

# Two-stage distributed parallel algorithm with message passing interface for maximum flow problem

Jincheng Jiang · Lixin Wu

Published online: 31 October 2014

© The Author(s) 2014. This article is published with open access at Springerlink.com

**Abstract** Maximum flow is one of the important and classical combinatorial optimization problems. However, the time complexity of sequential maximum flow algorithms remains high. In this paper, we present a two-stage distributed parallel algorithm (TSDPA) with message passing interface to improve the computational performance. The strategy of TSDPA has two stages, which push excess flows separately along cheap and expensive paths identified by a new distance estimate function. In TSDPA, stage 1 enhances the parallel efficiency by omitting high-cost paths and decentralizing calculations, and stage 2 guarantees the achievement of an optimal solution through divide-and-conquer method. The experimental test demonstrates that TSDPA runs 1.2–15.5 times faster than sequential algorithms and is faster than or almost as fast as the H\_PRF and Q\_PRF codes.

**Keywords** Distributed · Parallel algorithm · Maximum flow · Message passing interface · Two-stage strategy

## 1 Introduction

The ultimate goal of the maximum flow problem is to determine the maximum flow value from the *source* to *sink* in a directed capacitated network while satisfying capac-

---

J. Jiang

Academy of Disaster Reduction and Emergency Management, Beijing Normal University,  
No. 19, XinJieKouWai St., HaiDian District, Beijing 100875, China  
e-mail: jiangjincheng0305@126.com

L. Wu (✉)

IoT Perception Mine Research Center, China University of Mining and Technology,  
No. 1, Daxue Road, Quanshan District, Xuzhou, Jiangsu 221008, China  
e-mail: awulixin@263.net

ity and flow conservation constraints (except for the *source* and *sink*). As a classical combinatorial problem, maximum flow is of practical interest because of its extensive application in the fields of mining, transportation planning, communications, and operations research [1]. Many researchers have performed related theoretical and experimental studies, and numerous maximum flow algorithms have been proposed during the past half-century to improve complexity bounds and reduce actual computing time in large-scale networks and real-time applications [2].

In general, sequential maximum flow algorithms include mainly augmenting path [3,4], push-relabel [5], pseudo-flow [6], BK [7], draining [8], and several other excellent algorithms [9]. Detailed reviews have been provided by Ahuja et al. [1] and Asano and Asano [10]. These algorithms improve the computational performance, but their time complexity remains high. Fortunately, parallel computing has become an effective means of solving the computing bottleneck problem. A few existing parallel algorithms can be classified into two categories, namely shared-memory and distributed-memory modes.

- For shared-memory parallel mode, several early parallel algorithms [11,12] were designed to run on a parallel random access machine (PRAM) [13]. However, PRAM is not a physically realizable parallel model and thus lacks practical application value. To investigate the practical implementation of parallel algorithms, several researchers [14,15] focused on multi-core platforms [16] in shared-memory machines. However, the limited number of available cores prevents further speedup. Several other parallel algorithms [17,18] were proposed based on graphics processing units (GPUs), into which thousands of cores are integrated [19]. However, the GPU model is unsuitable for the strong correlative operations of maximum flow algorithms because of its mode of single-instruction multiple-data stream.
- For distributed-memory parallel mode [20], the programming model based on message passing interface (MPI) is unrestricted by the number of processors and can achieve significant improvements in execution time. However, few efficient distributed parallel maximum flow algorithms based on MPI were proposed because of the strong correlation of operations in the maximum flow algorithm and the high cost of passing messages.

Regardless of the parallel mode, most existing parallel algorithms utilize fine-grained parallelization and require a large amount of communication. As a result, many parallel algorithms do not perform well in practice. By contrast, coarse-grained parallelization has the advantage of reducing the amount of communication. Delong and Boykov [21] proposed a coarse-grained parallel algorithm (referred to as Delong–Boycov’s algorithm hereafter), in which the network is partitioned into several disjoint regions and the *discharge* operations are performed in non-interacting regions in parallel. The instances the researchers considered were N-D grids in the field of computer vision. On this basis, Shekhovtsov and Hlavac [22] proposed an improved algorithm (referred to as Shekhovtsov–Hlavac’s algorithm hereafter) guided by a different distance function. Unlike Delong–Boycov’s algorithm that utilizes push-relabel updates both inside and between regions, Shekhovtsov–Hlavac’s algorithm applies the augmenting path method inside each region and push-relabel updates between regions. Shekhovtsov–Hlavac’s algorithm improves the time complexity from a tight  $O(n^2)$  bound on the

number of *sweeps* to  $2|\mathcal{B}|^2 + 1$ , where  $n$  is the number of nodes in the network and  $\mathcal{B}$  is the set of nodes incident to inter-region edges. Even so, very a few improvements have been made to enhance the speedup of parallel maximum flow algorithms. Eleyat et al. [23] suggested that the maximum flow algorithm is difficult to be parallelized. This paper presents a two-stage distributed parallel algorithm (TSDPA) with MPI for large-scale sparse networks, in which the average number of arcs incident to a node is small. By limiting the total amount of communication between processors through several newly proposed strategies, TSDPA produces practical accelerating effects.

The rest of this paper is organized as follows. Section 2 provides a review of the background of maximum flow problem and presents related studies. TSDPA is described in detail in Sect. 3. The theoretical analysis of TSDPA is presented in Sect. 4. Section 5 presents the experiments and performance analysis of TSDPA. Section 6 provides the conclusions.

## 2 Background

### 2.1 Maximum flow problem

We consider a directed network  $G(V, E, s, t, c, f)$ , where  $V$  is the set of nodes and  $E$  is the set of arcs. Each arc  $(v_i, v_j) \in E$  has a capacity  $c(v_i, v_j)$ , and  $(v_i, v_j)$  is an outgoing arc for  $v_i$  and an incoming arc for  $v_j$ .  $G$  has two distinctive nodes: source node  $s \in V$  and sink node  $t \in V$ . Flow  $f$  in  $G$  is a function that satisfies the following constraints.

$$\text{Capacity constraint: } 0 \leq f(v_i, v_j) \leq c(v_i, v_j), \quad \forall (v_i, v_j) \in E \quad (1)$$

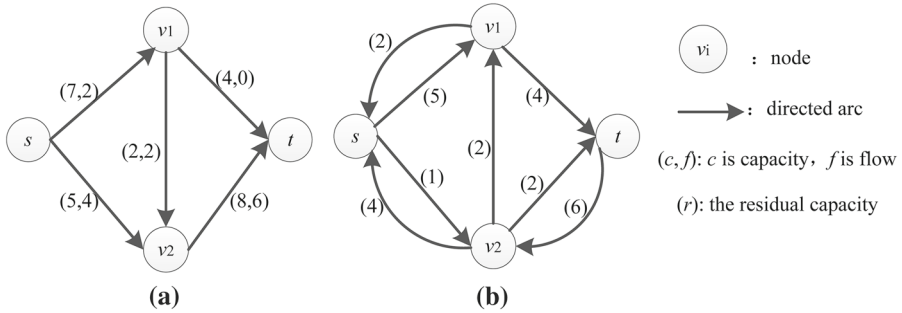
$$\begin{aligned} \text{Flow conservation constraint: } \sum_{(v_i, v_j) \in E} f(v_i, v_j) - \sum_{(v_j, v_k) \in E} f(v_j, v_k) &= 0, \\ \forall v_j \in V \setminus \{s, t\} & \end{aligned} \quad (2)$$

The value of flow  $f$  is defined as  $|f| = \sum_{(s, v_i) \in E} f(s, v_i)$ . The goal of the maximum flow problem is to find a flow with maximum value  $|f|$  from  $s$  to  $t$ .

### 2.2 Residual network

For existing flow  $f$  and arc  $(v_i, v_j) \in E$  in  $G$ , the flows in  $(v_i, v_j)$  can either increase or decrease. The residual capacity is defined as follows: (1) if  $f(v_i, v_j) = 0$ , then the residual capacity  $r(v_i, v_j) = c(v_i, v_j)$ ; (2) if  $0 < f(v_i, v_j) < c(v_i, v_j)$ , then  $r(v_i, v_j) = c(v_i, v_j) - f(v_i, v_j)$  and a reversed arc  $(v_j, v_i)$  is added,  $r(v_j, v_i) = f(v_i, v_j)$ ; (3) if  $f(v_i, v_j) = c(v_i, v_j)$ , then  $r(v_i, v_j) = 0$ ,  $r(v_j, v_i) = f(v_i, v_j)$ .

The residual network  $G_f(V, E_f, s, t, c, f)$  [5] contains the same node set  $V$  as that of  $G$  but a different arc set  $E_f$ , which is constructed by the original and reversed arcs with positive residual capacity, as depicted in Fig. 1. In the following paragraphs, pushing flows in the opposite direction of directed arc  $(v_i, v_j)$  means decreasing the flows of  $(v_i, v_j)$ .



**Fig. 1** An example of residual network: **a** the original network, **b** the residual network

### 2.3 Push-relabel method

Push-relabel method works on the residual network  $G_f(V, E_f, s, t, c, f)$  and relaxes the flow conservation constraint before the method terminates. For each node  $v_i \in V \setminus \{s, t\}$ , excess flow is defined as  $e(v_i) = \sum_{(v_j, v_i) \in E} f(v_j, v_i) - \sum_{(v_i, v_k) \in E} f(v_i, v_k)$ . Node  $v_i \in V \setminus \{s, t\}$  is an *active node* if  $e(v_i) > 0$ . Height  $h(v_i)$  is a valid function that refers to the shortest topological distance from  $v_i$  to  $t$  in  $G_f$  satisfying  $h(v_i) \leq h(v_j) + 1$  if  $(v_i, v_j) \in E_f$ . Arc  $(v_i, v_j)$  is identified as an *admissible arc* if  $(v_i, v_j) \in E_f$  and  $h(v_i) = h(v_j) + 1$ . Push-relabel method [5] maintains *preflow* and *height labeling*, where *preflow* is similar to a flow except that the total amount flowing into a node is allowed to exceed the total amount flowing out. *Preflow* and *height labeling* are iteratively modified by *push* and *relabel* operations (as shown below). Push-relabel method terminates when no more active nodes exist in  $G_f$ . Cherkassky and Goldberg [24] described push-relabel method in detail.

---

**Push ( $v_i$ )**

1. While ( $e(v_i) > 0$ )
2.   If (there is admissible arc  $(v_i, v_j) \in E_f$ )
3.     { Set  $\delta = \min(e(v_i), r(v_i, v_j))$ ,  $e(v_i) = e(v_i) - \delta$ ,  $e(v_j) = e(v_j) + \delta$ ;
4.      $r(v_i, v_j) = r(v_i, v_j) - \delta$ ,  $r(v_j, v_i) = r(v_j, v_i) + \delta$ ;
5.     If  $((v_i, v_j) \in E)$
6.          $f(v_i, v_j) = f(v_i, v_j) + \delta$ ;
7.     Else
8.          $f(v_j, v_i) = f(v_j, v_i) - \delta$ ;
9.     }
10.   Else
11.    Break;

**Relabel( $v_i$ )**

1. If  $(e(v_i) > 0)$  and  $v_i$  has no admissible arcs)
  2. { Select a node  $v_j$  with the minimum height  $h(v_j)$  from the neighbor set of  $v_i$  in  $G_f$ ;
  3. Set  $h(v_i) = \min\{h(v_j) + 1\}$ ;
  4. }
-

In the above procedures, the aim of steps 3–9 in the *Push* ( $v_i$ ) operation is to push flows from  $v_i$  to  $v_j$ .

## 2.4 Related work

Several excellent distributed parallel algorithms have been proposed for N–D grids in the domain of computer vision. Delong–Boykov’s algorithm [21] partitions the network into several regions and then discharges the flows for different non-interacting regions in parallel. After partitioning a network into regions, this algorithm repeats the following steps until no more active nodes exist.

- 
1. Several non-interacting regions with active nodes are dynamically selected
  2. Flows in the selected regions are discharged in parallel
  3. The region-gap, region-relabel and global relabeling heuristics are applied [21,22,24]
- 

Rather than using push-relabel updates inside regions as in Delong–Boykov’s algorithm, Shekhovtsov and Hlavac [22] adopt the augmenting path method to push flows inside regions to the boundary nodes in the order of their increasing labels, which are determined by a distance function. Given the same fixed partition as Delong–Boykov’s algorithm, the label of boundary nodes in Shekhovtsov–Hlavac’s algorithm indicates the sum of region boundaries crossed by a path to  $t$ . With a new distance function, Shekhovtsov and Hlavac present a new algorithm with the following steps.

- 
1. Init;
  2. While *there are active nodes* do
  3.     For  $k = 0, 1, \dots, d^\infty$ , do
  4.         Augment flow to boundary nodes with label  $d = k$ ;
  5.     Apply the new distance function to update the labels of boundary nodes;
  6.     Apply the global gap heuristic.
- 

The labels of boundary nodes in the above-mentioned competing algorithms are globally valid. In the following section, we present TSDPA. The labels of boundary nodes in stage 1 of TSDPA are locally valid.

## 3 TSDPA

TSDPA, which is designed based on push-relabel method, consists of two main stages (Fig. 2). After preprocessing, which aims to partition the network into several regions and distribute them to different processors, the flows are pushed along cheap and expensive paths to  $t$  with different strategies in stages 1 and 2, respectively.

### 3.1 Preprocessing

The network should be partitioned into several regions to enable multiple independent processors to work in parallel in a distributed environment. Two principles must be

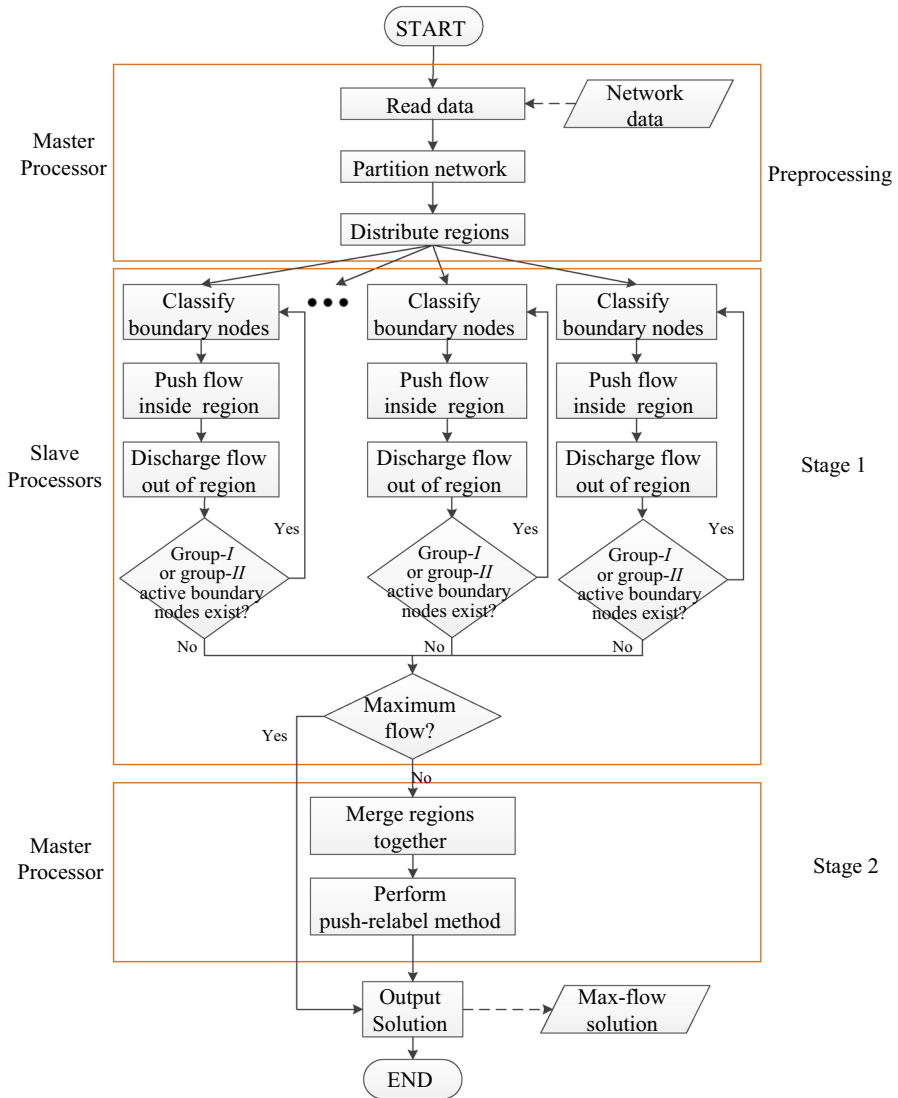
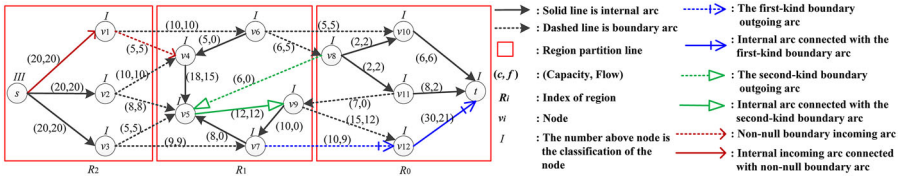


Fig. 2 Overall workflow of TSDPA

followed for partitioning the network. The first principle is load balance, i.e., all regions should have similar sizes because the total running time depends on the longest computing time among all processors. The second principle is that the number of nodes incident to the inter-region arcs should be as few as possible to reduce the cost of communication between regions.

Many methods [25] can be utilized to partition the network. In this paper, leveled nested dissection (LND) method ( $t$  is the initial seed) is selected because of its fast partition speed and the tendency of connected nodes to fit together. These qualities make the method well suited for coarse-grained parallelization.



**Fig. 3** Example of classifying boundary nodes:  $T(v_7)$  will be  $I$  because of  $(v_7, v_{12})$ ,  $T(v_8)$  will be  $II$  because of  $(v_8, v_5)$ , and  $T(v_1)$  will be  $III$

We assume that network  $G(V, E, s, t, c, f)$  with  $n = |V|$  nodes and  $m = |E|$  arcs is partitioned into  $p$  disjoint regions  $G_k$  ( $0 \leq k \leq p - 1$ ) through LND method, where  $p$  is the number of processors.  $G_k(V_k, E_k, c, f)$  is defined as follows.

Region node set  $V_k$  ( $V_k \subset V$ ) contains a subset of connected nodes in  $G$ , and the collection of regions  $V_k$  forms a partition of  $V$  (Fig. 3).  $V_k$  includes two parts: boundary node set  $V_k^B$  and internal node set  $V_k^R$ . The nodes in  $V_k^B$  are incident to inter-region edges, i.e.,  $V_k^B = \{v_i | \exists v_j \in G, (v_i, v_j) \in E \text{ or } (v_j, v_i) \in E, v_j \notin V_k, v_i \in V_k\}$ . Internal node set  $V_k^R$  is defined as  $V_k^R = V_k - V_k^B$ .

Region arc set  $E_k$  includes internal arc set  $E_k^R = \{(v_i, v_j) | (v_i, v_j) \in E, v_i \in V_k, v_j \in V_k\}$  and boundary arc set  $E_k^B$ , which are divided into boundary outgoing arc set  $E_k^{out}$  and boundary incoming arc set  $E_k^{in}$ . For region  $G_k$ , boundary outgoing arc set  $E_{kw}^{out}$  from regions  $G_k$  to  $G_w$  ( $k \neq w$ ) is defined as  $E_{kw}^{out} = \{(V_k^B, V_w^B)\}$ , and the corresponding boundary incoming arcs are defined as  $E_{kw}^{in} = \{(V_w^B, V_k^B)\}$ .  $E_k^{out} = \cup_{w=0}^{p-1} (E_{kw}^{out})$  and  $E_k^{in} = \cup_{w=0}^{p-1} (E_{kw}^{in})$ .

An important region distance function  $D_k$  is presented for region  $G_k$  in TSDPA.  $D_k$  equals the generated order of partitioned regions, e.g.,  $D_0 = 0, D_1 = 1$ , and  $D_2 = 2$  (Fig. 3). In other words,  $D_k$  is the number of regions between  $G_k$  and  $t$ .

As illustrated in Fig. 2, after partitioning the network into  $p$  regions, the master processor distributes the regions to several slave processors. Without loss of generality, region  $G_k$  is assigned to processor  $k$ . For any region  $G_k$ , stages 1 and 2 of TSDPA are described as follows.

### 3.2 Stage 1

In this section, we present a region-level heuristic function to evaluate the distance of boundary nodes (or to classify the boundary nodes) by which cheap and expensive feasible paths can be distinguished. Efficient methods are then proposed to push flows both inside and between regions for each processor  $k$  that is responsible for processing region  $G_k$  ( $0 \leq k \leq p - 1$ ).

#### 3.2.1 Region-level heuristic

Traditionally, the distance function of push-relabel method counts the length of a feasible path to  $t$ . In Shekhovtsov–Hlavac’s algorithm, the distance of boundary nodes

corresponds to the number of regions required to transfer the excess flow to  $t$ . This distance is an accurate global function with a maximum value of  $|\mathcal{B}| - 1$ , where  $\mathcal{B} = \cup(V_k^B)_{k=0}^{p-1}$  is the set of boundary nodes. Therefore, the number of passing messages along a single feasible path remains large, and these messages cannot be passed in batches. This drawback is unfavorable for MPI because its communication efficiency largely depends on the number of passing messages.

TSDPA uses a region-level heuristic to evaluate the distance of boundary nodes. This heuristic provides each boundary node a local classification, not a global distance. Boundary nodes in region  $G_k$  are classified into three groups (group  $I$ , group  $II$ , and group  $III$ ), which are determined by the residual capacity of boundary outgoing arcs and the region distance of neighboring regions. We employ  $T_i = I$ ,  $T_i = II$ , and  $T_i = III$  to denote the classification of boundary node  $v_i$  when it belongs to groups  $I$ ,  $II$ , and  $III$ , respectively.  $T_i = I$  implies that feasible paths from  $v_i$  to  $t$  exist and that the next regions in these paths are closer to  $t$  than  $G_k$ .  $T_i = II$  implies that there are feasible paths, but the region distances of the next regions in these paths are larger than those of  $G_k$ . Finally,  $T_i = III$  implies that no guaranteed feasible paths from node  $v_i$  to  $t$  exist. This region-level heuristic is described as follows.

$s$  and  $t$  are regarded as special boundary nodes, and  $T_s = III$  and  $T_t = I$  remain unchanged. Initially, all boundary nodes (except for  $s$ ) belong to group  $I$  because every boundary node has feasible paths to  $t$  crossing regions with decreasing region distances after the network is partitioned through LND method. At each iteration, the classification of any boundary node  $v_i$  ( $v_i \neq s$ ,  $v_i \neq t$ ) changes dynamically according to the following criteria.

1. If the first-kind boundary outgoing arc  $(v_i, v_j) \in E_{kw}^{out}$  exists, where  $T(v_j) = I$ ,  $c(v_i, v_j) > f(v_i, v_j)$  and  $D_k \geq D_w$ , then  $T(v_i) = I$ ;
2. Else, if the second-kind boundary outgoing arc  $(v_i, v_j) \in E_{kw}^{out}$  exists, where  $T(v_j) = I$ ,  $c(v_i, v_j) > f(v_i, v_j)$  and  $D_k < D_w$ , then  $T(v_i) = II$ ;
3. Else,  $T(v_i) = III$ , which indicates that no direct feasible boundary outgoing arcs to the neighboring regions exist.

After classifying with these criteria, we let  $S_0$ ,  $S_1$ , and  $S_2$  be the sets of groups  $I$ ,  $II$ , and  $III$  boundary nodes, respectively.

The above-mentioned criteria are based on the residual capacity of boundary arcs and the classification of boundary nodes in neighboring regions without considering the feasible paths inside the region. To make the classification more accurate, we update the classification of boundary nodes with region relabeling heuristics:

1. Breadth-first search (BFS) method starting from  $S_0$  is performed to search for feasible paths. If boundary node  $v_i$  with  $T(v_i) \neq I$  has feasible paths to  $S_0$ , then set  $T(v_i) = I$ .
2. BFS method is performed starting from  $S_1 - S_1 \cap S_0$ . If boundary node  $v_j$  with  $T(v_j) = III$  has feasible paths to  $S_1 - S_1 \cap S_0$ , then set  $T(v_j) = II$ .

We let  $S_3$ ,  $S_4$ , and  $S_5$  be the sets of the groups  $I$ ,  $II$ , and  $III$  boundary nodes after these updates, respectively.

The cheap and expensive feasible paths are identified with this region-level heuristic. For an active node  $v_i$  in region  $G_k$ , the feasible paths, whose boundary arcs only



contain the *first-kind boundary outgoing arc* or the *second-kind boundary outgoing arc*, cross boundary  $k - 1$  and  $k + 1$  times, respectively. These paths are identified as cheap paths, otherwise expensive paths. As illustrated in Fig. 3, although some calculation lags exist, reachable feasible paths from group *I* nodes to  $t$  are shorter than those from group *II* nodes in the same region. Meanwhile, the group *III* nodes have no guaranteed reachable feasible paths to  $t$ .

### 3.2.2 Region discharge

Shekhovtsov–Hlavac’s algorithm pushes excess flows to  $t$  and boundary nodes in the order of their increasing distances. In TSDPA, the distance of boundary nodes has only three values, which represent the three classifications. For convenience of expression, we let  $S_6$ ,  $S_7$  and  $S_8$  be the set of group *I* nodes with the *first-kind boundary outgoing arcs*, the set of group *II* nodes with the *second-kind boundary outgoing arcs*, and the set of group *III* nodes with the *non-null boundary incoming arcs* from  $G_{k+1}$ , respectively. A boundary arc  $(v_i, v_j)$  is defined as the *non-null boundary incoming arc* for  $v_j$  if  $f(v_i, v_j) > 0$ . Clearly,  $S_6 = S_0$ ,  $S_7 = S_1 - (S_1 \cap S_3)$ , and  $S_8 \subseteq S_5 \subseteq S_2$ .

We push excess flows inside the region to  $S_6$ ,  $S_7$ , and  $S_8$  successively. Then, we discharge the excess flows of these destinations along specific boundary arcs out of the region.

#### 1. Pushing flows inside the region

In Sect. 3.2.1, the region relabeling heuristics starting from  $S_6$  and  $S_7$  (in order) are performed inside the region to update the classification of boundary nodes. With these heuristics, corresponding sub-regions in which nodes have feasible paths to destinations  $S_6$ ,  $S_7$ , and  $S_8$  are separated and represented as  $R_1 \subseteq G_k$ ,  $R_2 \subseteq G_k$ , and  $R_3 \subseteq G_k$ . In other words, excess flows in  $R_1$ ,  $R_2$ , and  $R_3$  should be pushed to  $S_6$ ,  $S_7$ , and  $S_8$ , respectively.

Several good sequential methods can be utilized to push flows inside each region. Delong and Boykov apply push-relabel method, whereas the augmenting path method is adopted in Shekhovtsov–Hlavac’s algorithm. Push-relabel method is selected for TSDPA because it outperforms other methods in many instances [26].

Both communication complexity and load balance should be considered to achieve the desired practical accelerating effects. Therefore, highly efficient and parallelizable sequential methods must be selected or designed. However, existing sequential algorithms have difficulty in guaranteeing both qualities. For example, the H\_PRF code [24] is one of the most efficient algorithms in practice, but it loses parallelizability because of its highly concentrated operations. The F\_PRF code [24] is suitable for parallel implementation but does not perform as well as the H\_PRF code. We present two practical and highly efficient implementations of push-relabel method with high parallelizability. These two implementations have similar steps as follows.

1. Initially, we set height  $h(i) = \infty, \forall i \in V/\{t\}$ , and  $h(t) = 0$ .
2. BFS method is performed starting from  $t$  to label all nodes, as the global relabeling heuristics do. The searched active nodes (or  $s$  at the first iteration) are placed into

- S. If  $S$  is not empty, then all searched nodes are set as valid nodes; otherwise, the algorithm terminates.
3. An active node  $v_i$  with the largest height is selected from  $S$ . If there are valid neighbors with smaller heights, we push the excess flow of  $v_i$  along all admissible arcs to these valid neighbors until  $v_i$  has no admissible arcs or the excess flow of  $v_i$  is zero.  $v_i$  is identified as an invalid node if  $e(v_i) > 0$  after pushing flows along all admissible arcs.
  4. If  $e(v_i) > 0$ , we push the excess flow of  $v_i$  to valid neighbors with the same or larger height along feasible arcs until  $e(v_i) = 0$ .
  5. Steps 3 and 4 are repeated until no more valid active nodes exist.
  6. All nodes are set as invalid nodes except for  $t$ , and steps 2–5 are performed iteratively until no more active nodes exist.

In steps 3 and 4, the newly generated active nodes are placed in  $S$ . We call this approach multi-directional push (MDP) method if all steps are performed; otherwise, the single-directional push (SDP) method if step 4 is not performed.

In contrast to the traditional push-relabel method, no actual relabel operation is performed in MDP and SDP methods. The nodes' heights are updated only by global relabeling heuristics. In addition, node  $v_i$  is marked as valid or invalid to suggest whether there are good feasible paths from  $v_i$  to  $t$  or not. MDP and SDP methods provide several advantages for our parallel implementation. On one hand, MDP and SDP methods use global relabeling heuristics and inherit the advantages of highest-level selection strategy [24] because they always select the highest valid nodes. Thus, computational efficiency is enhanced. On the other hand, calculations in MDP and SDP methods are decentralized because generated invalid nodes reduce the contact among active nodes. Thus, multiple active nodes can perform operations in parallel with high efficiency. More detailed performance tests are described in Sect. 5.

## 2. Discharging flows out of the region

This step aims to discharge excess flows for the three kinds of boundary nodes out of the region along specific boundary arcs.  $S_6$  pushes excess flows along the *first-kind boundary outgoing arcs*,  $S_7$  pushes excess flows along the *second-kind boundary outgoing arcs*, and all excess flows in  $S_8$  can be pushed back to  $s$  or  $G_{k+1}$  along non-null boundary incoming arcs.

As an example, group *I* nodes include two kinds of boundary nodes. The first one (i.e.,  $S_6$ ) owns the *first-kind boundary outgoing arcs*, whereas the second one (i.e.,  $S_3 - S_6$ ) does not. Excess flows of  $S_6$  are discharged out of the region only along the *first-kind boundary outgoing arcs*. According to the classification criteria, feasible paths from  $S_3 - S_6$  to  $S_6$  may exist inside the region; thus, the excess flows of  $S_3 - S_6$  are pushed along these feasible paths rather than along the *second-kind boundary outgoing arcs* or *non-null boundary incoming arcs*.

The above discussions are also applicable to the group *II* nodes. All excess flows of groups *I* and *II* boundary nodes are discharged out of the region along boundary outgoing arcs. By contrast, the group *III* boundary nodes discharge excess flows along *non-null boundary incoming arcs*.

### 3.2.3 Feature of stage 1

Stage 1 of TSDPA performs the above region-level heuristic and region discharge operations iteratively until no more active groups *I* and *II* boundary nodes exist.

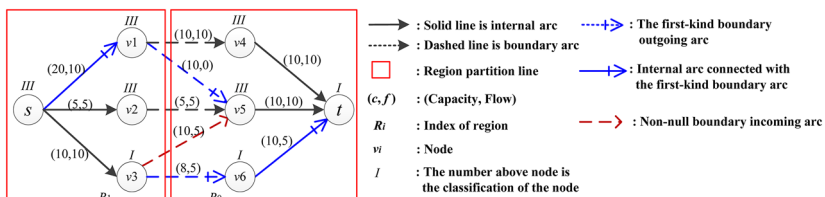
Two differences are observed between stage 1 of TSDPA and Shekhovtsov–Hlavac’s algorithm. The first difference is the label of boundary nodes. In place of three classification values, the label of boundary nodes in Shekhovtsov–Hlavac’s algorithm counts the region boundaries crossed by the shortest feasible path to *t*. These labels can be obtained by a global relabeling heuristic. The second difference is the sequential algorithm. Inside each region, stage 1 of TSDPA uses push-relabel method to push excess flows to groups *I*, *II*, and *III* boundary nodes in order. However, Shekhovtsov–Hlavac’s algorithm uses the augmenting path method to push excess flows to the boundary nodes in the increasing order of their labels.

### 3.3 Stage 2

After the termination of stage 1, the obtained flow may not be the maximum one because high-cost feasible paths are implicitly omitted. As described in Fig. 4, the omitted feasible path  $s - v_1 - v_5 - v_3 - v_6 - t$  is considered an expensive path according to classification criteria because it crosses boundaries many times. This stage aims to make the solution optimal.

In Shekhovtsov–Hlavac’s algorithm, the maximum label of a boundary node is  $|\mathcal{B}| + 1$ , which means that a feasible path crosses boundaries  $|\mathcal{B}| + 1$  times at most. The value of  $|\mathcal{B}|$  is difficult to decrease; thus, too many messages need to be passed. To avoid unfavorable cases, we adopt the divide-and-conquer strategy [27]. First, the master processor collects the information of all regions and merges them into an entire network. Second, sequential push-relabel method is performed by the master processor on this network until the optimal solution is obtained. Although this stage makes no contributions to the acceleration effect of TSDPA, undesirable cases with excessive communication are avoided and the optimal solution is guaranteed.

The acceleration of TSDPA depends on the workload of stages 1 and 2. If stage 1 performs most of the calculations, then the effect of acceleration will be significant and vice versa.



**Fig. 4** Example of omitted path: path  $s - v_1 - v_5 - v_3 - v_6 - t$  is considered an expensive path, where arc  $(v_5, v_3)$  is the reversed arc of  $(v_5, v_3)$  and the residual capacity  $r(v_5, v_3) = f(v_3, v_5)$

### 4 Communication complexity

The parallel performance of TSDPA relies mainly on the parallelization of stage 1; most messages are passed in this stage. In this section, we analyze the communication complexity of stage 1.

For group *I* boundary node *i* in region  $G_k$ , the shortest feasible path *P* from *i* to *t* crosses regions with monotonically decreasing region distances. In other words, the number of boundaries for *P* to cross is *k*. As for the excess flows of group *II* boundary nodes in  $G_k$ ,  $k + 2$  boundaries need to be crossed because the excess flows are first sent to the group *I* boundary nodes in  $G_{k+1}$  and then to *t*. The communication complexity can be analyzed in three steps.

First, we suppose that no discharge operation is performed for excess flows of  $S_7$  and  $S_8$ . No flow will be pushed to region  $G_{p-1}$  after the first iteration, where *p* is the number of processors (or regions). The excess flows in  $G_{p-1}$  will be discharged out of the region only by  $S_6$ . The maximum size of  $S_6$  in  $G_{p-1}$  is  $|V_{p-1}^B|$ .  $|S_6|$  is reduced by at least one for each iteration. The reason is that if active node  $v_i \in S_6$  remains active after the discharge operation, then  $v_i$  no longer belongs to  $S_6$ ; otherwise,  $S_6$  is empty. Therefore, the maximum number of iterations in region  $G_{p-1}$  is  $|V_{p-1}^B|$ . Similarly, the communication complexities are  $|V_{p-1}^B| + |V_{p-2}^B|, |V_{p-1}^B| + |V_{p-2}^B| + |V_{p-3}^B|, \dots, |V_{p-1}^B| + |V_{p-2}^B| + \dots + |V_0^B|$  for regions  $G_{p-2}, G_{p-3}, \dots, G_0$ , respectively. The maximum number of passing messages in this case is  $|U(V_u^B)_{u=k}^{p-1}|$  for any processor *k*.

Second, we suppose that the discharge operations are performed only for  $S_6$  and  $S_7$ . According to the region-level heuristic, region  $G_0$  never receives flows from group *II* boundary nodes in other regions. Region  $G_1$  can obtain flows from  $S_7$  in region  $G_0$ , and the maximum number of receiving flows from  $S_7$  in  $G_0$  is  $|V_0^B|$  according to the region-level heuristic. For any region  $G_k$  ( $1 \leq k \leq p - 1$ ), the maximum number of passing messages from group *II* boundary nodes is  $|U(V_u^B)_{u=0}^{k-1}|$ . Considering the messages from group *I* boundary nodes in other regions, the total number of messages for passing is  $|U(V_u^B)_{u=0}^{k-1} + U(V_u^B)_{u=k}^{p-1}| = |U(V_u^B)_{u=0}^{p-1}|$ . The worst case is the one in which flows of groups *I* and *II* nodes are sequentially processed.

Lastly,  $S_6, S_7$ , and  $S_8$  perform the discharge operations as TSDPA does. The terminal condition of TSDPA is that no more group *I* or *II* boundary active nodes exist, rather than all excess flows being pushed to *t* or back to *s*. According to the region-level heuristic, a flow will never be pushed along a cycle crossing boundaries in the above steps. Implicitly, *p* region heuristics relabel operations correspond to a global heuristic relabel; thus, the region-level heuristic is globally valid from this perspective. For the excess flow originating from group *III* boundary nodes in  $G_{k-1}$ , three destinations are available:  $S_6, S_7$ , and  $S_8$ . If a flow is pushed to  $S_8$ , then it must be discharged to  $G_{k+1}$  or remain at  $S_8$  until stage 1 terminates; otherwise, it is in a feasible path to *t*. After at most  $p + 1$  iterations, any flow in the network has four destinations: *s, t, S<sub>5</sub> – S<sub>8</sub>*, and  $v_i$  ( $v_i \in S_8$  and  $v_i$  is in a feasible path). The flows in the first three destinations no longer need to be processed. As for the last destination (i.e.,  $v_i$ ), at

least one group *I* or *II* boundary node in any region becomes the group *III* boundary node after  $2p$  iterations; otherwise, all excess flows are pushed to  $t$  or  $s$ . Thus, at most  $(2p)|U(V_u^B)_{u=0}^{p-1}|$  messages are added for passing if we discharge the excess flows for  $S_8$ .

In conclusion, the communication complexity of TSDPA is  $(2p + 1)|U(V_u^B)_{u=0}^{p-1}| = (2p + 1)|B|$  because the worst case occurs when discharge operations are performed sequentially for  $S_6$ ,  $S_7$ , and  $S_8$ .

## 5 Experimental test

### 5.1 Experiment design

Our experimental test includes two parts. The first part involves measuring the time performance of four sequential algorithms, namely MDP, SDP, and the well-known H\_PRF and Q\_PRF codes [24]. All tested algorithms use both global and gap relabeling heuristics. Global relabeling is performed after every  $n$  relabel operations. All sequential algorithms are implemented with the same programming style. The second part involves evaluating the parallel efficiency of TSDPA compared with corresponding sequential algorithms.

In this experiment, we do not compare TSDPA with Shekhovtsov–Hlavac’s algorithm because of the different network. The network utilized in Shekhovtsov–Hlavac’s algorithms is N-D grids, and nearly all nodes are connected to  $s$  and  $t$ . This kind of network is unsuitable for TSDPA because the LND method starting from  $t$  partitions N-D grids with excessive boundary arcs and thus compromises the advantages of TSDPA. Theoretically, TSDPA performs well for a network with a large topological distance between  $s$  and  $t$ . In other words, the parallel performance will be good if  $s$  is located at region  $G_{p-1}$  and  $t$  is located at region  $G_0$ . Moreover, the parallel versions of competing algorithms have difficulty in running faster than sequential algorithms in many practical instances. To evaluate thoroughly the parallel efficiency of TSDPA considering the partition time, five artificial networks are utilized for testing. These networks include a Washington-Line-Moderate (Line-Moderate) network with 65,538 nodes and 4,186,142 arcs, a Washington-RLG-Long (RLG-Long) network with 1,048,578 nodes and 3,145,664 arcs, a Washington-RLG-Wide (RLG-Wide) network with 524,290 nodes and 1,564,672 arcs, a Genrmf-Long network with 651,600 nodes and 3,170,220 arcs, and a Genrmf-Wide network with 3920 nodes and 18,256 arcs. All these networks were published in the first DIMACS workshop held in 1991 at Rutgers University. The details can be found in Badics, Boros, and Cepek [28]. They have been widely utilized to test maximum flow algorithms [26].

All tested algorithms are executed on a blade server cluster that provides adequate virtual machines. Each virtual machine has two cores, 2.53 GHz Intel Xeon CPU, and 4 GB available memory. The switch is QLogic 20-port 8 Gb SAN Switch, whose largest communication capability is 8Gb/s. The operating system is 64-bit Windows 7 Professional. All codes are written in C++ and compiled by *gcc* compiler with O2 optimization. The parallel library utilized is the MPICH2 library. Each result is averaged over three runs.

## 5.2 Results and discussion

With regard to correctness, TSDPA can obtain the optimal solution. The reason is that in stage 1, the capacity constraint is always satisfied when performing the *push* operation regardless of inside a region or between regions; however, active nodes may exist and the solution may be not the optimal one after stage 1 terminates. In stage 2, the sequential push-relabel algorithm is performed in the network; thus, all excess flows of active nodes are pushed to  $t$  or  $s$  until the solution is optimal when stage 2 terminates.

With regard to time performance, the following experimental results are presented to evaluate the performance of sequential and parallel algorithms. The acceleration of a parallel algorithm over the sequential algorithm is calculated as follows.

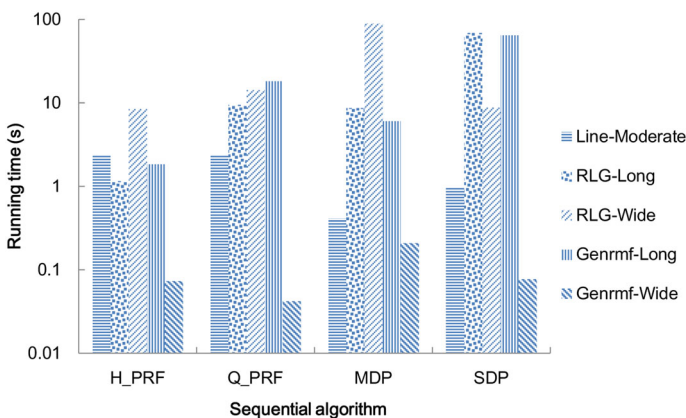
$$speedup = \frac{t_s}{t_p},$$

where  $t_s$  and  $t_p$  are the running time of sequential and corresponding parallel algorithms, respectively.

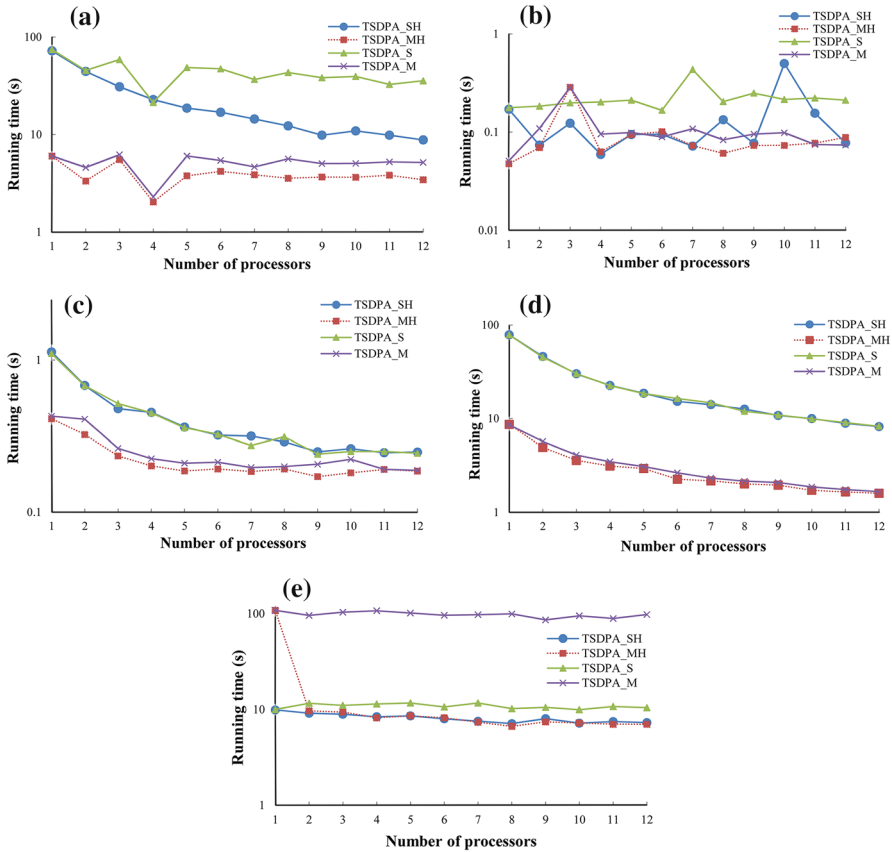
### 1. Performance evaluation for sequential algorithms

The sequential algorithm selected to be performed in each region by the corresponding processor affects not only the speedup of TSDPA, but also its overall performance. Thus, selecting a proper sequential algorithm with high time efficiency and parallelizability is crucial.

Figure 5 shows the time performance comparison of the sequential algorithms. The MDP algorithm shows good performance in the Line-Moderate network, and the SDP algorithm performs at high efficiencies in the Line-Moderate, RLG-Wide, and Genrmf-Wide networks. Compared with the Q\_PRF code, the better performance between the MDP and SDP algorithms is not far worse in all networks. Compared



**Fig. 5** Time performance comparison of sequential algorithms



**Fig. 6** Time efficiency of TSDPA for different networks: **a** Genrmf-Long network, **b** Genrmf-Wide network, **c** Line-Moderate network, **d** RLG-Long network, and **e** RLG-Wide network

with the H\_PRF code, the worst cases for the MDP and SDP algorithms occur in the RLG-Long and Genrmf-Long networks; in these two networks, the MDP algorithm outperforms the SDP algorithm and runs 8 and 6 times slower than the H\_PRF code, respectively. Furthermore, the MDP and SDP algorithms are only slightly worse than the Q\_PRF and H\_PRF codes. Both MDP and SDP algorithms outperform the Q\_PRF and H\_PRF codes in the Line-Moderate network. Thus, these two newly proposed sequential algorithms have certain superiority in time performance.

With regard to H\_PRF and Q\_PRF codes, the H\_PRF code runs faster than the Q\_PRF code in most of the networks, except for the Genrmf-Wide network. However, the parallelizability of the Q\_PRF code is higher than that of the H\_PRF code [24]. Each of the sequential algorithms has its unique advantages in time performance or parallel efficiency. The H\_PRF code performs with high efficiency for most of the tested networks. Thus, the H\_PRF code is a good choice for use as the sequential algorithm in stage 2 of TSDPA.

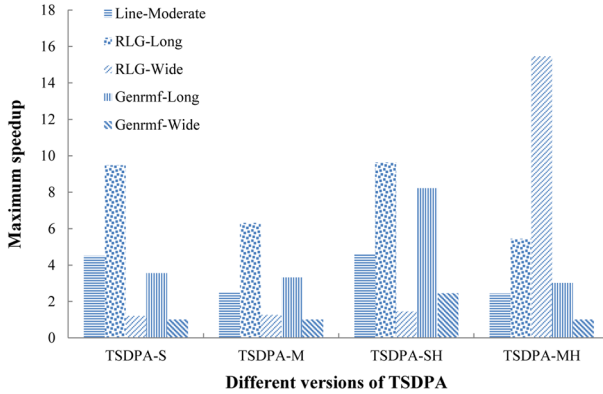


Fig. 7 Maximum speedup of TSDPA

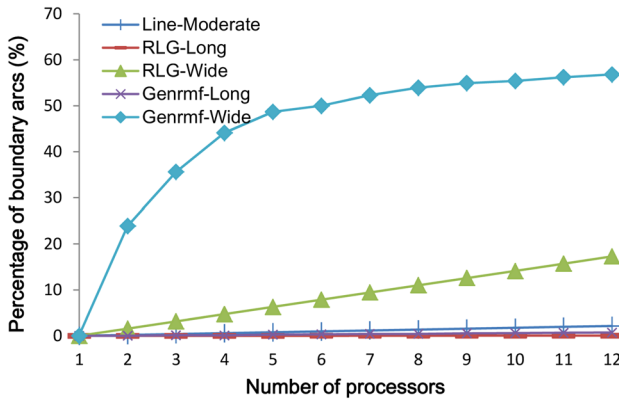


Fig. 8 Percentage of the number of boundary arcs for different numbers of processors

2. Performance evaluation for TSDPAs

Figure 6 shows the parallel efficiencies of TSDPAs for different networks. The method used in stage 1 of TSDPA-M and TSDPA-MH is the MDP algorithm. TSDPA-S and TSDPA-SH use the SDP algorithm in stage 1. The algorithms used in stage 2 of TSDPA-M, TSDPA-S, TSDPA-MH, and TSDPA-SH are MDP, SDP, H\_PRF, and H\_PRF, respectively. The following observations on the comparison of the computational time of parallel algorithms are obtained from Fig. 6.

- TSDPAs achieve desired parallel efficiencies in the Line-Moderate, RLG-Long, and Genrmf-Long networks. For example, the maximum speedups of the TSDPA-SH in Fig. 6a, c and d reach up to 8.22, 4.62, and 9.64 (Fig. 7), respectively. The parallel efficiencies are high because the TSDPAs exploit fully the cluster architecture, in which several processors work in parallel to reduce running time. The TSDPAs run efficiently in parallel for three reasons. First, the LND method, which produces few boundary arcs connected to different regions for the tested networks (Fig. 8), is utilized to partition the network and thus only slightly increases communication



time. Second, stage 1 limits the number of passing messages by omitting high-cost paths; thus, excessive communication is avoided. Third, operations are decentralized to make several processors work in parallel with load balance by designing efficient sequential algorithms with high parallelizability, such as the MDP and SDP methods in stage 1.

- For the comparison of TSDPAs and the H\_PRF code, the experimental results can be divided into three groups. The first group includes RLG-Long and Genrmf-Long networks. Although both MDP and SDP algorithms perform worse than the H\_PRF code in these networks, the time performances of TSDPAs are close to that of the H\_PRF code when the processors reach a certain number. The Line-Moderate network belongs to the second group. The MDP and SDP algorithms perform better than the H\_PRF code in this network, and the TSDPAs run much faster than the H\_PRF code with the increase in the number of processors. The last group includes Genrmf-Wide and RLG-Wide networks, in which the TSDPAs exhibit inferior performances because of excessive boundary arcs, but not much worse than that of the sequential algorithms.
- The other findings from Fig. 6 are that TSDPA-M exhibits always better parallel efficiency than TSDPA-S and that TSDPA-MH runs faster than TSDPA-M in all tested networks. In fact, TSDPAs with Q\_PRF and H\_PRF codes used in stage 1 are also tested, but their parallel performances are unbearable. The reasons for such performances are elaborated in Sect. 3.2.2. Several unusual cases in Fig. 6 need to be explained. First, the parallel efficiencies of TSDPAs in Genrmf-Wide and RLG-Wide networks are too low because of the large number of boundary arcs after partitioning the network (as shown in Fig. 8) and result in excessive workload for stage 2 without any parallel efficiency. Second, TSDPA-MH exhibits a super-linear speedup in the RLG-Wide network because the H\_PRF code runs much faster than the MDP method in stage 2. Third, the fluctuations in Fig. 6a, b are caused by the different workloads in stage 2. These differences are caused by the different numbers of expensive feasible paths omitted in stage 1.

Although the maximum flow algorithm is difficult to be parallelized [23], our distributed parallel algorithm using different sequential algorithms produced maximum speedups of 4.62, 9.64, 15.46, 8.22, and 2.45 for different networks. Compared with the speedups in other studies [14, 18, 23], we consider our speedups excellent.

## 6 Conclusion and future work

This paper presents TSDPA, which is a distributed parallel algorithm with MPI for the maximum flow problem. TSDPA is a coarse-grained parallelization that can expedite the computing speed for sparse networks in high-performance computing environment. To improve the parallel performance, a two-stage strategy is applied to push flows separately along cheap and expensive feasible paths identified by a novel distance function. Two strategies are implemented in stage 1 of TSDPA to accelerate solving the maximum flow problem. The first strategy involves omitting high-cost paths, which aims to limit the amount of communication. The second strategy entails the decentralization of calculations. It aims to improve parallelizability and thus allows multiple proces-

sors to work in parallel with load balance. These two strategies enhance the parallel efficiency of TSDPA in a distributed environment. The experimental test demonstrated that TSDPA generates considerable accelerating effects on the RLG-Long, Genrmf-Long, and Line-Moderate networks and runs faster than or almost as fast as the H\_PRF and Q\_PRF codes. The maximum speedups are 4.62, 9.64, 15.46, 8.22, and 2.45 in the Line-Moderate, RLG-Long, RLG-Wide, Genrmf-Long, and Genrmf-Wide networks, respectively.

The network partitioning method and sequential algorithm utilized inside each region are the main factors that influence the performance of TSDPA. Strandmark and Kahl [29] suggested that the parallelization of the maximum flow algorithm is not theoretically guaranteed to be fast for every network. Therefore, suitable partitioning methods, sequential maximum flow algorithms, and distance functions for boundary nodes should be designed and tested for particular topological networks. Also, hybrid parallel technologies, such as the use of distributed- and shared-memory modes in tandem, require further investigation.

**Acknowledgments** This study was funded by the National High-Tech Research and Development Program of China (2011AA120302).

**Open Access** This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

## References

1. Ahuja RK, Magnanti TL, Orlin JB (1993) Network flows: theory, algorithms and applications. Prentice-Hall, Englewood Cliffs
2. Nagy N, Akl SG (2003) The maximum flow problem: a real-time approach. *Parallel Comput* 6(29):767–794
3. Dinic EA (1970) Algorithm for the solution of a problem of maximal flow in networks with power estimation. *Soviet Math Doklady* 11:1277–1280
4. Ahuja RK, Orlin JB (1991) Distance-directed augmenting path algorithms for maximum flow and parametric maximum flow problems. *Nav Res Log* 3(38):413–430
5. Goldberg AV, Tarjan RE (1988) A new approach to the maximum flow problem. *J ACM* 4(35):921–940
6. Hochbaum DS (2008) The pseudoflow algorithm: a new algorithm for the maximum-flow problem. *Oper Res* 4(56):992–1009
7. Boykov Y, Kolmogorov V (2004) An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans Pattern Anal* 9(26):1124–1137
8. Dong JY, Li W, Cai CB, Chen Z (2009) Draining algorithm for the maximum flow problem, 2009 WRI International Conference on Communications And Mobile Computing: CMC 2009, Washington DC, IEEE Computer Society, vol 3, pp 197–200
9. Goldberg AV (2008) The partial augment-relabel algorithm for the maximum flow problem. In: Proceedings 16th annual European symposium. Algorithms-Esa 2008, pp 466–477
10. Asano T, Asano Y (2000) Recent developments in maximum flow algorithms. *J Oper Res Soc Jpn* 1(43):2–31
11. Shiloach Y, Vishkin U (1982) An  $O(\text{Log}n)$  parallel connectivity algorithm. *J Algorithm* 1(3):57–67
12. Johnson DB (1987) Parallel algorithms for minimum cuts and maximum flows in planar networks. *J ACM* 4(34):950–967
13. Barbosa VC (1996) An introduction to distributed algorithms. MIT Press, Cambridge
14. David AB, Vipin S (2005) A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic. In: Proceedings of 18th ISCA international conference on parallel and distributed computing systems. Las Vegas, NV, pp 41–48

15. Hong B, He ZY (2011) An asynchronous multithreaded algorithm for the maximum network flow problem with nonblocking global relabeling heuristic. *IEEE Trans Parallel Distr* 6(22):1025–1033
16. Alonso P, Cortina R, Martinez-Zaldivar FJ, Ranilla J (2011) Neville elimination on multi- and many-core systems: OpenMP, MPI, CUDA. *J Supercomput* 2(58):215–225
17. Vineet V, Narayanan PJ (2008) CUDA cuts: fast graph cuts on the GPU. In: 2008 IEEE computer society conference on computer vision and pattern recognition workshops, Anchorage, vols 1–3, pp 1070–1077
18. He Z, Hong B (2010) Dynamically tuned push-relabel algorithm for the maximum flow problem on cpu-gpu-hybrid platforms. In: IEEE international symposium on parallel and distributed processing, Atlanta, pp 19–23
19. Jian LH, Wang C, Liu Y, Liang SS, Yi WD, Shi Y (2013) Parallel data mining techniques on Graphics Processing Unit with Compute Unified Device Architecture (CUDA). *J Supercomput* 3(64):942–967
20. Park SY, Hariri S (1997) A high performance message-passing system for network of workstations. *J Supercomput* 2(11):159–179
21. Delong A, Boykov Y (2008) A scalable graph-cut algorithm for N-D grids. In: IEEE conference on computer vision and pattern recognition, Anchorage, pp 946–953
22. Shekhovtsov A, Hlavac V (2013) A distributed mincut/maxflow algorithm combining path augmentation and push-relabel. *Int J Comput Vis* 3(104):315–342
23. Eleyat M, Haugland D, Hetland ML, Natvig L (2012) Parallel algorithms for the maximum flow problem with minimum lot sizes. In: Klatt D, Lüthi HJ, Karl S (eds) *Operations research proceedings 2011. Selected papers of the international conference on operations research (OR 2011), August 30–September 2, 2011, Zurich, Switzerland*, Springer, Berlin, Heidelberg, pp 83–88
24. Cherkassky BV, Goldberg AV (1997) On implementing the push-relabel method for the maximum flow problem. *Algorithmica* 4(19):390–410
25. Schloegel K, Karypis G, Kumar V (2002) Parallel static and dynamic multi-constraint graph partitioning. *Concurr Comput Pract E* 3(14):219–240
26. Ahuja RK, Kodialam M, Mishra AK, Orlin JB (1997) Computational investigations of maximum flow algorithms. *Eur J Oper Res* 3(97):509–542
27. Cormen TH (2009) *Introduction to Algorithms*. MIT Press, Cambridge
28. Badics T, Boros E, Cepek O (1993) Implementing a new maximum flow algorithm. In: Johnson DS, McGeoch CC (eds) *Network flows and matching: first DIMACS implementation challenge*, DIMACS series in discrete mathematics and theoretical computer science, vol 12. American Mathematical Society, Providence
29. Strandmark P, Kahl F (2010) Parallel and distributed graph cuts by dual decomposition. In: 2010 IEEE conference on computer vision and pattern recognition (CVPR), San Francisco, pp 2085–2092