Computers & Geosciences 44 (2012) 1–9

Contents lists available at SciVerse ScienceDirect

Computers & Geosciences

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

Parallelization strategies to deal with non-localities in the calculation of regional land-surface parameters

Steffen Schiele[a,*], Markus Möller[b], Holger Blaar[a], Detlef Thürkow[b], Matthias Müller-Hannemann[a]

[a] University of Halle-Wittenberg, Institute of Computer Science, Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany
[b] University of Halle-Wittenberg, Institute of Geosciences, Von-Seckendorff-Platz 4, 06120 Halle (Saale), Germany

A R T I C L E I N F O

Article history:
Received 16 August 2011
Received in revised form
21 February 2012
Accepted 24 February 2012
Available online 8 March 2012

Keywords:
Parallel processing
Flow direction
Terrain analysis
High resolution elevation models
Threads
Multi-core

A B S T R A C T

Hand in hand with the increasing availability of high resolution digital elevation models (DEMs), an efficient computation of land-surface parameters (LSPs) for large-scale digital elevation models becomes more and more important, in particular for web-based applications. Parallel processing using multi-threads on multi-core processors is a standard approach to decrease computing time for the calculation of local LSPs based on moving window operations (e.g. slope, curvature). LSPs which require non-localities for their calculation (e.g. hydrological connectivities of grid cells) make parallelization quite challenging due to data dependencies. On the example of the calculation of the LSP "flow accumulation", we test the two parallelization strategies "spatial decomposition" and "two phase approach" for their suitability to manage non-localities.

Three datasets of digital elevation models with high spatial resolutions are used in our evaluation. These models are representative types of landscape of Central Europe with highly diverse geomorphic characteristics: a high mountains area, a low mountain range, and a floodplain area in the lowlands. Both parallelization strategies are evaluated with regard to their usability on these diversely structured areas. Besides the correctness analysis of calculated relief parameters (i.e. catchment areas), priority is given to the analysis of speed-ups achieved through the deployed strategies. As presumed, local surface parameters allow an almost ideal speed-up. The situation is different for the calculation of non-local parameters which requires specific strategies depending on the type of landscape. Nevertheless, still a significant decrease of computation time has been achieved. While the speed-ups of the computation of the high mountain dataset are higher by running the "spatial decomposition approach" (3.2 by using four processors and 4.2 by using eight processors), the speed-ups of the "two phase approach" have proved to be more efficient for the calculation of the low mountain and the floodplain dataset (2.6 by using four processors and 2.9 by using eight processors).

1. Introduction

A large variety of methods for space-oriented analyses of the earth's surface have been developed in the past couple of years. Methods based on photogrammetry and laser-scanning are able to produce digital elevation models (DEMs) with a geometric resolution within centimeters and a high quality, as well as a high quantity of data. The handling of high resolution DEMs is one of the most essential challenges in geomorphometrical and hydrological modeling. Algorithms have to be developed enabling computational efficient processing of large datasets in reasonable time (Wood, 2009). Processing time is of particular importance for the acceptance of web-based hydrological applications (Al-Sabhan et al., 2003; Gläßer et al., 2010). Therefore, new strategies for efficient implementation and parallelization of such computations are needed. These strategies have to find their ways into service-oriented architectures (SOA) for decision support systems (DSS). There, especially the use of computational intensive interactive visualization techniques would improve communication among stakeholders and scientists as well as the stakeholder interaction (Volk et al., 2010). Furthermore, the usage of high resolution input data would also enable small scale modeling leading to more accurate environmental risk assessments (Meyer et al., 2009).

The specific motivation for this study goes back to the development of an extension within the e-learning online portal GEOVLEX which aims at the geovisualization of hydrological processes (Gläßer et al., 2010). Against this background, a parallelized version of the D8 algorithm was implemented. However, its application leads to unexpected results. Especially, the derived watershed differed significantly from a reference watershed area

* Corresponding author.
E-mail address: steffen.schiele@informatik.uni-halle.de (S. Schiele).

0098-3004/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cageo.2012.02.023

which represents an officially confirmed boundary (Fig. 1). The reason for this inaccurate watershed delineation is related to the fact that the D8 algorithm assigns ambiguous flow directions arbitrarily to one direction within a moving window environment (Gruber and Peckham, 1999). We call this problem "neighborhood effect" (Fig. 2) which occurs in DEM sections with the same altitude (e.g. flat areas, lakes or other water surfaces)

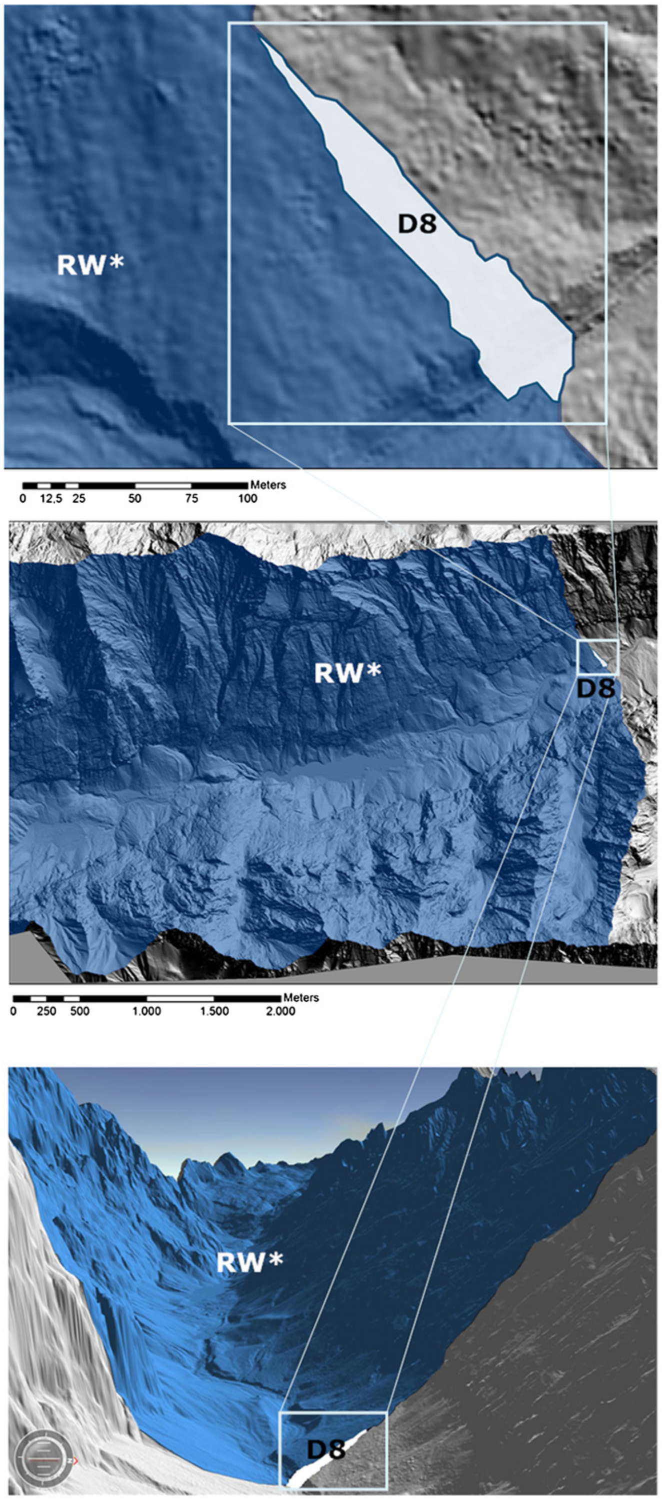

Fig. 1. Comparison of a reference (RW) and D8 watershed. The white marked watershed is calculated with the classical D8 algorithm. The image on the top is a magnified part of both lower images.

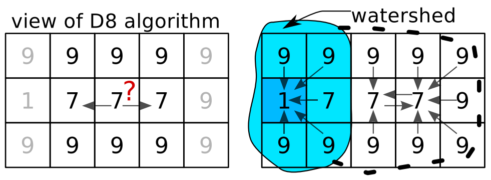

Fig. 2. Visualization of the neighborhood effect. The left part of the figure shows the 3 × 3 neighborhood of the cell in the center of the grid. Single flow algorithms, which use only the 3 × 3 neighborhood, are not able to determine the correct flow direction. This affects the determination, for instance, of the watershed, which is shown in the right part of the figure.

or after removing sinks by raising an area (Nardi et al., 2008). As a consequence, we have developed a modified version of the D8 algorithm. It is based on the concept of a neighborhood extension which allows to determine flow directions in a systematic and traceable manner (Section 3.1).

Extended neighborhoods are an example for non-localities. Non-localities represent raster cells which are located outside from the moving window neighborhood but have to be considered for the calculation of land-surface parameters (LSPs). The parallelization of such LSPs is challenging because of data dependencies which lead to random memory access. Thus, the main objective of this study is the investigation of parallelization strategies with regard to their efficiency to deal with non-localities.

In Section 2 we give a short review of parallel processing in geoscience and discuss some general problems. In Section 3, we first sketch our extended D8 algorithm for flow computations and then explain our two parallelization strategies. We also describe the sites used in our experimental study. Computational results for both parallelization strategies are given in Section 4. Finally, we summarize and discuss our observations in Section 5.

2. Parallel geoprocessing

It is no longer possible to speed-up applications by using faster processors with higher and higher clock frequencies. Besides the improvement of sequential algorithms' efficiency, the only way to get higher performance is to parallelize software to run on multi-cores. With the omnipresent multi-core processors with growing numbers of cores in the next years an essential alteration in software development arises.

Two main types of parallel computing systems can be distinguished (Rauber and Rünger, 2010):

1. Distributed memory systems are clusters of processors or shared memory nodes connected by an alignment network where data has to be transferred explicitly between the processors or nodes by message-passing routines.
2. Shared memory machines consist of one or more multi-core processors each with direct memory access.

As pointed out in Neal et al. (2010) the trend towards larger multi-core machines is expected to continue in the future. In our work we focus on parallelization methods for shared memory machines, in particular for multi-core systems. Instead of using multi-core processors, another interesting approach for parallelization is the usage of graphics processing units (GPUs). Ortega and Rueda (2010) have studied the applicability of this approach for parallel computation of drainage networks. The GPU is well suited for local processing and has only been used for this purpose.

There exist different basic approaches of parallelization like data-parallel computation, parallel calculation of independent components or parallelization of concurrent parts of an algorithm

(Grama et al., 2003; Rauber and Rünger, 2010). The most suitable method to implement parallel algorithms for shared memory computers is working with threads (Butenhof, 2007). Threads are subprocesses, started by a process. All threads use the context of this process. That means, all threads of a program have a common address space. The programmer has to synchronize the parallel work of all threads. One challenge regarding the efficiency is the scheduling of the tasks onto the processors or cores in such a manner that they work without long waiting time. Administration of threads causes always an overhead, for example synchronization cost, communication cost or I/O-operations (Nichols et al., 2003). The elimination of such a bottleneck might also cause another bottleneck at a deeper system level, which is more difficult to predict (Butenhof, 2007). A further problem is the possibility of false sharing. False sharing occurs when processors, that are accessing logically distinct data nevertheless conflict because the locations they are accessing lie on the same cache line (Herlihy and Shavit, 2008).

Thread-parallel applications can be implemented in major programming languages like C/C++, Fortran, Java, and C# by using thread libraries. The most popular one is the Pthreads library (Butenhof, 2007). POSIX threads (or Pthreads) is a POSIX standard to implement thread-parallel applications in C. In the Pthread library various functions, data types, programming interfaces, and macros are implemented to manage and coordinate threads and concurrent data access. The programmer must personally ensure that the parallel program works correctly and efficiently. Another one is the OpenMP library (Quinn, 2003). For example, Neal et al. (2009) describe and report experience with parallelization of procedures for flood inundation models using the OpenMP interface. OpenMP is particularly suitable for computations, which iteratively process the DEM, for example for the parallelization of the classic D8 algorithm. Compared with Pthreads, OpenMP is difficult to handle as soon as complex synchronizations are needed in the case for the parallelization of the modified D8 algorithm. For instance, in OpenMP an explicit control over the memory hierarchy is not possible. Another disadvantage is that statements in parallelized `for`-loops, which can lead to an early loop termination, are not allowed. Less often used, parallel implementations for multi-cores can also be built by using the Message Passing Interface (MPI; Gropp et al., 1999). For example, Tesfa et al. (2011) developed parallel approaches for the extraction of hydrological proximity measures using MPI. Pthreads as well as implementations using MPI or OpenMP lead to portable programs. However, MPI is most useful in distributed memory environments. Because of its flexibility, the Pthreads library was used in this study.

In contrast to our focus on algorithms which can be handled within internal memory, Mølhave et al. (2010) developed I/O-efficient external memory algorithms to cope with large-scale high resolution datasets.

3. Methods

3.1. Modification of the D8 algorithm

The attribute "flow direction" is the basis for the calculation of the most popular hydrological parameters like "specific catchment area" or "topographic wetness index" Wilson et al. (2007) and Gruber and Peckham (1999). Two general types of flow algorithms can be distinguished which differ in the handling of divergent flow. Single-neighbor algorithms do not enable divergent flow while multiple-neighbor algorithms can represent it (Gruber and Peckham, 1999). Each approach has specific strengths and weaknesses as well as specific fields of application which are discussed in detail by e.g. Wilson and Gallant (2000), Wilson et al. (2007), Kenny and Matthews (2005), Schäuble et al. (2008), Zhu and Lin (2009) and Gruber and Peckham (1999).

The so-called D8 algorithm was introduced by O'Callaghan and Mark (1984) and uses only eight adjacent cells for its computation. Two grid cells are called *adjacent* if they are neighbored. We use the standard neighborhood relation which gives rise to the name D8: a grid cell is neighbored with some other cell if and only if they share an edge or a common vertex. That means that a non-border cell is adjacent to eight other cells. The flow of a cell is rooted to the adjacent cell with the steepest descent. However, a problem occurs when the same minimum downslope gradient is found in more than one cell. Such ambiguous flow directions are mostly arbitrarily assigned to one direction (Gruber and Peckham, 1999). Particularly, this problem is related to surfaces with the same altitude (e.g. flat areas, lakes or other water surfaces) which could have considerable effects on the calculation of hydrological relief parameters (Nardi et al., 2008). It also occurs after removing sinks by raising an area. The example in Fig. 1 shows a white marked watershed area which was calculated using the classic D8 algorithm after removing sinks. It differs significantly from a blue marked reference watershed area (RW) which represents an officially confirmed boundary.

In order to assign systematically the ambiguous flow directions we propose a modified D8 algorithm. We call our algorithm D8*e* where *e* stands for neighborhood extension. In our implementation the 3×3 neighborhood of a cell is extended, if its flow direction and consequently its drainage cell is ambiguous. In the Appendix a pseudocode (Algorithm 1) is shown, which extends the neighborhood first and computes the flow directions using the extended neighborhood. Our algorithm starts at each potential drainage cell and assigns cells to the extended neighborhood, which meet the following three conditions:

- The cells have the same altitude as the potential drainage cell.
- The cells belong to one area which is spatially connected.
- At least one cell of the area is adjacent to the potential drainage cell.

The extension of the neighborhood is shown in Algorithm 2. Within the extended neighborhood the steepest descent of all potential drainage cells and their flow directions are calculated. For more details see Algorithm 3. Then the flow of the considered cell is routed to the potential drainage cell of the extended neighborhood which belongs to the drainage line with the steepest descent (Algorithm 1). The regional calculation step in Algorithm 3 is the most time-consuming and extensive calculation step and makes the parallelization particularly challenging. The size of the extended neighborhood varies widely depending on the terrain.

In the worst case, the running time of Algorithm 3 can be a constant times the squared number of grid cells, i.e. of order $\mathcal{O}(n^2)$ where n denotes the number of grid cells. However, we would like to point out that the actual running time can be much better. This crucially depends on the flatness of the given DEM.

3.2. Parallelization strategies

Two main strategies of parallelization and the effect on extended neighborhood were investigated. The first approach divides the DEM into squares and the second one divides the computation of the flow directions into two phases. All threads access the same DEM stored in shared memory. Therefore, there is no need to transfer data but the data access has to be synchronized among the threads.

3.2.1. *Dividing the DEM into squares*

If the DEM is divided into different squares and the flow directions of each square are computed by different threads at the same time complications often arise. Namely, the extended neighborhood computation can include cells of other squares. This means that the same flow direction of a single cell might be calculated several times and the number of such cells could be prohibitively high. An example is the area of the reservoir dam "Saidenbachtal" (Section 3.4). If this reservoir area is interpreted as a sink and is filled until the elevation of their lowest outflow point is reached, then all these cells will have the same altitude. Thus, they will be part of an extended neighborhood and redundantly considered several times.

To avoid such problems a sequential pre-computation is executed. At first, the algorithm chooses squares which are slightly overlapping at common boundary cells. Then the flow direction of all boundary cells and their neighbors are computed. In doing so, no thread will cross its square boundary. Fig. 3 exemplarily illustrates two cases where the algorithm extends the neighborhood of the cells with altitude 7 and precalculates the flow directions of all cells of the extended neighborhood (cells with altitude 5). Finally, the results are accessible by each thread. In the following, each thread computes the flow direction of the remaining cells (see Fig. 4). The pseudocode is given in the Appendix (Algorithm 4).

3.2.2. *Dividing the computation into two phases*

The main idea of this approach is to compute the flow direction using the original D8 algorithm during a first phase. Instead of extending the neighborhood of cells with ambiguous flow directions

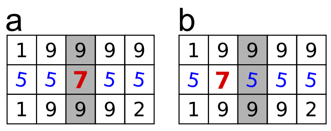

Fig. 3. Excerpts of a DEM where two squares overlap in the gray-shaded boundary cells. Numbers in each cell of the DEMs correspond to the elevation. In both examples, the D8*e* algorithm extends the neighborhood of the cells with altitude 7 and precalculates the flow directions of all cells with altitude 5.

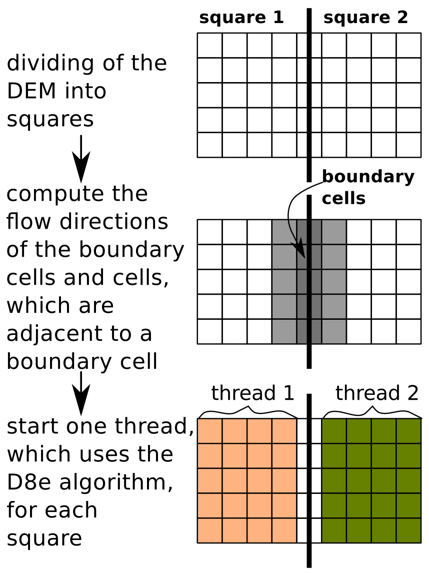
square 1
square 2
dividing of the DEM into squares
compute the flow directions of the boundary cells and cells, which are adjacent to a boundary cell
boundary cells
start one thread, which uses the D8e algorithm, for each square
thread 1
thread 2

Fig. 4. Illustration of the first parallelization strategy. First the DEM is divided into squares which overlap in their boundary cells. Afterwards, the flow directions of the boundary cells and of all cells which are adjacent to the boundary cells are computed using the D8*e* algorithm. Finally, each thread computes the flow directions of all cells of its square using the D8*e* algorithm.

(see Section 3.1) such cells are only marked. During the second phase, our algorithm extends the neighborhood of each marked cell and computes their flow directions.

The first phase can be parallelized by dividing the DEM into squares. Only using the 3×3-neighborhood of a cell for computing the flow direction, no pre-computation is needed. In the second phase, the marked cells are assigned to different groups in such a way that redundant calculations are avoided. At first, a marked cell is assigned to an empty group. Afterwards, a cell will be assigned to this group, if

- the marked cell is adjacent to one cell of the group, or
- there is at most one unmarked cell between the marked cell and a cell of the group.

The group assignment is done with breadth-first search (BFS; Cormen et al., 2009). BFS is a graph search algorithm. It begins at some root cell and explores all neighboring cells. For each of those cells, it explores their unexplored neighboring cells. This will be continued until all cells have been explored.

Fig. 5(a) shows marked cells for which the computation requires a neighborhood extension. In each of the three examples the algorithm puts the marked cells into the same group. The differently marked cells in the example of Fig. 5(b) belong to two different groups. The extended neighborhoods of both cells are disjoint. One thread is responsible for assigning cells to groups and manages these groups by a task pool (Rauber and Rünger, 2010). Each of the other threads takes a group out of the task pool and computes the flow direction of the marked cells of this group. In doing so, this approach ensures that two threads will never compute the flow direction of the same cell and that the computed extended neighborhoods will never overlap. An illustration is shown in Fig. 6. The corresponding pseudocodes appear in the Appendix (Algorithms 5–7).

3.3. *Efficiency measurement*

For measuring the runtime of a parallel implementation, the *real time* (wall clock time) can be used, but it can be influenced by other applications running on the system. *User* and *system CPU time* of a parallel application is the *accumulated* user and system CPU time on all processors. For instance, to get the actual parallel CPU time, these values have to be divided by the number of processors or cores used. Because of the disruptive effect of other processes running on the system the number of cores used by one job cannot easily be determined and can also vary during program execution. The operating system distributes the threads of all running programs onto the processors and cores (Blaar et al.,

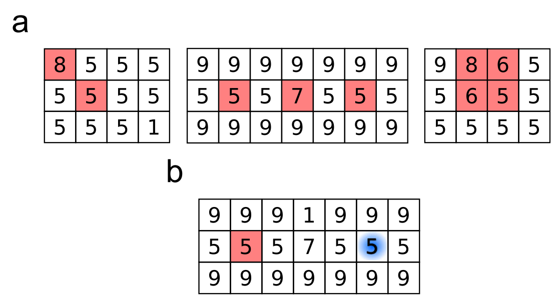

Fig. 5. Parallelization of the D8*e* algorithm by dividing the computation into two phases. Numbers in each cell of the DEMs correspond to the elevation. (a) In each example, cells with ambiguous flow direction are marked. Because of the neighborhood extension, the marked cells will be computed by the same thread. Therefore, our algorithm puts them into the same group. (b) In this example, differently marked cells are far enough from each other so that the extended neighborhoods are disjoint. Thus, our algorithm puts them into different groups.

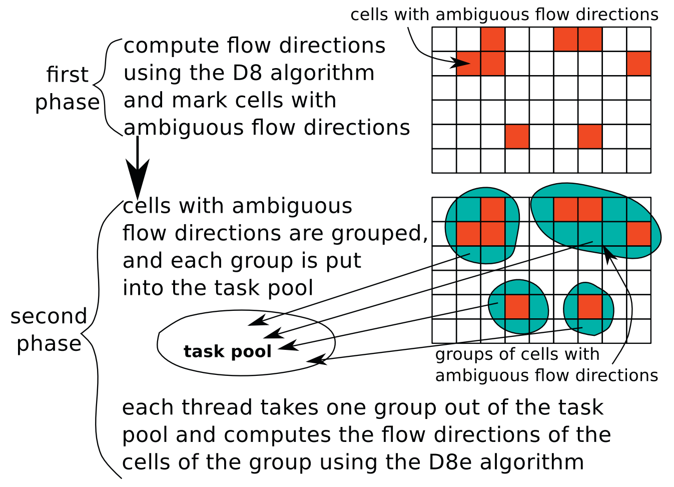
cells with ambiguous flow directions
first phase
compute flow directions using the D8 algorithm and mark cells with ambiguous flow directions
second phase
cells with ambiguous flow directions are grouped, and each group is put into the task pool
task pool
groups of cells with ambiguous flow directions
each thread takes one group out of the task pool and computes the flow directions of the cells of the group using the D8e algorithm

Fig. 6. Illustration of the second parallelization strategy. First the flow directions are computed using the D8 algorithm. All cells with ambiguous flow directions are marked. In the next phase one thread creates groups of cells with ambiguous flow directions and puts each group in the task pool. As long as the task pool is non-empty, each other thread takes a group out of the task pool and computes the flow directions of all cells of the group using the D8*e* algorithm. The grouping of cells ensures that threads will always work on disjoint sets of cells.

Table 1
Meta data parameters of the DEM datasets.

DEM dataset	Cell number	Columns and rows	Spatial resolution (m^2)	Filesize[a] (Mb)
Reintal	37,734,557	10,717 × 3521	1 × 1	290
Saidenbachtal	35,000,000	7000 × 5000	2 × 2	240
Mulde	116,674,076	6661 × 17,516	1 × 1	880

[a] Ascii-grid format (*.asc).

2002, 2005). Thus, we used an almost unloaded system for real-time measurements.

To determine the speed-up of a parallel algorithm or implementation the sequential runtime T_1 has to be measured. That is the runtime of a comparable implementation for one processor or core. The parallel runtime T_p is the runtime, in which a problem is solved using a parallel implementation with p processors or cores. The speed-up is defined by $S_p = T_1/T_p$, the efficiency by $E_p = S_p/p$. So the best or ideal speed-up would be $S_p = p$, the best efficiency $E_p = 1$. In practice these ideal values cannot be achieved in most cases because of parallelization overhead and strictly sequential parts of the algorithm. However, it is possible to get speed-ups higher than p because of cache effects. This means, more cache memory is available for the parallel program compared to cache memory usable by a sequential program running on one core.

3.4. *Study sites and datasets*

The computational experiments are executed on three different DEMs with varying high spatial resolutions (Table 1). All DEMs are based on airborne laser-scanning which are cleaned from vegetation and artificial objects like buildings. The datasets represent three geomorphological types of landscapes in Central Europe: high mountains (the Alps), low mountain ranges (the Ore Mountains) and floodplains of the lowlands (Floodplain of river Mulde; see Fig. 7). In all DEMs sinks were removed by filling within a pre-processing step.

*Study site*1—*Reintal*: The study site *Reintal* is a glacially carved valley located in the northern limestone Alps (*in German*: Wettersteingebirge) in Bavaria. At the gauging station "Blockhütte", the Partnach river drains an area of 27 km^2 below the Mount Zugspitze and flows into the Loisach river in Garmisch–Partenkirchen about 80 km south of Munich. The elevations of this catchment range from 1000 m to 2962 m with an average slope of 35.6° (Morche et al., 2008).

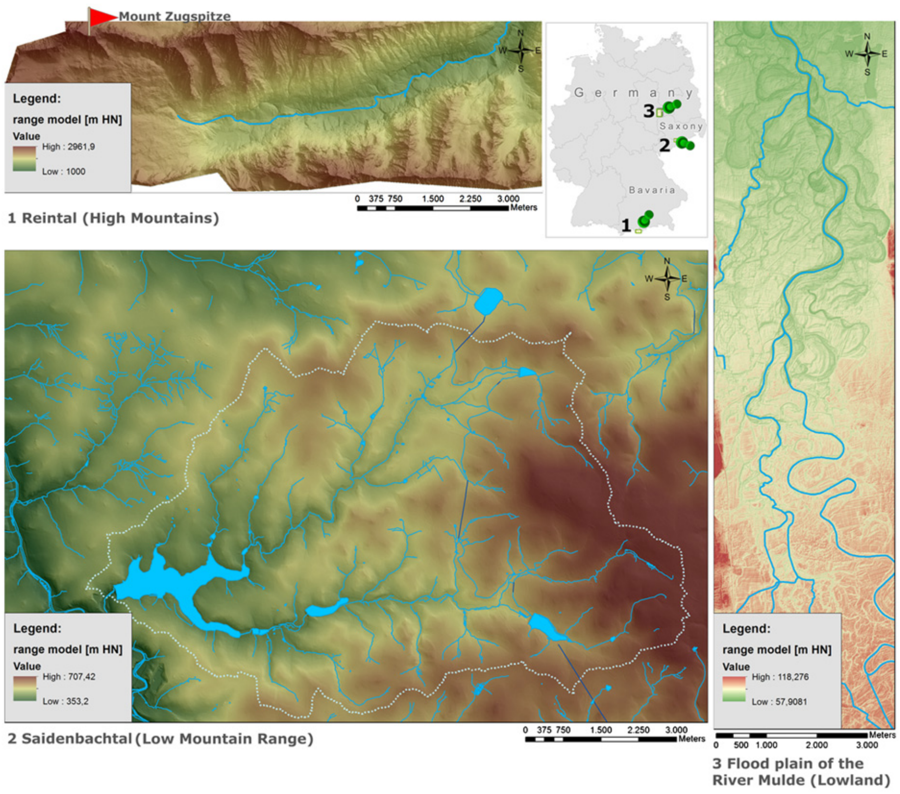

Fig. 7. Geographical location of the study sites Reintal, Saidenbachtal and Mulde in Germany. The range models of the DEMs show different types of typical landscapes in Germany. Data sources: http://www.landesvermessung.sachsen.de (Saidenbachtal), http://vermessung.bayern.de (Reintal), and http://www.lvermgeo.sachsen-anhalt.de (Mulde).

*Study site*2—*Saidenbachtal*: This study site corresponds to the catchment area of the *Saidenbachtal* reservoir and is part of the Middle Ore Mountains (*in German*: Mittleres Erzgebirge) in Saxony. The catchment area is situated near the cities Freiberg and Chemnitz. The reservoir dam was built between 1929 and 1933. It has a capacity of 22.38 million m^3, and a maximal storage area of 146.41 ha. The catchment area is about 61 km^2 and is mainly used for agriculture (Berlekamp et al., 2000). The relief represents a typical low mountain range landscape with elevations ranging from 350 m to 700 m and numerous deep and flat valleys which are flowed through by receiving waters (Thürkow, 2002). The average slope of this site is 5.5°.

*Study site*3—*Floodplain of the River Mulde*: This study site is part of an old moraine area of the "Leipzig Lowland Plain". It is situated in southern Saxony-Anhalt near the cities of Halle (Saale) and Leipzig. The soil and the relief of this area were dominated by processes under glacial and periglacial conditions during the Saalian and Weichselian glacial stages. Geomorphological, the study site is dominated by floodplains. The elevations range from 60 m to 120 m, the average slope is below 1°. The dikes are used for flood protection of farmland, settlements and commercial areas.

Table 1 shows the meta data of the used DEMs. Accordingly, all datasets are characterized by a high resolution and comparable file sizes. The first DEM has 377,125 cells with ambiguous flow directions. The largest extended neighborhood consists of 35,714 cells. In the second DEM there are 1,186,705 cells with ambiguous flow directions and the largest extended neighborhood contains 312,599 cells. The third DEM has 37,614,538 cells with ambiguous flow directions and the largest extended neighborhood consists of 3,339,361 cells. In Fig. 8, areas of extended neighborhood are shown.

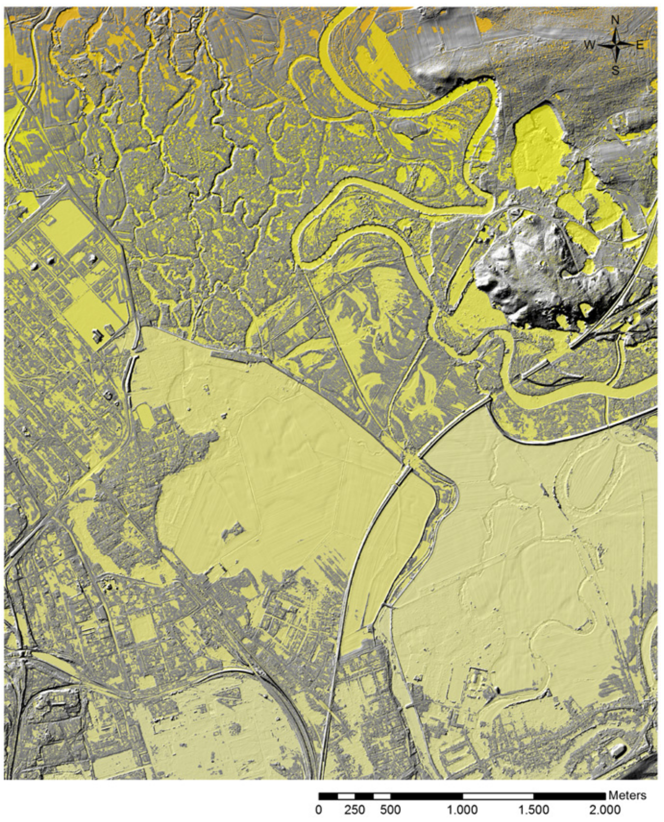

Fig. 8. Shaded relief of a DEM 3 subset and the areas of extended neighborhoods (yellow). There are many small extended neighborhoods visible, but only few huge extended neighborhoods. The average size of an extended neighborhood is about 39 cells, but two extended neighborhoods have more than one million cells. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

4. Results

The following runtime measurements were done on a symmetric multiprocessing computer (two Intel(R) Xeon(R) CPUs with four cores and 2.93 GHz each, and with 47 GB main memory). Additional runtime measurements were executed on different hardware, like a Linux Server with four AMD Opteron^{TM} processors 852 (1000 MHz) and 16 GB main memory, or a symmetric multiprocessing cluster (18 computation nodes with 16 CPU cores each and with at least 32 GB main memory each). In all cases the computations require up to 2 GB main memory. Computations on the symmetric multiprocessing cluster were executed only on one node. Speed-up and efficiency did not show significant differences on the available hardware systems. The algorithms are implemented in C++ and the Pthread library is used for parallelization, using compiler g++ in version 4.4.3.

4.1. Dividing the DEM into squares

The speed-ups for up to four threads are shown in Table 2. The number of threads is displayed in the form of "$a \cdot b$" where a is the number of rowwise partitions, and b the number of columnwise partitions.

In Fig. 9, the runtime (real time) is compared to the ideal runtime. The stacked bars show the sequential computing part and the cumulative runtime of threads. Added together they are a measure for the cost of computation. The cost of the computation is also directly obtained by measuring the CPU time used. Both clarify that rising the number of threads tends to result in an increase of used CPU time. But the effects differ between different runs with the same number of threads. The more threads we use, the more likely it is that the pre-computation steps are computationally more intensive. Note that by chance the boundary cells of the squares may have (almost) unambiguous flow directions. If so, the pre-computation step is computationally less intensive as we can see in the case 2×2 in Fig. 9(b). The load balance depends in particular on the topology and on the partition of the DEM. For instance, in the case of DEM 3 the runtime diagram shows that a rowwise partition leads to less pre-computations. In most but not all cases we have a sub-optimal load balance.

4.2. Dividing the computation into two phases

In Table 3, the speed-ups for up to four threads are presented. The runtime (real time) compared to the ideal runtime is shown in Fig. 10. The stacked bars show the cumulative calculation time of threads of both phases and the runtime which is needed to group the cells. The maximum waiting time becomes significant when using more than two threads ("Reintal" and "Mulde") or more than three threads ("Saidenbachtal"). For instance, using four threads for computing the flow directions of "Saidenbachtal", one of the four threads had to wait up to 63% of the whole parallel runtime. In case of less than four threads, a thread has to wait for a task up to 1% of the whole runtime. Regarding the dataset "Mulde" a thread has to wait up to 23% (four threads) or 28% (eight threads), respectively, of the whole parallel runtime. The

Table 2
Speed-ups of the parallel algorithm which divides the DEM into squares.

Number of threads	$1 \cdot 1$	$1 \cdot 2$	$2 \cdot 1$	$1 \cdot 3$	$3 \cdot 1$	$2 \cdot 2$
Speed-up (DEM 1 "Reintal")	0.99	1.80	1.78	2.33	2.23	3.23
Speed-up (DEM 2 "Saidenbachtal")	0.97	1.59	1.63	2.05	2.02	2.40
Speed-up (DEM 3 "Mulde")	0.98	1.53	1.55	1.46	1.81	1.82

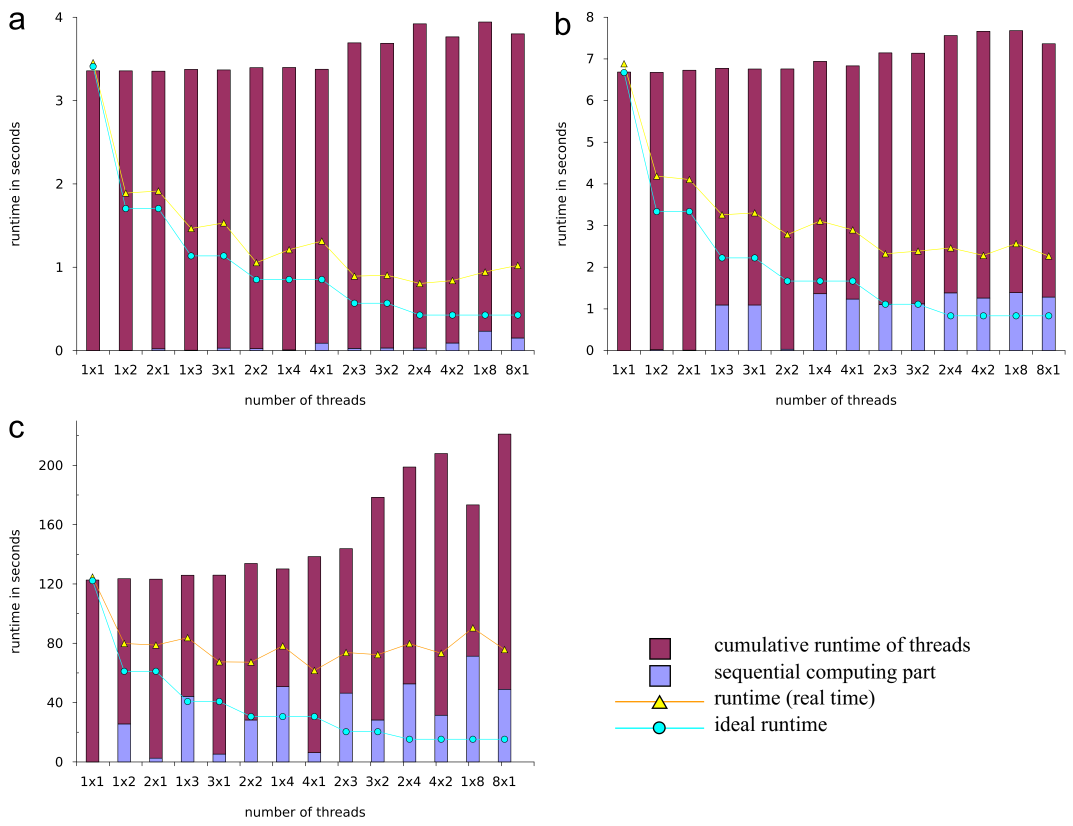

Fig. 9. Runtime diagrams of the parallel algorithm which divides the DEM into squares. The number of threads on the *x*-axis is displayed in the form of "$a \cdot b$" where a is the number of rowwise partitions, and b the number of columnwise partitions. (a) DEM 1 "Reintal". (b) DEM 2 "Saidenbachtal". (c) DEM 3 "Mulde".

Table 3
Speed-ups of the parallel algorithm which divides the computation into two phases.

Number of threads	**$1\cdot1$**	**$1\cdot2$**	**$2\cdot1$**	**$1\cdot3$**	**$3\cdot1$**	**$2\cdot2$**
Speed-up (DEM 1 "Reintal")	0.90	1.67	1.73	2.23	2.26	2.47
Speed-up (DEM 2 "Saidenbachtal")	0.94	1.82	1.84	2.38	2.40	2.58
Speed-up (DEM 3 "Mulde")	0.85	1.76	1.77	2.31	2.29	2.08

average waiting time of a thread is 14% (using four threads) or 27% (using eight threads), respectively, of its calculation time.

Working on the dataset "Saidenbachtal" and "Reintal", further investigations show that all threads have nearly equal computation time. The effort of calculation is well balanced, but in some periods there were no tasks for some of the threads. Thus, one or more threads had to wait. The maximum difference between computation time and the cumulated time of all threads were determined, too. The more the threads were used the more threads had to wait to get new tasks, but the threads had all nearly the same computation time.

Using the dataset "Mulde", we get a maximum difference between the calculation time of each thread of 10% (two threads are used) up to 18% (eight threads are used).

Several modifications of the algorithm were implemented to improve the runtime. One modifications eliminates recursion. Another one implements a heuristic which tries to change the processing order in such a way that large groups of cells were computed first. To avoid an overwhelming effort of data access, a modification – which merges small groups of cells together and puts those into the task pool – was also implemented. However, all these modifications did not show significant runtime differences.

As shown in Fig. 10, in all examples the runtime decreases but the accumulated calculation time increases with the number of threads. There is no explicit synchronization during the calculations because of the disjoint groups of cells. Some reasons are discussed in Section 2. An increased number of threads causes an increase of the overhead of administrations of the threads and an increase of random accesses which can induce for instance false sharing. The cumulated waiting time also increases.

5. Discussion and conclusion

Parallelization techniques open up new possibilities for handling large geodatasets. This is particularly true for high resolution DEMs which become increasingly available. The application of parallelization techniques for the computation of regional land-surface parameters requires specific strategies in order to enable an as efficient as possible processing. The challenge is to decrease the computing time in situations where non-localities of raster cells have to be considered.

In this study, we investigated the efficiency of parallelization techniques on the example of the non-local "extended neighborhood" of a raster cell. An extended neighborhood is used within the D8*e* algorithm to make the flow direction unique when ambiguous flow directions occur. The ordinary D8 neighborhood has ambiguous flow directions. The impact of the modified flow direction's assignment is apparent in Fig. 1 where a watershed – which was calculated using the classic D8 algorithm – is compared to a reference watershed. Using the D8*e* algorithm, the calculated watershed is almost identical to the reference watershed.

Two parallelization approaches were tested regarding their efficiency (Section 3.2):

1. The advantage of parallelization by dividing the DEM into squares is the low cost of synchronization. A disadvantage is

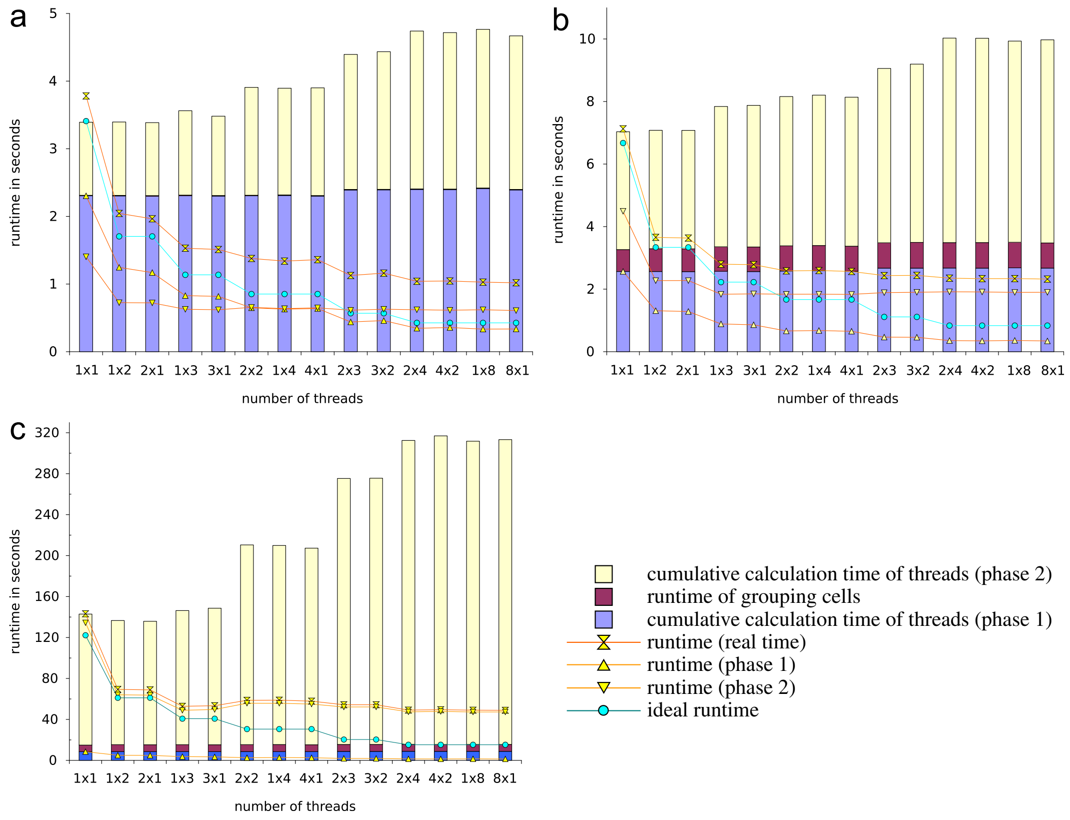

Fig. 10. Runtime diagrams of the parallel algorithm which divides the computation into two phases. The number of threads on the *x*-axis is displayed in the form of "$a \cdot b$" where *a* is the number of rowwise partitions, and *b* the number of columnwise partitions. This partitioning only applies to the first phase. (a) DEM 1 "Reintal". (b) DEM 2 "Saidenbachtal". (c) DEM 3 "Mulde".

the inefficient load balancing. On the one hand, if we have a well-adjusted load balance the speed-ups would be improved. On the other hand, speed-ups near to the best possible speed-up in all cases could not be achieved because of the sequential part (pre-computation step). Synchronization costs are insignificant in our implementations.

2. The advantage of parallelization by dividing the computation into two phases is the well-adjusted load balance. Disadvantages are the increasing cost of synchronization and data access. The sequentially grouping of cells causes increased computation expenditure and is independent from the number of threads. This parallelization alternative is more independent of the composition of the DEM because of the dynamic distribution of the cost-intensive calculations to the threads. Possible reasons for the non-ideal speed-ups have been examined. Unbalanced distribution of calculations, synchronization time and conflicts between threads because of a shared data structure can be excluded as being mainly responsible for these observations. Possible explanations are data access and the effect of false sharing caused by the high number of write and read accesses.

Neither parallelization strategy enabled speed-ups that are equal to the number of threads. This is in contrast to the parallelized calculation of local surface parameters like slope and aspect where speed-ups near to the number of threads have been achieved for all three study areas. The speed-up comparisons also revealed landscape-related dependencies. While the speed-ups for the high mountain dataset computation are higher by running the first strategy (DEM "Reintal"), the speed-ups of the second strategy have proved to be more efficient for the calculation on the low mountain and the floodplain datasets (DEM 2 "Saidenbachtal" and DEM 3 "Mulde"). The last mentioned datasets show the effect of flat areas (e.g. dams, filled sinks and floodplains) on the parallelization efficiency where a fixed DEM division into threads was carried out (first strategy). This implies that the distribution of the extended neighborhoods is fixed, too. This could lead to a sub-optimal load balancing. The second parallelization strategy is more appropriate to such DEMs because of the dynamic consideration of the extended neighborhood. This task pool-based strategy is more efficient for calculations on datasets with many data dependencies. However, the second parallelization strategy is computationally more intensive. Thus in the case of almost optimal load balances (e.g. dataset "Reintal") the speed-ups of the second parallelization strategy are lower than the speed-ups of the first one. In all datasets there are many small and a few huge extended neighborhoods (see e.g. Fig. 8). As a consequence one thread could work on just one huge extended neighborhood while the other threads have already finished.

In order to deal with the detected landscape-related dependencies of parallelization techniques, we recommend a pre-processing which enables a dataset assessment regarding the potential occurrence of non-localities. Therefore, we are testing terrain-related indices indicating flat areas and floodplains (e.g. Möller et al., 2012) or the usability of already existing geomorphological maps or classifications (Bishop et al., 2012) leading to an automatic selection of an appropriate parallelization strategy.

Further runtime measurements based on different spatial resolutions (area of DEM 1 in 2×2 m^2 and 5×5 m^2 resolution, area of DEM 2 in 5×5 m^2 resolution, and area of DEM 3 in 2×2 m^2 resolution) were executed, too. Because of the decreased runtime of the calculations caused by the lower number of cells, the overhead of administration of the threads becomes more significant. The speed-ups were a bit lower than the speed-ups presented above. Runtime measurements based on the DEMs (previous sink filling) were also executed. Because of significantly fewer flat areas, the speed-ups were significantly higher than the presented speed-ups.

The study's results are going to be implemented into the framework of the geoscientific e-learning platform GEOVLEX.[1] GEOVLEX is mainly intended to support academic education but also the knowledge exchange between scientists, students and the public while promoting user participation (Thürkow et al., 2009; Gläßer et al., 2010). Thus, the visualization of more complex, dynamic and interdependent landscape-related processes enables their better understanding.

In future work we will try to improve the computation by parallelizing the extended neighborhood computation. But this seems to be challenging because of data dependencies. We will also work on much larger datasets (about 10^9 cells). Because of runtimes greater than some minutes we will focus on applications besides web-based implementations, too. In anticipation of a growing number of cores per processor, it will be worth studying other parallelization strategies for shared memory machines.

Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at doi:10.1016/j.cageo.2012.02.023.

References

Al-Sabhan, W., Mulligan, M., Blackburn, G., 2003. A real-time hydrological model for flood prediction using GIS and the WWW. Computers, Environment and Urban Systems 27 (1), 9–32.

Berlekamp, J., Fuest, S., Gläßer, W., Matthies, M., Schreck, P., Thürkow, D., 2000. Trinkwasser in privaten Hausbrunnen Situation und Qualitätssicherung. Initiativen zum Umweltschutz, vol. 19. Deutsche Bundesstiftung Umwelt, Osnabrück, Germany.

Bishop, M., James, L., Shroder, J., Walsh, S., 2012. Geospatial technologies and digital geomorphological mapping: concepts, issues and research. Geomorphology 137, 5–26.

Blaar, H., Karnstedt, M., Lange, T., Winter, R., 2005. Possibilities to solve the clique problem by thread parallelism using task pools. In: Proceedings of the 19th IEEE Parallel & Distributed Processing Symposium, Workshop on Java for Parallel and Distributed Computing, April 4–8, 2005. IEEE, Denver, Colorado, USA.

Blaar, H., Legeler, M., Rauber, T., 2002. Efficiency of thread-parallel java programs from scientific computing. In: IPDPS 2002, Workshop on Java for Parallel and Distributed Computing. IEEE, Fort Lauderdale, USA.

Butenhof, D., 2007. Programming with POSIX Threads. Addison-Wesley, Boston.

Cormen, T., Leiserson, C., Rivest, R., Stein, C., 2009. Introduction to Algorithms, 3rd ed. MIT Press.

Gläßer, C., Thürkow, D., Dette, C., Scheuer, S., 2010. The development of an integrated technical–methodical approach to visualise hydrological processes in an exemplary post-mining area in central Germany. ISPRS Journal of Photogrammetry and Remote Sensing 65 (3), 275–281. (Theme issue "Visualization and exploration of geospatial data").

Grama, A., Karypis, G., Kumar, V., Gupta, A., 2003. Introduction to Parallel Computing, 2nd ed. Addison-Wesley, Harlow, England.

Gropp, W., Lusk, E., Skjellum, A., 1999. Using MPI. Portable Parallel Programming with the Message-Passing Interface, 2nd ed. The MIT Press, Cambridge, MA.

Gruber, S., Peckham, S., 2009. Land-surface parameters and objects in hydrology. In: Hengl, T., Reuter, H. (Eds.), Geomorphometry—Concepts, Software, Applications, Developments in Soil Science, vol. 33. Elsevier, Amsterdam, The Netherlands, pp. 171–194.

Herlihy, M., Shavit, N., 2008. The Art of Multiprocessor Programming. Elsevier, Amsterdam, The Netherlands.

Kenny, F., Matthews, B., 2005. A methodology for aligning raster flow direction data with photogrammetrically mapped hydrology. Computers & Geosciences 31 (6), 768–779.

Meyer, V., Haase, D., Scheuer, S., 2009. Flood risk assessment in European river basins—concept, methods, and challenges exemplified at the Mulde river. Integrated Environmental Assessment and Management 5, 17–26.

Mølhave, T., Agarwal, P.K., Arge, L., Revsbæk, M., 2010. Scalable algorithms for large high-resolution terrain data. In: Proceedings of the 1st International Conference and Exhibition on Computing for Geospatial Research & Application, COM.Geo'10. ACM, Washington, DC, pp. 20:1–20:7.

Möller, M., Koschitzki, T., Hartmann, K.-J., Jahn, R., 2012. Plausibility test of conceptual soil maps using relief parameters. CATENA 88 (1), 57–67.

Morche, D., Witzsche, M., Schmidt, K.-H., 2008. Hydrogeomorphological characteristics and fluvial sediment transport of a high mountain river (Reintal valley, Bavarian Alps, Germany). Zeitschrift für Geomorphologie 52 (Suppl. Issue 1), 51–77.

Nardi, F., Grimaldi, S., Santini, M., Petroselli, A., Ubertini, L., 2008. Hydrogeomorphic properties of simulated drainage patterns using digital elevation models: the flat area issue. Hydrological Science Journal 53 (6), 1176–1193.

Neal, J., Fewtrell, T., Bates, P., Wright, N., 2010. A comparison of three parallelisation methods for 2d flood inundation models. Environmental Modelling & Software 25, 398–411.

Neal, J., Fewtrell, T., Trigg, M., 2009. Parallelisation of storage cell flood models using OpenMP. Environmental Modelling & Software 24, 872–877.

Nichols, B., Buttlar, D., Farrell, J., 2003. Pthread Programming. O'Reilly Media.

O'Callaghan, J., Mark, D., 1984. The extraction of drainage networks from digital elevation data. Computer, Vision, Graphics and Image Processing 28 (3), 323–344.

Ortega, L., Rueda, A., 2010. Parallel drainage network computation on CUDA. Computer & Geosciences 36, 171–178.

Quinn, M., 2003. Parallel Programming in C with MPI and OpenMP. McGraw-Hill.

Rauber, T., Rünger, G., 2010. Parallel Programming: For Multicore and Cluster Systems. Springer, Berlin, Heidelberg.

Schäuble, H., Marinoni, O., Hinderer, M., 2008. A GIS-based method to calculate flow accumulation by considering dams and their specific operation time. Computers & Geosciences 34 (6), 635–646.

Tesfa, T., Tarboton, D., Watson, D., Schreuders, K., Baker, M., Wallace, R., 2011. Extraction of hydrological proximity measures from DEMs using parallel processing. Environmental Modelling & Software 26 (12), 1696–1709.

Thürkow, D., 2002. GIS-basierte Methoden zur Analyse der Wasserqualitätsentwicklung in Trinkwasserbrunnen am Beispiel des Einzugsgebietes der Saidenbachtalsperre Erzgebirge. Ph.D. Thesis, Martin-Luther-Universität Halle-Wittenberg.

Thürkow, D., Gläßer, C., Scheuer, S., Schiele, S., 2009. Visualisation of hydrological processes with GEOVLEX: introduction of an integrated methodical-technical online learning approach. In: König, G., Lehmann, H. (Eds.), Tools and Techniques for E-Learning, International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences. In: Proceedings of the ISPRS Working Group VI/1-VI/2, vol. XXXVIII-6/W7. International Society of Photogrammetry and Remote Sensing, Potsdam, Berlin, Germany.

Volk, M., Lautenbach, S., vanDelden, H., Newham, L.T.H., Seppelt, R., 2010. How can we make progress with decision support systems in landscape and river basin management? Lessons learned from a comparative analysis of four different decision support systems. Environmental Management 46, 834–849.

Wilson, J., Gallant, J., 2000. Terrain Analysis Principles and Applications. John Wiley and Sons.

Wilson, J., Lam, C., Deng, Y., 2007. Comparison of the performance of flow-routing algorithms used in GIS-based hydrologic analysis. Hydrological Processes 21 (8), 1026–1044.

Wood, J., 2009. Overview of software packages used in geomorphometry. In: Hengl, T., Reuter, H.I. (Eds.), Geomorphometry—Concepts, Software, Applications, Developments in Soil Science, vol. 33. Elsevier, Amsterdam, The Netherlands, pp. 257–267.

Zhu, Q., Lin, H.S., 2009. Simulation and validation of concentrated subsurface lateral flow paths in an agricultural landscape. Hydrology and Earth System Sciences 13 (8), 1503–1518.

[1] http://www.geovlex.de