作业 2: 基于 MPI 通信实现的卫星图像计算任务

龚舒凯 2022202790

1 数据集使用与计算任务

本作业采用的数据集是 Sentinel-2 MSI(Multispectral Instrument) 野火数据集，其中每张图像为多波段的卫星栅格数据，波段 1(沿海大气气溶胶)、2(蓝色)、3(绿色)、4(红色)、8(NIR, 近红外)、12(SWIR, 短波红外) 被合并为一个单一的 6 波段 GeoTIFF 文件，进一步切割成 512×512 分辨率的块。在本作业中，数据集位于 `dataset/chop_fire` 目录下。

对于野火数据集，我们考虑如下指标：对于每个像素点，我们需要计算其归一化灼烧指数 (NBR, Normalized Burned Ratio)，其计算公式为：

$$\text{NBR} = \frac{\text{NIR} - \text{SWIR}}{\text{NIR} + \text{SWIR}}$$

其中 NIR 和 SWIR 分别为近红外和短波红外波段的反射率。NBR 值介于-1 和 1 之间，如图1所示，NBR 值越接近 1，表示植被越健康；越接近-1，表示植被受损程度越高或存在裸露地表。利用 NBR 值，可以有效地监测火灾影响和植被恢复情况。

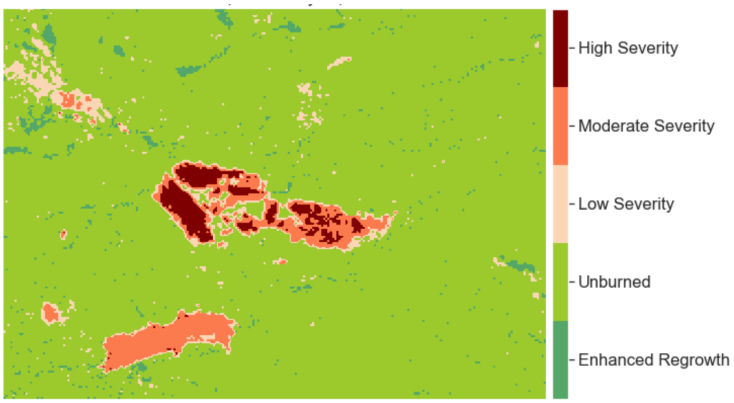

图 1: NBR 图像示例

对于每张图像，我们需要计算 512×512 个像素点的 NBR 值，这是一个典型的计算密集型任务，可以通过 MPI 通信框架实现。具体而言，计算 NBR 的大致算法如下：

1. 在主进程 (rank=0) 中：
   (a) 创建输出路径 `output`；
   (b) 读取数据集根目录下的所有图像文件名，并将其存储在数组 `tiff_files` 中；
   (c) 广播文件总数 `num_files` 和所有文件名 `filenames_buffer` 给所有进程；
2. 在其他进程 (rank≠0) 中：
   (a) 接收文件总数 `num_files` 和所有文件名 `filenames_buffer`；
3. 循环遍历所有的 GeoTiff 图像；

(a) 如果是其他进程 (rank≠0)，则计算 NBR。每个进程计算一部分图像块的 NBR 值；

(b) 如果是主进程 (rank=0)，则接收其他进程的计算结果，拼成一张完整的 NBR 图，并将其写入输出路径 `output`；

(c) 同步所有进程，准备处理下一张 GeoTiff 图像；

2 MPI 点对点通信

点对点通信实现 NBR 计算只涉及两个函数：`MPI_Send` 和 `MPI_Recv`。整个 NBR 计算过程中有两个核心问题：

- **广播：**在主进程中获得了全部图像的数组 `tiff_files` 后，因为每个进程都需要参与到所有图像的 NBR 计算中，所以每个进程都应该获得全部图像的数量和文件名信息。因此，
  - 在主进程中，通过在循环中

    ```
    MPI_Send(&num_files, 1, MPI_INT, dest, 0, MPI_COMM_WORLD)
    MPI_Send(filenames_buffer, num_files, MPI_CHAR, dest, 0, MPI_COMM_WORLD)
    ```

    广播文件总数 `num_files` 和所有文件名 `filenames_buffer` 给所有进程；
  - 在其他进程中，通过在循环中

    ```
    MPI_Recv(&num_files, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status)
    MPI_Recv(filenames_buffer, num_files, MPI_CHAR, 0, 0, MPI_COMM_WORLD, &status)
    ```

    接收文件总数 `num_files` 和所有文件名 `filenames_buffer`。
- **并行计算：**计算 NBR 时需要划分每个进程的处理区：对于指定的进程数 n，我们将图片按行数作如下划分

$$\text{rows\_per\_process} = 512//n,\ \text{remaining\_row} = 512\%n$$

每个进程需要处理的起始行号和终止行号分别按照如下方法计算

$$\text{start\_row} = \text{rank} \times \text{rows\_per\_process} + \min(\text{rank}, \text{remaining\_row})$$
$$\text{end\_row} = (\text{rank}+1) \times \text{rows\_per\_process} + \min(\text{rank}+1, \text{remaining\_row})$$

也就是说，对于前 `remaining_rows` 个进程，每个进程多处理一行。比如当 $n = 6$ 时，前 2 个进程处理 $512//6 + 1 = 86$ 行，剩余 4 个进程处理 $512//6 = 85$ 行。在计算 NBR 时，每个进程只需要处理从 `start_row` 到 `end_row` 的行即可。在 rank≠0 的进程计算完毕后，显式地通过 `MPI_Send` 函数将计算结果发送给主进程，主进程通过 `MPI_Recv` 函数接收计算结果。

处理后的 NBR 图像如图2所示。

2.1 编译与运行

使用点对点通信的 NBR 代码存放在 `nbr_p2p.cpp`，通过以下方法编译与运行：

```
mpicxx -o nbr_p2p nbr_p2p.cpp -lgdal
mpirun --allow-run-as-root --mca btl ^vader -np 4 nbr_p2p
```

这里`--allow-run-as-root` 是为了避免权限问题，`--mca btl ^vader` 是为了避免 vader 通信模块的使用导致的报错，`-np 4` 表示使用 4 个进程。

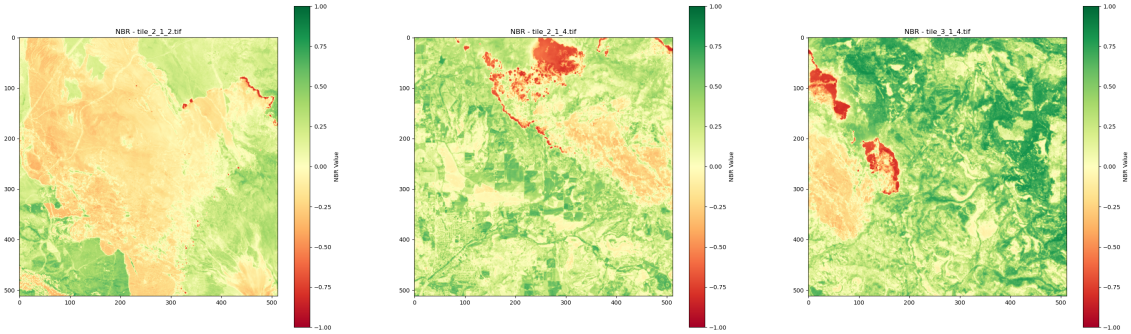

图 2: 并行计算生成的 NBR 图像示例

3 MPI 集合通信

由于本作业中频繁通过主进程广播数据与收集计算后的数据，通过其他进程计算 NBR 值并传递给主进程，相比点对点通信，MPI 的集合通信可以更好地实现这一目标。在本作业中，主要通过 `MPI_Bcsat` 和 `MPI_Gatherv` 函数实现集合通信。类似地，

- **广播：**直接通过 MPI 内置的 `MPI_Bcast` 函数广播文件总数 `num_files` 和所有文件名 `filenames_buffer` 给所有进程，而不用两个循环分别发送和接收；
- **并行计算：**注意到当 n 不能整除 512 时，`remaining_row` 不为 0，这时每个进程需要处理的数据量不同。`MPI_Gather` 假设了每个进程发送的数据量相同，这与实际情况相悖。因此，`MPI_Gatherv` 更为合适，其允许每个进程发送不同数量的数据，并将这些数据按正确的偏移量存储在根进程的接收缓冲区中。在主进程中，我们需要计算每个进程发送的数据量数组 `recvcounts[]` 和每个进程发送的数据的偏移量数组 `displs[]`，并在 `MPI_Gatherv` 函数中传递这两个参数。

```
MPI_Gatherv(nbr_data.data(), num_pixels, MPI_FLOAT,
            all_nbr_data.data(), recvcounts, displs, MPI_FLOAT,
            0, MPI_COMM_WORLD);
```

3.1 编译与运行

使用集合通信的 NBR 代码存放在 `nbr_collective.cpp`，通过以下方法编译与运行：

```
mpicxx -o nbr_collective nbr_collective.cpp -lgdal
mpirun --allow-run-as-root --mca btl ^vader -np 4 nbr_collective
```

4 两种通信方式的对比

表1对比了点对点通信和集合通信在设置不同进程数下平均每个进程的运行时间。由于集合通信在广播函数 `MPI_Bcast` 和收集函数 `MPI_Gatherv` 中使用了树形结构进行优化，因此集合通信的运行时间理论上应该相对于点对点通信更短。然而，介于待处理数据量的有限，在进程数较小的情况下，两种通信方式的运行时间差异并不明显，而当进程数增加时，集合通信的优势逐渐显现。

值得注意的是，直觉上而言，进程数越多，NBR 的并行程度越高，运行时间应该越短。然而，由于多进程的通信开销随进程数的增加迅速增长

- 对于点对点通信而言，通信开销的增长速度是线性的 ($O(n)$)；
- 对于集合通信而言，通信开销的增长速度是对数的 ($O(\log n)$)。

当进程数过多时，通信开销会超过计算开销，导致运行时间增加。因此，进程数的选择应该在计算开销和通信开销之间取得平衡。

进程数	2	4	6	8	10	12	14	16
点对点通信平均运行时间 (秒)	1.08	1.48	5.42	8.27	12.55	19.70	25.93	31.65
集合通信平均运行时间 (秒)	1.18	1.19	4.40	7.02	11.65	18.60	21.20	22.70

表 1: 点对点通信与集合通信的运行时间对比

运行配置

代码运行需安装 MPI 库、GDAL 库。实验结果在搭载 144 核 Intel Xeon Platinum 8352V CPU 的服务器上获得。