Comparison of different implementations of a raster map calculator

Netzel P.[a,*], Slopek J.[b]

[a] *University of Agriculture, al. 29 Listopada 35, Krakow, Poland*
[b] *University of Wroclaw, pl. Uniwersytecki 1, Wroclaw, Poland*

ARTICLE INFO

Keywords:
large rasters
map calculator
benchmark
OpenSource software

ABSTRACT

In the paper, we review selected existing solutions of raster map calculators and propose a new approach for map calculation tools. The main criteria to select raster maps calculators was the ability to run them in batch mode and to use them in external scripts. Such a working method is common in the processing and modeling of massive datasets. We compared the following solutions: r.mapcalc from GRASS GIS, Grid Calculator module in SAGA, gdal_calc.py from GDAL library, and 'calc' function from R raster package. Moreover, we propose another solution – plMapcalc. The solution has new features, such as multiple outputs, multi-pass processing, and a memory buffer to store temporary values. All raster calculators were compared according to their processing efficiency and precision. Two datasets of different sizes were used in the testing procedure, which started with GeoTIFF input files and produced GeoTIFF resultant files.

The results of the test show that the precision of the calculations is comparable. We also compared the processing times of all the calculators using a ranking procedure. The new solution for introducing extra functionalities is the best ranked raster map calculator.

1. Introduction

In raster systems, operations involving algebra layers are extremely important. Using arithmetic operations, it is possible to define the calculation of vegetation indices, implement environmental regression, and calculate the values of statistical models.

In the age of Big Data, the amount of input raster data has grown significantly. It is necessary to perform calculations on raster layers with sizes of 100,000 x 100,000 cells and larger. Many raster data applications require the combination of information stored in multiple raster layers or even the use of data stored in different formats to run complex tasks (Silva-Coira, Parama, Ladra, Lopez and Gutierrez, 2020). This leads to a situation in which either GIS workstations are considered insufficient and calculations are transferred to the cloud, to computing clusters, or the time to receive final results becomes very long, and calculations take hours. Parallel processing can allow faster calculations (Steinbach and Hemmerling, 2012; Yildirim, Watson, Tarboton and Wallace, 2015). Its use in GIS software dates back to 30 years (Guan and Clarke, 2010; Guan, Zeng, Gong and Yun, 2014). Some modern systems successfully use parallel processing, providing analysis results in a short time. Apart from parallel processing, research has also been conducted on alternate ways of reading and writing raster data (Rosario, Bordawekar and Choudhary, 1993; Qin, Zhan and Zhu, 2014), as input/output operations constitute another bottleneck in the processing of spatial data (Yildirim et al., 2015).

In recent years, the efficiency of geospatial data processing programs written in Python, or R and Python (Verbeurgt, Stal, De Sloover, Deruyter and De Wulf, 2020; Strimas-Mackey, 2020) have been analyzed. The interest in these topics is related to the need to process the rapidly growing amount of satellite remote

The contribution of authors: Pawel Netzel — conceptualization of this study, methodology, software, calculations; Jacek Slopek — tests and calculations, data curation, writing – original draft preparation

*Corresponding author
pawel.netzel@urk.edu.pl (N. P.); jacek.slopek@uwr.edu.pl (S. J.)
www.pawel.netzel.pl (N. P.)
ORCID(s): 0000-0003-4151-4908 (N. P.); 0000-0002-7194-3802 (S. J.)

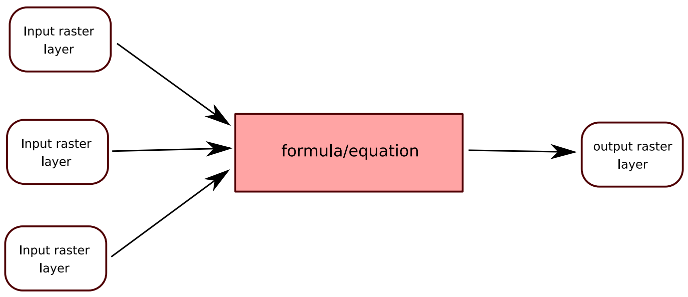

Figure 1: Basic calculation flow implemented in a raster calculator

sensing and aerial data. In this context, the biophysical spectral indices are also emphasized as a source of knowledge on the changes taking place in the natural environment (Achhab, Raissouni, Azyat, Chahboun and Lahraoua, 2010; Liu, Feld, Xue, Garcke, Soddemann and Pan, 2016; Saribekyan, 2013).

This study provides an overview of selected raster calculators of modern, open GIS systems and independent programs for raster map algebra As part of the comparative analysis, the currently used GIS systems and programs that are in a position to be ready to work with big data were identified. This study also presents a new solution in the field of raster calculators that allows complex calculations with large amounts of spatial data to be performed efficiently. This solution introduces new features to extend the range problems that can be solved with raster calculators (Netzel, 2020). The need for such software arises during the processing of large data files. When a user has to go beyond simple calculations completed by standalone raster calculators, the data needs to be imported into a GIS database to use GIS system functionalities, or a dedicated software needs to be developed. Moreover, such dedicated software must be optimized for processing time and memory usage. A convenient tool that can rapidly perform complex calculations with limited memory resources is therefore necessary. plMapcalc largely solves these problems.

All tested software is freely available under open licenses.

Raster map calculators were tested using local functions (Tomlin, 1990), i.e. those in which the input cells occupy the same position in the layer. The formulas used included basic arithmetic operations like addition, multiplication, division, subtraction, and simple mathematical functions, that can be implemented in any GIS system.

This paper has two main aims. The first is to introduce the new raster calculator that satisfies three requirements:

- it is easy to use in scripts and works with commonly used data formats (no import/export operations required);
- it has the ability to make complex calculations;
- it is fast.

The second aim is to compare five different map calculators using 2 datasets: medium size (ca. 10^8 raster cells) and large size (ca. 10^{10} raster cells). The testing procedure took a GeoTIFF raster files as an input and created an output GeoTIFF raster file(s). If an import or an export was necessary, the time of such procedures was taken into account. The new proposed solution – plMapcalc – is included in the raster map calculator's comparison procedure.

2. Raster map calculators

In general, currently used raster map calculators work based on a standard scheme:

$$new\_raster\_layer = f(input\_raster_1, input\_raster_2, \ldots)$$

where $f()$ is a function, equation, or formula that computes the cell values of the new layer.

The flow of the calculation process is illustrated in Figure 1. The calculator takes one or more input layers, applies a formula and stores the result in the output layer. In some systems, the user can run such processes in parallel.

It is necessary to use dedicated modules or functions to implement more complex algorithms. If they are missing, scripts need to be written. Moreover, existing solutions usually limit the computational possibilities for storing the result in only one newly created layer.

In this paper, the authors consider raster calculators that allow the user to work in batch mode and perform calculations using external scripts. This selection was dictated by the fact that this method is used most often when processing large data sets. The selected calculators can work on Windows or Linux.

The list of tested raster map calculators is as follows:

- plMapcalc - a new standalone raster calculator based on Tiny C Compiler (TCC) and GDAL libraries.
- gdal_calc.py - a raster calculator included in the GDAL toolset;
- r.mapcalc - a raster calculator from GRASS GIS;
- SAGA - a raster calculator built-in SAGA GIS software;
- R raster calc - a raster calculator from raster library in the R software;

2.1. plMapcalc (proposed solution)

The proposed solution provides new functionalities for map algebra calculus. The idea of the map calculator extension was modeled on the capabilities of the 'awk' text file processing program. In 'awk', it is possible to define the BEGIN script that should be run before scanning a file, the END script that can be used to summarize the results of file processing, and variables (and arrays) that are accessible during processing. It is also possible to restart the processing of a file.

plMapcalc uses a similar approach to raster data processing. Owing to such extensions and new features, plMapcalc can go beyond standard map one-pass and one-output formula calculations and enables a user to solve a wider range of problems (Netzel, 2020). The main idea behind this solution was to create a tool for issues more complicated than those having four arithmetic operations and less complex than spatial algorithms (such as spatial segmentation). Moreover, the new calculator should be fast enough to deal with large input files (10^{10} or more raster cells) in a reasonable time (see Table 1).

The properties of the proposed solution are as follows:

- entered calculation scripts are compiled into native machine code before execution,
- one or more result maps are generated in one run;
- calculations are conducted using input maps without the need to create the output layer;
- to propagate results of calculations from one cell to another, the formula can use a memory buffer with a user-defined size;
- to store partial results between program calls, the memory buffer can be written to or loaded from a text file;
- the user can enter three scripts: executed before starting the calculations, performed for each of the raster cells, and performed after the calculations are finished
- it is possible to restart layer calculations at any moment, allowing input layers to be scanned multiple times.

Owing to the above features it is possible to do calculations from simple raster map algebra or reclassification to building statistical models, and solving differential equations. plMapcalc is written in C language. It can be built with GNU C Compiler or Visual Studio Express. The parallel processing in plMapcalc is done with the help of OpenMP library version 5. Formulas for calculations are written in a language compatible with the ANSI C language. plMapcalc compiles these formulas into machine code using the TCC compiler library – Tiny C Compiler (Poletto, Hsieh, Engler and Kaashoek, 1999). plMapcalc provides an environment to run such compiled script code, feeds inputs, and stores outputs in files. It runs in command line mode and lacks a graphical user interface.

TCC is a swift, portable, and small C compiler. The TCC compiler is ANSI C compliant and can work as a compiler, C code script interpreter, and a C code compilation library. The last ability of TCC is used in plMapcalc. plMapcalc compiles ANSI C macros into machine code on-the-fly and runs it for consecutive cells.

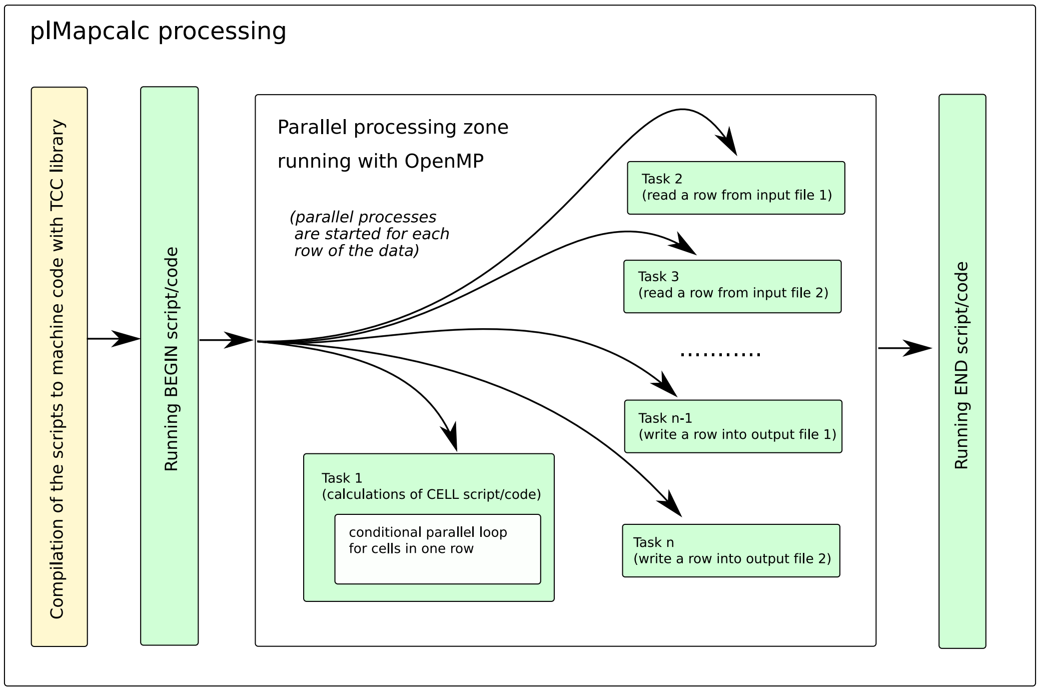
plMapcalc processing
Compilation of the scripts to machine code with TCC library
Running BEGIN script/code
Parallel processing zone running with OpenMP
(parallel processes are started for each row of the data)
Task 2 (read a row from input file 1)
Task 3 (read a row from input file 2)
....
Task n-1 (write a row into output file 1)
Task n (write a row into output file 2)
Task 1 (calculations of CELL script/code)
conditional parallel loop for cells in one row
Running END script/code

Figure 2: plMapcalc calculator's internal processing

2.1.1. Internal architecture of plMapcalc

plMapcalc can work with extensive data in a reasonable time. To limit memory requirements, plMapcalc processes the input data row-by-row, which allows the storage of only a small amount of the input file in the memory. As plMapcalc uses numbers in double-precision floating format to store the data, the memory footprint of each data file in bytes is eight times the number of columns. If plMapcalc runs with n input/output layers, and each layer is w cells wide and h cells high, then the memory requirement is $O(w * n + n * c)$, where c is the size of control structures necessary to open the GDAL dataset.

plMapcalc speeds up the processing by:

- compiling processing scripts to native machine code,
- running the code in parallel with OpenMP library.

Figure 2 illustrates internal processing implemented in plMapcalc.

The scripts entered by a user are compiled to a binary processor code. The library of Tiny C Compiler provides the ability to create binary machine code at the runtime. Internally, the scripts become bodies of functions that will be called at the processing start, for each cell, and at the end of the processing.

The primary process can be run in parallel. This is an option, and its usability depends on the hardware: the number of physical threads of the CPU and hard drive throughput. The first step of parallel processing is performed on the disk's read/write procedure level. Reading and writing of each file and row processing are done in parallel. The OpenMP tasks perform reading a row, writing a row, or calculating the current row. Such an approach requires the number of threads to be equal to or greater than the number of input and output files plus one extra thread for calculation.

The second step of parallelization is done during row processing. This step is conditional. It is used in the situation when the user does not declare a memory buffer to propagate calculation results from one cell to another. This buffer extends a range of algorithms that can be implemented in plMapcalc scripts, but it is a bottleneck in process parallelization.

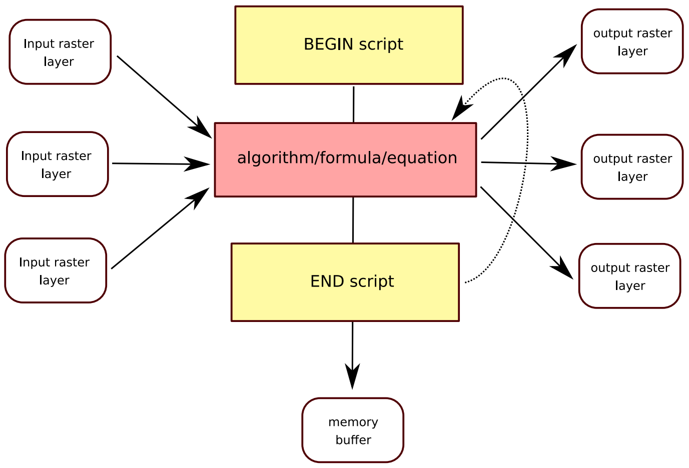

Figure 3: Proposed method of processing in the plMapcalc raster layer calculator

2.1.2. Workflow of plMapcalc

From a user point of view, the way plMapcalc functions is illustrated in Figure 3. The program accepts three scripts:

1. a STARTUP script, which is executed before starting data reading (upper yellow box);
2. a CELL script that is executed sequentially for each cell in the calculation region (red box);
3. an END script that is called after reading and calculating the input data (lower yellow box).

In the CELL script, the user can refer to input values through the IN array, to the output layers through the OUT array, and to the declared shared memory buffer through the MEM array. As OUT is an array, the user can define multiple output layers. That reduces the need to read input layers numerous times to calculate a set of resultant layers. plMapcalc allows the calculation of, for example, several vegetation indices at once. Such an approach is more efficient than iterating formulas that call on successive indices.

plMapcalc has an implemented memory buffer. This buffer is available in the calculation of each raster cell, and the user defines its size. The memory buffer enables storing, counting, summing, etc., values from raster cells or interim results. The buffer can be saved to a text file after the computation ends. In this way, the final results are stored even though the user does not specify an output raster layer. In such a case, plMapcalc works as a tool for calculating layer statistics, regression models, etc. Moreover, the memory buffer can be used for a quick reclassification of the input data.

When plMapcalc finishes the processing of the entire input, plMapcalc calls the END script. In this script, the user can do additional calculations based on the results stored in the memory buffer. If the algorithm requires it, the user can retrigger the calculations of input data from the beginning. During the next data scans, the BEGIN script is omitted.

Both the input data and the memory buffer are represented in the scripts in double format.

A set of examples that shows how to use plMapcalc is available on plMapcalc's web page (`http://plmapcalc.netzel.pl`). These examples are ordered from simple map algebra to complex classification problem.

2.1.3. Example of plMapcalc usage

To illustrate how plMapcalc can be applied, we show a simple example. The aim of the calculations is to nomalize air temperature to the range from 0 to 1. The input layeras are following t01.tif, t02.tif, ... , t12.tif. Each layer contains monthly averages of the air temperature. plMapcalc calculates global minimum and maximum air temperature. Next, it recalculates temperatures. The ouput consists of a set of layers.

The following macros do the calculations described above.
The BEGIN macro - file "init.mc":

```
 // the memory cells MEM[0] and MEM[1] will contain minimum and
 // maximum temperature respectively
 MEM[0] = 9999; //initial value for minimum
 MEM[1] = -9999 //initial value for maximum
```

The CELL macro - file "calc.mc":

```
int i;
// checking iteration number
if(ITERATION()==1) {
  // the first data scan
  //   reading the data from input layers
  for(i=0; i<INPNUM; i++) {
    // IN[i] contains value of i-th input layer
    // looking for global minimum
    if(IN[i]<MEM[0]) MEM[0] = IN[i];
    // looking for global maximum
    if(IN[i]>MEM[1]) MEM[1] = IN[i];
  }
} else {
  // the second data scan
  //   calculating
  for(i=0; i<INPNUM; i++) {
    // OUT[i] represens the cell of i-th output layer
    OUT[i]=(IN[i]-MEM[0])*MEM[1];
  }
}
```

The END macro - file "print.mc":

```
if(ITERATION()==1) {
  //the first data scan
  //  printing minimum and maximum
  printf("Global minimum: %.2lf\n", MEM[0]);
  printf("Global maximum: %.2lf\n", MEM[1]);
  //  calculate the scaling factor
  MEM[1]=1/(MEM[1]-MEM[0]);
  //  calling the restart of the data scanning
  RESTART();
}
```

plMapcalc should be run with following parameters to do the calculations:

```
  plMapcalc --memory=2
                -i t01.tif -i t02.tif -i t03.tif -i t04.tif -i t05.tif
                -i t06.tif -i t07.tif -i t08.tif -i t09.tif -i t10.tif
                -i t11.tif -i t12.tif
                -o n01.tif -o n02.tif -o n03.tif -o n04.tif -o n05.tif
                -o n06.tif -o n07.tif -o n08.tif -o n09.tif -o n10.tif
                -o n11.tif -o n12.tif
           --program-begin=init.mc
           --program=calc.mc
           --program-end=print.mc
           --threads=6
```

The parameter *memory* creates MEM array with two cells. At the beginning MEM array contains zeroes. The set of i parameters defines input layers. The IN array contains values from the input layers. In a similar way, the set of o parameters defines output layers. In the example, the output data will be stored in the default format. The user can define data type, no-data value, and compression level (see plMapcalc manual). The parameters with *program* prefix specify script files. The last parameter – *threads* determines the number of threads that will be used by plMapcalc.

2.2. GDAL (gdal_calc.py)

The calculator included in the GDAL library (Warmerdam, 2008) (which is a widely used open-source tool for manipulating spatial data) is gdal_calc.py. This tool is written in the Python language and is optimized for matrix computation (n-dimensional array objects) of the NumPy library, on which it depends. Thanks to this dependency, it has the speed of the NumPy library, which is written in C. gdal_calc.py reads input data and optimizes the calculation by trying to naturally divide data into blocks. gdal_calc.py can use data selection criteria to limit calculations and process the data that satisfies the given conditions.

2.3. GRASS (r.mapcalc)

r.mapcalc is a map algebra calculator that is part of GRASS (Geographical Resources Analysis Support System) software (Neteler and Mitasova, 2008). GRASS is one of the oldest Open Source GIS solutions. r.mapcalc enables the calculation of raster layers stored either in the internal spatial database of the GRASS system or the external raster layer made available through the GDAL library (and r.external tool). The external file (layer) must first be registered in the internal GRASS database. r.mapcalc provides a set of mathematical functions, logical operators, enables the definition of temporary variables, and allows multiple output layers to be created in one run.

It also allows operations in the surroundings - neighbors of the raster cell. A built-in C-based interpreter interprets the entered calculation formula. Moreover, r.mapcalc can work with multiple inputs and creates more than one output simultaneously (by using grouping and piping functions). According to r.mapcalc developers, it can accelerate calculations.

2.4. SAGA (Grid Calculator)

SAGA (Conrad, Bechtel, Bock, Dietrich, Fischer, Gerlitz, Wehberg, Wichmann and Böhner, 2015) provides a module library called Calculus for performing raster calculations. This library includes a Grid Calculator that allows a user to generate a new raster in the internal SAGA (grid) format. In the calculations, the user can use the functions implemented in the module and applies them to the grid files imported into the system. The Grid Calculator allows calculations with on-the-fly resampling (four interpolation types have been built for this purpose). The calculator saves output grids with nine data types (from Byte to Double Floating Point Precision). The use of SAGA saga_cmd - the SAGA frontend that uses the command line to access modules - saga_cmd - requires additional steps to be taken, such as data registration in the internal SAGA structure, and export to an external format (e.g. GeoTiff). SAGA is coded in C++ and uses OpenMP parallel processing library.

2.5. R raster (calc from raster library)

In the R system, the library 'raster' is used to perform calculations on spatial raster data. The library provides tools for performing calculations (function calc), preprocessing, and exporting calculation results to external files (function writeRaster). The raster library offers a class of R objects based on external spatial files. The system recognizes valid data sources among supported formats (including GeoTiff files). Such objects can be combined using the stack function into the RasterStack object, grouping files of the same resolution and size. The created RasterStack object can be converted to a multilayer raster RasterBrick object to accelerate calculations.

Moreover, the 'raster' library, with the help of 'clusterR' library, can accelerate calculations performed using the 'calc' function with the use of multiple processor cores.

3. Datasets and calculation procedure

Two data sets were selected for the calculations and performance tests of the raster calculators. The first is the scene recorded by the Sentinel-2B satellite. It is a relatively small dataset (120 million cells). The second set is an orthophoto map in CIR (Color Infrared, a combination of R, G, and NIR bands), created based on aerial photographs. This set was composed of approximately 5 billion raster cells with stored values (non-null values).

3.1. Sentinel-2B data

Four bands of the scene recorded by the Sentinel-2B satellite over the territory of Poland, covering over $12000km^2$, were selected for testing. The scene selected for the tests has a spatial extent from $17^o48'E$ to $19^o27'E$ and from $50^o25'N$ to $51^o24'N$. The test data consisted of four bands with a resolution of 10 m: Band 4 (664.9 nm, red), band 8 (832.9 nm, NIR), band 3 (559 nm, green), and band 5 (703.8 nm, Red Edge). The scene size for the recorded bands with a 10 m resolution consists of 10980 x 10980 (120 560 400) cells. The test scene was an L2A level product. The data used for the calculations were recorded in the UTM zone 33N geodetic projection (EPSG: 32633).

The test scene files were converted from the JPEG2000 format (as provided by the Copernicus Open Access Hub) to the non-compressed GeoTIFF format.

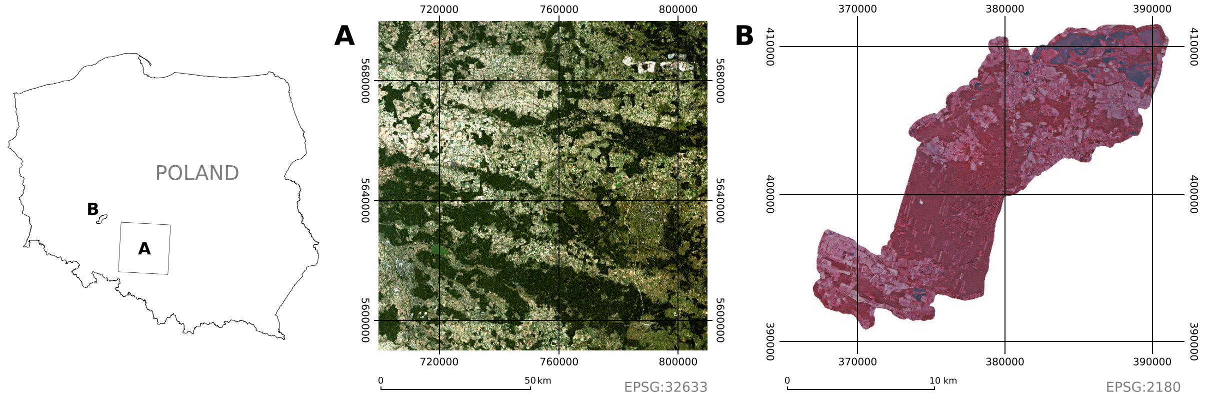

Figure 4: Datasets used for testing: A – Sentinel-2B scene, B – orthophoto map, Milicz forest district

The RGB composition of the scene selected for testing is shown in Figure 4 A.

3.2. Orthophoto map

The source aerial images for the orthophoto map were acquired over the Milicz forest district (Poland) (Figure 4 B) with an area of 181.94 km^2.

The digital orthophoto map was created by combining aerial photos taken with the Z/I DMC-II 230 aerial camera. The resulting image has a spatial resolution of 0.2 m per pixel and is a CIR composition containing the NIR band (670-910 nm), red band (570-730 nm), and green band (470-620 nm). The individual bands provided information in 8 bits per pixel (values 0-255). Image data was compressed using the JPEG method.

The orthophoto map is a mosaic composed of 63 scenes covering rectangular areas of approximately $5km^2$ in size. As a whole, the selected set of connected fragments has dimensions of 132680 x 115000 pixels (approx. 15.2 billion cells, approx. 26.5 x 23 km). The Milicz forest district constitutes approximately 35% of this area. The test data contained approximately 5.2 billion raster cells with an assigned value (not-null values).

The orthophoto map data was stored in the projected coordinate system for Poland — PUWG 1992 (EPSG: 2180).

3.3. Vegetation indices

A series of calculations were made on the test data to compare the performance of raster map calculators. Three spectral indices often used in natural sciences were used as test formulas:

3.3.1. Normalized Difference Vegetation Index, NDVI

$$NDVI = \frac{NIR - RED}{NIR + RED}$$

where
NIR - Near Infrared band, RED - RED band

NDVI is one of the most frequently used vegetation indices (Rouse, Haas, Schell and Deering, 1973; Silleos, Alexandridis, Gitas and Perakis, 2006) in nature, agricultural, and forest research. Its popularity is due to the simplicity of the mathematical formula, and therefore the speed with which it is possible to identify areas covered with vegetation containing chlorophyll, or to determine vegetation conditions in a given area.

The popularity of the NDVI is also due to the elimination of the area differentiated lighting problem related to the topography and the incidence angle of sunlight.

3.3.2. Corrected Transformed Vegetation Index, CTVI

In 1975, Deering et al. (Deering, Rouse, Haas and Schell, 1975) proposed a modification of the formula allowing the calculation of the NDVI by creating the TVI (Transformed Vegetation Index):

$$TVI = \sqrt{NDVI + 0.5}$$

The TVI allows the Poisson distribution, which approximates the NDVI distribution, to be changed to normal distribution. Additionally, entering a value of $+0.5$ eliminates some negative values. With NDVI values <-0.5, however, the TVI cannot be calculated. To solve this problem, Perry and Lautenschlager (Perry and Lautenschlager, 1984) proposed the Modification of the TVI formula by introducing the CTVI:

$$CTVI = \frac{NDVI + 0.5}{|NDVI + 0.5|}\sqrt{|NDVI + 0.5|}$$

In this study, the CTVI formula proposed for the Sentinel-2B satellite in the online database of vegetation indices indexdatabase.de was used:

$$CTVI = \frac{RI + 0.5}{|RI + 0.5|}\sqrt{|RI + 0.5|}$$

where
RI - Normalized Difference Red/Green Redness Index;

$$RI = \frac{RedEdge - Green}{RedEdge + Green}$$

In the case of the orthophoto map, the red band from the CIR image was used in place of the Sentinel's Red Edge band to calculate the CTVI as there was no Red Edge band on CIR data.

3.3.3. Soil Adjusted Vegetation Index. SAVI

The SAVI, which is another modification of the NDVI formula was introduced in 1988 by Huete (Huete, 1988). SAVI allows the minimization of the influence of soil on the signal coming from areas covered with vegetation. In the formula, the variable L was introduced. The value of L is selected depending on the density of the vegetation to be analyzed. L-values change with soil characteristics. The formula to calculate the index is:

$$SAVI = \frac{NIR - RED}{NIR + RED + L}(1 + L)$$

where
L – soil brightness factor with values ranging from -0.9 to 1.6 (usually 0.5 for intermediate vegetation density).

3.4. Calculation procedure

The NDVI was chosen due to the simple structure of the formula, requiring only three arithmetic operations. The CTVI additionally involves the calculation of the absolute value and the square root. The SAVI was chosen due to the possibility of performing a series of calculations in which the variable L has a value in the range from 0.9 to 1.6. SAVI was computed for L from the above range in the testing procedure, with a step of 0.1. This gave a total of 26 iterations of the loop. The results of all tests were stored in files as the Float32 type.

The iterative calculation of the SAVI also allows the use of alternative paths to perform this operation. In the case of GRASS GIS, both r.mapcalc's ability to generate many result layers in one calculator call, and the for loop in shell script were used to create a single result layer numerous times. A similar procedure was used in the case of the plMapcalc calculator, which allows the calculation of many result files during one program call. With the gdal_calc.py and SAGA calculators, it was only possible to use bash loops. The iterations necessary to compute the SAVI were performed with the internal for-loop command in the R system.

The input data files were not compressed. The calculations were performed on a single processor core. In the case of calculators with the ability to enable parallel processing (R and SAGA), the calculations were done additionally using all possible cores of the test computer.

If the calculator required data to be imported into a specific internal database format before performing the calculations, the import operation time was also calculated. The time of exporting the calculation results to external files and all additional operations performed in the tested GIS environment were also counted separately. For example, a GRASS GIS system in batch mode requires a system database structure (location and mapset) to be created on the fly. In the R system, additional operations are as follows: creating a group of layers using the stack function or creating a multi-layer object using the brick function.

3.5. Testing environment

Computational tests were performed on a computer with an i7 processor, 32 GB RAM, 10 TB HDD running under Linux Fedora 32. A set of console scripts was prepared to test the selected raster calculators. The runtime of raster calculator calls was measured with the GNU "time" utility. In the case of R scripts, the computation time of the raster calculator calls was determined using the Sys.time() function.

The following versions of software were used: GDAL 2.3.2, GRASS GIS 7.6.0, SAGA 2.3.1, plmapcalc 1.2.422, R 3.6.3 with libraries raster (3.3.7) and sp (1.4.2). All programs but plMapcalc were installed from the Linux Fedora system repository.

4. Results

During the calculations, 414 result files for the satellite scene and 244 files for orthophoto maps were created with a total size 14.4 TB (after compression approximately 3 TB). The resulting data were stored with a single precision. The small dataset – Sentinel-2B satellite scene – did not cause any problems in calculation. All calculators finished all operations, i.e. opening, processing the data, and storing (export) the results.

When calculations of indices were performed with an orthophoto dataset, two calculators did not generate results: SAGA and 'calc' from the R system. The large files exhausted computational resources required by these calculators. Other calculators were able to handle the amount of data and produced results.

4.1. Calculation efficiency

Two map calculators – gdal_calc.py and plMapcalc – work on GeoTIFF files directly, and there was no conversion necessary. The R 'calc' function requires the creation of a RasterStack object and an extra export procedure. SAGA and GRASS raster calculators need input data imported to the native GIS databases. These two calculators also need the results to be exported from the internal database to the external raster file. The data converted to the internal format allows faster access and accelerates the calculation. The GRASS calculator also has the possibility to work on layers in external mode. In this mode, the data are stored in external GeoTIFF files and these files are registered in the GIS database for the calculation times. There is no need to run an export procedure as r.mapcalc saves the results to the external file directly.

Each raster calculator can be run in different configurations (multiple outputs, parallel processing, etc.). The calculation times for all configurations are shown in Figures 5 and 6. The meaning of processing procedure names on these figures:

1. gdal_calc - gdal_calc.py was run on one CPU core
2. GRASS - r.mapcalc was run on the internal GRASS database
3. GRASS (ext) - r.mapcalc was run on external GeoTIFF files
4. GRASS (multi) - r.mapcalc was run on the internal GRASS database with improved data access
5. GRASS (ext, multi) - r.mapcalc was run on the external GeoTIFF files with improved data access

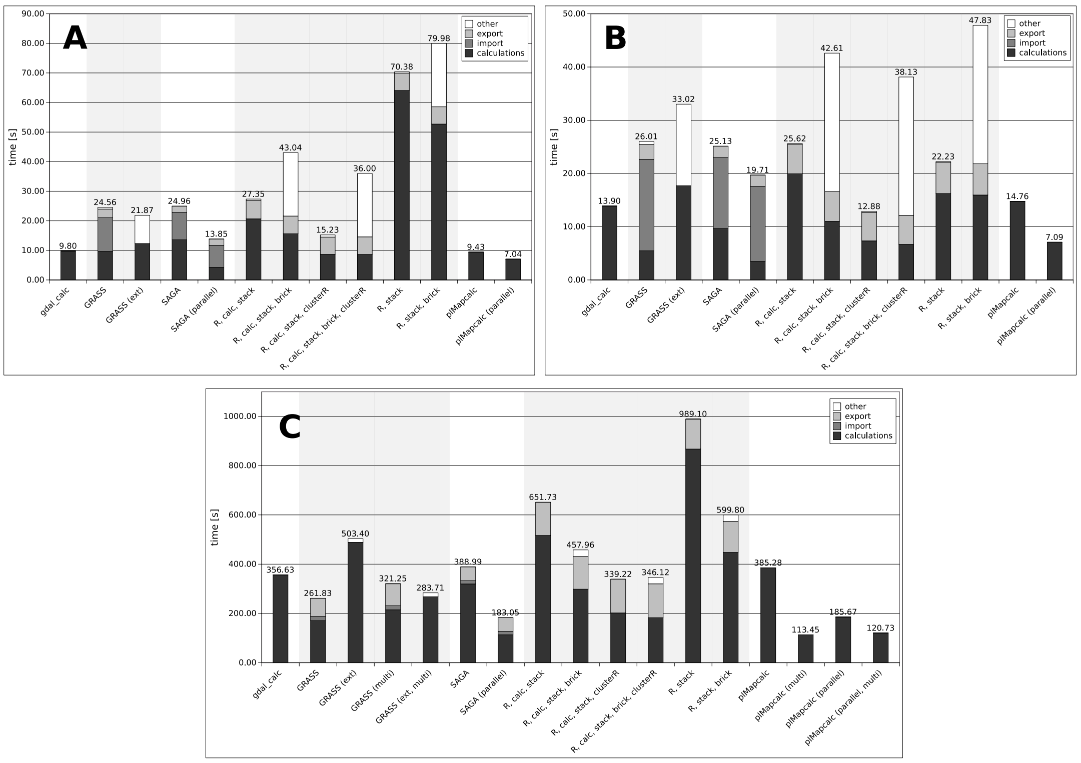

Figure 5: Operations times of CTVI (A), NDVI (B), and SAVI (C) calculation for Sentinel-2B scene. Total processing time is divided into calculation, import, export, and other tasks. The grey segments separate different raster calculators.

6. SAGA - SAGA calc was run on one CPU core
7. SAGA (parallel) - SAGA calc was run on 12 CPU cores
8. R, calc, stack - calc function run on a RasterStack object
9. R, calc, stack, brick - calc function run on a RasterBrick object
10. R, calc, stack, clusterR - calc function run on a RasterStack object in parallel mode
11. R, calc, stack, brick, clusterR - calc function run on RasterBrick object in parallel mode
12. R, stack - the formula was calculated on a RasterStack object
13. R, stack, brick - the formula was calculated on a RasterBrick object
14. plMapcalc - plMapcalc was run on one CPU core
15. plMapcalc (multi) - plMapcalc was run on one CPU core and created multiple output files in one run
16. plMapcalc (parallel) - plMapcalc was run on 12 CPU cores
17. plMapcalc (parallel, multi) - plMapcalc was run on 12 CPU cores and created multiple output files in one run

To compare the calculation efficiency, two parameters are considered: total processing time, and calculation time. Total processing time is important when a user needs to start with a file in a common raster format and wants to obtain a similar file as a result of the processing. The calculation time is more relevant if a user limits his/her activity to one GIS system and does not want to export the calculation results.

The total processing time differs significantly between calculators. For Sentinel-2B data, the best CTVI total processing time varies from 7.04 s for plMapcalc to 21.87 s for GRASS r.mapcalc (Figure 5 A). For all calculators but plMapcalc, NDVI calculations were longer than CTVI calculation – 12.88 to 26.01 s. plMapacal did this calculation in a comparable time (7.09 s) to the CTVI calculation (7.04 s) (Figure 5 B). SAVI was a batch calculation test of a set of output layers. The calculations can be done in serial mode in a

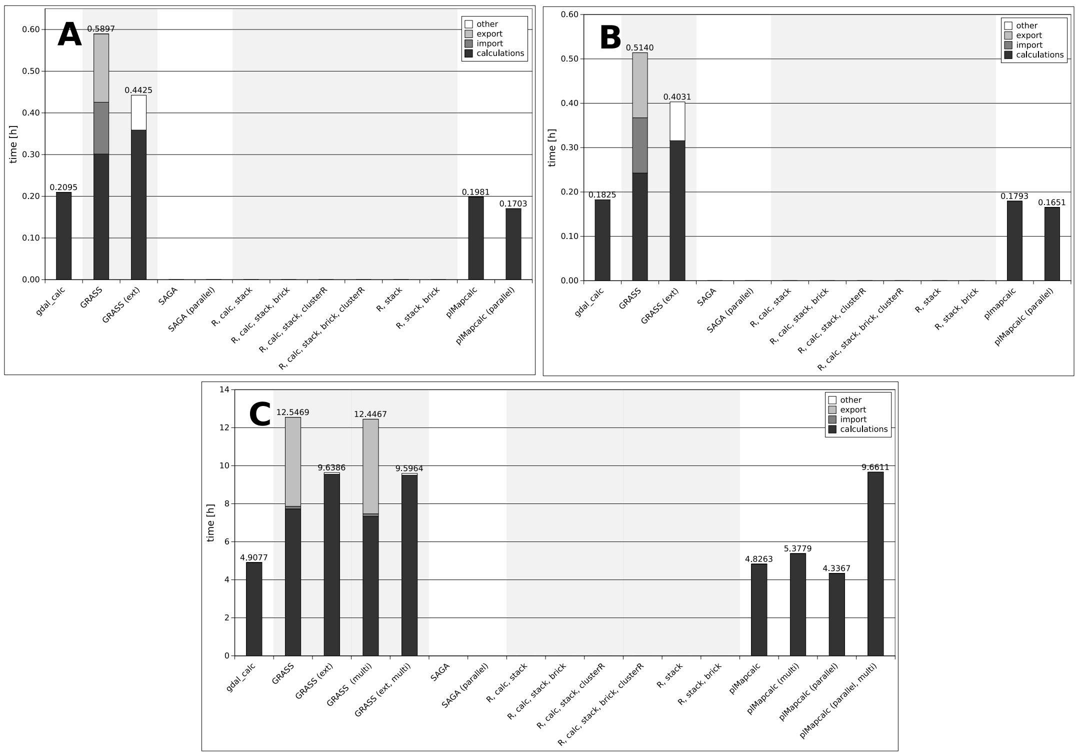

Figure 6: Operations times of CTVI (A), NDVI (B), and SAVI (C) calculation for the orthophoto map. The total processing time is divided into the calculation, import, export, and other tasks. Some bars are missing because the software could not process such a large file. The grey segments separate different raster calculators.

loop or using multiple outputs of raster calculators if the calculator provides such an option. plMapcalc was the most efficient (113 s) and overperformed the slowest solution by three times – gdal_calc.py – 356.63 s (Figure 5 C).

In general, gdal_calc.py has a similar total processing time as plMapcalc, but other calculators are slower. gdal_calc.py and plMapcalc do not need any extra import/export operations. In the case of GRASS and SAGA, import and export procedures add significant overhead, up to 3 times longer than the calculation itself. The 'raster' library in the R system can import the data to StackRaster object quickly but writing the output file is expensive in terms of time and adds approximately 30% to the calculation time.

Problems can arise with increasing amounts of data. R and SAGA cannot handle orthophoto map data to produce results. The total processing time varies from minutes for CTVI and NDVI to hours for SAVI. plMapcalc was the most efficient. A similar result was obtained by gdal_calc.py. GRASS (r.mapcalc) gave the worst results (Figure 6).

The best total processing and pure calculation times for each raster map calculator are presented in Table 1.

To compare the overall performance of all calculators, we conducted a ranking procedure. The calculated ranks are presented in the lower part of Table 1 and in Figure 7. In the case of issues with large data files and a lack of results, the raster calculator was ranked as the last. plMapcalc outperforms other calculators in the ranking. In the calculation time ranking, the advantage of the dedicated GRASS and SAGA GIS databases can be noticed. The R system is ranked as being the worst.

Table 1
Best total processing times, and best calculation times with ranking

	Total processing time					Calculation time				
	GDAL	GRASS	SAGA	R	PlMapcalc	GDAL	GRASS	SAGA	R	PlMapcalc
Sentinel-2B [s]										
CTVI	9.80	21.87	13.85	15.23	7.04	9.80	9.64	4.29	8.62	7.04
NDVI	13.90	26.01	19.71	12.88	7.09	13.90	5.48	3.48	6.68	7.09
SAVI	356.63	283.71	183.05	339.22	113.45	356.63	170.88	113.32	182.97	113.45
Ortophoto map [h]										
CTVI	0.2095	0.4425	—	—	0.1703	0.2095	0.3013	—	—	0.1703
NDVI	0.1825	0.4031	—	—	0.1651	0.1825	0.2427	—	—	0.1651
SAVI	4.9077	9.5964	—	—	4.3367	4.9077	7.7285	—	—	4.3367
	Ranks									
Sentinel-2B										
CTVI	2	5	3	4	1	5	4	1	3	2
NDVI	3	5	4	2	1	5	2	1	3	4
SAVI	5	3	2	4	1	5	3	1	4	2
Ortophoto map										
CTVI	2	3	4	4	1	2	3	4	4	1
NDVI	2	3	4	4	1	2	3	4	4	1
SAVI	2	3	4	4	1	2	3	4	4	1
TOTAL	16	22	21	22	6	21	18	15	22	11

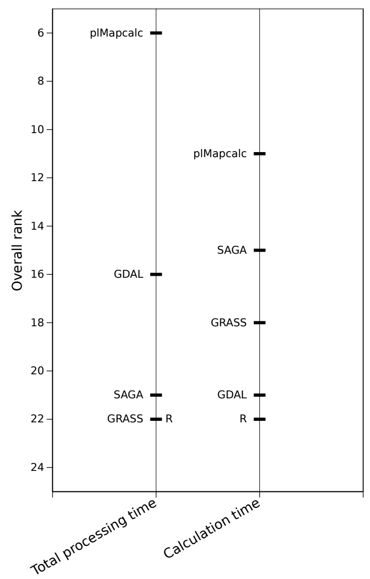

Figure 7: Final ranks of raster map calculators based on total processing time (left), and calculation time (right). A lower rank value and higher position on the graph indicates better performance.

4.2. Calculation precision

The results of all calculations were stored. In the next step, we compared resulting layers with each other. The maximum difference, MAE, and RMSE were calculated. For all comparisons, these statistics took a value of 0 with a single floating-point precision. The only difference was for treating the results of the division by zero. According to IEEE 754 (IEEE, 2019), standard division by zero may result in INF or NAN. All calculators except plMapcalc treated these values as no-data value. In plMapcalc, there is an option to choose whether these values should be no-data or real values in further calculations.

5. Discussion

Analyzing raster map calculators is a challenging task. Each calculator has its advantages and disadvantages. Some of them are part of GISs, while others are standalone.

In this paper, a new calculator is presented – plMapcalc. This solution tries to find a balance between speed of calculation, direct accession of many raster formats, and the ability to perform complex analysis.

Thanks to the memory buffer and multiple input file scanning options, it can be used to standardize raster map calculus and perform reclassification, modeling, or statistical calculations. All these possibilities can be used without a loss of calculation speed or significant requirement of memory.

The raster calculator's comparison according to the processing efficiency and data size scalability shows that plMapcalc is ranked as the most efficient calculator and it can handle large datasets.

For the remaining calculators, the test results show that for advanced modeling of relatively small datasets, 'calc' from the R system is the right choice. SAGA is a tool for fast processing of datasets in a GIS environment. The SAGA parallel processing optimization is very impressive. For large datasets, GRASS is the only tested solution in the GIS environment. r.mapcalc works fast with datasets stored in the internal GRASS database. For these files, gdal_calc.py is another tool for raster algebra calculations, and it comes as part of the toolset with the GDAL library. It does calculations almost as fast as plMapcalc.

plMapcalc comes as a standalone program without any GUI. The next step will be providing a Quantum GIS plugin to make this calculator available in the GIS.

At the current stage, plMapcalc can only perform calculations locally. Accessing the neighborhood of a cell will allow the analysis class to be extended to focal operations. This is planned as a next step in the development of plMapcalc.

6. Computer Code Availability

plMapacalc is an open source software written in C and is available under the GNU Public License. It was developed by Pawel Netzel (pawel at netzel dot pl) and can be run under Linux or Windows. The first release of plMapcalc was in 2015. The current version presented in the paper – plMapcalc 1.3 – is available since November 2020.

The main code repository of plMapcalc is available from plMapcalc's web page http://plmapcalc.netzel.pl. The repository contains the source code, binaries, examples, and the manual. It also includes the data and scripts used for calculations in this paper.

References

Achhab, N.B., Raissouni, N., Azyat, A., Chahboun, A., Lahraoua, M., 2010. High performance computing software package for multitemporal Remote-Sensing computations. International Journal of Engineering and Technology .

Conrad, O., Bechtel, B., Bock, M., Dietrich, H., Fischer, E., Gerlitz, L., Wehberg, J., Wichmann, V., Böhner, J., 2015. System for Automated Geoscientific Analyses (SAGA) v. 2.1.4. Geoscientific Model Development doi:10.5194/gmd-8-1991-2015.

Deering, D.W., Rouse, J.W., Haas, R.H., Schell, J.A., 1975. Measuring "FORAGE PRODUCTION" of grazing units from LANDSAT MSS data.

Guan, Q., Clarke, K.C., 2010. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. International Journal of Geographical Information Science doi:10.1080/13658810902984228.

Guan, Q., Zeng, W., Gong, J., Yun, S., 2014. pRPL 2.0: Improving the Parallel Raster Processing Library. Transactions in GIS doi:10.1111/tgis.12109.

Huete, A.R., 1988. A soil-adjusted vegetation index (SAVI). Remote Sensing of Environment doi:10.1016/0034-4257(88)90106-X.

IEEE, 2019. IEEE Standard for Floating-Point Arithmetic. IEEE STD 754-2019. IEEE , 1–84.

Liu, J., Feld, D., Xue, Y., Garcke, J., Soddemann, T., Pan, P., 2016. An efficient geosciences workflow on multi-core processors and GPUs: a case study for aerosol optical depth retrieval from MODIS satellite data. International Journal of Digital Earth doi:10.1080/17538947.2015.1130087.

Neteler, M., Mitasova, H., 2008. Open Source GIS: A GRASS GIS Approach (3rd Edition). doi:10.1186/1476-072X-7-53.

Netzel, P., 2020. The plMapcalc manual. http://pawel.netzel.pl/data/uploads/pdf/mapcalc_manual.pdf.

Perry, C.R., Lautenschlager, L.F., 1984. Functional equivalence of spectral vegetation indices. Remote Sensing of Environment doi:10.1016/0034-4257(84)90013-0.

Poletto, M., Hsieh, W.C., Engler, D.R., Kaashoek, M.F., 1999. 'C and tcc: A language and compiler for dynamic code generation. ACM Transactions on Programming Languages and Systems doi:10.1145/316686.316697.

Qin, C.Z., Zhan, L.J., Zhu, A.X., 2014. How to Apply the Geospatial Data Abstraction Library (GDAL) Properly to Parallel Geospatial Raster I/O? Transactions in GIS doi:10.1111/tgis.12068.

Rosario, J.M., Bordawekar, R., Choudhary, A., 1993. Improved parallel I/O via a two-phase run-time access strategy. ACM SIGARCH Computer Architecture News 21(5), 31–8.

Rouse, J.W., Haas, R.H., Schell, J.A., Deering, D., 1973. Monitoring the vernal advancement and retrogradation (green wave effect) of natural vegetation. Progress Report RSC 1978-1 .

Saribekyan, A.G., 2013. Performance of ndvi index on hpc resources. Mathematical Problems of Computer Science 39, 48–53.

Silleos, N.G., Alexandridis, T.K., Gitas, I.Z., Perakis, K., 2006. Vegetation indices: Advances made in biomass estimation and vegetation monitoring in the last 30 years. doi:10.1080/10106040608542399.

Silva-Coira, F., Parama, J., Ladra, S., Lopez, J., Gutierrez, G., 2020. Efficient processing of raster and vector data. PLoS ONE 15(1). doi:10.1371/journal.pone.0226943.

Steinbach, M., Hemmerling, R., 2012. Accelerating batch processing of spatial raster analysis using GPU. Computers and Geosciences doi:10.1016/j.cageo.2011.11.012.

Strimas-Mackey, M., 2020. Raster summarization in python. https://strimas.com/post/raster-summarization-in-python/.

Tomlin, C., 1990. Geographic Information Systems and Cartographic Modeling. Prentice-Hall, Englewood Cliffs, NJ, USA.

Verbeurgt, J., Stal, C., De Sloover, L., Deruyter, G., De Wulf, A., 2020. R and python benchmarking for geographical applications, p. 8.

Warmerdam, F., 2008. The Geospatial Data Abstraction Library, in: Open Source Approaches in Spatial Data Handling. doi:10.1007/978-3-540-74831-1_5.

Yildirim, A.A., Watson, D., Tarboton, D., Wallace, R.M., 2015. A virtual tile approach to raster-based calculations of large digital elevation models in a shared-memory system. Computers and Geosciences doi:10.1016/j.cageo.2015.05.014.