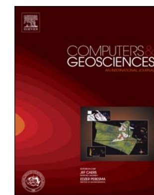
Computers & Geosciences 107 (2017) 20–27

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo



Research paper

Parallelization of interpolation, solar radiation and water flow simulation modules in GRASS GIS using OpenMP

Jaroslav Hofierka[a,*], Michal Lacko[a], Stanislav Zubal[b]

[a] Institute of Geography, Faculty of Science, Pavol Jozef Šafárik University in Košice, Jesenná 5, 04001 Košice, Slovak Republic
[b] Department of Software Engineering, Faculty of Information Technology, Czech Technical University in Prague, Thákurova 9, 160 00 Praha 6, Czech Republic

ARTICLE INFO

Keywords:
Parallel computing
OpenMP
Interpolation
Solar radiation
Water flow simulation
GRASS GIS

ABSTRACT

In this paper, we describe the parallelization of three complex and computationally intensive modules of GRASS GIS using the OpenMP application programming interface for multi-core computers. These include the v.surf.rst module for spatial interpolation, the r.sun module for solar radiation modeling and the r.sim.water module for water flow simulation. We briefly describe the functionality of the modules and parallelization approaches used in the modules. Our approach includes the analysis of the module's functionality, identification of source code segments suitable for parallelization and proper application of OpenMP parallelization code to create efficient threads processing the subtasks. We document the efficiency of the solutions using the airborne laser scanning data representing land surface in the test area and derived high-resolution digital terrain model grids. We discuss the performance speed-up and parallelization efficiency depending on the number of processor threads. The study showed a substantial increase in computation speeds on a standard multi-core computer while maintaining the accuracy of results in comparison to the output from original modules. The presented parallelization approach showed the simplicity and efficiency of the parallelization of open-source GRASS GIS modules using OpenMP, leading to an increased performance of this geospatial software on standard multi-core computers.

1. Introduction

Over the last decades, the advances in multi-core processor architecture enabled parallel processing that can better utilize the power of modern computers. The need for a better computer performance is also stimulated by the increasing availability of massive geospatial data, such as from laser scanning. However, this requires specific software with the ability to employ parallel computational capabilities (Guan and Wu, 2010).

First studies related to parallelization of Geographic Information System (GIS) operations were done by Healey et al. (1998) and Mineter and Dowers (1999). Several parallelization studies were published in the area of digital terrain modeling and analysis (for example, Huang and Yang, 2011; Huang et al., 2011; Schiele et al., 2012; Xie, 2012) and hydrological modeling (Cui et al., 2005; Sten et al., 2016). Still surprisingly, most of the current GIS software products exploit these advances in a very limited way and nearly all operations are executed by a single process. While most GIS operations are computationally simple and therefore executable quickly even by a single process, some geospatial tasks are more complex and computationally very intensive. For example, spatial interpolation techniques, often used in GIS for digital terrain modeling, usually require complex mathematical operations, and only rarely run in parallel. Most GIS operations are serially designed without any consideration of concurrent execution (Guan and Wu, 2010).

There are also various simulation models implemented in GIS ranging from simple models based on map algebra to fully dynamic simulation models such as, for example, hydrological or traffic models that need thousands of iterations to represent the dynamical behavior of the phenomenon. Moreover, these operations must be often run repeatedly with various input parameters in order to get the most accurate result. Thus, the issue of computation speed is very important for most GIS users. However, despite the growing performance of current computers, the tremendous volume of data make their processing using a standard GIS rather problematic. Obviously, the computationally intensive GIS operations originally run in a sequential mode by a single process need a parallelization modification to fully exploit the power of current multi-core systems.

The computer architecture and multiple processor organization determine various parallel approaches. These depend on the number of

* Corresponding author.
E-mail address: jaroslav.hofierka@upjs.sk (J. Hofierka).

http://dx.doi.org/10.1016/j.cageo.2017.07.007
Received 18 December 2016; Received in revised form 29 April 2017; Accepted 29 July 2017
Available online 31 July 2017
0098-3004/ © 2017 Elsevier Ltd. All rights reserved.

processors, access to physical memory and communication (Flynn, 1972). Typically, there are two major types of parallel architectures that use either a distributed memory model or a shared memory model. Distributed memory systems require a communication network to connect the inter-processor memory. In the shared memory model the access to the main memory is via Uniform Memory Access (UMA) or Non-Uniform Memory Access (NUMA). NUMA provides a higher scalability to systems with a higher number of processors (typically more than 8–12), however, it increases the complexity of parallel programming (Stallings, 2010). Recently, high performance computing platforms have increasingly used general purpose computing on graphics processing units (GPGPUs). The graphics processor unit is used to perform computation in applications traditionally handled by the central processing unit. The most common parallel computing platforms based on GPGPU are CUDA and OpenCL (Fang et al., 2011). The ideal GPGPU applications have large datasets, high parallelism exploiting a large number of threads, and minimal dependency between data elements.

There are several commonly used parallel programming approaches depending on the memory model. The distributed memory model uses a message passing system with the Message-Passing Interface (MPI) application programming interface being the most widely used. The shared memory model can be used by various parallel programming tools such as Open Multi-Processing (OpenMP), POSIX Threads, or Cilk. The distributed memory systems have dominated on network-connected computer clusters. Currently, however, most mainstream computers are equipped with multi-core central processing units (CPUs), thus opening the power of parallel processing using a shared memory model to any user with a single computer. In this study, we will explore the implementation of shared memory parallelism for multi-core systems using OpenMP and open-source GRASS GIS.

The OpenMP parallelization tool can be used to make new or modify existing programs to run in parallel simply by adding compiler directives or functions to the original program code. OpenMP (Open Multi-Processing) is an application programming interface (API) that supports shared memory multiprocessing (multithreading) on most common platforms ranging from desktop computers to supercomputers, processor architectures and operating systems (Chapman et al., 2007). It consists of a set of compiler directives, library routines, and environment variables that execute a task in parallel by several threads allocated to different processors (cores). The core elements of OpenMP are the constructs for thread creation, workload distribution (work sharing), data-environment management, thread synchronization, user-level runtime routines and environment variables. In C/C++, OpenMP uses #pragmas.

In geospatial tasks, the concept of parallel computing can be used to shorten computation time via decomposing the geospatial problem into several subtasks that are handled simultaneously by different threads. However, it is quite common that some subtasks depend on completion of some other subtasks. Therefore, different parallelization problems have different implementation schemes for shortening the duration of computation. Parallel algorithms are usually used to partition the temporal/spatial domain into sub-domains. Because almost all natural processes are temporally successive, domain decomposition is mostly carried out spatially (Li et al., 2011).

GRASS is an Open Source GIS with a wide-range functionality and applications (GRASS, 2016; Neteler and Mitasova, 2008; Neteler et al., 2012). It can handle 2D and 3D data in vector or raster format and offers many advanced geospatial modeling algorithms and visualization techniques. GRASS is not a monolithic application, but it rather consists of over 300 modules following the Unix philosophy that each module does a specific task. Therefore, each module is autonomous including a memory management and error handling. The core libraries and most modules are written in POSIX-conforming ANSI C. Some functions are written in the C++ and Python programming languages. Recently, Object Oriented Python Application Programming Interface for GRASS GIS was introduced to expand the capabilities of the software (Zambelli et al., 2013). The software is released under GNU General Public Licence (GPL) (Neteler et al., 2012). A few parallelization studies were done for GRASS GIS. For example, Sorokine (2007) parallelized GRASS GIS visualization using high-resolution tiled displays powered by Linux-based cluster of PCs, and Huang et al. (2011) also used a Linux-based cluster for a parallel inverse distance weighting interpolation algorithm using MPI. In GRASS GIS, the shared-memory parallelization model based on OpenMP has been used in the mathematical (gmath) and partial differential equations (PDE) libraries for matrix and vector calculations and linear equation solvers which are used by some GRASS GIS modules, such as *r.gwflow* or *r.solute.transport* (GRASS, 2016). There are still many computationally demanding modules that run using a single process and thus do not exploit the full power of the current multi-core technology. For example, the *r.sun* solar radiation module was used in several studies with large datasets representing Europe (Šúri et al., 2005) or large regions such as Andalusia (Romero et al., 2008). Pintor et al. (2015) have found the speed of *r.sun* to be a limiting factor for its application to high-resolution DEMs. A similar situation can be found for other modules, such as *r.sim.water* for dynamic hydrologic simulations or *v.surf.rst* for spatial interpolation that also require a lot of time to process large datasets.

The goal of this paper is to present our shared-memory parallelization approach based on OpenMP and applied to three existing and computationally very intensive modules of GRASS GIS with different functionality. These modules perform frequent geospatial operations: spatial interpolation (*v.surf.rst*), solar radiation modeling (*r.sun*) and hydrologic simulations (*r.sim.water*). The parallelized modules will be applied to the test area represented by massive airborne laser scanning data to document the efficiency of the parallel implementation.

2. Methods and data

2.1. OpenMP

The OpenMP standard was defined in 1997 as an API for writing portable, multithreaded applications. The multithreading is a method of parallelizing in which a master thread (a series of instructions executed consecutively) forks a specified number of slave threads and the system divides a task among them. The threads then run concurrently while being allocated to different processors.

The OpenMP programming model provides a set of compiler directives, function calls, and environment variables that instruct the compiler how and where to use parallelism in the application. The main task of OpenMP is to create and manage threads, distribute the tasks and manage the data environment. The directive based approach makes it possible to write sequentially consistent codes for easier maintenance. The well-known advantage of OpenMP is its global view of application memory address space that allows relatively fast development of parallel applications. OpenMP is supported by almost all major compilers (Akhter and Roberts, 2006).

OpenMP can be used effectively for the parallelization of the originally sequential code because this requires only minimal code changes, and thus minimizing the logical mistakes of the programmer. The programmer does not create threads directly within the application code, but just inserts OpenMP directives for the compiler, which generates the code for threads during the translation. The advantage of this approach is obvious; in most cases it is sufficient just to insert directives to indicate which section of code will run in parallel. The original code does not need to be changed. The functionality of the originally sequenced code is clearly visible and the parallelization code is separable. Obviously, this approach is especially useful when it is necessary to parallelize the existing code.

In our study, we have used the following OpenMP directives valid for C/C++ programming languages:

– #pragma omp parallel { } – defines a section of the parallel calculation
– #pragma omp for schedule (dynamic) – defines a parallel loop and each iteration will be assigned to threads dynamically
– #pragma omp critical { } – creates a critical section
– #pragma omp atomic – a simple critical section, a command under this directive becomes an atomic (indivisible), therefore a command during the execution is locked for the other threads
– firstprivate { } – variables that read the value from the main thread but later become private for each thread
– shared {} – defines shared variables between all threads

The pragmas control how the program works. By the C/C++ standards, even if the compiler does not support pragmas, the program will run correctly, however, without parallelism. Therefore adding the OpenMP directives can be done very safely.

2.2. *v.surf.rst*

Spatial interpolation is one of the most computationally intensive operations in GIS. There are many interpolation methods, but usually more advanced and accurate methods require more computer resources due to more complex calculations. The Regularized Spline with Tension (RST) is an interpolation method that belongs to the wide group of global methods based on radial basis functions (Mitas and Mitasova, 1999). The global interpolation methods use all input points to generate the resulting surface. Obviously, this is not possible for very large datasets containing millions of points. For example, RST requires solving a system of N linear equations that scales the computer time to N^3 (Mitas and Mitasova, 1999). Therefore software implementations rely on the spatial autocorrelation principles of continuous phenomena, i.e. they are used globally only for a smaller segment of the area. Then the proper segmentation procedure must be taken to ensure a smooth connection of all segments of the resulting surface. Still, the application of global interpolation methods is computationally intensive, and with the advance of the laser scanning technology producing massive datasets with hundreds of millions of input points it seems inevitable to use faster computational methods.

The bivariate RST interpolation has been implemented in GRASS GIS as *v.surf.rst* (Neteler and Mitasova, 2008). In this module, the interpolation process is controlled by several internal parameters such as tension, smoothing and anisotropy. These parameters control the character of the resulting surface. They can be selected empirically, based on the knowledge of the modeled phenomenon, or automatically, by minimization of the predictive error estimated by, for example, a cross-validation procedure (Hofierka et al., 2002). Another set of the *v.surf.rst* parameters controls the data processing: minimum and maximum distances between input points, number of points used within and outside of areal segments to ensure their smooth connection. The minimum distance between input points can be very efficient in reducing the number of points used in the interpolation process. It is based on the fact that it is not necessary to use all points that fall within one cell of the resulting grid. Therefore, the value of the minimal distance parameter can be set approximately to the cell size. However, with the increasing availability of dense point clouds it is also possible to compute high-resolution digital terrain models (DTM) leading to very large grids with tens of millions of grid cells. Thus the problem with efficient computation still needs to be addressed.

The *v.surf.rst* segmentation procedure uses a decomposition of the interpolated area into rectangular segments with a variable size dependent on the density of data points using 2d-trees (Mitasova et al., 2005). The interpolation process in the segment uses input data points from within this segment and its neighborhood to ensure a smooth connection of interpolated surfaces from all segments (Mitas and Mitasova, 1999). Thus the segmentation procedure of the *v.surf.rst* module presents a task decomposition into several subtasks (interpolation segments) that can be parallelized.

2.3. *r.sun*

Spatial distribution of solar radiation at the land surface plays an important role in many environmental applications. It depends on many factors, such as the Earth's geometry, terrain morphology and atmospheric conditions. These factors are described by a set of equations creating a complex model, such as *r.sun* (Šúri and Hofierka, 2004), that is capable of estimating the amount of global solar radiation in all its components (beam, diffuse and reflected radiation) for any point on land surface. The *r.sun* model is one of the most widely used GIS-based solar radiation models that is implemented in the open-source environment of GRASS GIS (Neteler and Mitasova, 2008). Originally developed as a clear-sky model (Hofierka, 1997), it was later further substantially improved by Šúri and Hofierka (2004) to include diffuse and reflected components of solar radiation for clear-sky and real-sky conditions. It has been used in a wide range of applications at various scales (e.g., Romero et al., 2008, Ruiz-Arias et al., 2009, Bergamasco and Asinari, 2011).

The *r.sun* module in GRASS GIS computes grids of solar radiation using several input data such as DTM, atmospheric and land cover conditions (Neteler and Mitasova, 2008). The most computationally intensive part of the calculation is a sky obstruction (shadowing) by terrain. Solar rays can be blocked by terrain features especially at lower solar altitudes or in mountainous regions. The shadowing algorithm is based on solar ray tracing and comparison of solar rays' altitude to terrain elevation in selected positions along the solar ray line. This is calculated for each grid cell and repeatedly for selected times from sunrise to sunset throughout the day if a daily sum of solar radiation is calculated. The calculations for grid cells can be assigned as subtasks to threads and computed in parallel.

2.4. *r.sim.water*

Overland flow (surface runoff) is the flow of water over the land that usually occurs during heavy rainfalls, rapid snow thawing or from other sources during which the amount of water exceeds the infiltration capacity of the soil. Overland flow is controlled mainly by the terrain gradient and roughness of the land surface and drives the processes of soil erosion. The Saint-Venant differential equations are commonly used to describe the shallow water flow conditions in surface runoff (Maidment, 1993). There are several approximation techniques used for overland flow simulations. Because overland flow is a dynamic phenomenon, the simulation also needs to be dynamic. The overland flow is usually simulated by iterations replicating the flowing water (flow routing).

The *r.sim.water* module in GRASS GIS is part of the SIMWE model proposed by Mitas and Mitasova (1998). It is based on the Monte Carlo simulation and diffusion-wave approximation of the Saint-Venant differential equations (Mitasova et al., 2004; Hofierka and Knutová, 2015). The key input parameters for the *r.sim.water* module include elevation, water flow gradient (defined by the first-order partial derivatives of the elevation grid), rainfall excess rate and a surface roughness coefficient given by the Manning's n. The Monte Carlo part is represented by walkers (particles) that simulate the flowing water. The number of walkers depends on the size of grid. The duration of the simulation is defined by the number of iterations. The number of iterations and the grid size have a strong influence on the speed of calculations. The outputs include grids of water depth [m], and water discharge [m^3/s] (Neteler and Mitasova, 2008). The module also contains several other parameters controlling the movement behavior of the water flow especially in depressions or near terrain obstacles.

The module runs in iterations representing time steps in the water flow simulation. The water depth is calculated for each grid cell during the iteration, however, the lateral movement of water between the cells makes a parallelization more complicated. The concurrent addition of walkers to the same cell by different threads must be prohibited.

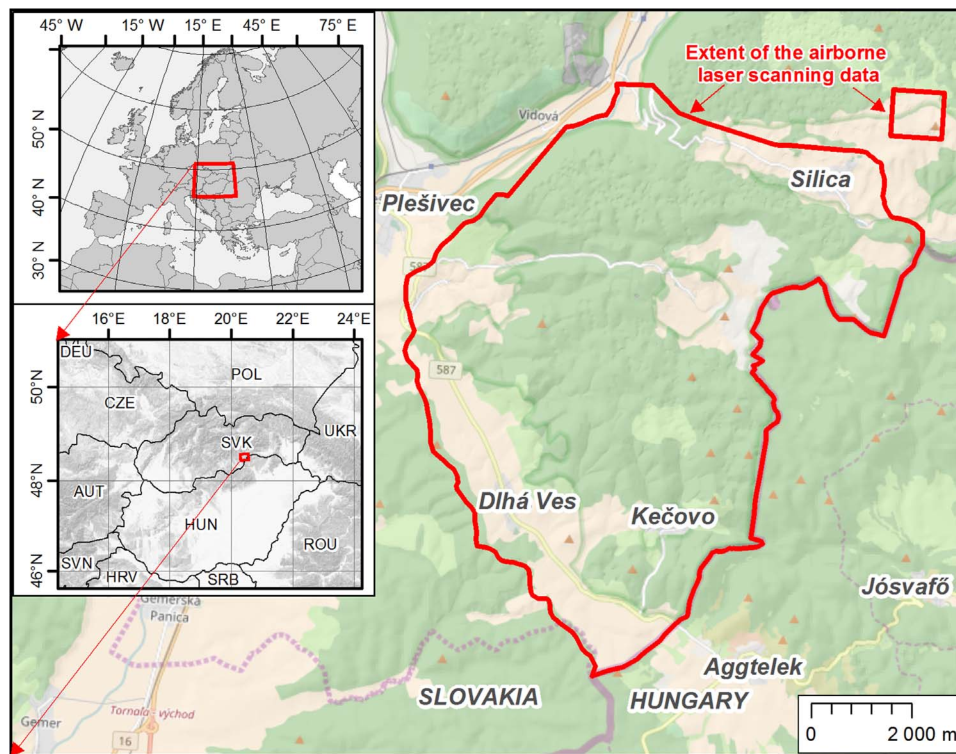


Fig. 1. Location of the test area.

Therefore the parallelization must be done in blocks of grid cells sufficiently apart from each other.

2.5. Test area and the data

We have selected a small test area near the state border of Slovakia with Hungary, Central Europe, to test the parallelized GRASS GIS modules (Fig. 1). The area covers 68 km^2 of karst landscape with many karst landform features and a diverse mosaic of forests, shrubs, permanent meadows and some arable land. The area was mapped by airborne laser scanning (ALS) in 2014. A very detailed ALS was done to map specific karst landform features such as sinkholes with varying sizes even under a dense forest canopy. The complete ALS dataset contains 1.99 billion of points with an average point density of 29 points per sq. metre representing the canopy as well as the land surface. The average density of ground returns representing the land surface is 4 points per sq. meter with a varying point density in forested and open areas. The workflow and technical details on processing the ALS dataset is described in (Hofierka et al., 2017).

The size of the dataset was further reduced to 25% of the original file in the LAStools software by random selection of points leading to the average point cloud density of 1.1 point/m^2. This operation was done outside of this study for other reasons. However, this is still more than sufficient for grids with 2-meter (6250 × 5500) and 5-meter (2500 × 2200) spatial resolutions selected for the test area. Thus, the resulting dataset used in this study contains about 75 million of points representing the land surface. The workflow and technical details on processing the ALS dataset is described in (Hofierka et al., 2017).

3. Results

The parallelization of GRASS GIS modules required several steps. First, the functionality and algorithm of the module needs to be fully understood. Then the source code must be thoroughly analyzed including the libraries called from within the program. This analysis helps to identify possible code segments suitable for parallelization. The test data must be sufficiently complex to identify software bugs and yet small enough to minimize the time for repeated calculation. The identified code segments should be analyzed by a timer control to identify the parts of the most time-consuming code. These parts of the code might be good candidates for a parallelization and therefore need a further analysis. Usually they contain a recursion or a nested loop. A parallelization should be done at the top layer (upper loop) since this is the most effective way of parallelization without creating and destruction of threads at lower layers as this consumes time. After these steps and identification of the code that could be parallelized, we can add necessary OpenMP parallelization code. The result of the calculation is compared to the original, sequential version. The results of both versions must be identical.

The tests were performed using the dataset described in the Section 2.1 and a high-resolution DTM derived by *v.surf.rst*. The modules were executed on a Linux-based desktop computer with an Intel i7-3770K hyperthreaded quad-core processor and 8 GB of RAM. In this study, we have used the current stable version of GRASS GIS (v. 7.2). The source code was compiled using GNU Compiler Collection (gcc 4.4.6) as a component of the Scientific Linux v. 6.3 operating system.

3.1. Parallelization of the v.surf.rst module

The *v.surf.rst* module uses the segmentation procedure for processing large datasets. The interpolated area is split into quadtree-based rectangular segments. The size of each segment is adjusted to the density of points. To ensure smooth connection of segments, the interpolated surface in each segment is computed using the points in the given segment and the points in its neighborhood which is defined by a rectangular window surrounding the given segment (Mitasova et al., 2005). In the source code, the division into quad-tree segments is done by the *multtree* data structure. The tree is searched recursively with the interpolation segments saved in the leaves of the tree. However, the recursion is not very appropriate for parallelization because calculation tasks should be divided equally between the threads. In recursion, this is not possible because each task depends

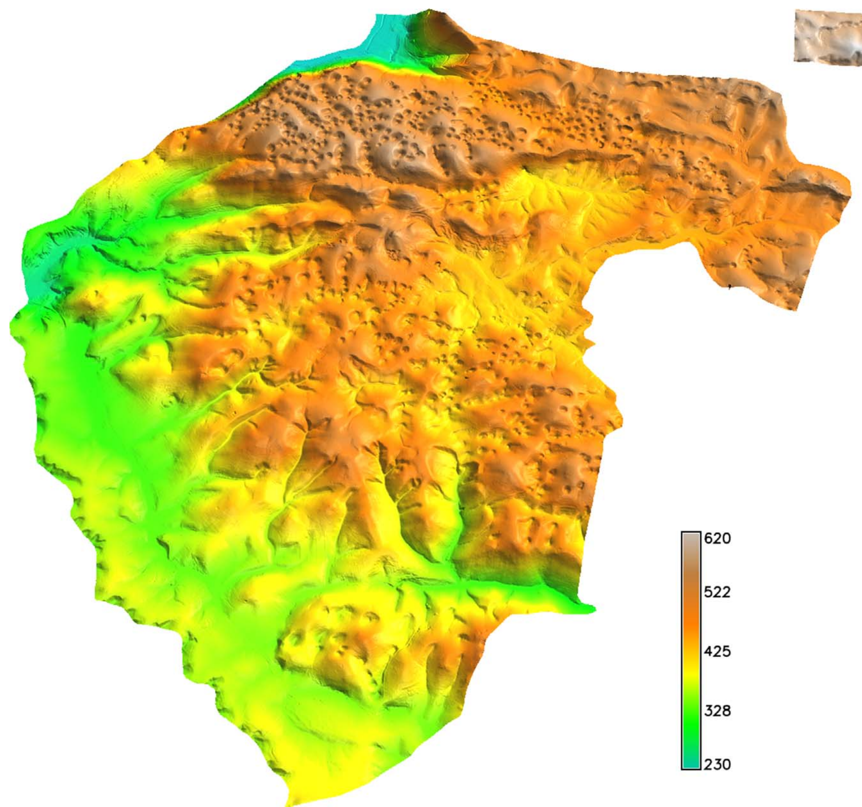


Fig. 2. The DTM (elevation in meters) computed by the parallelized *v.surf.rst* module.

on some other task. Therefore we have removed the recursive calculations and replaced them by the *cut_tree* function. This function cuts the tree, and the required data segments are stored in an array. The components of the array can be accessed in parallel thus leading to an easy access to data.

After splitting the region into segments and filling the arrays, the parallel computation is performed. The number of threads (1–4) is defined by the user using the *threads* parameter. Each thread takes the data from the array for a single interpolation segment. After the task is finished, it passes the information about it to the management of the calculation and selects the next segment. Consequently, the threads process all interpolation segments, and results are stored in a final grid. Hence, the parallelization solution is based on the segmentation procedure that is part of the interpolation algorithm.

The parallelized *v.surf.rst* module was applied to the test area and data (Fig. 2). The output elevation grids were computed with a 2-m and 5-m spatial resolution (cell size). To reduce a still oversampled input dataset, we have used the minimum distance between the input points parameter in *v.surf.rst* with the value of 2 and 5 m reflecting the spatial resolution of the grids. Thus we reduced the number of input points actually used in the calculation to 8.8 million and 2.1 million points for 2-m and 5-m spatial resolutions, respectively. The execution times of the original and parallelized *v.surf.rst* are presented in Table 1. A higher resolution requires substantially more computer resources and time. The speed-up of computation (execution time of the original version/execution time of the parallelized version) is 2.31 and 2.33 for 2 threads and 4.08 and 4.18 for 4 threads, respectively. The speed-up is slightly higher for 2-meter resolution due to a lower share of I/O operations in the computation. Normally, the speed-up will not exceed the number of processor cores. The parallel algorithm is considered highly efficient if the speed-up is equal or very close to the number of cores. The speed-up number in this case is exceeding the number of threads because the parallel version uses a more efficient algorithm dealing with the interpolation segments.

The source code of the parallelized *v.surf.rst* module is already included in the active development version of GRASS GIS (v. 7.3) and it is available at https://grass.osgeo.org/download/software/.

3.2. Parallelization of the r.sun module

The *r.sun* code has been parallelized relatively easily at the top level of the program without excessive code changes. The primary parallelization task was to split the grid cells between the processor cores. The *forscheduledynamic* function assigns the blocks of input grid cells to the threads. The cells are read sequentially from the grid. After the completion of the task, the thread asks sequentially for another cell. The variables that should not be shared between the threads because they would be overwritten within different threads were assigned as

Table 1
Execution times (in seconds) and speed-up for the original and parallelized version of *v.surf.rst*.

Spatial resolution	Original version	1 Thread		2 Threads		4 Threads	
	Time (s)	Time (s)	Speed-up	Time (s)	Speed-up	Time (s)	Speed-up
2 m	13,598	11,323	1.20	5,832	2.33	3,255	4.18
5 m	2,622	2,184	1.20	1,137	2.31	642	4.08

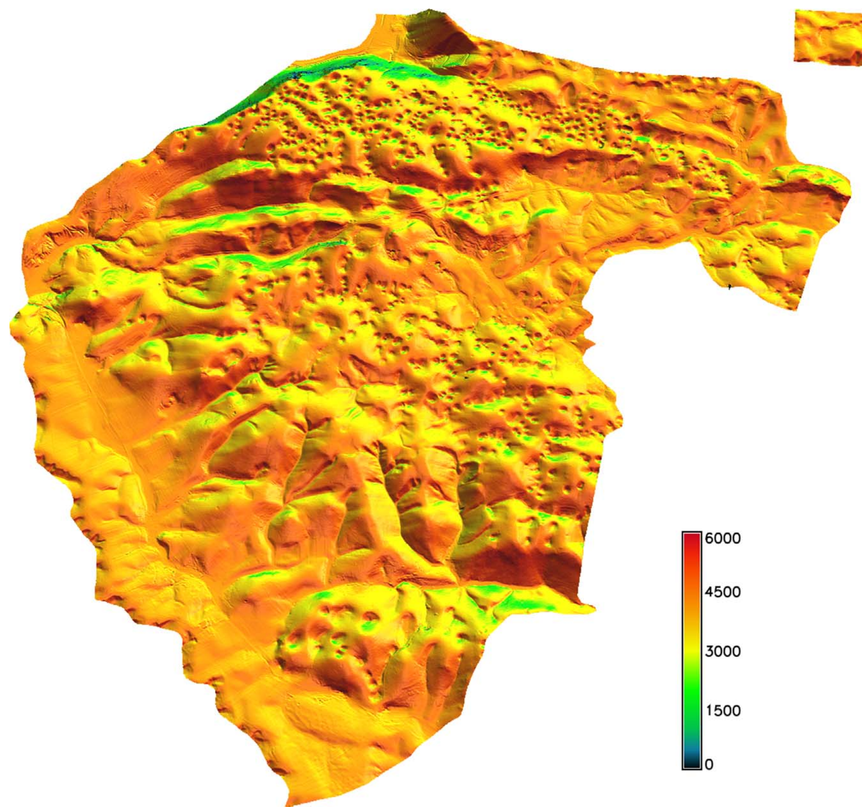


Fig. 3. Beam solar radiation (daily sum in W h/m^2) computed using the parallelized *r.sun* module.

private. Also we modified the calculation of variables representing the indexes of the grid cells in order to make the calculation by threads independent from the iteration.

To demonstrate the applicability of the parallelized *r.sun* module, we used the DTM computed by the *v.surf.rst* module. The area used for the analysis contains approximately 2.7 million of cells in the 5-meter resolution grid and 17 million of cells in the 2-meter resolution grid. We computed the beam radiation for the spring equinox day (Fig. 3). The execution times of the original and parallelized *r.sun* are presented in Table 2. The speed-up of computation is 1.95 and 1.97 for 2 threads and 3.44 and 3.72 for 4 threads, respectively. This indicates that the software is efficiently parallelized despite the fact that some sequential parts of the code are identical (such as input/output operations). The speed-up is clearly higher for the 2-meter resolution.

The parallelized source code of *r.sun* is available from the Add-ons repository of GRASS GIS at https://trac.osgeo.org/grass/browser/grass-addons/grass7/raster/r.sun.mp.

3.3. Parallelization of the r.sim.water module

The analysis of the *r.sim.water* module showed that this module depends on the Monte Carlo simulation library (r.sim). Therefore the parallelization had to be done in this library and *r.sim.water* contains just a parameter for selecting the number of threads. This module is impossible to parallelize at the top level of the program because the final grid is created iteratively and each step depends on the previous step. Therefore the parallelization has to be done at lower level of the program. This also causes a slightly lower speed-up of the parallelization in comparison to previous modules (Table 3). The parallelization is applied to large segments of grid cells, and threads work on sufficiently distant grid cells. The grid is split into regular segments via spatial decomposition according to the number of threads. Each thread works on its own segment.

The parallelized *r.sim.water* module was applied to the test area using the same high-resolution DTM derived by *v.surf.rst*. We simulated 120-min. rainfall event using default simulation parameters and uniform soil and land cover properties. The output grids include water flow depth and water discharge (Fig. 4). The computation times of the original and parallelized *r.sim.water* are presented in Table 3. The speed-up of computation is 1.83 and 1.90 for 2 threads and 2.87 and 2.97 for 4 threads, respectively. The speed-up of the computation is slightly lower than in previous modules due to a less efficient parallelization as discussed above. Still, the speed-up is higher for 2-meter resolution showing a better performance for large grids.

The parallelized source code of *r.sim.water* is available from the

Table 2
Execution times (in seconds) and speed-up for the original and parallelized version of *r.sun*.

Spatial resolution	Original version/1 Thread		2 Threads		4 Threads	
	Time (s)	Speed-up	Time (s)	Speed-up	Time (s)	Speed-up
2 m	2,652	–	1,345	1.97	713	3.72
5 m	189	–	97	1.95	55	3.44

Table 3
Execution times (in seconds) and speed-up for the original and parallelized version of r.sim.water.

Spatial resolution	Original version/1 Thread		2 Threads		4 Threads	
	Time (s)	Speed-up	Time (s)	Speed-up	Time (s)	Speed-up
2 m	21,824	–	11,501	1.90	7,343	2.97
5 m	2,631	–	1,440	1.83	916	2.87

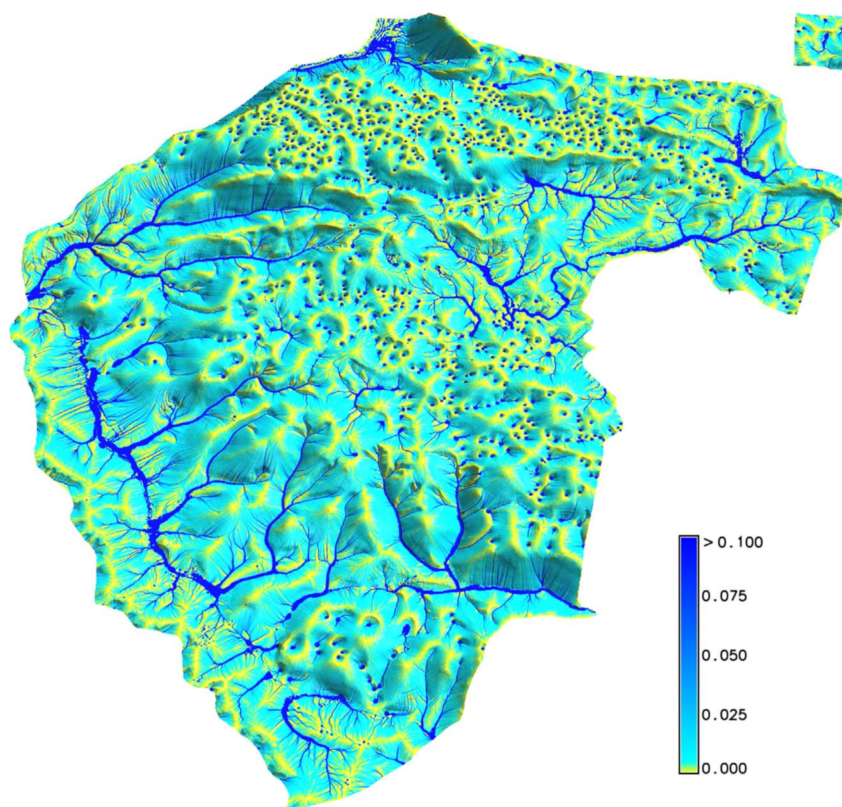


Fig. 4. Water flow discharge (in m^3/s) computed using the parallelized *r.sim.water* module.

Add-ons repository of GRASS GIS at https://trac.osgeo.org/grass/browser/grass-addons/grass7/raster/r.sim.water.mp.

4. Discussion

If we compare results of all parallelized modules and their original versions presented in Tables 1–3, we can see a substantial improvement in execution times depending on the number of threads. The most efficiently parallelized is the *v.surf.rst* module because the modification of the original code also included changes in the manipulation of interpolation segments, which proved to be more efficient than in the original version. The results of calculations were verified with the results of the original versions and they are identical. Very small variations in the resulting output grids exist in the case of the *r.sim.water* module because of the independent use of the *g_drand48* random generator within each thread. However, the differences are very small and within the expected accuracy of the Monte Carlo model.

Spatial resolution has a major impact on execution times in all modules. The least efficient is the *r.sun* module; the original version calculates an output for the 2-m resolution grid with (6250 × 5500) 14 times longer than for the 5-m resolution grid (2500 × 2200) but still within one hour. However, the calculation of a high-resolution grid by the original versions of *r.sim.water* and *v.surf.rst* lead to execution times that are rather too long (4–6 h). Both modules require thorough parameterizations, so the user often needs to run the modules repeatedly to analyze results. Still, the 6250 × 5500 grid is nothing unusual in geoscientific applications. The OpenMP parallelization has led to a substantial reduction of execution times, almost linearly with the number of processor cores. The speed-up factors for 4 threads are 4.18 (*v.surf.rst*), 3.72 (*r.sun*) and 2.97 (*r.sim.water*) using a 2-meter resolution grid. Schiele et al. (2012) report a speed-up in a range of 2.58–3.23 in a study with similar data using the POSIX Threads parallelization library. The authors calculated flow directions using the grid-based DEM. Guan and Wu (2010) achieved a speed-up factor of 3.66 for spatial interpolation of DEM using Intel Threading Building Blocks parallelization library and similar laser scanning data. In both studies, the new custom-based algorithms were developed. In contrast, this study demonstrates the efficiency of the OpenMP parallelization approach applied to existing open-source algorithms.

Several parallelization studies (for example, Guan and Wu, 2010; Xie, 2012) discuss the scalability of the parallelization relating the number of threads to the speed-up. Depending on the amount of data, the optimal number of threads is around 4–5. When the number of concurrent threads is greater than 6, the speed-up quickly decreases due to serial input/output operations (Guan and Wu, 2010). The use of GPGPU with thousands of threads may be a good solution to algorithms with a high level of parallelism and local character of the calculations applied to large datasets (Sten et al., 2016). The *r.sun* module could be a good candidate for such a parallelization approach, especially for cell by cell calculations of local solar parameters.

5. Conclusions

Current geospatial software products include several computationally intensive operations that could be efficiently parallelized using appropriate tools such as OpenMP or MPI depending on the hardware architecture. In this study, we present an OpenMP-based parallelization approach suitable for multi-core systems with a shared memory model applied to existing open-source GRASS GIS modules.

We have parallelized three computationally intensive GRASS GIS modules: *v.surf.rst*, *r.sun* and *r.sim.water*. These modules often use as an input massive and high-resolution geospatial data presenting challenges for computational capacity. Moreover, they are often run repeatedly in order to get optimized input parameters or the most accurate result.

The parallelization was done in several steps. These include the analysis of the module's functionality, identification of source code segments suitable for the parallelization and proper application of

parallelization code to create efficient threads processing the subtasks. The parallelized modules were applied to the test area represented by massive airborne laser scanning data and derived high-resolution DTMs. The parallelized modules showed a substantial speed-up of computation depending on the number of threads. This documents that the parallelization was implemented efficiently, thus maximizing the processor performance.

In this study, we demonstrated the simplicity and efficiency of the parallelization of open-source GRASS GIS modules using OpenMP, leading to an increased performance of this geospatial software on standard multi-core computers. The parallelization can substantially improve the speed of complex geospatial operations such as spatial interpolation or landscape process simulations. This paper showed that parallelization of existing GIS operations can be done easily and without excessive code changes. The parallelized GIS modules should very well benefit the GIS users and inspire for further efforts in this field.

Acknowledgement

This work originated within the APVV-0176-12 and VEGA 1/0474/16 research projects supported by the Slovak Research and Development Agency and Slovak Research Grant Agency VEGA, respectively.

References

Akhter, S., Roberts, J., 2006. Multi-Core Programming. IntelPress, Hillsboro, USA, 360.

Bergamasco, L., Asinari, P., 2011. Scalable methodology for the photovoltaic solar energy potential assessment based on available roof surface area: application to Piedmont Region (Italy). Sol. Energy 85, 1041–1055.

Chapman, B., Jost, G., van der Pas, R., 2007. Using OpenMP: Portable Shared Memory Parallel Programming. The MIT Press, Cambridge, 384.

Cui, Z., Vieux, B.E., Neeman, H., Moreda, F., 2005. Parallelisation of a distributed hydrologic model. Int. J. Comput. Appl. Technol. 22, 42–52.

Fang, J., Varbanescu, A.L., Sips, H., 2011. A comprehensive performance comparison of CUDA and OpenCL. In: Proceedings of the 40th International Conference on Parallel Processing, Taipei City, Taiwan, 13-16 September 2011, pp. 216–225.

Flynn, M.J., 1972. Some computer organizations and their effectiveness. IEEE Trans. Comput. C–21 (9), 948–960.

GRASS, 2016. GRASS GIS [online]. ⟨http://grass.osgeo.org/⟩ December 2016.

Guan, X., Wu, H., 2010. Leveraging the power of multi-core platforms for large-scale geospatial data processing: exemplified by generating DEM from massive LiDAR point clouds. Comput. Geosci. 36, 1276–1282.

Healey, R., Dowers, S., Gittings, B., Mineter, M., 1998. Parallel Processing Algorithms for GIS. Taylor & Francis, London, UK, 460.

Hofierka, J., Gallay, M., Kaňuk, J., Šašak, J., 2017. Modelling karst landscape with massive airborne and terrestrial laser scanning data. In: Ivan, I., Singleton, A., Horák, J., Inspektor, T. (Eds.), The Rise of Big Spatial Data, Lecture Notes in Geoinformation and Cartography. Springer International Publishing, Cham, Switzerland, 141–154.

Hofierka, J., 1997. Direct solar radiation modelling within an open GIS environment. In: Proceedings of the 1997 Joint European GI Conference, Vienna, Austria, pp. 575–584.

Hofierka, J., Knutová, M., 2015. Simulating spatial aspects of a flash flood using the Monte Carlo method and GRASS GIS: a case study of the Malá Svinka Basin (Slovakia). Open Geosci. 7, 118–125.

Hofierka, J., Parajka, J., Mitasova, H., Mitas, L., 2002. Multivariate interpolation of precipitation using regularized spline with tension. Trans. GIS 6, 135–150.

Huang, F., Liu, D., Tan, X., Wang, J., Chen, Y., He, B., 2011. Explorations of the implementation of a parallel IDW interpolation algorithm in a Linux cluster-based parallel GIS. Comput. Geosci. 37, 426–434.

Huang, Q., Yang, C., 2011. Optimizing grid computing configuration and scheduling for geospatial analysis: an example with interpolating DEM. Comput. Geosci. 37, 165–176.

Li, T., Wang, G., Chen, J., Wang, H., 2011. Dynamic parallelization of hydrological model simulations. Environ. Model. Softw. 26, 1736–1746.

Maidment, D.R., 1993. Handbook of Hydrology. McGraw-Hill, New York, 1424.

Mineter, M.J., Dowers, S., 1999. Parallel processing for geographical applications: a layered approach. J. Geogr. Syst. 1, 61–74.

Mitas, L., Mitasova, H., 1998. Distributed soil erosion simulation for effective erosion/deposition modeling and enhanced dynamic visualization. Water Resour. Res. 34, 505–516.

Mitas, L., Mitasova, H., 1999. Spatial interpolation. In: Longley, P., Goodchild, M.F., Maguire, D.J., Rhind, D.W. (Eds.), Geographical Information Systems: Principles, Techniques, Management and Applications. GeoInformation International, Wiley, New York, 481–492.

Mitasova, H., Mitas, L., Harmon, R.S., 2005. Simultaneous spline interpolation and topographic analysis for lidar elevation data: methods for Open source GIS. IEEE Geosci. Remote Sens. Lett. 2, 375–379.

Mitasova, H., Thaxton, C., Hofierka, J., McLaughlin, R., Moore, A., Mitas L., 2004. Path sampling method for modeling overland water flow, sediment transport and short term terrain evolution in Open Source GIS. In: C.T. Miller, M.W. Farthing, V.G. Gray, G.F. Pinder (Eds.) Proceedings of the XVth International Conference on Computational Methods in Water Resources (CMWR XV), June 13–17 2004, Chapel Hill, NC, USA, Elsevier, pp. 1479–1490.

Neteler, M., Bowman, M.H., Landa, M., Metz, M., 2012. GRASS GIS: a multi-purpose open source GIS. Environ. Model. Softw. 31, 124–130.

Neteler, M., Mitasova, H., 2008. Open source GIS: A GRASS GIS approach. third ed. In: The International Series in Engineering and Computer Science, volume 773, Springer, New York.

Pintor, B.H., Sola, E.F., Teves, J., Inocencio, L.C., Ang, M.R.C., 2015. The resolution dilemma: Finding the optimum DEM resolution for large-scale solar energy resource assessment using R.sun. In: Lagmay, A.M. (Ed.) Proceedings of the 36th Asian Conference on Remote Sensing 2015 (ACRS 2015): Fostering Resilient Growth in Asia, 19–23 October 2015, Quezon City, Manila, Philippines, Asian Association on Remote Sensing, pp. 1357–1366.

Romero, L.F., Tabik, S., Vías, J.M., Zapata, E.L., 2008. Fast clear-sky solar irradiation computation for very large digital elevation models. Comput. Phys. Commun. 178, 800–808.

Ruiz-Arias, J.A., Tovar-Pescador, J., Pozo-Vázquez, D., Alsamamra, H., 2009. A comparative analysis of DEM-based models to estimate the solar radiation in mountainous terrain. Int. J. Geogr. Inf. Sci. 23, 1049–1076.

Schiele, S., Möller, M., Blaar, H., Thürkow, D., Müller-Hannemann, M., 2012. Parallelization strategies to deal with non-localities in the calculation of regional land-surface parameters. Comput. Geosci. 44, 1–9.

Sten, J., Lilja, H., Hyväluoma, J., Westerholm, J., Aspnäs, M., 2016. Parallel flow accumulation algorithms for graphical processing units with application to RUSLE model. Comput. Geosci. 89, 83–95.

Stallings, W., 2010. Computer Organization and Architecture: Designing for Performance eighth ed.. Prentice Hall, Upper Saddle River, New Jersey.

Sorokine, A., 2007. Implementation of a parallel high-performance visualization technique in GRASS GIS. Comput. Geosci. 33, 685–695.

Šúri, M., Hofierka, J., 2004. A new GIS-based solar radiation model and its application to photovoltaic assessments. Trans. GIS 8, 175–190.

Šúri, M., Huld, T.A., Dunlop, E.D., 2005. PVGIS: a web-based solar radiation database for the calculation of PV potential in Europe. Int. J. Sustain. Energy 24, 55–67.

Xie, J., 2012. Implementation and performance optimization of a parallel contour line generation algorithm. Comput. Geosci. 39, 21–28.

Zambelli, P., Gebbert, S., Ciolli, M., 2013. Pygrass: an object oriented python application programming interface (API) for geographic resources analysis support system (GRASS) geographic information system (GIS). ISPRS Int. J. Geo-Inf. 2, 201–219.