CUDA 驱动的并行连通域标记

余笑轩
xiaoxuan_yu@pku.edu.cn
化学与分子工程学院

Jun 22, 2024

Abstract

Keywords 连通域标记 · 并行计算 · CUDA

1 连通域标记问题

连通域标记（Connected Component Labeling, CCL）是计算机视觉中的一种基本操作, 广泛应用于图像分割、目标检测和图像分析等领域。它的主要任务是识别和标记图像中相连的像素块, 即连通域。连通域标记在图像处理、模式识别和计算机视觉的许多应用中起着关键作用。

在图像中，连通域是指所有像素值相同且通过某种连通性准则（如 4-连通或 8-连通，如 Figure 1 所示）相连的区域。连通域标记算法的目标是为每个连通域分配一个唯一的标签, 以便后续的图像处理和分析工作。具体而言，本次作业将实现对于二维二值图像的 8-连通域标记算法。

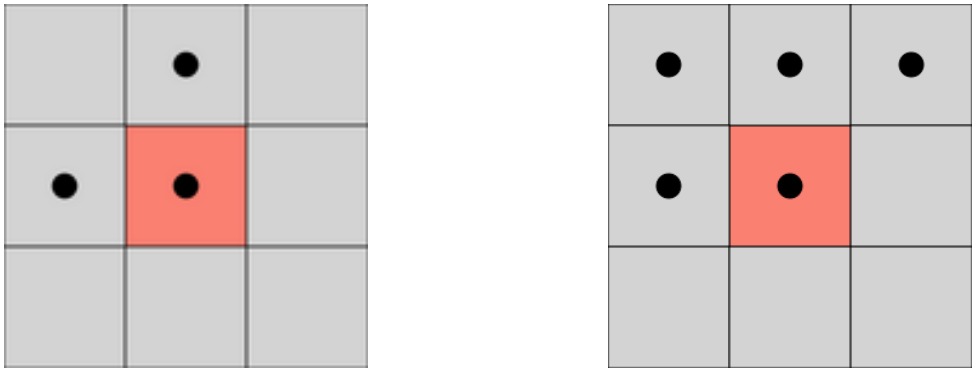

Figure 1: 4-连通性和 8-连通性的示意图 (Wikipedia contributors 2023)

2 算法

2.1 CCL 的串行算法: 并查集

基于并查集的串行算法是一种经典的连通域标记算法。并查集 (Union-Find) 是一种常用的数据结构, 能够高效地处理连通域标记问题。并查集主要包含两个操作: 查找 (Find) 和合并(Union), 如 Figure 2 所示。

- 查找: 确定某个元素属于哪个连通域。
- 合并: 将两个连通域合并为一个。

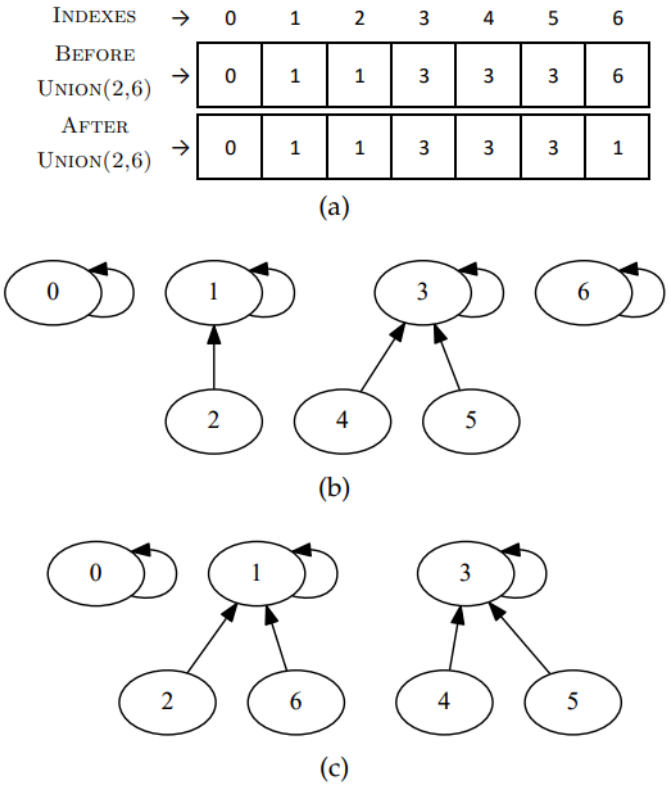

Figure 2: 并查集示意图 (Allegretti, Bolelli, and Grana 2020)

基于并查集的算法通过逐像素扫描图像, 使用并查集记录和合并相邻像素的连通信息, 从而实现连通域标记。具体步骤如下:

1. 初始化并查集，每个像素作为一个独立的集合。
2. 逐像素扫描图像，对每个像素检查其上方和左方像素的连通情况，进行合并操作。
3. 第二次扫描图像，对每个像素进行查找操作，确定其最终的连通域标签。

2.2 GPU 并行的 CCL: Komura Equivalence 算法

对于并查集的并行化并不是显而易见的。传统的并查集算法是基于串行处理的, 直接并行化会面临许多挑战, 尤其是在处理等价类合并时, 需要解决多个线程之间的同步和冲突问题。为了有效地在 GPU 上实现并行的连通域标记, 研究者们提出了多种改进方案, KE(Komura-Equivalence) 算法就是其中之一。KE 算法(Komura 2015) 通过一系列步骤来实现高效的 GPU 并行连通域标记, 其过程如 Figure 3 所示:

- 初始化: 为每个像素分配一个唯一的初始标签，通常使用其线性索引值。
- 等价类更新: 在此步骤中, 多个 GPU 线程并行处理像素, 检查每个像素与其相邻像素的连通性, 并更新等价类信息。这一步骤通常需要多次迭代, 直到所有像素的标签稳定下来。
- 标签压缩: 使用路径压缩技术对等价类进行压缩，确保所有等价像素的标签一致。这一步骤通过在并查集的“查找”操作中进行路径压缩来实现。
- 标签传播: 将最终标签传播到所有连通像素，确保每个连通域的所有像素共享相同的标签。

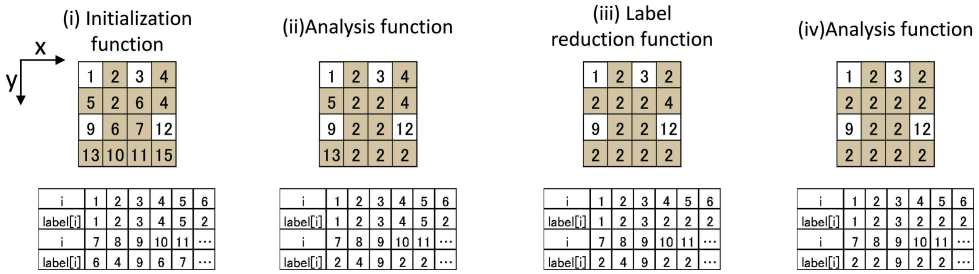

Figure 3: KE 算法示意图

KE 算法通过分阶段处理和并行化技术, 有效地克服了传统并查集在 GPU 上的并行化困难, 提高了连通域标记的效率。2018 年, Allegretti 等人给出了 8 连通的 KE 算法 (Allegretti et al. 2018), 该算法主要对 Figure 3 中的 reduction 部分进行了改进, 如 Figure 4 所示。

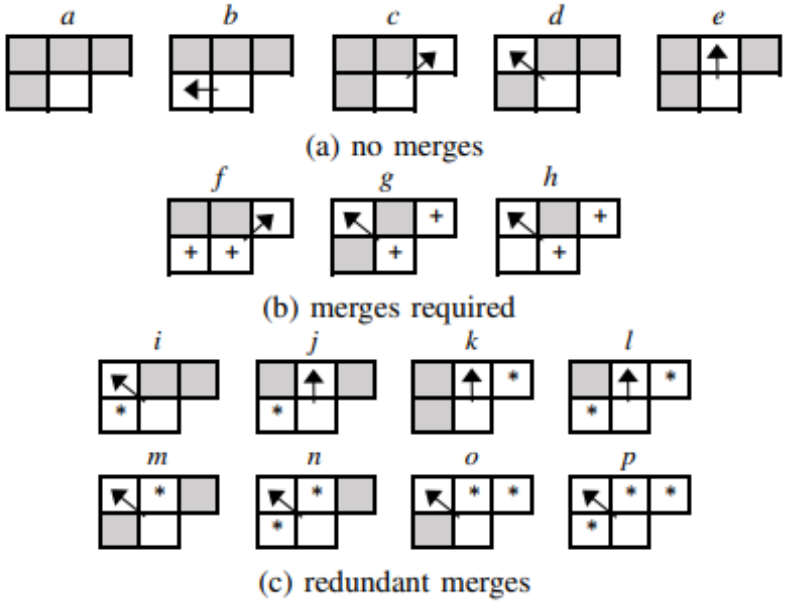

Figure 4: 8-连通的 KE 算法示意图

3 实现

我们使用 CUDA 实现了 KE 算法, 并对其进行了性能测试, 参见 algorithm/ 文件夹下的相关代码。

```
.
├── CMakeLists.txt
└── src
    ├── alg
    │   ├── KE.cu
    │   └── UFTree.cuh
    ├── CMakeLists.txt
    ├── include
    │   ├── alg_runner.cuh
    │   ├── Matrix.cuh
    │   └── timer.cxx
    ├── main.cu
    └── serial.cxx
```

4 结果与讨论

4.1 正确性验证

以作业中给定的串行程序作为参考，我们对并行程序的结果进行了验证。在正确性验证中，一个难以处理的问题是即使结果正确，标签的顺序和值也都可能不同。因而，我们实现了一个映射算法，将并行程序的标签映射到串行程序的标签，从而验证两者的结果是否一致。映射算法的实现非常平凡，我们将每个标签对应的像素坐标全部记录并进行匹配，从而得到一个标签之间的映射表。根据这个映射表执行映射后，我们可以直接比较两个 `label` 数组是否一致从而验证正确性。正确性验证的相关代码参见 `validation/val.py` 和 `validation/validation.ipynb`。此处给出核心功能函数的实现。

```
def get_match_dict(labels_a, labels_b):
    uni_labels_a = np.unique(labels_a)
    uni_labels_b = np.unique(labels_b)
    labels_a_dict = {label: [] for label in uni_labels_a}
    labels_b_dict = {label: [] for label in uni_labels_b}
    for i in range(labels_a.shape[0]):
        for j in range(labels_a.shape[1]):
            labels_a_dict[labels_a[i][j]].append(i * labels_a.shape[1] + j)
    for i in range(labels_b.shape[0]):
        for j in range(labels_b.shape[1]):
            labels_b_dict[labels_b[i][j]].append(i * labels_b.shape[1] + j)
    match_dict = {}
    for label in uni_labels_a:
        label_index_list = labels_a_dict[label]
        for b_label in uni_labels_b:
            b_label_index_list = labels_b_dict[b_label]
            if len(label_index_list) != len(b_label_index_list):
                continue
            if all(
                [
                    label_index_list[i] == b_label_index_list[i]
                    for i in range(len(label_index_list))
                ]
            ):
                match_dict[label] = b_label
                break
    return match_dict


def map_with_dict(labels, match_dict):
    labels = labels.copy()
    for i in range(labels.shape[0]):
        for j in range(labels.shape[1]):
            labels[i][j] = match_dict[labels[i][j]]
    return labels


def map_label(labels_a, labels_b):
    match_dict = get_match_dict(labels_a, labels_b)
    labels_a = map_with_dict(labels_a, match_dict)
    return np.all(labels_a == labels_b), labels_a
```

4.2 性能测试

4.3 计算速度的分析与讨论

5 总结

Bibliography

[1] Wikipedia contributors, “Connected-component labeling — Wikipedia, The Free Encyclopedia.” [Online]. Available: https://en.wikipedia.org/w/index.php?title=Connected-component_labeling&oldid=1192036140

[2] S. Allegretti, F. Bolelli, and C. Grana, “Optimized Block-Based Algorithms to Label Connected Components on GPUs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 2, pp. 423–438, 2020, doi: 10.1109/TPDS.2019.2934683.

[3] Y. Komura, “GPU-based cluster-labeling algorithm without the use of conventional iteration: Application to the Swendsen–Wang multi-cluster spin flip algorithm,” *Computer Physics Communications*, vol. 194, pp. 54–58, 2015, doi: https://doi.org/10.1016/j.cpc.2015.04.015.

[4] S. Allegretti, F. Bolelli, M. Cancilla, and C. Grana, “Optimizing GPU-Based Connected Components Labeling Algorithms,” in *2018 IEEE International Conference on Image Processing, Applications and Systems (IPAS)*, 2018, pp. 175–180. doi: 10.1109/IPAS.2018.8708900.