http://bhxb.buaa.edu.cn jbuaa@buaa.edu.cn
DOI: 10.13700/j.bh.1001-5965.2023.0208

基于 RISC-V 向量扩展的图像预处理加速方法

刘强[1, 2, *]，尹蔚[1, 2]，李凯[1, 2]

(1. 天津大学 微电子学院，天津 300072；2. 天津市成像与感知微电子技术重点实验室，天津 300072)

摘 要：作为卷积神经网络 (CNN) 计算的前序步骤，图像预处理不可或缺又非常耗时。为加速图像预处理，提出一种基于 RISC-V 向量扩展的加速方法，对灰度化、标准化、高斯滤波等 11 种图像预处理算法进行加速。从计算模式上将 11 种图像预处理算法归为 4 类，并基于 RISC-V 向量扩展对各类图像预处理算法设计了加速方案；为进一步提高性能，新增 6 条自定义的向量指令，并通过修改编译器和设计硬件模块实现了 6 条自定义向量指令；使用现场可编程门阵列 (FPGA) 进行测试，并分析了向量处理器配置对性能和资源消耗的影响。结果显示：所提方法相比标量处理器实现了 3.13～9.97 倍的加速效果，可有效解决图像预处理在深度学习过程中的性能瓶颈问题。

关 键 词：卷积神经网络；预处理；RISC-V；向量扩展；算法加速
中图分类号：TN402
文献标志码：A **文章编号**：1001-5965(2025)04-1074-11

当前，卷积神经网络 (convolutional neural network, CNN) 广泛应用于图像分类、人脸识别等领域。作为 CNN 计算的前序步骤，图像预处理在扩充训练数据集、降低 CNN 推理或训练误差等方面均有重要意义。Pal 等[1] 调研了归一化、标准化和零分量分析 (zero component analysis, ZCA) 对 3 种不同结构的 CNN 识别准确率的影响，结果表明，经过预处理的数据识别准确率相比原始数据提升约 10%。Şaban 和 Akdemir[2] 研究了归一化、中值滤波和维纳滤波等图像预处理算法对病理学图像识别的影响，结果显示，预处理使 CNN 训练收敛速度更快，并使推理准确率提升约 1.7%。多项研究表明，高效的图像预处理算法可有效改善 CNN 的性能[3-4]。

CNN 常用的图像预处理算法从应用角度可以分为 3 类：①统一图像或数据格式，以满足 CNN 的输入要求，如标准化、归一化、图像缩放、通道转换等；②扩充 CNN 训练数据集，在许多应用场景中，供 CNN 训练的数据集往往是不充分的，预处理能够有效增加 CNN 训练的样本量，从而提高 CNN 的泛化能力，常见的扩充数据集的图像预处理算法有旋转、镜像、平移、裁剪、亮度调整、对比度调整、添加随机噪声等；③图像增强，目的是提高推理速度和准确率，常见的方法包括灰度化、二值化、各类滤波算法、图像分割、边缘检测、锐化 (边缘增强) 等。

在嵌入式应用环境中，CPU+CNN 加速器的异构组合是常见的深度学习计算平台。在该异构平台中，CPU 负责图像的预处理和加速器的控制，加速器负责 CNN 的计算。在先前的研究中，大量的工作集中在 CNN 推理或训练部分的优化[5-6]。然而，在嵌入式平台中，由于 CPU 并行度低，性能不佳，图像预处理也会成为瓶颈。在多项研究[7-9] 中，图像预处理占据了 60% 以上的计算时间。因此，加速图像预处理成为了进一步提升性能的关键。

为解决这一问题，NVIDIA 发布了图像预处理计算库 NVIDIA Data Loading Library (DALI)[10]，该

收稿日期：2023-04-24；**录用日期**：2023-05-26；**网络出版时间**：2023-06-09 10:13
网络出版地址：link.cnki.net/urlid/11.2625.V.20230608.1155.002
基金项目：国家自然科学基金 (U21B2031)
* **通信作者**. E-mail：qiangliu@tju.edu.cn

引用格式：刘强，尹蔚，李凯. 基于 RISC-V 向量扩展的图像预处理加速方法 [J]. 北京航空航天大学学报，2025，51（4）：1074-1084.
LIU Q，YIN W，LI K. Image preprocessing acceleration method based on RISC-V vector extension[J]. Journal of Beijing University of Aeronautics and Astronautics，2025，51（4）：1074-1084（in Chinese）.

计算库能够高效地进行 CNN 计算中的图像解码和预处理等操作, 其加速原理是将在 CPU 上进行的图像预处理工作部署到图形处理单元 (graphics processing unit, GPU) 上, 但是, DALI 仅能被应用在 CPU+GPU 的深度学习平台, 不适用于 CPU+CNN 加速器的嵌入式深度学习环境。Ma[11] 通过高层次综合 (high level synthesis, HLS) 的方法将灰度化、图像缩放和显著性分析算法固化成 IP 核, 相比 ARM CPU 实现了约 20 倍的加速效果。

RISC-V 是一种开源的 CPU 指令集架构 (instruction set architecture, ISA)。除了标准的整数指令集外, RISC-V 还针对不同的应用需求定义了扩展指令集, 如嵌入式扩展 (embedded extension)、位操作扩展 (bit manipulation extension)、向量扩展 (vector extension)、加密扩展 (cryptographic extension) 等。此外, RISC-V 还支持用户自定义指令集。RISC-V 指令集扩展在算法加速、硬件安全等领域表现出优秀的性能和良好的灵活性。Kuo 等[12] 扩展了一组迦瓦罗域的 RISC-V 计算指令, 对非二进制纠错码和后量子密码算法进行加速, 实现了约 5 倍的加速效果, 同时硬件逻辑资源消耗仅增加 1.27%; Razilov 等[13] 使用高性能 RISC-V 向量处理器 Ara[14] 和 RISC-V 向量扩展 (RISC-V vector extension, RVV) 指令来加速广义频分复用 (generalized frequency division multiplexing, GFDM) 算法, 实现了约 60 倍的加速效果; 刘强和李一可[15] 基于 RISC-V 指令扩展提出了一种可配置的故障注入检测方法, 支持时间冗余和信息冗余 2 种检测模式, 相比单一的信息冗余检测方法, 故障检测率提高了 13.34%, 仅引入 4.4% 的资源开销。

RVV 是一种单指令多数据流 (single instruction multiple data, SIMD) 的技术。RVV1.0[16] 是 RVV 的技术规范, 在 2021 年被批准。支持 RVV 的 RISC-V 处理器有用于实时和嵌入式计算的向量处理器 Vicuna[17]、64 位高性能向量处理器 Ara[14] 等。

由于不同网络结构和应用场景所需的图像预处理算法不同, 图像预处理难以被设计成专用集成电路 (application specific integrated circuit, ASIC) 进行加速。同时, 图像预处理算法对并行度要求高, 通用的嵌入式 CPU 往往并行计算能力较差。RVV 能够增强 RISC-V CPU 的并行计算能力, 还可以通过自定义向量指令实现对特定应用的定制化加速。因此, 本文采用 RVV 方案对图像预处理进行加速。

使用 RISC-V 向量处理器加速图像预处理算法面临以下 2 个问题: ①图像预处理算法种类多, 难以设计通用加速方案; ②现有标准 RISC-V 指令集不能完全满足图像预处理加速的需求。

为解决上述问题, 本文面向常用的 CNN 图像预处理算法提出了一种基于 RVV 的加速方案。

1) 针对标准化、缩放、旋转、镜像、平移、亮度调整、灰度化、二值化、高斯滤波、拉普拉斯滤波、索贝尔边缘检测 11 种常用的 CNN 图像预处理算法, 按照计算模式分为 4 类, 对每类算法设计了加速方案。

2) 在标准向量扩展的基础上新增了 6 条自定义向量指令, 用于进一步加速图像预处理算法中的操作。为实现这些指令, 对编译器进行修改, 并设计了相应的译码和计算模块。

3) 使用现场可编程门阵列 (field programmable gate array, FPGA) 对本文方法和硬件架构进行验证和评估, 实验结果显示, 与标量处理器相比, 本文设计的加速方法能够实现 3.13～9.97 倍的加速。

1　硬件架构

在嵌入式深度学习应用中, 硬件资源和功耗往往是受限的。因此, 本文采用轻量级处理器 ibex[18]+向量协处理器 Vicuna[17] 的组合来开展研究, 如图 1 所示。其中, ibex 是 32 位的标量主处理器, 设有 2 级流水线, 支持 RV32IEMCB 指令集, 该处理器核的优点是可配置且轻量化, 适合嵌入式控制应用。Vicuna 是 32 位整数向量协处理器, 支持 8、16、32 位的向量元素宽度, 其遵循 RVV1.0 规范, 实现了 Zve32x 指令集 (向量扩展的一个子集)。

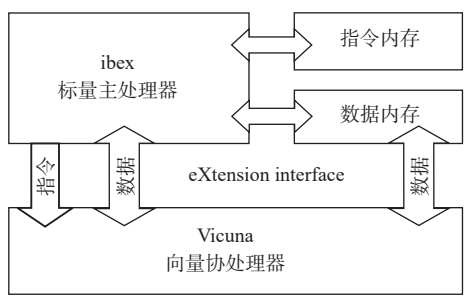

图 1　硬件平台架构

Fig. 1　Hardware architecture

ibex 和 Vicuna 通过 eXtension interface[19] 相互连接。当取到标量指令时, 由主处理器 ibex 执行; 当取到向量指令时, ibex 会将该指令通过 eXtension interface 发送至 Vicuna 执行。Vicuna 在执行向量指令时, 能够直接从数据内存中读取数据到向量寄存器文件 (vector register file, VRF) 中, 并在计算完成后将结果写回数据内存。在此期间, 如果存在数据依赖, 则 ibex 会停滞流水线直到 Vicuna 完成写

回，如果不存在数据依赖，则 ibex 可以和 Vicuna 同时运行。

2 图像预处理算法加速方案设计

由于卷积运算的独特性，本文按照是否涉及卷积运算，将 11 种图像预处理算法分为非卷积类和卷积类。卷积类算法包括高斯滤波、拉普拉斯滤波和索贝尔边缘检测。非卷积类算法按照特点又可以分为 3 类，分别为像素位置变化类、像素数值变化类及求全局平均值类。其中，像素位置变化类包括旋转、镜像、平移和缩放，像素数值变化类包括灰度化、二值化和亮度调整，求全局平均值类包括标准化。

2.1 非卷积类算法

2.1.1 像素位置变化类算法

涉及像素位置变化的算法包括旋转、镜像、平移和缩放。其中，旋转和缩放采用最近邻插值法，其他插值方法（如线性插值法）由于需要单独计算每个像素点数值的变化，不适合进行向量化。

这类算法的特点是：输出图像与输入图像之间，仅像素值的存储地址发生了变化，像素值本身并未发生变化。本文方法使用 RVV 中的索引加载 (indexload) 指令对这类算法实现定制化加速。

RVV 中的加载指令用于实现从数据内存中读取数据并加载到向量处理器的向量寄存器文件中。按照加载方式的不同，加载指令可以分为以下 3 类：①unit-strideload，该指令能够从内存中加载一组相邻的数据元素到向量寄存器中，是最常用的加载指令；②stridedload，该指令能按照一定的步长 (stride) 间隔加载一组固定间隔的数据到向量寄存器中；③indexload，即索引加载指令，该指令能够按照索引矩阵中描述的顺序加载一组数据元素到向量寄存器中。

索引加载指令能够按照既定的顺序加载数据。而涉及像素位置变化的图像预处理算法本质上是数据存储位置的变化。当使用索引加载指令实现这些算法时，原有的图像数据被按照索引矩阵描述的顺序读取到向量寄存器中重新排列，再写回数据内存，以此完成像素数据位置的变化。

如图 2 所示，以一张 3×3 大小的图像顺时针旋转 90°为例，说明索引加载指令如何实现像素位置的变化。

图 2 最上方是索引矩阵，左侧是旋转前后的图像，右侧是旋转前后的图像像素值在一维数组中存储的变化。在索引加载指令操作过程中，输入像素值按照式（1）进行重排序：

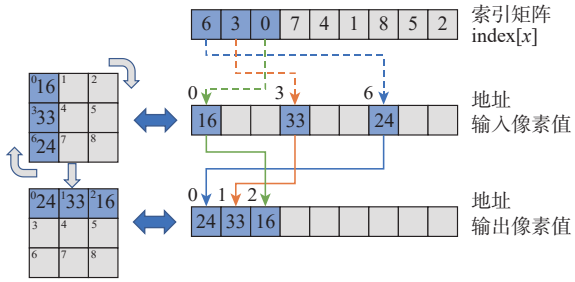

图 2 索引加载指令实现像素位置变化

Fig. 2 Pixel location change by index loading instruction

$$\text{output}[x] = \text{input}\left[\text{index}[x]\right] \tag{1}$$

例如，索引矩阵中第一个元素为 6，则向量处理器会加载输入数组地址为 6 的元素到向量寄存器地址 0 的位置，后续元素同理。加载完毕后，将向量寄存器中的数据写回数据内存，以此来实现像素位置的变化。标量处理器完成同样的操作需要对每个数据进行单独读写，这需要大量的指令开销，而一条索引加载指令能够完成一批数据的位置变化，显著提高效率。

另外，索引矩阵决定了算法中像素位置变化的规则，如果多张图像都采用同一种变换方式，则索引矩阵可以被提前计算好并作为常量存储，以减少计算量。

2.1.2 像素数值变化类算法

涉及像素数值变化的算法包括灰度化、二值化和亮度调整。这类算法的特点是：输出图像和输入图像之间仅像素值发生变化，像素位置并未发生变化。在实现过程中，可以使用常规的 unit-strideload 加载指令加载数据，按照算法原理使用对应的向量算数运算指令进行计算，将计算结果写回数据内存。

本节以灰度化为例说明加速这类算法的方法。灰度化有多种实现方法，如分量法、最大值法、均值法和加权平均法。本文采用图像处理库 OpenCV 中的加权平均来保证通用性。

$$\text{Gray}[x] = 0.072B[x] + 0.715G[x] + 0.213R[x] \tag{2}$$

式中：Gray[x] 表示像素 x 的灰度值；B、G、R 分别表示蓝、绿、红 3 个颜色通道。

计算灰度值的加权平均公式有浮点乘法，需要消耗大量的计算资源。因此，本文将式（2）中等号右侧所有系数都乘以 256 再右移 8 位，如式（3）所示。这种方式使用 8 位的整数乘法和移位来代替浮点乘法，易于硬件实现。经过评估，由此带来的精度损失不超过 1%。

$$\text{Gray}[x] = (18B[x] + 183G[x] + 55R[x]) \gg 8 \tag{3}$$

使用 RVV 指令对算法进行实现，伪代码如算法 1 所示。其中，输入为 B、G、R 这 3 个通道的像

素值, 输出结果 RES 为灰度值; 循环次数 n 取决于需要处理的数据量和向量寄存器的长度; src 表示该变量存储在数据内存中, vec 表示该变量存储在向量处理器的寄存器中。

使用向量指令实现的灰度化算法可以分为以下 4 步: ①将数据从内存加载到向量寄存器中 (算法 1 的第 2～4 行)。②在 3 个通道上分别进行乘累加操作 (第 5～7 行), 其中, 括号内的第 1 个参数代表需要累加的数据, 后 2 个参数代表指令中的操作数 1 和操作数 2。在该算法使用的乘累加指令中, 操作数 1 的数据类型是整形, 操作数 2 的数据类型是向量寄存器的地址。在实际运算中, 会根据地址调取该向量寄存器中的所有元素进行计算, 实现 SIMD 的效果。③将乘累加后的结果右移 8 位 (第 8 行)。④将结果从向量寄存器写回数据内存 (第 9 行)。

算法 1　灰度化算法的向量实现。

输入: src_B, src_G, src_R。

输出: src_RES。

1. for i = 1 to n do
2. vec_B_i = VLOAD(src_B_i);
3. vec_G_i = VLOAD(src_G_i);
4. vec_R_i = VLOAD(src_R_i);
5. vec_RES$_i$ = VMAC(vec_RES$_i$,18,vec_B_i);
6. vec_RES$_i$ = VMAC(vec_RES$_i$,183,vec_G_i);
7. vec_RES$_i$ = VMAC(vec_RES$_i$,55,vec_R_i);
8. vec_RES$_i$ = VSHIFT(vec_RES$_i$,8);
9. src_RES$_i$ = VSTORE(vec_RES$_i$);
10. end for

2.1.3　求全局平均值类算法

许多 CNN 算法在图像数据输入卷积层前需要进行数据标准化的操作。标准化涉及求图像全局的平均值。求平均值要先求全局像素值的和, 本文使用规约求和指令实现该类算法的加速。

规约指令是 RVV 指令中的一类特殊指令, 能够对指定向量寄存器中的元素两两运算, 将其规约为一个标量值。RVV 中的规约指令有规约求和、规约求最大值、规约求与等。

首先, 使用规约求和指令求得所有像素的和; 然后, 使用移位代替除法求得图像的全局平均值; 最后, 在当前像素值的基础上调用向量减法指令减去全局平均值, 即可实现数据标准化。

2.2　卷积类算法

高斯滤波、拉普拉斯滤波和索贝尔边缘检测等算法的共同特点是需要对比或计算像素点与其周围像素点的关系, 类似卷积运算。因此, 本文设计了一种基于 RVV 的卷积加速方案。

2.2.1　卷积加速方案

使用 RVV 指令实现卷积操作的难点在于向量指令易于实现地址连续的操作, 但是卷积运算处理的二维数据的地址是不连续的。如图 3 所示, 左侧为原始图像, 中间为卷积核, 右侧为计算结果。在原始图像中, 像素点 a_i、b_i、c_i 的存储地址是连续的, 而像素点 c_i 和像素点 d_i 由于处在不同行, 其地址不是连续的, 这不利于向量指令进行算法实现。

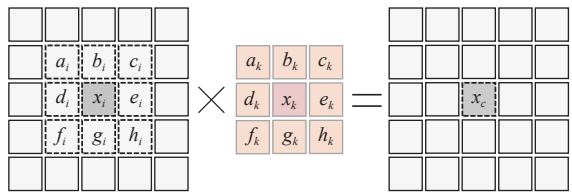

图 3　卷积计算过程

Fig. 3　Convolution computation process

在卷积运算过程中, 卷积核按照一定的步长遍历整个图像。以图 3 为例, 当卷积的步长为 1 时, 卷积核会均匀地滑过图像中的所有像素点。图像中的每个元素都要与卷积核中每个权重相乘。

基于以上分析, 本文方案第 1 步为计算该图像和卷积核中所有元素的乘积。这一步操作是地址连续的向量与标量的相乘, 易于向量实现。以图 3 中的卷积核为例, a_k 与图像所有像素点相乘得到第 1 张图像, b_k 与图像所有像素点相乘得到第 2 张图像, 后续同理, 一共产生 9 张图像 img1～img9, 如图 4 左侧所示。这 9 张图像称为临时卷积图像。

卷积运算的过程实际上是乘累加的过程, 第 1 步相当于完成了“乘累加”中的乘法操作, 下一步需要从 9 张临时卷积图像中找到需要的值并累加。

图 4 的最右侧是计算结果 RES, 其中, x_c 对应的 9 个临时乘积分别位于临时卷积图像 img1～img9 中 x_1～x_9 的位置。因此, $x_c = x_1 + x_2 + \cdots + x_9$。假设 x_c 在数组中的位置为 m, 在计算 x_c 时, 需采用以下公式:

$$\begin{aligned} x_c = \mathrm{RES}[m] = {} & \mathrm{img1}[m-r-1] + \mathrm{img2}[m-r] + \\ & \mathrm{img3}[m-r+1] + \mathrm{img4}[m-1] + \mathrm{img5}[m] + \\ & \mathrm{img6}[m+1] + \mathrm{img7}[m+r-1] + \\ & \mathrm{img8}[m+r] + \mathrm{img9}[m+r+1] \end{aligned} \tag{4}$$

式中: r 为临时卷积图像一行的元素数量。然而, 由于 x_1～x_9 在数组中的位置不统一, 这种计算方式也难以进行向量实现。

2.1.1 节中提到的索引加载指令能够改变像素的存储位置。为使操作易于向量化, 本文方法使用索引加载指令索引 img1～img9, 以此统一 x_1～x_9 在数组中的位置。索引矩阵决定了像素位置变换的

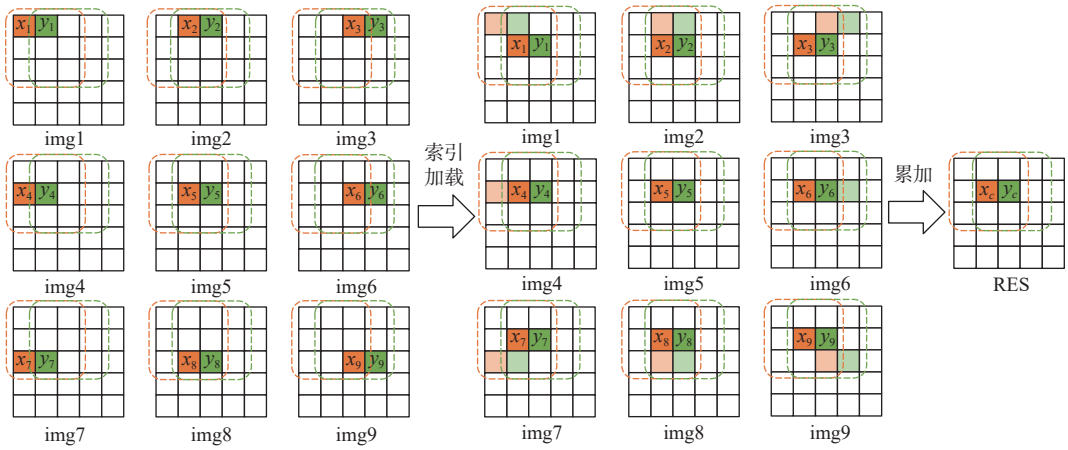

图4 索引加载和累加计算过程

Fig. 4 Computation process of index loading and accumulation

规则，如img1对应的索引矩阵负责将所有像素往右下移动一格，img2对应的索引矩阵负责将所有像素往下移动一格，其余索引矩阵同理，共9个。索引加载过程如图4所示。

索引加载完成后，计算 x_c 采用的公式为

$$\begin{aligned} x_c = \mathrm{RES}[m] = {} & \mathrm{img1}[m] + \mathrm{img2}[m] + \mathrm{img3}[m] + \\ & \mathrm{img4}[m] + \mathrm{img5}[m] + \mathrm{img6}[m] + \\ & \mathrm{img7}[m] + \mathrm{img8}[m] + \mathrm{img9}[m] \end{aligned} \tag{5}$$

这种计算方式易于进行向量实现。此外，不仅计算 x_c 需要的 x_1～x_9 位置进行了统一，图像中的其他点，如计算 y_c 需要的 y_1～y_9 位置也得到了统一。调用向量指令将索引后的9张临时卷积图像累加，便得到了整张图像的卷积计算结果。至此，完成了“乘累加”中的累加。

本文提出的卷积加速方法可以概括为以下4步：①将图像与卷积核中的每个权重相乘，得到9张临时卷积图像；②计算9个索引矩阵；③用索引矩阵索引9张临时卷积图像；④累加，得到实际卷积结果。值得注意的是，如果需要处理大量图像的卷积，9个索引矩阵实际上只需要被计算一次。

2.2.2 卷积加速方案实例——高斯滤波

高斯滤波算法通过计算像素值的加权和来平滑图像并减少噪声，提高CNN推理的准确率。其算法原理是：取图像中每个像素点相邻的矩形窗口，按照权重矩阵计算窗口内所有像素点的加权和。计算式如式(6)所示。高斯滤波的计算过程本质上是一种卷积。权重矩阵的权重值遵循二元正态分布，越接近矩阵中心，权重值越大。

$$C(x,y) = \sum_{j=y-n}^{y+n} \sum_{i=x-n}^{x+n} W(i,j) * P(i,j) \tag{6}$$

式中：C 为计算结果；W 和 P 分别为权重和该点周围的像素。

本文提出的卷积加速方法还有一个常规卷积计算方法不具备的优势，即当卷积核中有重复的权值时，可以避免重复计算。图5展示了图像中像素点 a、b、c 的高斯滤波计算过程。高斯滤波权重矩阵中，距离卷积核中心距离相等的点权值相同。根据2.2.1节的分析，图像中每个像素点在卷积计算的过程中会与权重矩阵中的所有元素相乘，即乘4次 W_1，4次 W_2，1次 W_3。重复的计算会导致资源和功耗的浪费，实际上，每个点只需要分别与 W_1、W_2、W_3 相乘1次。因此，本文提出的卷积加速方法能够规避卷积核中有重复权值产生的额外乘法计算。高斯滤波算法向量实现伪代码如算法2所示。

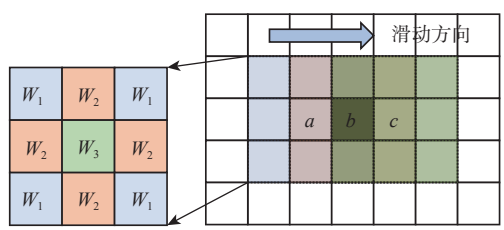

图5 高斯滤波计算过程

Fig. 5 Gaussian filter computation process

将输入图像从数据内存加载到向量寄存器(算法2的第2行)后，先计算图像的所有像素点与 W_1、W_2、W_3 这3个权重的乘积，由于权重均为浮点数据，采取2.1.2节中用到的方法，用整数乘法和移位来代替浮点乘法，得到3张临时卷积图像(第4～6行)。计算完毕后，利用索引加载指令，索引在上一步中计算完毕的临时卷积图像。直到9张临时卷积图像都被索引并累加完毕(第9～11行)。最后，将计算结果写回数据内存(第13行)。

算法2 高斯滤波算法的向量实现。

输入：src_input，W_1，W_2，W_3，src_index$_1$～src_index$_9$。

输出：src_RES。

1. for i = 1 to n do
2. vec_input$_i$ = VLOAD(src_input$_i$);
3. for j = 1 to 3 do
4. vec_W_j = VMUL(vec_input$_i$, W_j);
5. vec_W_j = VSHIFT(vec_W_i, 8);
6. src_W_j = VSTORE(vec_W_j);
7. end for
8. for k = 1 to 9 do
9. vec_index$_k$ = VLOAD(src_index$_k$);
10. vec_RES$_k$ = VINDEX(src_W, vec_index$_k$);
11. vec_RES$_k$ = VADD(vec_RES$_k$, vec_RES$_k$);
12. end for
13. src_RES = VSTORE(vec_RES);
14. end for

3　自定义向量指令

在使用 RVV 指令对图像预处理算法进行加速时, 标准的 RVV 并不能完全满足算法向量化的要求。例如, 索贝尔边缘检测中需要进行阈值判断并置位, 这在标准 RVV 中不被支持。

因此, 为实现标准 RVV 中不支持的功能和进一步对算法进行加速, 本文新增了 6 条自定义向量指令, 并通过修改编译器、设计硬件电路的译码和执行模块实现了这 6 条指令。

3.1　指令定义与结构

新增的 6 条自定义向量指令如图 6 所示。指令的长度为 32 位, 其中, 不同的编码段代表的意义不同。OP-V major code 表示该指令属于 RVV 指令; funct6 用于和该指令集内的其他指令进行区分。因此, 在自定义向量指令时, 该编码段不能与已有的标准向量指令重复。考虑到自定义向量指令的需求, RVV1.0 预留了一些没有被使用的编码空间。每条指令都有 2 个操作数, 使用 funct3 来区分 2 个操作数的类型, 其中, funct3 = 110 和 100 分别表示该指令的 2 个操作数分别为向量和标量; vs2 和 rs1 分别表示向量源寄存器和标量源寄存器的地址, vd 表示目的寄存器的地址; vm 表示掩码位。

funct6				funct3		OP-V major code	
111001	vm	vs2	rs1	100	vd	1010111	vset255.vx
010110	vm	vs2	rs1	100	vd	1010111	vset0.vx
111001	vm	vs2	rs1	110	vd	1010111	vabs.vx
010110	vm	vs2	rs1	110	vd	1010111	vcmpset.vx
010101	vm	vs2	rs1	110	vd	1010111	vwmsau.vx
001101	vm	vs2	rs1	110	vd	1010111	vmuls.vx

图 6　自定义向量指令格式

Fig. 6　Customized vector instruction format

在本文实现的图像预处理算法中, 需要扩展指令的操作有: 在拉普拉斯滤波和亮度调整算法中, 需要判断像素值改变后是否越界 (0～255); 在索贝尔边缘检测和二值化算法中, 需要判断像素值是否大于阈值并进行置位; 在索贝尔边缘检测算法中, 需要计算数据的绝对值。

针对上述操作, 本文新增了 4 条自定义指令, 其逻辑功能如表 1 所示。vset255 和 vset0 这 2 条指令能够将越界的数据调整为上限或下限数据; vabs 指令判断输入操作数的正负, 如果操作数为负则取反, 否则返回原值; vcmpset 指令先判断 2 个操作数的大小, 如果操作数 1 大于操作数 2, 则返回一个置位值, 否则返回另一个置位值。

此外, 本文使用乘法和移位代替浮点运算, 为进一步减少指令调度, 提高加速效果, 本文将这 2 步操作进行合并, 新增了无符号加宽型向量乘移位累加 vwmsau 和向量乘移位 vmuls, 其逻辑功能如表 1 所示, 2 条指令的主要区别在于是否累加。

表 1　自定义向量指令功能

Table 1　Function of customized vector instructions

指令	功能
vset255	vd = vs1 >rs2 ? 255: vs1
vset0	vd = vs1 >rs2 ?vs1:0
vabs	vd = vs1 > rs2 ? vs1: −vs1
vcmpset	vd = vs1 > rs2 ?255: 0
vwmsau	vd = vd + (vs1·rs2 >> 8)
vmuls	vd = vs1·rs2 >>8

3.2　自定义向量指令的实现

自定义指令的实现分为 2 部分: 对编译器的修改和对处理器本身的修改。

以 GNU 编译套件 (GNU compiler collection, GCC) 为例, 在修改编译器前, 按照 RVV 指令规范定义向量编码。将自定义的指令变量添加到 GCC 的二进制工具 (RISC-V binutils) 中并重新编译, 使 GCC 能够成功识别自定义向量指令并生成对应的机器码。

自定义向量指令的硬件实现如图 7 所示。左侧是标量处理器 ibex 和数据内存, 右侧是向量处理器 Vicuna。Vicuna 中的比较置位模块实现了 vset255、vset0、vabs 和 vcmpset 指令, 乘移位模块实现了 vwmsau 和 vmuls 指令。

实现自定义向量指令需要先对指令译码器进行修改, 译码器根据指令的 funct6 和 funct3 编码段来辨别自定义向量指令, 按照指令的操作类型生成控制信号及操作数据, 并发送到对应的执行模块。

为实现 vset255、vset0、vabs 和 vcmpset 指令, 本

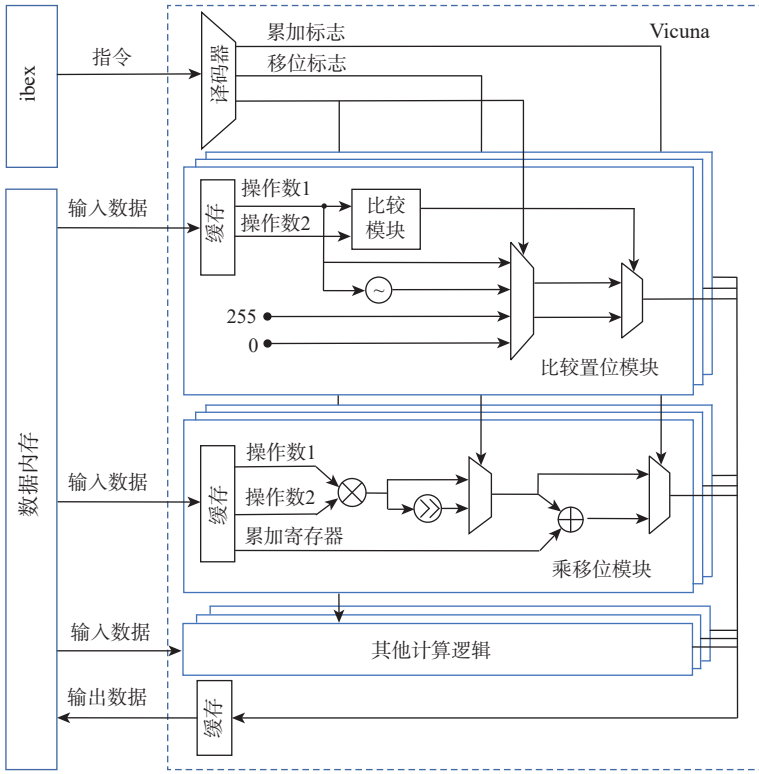

图7 自定义向量指令的硬件实现

Fig. 7 Hardware implementation of customized vector instructions

文设计了比较置位模块。数据进入该模块时，会先进入比较置位模块进行大小比较。在该模块中，根据指令的功能定义设置了4种不同的置位值：vs1(操作数1)、−vs1、255和0，根据译码阶段的控制信号及比较置位模块的输出信号，通过两级多路复用器(multiplexer, MUX)选择置位值输出。这4条指令复用了比较置位模块。

为实现vwmsau和vmuls指令，本文设计了乘移位模块。该模块复用了Vicuna原有的乘法模块逻辑，仅增加移位标志、移位器和MUX，以减少资源开销。如果译码模块遇到这2条指令，移位标志就会被置为1。由于没有影响原乘法器的功能，乘移位模块根据不同指令能够实现4种操作，分别为乘法、乘累加、乘移位和乘移位累加。

4 实验结果与分析

4.1 仿真及实验环境

本文选择riscv32-unknown-elf-gcc v12.0.1作为编译工具，选择Verilator v4.210作为仿真工具来测试自定义模块的功能正确性和图像预处理算法的加速效果。使用Vivado v2019.2进行硬件电路的综合和实现，并部署到Xilinx Artix-7 FPGA开发板上进行性能验证，评估面积和功耗，工作频率为50 MHz。用于测试的图像数据来自数据集CIFAR-10[20]，图像大小为32×32×3。

4.2 不同处理器配置下的性能与资源消耗

本文以周期数为指标评价性能。由于时间=周期数/频率，经评估，本文设计并未影响处理器的关键路径，其依然能按照原有的最大频率工作。因此，周期数的变化能准确反映计算时间的变化。

不同的处理器配置会对性能和面积产生影响。本节就2种参数的配置对图像预处理加速实验结果的影响展开讨论，分别为向量寄存器长度VLEN和向量处理单元位宽PIPE_W。

4.2.1 向量寄存器长度

RISC-V向量处理器设有32个向量寄存器v0～v31，每个向量寄存器的长度为VLEN，单位为bit。RVV中，向量寄存器长度不是固定的，Vicuna的向量寄存器长度可配置的范围为64～2 048 bit。向量寄存器长度可配置是指在硬件电路中，向量寄存器有不同长度的实现，而不是软件层面的配置。

图8展示了随向量寄存器长度的变化，单张图像灰度化算法向量实现周期数的变化(向量处理单元位宽固定为32 bit)。可以发现，随着向量寄存器长度增长，算法需要的周期数显著减少，向量寄存器长度从64 bit增加到1 024 bit，周期数减少了60.2%。当向量寄存器长度增大时，单条向量指令(如加载指令、运算指令和存储指令)处理的数据量得到提升。

不同向量寄存器长度配置下，FPGA查找表

(look up table, LUT)、触发器 (flip flop, FF)、块随机存取存储器 (block random access memory, BRAM) 和数字信号处理单元 (digital signal processor, DSP) 的消耗量如表 2所示。向量寄存器长度的变化不会影响 BRAM 和 DSP 的用量, 但 LUT 和 FF 的消耗会增加。向量寄存器长度的变化对性能和资源消耗均有明显影响, 因此, 应针对不同的应用环境要求选择合适的向量寄存器长度配置。

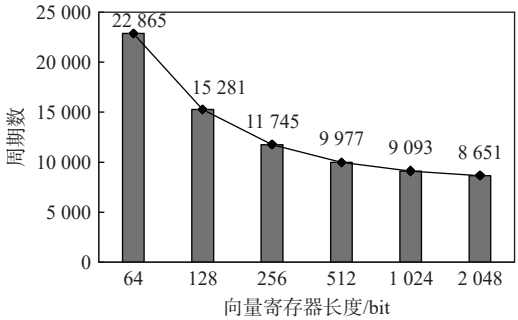

图 8　不同向量寄存器长度配置下的灰度化周期数

Fig. 8　Cycles of grayscale processing under different vector register length configurations

表 2　不同向量寄存器长度配置下资源消耗及功耗

Table 2　Resource and power consumption under different vector register length configurations

向量寄存器长度/bit	LUT	FF	BRAM	DSP	功耗/W
64	9 880	5 976	64	7	0.336
128	10 373	6 964	64	7	0.346
256	11 312	8 716	64	7	0.353
512	13 262	12 168	64	7	0.373
1 024	17 374	19 079	64	7	0.424
2 048	41 836	33 068	64	7	0.533

4.2.2　向量处理单元位宽

向量处理单元位宽是向量处理单元的位宽, 即算术逻辑单元 (arithmetic logical unit, ALU)、乘法单元等运算模块能同时处理的数据量, 代表了向量处理单元的计算性能。不同向量处理单元位宽对灰度化算法周期数的影响如图 9 所示。资源消耗如表 3 所示 (向量寄存器长度固定为 2 048 bit)。向量处理单元位宽取值范围为 32～VLEN/2, 单位为 bit。

向量处理单元位宽影响了 LUT、FF 和 DSP 资源的使用量, 但其增加对周期数的影响并不明显。向量处理单元位宽从 32 bit 提升到 512 bit, 周期数仅减少了 2.5%。由于图像预处理算法是数据密集型的应用, 限制算法效率的往往不是向量处理单元的计算性能, 而是可用的内存带宽。Vicuna 的内存读取端口宽度为 32 bit, 限制了算法的效率, 因此, 向量处理单元位宽的变化并不会显著影响性能。此外, 只涉及像素位置变化的算法, 如旋转、镜像、

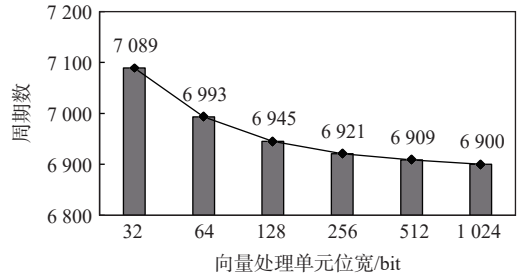

图 9　不同向量处理单元位宽配置下的灰度化周期数

Fig. 9　Cycles of grayscale processing under different vector processing pipe width configurations

表 3　不同向量处理单元位宽配置下资源消耗及功耗

Table 3　Resource and power consumption under different vector processing pipe width configurations

向量处理单元位宽/bit	LUT	FF	BRAM	DSP	功耗/W
32	41 386	33 068	64	7	0.533
64	42 693	33 642	64	11	0.539
128	44 133	34 772	64	19	0.545
256	47 470	37 058	64	35	0.577
512	52 061	41 584	64	67	0.607
1 024	81 279	48 705	64	131	0.738

平移等, 不会用到逻辑运算单元, 向量处理单元位宽的增加对这些算法的效率没有影响。因此, 在图像预处理类应用中, 本文选择尽量小的向量处理单元位宽来减少资源消耗。

4.3　加速效果及分析

结合 4.2 节中对向量处理器关键参数配置的分析, 本节的实验结果均基于向量寄存器长度为 2 048 bit、向量处理单元位宽为 32 bit 的处理器配置。以标量处理器 ibex 的执行周期数为基准, 列举了文中方法对各类图像预处理算法的加速效果。

使用索引加载指令对像素位置变化类算法进行加速, 效果如表 4 所示。实验结果显示, 本文方法对像素位置变化类算法相比标量处理器实现了 6.93～9.75 倍的加速效果。索引加载指令能够对这类算法实现高效的加速。

对像素数值变化类算法进行加速, 效果如表 5 所示。实验结果显示, 本文方法对 3 种像素数值变化类算法相比标量处理器实现了 6.69～9.97 倍的加速。标准向量扩展仅对灰度化算法有较好的加

表 4　像素位置变化类算法加速效果

Table 4　Acceleration results of pixel location change algorithms

算法	标量基准周期数	RVV周期数	加速倍数
旋转/镜像	46 728	4 791	9.75
平移	47 439	5 679	8.35
缩放	40 681	5 868	6.93

速效果，对二值化和亮度调整算法没有明显的加速效果。其中，二值化算法无法使用标准向量扩展指令加速，因此，其周期数与标量基准一致；由于标准向量指令集不支持亮度调整算法中的比较置位运算，这部分运算需要在标量处理器上执行，额外的标量处理器读写和运算操作导致加速效果被严重稀释。

高效的自定义RVV指令实现了标准RVV指令集不支持的功能，解决了加速效果被标量处理器稀释的问题，使得二值化和亮度调整算法的效率大幅提升。合并乘法与移位的扩展指令也使得灰度化算法得到进一步的加速。

求全局平均值类算法为标准化算法。本文使用规约求和指令对这类算法进行加速，效果如表6所示。实验结果显示，本文方法对求全局平均值类算法实现了7.52倍的加速效果。

表5 像素数值变化类算法加速效果

Table 5 Acceleration results of pixel value change algorithms

算法	标量基准周期数	标准RVV周期数	标准RVV加速倍数	自定义RVV周期数	自定义RVV加速倍数
灰度化	60 488	10 786	5.61	8 651	6.99
二值化	42 576	42 576	1.00	4 272	9.97
亮度调整	116 627	80 730	1.44	17 422	6.69

表6 求全局平均值类算法加速效果

Table 6 Acceleration results of global average computation algorithms

算法	标量基准周期数	RVV周期数	加速倍数
标准化	65 620	8 724	7.52

卷积类算法包括高斯滤波、拉普拉斯滤波和索贝尔边缘检测，应用本文提出的卷积加速方案对这类算法进行加速，结果如表7所示。实验结果显示，本文方法对3种卷积类算法相比标量处理器实现了3.13～4.26倍的加速。应用本文提出的卷积加速方案，在使用标准RVV指令时，能取得一定的加速效果，但仍存在加速效果被稀释的现象。而结合卷积加速方案和自定义RVV指令能够解决这一问题，运算效率得到进一步提升。

表7 卷积类算法加速效果

Table 7 Acceleration results of convolution algorithms

算法	标量基准周期数	标准RVV周期数	标准RVV加速倍数	自定义RVV周期数	自定义RVV加速倍数
高斯滤波	268 604	66 178	4.06	63 037	4.26
拉普拉斯滤波	214 456	105 235	2.04	57 650	3.72
索贝尔边缘检测	214 783	197 202	1.09	68 542	3.13

4.4 CNN应用中的资源消耗与功耗

以文献[21]中的MobileNet加速器为例，预估本文硬件平台在实际CNN加速应用中的资源消耗，如表8所示。Vicuna的配置为：向量寄存器长度为2 048 bit，向量处理单元位宽为32 bit。应用于CPU+CNN加速器的深度学习平台中，向量处理器导致的资源消耗增长仅约10%，自定义指令导致的额外硬件资源开销小于0.1%。

表8 资源消耗及功耗

Table 8 Resource and power consumption

硬件架构	LUT	FF	BRAM	DSP	功耗/W
ibex+CNN	310 840	279 836	957	2 161	11.65
ibex+Vicuna+CNN	349 618	311 699	1 005	2 168	11.88
ibex+自定义Vicuna+CNN	349 835	311 964	1 005	2 168	11.88

4.5 在CNN中的应用实例

本节选取嵌入式图像分类应用[9]和车牌识别应用为例，预估本文方法在深度学习应用场景中的效果。

如图10所示，在图像分类应用中，CPU预处理阶段先进行图像的缩放、灰度化和归一化，再由二值神经网络(binary neural network, BNN)加速器进行推理分类。整个计算过程中，加速器占用时间为24%(137 ms)，CPU图像预处理占用时间为76%(433 ms)，其中，缩放占用30%(171 ms)，灰度化占用32%(182 ms)，归一化占用14%(80 ms)。应用本文方法，CPU图像预处理的占用时间大幅缩短，整体计算时间相比原来减少54.2%(309 ms)。

如图11所示，在车牌识别应用中，预处理阶段进行灰度化、高斯滤波和索贝尔边缘检测，再由3层的反向传播神经网络(back propagation neural

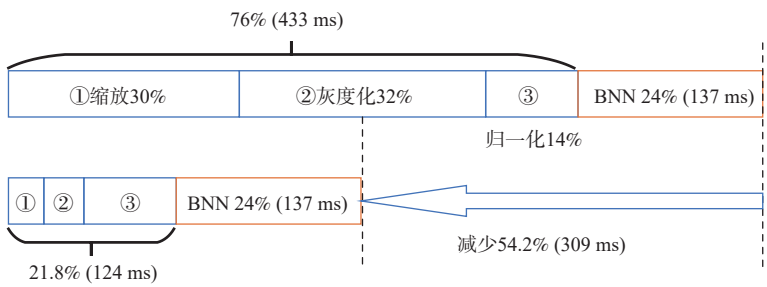

图10 图像分类应用中的效果

Fig. 10 Effect in image classification application

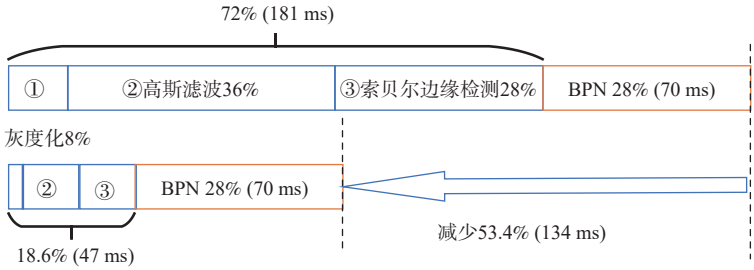

图 11　车牌识别应用中的效果

Fig. 11　Effect in license plate recognition application

network, BPN) 进行识别。BPN 占用时间为 28% (70 ms), CPU 图像预处理占用时间为 72%(181 ms), 其中, 灰度化占用 8%(20 ms), 高斯滤波占用 36% (91 ms), 索贝尔边缘检测占用 28%(70 ms), 应用本文方法, 整体计算时间相比原来减少 53.4%(134 ms), 有效解决了 CPU 图像预处理在 CNN 计算中的性能瓶颈问题。

5　结　论

1) 针对 11 种常见的图像预处理算法, 按照算法特点分为 4 类, 并针对每种分类设计了基于 RISC-V 向量扩展的加速方案。

2) 为解决标准向量扩展不能满足图像预处理加速需求的问题, 自定义了 4 条向量指令; 为进一步提高算法运算效率, 自定义了 2 条向量指令以合并乘法与移位运算。通过设计 RISC-V 处理器微架构实现了这 6 条指令。

3) 实验结果表明, 本文提出的方案与标量处理器相比实现了 3.13～9.97 倍的加速效果。在 2 种实际应用中, 预估能减少 50% 左右的整体计算时间, 可有效解决 CPU 图像预处理在 CNN 整体计算中的性能瓶颈问题。

参考文献 (References)

[1] PAL K K, SUDEEP K S. Preprocessing for image classification by convolutional neural networks[C]//Proceedings of the IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology. Piscataway: IEEE Press, 2016: 1778-1781.

[2] ŞABAN Ö, AKDEMIR B. Effects of histopathological image preprocessing on convolutional neural networks[J]. Procedia Computer Science, 2018, 132: 396-403.

[3] PITALOKA D A, WULANDARI A, BASARUDDIN T, et al. Enhancing CNN with preprocessing stage in automatic emotion recognition[J]. Procedia Computer Science, 2017, 116: 523-529.

[4] TABIK S, PERALTA D, HERRERA-POYATOS A, et al. A snapshot of image pre-processing for convolutional neural networks: case study of MNIST[J]. International Journal of Computational Intelligence Systems, 2017, 10(1): 555-568.

[5] DU C Y, TSAI C F, CHEN W C, et al. A 28 nm 11.2 TOPS/W hardware-utilization-aware neural-network accelerator with dynamic dataflow[C]//Proceedings of the IEEE International Solid-State Circuits Conference. Piscataway: IEEE Press, 2023: 1-3.

[6] KELLER B, VENKATESAN R, DAI S, et al. A 95.6-TOPS/W deep learning inference accelerator with per-vector scaled 4-bit quantization in 5 nm[J]. IEEE Journal of Solid-State Circuits, 2023, 58(4): 1129-1141.

[7] KARNIK T, KURIAN D, ASERON P, et al. A cm-scale self-powered intelligent and secure IoT edge mote featuring an ultra-low-power SoC in 14 nm tri-gate CMOS[C]//Proceedings of the IEEE International Solid-State Circuits Conference. Piscataway: IEEE Press, 2018: 46-48.

[8] NARAYANAN D, SANTHANAM K, PHANISHAYEE A, et al. Accelerating deep learning workloads through efficient multi-model execution[C]//Proceedings of the NeurIPS Workshop on Systems for Machine Learning. [S. l.]: NeurIPS, 2018: 20-27.

[9] JIA T Y, JU Y H, JOSEPH R, et al. NCPU: an embedded neural CPU architecture on resource-constrained low power devices for real-time end-to-end performance[C]//Proceedings of the 53rd Annual IEEE/ACM International Symposium on Microarchitecture. Piscataway: IEEE Press, 2020: 1097-1109.

[10] NVIDIA. NVIDIA data loading library (DALI) [EB/OL]. [2023-04-01]. https://github.com/NVIDIA/DALI.

[11] MA N. A SoC-based acceleration method for UAV runway detection image pre-processing algorithm[C]//Proceedings of the 25th International Conference on Automation and Computing. Piscataway: IEEE Press, 2019: 1-6.

[12] KUO Y M, GARCÍA-HERRERO F, RUANO O, et al. RISC-V Galois field ISA extension for non-binary error-correction codes and classical and post-quantum cryptography[J]. IEEE Transactions on Computers, 2023, 72(3): 682-692.

[13] RAZILOV V, MATÚŠ E, FETTWEIS G. Communications signal processing using RISC-V vector extension[C]//Proceedings of the International Wireless Communications and Mobile Computing. Piscataway: IEEE Press, 2022: 690-695.

[14] CAVALCANTE M, SCHUIKI F, ZARUBA F, et al. Ara: a 1-GHz scalable and energy-efficient RISC-V vector processor with multi-precision floating-point support in 22-nm FD-SOI[J]. IEEE Transactions on Very Large Scale Integration Systems, 2020, 28(2): 530-543.

[15] 刘强, 李一可. 基于指令扩展的 RISC-V 可配置故障注入检测方法[J/OL]. 北京航空航天大学学报, 2023(2023-03-10)[2023-04-01]. https://bhxb.buaa.edu.cn/bhzk/cn/article/doi/10.13700/j.bh.1001-5965.2022.0995.

LIU Q, LI Y K. Configurable fault detection method for RISC-V processors based on instruction extension[J/OL]. Journal of Beijing University of Aeronautics and Astronautics, 2023(2023-03-10)[2023-04-01]. https://bhxb.buaa.edu.cn/bhzk/cn/article/doi/10.13700/j.bh.1001-5965.2022.0995(in Chinese).

[16] ASANOVIC K. Vector extension 1.0[EB/OL]. (2021-09-20)[2023-04-01]. https://github.com/riscv/riscv-v-spec/releases/tag/v1.0.

[17] PLATZER M, PUSCHNER P. Vicuna: a timing-predictable RISC-V vector coprocessor for scalable parallel computation[C]//Proceedings of the 33rd Euromicro Conference on Real-Time Systems. Porto: [s. n.], 2021: 1-18.

[18] SCHIAVONE P D, CONTI F, ROSSI D, et al. Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-things applications[C]//Proceedings of the 27th International Symposium on Power and Timing Modeling, Optimization and Simulation. Piscataway: IEEE Press, 2017: 1-8.

[19] OpenHW Group. OpenHW group eXtension interface[EB/OL]. [2023-04-01].https://docs.openhwgroup.org/projects/openhw-group-core-v-xif/en/latest/index.html.

[20] ALEX K, VINOD N. The CIFAR-10 dataset[EB/OL]. [2023-04-01]. http://www.cs.toronto.edu/~kriz/cifar.html.

[21] YAN S, LIU Z Y, WANG Y, et al. An FPGA-based MobileNet accelerator considering network structure characteristics[C]//Proceedings of the 31st International Conference on Field-Programmable Logic and Applications. Piscataway: IEEE Press, 2021: 17-23.

Image preprocessing acceleration method based on RISC-V vector extension

LIU Qiang[1, 2, *], YIN Wei[1, 2], LI Kai[1, 2]

(1. School of Microelectronics, Tianjin University, Tianjin 300072, China;
2. Tianjin Key Laboratory of Imaging and Sensing Microelectronic Technology, Tianjin 300072, China)

Abstract: As the pre-order step of convolutional neural network (CNN) computing, image preprocessing is indispensable but time-consuming. To accelerate image preprocessing, a method based on RISC-V vector extension was proposed to accelerate eleven image preprocessing algorithms such as gray scale processing, standardization, and Gaussian filtering. Firstly, eleven image preprocessing algorithms were classified into four categories according to the computing mode, and acceleration schemes for the preprocessing algorithms were designed based on RISC-V vector extension. In order to further improve the performance, six customized vector instructions were added. The customized instructions were implemented by modifying the compiler and designing the hardware module. Finally, a field programmable gate array (FPGA) was used for testing, and the impact of vector processor configuration on performance and resource consumption was analyzed. The results showed that the proposed method achieves 3.13–9.97 times speedup compared with scalar processors, which effectively solves the performance bottleneck problem of image preprocessing in deep learning.

Keywords: convolutional neural network; preprocessing; RISC-V; vector extension; algorithm acceleration

Received: 2023-04-24; **Accepted**: 2023-05-26; **Published Online**: 2023-06-09 10:13

URL: link.cnki.net/urlid/11.2625.V.20230608.1155.002

Foundation item: National Natural Science Foundation of China (U21B2031)

* **Corresponding author**. E-mail: qiangliu@tju.edu.cn