

基于嵌入式 CPU+GPU 异构平台的 遥感图像滤波加速*

谭鹏源^{1,2} 薛长斌¹ 周莉¹

1(中国科学院国家空间科学中心 北京 100190)

2(中国科学院大学 北京 100049)

摘要 针对遥感图像在轨实时处理提出一种基于嵌入式 CPU + GPU 异构平台的遥感图像滤波加速设计方法. 以加速拉普拉斯滤波为例, 利用 GPU 的并行计算特点, 通过数据划分及数据映射的方法对算法进行并行设计; 利用 GPU 的向量单元和缓存等硬件资源, 通过采取向量化和向量重组以及工作组调优方法进一步提高了算法的运行速度. 在嵌入式开发板上验证了加速设计的可行性和高效性. 实验结果表明, 相比于单 CPU 的串行实现, 在增加 GPU 并行处理后的拉普拉斯滤波获得了 4.08~16.92 倍的加速比. 进一步利用 GPU 硬件资源优化性能后, 加速比可达 15.38~56.41 倍.

关键词 嵌入式 GPU, 遥感图像滤波, OpenCL, 向量化, 向量重组

中图分类号 V19, TP391

Acceleration of Remote Sensing Image Filtering Based on Embedded CPU+GPU Heterogeneous Platform

TAN Pengyuan^{1,2} XUE Changbin¹ ZHOU Li¹

1(National Space Science Center, Chinese Academy of Sciences, Beijing 100190)

2(University of Chinese Academy of Sciences, Beijing 100049)

Abstract A method is proposed for accelerating remote sensing image filtering in real-time using an embedded CPU + GPU heterogeneous platform for satellite-based image processing. the algorithm was initially parallelized through data division and mapping, leveraging the parallel computing capabilities of the GPU. Subsequently, hardware resources like the vector unit and cache of the GPU were employed to enhance algorithm speed through vectorization, vector permutation, and workgroup tuning. The feasibility and efficiency of this accelerated design were validated on an embedded development board. The experiments demonstrate a speedup ranging from 4.08 to 16.92 times when incorporating GPU parallel processing, compared to the serial implementation on a single CPU. Further optimization using GPU hard-

* 中国科学院国防科技重点实验室基金项目资助 (CXJJ-20S017)

2023-03-02 收到原稿, 2023-04-26 收到修定稿

ware resources can push the speedup to 15.38 to 56.41 times.

Key words Embedded GPU, Remote sensing image filtering, OpenCL, Vectorization, Vector permutation

0 引言

随着空间探测领域的扩大,空间探测任务对航天器的实时数据处理能力要求越来越高.目前,在轨处理以 CPU, DSP, FPGA 实现为主,随着抗辐照芯片设计技术的成熟,将地面上成熟的嵌入式 ARM CPU+GPU 异构处理器进行抗辐照加固应用于航天,有助于满足空间任务对处理器的性能需求,并降低工程开发的难度.

在遥感图像处理中,图像滤波发挥十分重要的作用,可以对遥感图像进行去噪、模糊、锐化等多种操作,是图像增强、图像识别、特征提取、图像分割等后续工作步骤的基础.常见的图像滤波算法有均值滤波、高斯滤波、中值滤波和拉普拉斯滤波等^[1].随着遥感图像分辨率和帧频不断提高,图像数据量越来越大.因为图像滤波处理需要对输入图像中的每个像素及其邻域内的像素进行处理,所以当数据量庞大时,在轨计算过程将非常耗时.在地面,采用 GPU 处理图像滤波可以达到很高性能.随着抗辐照嵌入式 GPU 的不断发展,采用嵌入式 CPU+GPU 异构平台提升航天器在轨处理性能已成为目前的重要方向^[2].因此,基于嵌入式 CPU+GPU 异构平台利用嵌入式 GPU 对遥感图像滤波并行加速,已成为未来空间应用的趋势.

目前,对图像滤波的硬件加速方法预先研究不断深入. Xiao 等^[3]基于 NVIDIA GPU 设计并实现了一种图像加权均值滤波 OpenCL(Open Computing Language)并行算法,并通过使用常量内存、局部内存、工作组调优、零拷贝等方法进一步优化算法性能,最终取得了 20.88 倍的加速比. Xiao 等^[4]基于 NVIDIA GPU 提出了一种 Sobel 滤波 OpenCL 并行算法,并采用多级存储和向量访问两种技术优化了数据存储结构,最终取得了 9.55 倍的加速比.然而,以上研究均基于桌面 GPU,桌面 GPU 功耗大,无法应用于功耗受限的星上平台. Pang 等^[5]在 FPGA 上通过并行处理和流水线技术提高了中值滤波的运行速度,在 XC7 K325 T 平台上处理尺寸为 4096×4096 的遥感

图像只耗费了 40 ms,能够满足星上实时处理的要求.然而,与嵌入式 GPU 相比, FPGA 存在开发门槛高、浮点运算能力低等问题.

以往的研究主要集中在桌面 GPU 和 FPGA 等平台上进行图像滤波加速,而针对嵌入式 GPU 的加速研究较少.由于嵌入式 GPU 计算资源有限,并行计算及其优化技术是实现高性能的关键.因此,本文根据嵌入式 GPU 硬件架构和并行编程框架 OpenCL,以拉普拉斯滤波为例说明具体的并行设计和优化方法,并在硬件平台上进行了验证,然后推广到其他常用的图像滤波算法上.

1 GPU 硬件架构与并行编程框架

基于地面成熟的嵌入式 ARM CPU+GPU 异构多核处理器,与目前国外抗辐照设备架构类似,具体方法也可以适用于中国即将推出的抗辐照处理器. GPU 采用 Midgard 架构的 Mali GPU,如图 1 所示. Midgard 架构的 GPU 采用统一渲染核架构,着色器核心(Shader Core)可扩展至 16 个,每个核可运行多达 256 个线程,所有渲染核共用 1 个二级缓存.在每个核心中包含 1 个 Load/Store 流水线、2 个算术流水线(Arithmetic Pipeline).数据 Load/Store 流水线用于对内存数据进行读写操作,单位时间内可以读写 128 bit 的数据.其还拥有 1 个 16 kByte 的一级缓存,缓存行大小为 64 Byte.算术流水线基于单指令多数据(Single Instruction Multiple Data, SIMD),单位时间内可同时对向量寄存器中的多个数据进行处理.向量寄存器的长度为 128 bit,因此 SIMD 的向量长度最大为 128 bit^[6].

采用 OpenCL 进行并行编程设计. OpenCL 是一种跨平台的并行编程框架,用于编写并行计算应用程序. OpenCL 程序由主机程序(Host program)和内核(Kernel)组成.主机程序在 CPU 上执行,负责为内核定义上下文并管理内核的执行,内核则在 GPU 上执行具体的计算任务^[7].

工作项(work item)是内核的实例,为 Open-

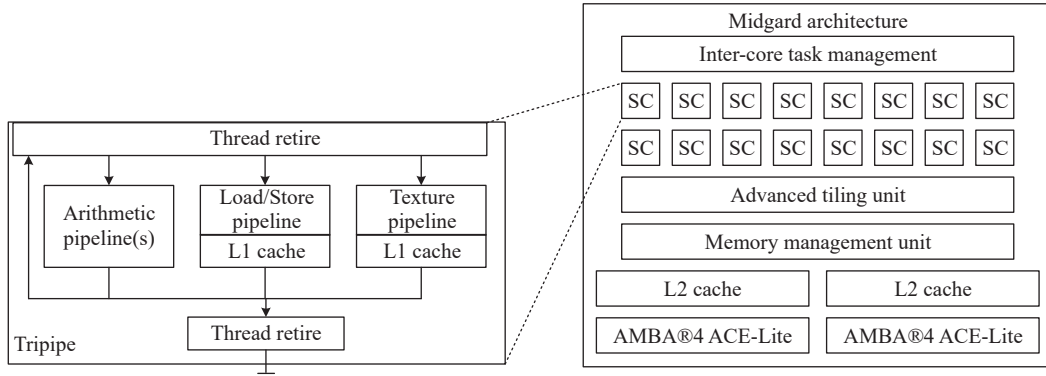


图 1 Mali GPU Midgard 架构

Fig. 1 Mali GPU Midgard architecture

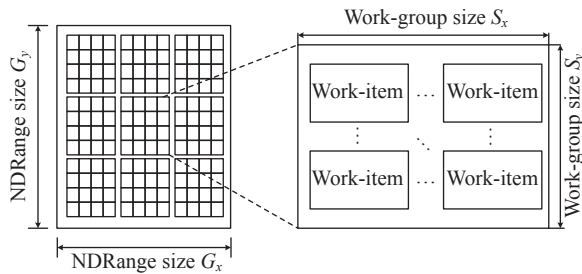


图 2 二维索引空间

Fig. 2 Two-dimensional NDRange

CL 中执行计算任务的最小单元. 若干工作项构成工作组 (work group). 索引空间 (NDRange) 是工作项的集合, 描述了工作项的坐标范围, 在内核中用于对数据进行寻址. 图 2 描述了在二维索引空间中工作项、工作组和索引空间三者之间的关系, 工作组是对索引空间的粗粒度分解, 工作项是对索引空间的细粒度分解.

2 图像滤波并行设计与优化

图像滤波可分为空间域滤波和频率域滤波两种方法. 空间域滤波通过窗口或卷积核进行, 通过参照相邻像素改变中心像素的灰度值, 适用于滤波窗口较小的场合. 频率域滤波是对图像进行傅里叶变换, 再对变换后的频率域图像中的频谱进行滤波, 适用于滤波窗口较大的场合^[1]. 在实际使用中, 一般滤波所采用的窗口尺寸较小, 因此本文采用空间域滤波的方法实现图像滤波. 图像滤波均为邻域操作, 其加速方法具有通用性, 因此为了便于分析, 仅以拉普拉斯滤波为例, 说明具体的并行设计和优化方法, 后续可将方

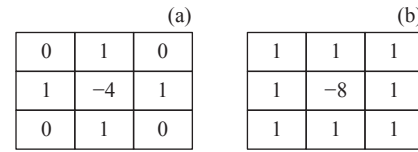


图 3 两种常见的拉普拉斯模板

Fig. 3 Two common types of Laplacian templates

法推广到其他常见图像滤波的实现上.

2.1 拉普拉斯滤波

拉普拉斯滤波是遥感图像锐化的基本工具, 可增强原始图像中灰度突变区域的对比度, 使得图像的细节比原始图像更加清晰. 其理论依据是拉普拉斯算子, 定义如下:

$$\nabla^2 f(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \quad (1)$$

在数字图像中, 任意阶的微分表现为差分的形式, 由此可得如下离散二维空间上的拉普拉斯算子:

$$\nabla^2 f(x, y) = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y). \quad (2)$$

式 (2) 的应用一般会采用滤波模板来实现. 如图 3(a) 所示, 图中模板系数分别为式 (2) 的各项系数. 例如模板中心系数为 $f(x, y)$ 的系数 -4, 模板中心左边系数为 $f(x-1, y)$ 的系数 1, 以此类推. 另外, 还有一种常见的模板, 在保证各向同性的同时, 还将对角线上的像素加入到模板中^[8], 则式 (2) 可扩展为如下形式:

$$\begin{aligned} \nabla^2 f(x, y) = & f(x-1, y-1) + f(x, y-1) + \\ & f(x+1, y-1) + f(x-1, y) - 8f(x, y) + \\ & f(x+1, y) + f(x-1, y+1) + f(x, y+1) + \\ & f(x+1, y+1). \end{aligned} \quad (3)$$

式 (3) 对应的滤波模板如图 3(b) 所示。

在遥感图像处理中, 拉普拉斯滤波的实质是拉普拉斯模板与输入图像滑动卷积的过程, 如图 4 所示。在输入图像上不断移动滤波模板, 将模板系数与模板覆盖区域对应的像素灰度值进行点乘再求和, 即可得到输出图像中目标像素的灰度值。图像卷积是基于邻域的运算, 而图像边缘上的像素缺乏完整的邻域, 因此需要对图像边缘上的像素进行特殊处理。本文采用在输入图像外部填充常数的方法处理图像边缘上的像素。

2.2 拉普拉斯滤波的并行设计

2.2.1 算法可并行性分析

拉普拉斯滤波算法使用 3×3 模板计算图像中每个目标像素的卷积值, 即将滤波模板系数与模板覆盖区域对应的像素灰度值进行点乘再求和。该算法的特点是: 每个目标像素的计算过程相同, 而且目标像素的计算没有特定顺序, 即一个目标像素的计算结果不会影响其他像素的计算结果。因此, 每个目标像素的计算互相独立, 可以同时进行。利用 OpenCL 在 GPU 上进行并行计算, 可以加速拉普拉斯滤波。

2.2.2 算法的并行设计

2.2.2.1 数据划分

为了发挥 GPU 大规模并行计算的能力, 需要对数据进行合理的划分。OpenCL 通过配置索引空间划分数据, 包括工作项配置和工作组配置。

在工作项配置中, 根据数字图像的特点, 将索引空间的维度设置为二维。因为拉普拉斯滤波对每个目标像素的计算可以同时进行, 因此, 为获得最大的并行性, 将每个目标像素与工作项一一对应, 即每个工作项处理一个目标像素。假设图像的宽度为 W , 高度为 H , 则二维索引空间的宽度为 W , 高度为 H , 包含

$W \times H$ 个工作项。

在工作组配置中, 将整个图像划分为若干不重叠的子块, 并将图像子块与工作组一一对应。假设每个工作组的宽为 S_x , 高为 S_y , 则整张图像被划分为 $(W/S_x) \times (H/S_y)$ 个子块。 S_x 和 S_y 的选择会影响图像数据的局部性, 进而影响到缓存的命中率, 因此对程序的性能有重要影响。在 OpenCL 中, 可以通过 `clEnqueueNDRangeKernel` 函数指定 S_x 和 S_y , 实现图像子块的划分。另外, 在指定工作组大小时, 应满足 $W\%S_x = 0, H\%S_y = 0$ 这个条件, 否则, 需要通过添加虚假工作项的方法来填充索引空间^[9]。

2.2.2.2 数据映射

在上述数据划分方案中, 每个工作项负责处理一个目标像素。通过索引空间计算相应像素的内存地址, 然后到内存中读取相应的像素并进行卷积运算, 最后把结果存储到对应的位置中。在上述过程中, 需要将每个工作项在索引空间的位置与一个目标像素进行映射。假设像素坐标为 (I_x, I_y) , 工作项 ID 为 (μ, λ) , 则 $I_x = \mu, I_y = \lambda$ 。其中, μ 和 λ 分别为工作项在索引空间 x 方向和 y 方向上的坐标。这样, 每个工作项就可以通过像素坐标 (I_x, I_y) 计算相应像素的内存地址, 并确定拉普拉斯模板覆盖的像素区域。

2.3 基于硬件资源的优化方法

在上述拉普拉斯滤波算法的并行设计中, 为获得最大的并行性, 安排每个工作项处理一个目标像素。然而, 这样做并未充分利用 GPU 的硬件资源。GPU 的 Load/Store 流水线单位时间可读写 128 bit 的数据, 算术流水线单位时间可计算 128 bit 的数据, 而图像滤波的输入图像数据类型一般是 char 类型, 宽度为 8 bit。因此, 当每个工作项只处理一个宽度为 8 bit 的目标像素时, 实际上浪费了 120 bit 的带宽和

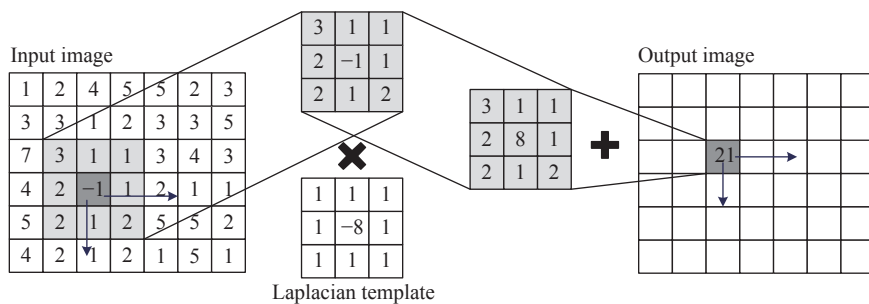


图 4 拉普拉斯模板滑动卷积

Fig. 4 Laplacian template sliding convolution

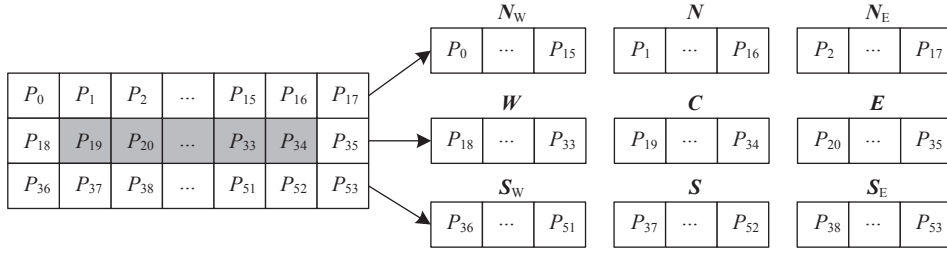


图 5 拉普拉斯滤波内核向量化示例

Fig. 5 Example of vectorizing the Laplacian filtering kernel

计算资源. 因此, 下面从合理利用 GPU 硬件资源的角度进一步提高算法的执行效率.

2.3.1 向量化

Midgard 架构的 GPU 基于 SIMD 设计, 在硬件上支持长度为 16 Byte 的向量操作, 这意味着 GPU 一次处理 16 Byte 的数据与一次只处理 1 Byte 的数据所需的时间相同. 因此, 为了最大限度地利用 GPU 的带宽和计算资源, 应尽可能使用向量化^[10]技术.

拉普拉斯滤波算法具有数据连续访问的特点, 因此可以使用向量化技术优化算法的性能. 如图 5 所示, 灰色区域为要计算的连续 16 个目标像素的位置. 计算这 16 个像素需要加载 3 行共 54 个像素, 并且每行有连续的 18 个像素. 每个像素大小一般为 1 Byte, 因此 GPU 可以使用向量一次加载 16 个像素. 于是, 使用 3 次向量加载即可加载完每行的 18 个像素. 例如, 在加载第一行的像素时, 使用向量 N_W 加载 $\{P_0, P_1, \dots, P_{14}, P_{15}\}$, 向量 N 加载 $\{P_1, P_2, \dots, P_{15}, P_{16}\}$, 向量 N_E 加载 $\{P_2, P_3, \dots, P_{16}, P_{17}\}$. 以此类推, 用 W , C , E 三个向量加载第二行像素, 用 S_W , S , S_E 三个向量加载第三行像素. 这样只需要 9 次向量加载操作即可加载完这 54 个像素.

基于以上分析, 安排每个工作项使用向量化技术计算连续的 16 个目标像素. 在使用向量化技术之前, 每个工作项只计算一个目标像素, 因为每计算一个目标像素点需要 9 次内存加载操作, 因此计算 16 个目标像素总共需要 144 次内存加载操作. 由此可知, 使用向量化技术后, 内存加载次数为原来的 $\frac{1}{16}$.

2.3.2 向量重组

如果相邻的两个向量加载操作访问的数据有重叠, 那么就会存在向量加载冗余. 分析拉普拉斯滤波算法的向量加载过程, 可以发现向量加载冗余的存在. 如图 6 所示, 计算一个目标向量需要用 9 个向量

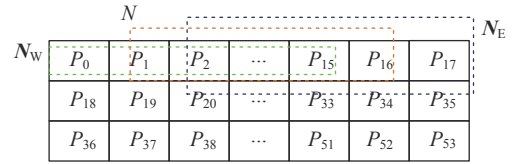


图 6 向量加载冗余

Fig. 6 Redundancy in vector loading

加载数据, 分别为 N_W , N , N_E , W , C , E , S_W , S , S_E . 其中, $N_W = \{P_0, P_1, \dots, P_{14}, P_{15}\}$, $N = \{P_1, P_2, \dots, P_{15}, P_{16}\}$, $N_E = \{P_2, P_3, \dots, P_{16}, P_{17}\}$, 以此类推. 可以看到, N_W , N , N_E 三个向量中存在相同的数据, 向量 N 完全可由向量 N_W 和向量 N_E 中的数据重组得到, 向量 W , C , E 和向量 W , S , S_E 同样有类似规律. 因此, 可以使用向量重组^[11]的方法去构造向量 N 、向量 C 和向量 S , 而无须到内存中重复加载.

向量重组通常使用移位、提取和拼接等方法构造新向量^[12]. 本文采用提取、拼接的方式实现向量重组. 以向量 N_W , N , N_E 为例, 首先提取向量 N_W 的元素 $\{P_1, \dots, P_{14}, P_{15}\}$ 和向量 N_E 的元素 $\{P_{16}\}$, 然后将 $\{P_1, \dots, P_{14}, P_{15}\}$ 和 $\{P_{16}\}$ 拼接成向量 N .

除了在向量加载同一行的像素时存在数据冗余外, 对于邻行两个相邻目标向量的计算, 也存在向量加载冗余. 如图 7 所示, D_1 和 D_2 为要计算的邻行两个相邻目标向量. 为计算 D_1 , 需要用向量 N_{W1} , N_1 , N_{E1} , W_1 , C_1 , E_1 , S_{W1} , S_1 , S_{E1} 加载数据, 为计算 D_2 , 需要用向量 N_{W2} , N_2 , N_{E2} , W_2 , C_2 , E_2 , S_{W2} , S_2 , S_{E2} 加载数据. 从图 7 可以看到, N_{W2} 和 W_1 的数据相同, N_{W2} 完全可重用 W_1 的数据. 以此类推, N_2 , N_{E2} , W_2 , C_2 , E_2 可分别完全重用 C_1 , E_1 , S_{W1} , S_1 , S_{E1} 中的数据, 而无需到内存中重复加载. 基于以上分析, 可以安排每个工作项计算邻行两个相邻目标向量, 这样在计算第二行目标向量时, 可以重用在计算第一行目标向量时加载的数据, 从而向量加载次数可

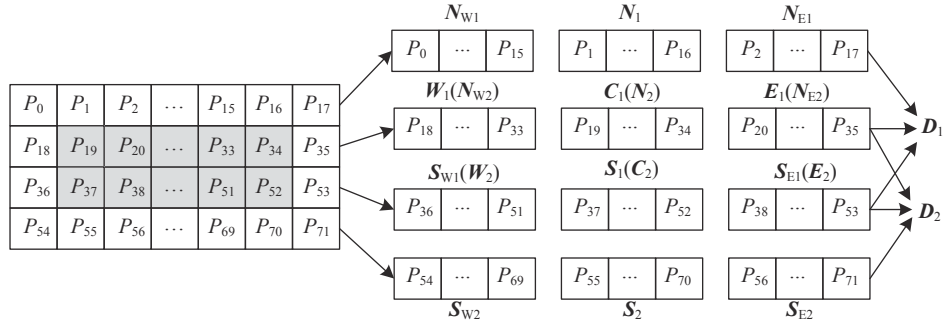


图 7 相邻行目标向量的计算存在重复使用的数据

Fig. 7 Calculation of two target vectors in adjacent rows involves duplicated data

从 18 次减少为 8 次. 以此类推, 当每个工作项计算的相邻目标向量行数增大时, 重用的数据随之增多, 内存加载次数可以进一步减少, 算法执行效率理应提高. 然而, 这也可能会导致每个工作项占用的寄存器数量增大, 使得 GPU 每一时刻并行的活跃工作项数减少, 最终反而导致算法的性能下降. 因此要合理安排每个工作项计算的相邻目标向量行数, 才能达到优化的目的.

2.3.3 工作组调优

工作组的大小和形状对程序的性能有重要影响^[13]. 一方面, 如果工作组过大, 会增加向量寄存器的使用量, 当寄存器不够用时, 数据会溢出到内存中, 导致内核性能下降. 另一方面, 工作组的选择影响缓存的使用, 因为通过调整工作组的大小和形状, 可以让相邻线程尽可能访问相同的缓存行, 从而实现合并访问, 提高访存效率. 在 OpenCL 中, 将 `clEnqueueNDRangeKernel` 函数中的工作组参数设置为 NULL, 编译器会自动确定一个适合的工作组. 然而, 这样挑选出来的工作组性能往往不是最优的, 为获得最优工作组需要人为通过实验测定. 可采用穷举的方法测定最优工作组. 由于 GPU 最适合的工作组大小为 2 的幂次方以及工作组大小最大为 256, 因此分别测试大小为 2^M ($2 \leq M \leq 8$) 的工作组及对应各种形状的内核性能, 然后通过比较确定最优工作组.

以尺寸为 2048×2048 的输入图像为例, 通过调整工作组的大小和形状, 测定拉普拉斯滤波内核的执行时间, 从而找到最优的工作组, 实验环境详见 3.1 节. 测定结果如图 8 所示, 图 8 显示了在不同工作组大小下最优形状和最坏形状对应的内核运行时间, 其中深色条纹表示最坏形状对应的内核运行时间, 浅

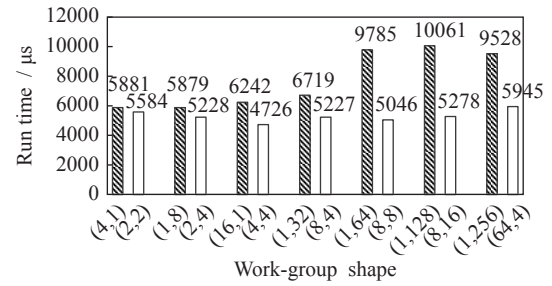


图 8 不同工作组大小下最坏形状 (深色条纹) 和最优形状 (浅色条纹) 对应的内核执行时间

Fig. 8 Kernel execution times corresponding to worst (dark stripes) and optimal (light stripes) shapes for different workgroup sizes

色条纹表示最优形状对应的内核运行时间.

从图 8 可以看出, 在工作组大小为 16, 形状为 (4, 4) 时, 内核的执行时间最短, 对应的内核执行时间为 4726 μs , 此时的工作组即为最优工作组. 性能最差的工作组大小为 128, 形状为 (1, 128), 对应的内核执行时间为 10061 μs . 与性能最差的工作组相比, 采用最优的工作组后, 性能提升了 53.02%. 当输入图像为其他尺寸时, 也获得了类似的结果.

3 实验结果与分析

3.1 实验环境

实验平台采用瑞芯微的 RK3399. RK3399 包含 2 个 ARM Cortex-A72 核、4 个 ARM Cortex-A53 核和四核 Mali-T860 GPU. 其中, Mali-T860 GPU 的频率为 650 MHz. 编程语言为 OpenCL1.2, 操作系统是 Fedora 28. 在测量内核时间之前, 先对 GPU 进行预热, 随后循环调用内核 10 次, 并取平均值作为内核的最终执行时间.

3.2 实验结果

3.2.1 评价指标

采用加速比衡量 GPU 并行算法相较于 CPU 串行算法执行效率的提升效果. 定义如下:

$$S_p = \frac{T_c}{T_g}. \quad (4)$$

其中, T_c 为 CPU 串行算法的执行时间, T_g 为 GPU 并行算法的执行时间.

3.2.2 CPU 与 GPU 的对比分析

为了验证嵌入式 GPU 的并行计算优势, 在嵌入式 CPU+GPU 异构平台上, 比较拉普拉斯滤波 GPU 并行实现版本与 CPU 串行版本的执行效率. 实验结果如图 9 所示, 输入图像的尺寸分别为 256×256 , 512×512 , 1024×1024 , 2048×2048 , 4096×4096 .

从实验结果可知, 拉普拉斯滤波在 GPU 上并行实现后, 相比于同一异构平台上的 CPU 串行实现, 获得了 4.08~16.92 倍的加速比, 且随着输入图像尺寸的增大, 加速比越大.

3.2.3 优化方法的加速效果测试

如图 10 所示, 分别使用尺寸为 256×256 , 512×512 , 1024×1024 , 2048×2048 , 4096×4096 的图像测试了不同优化方法在 GPU 平台上对拉普拉斯滤波执行效率的影响. 其中 Vector 是在 GPU 基础实现版本上进行向量化后的版本, Vector+Permutation 是在 Vector 版上进行向量重组后的版本, 这两个版本的工作组的大小和形状由编译器自动确定. Vector+Permutation+Workgroup 是在 Vector+Permutation 版本上进行了工作组调优后的版本.

实验结果表明, 内核向量化后, 算法执行效率有明显的提升, 获得了 10.85~40.74 倍的加速比. 这是

因为向量化技术可以使用一条指令同时读取同一行多个像素, 有利于不同目标像素点重用数据, 避免重复加载数据, 从而减少了内存加载次数. 在向量化的基础上, 向量重组使算法执行效率平均提升了 34.81%. 这是因为内核向量重组后, 一些向量的数据直接从其他向量寄存器获取, 消除了不必要的向量内存加载. 而工作项访问寄存器的速度远高于访问内存的速度, 因此向量重组能够提升算法的执行效率. 内核工作组调优后, 算法执行效率在向量重组的基础上平均提升了 12.46%, 这是因为工作组调优改善了数据的空间局部性, 使得相邻线程尽可能访问相同的缓存行, 提高了缓存的命中率.

经过一系列优化后, 最终拉普拉斯滤波在 GPU 上获得了 15.38~56.41 倍的加速比.

3.2.4 加速方法推广测试

为了验证本文方法应用于其他图像滤波的加速效果, 在嵌入式 CPU+GPU 异构平台上加速了其他常见图像滤波, 并进行了测试. 测试用例统一使用 1024×1024 尺寸的图像, 各图像滤波的窗口大小均为 3×3 . 实验结果如图 11 所示. 从图 11 可以看出, 相比基于 CPU 的串行实现, 在 GPU 上并行实现后的图像滤波的加速比可达 26.71 倍以上.

4 结语

以加速拉普拉斯滤波为例, 提出了一种基于嵌入式 CPU+GPU 异构平台的遥感图像滤波加速设计方法. 在对滤波算法进行并行设计时, 通过工作项配置和工作组配置完成图像数据划分, 并实现图像数据与工作项之间的映射. 在此基础上, 针对 GPU 的硬件

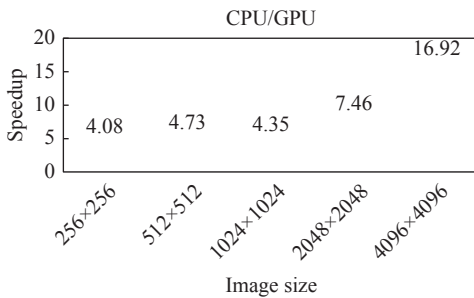


图 9 拉普拉斯滤波 GPU 版本相对于 CPU 版本的加速比

Fig. 9 Speedup of the Laplacian filtering GPU version relative to its CPU counterpart

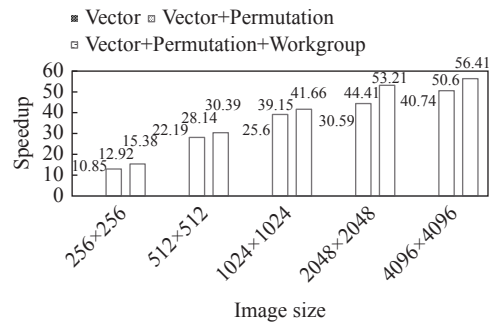


图 10 不同优化方法在 GPU 上获得的性能

Fig. 10 Performance obtained on the GPU through various optimization methods

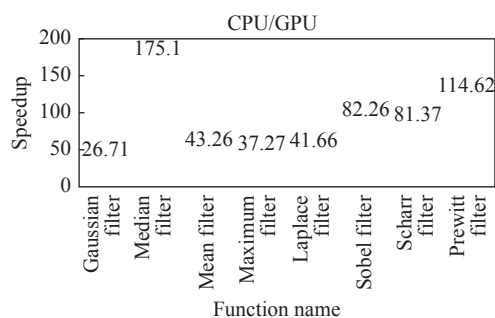


图 11 图像滤波加速比

Fig. 11 Speedup of image filtering

架构特性进一步提高算法的性能. 采用向量化技术减少算法的内存加载次数, 通过向量重组的方法减少向量化带来的内存加载冗余, 通过工作组调优改善数据的空间局部性. 在硬件平台上的实验结果验证了方法的高效性, 并可为加速其他遥感图像处理算法提供参考.

参考文献

- [1] WEI Yuchun, TANG Guoan, YANG Xin, *et al.* Remote Sensing Digital Image Processing Course[M]. Beijing: Science Press, 2007: 174-184 (韦玉春, 汤国安, 杨昕, 等. 遥感数字图像处理教程 [M]. 北京: 科学出版社, 2007: 174-184)
- [2] KOSMIDIS L, RODRIGUEZ I, JOVER-ALVAREZ A, *et al.* GPU4S: Major project outcomes, lessons learnt and way forward[C]//2021 Design, Automation & Test in Europe Conference & Exhibition (DATE). Grenoble, France: IEEE, 2021: 1314-1319
- [3] XIAO H, GUO B Y, ZHANG H Y, *et al.* A parallel algorithm of image mean filtering based on OpenCL[J]. *IEEE Access*, 2021, **9**: 65001-65016
- [4] XIAO H, XIAO S Y, MA G, *et al.* Image Sobel edge extraction algorithm accelerated by OpenCL[J]. *The Journal of Supercomputing*, 2022, **78**(14): 16236-16265
- [5] PANG Y L, JIANG S, CHENG B W, *et al.* Design and implement of median filter toward remote sensing images based on FPGA[C]//2021 IEEE 14th International Conference on ASIC (ASICON). Kunming, China: IEEE, 2021: 1-4
- [6] HARRIS P. The Mali GPU: An Abstract Machine, Part 3-The Midgard Shader Core[OL]. (2014-03-12)[2023-02-10]. <https://community.arm.com/arm-community-blogs/b/graphics-gaming-and-vr-blog/posts/the-mali-gpu-an-abstract-machine-part-3--the-midgard-shader-core>

- [7] Khronos OpenCL Working Group. The OpenCL Specification V1.2[EB/OL]. (2011-11-14)[2013-02-10]. <https://registry.khronos.org/OpenCL/specs/opencl-1.2.pdf>
- [8] ZHOU Xun. Research on Industrial Ray Image Enhancement Algorithm[D]. Guangzhou: South China University of Technology, 2020 (周浔. 工业射线图像增强算法的研究 [D]. 广州: 华南理工大学, 2020)
- [9] SEO S, LEE J, JO G, *et al.* Automatic OpenCL workgroup size selection for multicore CPUs[C]//Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques. Edinburgh, UK: IEEE, 2013: 387-397
- [10] USAMENTIAGA R. Real-time filtering on parallel SIMD architectures for automated quality inspection[J]. *Journal of Real-Time Image Processing*, 2021, **18**(1): 127-141
- [11] LI K, YUAN L, ZHANG Y Q, *et al.* Reducing redundancy in data organization and arithmetic calculation for stencil computations[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. St. Louis, Missouri: ACM, 2021: 84
- [12] DONG Yushan. Optimizations of Memory-access for Stencil Computations on Shared-memory Multi-core Processor [D]. Changsha: National University of Defense Technology, 2015 (董钰山. 面向 SMP 的模板计算访存优化研究 [D]. 长沙: 国防科学技术大学, 2015)
- [13] JIANG S Q, RAN L H, CAO T, *et al.* Profiling and optimizing deep learning inference on mobile GPUs[C]//Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems. Tsukuba, Japan: ACM, 2020: 75-81

作者简介



谭鹏源 男, 1996 年 9 月出生于广西壮族自治区钦州市. 现为中国科学院国家空间科学中心硕士研究生, 主要研究方向为遥感图像并行处理.

E-mail: tanpengyuan19@mailsucas.ac.cn



薛长斌 (通信作者) 男, 1972 年 5 月出生于辽宁省锦州市. 现为中国科学院国家空间科学中心研究员, 博士生导师, 主要研究方向为空间在轨精密过程控制技术、星上数据管理技术及航天系统工程等.

E-mail: xuechangbin@nssc.ac.cn