doi:10.3969/j.issn.1001-4616.2025.04.014

一种基于 GPU 计算的自适应局部降噪并行算法

肖　汉[1,3],杜　莹[2],王　平[1],周清雷[3]

(1.郑州师范学院信息科学与技术学院,河南 郑州 450044)
(2.郑州师范学院地理与旅游学院,河南 郑州 450044)
(3.郑州大学计算机与人工智能学院,河南 郑州 450000)

[**摘要**]　随着获取图像像幅规模的增大和分辨率的提高,自适应局部降噪算法的性能成为制约图像实时处理的关键. 本文提出了一种基于 GPU 的自适应局部降噪并行算法. 从向量化访存、数据本地化计算以及资源配置优化 3 个方面出发,结合算法特性和底层硬件架构特征,研究了自适应局部降噪算法在 CPU+GPU 异构计算平台上的并行计算和性能优化. 实验结果显示,在处理 8 182×8 182 分辨率的图像时,相比 CPU 串行计算获得了 27.39 倍加速比,具有较好的数据扩展性. 并行算法充分发挥了 GPU 的并行处理能力. 文中提出的方法对图像处理算法的 GPU 加速提供了新的研究思路.

[**关键词**]　自适应滤波器,局部噪声,信噪比,GPU,CUDA,并行算法
[**中图分类号**]TP311　[**文献标志码**]A　[**文章编号**]1001-4616(2025)04-0139-14

An Adaptive Local Denoising Parallel Algorithm Based on GPU Computing

Xiao Han[1,3],Du Ying[2],Wang Ping[1],Zhou Qinglei[3]

(1.School of Information Science and Technology,Zhengzhou Normal University,Zhengzhou 450044,China)
(2.School of Geography and Tourism,Zhengzhou Normal University,Zhengzhou 450044,China)
(3.School of Computer and Artificial Intelligence,Zhengzhou University,Zhengzhou 450000,China)

Abstract:With the increase of the size of the acquired image and the improvement of the resolution,the performance of the adaptive local denoising algorithm becomes the key to restrict real-time processing of image. In this paper,a parallel algorithm of adaptive local denoising based on GPU is proposed. From the three aspects of vectorized memory access,data localization computing and resource allocation optimization,combined with the characteristics of the algorithm and the underlying hardware architecture,the parallel computing and performance optimization of adaptive local denoising algorithm on CPU + GPU heterogeneous computing platform are studied. The experimental results show that when processing 8 182×8 182 resolution images,compared with the CPU serial calculation,the speedup ratio is 27.39 times,which has better data scalability. The parallel algorithm gives full play to the parallel processing ability of GPU. The method proposed in this paper provides a new research idea for GPU acceleration of image processing algorithms.

Key words:adaptive filter,local noise,signal-to-noise ratio,GPU,CUDA,parallel algorithm

图像信息在采集、传输和记录过程中,常常遭到各种噪声的干扰,图像的视觉效果受到了严重地影响. 因此,采用适当的方法尽量减少噪声,是在进行边缘检测、特征提取、图像分割、模式识别等处理之前的一项非常重要的预处理步骤[1]. 目前,噪声滤波器从整体上常可以分为线性滤波器和非线性滤波器两种. 两者算法针对的是不同的噪声源,其中,线性滤波器对加性高斯噪声有较好的平滑作用,抑制脉冲噪声则主要用非线性滤波器. 而处理高斯噪声和脉冲噪声组成的混合噪声时,两类滤波器均无法获得很好的效果[2].

自适应滤波器是根据对图像的局部特征的判断进行抑制噪声,滤波器充分利用处理的像素点的局部

收稿日期:2024-08-20.
基金项目:国家自然科学基金项目(61572444、61250007).
通讯作者:肖汉,博士,教授,研究方向:并行算法研究与设计、遥感大数据并行处理的研究. E-mail:xiaohan@163.com

信息,以区分图像局部的变化是由于噪声引起的还是因图像特征的变化(如边缘等)所引起,以便更好地发挥线性滤波器和非线性滤波器的作用. 自适应滤波器的输出是根据图像的局部方差来调整计算,其滤波结果优于线性滤波器,同时图像的边缘和高频细节信息可以更好地保存. 由于自适应滤波器具有更强的适应性和更优的滤波性能,其在通信信道的自适应均衡,系统模型识别,噪声中信号的检测、跟踪、增强和线性预测等领域中得到广泛的应用[3]. 自适应局部降噪算法中含有二维相关运算,当图像像幅较大时,运算工作量很大,降低算法时间复杂度以及在高性能硬件平台上提高算法的运算效率,已经成为图像处理领域中的研究重点.

FPGA 并行实现要求设计人员对底层硬件的执行细节有较深刻的理解,编译生成的 FPGA 配置文件需要利用专门工具,系统设计难度高、效率低. 随着硬件技术的发展,GPU 计算逐渐受到人们的关注. GPU 计算是利用众核计算对同一问题进行求解,和使用传统 CPU 单核/多核计算形成对比,能够在一定程度上解决算法运算耗时的问题. 自适应局部降噪算法研究主要是采用一些传统的并行计算方式进行去噪声处理,算法性能提升的效果不理想,不能很好地满足实时处理. 因此,本文提出了一种基于 GPU 的自适应局部降噪(GPU_ALD)并行算法. 它采用向量计算模式和共享存储器技术,通过对不同线程地址映射方法下自适应局部降噪并行算法的计算效率的分析,进行算法并行优化研究,试图找到基于此条件下的最优数据划分方法,并在众核计算环境下启用的线程块维度和自适应局部降噪并行算法计算效率之间建立关联,从而为算法优化提供参考. GPU_ALD 并行算法是在灰度图像上直接进行并行化,最高可获得 27.39 倍的加速比,并具有良好的数据可扩展性,可有效提升去噪算法运算过程的效率.

1 相关研究工作

根据环境的变化,自适应噪声滤波器使用自适应算法来修正滤波器的参数和结构. 这方面一直备受学者和研究人员的关注. 燕红文等[4]提出了不同的自适应中值滤波方法,改善了图像质量和降低噪声,提高了信噪比. 刘智嘉等[5]提出了一种自适应窗口的中值滤波改进算法,图像的细节、边界得到保持,提高了运算速度. 黄琪等[6]介绍了一种信噪比较高的新型迭代自适应滤波算法,算法的实时性较为突出. 薄振桐[7]实现了基于 FPGA 的自适应中值滤波器,提高了图像处理的实时性. Srinivas 等[8]提出了一种高效的一维非局部均值自适应滤波器,提高了算法处理的实时性.

Kumar 等[9]基于 FPGA 研究了一种分数阶的自适应滤波器,实现了有效去噪. Maier 等[10]设计了自适应各向异性滤波算法降低了噪声,利用 CUDA 将速度提高了 8.9 倍. Wang 等[11]在 CUDA 上实现了自适应差分高斯滤波器进行彩色视频图像的卡通风格化,性能提高了 45~180 倍. Doronin 等[12]利用 CUDA 技术实现了自适应 Wiener 滤波器,加快了计算速度.

综上所述,目前自适应滤波算法的研究主要集中在自适应中值滤波算法、自适应非局部均值算法以及利用一些专门的改进技术实现的自适应滤波算法,同时能够利用 FPGA 和 GPU 对这些算法进行加速. 但是,这些算法和应用都主要集中在对各自提出的自适应滤波器的并行处理上,对自适应局部降噪算法在 GPU 上的设计和优化的研究较少,并没有对算法并行性进行充分挖掘,更没有提出如何对 GPU 并行算法的性能进行全面客观评价. 本文基于 CUDA 并行计算平台,有效地对 GPU_ALD 并行算法进行了一系列理论分析. 该方法基于算法核心运算的独立与并行性,对传统算法进行了改进,最大程度地体现了各计算部分的独立与并行关系,从而实现了总体系统的加速. 采用目前主流的多核 CPU 和众核 GPU,分别实现了基于 OpenMP 的自适应局部降噪算法(OMP_ALD)和 GPU_ALD 并行算法. 通过给出不同并行架构下的图像降噪多平台并行算法,提高图像降噪多平台可应用性.

2 算法的研究与分析

2.1 CUDA 架构背景

在 CUDA 架构中,GPU 可以被看成是一组流处理器,配置有多种形式的存储器,如全局存储器、局部存储器、纹理存储器和共享存储器等. 一个多线程系统被划分成多个 CUDA 子系统,不同子系统之间相互独立,被自动平均分配给 GPU 的不同流处理器. 此外,CUDA 运算平台还为 GPU 提供了一套编程模型,在 CUDA 编程模型中,在 CPU 端以调用核函数的方式来访问 GPU,并且可以指定需要调用的线程块数量以

及线程块中的线程数量. 这些线程会组织成三级层次结构,线程组成线程块,线程块在网格上运行,每一个线程和线程块都会有相应的内置标识符 threadId 和 blockId 来进行标识[13].

2.2　算法论述

当一幅图像被加性噪声退化时,加性噪声项对输入图像 $f(x,y)$ 进行运算,生成一幅退化图像 $g(x,y)=f(x,y)+\eta(x,y)$. 噪声项通常是未知的,因此不能从 $g(x,y)$ 中减去噪声项来得到 $f(x,y)$. 只存在加性随机噪声时,可用空间滤波的方法来估计 $f(x,y)$. 自适应滤波器的特性会根据 $n\times n$ 图像邻域 S_{xy} 定义的滤波区域内的图像的统计特性变化. 一个随机变量的最简单的统计测度是其均值和方差,这些参数是自适应滤波器的基础,因为它们与图像的外观高度相关. 自适应局部噪声消除滤波器是一种基于图像的位置独立和噪声加性独立的模型,并且全部噪声的方差 σ_η^2 不大于去噪滤波器所在像素点的局部方差 σ_L^2,在这种假设前提下,建立了一种自适应、局部噪声消除滤波器. 若滤波器作用于局部区域 S_{xy},中心化区域中任意点 (x,y) 上的滤波器响应基于下面 4 个量[14]:

(1)带噪图像在点 (x,y) 上的灰度值为 $g(x,y)$;

(2)干扰 $f(x,y)$ 以形成 $g(x,y)$ 的噪声方差 σ_η^2;

(3)在 S_{xy} 中像素点的局部均值为 m_L;

(4)在 S_{xy} 中像素点的局部方差为 σ_L^2.

其中,像素点坐标为 (x,y),S_{xy} 是作用于局部区域内的自适应滤波器所在的 $n\times n$ 窗口,$f(x,y)$ 是无噪图像中 (x,y) 像素点的灰度值[15]. 假如 $\hat{f}(x,y)$ 是滤波后图像中像素点 (x,y) 的灰度值,那么滤波器的预期性能如下:

(1)如果 σ_η^2 为零,则滤波器简单地返回 $g(x,y)$ 的值,即在无噪声情况下,$g(x,y)$ 等于 $f(x,y)$;

(2)如果局部方差与 σ_η^2 高度相关,则滤波器返回一个 $g(x,y)$ 的近似值,即局部方差与边缘相关,应该保留边缘;

(3)如果两个方差相等,则滤波器返回局部区域 S_{xy} 中像素算术均值. 该情况发生在 S_{xy} 与全部图像有相同特性的条件下,这时的局部噪声可简单地用求平均来降低[16].

基于这些假设的自适应局部噪声消除滤波算法中的 $\hat{f}(x,y)$ 的表达式可以写成:

$$\hat{f}(x,y)=g(x,y)-\frac{\sigma_\eta^2}{\sigma_L^2}[g(x,y)-m_L]. \tag{1}$$

全部噪声的方差 σ_η^2 是式(1)中唯一需要知道或估计的量,其他参数可从像素点 (x,y) 处的 S_{xy} 中的像素点计算获得,滤波器窗口的中心在 (x,y) 处. 式(1)中,假设 $\sigma_\eta^2\leqslant\sigma_L^2$,据模型中的噪声是加性和位置独立的,因此,这是一个合理的假设,因为 S_{xy} 是 $g(x,y)$ 的子集. 但是,实际中很少有确切 σ_η^2 的知识,所以实际中很可能违背这个假设条件. 由于该原因,对式(1)的实现应构建一个测试,以便当条件满足时,将比率设置为 1. 这使该滤波器成为非线性,这样可以防止由于缺乏图像噪声方差的知识而产生无意义的结果,即负灰度值. 另一种方法是允许产生负值,在最后重新标定灰度值. 但这种方法将损失图像的动态范围[17].

2.3　自适应局部降噪算法的执行特征

2.3.1　算法热点分析

对自适应局部降噪算法并行设计时,首先需要对算法的热点进行分析,以确定算法性能瓶颈的位置. 设测试图像像幅为 8 182×8 182,所取图像局部区域大小为 3×3. 使用 float 作为数据元素类型. 记录结果时为保证测试时间的准确性,取多次运算的结果平均值作为实验的测试结果. 算法各功能模块的计算过程时间及在计算总时间中占比如表 1 所示.

表 1　自适应局部降噪各个模块执行时间及占比

Table 1　Running time and proportion of each step of adaptive local denoising

主要计算步骤	执行时间/ms	时间占比/%
输入图像初始化	971.00	9.88
扩充图像	1 130.90	11.51
自适应局部降噪	5 846.20	59.50
滤波图像输出	1 877.70	19.11
总计	9 825.80	100.00

通过表 1 数据可以看出,在计算过程中,自适应局部降噪在计算过程中时间占比为 59.50%,是计算量最大步骤. 局部均值和局部方差需要在计算点周围取一个 3×3 的二维窗口,局部均值需要在一个二维数据块上求和并求出均值. 那么,在二维图像数据

中的每一个像素点上依次进行局部均值、局部方差和自适应局部降噪滤波处理会产生很大的计算量,所以减少降噪滤波处理的计算时间开销是该算法需要解决的主要问题.

2.3.2　时间复杂度分析

假设噪声图像像幅大小为 $H\times H$,图像局部区域大小为 $n\times n$. 算法的第一步输入图像初始化主要是读入噪声图像,可以不进行考虑;第二步中,图像扩展步骤的时间复杂度为 $O(H^2)+O(H\times n)$;第三步的自适应局部降噪步骤中,需要对每个像素点在 $n\times n$ 图像区域内计算其局部均值和局部方差,他们的时间复杂度均为 $O(n^2)$. 同时这些操作需要对 $H\times H$ 范围内的所有像素点进行计算,所以,本步的时间复杂度为 $O(H^2n^2)$;第四步滤波图像输出主要是输出滤波后图像,可以不进行考虑. 至此可见,整体自适应局部降噪算法时间复杂度为 $O(H^2n^2)$,是由第三步自适应局部降噪处理过程决定. 因此,第三步成为算法并行化的关键步骤,这与上面的运算时间占比分析情况是一致的. 当 $H\geqslant n$ 时,算法时间复杂度可视为 $O(H^2)$.

2.4　并行化可行性分析

并行算法要在 GPU 上正确实现,需要确保循环中不存在数据间的依赖性和数据处理间的竞争性. 在自适应局部降噪处理步骤中,需要在每个像素点上进行其局部均值和局部方差的计算操作. 可以发现,$H\times H$ 范围内的所有像素点数据相互独立可单独操作. 同时,对像素点的局部均值和局部方差的计算互不影响,相互独立. 因此,可以利用 GPU 并行计算图像的自适应局部降噪过程,提高算法运算效率.

利用 CUDA 进行并行加速,为图像的像素点创建线程. 此时,让 GPU 中的每一个线程来计算每一个像素点的局部均值和局部方差. 由于所有线程同时执行相同的计算过程,自适应局部降噪算法时间复杂度将减少到 $O(1)$,这是一个恒量等级. 若未能在内核函数中处理完所有像素点,那么自适应局部降噪内核函数将至少被每个线程执行 $\frac{H^2}{tsum}$ 次,其中 $tsum$ 是网格中活动线程总量. 这种情形下,时间复杂度为 $O\left(\frac{H^2}{tsum}\right)$. 在 GPU 架构下,一般系统中能够维持的活动线程量很大,即 $tsum$ 将是一个很大的量值. 因此,存在自适应局部降噪并行算法时间复杂度 $O\left(\frac{H^2}{tsum}\right)\leqslant O(H^2)$.

3　自适应局部降噪算法的异构计算平台实现

3.1　并行算法描述

利用 GPU 的计算网格把图像处理任务划分为不同子图像块,并通过 CUDA 线程施以并行计算. 具体的自适应局部图像降噪并行算法描述如下.

算法 1　基于 GPU 平台的自适应局部图像降噪并行算法

输入:图像大小为 $H\times H$ 的带噪图像矩阵 ***srcImageData***,局部区域 $n\times n$,每个线程对 $BX\times BY$ 个像素点自适应局部降噪
输出:去噪图像矩阵 ***desImageData***
Begin
　　srcImageData ← 图像大小为 $H\times H$ 的带噪图像
　　在 x 方向和 y 方向分别初始化线程块索引 bx 和 by
　　定义在每个线程块(bx,by)中的共享存储器中的子图像块空间大小 *SubImage_ds*
　　将线程块(bx,by)中的所有线程需要访问的输入子图像块数据从全局存储器中加载到大小为 *SubImage_ds* 的共享存储器中
　　所有线程都等待在线程块(bx,by)的共享存储器缓冲区中的数据运算完成.
　　for $i=0$ to n do
　for $j=0$ to n do
　　对线程块(bx,by)中所有线程对应大小为 $n\times n$ 的图像区域进行局部均值计算
　end for
　　end for
　for $i=0$ to n do
　for $j=0$ to n do
　　对线程块(bx,by)中所有线程对应大小为 $n\times n$ 的图像区域进行局部方差计算

```
end for
    end for
    for i=0 to BX do
for j=0 to BY do
    线程块(bx,by)中的每个线程处理 BX×BY 个像素的自适应局部去噪
end for
    end for
    将自适应局部去噪处理结果 desImageData 从全局存储器传输到主存
End
```

3.2　方法设计

自适应局部降噪并行处理详细的建模流程如图 1 所示. 在自适应局部降噪算法 GPU 并行化处理过程中,首先载入图像获取图像的宽、高和大小等信息. 然后定义算法处理过程中需要的所有变量,并分配对应大小的存储空间. 其中,主机端用 malloc 函数分配空间,设备端用 CUDA 指令 cudaMalloc 分配空间. 利用 cudaMemcpy 命令将原图像数据以及所需的中间数据从主机存储器复制到设备内存. 自适应局部降噪算法在遍历到图像边缘的时候,计算会发生图像二维矩阵寻址越界的情况,需要对原图像进行扩充. 针对自适应局部降噪算法的计算特点,设计一个 GPU 核函数用于完成图像中各子图像块的局部均值和方差的计算任务,以及局部降噪滤波结果的计算. 依据分配的线程空间利用相应的 ID 进行子图像块数据的读取和处理,以及调整图像数据存储器类型. 之后通过 GPU 核函数完成自适应局部降噪算法的并行过程,最后将局部降噪处理结果传输至主机端进行显示.

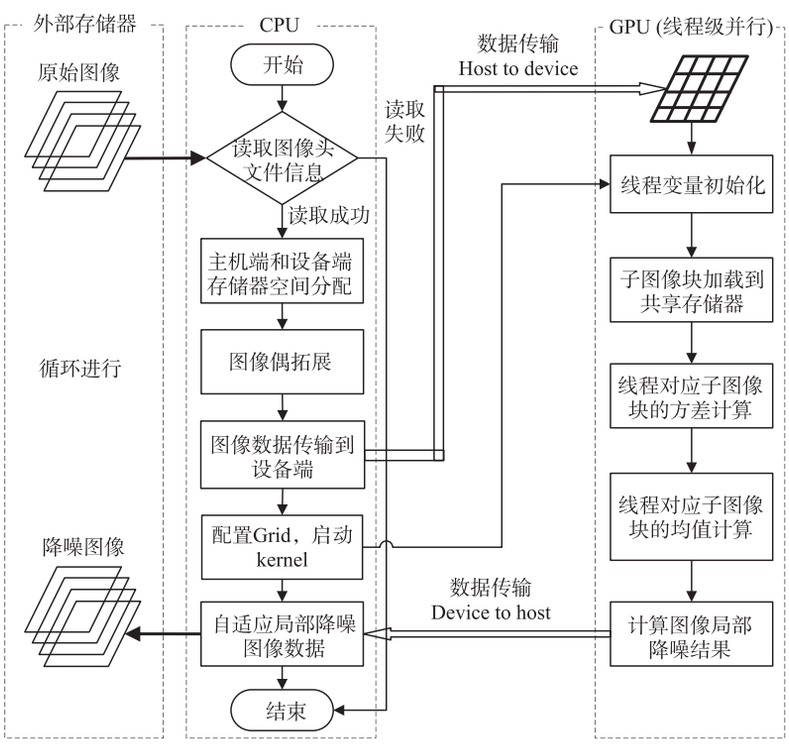


图 1　基于 CUDA 的自适应局部降噪并行计算流程图

Fig. 1　Parallel computing flow diagram of adaptive local denoising based on CUDA

3.3　基于 CPU+GPU 异步计算的并行化

3.3.1　*并行方案设计*

处理单元与执行线程的映射主要在于 CUDA 核心与线程实现一一逻辑对应. GPU_ALD 并行算法的并行设计方案如图 2 所示. 图中显示了 GPU 线程 ID 与噪声图像矩阵的映射关系. 通过 GPU 线程 ID 与图像像素点坐标的映射,实现每个 GPU 线程使用唯一的坐标或者线程索引计算各自需要处理的数据,达到最大程度的细粒度并行.

整体自适应局部降噪处理由一个对应的内核函数进行处理. 算法执行过程中,内核函数与 Grid 一一对应. 将图像区域 Width×Height 分解为若干子图像块,Grid 将子图像块的处理任务分配给相应的线程块,从而实现多个子图像块并行处理的粗粒度并行. 线程块将黑色实心方块像素点的处理任务再分配给相应的线程进行处理. 内核函数在主机端进行调用,在 GPU 端运行. 图 2 中的计算网格启动的线程块所覆盖的范围比整个图像大,以保证所有像素点都有相应的线程进行处理. 计算网格中的阴影表示的线程块超过图像边界的范围,算法不作任何处理.

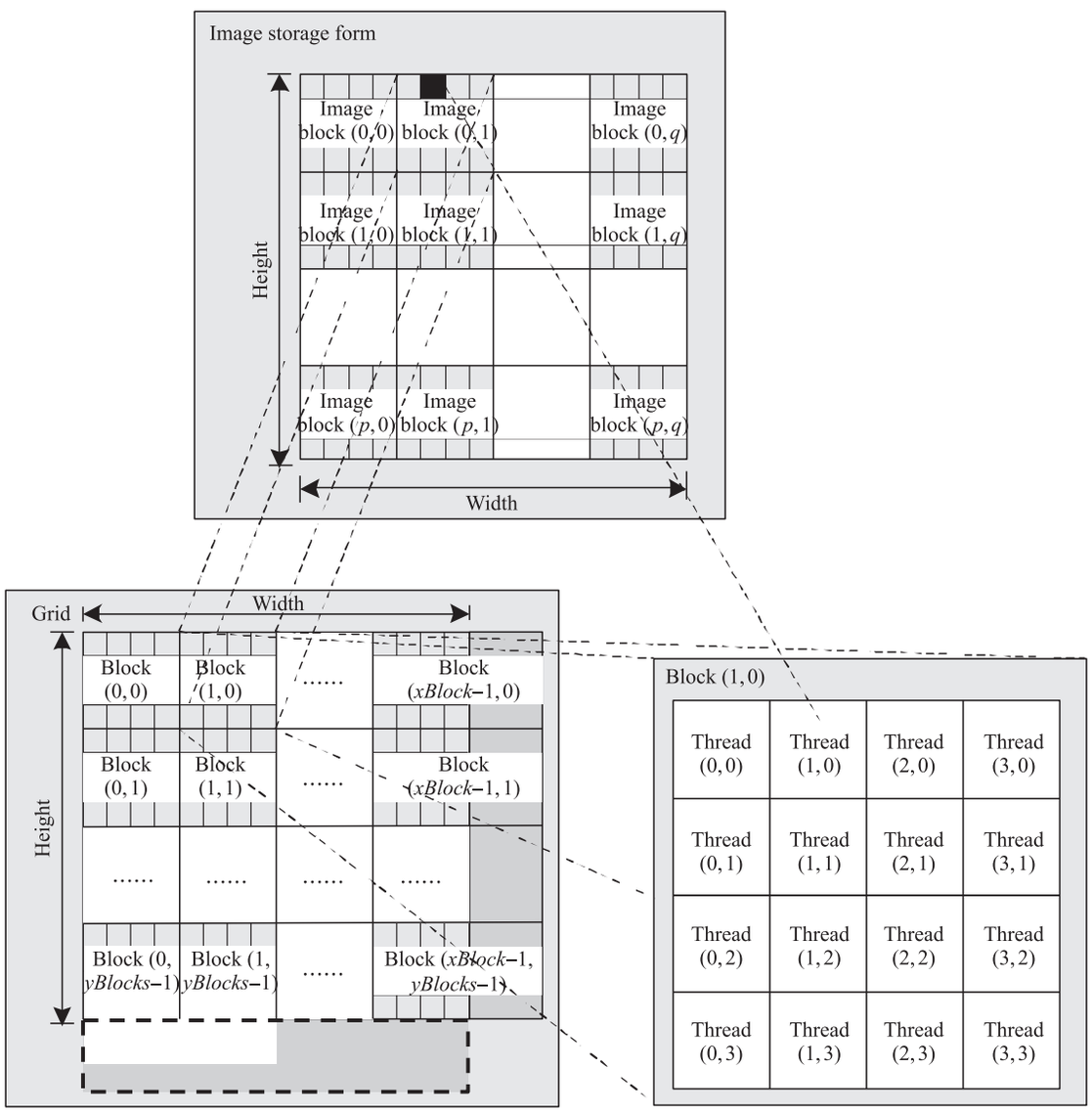


图 2　算法并行处理图解

Fig. 2　Diagram illustration of algorithm parallel processing

3.3.2　像素的坐标变换

GPU_ALD 算法具有显著的数据计算并行性. 首先根据图像数据的大小确定所需线程块的数量,其计算公式为:

$$xBlocks=\left|\frac{(W+xTHREADS-1)}{xTHREADS}\right|, \tag{2}$$

$$yBlocks=\left|\frac{(H+yTHREADS-1)}{yTHREADS}\right|, \tag{3}$$

式中,$xBlocks$、$yBlocks$ 分别为线程块在 x、y 方向上的数量,W、H 分别为图像的宽度和高度,$xTHREADS$、$yTHREADS$ 分别为在 x、y 方向上线程块包含线程的数量. 然后根据相应图像数据与线程块中线程的对应

关系构造线程索引如下:

$$tidx = threadIdx.x + blockIdx.x * blockDim.x, \tag{4}$$

$$tidy = threadIdx.y + blockIdx.y * blockDim.y, \tag{5}$$

式中,*threadIdx.x* 和 *threadIdx.y* 分别为线程块中的线程在 x 和 y 方向上的索引,*blockIdx.x* 和 *blockIdx.y* 分别为计算网格中的线程块在 x 和 y 方向上的索引,*blockDim.x* 和 *blockDim.y* 分别为线程块在 x 和 y 方向上的维度. *tidx* 和 *tidy* 分别代表线程块中的某一个线程在整个网格中在 x 和 y 方向上索引号.

3.4　优化策略

3.4.1　向量模式

在算法的并行设计中设定一个线程处理一个输出像素点. 由图 3 可见,如果要处理深色阴影区的 16 个像素点则需要 16 个线程. 为了增加每个线程的工作负载,进一步提高带宽利用率,每个线程需要处理多个像素点. 根据向量访问技术,图中线程 $thread(i,j)$ 处理 $p(i,j)$、$p(i+1,j)$、$p(i,j+1)$ 和 $p(i+1,j+1)$ 四个像素输出点,以此类推. 若算法中的局部区域大小为 3×3,每个线程需要一次读取 16 个图像数据,与原来的访问 36 个图像数据相比,减少了 56%的存储器访问. 而每个线程块则需一次读取 36 个数据(一个线程块开启了 4 个线程). 在传输降噪图像时,每次获得的 4 个存储空间位置连续的 4 B 数据(图像数据采用 4 B 表示)可以视同一个 16 B 数据. 在 GPU 存储器访问事务中,传输一个 1 B 数据或 16 B 数据所用时间相同. 因此,每次产生 4 个输出点比产生 1 个输出点可以减少 75%的访存时间,向量访问技术能有效提高存储访问效率.

由于全局存储器访存效率较低,导致访存片外的存储器开销增大. 为了提高算法的性能,将图像数据从片外存储器转移至片内并实现数据共享,以此来降低对片外存储器访存的依赖. 共享存储器属于片上存储器,其访问延迟远低于全局存储器的访问延迟. 因此,GPU_ALD 算法通过数据共享存储器化,减少对全局存储器的访存依赖,实现算法性能的提升.

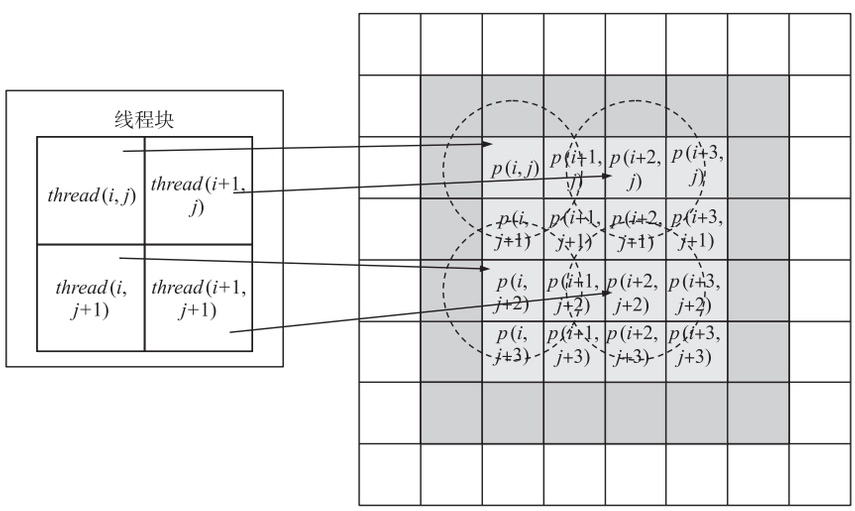


图 3　线程块内的向量化访问

Fig. 3　Vectorized access within a thread block

3.4.2　数据传输优化

噪声图像、图像局部均值和局部方差数据占用的存储空间较大,在自适应局部降噪算法中多次被用于后续计算. 如果这些数据多次在主机与设备之间频繁传输,将会耗费大量的系统时间. 另一方面,PCI-E 的数据传输带宽相比 GPU 存储器传输带宽较小,与 GPU 访问设备存储器的速度存在着巨大的差距. 为了避免这种无谓的数据传输,系统中将图像局部均值和局部方差数据设置在全局存储器,尽量让数据保持在设备存储器中. 并且,将噪声图像数据一次性的从主机存储器传输到设备存储器中. 这样可以有效避免增加 CPU 和 GPU 之间交换数据的传输时间. 同时,这种数据重用方式可实现一次取数和生成数据,多次运算. 因此,数据重用方式减少了数据在设备间的换出换入操作,减少了系统访存延时,提升了数据利用效率和系统的并行处理能力.

3.4.3　线程块资源配置优化

并行算法中线程块的大小要合理划分. 若尺寸较大,系统将会启动更多的线程参与并行计算,有助于提高算法执行效率;若线程块尺寸较小,则占用的系统资源也较少,因而可以提高流多处理器中的活动 warp 的数量. 一般考虑线程块中线程的数量应保持在 32 到 512 之间,并且取为 32 的整数倍. 然而不同 GPU 的硬件参数不同,所以,在自适应局部降噪算法中需要调节线程量参数以达到最优的性能,这样系统的执行效率将会提高. 当图像大小为 8 182×8 182,局部区域大小为 3×3 时,线程块大小对系统运算时间的影响情况如表 2 所示.

表 2　不同线程块大小的运算时间

Table 2　Operation time of different thread block sizes

线程块大小	运算时间/ms	线程块大小	运算时间/ms
8×8	223.15	24×24	217.09
16×16	213.44	32×32	219.45

以上实验结果显示,对于 GTX1070 显卡来说,在线程块大小为 16×16 时系统运行效率较高,其余情况下系统运行效率均会下降.

4　性能评测与分析

4.1　测试环境

CPU 采用 Intel Core 六核 i7 8700K,3.7 GHz 的主频;DDR4 的系统内存,16 GB 的内存容量,工作频率为 2 400 MHz;显卡采用的是 NVIDIA Geforce GTX1070,属于 Pascal 构架. GTX1070 具有 1 920 个 CUDA 核,计算能力 6.1,核心频率 1 506 MHz,搭载 8 GB GDDR5 显示存储器,256 bit 显存位宽,256 GB/s 显存带宽.

软件操作系统采用 Microsoft Window 11 64-bit,GPU 软件开发包为 CUDA 11.5,集成环境为 Microsoft Visual Studio 2019,图像编辑软件 Photoshop CS6.

4.2　图像质量评价

为了对自适应局部降噪并行滤波算法的有效性、优越性进行验证,文中分别运用主观评判法和客观评判法衡量滤波器消除噪声的图像质量.

4.2.1　主观评判结果

对电路板图像(455×440)、船舶图像(720×576)、Lena 图像(512×512)和飞机场图像(1 024×1 024)分别用均值为 0,发生概率为 0.5%和 1.0%的高斯噪声,以及噪声密度为 1.0%和 2.0%的椒盐噪声作为图像的噪声污染模型进行实验. 基于 CPU 的自适应局部降噪串行算法 CPU_ALD,OMP_ALD 和 GPU_ALD 并行算法的滤波实验采用 3×3 大小的局部滤波区域,实验结果如图 4 和图 5 所示. 图 4(a)和图 5(a)分别是高斯噪声图像和椒盐噪声图像,图 4(b)、图 5(b),图 4(c)、图 5(c)和图 4(d)、图 5(d)三组图像分别为用 CPU_ALD、OMP_ALD 和 GPU_ALD 系统进行处理后的结果. 可以明显地从滤波后的图像中看出,电路板顶部的连接片更为清晰,孔和图像左下方较暗部件的 8 根引脚等也更加清楚. 船头部分字母更加明显,Lena 的左侧鼻子和嘴角两侧的白色噪声已经消失. 飞机场左侧浅色的农田部分可以明显的看到噪声点的减少. 通过图像对比可知,本文提出的自适应局部噪声并行滤波算法达到了消除不同噪声类型和不同噪声程度的目的. 同时,经过对三种降噪滤波算法结果的对比可看出均得到了比较清晰的图像,肉眼难以区分三者之间的差异.

4.2.2　客观评判结果

除了对图像滤波效果直接对比观测,为了确保实验的准确性和客观性,文中还通过图像峰值信噪比(peak signal to noise ratio,PSNR)和结构相似性(structural similarity index,SSIM)定量对比多种计算方式对噪声图像处理的效果.

PSNR 作为一种衡量噪声水平或图像失真的客观标准,常用于图像降噪前后对图像劣化程度的客观评价. 其表达式如下:

$$\mathrm{PSNR}=10\log_{10}\frac{L^2}{\mathrm{MSE}}(\mathrm{d}B), \tag{6}$$

式中,MSE(mean square error)为图像均方误差:

$$\mathrm{MSE}=\frac{1}{MN}\sum_{x=1}^{M}\sum_{y=1}^{N}(f(x,y)-\hat{f}(x,y))^2, \tag{7}$$

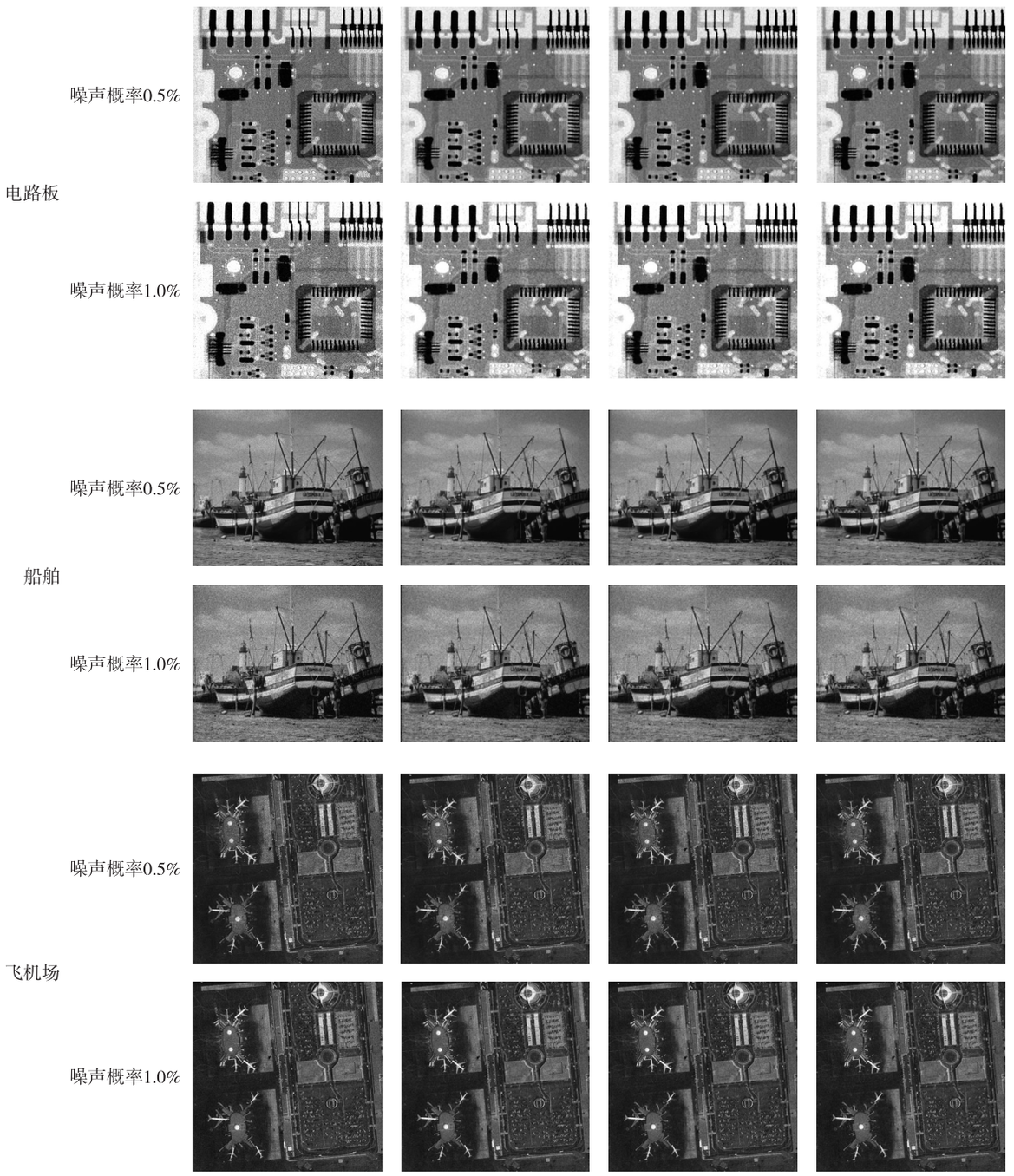


图 4　自适应局部降噪效果图(高斯噪声)

Fig. 4　Adaptive local denoising effect shows(Gaussian noise)

式中,M、N 分别是 x、y 方向图像像素点个数,$f(x,y)$ 和 $\hat{f}(x,y)$ 分别是参考图像和失真图像在点(x,y)上的取值,L 是图像量化的灰度级数. PSNR 值越大,去噪效果就越好. 即失真越少,图像质量越高.

SSIM 是用于衡量两幅图像相似性的一种方法,它综合考虑了亮度、对比度和结构三个方面的信息,其表达式如下:

$$\mathrm{SSIM}(x,y)=\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)}, \tag{8}$$

式中,μ_x,μ_y 分别为图像 $f(x,y)$ 和 $\hat{f}(x,y)$ 的均值,σ_x^2,σ_y^2 分别为图像 $f(x,y)$ 和 $\hat{f}(x,y)$ 的方差,σ_{xy} 为图像 $f(x,y)$ 和 $\hat{f}(x,y)$ 的协方差,c_1,c_2 是两个常数. 表 3 显示了测试图像在高斯噪声发生概率为 0.5%和椒盐噪声噪声密度为 1.0%时的 PSNR 数据,表 4 显示了测试图像在高斯噪声发生概率为 0.5%和椒盐噪声噪声

密度为 2.0%时的 SSIM 数据.

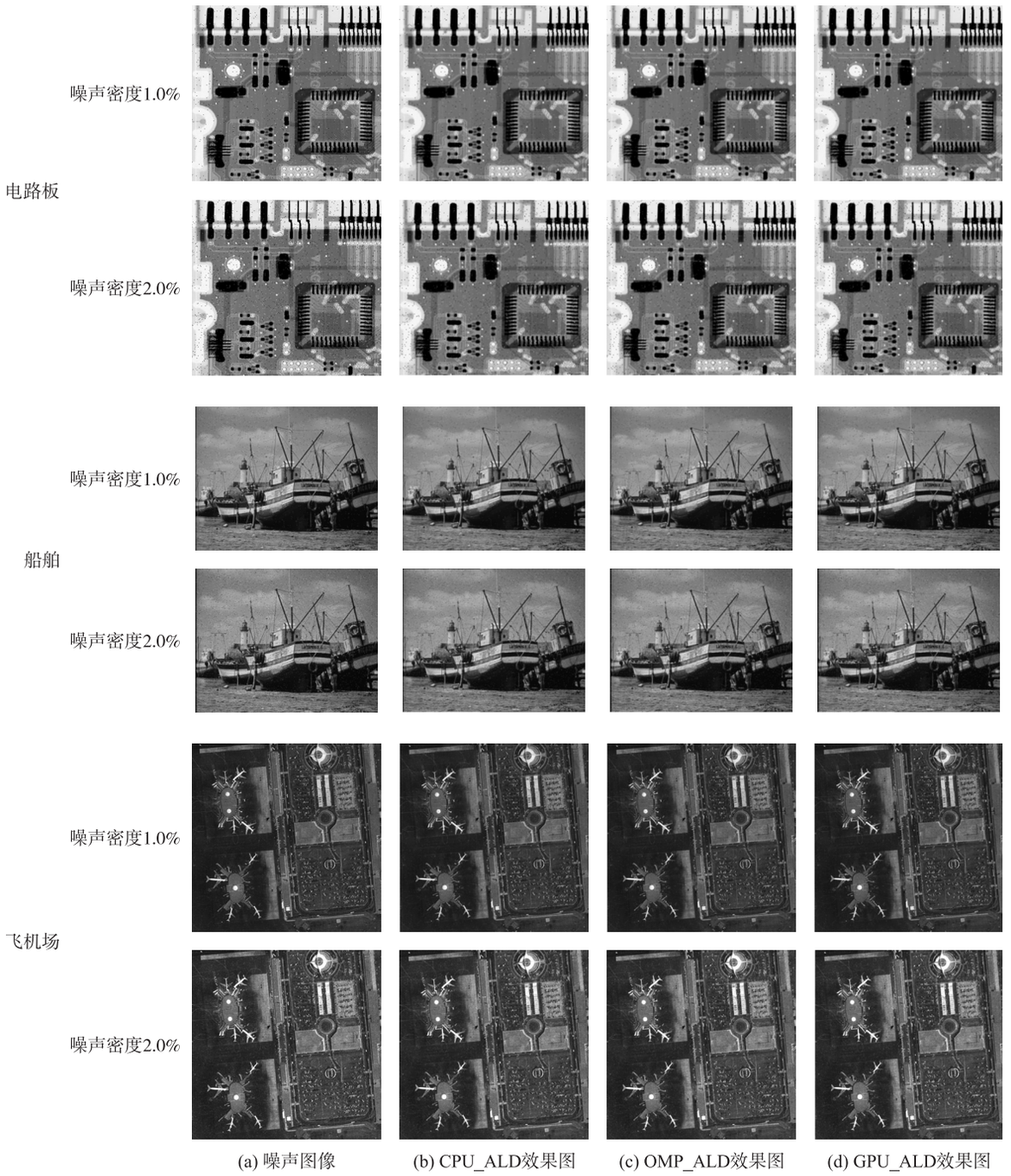


图 5　自适应局部降噪效果图(椒盐噪声)

Fig. 5　Adaptive local denoising effect shows(salt-and-pepper noise)

表 3　各图像 PSNR 对照

Table 3　Comparison of PSNR in each image

图像名称	噪声类型	无处理	CPU_ALD	OMP_ALD	GPU_ALD
电路板	高斯噪声	7.54	7.66	7.66	7.66
	椒盐噪声	24.96	27.31	27.31	27.31
船舶	高斯噪声	25.87	30.99	30.99	30.99
	椒盐噪声	25.38	29.91	29.91	29.91
飞机场	高斯噪声	19.15	24.32	24.32	24.32
	椒盐噪声	25.20	27.75	27.75	27.75

表 4　各图像 SSIM 对照

Table 4　Comparison of SSIM in each image

图像名称	噪声类型	无处理	CPU_ALD	OMP_ALD	GPU_ALD
电路板	高斯噪声	0.51	0.79	0.79	0.79
	椒盐噪声	0.62	0.83	0.83	0.83
船舶	高斯噪声	0.40	0.75	0.75	0.75
	椒盐噪声	0.52	0.79	0.79	0.79
飞机场	高斯噪声	0.49	0.69	0.69	0.69
	椒盐噪声	0.53	0.78	0.78	0.78

由表 3 和表 4 数据可见,对比分辨率大小不同的四种图像,自适应局部降噪算法在处理噪声图像时取得了较高的 PSNR 值和 SSIM 值,与之前各组图像优劣对比的主观感知相符. 同时,通过观察,发现各组图像串/并行算法的 PSNR 值和 SSIM 值基本一致,验证了本文提出的 GPU_ALD 并行算法的正确性和有效性.

4.3　实验数据分析

4.3.1　算法运算时间分析

由于并行算法设计中既有不可并行化的部分,又有并行化部分,对算法的整体计算速度进行统计分析并不能准确体现 GPU 加速算法执行的能力. 因此,以自适应局部降噪部分功能为对象,统计该算法部分的运行时间. 同时,实现相同功能、相同参数的串行算法 CPU_ALD 并统计运行时间,以此分析 GPU_ALD 并行算法的加速性能.

首先通过图像编辑软件工具 Photoshop 产生一批不同尺寸的图像,具体包括图像分辨率为 236×256、432×512、640×480、1 024×1 244、2 048×2 352、4 096×4 136 和 8 182×8 182 的 7 组实验数据. 然后采用 3×3 的局部窗口大小,对比测试 CPU_ALD、OMP_ALD 和 GPU_ALD 三种串行/并行算法分别处理这组图像的耗时. 由于这些算法采用对图像文件一致的读/写方式,并运行在 CPU 端. 所以,图像文件的读/写和算法中不可并行化的部分耗时不计入. 三种自适应局部降噪串行/并行算法运算耗时对比如表 5 所示,其中时间数据采用经过多次的算法测试取平均而得.

根据表 5 绘制折线图 6,以便将自适应局部降噪算法在串行平台和不同并行平台上的执行时间更直观的展现出来. 从图中可以看出,OMP_ALD 和 GPU_ALD 并行算法的运算时间相比 CPU_ALD 串行算法有了较明显的减少. 特别是在 GPU_ALD 并行算法中,随着像幅大小的增加,算法中启动的线程量增多,计算资源的利用率提高. 使得 GPU_ALD 并行算法的运行时间得到了更大幅度的降低,算法执行速度提升明显.

表 5　自适应局部降噪算法耗时比较

Table 5　Time-consuming comparison of adaptive local denoising algorithm

像幅大小/px	串行处理时间/ms	并行处理时间/ms		像幅大小/px	串行处理时间/ms	并行处理时间/ms	
		OMP_ALD	GPU_ALD			OMP_ALD	GPU_ALD
236×256	5.79	2.16	0.71	2 048×2 352	328.00	79.42	13.92
432×512	22.40	6.16	1.06	4 096×4 136	1 452.20	333.60	55.29
640×480	24.22	6.46	1.08	8 182×8 182	5 846.20	1 270.80	213.44
1 024×1 244	88.78	23.26	3.86				

4.3.2　算法加速性能分析

在并行算法中,加速比是较为直接的性能评估指标,体现了并行算法求解实际问题时性能提升的能力.

定义 1　CPU_ALD 串行算法的执行时间 $T_{\text{CPU_ALD}}$ 与 OMP_ALD 并行算法的执行时间 $T_{\text{OMP_ALD}}$ 之比,被称为 S_{OMP} 加速比,如式(9)所示.

$$S_{\text{OMP}}=\frac{T_{\text{CPU_ALD}}}{T_{\text{OMP_ALD}}}. \tag{9}$$

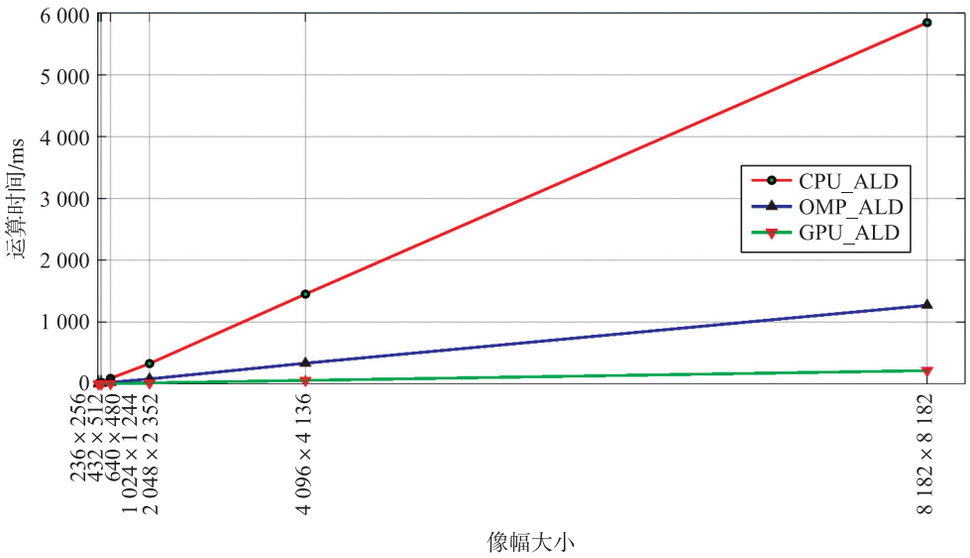


图 6　图像数据集运行时间比较

Fig. 6　Running time comparison of image dataset

定义 2　CPU_ALD 串行算法的执行时间 $T_{\text{CPU_ALD}}$与 GPU_ALD 并行算法的执行时间 $T_{\text{GPU_ALD}}$之比,被称为 S_{GPU}加速比,如式(10)所示.

$$S_{\text{GPU}}=\frac{T_{\text{CPU_ALD}}}{T_{\text{GPU_ALD}}}. \tag{10}$$

定义 3　OMP_ALD 并行算法的执行时间 $T_{\text{OMP_ALD}}$与 GPU_ALD 并行算法的执行时间 $T_{\text{GPU_ALD}}$之比,被称为 $RS_{\text{OMP-GPU}}$相对加速比,如式(11)所示.

$$RS_{\text{OMP-GPU}}=\frac{T_{\text{OMP_ALD}}}{T_{\text{GPU_ALD}}}. \tag{11}$$

OMP_ALD 和 GPU_ALD 并行算法在各组测试图像上取得的加速比如表 6 所示.

表 6　加速结果对比

Table 6　Accelerated results comparison

像幅大小/px	加速比		相对加速比	像幅大小/px	加速比		相对加速比
	S_{OMP}	S_{GPU}	$RS_{\text{OMP-GPU}}$		S_{OMP}	S_{GPU}	$RS_{\text{OMP-GPU}}$
236×256	2.68	8.15	3.04	2 048×2 352	4.13	23.56	5.71
432×512	3.64	21.13	5.81	4 096×4 136	4.35	26.27	6.03
640×480	3.75	22.43	5.98	8 182×8 182	4.60	27.39	5.95
1 024×1 244	3.82	23.00	6.03				

为更直观地观察自适应局部降噪并行算法的性能改善,绘制图 7. 据图分析可知,图像像幅大小在 236×256 至 4 096×4 136 之间时,随着图像大小的增加 S_{OMP}和 S_{GPU}的值均呈现增值态势,并且曲线斜率较大. OMP_ALD 和 GPU_ALD 两种并行算法都体现出了较好的数据可扩展性. 并且显然 GPU_ALD 获得了相比 OMP_ALD 更高的加速比. 由于多线程并行算法的设计在线程启动、同步、调度方面都需要耗时,以及 CPU 物理核心数的限制,S_{OMP}和 CPU 核心数成正比,最大加速比取得了 4.6 倍. 当像幅较小时,GPU_ALD 并行算法的加速比增幅较大. 像幅大小达到 4 096×4 136 时,GPU 的并行扩展性得以完全展开. 这主要是由于此时数据在主机端与设备端之间的传输耗时不大,且 GPU 充裕的计算资源可以创建充足的线程以满足大数据量的并行处理. 随着图像像幅的进一步增加,虽然 S_{GPU}仍然保持了一个较高的数值,GPU_ALD 并行算法在处理大幅图像时最大获得了 27.39 倍加速比. 然而,像幅大小超过 4 096×4 136 时,加速比曲线呈现逐渐平缓趋势. 主要是由于主机与设备间的数据传输的时间开销逐步增大,严重影响了算法的执行效率.

自适应局部降噪算法在不同并行架构下加速情况不同. 随着像幅的增大,OMP_ALD 并行算法的加速比在稳步增大,但是,加速比只能接近 CPU 核数. 而 GPU_ALD 并行算法在整个测试图像集上体现出了更

好的加速效果. 由图 8 可见,GPU_ALD 并行算法相比 OMP_ALD 并行算法可取得 3.04~6.03 倍的加速比. 分析原因,主要是异构众核架构 GPU 在像幅较大时,利用丰富的计算资源启动足够的线程参与并行计算.

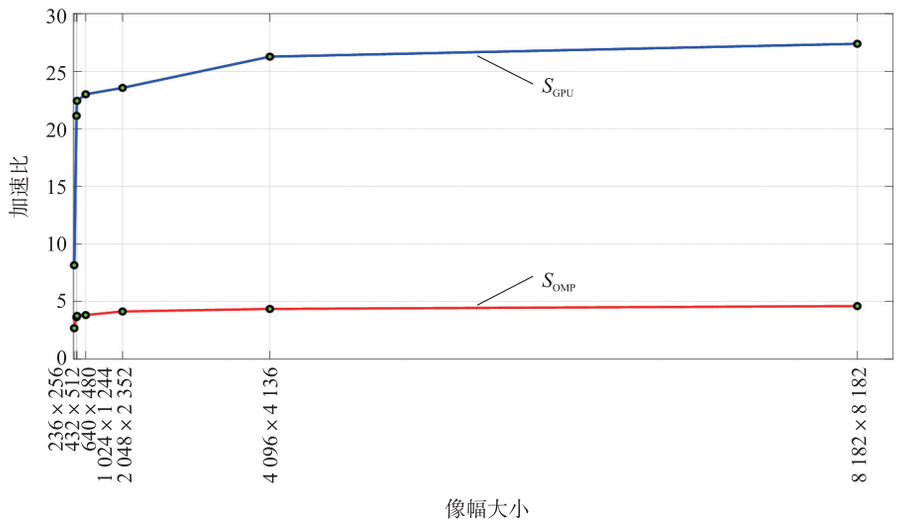


图 7　两种并行算法加速比对比图

Fig. 7　Comparison of speedup of two parallel algorithms

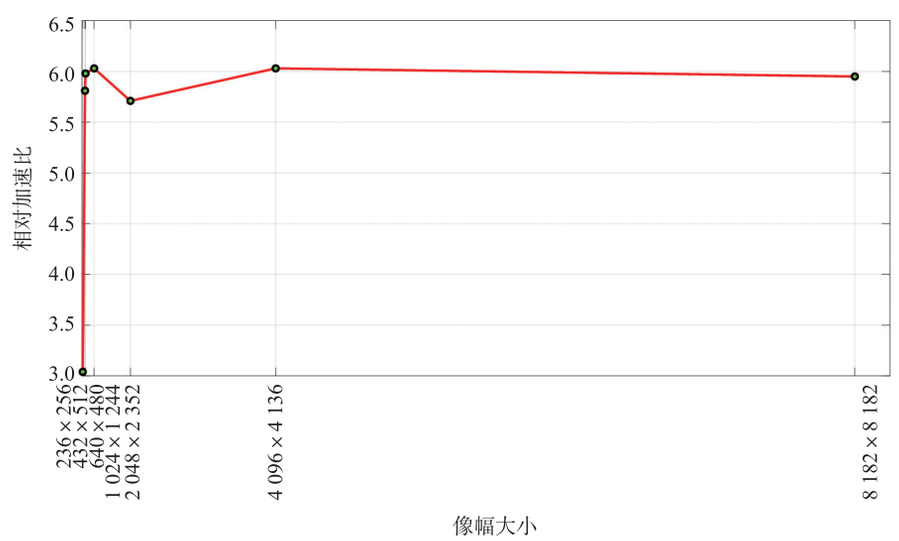


图 8　相对加速比趋势图

Fig. 8　Relative acceleration ratio trend graph

4.3.3　系统性能瓶颈分析

通过分析可知,GPU_ALD 并行算法中的自适应局部降噪处理步骤需要进行大量的存储器读写操作. 需要对数据进行 $H^2n^2\times15$ 次存储器读取,对结果图像进行 H^2n^2 次存储器写入操作. 设原始图像大小为 4 096×4 136,图像局部区域大小为 3×3,每个像素点占用的存储器大小是 4 B. 因此,存取存储器数据总量约为 8 GB. 除以内核执行的实际时间 0.037 s,得到的带宽约为 216.22 GB/s,已是接近 GeForce GTX1070 显存的 256 GB/s 理论带宽. 所以,可以很明显的看出,GPU_ALD 并行算法效率现受限于全局存储器带宽.

图 8 显示,当图像像幅较小时,GPU_ALD 并行算法运算速度的提升较快. 但当图像像幅较大时,GPU_ALD 并行算法的加速效果呈现出缓慢平稳的趋势. 主要原因是在 GPU 并行算法处理中,主机负责读取和输出图像的过程并没有加速. 随着需要处理图像规模的增加,读取和输出图像的耗时逐步增加. 因此,显存带宽和主存与显存之间数据传输带宽成为 GPU_ALD 并行算法的性能瓶颈.

5 结论

本文对 CUDA 并行计算架构和 CPU_ALD 算法的并行性进行了分析和讨论,设计了像素级图像处理 CPU_ALD 算法的 GPU 并行化实现方法. 本文提出的 GPU_ALD 并行算法同时支持了粗粒度并行和细粒度并行. 进一步探讨了像素级的向量化并行处理,采用了一种新的数据存储模式,该数据存储模式提高了共享存储器中线程计算的密度. 实验结果表明,OMP_ALD 算法,特别是 GPU_ALD 并行算法,能够大幅度提高图像降噪的处理速度. 随着图像像幅的增大,GPU_ALD 算法的加速比呈上升趋势,表现出良好的数据可扩展性. GPU_ALD 并行算法最高可取得 27.39 倍的加速比,它将广泛应用于遥感图像去噪、图像增强、医学成像以及图像识别等诸多领域. 文中采用的是单 GPU 与 CPU 协同工作,今后的工作是将单 GPU 上的 GPU_ALD 算法移植到多 GPU 上,以便进一步优化 GPU_ALD 算法,挖掘更大的性能提升潜力.

[参考文献]

[1] HAJIABADI M. Exploring the potential of parallel adaptive filters for audio noise removal[J]. Signal image and video processing,2024,18(1):445-454.

[2] LIU X B,ZHANG Y W,JING H D,et al. One image segmentation method using U-net and res_unet convolutional networks[J]. Rsc advances,2020,10(16):9396-9406.

[3] GAO W F,LIU F,LIU J,et al. Spatial-spectral adaptive learning with pixelwise filtering for hyperspectral image classification[J]. IEEE transactions on geoscience and remote sensing,2023,61(8):1-15.

[4] 燕红文,邓雪峰. 中值滤波在数字图像去噪中的应用[J]. 计算机时代,2020,39(2):47-49.

[5] 刘智嘉,夏寅辉,杨德振,等. 基于中值滤波器的红外图像噪声处理的改进方法[J]. 激光与红外,2019,49(3):376-380.

[6] 黄琪,李鼎权,施涛,等. 一种针对椒盐噪声的新型迭代自适应滤波器[J]. 南开大学学报(自然科学版),2015,48(5):84-89.

[7] 薄振桐. 基于 FPGA 的实时图像自适应中值滤波器设计[J]. 电子科技,2014,27(1):121-124.

[8] SRINIVAS N,SINGH P,PRADHAN G. A novel hardware architecture for non-local means adaptive filter[J]. National academy science letters:India,2020,62(2):1-4.

[9] KUMAR S,JHA R K. An FPGA-based design for a real-time image denoising using approximated fractional integrator[J]. Multidimensional systems and signal processing,2020,26(2):1-23.

[10] MAIER A,WIGSTROM L,HOFMANN H G,et al. Three-dimensional anisotropic adaptive filtering of projection data for noise reduction in cone beam CT[J]. Medical physics,2011,38(11):5896-5909.

[11] WANG Q Y,CHEN D S,LI S R,et al. An adaptive cartoon-like stylization for color video in real time[J]. Multimedia tools and applications,2017,76(15):16767-16782.

[12] DORONIN A,MEGLINSKI I. Imaging of subcutaneous microcirculation vascular network by double correlation optical coherence tomography[J]. Laser and photonics reviews,2013,7(5):797-800.

[13] RUBIO C,GONZALO J,SIMINSKI J,et al. Efficient computation of the geopotential gradient in graphic processing units[J]. Advances in space research,2024,74(1):332-347.

[14] CHOWTHRI V,UMA A,KALPANA P. Design and implementation of low complexity LMS adaptive filter[J]. International journal of nonlinear analysis and applications,2021,12(SI):1827-1833.

[15] ENGINOGLU S,ERKAN U,MEMIS S. Pixel similarity-based adaptive riesz mean filter for salt-and-pepper noise removal[J]. Multimedia tools and applications,2019,78(24):35401-35418.

[16] REN Z L,ZHANG X R,HU Z J,et al. Reducing radiation dose and improving image quality in CT portal venography using 80 kV and adaptive statistical iterative reconstruction-V in slender patients[J]. Academic radiology,2020,27(2):233-243.

[17] KARCZEWICZ M,HU N,TAQUET J,et al. VVC in-loop filters[J]. IEEE transactions on circuits and systems for video technology,2021,31(10):3907-3925.

[责任编辑:杜忆忱]