March 12, 2023

Project 2: Connected component labeling

MPI, 2D topology, ghost layers

Introduction

Connected component labeling is a class of image analysis algorithms. Given a 2D image of pixel values, such an algorithm scans it and assigns labels to pixels. These labels identify groups of pixels that are topologically connected. Labeled pixels either belong to the same connected component or not depending on whether their labels are same or different. For two pixels of the same label there exists a contiguous path of pixels of the same label that connects them together.

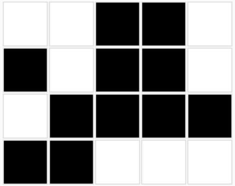


image of pixels

X	X	.	.	X
.	X	.	.	X
X	.	.	.	.
.	.	X	X	X

input

T	T	F	F	T
F	T	F	F	T
T	F	F	F	F
F	F	T	T	T

logical representation

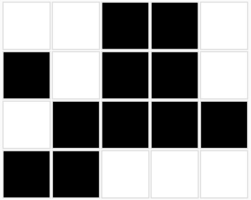
We consider 2D bi-level images with either black or white pixels, represented by the characters '`.`' and '`X`' on input, and by the logical values of `.false.` and `.true.` in arrays.

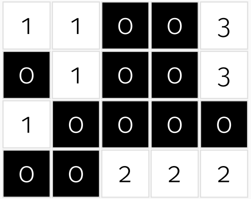
Illustrative example

Such algorithms are frequently used in astronomy. For example, one may have a 2D field of stars or galaxies that need to be identified. Fitting 2D Gaussians of ellipsoids is expensive, so before doing this the image may be binarized for a specific color range, all connected components counted and labeled using the CCL algorithm, their centroids computed and fitting performed for each component independently and much faster. Another possibility is measuring areas of irregular structures, for example, the characteristic size of a granular cell in MHD turbulence etc.

Connectivity

We will only consider the *8-connectivity* case: all eight nearest neighbors can be connected to the current pixel, two vertically, two horizontally, and four diagonally.


Input bi-level image



Output labels

Textbook solution

Textbooks present the most common solution to this problem using a simple but time-inefficient algorithm. Let pixel values of the image be $I_{ij} : i \in \{1..m\}, j \in \{1..n\}$. Firstly, assign for each white pixel some unique label, usually, the memory offset, $L_{ij} = (i-1) + m(j-1)$, and give some very big label for each black pixel. Secondly, for each white I_{ij} assign the minimum label among the nearest neighbors, $L_{ij} = \min L_{i-1..i+1, j-1..j+1}$. Repeat this iteratively while labels are changing. Thirdly, count the number of connected components K and renumber labels from 1 to K for white pixels and make them 0 for false pixels.

	1	2	3	4	5
1	0	4	∞	∞	16
2	∞	5	∞	∞	17
3	2	∞	∞	∞	∞
4	∞	∞	11	15	19

unique labels assigned

	1	2	3	4	5
1	0	0	∞	∞	16
2	∞	0	∞	∞	16
3	0	∞	∞	∞	∞
4	∞	∞	11	11	11

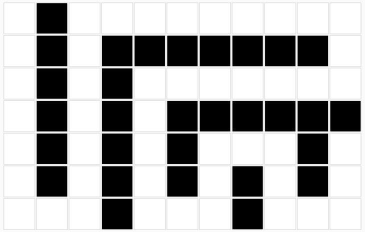
after many iterations

1	1	0	0	3
0	1	0	0	3
1	0	0	0	0
0	0	2	2	2

labels renumbered

Textbook solution

The iterative min-reduction is akin to the relaxation Jacobi-type operator in the heat transport solution. It is as slow as heat diffusion. In the worst case the time scaling is $O(mn)$, which is the image size. This happens when there is a single component that covers the entire image like a snake. This algorithm is simple and needs memory only for two images as well as labels, but is very time-inefficient for big inputs with long components. Also, renumbering labels in parallel is not trivial.


worst case example

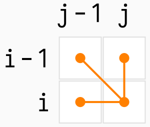
Better methods employ either topological transforms or recursive image partitioning with algorithms on graphs. First are simple and easy to parallelize at the expense of heavy memory consumption. Second are fast and memory-frugal although difficult to parallelize. We'll consider only the first class.

Levialdi's algorithm

The algorithm of Levialdi (1972) uses one of the simplest topological transforms, which preserves the 8-connectivity of image components. It consists of two phases: shrink and growth. Both take $O(m+n)$ iterations.

In the shrink phase a 2×2 operator is applied to the neighborhood of each image pixel I_{ij}. This operator is chosen so as each component shrinks away but neither connects to other components nor splits up. This is repeated iteratively until each component is reduced to an isolated pixel and finally removed. At each iteration $t : t \in 1..T$ the current image state I^t_{ij} must be remembered. The process stops when all pixels $I^T_{1..m,1..n}$ become black. Pixel values outside the domain $\{1..m, 1..n\}$ are assumed to be always black.

In the growth phase the process is reversed from $t = T$ to 1 to compute labels by restoring saved images. For each isolated pixel I^t_{ij}, appearing in the current image, a new label L_{ij} is assigned and then copied in the following iterations to the nearest neighbors from the operator stencil.



shrink operator

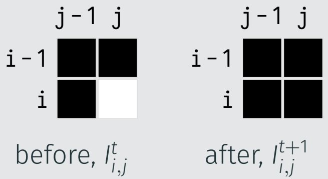
Shrinking components

For a 2×2 operator there are 16 pixel configurations of bi-level values. Only three of them will change the current pixel value at `(i, j)` in the next image state, while others will not.

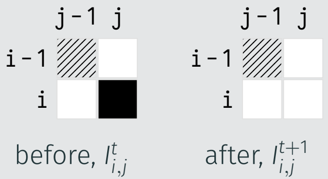
If in the image state *t* the current pixel at `(i, j)` is white and the three neighbors at `(i, j-1)`, `(i-1, j-1)`, and `(i-1, j)` are black, then in the next state $t+1$ the current pixel becomes black.

If in the image state *t* the current pixel at `(i, j)` is black and the two neighbors at `(i, j-1)` and `(i-1, j)` are white, then in the next state $t+1$ the current pixel becomes white. The third neighbor at `(i-1, j-1)` can be either black or white.

$I^t_{i,j}$ is white/ `.true.`



$I^t_{i,j}$ is black/ `.false.`



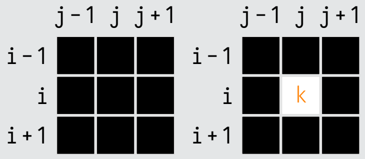
Growing components

All images $I^t_{i,j}$ are stored for states $1 \leq t \leq T$, where $I^1_{i,j}$ is the initial image, and $I^T_{i,j}$ is all black. Now we go back from T to 1 and calculate labels $L_{i,j}$ at each state t. Let t be the current state and $t+1$ be the previous one.

If in the current image state t a new isolated pixel appears at `(i, j)`, then it is assigned a new label k at $L_{i,j}$. This label will propagate and fill its component in the subsequent states. Initially, $k = 0$ at $t = T$ and is incremented by one every time a new isolated pixel appears. Finally, $k = K$ is the total number of components at $t = 1$, when the original image is restored.

All black pixels have the special label of zero, $L_{i,j} = 0$.

white pixel $I^t_{i,j}$ is isolated



previous $I^{t+1}_{i,j}$ current $I^t_{i,j}$

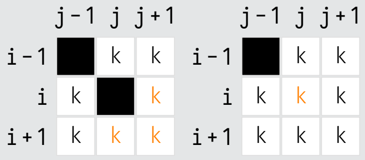
Growing components

If in the current state t the white pixel at $(\mathtt{i},\mathtt{j})$ has neighbors, then it gets its label from the previous state $t+1$ from any of the three neighbors at $(\mathtt{i},\mathtt{j+1})$, $(\mathtt{i+1},\mathtt{j})$ or $(\mathtt{i+1},\mathtt{j+1})$, which is white.

This can be thought the other way round. Any white pixel $(\mathtt{i},\mathtt{j})$ at the previous state $t+1$ propagates its label into the next state t to any of the tree neighbors at $(\mathtt{i-1},\mathtt{j})$, $(\mathtt{i},\mathtt{j-1})$ and $(\mathtt{i-1},\mathtt{j-1})$, which are white.

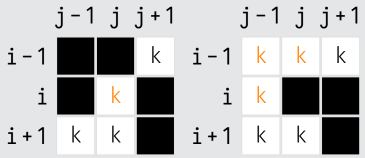
As before, black pixels have the special label of zero, $L_{i,j}=0$.

white pixel $I^t_{i,j}$ gets its label from 3 neighbors of $I^t_{i,j}$



previous $I^{t+1}_{i,j}$ current $I^t_{i,j}$

white pixel $I^{t+1}_{i,j}$ propagates its label to 3 neighbors of $I^t_{i,j}$

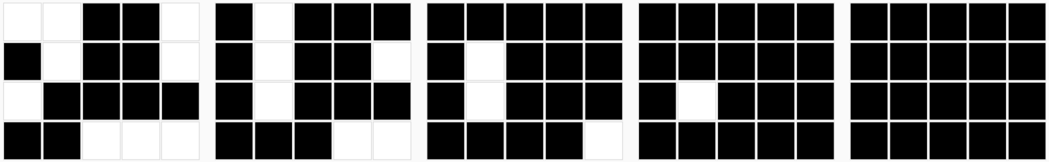


previous $I^{t+1}_{i,j}$ current $I^t_{i,j}$

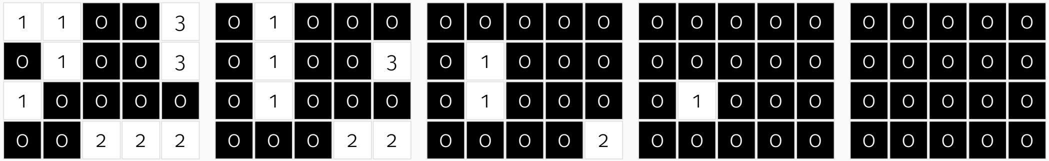
Example: 4×5 image with 3 components

$t = 1$ $t = 2$ $t = 3$ $t = 4$ $t = 5$

$\rightarrow$ shrink phase $\rightarrow$



$\leftarrow$ growth phase $\leftarrow$



1	1	0	0	3
0	1	0	0	3
1	0	0	0	0
0	0	2	2	2

0	1	0	0	0
0	1	0	0	3
0	1	0	0	0
0	0	0	2	2

0	0	0	0	0
0	1	0	0	0
0	1	0	0	0
0	0	0	0	2

0	0	0	0	0
0	0	0	0	0
0	1	0	0	0
0	0	0	0	0

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Input/output format

```
4  5  2  2
xx..x
.x..x
x....
..xxx
```

The input format specifies the size of the image $m \times n$ and the number of block rows n_{rows} and columns n_{cols} in a 2D topology (read by the parallel version only). What follows is a 2D grid of dots and crosses that represents the image $I_{1..m,1..n}$. Dots are black or `.false.` pixels, crosses are white or `.true.` pixels.

```
3
 1 1 0 0 3
 0 1 0 0 3
 1 0 0 0 0
 0 0 2 2 2
```

The output format specifies the total number of components K and the array of labels, which enumerate components from 1 to K as well as black pixels with zero. Array values must be printed with one leading space followed by the full width of the biggest label K. For example, if $100 \leq K \leq 999$, then different labels will be printed as ·`521`, ··`42`, or ···`7`.

What needs to be done?

Firstly, implement the serial version of this solution. Prepare several simple inputs for tests and run them to get correct outputs. Use them later to test your parallel version. The parallel version must use the last two numbers from the 1st input row to create a 2D topology of $n_{\text{rows}} \times n_{\text{cols}}$ ranks. The parallel version must run with any reasonable number of processes up to 128 and possible topology sizes, such as 1×1, 16×1, 8×12 and so on. It must produce similar outputs but it is not required that the order of labels is the same as it is hard to maintain it in parallel. You are free to choose names of variables, subprograms, whether it is coded as one big main program, with modules, or classes. For the final test I'll run your parallel version on several big images of Mpix in size, e.g., a scanned page of a book to count printed characters.

Project 2 mark contributes 1/3 to the total mark of the course. Get at least a 50% mark to pass this project. You must submit only the parallel version. I might ask you for the serial version in case something is not clear in the parallel solution. This project is individual. You can discuss general ideas and technical details between each other, but the code you submit you must write yourself.

Breakdown of marks

5% for reading in and distributing the input numbers to all processes.

10% for creating a 2D topology. You must choose the right type of it, check if its size corresponds to the number of processes, find the block coordinates for a given rank, find the rank for given coordinates, calculate ranks for the neighbor processes.

5% for partitioning the image grid between processes as evenly as possible. Local array indices must be contiguous along corresponding topological rows or columns with respect to the global grid range.

10% for correctly declaring, using, and releasing MPI datatypes, which are needed for synchronizing boundaries and doing I/O in parallel.

10% for correctly choosing boundary conditions and synchronizing domain boundaries between the neighbors through their ghost cells without deadlocks;

Breakdown of marks

20% for doing I/O line-by-line in parallel via the root rank. It must be root-to-all for reading and all-to-root for printing. Use only point-to-point calls, not collectives. On reading, image characters must be converted to logicals. On writing, correct tight and common format must be used for integer labels. Any auxiliary I/O routine used for intermediate debugging won't be evaluated.

10% for the forward phase: correctly implementing the shrink operator, doing iterations, handling and saving image states, checking in parallel when iterations must be stopped. Remember that the number of iterations is not known in advance.

20% for the backward phase: correctly counting and propagating labels in parallel when components grow, doing iterations in reverse, handling and restoring image states.

10% for correctly creating, using, and releasing dynamic memory variables. In the worst case of a 1 Mpix image, the 2 GiB memory limit per one process might be broken. To get full mark try to minimize memory usage in the most consuming part of the program.