Advances in Water Resources 67 (2014) 1–15

Contents lists available at ScienceDirect

Advances in Water Resources

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

A parallel dynamic programming algorithm for multi-reservoir system optimization

Xiang Li [a], Jiahua Wei [a], Tiejian Li [a], Guangqian Wang [a], William W.-G. Yeh [b,*]

[a] State Key Laboratory of Hydroscience & Engineering, Tsinghua University, Beijing 100084, China
[b] Department of Civil and Environmental Engineering, University of California, Los Angeles, CA 90095, USA

ARTICLE INFO

Article history:
Received 8 May 2013
Received in revised form 8 January 2014
Accepted 12 January 2014
Available online 30 January 2014

Keywords:
Dynamic programming
Multi-reservoir system optimization
Joint operation
Parallel computing

ABSTRACT

This paper develops a parallel dynamic programming algorithm to optimize the joint operation of a multi-reservoir system. First, a multi-dimensional dynamic programming (DP) model is formulated for a multi-reservoir system. Second, the DP algorithm is parallelized using a peer-to-peer parallel paradigm. The parallelization is based on the distributed memory architecture and the message passing interface (MPI) protocol. We consider both the distributed computing and distributed computer memory in the parallelization. The parallel paradigm aims at reducing the computation time as well as alleviating the computer memory requirement associated with running a multi-dimensional DP model. Next, we test the parallel DP algorithm on the classic, benchmark four-reservoir problem on a high-performance computing (HPC) system with up to 350 cores. Results indicate that the parallel DP algorithm exhibits good performance in parallel efficiency; the parallel DP algorithm is scalable and will not be restricted by the number of cores. Finally, the parallel DP algorithm is applied to a real-world, five-reservoir system in China. The results demonstrate the parallel efficiency and practical utility of the proposed methodology.

1. Introduction

Dynamic programming (DP), an algorithm attributed largely to Bellman [3], is developed for optimizing a multi-stage (the term “stage” represents time step throughout the paper) decision process. If the return or cost at each stage is independent and satisfies the monotonicity and separability conditions [23], the original multi-stage problem can be decomposed into stages with decisions required at each stage. The decomposed problem then can be solved recursively, two stages at a time, using the recursive equation of DP. DP is particularly suited for optimizing reservoir management and operation as the structure of the optimization problem conforms to a multi-stage decision process. Over the past four decades, DP had been used extensively in the optimization of reservoir management and operation [4,6,8,13,22,35,37–40].

In the discrete form of DP, storage of each reservoir is discretized into a finite number of levels. By exhaustive enumeration over all possible combinations of discrete levels at each stage for all reservoirs in a system, global optimality can be assured in a discrete sense. However, the well-known “curse of dimensionality” [2] limits the application of DP to multi-state variable problems, as the state space increases exponentially with an increase in the number of state variables. This drastic increase in state space and the consequent random access memory (RAM) requirement quickly can exceed the hardware capacity of a modern computer [13]. A variety of DP variants, such as incremental dynamic programming (IDP) [15], dynamic programming successive approximations (DPSA) [14], incremental dynamic programming and successive approximations (IDPSA) [32] and discrete differential dynamic programming (DDDP) [9] have been proposed to alleviate the dimensionality problem. However these variants all require an initial trajectory for each state variable. For a non-convex problem, there is no assurance of convergence to the global optimum. Recently, Mousavi and Karamouz [22] reduced the computation time of a DP model for a multi-reservoir system by diagnosing infeasible storage combinations and removing them from further computations. Zhao et al. [40] proposed an improved DP model for optimizing reservoir operation by taking advantage of the monotonic relationship between reservoir storage and the optimal release decision. However, the model only can be applied to reservoir operation with a concave objective function.

Because of the hardware limitations of a single computer as well as large-scale computing requirements, parallel computing has been applied in many fields [25]. In the water resources field, there are several successful examples. Bastian and Helmig [1]

* Corresponding author. Tel.: +1 3108252300; fax: +1 3108257581.
E-mail addresses: l-xiang09@mails.tsinghua.edu.cn, ideal.thu@gmail.com (X. Li), weijiahua@tsinghua.edu.cn (J. Wei), litiejian@tsinghua.edu.cn (T. Li), dhhwgq@tsinghua.edu.cn (G. Wang), williamy@seas.ucla.edu (W.W.-G. Yeh).

http://dx.doi.org/10.1016/j.advwatres.2014.01.002
0309-1708/© 2014 Elsevier Ltd. All rights reserved.

employed a data parallel implementation of a Newton-multi-grid algorithm for two-phase flow in porous media. Kollet and Maxwell [10] incorporated an efficient parallelism into an integrated hydrologic model – ParFlow. Tang et al. [31] used the master–slave and multi-population parallelization schemes for the Epsilon-Nondominated Sorted Genetic Algorithm-II and applied them to several water resources problems. Wang et al. [33] designed a common parallel framework, while Li et al. [16] introduced a dynamic parallel algorithm for hydrological model simulations. Rouholahnejad et al. [27] proposed a parallelization framework for hydrological model calibration. Wu et al. [36] parallelized the Soil and Water Assessment Tool. These previous studies demonstrated that by using parallel computing associated with proper parallelization strategies for optimization or simulation, solutions can be improved and computation times can be reduced dramatically. Although parallel computing has been used extensively in hydrologic models, the potential has not been explored fully in the field of reservoir operation [29].

Over the last several decades, with the rapid development of high-performance computing (HPC) environments, parallel dynamic programming algorithm has been studied from theories gradually to applications. From the theoretical point of view, Casti et al. [5] presented various forms of parallelism and the corresponding parallel algorithms for DP. Rytter [28] considered some DP problems and estimated the computation complexity and the number of computing processes based on a hypothetical parallel random access machine, namely the shared memory architecture. However, the shared memory architecture may not be supportable for large RAM requirement problems, since all parallel tasks access the finite RAM in the architecture (because of motherboard size restrictions). From the practical point of view, El Baz and Elkihel [7] designed several load-balancing strategies and applied a parallel DP algorithm with Open MP, a protocol based on the shared memory architecture, to the 0–1 knapsack problem. Martins et al. [19] and Tan et al. [30] parallelized DP algorithms for a class of problems, such as biological sequence comparisons. Piccardi and Soncini-Sessa [24] studied the discretization and inflow correlation on the solution reliability of a stochastic dynamic programming (SDP) and exploited parallel computing to a one-reservoir optimal control problem; the parallelization was attributed largely to a vectorizing compiler or parallelizing compiler. Li et al. [17] implemented a knowledge-based approach to accurately determine hydropower generation and developed a master–slave parallel DP algorithm on an HPC system to optimize the coordinated operation of the Three Gorges Project and Gezhouba Project cascade hydropower plants in China. The master–slave is a frequently-used parallel paradigm for parallelizing a DP algorithm, where the master process has a full version of the DP algorithm, and calls slave processes to evaluate and return objective values and saves all variables in the RAM of the master process. However, this parallel paradigm merely shortens computation time but does not alleviate the RAM requirement.

In this paper, we develop a parallel DP algorithm for multi-reservoir system optimization. The parallel DP algorithm aims at reducing the computation time and alleviating the RAM requirement, taking advantage of parallel computing on the distributed memory architecture. The paper is organized as follows: Section 2 formulates a DP model for a multi-reservoir system optimization problem; Section 3 analyzes the parallelism inherent to the DP algorithm and then proposes a peer-to-peer parallel paradigm for the DP algorithm; Section 4 considers the classic four-reservoir example and tests the parallel DP algorithm on the problem; Section 5 applies the parallel DP algorithm to a real-world five-reservoir system in China; finally, Section 6 presents the conclusions.

2. Dynamic programming model

2.1. Objective function

A typical objective function for the optimal operation of a multi-reservoir system is either to maximize benefit or minimize operational cost. If more than one objective is involved, they can be combined by the weighting method to form a composite objective function. Without loss of generality, let us consider the maximization problem. According to Bellman's principle of optimality [3], the traditional forward recursive equation of an n-dimensional DP model (the term "dimension" here refers to the number of reservoirs) can be represented as:

$$F^*_{t+1}(\mathbf{S}(t+1)) = \max\{f_t(\mathbf{S}(t), \mathbf{S}(t+1)) + F^*_t(\mathbf{S}(t))\} \tag{1}$$

where t is the time index, $t \in [1, T]$; $\mathbf{S}(t)$ is the storage vector at the beginning of time step t; $\mathbf{S}(t) = [S_1(t), \ldots, S_i(t), \ldots, S_n(t)]^T$; $\mathbf{S}(t+1)$ is the storage vector at the end of time step t; i is the reservoir index, $i \in [1, n]$; $F^*_t(\bullet)$ is the maximum cumulative return from the first time step to the beginning of the tth time step resulting from the joint operation of n reservoirs; initially, $F^*_1(\bullet) = 0$; $F^*_{t+1}(\bullet)$ is the maximum cumulative return from the first time step to the end of the tth time step resulting from the joint operation of n reservoirs; and $f_t(\bullet)$ is the objective function to be maximized during time step t. Note that Eq. (1) is the inverted form of a DP model with reservoir storages as the decision variables. In the non-inverted DP model, the releases are the decision variables. For a deterministic DP model, the ending storage is related to the beginning storage by the continuity equation.

2.2. Constraints

In the operation of a multi-reservoir system, each individual reservoir is subject to its own set of constraints, while the reservoir system is subject to system constraints brought by the interconnection of reservoirs. Specifically, we consider the following constraints:

- Continuity equation:

$$\mathbf{S}(t+1) = \mathbf{S}(t) + \mathbf{I}(t) - \mathbf{M} \bullet \mathbf{R}(t) \quad \forall t \tag{2}$$

where $\mathbf{I}(t)$ is the vector of inflows to reservoirs $(i = 1, \ldots, n)$ during time step t; $\mathbf{R}(t)$ is the vector of total releases from reservoirs $(i = 1, \ldots, n)$ during time step t; $\mathbf{R}(t) = [R_1(t), \ldots, R_i(t) \ldots, R_n(t)]^T$; and $\mathbf{M}$ is the $n \times n$ reservoir system connectivity matrix. Without loss of generality, we assume that evaporation loss is balanced by precipitation.

- Initial and final reservoir storages:

$$\mathbf{S}(1) = \mathbf{S}^{\text{initial}} \tag{3}$$

$$\mathbf{S}(T+1) \geqslant \mathbf{S}^{\text{final}} \tag{4}$$

where $\mathbf{S}^{\text{initial}}$ and $\mathbf{S}^{\text{final}}$ are the vectors of initial storages and final expected storages of reservoirs $(i = 1, \ldots, n)$.

- Lower and upper bounds on storages:

$$\mathbf{S}^{\min}(t+1) \leqslant \mathbf{S}(t+1) \leqslant \mathbf{S}^{\max}(t+1) \quad \forall t \tag{5}$$

where $\mathbf{S}^{\min}(t+1)$ and $\mathbf{S}^{\max}(t+1)$ are the vectors of minimum and maximum storages of reservoirs $(i = 1, \ldots, n)$ at the end of time step t.

- Lower and upper bounds on releases:
  For all reservoirs:

$$\mathbf{R}^{\min}(t) \leqslant \mathbf{R}(t) \leqslant \mathbf{R}^{\max}(t) \quad \forall t \tag{6}$$

where $\mathbf{R}^{\min}(t)$ is the vector of the minimum required releases from reservoirs $(i = 1, \ldots, n)$ during time step t; and $\mathbf{R}^{\max}(t)$ is the vector of maximum allowable releases from reservoirs ($i = 1, \dots, n$) during time step t.

For reservoirs in parallel:

$$PR_l^{\min}(t) \leqslant \sum_{i \in U_l} R_i(t) \leqslant PR_l^{\max}(t) \quad \forall t, \forall j \tag{7}$$

where l is the index of river confluence points; $l \in [1, L]$ and L is the number of river confluence points; U_l is the set of reservoirs in parallel at river confluence point l; and $PR_l^{\min}(t)$ and $PR_l^{\max}(t)$ are the minimum and maximum releases at river confluence point l during time step t.

- Lower and upper bounds on outputs:

For all reservoirs:

$$\mathbf{N}^{\min}(t) \leqslant \mathbf{N}(t) \leqslant \mathbf{N}^{\max}(t) \quad \forall t \tag{8}$$

where $\mathbf{N}(t)$ is the vector of outputs produced from reservoirs ($i = 1, \dots, n$) during time step t; $\mathbf{N}(t) = [N_1(t), \dots, N_i(t) \dots, N_n(t)]^T$; $\mathbf{N}^{\min}(t)$ is the vector of minimum required outputs from reservoirs ($i = 1, \dots, n$) during time step t; and $\mathbf{N}^{\max}(t)$ is the vector of maximum allowable outputs from reservoirs ($i = 1, \dots, n$) during time period t.

For the entire reservoir system:

$$N^{\min}(t) \leqslant \sum_{i=1}^{n} N_i(t) \leqslant N^{\max}(t) \quad \forall t \tag{9}$$

where $N^{\min}(t)$ is the minimum required output from the reservoir system during time step t; and $N^{\max}(t)$ is the maximum allowable output from the reservoir system during time step t.

Note that reservoir storages, releases and outputs are variables to be solved; the others are the known input data in the optimization.

3. Parallelization strategy

In this study, we use the Windows HPC Server 2012 R2 operating system (OS). The HPC system consists of 20 IBM HS22 blades and an Infiniband 40 GBps network. Each blade has two Intel® Xeon® E5645 2.40 GHz CPUs (each CPU has six physical cores) and 12 GB of RAM. The Intel® Xeon® CPU employs hyper-threading technology that includes key features: (1) for each physical core, the OS addresses two logical cores and can schedule two processes simultaneously (i.e. a logical core conducts one process). Thus there are a total of 480 logical cores in the HPC system; (2) each logical core within a physical core has its own execution resources and also shares execution resources with the other logical core; (3) when two processes on two logical cores belong to the same physical core and need same shared execution resources, one process should stall and give the execution resources to the other process until it finishes. In other words, the two logical cores within a physical core cannot work as two independent physical cores. Fig. 1 presents the schematic representation of the HPC system, which is the distributed memory architecture.

Section 3.1 establishes the foundation for using the serial DP algorithm to solve the multi-reservoir system optimization problem. We summarize the procedures of DP algorithm, illustrate the DP algorithm's solution space with the help of a matrix, convert the vector form Eq. (1) into a scalar form, and estimate the computation time and RAM requirement for a multi-reservoir DP problem. Section 3.2 analyzes the concurrency and dependency of executing the DP algorithm and employs a peer-to-peer parallel paradigm to parallelize the DP algorithm. The aim of this paradigm is to shorten computation time as well as alleviate the RAM requirement. We develop the parallel programs using C++ language under the Microsoft Visual Studio 2010 platform and the message passing interface (MPI) protocol [21], allowing us to coordinate a parallel program running on multiple computing processes in the distributed memory architecture.

3.1. Preparation for parallelization

In general, the solution of the DP algorithm includes two procedures [5,38]:

Procedure I: Solve the recursive equation (1) sequentially, two stages (corresponding to time steps) at a time, save the optimal transitions or candidate paths (i.e. $\mathbf{S}(t) \to \mathbf{S}(t+1)$) in the memory, and save the maximum cumulative returns (i.e. $F_{t+1}^*(\mathbf{S}(t+1))$) in the memory for use in the next time step. The procedure continues until the final time step is reached.

Procedure II: Trace back the optimal path, based on the saved optimal transitions, from the final time step to the first time step to determine the consequent storage trajectories (i.e. $\mathbf{S}(T+1) \to \cdots \to \mathbf{S}(2) \to \mathbf{S}(1)$) and release trajectories (i.e. $\mathbf{R}(T) \to \cdots \to \mathbf{R}(2) \to \mathbf{R}(1)$).

For this analysis, we perform a conversion for Eq. (1). Assuming the storage of each reservoir is discretized into m levels, the number of storage combinations of all interconnected reservoirs is m^n at any time step (see Fig. 2). Note that storage is defined either at the beginning or the end of a time step. For the sake of clarity, we introduce $C(p_t, t)$ to denote a storage combination of all interconnected reservoirs in a system at the beginning of time step t, where p_t denotes the serial number of a storage combination at the beginning of time step t, $p_t \in [1, m^n]$. Thus all possible storage combinations of all reservoirs over the entire planning horizon can be expressed abstractly as:

$$\mathbf{C} = \begin{bmatrix} C(1,2) & \cdots & C(1,t) & \cdots & C(1,T+1) \\ \vdots & \ddots & \vdots & \iddots & \vdots \\ \vdots & \cdots & C(p_t,t) & \cdots & \vdots \\ \vdots & \iddots & \vdots & \ddots & \vdots \\ C(m^n,2) & \cdots & C(m^n,t) & \cdots & C(m^n,T+1) \end{bmatrix}_{m^n \times T} \tag{10}$$

where $\mathbf{C}$ is the $m^n \times T$ matrix; $\mathbf{C} = [\mathbf{C}(2), \dots, \mathbf{C}(t), \dots, \mathbf{C}(T+1)]$; $C(p_1, 1)$ is the initial storage combination, often fixed; $\mathbf{C}^*$ is denoted as the $m^n \times T$ matrix of optimal transitions or candidate paths, used to save the optimal previous storage combinations to the current ones so as to trace back the optimal path; and $\mathbf{C}^*$'s element $C^*(p_{t+1}, t+1)$ saves the optimal storage combination at the beginning of time step t to storage combination p_{t+1} at the end of time step t. In addition, all maximum cumulative returns from the first time step to the beginning of the tth time step are denoted as $\mathbf{F}_t^*$, where $\mathbf{F}_t^*$ is the $m^n \times 1$ matrix. By substituting $C(p_t, t)$ ($p_t = 1, \dots, m^n$) for $\mathbf{S}(t)$, the vector form Eq. (1) can be rewritten into an equivalent scalar form Eq. (11):

$$F_{t+1}^*(C(p_{t+1}, t+1)) = \max\{f_t(C(p_t, t), C(p_{t+1}, t+1)) + F_t^*(C(p_t, t))\} \tag{11}$$

Fig. 3 schematically illustrates the computing procedures and computer memory used for the serial DP algorithm. Fig. 4 shows the two computing procedures of the serial DP algorithm, based on the scalar form Eq. (11). First, the DP algorithm executes *Procedure I*: For a given p_{t+1}, to determine the maximum cumulative return $F_{t+1}^*(C(p_{t+1}, t+1))$ and optimal transition $C^*(p_{t+1}, t+1)$, the objective function $f_t(\bullet)$ of all possible transitions must be examined between $C(p_t, t)$ ($p_t = 1, 2, \dots, m^n$) and $C(p_{t+1}, t+1)$ and $F_t^*(C(p_t, t))$ ($p_t = 1, 2, \dots, m^n$) has to be added as well. Then, the maximum cumulative return $F_{t+1}^*(C(p_{t+1}, t+1))$ and optimal transition $C^*(p_{t+1}, t+1)$ are saved in computer memory. After all maximum cumulative return $F_{t+1}^*(C(p_{t+1}, t+1))$ ($p_{t+1} = 1, 2, \dots, m^n$) and optimal transition $C^*(p_{t+1}, t+1)$ ($p_{t+1} = 1, 2, \dots, m^n$) are derived at the

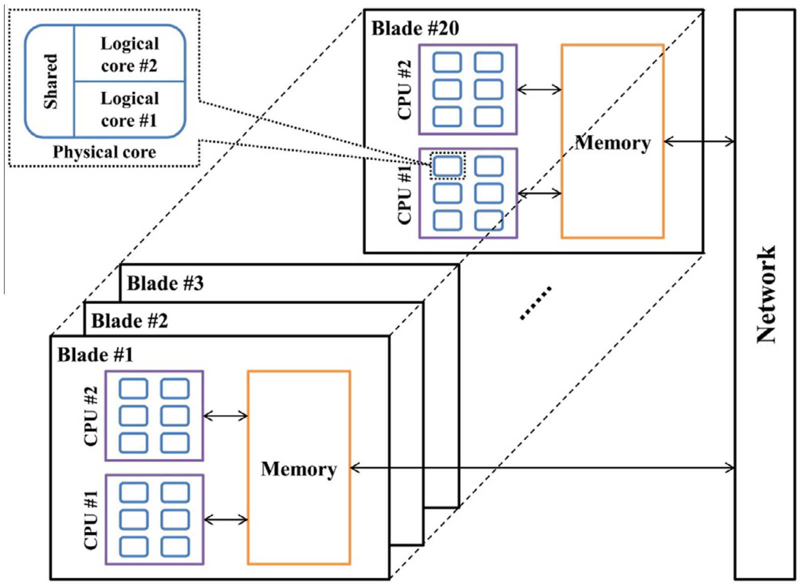


Fig. 1. Schematic representation of the HPC system.

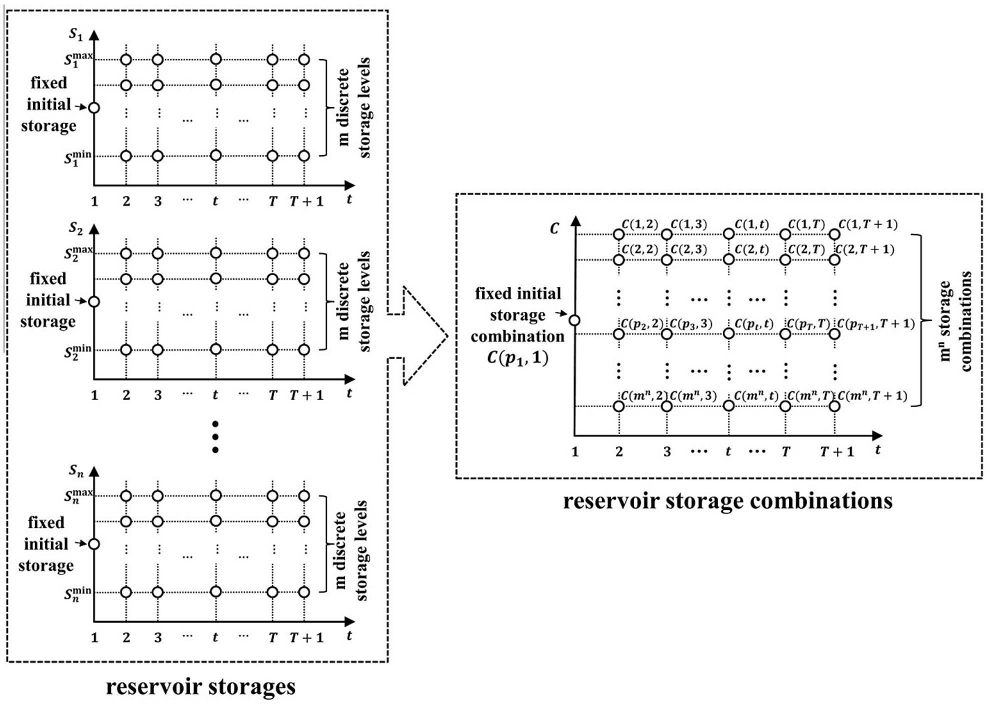

fixed initial storage
m discrete storage levels
reservoir storages
fixed initial storage combination $C(p_1,1)$
m^n storage combinations
reservoir storage combinations

Fig. 2. Conversion from reservoir storages to reservoir storage combinations.

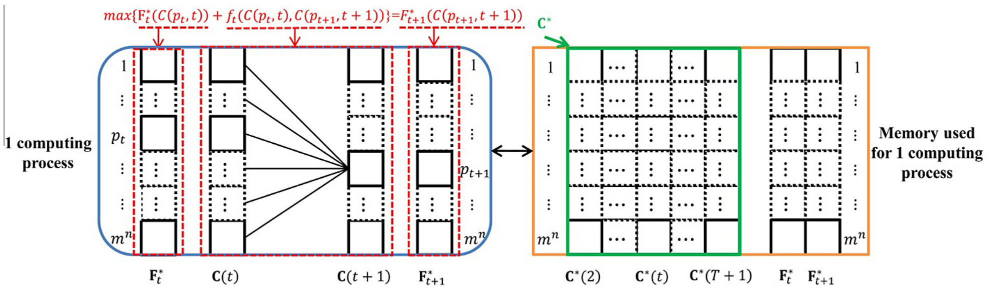

$max\{F_t^*(C(p_t,t)) + f_t(C(p_t,t), C(p_{t+1},t+1))\} = F_{t+1}^*(C(p_{t+1},t+1))$
C^*
1 computing process
Memory used for 1 computing process
F_t^*
$C(t)$
$C(t+1)$
F_{t+1}^*
$C^*(2)$
$C^*(t)$
$C^*(T+1)$

Fig. 3. Schematic illustration of computing procedures and computer memory used for the serial DP algorithm.

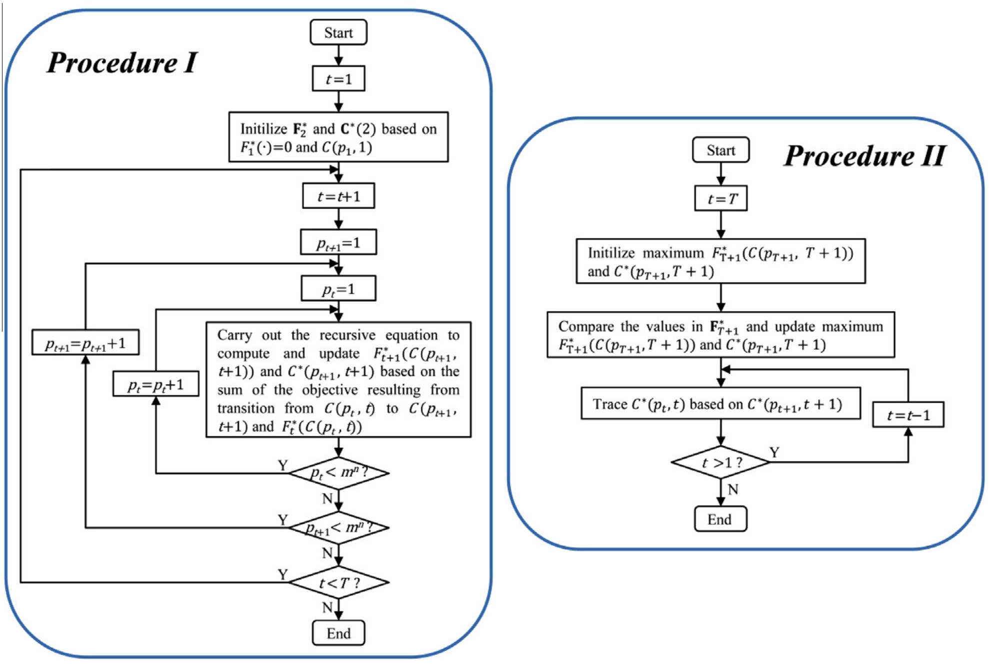

Procedure I
Start
t=1
Initilize $\mathbf{F}_2^*$ and $\mathbf{C}^*(2)$ based on $F_1^*(\cdot)=0$ and $C(p_1, 1)$
t=t+1
$p_{t+1}=1$
$p_t=1$
Carry out the recursive equation to compute and update $F_{t+1}^*(C(p_{t+1}, t+1))$ and $C^*(p_{t+1}, t+1)$ based on the sum of the objective resulting from transition from $C(p_t, t)$ to $C(p_{t+1}, t+1)$ and $F_t^*(C(p_t, t))$
$p_{t+1}=p_{t+1}+1$
$p_t=p_t+1$
$p_t< m^n$?
$p_{t+1}< m^n$?
$t<T$?
Y
N
End
Procedure II
Start
t=T
Initilize maximum $F_{T+1}^*(C(p_{T+1}, T+1))$ and $C^*(p_{T+1}, T+1)$
Compare the values in $\mathbf{F}_{T+1}^*$ and update maximum $F_{T+1}^*(C(p_{T+1}, T+1))$ and $C^*(p_{T+1}, T+1)$
Trace $C^*(p_t, t)$ based on $C^*(p_{t+1}, t+1)$
t=t−1
$t>1$?
Y
N
End

Fig. 4. Flowcharts of *Procedure I* and *Procedure II* of the serial DP algorithm.

end of time step t, the algorithm proceeds to the next time step until the end of the planning horizon. Then the DP algorithm carries-out *Procedure II*: The optimal storage transition for each reservoir as well as the corresponding release policy can be traced by a backward sweep.

The computation time of the DP algorithm, mainly from *Procedure I*, can be approximately estimated as:

$$\tau_1 = m^{2n} \times \Delta\tau \times T \tag{12}$$

where τ_1 is the wall clock time of using one computing process; there are a total number of m^{2n} time evaluations for Eq. (1) at each time step; all evaluations for Eq. (1) are assumed to require the same average wall clock time $\Delta\tau$. It should be noted that this is an upper bound estimation that includes the possible infeasible transitions, which are discarded in the actual computation. The infeasible transitions are the transitions that cannot be reached due to insufficient inflow to a reservoir [22].

On the other hand, *Procedure I* requires large RAM capacity. For illustration, we consider the smallest amount of RAM to occupy when running the DP algorithm on a single computer, as shown in Fig. 3 (right). There are two main types of variables that need to be saved in the sequential decision-making process: one is in the form of integer variables, used to save the optimal transitions $\mathbf{C}^*$ in a two-dimensional array, with the horizontal direction representing the time step and the vertical direction representing the storage combinations. The other is in the form of floating variables, used to save the maximum cumulative returns $\mathbf{F}_t^*$ and $\mathbf{F}_{t+1}^*$ for time steps t and $t+1$ in two one-dimensional arrays, respectively. It should be noted that the maximum cumulative returns at time steps t and $t+1$ are updated until the final time step is reached; that is to say, $\mathbf{F}_{t-1}^*, \mathbf{F}_{t-2}^*, \ldots, \mathbf{F}_1^*$ are no longer saved in the RAM during time step t (see Fig. 3). For the sake of simplicity, we assume the two main variables occupy the same amount of RAM, Φ bytes. Thus the DP algorithm's total RAM amount simply can be expressed as:

$$RAM_1 = m^n \times (T+2) \times \Phi \tag{13}$$

where RAM_1 is the RAM amount running the DP algorithm on one computing process (byte).

From Eqs. (12) and (13), it can be seen that computation time and computer memory are proportional to m^n for a multi-reservoir DP problem, and for this reason the "curse of dimensionality" issue arises.

3.2. Peer-to-peer parallel paradigm

In order to apply the DP algorithm to multi-reservoir system optimization, it is necessary to develop an effective parallelization strategy for the DP algorithm in order to shorten computation time as well as alleviate the RAM bottleneck. The RAM bottleneck would make the DP algorithm un-implementable when a single computing process is used or several computing processes are used on the shared memory architecture.

The purpose of parallelization is to decompose the original task into several subtasks. Moreover, the parallelization distributes the total RAM associated with the task into several subtasks, each of which requires less RAM. Fig. 5 illustrates the way we distribute the above-mentioned two types of variables among K computing processes. We believe that the distribution only can be made along the vertical direction since the DP model features accumulation time step by time step (refer to Eq. (1)). Parallelization is based on the peer-to-peer parallel paradigm, consisting of two types of computing processes, i.e. K peer processes and one transfer process. Each peer process is in charge of sub-allocation of the total RAM. The total number of storage combinations m^n are distributed among K peer processes, with the number of storage combinations m_k allocated to peer process k, $k \in [1, K]$, yielding $m^n = \sum_{k=1}^{K} m_k$. For the sake of clarity, let $\mathbf{C}^* = [\mathbf{C}_1^*, \ldots, \mathbf{C}_k^*, \ldots, \mathbf{C}_K^*]^T$ and $\mathbf{F}_t^* = [\mathbf{F}_{1,t}^*, \ldots, \mathbf{F}_{k,t}^*, \ldots, \mathbf{F}_{K,t}^*]^T$, as shown in Fig. 5.

Parallelization should consider the concurrency and dependency among subtasks undertaken by the K peer processes. Here, the term "concurrency" refers several subtasks that can be executed simultaneously on multiple computing processes. The term "dependency" refers to a computing process that can perform a subtask only after the other computing processes have finished certain subtasks. As far as concurrency is concerned, during time step t the computation of each optimal transition, say $C^*(p_{t+1}, t+1)$, is independent of the others in the DP algorithm;

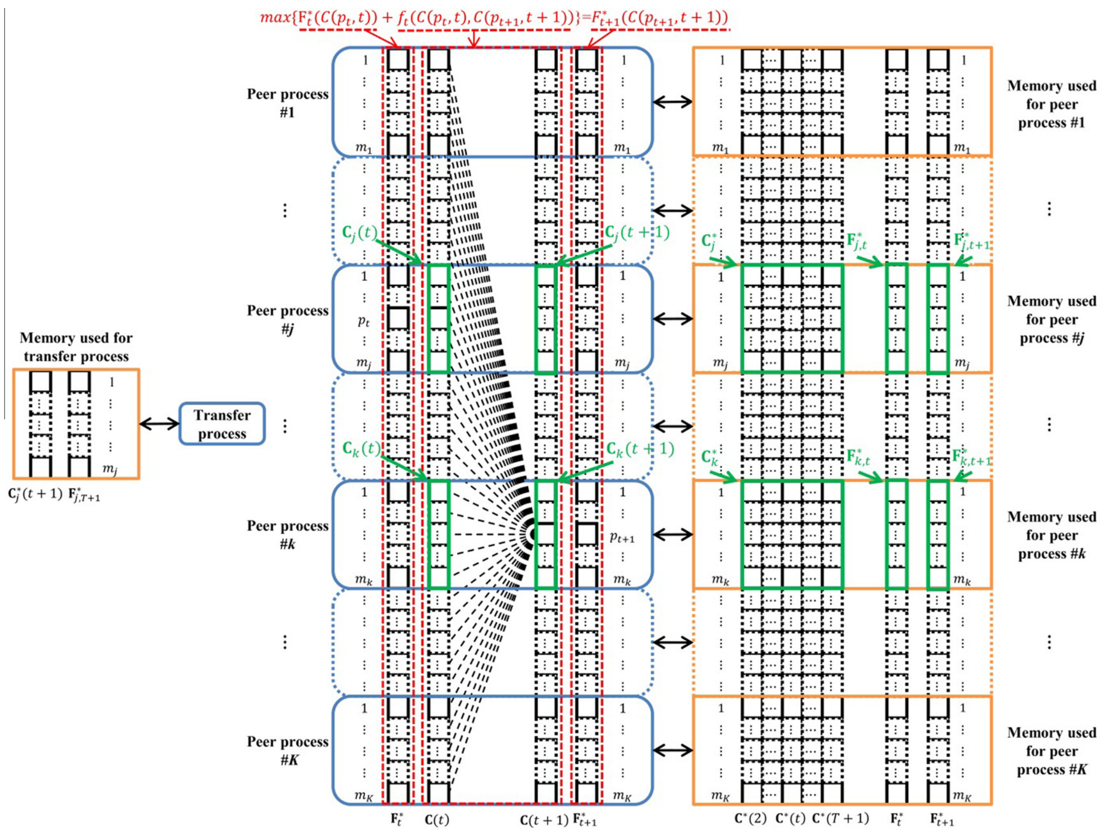

$max\{F_t^*(C(p_t,t)) + f_t(C(p_t,t),C(p_{t+1},t+1))\}=F_{t+1}^*(C(p_{t+1},t+1))$
Peer process #1
Peer process #j
Peer process #k
Peer process #K
Memory used for transfer process
Transfer process
$C_j^*(t+1)$ $\mathbf{F}_{j,T+1}^*$
$C_j(t)$
$C_j(t+1)$
$C_k(t)$
$C_k(t+1)$
p_t
p_{t+1}
$\mathbf{F}_t^*$ $\mathbf{C}(t)$ $\mathbf{C}(t+1)$ $\mathbf{F}_{t+1}^*$
$C^*(2)$ $C^*(t)$ $C^*(T+1)$ $\mathbf{F}_t^*$ $\mathbf{F}_{t+1}^*$
$\mathbf{C}_j^*$ $\mathbf{F}_{j,t}^*$ $\mathbf{F}_{j,t+1}^*$
$\mathbf{C}_k^*$ $\mathbf{F}_{k,t}^*$ $\mathbf{F}_{k,t+1}^*$
Memory used for peer process #1
Memory used for peer process #j
Memory used for peer process #k
Memory used for peer process #K

Fig. 5. Schematic illustration of computing procedures and computer memory used for the parallel DP algorithm.

that is, the computations of all $C^*(p_{t+1}, t+1)$ $p_{t+1} = 1, 2, \ldots, m^n$ can be executed simultaneously. As far as dependency is concerned, to compute $C^*(p_{t+1}, t+1)$ during time step t, all optimal transition $C^*(p_t, t)$ $p_t = 1, 2, \ldots, m^n$ and maximum cumulative returns $F_t^*(C(p_t, t))$ $p_t = 1, 2, \ldots, m^n$, which are saved among K peer processes in this paradigm (see Fig. 5), should be known prior. In other words, a peer process cannot accomplish the computation for $C^*(p_{t+1}, t+1)$ unless all peer processes have finished all computations for $C^*(p_t, t)$ $p_t = 1, 2, \ldots, m^n$ and $F_t^*(C(p_t, t))$ $p_t = 1, 2, \ldots, m^n$. Thus there is an underlying synchronization during each time step in this paradigm.

The two processes have basic functions. Each peer process evaluates a subtask of the current maximum cumulative returns and optimal transitions based on the previous maximum cumulative returns completed by the K peer processes and after the evaluations the subtask of current maximum cumulative returns and the optimal transitions are saved in the RAM. The transfer process is in charge of communicating maximum cumulative returns among all peer processes and tracing back the optimal path.

Figs. 6 and 7 show the two procedures (*Procedure I* and *Procedure II*) of the parallel DP algorithm. Unlike the procedures of serial DP algorithm in Fig. 4, some inter-process communicating statements are added between the peer processes and the transfer process.

Fig. 6 presents the flowchart of peer process k and the transfer process of *Procedure I* of the parallel DP algorithm. Peer process j is any one of the K peer processes that has the same workflows as peer process k, and peer process k or the transfer process receive the message of peer process j based on the "first come, first served" principle; and w is a counter variable. The more specific workflows of *Procedure I* of the parallel DP algorithm include the following steps:

(1) Start to allocate a sub-allocation of the total RAM for all processes;
(2) When $t = 1$, for each peer process, say peer process k, initialize $\mathbf{F}_{k,2}^*$ and $\mathbf{C}_k^*(2)$ under the given $F_1^*(\bullet) = 0$ and $C(p_1, 1)$;
(3) Each peer process, say peer process k, sends $\mathbf{F}_{k,t+1}^*$ to the transfer process at time step t;
(4) The transfer process receives $\mathbf{F}_{j,t+1}^*$ from a certain peer process j based on the principle of "first come, first served" and sends $\mathbf{F}_{j,t+1}^*$ to all the peer processes;
(5) Repeat Step (4) until $\mathbf{F}_{j,t+1}^*$ from each of the peer processes (i.e. $j = 1, \ldots, K$) is received and sent by the transfer process;
(6) Each peer process, say peer process k, receives $\mathbf{F}_{j,t}^*$ at time step $t+1$ (i.e. $\mathbf{F}_{j,t+1}^*$ at time step t) from the transfer process based on the principle of "first come, first served" and carries-out the recursive equation to compute and update $\mathbf{F}_{k,t+1}^*$ and $\mathbf{C}_k^*(t+1)$ based on the sums of the objective function resulting from the transitions from $\mathbf{C}_j(t)$ to $\mathbf{C}_k(t+1)$ and $\mathbf{F}_{j,t}^*$;
(7) Repeat Step (6) until $\mathbf{F}_{j,t}^*$ is received K times from the transfer process (i.e. $j = 1, \ldots, K$);
(8) Repeat Steps (3)–(7) until the final time step T is reached.

After *Procedure I*, each peer process, say peer process k, saves $\mathbf{C}_k^*$, $\mathbf{F}_{k,T}^*$ and $\mathbf{F}_{k,T+1}^*$ in its RAM. Then, *Procedure II* of the parallel DP algorithm begins, as shown in Fig. 7, and the workflows include the following steps:

(1) Each peer process, say peer process k, sends $\mathbf{F}_{k,T+1}^*$ and $\mathbf{C}_k^*(T+1)$ to the transfer process;
(2) The transfer process receives $\mathbf{F}_{j,T+1}^*$ and $\mathbf{C}_j^*(T+1)$ from a certain peer process j based on the principle of "first come, first served" and compares the elements in $\mathbf{F}_{j,T+1}^*$ and updates the maximum $F_{T+1}^*(C^*(p_{T+1}, T+1))$ and consequent $C^*(p_{T+1}, T+1)$;

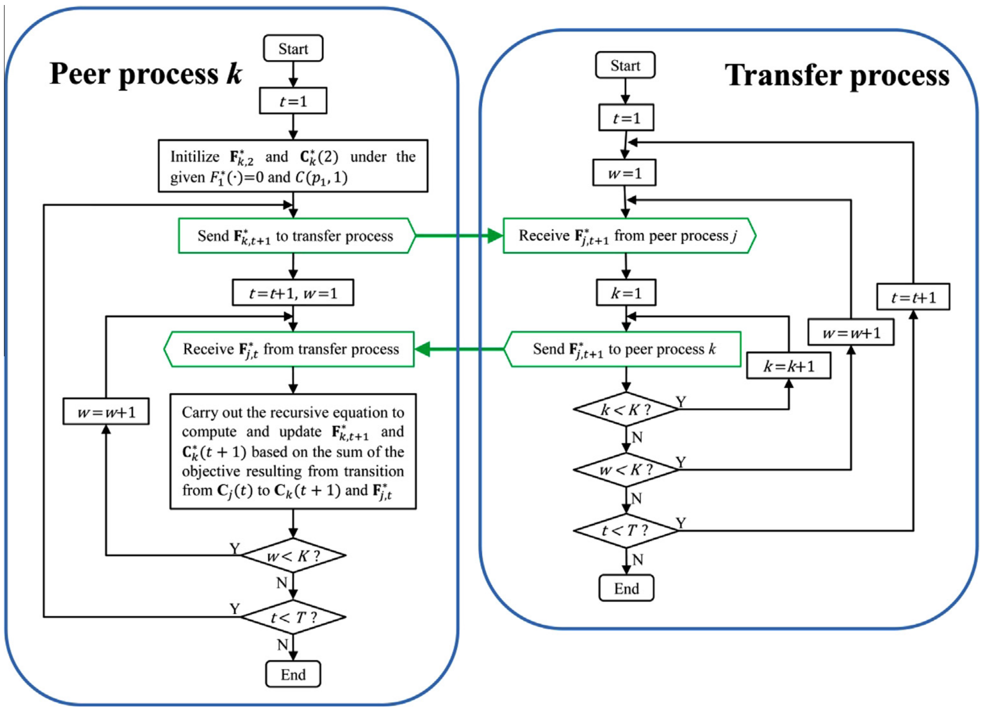

Peer process k
Start
t=1
Initilize $\mathbf{F}^*_{k,2}$ and $\mathbf{C}^*_k(2)$ under the given $F^*_1(\cdot)$=0 and $C(p_1,1)$
Send $\mathbf{F}^*_{k,t+1}$ to transfer process
t=t+1, w=1
Receive $\mathbf{F}^*_{j,t}$ from transfer process
w=w+1
Carry out the recursive equation to compute and update $\mathbf{F}^*_{k,t+1}$ and $\mathbf{C}^*_k(t+1)$ based on the sum of the objective resulting from transition from $\mathbf{C}_j(t)$ to $\mathbf{C}_k(t+1)$ and $\mathbf{F}^*_{j,t}$
$w<K$?
$t<T$?
Y
N
End
Transfer process
Start
t=1
w=1
Receive $\mathbf{F}^*_{j,t+1}$ from peer process j
k=1
Send $\mathbf{F}^*_{j,t+1}$ to peer process k
k=k+1
w=w+1
t=t+1
$k<K$?
$w<K$?
$t<T$?
End

Fig. 6. Flowchart of peer process k and transfer process of *Procedure I* of the parallel DP algorithm.

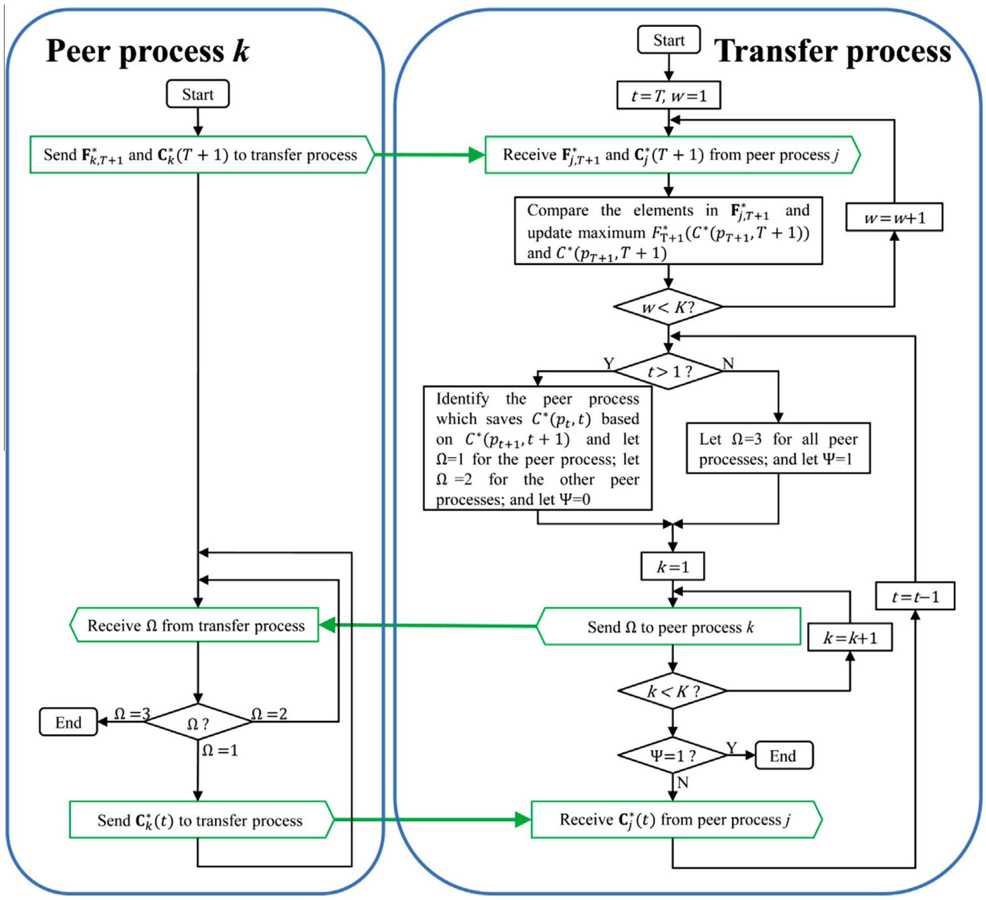

Peer process k
Start
Send $\mathbf{F}^*_{k,T+1}$ and $\mathbf{C}^*_k(T+1)$ to transfer process
Receive Ω from transfer process
Ω ?
Ω =3
Ω =2
Ω =1
End
Send $\mathbf{C}^*_k(t)$ to transfer process
Transfer process
Start
t=T, w=1
Receive $\mathbf{F}^*_{j,T+1}$ and $\mathbf{C}^*_j(T+1)$ from peer process j
Compare the elements in $\mathbf{F}^*_{j,T+1}$ and update maximum $F^*_{T+1}(C^*(p_{T+1},T+1))$ and $C^*(p_{T+1},T+1)$
w=w+1
$w<K$?
$t>1$?
Y
N
Identify the peer process which saves $C^*(p_t,t)$ based on $C^*(p_{t+1},t+1)$ and let Ω=1 for the peer process; let Ω =2 for the other peer processes; and let Ψ=0
Let Ω=3 for all peer processes; and let Ψ=1
k=1
Send Ω to peer process k
k=k+1
t=t−1
$k<K$?
Ψ=1 ?
End
Receive $\mathbf{C}^*_j(t)$ from peer process j

Fig. 7. Flowchart of peer process k and transfer process of *Procedure II* of the parallel DP algorithm.

(3) Repeat Step (2) until $\mathbf{F}^*_{j,T+1}$ and $\mathbf{C}^*_j(T+1)$ from all peer processes (i.e. $j = 1,\ldots,K$) are received and the maximum $F^*_{T+1}(C^*(p_{T+1},T+1))$ and consequent $C^*(p_{T+1},T+1)$ are derived;
(4) If $t > 1$, go to Step (5); otherwise go to Step (10);
(5) Identify the peer process that saves $C^*(p_t,t)$ based on $C^*(p_{t+1},t+1)$ and let $\Omega = 1$ (where Ω is used to judge which choice the peer process should make) for the peer process, $\Omega = 2$ for the other peer processes, and $\Psi = 0$ (where Ψ is used to judge whether to end the transfer process);
(6) The transfer process sends Ω to all the peer processes;
(7) Each peer process, say peer process k, receives Ω from the transfer process. If $\Omega = 1$, the peer process should send $\mathbf{C}^*_k(t)$ to the transfer process and then repeat to receive Ω for the next time step. If $\Omega = 2$, the peer process directly repeats to receive Ω for the next time step;
(8) The transfer process receives the required $\mathbf{C}^*_j(t)$, which saves the $C^*(p_t,t)$ at time step t (i.e. $C^*(p_{t+1},t+1)$ at time step $t-1$) from the required peer process, say peer process j;
(9) Repeat Steps (4)–(8);
(10) Let $\Omega = 3$ for all peer processes and let $\Psi = 1$;
(11) Each peer process ends its process, once receive $\Omega = 3$ from the transfer process;
(12) The transfer process ends its process when $\Psi = 1$.

The optimal path is derived by the transfer process with *Procedure II*. Furthermore, the consequent storages and releases of multiple reservoirs also can be determined for all time steps with the derived optimal path.

We consider a straightforward workload balancing strategy to decrease the idle computing processes and further obtain good performance on parallel efficiency. The strategy is to approximately allocate the same amount of subtask to each peer process. Thus the number of storage combinations saved in a peer process is defined as:

$$\begin{cases} m_k = \alpha + 1 & k \leqslant \beta \\ m_k = \alpha & \beta < k \leqslant K \end{cases} \tag{14}$$

where

$$\alpha = u(m^n/K) \tag{15}$$

$$\beta = v(m^n, K) \tag{16}$$

and where $u(\bullet)$ is the floor integer function (e.g. $u(5/3) = 1$) and $v(\bullet)$ is the remainder function (e.g. $v(5,3) = 2$). However, we note that the discarded infeasible solutions (see Section 3.1) still result in imbalance tasks assigned to all peer processes.

The computation time of using the parallel DP algorithm is estimated as:

$$\tau_K = (\tau' + \tau'' + \tau''')/K \tag{17}$$

where τ_K is the wall clock time of using K peer processes, consisting of the computation time fraction τ', the communication time fraction τ'' and the workload imbalance time cost τ'''. We employ the parallel efficiency E_K as a measurement to evaluate the parallel performance, calculated as:

$$E_K = \tau_1/(K \times \tau_K) \tag{18}$$

The parallel efficiency of a parallel program is dependent on the ratio between the communication time and the computation time. If the ratio is small, the parallel efficiency is high; otherwise the parallel efficiency is low.

The RAM of the parallel DP algorithm allocated to each peer process, say peer process k, is estimated as:

$$RAM_k = [m^n \times (T+2) \times \Phi]/K \tag{19}$$

where RAM_k is the RAM amount for each peer process (byte). For a system of distributed memory architecture, such as Fig. 1, all computing processes in a blade share RAM. Suppose Θ computing processes share RAM whose size is *RAM* bytes. When using the parallel DP algorithm, we roughly can determine the upper bound of RAM by the following equation:

$$[m^n \times (T+2) \times \Phi]/K \times \Theta \leqslant RAM \tag{20}$$

From Eq. (20), we see that the RAM requirement for a multi-reservoir DP problem can be alleviated by the number of computing processes (K). This is a breakthrough in that multi-reservoir system optimization problems that previously could not be solved by a serial DP algorithm on a single computer because of RAM requirements now can be solved with parallel computing. However, we note that an increase in reservoir number will result in an increase in computing process demand.

4. The four-reservoir example

4.1. Problem description

We first apply the developed parallel DP algorithm to the classic, hypothetical four-reservoir problem (see Fig. 8). This problem has been studied in the literature by several researchers. Larson [15] solved the problem by IDP. Heidari et al. [9] solved the same problem by DDDP. More recently, Wardlaw and Sharif [34] used a genetic algorithm while Kumar and Reddy [12] used particle swarm optimization to evaluate the performance of their heuristic techniques.

From Fig. 8, we see that the reservoir system consists of both series and parallel connections ($L = 1$, $U_1 = [1,3]$). The reservoir system connectivity matrix is:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ -1 & 0 & -1 & 1 \end{bmatrix} \tag{21}$$

The inflows are assumed to be constant for the entire operating period and are set as:

$$\mathbf{I} = \begin{bmatrix} 2 \\ 3 \\ 0 \\ 0 \end{bmatrix} \tag{22}$$

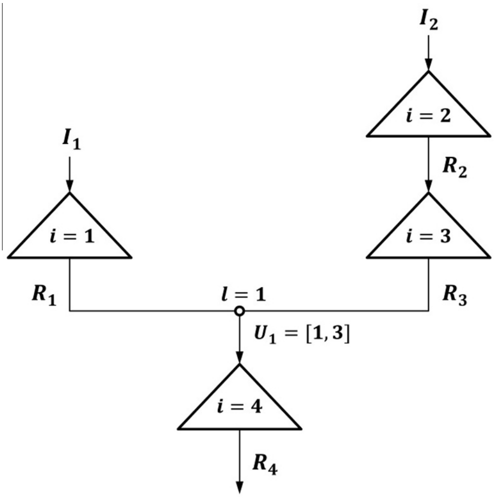


Fig. 8. The four-reservoir problem.

The main objectives of the system operation are to maximize the benefits from hydropower generation from reservoirs $i = 1, 2, 3, 4$ as well as the benefits from irrigation from reservoir $i = 4$ over 12 time steps ($n = 4$, $T = 12$). We expand the objective function $f_t(\bullet)$ to be maximized during time step t to:

$$f_t(\bullet) = \begin{cases} \sum_{i=1}^{4} b_i(t) \bullet R_i(t) + b_5(t) \bullet R_4(t) & t < 12 \\ \sum_{i=1}^{4} b_i(t) \bullet R_i(t) + b_5(t) \bullet R_4(t) + \sum_{i=1}^{4} g_i(S_i(13), d_i) & t = 12 \end{cases} \tag{23}$$

where $R_i(t)$ ($i = 1, 2, 3, 4$) can be computed directly from Eq. (2) once $S_i(t)$ and $S_i(t + 1)$ ($i = 1, 2, 3, 4$) are chosen; $b_i(t)$ is the benefit function (refer to [15]); and $g_i(\bullet)$ is the penalty function for not meeting the ending storage constraint (d_i):

$$g_i(S_i(13), d_i) = \begin{cases} -40 \bullet (S_i(13) - d_i)^2 & S_i(13) \leqslant d_i \\ 0 & S_i(13) > d_i \end{cases} \tag{24}$$

Other constraints include:

$$\mathbf{S}^{\min} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{S}^{\max} = \begin{bmatrix} 10 \\ 10 \\ 10 \\ 15 \end{bmatrix} \tag{25}$$

$$\mathbf{S}^{initial} = \begin{bmatrix} 5 \\ 5 \\ 5 \\ 5 \end{bmatrix}, \quad \mathbf{S}^{final} = \begin{bmatrix} 5 \\ 5 \\ 5 \\ 7 \end{bmatrix} \tag{26}$$

$$\mathbf{R}^{\min} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{R}^{\max} = \begin{bmatrix} 3 \\ 4 \\ 4 \\ 7 \end{bmatrix} \tag{27}$$

Note that Eqs. (7)–(9) are inactive in this example.

4.2. Results and discussion

In this example, the storage in each reservoir is discretized with $\Delta S = 1$ unit, and thus the total number of storage combinations is $11 \times 11 \times 11 \times 16 = 21{,}296$. We solved the four-dimensional DP model on the HPC system described above. First, we applied the serial DP algorithm with a single computing process. The global optimum is 401.3. The wall clock time (T_1) was 1818.6 s. Then we performed several executions using the developed parallel DP algorithm by varying the peer process numbers. As expected, all executions produced the same global optimum, except with different wall clock times. The computation time fraction τ', the sum of communication time fraction and workload imbalance time cost $\tau'' + \tau'''$, the wall clock time τ_K and the parallel efficiency E_K are presented in Table 1. As we can see, the wall clock time T_K is drastically reduced, from 1818.6 s to 9.7 s with 350 peer processes. This reduction is quite substantial.

Several probable reasons affect the parallel efficiency:

(1) The hyper-threading technology. (a) If there are one peer process and one transfer process, we can conclude from the result that the OS schedules the two processes on one physical core. The peer process does not compete with the transfer process for execution resources, and thus the computation time fraction is almost the same with the serial DP algorithm. (b) If there are two peer processes and one transfer process, we conclude that one peer process and the transfer process are scheduled on one physical core, while the other peer process is scheduled on the other physical core. Because the three processes do not compete for execution resources, the parallel efficiency is as high as 0.99. (c) If there are three peer processes and one transfer process, there would be two peer processes scheduled on the same physical core. Because the two peer processes compete for execution resources, the parallel efficiency is reduced to 0.72. (d) By this reasoning, parallel efficiencies increase when there are even numbers of peer processes and decrease when there are odd numbers of peer processes. However, as the number of peer processes becomes large, the fluctuations are not obvious.

(2) The workload imbalance. From Fig. 9, we see that as the number of peer processes increases, the sum of communication time fraction and workload imbalance time cost increases, then decreases, and then increases again. This is because when the number of peer processes is small, the workload imbalance (because various numbers of infeasible solutions assigned to the peer processes will be discarded, [see Section 3.1]) is significant, and the fast peer processes must wait until the slowest peer process finishes its task. However, when the number of peer processes becomes large, the amounts of task assigned to peer processes decreases and the influence of workload imbalance diminishes. Although there are still imbalance tasks among various peer processes, the fast peer processes do not have to wait long. Because the sum of communication time fraction and workload imbalance time cost increases linearly as the number of peer processes increases, we can infer that the developed parallel DP algorithm is scalable and not restricted by the increase in the number of cores.

(3) The turbo boost technology (referred to as dynamic overclocking). The clock rate depends on the CPU's thermal limit, the number of cores in use as well as the maximum frequency of the active cores. If the CPU is below its thermal limits, the operating frequency will increase; otherwise the operating frequency will be fixed on the standard frequency.

Table 1
The computation time fraction, the sum of communication time fraction and workload imbalance time cost, the wall clock time and the parallel efficiency of the parallel DP algorithm for the classic four-reservoir problem.

No. of peer processes	τ' (s)	$\tau'' + \tau'''$ (s)	τ_K (s)	E_K
Serial	1818.6	–	–	1.00
1	1836.2	0.0	1836.2	0.99
2	1833.8	1.3	917.5	0.99
3	2427.5	100.6	842.7	0.72
4	2226.3	123.4	587.4	0.77
5	2595.5	115.2	542.1	0.67
6	2373.6	181.3	425.8	0.71
7	2688.3	140.5	404.1	0.64
8	2510.6	194.1	338.1	0.67
9	2732.9	132.6	318.4	0.63
10	2593.3	153.7	274.7	0.66
25	2948.3	90.9	121.6	0.60
50	2987.1	60.6	61.0	0.60
75	3016.1	33.3	40.7	0.60
100	3036.3	93.7	31.3	0.58
125	3073.2	75.3	25.2	0.58
150	3058.1	106.0	21.1	0.57
175	3066.6	140.8	18.3	0.57
200	3059.8	134.0	16.0	0.57
225	3115.4	147.1	14.5	0.56
250	3087.7	197.5	13.1	0.55
275	3106.8	210.5	12.1	0.55
300	3113.3	216.7	11.1	0.55
325	3080.3	250.6	10.2	0.55
350	3137.0	243.0	9.7	0.54

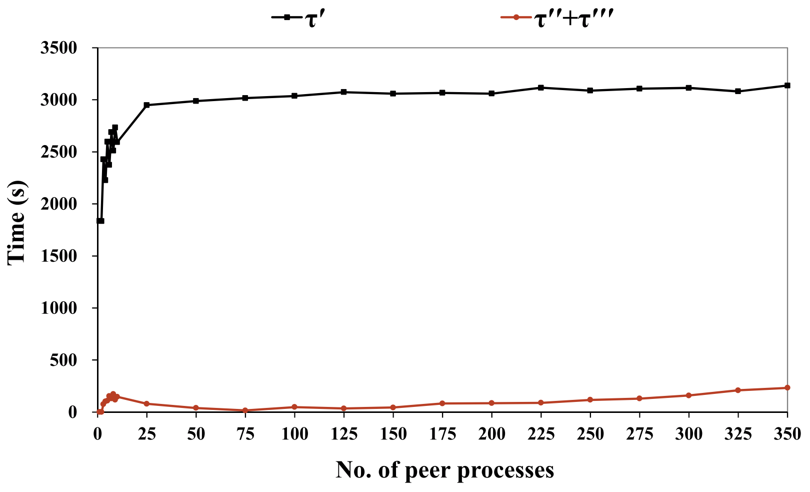


Fig. 9. Computation time fraction versus the sum of communication time fraction and workload imbalance time cost.

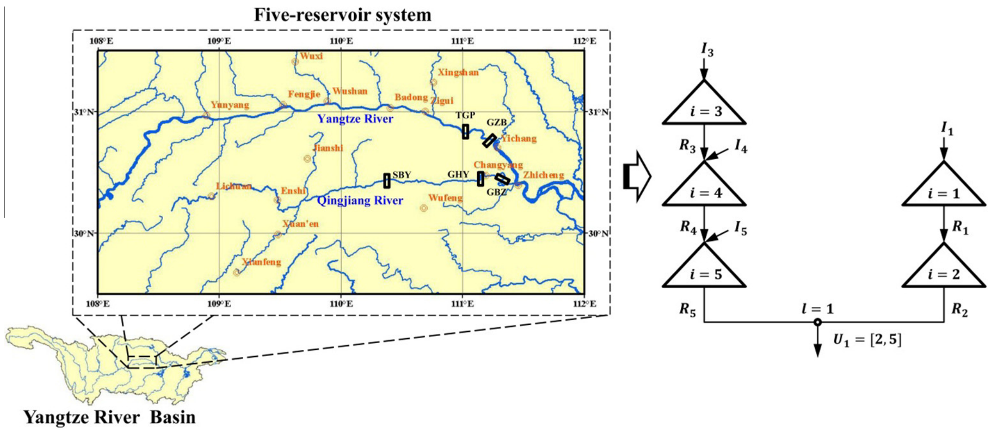


Fig. 10. The five-reservoir system.

5. A real-world application

5.1. Problem description

We now apply the developed parallel DP algorithm to a real-world reservoir system located in the central Yangtze River Basin, China (see Fig. 10). The system consists of five reservoirs ($n = 5$): the Three Gorges Project (TGP) and the Gezhouba (GZB) on the Yangtze River; and the Shuibuya (SBY), the Geheyan (GHY) and the Gaobazhou (GBZ) on the Qingjiang River (which joins the Yangtze River at the town of Zhicheng). As shown in Fig. 10 (right), these reservoirs are numbered from $i = 1$ to $i = 5$ and the river confluence point is denoted as $l = 1$ ($U_1 = [2, 5]$). The natural inflow between the TGP and the GZB can be ignored because of their close proximity. At present, all five reservoirs are operated by two corporations. The TGP and GZB cascade hydropower plants (TGP–GZB) are managed by the China Three Gorges Corporation, while the SBY, GHY and GZB cascade hydropower plants (SBY–GHY–GBZ) are under the jurisdiction of the Hubei Qingjiang Hydroelectric Development Co., Ltd. The joint operation of this five-reservoir system is of major interest to the Ministry of Science and Technology of China.

In this reservoir system, the SBY is a multi-year storage reservoir with $24.0 \times 10^8\ \mathrm{m}^3$ of active storage capacity; the GHY is a yearly storage reservoir with $11.5 \times 10^8\ \mathrm{m}^3$ of active storage capacity; the TGP is a seasonal storage reservoir with $221.5 \times 10^8\ \mathrm{m}^3$ of active storage capacity; and the GZB and GBZ are daily storage reservoirs with active storage capacities $0.8 \times 10^8\ \mathrm{m}^3$ and $0.5 \times 10^8\ \mathrm{m}^3$, respectively – much smaller than the other three reservoirs. The main characteristics and the operating rules of the five reservoirs are listed in Table 2, where $(S^{\max} - S^{\min})$ denotes the active storage capacity; $H(\bullet)$ is the conversion function from reservoir storage to water level; $N^{\min}$ denotes the guaranteed output production of a hydropower plant; and $N^{\max}$ denotes the installed capacity of a hydropower plant.

Table 2
Main characteristics and operating rules of the five reservoirs.

Reservoir	TGP	GZB	SBY	GHY	GBZ
$S^{\min}$ (10^8 m^3)	171.5	6.3	19.0	18.7	3.5
$S^{\max}$ (10^8 m^3)	393.0	7.1	43.0	30.2	4.0
$(S^{\max} - S^{\min})$ (10^8 m^3)	221.5	0.8	24.0	11.5	0.5
$H(S^{\min})$ (m)	145.0	63.0	350.0	180.0	78.0
$H(S^{\max})$ (m)	175.0	66.0	400.0	200.0	80.0
$R^{\min}$ (m^3/s)	6000	6000	0	0	0
$R^{\max}$ (m^3/s)	101,700	113,400	13,200	18,000	18,400
$N^{\min}$ (MW)	4990	1040	310	241.5	77.3
$N^{\max}$ (MW)	22,400	2757	1840	1212	270

Table 3
Three scenarios used in the real-world problem.

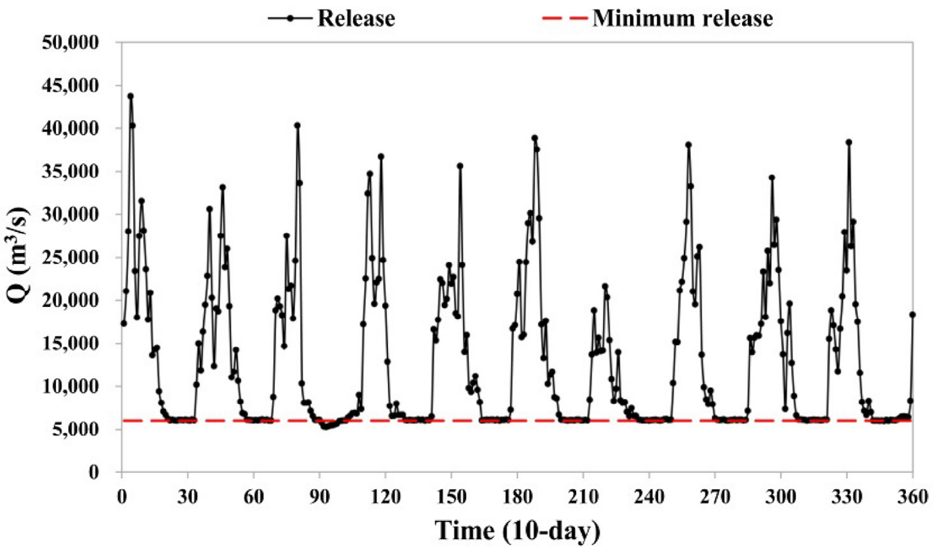
Scenario	System	PR^{min} (m^3/s)	PR^{max} (m^3/s)	N^{min} (MW)	N^{max} (MW)
1	TGP–GZB	6000	56,700	6030	20,957
2	SBY–GHY–GBZ	0	18,400	628.8	3322
3	The five-reservoir system	6000	56,700	6658.8	24,279

Table 4
The optimal energy generation from the three scenarios (10^8 KW h).

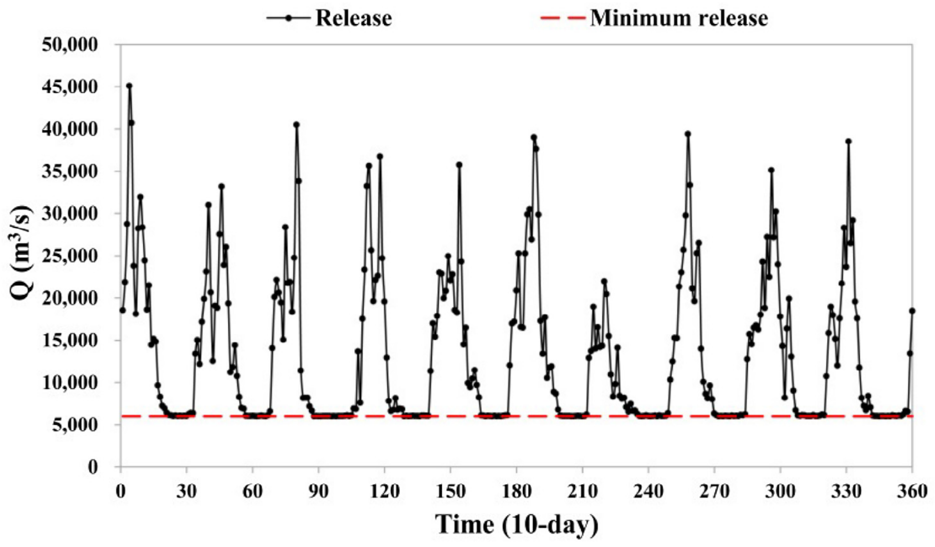
Scenario	TGP	GZB	SBY	GHY	GBZ	Total
1	8364.9	1528.8	–	–	–	9893.7
2	–	–	330.5	274.9	87.8	693.2
3	8411.9	1525.0	342.1	271.5	86.0	10636.5
(3-1-2)*	47.0	−3.8	11.6	−3.4	−1.8	49.6

* *Note*: (3-1-2) denotes the scenario 3 minus scenario 1 minus scenario 2.

We use three scenarios for the case study, as shown in Table 3, where PR^{min} and PR^{max}, respectively, denote the mandatory release and the maximum allowable release at river confluence point $l = 1$. Scenario 1 is built for the TGP–GZB; scenario 2 is for the SBY–GHY–GBZ; and scenario 3 encompasses the entire five-reservoir system. Scenario 3 is conducted under the assumption that the five-reservoir system is under joint operation. To test our proposed methodology, we further assume that energy generation is transmitted to the same grid. This assumption may differ slightly from what actually occurs in practice. Furthermore, for confluence point $l = 1$, the

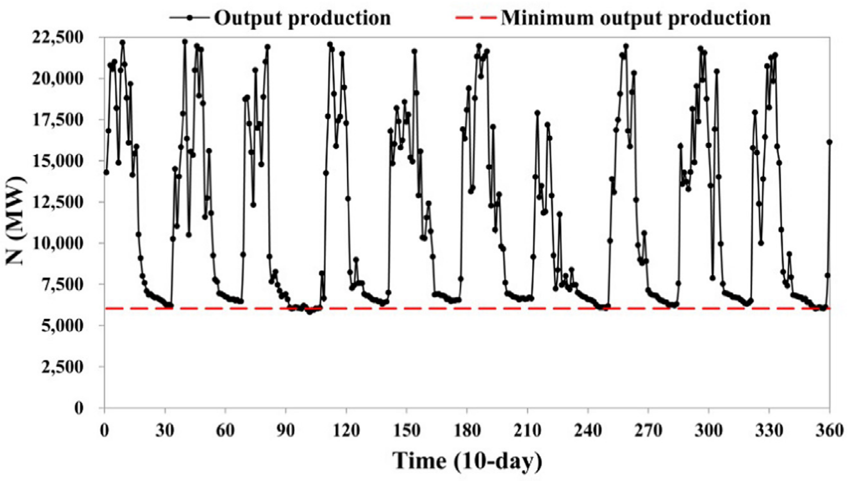


(a) Release at river confluence point $l = 1$ from scenario 1.

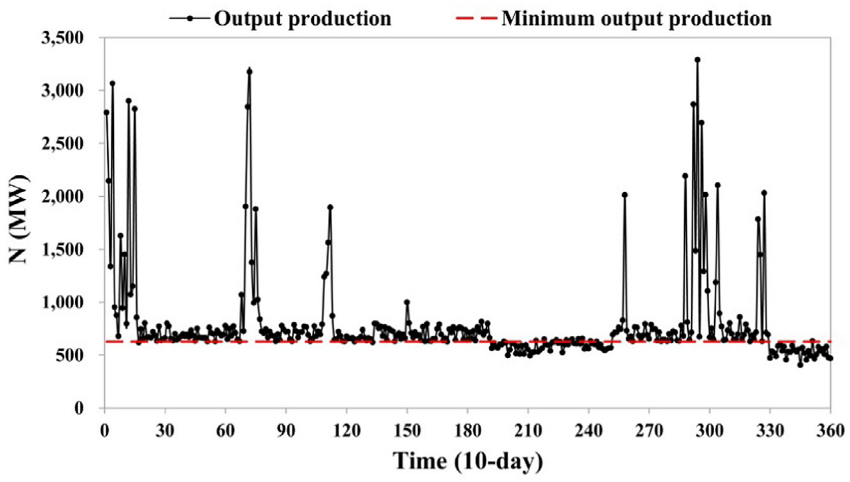


(b) Release at river confluence point $l = 1$ from scenario 3.

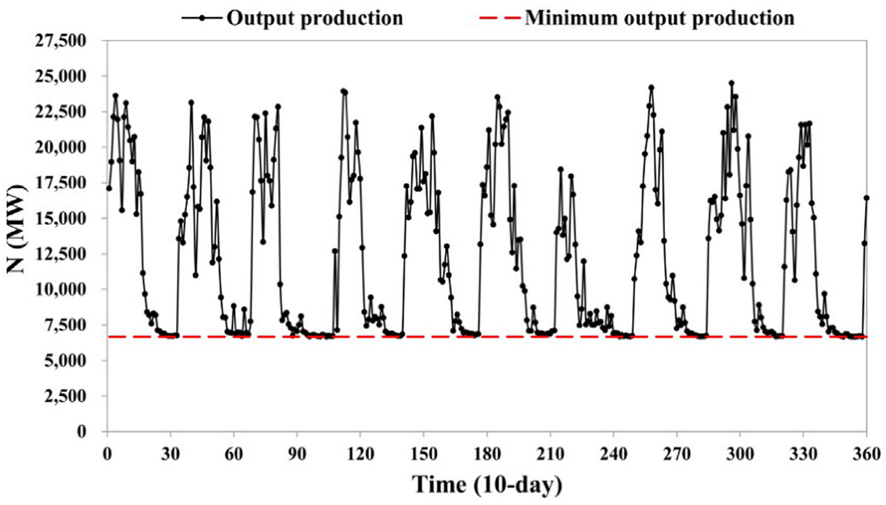
Fig. 11. Comparison of releases at river confluence point $l = 1$, scenarios 1 versus 3.



(a) Output production from scenario 1.



(b) Output production from scenario 2.



(c) Output production from scenario 3.

Fig. 12. Comparison of output productions, scenarios 1 and 2 versus scenario 3.

mandatory release is 6000 m^3/s for both navigational and ecological purposes, and the maximum allowable release is 56,700 m^3/s for downstream flood protection.

The objectives of the three scenarios are to maximize energy generation over a recent 10-year horizon (June 2000 through May 2010). The operating period is divided into 360 time steps

Table 5
The wall clock time and the parallel efficiency of the parallel DP algorithm for the five-reservoir system.

No. of peer processes	Serial	50	100	150	200	250	300	350
τ_K (h)	266.83	10.11	5.17	3.51	2.61	2.12	1.77	1.54
E_K	1.00	0.53	0.52	0.51	0.52	0.50	0.50	0.50

(each with a length of about 10 days and $T = 360$). We expand the objective function $f_t(\bullet)$ to be maximized during time step t to:

$$f_t(\bullet) = \sum_i^I N_i(t) \bullet \Delta t = \sum_i^I 9.81 \bullet \eta_i \bullet R_i'(t) \bullet \overline{H}_i(t) \times \Delta t \quad \forall t \tag{28}$$

where

$$R_i(t) = R_i'(t) + R_i''(t) \quad \forall t \tag{29}$$

$$\overline{H}_i(t) = HF_i(t) - HT_i(t) \quad \forall t \tag{30}$$

η_i is the hydropower plant efficiency at reservoir i; $R_i'(t)$ and $R_i''(t)$ are the power release and non-power release from reservoir i during time step t; $\overline{H}_i(t)$ is the average head at reservoir i during time step t, which is the difference between the average reservoir forebay water level $HF_i(t)$ (as a function of beginning and ending time period reservoir storages) and the tail-race water level $HT_i(t)$ (as a function of total release) at reservoir i during time step t; and Δt is the time interval. In this case, each reservoir operates under its own constraints (see Table 2), while the reservoir system operates under the system constraints (see Table 3).

5.2. Results and discussion

In the real-world case, the GZB and GBZ are treated as run-of-river hydropower plants because of their much smaller storages compared with the TGP, SBY and GHY (see Table 2), which are operated as storage reservoirs. The ratio of active storages of the three reservoirs is approximately 20:2:1 (i.e. 221.5×10^8 m^3: 24.0×10^8 m^3: 11.5×10^8 m^3), and thus we discretize them into 200, 20 and 10 levels, respectively. The total numbers of storage combinations are $200 \times 1 = 200$, $20 \times 10 = 200$ and $200 \times 20 \times 10 = 40{,}000$ for the three scenarios. Similarly, the computation was performed on the HPC system with Oracle 11g serving as the database system for data access.

The optimal energy generations of the three scenarios are shown in Table 4. The sum of optimal energy production of scenario 1 and scenario 2 is $9893.7 + 693.2 = 10586.9 \times 10^8$ KW h, less than the 10636.5×10^8 KW h of scenario 3. This indicates that the coordinated operation of the five reservoirs can result in an average 4.96×10^8 KW h energy production increase (or about \$$1.24 \times 10^8$ CNY increase, assuming the energy price of the five reservoirs is \$0.25 CNY/KW h [18]) per year for the system. The comparisons of releases at river confluence point $l = 1$ and the output productions between scenarios 1 and 2 versus scenario 3 are shown in Figs. 11 and 12, respectively. Obviously, the SBY–GHY–GBZ system helps the TGP–GZB system relieve the stress of water supply demand at river confluence point $l = 1$. The five-reservoir system can provide the grid with more secure and reliable output production.

For scenario 3, the serial DP algorithm took more than 10 days (T_1). Similar to the four-reservoir problem, we performed several executions using the developed parallel DP algorithm by varying the number of peer processes. The computing procedures are completely the same for various executions, and thus the objective function value and the consequent storages and releases are the same. The wall clock time T_K and the parallel efficiency E_K for various numbers of peer processes are shown in Table 5. As we can see, the wall clock time T_K is reduced from 266.83 h to 1.54 h with 350 peer processes. Again, this reduction is quite substantial. It is clear that an increase in the number of peer processes is accompanied by a decrease in wall clock time. Although each peer process is allocated the same approximate amount of subtask, workload imbalance still exists, which may result in a little less efficiency relative to the hypothetical problem.

6. Conclusions

In this paper, we establish a multi-dimensional DP model for optimizing the joint operation of a multi-reservoir system, considering several indispensable constraints for individual reservoirs and the reservoir system. We illustrate the DP algorithm's solution space with the help of a matrix and estimate the smallest RAM required by the DP algorithm, making full preparations for parallelization. We believe an effective parallelization strategy for the DP algorithm should be designed specially to alleviate the RAM bottleneck. For instance, the discretization should be sufficiently fine for an accurate simulation in real-time operation. If we use a discretization level of 10 cm for each of the five reservoirs in Section 5 and 30 days as the inflow forecast period, the RAM requirement will increase to approximately 2.09 TB when using the serial DP algorithm. In this situation, a single computer or the shared memory architecture no longer can meet the large RAM requirement. The distributed memory architecture and the message passing interface (MPI) protocol make it possible for us to develop a parallel DP algorithm that considers both the distributed computing and distributed computer memory, and further, to solve previously unsolvable multi-reservoir DP problems with parallel computing. In the above instance, the RAM requirement will be reduced to approximate $2.09/K$ TB to each peer process, where K is the number of peer processes.

Using the developed parallel DP algorithm based on the peer-to-peer parallel paradigm, we solve the classic four-reservoir problem and a real-world five-reservoir system on a HPC system with up to 350 cores. The results indicate that the wall clock times are reduced drastically when the number of computing processes is increased. In both cases, we observe good performance in parallel efficiency. Furthermore, the real-world problem results indicate that operational effectiveness can be improved and the beneficial uses can be maximized by joint operation of the interconnected five reservoirs. Specifically, (1) the energy production of the system can be increased by an average of 4.96×10^8 KW h per year; (2) the stress of meeting the minimum water supply demand at river confluence point $l = 1$ can be relieved greatly by the inclusion of the SBY–GHY–GBZ system with the TGP–GZB system; and (3) more secure and reliable output productions can be guaranteed and transmitted to the grid.

The Casti et al. [5] paper sends a message of confidence in constructing computers with many processing elements to deal with some future DP problems. Even at the time of that study (i.e. 1970s), NASA's parallel computer had only 64 processing elements. Over last twenty years, however, we have witnessed the rapid development of supercomputing worldwide. According to TOP 500 supercomputer sites' historical lists (http://www.top500.org/), the number of cores in top supercomputers has increased from several thousands to several millions, and computer memory has increased correspondingly. Given such advances, we predict that the barrier in computing resources may be inconsequential in the future. Moreover, we believe the benefits resulting from hydropower generation of a multi-reservoir system far exceed the expense of computing resources. Thus, we hope this paper will stimulate future research on parallel computing in the field of reservoir operation.

Future work should implement the parallel DP algorithm using massively parallel computing resources. Indeed, we have noticed the trend to use massively parallel computing resources for some water resources problems. For instance, Reed and Kollat [26] employed a massively parallel multi-objective evolutionary algorithm for a groundwater monitoring application with a maximum of 8192 processors; while Kollet et al. [11] and Maxwell [20], respectively, performed a ParFlow of various coupled modes utilizing the maximum number of 16,384 processors.

Future work also could introduce the distributed database management technique to further alleviate RAM requirements for multi-reservoir DP problems. This paper distributes the optimal transitions $\mathbf{C}^*$ and maximum cumulative returns $\mathbf{F}_t^*$ and $\mathbf{F}_{t+1}^*$ and saves them in RAMs of several computing processes. Future work could distribute and save them in several databases in various computers. This would take advantage of large hard disk capacities and access the required data from the database of identified computer when the DP algorithm solves the recursive equation and traces back the optimal path.

Finally, the developed parallel DP algorithm easily can be applied to other DP-based variants, such as SDP, IDP and DDDP.

Notation

t	time index, $t \in [1, T]$
i	reservoir index, $i \in [1, n]$
l	river confluence point index, $l \in [1, L]$
k	peer process index, $k \in [1, K]$
$\mathbf{S}(t)$	reservoir storage vector at the beginning of time step t, $\mathbf{S}(t) = [S_1(t), \ldots, S_i(t), \ldots, S_n(t)]^T$
$\mathbf{S}(t+1)$	reservoir storage vector at the end of time step t
$\mathbf{S}^{\text{initial}}$	initial reservoir storage vector
$\mathbf{S}^{\text{final}}$	final expected reservoir storage vector
$\mathbf{S}^{\min}(t+1)$	minimum reservoir storage vector at the end of time step t
$\mathbf{S}^{\max}(t+1)$	maximum reservoir storage vector at the end of time step t
$F_t^*(\bullet)$	maximum cumulative return from the first time step to the beginning of the tth time step resulting from the joint operation of n reservoirs
$f_t(\bullet)$	objective function to be maximized during time step t
$\mathbf{I}(t)$	inflow vector during time step t
$\mathbf{R}(t)$	total release vector during time step t, $\mathbf{R}(t) = [R_1(t), \ldots, R_i(t), \ldots, R_n(t)]^T$
$\mathbf{R}^{\min}(t)$	minimum required release vector during time step t
$\mathbf{R}^{\max}(t)$	maximum allowable release vector during time step t
η_i	hydropower plant efficiency at reservoir i
$R_i'(t)$	power release from reservoir i during time step t
$R_i''(t)$	non-power release from reservoir i during time step t
$\overline{H}_i(t)$	average head at reservoir i during time step t, which is equal to the average reservoir fore-bay water level $HF_i(t)$ minus the tail-race water level $HT_i(t)$
Δt	time interval
$\mathbf{M}$	reservoir system connectivity matrix
U_l	set of reservoirs in parallel at river confluence point l
$PR_l^{\min}(t)$	minimum required release at river confluence point l during time step t
$PR_l^{\max}(t)$	maximum allowable release at river confluence point l during time step t
$\mathbf{N}(t)$	output vector during time step t, $\mathbf{N}(t) = [N_1(t), \ldots, N_i(t), \ldots, N_n(t)]^T$
$\mathbf{N}^{\min}(t)$	minimum required output vector during time step t
$\mathbf{N}^{\max}(t)$	maximum allowable output vector during time step t
$N^{\min}(t)$	minimum required output from a reservoir system during time step t
$N^{\max}(t)$	maximum allowable output from a reservoir system during time step t
$\mathbf{C}$	possible storage combination $m^n \times T$ matrix, $\mathbf{C} = [\mathbf{C}(2), \ldots, \mathbf{C}(t), \ldots, \mathbf{C}(T+1)]$
$\mathbf{C}^*$	optimal transitions, $\mathbf{C}^* = [\mathbf{C}_1^*, \ldots, \mathbf{C}_k^*, \ldots, \mathbf{C}_K^*]^T$
$\mathbf{F}_t^*$	maximum cumulative returns from the first time step to the beginning of the tth time step, $\mathbf{F}_t^* = [\mathbf{F}_{1,t}^*, \ldots, \mathbf{F}_{k,t}^*, \ldots, \mathbf{F}_{K,t}^*]^T$
m	number of discretization levels of each reservoir
m_k	allocation number of storage combinations to peer process k
τ_1	wall clock time of using 1 computing process
τ_K	wall clock time of using K peer processes
$\Delta\tau$	average wall clock time for one-time objective function evaluation
τ'	computation time fraction
τ''	communication time fraction
τ'''	workload imbalance time cost
E_K	parallel efficiency
RAM_1	RAM amount for a single computing process (byte)
RAM_k	RAM amount for each peer process (byte)

Acknowledgements

The research is supported by the National Key Technologies R&D Program #2013BAB05B03 and #2009BAC56B03 and the National Natural Science Foundation #51109114 in China. The first author is supported by a fellowship from the Chinese government for his visit to the University of California, Los Angeles. Partial support is also provided from an AECOM endowment. The authors are very grateful to the two anonymous reviewers for their in-depth reviews and constructive comments, which greatly helped improve the paper.

References

[1] Bastian P, Helmig R. Efficient fully-coupled solution techniques for two-phase flow in porous media: parallel multigrid solution and large scale computations. Adv Water Resour 1999;23(3):199–216. http://dx.doi.org/10.1016/S0309-1708(99)00014-7.

[2] Bellman R. Adaptive control processes: a guided tour. Princeton, NJ: Princeton University Press; 1961.

[3] Bellman R. Dynamic programming. Princeton, NJ: Princeton University Press; 1957.

[4] Bhaskar NR, Whitlatch EE. Derivation of monthly reservoir release policies. Water Resour Res 1980;16(6):987–93. http://dx.doi.org/10.1029/WR016i006p00987.

[5] Casti J, Richardson M, Larson R. Dynamic programming and parallel computers. J Optim Theory App 1973;12(4):423–38. http://dx.doi.org/10.1007/BF00940421.
[6] Chandramouli V, Raman H. Multireservoir modeling with dynamic programming and neural networks. J Water Resour Plann Manage 2001;127(2):89–98. http://dx.doi.org/10.1061/(ASCE)0733-9496(2001)127:2(89).
[7] El Baz D, Elkihel M. Load balancing methods and parallel dynamic programming algorithm using dominance technique applied to the 0–1 knapsack problem. J Parallel Distrib Comput 2005;65(1):74–84. http://dx.doi.org/10.1016/j.jpdc.2004.10.004.
[8] Hall WA, Butcher WS, Esogbue A. Optimization of the operation of a multiple-purpose reservoir by dynamic programming. Water Resour Res 1968;4(3):471–7. http://dx.doi.org/10.1029/WR004i003p00471.
[9] Heidari M, Chow VT, Kokotovic PV, Meredith DD. Discrete differential dynamic programming approach to water resources systems optimization. Water Resour Res 1971;7(2):273–82. http://dx.doi.org/10.1029/WR007i002p00273.
[10] Kollet SJ, Maxwell RM. Integrated surface-groundwater flow modeling: a free-surface overland flow boundary condition in a parallel groundwater flow model. Adv Water Resour 2006;29(7):945–58. http://dx.doi.org/10.1016/j.advwatres.2005.08.006.
[11] Kollet SJ, Maxwell RM, Woodward CS, Smith S, Vanderborght J, Vereecken H, Simmer C. Proof of concept of regional scale hydrologic simulations at hydrologic resolution utilizing massively parallel computer resources. Water Resour Res 2010;46(4). http://dx.doi.org/10.1029/2009WR008730.
[12] Kumar DN, Reddy MJ. Multipurpose reservoir operation using particle swarm optimization. J Water Resour Plann Manage 2007;133(3):192–201. http://dx.doi.org/10.1061/(ASCE)0733-9496(2007)133:3(192).
[13] Labadie JW. Optimal operation of multireservoir systems: state-of-the-art review. J Water Resour Plann Manage 2004;130(2):93–111. http://dx.doi.org/10.1061/(ASCE)0733-9496(2004)130:2(93).
[14] Larson RE, Korsak AJ. A dynamic programming successive approximations technique with convergence proofs. Automatica 1970;6(2):245–52. http://dx.doi.org/10.1016/0005-1098(70)90095-6.
[15] Larson RE. State increment dynamic programming. New York: Elsevier Science; 1968.
[16] Li T, Wang G, Chen J, Wang H. Dynamic parallelization of hydrological model simulations. Environ Modell Softw 2011;26(12):1736–46. http://dx.doi.org/10.1016/j.envsoft.2011.07.015.
[17] Li X, Wei J, Fu X, Li T, Wang G. A knowledge-based approach for reservoir system optimization. J Water Resour Plann Manage, in press. doi: http://dx.doi.org/10.1061/(ASCE)WR.1943-5452.0000379.
[18] Li X, Li T, Wei J, Wang G, Yeh WWG. Hydro unit commitment via mixed integer linear programming: a case study of the three gorges project, China, IEEE Trans Power Syst, in press. doi: http://dx.doi.org/10.1109/TPWRS.2013.2288933.
[19] Martins WS, Del Cuvillo JB, Useche FJ, Theobald KB, Gao GR. A multithreaded parallel implementation of a dynamic programming algorithm for sequence comparison. Pac Symp Biocomput 2001;6:311–22.
[20] Maxwell RM. A terrain-following grid transform and preconditioner for parallel, large-scale, integrated hydrologic modeling. Adv Water Resour 2012;53:109–17. http://dx.doi.org/10.1016/j.advwatres.2012.10.001.
[21] Message passing interface forum. <http://www.mpi-forum.org/index.html>.
[22] Mousavi SJ, Karamouz M. Computational improvement for dynamic programming models by diagnosing infeasible storage combinations. Adv Water Resour 2003;26(8):851–9. http://dx.doi.org/10.1016/S0309-1708(03)00061-7.
[23] Nemhauser GL. Introduction to dynamic programming. New York: John Wiley; 1966.
[24] Piccardi C, Soncini-Sessa R. Stochastic dynamic programming for reservoir optimal control: dense discretization and inflow correlation assumption made possible by parallel computing. Water Resour Res 1991;27(5):729–41. http://dx.doi.org/10.1029/90WR02766.
[25] Pool R. Massively parallel machines usher in next level of computing power. Science 1992;256:50–1. http://dx.doi.org/10.1126/science.256.5053.50.
[26] Reed PM, Kollat JB. Visual analytics clarify the scalability and effectiveness of massively parallel many-objective optimization: a groundwater monitoring design example. Adv Water Resour 2013;56:1–13. http://dx.doi.org/10.1016/j.advwatres.2013.01.011.
[27] Rouholahnejad E, Abbaspour KC, Vejdani M, Srinivasan R, Schulin R, Lehmann A. A parallelization framework for calibration of hydrological models. Environ Modell Softw 2012;31:28–36. http://dx.doi.org/10.1016/j.envsoft.2011.12.001.
[28] Rytter W. On efficient parallel computations for some dynamic programming problems. Theor Comput Sci 1988;59(3):297–307. http://dx.doi.org/10.1016/0304-3975(88)90147-8.
[29] Sulis A. GRID computing approach for multireservoir operating rules with uncertainty. Environ Modell Softw 2009;24(7):859–64. http://dx.doi.org/10.1016/j.envsoft.2008.11.003.
[30] Tan G, Sun N, Gao GR. A parallel dynamic programming algorithm on a multi-core architecture. In: Proceedings of the nineteenth annual ACM symposium on parallel algorithms and architectures (ACM 2007), 2007. p. 135–44. http://dx.doi.org/10.1145/1248377.1248399.
[31] Tang Y, Reed PM, Kollat JB. Parallelization strategies for rapid and robust evolutionary multiobjective optimization in water resources applications. Adv Water Resour 2007;30(3):335–53. http://dx.doi.org/10.1016/j.advwatres.2006.06.006.
[32] Trott WJ, Yeh WWG. Optimization of multiple reservoir system. J Hydraul Eng Div 1973;99(10):1865–84.
[33] Wang H, Fu X, Wang G, Li T, Gao J. A common parallel computing framework for modeling hydrological processes of river basins. Parallel Comput 2011;37(6–7):302–15. http://dx.doi.org/10.1016/j.parco.2011.05.003.
[34] Wardlaw R, Sharif M. Evaluation of genetic algorithms for optimal reservoir system operation. J Water Resour Plann Manage 1999;125(1):25–33. http://dx.doi.org/10.1061/(ASCE)0733-9496(1999)125:1(25).
[35] Wurbs RA. Reservoir-system simulation and optimization models. J Water Resour Plann Manage 1993;119(4):455–72. http://dx.doi.org/10.1061/(ASCE)0733-9496(1993)119:4(455).
[36] Wu Y, Li T, Sun L, Chen J. Parallelization of a hydrological model using the message passing interface. Environ Modell Softw 2013;43:124–32. http://dx.doi.org/10.1016/j.envsoft.2013.02.002.
[37] Yakowitz S. Dynamic programming applications in water resources. Water Resour Res 1982;18(4):673–96. http://dx.doi.org/10.1029/WR018i004p00673.
[38] Yeh WWG. Reservoir management and operations models: a state-of-the-art review. Water Resour Res 1985;21(12):1797–818. http://dx.doi.org/10.1029/WR021i012p01797.
[39] Young GK. Finding reservoir operating rules. J Hydraul Eng Div 1967;93(HY6):297–321.
[40] Zhao T, Cai X, Lei X, Wang H. Improved dynamic programming for reservoir operation optimization with a concave objective function. J Water Resour Plann Manage 2012;138(6):590–6. http://dx.doi.org/10.1061/(ASCE)WR.1943-5452.0000205.