Connected Components Labeling for Giga-Cell Multi-Categorical Rasters

Pawel Netzel[a,b], Tomasz F. Stepinski[a,1]

[a] *Space Informatics Lab, Department of Geography,University of Cincinnati, Cincinnati, OH 45221-0131, USA*
[b] *Department of Climatology and Atmospheric Protection, University of Wroclaw, Kosiby 6/8, 51-621, Wroclaw, Poland*

Abstract

Labeling of connected components in an image or a raster of non-imagery data is a fundamental operation in fields of pattern recognition and machine intelligence. The bulk of effort devoted to designing efficient connected components labeling (CCL) algorithms concentrated on the domain of binary images where labeling is required for a computer to recognize objects. In contrast, in the Geographical Information Science (GIS) a CCL algorithm is mostly applied to multi-categorical rasters in order to either convert a raster to a shapefile, or for statistical characterization of individual clumps. Recently, it has become necessary to label connected components in very large, giga-cell size, multi-categorical rasters but performance of existing CCL algorithms lacks sufficient speed to accomplish such task. In this paper we present a modification to the popular two-scan CCL algorithm that enables labeling of giga-cell size, multi-categorical rasters. Our approach is to apply a divide-and-conquer technique coupled with parallel processing to a standard two-scan algorithm. For specificity, we have developed a variant of a standard CCL algorithm implemented as *r.clump* in GRASS GIS. We have established optimal values of data blocks (stemming from the divide-and-conquer technique) and optimal number of computational threads (stemming from parallel processing) for a new algorithm called *r.clump3p*. The performance of the new algorithm was tested on a series of rasters up to 160Mcells in size; for largest size test raster a speed up over the original algorithm is 74 times. Finally, we have applied the new algorithm to the National Land Cover Dataset 2006 raster with 1.6×10^{10} cells. Labeling this raster took 39 hours using two-processors, 16 cores computer and resulted in 221,718,501 clumps. Estimated speed up over the original algorithm is 450 times. The *r.clump3p* works within the GRASS environment and is available in the public domain.

Keywords: connected components labeling, divide-and-conquer technique, parallel processing, land cover dataset

1. Introduction

Connected component labeling (CCL) is one of the most fundamental operations in pattern analysis. The original CCL algorithm (Rosenfeld and Pfaltz, 1966) was intended for binary images; its purpose was to identify all 4- or 8-connected regions of pixels (*clumps*) having values of 1 and to assign each of them a unique label. Subsequently, many different CCL algorithms have been proposed including multi-scan algorithms (Haralick, 1981; Hashizume et al., 1990), two-scan algorithms (Rosenfeld and Pfaltz, 1966; Rosenfeld, 1970), hybrid algorithms (Suzuki et al., 2003), and tracing-type algorithms (Rosenfeld, 1970; Hu et al., 2005; Chang et al., 2004). The most popular CCL algorithm is the two-scan algorithm (Rosenfeld and Pfaltz, 1966; Rosenfeld, 1970). In image analysis, the CCL is a fundamental step in segmentation of binary image into constituent objects that a computer system needs to recognize. Applications include optical character recognition, automated inspection, target recognition, medical image analysis, and computer-aided diagnosis (Ronsen and Denjiver, 1984). Note that the aforementioned applications usually involve images with $n \leq 10^6$ pixels, which is convenient because the conventional two-scan algorithm has, in general, performance that depends steeply on size and complexity of an image and becomes impractical for large and complex images. Previous work on fast and efficient CCL algorithms (Asano and Tanaka, 2010; Stefano and Bulgarelli, 1999; Bock and Philips, 2010) is restricted to binary images and may not be extensible to multi-categorical rasters.

In the GIS application, the CCL is needed to identify clumps in a categorical raster (often a classified image) in order to convert the raster data into a shapefile

*Corresponding author. stepintz@uc.edu, *tel/fax:* 513-556-3583/513-556-3370

format, or for statistical characterization of the clumps. If clumps of a single category needs to be identified, a standard binary raster CCL algorithm can be applied, but if an application calls for identification of all clumps in all categories, the standard CCL algorithm needs to be extended in order to handle multi-categorical data. Such extensions to the conventional two-scan algorithm have been implemented in all major GIS software packages. For example, in the GRASS (Geographic Resources Analysis Support System) (Neteler and Mitasova, 2007) the CCL algorithm is implemented as *r.clump*. Because *r.clump* implementation is based on the conventional two-scan algorithm, there is a practical limit on the size of the raster to which it can be applied.

Advances in remote sensing result in ever increasing volume of high resolution imagery, many of which are automatically classified and turned into land products, such as, for example, the National Land Cover Database (NLCD) (Fry et al., 2009; Xian et al., 2009) that maps land cover/land use over the entire conterminous United States or Coordination of Information on the Environment (CORINE) (Lima, 2005) that maps land cover/land use over most of Europe. Similarly large rasters, depicting spatial distribution of natural and/or anthropogenic features, can be constructed from other remotely sensed and/or ground gathered non-imagery data. We refer to such datasets as giga-cell rasters because they often contain $> 10^9$ cells; for example, the NLCD is a 16-classes raster containing 1.6×10^{10} cells. Recently, calculating all clumps in a giga-cell raster become an issue in connection with development of a system for querying such rasters for local regions having patterns of categories similar to a given example (Jasiewicz and Stepinski, 2012). Note that calculating clumps in a giga-cell raster from smaller tiles and combining them together is not a solution because it would lead to some clumps being artificially cut by tiling process resulting in erroneous statistics of clump sizes and shapes. An optimistic estimate of the time needed for a conventional two-scan CCL algorithm (as implemented in *r.clump*) to label all clumps in the entire NLCD raster is about 2 years using a two-processors, 16 cores computer.

This paper presents an extension to the two-scan CCL algorithm aimed at reduction of the time necessary to label all clumps in a giga-cell raster by two-to-three orders of magnitude. For the sake of specificity, we concentrate on modifying the GRASS module *r.clump* using a divide and conquer technique and parallel processing to achieve a desired speed up. Our idea is based on an earlier work (Park et al., 2000) but extends it by the following:

- Input is not restricted to a binary raster, instead we allow for a multi-categorical raster with no limits on the number of categories.
- Parallelization of computing processing
- Performance tested up to rasters with 10^{10} cells.
- Optimized implementation in GRASS takes advantage of GRASS custom spatial database and its ability of fast row-by-row data processing.

Algorithm 1: Basic structure of *r.clump*

```
input  : Multi-categorical raster 𝒜
output : Connected components labels raster Q
for row = 1 to N do
    read focus and previous rows;
    execute algorithm assign_labels to assign
    temporary labels to cells in the focus row;
    update dictionary 𝒟 with temporary labels;
end
re-order labels in dictionary to obtain consecutive
numbering;
for row = 1 to N do
    read focus and previous rows;
    execute algorithm assign_labels to assign final
    labels to cells in the focus row;
    update dictionary 𝒟 with final labels;
    write a focus row of labels to output Q;
end
```

2. Multi-categorical connected components labeling algorithm

Multi-categorical CCL algorithm identifies all 4- or 8- connected regions of cells sharing the same categorical values and assigns each of them a unique label. The *r.clump*, a multi-categorical CCL algorithm on which this work is based, is a variant of a two-scan algorithm modified for use in multi-categorical rasters; it assumes 4-connectivity - a preferred type of connectivity when working with remotely sensed geospatial data. An input to *r.clump* is a raster $\mathcal{A}$ that has N rows and M columns. $\mathcal{A}(i, j)$ refers to the element in row i and column j. To each cell in $\mathcal{A}$ a category class L is assigned; class $L = 0$ indicates *noData* while the values $L \geq 1$ indicate actual classes. An output of *r.clump* is a raster Q having the same dimensions as $\mathcal{A}$ and holding labels identifying unique connected components.

Algorithm 2: Function *assign_labels*

```
input  : focus and previous rows of 𝒜, previous
         row of Q, label directory 𝒟
output: current row of Q

for column = 1 to M do
    if class ≠ noData then
        if class ≠ classUp and class ≠ classLeft
        then
            assignNewLabel;
            addNewLabelToDirectory;
        else
            if class ≠ classUp and class = classLeft
            then
                assignLeftLabel
            else
                if class = classUp and class ≠
                classLeft then
                    assignUpLabel
                else
                    if LeftLabel = UpLabel then
                        assignUpLabel
                    else
                        assignUpLabel;
                        if pass = 1 then
                            updateDictionary
                        end
                    end
                end
            end
        end
    end
end
```

Algorithm 1 shows the basic structure of *r.clump*. The algorithm passes the raster twice. In the first pass it assigns temporary labels to the connected components and builds-up an array holding all already assigned labels and their equivalences; we refer to this array as “dictionary” and use symbol $\mathcal{D}$ do denote it. Because of the design of the algorithm the first pass results in possible over-labeling and existence of non-consecutive labels. The purpose of the second pass is to eliminate unnecessary labels and to make remaining labels consecutive. The key part of Algorithm 1 is the function *assign_labels* that assigns labels to the cells in the focus row; its design is shown in Algorithm 2.

Algorithm 2 operates cell-by-cell in a focus row of $\mathcal{A}$. Because of assumed 4-connectivity, a focus cell (having class = *class*) is compared with only two other cells, a cell immediately to its left (having class = *classLeft*) and a cell immediately up (having class = *classUp*). Note that because of row-by-row, left-to-right processing of $\mathcal{A}$ both of these neighboring cells have already clump labels (*LeftLabel* and *UpLabel*) assigned to them before a focus cell is processed. As Algorithm 2 shows, assigning a clump label to a focus cell is straightforward, except in the case where both of the neighbors happens to have the same class as the focus cell but are assigned different clump labels. In this case the focus cell fuses the two previously separate clumps. In Algorithm 2 *assignLeftLabel* and *assignUpLabel* denotes operations of assigning the focus cell with labels from its respective neighbors, whereas *assignNewLabel* denotes issuing a new label and *addNewLabelToDirectory* denotes appending $\mathcal{D}$ by its value. When focus cell fuses two clumps, it becomes necessary to record this equivalence using a process denoted by *updateDictionary*.

The two parameters critical for the time of execution of *r.clump* are the length of the row M and the length of $\mathcal{D}$. For a giga-cell rasters, like the NLCD, $M = 161,000$ and the length of $\mathcal{D} \geq 220,000,000$. In contrast a small block of the NLCD raster (with size of 500×500 cells) has $M = 500$ and the length of $\mathcal{D} \sim 1000$. Thus, the major bottleneck in applying *r.clump* to giga-cell rasters is the great length of $\mathcal{D}$.

3. Divide-and-conquer approach

Our solution to overcome this bottleneck is to apply the divide-and-conquer technique. The idea (first proposed in the context of much smaller binary images by Park et al. (2000)) is to divide a raster $\mathcal{A}$ into a number of much smaller blocks which each block having dimensions of $n \times m$ with $n \ll N$ and $m \ll M$. In this paper we will use blocks with $n = m = 500$ cells.

In designing our divide-and-conquer CCL algorithm we take advantage of GRASS database structure which works most efficiently if the data are processed row by row. Therefore, instead of dividing entire raster $\mathcal{A}$ into blocks, we first divide it into horizontal buffers. A buffer has a height n, equal to the size of the block, and a width M, equal to the width of $\mathcal{A}$. Thus, a giga-cell raster is processed in a buffer-by-buffer fashion. This is shown schematically on Fig. 1 where two (of many possible) buffers are shown. Each buffer is in turn divided into the set of square ($n \times n$ cells) blocks and each block is processed individually using original *r.clump* algorithm (see Algorithms 1 and 2) resulting in creation of local, block-specific temporary and small dictionaries of clump labels (see Fig. 1). Because each block is small ($n = 500$ cells in our calculations) local CCL calculations are very rapid. Moreover, because calculating connected components for each block is independent from the data in the other blocks, the algorithm is ideally suited for parallel processing. We use OpenMP library (Chapman et al., 2007) to enable parallel processing (see the line "#pragma omp parallel for" in Algorithm 3).

Algorithm 3: Divide-and-conquer CCL

```
input  : Multi-categorical raster A, blockSize
output: Connected components labels raster Q

Calculate a number, N_buff, of horizontal buffers in A;
Calculate a number, M_blocks, of blocks in a buffer;
Create buffers and blocks;

for buffer = 1 to N_buff do
    load horizontal buffer;
    #pragma omp parallel for;
    for block = 1 to M_blocks do
        call Algorithm 1
    end
    for block = 1 to M_blocks do
        merge labels in focus block with labels in
        block to the left and a single row of data
        locate up
    end
    save all labels in horizontal buffer;
end
```

Algorithm 3 show schematically the working of our divide-and-conquer algorithm. The ability to process multi-categorical data is achieved by using original GRASS *r.clump* algorithm as a base clumping algorithm. In the algorithm proposed in (Park et al., 2000) each block of data was clumped and its local label directory was reconciled and merged with a global directory resulting from blocks that have been already processed. This design would not allow for parallelization. In our design a number of blocks are clumped in parallel before their labels are reconciled and merged with the global directory. This design feature is reflected in Algorithm 3 by existence of two separate loops over the blocks: the first loop clumps blocks in parallel and, after it finishes, the second loop merges the labels. The merging of labels is performed using a technique described in (Park et al., 2000) extended to multicategorical data.

[Figure 1 about here.]

[Table 1 about here.]

4. Experimental results

In this section, we evaluate the effectiveness of our divide-and-conquer approach to connected components labeling of giga-cell multi-categorical rasters. The evaluation is performed on the NLCD 2006 dataset.

4.1. Data

National Land Cover Database 2006 (NLCD2006) is a 16-class land cover classification scheme that has been applied consistently across the conterminous United States at a spatial resolution of 30 meters. NLCD2006 is based primarily on the unsupervised classification of Landsat Enhanced Thematic Mapper+ (ETM+) using 2006 satellite data. The data is given in Albers Equal Area projection. In this projection, the spatial region is bounded by following coordinates: north 3310020 m, south 177270 m, east 2342670 m, west -2493060 m. The entire region has 161,191 rows and 104,425 columns of raster cells and it contains 16,832,787,875 cells. Because of its size, the connected components labels of the NLCD raster cannot be calculated (in a practical time frame) using the *r.clump* algorithm, or, to the best of our knowledge, any other existing clumping algorithm. Therefore we cannot test various algorithms on the entire NLCD; instead, we use a series of smaller regions (subsets of the entire NLCD) for testing the performance of our algorithm versus the standard *r.clump* algorithm.

Table 1 summarize the six test regions selected for testing and referred to as regions Test_0 (the smallest) to Test_5 (the largest). The testing regions varies in raster size from ~ 300 Kcells to ~ 100 Mcells. Even the biggest testing region contains less than 1% of the cells of the entire NLCD raster. The geographical context of testing regions is shown on Fig 2. Testing regions cover the portion of Midwest US including the city of Chicago.

4.2. Calculations

Our calculations proceeded as described in section 3 and outlined in Algorithm 3. Entire process could be described as wrapping the divided-and-conquer technique over the existing multi-categorical CCL algorithm *r.clump*. The resulting code is referred to as *r.clump3p*; the letter "p" at the end of the name indicates that the code was optimized for parallel processing. In addition to the size of a raster and its complexity, there are two parameters that influence the performance of *r.clump3p*; (1) the number of blocks, and (2) the number of threads in parallel processing.

[Figure 2 about here.]

[Figure 3 about here.]

We have evaluated the impact of block size on the efficiency of computation by using different block sizes: 50, 100, 500, 750, 1000 and 2000 cells, respectively. The smaller the block the more efficient is the core algorithm *r.clump* because of the shortness of the label dictionary. However, larger number of blocks leads to a larger overhead associated with merging labels from individual blocks. Because of this trade-off we expect that there exists an optimal size of the block for which our algorithm exhibits optimal performance. Fig. 3 shows the results of testing dependence of algorithm performance on the block size (and thus, on the number of blocks). Calculations are performed using the two largest testing regions Test_4 (33 Mcells) and Test_5 (116 Mcells). As explained in section 3, all blocks have square size and the buffer height is equal to the block size. For this experiment we use only a single computational thread. The results indicate that block size of 500 or 750 cells is optimal from computational efficiency point of view.

Next, we evaluate the impact the number of threads has on code performance. The computer available to us was equipped with two processors each having 8 physical cores. With Hyper-Threading Technology, it allows running of up to 32 threads in parallel. Experiment aimed at establishing dependence of code performance on the number of threads was conducted using the largest test region (Test_5) having size of 116 Mcell and the two optimal choices for block size: 500 and 750 cells. Fig. 4 shows the results which indicate that the optimal number of threads is 15-16, approximately equal to the number of physical cores in the computer.

[Figure 4 about here.]

[Table 2 about here.]

[Figure 5 about here.]

Based on these experiments we have concluded that block size of 500 cells and 15 threads running in parallel offer the best performance of *r.clump3p* algorithm. We conducted experiments aimed at comparing performance of *r.clump3p* with performance of *r.clump*. This comparison includes the original *r.clump* algorithm running on a single thread (this algorithm cannot be parallelized), the *r.clump3p* algorithm running on a single thread and having a block size of 50 cells (as suggested in Park et al. (2000)), the *r.clump3p* algorithm running on a single thread and having a block size of 500 cells (an optimal size as suggested by our experiments), and the *r.clump3p* algorithm running on 15 threads and having a block size of 500 cells. The results are summarized in Table 2. Examining a row in Table 2 corresponding to the largest test raster (Test_5) we note that the our optimally-tuned algorithm achieved an overall speed-up of 74 times over the *r.clump*. The divide-and-conquer approach yields a speed-up of 23 times; additional speed up of 3.23 times is due to parallel processing.

Fig. 5 shows functional trends of execution time with the number of cells in a raster. Note that these trends are established on the basis of six test runs covering rasters with sizes up to ~120Mcells. Because of an empirical character of these trends, their extrapolations to larger rasters needs to be taken with caution; it is likely that they underestimate the actual times needed. For the original *r.clump* algorithm a third order polynomial offer the best fit to the six test measurements, although a second order polynomial also offered a reasonably good fit. Extrapolating these fitted trends to a raster with 16,000 Mcells (like the NLCD) yields about 2 years for the second order polynomial fit and 158 years for the third order polynomial fit. On the basis of these estimates we claim that, in practice, the original *r.clump* algorithm cannot be used for labeling connected components in giga-cell rasters.

For our *r.clump3p* algorithm with optimal values of parameters (block size = 500 cells, number of threads = 15) the best fit to the six test measurements is provided by either a linear function or the second order polynomial. Extrapolating these fitted trends to a raster with 16,000 Mcells yields about 2 hours for the linear fit and

16 hours for the second order polynomial fit. The actual calculations took 39 hours. Comparing this time with the most optimistic estimate for *r.clump* (2 years) yield a speed up of about 450 times.

We used *r.clump3p* algorithm with optimal settings to label all connected components in the NLCD 2006 giga-cell raster. The calculation took 39 hours (1.6 days) and resulted in labeling of 221,718,501 clumps. In order to better appreciate the enormity of this task consider labeling of raster Test_0 shown in Fig. 2C. On this figure the 12,502 connected components of 0.36 Mcell raster are shown using random colors. It is clear, from the pattern of clumps seen on Fig. 2C, that NLCD raster has larger complexity than most binary images for which bulk of CCL analysis has been conducted; large complexity of image/raster results in more time demand on a CCL algorithm. Note that Test_0 raster contains only 0.002% of cells in the entire NLCD raster. Thus, a task of labeling connected components in giga-cell rasters stemming from remote sensing applications is truly enormous.

Conclusions

The aim of this paper is to present a design of connected components labeling algorithm capable of being applied to giga-cell size multi-categorical rasters. A necessity to label connected components in such large rasters arose in connection with a recent work (Jasiewicz and Stepinski, 2012) on pattern-based query system for retrieval of alike land cover scenes from high resolution, continental-scale dataset (NLCD 2006). In such a system an analyst selects a reference scene of interest and the system identifies all scenes in the dataset having similar patterns of land cover categories. A similarity function between two scenes is based on statistics of constituent clumps (their classes, sizes, and shapes) in each scene - since a need for clumping the entire NLCD. Note that an idea of pattern-based query is not restricted to land cover data as it can be utilized in a number of high resolution, high complexity continental or global scale rasters pertaining to natural or anthropogenic phenomena.

Using a divide-and-conquer technique and parallel processing we have designed an CCL algorithm with performance that is two-three orders of magnitude better than standard CCL algorithms. The specific speed up depends on the size of the data and its complexity and is greatest for very large and complex rasters. We have implemented the proposed algorithm as a GRASS module *r.clump3p* and demonstrated its usability by performing the connected components labeling for the entire 16 giga-cell raster containing NLCD 2006. The *r.clump3p* algorithm required 39 hours to complete the calculation on a computer equipped with 2 processor each having 8 cores. This is a very reasonable execution time considering that such labeling needs to be performed only occasionally. An estimate of speed up over a conventional CCL algorithm *r.clump* is 450 times. Thus, to the best of our knowledge, *r.clump3p* is the only algorithm capable of labeling this dataset in practical time frame. The *r.clump3p* implementation of our algorithm is available for download at http://sil.uc.edu and http://www.wgug.org.

5. Acknowledgments

This work was supported in part by the National Science Foundation under Grant BCS-1147702 and by the University of Cincinnati Space Exploration Institute.

References

Asano, T., Tanaka, H., 2010. In-place algorithm forest connected components labeling. Journal of Pattern Recognitions Research 1, 10–22.

Bock, J. D., Philips, W., 2010. Fast and memory efficient 2–d connected components using linked lists of line segments. IEEE Transactions on Image Processing 19, 3222–3231.

Chang, F., Chen, C. J., Lu, C. J., 2004. A linear-time component-labeling algorithm using contour tracing technique. Comput. Vision Image Understanding 93, 206–220.

Chapman, B., Jost, G., van der Pas, R., Kuck, D. J., 2007. Using OpenMP: Portable Shared Memory Parallel Programming. The MIT Press.

Fry, J. A., Coan, M. J., Homer, C. G., Meyer, D. K., Wickham, J. D., 2009. Completion of the National Land Cover Database (NLCD) 1992–2001 land cover change retrofit product. Tech. rep., U.S. Geological Survey Open-File Report 2008–1379.

Haralick, R. M., 1981. Some neighborhood operations. In: Real Time/Parallel Computing Image Analysis. Plenum Press, New York, pp. 11–35.

Hashizume, A., Suzuki, R., Yokouchi, H., etal., 1990. Analgorithm of automated rbc classification and its evaluation. Bio Med. Eng. 28, 25–32.

Hu, Q., Qian, G., Nowinski, W. L., 2005. Fast connected-component labeling in three-dimensional binary images based on iterative recursion. Comput. Vision Image Understanding 99, 414–434.

Jasiewicz, J., Stepinski, T. F., 2012. Example-based retrieval of alike land-cover scenes from nlcd2006 database. IEEE Geoscience and Remote Sensing Letters in print.

Lima, M., 2005. IMAGE2000 and CLC2000: Products and Methods. European Commission Joint Research Centre (DG JRC), Institute for Environment and Sustainability (IES), Land Management Unit, I-21020 Ispra (VA), Italy.

Neteler, M., Mitasova, H., 2007. Open source GIS: a GRASS GIS approach, 3rd Edition. Springer, New York.

Park, J. M., Looney, C. G., Chen, H. C., 2000. Fast connected component labeling algorithm using a divide and conquer technique. In: Shin, S. Y. (Ed.), Proceedings of the ISCA 15th International Conference Computers and Their Applications, March 29-31, 2000, New Orleans, Louisiana, USA. pp. 373–376.

Ronsen, C., Denjiver, P. A., 1984. Connected Components in Binary Images: The Detection Problem. Research Studies Press.

Rosenfeld, A., 1970. Connectivity in digital pictures. J. ACM 17, 146–160.

Rosenfeld, A., Pfaltz, J. L., 1966. Sequential operations in digital processing. J. ACM 13, 471–494.

Stefano, L. D., Bulgarelli, A., 1999. A simple and efficient connected components labelling algorithm. In: Preceding of the Tenth International Conference on Image Analysis, and Processing, Sept. 27-29, 1999, Venice, Italy. pp. 322–327.

Suzuki, K., Horiba, I., Sugie, N., 2003. Linear-time connected-component labeling based on sequential local operations,. Comput. Vision Image Understanding 89, 1–23.

Xian, G., Homer, C., Fry, J., 2009. Updating the 2001 national land cover database land cover classification to 2006 by using landsat imagery change detection methods. Remote Sensing of Environment 113(6), 1133–1147.

List of Figures

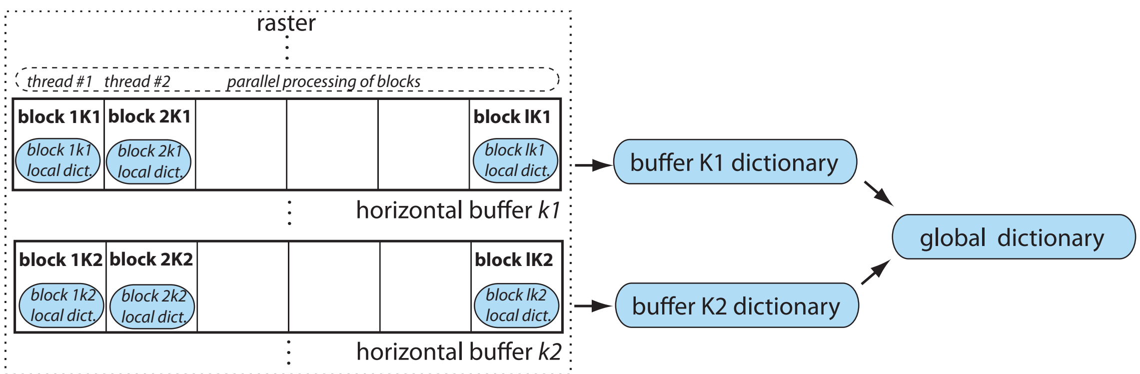

Figure 1: Schematic diagram showing the idea of divide-and-conquer approach to CCL.

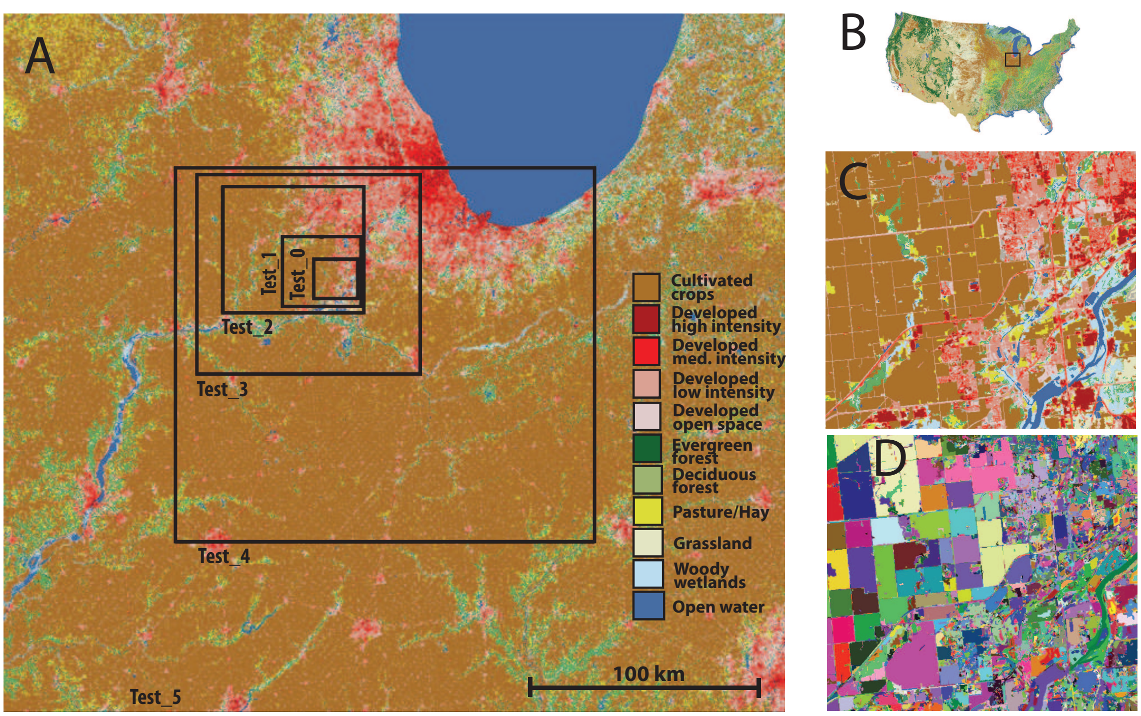

Figure 2: Geographical context of testing regions. (A) Six testing regions overlaid on the map of land cover; different colors on the map indicate different classes of land cover as indicated by the legend. (B) The location of the largest testing region. (D) Zoom-in into the test region Test_0. (D) Individual connected components (clumps) in the test region Test_0 are shown by randomly assigned colors.

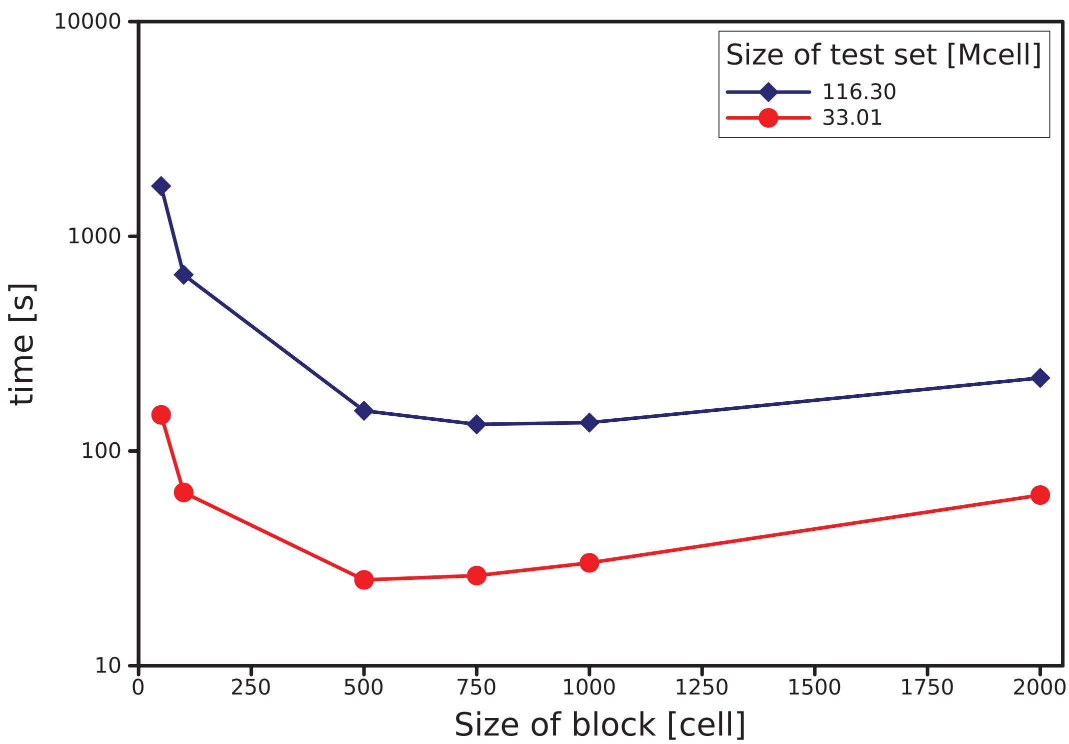

Figure 3: Dependence of calculation time on block size.

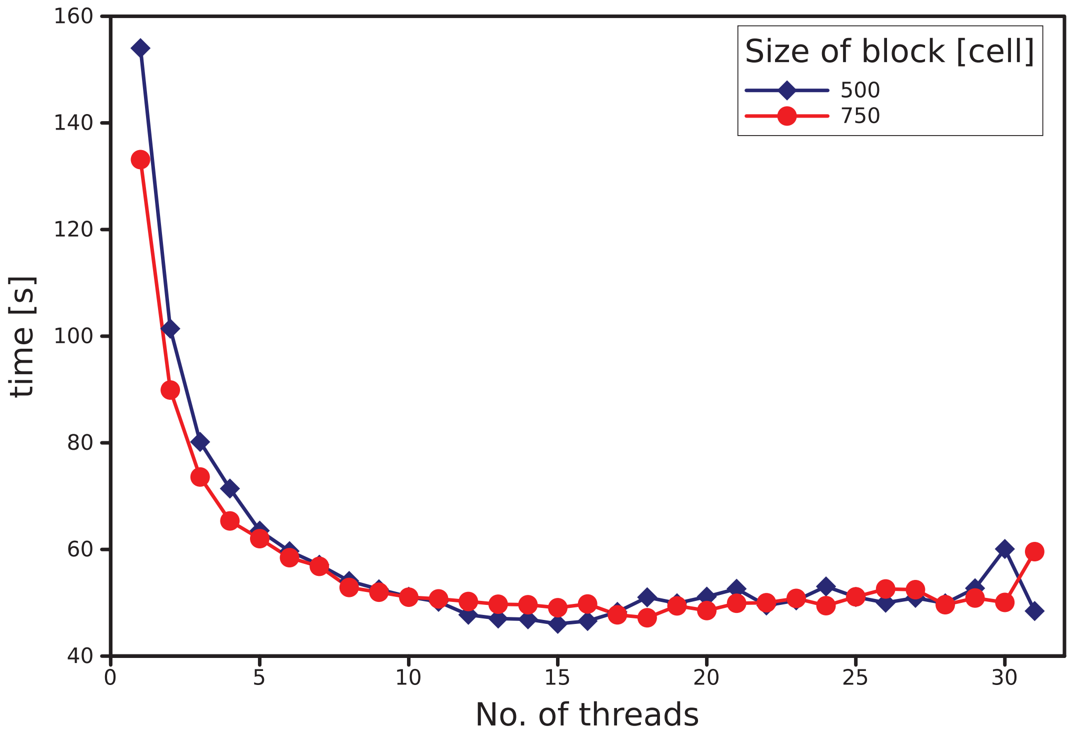

Figure 4: Dependence of calculation time on number of threads.

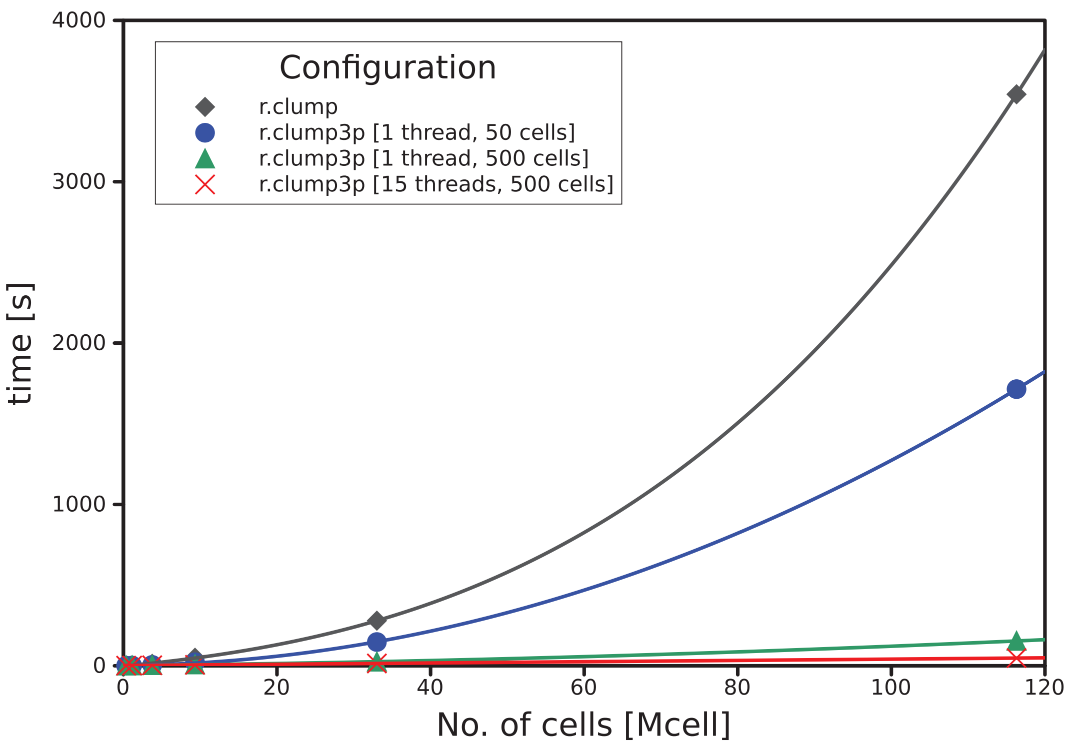

Figure 5: Empirically established dependence of computation time on the size of raster for various setting parameters of *r.clump3* algorithm.

List of Tables

Table 1: Summary of test regions

Name	west	east	south	north	# of cols	# of rows	# of cells
Test_0	631937.32	650924.47	2070152.87	2087111.15	633	565	357,645
Test_1	618484.33	652332.65	2066638.38	2096869.84	1,128	1,008	1,137,024
Test_2	592075.21	653572.51	2063838.06	2118764.07	2,050	1,831	3,753,550
Test_3	581042.95	677994.21	2037352.38	2123943.93	3,232	2,886	9,327,552
Test_4	571882.80	754258.21	1963997.75	2126885.47	6,079	5,430	33,008,970
Test_5	496041.57	838381.87	1889801.01	2195560.59	11,411	10,192	116,300,912
NLCD	-2493045.00	2342655.00	177285.00	3310005.00	161,191	104,425	16,832,787,875

Table 2: Summary of performance of different CCL algorithms on testing regions

Name of test set	Number of cells	r.clump	r.clump3p (1 thread, 50 cells)	r.clump3p (1 thread, 500 cells)	r.clump3p (15 threads, 500 cells)	Number of segments
	[Mcell]	[s]	[s]	[s]	[s]	[#]
Test_0	0.36	1.49	0.78	0.81	0.81	12,502
Test_1	1.14	3.27	1.69	1.61	1.55	31,110
Test_2	3.75	12.20	5.62	3.76	3.28	89,887
Test_3	9.33	49.93	22.09	8.04	5.84	209,422
Test_4	33.01	279.57	97.92	25.11	14.48	532,079
Test_5	116.30	3,541.91	1,714.64	154.01	47.75	1,841,037
NLCD	16,822.78	-	-	-	141,039	221,718,501