适用于高分辨率图像的 ASIFT 算法的 GPU 实现

摘要

随着技术进步，图像处理的需求日益增长，而特征匹配作为图像处理的核心环节，在多种应用中扮演着至关重要的角色。然而，针对高分辨率图像的特征匹配算法在计算上的复杂性，通常阻碍了它们的实时处理能力。为了解决这一问题，本文采用 CUDA 框架，对 ASIFT 算法实施了五种优化策略。本文选取高分辨率卫星遥感图像作为测试数据，旨在评估优化后算法的性能。实验结果显示，尽管这五种优化策略相比于 ASIFT 算法的原始实现在成功匹配的特征点对数量上有所下降，但它们在运行效率上均实现了显著提升，达到了最高约 131 倍的加速比。

关键词：图像匹配；高性能计算（HPC）；GPU

1 引言

尺度不变特征变换（Scale-Invariant Feature Transform，SIFT）[8] 是一种基于局部特征描述子的图像特征提取算法。该算法分别在 1999 和 2004 年提出和发扬光大，成为计算机视觉领域中广泛应用的图像识别和目标检测算法之一。尽管 SIFT 在尺度缩放下表现出色，但在仿射变换下的性能表现不佳。

为解决这一局限性，仿射尺度不变特征变换（ASIFT）算法 [9] 于 2009 年被提出，ASIFT 通过模拟由正面位置的相机光轴方向变化引起的所有可能的仿射失真来转换每个图像，这些失真取决于经度角 和纬度角 ，其具体的转换方式是图像首先经过 的旋转，然后进行倾斜，由于倾斜是对于数字图像进行的，因此在进行倾斜前需要在 x 方向使用抗混叠滤波器。通过以上流程获取的所有图像，需要使用 SIFT 进行特征提取 [9]。ASIFT 显示了完全的仿射不变性，提供了更精确的特征匹配。尽管 ASIFT 算法在特征匹配方面取得了显著的优势，但其算法流程中包含大量卷积、滤波等操作，计算复杂度较高，这导致其在处理大规模高分辨率图像时面临着显著的实时性挑战。为了克服这些挑战，研究者们提出了多种优化策略，如基于 CPU 的多线程计算、使用更高效的描述子替代 SIFT [2] 等。尽管这些优化方法有所帮助，但在加速方面仍有限制。

近年来，图形处理器（GPU）并行计算技术在计算机视觉和图像处理领域取得了显著进展。特别是对于计算密集型的任务，基于 GPU 的并行计算可以大幅提高算法的处理效率。基于 CUDA 框架的 GPU 并行计算技术为 ASIFT 算法提供了一条高效的优化路径。实际上，已有研究显示，使用单个 GPU 实现的 ASIFT 算法在高分辨率图像上可以获得近 70 倍的加速比 [4]。

本文致力于研究和实现基于 GPU 并行计算的 ASIFT 算法优化，旨在为处理高分辨率图像的特征匹配提供一种高效且实时的解决方案。我们希望这项工作能为图像处理领域的其他任务提供并行计算优化的宝贵参考。

2 相关工作

对于图像的特征提取，一般来说有两个步骤：特征检测和特征匹配。图像中的特征点将在特征检测阶段生成，随后将构造描述子并将其与每个特征相关联，这些描述子通常以向量的形式出现，并将被用于特征匹配阶段。算法效率的优化通常在特征检测阶段进行。

2.1 基于非并行计算的优化方案

对于 ASIFT ，优化策略之一是在特征检测阶段用 SURF 等效率更高的检测器替换 SIFT ，这可以达到 3 倍左右的加速比 [11]。但是 SURF 的精度低于 SIFT [10]，在对于精度要求较高的场景下难以满足图像匹配的需求。另一种优化策略是简化 ASIFT ，但其仅对某些特定图像有效 [5]。此外，也有基于硬件对 SIFT 算法进行优化的方案 [12]。总的来说，这些优化策略的加速比都比较低，在图像规模较大的情况下很难得到可接受的结果。

2.2 基于并行计算的优化方案

使用并行计算优化 ASIFT 是另一种可行的方法，这种方案在实现较高加速比的同时，又可以保持接近 ASIFT 原始实现的精度。ASIFT 的主要流程包括：倾斜、旋转、高斯卷积以及 SIFT，事实上，这四个部分占据了 ASIFT 运行时长的 97% [4]。并行计算包括中央处理器（CPU）多线程计算和 GPU 多线程计算。

2.2.1 基于 CPU 的并行计算优化方案

使用 OpenMP 进行基于 CPU 的并行计算来加速 ASIFT 是一种易于实现的优化，也可以与原始的 ASIFT 实现保持相同的精度。其要点在于使用大量线程将通过经度角和纬度角对图像进行视点转换的过程并行化（事实上，在实践中通常也包括对转换后的图像使用 SIFT 进行特征提取的部分）。该方案的问题一方面在于通过纬度角和经度角进行转换后的图像数量往往远大于 CPU 的核心数量。另一方面，该方案实际上并没有对上述 ASIFT 的四个主要部分进行并行计算优化。在这些因素的共同作用下，单独使用 OpenMP 对 ASIFT 进行并行计算优化难以达到较高的加速比，因此其常与其他优化方案相结合。

2.2.2 基于 CPU 的并行计算优化方案

使用 GPU 进行并行计算是更为有效的方法。一般来说，图形处理单元上的通用计算（GPGPU）是利用处理图形任务的 GPU 来计算原本 CPU 处理的通用计算任务，这些通用计算任务通常来说与图形处理没有任何关系，由于现代 GPU 具有的强大并行处理能力和可编程流水行，令图形处理器也可以处理非图形数据，特别是在面对单指令流多数据流（SIMD）且数据处理的运算量远大于数据调度和传输的需要时，通用图形处理器在性能上大大超越了传统的中央处理器。目前，常用的 GPGPU 框架包括 OpenCL 和 CUDA 等，总的来说，CUDA

框架与 NVIDIA 的硬件结合的更好，具有良好的易用性和较高的效率，因而被广泛使用，目前已有的针对 ASIFT 的基于 GPU 的并行计算优化方案也多基于 CUDA 框架开展。

GPU-ASIFT [4] 使用 CUDA 框架对 ASIFT 进行并行计算优化，在使用单 GPU 的情况下可在高分辨率图像上达到近 70 倍的加速比。该优化方案将 ASIFT 中的图像变换部分（上述倾斜、旋转、高斯卷积部分）迁移至 GPU，即实现了完成这些图像变换的核函数，并使用 SIFT 的一个 GPU 实现——siftGPU 替换了 SIFT ，从而完成了 ASIFT 的并行优化。GPU-ASIFT 的优化思路具有极强的可扩展性，按照该方案，SIFT 实际上可替换为其任意的 GPU 实现，图像变换部分也可以使用任意等价的模块替代而无需局限于自行实现核函数。

另一个基于 CUDA 框架开展的针对 ASIFT 的并行计算优化方案是由何婷婷等人在 2014 年提出的 GASIFT [13]，该方案的整体思路与 GPU-ASIFT 类似，同样在 GPU 端实现了 ASIFT 中的图像变换部分，但在 SIFT 部分却没有采用现有的 GPU 实现，而是自行提出了一个使用 GPU 并行计算加速的 SIFT 实现，而该实现由于引入了显存池的概念，也成为该优化方案中的一大亮点。内存池（Memory Pool），又被称为固定大小区块规划（fixed-size-blocks allocation），允许程序进行动态的存储器规划（内存分配），对于常见的动态内存分配方案，由于会产生因变动存储器区块大小导致的碎片问题，导致其性能受限。而内存池就这一点提供了更有效率的解决方案：预先规划一定数量的存储器区块，使得整个程序可以在运行时分配、使用和释放存储器区块。在 GASIFT 的实现中，显存池主要用来避免 SIFT 在运行过程中对于显存空间频繁的申请和释放，即 GASIFT 在运行过程中只需对显存池进行一次初始化（这意味着申请一块很大的显存），随后需要的所有显存都从显存池中进行分配。同时 GASIFT 还使用了共享内存、合并访存等特性提高了数据访问效率。最终，该优化方案相比原始 ASIFT 达成了约 16 倍的加速比，相比 OpenMP 优化后的 ASIFT 达成了约 7 倍的加速比。

同时，SIFT 作为 ASIFT 的一部分，由 GPU-ASIFT 可以获得启发: 所有针对 SIFT 的优化方案都可以直接应用在 ASIFT 中，因此 SIFT 的 GPU 并行计算优化方案也是值得关注的。GPU-ASIFT 中使用的 siftGPU 就是 SIFT 的一种 GPU 实现，其代码已经开源，但该方案提出时间较早，且后续缺乏改进，因此性能相比其他实现并不占优。CudaSift [3]，其代码同样已经开源，该实现经过多次迭代，具有良好的易用性和极高的加速比，但其精度相比原始的 SIFT 实现有所降低。HartSift 是一个针对 SIFT 的改进方案 [7]，该实现较之 siftGPU 具有 4.01 至 6.49 倍的加速比，与 CudaSift 性能相当，但精度远高于 CudaSift，遗憾的是，该实现并没有开源。上述三种 SIFT 的 GPU 实现都基于 CUDA 框架。

3 本文方法

3.1 本文方法概述

本文首先对 640×480 像素的图像进行了原始 CPU 实现的 ASIFT 特征检测分析。图1揭示了 ASIFT 算法中几个核心部分的时间占用情况：倾斜处理、旋转调整、高斯滤波以及 SIFT 特征检测这几个步骤占据了算法执行时间的大部分，而其余部分的时间消耗则不足总时间的 5%。根据阿姆达尔定律 [1]，为了提高整体性能，优化每个主要过程至关重要。

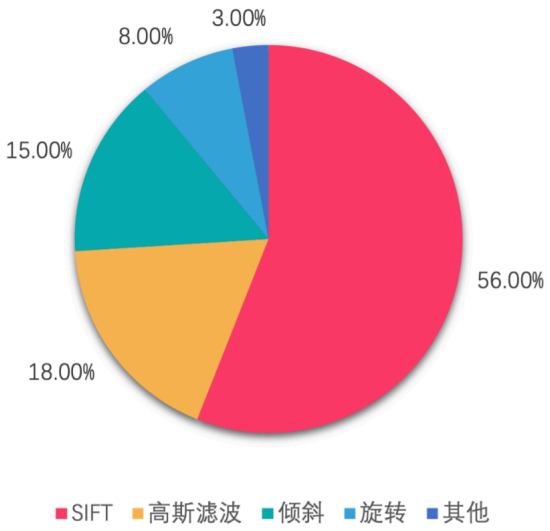


图 1. ASIFT 算法运行时间分析

优化后的算法分为四个步骤:

1. 对原始图像执行自适应图像分块算法。然后将图像（或分块后的图像）复制到显存中。
2. 如果使用 OpenMP，其将创建数个线程，每个线程将对图像执行一系列图像变换。如果没有使用 OpenMP，则图像变换操作将在单个线程上运行。
3. 在 GPU 上执行旋转、高斯模糊和倾斜操作，然后使用 CudaSift 进行特征提取。
4. 使用 GPU 优化的 Brute-Force 方法，将获取的特征点和描述子数据进行匹配。

优化后的基本流程如图2所示。

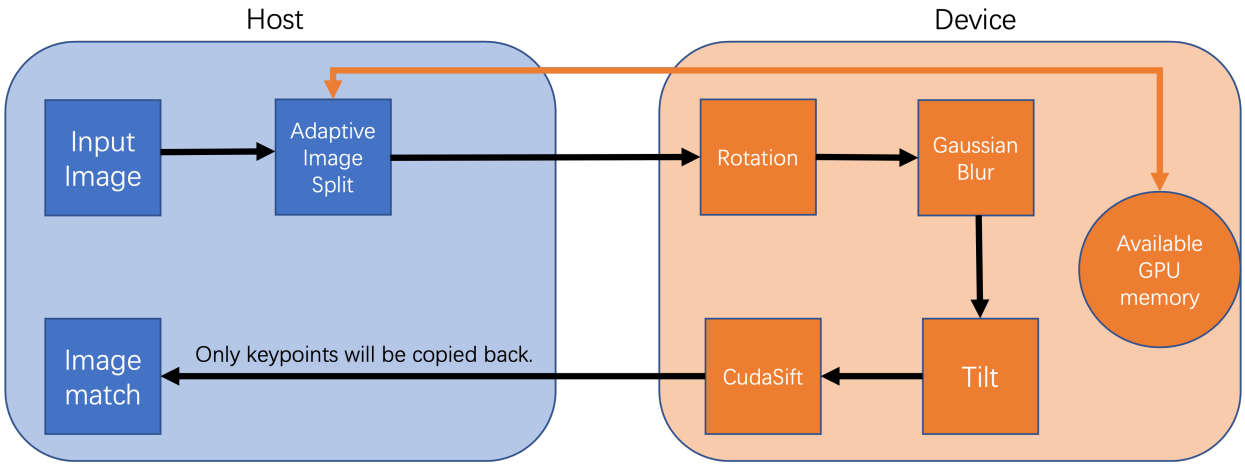


图 2. 基本流程

根据上述阶段, 本文共进行了如表1所示五种不同优化策略的对比。

优化策略	解释
ASIFT(原始)	原始 ASIFT 实现
CUDASIFT	将原始 ASIFT 实现中的 SIFT 特征提取部分替换为 CudaSift
OPENMP_CUDASIFT	在 CUDASIFT 优化策略的基础上，启用 OpenMP
RTG_CUDASIFT	在 CUDASIFT 优化策略的基础上， 将原始 ASIFT 实现中的图像变换部分替换为自行实现的核函数
OPENMP_RTG_CUDASIFT	在 RTG_CUDASIFT 优化策略的基础上，启用 OpenMP
NPP_CUDASIFT	在 CUDASIFT 优化策略的基础上， 将原始 ASIFT 实现中的图像变换部分替换为 Nvidia Performance Primitive 库中相应的函数

表 1. ASIFT 优化策略

3.2 用于 GPU 并行计算的自适应图像分块

面对大尺寸的卫星图像，显存大小的限制常常是 GPU 处理中的一个关键问题。受限于显存的大小，GPU 可能无法一次性处理全尺寸的大图像，降低图像分辨率似乎是一个直接的解决方案。然而，这种做法往往会导致图像失去部分细节信息，影响最终处理的效果。

为了有效解决这一挑战，本文提出了一种自适应图像分块算法。如图3所示，该算法能够根据显存的大小和图像的尺寸自动对图像进行分割。这种方法的优势在于，只需将分块后的数据传输到 GPU 进行处理，处理完成后再将各个块的数据重新组合成完整图像。通过这种方式，算法既克服了显存大小的限制，又保持了图像的完整性和细节。此外，这种分块方法能够更高效地利用 GPU 的并行处理能力，进一步提高算法的整体性能和处理速度。

本文发现算法运行时所需的显存可以根据图像大小计算，图像变换和 CudaSift 所需的显存大约是存储图像所需显存的倍数，如公式 (1) 所示，根据本文对基于 GPU 的图像变换实现和 CudaSift 的 GPU 实现，发现这个倍数大约是 5 倍。

$$\text{RequireMem} = \frac{\text{ImageSize}}{(\text{RowCut}+1)(\text{ColCut}+1)} \times 5 \tag{1}$$

在公式 (1) 中，变量 *RowCut* 和 *ColCut* 表示原图沿水平方向和竖直方向被分割的次数。

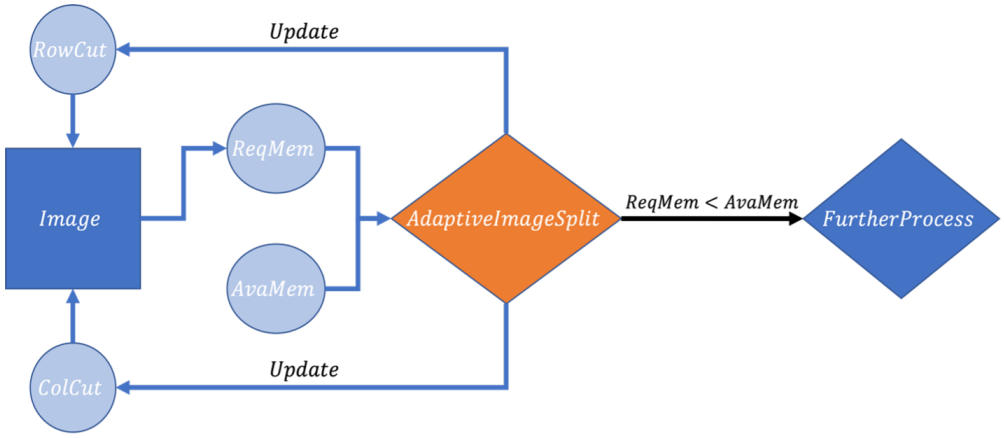


图 3. 图像分块过程

计算切割次数的伪代码如下所示, 变量 $AvailMem$ 表示可用的 GPU 内存，它可以通过 Cuda 内存管理函数 CudaMemGetInfo() 更新。进行分块后（如果需要），图像将分别传输到 GPU 内存做进一步处理。需要指出是，若启用自适应图像分块算法，则图像从 CPU 传输至 GPU 的过程可能会增加，具体的增量取决于图像被划分为几个部分。

Algorithm 1 Calculate RowCut and ColCut

Input: ImageSize, RequireMem, AvailMem

Output: RowCut, ColCut

Initialize: $RowCut = 0$, $ColCut = 0$
Update AvailMem
Update RequireMem based on Equation (1)
while $RequireMem > AvailMem$ **do**
    if $RowCut = ColCut$ **then**
        $RowCut \leftarrow RowCut + 1$
    else
        $ColCut \leftarrow ColCut + 1$
    end if
    Update AvailMem
    Update RequireMem based on Equation
end while

3.3 基于 GPU 的图像变换实现

图像变换过程包括三个主要步骤：旋转、高斯模糊和倾斜操作，这些步骤都旨在模拟仿射失真。

首先，内存和显存之间的数据传输是 GPU 实现中必须考虑的重要问题。因为这两个平台之间的数据传输通常需要消耗大量时间，因此减少数据传输次数是提高效率的关键。其次，显存空间的分配也是一个值得关注的问题，这个过程同样可能耗费大量时间。在 CUDA 中，显存分配既可以在 CPU 端完成，也可以在 GPU 上（通过核函数）完成。当在 CPU 端分配显存时，通常一次性分配所需的全部显存。而在 GPU 上，显存通常被分配用于临时存储信息，这种情况下所需的显存量通常是不确定的，并且显存的分配在 GPU 上相比 CPU 需要更多的时间。

针对这些问题，本文提出的流程将图像数据从 CPU 传输至 GPU 的过程减少到仅一次（在不分块的情况下）。考虑到图像变换的核函数必须有输入和输出（输入是原始图像，输出是处理后的图像），且通常输入和输出图像大小不同，它们会被存储在不同的内存地址中。这意味着在内核初始化之前，应为输出图像所需的内存分配空间。通过这种策略，图像可以始终存储在显存中，从而减少了不同平台之间的数据复制过程。实际上，尽管核函数所需的临时内存大小不确定，但仍然可以在初始化之前为其分配足够的空间，因为核函数运行时所需的临时内存相对较小，可以预先计算出所需的最大值。即在核函数启动之前，按照所需的最大值分配显存。

除了自行实现的核函数外，本文也利用了高性能 CUDA 库——Nvidia Performance Primitive（NVIDIA NPP），这是一个提供 GPU 加速的图像、视频和信号处理函数的库。该库相比纯 CPU 实现具有显著的加速比，大约有 30 倍，且易于使用。它已经在计算机视觉、工业检测、机器人技术、医学成像、电信和深度学习等需要高性能计算的领域得到了广泛应用。本文同样基于 NPP 库实现了图像变换部分，并将在结果分析部分展示与自行实现的核函数的性能对比。

3.4 基于 GPU 的特征提取

在图像变换处理完成后，接下来的步骤是对图像进行 SIFT 特征提取。在这一阶段，本文选择了使用 CudaSift 库，并对其进行了必要的修改，以便更好地融入整体的算法策略。

CudaSift 原始代码中的一个重要特征是是它在 GPU 端使用了常量内存。常量内存是一种特殊的只读内存区域，位于 GPU 端，尽管其容量较小（通常为 64KB），却能提供高效的访问速度。这意味着，当多个线程同时访问同一个常量内存位置时，性能会得到优化。这种内存区域通常用于存储在核函数执行期间不会更改的数据，例如预定义的常量或参数。CudaSift 通过这种方式提高了其运行效率。

然而，这也带来了一个问题：CudaSift 本身并不支持并行化执行。当将本文提出的优化策略与 OpenMP 同时使用时，CudaSift 部分会产生大量的访存错误。这些错误是由于多个线程同时访问同一常量内存位置时产生的竞争条件（race condition）导致的。

为了解决这个问题，本文对 CudaSift 进行了一些修改，将使用常量内存的变量转换为使用全局内存的局部变量。这样做虽然消除了访存错误，但相应的代价是降低了程序的运行效率。这是因为全局内存的访问速度相比常量内存来说较慢，且更容易出现竞争条件。在 SIFT 特征提取完成后，将生成一系列特征点和描述子数据。这些数据将用于后续的特征点匹配过程。尽管修改 CudaSift 可能会对性能产生一定影响，但它确保了整个流程的顺畅执行和并行化优化的实现。

3.5 特征点匹配

在图像处理和计算机视觉领域，特征点匹配算法扮演着至关重要的角色。它的主要任务是将从不同图像中提取的特征点进行匹配，以确定图像之间的对应关系。

在常见的特征点匹配方法中，Brute-Force（暴力匹配）方法是最直接的一种。这种方法通过将一个图像中的每个特征点与另一个图像中的所有特征点进行逐一比较，计算它们之间的特征描述符距离（例如欧几里得距离或汉明距离），并选择距离最近的特征点作为匹配点。尽管 Brute-Force 方法直观且易于实现，但它的计算复杂度较高，可能不适合实时应用场景。另一种常用的方法是使用 FLANN（Fast Library for Approximate Nearest Neighbors），这是一个专门用于近似最近邻搜索的快速库。FLANN 能够自动选择最合适的算法（如 K-D 树或其他方法）来执行特征匹配。在处理大型数据集时，FLANN 显示出了良好的性能和准确性，因此在多个计算机视觉应用中得到了广泛的使用。

本文在特征点匹配方面并未进行深入研究，但实验了 OpenCV 库中的 FLANN 方法和使用 GPU 优化的 Brute-Force 方法。结果表明，GPU 优化的 Brute-Force 方法在运行效率上优于 FLANN 方法，因此被选为本文的主要特征点匹配方法。

在特征点匹配完成后，还需应用随机抽样一致性（Random Sample Consensus，RANSAC）方法来剔除错误的匹配对。RANSAC 是一种广泛使用的参数估计方法，特别是在目标检测和特征匹配等计算机视觉问题中。其核心原理是从数据集中随机选取一部分样本进行模型估计，然后使用剩余的样本来判断模型的内外点，最终确定最佳模型参数。RANSAC 适合处理存在噪声和异常点的数据集，有效地减少这些异常点和噪声对模型参数估计结果的干扰

4 复现细节

4.1 与已有开源代码对比

在 3.1 所提到的 4 个步骤中:

- 步骤 1 中自适应图像分块技术为本文原创设计并实现。
- 步骤 2 中使用的OpenMP是一个跨平台的多线程实现，主线程 (顺序的执行指令) 生成一系列的子线程，并将任务划分给这些子线程进行执行。这些子线程并行的运行，由运行时环境将线程分配给不同的处理器。在 C++ 中使用时直接在开头声明引用即可。
- 步骤 3 中 GPU 上的图像变换操作为本文自行实现，在进行不同优化策略对比时，NPP_CUDASIFT 使用了Nvidia NPP库中相应的函数替换了自行实现的图像变换部分和相关内存拷贝部分，以评估不同实现策略的性能。使用的CudaSift是在 GitHub 上已开源的项目，本文直接进行使用，并在其基础上将使用常量内存的变量全部转换为使用全局内存的局部变量，解决了并行执行时的访存错误问题。
- 步骤 4 特征点匹配中，本文直接调用了 OpenCV 提供的使用 GPU 优化的 Brute-Force 方法。

最后，本文在进行优化效果对比时，原始 ASIFT 使用了原论文提供的代码 [9]，同时在编写其他优化策略时，图像变换 (除修改部分外) 也直接使用了 SIFT 中的代码。

4.2 实验环境搭建

实验基于 WHU-TLC 数据集 [6] 的图像 (5120x5120 像素) 进行，该数据集是面向卫星域 MVS (Multi-View Stereo) 任务的公开数据集，由三视图卫星图像、RPC 参数和地面真值 dsm 组成，三视图图像是由安装在资源 3 号 (ZY-3) 卫星上的 TLC 相机收集的，这三幅图像几乎是同时拍摄的，没有受到光照和季节变化的影响。

实验将倾斜参数 t 的最大值设定为 8，这是一个相当大的数值，可以提取更多的特征点，与针对卫星影像的特征提取的真实场景更为接近。

测试所用设备参数如表2所示。

CPU	Intel® Core™ i5-11400F Processor
内存	32GB
GPU	GEFORCE GTX 1080 Ti

表 2. 实验用设备参数

4.3 界面分析与使用说明

```
ASIFT_CUDA [detectOption] [image1] [image2] [resultPath]
    [dectectOption]      -origin                 original asift
                         -openmp                 original asift with openmp
                         -cudasift               using cudasift to replace
                                                 sift in original asift
                         -openmp_cudasift        using cudasift and enable
                                                 openmp(image maybe splited
                                                 due to insufficient of gpu
                                                 memory)
                         -rtg                    running image rotate, tilt
                                                 and gaussian blur on gpu
                         -rtg_cudasift           running image processing on
                                                 gpu and using cudasift
                         -openmp_rtg_cudasift    running image processing on
                                                 gpu, using cudasift and
                                                 enable openmp(image may be
                                                 splited due to insufficient
                                                 of gpu memory)
                         -npp_cudasift           running image processing on
                                                 gpu using Nvidia 2D Image
                                                 And Signal Performance
                                                 Primitives and using
                                                 cudasift
```

图 4. 使用说明

4.4 创新点

本文构建了一个自适应图像分块算法，这一算法能够根据硬件条件，将大尺度图像分割成多个小尺度的图像块。对于每个小尺度图像块，算法执行特征提取，并将提取出的特征点坐标恢复至原图像的相应位置。这一策略使得基于 GPU 的改进后的 ASIFT 算法能够在几乎所有平台上针对任意大小的图像进行有效运行，从而大大提高了算法的适用性和灵活性。

此外，本文在 GPU 上自行实现了图像变换操作。通过优化数据传输流程，将图像从 CPU 传输到 GPU 的过程减少到最少（在不进行图像分块的情况下只需要一次传输），这显著减少了数据传输时间，提高了整体处理效率。同时，本文结合了 OpenMP 和 Nvidia NPP 提供的函数，执行了多种优化策略的性能对比，以确定最佳的处理方法。

在使用 CudaSift 进行特征点提取时，本文也做出了关键的修改。为了支持并行化执行，本文牺牲了部分运行效率，改变了原有的内存使用方式，从而解决了并行执行中的访存错误问题。这一修改虽然降低了一定的运行效率，但提高了算法的并行化能力和适用性。

总的来说，本文的研究通过综合应用多种优化技术和策略，显著提升了基于 GPU 的 ASIFT 算法的实用性和效率，为处理大尺寸高分辨率图像提供了一种有效的解决方案。

5 实验结果分析

本文的实验部分主要集中在图像变换和特征提取两个关键环节的优化上。表3中的耗时不包含特征点匹配部分的耗时。由表3可以看出，与原始 ASIFT 算法相比，本文提出的所有优化策略都实现了运行效率提升。

优化策略	平均耗时 (s)	最大耗时 (s)	最小耗时 (s)	加速比
ASIFT(原始)	533.36	570.02	496.56	1.00
CUDASIFT	239.93	247.55	238.20	2.22
OPENMP_CUDASIFT	56.78	72.40	51.17	9.39
RTG_CUDASIFT	21.84	22.77	21.14	24.42
OPENMP_RTG_CUDASIFT	36.38	39.66	34.89	14.66
NPP_CUDASIFT	4.06	5.09	3.29	131.37

表 3. 不同优化策略的运行效率对比（加速比基于平均耗时计算）

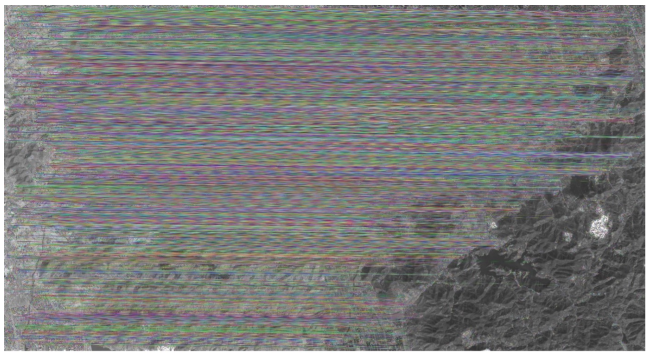
本文提出的优化策略中，使用 GPU 并行计算优化的部分主要是图像变换和特征提取部分。在特征提取部分，OPENMP_CUDASIFT 相对于 CUDASIFT 的加速比为 4.23，这表明在仅优化特征提取算法的情况下，启用 OpenMP 可以获得显著的效率提升；但在对图像变换部分做优化后，情况发生了意料之外的变化，OPENMP_RTG_CUDASIFT 相对于 ORIGIN 的加速比较之 RTG_CUDASIFT 的加速比反而出现了剧烈的下降，一个合理的解释是：这两种优化策略之间最显著的区别在于前者启用了自适应图像分块算法，这就意味着图像数据从 CPU 拷贝至 GPU 的过程要进行多次 (本文的实验中是 4 次)，而该过程如前所述，效率较低。但也应当注意到，RTG_CUDASIFT 相对于 CUDASIFT 的加速比为 10.99，这表明对于图像变换部分的 GPU 并行计算优化策略是有效的。在所有的优化策略中，NPP_CUDASIFT 取得了高达 131.37 的加速比，远高于其余所有优化策略，表明 NPP 库中的图像处理函数普遍具有高效的实现，在常见的使用场景下，使用库函数较之自行实现的核函数有可能取得更高的加速比。

对于结果准确性的评估聚焦于成功匹配的特征点对的数量，图5展示了不同优化策略对采用 WHU-TLC 数据集中 index0_idx7 部分的前两张图像进行特征提取和匹配的结果，两张图像被拼接在一张图中，特征点对将被连接在一起。

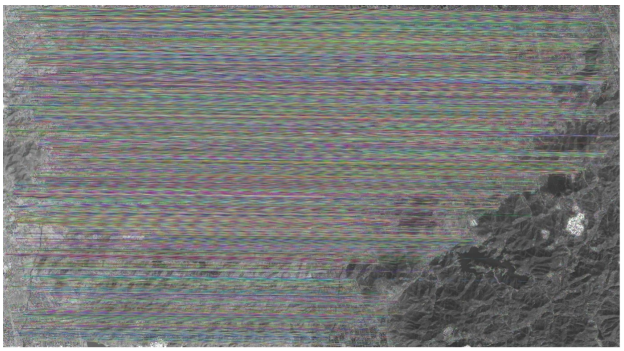
优化策略	成功匹配的特征点对平均数
ASIFT	383630
CUDASIFT	71244
OPENMP_CUDASIFT	71219
RTG_CUDASIFT	41486
OPENMP_RTG_CUDASIFT	67468
NPP_CUDASIFT	41016

表 4. 不同优化策略的结果准确性对比

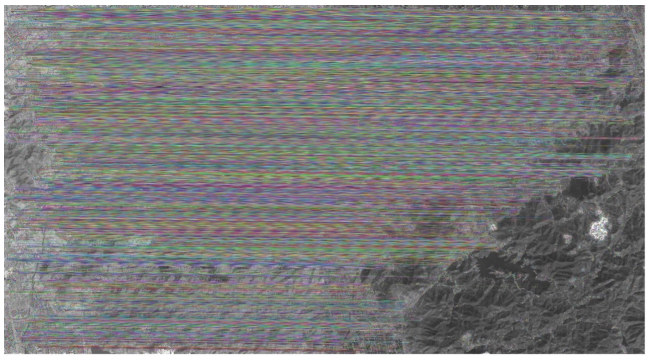
同时, 我们对所有优化策略在成功匹配的特征点对数量进行统计形成表4, 可以看出, 本文提出的所有优化策略在成功匹配的特征点对数量上都出现了不同程度的下降。CUDASIFT 相比 ASIFT 在该项数据上出现了大幅度的下降，RTG_CUDASIFT 和 NPP_CUDASIFT 相比 CUDASIFT 在该项数据上也出现了小幅度的下降，由此推测, 当采用 GPU 优化的部分越多，可获取的成功匹配的特征点对越少。OPENMP_RTG_CUDASIFT 相比 RTG_CUDASIFT 在该项数据上反而出现了小幅度的提升，这意味着将规模较大的图像划分为若干部分后分别进行特征提取可以获取更多的数据。



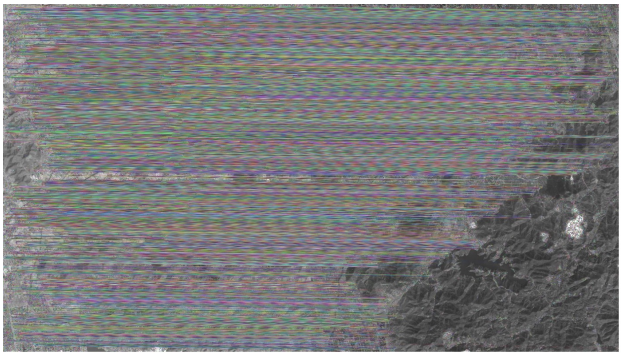
(a) CUDASIFT



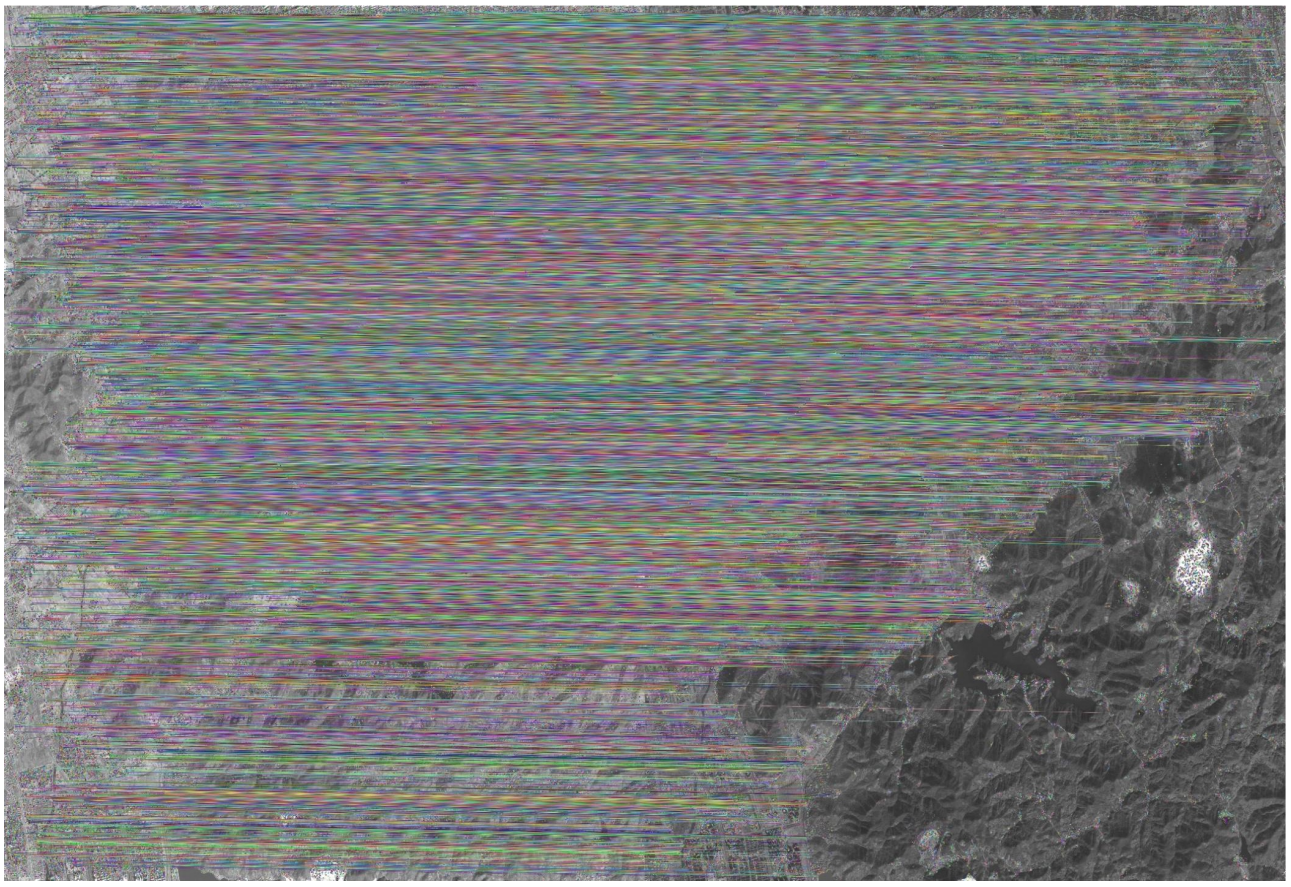
(b) OPENMP_CUDASIFT



(c) RTG_CUDASIFT



(d) OPENMP_RTG_CUDASIFT



(e) NPP_CUDASIFT

图 5. 结果展示

6 总结与展望

本文针对使用 GPU 并行计算优化 ASIFT 算法出了五种有效的策略，并将这些策略在 WHU-TLC 数据集上进行了测试。结果表明，本文提出的优化策略相比 ASIFT 算法的原始实现可取得最高约 131 倍的加速比，优于目前最快的实现。同时，本文还提出了一种自适应图像分块算法，可以根据当前可用的显存自动将大尺寸的图像分割为若干个尺寸较小的图像，该算法极大的提高了优化策略的适用性，使得显存较小的设备也可以处理大尺寸的图像。但本文提出的优化策略也伴随着一定程度的精度下降，使得成功匹配的特征点对数量降低。这种精度损失主要源于 GPU 和 CPU 在硬件设计、计算稳定性以及舍入误差等方面的显著差异，这些差异在进行并行计算时通常会导致精度（误差）问题。为了尽量减少这种影响，本文采用了 Kahan 求和算法。

尽管目前的优化策略已取得显著成效，但仍有进一步探索的空间。例如，将计算不同图像视图和计算 SIFT 特征的过程，并行化地迁移至 GPU 侧，这意味着需要在 GPU 启动的线程中启动更多的线程。CUDA 为计算能力在 3.5 以上的 GPU 设备提供了称为动态并行（Dynamic Parallelism）的扩展，使得在 GPU 线程中启动更多 GPU 线程理论上是可行的。此外，GPU 侧还需要一个动态的资源分配算法，以充分利用计算资源，这意味着图像的分割可能是非均匀的。实际上，这一过程也可以在多个 GPU 上执行，但目前针对 ASIFT 的多 GPU 优化尚未涉及此方面。当然，分块算法也有进一步优化的空间，实验中发现，分块越少，效率越高，这是因为分块越多，内存到显存的拷贝次数就越多（不分块时仅需拷贝一次）。因此，如何减少拷贝次数是进一步提高效率的关键。这些方向的探索可能会进一步提升算法的效率和适用性。

参考文献

[1] Gene M Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 483–485, 1967.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3):346–359, 2008.

[3] Mårten Björkman, Niklas Bergström, and Danica Kragic. Detecting, segmenting and tracking unknown objects using multi-label mrf inference. *Computer Vision and Image Understanding*, 118:111–127, 2014.

[4] Valeriu Codreanu, Feng Dong, Baoquan Liu, Jos BTM Roerdink, David Williams, Po Yang, and Burhan Yasar. Gpu-asift: A fast fully affine-invariant feature extraction algorithm. In *2013 International Conference on High Performance Computing & Simulation (HPCS)*, pages 474–481. IEEE, 2013.

[5] Han Fu, Donghai Xie, Ruofei Zhong, Yu Wu, and Qiong Wu. An improved asift algorithm for indoor panorama image matching. In *Ninth International Conference on Digital Image Processing (ICDIP 2017)*, volume 10420, pages 285–292. SPIE, 2017.

[6] Jian Gao, Jin Liu, and Shunping Ji. Rational polynomial camera model warping for deep learning based satellite multi-view stereo matching. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6148–6157, October 2021.

[7] Zhihao Li, Haipeng Jia, and Yunquan Zhang. Hartsift: A high-accuracy and real-time sift based on gpu. In *2017 IEEE 23rd International Conference on Parallel and Distributed Systems (ICPADS)*, pages 135–142. IEEE, 2017.

[8] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60:91–110, 2004.

[9] Jean-Michel Morel and Guoshen Yu. Asift: A new framework for fully affine invariant image comparison. *SIAM journal on imaging sciences*, 2(2):438–469, 2009.

[10] Payal Panchal et al. A comparison of sift and surf. *International Journal of Innovative Research in Computer and Communication Engineering*, 1:323–327, 2013.

[11] Yanwei Pang, Wei Li, Yuan Yuan, and Jing Pan. Fully affine invariant surf for image matching. *Neurocomputing*, 85:6–10, 2012.

[12] Joohyuk Yum, Chul-Hee Lee, Jinwoo Park, Jin-Sung Kim, and Hyuk-Jae Lee. A hardware architecture for the affine-invariant extension of sift. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(11):3251–3261, 2017.

[13] 何婷婷, 芮建武, and 温腊. Cpu-gpu 协同计算加速 asift 算法. *计算机科学*, 41(5):14–19, 2014.