计算机科学
COMPUTER SCIENCE

基于 ARM 的图像几何变换算法库实现和优化技术研究

王麓涵，贾海鹏，张云泉，张广婷

引用本文

王麓涵，贾海鹏，张云泉，张广婷. 基于 ARM 的图像几何变换算法库实现和优化技术研究[J]. 计算机科学, 2022, 49(10): 10-17.

WANG Lu-han, JIA Hai-peng, ZHANG Yun-quan, ZHANG Guang-ting. Study on Implementation and Optimization of ARM-based Image Geometric Transformation Library[J]. Computer Science, 2022, 49(10): 10-17.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220100128

基于 ARM 的图像几何变换算法库实现和优化技术研究

王麓涵[1,2]　贾海鹏[1]　张云泉[1]　张广婷[1]
1 中国科学院计算技术研究所计算机体系结构国家重点实验室　北京 100190
2 中国科学院大学计算机科学与技术学院　北京 100049
(iswangluhan@foxmail.com)

摘　要　高性能原语基础算法库(Intel © Integrated Performance Primitives,Intel IPP)是面向信号、图像处理领域的高性能多媒体加速库。然而,截至目前,暂时没有基于 ARM 架构的高性能 IPP 库。文中针对镜像变换、重映射、仿射、透视变换等基础图像几何变换算法,实现了一个基于 ARM 计算平台的高性能算法库 PerfIPP,并通过 SIMD 汇编优化、内存对齐、数据预计算、高性能矩阵转置等优化技术,显著提升了上述算法的性能。同时,通过对比不同指令组合、不同指令排列、不同取数存储方式等所带来的性能差异,总结图像几何变换算法在 ARM 计算平台上实现与优化的关键技术。实验结果表明,在华为鲲鹏 920 平台上,相比开源计算机视觉库 OpenCV,PerfIPP 在满足精度要求的同时,在上述基础图像几何变换上获得了 108.08%～435.5% 的性能提升,并达到了在英特尔至强 E5-2640 处理器上 Intel IPP 库平均性能的 83.79%。

关键词:IPP;ARM;NEON Intrinsic;几何变换;插值
中图法分类号　TP391

Study on Implementation and Optimization of ARM-based Image Geometric Transformation Library

WANG Lu-han[1,2],JIA Hai-peng[1],ZHANG Yun-quan[1] and ZHANG Guang-ting[1]
1 State Key Laboratory of Computer Architecture,Institute of Computing Technology,Chinese Academy of Sciences,Beijing 100190,China
2 School of Computer Science and Technology,University of Chinese Academy of Sciences,Beijing 100049,China

Abstract　Intel © integrated performance primitives is a high-performance multimedia acceleration library for signal and image processing. However,as of now,there is no high-performance IPP library based on the ARM architecture. This paper implements a high-performance algorithm library PerfIPP based on the ARM computing platform for basic image geometric transformation algorithms such as mirror,remap,and affine/perspective transformation. The PerfIPP,optimized through SIMD assembly,memory alignment,data pre-calculation,high-performance matrix optimization techniques,has significantly improved the performance of the above algorithms. At the same time,This paper summarizes the key technologies for the realization and optimization of image geometric transformation algorithms on the ARM computing platform by comparing the performance differences brought about by different instruction combinations,different instruction arrangements,and different access and storage methods. Experimental results show that,on the Huawei Kunpeng 920 platform,the PerfIPP proposed in this paper can achieve 108.08%～435.5% performance improvement in image transformation compared with the open source computer vision library while meeting accuracy. It also achieves 83.79% of the average performance of Intel IPP library on Intel Xeon E5-2640 processor.

Keywords　IPP,ARM,NEON Intrinsic,Geometry Transforms,Interpolation

Intel IPP 是拥有广泛多媒体功能的高性能接口集合[1]。IPP 提供的针对矩阵、向量计算、颜色转换和视觉计算的函数,目前已被广泛应用于生物医学、航空航天、数据通信、数字媒体等领域。在图像处理上,几乎所有的 IPP 函数都充分利用了 MMX,SSE 和 SSE2 等 Intel 处理器的指令集。单指令多数据技术使其具备了出色的高性能计算能力[2],实现了更高的程序执行效率。著名的开源计算机视觉库 OpenCV 可在 Intel 计算平台上,利用底层 IPP 库来获得非常显著的性能加速。不同版本的图像处理算法库在处理 512×512 图像规模的离散傅里叶变换、512×512 至 384×384 规模的图像放缩、

到稿日期:2022-01-14　返修日期:2022-04-28
基金项目:国家重点研发计划(2017YFB0202105);国家自然科学基金(61972376);北京市自然科学基金(L182053)
This work was supported by the National Key R&D Program of China(2017YFB0202105),National Natural Science Foundation of China(61972376) and Natural Science Foundation of Beijing,China(L182053).
通信作者:贾海鹏(jiahaipeng@ict.ac.cn)

三维空间的光流操作以及快速人工神经网络时的运行耗时如图 1 所示。该实验平台是 Pentium M，1.7 GHz。

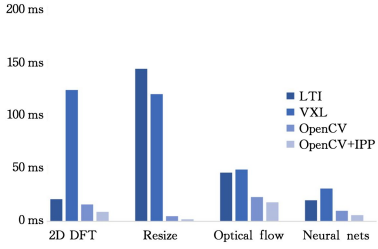

图 1 不同算法库在 4 种操作上的耗时

Fig. 1 Time consuming of different algorithm libraries

可以看出，IPP 优化的 OpenCV 相比其他 3 种主流图像处理算法库获得了最高的性能加速[3]。然而，作为商业版本的 Intel IPP 的劣势是仅能工作在 Intel 处理器上[4]，且其代码是闭源的。

随着 ARM 架构的发展，ARM 的应用越来越广泛[5]。针对 ARM 架构高性能 IPP 算法库的缺失问题，本文通过单指令多数据编程、算法调整、高性能矩阵转置、内存对齐、数据预计算等技术，实现了一个基于 ARM-v8[6-8] 架构的高性能 IPP 图像库 PerfIPP，该算法库在功能上将实现 Intel IPP 算法库的部分图像处理接口，并在性能上进行充分优化。

本文完成了镜像变换算法、重映射算法、仿射变换算法、透视算法在 ARM 架构上的实现和优化，并在精度上与 Intel IPP 的相关函数保持一致。通过对这些几何变换的高性能实现，本文提出并总结了 IPP 库在 ARM 架构上的关键优化技术。

本文第 1 节介绍了镜像函数的实现与优化，镜像是应用较为广泛的几何变换操作，Mirror 涉及大量的像素位置变换，在并行处理时各个点的相关性较强；第 2 节介绍了以重映射为基础的仿射透视变换，仿射变换、透视变换与重映射作为重要的几何变换算法，应用极为广泛，其在插值过程中包含大量的像素值计算，优化并分析这些几何变换算法的性能，对总结 IPP 的关键优化技术具有较高的参考价值；第 3 节将 PerfIPP 与 Intel IPP、开源计算机视觉库 OpenCV 等图像库的性能进行对比，并分析产生此性能的原因；最后总结全文。

1 镜像算法实现与优化

镜像变换仅涉及目标图像中像素位置间的调整，在进行 SIMD 优化时，一次取出的像素点在后续变换过程中的相关性较强。

1.1 算法原理

Mirror 算法实现了对图像的镜像变换，具体功能是将图像绕着指定的轴进行镜像翻转。IPP 算法库中该函数支持的类型有 unsigned char，unsigned short，short，int，float，分别支持通道为 C_1，C_3，C_4 以及 AC_4 的情况。在 IPP 库中使用该函数时，可以通过 *flip* 参数值的设定指定翻转轴。

如图 2 所示，当元素个数为偶数时，中间轴两侧的元素均要镜像翻转。

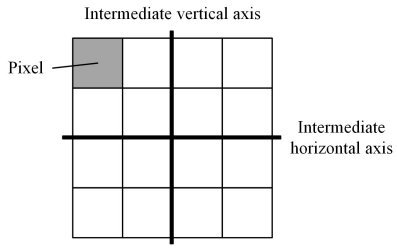

图 2 偶数元素镜像

Fig. 2 Mirror even elements

如图 3 所示，当元素个数为奇数时，中间轴所在的列（行）不动，该列（行）两侧的元素均要镜像翻转。

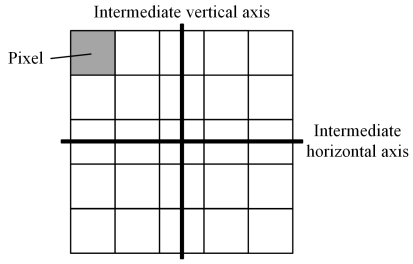

图 3 奇数元素镜像

Fig. 3 Mirror odd elements

PerfIPP 支持 5 种情况的镜像轴：中间水平轴翻转、中间竖直轴翻转、中间竖直轴翻转后沿着中间水平轴翻转。其中，当通道数量为 1 时，IPP 库还支持沿着 45°，135°轴翻转。

1.2 优化方法

使用 SIMD 指令的整体框架如图 4 所示。

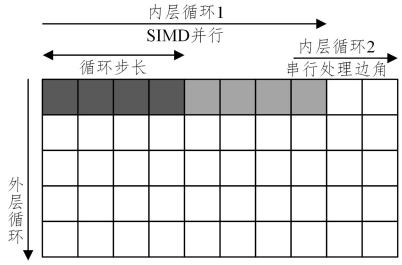

图 4 SIMD 优化框架

Fig. 4 SIMD optimization framework

外层循环按行扫描图片，对于当前行，内层设有两个循环，用于按列遍历。内层循环 1 使用 SIMD 指令，每次并行地处理 N 个值，若剩余不足 N 列，则由内层循环 2 逐个值处理。

1.2.1 45°与 135°镜像翻转——高性能转置

单通道图像沿着 45°轴进行的翻转操作等价于图像的转置。在上文利用 SIMD 加速时，是在一维空间中连续处理多个元素，而转置操作面向的是二维空间。可以根据不同的数据类型一次处理大小为 4×4 或 8×8 的矩阵。本文以 u8 数据类型单次循环转置大小为 8×8 的矩阵为例展开介绍。

本文对两层 for 循环做 8×8 展开，单次处理 8×8 大小的矩阵。vtrn 与 vzip 指令均可实现矩阵的转置。经测试，在转置大小为 4×4 的 f32 数据类型的矩阵时使用两条指令的耗时几乎没有差别。本文选择 vtrn 指令对大小为 8×8 的

矩阵进行转置操作，步骤如下。

(1)用 vtrn_u8 指令处理图像相邻的两行像素，如图 5 所示。

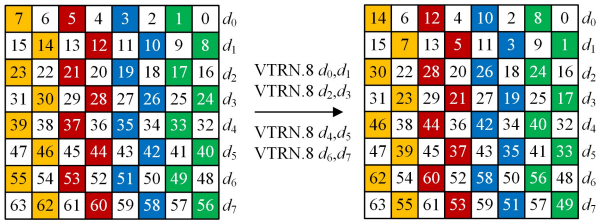

图 5 高性能转置大小为 8×8 的矩阵(1)

Fig. 5 High-performance transposed 8×8 matrix (1)

(2)vreinterpret_u16_u8 将寄存器 D 中的 8 bit×8 转化为 16 bit×4，用 vtrn_u16 处理第 1 行与第 3 行、第 2 行与第 4 行、第 5 行与第 7 行、第 6 行与第 8 行，如图 6 所示。

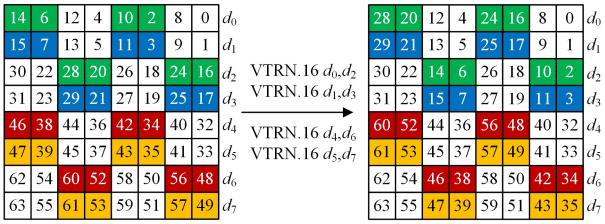

图 6 高性能转置大小为 8×8 的矩阵(2)

Fig. 6 High-performance transposed 8×8 matrix (2)

(3)vreinterpret_u32_u16 将寄存器 D 中的 16 bit×4 转化为 32 bit×2，用 vtrn_u32 处理第 1 行与第 5 行、第 2 行与第 6 行、第 3 行与第 7 行、第 4 行与第 8 行，如图 7 所示。

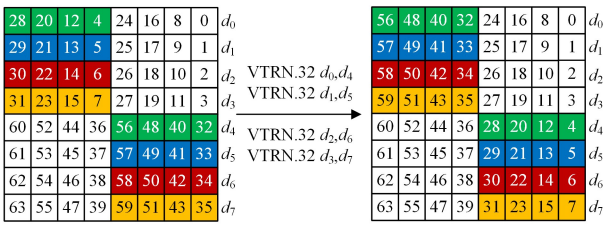

图 7 高性能转置大小为 8×8 的矩阵(3)

Fig. 7 High-performance transposed 8×8 matrix (3)

经过以上优化，在处理 45°，135°图像转置时，PerfIPP 相比基准代码性能获得了平均 545.5%的加速比。

1.2.2 竖直镜像翻转

在处理沿竖直轴或水平竖直轴同时翻转的情况时，需要将元素连续取出，而后在向量中将像素的顺序进行翻转(像素内部的通道顺序不变)，而后将结果存到目标位置。因此，在处理不同类型的通道与数据类型时便有不同的策略。这里可采用两种方法：vld3＋vrev＋vst3 的指令组合与 vld1＋vtbl＋vst1 的指令组合。

(1)指令选择：vld3＋vrev＋vst3

用 vld3 指令交织地取出 3 个向量(每个通道对应一个向量)后，在单通道情况下对每个向量进行相同的处理，再用 vst3 将 3 个向量交织地存入目标矩阵处，如图 8 所示。

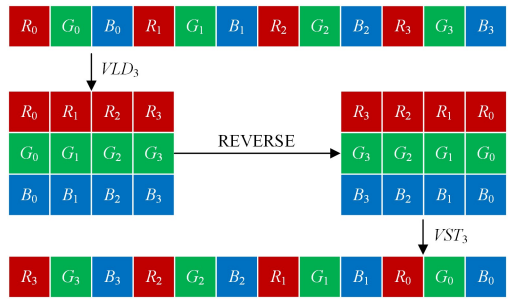

图 8 处理三通道竖直镜像指令组合 a

Fig. 8 Command combination for processing three-channel vertical mirroring a

(2)指令选择：vld1＋vtbl＋vst1

除了以上方法，处理三通道时也可用 vld1q 取出和 vst1q 存入。对于 float 数据类型而言，它一次可取出 4 个通道并将其放到一个向量中，这个向量中必然有不同像素的元素，因此需采用查表指令 tbl。

查表指令可建立索引，为 32×4×3 这 12 个元素分别加上索引，将它们的位置倒转存入目标位置。由于每个索引只有 8 bit 且只能为 8 bit 的数据找到位置，因此用 vreinterpretq 指令将大小为 32×4 的矩阵转为大小为 8×16 的矩阵来进行索引查表操作，效果如图 9 所示。

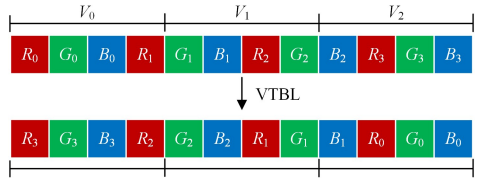

图 9 处理三通道竖直镜像指令组合 b

Fig. 9 Command combination for processing three-channel vertical mirroring b

以上两种指令组合性能的对比如图 10 所示。其中，数值部分为当前指令组合迭代 60 次的耗时(ms)。

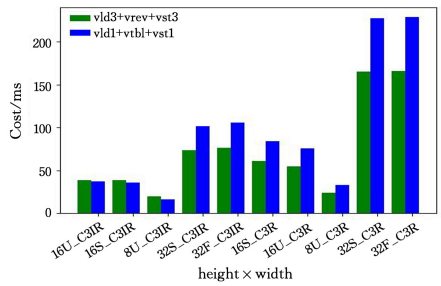

图 10 三通道竖直镜像两种指令组合耗时(ms)对比

Fig. 10 Comparison of the time consuming (ms) of two command combinations of three-channel vertical mirroring

这两种指令组合在处理三通道不同数据类型图像时的表现是不同的。本文在处理不同类型的三通道数据时，优先选择耗时较低的指令组合，因此在处理 32f_C3R，32s_C3R 等数据类型时采用 vld3＋vrev 翻转指令的组合，在处理 8u_C3IR，16s_C3IR，16uC3IR 数据类型时采用 vld1＋vtbl 查表法的指令组合。针对不同接口，采用性能最高的指令组合，PerfIPP

相对于-O3 编译优化的基准代码获得了 206.4%的平均性能加速。

2 仿射、透视、重映射变换算法的实现与优化

仿射变换与透视变换使图像变形并映射到目标图像。二者在图像中的应用极为广泛，如 OCR 文字识别操作的第一步——歪斜图片矫正。重映射是几何变换中的基础操作，本文实现了高性能的重映射函数，并在仿射透视变换中的映射操作中调用重映射以完成源图像到变形目标图像的高性能映射。实际上，为了减少变形过程中的锯齿状失真，本文的重映射操作是以相反的顺序完成的，即从目标图像到源图像的顺序。

2.1 重映射原理

重映射从源图像中某个位置提取像素并将该像素定位到目的图像中的另一个位置，如图 11 所示。

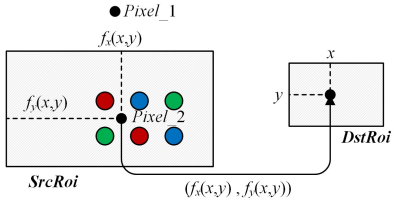

图 11 remap 重映射原理

Fig. 11 Principle of remap

在映射时存在两个主要问题：

(1)通常情况下，$f_x(x,y)$ 和 $f_y(x,y)$ 是浮点数，这意味着变换〈f_x，f_y〉可以是仿射、透视变换，或径向畸变校正等，因此需要检索分数坐标处的像素值。在最简单的情况下，坐标可以四舍五入到最接近的整数坐标，并且可以使用相应的像素，这称为最近邻插值。使用更复杂的插值方法可以获得更好的结果，如本文实现的 *Linear*，*Cubic*，*Catmull*，*Lanczos*4 分别从当前浮点坐标周围取 2×2，4×4，4×4，6×6 数量的像素点，然后进行加权计算，得到目标点的数值。它们引入了更多的计算与访存操作来解决图像的锯齿和失真问题。

(2)源图像外不存在像素的外推。对于某些 (x,y)，$f_x(x,y)$，$f_y(x,y)$ 或两者都可能落在图像之外。在这种情况下，需要使用外推法，如常数边界、复制边界、混合边界、透明边界等。

在算法库的正确性上，每个算法库中同一插值方式的具体算法的实现过程有细微的差别，这导致了运算结果的不同。在 PerfIPP 中，本文在每个接口中都实现了最近邻插值、线性插值、三次样条插值与兰索斯插值的选项，在精确度测试中与 Intel IPP 保持一致。下文对 PerfIPP 插值算法的描述中使用的符号如表 1 所列。

表 1 插值符号及其含义

Table 1 Interpolation symbols and their meanings

符号	含义
(x_D,y_D)	目的图像的像素坐标(整数值)
(x_S,y_S)	将要被映射到(x_D,y_D)的源图像的像素点的坐标
$S(x,y)$	源图像坐标为(x,y)像素点的像素值
$D(x,y)$	目的图像坐标为(x,y)像素点的像素值

以线性插值为例，该插值方法综合了该小数坐标周围的 4 个像素值的信息，如图 12 所示。

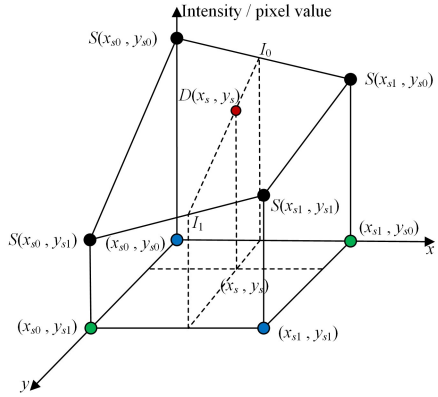

图 12 线性插值原理

Fig. 12 Principle of linear interpolation

4 个整数坐标(x_{S0},y_{S0})，(x_{S0},y_{S1})，(x_{S1},y_{S0})，(x_{S1},y_{S1})上分别有 4 个像素点，分别确定了 4 个像素值 $S(x_{S0},y_{S0})$，$S(x_{S0},y_{S1})$，$S(x_{S1},y_{S0})$，$S(x_{S1},y_{S1})$。由于像素点的坐标均为整数，因此对于小数坐标(x_S,y_S)，其坐标点的强度 $D(x_S,y_S)$需要由周围 4 个像素值经过特定的运算确定。

2.2 仿射与透视算法原理

仿射与透视变换又可以分别称为二维坐标变换与三维坐标变换。如图 13 所示，在图片文字识别的预处理时，对歪斜图片进行平铺操作使用的便是仿射或透视变换。

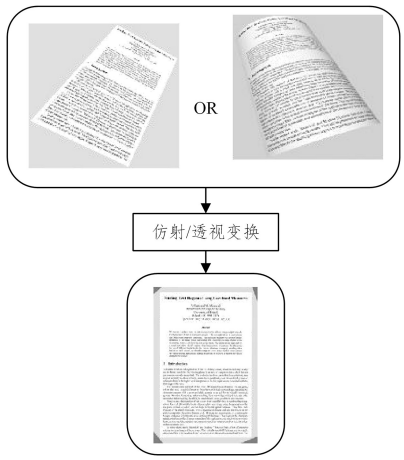

图 13 OCR 操作图片预处理

Fig. 13 Image preprocessing for OCR operation

仿射变换保持了二维图形的平直性与平行性，表达式如下[9]：

$$M=[A\ B]=\begin{bmatrix}a_{00} & a_{01} & b_{00}\\ a_{10} & a_{11} & b_{10}\end{bmatrix}_{2\times 3}$$

$$\boldsymbol{T}=\boldsymbol{M}\cdot[x,y,1]^{\mathrm{T}}$$

其中，$\boldsymbol{M}$ 表示变换矩阵，包含线性变换矩阵 $\boldsymbol{a}$ 与平移矩阵 $\boldsymbol{b}$；x 代表像素的横坐标；y 代表像素的纵坐标；$\boldsymbol{T}$ 表示仿射变换后的图像矩阵。仿射变换可以做图像的旋转操作，表达式如下：

$$M=\begin{bmatrix}\alpha & \beta & (1-\alpha)\cdot center.x-\beta\cdot center.y\\ -\beta & \alpha & \beta\cdot center.x+(1-\alpha)\cdot center.y\end{bmatrix},$$

$$\alpha=scale\cdot\cos\theta,\beta=scale\cdot\sin\theta$$

其中，*center* 代表旋转中心点，θ 代表旋转角度，*scale* 代表缩放比例。

透视变换相比仿射变换可改变二维图像的平行性，变换公式如下[10]：

$$[x',y',w']=[u,v,w]\cdot\begin{bmatrix}a_{11}&a_{12}&a_{13}\\a_{21}&a_{22}&a_{23}\\a_{31}&a_{32}&a_{33}\end{bmatrix}$$

$$x=\frac{x'}{w'},y=\frac{y'}{w'}$$

其中，w 通常为 1，$[a_{11}\ \ a_{12}\ \ a_{21}\ \ a_{22}]$ 为线性变换矩阵，$[a_{31}\ \ a_{32}]$ 为平移矩阵，二者等价于上文的仿射变换矩阵，$[a_{13}\ \ a_{23}]$ 产生透视变换；u,v 代表原始图片坐标；x,y 代表透视变换后图片的坐标，w 通常为 1。

2.3 优化方法

2.3.1 访存优化

以重映射变换的 16U 数据类型的线性插值方法为例，若要实现一次循环处理 4 个数据，则首先要从 *xMap* 与 *yMap* 数组中一次取得 4 个数据。

此时，4 个 *yMap* 确定了 4 个纵坐标，4 个 *xMap* 确定了 4 个横坐标，二者组成了 4 个浮点数坐标，然后根据这 4 个浮点数坐标从源图像中取出并计算 4 个坐标点对应的值，并将其放入目的图像中。线性插值取坐标的过程如图 14 所示。

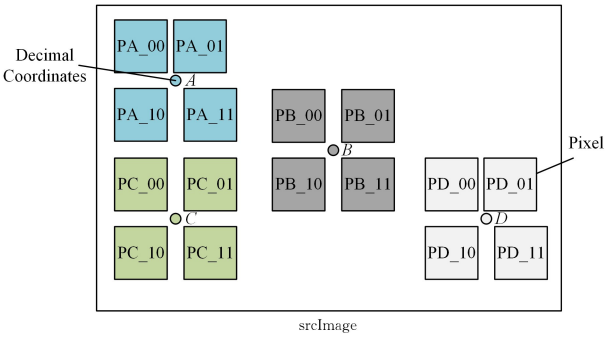

图 14 线性插值取坐标

Fig. 14 Take coordinates when linear interpolation

此时，确定 4 个分数坐标点 A,B,C,D，在线性插值中，它们位置上的值由其周围的 4 个像素值决定，因此需要在取数阶段取出 16 个值存入向量寄存器中。16 个值在向量寄存器中的排布可以采用两种方法，如图 15 所示。

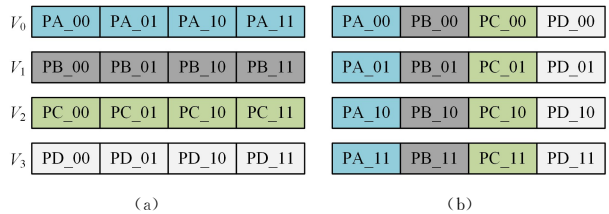

图 15 16 个值在向量寄存器中的两种排布方法

Fig. 15 Two ways of arranging 16 values in vector registers

在图 15(b)中，将位于 4 个坐标点左上位置的 4 个像素、右上位置的 4 个像素、左下位置的 4 个像素和右下位置的 4 个像素，分别存放于 4 个寄存器 V_0— V_3 中。由 2.1 节对线性插值的介绍可知，每个位置对应的权重计算方法相同，如对于 00 号位置而言，可以同时算出 4 个点在该位置的权重，如图 16 所示。

图 16 位置 00 处的权重寄存器

Fig. 16 Weight register at position 00

图 15(a)显示了同一个坐标点周围的 4 个像素被放在一个寄存器中。因此，需要通过 zip 指令将寄存器内的权重值进行重新排列，以适应当前的像素顺序，如图 17 所示。

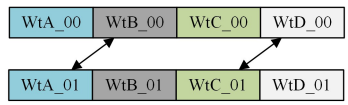

图 17 zip 指令重排权重寄存器

Fig. 17 Rearrange weight register with zip instruction

在对整数数据类型进行优化时，由于整数数据开始存放于通用寄存器，将其放入向量寄存器进行 simd 优化时涉及通用寄存器到向量寄存器的拷贝，该过程较为耗时。此时，虽然多了 zip 指令，但由于 a 方法可将左上、右上、左下、右下的像素点作为一个 32u 取出到标量寄存器中，因此标量寄存器赋值到向量寄存器只需赋值两次，而 b 方法需要赋值 4 次，二者在 256×256～1 920×1 080 像素规模的图片下的耗时如图 18 所示。在线性插值时，本文均采用较快的寄存器排布方式，即方式 a。

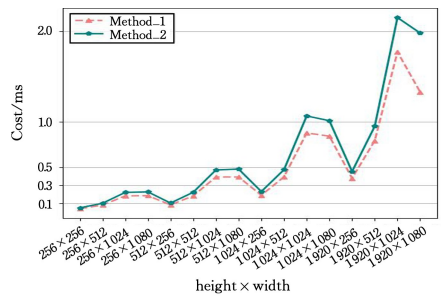

图 18 不同图像规模下两种取数方式的耗时

Fig. 18 Time cost for two access methods

2.3.2 计算优化

在兰索斯插值与 3 次样条插值中均使用预计算。兰索斯插值算法中涉及三角函数、多项式的计算，而无精度损失的三角函数计算量非常大，很容易成为算法库的性能瓶颈。该计算结果只与待插值点坐标的小数部分有关，可以预先生成插值权重因子并将其存储在内存中，在执行的过程中，算法库根据索引直接查找的方式来提高插值算法的性能，避免了权重因子的计算过程，以空间资源换取时间资源。

3 实验结果与分析

3.1 实验环境搭建

本文的 ARM 测试平台采用的处理器是鲲鹏 920，对照性能测试的 X86 平台采用的处理器是 Intel Xeon E5-2640，二者主频均为 2.6GHz，本文实验环境配置如表 2、表 3 所列。

表 2 ARM 实验环境

Table 2 ARM experimental environment

处理器	Kunpeng920
Architecture	aarch64
L1 cache/Byte	64 000
L2 cache/Byte	512 000
CacheLine/Byte	64
向量寄存器长度/bit	128
操作系统	Red Hat 4.8.5-36
测试工具	Google Test
编译工具	CMake
链接库	OpenCV

表 3 X86 对比实验环境

Table 3 X86 experimental environment

处理器	Intel Xeon E5-2640
Architecture	x86-64
L1 cache/Byte	64 000
L2 cache/Byte	256 000
CacheLine/Byte	64
向量寄存器长度/bit	256
操作系统	CentOS release 6.3
测试工具	Google Test
编译工具	CMake
链接库	IntelIPP

Intel IPP 运行的硬件架构为 Intel© Haswell[11-15] 微体系结构，该微架构支持 Intel AVX2[16-17] 与 Intel © TSX[18-19]。对比二者的向量寄存器最大长度，X86 平台是 ARM 平台的一倍，该参数将会对 SIMD 编程的性能产生影响。

3.2 不同策略下的性能

在 2.3 节中，重映射的两种访存方式的性能如图 18 所示。图 18 中，纵坐标表示当前访存方式的耗时，单位为 ms。

由图 18 可知，方式 a 的耗时 Method_1 更低，因此在线性插值的 16u 数据类型下采用方式 a 的寄存器排布方式。

3.3 Mirror 性能分析

在测试中，对于每一个接口，均测试了规模为 256×256 到 1 920×1 080 的不同图像。为了便于性能分析，以 1 080p (1 920×1 080 像素)的图像为例来说明 mirror 的性能。数据类型采用 unsigned char(8u)，即目前大量应用的光学三原色 RGB 模式，像素的取值是 0～255。

Mirror 的性能分析结果如图 19、图 20 所示。

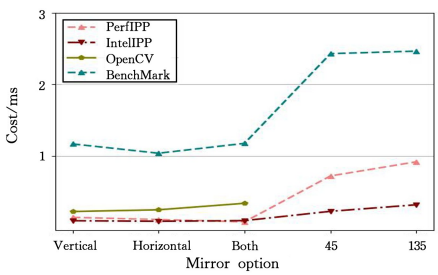

图 19 单通道镜像 1080p 图像耗时对比

Fig. 19 Comparison of time cost of single-channel mirroring 1080p images

图 19 给出了单通道情况，图 20 给出了 3 通道、4 通道的镜像变换过程。横坐标代表了不同的镜像轴，非单通道的情况支持 3 种基本的镜像轴，单通道下除了 3 种基本镜像轴，还支持 45°与 135°镜像轴，基准代码 Benchmark 为-O3 编译选项优化的 C 语言实现代码。其中，OpenCV 的镜像函数不支持 45°与 135°的转置选项。

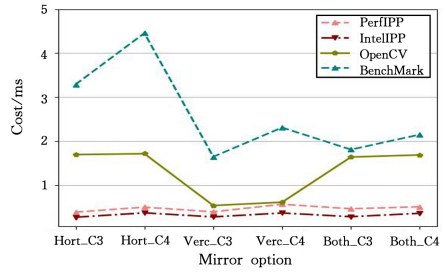

图 20 多通道镜像 1080p 图像耗时对比

Fig. 20 Comparison of time cost of multi-channel mirroring 1080p images

由图 19、图 20 可知，PerfIPP 在各情况下均稳定在较低耗时。在 PerfIPP 中，沿竖直轴镜像操作与水平轴相比多了翻转指令，因此耗时相对较长。图 19 中，由于 45°与 135°翻转涉及转置，其耗时明显长于其他翻转，大约为 Intel IPP 性能的 40%左右。

以 ippiMirror_32f_C1R 为例：做图像的转置操作时(45°)，PerfIPP 库在 ARM 上的绝对性能平均值约为 Intel IPP 库在 X86 上的绝对性能平均值的 40.6%。其产生性能差异的原因如下：在测试案例中，给出的图像规模为 256×256 像素，在 ARM 平台上实现转置时，一次循环可用 6 条 *vzip* 转置指令处理 4×4 的矩阵；由于 Intel 平台的向量寄存器长度可达到 ARM 平台的两倍，即 256 bit，IntelIPP 在处理过程中利用 256 bit 的 *ymm* 寄存器与 128 bit 的 *xmm* 寄存器的配合，使用了相隔两个元素交织的 *vunpcklps*，*vunpckhps* 指令、将两个 128 bit 向量合并的 *vinsertf*128 指令等，共计 24 条转置指令，每一次循环转置了规模为 16×8 的矩阵。

如此看来，在处理转置的平均指令条数上，同样是处理规模为 16×8 的矩阵，X86 的指令条数是 NEON 指令的一半。而根据 Intel 平台的向量寄存器长度，IntelIPP 一次可存 8 个像素，而在 ARM 平台上我们利用 128bit 的寄存器一次仅能存 4 个像素。综合计算指令及访存指令，因此 PerfIPP 的转置变换仅能达到 IntelIPP 的 40.6%。

3.4 重映射、仿射、透视性能分析

性能测试的实验硬件平台与软件环境请见 3.1 节。图 21、图 22 给出了 Remap 中计算较为复杂的 LINEAR 与 CUBIC 插值算法的性能比对结果，实验的图像通道数为 3，像素数据类型为 unsigned short 类型。

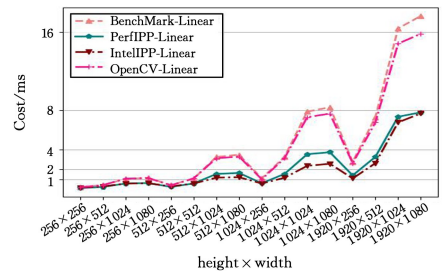

图 21 重映射-线性插值耗时对比

Fig. 21 Comparison of time cost of remap-linear

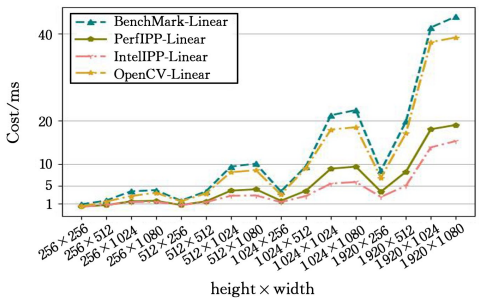

图 22 重映射-三次插值耗时对比

Fig. 22 Comparison of time cost of remap-cubic

图 21、图 22 中，横坐标代表各个图像的规模，本文测试了从 256×256 到 1920×1080 的图像规模，纵坐标表示当前函数执行一次的平均消时。基准代码 Benchmark 为-O3 编译选项优化的 C 语言实现代码。

可以看出，PerfIPP 的性能均高于开源 OpenCV 的性能，可以在 LINEAR 插值方式下达到 Intel IPP 性能的 66.59%～125.51%，在 CUBIC 插值下达到 IPP 性能的 61.33%～98.04%。在 1920×1024 像素的图像规模时，三插值算法耗时均有明显的正向突变。其原因在于，此时单幅图大小超过 cache 大小，cache 命中率下降导致性能大幅度降低。

在开源计算机视觉库中，仿射变换与透视变换的映射操作调用了算法库的重映射函数，PerfIPP 也采用此方法调用自己的 Remap。对最近邻插值的 unsigned char 数据类型、单通道进行仿射变换与透视变换实验，仿射变换性能对比如图 23 所示。

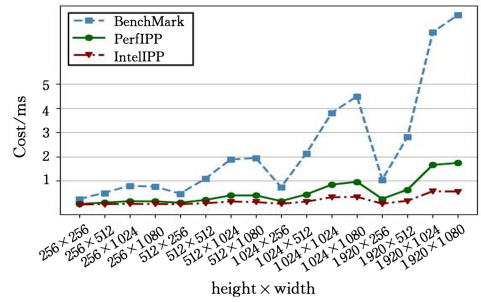

图 23 仿射变换耗时对比

Fig. 23 Comparison of time cost of affine transformation

透视变换的性能如图 24 所示。

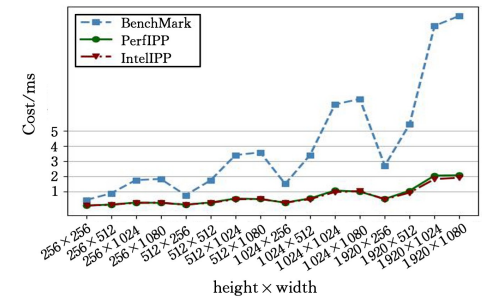

图 24 透视变换耗时对比

Fig. 24 Comparison of time cost of perspective transformation

可以看出，透视变换时，PerfIPP 的性能与 IntelIPP 基本保持接近。

由于 ARM 平台向量寄存器的长度为 X86 平台的一半，因此其理论性能为 Intel IPP 的 50%左右。本文实现的接口达到了英特尔至强 E5-2640 处理器上 Intel IPP 库平均性能的 83.79%。

结束语 高性能原语基础(IPP)算法库作为底层加速库，在处理器的软件生态中占据着十分重要的地位。本文实现并优化了一个面向华为鲲鹏 920 处理器的 IPP 库——PerfIPP。在本文的几何变换模块中，PerfIPP 相比 OpenCV 性能提升了 1.08～4.35 倍。本文总结的面向 ARM 计算平台 IPP 算法库的实现和优化关键技术主要包括：

(1)迭代间 SIMD 的使用，以及在取数时尽量减少通用寄存器到向量寄存器的拷贝次数。

(2)计算时大多数情况下尽量采用乘加指令、部分有_high 下标的混合指令来减少计算指令条数。

(3)在相同功能下采用较快指令：位操作指令 vand 的速度明显快于 vmin；左移操作应选择向量寄存器内的左移指令 vshl 来代替通用寄存器内的左移 vshl_n；尽量用性能更高的 vld1 指令替代 vldn 指令；利用 vzip 和 vtsrn 指令进行高效矩阵转置；在三通道不同数据类型时，vrev＋vldn/vstn 指令组合与 vtbl＋vldn/vst1 指令组合各有优势，PerfIPP 在各数据类型下均选择了最优的指令组合。

(4)将预计算的值放入内存空间并建立索引，减少程序运行时的计算量。

参 考 文 献

[1] TENG S H, WANG F, ZHAO Z S, et al. Application of Intel IPP to comprehensive experiments for digital image processing [J]. Laboratory Science, 2016, 19(5): 76-79.

[2] LI J, WEI J, SHI J H. Design of MPEG-4 Video Transmission System based on IPP library [J]. Microcomputer Information, 2008, 24(11): 16-17, 33.

[3] DEVIRANGALAKSHMI A, INABITHINI S R, VENKATARAMANA P. Realization of signal processing algorithms using Intel integrated performance primitives(IPP)[C]// 2017 International Conference on Innovations in Information, Embedded and Communication Systems(ICIIECS). IEEE, 2017: 1-4.

[4] OME LANDR J R. Programming with intel ipp (integrated performance primitives) and intel opencv (open computer vision) under gnu linux[EB/OL]. http://www4.comp.polyu.edu.hk/~csajaykr/myhome/teaching/biometrics/ippocv.pdf.

[5] CHEN T, LI Z H, JIA H P, et al. Implementation and optimization of multi-dimensional FFT based on ARMv8 platform [J]. Chinese Journal of Computers, 2019, 42(11): 2384-2402.

[6] STEPHENS N. ARMv8-A next-generation vector architecture for HPC[C]// 2016 IEEE Hot Chips 28 Symposium(HCS). IEEE, 2016: 1-31.

[7] GRISENTHWAITE R. Armv8 technology preview[C]// IEEE Conference. 2011.

[8] FLUR S, GRAY K E, PULTE C, et al. Modelling the ARMv8 architecture, operationally: concurrency and ISA[C]// Procee-

dings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. 2016:608-621.

[9] GUAN Y R,GUAN Y Q. Research and application of affine transform based on OpenCV [J]. Computer Technology and Development,2016,26(12):58-63.

[10] ROBERT C,ASEEM A,MANEESH A. Image warps for artistic perspective manipulation[J]. ACM Transactions on Graphics,2010,29(4CD):127.1-127.9.

[11] HAMMAR L P,MARTINEZ A J,BAJWA A A,et al. Haswell: The fourth-generation intel core processor[J]. IEEE Micro,2014,34(2):6-20.

[12] KURD N,CHOWDHURY M,BURTON E,et al. Haswell: A family of IA 22 nm processors[J]. IEEE Journal of Solid-State Circuits,2014,50(1):49-58.

[13] HACKENBERG D,SCHÖNE R,ILSCHE T,et al. An energy efficiency feature survey of the intel haswell processor[C]// 2015 IEEE International Parallel and Distributed Processing Symposium Workshop. IEEE,2015:896-904.

[14] MOLKA D,HACKENBERG D,SCHÖNE R,et al. Cache coherence protocol and memory performance of the intel haswell-ep architecture[C]// 2015 44th International Conference on Parallel Processing. IEEE,2015:739-748.

[15] KANTER D. Intel's haswell cpu microarchitecture[J/OL]. Real World Technologies,2012. https://scholar.google.co.kr/citations?view_op=view_citation&hl=vi&user=jLyty0sAAAAJ&citation_for_view=jLyty0sAAAAJ:ufrVoPGSRksC.

[16] WATANABE H,NAKAGAWA K M. SIMD vectorization for the Lennard-Jones potential with AVX2 and AVX-512 instructions[J]. Computer Physics Communications,2019,237:1-7.

[17] HAMMARLUND P,MARTINEZ A J,BAJWA A,et al. 4th generation Intel core processor,codenamed haswell[C]// Hot chips. 2013.

[18] JEONG S,YANG S,BURGSTALLER B. Lock Elision for Protected Objects Using Intel Transactional Synchronization Extensions[C]// Ada-Europe International Conference on Reliable Software Technologies. Cham:Springer,2017:121-136.

[19] OLEKSENKO O,KUVAISKII D,BHATOTIA P,et al. Efficient Fault Tolerance using Intel MPX and TSX[C]// 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks. 2016.

WANG Lu-han, born in 1999, postgraduate. His main research interests include high performance computing and parallel software, etc.

JIA Hai-peng, born in 1983, Ph.D. His main research interests include high performance computing, many-core programming method and key optimization technologies for many-core platforms.

(责任编辑:喻藜)