自动向量化：近期进展与展望

冯竞舸[1,2]，贺也平[1,2,3]，陶秋铭[1,2]

（1. 中国科学院软件研究所基础软件国家工程研究中心，北京 100190；2.中国科学院大学研究生院，北京 100049；
3. 中国科学院软件研究所计算机科学国家重点实验室，北京 100090）

摘 要：随着单指令流多数据流（SIMD）技术的迅速发展，近年来许多面向 SIMD 扩展部件的自动向量化编译方法被提出，有效缓解了程序员手写向量程序的压力，并发挥了 SIMD 扩展部件的加速效能。基于此，分析总结了自动向量化领域近 10 年的研究成果，从保义分析和变换、向量化分组分析和变换、面向处理器支持特性的分析和变换以及性能评估分析这 4 个方面分类归纳了自动向量化的关键问题和主要突破，进而对 4 个方面的发展趋势和研究方向进行了展望。

关键词：自动向量化；SIMD 扩展；编译技术；数据级并行；性能优化

中图分类号：TP312

文献标志码：A

DOI: 10.11959/j.issn.1000−436x.2022051

Auto-vectorization: recent development and prospect

FENG Jingge[1,2], HE Yeping[1,2,3], TAO Qiuming[1,2]

1. National Engineering Research Center for Fundamental Software, Institute of Software Chinese Academy of Sciences, Beijing 100190, China
2. Graduate University, University of Chinese Academy of Sciences, Beijing 100049, China
3. China State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing 100090, China

Abstract: The technology of SIMD is developing rapidly, and quite a few auto-vectorization methods have been proposed. Auto-vectorization can automatically translate scalar programs into vector programs based on SIMD extension, decrease workload of the programmers in coding vector programs, and effectively improve performance of programs. Based on that, the research achievements in the field of automatic vectorization in recent 10 years were analyzed and summarized. The key problems and major breakthroughs in automatic vectorization were classified from four aspects: semantic-maintaining analysis and transformation, vectorization grouping analysis and transformation, processor-oriented analysis and transformation, and performance evaluation analysis. Furtherly, the development trends and research directions of the four aspects were prospected.

Keywords: auto-vectorization, SIMD extension, compiling technology, data level parallelism, performance optimization

0 引言

芯片和基础软件的发展，不仅影响国家信息安全[1]，也影响产业供应链安全。编译器作为基础软件之一，对发挥芯片特性、提升程序性能至关重要，国内越来越多的企业和科研机构高度重视编译技术研发，例如华为、阿里巴巴、腾讯、中国科学院、国防科学技术大学、清华大学、中国科学技术大学、上海交通大学、武汉大学，等等。

自动向量化是一种重要的编译优化方法，它利用单指令流多数据流（SIMD, single-instruction stream multiple-data stream）系统扩展部件[2]提供的数据并行处理能力，有效提升程序性能，在数字信号处理[3]、大数据[4]、人工智能[5]、高性能计算[6]等众多应用场景[7-10]中发挥着重要的作用。

自动向量化不仅可显著提升程序性能[11-12]，也

收稿日期：2021−12−06；**修回日期**：2022−02−09

基金项目：中国科学院战略性先导科技专项基金资助项目（No.XDA-Y01-01, No.XDC02010600）

Foundation Item: The Strategic Priority Research Program of Chinese Academy of Sciences (No.XDA-Y01-01, No.XDC02010600)

能降低程序功耗[13]。与程序员以手动方式编写 SIMD 向量程序（如利用内嵌汇编[14]、内部函数[15]、SIMD 函数库等）相比，自动向量化能将程序员编写的标量程序自动转换为 SIMD 向量程序，不需要程序员深入理解 SIMD 扩展部件的功能和特性，从而减少程序员的负担。

人们对计算性能的不懈追求和 SIMD 扩展部件的迅速发展推动着自动向量化技术的进步。SIMD 扩展部件的应用并不局限于多媒体应用[16]中，还大量应用于大数据、人工智能、云计算等科学计算领域和访存密集型应用领域。处理器对 SIMD 扩展部件的支持是自动向量化的硬件基础。目前绝大部分处理器都支持 SIMD 扩展部件，例如 Intel 推出的 x86 AVX512 指令集，ARM 推出的 SVE[17]指令集和美国加州大学推出的 RISC-V[7]指令集等。随着硬件处理器技术的快速发展，SIMD 扩展部件支持的并行长度越来越长，功能越来越丰富[18]。如何充分利用这些功能日渐强大的 SIMD 扩展部件，给自动向量化带来新的挑战，并推动着自动向量化技术的进步。然而，尽管目前主流编译器如 GCC、LLVM 和 ICC 等都初步实现了自动向量化功能，但仍存在较大的提升空间。国内外研究人员对 ICC 和 GCC 的自动向量化功能评测后发现，自动向量化获得的性能与手工向量化相比，依然存在较大的差距[11,19]。

近年来，大量的自动向量化研究成果被公布。为了对该研究问题的进展进行系统梳理和归纳总结，本文搜集和分析了 2011—2021 年关于自动向量化的重要文献。

早在 2007 年，张为华等[20]针对自动向量化的研究进展进行了综述。2015 年，高伟等[2]也对相关研究进行了成果综述。2015 年以来，自动向量化领域出现了一系列新的问题和研究成果，例如运算类型语句的自动向量化[21]（CGO2015），基于特殊 SIMD 指令的自动向量化[22]（ASPLOS 2021）等，以及新的解决方法，例如利用机器学习方法解决自动向量化问题[23]（CGO2020），利用部分线性化技术解决分支判定语句的自动向量化问题（PLDI2018）[24]等。本文旨在对近年的自动向量化技术研究成果进行更加全面的分析和归纳总结。基于目前该领域的研究现状，展望了该领域的发展趋势，总结了可能的研究方向，为未来的研究提供参考。

鉴于已有的综述论文，本文对部分较早期的成果不做赘述，更聚焦于 2015 年之后的研究成果。当然由于个别类型的自动向量化方法是基于早期研究成果（2015 年之前），为了完整性和便于读者理解，本文对部分相关研究也会做适当的介绍和分析。

1　自动向量化问题概述

1.1　自动向量化问题抽象描述

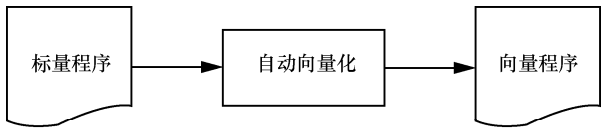
自动向量化问题可描述为：在保证转换前后语义具有可观察等价性[25]，满足处理器支持 SIMD 扩展部件的限定下，评估有性能收益时，利用编译器将标量程序转换为向量程序，有效提升程序性能，其具体解释如下。

1) 保持程序转换前后语义“可观察等价性”是保证自动向量化正确性的前提条件。Plotkin[25]给出了可观察等价性的定义，即对于 2 个表达式 M 和 N，当且仅当在 M 和 N 均为封闭（即没有自由变量）的上下文 C 中，对 $C[M]$和 $C[N]$求值，两者或者产生相同的结果或者均不停止时，那么 M 和 N 是可观察等价的。

2) 自动向量化要在处理器支持特性的限制条件下生成 SIMD 扩展指令，不能生成处理器不支持的 SIMD 扩展指令。

3) 自动向量化目标是提升程序运行的性能。只有当编译器判定自动向量化有收益时，才进行向量化。

自动向量化问题描述如图 1 所示，限制条件 1 中的 Origin_Semantics 和 Sem 分别是向量化转换前后的程序语义，它们必须是可观察等价的。限制条件 2 中 SimdInst 是向量化后生成的向量指令集合，Support_SimdInst 是 SIMD 扩展部件支持的指令集合。SimdInst 必须包含于 Support_SimdInst。限制条件 3 中，性能评估自动向量化有收益，自动向量化的目标就是在保证限制条件 1~3 下，生成向量指令，以有效提升程序性能。



限制条件1：Sem==Origin_Semantics
限制条件2：SimdInst∈Support_SimdInst
限制条件3（目标）：有效提升程序性能（Performance）

图 1　自动向量化问题描述

下面用一个示例说明上述描述。如图 2 所示，图 2(c)中的向量加载指令（vload）、向量存储指令（vstore）和向量加法指令（vadd）在多数处理器中都支持。图 2(a)程序执行时需 4 个加载访存指令（load）、2 个加法指令（add）和 2 个存储访存指令（store），图 2(b)程序执行时（如图 2(c)汇编代码）只需要 2 个 vload、一个 vadd 和一个 vstore，比图 2(a)程序的代价低。并且，图 2(b)与图 2(a)程序语义一致，处理器支持所有生成的 SIMD 指令，且可有效提升性能。因此，图 2(a)标量程序可转换为图 2(b)向量程序。

$$a[0] = b[0] + c[0]$$
$$a[1] = b[1] + c[1]$$
$$\Rightarrow \quad a[0:1] = b[0:1] + c[0:1]$$

（a）标量程序　　（b）向量程序

```
vload R1,&(b+0);/*vload (b[0–1]) */
vload R2,&(c+0);/*vload (c[0–1]) */
vadd R2,R1;/*vadd (b[0–1],c[0–1]) */
vstore & (a+0),R2;/*vstore (a[0–1]) */
```

（c）向量汇编程序

图 2　自动向量化示例

1.2　自动向量化关键问题分类描述

基于自动向量化问题的“语义可观察等价性”和“性能有收益”等限制条件分析，自动向量化的关键问题可分类归纳为以下 4 个方面。

1) 自动向量化与编译器的保义分析和变换能力相关，保义分析和变换能力越强，向量化的适用范围越大。向量化从程序执行顺序角度讲，本质上是将原始串行程序转换为局部并行程序，在同一 SIMD 向量指令中运行的操作不能有依赖关系，否则将改变原始程序语义。面向向量化的保义分析的核心是依赖关系分析。依赖关系分析与别名关系分析紧密相关，别名关系分析能力的不足有时会阻碍依赖关系的判定。

2) 自动向量化与其自身的分组方法有关。将多个标量数据转为一个向量数据的过程称为分组。分组直接影响向量化的收益。根据程序分析粒度，向量化分组分析和变换问题可分为基本块级、循环级、函数级[2]和混合级向量化分组问题。基本块级分组问题旨在寻找基本块内向量化分组，循环级分组问题旨在寻找包含循环的程序中迭代间语句分组，函数级分组问题旨在寻找不同函数之间的语句分组，混合级分组问题是前面 3 种分组方式的扩展及融合问题。

3) 向量化与处理器特性相关。向量化从使用指令角度讲，本质是利用 SIMD 扩展部件的运算并行和访存并行特性提升程序性能。在运算并行特性方面，目前大多处理器只支持相同类型运算并行的向量指令。在访存并行特性方面，大多处理器只支持对齐/连续访问指令[26]，或者尽管支持非对齐/非连续指令，但其指令时延较大。在并行度支持特性方面，大多处理器只支持处理长度为 2 的整数次幂[27]的向量。由于上述限制因素，编译器不能直接使用 SIMD 扩展指令，对不满足上述特征的程序进行向量化。本文将面向处理器支持特性的分析和变换问题分为运算并行相关问题、内存访问并行相关问题和并行长度相关问题。

4) 向量化与性能评估分析相关。性能评估最终决定编译器是否进行向量化。性能与处理器支持的指令时延、访存模型和寄存器相关特性紧密相关。设计精确的向量化代价评估模型，能够有效指导向量化的程序变换，并减少因不恰当的程序变换导致程序性能下降的情形。图 3 展示了自动向量化流程与上述 4 个关键问题的关系。

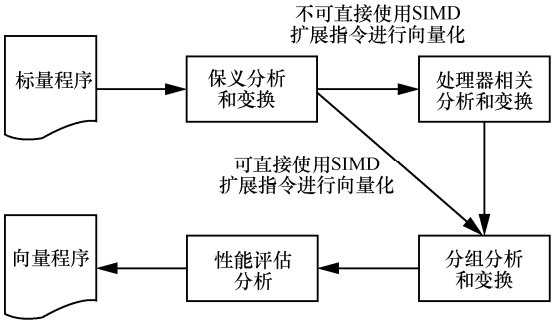


图 3　自动向量化流程

根据上述分析，向量化问题可归纳为 4 类：保义分析和变换问题、分组分析和变换问题、处理器相关分析和变换问题以及性能评估分析问题。本文从这 4 个角度，对近几年自动向量化的研究成果进行分析和总结。

2　保义分析和变换问题及方法

保义分析和变换是编译优化的基础和前提。几乎所有的编译器优化方法都涉及保义分析和变换问题。本文只关注自动向量化的保义分析和变换问题。目前面向自动向量化的保义分析和变换的研究集中于依赖和别名的分析和变换。

2.1　依赖分析和变换

2.1.1　数据依赖分析和变换

数据的访问和重用常引入数据依赖，如图 4 所示，当两条语句访问相同的存储单元，且其中有一个写数据，则发生数据依赖。

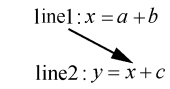


图 4　数据依赖说明

近几年对于数据依赖分析，学术界有三方面的主要突破。在研究对象方面，从原始的简单依赖关系的判定，逐渐转向多层嵌套复杂依赖关系的判定；在理论方法方面，从传统静态分析方法，转向动态投机执行的方法，从软件表达式等价变换的方法，转向基于特殊硬件处理的方法；在分析和转换流程方面，从传统的“分析–判定”流程模式，转向“分析–转换–判定”迭代模式。

起初，向量化主要采用编译器中通用的依赖分析和变换方法，例如 GCD 测试[28]和 Banerjee 测试[29]等，这些方法没有考虑 SIMD 的特点，如 SIMD 扩展部件的并行长度有限，在循环中，当操作的依赖距离大于其并行长度时，把这些操作存放到 SIMD 寄存器中，不改变原始程序语义。Bulić 等[30]基于 Banerjee 方法将 SIMD 扩展部件并行长度信息应用于依赖测试中，扩展面向向量化的依赖测试识别范围，有效解决在循环中依赖距离大于 SIMD 并行化长度的数据依赖分析问题。

传统编译器的依赖测试主要基于静态分析的方法，然而有时候该方法不能准确判定语句的依赖关系。2017 年，Jensen 等[31]首次提出根据原始程序中已有的 OpenMP 并行编译指示信息，指导面向向量化的依赖分析，基于标识信息的依赖判定方法一定程度上弥补了单纯静态分析本身程序特征方法的不足，该方法实质是利用了原始程序中指导多线程并行的信息指导向量化。还有研究者采用动静态分析结合的方法解决依赖判定问题。例如 2017 年，Sampaio 等[32]提出了基于动静态结合的依赖分析及变换方法，对于不能完全由静态分析确定依赖关系的程序，建立多个可能的执行处理版本，在实际运行中确定具体执行的版本，该方法能够解决因静态分析信息不全导致的依赖分析失败的问题。

当编译器判定存在阻碍向量化的依赖时，传统的向量化方法不进行程序转换。然而，有的数据依赖关系可通过程序变换改变，从而使向量化成为可能。2017 年，赵捷等[33]根据 SIMD 特征及等价变换关系式，提出并论证了 7 个依赖分析及转换的定理，为利用等价关系式解除依赖奠定理论基础，并基于数组依赖图构建包含数组和语句依赖信息的有向图，在此基础上，利用语句重排和节点分裂等技术在数组依赖图中改变程序中阻碍向量化的依赖关系，增强了向量化能力。

2.1.2　控制依赖分析和变换

程序中分支判定和循环语句经常引入控制依赖，如图 5 所示，语句 C 是否执行依赖于语句 B 的判定条件。控制依赖给程序分析及优化带来复杂性和不确定性，增加了向量化难度。

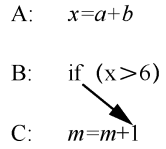


图 5　控制依赖说明

近几年对于面向向量化的控制依赖问题，学术界在研究对象方面，从原始的简单分支判定语句的向量化，逐渐转向多层嵌套分支判定语句的向量化；在向量并行化方面，从原始迭代间的分支判定语句的向量化，转向多角度综合考虑迭代内和迭代间的语句自动向量化；在理论方法方面，从先前的基于控制流转换数据流的全分支转换法，转向更加激进的分支判断转换方法，如部分线性化和投机执行等方法。

Smith 等[34]将控制依赖转换成数据依赖（If-conversion），然后利用传统向量化方法处理[35]。基于 If-conversion 的向量化方法示例如图 6 所示，图 6(a)是原始程序，图 6(b)是通过 If-conversion 并利用选择指令（select）实现对控制流程序的向量化。

基于 If-conversion 的向量化方法在没有代价模型指导的情况下，转换所有分支判定语句，生成较多额外的选择指令。Shin[36]利用谓词判定是否会真正执行，绕过一些不实际执行的语句，减少了在包含多层嵌套分支程序中产生的冗余指令。还有基于分支分类处理的方法，例如 2015 年，孙回回等[37]解决嵌套分支的向量化问题，根

据控制流分支控制条件与循环迭代之间的关系将控制流条件变量分为循环变量和循环不变量两类，外提循环不变量，将循环变量转换为数据流形式，并利用递归方法解决包含嵌套分支程序的向量化问题。利用分支逆转方法将向量化没有收益的分支退回为控制流，该方法能够有效减少包含嵌套分支程序向量化的指令代价。近几年，分支线性化选择方面的研究有了新的突破。Moll 等[24]在 PLDI 2018 会议中首次提出了用部分分支线性化技术及基于控制流及数据流分析技术，来减少条件分支转换的数量，与先前的所有分支都进行线性化不同，该研究根据程序分支的特点，并结合不同迭代的一致性分析，来选择向量化收益更大的分支进行线性化处理。该方法有效减少了分支依赖引入的冗余重组指令，显著提高了向量化对分支依赖程序的性能，该方法对 SPEC CPU 2017 和 NAB Benchmark 的测试程序平均性能提升了 146%。

```
for(i=0;i<256;i++)                for(i=0;i<256;i+=4)
{                                 {
    if(m[i]!=4)                       v4=(4,4,4,4);
      n[i]=m[i];                      vp=m[i:i+3]!=v4;
}                                     [n:i+3]=select(n[i:i+3],
                                      m[i:i+3],vp);
                                  }
```

(a) 原始程序　　(b) If-conversion转化后的程序

图 6 基于 If-conversion 的向量化方法示例

传统控制依赖分析主要采用静态分析方法，没有考虑动态运行时信息，忽略了一些优化机会。Sujon 等[38]利用投机执行的方法，解决包含控制依赖程序的向量化问题，根据运行时程序执行的路径判定是否向量化，该方法能够有效处理静态分析中不可知依赖信息的程序，适用于总是执行可向量化特定分支的程序。类似地，Baghsorkhi 等[39]采用投机执行法，对包含控制依赖且运行执行次数少的程序进行部分向量化。基于运行时统计的方法也应用于分支判定语句的向量中，例如，Sun 等[40]利用静态仿射分析和动态统计控制流运行比例的方法，探测一致性运行分支，插入运行时检查分支语句，该方法基于运行程序，根据统计的分支运行状态进行动态调优，能够有效减少因控制流向量化引入的冗余分支和掩码向量指令，该方法使 Rodinia 测试程序平均性能提升 1.14 倍。

此外，除了上述从控制依赖关系角度出发解决向量化问题，还出现了从分支判定语句内部挖掘向量化并行性的研究。2017 年，高伟等[41]分析分支语句输出的基本块内蕴含的向量并行性，首次提出考虑基本块间向量重用的直接控制流向量化方法，向量化包含同构语句条数较多的循环，然后利用代价模型指导向量指令生成。

2.2 别名分析和变换

别名分析是指判定 2 个变量是否访问同一地址空间的分析过程[42]。别名分析的局限，会限制编译器对依赖关系的判定和对优化机会的识别，进而限制向量化。对于如图 7 所示的示例程序，在考虑指针别名引起的依赖时，指针 *a* 和 *b* 作为形式参数传入函数体，在无法取得 *a* 和 *b* 的内存地址信息时，指针 *a* 可能存在循环携带的真依赖，该循环无法进行向量化。

```
void sum(int* a,int* b,int n)
{
  for (int i=0;i<n;i++)
      a[i]=b[i]+a[i];
}
```

图 7 别名分析示例程序

针对别名分析，近几年学术界逐渐从研究传统的静态的别名分析法转为动静态结合的分析方法。如侯永生[43]利用动静态结合的方法，解决别名和非线性表达式的动态检测条件构建问题，该方法插入动态监测代码，运行时判定是否进行向量化。类似地，2015 年，刘鹏等[44]通过动态插桩和试运行提取指针别名信息，并反馈到向量化阶段指导向量化代码生成。

除了动静结合的方法外，近几年还出现了挖掘程序别名相关过程间的特征信息的突破性方法。例如，2016 年，Sui 等[45]发现在自动向量化中应用的别名分析方法较保守，例如编译器没有将过程间分析方法应用于自动向量化的别名分析中。别名分析方法的局限阻碍了依赖关系的判定，丢失了一些自动向量化的机会。对于包含指针、数组访问和结构体访问循环的程序，文献[45]根据访存基地址，结合循环特征及访存的范围，进

行过程间别名分析，该方法能够有效处理包含数组和结构体等程序的向量化的别名分析问题，可有效提升 SPEC CPU 2000 基准测试的 177.mesa 程序 7.18%的性能收益。

3　分组分析和变换问题及方法

自动向量化分组策略直接影响能否向量化及其收益[46]。近几年对于分组问题，在研究对象方面，从原始单一形式的分组方法研究，逐渐转向多级融合的分组方法研究；在程序信息分析粒度方面，从先前单一基本块或循环特征信息分析，转向跨基本块以及多层嵌套循环的特征分析；在理论方法方面，从先前的启发式贪心方法转向更加高级的组合优化方法，如人工智能、动态规划和整数线性规划，等等。

3.1　基本块级分组问题及方法

基本块级分组旨在寻找基本块内语句的向量并行性，如图 8 所示，将同一个基本块中的 4 条语句，分组转换为一个向量语句。Larsen 等[47]于 2000 年首次提出了奠基性的基本块级向量化方法，即超字并行 （SLP, superword level parallelism）方法，基于连续内存访问，利用基本块内数据的复用信息，寻找多条可并行执行的同构语句，并将其向量化。

$$a[i]=b[i]+c[i];$$

$$a[i+1]=b[i+1]+c[i+1]; \Longrightarrow a[i:i+3]=b[i:i+3]+c[i:i+3];$$

$$a[i+2]=b[i+2]+c[i+2];$$

$$a[i+3]=b[i+3]+c[i+3];$$

图 8　分组转换示例

学术界对 SLP 进一步展开研究，提出了许多新的改进方法，其分组策略主要包括局部贪心和全局的分组策略。贪心策略和全局策略互有优劣。贪心策略编译速度快，但分析信息较局部，无法有效向量化不规则的内存访问。全局策略分析信息较全局，能够向量化不规则的内存访问，但编译时间较长。

局部贪心分组指基于“定义/使用”和“使用/定义”关系对向量进行分组。近几年出现了许多相关的扩展优化工作[48-51]。例如，2015 年，Porpodas 等[49]在建立“定义/使用”或“使用/定义”依赖图时，实时计算代价，找到最小生成指令代价，去除生成破坏代价的指令。2017 年，Porpodas 等[52]同时利用“定义/使用”和“使用/定义”关系信息扩展同构语句，首次在向量分组中利用了不同构建同构链间的重用信息，来减少向量化中重组指令的生成。2019 年，Mendis 等[53]将启发式 SLP 方法扩展应用于嵌套汇编形式的向量程序，分析向量程序的并行度结合处理器所支持的 SIMD 功能，并利用混洗指令进一步提升程序的向量并行度，其实质是将向量化的范畴从“多对一转换”扩展到了“多对多转换”。

全局分组指在程序中进行全局搜索可向量化的指令。例如，Barik 等[54]在基本块中进行全局搜索，利用动态规划方法综合考虑标量和向量指令代价对程序进行分组。Liu 等[55]扩展了初始对象选择范围，除了连续内存访问外，将具有相同类型的运算也作为初始向量对象，以减少重组指令为目标，基于依赖图全局搜索，并启发式选择相对最优的分组方法。Huh 等[56]在 MICRO 2017 会议上首次提出了将分级处理的思路应用到全局 SLP 分组中，先局部选择收益较高的向量化对象，再全局将它们重组构建向量化程度更高的向量对象集合。该方法适用于包含尺寸较大的基本块，平均提升 SPEC CPU 2006 和 NAS 测试程序的性能 8.6%。值得一提的是，Mendis 等[57]在 OOPSLA 2018 会议上提出将整数线性规划方法应用于全局 SLP 向量化分组中，并利用 IBM CPLEX 商用的整数线性规划调度器，能够找到在函数中相对最优分组的向量化策略，该方法使 SPEC CPU 2017 的浮点测试程序平均提升了 7.58%。此外，近几年出现了基于人工智能技术的分组方法，代表性工作如 Mendis 等[58]在 NeurIPS2019 提出的利用图网络学习方法模拟 SLP 方法的分组打包过程。

3.2　循环级分组问题及方法

循环级向量化分组旨在寻找循环中迭代间的向量化并行。Allen 等[59]首次提出基于循环的（Loop-based）自动向量化方法，如图 9 所示。Loop-based 自动向量化方法针对内层循环的迭代空间，将整个数组作为一个向量单元进行操作，基于依赖关系分析将在不同迭代间且相互间不会形成依赖环的多条语句转换为向量形式。

```
for(i = 0; i < n; i++)          for(i = 0; i < n; i+= 4)
{                               {
  a[i] = b[i] + c[i]              a[i:i+3] = b[i:i+3] + c[i:i+3]
}                               }
```

图 9 Loop-based 自动向量化方法示例

学术界在 Loop-based 方法的基础上进一步改进优化，集中于外层循环向量化及多层嵌套循环向量化。其中大多数研究集中于多层嵌套循环向量化。

对于外层循环向量化，Nuzman 等[60]利用循环展开实现外层循环向量化，该方法能够有效提升包含多层嵌套循环且外层循环包含较多并行性的程序性能。

对于多层嵌套循环向量化，魏帅等[61]发现对于多层嵌套循环而言，有时会有多种向量化分组方法，如何进行分组选择影响向量化的收益，综合考虑多层嵌套循环的内存访问和依赖等信息，确定相对最优的自动向量化方案，该方法可有效选择包含多层嵌套循环程序的分组向量化方案。研究者也将多面体技术[62]应用于多层嵌套循环的分组选择中。多面体技术是一种统一化的程序变换表示技术，它通过迭代域、仿射调度和访存函数表示代码，有利于表示程序变换的组合，有助于解决包含循环程序的向量化问题。Trifunovic 等[63]建立一种考虑对齐、连续和类型转换等因素的代价模型指导多面体变换，选择收益最大的向量化方案。Kong 等[64]基于多面体模型，结合向量化、并行化和局部性因素分步骤进行程序变换；基于依赖和局部访问信息，利用循环分块和循环融合等方法进行空间局部性优化，选择相对最优的向量化方案。

3.3 函数级分组问题及方法

函数级向量化从函数粒度[65]识别程序中的数据级并行，发展到过程间分析。函数包含的程序类型复杂多变，程序的复杂性和过程间的分析给函数级向量化分组分析和变换带来挑战。

起初，编译器通过基于模式匹配的内建函数（build-in function）[66]实现函数级向量化，如基础数学向量函数已在编译器中广泛使用。近几年，相关研究者深入研究函数级向量化分析和变换方法。2015 年，Karrenberg 等[67]在静态单赋值表示形式（SSA, static single assignment form）下，基于数据流分析和变换利用掩码和选择指令解决函数向量化中运行操作不一致的问题。2017 年，Reiche 等[68]对 Karrenberg 的方法进一步扩展，针对图像处理专用语言，结合领域语言的相关信息，实现了高级语言程序到高级语言程序的函数级向量化。

3.4 多级向量化分组扩展及融合

学术界近期关注如何综合利用上述不同级别的分组方法，主要包括多种转换方法的融合及多模式分组的选择。多种转换方法的融合指同时利用多种转换方法来提升向量化的适用范围。例如，If-conversion 与 SLP 结合实现跨基本块语句的向量化[69-70]；循环展开与 SLP 的结合实现循环的向量化[71-72]。

多模式分组的选择指对多种分组模式进行选择来提升向量化的收益。2017 年，高伟等[46]提到程序蕴含的并行性是固有的，根据程序的特征分析向量化的并行度，来选择具体的分组方法，分组方法包括 Loop-based 方法、SLP 方法和 Loop-aware 方法。该方法平均提升 SPEC CPU 2006、SPEC CPU 2000 和 NPB 中部分测试程序的 12.1%的性能。Zhou 等[73]根据语句重用及并行相关信息，同时综合考虑基本块级和循环级分组方法，该方法能够有效减少包含循环结构的程序在自动向量化中产生的重组指令负载。

4 处理器相关分析和变换问题及方法

目前绝大部分处理器都支持 SIMD 扩展部件，如 Intel、AMD 和 ARM 等，如表 1 所示。SIMD 扩展部件支持的向量寄存器长度越来越长[74]，功能也越来越丰富，Intel SIMD 扩展部件的指令集统计如表 2 所示。

SIMD 扩展部件的发展推动着向量化方法的不断改进。1) 向量寄存器长度的增长增加了向量化的并行度。2) SIMD 扩展部件功能越来越丰富，使编译器能够向量化更多的程序。例如，AVX512 指令集的冲突探测指令有助于编译器实现对包含间接数组访问程序的自动向量化。

然而，SIMD 相关处理器在运算并行、访存并行及并行度支持方面依然存在以下局限。

1) 在运算并行特性支持方面，大部分处理器只支持相同类型运算并行的向量指令，不支持不同类型运算并行的向量指令，或者尽管支持不同类型运算并行的向量指令，但是其指令的时延较大。

表 1　　支持带有 SIMD 扩展部件的处理器

厂商	处理器	指令集	长度/bit
Intel	Pentium	MMX	64
		SSE	128
	Core	AVX128	128
		AVX256	256
		IMCI	512
		AVX512	512
IBM	P6	VMX	128
	BG/L	BG/L	256
DEC	Alpha	MVI	64
SGI	MIPS V	MDMX	64
Sun	SPARC v9	VIS	64
HP	PA-RISC	MAX2-2	64
Motorola	G4	AltiVec	128
AMD	Athlon	3Dnow!	128
	Jaguar	F16C	128
	Bulldozer	FMA	256
Ingenic	XBurst	MXU	128
Sony	Cell	AltiVec	128
CAS	Godson	Godson	256
NRCPC	SW26010	SW26010	256
NUDT	Matrix	Matrix	1024
ARM	ARMv6	NEON	128
	PPC970	VMX	128
	ARMv8	SVE	2 048

2) 在访存并行特性支持方面，大部分处理器只支持连续向量内存访问指令，或者尽管支持非连续内存访问指令，但指令时延较大。大部分处理器只支持对齐访问指令，或者尽管支持非对齐访问指令，但指令时延较大。

3) 在并行度支持特性方面，SIMD 扩展部件支持的并行长度有限，且只支持向量处理长度为 2 的整数次幂。

由于上述局限，对于有些程序，编译器不能直接使用 SIMD 扩展指令进行向量化，进而限制了向量化的适用范围。为此，本节分别从以上三方面介绍面向处理器支持特性的向量化分析和变换问题及方法。

4.1　运算相关分析和变换问题及方法

如前文所述，编译器不能直接使用 SIMD 扩展部件支持的指令实现对运算类型不相同程序向量化。然而，现实中存在大量包含不相同类型运算的程序，且有些实际上运算类型相同的程序语句在编译过程中由于某些标量优化（如常量折叠和强度削弱[75]等）的实施，会被转换为运算类型不同的程序，这类情形广泛存在于 SPEC CPU 测试程序中。如果能够对这类程序实现自动向量化，可有效提升这些程序的性能。

运算类型不相同程序的向量化问题是 Porpodas 等[21]在 2015 年 CGO 会议中首次提出的，近年来有多篇论文对该问题从不同角度提出了解决方法，主要包括基于硬件特殊指令的方法和基于表达式等价变换的方法。

1) 基于硬件特殊指令的方法。其主要利用 SIMD 特殊指令实现运算类型不同语句的向量化，

表 2　　Intel SIMD 扩展部件的指令集统计

序号	时间	指令集	处理器	长度/bit	特征
1	2015 年	AVX512	Xeon Phi	512	Masked blend、Perm、Bitwise、Conflict detection、Gather、Scatter and Prefetch instruction
2	2013 年	AVX256	Haswell	256	Mul and add instruction、Gather instruction、Broadcast instruction Masked load and store instruction
3	2008 年	AVX128	Sandy Bridge	128	Data reorganization and Unaligned memory instruction
4	2006 年	SSE4	Penryn/Nehalem	128	Insert、Extract、Search and String processing instruction
5	2004 年	SSE3	Pentium4	128	Unaligned memory instruction，Horizontal add instruction
6	2001 年	SSE2	Pentium4	128	Type conversion instruction
7	1999 年	SSE	Pentium3	128	128 length of Basic logical operation，shift and compare operation
8	1996 年	MMX	Pentium	64	Basic logical operation，shift and compare operation

例如，PSLP 方法[21]将运算类型不同的语句中差异的部分相互参照并分别通过添加选择指令进行扩充，从而将运算类型不同的语句转换成运算类型相同的形式；VeGen 方法[22]实现了一个编译框架，利用 Non-SIMD 指令实现对运算类型不同语句的向量化。基于硬件特殊指令的方法一般会受到处理器平台的限制，且会引入额外的运行代价。

2) 基于表达式等价变换的方法。其主要利用表达式等价变换将满足特定条件的运算类型不同的语句转换为运算类型相同的语句，从而为实施 SLP 创造条件。例如，LSLP[50]（look-ahead SLP）方法针对运算顺序存在差异的多条语句分析和处理，当条件合适时基于交换律对可交换运算操作和操作数进行重排，从而获得运算顺序相同的语句。SN-SLP[76]（super-node SL）方法与 LSLP 方法类似，当条件合适时基于减法或者除法的等价关系式（如 $a-b-c=a-c-b$）对不同语句中的运算进行重排。基于表达式等价变换的方法一般对处理器硬件没有特殊要求，不会引入额外的运行代价。

4.2 访存相关分析和变换问题及方法

访存并行特性主要包括连续和对齐 2 个方面，下面分别介绍相关问题及方法。

4.2.1 连续访存相关分析和变换问题及方法

非连续内存访问在程序中大量存在，例如间接数组[77]和结构体等。对于非连续内存访问的向量化，学术界有了新的突破。在分析方法方面，从传统的静态分析方法，扩展到基于动态投机执行的方法、基于特殊硬件指令的方法以及针对特定类型程序的访存排布方法；在研究对象方面，从传统的固定步长的非连续内存访问的向量化研究，扩展到不固定步长[78]的非规则连续内存访问的向量化研究。

在访存信息的分析方面，在编译器中对于连续内存访问的判定主要基于连续内存访问定义的静态分析法。单纯通过静态分析法有时难以判定访存是否连续，有研究者利用动静分析结合的方法解决该问题，例如，姚远[79]记录指针引用信息，进而确定指针引用对齐和连续信息，根据循环迭代信息推出运行时指针地址信息判断对齐和连续性信息指导自动向量化，该方法利用动态运行时信息指导对齐和连续访存指令生成，减少了非对齐和非连续内存访问。

在分析对象方面，近期研究集中于规则步长的内存访问、结构体、间接数组和非规则内存访问方面的向量化研究。

对于规则步长的内存访问，Nuzman 等[80]发现循环中包含数组访问步长为 2 的整数次幂内存访问程序大量存在，为了实现此类程序的向量化，基于线性数组访问的基地址和步长信息，给出了循环中访问步长为常量操作的识别方法，抽象定义了交错操作和提取操作，如图 10 所示，利用该操作实现循环中包含数组访问步长为 2 的整数次幂内存访问程序的向量化。此后，Anderson 等[81]利用混洗指令实现基于循环的数组访问步长为任意长度的向量化，该研究也利用了数据流分析，改变不同迭代的执行顺序以减少不必要的混洗指令生成，能有效减少重组指令的数量。

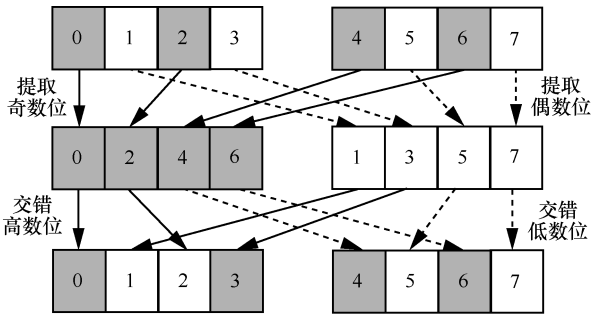


图 10 交错和提取操作

结构体在程序中很常见，由于其数据的非连续和非对齐访问特性[82]，导致对其向量化[83]较困难。近几年，研究者关注对结构体的向量化。2015 年，Li 等[84]提出一种转换结构体中数据重组的方法，利用矩阵转置运算，使结构体中相同类型的访问变量数据排布连续，进而减少了非连续内存访问。2016 年，于海宁等[85]提出利用结构体中访问的数据相似性信息指导结构体拆分，将引用的结构体数组的非数组域映射到二维数组来满足结构体数组向量化时的访存连续和对齐要求，以降低缓存失效的次数。

有些包含间接数组访问的程序[86]由于在编译阶段不知是否访问连续、对齐或依赖关系信息，导致很难向量化[87]。2018 年，姚金阳等[88]利用局部数据重组方法解决间接数组索引向量化问题，首先将间接数组赋值到临时数组中，然后将临时数组中的数据加载到 SIMD 向量寄存器，进而实现向量化，运算结束后根据需求判断是否将其存入间接数组。

非规则内存访问形式的代码在人工智能和大

数据等应用领域广泛存在，近几年对该类型程序的向量化研究越来越引起学术界的关注。非规则内存访问的随机性和不确定性，给向量化的依赖、连续和对齐相关分析及变换增加了难度[2]。近几年，关于非规则内存访问的向量化研究出现了新的研究方法，包括排布内存访问的方法和基于硬件特殊指令的方法。其中访存排布代表性的工作有：Chen 等[89]在 CGO2016 会议上将不规则的内存访问抽象为稀疏矩阵的计算，通过排布内存访问的结构，增强内存访问的时间和空间局部性，进而有效提升了非规则内存访问的性能收益，对于一些包含不规则的图计算的应用平均提升了 2.81 倍的性能。基于特殊硬件指令的代表性方法有：Jiang 等[90]在 CGO2018 会议中利用 AVX512 中的冲突探测指令，并结合运算的结合律进行对 reduction 型非规则内存访问的向量化，在部分图计算的应用中可提升 1.4~11.8 倍的性能。

4.2.2　对齐访存相关分析和变换问题及方法

目前针对非对齐内存访问的研究问题主要包括静态分析访存的对齐信息方法[26,91]、基于 OpenMP 的编译指示方法[92]、基于硬件特殊指令的方法[93]、基于动态运行时分析的方法[94]以及访存排布方法[28]，等等。由于近几年并未发现有新的相关研究工作，因此本文不做详述。传统的非对齐内存访问的向量化研究可详见文献[2]。

4.3　并行度相关分析和变换问题及方法

并行度选择直接影响向量化能否实施及其收益。并行度越大，通常向量化的收益越高，可向量化的条件越严格。选择适合的并行度并不是显而易见的，与程序本身蕴含的并行特性以及处理器支持特性都有关系。

近几年，并行度的选择研究在分析粒度方面越来越精细，从传统循环级的分析[59]，到后来的语句级的分析[56]，再到近几年新出现指令级的分析[27]。其中代表性的工作是 VW-SLP（PACT2018），根据程序蕴含的并行特性；在指令级别调整向量的并行度，当可并行语句较多时候，增大向量并行度，充分利用 SIMD 的并行特性；当可并行语句较少时，减少并行度，避免向量分组过早停止。除了传统的并行度分析和转换方法外，近几年也出现了面向并行度调节的机器学习方法[95-96]。例如，2019 年，Haj-Ali 等[23]将端到端的深度学习方法应用到循环向量化并行度选择中，与原有的方法相比，性能提升了 1.29~4.73 倍的性能提升。

并行度的选择除了与程序特征有关系，也受限于处理器的特性。如前文所述，目前大部分处理器 SIMD 扩展部件只支持向量处理的长度为 2 的整数次幂。因此编译器无法直接利用 SIMD 指令实现非 2 的整数次幂并行度的向量化。然而，这类程序在现实中普遍存在，如基于红绿蓝颜色标准表示的多媒体应用中经常出现并行语句数量为 3 的程序。

非 2 的整数次幂并行度向量化是近几年出现的新问题，目前的解决方法主要是基于指令生成优化的方法。例如，2016 年， Zhou 等[97]在把向量寄存器数据存储到内存时，先把不需要处理的数据拿出来，等待计算完成后把数据存放回去，该方法能够处理并行度为非 2 整数次幂程序的向量化，并减少了重组指令的数量。类似地，2017 年，高伟等[46]部分利用 SIMD 扩展指令，实现包含循环次数或者基本块内可并行语句较少程序的向量化，区分向量寄存器中的有效部分和无效部分，把有效部分的数据经过计算传进内存，无效部分的数据不传送到内存，该方法能够处理并行度为非 2 的整数次幂程序的向量化。

5　性能评估分析问题及方法

编译器利用代价评估模型辅助判定是否实施向量化。代价评估模型需要精确且评估时间复杂度不能太高。然而，这两方面往往是矛盾的。代价评估模型越精确，需考虑处理器的因素越多，导致代价评估模型算法复杂度越大。如何找到有效的代价评估算法是个挑战。

近几年，关于向量化性能评估分析问题，学术界出现了较多新的方法，从传统的基于指令计数的评估方法，转向基于人工智能的方法。在评估模式使用方面，从传统的应用于判定是否向量化，转向利用评估模型指导面向向量化相关的程序变换。

在评估方法方面，出现了较多基于机器学习的方法。例如，Stock 等[98]等针对向量化代价模型不精确的问题，利用机器学习方法提取汇编代码特征，结合循环重排、循环展开以及向量化的方案选择，辅助向量指令生成。2020 年，Pohl 等[99]分析程序的中间表示特征以及内存访问特征，采用基于线性回归技术的机器学习方法训练评估模型。

在评估模型的应用方面，Trifunovic 等[63]设计性能代价评估模型，指导多层嵌套循环的相对最优向量化方案选择，该模型结合了向量化因子、访存

步长宽度和对齐访存等因素。类似地，张媛媛等[100]结合内存访问的非连续和非对齐等因素，利用基于多面体指导循环变换的向量化代价模型，指导向量化分组方案。

自动向量化代价评估模型可从功能设计上进一步扩展。从性能评估模型的功能角度讲，传统编译器的代价评估模型只指导最终是否进行向量化，其实代价评估模型还可以指导与向量化相关的程序变换方法。从代价评估模型自身考虑的因素角度讲，传统代价评估模型只考虑指令时延因素，其实处理器运行程序的性能与多方面因素有关系，如指令时延、访存对齐等，如图 11 所示。如何考量多种影响性能的因素[12]，如何设计性能衡量权重，以及如何结合程序自身的逻辑特征设计快速精确的向量化评估模型都是值得进一步研究的。

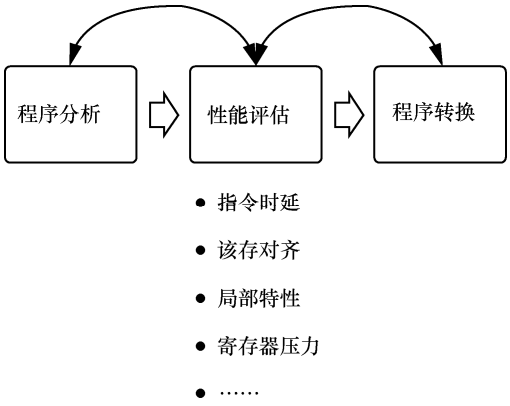


图 11 改进自动向量化评估模型

6 自动向量化的展望

自动向量化领域还存在众多的挑战需要突破，基于近年来相关关键问题突破及暴露出来的不足之处，本节梳理出一些值得进一步关注和探索的研究问题。

6.1 保义分析和变换相关问题

1) 一些程序的依赖关系比较复杂，只有在动态运行时才能够确定。目前的研究工作主要是利用基于动静态结合的多版本技术处理方法，但是这类方法额外增加了运行时判定代码，当实际运行时存在较多依赖时，反而可能会降低程序性能。如何进一步减少动态运行时不必要的判定语句是值得进一步研究的问题。

2) 当编译器确定程序依赖关系时，可通过一些程序变换方法应用于向量化中，改变依赖关系，进而提高向量化的适用范围。

6.2 分组分析和变换相关问题

程序尺寸越来越大，语句形式越来越复杂。有时一些语句同时存在多种向量化的分组方案。向量化分组直接决定能否实现向量化及是否有收益。

1) 运算类型不同语句的向量化问题是近几年出现的新问题，由于运算类型不同语句比运算类型相同语句在程序中的占比更高，若能对运算类型不同语句进行向量化，可有效提升程序的性能，因而该问题近几年得到广泛的关注。如何利用更多类型的等价表达式变换来对运算类型不同语句进行转换以及多种变换如何进行选择是未来值得研究的重要方向。对于上述研究问题，应用端到端的深度学习等[101]方法值得关注。

此外，编译指令融合优化和强度消减优化也会影响运算排布，目前少有研究将这些优化与向量化进行有机综合考虑，该问题是值得结合应用场景进一步深入研究的。

2) 并行度选择直接影响向量化的性能收益，目前的研究深入到指令级别的自适应并行度选择，但仅采用基于遍历搜索启发式策略进行选择。可引入组合优化方法进行并行度的选择，例如，引入动态规划、整数线性规划等方法是值得深入探索的可能解决思路。

3) 函数级向量化研究近几年越来越得到研究者的关注，可取得任务级并行的效果[2]。然而，其受限于向量化分组方法、编译过程间分析和别名分析。当函数运行路径不一致时，很有可能其操作的数据类型长度也不一致，为了向量化这些操作，需要额外添加重组指令。如果能减少因为函数执行路径不一致而引入的重组指令，则有利于提高向量化的收益，该问题有可能是引导打破函数执行路径差异限制的向量化方法的研究方向。

4) 自动向量化可描述为逻辑约束求解问题，因而可尝试利用 SMT 等理论方法解决向量化分组问题，同时结合向量化相关的限制因素，以加快求解速度，这方面的研究较少，值得尝试。

6.3 处理器支持特性相关问题

自动向量化实质就是利用 SIMD 扩展部件提供的运算并行和访存并行特性提升程序性能。随着处理器对 SIMD 扩展部件支持能力的提高，出现了一些新的向量化方法。

1) 处理器不仅支持基于 SIMD 扩展的数据级并行，还支持基于多发射的指令级并行和基于多核

多线程的任务级并行。目前向量化研究集中于发掘程序的数据级并行，可综合考虑指令级并行和任务级并行，这样更有助于充分利用处理器的多层次并行能力。

2) 非规则内存访问广泛存在于科学计算领域和信号处理领域[78]。然而，对该类型程序的自动向量化研究并不充分。非规则内存访问的随机性和不确定性，给自动向量化的依赖、连续和对齐相关分析及变换增加了难度。如何克服上述困难，对该类型程序的自动向量化是值得研究的。

3) 大部分处理器 SIMD 扩展部件支持的并行长度有限，如能提高向量化的并行度，有助于提升程序性能。对于一些特定应用领域，并不需要高精度的计算，只需“够用”就好，因此可适当减少数据类型长度，来提高程序的并行度。

4) 如何充分利用功能日益丰富的 SIMD 指令，给编译器带来新的挑战。如 Intel 推出的 AVX512 包含内存冲突检测、扩展压缩和掩膜操作指令等，利用这些指令有利于向量化更多类型程序。尽管目前很多编译器已经支持这些新的指令，但是目前采用的方法大多是模板匹配法，并未深入分析程序特征、指令特性并与向量化有机结合起来，不具备通用性和可扩展性。如何改进向量化方法并利用这些指令提升向量化的能力是值得进一步研究的。

另外，指令集发展给向量化提供了更多选择，如实现向量化非连续访存，既可以通过连续内存访问结合重组指令的方式优化，也可直接通过非连续的内存访问指令优化。不同优化方案带来的收益与具体的硬件平台和程序特征都相关，如何选择更优的指令生成方案是值得结合应用场景深入研究的。

6.4 性能评估分析相关问题

处理器技术快速发展有效提升了处理器的性能，如超标量和乱序执行等。传统编译器通过基于指令计数的评估方法早已不能准确评估处理器的性能。不精确的评估方法会严重影响编译器程序变换的具体实施方法和收益。向量化性能与处理器的指令时延和执行单元等多方面因素都有关系，要精确评估比较困难，通过引入机器学习等新方法可能解决该问题。

7 结束语

自动向量化能够利用 SIMD 并行特性有效提升程序性能，受到越来越多的关注，众多向量化技术先后被提出。本文针对自动向量化相关的文献进行了系统整理，尤其对近些年的一些研究成果和技术突破进行了分析和总结。本文基于自动向量化问题的抽象描述识别出自动向量化领域 4 个方面的关键问题，即保义分析和变换、分组分析和变换、处理器相关分析和变换以及性能评估分析，从以上 4 个方面对研究成果和技术突破进行了全面分析和梳理，展望了该领域未来可能的研究方向，为该领域研究人员提供有益的参考。

参考文献：

[1] 杨毅宇, 周威, 赵尚儒, 等. 物联网安全研究综述：威胁、检测与防御[J]. 通信学报, 2021, 42(8): 188-205.
YANG Y Y, ZHOU W, ZHAO S R, et al. Survey of IoT security research: threats, detection and defense[J]. Journal on Communications, 2021, 42(8): 188-205.

[2] 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.

[3] ZHENG R H, PAI S. Efficient execution of graph algorithms on CPU with SIMD extensions[C]//Proceedings of 2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). Piscataway: IEEE Press, 2021: 262-276.

[4] BÖHM C, PLANT C. Massively parallel graph drawing and representation learning[C]//Proceedings of 2020 IEEE International Conference on Big Data (Big Data). Piscataway: IEEE Press, 2020: 609-616.

[5] YAMAZAKI S. Future possibilities and effectiveness of JIT from elixir code of image processing and machine learning into native code with SIMD instructions[R]. 2021.

[6] BIAN H D, HUANG J Q, LIU L B, et al. ALBUS: a method for efficiently processing SpMV using SIMD and load balancing[J]. Future Generation Computer Systems, 2021, 116: 371-392.

[7] PAPAPHILIPPOU P, PAUL H J K, LUK W. Simodense: a RISC-V soft-core optimised for exploring custom SIMD instructions[C]//Proceedings of 2021 31st International Conference on Field-Programmable Logic and Applications (FPL). Piscataway: IEEE Press, 2021: 391-397.

[8] GAO Y, LIU Y Z, MA Y M, et al. abPOA: an SIMD-based C library for fast partial order alignment using adaptive band[J]. Bioinformatics, 2020, 37(15): 2209-2211.

[9] BARREDO A, CEBRIAN J M, MORETÓ M, et al. Improving predication efficiency through compaction/restoration of SIMD instructions[C]//Proceedings of 2020 IEEE International Symposium on High Performance Computer Architecture. Piscataway: IEEE Press, 2020: 717-728.

[10] LI H J, HAN J, HAN D S. Leveraging SIMD parallelism for accelerating network applications[C]//Proceedings of APNet'20 4th Asia-Pacific Workshop on Networking. New York: ACM Press, 2020: 23-29.

[11] MALEKI S, GAO Y Q, GARZAR´N M J, et al. An evaluation of vectorizing compilers[C]//Proceedings of 2011 International Confe-

rence on Parallel Architectures and Compilation Techniques. Piscataway: IEEE Press, 2011: 372-382.

[12] SISO S, ARMOUR W, THIYAGALINGAM J. Evaluating auto-vectorizing compilers through objective withdrawal of useful information[J]. ACM Transactions on Architecture and Code Optimization, 2020, 16(4): 1-23.

[13] INOUE H. How SIMD width affects energy efficiency: a case study on sorting[C]//Proceedings of 2016 IEEE Symposium in Low-Power and High-Speed Chips. Piscataway: IEEE Press, 2016: 1-3.

[14] AMIRI H, SHAHBAHRAMI A. SIMD programming using Intel vector extensions[J]. Journal of Parallel and Distributed Computing, 2020, 135: 83-100.

[15] STOJANOV A, TOSKOV I, ROMPF T, et al. SIMD intrinsics on managed language runtimes[C]//Proceedings of the 2018 International Symposium on Code Generation and Optimization. 2018: 2-15.

[16] BOGAEVSKIY D, MINENKO M, EZHOV S, et al. Development and implementation of the H.264-codec deblocking filter based on the MIPS SIMD architecture[C]//Proceedings of 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus). Piscataway: IEEE Press, 2021: 246-251.

[17] STEPHENS N, BILES S, BOETTCHER M, et al. The ARM scalable vector extension[J]. IEEE Micro, 2017, 37(2): 26-39.

[18] KUMAR R, MARTINEZ A, GONZALEZ A. A variable vector length SIMD architecture for HW/SW co-designed processors[J]. arXiv Preprint, arXiv: 2102.13410, 2021.

[19] MITRA G, JOHNSTON B, RENDELL A P, et al. Use of SIMD vector operations to accelerate application code performance on low-powered ARM and Intel platforms[C]//Proceedings of 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. Piscataway: IEEE Press, 2013: 1107-1116.

[20] 张为华, 藏斌宇. SIMD 编译优化技术研究概述[J]. 中国计算机学会通讯, 2007, 3(2): 27-36.
ZHANG W H, ZANG B Y. A survey on SIMD vectorization technology[J]. Communications of CCF, 2007, 3(2): 27-36.

[21] PORPODAS V, MAGNI A, JONES T M. PSLP: padded SLP automatic vectorization[C]//Proceedings of 2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). Piscataway: IEEE Press, 2015: 190-201.

[22] CHEN Y S, MENDIS C, CARBIN M, et al. VeGen: a vectorizer generator for SIMD and beyond[C]//Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. New York: ACM Press, 2021: 902-914.

[23] HAJ-ALI A, AHMED N K, WILLKE T, et al. NeuroVectorizer: end-to-end vectorization with deep reinforcement learning[C]//Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization. New York: ACM Press, 2020: 242-255.

[24] MOLL S, HACK S. Partial control-flow linearization[C]//Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 2018: 543-556.

[25] PLOTKIN G D. Call-by-name, call-by-value and the λ-calculus[J]. Theoretical Computer Science, 1975, 1(2): 125-159.

[26] EICHENBERGER A E, WU P, O'BRIEN K. Vectorization for SIMD architectures with alignment constraints[J]. ACM SIGPLAN Notices, 2004, 39(6): 82-93.

[27] PORPODAS V, ROCHA R C O, GÓES L F W. VW-SLP: auto-vectorization with adaptive vector width[C]//Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques. New York: ACM Press, 2018: 1-15.

[28] ALLEN R, KENNEDY K. Optimizing compilers for modern architectures: a dependence-based approach[M]. San Francisco: Morgan Kaufmann Publishers Inc, 2001.

[29] PSARRIS K, KLAPPHOLZ D, KONG X Y. On the accuracy of the Banerjee test[J]. Journal of Parallel and Distributed Computing, 1991, 12(2): 152-157.

[30] BULIĆ P, GUSTIN V. D-test: an extension to Banerjee test for a fast dependence analysis in a multimedia vectorizing compiler[C]//Proceedings of the 18th International Parallel and Distributed Processing Symposium, 2004. Piscataway: IEEE Press, 2004: 230.

[31] JENSEN N B, KARLSSON S. Improving loop dependence analysis[J]. ACM Transactions on Architecture and Code Optimization, 2017, 14(3): 1-24.

[32] SAMPAIO D N, POUCHET L N, RASTELLO F. Simplification and runtime resolution of data dependence constraints for loop transformations[C]//Proceedings of the International Conference on Supercomputing. New York: ACM Press, 2017: 1-11.

[33] 赵捷, 赵荣彩. 基于有向图可达性的 SLP 向量化识别方法[J]. 中国科学: 信息科学, 2017, 47(3): 310-325.
ZHAO J, ZHAO R C. Identifying superword level parallelism with directed graph reachability[J]. Scientia Sinica (Informationis), 2017, 47(3): 310-325.

[34] SMITH J E, FAANES G, SUGUMAR R. Vector instruction set support for conditional operations[J]. ACM SIGARCH Computer Architecture News, 2000, 28(2): 260-269.

[35] HALL M, SHIN J. Compiler optimizations for architectures supporting superword-level parallelism[M]. Los Angeles: University of Southern California, 2005.

[36] SHIN J. Introducing control flow into vectorized code[C]//Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques (PACT 2007). Piscataway: IEEE Press, 2007: 280-291.

[37] 孙回回, 赵荣彩, 高伟, 等. 基于条件分类的控制流向量化[J]. 计算机科学, 2015, 42(11): 240-247.
SUN H H, ZHAO R C, GAO W, et al. Control flow vectorization based on conditions classification[J]. Computer Science, 2015, 42(11): 240-247.

[38] SUJON M H, WHALEY R C, YI Q. Vectorization past dependent branches through speculation[C]//Proceedings of the 22nd International Conference on Parallel Architectures And Compilation Techniques. Piscataway: IEEE Press, 2013: 353-362.

[39] BAGHSORKHI S S, VASUDEVAN N, WU Y F. FlexVec: auto-vectorization for irregular loops[J]. ACM SIGPLAN Notices, 2016, 51(6): 697-710.

[40] SUN H H, FEY F, ZHAO J, et al. WCCV: improving the vectorization of IF-statements with warp-coherent conditions[C]//Proceedings of the ACM International Conference on Supercomputing. New York: ACM Press, 2019:

319-329.

[41] 高伟, 李颖颖, 孙回回, 等. 一种改进的控制流 SIMD 向量化方法[J]. 软件学报, 2017, 28(8): 2046-2063.
GAO W, LI Y Y, SUN H H, et al. Improved SIMD vectorization method in the presence of control flow[J]. Journal of Software, 2017, 28(8): 2046-2063.

[42] MAALEJ M, PAISANTE V, RAMOS P, et al. Pointer disambiguation via strict inequalities[C]//Proceedings of 2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). Piscataway: IEEE Press, 2017: 134-147.

[43] 侯永生. 多重循环 SIMD 向量化方法及性能优化技术研究[D]. 郑州: 解放军信息工程大学, 2014.
HOU Y S. Research on SIMD vectorization of loop nests and its optimization techniques[D]. Zhengzhou: PLA Information Engineering University, 2014.

[44] 刘鹏, 赵荣彩, 李朋远. 一种面向向量化的动态指针别名分析框架[J]. 计算机科学, 2015, 42(3): 26-30.
LIU P, ZHAO R C, LI P Y. Dynamic pointer alias analysis framework for vectorization[J]. Computer Science, 2015, 42(3): 26-30.

[45] SUI Y L, FAN X K, ZHOU H, et al. Loop-oriented array and field-sensitive pointer analysis for automatic SIMD vectorization[J]. ACM SIGPLAN Notices, 2016, 51(5): 41-51.

[46] 高伟, 韩林, 赵荣彩, 等. 向量并行度指导的循环 SIMD 向量化方法[J]. 软件学报, 2017, 28(4): 925-939.
GAO W, HAN L, ZHAO R C, et al. Loop vectorization method guided by SIMD parallelism[J]. Journal of Software, 2017, 28(4): 925-939.

[47] LARSEN S, AMARASINGHE S. Exploiting superword level parallelism with multimedia instruction sets[C]//Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation. New York: ACM Press, 2000: 145-156.

[48] 徐金龙, 赵荣彩, 韩林. 分段约束的超字并行向量发掘路径优化算法[J]. 计算机应用, 2015, 35(4): 950-955.
XU J L, ZHAO R C, HAN L. Vector exploring path optimization algorithm of superword level parallelism with subsection constraints[J]. Journal of Computer Applications, 2015, 35(4): 950-955.

[49] PORPODAS V, JONES T M. Throttling automatic vectorization: when less is more[C]//Proceedings of 2015 International Conference on Parallel Architecture and Compilation (PACT). Piscataway: IEEE Press, 2015: 432-444.

[50] PORPODAS V, ROCHA R C O, GÓES L F W. Look-ahead SLP: auto-vectorization in the presence of commutative operations[C]//Proceedings of the 2018 International Symposium on Code Generation and Optimization. New York: ACM Press, 2018: 163-174.

[51] PORPODAS V, RATNALIKAR P. PostSLP: cross-region vectorization of fully or partially vectorized code[C]//Languages and Compilers for Parallel Computing. Berlin: Springer, 2021: 15-31.

[52] PORPODAS V. SuperGraph-SLP auto-vectorization[C]//Proceedings of 2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). Piscataway: IEEE Press, 2017: 330-342.

[53] MENDIS C, JAIN A, JAIN P, et al. Revec: program rejuvenation through revectorization[C]//Proceedings of the 28th International Conference on Compiler Construction. 2019: 29-41.

[54] BARIK R, ZHAO J S, SARKAR V. Efficient selection of vector instructions using dynamic programming[C]//Proceedings of 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE Press, 2010: 201-212.

[55] LIU J, ZHANG Y R, JANG O, et al. A compiler framework for extracting superword level parallelism[J]. ACM SIGPLAN Notices, 2012, 47(6): 347-358.

[56] HUH J, TUCK J. Improving the effectiveness of searching for isomorphic chains in superword level parallelism[C]//Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE Press, 2017: 718-729.

[57] MENDIS C, AMARASINGHE S. goSLP: globally optimized superword level parallelism framework[C]//Proceedings of the ACM on Programming Languages. New York: ACM Press, 2018: 1-28.

[58] MENDIS C, YANG C, PU Y, et al. Compiler auto-vectorization with imitation learning[C]//Proceedings of the 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Cambridge: MIT Press, 2019: 32.

[59] ALLEN J R, KENNEDY K, PORTERFIELD C, et al. Conversion of control dependence to data dependence[C]//Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages. New York: ACM Press, 1983: 177-189.

[60] NUZMAN D, ZAKS A. Outer-loop vectorization: revisited for short SIMD architectures[C]//Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques. New York: ACM Press, 2008: 2-11.

[61] 魏帅, 赵荣彩, 姚远. 面向 SLP 的多重循环向量化[J]. 软件学报, 2012, 23(7): 1717-1728.
WEI S, ZHAO R C, YAO Y. Loop-nest auto-vectorization based on SLP[J]. Journal of Software, 2012, 23(7): 1717-1728.

[62] ZHAO J, LI B J, NIE W, et al. AKG: automatic kernel generation for neural processing units using polyhedral transformations[C]//Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation. New York: ACM Press, 2021: 1233-1248.

[63] TRIFUNOVIC K, NUZMAN D, COHEN A, et al. Polyhedral-model guided loop-nest auto-vectorization[C]//Proceedings of 2009 18th International Conference on Parallel Architectures and Compilation Techniques. Piscataway: IEEE Press, 2009: 327-337.

[64] KONG M, VERAS R, STOCK K, et al. When polyhedral transformations meet SIMD code generation[J]. ACM SIGPLAN Notices, 2013, 48(6): 127-138.

[65] MOREIRA R E A, COLLANGE C, QUINTÃO F M. Function call re-vectorization[J]. ACM SIGPLAN Notices, 2017, 52(8): 313-326.

[66] GNU. Using vector instructions through build-in functions[R]. 2018.

[67] KARRENBERG R. Automatic SIMD vectorization of SSA-based control flow graphs[M]. Wiesbaden: Springer Vieweg, 2015.

[68] REICHE O, KOBYLKO C, HANNIG F, et al. Auto-vectorization for image processing DSLs[C]//Proceedings of the 18th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Em-

bedded Systems. New York: ACM Press, 2017: 21-30.

[69] SHIN J, HALL M, CHAME J. Superword-level parallelism in the presence of control flow[C]//Proceedings of International Symposium on Code Generation and Optimization. Piscataway: IEEE Press, 2005: 165-175.

[70] TANAKA H, OTA Y, MATSUMOTO N, et al. A new compilation technique for SIMD code generation across basic block boundaries[C]//Proceedings of 2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC). Piscataway: IEEE Press, 2010: 101-106.

[71] LARSEN S, RABBAH R, AMARASINGHE S. Exploiting vector parallelism in software pipelined loops[C]//Proceedings of 38th Annual IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE Press, 2005: 11-129.

[72] ROCHA R C O, PORPODAS V, PETOUMENOS P, et al. Vectorization-aware loop unrolling with seed forwarding[C]//Proceedings of the 29th International Conference on Compiler Construction. New York: ACM Press, 2020: 1-13.

[73] ZHOU H, XUE J L. Exploiting mixed SIMD parallelism by reducing data reorganization overhead[C]//Proceedings of 2016 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). Piscataway: IEEE Press, 2016: 59-69.

[74] YAZDANPANAH F. An approach for analyzing auto-vectorization potential of emerging workloads[J]. Microprocessors and Microsystems, 2017, 49: 139-149.

[75] RODRIGUEZ-CANCIO M, COMBEMALE B, BAUDRY B. Automatic microbenchmark generation to prevent dead code elimination and constant folding[C]//Proceedings of 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE). Piscataway: IEEE Press, 2016: 132-143.

[76] PORPODAS V, ROCHA R C O, BREVNOV 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 (CGO). Piscataway: IEEE Press, 2019: 206-216.

[77] SUN H H, ZHAO R C, GAO W, et al. Exploiting pure superword level parallelism for array indirections[C]//Proceedings of 2015 Seventh International Symposium on Parallel Architectures, Algorithms and Programming (PAAP). Piscataway: IEEE Press, 2015: 13-19.

[78] ASHFAQ M, HUANG R B, OMARI M. FSCS-SIMD: an efficient implementation of fixed-size-candidate-set adaptive random testing using SIMD instructions[C]//Proceedings of 2020 IEEE 31st International Symposium on Software Reliability Engineering. Piscataway: IEEE Press, 2020: 277-288.

[79] 姚远. SIMD 自动向量识别及代码调优技术研究[D]. 郑州: 解放军信息工程大学, 2012.
YAO Y. Research on automatic SIMD vectorization recognization and code tuning technology[D]. Zhengzhou: PLA Information Engineering University, 2012.

[80] NUZMAN D, ROSEN I, ZAKS A. Auto-vectorization of interleaved data for SIMD[J]. ACM SIGPLAN Notices, 2006, 41(6): 132-143.

[81] ANDERSON A, MALIK A, GREGG D. Automatic vectorization of interleaved data revisited[J]. ACM Transactions on Architecture and Code Optimization, 2016, 12(4): 50.

[82] 李玉祥, 施慧, 陈莉. 面向向量化的局部数据重组[J]. 小型微型计算机系统, 2009, 30(8): 1528-1534.
LI Y X, SHI H, CHEN L. Vectorization-oriented local data regrouping[J]. Journal of Chinese Computer Systems, 2009, 30(8): 1528-1534.

[83] LI P Y, ZHANG Q H, ZHAO R C, et al. Data layout transformation for structure vectorization on SIMD architectures[C]//Proceedings of 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). Piscataway: IEEE Press, 2015: 1-7.

[84] LI P Y, ZHANG Q H, ZHAO R C, et al. Data layout transformation for structure vectorization on SIMD architectures[C]//Proceedings of 2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). Piscataway: IEEE Press, 2015: 1-7.

[85] 于海宁, 韩林, 李鹏远. 面向自动向量化的结构体优化[J]. 计算机科学, 2016, 43(2): 210-215.
YU H N, HAN L, LI P Y. Structure optimization for automatic vectorization[J]. Computer Science, 2016, 43(2): 210-215.

[86] WANG Q, HAN L, YAO J Y, et al. Research on vectorization technology for irregular data access[C]//Communications in Computer and Information Science. Singapore: Springer Singapore, 2017: 321-334.

[87] KIM S, HAN H. Efficient SIMD code generation for irregular kernels[J]. ACM SIGPLAN Notices, 2012, 47(8): 55-64.

[88] 姚金阳, 赵荣彩, 王琦, 等. 面向间接数组索引的向量化方法[J]. 计算机科学, 2018, 45(9): 220-223, 236.
YAO J Y, ZHAO R C, WANG Q, et al. Vectorization methods for indirect array index[J]. Computer Science, 2018, 45(9): 220-223, 236.

[89] CHEN L C, JIANG P, AGRAWAL G. Exploiting recent SIMD architectural advances for irregular applications[C]//Proceedings of the 2016 International Symposium on Code Generation and Optimization. New York: ACM Press, 2016: 47-58.

[90] JIANG P, AGRAWAL G. Conflict-free vectorization of associative irregular applications with recent SIMD architectural advances[C]//Proceedings of the 2018 International Symposium on Code Generation and Optimization. 2018: 175-187.

[91] PRYANISHNIKOV I, KRALL A. Pointer alignment analysis for processors with SIMD instructions[J]. Software-Practice and Experience, 2007, 37: 93-113.

[92] CIORBA F M, IWAINSKY C, BUDER P. OpenMP loop scheduling revisited: making a case for more schedules[C]//Evolving OpenMP for Evolving Architectures. Berlin: Springer, 2018: 21-36.

[93] SHAHBAHRAMI A, JUURLINK B, VASSILIADIS S. Performance impact of misaligned accesses in SIMD extensions[C]//Proceedings of the 17th Annual Workshop on Circuits, Systems and Signal Processing (ProRISC 2006). [S.l.:s.n.], 2006: 334-342.

[94] WU P, EICHENBERGER A E, WANG A. Efficient SIMD code generation for runtime alignment and length conversion[C]//Proceedings of International Symposium on Code Generation and Optimization. Piscataway: IEEE Press, 2005: 153-164.

[95] CRUZ-AYOROA A J. Machine learning driven compiler tuning[D].

Fukuoka: Kyushu University, 2015.

[96] TROUVÉ A, CRUZ A J, MURAKAMI K J, et al. Guide automatic vectorization by means of machine learning: a case study of tensor contraction kernels[J]. IEICE Transactions on Information and Systems, 2016, E99.D(6): 1585-1594.

[97] ZHOU H, XUE J L. A compiler approach for exploiting partial SIMD parallelism[J]. ACM Transactions on Architecture and Code Optimization, 2016, 13(1): 1-26.

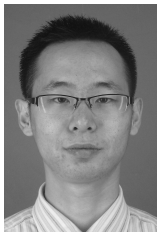
[98] STOCK K, POUCHET L N, SADAYAPPAN P. Using machine learning to improve automatic vectorization[J]. ACM Transactions on Architecture and Code Optimization, 2012, 8(4): 1-23.

[99] POHL A, COSENZA B, JUURLINK B. Vectorization cost modeling for NEON, AVX and SVE[J]. Performance Evaluation, 2020, 140/141: 102106.

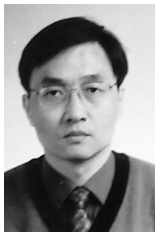
[100] 张媛媛, 赵荣彩, 韩林. 基于多面体表示的向量化收益评估方法[J]. 计算机工程, 2012, 38(7): 266-268, 272.
ZHANG Y Y, ZHAO R C, HAN L. Vectorization benefit evaluation method based on polyhedron representation[J]. Computer Engineering, 2012, 38(7): 266-268, 272.

[101] 杜丽娜, 卓力, 杨硕, 等. 基于强化学习的移动视频流业务码率自适应算法研究进展[J]. 通信学报, 2021, 42(9): 205-217.
DU L N, ZHUO L, YANG S, et al. Survey on reinforcement learning based adaptive bit rate algorithm for mobile video streaming services[J]. Journal on Communications, 2021, 42(9): 205-217.

[作者简介]



冯竞舸（1988− ）男，满族，河北临城人，中国科学院大学博士生，主要研究方向为编译技术及性能优化技术。



贺也平（1962− ），男，甘肃兰州人，博士，中国科学院软件研究所研究员、博士生导师，主要研究方向为基础软件、系统安全。

陶秋铭（1979− ），男，江苏南通人，博士，中国科学院软件研究所副研究员、硕士生导师，主要研究方向为操作系统、编译技术、软件工程。