Contents lists available at ScienceDirect

Computers & Geosciences

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

Regional-scale calculation of the *LS* factor using parallel processing

Kai Liu[a,b,c], Guoan Tang[a,b,c,*], Ling Jiang[d], A-Xing Zhu[c,e], Jianyi Yang[a,b,c], Xiaodong Song[f]

[a] *Key Laboratory of Virtual Geographic Environment, Nanjing Normal University, Ministry of Education, Nanjing 210023, China*
[b] *State Key Laboratory Cultivation Base of Geographical Environment Evolution, Jiangsu Province, Nanjing 210023, China*
[c] *Jiangsu Center for Collaborative Innovation in Geographical Information Resource Development and Application, Nanjing 210023, China*
[d] *Anhui Center for Collaborative Innovation in Geographical Information Integration and Application, Chuzhou University, Chuzhou 239000, China*
[e] *Department of Geography, University of Wisconsin-Madison, Madison, USA*
[f] *State Key Laboratory of Soil and Sustainable Agriculture, Institute of Soil Science, Chinese Academy of Sciences, Nanjing, Jiangsu 210008, China*

ARTICLE INFO

Article history:
Received 22 August 2014
Received in revised form
20 December 2014
Accepted 2 February 2015
Available online 11 February 2015

Keywords:
LS factor
Parallel computing
Soil erosion model
Regional scale
Digital terrain model

ABSTRACT

With the increase of data resolution and the increasing application of USLE over large areas, the existing serial implementation of algorithms for computing the *LS* factor is becoming a bottleneck. In this paper, a parallel processing model based on message passing interface (MPI) is presented for the calculation of the *LS* factor, so that massive datasets at a regional scale can be processed efficiently. The parallel model contains algorithms for calculating flow direction, flow accumulation, drainage network, slope, slope length and the *LS* factor. According to the existence of data dependence, the algorithms are divided into local algorithms and global algorithms. Parallel strategy are designed according to the algorithm characters including the decomposition method for maintaining the integrity of the results, optimized workflow for reducing the time taken for exporting the unnecessary intermediate data and a buffer–communication–computation strategy for improving the communication efficiency. Experiments on a multi-node system show that the proposed parallel model allows efficient calculation of the *LS* factor at a regional scale with a massive dataset.

1. Introduction

Among the factors in the Universal Soil Loss Equation (USLE) (Wischmeier and Smith, 1978) and the revised USLE (RUSLE) (Renard et al., 1997), the extraction of the *LS* factor, which reflects the influence of terrain on soil erosion, is a key issue in the applications of these models (Kinnell, 2010). The *LS* factor contains two parts: the slope length factor (*L*) and the slope factor (*S*). It is widely believed that slope length is the more problematic part. The development of GIS allows for automatic extraction of slope length from high resolution DEMs, thus an inefficient manual process is avoided. In general, there are three kinds of extraction methods: unit stream power methods (Moore and Burch, 1986; Mitasova, 1996), contributing area methods (Moore and Wilson, 1992; Desmet and Govers, 1996) and grid-based methods (Hickey, 1994, 2000). In recent years, researches have primarily adopted grid-based methods, which overcome the limitation of the former two methods in predicting soil deposition on topographically driven zones (Winchell et al., 2008).

In the grid-based method proposed by Hickey et al. (1994), slope length is calculated from the high points along the direction with maximum downhill slope angle at the same time that converging flows and deposition areas are taken into consideration. Based on the work of Van Remortel et al. (2001, 2004), Zhang et al. (2013) proposed a definition of distributed watershed erosion slope length (DWESL) with the following two improvements: (1) for the runoff node, the slope length equals to the total lengths of all the surrounding cells flowing into it, instead of just considering the longest value; and (2) the slope length calculation must stop at a channel. The experiments showed that the results got by DWESL method were more similar to the manual method than those by existing algorithms.

In many studies, USLE/RUSLE has been employed for the quantitative assessment of soil erosion at large watersheds or even on a regional scale (Yang et al., 2003; Fu et al., 2005). At the same time, the development of acquisition method for spatial data means that the geospatial dataset size is increasing at an exponential rate (Cheng et al., 2013) and high resolution DEM data for large regions are becoming more available. As a result, the computational time increases significantly and to the level becoming a bottleneck for applying these models over large areas

* Corresponding author at: Key Laboratory of Virtual Geographic Environment, Nanjing Normal University, Ministry of Education, Nanjing 210023, China. Fax: +86 25 85891347.
E-mail addresses: liukaigis@gmail.com (K. Liu), tangguoan@njnu.edu.cn (G. Tang).

http://dx.doi.org/10.1016/j.cageo.2015.02.001
0098-3004/© 2015 Elsevier Ltd. All rights reserved.

(Jiang et al., 2013). It is limited by computer storage and calculation ability that traditional serial processing cannot meet user demand due to the long response time. Currently, two methods are adopted for *LS* factor calculation on regional scale. One method employs a set of prediction rules to generate *LS* factor based on a number of attributers which control the landform types such as materials, climate and regional geomorphology (Lu et al., 2003). Another extracts the topographic index based on low resolution DEMs, and then gets available values via a scale transformation (Cheng et al., 2009). However, both methods could only be regarded as an expedient measure, the fast and accurate calculation of *LS* factor on regional scale is yet to be developed.

Developments in computer technology have improved computation ability by using parallel processing. Recently, parallel technology has been widely used in digital terrain analysis, such as parallel drainage extraction (Gong and Xie, 2009; Qin and Zhan, 2012), parallel visibility analysis (Kidner et al., 1997; Wang et al., 2015), and parallel hydrological analysis (Tesfa et al., 2011; Wu et al., 2013; Liu et al., 2013). In order to make the parallel programming become easy, some raster-based programming libraries are proposed such as Parallel Raster Processing Programming Library (pRPL) (Guan and Clarke, 2010) and Parallel Raster-based Geocomputation Operators (PaRGO) (Qin et al., 2014). Among the existing literatures, Message Passing Interface (MPI) is widely used to parallelize algorithms for its adaptable to various parallel computing environments and the rich programming interfaces (Cheng et al., 2013). Compared with the serial algorithms, the MPI parallel algorithms can achieve a huge improvement in processing time (Jiang et al., 2013). There is, however, little published research on the *LS* factor calculation using parallel computing.

The aim of this paper is to propose a parallel approach that can be applied on a regional scale calculation of the *LS* factor. The structure of this paper is as follows. Section 2 details each algorithm for the *LS* factor calculation. Section 3 introduces the parallel approach. According to the algorithm characters, two parallel strategies have been designed for both local algorithms and global algorithms. Section 4 discusses the effectiveness of the parallel algorithms, and finally, the concluding remarks are given in Section 5.

2. Serial *LS* factor calculation algorithm

2.1. Overview of the LS factor algorithms

The *LS* factor can be obtained according to the following expressions (McCool et al., 1989; Liu et al., 2000):

$$LS = L*S \tag{1}$$

$$L = \left(\frac{l}{22.13}\right)^m \tag{2}$$

$$\lambda_{i,j} = \sum_{x=0,y=0}^{x=i,y=j} \sum_{k=1}^{m} \lambda_{x,y} \tag{3}$$

$$\beta = (\sin\theta/0.089)/[3\sin\theta^{0.8} + 0.56] \tag{4}$$

$$S = \begin{cases} 10.8\sin\theta + 0.03 & \theta < 5^\circ \\ 16.8\sin\theta - 0.05 & 5^\circ \le \theta \le 14^\circ \\ 21.91\sin\theta - 0.96 & 14^\circ \le \theta \end{cases} \tag{5}$$

where l is the slope length, m is a variable length-slope exponent, β is the ratio between rill erosion and interrill erosion, θ is the angle of the slope, S is the slope factor and L is the slope length factor.

In the calculation of *LS* factor, the slope algorithm is relatively simple, while the slope length algorithm is more complicated. In this paper, the DWESL method proposed by Zhang et al. (2013) is used and its formula is as follows:

$$\lambda_{i,j} = \sum_{x=0,y=0}^{x=i,y=j} \sum_{k=1}^{m} \lambda_{x,y} \tag{6}$$

where k is the code of the surrounding cells at coordinates (x, y), and $\lambda_{x,y}$ is the slope length of each cell.

According to Eq. (6), for the cells where convergent flow occurs, the slope length of the center cell should equal to the sum value of the slope length of the surrounding cells that flow into it, which shows the particularity between DWESL and the other grid-based approaches. As shown in Fig. 1, point A is the convergent point of flow AC and BC. The slope length of A takes the sum value of projection length $A'B'$ and $A'C'$ instead of using the longest length. In addition, two cutoff factors are considered in DWESL. One is slope cutoff factor, such as Point D, where slope becomes gentle and deposition happens. In this case, the calculation of slope length should be restarted. The other one is the channel network cutoff factor. At Point F, where channel network occurs, the slope length should be set to a constant, generally zero.

2.2. Algorithm flow

The algorithm flow is based on the definition above and contains the six steps shown in Fig. 2. The input DEM data need to be preprocessed before the calculation in order to fill surface depressions and remove flat areas. In this paper, the preprocessed processing has been implemented using TauDEM software (Wallis et al., 2009).

2.3. Flow accumulation calculation

Among the calculation workflow above, the algorithms for flow accumulation and slope length calculations are more complex due to their accumulation process. It is obvious that the flow accumulation calculation for each cell depends on its upstream cells. In other word, to calculate the flow accumulation, the number of all cells that drain into it directly or indirectly should be figured up (Wallis et al., 2009). In the process, all cells are actually composed of two kinds of cells: (1) source cell, located in the peak or ridge or around the edge of DEM dataset with no cell drain into; and (2) normal cell, which does not belong to source cells (Fig. 3a). The serial algorithm for flow accumulation is as follows:

Step 1: Each cell calculates the number of neighbor cells which flow into it. If the number equals to n, then cell is assigned to $-n$; if the number equals to 0, then the cell value is assigned to zero and it should be pushed onto the stack.

Step 2: Each cell in the stack should be popped off in turn, if its next cell n in flow direction is less than -1, the value of cell n pluses one and waits for the next calculation. If its next cell n in flow direction equals to -1, which means the computational condition is achieved, cell n is assigned to the sum value of neighbor cells. Iterating the calculation for each cell from cell n until the next cell does not equal to -1.

Step 3: As soon as the stack is getting empty, which means all the cells have been assigned to its flow accumulation value, then the computation task can be completed.

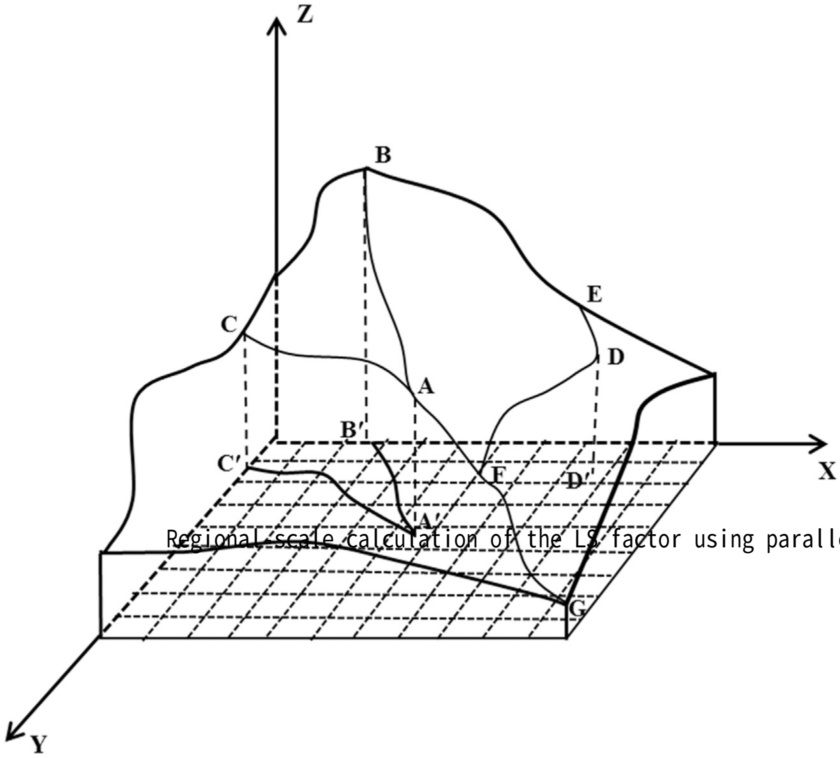

Fig. 1. Schematic representation of distributed watershed erosion slope length.

2.4. *Slope length calculation*

Slope length calculation is the key step in *LS* factor extraction. As required in DWESL, flow direction is the necessary condition in the calculation, while slope cutoff and channel cutoff must be taken into consideration as well. In the process, all cells can be classified into four types: (1) source cell, whose definition is the same as in the flow accumulation described above; (2) slope cutoff cell, where the sediment is deposited. It is identified by the change ratio in slope angle between one cell and the next along the flow direction. If the change ratio is greater than the cutoff factor, the next cell is regarded as the slope cutoff cell and the slope length from this direction cannot be accumulated; (3) channel cutoff cell, where the cell is a part of a channel and the slope length value should be set as a constant, which is generally zero; and (4) normal cell, which does not belong to source cell, slope cutoff cell or channel cutoff cell (Fig. 3b).

The calculation process contains three steps:

Step 1: Cell identification. An array *T* is created for identifying the cell type. If the cell *m* is channel cutoff cell, *T(m)* is assigned to 1 and its slope length value is 0. If the cell *m* belongs to source cell, it is pushed onto the stack, and *T(m)* is assigned to 1. For the other cells, if the number of the neighbor cells flowing inside is *n*, *T(m)* is assigned to −*n*.

Step 2: Non-accumulation slope length calculation. Non-accumulation slope length, only considering the length of the cell itself, has the following regulation:

$$NSL=\begin{cases}0 & \text{channel cutoff cell}\\ \frac{1}{2}\text{cell size} & \text{source cell or slope cutoff cell with horizontal}\\ & \text{flow direction}\\ \frac{\sqrt{2}}{2}\text{cell size} & \text{source cell or slope cutoff cell with diagonal lir}\\ \text{cell size} & \text{normal cell with horizontal or vertical flow dir}\\ \sqrt{2}\,\text{cell size} & \end{cases} \tag{7}$$

Step 3: Accumulation slope length calculation. The calculation strategy for slope length is similar to that for flow accumulation. Each cell in the stack should be popped off in turn, if the next cell *n* in the direction of flow does not equal -1, the value of cell *n* increases by one and waits for the next calculation. If the next cell *n* along flow path equals -1, then cell *n* is assigned to the sum slope

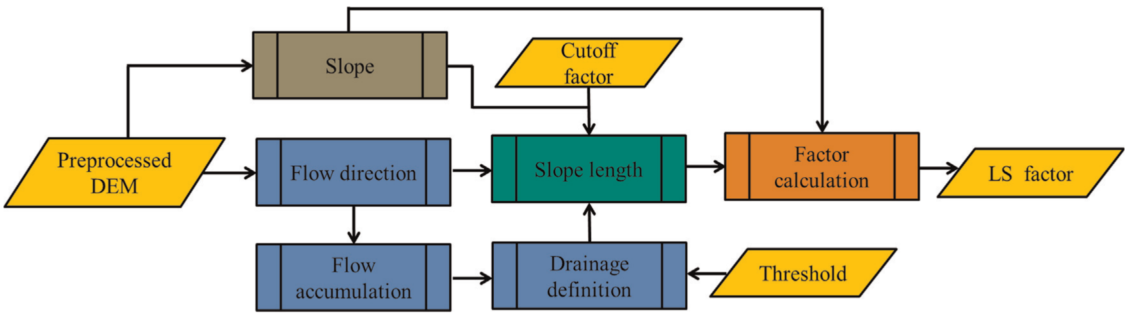

Fig. 2. Workflow for *LS* factor calculation from DEM.

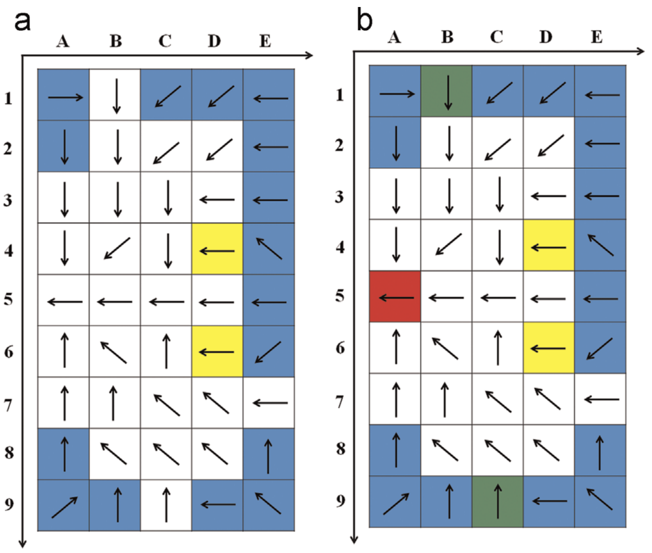

Fig. 3. Two flow-direction matrixes to illustrate cell types in flow accumulation algorithm and slope length algorithm. (a) Blue cells which are located in the border and yellow cells with no cell draining into are regarded as source cells, and the rest white cells are normal cells. (b) Blue cells and yellow cells are source cells. Assuming that the green cells B1 and C9 satisfy the conditions of slope cutoff according to the slope matrix, and then they are regarded slope cutoff cells; Assuming that the red cell satisfies the conditions of channel cutoff according to channel matrix, it is regarded as channel cutoff cell. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

length of neighbor cells draining into it. In the calculation, the slope length cannot be accumulated at slope cutoff cell. The same calculation is iterated from cell n until the next cell does not equal -1. The calculation process is repeated until the stack becomes empty.

3. Parallel computations

3.1. Approach to parallelization

In the *LS* factor calculation process, all the algorithms can be divided into two types, local algorithms and global algorithms, based on the existence of data dependency during the computation process. In local algorithm, the calculation for each cell is independent, which means the calculation results rely only on the input data and can be completed after traversing the input data once. The local algorithm can be further divided into two types: (1) point computation, such as the algorithm for drainage-network definition and factor calculation, which has no relevance with other grids; (2) regional computation, such as the algorithm for slope and flow direction, in which the computation process needs a limited region (Xie, 2012). Regional computation depends on the center cells and its neighbor cells and different algorithm employs different scope transition, the Moore neighborhood (Cheng et al., 2012) are used for both slope algorithm and flow direction algorithm in *LS* factor calculation. Compared with the local algorithms, the processing of global algorithm is not independent, which means the calculation of most cells may rely on the calculation results of other cells.

The existing researches prove that local algorithms have better parallelism and communication strategy should be designed for the parallelization of global algorithms to accommodate the influence of data dependency. The parallel strategies in this paper are designed according to the algorithmic characters discussed above, including the decomposition method to maintain the integrity of the results, optimized workflow to reduce the time taken for exporting the unnecessary intermediate results and the communication strategy to improve the communication efficiency.

3.2. Decomposition method

Data partitioning is a key problem in parallel computing, and requires particular attention (Gong and Xie, 2009; Song et al., 2013). The existing partition strategies can be divided into two types: regular strategy and irregular strategy. Regular decomposition is widely used for its simplicity and high efficiency, while irregular strategy is more concerned with maintaining the integrity of the terrain in the process.

In this paper, a striped partitioning approach which is a regular strategy proposed by Wallis et al. (2009) is used. The DEM data are divided horizontally into parts of equal size and mapped to size processes, with any portion of the grid that is remaining is attached to the last divided portion. Each portion contains three elements: (1) data region: the data for computing in each process. According to the partitioning approach, row number of each process is

$$\text{Prows} = \begin{cases} \text{Total Rows/size} & \text{rank} \neq \text{size} - 1 \\ \text{Total Rows/size} + \text{Total Rows \% size} & \text{rank=size} - 1 \end{cases} \quad (8)$$

where Total Rows denotes the total row numbers of global dataset, size is the rank number, rank is the identification of each process (rank=1, 2,…., size).

(2) Anchor point: the starting point for reading data from a grid file and through which the result of each process can be written to the result file, and the coordinate of anchor point is

$$\text{col}=0$$

$$\text{row} = \begin{cases} \text{rank} \times \text{Prows} & \text{rank} \neq \text{size} - 1 \\ \text{rank}\times(\text{Total Rows/size}) & \text{rank=size} - 1 \end{cases} \quad (9)$$

where col and row denote the column number and row number of the anchor point respectively.

(3) Buffer area: a row of cells as overlapping area from adjacent process, above and below it. Through buffers, each process has access to an adjacent subdomain with litter commutation. Fig. 4 illustrates the striped partitioning strategy.

3.3. Optimized design of the workflow

To achieve the final *LS* factor, the workflow mentioned in Section 2.2 contains six parts which is too complex for the parallel algorithms, especially considering the time taken for the I/O disk. The method for solving this problem is to reduce intermediate results by merging the algorithms. In this paper, the optimized design of the workflow considering two factors. One is whether the calculation results will be used for more than one algorithm or not. For example, the flow-direction matrix which is the input dataset for both flow accumulation and slope length, therefore flow direction is not suitable for merging with other algorithms. Another factor depends on the algorithm characters. It is obvious that point computation can be easily merged with other algorithms for its independent calculation of each cells. In this paper, flow accumulation and drainage definition can be merged; Slope length and factor calculation can be merged as well. After optimization, the workflow includes only four algorithms, namely flow direction, stream (combination of flow accumulation and drainage definition), slope and *LS* factor (combination of slope length and factor calculation).

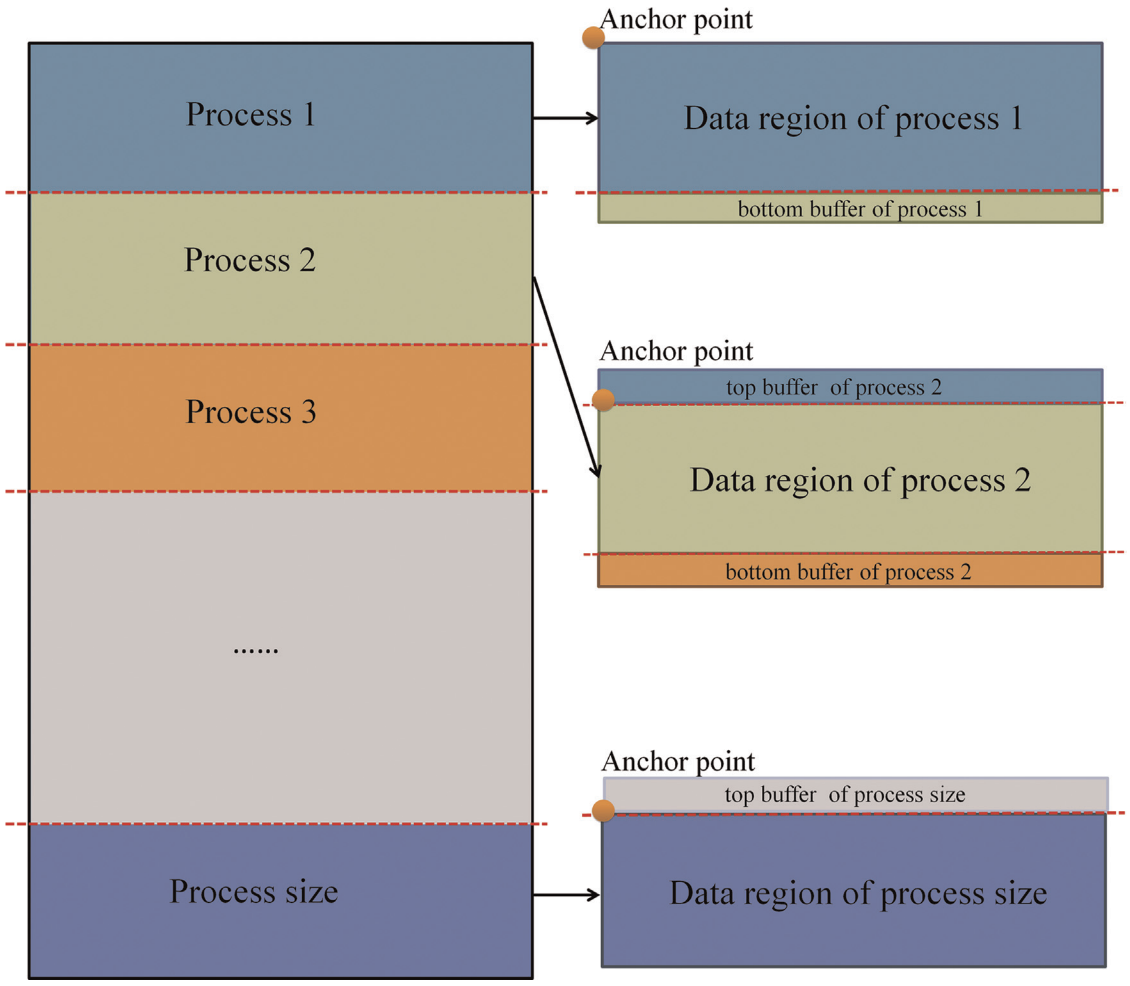

Fig. 4. Illustration of striped partitioning strategy.

3.4. *Buffer–communication–computation strategy*

In the parallel process of global algorithms, the computation is implemented in two steps: internal and external computation. In the internal computation step, if the cell and its dependent cells, labeled upstream cells in this paper, are all in the same subdomain, the computation can be completed within this step. After internal computation, some more cells can get their values. However, the cell and its dependent cells can also be divided into different subdomains according to the partitioning strategy, in which an external computation step is carried out to exchange the processing results from adjacent processes. The illustration of data dependency is shown in Fig. 5. According to flow direction in Fig. 5a, the blue cells in Fig. 5b are the source cells which can achieve their values directly without considering other cells. After internal computation, the computation result is shown in Fig. 5c in which the white cells which have the data dependency on other process should wait for the external computation.

Further study shows that, the data dependency between processes originates from the border rows of subdomains and the dependency information for each subdomain can be obtained through the communications between the border rows of adjacent processes. If all the cells in border rows have no data dependency on the adjacent processes, it proves that all the cells in this subdomain can obtain their values within internal computation. However, if some cells in border rows have the data dependency on the cells in other processes, these cells and their downstream cells should wait for the data exchange before beginning their computations. That is to say the statuses of the cells in border rows determine the status of other cells. If all the cells in the border rows have finished the calculations, which proves that all the cells have obtained their values and there is no need to exchange the information further.

In this paper, the implementation of data communication between processes is simplified by MPI. To overcome any extra communication, a buffer–communication–computation strategy, the core part of the parallelization strategy for global algorithms, is designed according to the following steps.

Step 1: The calculation is implemented in each process once. The ability of the cells to complete the calculation depends on whether or not the cell and its dependent cells are in the same subdomain.

Step 2: Upon completing step 1, if any cell in the boundary (top or bottom row) has not completed the calculation, each process updates the buffers by swapping its boundaries with neighboring processes (Fig. 6a).

Step 3: Each process loops through the buffers. If the cell in the buffer has achieved its final value and does not have to be visited, it should be pushed onto the stack and its visited flag is assigned 1. Each cell in the stack is popped off, which can be regarded as the source cell. The calculation is repeated from the source cell to update the unfinished cells (Fig. 6b).

Step 4: If any cell in the top or bottom rows of the subdomain has not completed the calculation, step 2 and 3 are repeated until all the cells have achieved their values.

To illustrate the process of the buffer–communication–computation strategy, an example of a slope length algorithm is given

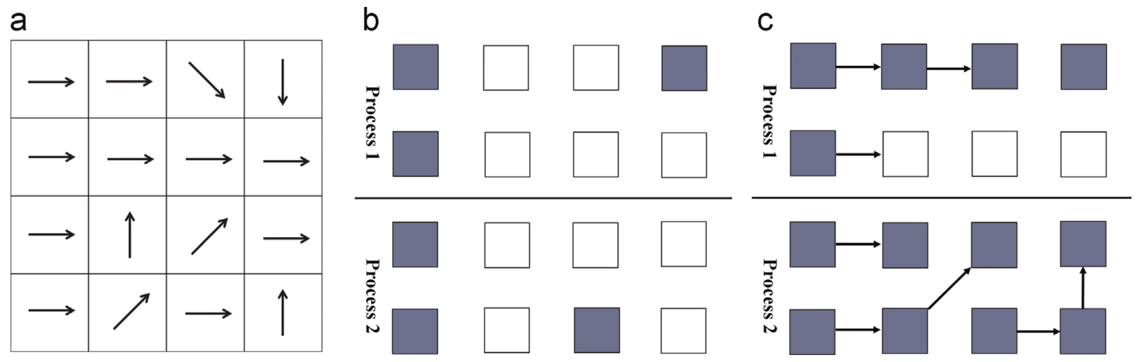

Fig. 5. Illustration of data dependency in computation processing. (a) Flow direction matrix, (b) blue cells are the sources cells, (c) internal computation result in which white cells are the unfinished cells. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

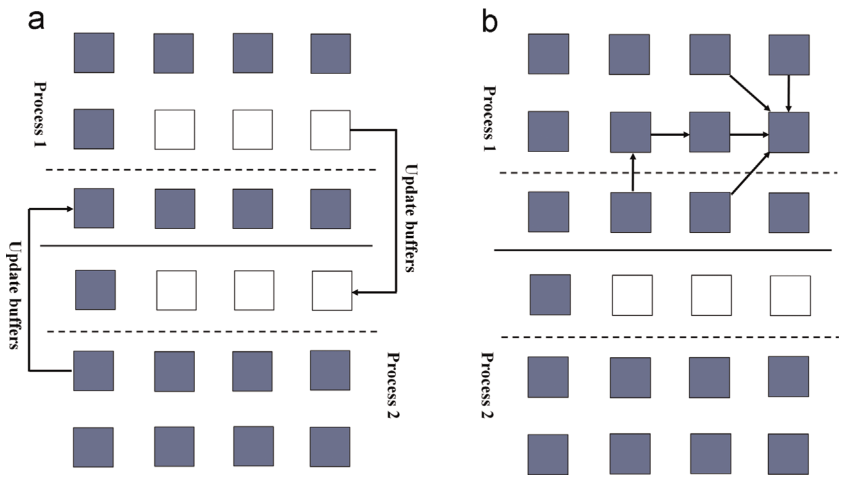

Fig. 6. Illustration of buffer–communication–computation strategy. (a) Buffer areas are updated through communication, (b) buffer areas are used for the computation of the unfinished cells.

with a nine-row flow-direction matrix (see Fig. 3). Because the process number is three, each process can be assigned a three-row sub-dataset. After the first-time calculation, some cells have achieved a slope length value without communication (Fig. 7a). Each process then searches the unfinished cells from both top and button rows and the buffers are updated through communication (Fig. 7b). In each buffer, the cells that have achieved their value and have not been visited are regarded as the source cells. Each process updates the unfinished cell's value from the source cells (Fig. 7c). If unfinished cells still exist after a respective search of the top and bottom rows, one more buffer communication is completed (Fig. 7d). Each process then repeats the search of the source cell from which the recalculation is done (Fig. 7e). The calculation is not complete until there are no unfinished cells in all processes (Fig. 7f).

3.5. *Programming implementation*

The MPI parallel algorithm in this paper was developed in C++ programming language. Fig. 8 shows an integral flowchart of the parallelization workflow in the extraction of the *LS* factor and it is described in the following steps.

Step 1: Each process is initialized with the number of processes and parameters involving the algorithm.

Step 2: The input dataset is subdivided into subdomains and the result file is created by process 1.

Step 3: Each process reads data from the input file according to its anchor point.

Step 4: Subtasks are implemented in each process.

Step 5: If the algorithm is the global type, the buffer–communication–computation strategy should be used to make sure that all the cells finish their calculations.

Step 6: Each process writes the computation results to the result file. As illustrated in Fig. 4, the computation results of each part are written to the global dataset according to their anchor points.

Step 7: Each process frees memory and the task is completed.

For the local algorithms, the implementation of parallelization contains three parts: reading input data, performing the calculation task and writing result data. For the parallelization of global algorithms, the communication operations should be considered. The C++ pseudo code of the parallelization strategy for local and global algorithms can be found in Appendices A.1 and A.2, their total time spent are given as follows:

$$T_{local} = \max_{i\in[1,np]} t_{read} + \max_{i\in[1,np]} t_{cpt} + \max_{i\in[1,np]} t_{write} \tag{10}$$

where t_{read} and t_{write} denotes the time-taken for reading and writing data respectively, t_{cpt} is the time overhead to execute the calculation task.

According to the optimized workflow, all the algorithms should be implemented as the following sequence: slope, flow direction, stream, and the final *LS* factor. Each process executes the next step simultaneously when all the processes have finished the prior step. The total time spent for the workflow of *LS* factor calculation contains four parts which can be estimated as follows:

$$T = T_{slope} + T_{flowdir} + T_{stream} + T_{LSfactor} \tag{11}$$

where T_{slope}, $T_{flowdir}$, T_{stream}, $T_{LSfactor}$ denotes the time for the calculation of slope, flow direction, stream and the final *LS* factor calculation.

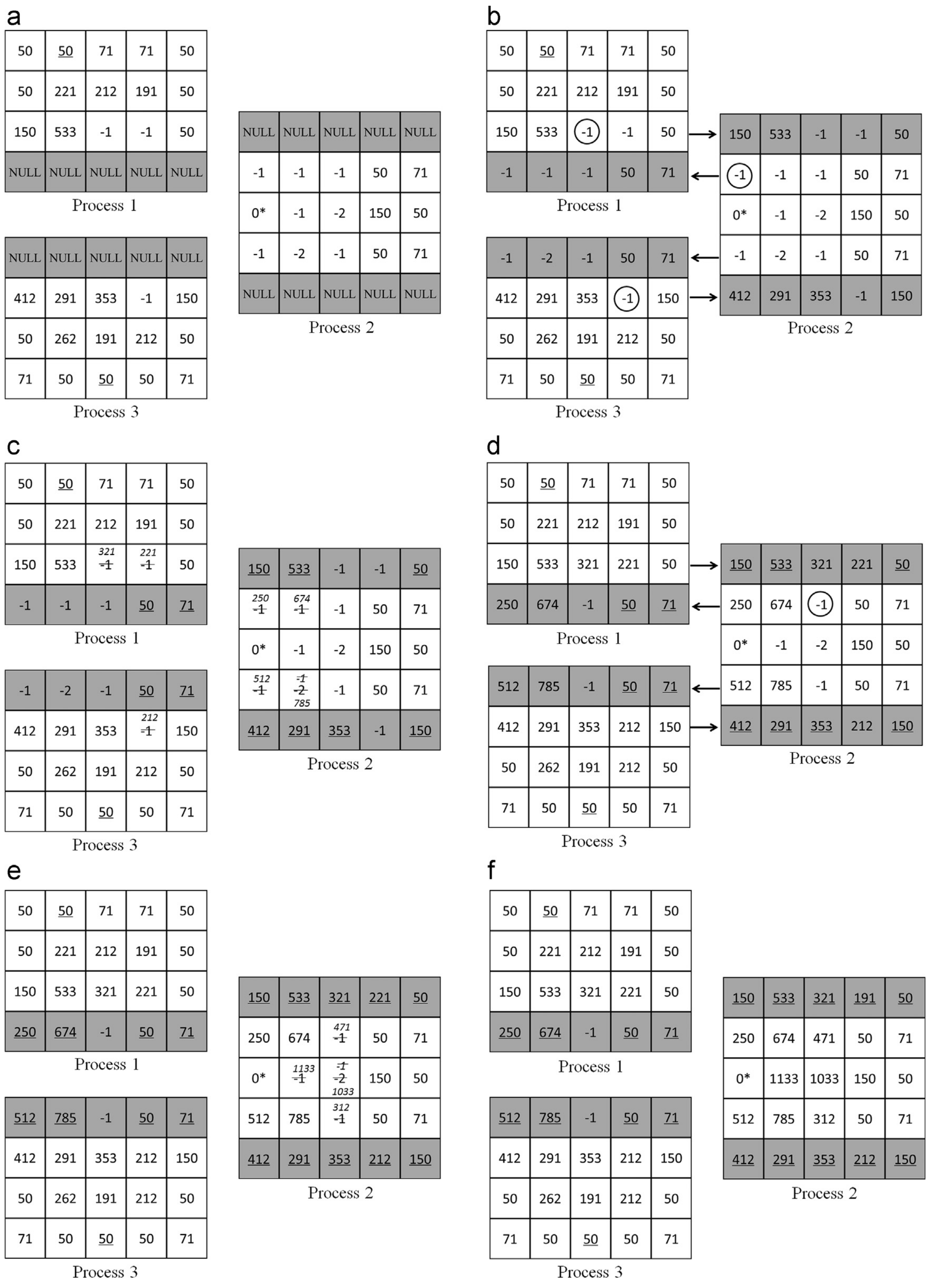

Fig. 7. Illustration of buffer–communication–computation strategy using flow-direction matrix in Fig. 3 and slope length algorithm is taken for example. (a) Calculation results after independent processing (gray rows mean the buffers, the cells with underlines in process 1 and process 3 are the slope cutoff cells, the cell with an asterisk in process 2 is the channel cutoff cell). (b) Searching unfinished cells in top and bottom rows and updating the buffers. (c) Calculation results after updating unfinished cells from buffers. (d) Searching unfinished cells in top and bottom rows and updating the buffers. (e) Updating unfinished cells from source cells in buffers. (f) Searching unfinished cell in top and bottom rows (all cells of data regions have achieved their slope length values at the moment).

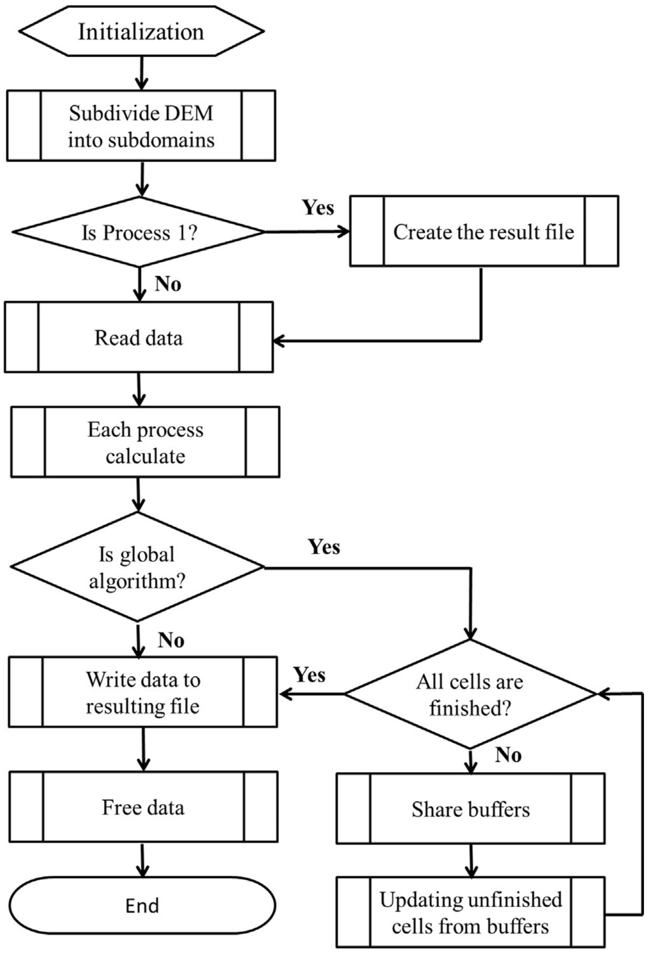

Fig. 8. Parallelization workflow.

4. Experiments and results

4.1. Experimental basis

In order to evaluate its accuracy and effectiveness, the parallel algorithm were conducted on an 80-core cluster composed of 10 diskless nodes that were connected through an Ethernet network with Gigabit (1000 Mbit/s) speed. Each node had dual quad-core Intel(R) Xeon(R) E5620 processors (2.40 GHz, 16 GB RAM) under a Linux operating system.

The Loess Plateau of China, with a total area of 640,000 km^2, is world-renowned for its unique loess landform and severe soil erosion. The USLE/RUSLE model is usually applied to model the soil erosion in this area. However, the massive data volume due to the size of the area presents a computational challenge to traditional serial algorithms. In this paper, two different datasets covering the entire loess plateau were employed as the test data (Fig. 9). The 90- m resolution dataset with dimensions of 11472×17087 (748 MB) is regarded as the smaller one and the larger one is at 30-m resolution with dimensions of 34417×51261 (6.57 GB).

4.2. Experimental results

To judge the accuracy of the parallel algorithm, serial algorithm for *LS* factor calculation was implemented with the same dataset. There is no difference of the calculation results between the serial algorithm and the parallel computing approach, which proves the availability by using the parallel processing. Fig. 10 shows the calculation results by using the proposed parallel computing approach in this paper with a 30-m resolution dataset. Fig. 10a is a map of the *LS* factor for the entire loess plateau and shows its spatial distribution at a regional scale. Fig. 10b, c and d are the calculation results from three small watersheds. It confirms that the calculation results can not only be applied for some regional studies, but also maintain their utility at a small watershed scale.

Since the main purpose of the study is to improve computation efficiency, more attention is paid here to evaluate its parallel efficiency. Fig. 11 shows the run time taken to complete each steps of

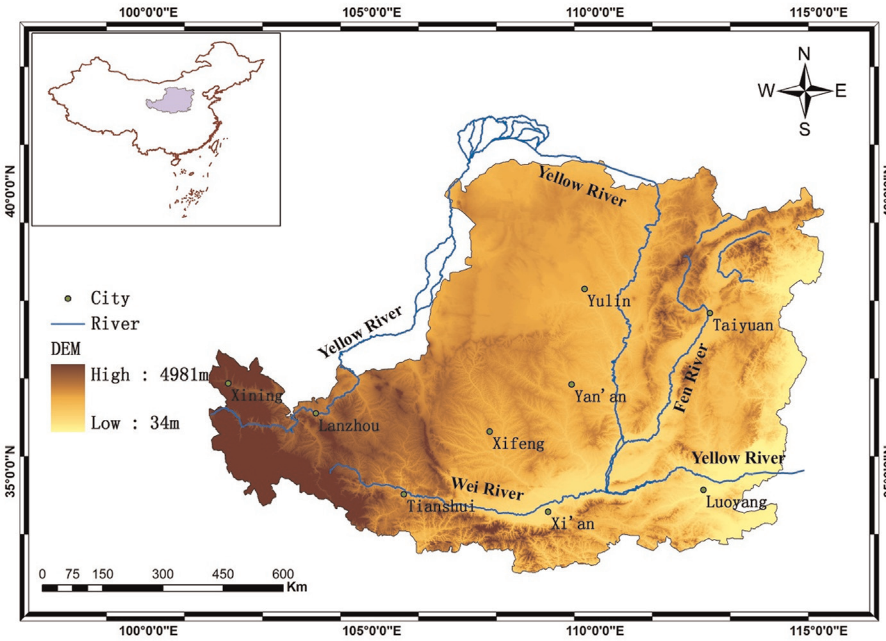

Fig. 9. The Loess plateau study area.

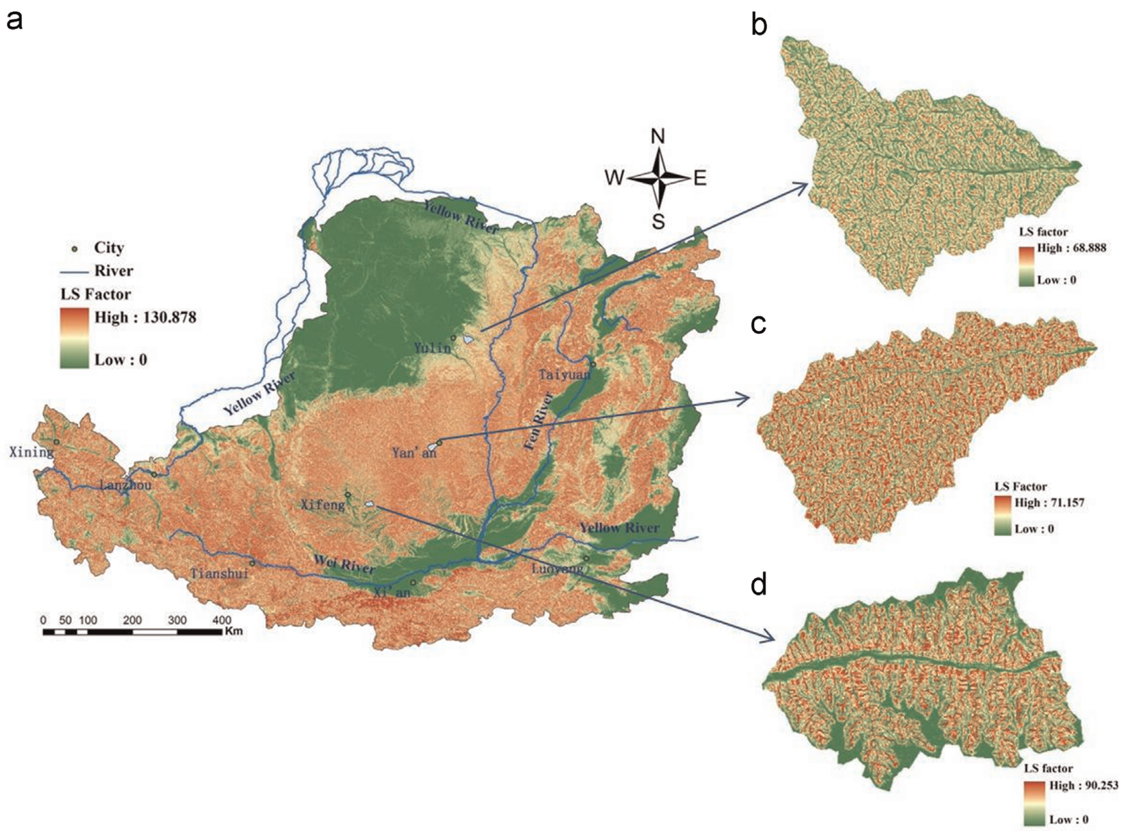

Fig. 10. Map of calculation results.

the *LS* factor calculation using both serial algorithm and the parallel MPI algorithm. Between 1 and 16 processes, the execution time of both the 90-m dataset and 30-m dataset are significantly reduced, which are shown in Fig. 11a and b, respectively. Because all the processes are running simultaneously, more processes can be completed in less execution time. However, from 16 to 64 processes the run time decreases slowly, even experiencing a rise trend from 32 to 64 processes. This is because that as the increase of the processes number, the subdata assigned to each process gets smaller, resulting in more overheads of communications. It should be noted that the time taken by using parallel algorithm with one process costs more time for that by using serial algorithm, due to the additional preprocessing of the parallel algorithm.

In order to extend the evaluation, the speedup which is defined as the ratio between execution time on one processor and parallel execution time was used. There are three kinds of speedups shown in Fig. 12, the ideal speedup, total time speedup and the compute time (total time minus the time for disk I/O) speedup. It is obvious that data volume has a clear influence on speedups. By using larger dataset, higher compute time speedup could be achieved especially between 2 and 16 processes; meanwhile the gaps between the compute time speedup and total time speedup become

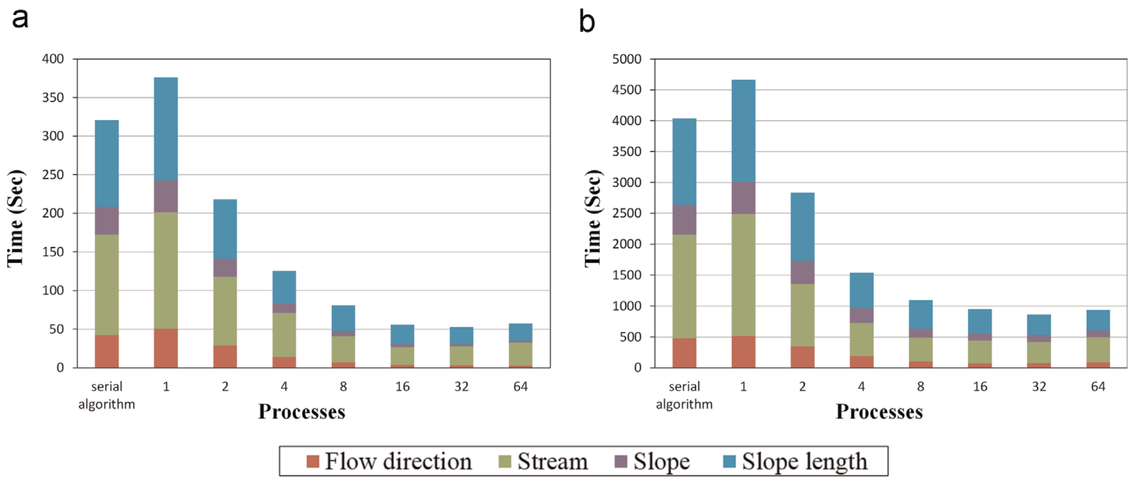

Fig. 11. Time taken to parallel algorithms for *LS* calculation with different DEM datasets: (a) 90-m dataset (b) 30-m dataset.

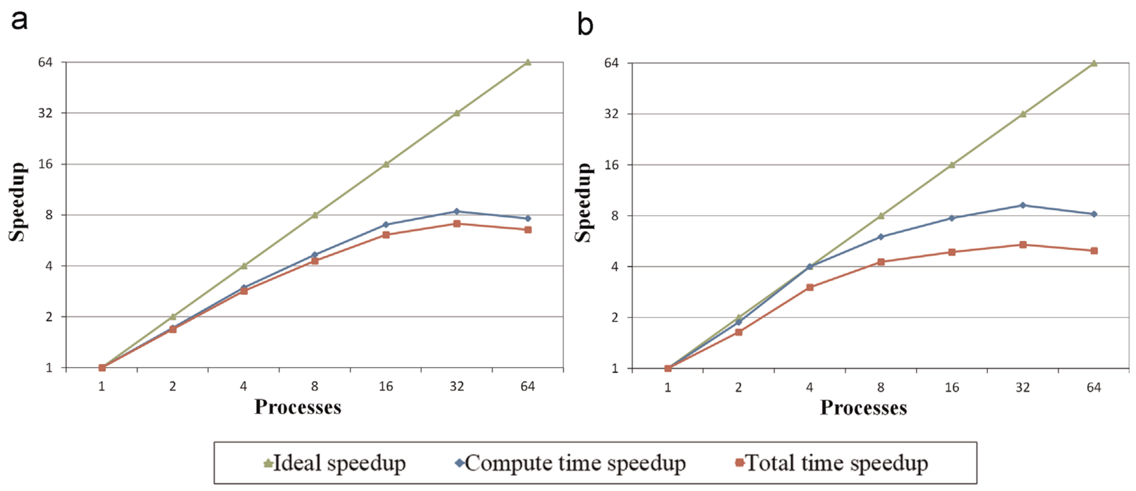

Fig. 12. A comparison of compute time speedup and total time speedup of parallel algorithms for *LS* calculation with different DEM datasets: (a) 90-m dataset (b) 30-m dataset.

wider. This is because the time taken for computation increases with the data volume, leading a better compute time speedup; however the overhead for disk I/O also increases by using larger dataset, which has a bad influence on the total time speedup. It is also noted that the advantage of larger dataset in compute time speedup becomes smaller as the processes getting larger, it is because that the communication gets more frequent for larger dataset under such condition.

Because of the differences in serial algorithms and the parallel strategy, there are clearly variations in total time speedups of different algorithms. As shown in Fig. 13, the speedups of local algorithms are obvious higher than that of global algorithms by using the smaller dataset. However, the differences between local and global algorithms get narrower by using larger dataset. The phenomenon is caused by the increasing time taken for the I/O disk, as a result the global algorithms which are more complex with longer computation time can achieve higher speedups. Affected by the communication in global algorithms, more processes increase the burden of communication between processes, which reduces the parallel efficiency. Fig. 14 shows the communication overhead ratios of the stream algorithm and the *LS* factor algorithm. From 2 to 16 processes, the communication ratios, defined as the ratio between time taken for communication and total execution time, increase significantly for both datasets and both parallel algorithms, but their growth rate slows down from 16 to 64 processes. The differences of communication overhead ratios can also reflect the influence of I/O. It is obvious that the *LS* factor algorithm owns less communication overhead ratios than the stream algorithm, because *LS* factor needs more input datasets which bring more burdens of I/O.

5. Conclusions and future work

With the improvement of DEM resolution and the extension of the research area, serial algorithms are insufficient, sometime even incapable to process the massive terrain datasets. In this paper, a parallel processing model is developed for calculating the *LS* factor which in turn includes the calculation of slope, flow direction, stream, and final *LS* factor. From the experimental results on the multi-node cluster the following findings are observed: (1) the parallel implementation of the *LS* factor calculation dramatically reduces the computation time, even with the datasets which completely overwhelms the serial recursive implementation;

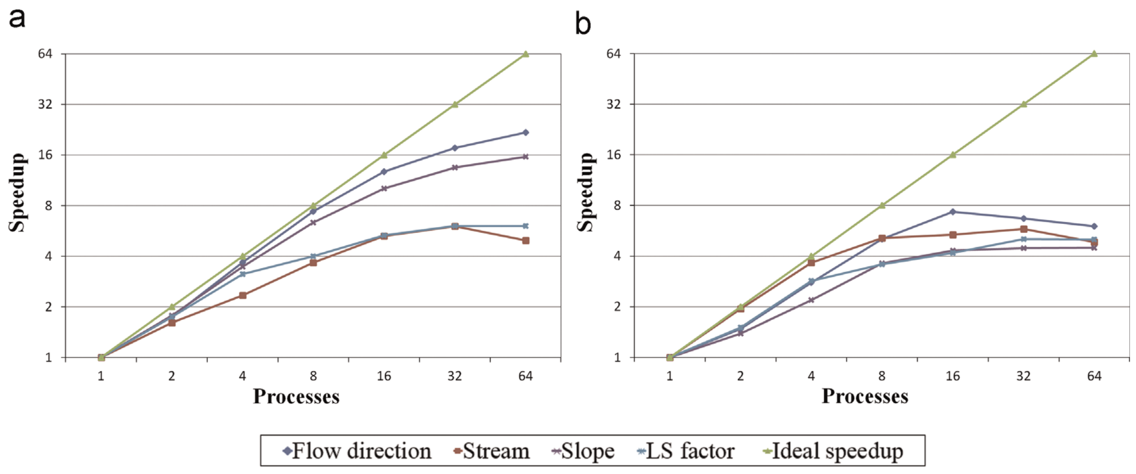

Fig. 13. A comparison of speedup of parallel algorithms for *LS* calculation with different DEM datasets: (a) 90-m dataset (b) 30-m dataset.

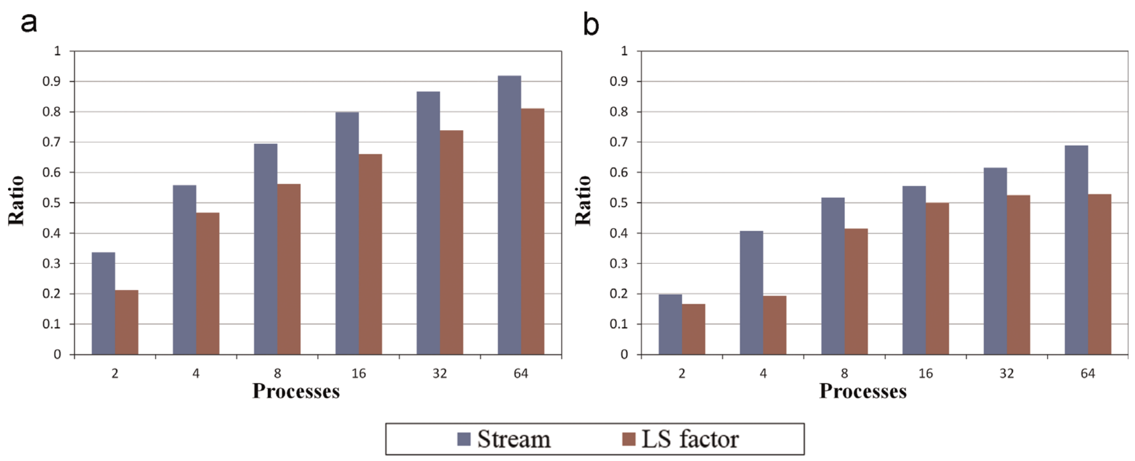

Fig. 14. A comparison of communication overhead ratios of parallel algorithms for *LS* calculation with different DEM datasets: (a) 90-m dataset (b) 30-m dataset.

(2) due to the influence of communication overhead, the parallel efficiency of global algorithms is inferior to that of local algorithms; and (3) the parallelized *LS* factor algorithm achieves a higher compute time speedup with the increase of data volume especially when the number of process is not very large. However the increase of the I/O overhead has a strong negative influence on the total time speedup.

Some shortcomings still exist in the new approach. One is the contradiction between the limited computing resources and the massive dataset. The parallel algorithm can make it possible to process the big data which cannot be calculated by using the traditional serial algorithms. However, if the assigned dataset for each processor is out of its available memory, especially when considering the cluster which may be used for some other computation tasks at the same time, the parallel algorithms may not process the massive dataset. The granularity control has been regarded as a helpful tool to solve this problem. An improved parallel algorithm with granularity control for the *LS* factor calculation could be implemented in the further study. Another problem focuses on the disk I/O time. The experiments show that the time taken for I/O increases rapidly with the increase of data volume. The I/O overhead ratio could even exceed that for computation by using a large number of processes, which makes a negative influence on the parallel efficiency. In further study, a shared file system with high parallel I/O throughput can be used to eliminate this bottleneck so that the parallel algorithms can be more efficient and practical.

Acknowledgments

This work was supported by the Major Scientific Research Projects of Universities in Jiangsu Province (13KJA170001), the Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions, the National Natural Science Foundation of China (Nos. 41471316, 41201464 and 41271438) and the Foundation of Graduate Innovation Plan of Jiangsu Province (KYLX_0701). We thank anonymous reviewers for their constructive comments on the earlier version of this paper.

Appendix A.1. Pseudo code of local algorithm parallelization. (inPutfile and outPutfile are the input DEM file and the output DEM file respectively, pSize is the process number.)

```
void LocalParallelization (inPutfile,outPutfile,pSize)
    SubDomain(pSize)
    if rank=1 then
        Createfile(outPutfile)
    end if
    inputData=Read(inPutfile,rank)
    outputData=Init(inputData)
    Calculation(outputData)
    WriteData(outPutfile)
    Freedata()
```

Appendix A.2. Pseudo code of parallelization strategy for global algorithms. (inPutfile and outPutfile are the input DEM file and the output DEM file respectively, pSize is the process number. Function *RingTermination( )* serves to transfer signals among all the processes, the final signal becomes false if the signal of any process is false. Function *SwapBuffers( )* denotes a function for swapping the buffers with neighboring processes.)

```
1. void GlobalParallelization(inPutfile,outPutfile,pSize)
2.    SubDomain(pSize)
3.    if rank=1 then
4.       Createfile(outPutfile)
5.    end if
6.    inputData=Read(inPutfile,rank)
7.    outputData=Init(inputData)
8.    for each cell m of inputData do
9.       if m is sourcePoint then
10.         Calculation(outputData,m)
11.      end if
12.   end for
13.   allFinished=false
14.   While (!allFinished)
15.      allFinished=true
16.      for each cell m in both topRow and bottomRow do
17.         if m is not calculated then
18.            allFinished=false
19.         end if
20.      end for
21.      allFinished=RingTermination(allFinished)
22.      if allFinished then
23.         Goto line 36
24.      end if
25.      SwapBuffers( )
26.      for each c in both topBuffer and bottomBuffer do
27.         if Buffer(c) is calculated and has not been used for
   updating then
28.            sourcePoints.push(c)
29.         end if
30.      end for
31.      while (!sourcePoints.empty())
32.         bufferCell=sourcePoints.pop()
33.         Calculation(outputData,bufferCell)
34.      end while
35.   end while
36.   WriteData(outPutfile)
37.   Freedata()
```

References

Desmet, P.J.J., Govers, G., 1996. A GIS procedure for the automated calculation of the USLE LS factor on topographically complex landscape units. J. Soil Water Conserv. 51 (5), 427–433.

Fu, B.J., Zhao, W.W., Chen, L.D., Zhang, Q.J., Lv, Y.H., Gulinck, H., Poesen, J., 2005. Assessment of soil erosion at large watershed scale using RUSLE and GIS: a case study in the Loess Plateau of China. Land Degrad. Dev. 16 (1), 73–85.

Gong, J., Xie, J., 2009. Extraction of drainage networks from large terrain datasets using high throughout computing. Comput. Geosci. 35 (2), 337–346.

Cheng, G., Liu, L., Jing, N., Chen, L., Xiong, W., 2012. General-purpose optimization methods for parallelization of digital terrain analysis based on cellular automata. Comput. Geosci. 45, 57–67.

Cheng, G., Jing, N., Chen, N., 2013. A theoretical approach to domain decomposition for parallelization of Digital Terrain Analysis. Ann. GIS 19 (1), 45–52.

Cheng, L., Yang, Q.K., Xie, H.X., Wang, C.M., Guo, W.L., 2009. GIS and CSLE based quantitative assessment of soil erosion in Shaanxi, China. J. Soil Water Conserv. 23 (5), 61–66 (in Chinese).

Guan, Q.F., Clarke, K.C., 2010. A general-purpose parallel raster processing programming library test application using a geographic cellular automata model. Int. J. Geogr. Inf. Sci. 24 (5), 695–722.

Hickey, R., 2000. Slope angle and slope length solutions for GIS. Cartography 29, 1–8.

Hickey, R., Smith, A., Jankowski, P., 1994. Slope length calculations from a DEM within Arc/Info GRID. Comput. Environ. Urban Syst. 18 (5), 365–380.

Jiang, L., Tang, G.A., Liu, X.J., Song, X.D., Yang, J.Y., Liu, K., 2013. Parallel contributing area calculation with granularity control on massive grid terrain datasets. Comput. Geosci. 60, 70–80.

Kidner, D.B., Railings, P.J., Ware, J.A., 1997. Parallel processing for terrain analysis in GIS: visibility as a case study. Geoinformatica 1 (2), 183–207.

Kinnell, P.I.A., 2010. Event soil loss, runoff and the Universal Soil Loss Equation family of models: a review. J. Hydrol. 385, 384–397.

Liu, B.Y., Nearing, M.A., Shi, P.J., Jia, Z.W., 2000. Slope length effects on soil loss for steep slopes. Soil Sci. Soc. Am., 64; , pp. 1759–1763.

Liu, J.Z., Zhu, A.X., Liu, Y.B., 2013. A layered approach to parallel computing for spatially distributed hydrological modeling. Environ. Model. Softw. 51, 221–227.
Lu, H., Prosser, I.P., Moran, C.J., Gallant, J.C., Priestley, G., Stevenson, J.G., 2003. Predicting sheet wash and rill erosion over the Australian continent. Aust. J. Soil Res. 41 (6), 1037–1062.
McCool, D.K., Foster, G.R., Mutchler, C.K., Meyer, L.D., 1989. Revised slope length factor for the universal soil loss equation. Trans. Am. Soc. Agric. Eng. 32, 1571–1576.
Mitasova, H., 1996. Modelling topographic potential for erosion and deposition using GIS. Int. J. Geogr. Inf. Sci. 10 (5), 629–641.
Moore, I., Burch, G., 1986. Physical basis of the length–slope factor in the Universal Soil Loss Equation. Soil Soc. Am. J. 50, 1294–1298.
Moore, I.D., Wilson, J.P., 1992. Length–slope factors for the revised universal soil loss equation: simplified method of estimation. J. Soil Water Conserv. 47 (5), 423–428.
Qin, C.Z., Zhan, L.J., 2012. Parallelizing flow-accumulation calculations on graphics processing units – from iterative DEM preprocessing algorithm to recursive multiple-flow-direction algorithm. Comput. Geosci. 43, 7–16.
Qin, C.Z., Zhan, L.J., Zhu, A.X., Zhou, C.H., 2014. A strategy for raster-based geo-computation under different parallel computing platforms. Int. J. Geogr. Inf. Sci. 28 (11), 2127–2144.
Renard, K.G., Foster, G.R., Weesies, G.A., McCool, D.K., Yoder, D.C., 1997. Predicting soil erosion by water: a guide to conservation planning with the Revised Universal Soil Loss Equation (RUSLE), Agriculture Handbook.
Song, X.D., Tang, G.A., Li, F.Y., Jiang, L., Zhou, Y., Qian, K.J., 2013. Extraction of loess shoulder-line based on the parallel GVF snake model in the loess hilly area of China. Comput. Geosci. 52, 11–20.
Tesfa, T.K., Tarboton, D.G., Waston, D.W., Schreuders, K.A.T., Baker, M.E., Wallace, R. M., 2011. Extraction of hydrological proximity measures from DEMs using parallel processing. Environ. Model. Softw. 26 (12), 1696–1709.
Van Remortel, R.D., Hamilton, M.E., Hickey, R.J., 2001. Estimating the LS factor for RUSLE through iterative slope length processing of digital elevation data. Cartography 30, 27–35.
Van Remortel, R.D., Maichle, R.W., Hickey, R.J., 2004. Computing the LS factor for the Revised Universal Soil Loss Equation through array-based slope processing of digital elevation data using a Cpp executable. Comput. Geosci. 30, 1043–1053.
Wallis, C., Watson, D., Tarboton, D., Wallace, R., 2009. Parallel flow-direction and contributing are a calculation for hydrology analysis in digital elevation models. In: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. Las Vegas, Nevada, pp. 467–472.
Wang, F., Wang, G., Pan, D.J., Liu, Y., Yang, L.Z., Wang, H.B., 2015. A parallel algorithm for viewshed analysis in three-dimensional Digital Earth. Comput. Geosci. 75, 57–65.
Winchell, M.F., Jackson, S.H., Wadley, A.M., Srinivasan, R., 2008. Extension and validation of a geographic information system-based method for calculating the revised universal soil loss equation length–slope factor for erosion risk assessments in large watersheds. J. Soil Water Conserv. 63, 105–111.
Wischmeier, W.H., Smith, D.D., 1978. Predicting Rainfall Eosion losses: A Guide to Conservation Planning with Universal Soil Loss Equation (USLE) Agriculture Handbook. Department of Agriculture, Washington, DC, p. 703.
Wu, Y., Li, T., Sun, L., Chen, J., 2013. Parallelization of a hydrological model using the message passing interface. Environ. Model. Softw. 43, 124–132.
Xie, J.B., 2012. Implementation and performance optimization of a parallel contour line generation algorithm. Comput. Geosci. 49, 21–28.
Yang, D., Kanae, S., Oki, T., Koike, T., Musiake, K., 2003. Global potential soil erosion with reference to land use and climate changes. Hydrol. Process. 17 (14), 2913–2928.
Zhang, H.M., Yang, Q.K., Li, R., Liu, Q.R., Moore, D., He, P., Ritsema, C.J., Geissen, V., 2013. Extension of a GIS procedure for calculating the RUSLE equation LS factor. Comput. Geosci. 52, 177–188.