CUDA 实现的非局部均值图像去噪算法

摘 要：非局部均值滤波是一种能够在有效去除图像噪声的同时保持图像边缘清晰度的图像去噪方法。但是该算法具有高时间复杂度，在处理图像时需要消耗大量时间。这使得该方法难以实用化。因此，提升非局部均值算法的处理速度非常有必要。本文使用 CUDA 实现非局部均值图像去噪方法的并行化。结果表明，在保持图像处理质量不变的前提下，并行化后的处理时间显著缩短。

关键字：CUDA；GPU；非局部均值滤波；图像去噪

1 背景介绍

图像去噪是一类经典的计算机视觉问题。常见的方法有中值滤波、高斯滤波和双边滤波等。这些方法思想是：像素点的真值和他周围的像素是最相关的，通过算法用周围的像素点表示待去噪的像素点。这些方法使用的信息只有像素周围的若干个像素，而离像素点更远的信息将不会参与到像素的去噪过程中。

为了充分的利用图像中的冗余信息，Baudes[1][2]等人于2005年在双边滤波器的基础上行提出了效果更好的非局部均值滤波(Non-local Means Filtering，NLM)。该去噪方法的处理思想是：通过在全局范围内搜索待去噪像素点的相似像素，然后使用加权平均的方法来得到处理后的图像。这种方法在去噪的同时能最大程度地保持图像的细节特征。但是该算法的最大缺陷就是计算复杂度太高，程序非常耗时，导致该算法不够实用。通过 CUDA 在 GPU 上实现 NLM 算法能够显著提升该算法的运算效率。

2 算法分析

2.1 图像降质模型

在图像获取过程中，通常会受到各种因素的干扰而产生图像质量的退化。基本的图像降质过程如图 2.1 所示。

图 2.1 展示的是加性噪声产生的原理。其中，图像 X 表示原始不含噪声的数字图像；n 为零均值的高斯噪声；图像 Y 表示带噪的图像。

图像的去噪问题是一个病态(ill-posed)问题。从模型的建立可以知道，我们已知的是图像 Y，通过作出合理的假设，求得 X 的一个尽可能逼近真值的估计值。

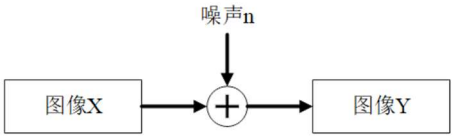


图 2.1 图像降质模型

Fig.2.1 Image degradation model

2.2 非局部均值图像去噪算法原理

根据图像块的正则性原理，在自然图像的图像块中，相似图像块的中心像素一般也是相似的。这一原理最早被用在纹理合成中。随后 Baudes 在这些研究的基础上提出了有着优异的边缘、纹理保持能力的非局部均值图像去噪算法。

一幅图像是由众多像素点构成的，每个像素点和与之邻近的像素点构成了图像子块。而图像子块则在一定程度上反映了图像所包含的纹理信息。如果用所有像素点的图像子块来表示一幅图像，这些信息是冗余的。

举例来说明图像中一定存在重复的结构，比如图像中位于同一纹理边界上的点肯定具有相似的纹理结构，图像中组成同一块图案的像素点也一定有类似的结构。一般情况下定义邻域，是把某一像素点和其周围像素组成的图像子块称为邻域。但引入自然图像自相似性和冗余性以后，图像中的某个像素点的邻域概念可以转变为：若以像素点 i 为中心的图像子块和以像素点 j 为中心的图像子块是相似的，满足这样条件的所有像素点组成的集合被称为邻域。再处理图像中某个像素点时，这样的概念转变使得图像中更多的像素点都彼此的关联起来，能够使用的参与处理某一像素点的相似的像素点数目变多，范围变大。上述依据图像普遍规律得

出的思想就是本文使用的 NLM 滤波的非局部的核心思想。

NLM 滤波把非局部思想和图像块的相似度这两种新颖的方法引入滤波，第一，非局部思想是指自然图像中与每个像素相似的像素点不局限于其周围较小范围区域内，还有可能处于更广阔全局图像中。并且图像去噪效果通常和参与去噪的相似像素点的数量相关，更多的像素参与去噪过程，去噪后的效果有可能更高。第二，像素之间相似度量的准确性是去噪结果准确的先决条件，如果大量不相似的像素被误判为相似的像素，从而参与到去噪过程中，那么这种相似性的误判可想见地必然导致去噪效果的变差。NLM 去噪的基本思路是：不局限于待去噪像素所在的物理距离的邻域而将搜索范围扩张到整幅图像，寻找与待去噪像素相似度高的像素点，利用这些相似像素点采取加权平均的机制进行去噪处理，即相似度越高的像素获得的权重越高，其作用于待去噪像素点的能力也就越大。

NLM 去噪的数学表达式[3]如公式 2.1，i 表示受噪声污染图像中任意一像素点，$I(i)$ 表示其灰度值，i 点去噪后的估计值为：

$$I_{NLM}(i)=\frac{1}{C_j}\sum_{j\in\Omega}w(i,j)I(j)$$

其中，Ω 表示以像素点 i 为中心的搜索区域；j 为 Ω 内任意一个像素点，$I(j)$为像素点 j 的灰度值；权值 $w(i,j)$反映了分别以像素 i,j 为中心的图像块N_i和N_j的相似程度，且满足$0\le \mathrm{w(i,j)}\le 1$和$\sum w(i,j)=1$两个条件；$C_j$为权值归一化系数。NLM 一般用图像块向量间的高斯(Gaussian)加权欧式距来表示图像块中心像素间的相似度距离，权值计算式为：

$$w(i,j)=\frac{1}{Z_j}exp\{-\left\|N_i-N_j\right\|_{2,a}^2/h^2\}*d(i,j)$$

$\left\|N_i-N_j\right\|_{2,a}^2$代表 Gaussian 加权欧氏距离，$a>0$是 Gaussian 标准方差；$h$ 是滤波参数；$d(i,j)$衡量的是像素点 i,j 之间的物理位置远近关系。NLM 滤波器正是因为融入了非局部的邻域思想，它能够在非局部邻域内搜索到更多相似性的像素点，所以获得了比局部平滑滤波器更好的稳健性和滤波效果。

3 CUDA 实现

3.1 CUDA 简介

CUDA(Compute Unified Device Architecture，通用并行计算架构)，是英伟达在 2007 年推出的一款专门针对其旗下 GPU 开发的基于并行编程模型和指令集架构的通用计算架构。CUDA 的优化程序有两大特点[4][5]:

a)使开发线程级并行，在硬件中可以被动态地调度和执行线程。CUDA 把 GPU 看做是一个由若干个线程块(block)组成的线程网格(grid)，每个线程块又包含多个线程线(thread)，线程为 GPU 计算中最小的执行单元。在处理图像数据时，一幅图像可以被划分成若干图像子块，每一个图像子块可映射为一个线程网格。在图像去噪时，每个带噪像素点所经历的处理过程是类似的，因此可以把每个像素的执行作为一个单独的线程。

b)存储层次性访问。在 CUDA 架构中，GPU 的存储空间被划分为不同的层次，包括全局存储器、共享存储器、寄存器、常量存储器、纹理存储器等，这些存储器的大小和访问速度各不相同，在具体应用中可依据实际应用的需要设计数据存取模式，合理地运用存储器也有助于程序时间优化。

3.2 NLM 图像去噪算法的 CUDA 加速

NLM 算法在 CPU 端是运行可以看作是串行化的。而考虑一个算法能否被改为串行运行，需要考虑三个问题：1.并行化后的算法能否得到正确的运算结果；2.并行化如何实现；3.已经并行化后的程序能够进一步优化。

首先，考虑第一个问题，程序并行化后能否得到正确的运算结果。并行化的原理是在同一个时间内同时处理多个相同或相似的任务。这些任务的处理数据可以不同，但是处理的过程一定是相似的。另一方面，这些任务应是不相干的，每个并行化单元不需要利用其它单元的结果作为自己计算的初

始条件或中间步骤，并行个体间互相不依赖，计算才能划分至最小独立单元。

针对图像处理过程并行化问题，图像作为被处理数据。从数学角度可将其看做一个二维矩阵，确定(*x,y*)坐标在二维空间中位置后，即可获得该坐标位置相应像素灰度值。结合 CUDA 的线程模型考虑，将一幅图像划分成若干图像子块，每一个图像子块映射到一个线程网格上进行图像处理，从 CUDA 线程模型角度，图像数据是可划分成独立单元的。在进行图像去噪过程中，每个带噪的像素点所需要经历的处理过程是相似的，且需要经历这个处理过程的像素点个数非常庞大的，并且计算规模是随着图像大小递增。因此把每个像素的重建过程分配到一个线程上独立计算是可行的，总体上看整幅图像处理过程是并行化也是可实现的。

本文 NLM 算法 CUDA 并行化的算法流程如下：

a)CPU 端初始化。在 CPU 端将带噪图像进行延拓，预处理相关参数和创建处理时需要的对数表。

b)GPU 端初始化。在 GPU 端分配对数表空间和去噪结果的空间，它们均是全局内存空间。将延拓后的带噪图像绑定至纹理内存中，然后拷贝相关数据到纹理内存中。

c)运行核函数。CPU 端在完成初始化后调用核函数，GPU 开始并行执行去噪算法。调用时，block 大小设为默认的(16,16)，grid 大小根据图像的大小进行调整。在进行核函数运行时，CPU 端会对核函数的状态进行捕获。若核函数执行发生溢出、越界、内存耗尽等问题，核函数将放弃执行。此时，CPU 端的捕获将会显示相关错误信息。

d)数据拷贝回内存。在核函数正常执行结束后，CPU 端控制相关方法，将显存中的处理结果拷贝回内存中。

e)完成数据最终存储和释放资源。对拷贝回内存的数据需要进行限幅等处理。在完成上述左右步骤后，释放相关资源，并且重设 GPU。

分析相关算法的时间复杂度。假设待去噪图像的大小为 $H*W$，搜索窗口的大小为 $(2*s+1)*(2*s+1)$，比较窗口的大小为 $(2*p+1)*(2*p+1)$。则 CPU 端串行算法的时间复杂为 $O(H*W*s*s*p*p)$。而在进行 CUDA 并行化后，每次像素点的运行是可以同时运行的。理想情况下，并行化后的算法时间复杂度为 $O(s*s*p*p)$。但是实际运行时，受制于数据拷贝、GPU 的实际运算单元数量等因素，时间复杂度将会略大于 $O(s*s*p*p)$。

3.3 纹理内存使用

纹理内存是 GPU 全局存储器中的一个特殊区域，但是其访问速度要比普通的全局存储区更快。这是纹理内存是只读的，其中的数据不随对应的矩阵改变而改变，即一旦设定不可就不可以对其进行数据修改。纹理内存区别于全局存储器等其他存储器有下面三个方面显著特点：

a)纹理内存是显卡存储器中的一个专用高速缓存区域，是显卡芯片专门针对绘图应用而设计的，具有访问上的特殊性质。

b)纹理寄存器没有访问模式的限制。对于全局内存、常量内存来说，一般得遵循一定的访存模式，如半数 warp 中每个线程应该进行排列，当其访问全局内存能够合并到邻近的对齐的内存中时，访问数据的效率才高；而纹理内存的访问没有如此限制，因为纹理内存线程的存取动作有区域性，本身就成块状，不像一般线程的随机存取，当同一个 warp 中的所有线程读取地址都是相近的数据时，线程访问内存效率才能达到最高的。

c)隐藏线程寻址计算的延迟，程序执行随机访问数据的性能非常高。

CUDA 中提供纹理内存使用的相关 API 有纹理饮用和纹理对象两类。二者在使用上最大的区别在于，纹理引用在每次使用后需

表 4.1 不同去噪算法的定量结果(时间：毫秒)

Table.4.1 Quantitative results of different image denoising methods(time: ms)

(a)img1-cameraman.tif

sigma	中值滤波	高斯滤波	双边滤波	NLM

	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM
5	0.11	27.14	0.84	7.19	23.12	0.74	1.49	36.11	0.91	1013.39	35.35	0.92
10	0.14	26.64	0.77	6.31	23.01	0.72	1.65	30.91	0.77	1011.55	32.76	0.90
20	0.07	25.18	0.61	17.33	22.76	0.67	1.50	25.71	0.56	1025.40	28.52	0.83

(b)img2-lena.png

sigma	中值滤波			高斯滤波			双边滤波			NLM		
	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM
5	0.18	35.10	0.90	5.27	30.41	0.86	2.74	36.12	0.90	3990.01	36.80	0.92
10	0.14	32.71	0.83	6.57	30.22	0.84	3.00	31.07	0.74	3996.40	35.42	0.91
20	0.17	28.69	0.65	4.82	29.34	0.79	2.75	25.71	0.49	4043.63	31.55	0.84

(c)img3-fence.png

sigma	中值滤波			高斯滤波			双边滤波			NLM		
	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM
5	0.23	33.96	0.86	5.77	27.82	0.80	7.86	42.46	0.98	9787.17	34.00	0.84
10	0.22	34.06	0.90	7.03	28.65	0.91	7.25	31.33	0.80	9738.22	37.01	0.96
20	0.23	29.12	0.74	5.44	27.64	0.87	6.16	26.02	0.59	9742.37	31.75	0.91

(d)img4-wood.png

sigma	中值滤波			高斯滤波			双边滤波			NLM		
	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM
5	0.63	40.99	0.95	8.22	37.95	0.97	10.37	37.79	0.90	21843.65	42.46	0.97
10	0.40	35.85	0.85	5.33	37.53	0.95	10.09	32.72	0.74	21846.10	39.87	0.96
20	0.55	30.42	0.64	5.41	31.71	0.89	9.55	27.42	0.51	21856.13	33.99	0.93

(e)img5-roof.png

sigma	中值滤波			高斯滤波			双边滤波			NLM		
	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM	Time	PSNR	SSIM
5	1.36	40.57	0.96	5.60	37.05	0.98	34.73	37.17	0.90	90437.83	43.36	0.98
10	1.75	35.37	0.87	5.59	36.65	0.96	36.43	32.07	0.76	90291.28	40.13	0.97
20	1.57	30.06	0.68	5.78	32.12	0.90	33.24	27.15	0.55	90238.60	34.09	0.93

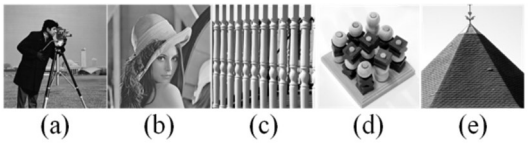
要重新进行绑定；纹理对象则不需要重新绑定。同时，纹理对象适用于算力更高的显卡(Compute Capability >= 3.0)，且访问速度更快。

本文使用纹理对象的方式，将延拓后的带噪图像绑定至纹理内存中，进一步提升带噪图像的访问速度。

4 实验对比

本文的测试硬件平台为：GPU 为 NVIDIA GTX 780(3GB)；CPU 为 Intel Core i7-4790K(4C8T)；内存为 16GB。

测试软件平台为：Visual Studio 2015，CUDA 9.2.148，驱动程序版本为 441.41，OpenCV 3.4.5。



(a)cameraman(256*256), (b)lena(512*512),
(c)fence(800*800), (d)wood(1200*1200),
(e)roof(2448*2448)

图 4.1 纯净图像

Fig.4.1 Images without noise)

实验时，待测试的图像是 5 张分辨率不同的图像。这 5 张图像均是单通道灰度图像，在去噪前均添加了标准差为 5、10、20 的零均值高斯白噪声。图 4.1 是 5 张纯净图像。

图 4.2 是其中一张纯净图像添加不同标准差的噪声后的结果。

(a)sigma=5, (b)sigma=10, (c)sigma=20

图 4.2 噪声标准差不同的带噪图像

Fig.4.2 Noise images with different σ

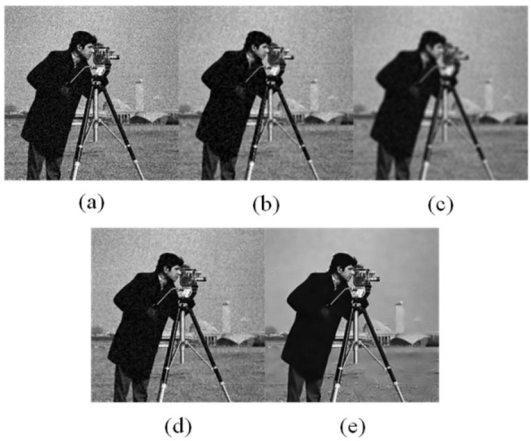
实验主要分为两个部分，一部分是 NLM 算法和其他常见算法的对比，包括处理时间和处理效果；另一部分是 NLM 算法在 CPU 和 GPU 上运行的时间和处理效果的对比。

4.1 与其他常见算法对比

该部分实验对比的去噪算法有中值滤波、高斯滤波和双边滤波。这些算法的实现均是由 OpenCV 提供的。这些方法的的参数尽可能的和 NLM 算法保持一致。

对比的指标主要有算法在 CPU 上处理带噪图像消耗的时间、PSNR 和 SSIM。

处理的结果如图 4.3 所示，实验数据的对比如表格 4.1 所示。



(a)噪声图像(sigma=20), (b)中值滤波,

(b)(c)高斯滤波, (d)双边滤波, (e)NLM

图 4.3 不同去噪方法视觉对比

Fig.4.3 Visual comparison of different image denoising methods

从处理的结果图像可以看出，图像分辨率不同时，各个算法的处理时间均会随着处理数据量的增加而增加。但是不管怎样，NLM 算法的时间消耗总是最多的，且显著高于其他算法。

此外，对比处理效果，不管是 PSNR 还是 SSIM，NLM 都要好于其他算法。且到图像中的噪声程度增加时，NLM 的这一优势会进一步扩大。

同时，从主观表现上来看，NLM 算法的去噪效果最好；中值滤波的去噪能力明显不足；高斯滤波容易产生多余的模糊，虽然 PNSR 和 SSIM 比中值滤波高，但是实际观感不如中值滤波优秀；双边滤波去噪效果不是特别理想。

4.2 NLM 算法在 CPU 和 GPU 上运行的对比

该部分算法比较的是算法在不同运算平台上的运行的速度和处理质量对比。σ 算法在不同平台上处理时，参数一致。

实验测试时对比了 CPU 串行运行、CPU 多核运行和 CPU+GPU 运行的处理实现和效果对比。实验时所添加的零均值高斯噪声的标准差均为 10。对比的指标有处理图像消耗的时间、PSNR、SSIM 和加速比。这里加速比是指代码在 CPU 上串行运行的时间除以在其他平台上运行的时间所得数值。例如 CPU 串行运行的加速比始终为 1。

实验定量处理结果如表格 4.2 所示。统计对比图如图 4.4 所示。

实验结果表明，在测试的五张图像中，GPU 对于 NLM 图像去噪算法均有加速效果。当图像的分辨率较小时，加速效果不明显；当图像的分辨率较大时，算法的运行速度有显著提升。同时三种不同平台上运行的图像结果是相同的，没有显著差异。

(a)噪声图像(sigma=10), (b)CPU 结果

(c)OMP 结果, (d)GPU 结果

图 4.4 不同平台上 NLM 对比图

Fig.4.4 Comparison of performance of NLM on different processing platforms

表 4.2 不同平台上 NLM 的定量结果(时间：毫秒)

Table.4.2 Quantitative performance of NLM on different processing platforms(time: ms)

img	CPU				CPU 多核 OpenMP				CPU+GPU			
	Time	PNSR	SSIM	Ratio	Time	PNSR	SSIM	Ratio	Time	PNSR	SSIM	Ratio
1	1029.00	32.76	0.90	1	198.02	32.76	0.90	5.20	137.69	32.76	0.90	7.47
2	4010.78	35.42	0.91	1	715.84	35.42	0.91	5.60	160.54	35.42	0.91	24.98
3	9789.09	37.01	0.96	1	1748.22	37.01	0.96	5.60	231.33	37.01	0.96	42.32
4	21888.18	39.87	0.96	1	3904.81	39.87	0.96	5.61	328.57	39.87	0.96	66.62
5	90400.88	40.13	0.97	1	16184.40	40.13	0.97	5.59	937.27	40.13	0.97	96.45

分析上面结论的原因。首先对比使用 GPU 进行通用计算的流程和只使用 CPU 进行计算的流程之间的差异。我们不难发现，GPU 进行通用计算仅是将大量可并行的任务并行化运行了，这是 GPU 能够加速算法运行的主要原因。此外，相比较 CPU 计算，进行 GPU 通用计算还需要在处理任务前将数据从内存拷贝至显存，处理结束后需要将结果从显存拷贝回内存。这是 GPU 通用计算多消耗的时间。那么在进行 NLM 图像去噪时，如果数据拷贝时间和处理时间之和接近或者大于 CPU 处理时间时使用 GPU 进行运算将不是一个明智而合理的选择。

从表 4.2 我们可以知道，不管图像的分辨率大小是使用 OpenMP 的加速比始终是 5.5 左右。实验使用的机器为 4 核心 8 线程处理器。除了每个核心固有的 4 个固定线程外，剩余的 4 个线程是通过超线程技术获得的。而根据 Intel 超线程技术手册，其对处理器性能的提升约为 20%-30%。换算后，理论上 OpenMP 的加速比应为 5.2 左右，和实验所得的结果是相近的。

而 NLM 算法在 GPU 上的提升十分明显，最高可以达到近 100 倍的提升。当处理的数据量较小时，GPU 的性能并未完全发挥，加速比不到 10；而当图像分辨率较高时，需要处理的数据量明显增大，GPU 的并行化处理优势便体现出来了。

5 总结

本文在实现非局部均值图像去噪算法的基础上，使用 GPU 对该算法进行并行化加速。实验证明，改进后的非局部均值图像去噪算法能够在保持图像去噪效果不变的情况下，使用原始算法约 1/10-1/100 的时间完成处理。

此外，针对 NLM 算法本身，本文使用的测试代码使用的并不是高斯加权的欧式距离，而是等权的欧式距离。当使用其他相似性度量方式时，NLM 算法的处理效果将会有进一步提升；NLM 算法本身实现方式上，GPU 算法还有进一步优化提升的空间，可以将 GPU 中每个线程处理的任务进步划分，进一步提升处理的粒度。因此，CUDA 实现的 NLM 图像去噪算法在处理效果和处理速度上都还有进一步提升的空间。

参 考 文 献

[1] Buades A , Coll B , Morel J M . A non-local algorithm for image denoising[C]// Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. IEEE, 2005.

[2] Coll B . A Review of Image Denoising Algorithms, with a New One[J]. Siam Journal on Multiscale Modeling & Simulation, 2005, 4(2).

[3] Buades A , Coll B , Morel J M . Non-Local Means Denoising[J]. Image Processing On Line, 2011, 1.

[4] NVIDIA. (2020). CUDA c programming guide(v10.2). NVIDIA Corporation. 120.

[5] 李瑶,孙涛,黄驰,张子健.序列图像的非局部均值超分辨率重建算法及 GPU 实现 [J]. 计 算 机 应 用 研 究 ,2016, 33(07):2201-2205.

A CUDA implementation of Non-local Means Image Denoising Method

Abstract Non-local means filtering is an image denoising method that can effectively remove image noise while maintaining the sharpness of the image edges. However, this algorithm has a high time complexity and requires a lot of time when processing images. This makes the method difficult to put into practice. Therefore, it is necessary to improve the processing speed of the non-local means algorithm. This paper uses CUDA to realize the parallelization of non-local means image denoising method. The results show that, while maintaining the same image processing quality, the processing time after parallelization is significantly shortened.

Key Words CUDA, GPU, Non-local Means, Image Denoising