Environmental Modelling and Software 144 (2021) 105142

Contents lists available at ScienceDirect

Environmental Modelling and Software

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

An automatic partition-based parallel algorithm for grid-based distributed hydrological models

Zhenwu Xu, Guoping Tang[*], Tao Jiang[**], Xiaohua Chen, Tao Chen, Xiangyu Niu

Department of Physical Geography, Resources and Environment, School of Geography and Planning, Sun Yat-Sen University, Guangzhou, 510275, China

ARTICLE INFO

Keywords:
Parallelization
Grid-based distributed models
Automatic partition
Speedup ratio
Computing efficiency
Watershed modeling

ABSTRACT

Parallel computing is a primary way to increase computing efficiency of grid-based distributed hydrological models. This study proposed an automatic partition-based parallel algorithm (APPA) to approach the theoretical maximum speedup ratio (TMSR). Through a combination of flexible partition for the domain decomposition and the load balance of parallel simulation, APPA optimizes the parallelization of hillslope and channel flow routing processes at sub-basin and channel unit level, respectively. To illustrate APPA's performance, we embedded it in a distributed ecohydrological model, and then applied the updated model to three watersheds at different spatial scales. The results indicate that APPA effectively promoted parallel performance. The estimated speedup ratio approached 93–97% of the TMSR for simulating hillslope processes and 91–98% of the TMSR for simulating channel processes using 26-threads in all three watersheds. These improvements justify that APPA is effective in accelerating model simulation and thus benefits future model-based research.

Software availability

Software: APPA (Automatic Partition-based Parallel Algorithm)
Language: C++
Availability: APPA is open-sourced on Github (https://github.com/xuzhenwu/APPA).

1. Introduction

Using distributed models to simulate coupled hydro-ecological processes at large spatial and long-term temporal scales can be cumbersome and time-consuming. With the rapid development of computer science, the parallelization of model simulation becomes an effective strategy for speeding up the computing efficiency (compared to the serial computing). One of the most important approaches for parallel computing is to partition a heavy computation task into several independent loads so that the subtasks can be allocated to multiple processors simultaneously. For watershed modeling, parallel simulation often divides the study area into several independent sub-basins. Thus, each of sub-basins is treated as a single task and processed by an individual thread under the assumption that there are no, or relatively low, communications between sub-basins (Vivoni et al., 2011). However, the task-partition-based parallel computing faces difficulties in finding the best partition solution, largely due to the spatial connectedness of coupled hydro-ecological process (e.g., water and solutes routing) in distributed models.

In spite of aforementioned disadvantages, the parallel algorithm by partitioning a watershed into several sub-basins has proved to be effective in speeding up model simulations (Tian et al., 2008; Wang et al., 2011; Qin and Zhan, 2012). The partitioned-based parallel algorithm usually involves spatial domain decomposition, computational task allocation, and inter-process communication (Zhang and Zhou, 2019). The spatial domain decomposition is to partition a large domain into smaller individual sub-basins that are arranged from upstream to downstream (Apostolopoulos and Georgakakos, 1997; Arnold et al., 1998; Li et al., 2011). In addition to decomposition at sub-basin level, Liu et al. (2014) found that the distributed models can be parallelized at basic simulation-unit levels through a layered approach. Consequently, the computational tasks of these parallelized units are allocated for parallel computing. However, the dependency resulting from flow routing still coexists with concurrency. This can be either resolved by

[*] Corresponding author. Department of Physical Geography, Resources and Environment, School of Geography and Planning, Sun Yat-Sen University, Guangzhou, 510275, China.
[**] Corresponding author. Department of Physical Geography, Resources and Environment, School of Geography and Planning, Sun Yat-Sen University, Guangzhou, 510275, China.
E-mail addresses: tanggp3@mail.sysu.edu.cn (G. Tang), eesjt@mail.sysu.edu.cn (T. Jiang).

https://doi.org/10.1016/j.envsoft.2021.105142
Accepted 15 July 2021
Available online 20 July 2021
1364-8152/© 2021 Elsevier Ltd. All rights reserved.

simulation from upstream to downstream units at thread-level with shared-memory programing standard (Hwang et al., 2014; Liu et al., 2014) or by simulation of units at process-level with message-passing programming standard (Tian et al., 2008; Li et al., 2011; Vivoni et al., 2011; Wang et al., 2011). In practice, the former may be constrained due to the consideration of the task arrangement while the latter may suffer from poor efficiency caused by a heavy load of inter-process communications (Li et al., 2011; Vivoni et al., 2011). Even so, a joint of them can break through the speedup ratio limit compared with using either of them (Liu et al., 2016; Zhu et al., 2019). Nevertheless, the exploration of partition-based parallel algorithms has greatly enhanced the computing speed of distributed models. Apart from the effectiveness (i.e., speedup ratio), the existing parallel algorithms need improving since the computational efficiency (i.e., parallel efficiency) is still restricted to the unit partition at a task parallelism level (Zhang and Zhou, 2019).

To quantify the efficiency and guide the parallel algorithm development, the theoretical maximum speedup ratio (TMSR) has been proposed as a target to achieve the potential of parallel computing under a given number of processors (Wang et al., 2012). To approach TMSR in the flow routing systems of hydrological models, the critical path heuristic algorithm has proven to be effective in scheduling static tasks of partitioned sub-basins (Shirazi et al., 1990; Liu et al., 2013). Through analyzing basin width function, Wang et al. (2012) insisted that there is a maximum speedup curve for an arbitrary drainage network. In a preliminary application, they found that the more sub-basins are partitioned, the higher the TMSR is. In addition, the compact watersheds generally have higher TMSR than long and narrow watersheds. Liu et al. (2013) developed a method to estimate TMSR for parallel computing of hillslope and channel processes using grid-based distributed hydrological models. Efforts are continuously made to maximize the parallel efficiency. Some researchers noticed that the parallel performance is improved with the increase of the number of partitioned sub-basins as a result of better load balance (Wang et al., 2012; Liu et al., 2016). Besides, the load balance is proved to significantly improve the parallel speed-up with proportionally faster runs as simulation complexity (i.e., domain resolution and channel network extent) increases (Vivoni et al., 2011). These explorations help researchers recognize the potential of parallel computing under a given parallel algorithm.

Application of distributed models at large-spatial scales driven by high-spatial resolution data can involve a huge number of simulated grids. Compared with models built at coarser units (e.g., sub-basins, hillslope), grid-level simulation usually consists of more simulation units (grids) and thus has a more extended and complex flow network (hillslope flow routings and channel flow routings). The increasing complexity of simulation not only increases computing burden, but also may bring greater potential of parallelization. Although channel reaches are usually set as the basis for sub-basin unit decomposition in prior partition-based algorithms (Arnold et al., 1998; Li et al., 2011; Vivoni et al., 2011), the grid-based models have no such restricts and each grid can be set as the potential outlet for extracting a sub-basin. As partitioned units are further allocated to different processors, the computation efficiency is sensitive to the spatial domain decomposition (Li et al., 2011; Zhang and Zhou, 2019). However, previous studies usually consider the spatial domain partition as a preliminary process before allocating computation tasks for parallelization. It is worth of investigating how the parallel efficiency will respond to the flexibility of partition for grid-based distributed models. To maximize the parallel performance, we argue that an optimal partition-based parallel algorithm should have following features: (a) the utilization of the load balance strategy throughout the preparation of a parallel scheme; (b) the search of potential parallel schemes based on flexible spatial domain partition for the optimal efficiency; (c) an estimate of real performance compared with the theoretical maximum performance; and (d) the applicability to most grid-based distributed models.

To meet these criteria, this study proposed an automatic partition-based parallel algorithm (APPA) that aims to parallelize grid-based distributed models such that the theoretical maximum speedup ratio (TMSR) is approachable. By combining global search for potential partition schemes with the load balance strategy in parallel simulation, APPA optimizes parallelization of hillslope and channel flow routing processes at both sub-basin and channel unit levels, respectively. To illustrate APPA's performance, we embedded it into the Coupled Hydro-Ecological Simulation System (CHESS, Tang et al., 2014, 2019), and applied the updated CHESS to three watersheds using various threads. The subsequent computing performances of parallel simulations were evaluated using the TMSR, estimated speedup ratio and real speedup ratio. Overall, this study provides a new parallel algorithm that can maximize the computing efficiency of parallel simulation, thus contributing to future model-based research.

2. Developing the automatic partition-based parallel algorithm

2.1. Parallel assumptions of grid-based hydrological models

The automatic partition-based parallel algorithm (APPA) assumes that a typical grid-based distributed model has following characteristics:

(1) Grids are basic units for simulating hydrological processes (e.g., infiltration, evaporation, runoff).
(2) The overall hydrological processes can be classified as hillslope and channel processes in terms of flow equations (e.g., hillslope surface and sub-surface flow, and channel flow).
(3) Flow routing among grids is generated by a single flow routing algorithm (e.g., D8 algorithm) such that runoff yield in a grid is routed to only a downslope neighboring grid. In addition, directions of surface and sub-surface flow are uniform for each hillslope grid.
(4) Given the continuity and nature of downslope water movement, grids located in upslope or upstream areas take precedence during simulation. In practice, this is realized by arranging all grids in descending order according to their elevations.
(5) The runtime for a grid simulation is assumed to be equal, or is determined by its own type.

2.2. General parallelization scheme for the automatic partition-based algorithm

The proposed APPA applies a two-level partition scheme for parallel computing so that the computing performance of grid-based distributed models can be enhanced. First, grids are divided into "hillslope grids" and "channel grids", corresponding to hillslope processes (including surface and sub-surface flow) and channel processes (only containing channel flow). Second, the two processes are further partitioned as sub-basin units and channel units that are allocated to different threads for parallel computing. By doing so, the original serial simulation is divided into several rounds of parallel simulations. Within each round of parallel simulation, the partitioned units are assumed to be independent and allocated to different threads that do not communicate with each other.

Fig. 1 depicts the two-level parallel scheme and the corresponding parallel algorithm developed for a four-core computer. Daily simulation of overall processes is divided into simulation of hillslope and channel processes. For hillslope processes, the computation tasks of five partitioned sub-basins are eventually allocated to four threads (Fig. 1a). For channel processes, the parallel computing consists of five sequential layers. The first layer contains seven independent units, which are further allocated to four threads (Fig. 1b). Similarly, three middle layers are sequentially parallelized. The last layer of channel processes is computed by one thread, as it has no branches for partitioning. Based on unit partitions and thread allocations, the proposed algorithm adopts a fork/join structure to achieve parallel computing, and considers the load balance among processors to improve parallel performance (Fig. 1c). In addition, no communication costs exist among threads in each round of

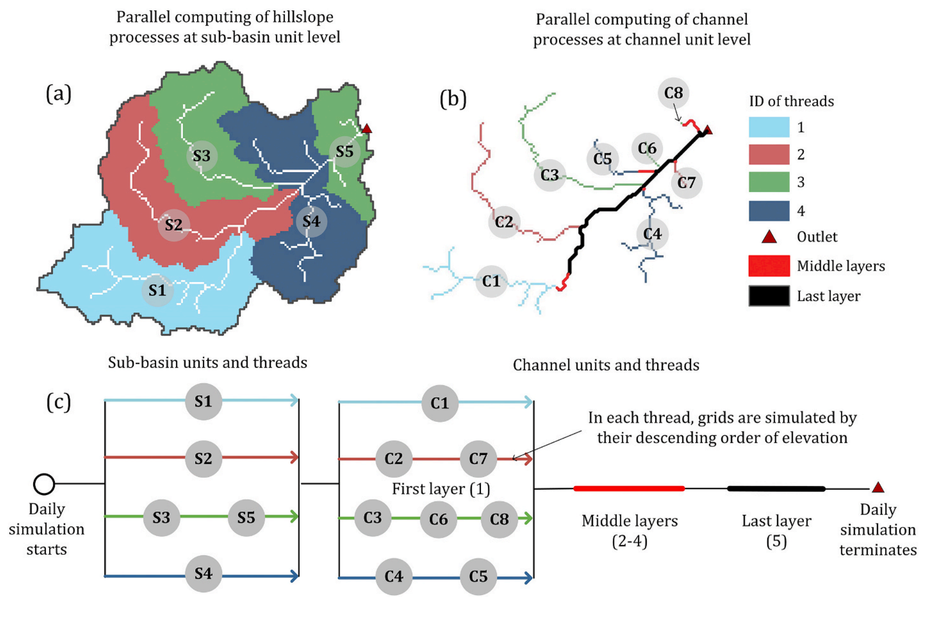
Parallel computing of hillslope processes at sub-basin unit level
(a)
S1
S2
S3
S4
S5
Parallel computing of channel processes at channel unit level
(b)
C1
C2
C3
C4
C5
C6
C7
C8
ID of threads
1
2
3
4
Outlet
Middle layers
Last layer
Sub-basin units and threads
(c)
Daily simulation starts
Channel units and threads
In each thread, grids are simulated by their descending order of elevation
First layer (1)
Middle layers (2-4)
Last layer (5)
Daily simulation terminates

Fig. 1. A parallel scheme developed for a four-core computer when applying a distributed model to a watershed. S1 to S5 are five independent sub-basin units partitioned for hillslope processes. C1 to C8 are eight independent channel units partitioned for the first layer of channel processes. These units are allocated to different threads as per the load balance strategy. The bold shiny red lines represent all grids of the sequential parallelized middle layers (layer 2–4). The bold black line represents the last layer that is processed by one thread (layer 5). The fork/join structure is adopted for each round of parallel computing.

parallel computing (e.g., hillslope processes, channel processes in first layer). Within each thread, grids are simulated in descending order of elevation to keep flow movement. Daily simulation of hydrological processes terminates when the outlet grid is complete.

2.3. *Principles of optimization for automatic partition-based parallelization*

To approach the optimal state of partition-based parallelization, APPA adopts the following principles:

(1) Searching the potential parallel scheme repeatedly to select the optimal scheme. As the selected model is grid-based, the size of partitioned units can vary due to many choices of the potential outlet grids. Each channel grid has the potential to be the reference point to trace all upstream grids that can be grouped into a sub-basin or channel unit. Given this, the accumulated runtimes for grid i ($T_{G,i}$, Figs. 2b and 5b) are used to estimate the required time for simulating all grids within a sub-basin or channel unit. When the runtimes for each grid are specified, the total T_G accumulates as the flow routes downslope to the outlet. In theory, if the runtimes for simulating each grid are identical, the number of grids is just a proxy of the accumulated runtimes (T_G). Due to the flexibility of selecting outlets for sub-basin or channel units, it is likely to obtain the best parallel scheme by comparing computing performances under different domain partitions.

(2) Utilizing the load balance strategy to composite the partitioned units as the final parallel scheme. For example, the partitioned sub-basin units are assigned to groups with possible balanced runtime so that the final runtime for hillslope processes is minimized (Fig. 1a).

(3) Measuring the parallel performance to approach theoretical performance. As the goal of partitioning is to fully utilize the computation power of computers with n available processors, this study used three kinds of speedup ratios (SR) to evaluate the parallel performance: the theoretical maximum speedup ratio (TMSR), the estimated speedup ratio (ESR), and the real speedup ratio (RSR). SR is defined as the ratio of the time cost of serial computing (T_S) to that of the parallel computing with n processors (T_n) (Hwang et al., 2014).

$$SR = T_S / T_n \tag{1}$$

The overall processes are divided into hillslope processes (hs) and channel processes (ch), and there are three types of SR for each of processes. TMSR indicates the full utilization of all allocated threads. As there are no communications for hillslope processes among sub-basins, the TMSR of hillslope processes equals the number of threads (n) when the load balance among threads is fully achieved ($\text{TMSR}_{hs} = n$). For channel processes, TMSR_{ch} is determined by the longest exit length, which represents the minimum possible execution time (Shirazi et al., 1990; Liu et al., 2013). It is computed as the ratio of the serial runtime spent for the longest channel exit length (T_{ls}) to the required runtime for serial computing of channel grids (T_{ch}):

$$\text{TMSR}_{ch} = \frac{T_{ch}}{T_{ls}} \tag{2}$$

Then, the overall TMSR is computed as follows:

$$\text{TMSR} = 1/(p_{hs} / \text{TMSR}_{hs} + p_{ch} / \text{TMSR}_{ch}) \tag{3}$$

where, p_{hs} and p_{ch} refer to the runtime proportion for hillslope and channel processes, respectively, relative to the overall simulation values ($p_{hs} + p_{ch} = 1$).

The estimated speedup ratio (ESR) is the estimate of speedup ratio for a given parallel scheme based on the load balance in the fork/join structure (Fig. 1c). To compute ESR, T_S is defined as the total runtimes of all partitioned grids, and T_n is defined as the thread that is allocated with the maximum runtime of partitioned grids (Eq. (1)). RSR is computed by the testing results of real runtimes of serial and parallel computing. In comparison, RSR is generally the smallest value since it accounts more costs (TMSR > ESR > RSR).

Parallel efficiency (EP) is used to quantify the efficiency in parallel computing and defined as the speedup ratio to the number of available processors:

$$EP = SR / n \tag{4}$$

2.4. *Parallel simulation of hillslope processes*

The simulation of hillslope processes is most time-consuming, as it involves the most of grids to be simulated. In practice, parallel computing of hillslope processes involves sub-basin partitions and thread allocation. As the partitioned sub-basin units are independent, the parallel computing of hillslope processes should be finished in one round of parallel computing with optimal thread allocation. In addition,

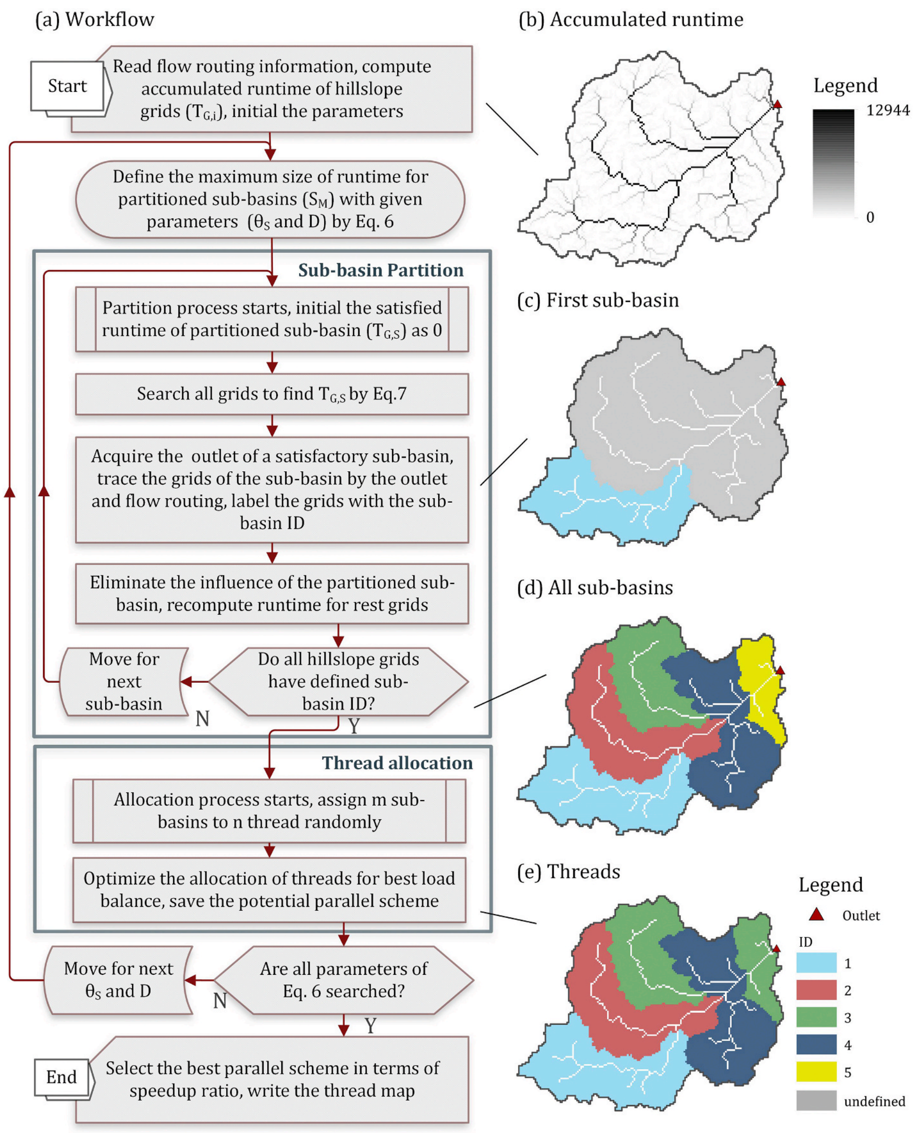
(a) Workflow
Start
Read flow routing information, compute accumulated runtime of hillslope grids ($T_{G,i}$), initial the parameters
Define the maximum size of runtime for partitioned sub-basins (S_M) with given parameters (θ_S and D) by Eq. 6
Sub-basin Partition
Partition process starts, initial the satisfied runtime of partitioned sub-basin ($T_{G,S}$) as 0
Search all grids to find $T_{G,S}$ by Eq.7
Acquire the outlet of a satisfactory sub-basin, trace the grids of the sub-basin by the outlet and flow routing, label the grids with the sub-basin ID
Eliminate the influence of the partitioned sub-basin, recompute runtime for rest grids
Do all hillslope grids have defined sub-basin ID?
Move for next sub-basin
N
Y
Thread allocation
Allocation process starts, assign m sub-basins to n thread randomly
Optimize the allocation of threads for best load balance, save the potential parallel scheme
Are all parameters of Eq. 6 searched?
Move for next θ_S and D
N
Y
End
Select the best parallel scheme in terms of speedup ratio, write the thread map
(b) Accumulated runtime
Legend
12944
0
(c) First sub-basin
(d) All sub-basins
(e) Threads
Legend
Outlet
ID
1
2
3
4
5
undefined

Fig. 2. Workflow of building parallel schemes for simulating hillslope processes. Data shown here are based on an application in the Yuecheng watershed in southern China using a four-core computer, which contains 12,944 hillslope grids.

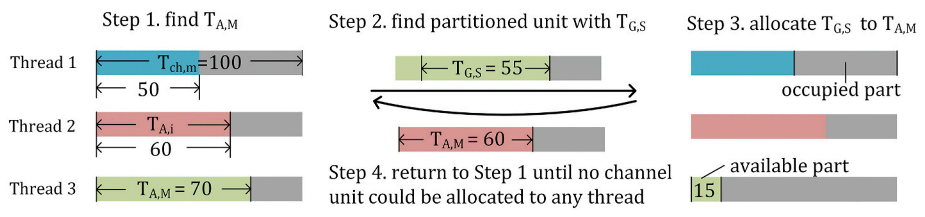
Step 1. find $T_{A,M}$
Step 2. find partitioned unit with $T_{G,S}$
Step 3. allocate $T_{G,S}$ to $T_{A,M}$
Thread 1
$T_{ch,m}$=100
50
Thread 2
$T_{A,i}$
60
Thread 3
$T_{A,M}$ = 70
$T_{G,S}$ = 55
$T_{A,M}$ = 60
occupied part
available part
15
Step 4. return to Step 1 until no channel unit could be allocated to any thread

Fig. 3. Illustration of variables used in the APPA for best load balance in parallelization of channel processes.

an extra scheme selection process is provided for obtaining optimal parallel performance. The three steps are described in the following paragraphs.

The sub-basin partition starts with the definition of partition goals. First, we define the total computation time needed for hillslope processes as the required time for simulating all hillslope grids (T_{hs}). Thus, the theoretical minimum runtime for all n threads ($T_{hs,n}$) is expressed as:

$$T_{hs,n} = T_{hs}/n \tag{5}$$

Specially, the maximum computation time demanded for partitioned sub-basins (S_M) is determined by $T_{hs,n}$:

$$S_M = \frac{T_{hs,n}}{D}(1 + \theta_S) \tag{6}$$

where, the deviation index (θ_S) and decomposition index (D) indicate the variation of S_M; θ_S represents minor variations of S_M, which will not significantly increase the number of final partitioned sub-basins (m); and D represents significant changes in S_M. For example, m approximates twice of the value of n when D is 2.

In practice, a potential outlet grid with satisfactory accumulated runtime ($T_{G,S}$) will be searched from all channel grids. This potential channel grid assures that the subsequently partitioned sub-basins are independent in terms of hillslope processes, and it will retain a closest value to, but lower than S_M.

$$T_{G,S} = \begin{cases} T_{G,i} & when\ 0 \le S_M - T_{G,i} < S_M - T_{G,S} \\ T_{G,S} & others \end{cases} \tag{7}$$

where $T_{G,S}$ is initialized as zero before the searching process. The value of $T_{G,S}$ may be updated as the algorithm searches through all channel grids.

When a grid is captured ($T_{G,S} > 0$), it will be set as the reference outlet to trace the corresponding sub-basin (Fig. 2c). Once a sub-basin is obtained, the accumulated runtime for all grids (T_G) will be recomputed, and then the program starts to search for next sub-basin until all hillslope grids are grouped into the partitioned sub-basins (Fig. 2d).

The thread allocation process allocates the computation tasks of independent sub-basins to available threads in one loop of parallel computing (Fig. 1c). To optimize the allocation for obtaining the best load balance, APPA tries to reduce the maximum runtime among all threads. First, the thread with maximum runtime and the thread with minimum runtime are chosen, respectively. Then, the smallest sub-basin in the thread with maximum runtime is moved to the minimum thread. If it successfully reduces the maximum runtime among all threads that limits the parallel performance, the redistribution process continues. Otherwise, the optimal state of allocation is achieved. For example, the first-to-fifth sub-basins in Fig. 1a are assigned to four threads in order, i.e., sub-basin 5 is assigned to thread 1. As thread 1 is the thread with the maximum runtime, the smallest sub-basin in thread 1 (sub-basin 5) is reassigned to thread 3 with the minimum runtime. It retained as the optimal result of allocation where sub-basin 5 and sub-basin 3 are processed by the same thread (Fig. 1c). After all hillslope grids of sub-basins are allocated to specific threads, the applicable parallel scheme for hillslope processes is completed.

The strategy for the outer selection process is to choose the optimal scheme in terms of ESR (Fig. 2a). Since the maximum runtime of sub-basins (S_M) is subject to changes in the deviation index (θ_S) and decomposition index (D), the proposed algorithm examines all potential parallel schemes. Empirically, θ_S has a value ranging from 0 to 0.4 and iterates by a step of 0.05. The decomposition index (D) is designed to increase the number of sub-basins several times. As D iterates from 1 to a larger value, it will reveal how the variation of domain decomposition affects parallel performance. Then, the best parallel scheme is selected from all schemes based on the ESRs.

2.5. Parallel simulation of channel processes

Channel routing processes significantly affect the computing performance of model simulation. Although channel units are interconnected units with binary structures, they can still be considered independent when all channel units are treated in a same “layer” (Liu et al., 2014). As a result, the upstream-downstream computation dependence only imposes restrictions on the outer and inner layers where the channel units in outer layers take precedence over those in inner layers during simulation. Thus, the aforementioned partition and allocation processes are applicable in parallelizing channel processes into several layers. However, the more layers there are, the more rounds of parallel computing are needed. Because APPA adopts flexible partition strategy for grid-based models, the partitioned channel units do not necessarily follow the locations of stream reaches. In general, similar to hillslope processes, the parallel computing of channel processes involves channel unit partition and thread allocation. In addition, an extra scheme selection process is provided for obtaining the optimal parallel performance in simulating channel processes.

The channel unit partition and thread allocation are synchronous processes. As with simulating hillslope processes, we define a maximum runtime of threads for channel processes ($T_{ch,m}$ in Fig. 3), and thus the strategy of partition is to fill up $T_{ch,m}$, which is assumed to be the same in all threads for load balance. Specially, we defined the available runtime of thread i ($T_{A,i}$) to limit the size of rest of the runtimes of thread i that can be allocated for a potential channel unit ($T_{G,S}$). Like setting the priority for larger sub-basin units in hillslope parallel computing, the thread with the largest available runtime ($T_{A,M}$) should be filled up first with a new channel unit (Step 1 in Fig. 3):

$$T_{A,M} = \max\left(T_{A,1}, \ldots, T_{A,n}\right) \tag{8}$$

As with the available runtime of a sub-basin (S_M, Eq. (6)), $T_{A,M}$ limits the size of accumulated runtimes of the partitioned channel unit ($T_{G,i}$) for channel processes. The final outlet should meet the requirement that $T_{G,S}$ has a slightly lower value than $T_{A,M}$ (step 2 in Fig. 3):

$$T_{G,S} = \begin{cases} T_{G,i} & when\ 0 \le T_{A,M} - T_{G,i} < T_{A,M} - T_{G,S} \\ T_{G,S} & others \end{cases} \tag{9}$$

where $T_{G,S}$ is initialized as zero before the searching process. The value of $T_{G,S}$ may be updated as the algorithm searches through all unpartitioned channel grids.

If $T_{G,S}$ is greater than zero, the corresponding grid is set as the reference outlet to trace the channel unit (Fig. 5c), which is further allocated to $T_{A,M}$ (Step 3 in Fig. 3). To eliminate the influence of this unit, all the other channel grids located downstream will be excluded from the subsequent searching of $T_{G,S}$ to avoid upstream-downstream connection in this layer. The accumulated runtimes of other grids $(T_{G,i})$ will be re-computed. Specifically, the available runtime of the maximum thread will be updated using the following equation:

$$T_{A,M} = T_{A,M} - T_{G,S} \tag{10}$$

If $T_{G,S}$ remains zero, the computing for this layer is finished and the unit partitioning and the thread allocation process is terminated.

After several rounds of parallel simulation, the channel processes are finished and the maximum runtime for each thread ($T_{ch,m}$) in the layer is determined. Then, the runtime of all threads ($T_{O,V}$) in each layer is defined as:

$$T_{O,V} = \sum T_{O,i} \tag{11}$$

where $T_{O,i}$ is the total runtime of thread i and $T_{ch,m} = T_{O,i} + T_{A,i}$. To search all the possibilities of parallel schemes in the layer, $T_{ch,m}$ is set as the smallest value in the first iteration, i.e., the runtime for a channel grid ($T_{ch,m} = T_{CG}$, Fig. 4b), and kept updating as θ_G increases:

$$T_{ch,m} = \max\left(T_{ch,m}(1+\theta_G),\ T_{ch,m} + T_{CG}\right) \tag{12}$$

where max is the maximum function that returns the greatest value. Once a new $T_{ch,m}$ is obtained, the program records the estimated runtime of all threads ($T_{O,V}$) and the estimated speed up ratio (ESR_{ch}). This process will be terminated once $T_{O,V}$ has covered all channel grids with undefined layers and threads (Fig. 4d). ESR_{ch} is the serial time ($T_{O,V}$) divided by the maximum runtime in a thread in parallel computing:

$$ESR_{ch} = T_{O,V} / \max\left(T_{O,i}\right) \tag{13}$$

Fig. 4a shows how ESR_{ch} and $T_{O,V}$ vary as the key variable $T_{ch,m}$ increases using a four-core computer. As $T_{ch,m}$ increases from 1 to 113, ESR_{ch} remains having the maximum value of 4 and $T_{O,V}$ keeps increasing (Fig. 4a). However, the applicability of parallelization drops when $T_{ch,m}$ exceeds 113 and ESR_{ch} drops. When $T_{ch,m}$ exceeds the maximum accumulated runtime among all channel grids, they will be allocated to the same thread and ESR_{ch} is 1 (Fig. 4d). As the speedup ratio is the top priority of parallelization, the schemes with the largest ESR_{ch} are picked for next-round comparisons ($T_{ch,m}$ ranges from 1 to 113, Fig. 4a). Then, the scheme with the highest $T_{O,V}$ is selected (Fig. 4c). This selection process can avoid involving too many layers and thus reduce time spent

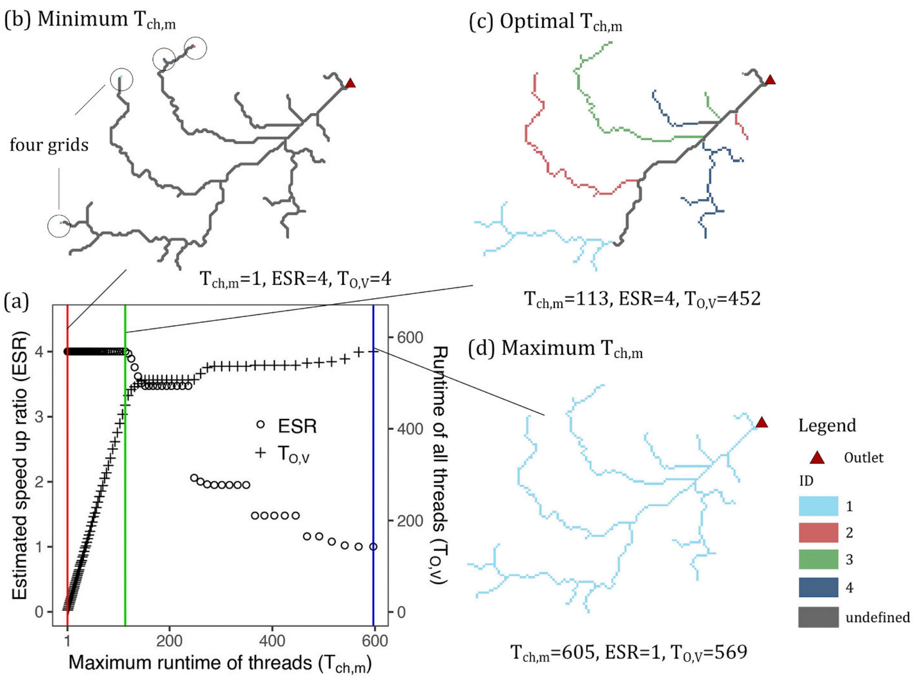

Fig. 4. Selection of the optimal parallel scheme based on estimated speedup ratio (ESR_{ch}) and the runtime of all threads ($T_{O,V}$). Data shown here are based on the first layer of parallel computing applied in the Yuecheng watershed using a four-core computer.

in allocating threads.

Similar steps in scheme generation processes are performed until all channel grids are allocated to a specific layer and thread (Fig. 5a). As the partition processes move to the outlet, the applicability of parallelization decreases and the last layer with a channel unit is processed using one thread (the black bold line in Fig. 1b). Empirically, the only tunable parameter θ_G is set as 0.05.

According to the characteristics of layers, several rounds of parallel computing can accelerate the simulation of channel processes. Model simulation for a given time scale (e.g., day) will terminate once the simulation for the outlet of the watershed was complete (Fig. 1c).

3. Application and testing of APPA: A case study

The Coupled Hydro-Ecological Simulation System (CHESS, Tang et al., 2014, 2019), a grid-based distributed model, was applied to test the proposed APPA. CHESS simulates integrated water, carbon, nutrient dynamics as well as vegetation growth in terrestrial ecosystems at watershed and regional scales. In CHESS, the routing of water and solutes among grids is explicitly simulated. When the model is applied at high spatial resolution and large spatial scales, the implementation of parallel computing becomes crucial for improving the computation efficiency.

In this study, the parallelization of CHESS was implemented through the Open Multi-Processing (OpenMP) library by C/C++ compilation (Fig. A1). First, CHESS reads the thread information and lists all grids of each thread in a descending order of grid elevations. In each loop of parallel computing, the threads compute corresponding grids whose addresses are stored in a specific array. To perform the fork/join structure, the main thread can continue until all parallelized threads are completed. Model simulation for a given time scale (e.g., day) will terminate once the parallel simulation of all grids is completed.

Three watersheds located in southern China were selected for evaluating the performance of proposed parallel algorithms (Fig. 6). They differ in total number of grids and areal coverage (Table 1). The two smaller watersheds are actually sub-watersheds of the biggest, and named corresponding to hydrological stations (Yuecheng and Longchuan). They represent ten-thousand-grid and hundred-thousand-grid level simulations, respectively. In contrast, the largest watershed, Dongjiang, consists of almost a million of grids in 200-m resolution, and represents another level of simulation.

One super-computer, which is equipped with two-way, 14-core, Intel Xeon E5-2683 2.0 GHz CPUs, and supports the parallel computing by a maximum of 28 processors/threads, was used to analyze parallel efficiency of the proposed algorithm. The parallel simulations are performed using different numbers of processors and the resultant parallel performances, based on different numbers of threads that vary from 2 to 32 (by an interval of 2), are compared with the original serial simulation to analyze the computing performance of the proposed algorithm.

4. Results

4.1. Optimizing the parallel computing of hillslope processes

Since sub-basins are basic units in the parallel computing of hillslope processes, the partition of sub-basins significantly affects the performance of parallel computing. To find the best parallel scheme, APPA uses the deviation index (θ_S) and the decomposition index (D) to adjust the size of estimated runtime for partitioned sub-basins (S_M). Fig. 7 shows the variation of estimated speedup ratio (ESR) and the relative increments of ESR under different parallel schemes as the number of involved threads increases. In the reference parallel scheme series ($\theta_s = 0, D = 1$), ESR increases linearly as more threads are involved (black line, Fig. 7a). When change the size of S_M, θ_s varies from 0 to 0.4 by a step length of 0.05, and the optimal result is selected from 9 parallel scheme series (blue lines). Based on optimal results, the maximum ESR (green line) increases by about 20% compared with the reference series when the number of threads ranges from 8 to 32, which approximates half the space between the TMSR (red line) and the reference ESR (black line, Fig. 7b).

The number of partitioned sub-basins also poses a great effect on the speedup ratio of parallel computing. For comparison, the reference parallel scheme series (black line in Fig. 7c) are set as optimal results generated by searching different θ_s, under the condition that the decomposition index (D) equals 1 and the number of sub-basins approximates the number of threads (green line in Fig. 7a). To change the

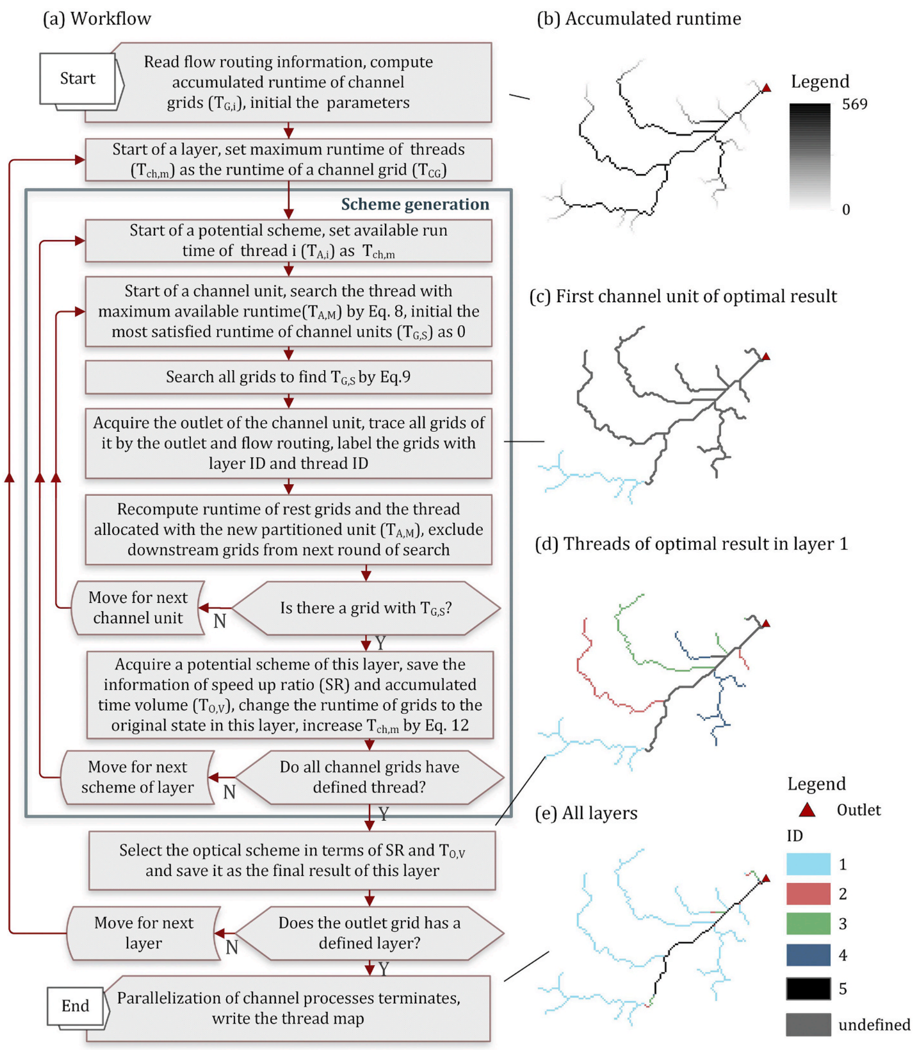
(a) Workflow
Start
Read flow routing information, compute accumulated runtime of channel grids ($T_{G,i}$), initial the parameters
Start of a layer, set maximum runtime of threads ($T_{ch,m}$) as the runtime of a channel grid (T_{CG})
Scheme generation
Start of a potential scheme, set available run time of thread i ($T_{A,i}$) as $T_{ch,m}$
Start of a channel unit, search the thread with maximum available runtime($T_{A,M}$) by Eq. 8, initial the most satisfied runtime of channel units ($T_{G,S}$) as 0
Search all grids to find $T_{G,S}$ by Eq.9
Acquire the outlet of the channel unit, trace all grids of it by the outlet and flow routing, label the grids with layer ID and thread ID
Recompute runtime of rest grids and the thread allocated with the new partitioned unit ($T_{A,M}$), exclude downstream grids from next round of search
Move for next channel unit
N
Is there a grid with $T_{G,S}$?
Y
Acquire a potential scheme of this layer, save the information of speed up ratio (SR) and accumulated time volume ($T_{O,V}$), change the runtime of grids to the original state in this layer, increase $T_{ch,m}$ by Eq. 12
Move for next scheme of layer
N
Do all channel grids have defined thread?
Y
Select the optical scheme in terms of SR and $T_{O,V}$ and save it as the final result of this layer
Move for next layer
N
Does the outlet grid has a defined layer?
Y
End
Parallelization of channel processes terminates, write the thread map
(b) Accumulated runtime
Legend
569
0
(c) First channel unit of optimal result
(d) Threads of optimal result in layer 1
(e) All layers
Legend
Outlet
ID
1
2
3
4
5
undefined

Fig. 5. Workflow of building the parallel scheme for simulating channel processes. Data shown here are based on application in the Yuecheng watershed in southern China using a four-core computer. The watershed contains 569 channel grids.

number of partitioned sub-basins, we have D vary from 1 to 8 by a step of 1 and the optimal results are selected from 8 parallel scheme series (green lines, Fig. 7c). Fig. 7b illustrates how *D* promotes the maximum ESR by searching potential schemes under different *D*. Fig. 7d indicates that the increasing partition of sub-basins can increase ESR when the number of threads exceeds 8; however, it does not necessarily increase ESR when the number of threads ranges from 2 to 6. In general, the maximum ESR increases by 4–17% compared with the reference ESR. By searching different parallel schemes with different θ_s and *D*, APPA achieves the potential of parallel computing while the maximum ESR is raised up to 97–99% of TMSR in Dongjiang watershed (Fig. 7c).

4.2. Optimizing the parallel computing of channel processes

APPA aims to increase the speedup ratio and reduce rounds of parallel computing in the implementation of the layered approach for simulating channel processes. In the case of Dongjiang watershed, the parallel simulation based on the proposed algorithm generates 18 layers for parallel simulation. In contrast, the original layered approach generates more than 2898 layers for the channel processes (grids in the critical exit length) if it simply allocates a grid to a single thread (Liu et al., 2014). Although ESR is the top priority in each layer of the proposed algorithm, the potential of parallelization decreases as more channel units are partitioned (Fig. 8a). As a result, the ESR for inner layers drops significantly. For the last channel unit (that has no branches), it cannot be parallelized and ESR is 1 for all three watersheds. In general, ESR increases as the number of involved threads increases (Fig. 8b). However, the increase in ESR has a ceiling when ESR approaches TMSR. Because the proposed algorithm explores all the possible partition-based schemes and considers the load balance of parallelization, the final ESR of channel processes is comparable with TMSR. The results based on three watersheds show that the maximum ESR can reach 91–98% of TMSR using 26-threads (Table 1).

4.3. Evaluating real parallel performance

Fig. 9 illustrates the variations in execution time, speedup ratio, parallel efficiency, and the efficiency deficit as the number of threads changes for the Dongjiang watershed. The execution time for hillslope processes drop sharply, and the speedup ratio increases, as the number of threads increases (Fig. 9a and b). The estimated speedup ratio for hillslope processes (ESR_{hs}) reaches 25.1, and the real speedup ratio (RSR_{hs}) reaches 21.6 (86.1% of ESR_{hs}) under 26-threads. For channel processes, the real speedup ratio under 26-threads ($RSR_{ch,26}$) is 6.6, approximating 89.2% of the estimated value (7.4). As hillslope grids account for 97.9% of all grids, the overall performance is mainly determined by performance of simulating hillslope processes.

In general, the parallel efficiency using more threads is lower

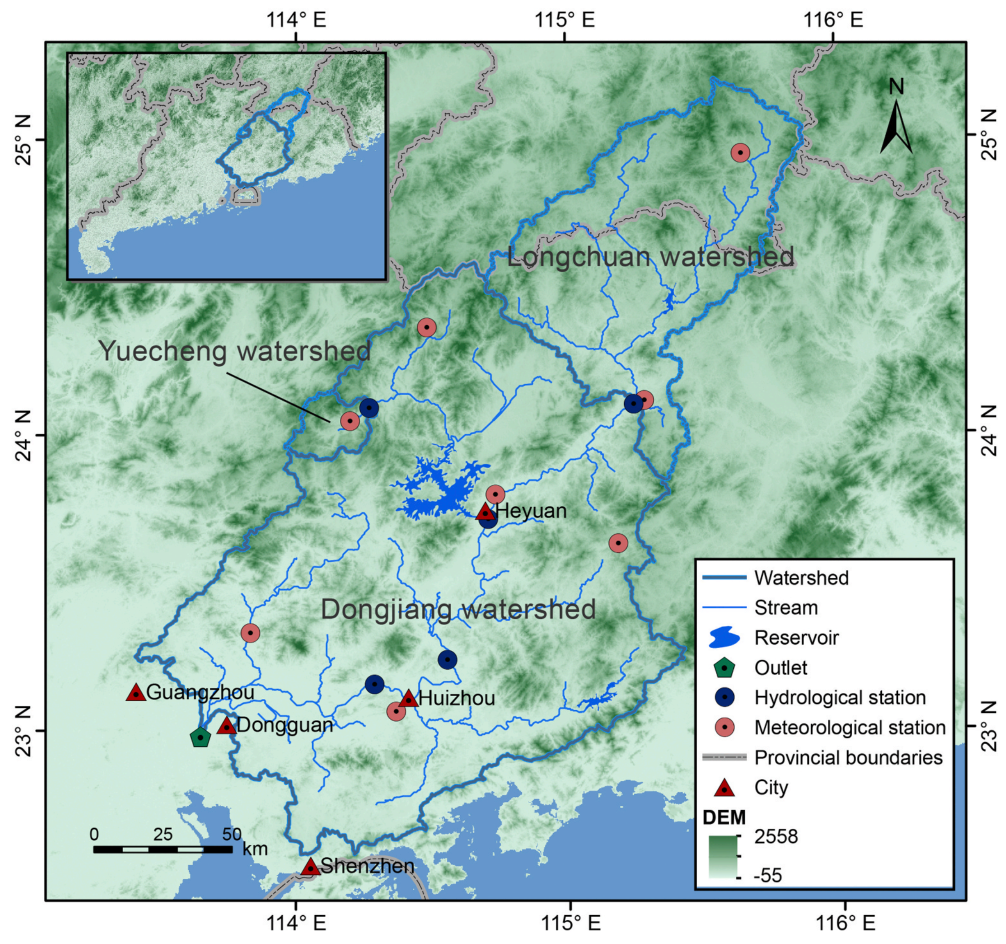

Fig. 6. Three watersheds located in southern China, and the drainage networks in these watersheds.

Table 1
The parallel performance for the three watersheds using 26 threads.

Watershed	Dongjiang	Longchuan	Yuecheng
Area (km^2)	31,612	8,157	541
Number of grids	790,304	203,914	13,513
Portion of channel grids	2.1%	2.7%	4.2%
$TMSR_{hs,26}$	26	26	26
$TMSR_{ch,\ 26}$	8.1	7.2	3.6
$TMSR_{26}$	24.8	24.2	20.6
$ESR_{hs,\ 26}$	25.1	25.0	24.3
$ESR_{ch,\ 26}$	7.4	6.7	3.6
ESR_{26}	23.9	23.3	19.5
$RSR_{hs,\ 26}$	21.6	21.8	23.3
$RSR_{ch,\ 26}$	6.6	6.1	3.3
RSR_{26}	20.6	20.3	19.4

*TMSR, ESR, and RSR denote the theoretical maximum speedup ratio, estimated speedup ratio, and real speedup ratio, respectively. *hs* and *ch* represent hillslope processes and channel processes, respectively. The subscript 26 refers to the number of threads.

(Fig. 9c). The estimated parallel efficiency for hillslope processes is relatively stable, approximating 97% of the theoretical efficiency. After the estimated speed ratio for channel process (ESR_{ch}) peaked at 7.4, the parallel efficiency drops continuously, which greatly reduces the parallel efficiency for simulating overall processes.

The efficiency deficit, defined as the difference between the real and estimated parallel efficiencie, indicates how the real speedup ratio approximates estimated performance. The efficiency deficit is less than 10% for all processes when the number of threads is less than 24. However, when the number of threads exceeds 28, the real efficiency decreases dramatically as the super-computer used in this study only has 28 cores (Fig. 9d).

Fig. 10 shows the overall performance of parallelization applied in three watersheds. The ESR_{hs} using 26-threads varies from 24.3 to 25.1 for the three watersheds and accounts for 93–97% of $TMSR_{hs}$. The performances among the three watersheds differ mainly in $TMSR_{ch}$, which is quantified by channel width function. As a result, the parallel simulation of the Dongjiang watershed has a larger speedup ratio compared with the two smaller watersheds (Table 1). Meanwhile, there is no significant difference in the efficiency deficit among the three watersheds, which is less than 10% when the number of threads is less than 24 (Fig. 10d). In general, APPA is efficient in parallel computing and the estimated performance is consistent with real performance.

5. Discussion

Grid-based distributed mechanistic models are increasingly used for evaluating surface and subsurface hydrological processes. Parallel computing attempts to tackle the computing challenge for applying distributed models at large spatial and long-term temporal scales. In general, a parallel algorithm is only feasible for models that fit specific parallel assumptions, which regulate the implementation of the parallel framework with little or no loss of model's accuracy. By setting assumptions as pre-conditions, a parallel algorithm can better describe ideal state of high-performance computation and seek the optimal method to reach the ideal performance. In this study, the proposed APPA algorithm is designed for grid-based distributed hydrological models to

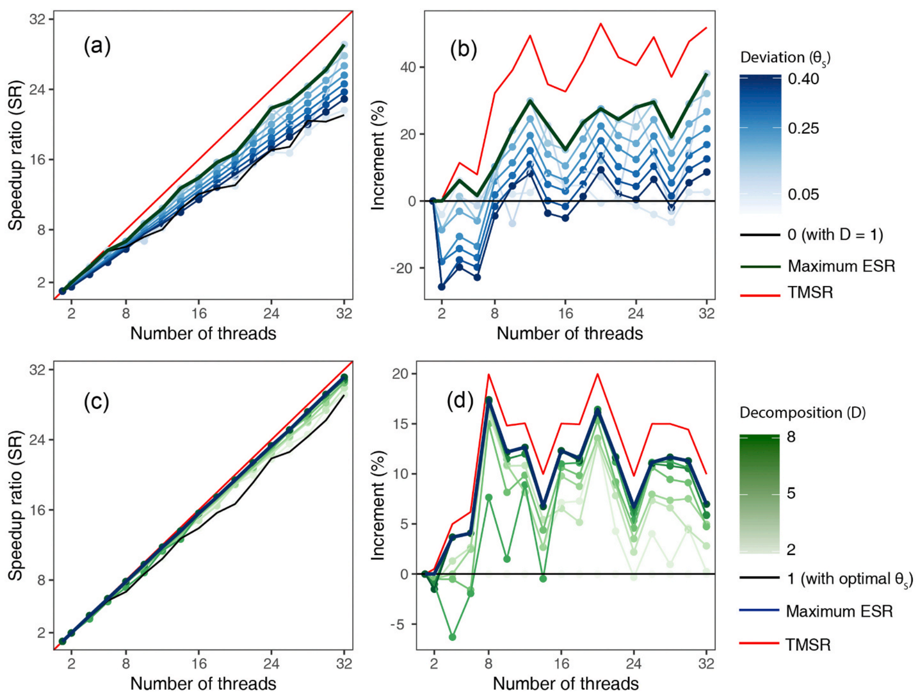

Fig. 7. Comparison of parallel performances for various partition-based parallel schemes for hillslope processes generated by different deviation (θ_s) and composition indexes (D). ESR is the estimated speedup ratio given by APPA. TMSR refers to theoretical maximum speedup ratio. θ_s ranges from 0 to 0.4, by a step of 0.05, and D ranges from 1 to 8 by a step of 1.

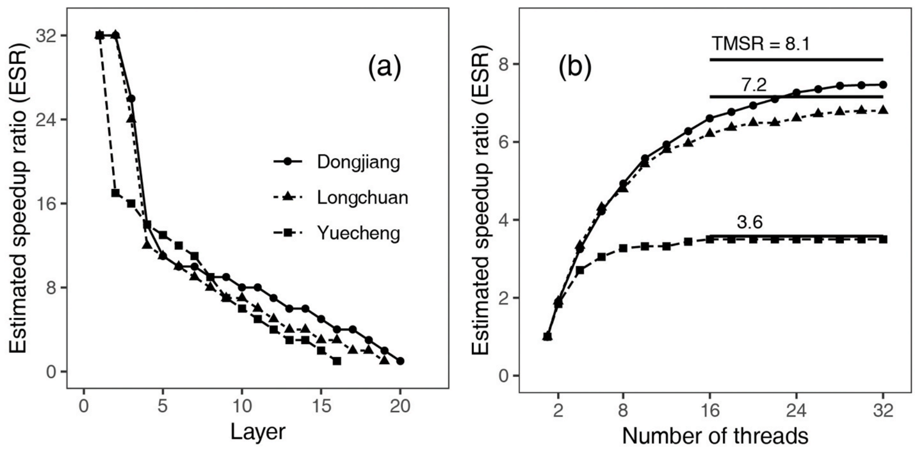

Fig. 8. (a) The estimated speedup ratio decreases as the simulated layer moves downstream to the outlet using 32 threads; (b) The estimated speedup ratio increases sharply at first, and then the trend flattens as the number of threads increases. The horizontal lines denote the TMSR of parallelization for channel processes in three watersheds.

obtain the optimal computing performance under given hardware condition.

Flow routings, which keep water flow along the gravity gradient, are the source of challenges in parallelization for hydrological models. Our proposed APPA assumes that models should adopt a single flow routing algorithm, by which flow generated in a center grid is routed to only one downslope neighboring grid (O'Callaghan and Mark, 1984). As a result, sub-basins are assumed to be independent in terms of hillslope processes. Otherwise, additional approaches are required to meet the demand of message passing among these parallelized units. For example, Vivoni et al. (2011) used ghost cells to simulate lateral subsurface flow for ridge grids, which are located on shared boundaries between sub-basins. In fact, multiple flow routing algorithms, based on topography (Quinn et al., 1991; Seibert and McGlynn, 2007) or real-time water level (Dai et al., 2019), consider flow dispersion at each grid and grids at watershed ridges, in which flow can belong to two or more partitioned sub-basins. Nevertheless, it is still likely for these models to adopt an independent partition-based parallelization without communication among partitioned units if flow generated at ridges of sub-basins is only routed in one direction. Since ridge-grids account for a small portion of all simulated grids and model outputs are averaged for the whole watershed, differences between parallel and serial computation may be negligible. Apart from this, the use of parallel algorithm can be performed flexibly with better understanding of model structures related to flow routing. In this study, APPA was applied for the whole model simulation as CHESS has closely coupled runoff production and routing processes at grid level. For models that have independent runoff generation and routing processes, which are usually simulated in a sequential order, APPA can also be selectively used for only routing processes (Liu et al., 2014).

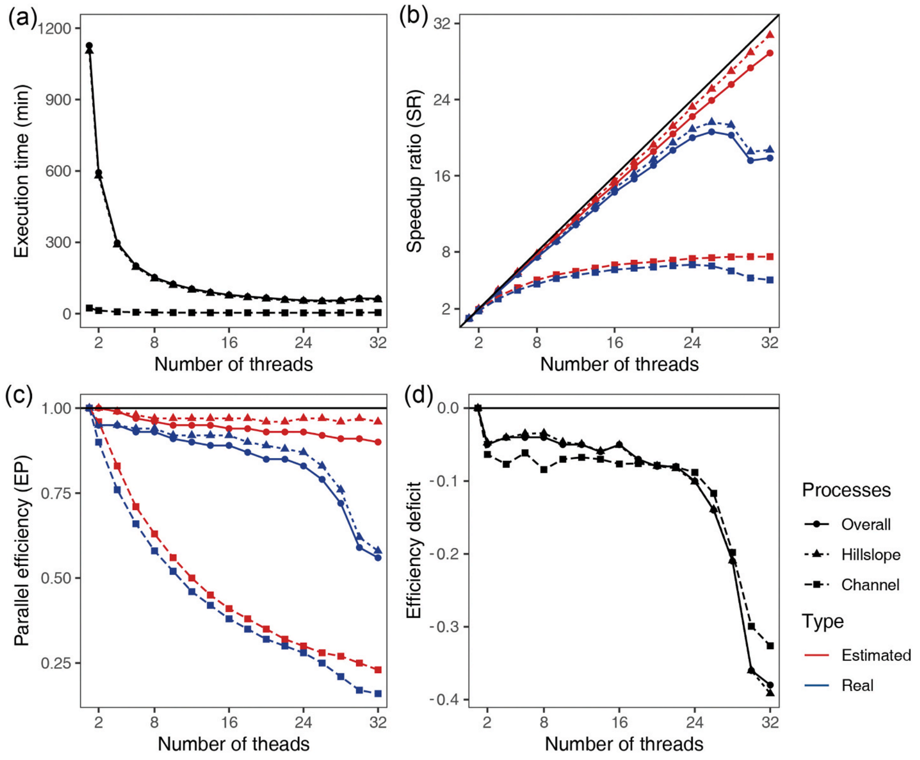

Fig. 9. Evaluation of parallel performance in simulating hillslope, channel, and overall processes based on a ten-year simulation for the Dongjiang watershed under different numbers of threads. The efficiency deficit denotes the difference between the real and estimated parallel efficiencies.

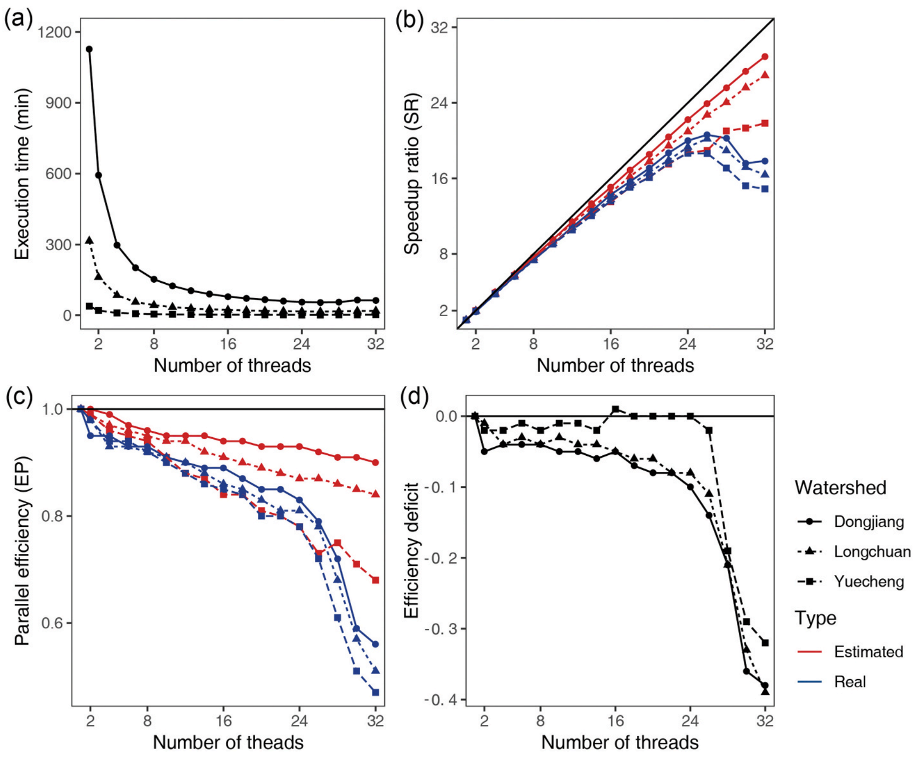

Fig. 10. Evaluation of the overall parallel performance in three watersheds. Data shown here are based on a ten-year simulation under different numbers of threads. The efficiency deficit denotes difference between the real and estimated parallel efficiencies.

APPA classifies simulated grids as either hillslope grids or channel grids. The former indicates where hillslope processes (e.g., surface and sub-surface flow routing) occur, and the later represents where channel processes (e.g., channel flow routing) take place. This assumption is suitable for most grid-based distributed models at regional or large-spatial scale (e.g., VIC, Liang et al., 1994; RHESSys, Tague and Band, 2004). For models that have no specification between the two processes, users of APPA can set a boundary value for accumulated runtime ($T_{G,BHC}$) to separate “hillslope grids” from “channel grids”. As suggested by Vivoni et al. (2011), the best strategy of parallelization for large river basins is to combine a balanced partitioning with an extended channel network. As the outlet of sub-basins must be a channel grid such that sub-basins are independent in terms of hillslope processes, the magnitude of $T_{G,BHC}$ will limit the size of the smallest sub-basin and imposes restriction on uses of more threads used in parallel computing. For example, when the value of $T_{G,BHC}$ is 10% of time for serial computing, the estimated speedup ratio for channel processes will approach the limit of 10 in spite of increasing number of threads. Empirically, the value of $T_{G,BHC}$ should be within the range of 0.5–5% to tap the potential of hillslope parallelization.

For grid-based models, if aforementioned assumptions are meet, APPA will approach the TMSR by searching for the optimal spatial domain decomposition and computational task allocation. To better illustrate the parallel performance, APPA was tested using a typical grid-based model, i.e., CHESS. In experiments, the proposed approach is not only efficient as a result of optimization for partition-based parallelization, but also easy to use since the deployment of parallel simulation with APPA requires little knowledge of shared memory programing to achieve the fork/join structure (Fig. A1). For parallel performance, the estimated speedup ranges from 19.5 to 23.9 using 26 threads for the three watersheds, which almost reach the theoretical limit (~96% of the TMSR, Table 1). The corresponding real speedup ranges from 19.4 to 20.6.

A comparison between our results and prior work in adoption of high performance using distributed hydrologic models indicate a set of general outcomes in parallel performance (Vivoni et al., 2011). In Table 2, we investigated various parallel frameworks and briefly summarized their key findings from case studies. The source of differences between those approaches lies in the basic simulation unit in hydrological simulation, which influences the complexity of model simulation. The

Table 2
Summary of various parallel frameworks.

References	Parallel framework/ Hydrological model	Basic unit	Protocol	Key features/findings	Best RSR (np*nt)	Derived REP	Data source in references	Additional notes
Apostolopoulos and Georgakakos (1997)	/	Sub-basin	/	A primary exploration for parallel computing in hydrological modelling	~2.3/4 (nt)	0.58	Fig. 6	/
Vivoni et al. (2011)	tRIBS model	Triangular irregular network (TIN)	MPI	Load balancing significantly improves RSR with proportionally faster runs as simulation complexity (domain resolution and channel network extent) increases	~23/32 (np*nt) ~70/512 (np*nt)	0.72 0.14	Fig. 10	Suggested combining a balanced partitioning with an extended channel network
Li et al. (2011)	DYRIM model	Hillslope/sub-basin	MPI	A new dynamic basin decomposition method	4.39/8 (nt) 5.34/13 (nt)	0.55 0.41	Table 1	/
Wang et al. (2011)	Xinanjiang model + Muskingum flow routing model	Hillslope/sub-basin	MPI	Found limited enhancement of computing efficiency due to constraints of the binary-tree-based drainage network	7.8/10 (nt)	0.78	Fig. 12	TMSR: 9.1
Wang et al. (2012)	Xianjiang model + diffuse wave flow routing model	Hillslope/sub-basin	MPI	Proved the existence of maximum speedup ratio curve in parallel computing of the binary-tree-based drainage network	10.1/13 (nt)	0.78	Fig. 8	TMSR:11.26
Liu et al. (2014)	A layered approach	Sub-basin or basic unit	OpenMP	Dispatched simulation tasks in sequential layers at sub-basin level or basic unit level	~6.7–8.4/ 12 (nt) ~9.1–12.5/ 24 (nt)	0.56–0.70 0.38–0.52	Fig. 5	Discussed different results for different resolutions
Liu et al. (2016)	A two-level layered approach	Sub-basin and basic unit	OpenMP + MPI	Dispatched simulation tasks at sub-basin level and basic unit level to multiple processes and threads, respectively	~12–20/24 (np*nt) ~14–34/48 (np*nt)	0.50–0.83 0.29–0.71	Figs. 8 and 9	Discussed different results for different resolutions or numbers of sub-basins
Zhu et al. (2019)	SEIMS	Sub-basin, hillslope, or basic simulation unit (e,g., grid, HRU)	OpenMP + MPI	A modular and parallelized watershed modeling framework	~18/ (10np*4nt)	0.45	Fig. 12	Suitable for a wide range of hydrological models
Zhang and Zhou (2019)	A particle-set strategy	Flow path network (FPN)	MPI	Decomposed the computational workload by randomly allocating runoff particles to concurrent computing processors	~24/32 (np*nt) ~56/128 (np*nt)	0.75 0.44	Fig. 15	Similar prediction accuracy and REP to that of tRIBS (Vivoni et al., 2011)
This study	APPA/CHESS model	Grid	OpenMP	A combination of flexible partition for the domain decomposition and load balance for approaching TMSR	10.1–10.8/ 12(nt) 19.4–20.6/ 26(nt)	0.84–0.90 0.75–0.79	Fig. 10	TMSR: 10.9–11.9/ 12 (nt) TMSR: 20.6–24.8/ 26 (nt) for three watersheds

* RSR: real speedup ratio; Derived REP: derived value of real parallel efficiency; TMSR: theoretical maximum speedup ratio; OpenMP: Open Multi-Processing, a shared-memory programing standard; MPI: Message Passing Interface, a message-passing standard; “~” means values are estimated from the figures; nt: number of threads; np: number of processes (for MPI).

larger the simulation unit in terms of the spatial scale (e.g., sub-basin, hillslope) is, the less complexity the flow routing network is. As a result, the enhancement of computing efficiency is limited in spite of increasing parallel resources due to constraints of the binary-tree structure of drainage network, which generally have fewer simulation units (RSR < 12, Li et al., 2011; Wang et al., 2011; Wang et al., 2012). In comparison, modelling with more simulation units, implemented in finer basic units (e.g., grids vs sub-basins), larger area coverage (e.g., larger watersheds vs smaller watersheds) or higher spatial resolutions (e.g., 90 m vs 270 m), is proved to have a larger speed up ratio (Vivoni et al., 2011; Liu et al., 2016; Zhang and Zhou, 2019). So as to better quantify the progress of proposed parallel framework, the theoretical maximum speedup ratios are defined for given processers and given model's applications (Wang et al., 2012; Liu et al., 2013). Compared with TMSR, the real parallel efficiency is mainly hampered by two main factors: the balance of the computation loads and the increase in input/output (I/O) costs (Vivoni et al., 2011; Zhang and Zhou, 2019). In this study, the combination of spatial domain decomposition and load-balanced task allocation is proved effective in reducing the former costs. A better load balance is achieved with a different extent of disaggregation (Fig. 7). Meanwhile, the communication costs are avoided since we adopted the fork/join structure of OpenMP. As a result, the real performance is close to the estimated with efficiency deficit stably less than $<10\%$ when the number of available threads are less than 24. This extra loss of efficiency can be attributed to other factors (e.g., the cost of creating threads and building parallel environment, hardware limit, data management tools). For example, the best fit between real and estimated speedup occurs in the smallest Yuecheng watershed (efficiency deficit $< 2\%$ across all experiments used different threads), indicating that the real performance of APPA can be improved with an advanced data management method in parallelization.

Although this study demonstrated the possibility for grid-based hydrological models to approach TMSR at thread-level, the proposed APPA can be modified for other types of models. For example, APPA can be modified to facilitate the need of parallelization for models based on other basic units (e.g., TIN, HRU) or other routing-related computations with huge simulation complexity. If the runtime and flow routing of each basic unit are defined, APPA can be used for spatial decomposition to maximize speedup. Meanwhile, it seems that APPA can be deployed at process-thread level with MPI programing, grepping the power of computer clusters. The adoption of multiple parallelization levels (e.g., threads and processes) can break through the limit of TMSR at thread level, which depends on available cores in a computer (RSR > 30, Vivoni et al., 2011; Liu et al., 2016). However, the incorporation of MPI is relatively more complex than that of OpenMP and may have lower efficiency due to the load of communication among processes (Zhang and Zhou, 2019).

6. Conclusion

This study proposed the automatic partition-based parallel algorithm (APPA) to achieve the optimal state of parallelization for grid-based distributed models. The results suggest that the performance of parallel computation using grid-based distributed models is sensitive to the spatial domain decomposition. By combining flexible partition of units (e.g., sub-basins or channel units) with the load balance of thread allocation, the estimated speedup ratio in APPA increases noticeably and reaches 93–97% of TMSR for simulating hillslope processes and 91–98% of TMSR for simulating channel processes using 26-threads. The real speedup ratio is greater than 90% of the estimated speedup ratio when the number of threads does not exceed the number of cores in our case study. Overall, our proposed APPA is useful for parallelizing grid-based models to maximize the parallel performance under a given number of threads and thus increase the computation efficiency.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This study was supported by the "National Key R&D Program of China (2017YFC0405900)", the Chinese National Natural Science Foundation (41671192), and the Guangzhou Municipal Scientific Program (42050441).

Appendix

```
// hillslope processes
#pragma omp parallel for schedule (static, 1) num_threads(nthread)
for (ithread = 0; ithread < nthread; ithread ++) {//nthread is the number of threads
    parallel_hillslope_daily (ithread_grid_array); // grid array that stores addresses of grids
}
// channel processes
for (jlayer = 0; jlayer < nlayer; jlayer++){//nlayer is the number of layers
    #pragma omp parallel for schedule (static, 1) num_threads(nthread)
    for (ithread = 0; ithread < nthread; ithread ++) {//nthread is the number of threads
        parallel_channel_daily(jlayer_ithread_grid_array);// grid array that stores addresses of grids
    }
}
```

Fig. A1. C++ pseudo code of the parallel algorithm.

References

Apostolopoulos, T.K., Georgakakos, K.P., 1997. Parallel computation for streamflow prediction with distributed hydrologic models. J. Hydrol. 197 (1–4), 1–24.

Arnold, J.G., Srinivasan, R., Muttiah, R.S., Williams, J.R., 1998. Large area hydrologic modeling and assessment part I: model development 1. J. Am. Water Resour. Assoc. 34 (1), 73–89.

Dai, Y., Chen, L., Zhang, P., Xiao, Y.C., Hou, X.S., Shen, Z.Y., 2019. Construction of a cellular automata-based model for rainfall-runoff and NPS pollution simulation in an urban catchment. J. Hydrol. 568, 929–942.

Hwang, H.T., Park, Y.J., Sudicky, E.A., Forsyth, P.A., 2014. A parallel computational framework to solve flow and transport in integrated surface–subsurface hydrologic systems. Environ. Model. Software 61, 39–58.

Li, T., Wang, G., Chen, J., Wang, H., 2011. Dynamic parallelization of hydrological model simulations. Environ. Model. Software 26 (12), 1736–1746.

Liang, X., Lettenmaier, D.P., Wood, E.F., Burges, S.J., 1994. A simple hydrologically based model of land surface water and energy fluxes for general circulation models. J. Geophys. Res. 99 (D7), 14415.

Liu, J., Zhu, A.X., Liu, Y., Zhu, T., Qin, C.-Z., 2014. A layered approach to parallel computing for spatially distributed hydrological modeling. Environ. Model. Software 51, 221–227.

Liu, J., Zhu, A.X., Qin, C.-Z., 2013. Estimation of theoretical maximum speedup ratio for parallel computing of grid-based distributed hydrological models. Comput. Geosci. 60, 58–62.

Liu, J., Zhu, A.X., Qin, C.-Z., Wu, H., Jiang, J., 2016. A two-level parallelization method for distributed hydrological models. Environ. Model. Software 80, 175–184.

O'Callaghan, J.F., Mark, D.M., 1984. The extraction of drainage networks from digital elevation data. Comput. Vis. Graph Image Process 28 (3), 323–344.

Qin, C.-Z., Zhan, L., 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.

Quinn, P., Beven, K., Chevallier, P., Planchon, O., 1991. The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models. Hydrol. Process. 5 (1), 59–79.

Seibert, J., McGlynn, B.L., 2007. A new triangular multiple flow direction algorithm for computing upslope areas from gridded digital elevation models. Water Resour. Res. 43 (4).

Shirazi, B., Wang, M., Pathak, G., 1990. Analysis and evaluation of heuristic methods for static task scheduling. J. Parallel Distr. Comput. 10 (3), 222–232.

Tague, C.L., Band, L.E., 2004. RHESSys: regional hydro-ecologic simulation system—an object-oriented approach to spatially distributed modeling of carbon, water, and nutrient cycling. Earth Interact. 8 (19), 1–42.

Tang, G., Hwang, T., Pradhanang, S.M., 2014. Does consideration of water routing affect simulated water and carbon dynamics in terrestrial ecosystems? Hydrol. Earth Syst. Sci. 18 (4), 1423–1437.

Tang, G., Li, S., Yang, M., Xu, Z., Liu, Y., Gu, H., 2019. Streamflow response to snow regime shift associated with climate variability in four mountain watersheds in the US Great Basin. J. Hydrol. 573, 255–266.

Tian, Y., Peters-Lidard, C.D., Kumar, S.V., Geiger, J., Houser, P.R., Eastman, J.L., Dirmeyer, P., Doty, B., Adams, J., 2008. High-performance land surface modeling with a Linux cluster. Comput. Geosci. 34 (11), 1492–1504.

Vivoni, E.R., Mascaro, G., Mniszewski, S., Fasel, P., Springer, E.P., Ivanov, V.Y., Bras, R. L., 2011. Real-world hydrologic assessment of a fully-distributed hydrological model in a parallel computing environment. J. Hydrol. 409 (1–2), 483–496.

Wang, H., Fu, X., Wang, G., Li, T., Gao, J., 2011. A common parallel computing framework for modeling hydrological processes of river basins. Parallel Comput. 37 (6–7), 302–315.

Wang, H., Zhou, Y., Fu, X., Gao, J., Wang, G., 2012. Maximum speedup ratio curve (MSC) in parallel computing of the binary-tree-based drainage network. Comput. Geosci. 38 (1), 127–135.

Zhang, F., Zhou, Q., 2019. Parallelization of the flow-path network model using a particle-set strategy. Int. J. Geogr. Inf. Sci. 33 (10), 1984–2010.

Zhu, L.-J., Liu, J., Qin, C.-Z., Zhu, A.X., 2019. A Modular and Parallelized Watershed Modeling Framework, vol. 122. Environmental Modelling & Software.