包含控制流的代价计算后, 根据基本块执行的概率来扩展代价<sup>[20]</sup>, 循环级向量化认为每个基本块执行的概率均为50%, 因此, 分支基本块的代

价为  $C^{\text{Cost}} = C^{\text{Cost}} / 2$ 。

3) 将循环中所有指令进行累加, 对比以上3种不连续访存的收益, 选择最优收益方案进行下一步向量因子的收益分析。

最后对比标量与向量形式下的收益, 选择进行向量代码生成或保持原有的标量执行。

### 3.2 面向基本块级向量化的收益评估

SLP向量化收益开销计算算法以抽象语法树为基础, 满足  $E \geq V_{\text{VF}}$  条件后以存储指令为根节点遍历树形结构, 通过“使用-定义链”自下而上进行打包, 其中, 每个节点都包含可并行处理且数目相同的同构语句<sup>[21-22]</sup>。

代价模型获取寄存器信息计算向量化所需的同构语句数, 根据指令代价选择收益最大的进行向量化。首先确定同构语句长度(即同构语句数量)与向量化因子: 设同构语句长度为  $E$ , 向量化因子为  $V_{\text{VF}}$  (设向量寄存器长度为256位, 标量元素double为64 bit, 则向量化因子  $V_{\text{VF}}$  等于4)。假设标量指令代价为  $S_i$ , 向量指令代价为  $V_{\text{Cost}}$ , 则每条向量指令的收益为:

$$C^{\text{Cost}} = V_{\text{Cost}} - V_{\text{VF}} \cdot S_i \quad (7)$$

该基本块向量的收益总和为:

$$C^{\text{Cost}} = \sum_{i=1}^N C_{\text{InstVec}[i]}^{\text{Cost}} \quad (8)$$

考虑寄存器溢出与指令重组对向量化性能存在影响, 因此, 自底向上遍历语法树以计算额外开销  $C_{\text{extra}}^{\text{Cost}}$ , 其值在理想状态下为0, 则总体收益为:

$$C_1^{\text{Cost}} = C_N^{\text{Cost}} + C_{\text{extra}}^{\text{Cost}} \quad (9)$$

另外一些计算如标量的规约计算、向量化指针指令的索引计算, 没有被基于同构store组的自底向上的遍历过程所捕获, 因为它们不包含这样的存储指令, 此类收益分析需考虑插入/抽取指令以及索引信息的代价  $C_{\text{user}}^{\text{Cost}}$  (从后端相关指令延迟表中获取, 默认值为0), 则总体收益为:

$$C_2^{\text{Cost}} = C_1^{\text{Cost}} + C_{\text{user}}^{\text{Cost}} \quad (10)$$

将基本块总收益  $C_1^{\text{Cost}}$  或  $C_2^{\text{Cost}}$  与阈值(默认值为0)进行对比, 若有收益, 则调用函数完成向量代码生成; 否则, 放弃SLP向量化。

## 4 测试分析

本文采用申威1621处理器为测试平台, 进行LLVM编译器自动向量化移植功能测试, 编译器版本为7.0。分别采用SPEC2006与TSVC测试集以及向量化应用测试, 从正确性与向量化性能2个方面进行对比分析。

### 4.1 功能测试与分析

SPEC2006标准测试集是SPEC组织推出的CPU子系统评估软件。为验证本文方法改进下编译器的健壮性, 对SPEC2006标准测试集中的29道题进行测试, 结果如表1所示。

表1 SPEC测试结果  
Table 1 SPEC test results

程序名称	编程语言	自动向量化 移植前	自动向量化 移植后
400.perlbench	C	通过	通过
401.bzip2	C	通过	通过
403.gcc	C	通过	通过
410.bwaves	F77	结果错	通过
416.games	Fortran	结果错	通过
429.mcf	C	通过	通过
433.milc	C	通过	通过
434.zeusmp	F77	通过	通过
435.gromacs	C, Fortran	通过	通过
436.cactusADM	F90, C	通过	通过
437.leslie3d	F90	通过	通过
444.namd	C++	通过	通过
445.gobmk	C	结果错	通过
447.dealII	C++	通过	通过
450.soplex	C++	通过	通过
453.povray	C++	通过	通过
454.calculix	F90, C	结果错	通过
456.hmmer	C	通过	通过
458.sjeng	C	通过	通过
459.GemsFDTD	F90	结果错	通过
462.libquantum	C	通过	通过
464.h264ref	C	段错	通过
465.tonto	F95	编译错	通过
470.lbm	C++	通过	通过
471.omnetpp	C++	通过	通过
473.astar	C++	通过	通过
481.wrf	F90, C	段错	通过
482.sphinx3	C	结果错	通过
483.xalancbmk	C++	通过	通过

移植前中间表示代码在自动向量化阶段生成了后端不支持的向量类型,在后端指令降级过程中找不到匹配的降级方法与指令模板,导致测试题出现段错误以及结果错误的问题。从表1可以看出,移植优化后410、416、454等测试题依靠后端信息的精准描述以及收益分析的正确引导,向量化程序在test、train、ref规模下正确运行。

#### 4.2 精准代价指导下的测试分析

基本运算指令代价的精准描述,可以使得自动向量化做出符合后端要求的向量化决策,生成简洁高效的向量汇编指令。本文采用TSVC测试集中的典型例题,对比移植优化前后的加速性能,结果如图4所示,从图4可以看出,相对移植前,移植优化后平均加速比提升42%。

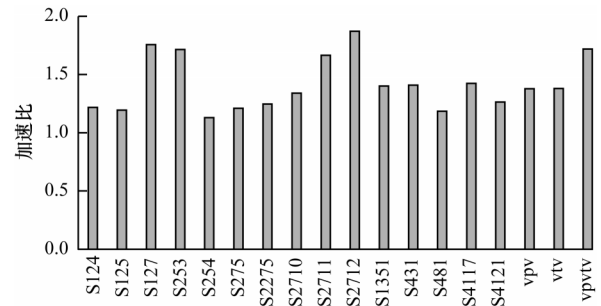


图4 精准代价指导下的加速性能

Fig.4 Accelerated performance under precise cost guidance

当后端对不支持的向量指令进行降级处理时,会生成冗余标量指令导致倒加速。精准代价模型指导下的自动向量化可排除后端不支持的向量类型,防止向量化倒加速产生。从图5可以看出,使用精准代价指导后,选择与后端匹配的向量化因子以及向量化方法,可以使倒加速情况得到明显改善,在原有移植的基础上平均加速比提升28%。

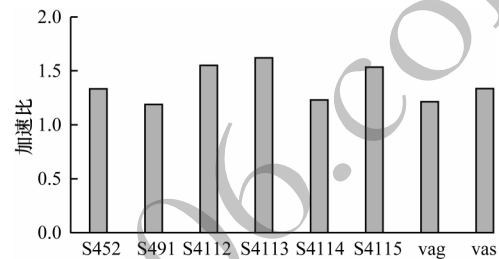


图5 精准代价指导下的倒加速改善结果

Fig.5 Reverse acceleration improvement results under precise cost guidance

#### 4.3 性能测试与分析

本文分别采用SPEC CPU2006测试题、TSVC测试集以及被广泛应用的矩阵运算测试题进行性能测试。

##### 4.3.1 SPEC测试分析

本次实验对SPEC测试集的29道题进行性能测试分析,编译优化选择-O3-static,ref规模,单进程运行。选取代表性测试题进行分析,对比X86平台与国产平台的向量化能力,从而验证移植的有效性。测试结果如图6所示,其中,X86选用与申威SIMD长度相同的AVX指令集。

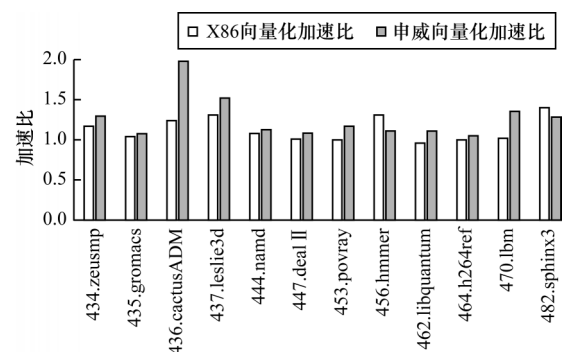


图6 SPEC测试分析结果

Fig.6 SPEC test analysis results

前端将测试题解析为中间表示代码, 自动向量化在此基础上进行向量代码优化生成, 然后由后端进行降级处理, 生成特定指令集的汇编码。从图6可以看出, 移植优化后的向量代码简洁高效且符合后端指令集特征需求, 移植后SPEC整体性能提升10.8%, 国产平台的向量化能力优于X86, 其中, 436加速效果最为明显, 提升了97%, 437加速比提升52%, 434、470、482加速比平均提升21%。由于指令集存在差异, 国产处理器不支持一些特殊的向量指令, 导致其对456与482的加速性能低于X86。

#### 4.3.2 TSVC测试分析

TSVC测试集主要用来对编译器的自动向量化能力进行性能测试, 本文采用TSVC测试集对比移植优化前后的加速性能, 测试结果表明, 移植优化后整体加速比提升16%。对掩码内建指令进行相关修改, 可以使得后端兼容原本不支持的向量化实现方法, 从而充分利用申威后端的向量指令。从图7可以看出, 在控制流优化下, 循环识别率提升48%, 平均加速比提升51%。

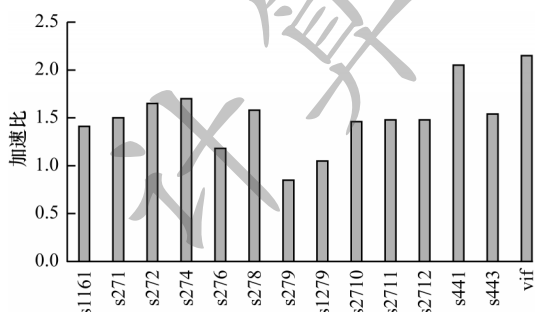


图7 控制流优化下的加速比结果

Fig.7 Acceleration ratio results under control flow optimization

#### 4.3.3 应用测试分析

快速傅立叶变换被广泛应用于信号分析任务, 其性能提升离不开矩阵运算优化, 另外, 图形处理、游戏开发、科学计算中包含了大量的矩阵运算, 因此, 本次实验以矩阵乘为代表, 进行应用程序测试分析。实验采用3层循环, 以3种 $N \times N$ 规模的矩阵进行测试, 每个规模运行3次并取平均值, 以对比移植优化前后的运算性能, 结果如图8所示。从图8可以看出, 矩阵乘运算总体性能提升了72%, 矩阵规模为 $1\,024 \times 1\,024$ 、 $2\,048 \times 2\,048$ 、 $4\,096 \times 4\,096$ 时, 加速比分别提升了77%、79%、59%。矩阵乘性能收益主要来源于合适的跨幅因子与向量化因子, 其能够最大程度地发挥SIMD向量指令的优势。

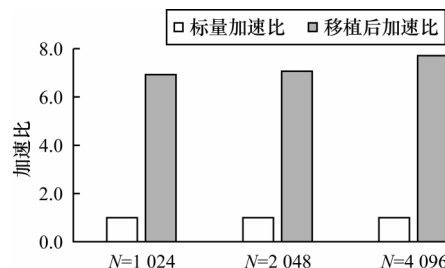


图8 矩阵乘运算性能分析

Fig.8 Performance analysis of matrix multiplication operation

## 5 结束语

本文针对申威1621处理器进行LLVM自动向量化功能移植, 以解决自动向量化过程中后端信息不匹配和向量化实现方法不兼容的问题。实验结果表明, 移植优化后不再生成后端不支持的向量类型以及不合法的向量指令, 控制流向量化程度显著提升, 在TSVC测试集中, 相对自动向量化移植前, 移植优化后平均加速比提升16%。通过对自动向量化移植进行研究可以看出, 在申威后端存在不支持某些相对重要的向量指令的情况, 这将严重影响向量化的效果。因此, 下一步将综合考虑申威指令集与自动向量化后代码生成的关系, 将申威指令集中的基本向量指令进行拼接与重组, 从而提高向量指令的适用性。

## 参考文献

- [1] 高伟, 赵荣彩, 韩林, 等. SIMD自动向量化编译优化概述[J]. 软件学报, 2015, 26(6): 1265-1284.  
GAO W, ZHAO R C, HAN L, et al. Research on SIMD auto-vectorization compiling optimization[J]. Journal of Software, 2015, 26(6): 1265-1284. (in Chinese)
- [2] ZHOU H, XUE J L. Exploiting mixed SIMD parallelism by reducing data reorganization overhead[C]//Proceedings of 2016 International Symposium on Code Generation and Optimization. New York, USA: ACM Press, 2016: 59-69.
- [3] PANDEY M, SARDA S. LLVM cookbook[M]. Packt Publishing Ltd., [s. n.], 2015.
- [4] RALF K, SEBASTIAN H. Whole-function vectorization[C]//Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization. Washington D. C., USA: IEEE Press, 2011: 141-150.
- [5] TIAN X M, SAITO H, SU E, et al. LLVM framework and IR extensions for parallelization, SIMD vectorization and offloading[C]//Proceedings of the 3rd Workshop on LLVM Compiler Infrastructure in HPC. Washington D. C., USA: IEEE Press, 2016: 21-31.
- [6] DORIT N, IRA R, AYAL Z. Auto-vectorization of interleaved data for SIMD[J]. Association for Computing Machinery, 2006, 6: 132-143.
- [7] PETROGALLI F, WALKER P. LLVM and the automatic vectorization of loops invoking math routines: -fsimdmath[C]//Proceedings of 2018 IEEE/ACM Workshop on the LLVM Compiler Infrastructure in HPC. New York, USA: ACM Press, 2018: 30-38.



- [ 8 ] PORPODAS V. SuperGraph-SLP auto-vectorization[C]// Proceedings of 2017 International Conference on Parallel Architecture and Compilation. Washington D. C. , USA: IEEE Press, 2017: 330-342.
- [ 9 ] PORPODAS V, ROCHA R C O, LUÍS F W. Look-ahead SLP: auto-vectorization in the presence of commutative operations[C]//Proceedings of International Symposium on Code Generation and Optimization. Washington D. C. , USA: IEEE Press, 2018: 163-174.
- [ 10 ] HAO Z, XUE J L. A compiler approach for exploiting partial SIMD parallelism [J]. ACM Transactions on Architecture and Code Optimization, 2016, 13(1): 1-26.
- [ 11 ] MOLDOVANOVA O V, KURNOSOV M G, MELNIKOV A. Energy efficiency and performance of auto-vectorized loops on Intel Xeon processors[C]//Proceedings of 2018 Russian-Pacific Conference on Computer Technology and Applications. Washington D. C. , USA: IEEE Press, 2018: 1-6.
- [ 12 ] 李威, 梁军, 张桢, 等. 基于 ARM GPU 的机载 SAR 成像算法并行优化策略[J]. 计算机工程, 2020, 46(10): 240-247.  
LI W, LIANG J, ZHANG Z, et al. Parallel optimization strategy of airborne SAR imaging algorithm based on ARM GPU[J]. Computer Engineering, 2020, 46(10): 240-247. (in Chinese)
- [ 13 ] VASILEIOS P, RODRIGO C O R, LUÍS F W G. VW-SLP: auto-vectorization with adaptive vector width[C]//Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques. New York, USA: ACM Press, 2018: 4-15.
- [ 14 ] ZHOU H, XUE J L. A compiler approach for exploiting partial SIMD parallelism [J]. ACM Transactions on Architecture and Code Optimization, 2016, 13(11): 26-35.
- [ 15 ] RODRIGO C O R, VASILEIOS P, PAVLOS P, et al. Vectorization-aware loop unrolling with seed forwarding [C]//Proceedings of the 29th International Conference on Compiler Construction. New York, USA: ACM Press, 2020: 1-13.
- [ 16 ] ZHOU H, XUE J. Exploiting mixed SIMD parallelism by reducing data reorganization overhead[C]//Proceedings of 2016 International Symposium on Code Generation and Optimization. New York, USA: ACM Press, 2016: 59-69.
- [ 17 ] SIMON M, SHREY S, MATTHIAS K, et al. Multi-dimensional vectorization in LLVM[C]//Proceedings of the 5th Workshop on Programming Models for SIMD/Vector Processing. New York, USA: ACM Press, 2019: 1-8.
- [ 18 ] ANDREW A, AVINASH M, DAVID G. Automatic vectorization of interleaved data revisited [J]. ACM Transactions on Architecture and Code Optimization, 2016, 13(2): 1-25.
- [ 19 ] LIU J, ZHANG Y R, JANG O Y, et al. A compiler framework for extracting superword level parallelism[C]// Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA: ACM Press, 2012: 347-358.
- [ 20 ] BOHM C, PLANT C. Mining massive vector data on single instruction multiple data microarchitectures[C]//Proceedings of 2015 IEEE International Conference on Data Mining Workshop. Washington D. C. , USA: IEEE Press, 2015: 597-606.
- [ 21 ] PORPODAS V, ROCHA R, BREVN OV E, et al. Super-node SLP: optimized vectorization for code sequences containing operators and their inverse elements [C]// Proceedings of 2019 IEEE/ACM International Symposium on Code Generation and Optimization. Washington D. C. , USA: IEEE Press, 2019: 206-216.
- [ 22 ] YAO J Y, ZHAO R C, WANG Q, et al. Loop-nest auto-vectorization method based on benefit analysis [C]// Proceedings of the 2nd International Conference on Advances in Image Processing. New York, USA: ACM Press, 2018: 240-244.

编辑 吴云芳