Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Distributed sparse bundle adjustment algorithm based on three-dimensional point partition and asynchronous communication*

Xiao-long SHEN‡[1,2], Yong DOU[1,2], Steven MILLS[3], David M EYERS[3], Huan FENG[4], Zhiyi HUANG[3]

[1]*College of Computer, National University of Defense Technology, Changsha 410000, China*

[2]*Science and Technology on Parallel and Distributed Laboratory, National University of Defense Technology, Changsha 410000, China*

[3]*Department of Computer Science, University of Otago, Dunedin 9016, New Zealand*

[4]*Department of Computer Science, Tsinghua University, Beijing 100084, China*

E-mail: {shenxiaolong11, yongdou}@nudt.edu.cn; {steven, dme}@cs.otago.ac.nz; fenghuan517@gmail.com; zhuang@cs.otago.ac.nz

Received Mar. 20, 2018; Revision accepted July 2, 2018; Crosschecked July 13, 2018

Abstract: Sparse bundle adjustment (SBA) is a key but time- and memory-consuming step in three-dimensional (3D) reconstruction. In this paper, we propose a 3D point-based distributed SBA algorithm (DSBA) to improve the speed and scalability of SBA. The algorithm uses an asynchronously distributed sparse bundle adjustment (A-DSBA) to overlap data communication with equation computation. Compared with the synchronous DSBA mechanism (S-DSBA), A-DSBA reduces the running time by 46%. The experimental results on several 3D reconstruction datasets reveal that our distributed algorithm running on eight nodes is up to five times faster than that of the stand-alone parallel SBA. Furthermore, the speedup of the proposed algorithm (running on eight nodes with 48 cores) is up to 41 times that of the serial SBA (running on a single node).

Key words: Sparse bundle adjustment; Parallel; Distributed sparse bundle adjustment; Three-dimensional reconstruction; Asynchronous

https://doi.org/10.1631/FITEE.1800173 **CLC number:** TP312; TP217.4

1 Introduction

Recently, using the structure from motion (SfM) method to solve large-scale three-dimensional (3D) reconstruction problems has received considerable attention in the computer vision community (Snavely et al., 2006, 2008; Li et al., 2008; Agarwal et al., 2011; Zheng and Wu, 2015). SfM has been widely used in surveying and mapping, military reconnaissance, city-scale 3D reconstruction, etc. The SfM pipeline has several steps. The joint optimization of camera positions and point coordinates using the bundle adjustment (BA) algorithm is the last step. The BA process is the primary bottleneck in SfM. It consumes about half of the total computation time (Choudhary et al., 2010) and a great deal of memory. As the scale of a 3D reconstruction problem grows, the scalability of the BA algorithm becomes a critical issue. Speeding up the BA through parallelism can considerably improve the performance of SfM.

‡ Corresponding author

* Project supported by the National Natural Science Foundation of China (Nos. U1435219, U1435222, and 61572515), the National Key R & D Program of China (No. 2016YFB0200401), and the Major Research Plan of the National Key R & D Program of China (No. 2016YFC0901600)

ORCID: Xiao-long SHEN, http://orcid.org/0000-0002-6481-4287

Over the last decade, large-scale 3D reconstruction problems have been extensively studied to improve the computational efficiency and storage scalability. Lourakis and Argyros (2009) made full use of data sparsity in 3D reconstruction to reduce the time and space complexity of the BA algorithm. Agarwal et al. (2011) proposed a cluster-based distributed 3D reconstruction system. However, the BA algorithm in their system was operated on individual nodes. Ni et al. (2007) decomposed the original problem into several sub-maps and iteratively applied the BA algorithm to each sub-map and then to the global problem. However, due to the dependence upon the inherent relations among the input parameters, this method could solve only a few BA problems efficiently. Wu et al. (2011) improved two of Agarwal's PCG-based BA algorithms and implemented them on a multi-core and a single GPU, respectively. Frahm et al. (2010) proposed a multiple GPU based fast SfM system, where the BA algorithm was operated with an individual CPU thread without any GPU/CPU parallel acceleration. Choudhary et al. (2010) proposed a hybrid BA algorithm, where the Hessian matrix and Schur complements were generated in the GPU and others were computed using the CPU. Hänsch et al. (2016) improved the accuracy and the speed of three state-of-the-art BA algorithms by introducing numerical methods and GPU-based implementations. The scalability of the algorithms mentioned above (Choudhary et al., 2010; Frahm et al., 2010; Wu et al., 2011; Hänsch et al., 2016) was restricted by the memory space of the GPU. Liu et al. (2012) proposed a distributed BA algorithm to improve the efficiency and scalability, but their method was applicable mainly to the massive-point bundle adjustment (MPBA) problem. Eriksson et al. (2016) and Ramamurthy et al. (2017) proposed an optimization-based distributed BA. Although they distributed the camera parameters, the scalability of the algorithm would be limited by the number of 3D points, which was usually 2–4 orders of magnitude higher than the number of cameras.

The scalability and real-time issues in the BA algorithm are investigated in this study, and the contributions are summarized as follows:

1. A 3D point set partition algorithm, namely distributed sparse bundle adjustment (DSBA), is proposed to improve the scalability by eliminating data correlation within the BA problem.

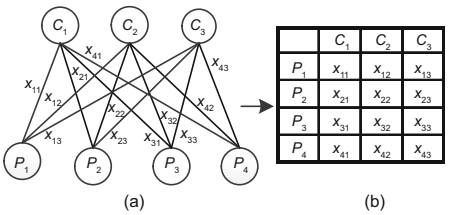
2. A circulant matrix based asynchronously distributed algorithm, namely asynchronously distributed sparse bundle adjustment (A-DSBA), is proposed to solve the equation set. A-DSBA develops an asynchronous communication protocol to reduce the Schur complement $\boldsymbol{S}$ (Lourakis and Argyros, 2009) during the equation solving process.

2 Background

In this section, we first introduce the 3D reconstruction model based on multiple views. Then, we briefly describe the BA algorithm used in the last step of the reconstruction.s

2.1 Three-dimensional reconstruction model based on multiple views

The principles of multiple view 3D reconstruction are similar to the observation of the world through human eyes. According to the principle of multiple view geometry (Hartley and Zisserman, 2000), if several images of a static scene are captured from several views and these images have content in common, it is theoretically possible to automatically restore the internal and external parameters of the camera, as well as the 3D information of the scene. For simplicity, assume that all points can be seen by every camera. Fig. 1a shows a simple network model for a multiple view 3D reconstruction, where the circle C_j at the top layer represents the j^{th} camera, and the circle P_i at the bottom layer represents the i^{th} 3D point. The straight line connecting a camera and a 3D point in the network model denotes a projection point and x_{ij} depicts the projection point of the i^{th} 3D point on the j^{th} camera. All the connecting lines, i.e., projection points, corresponding to the camera C_j, constitute an image. Fig. 1b shows the abstract data model.



	C_1	C_2	C_3
P_1	x_{11}	x_{12}	x_{13}
P_2	x_{21}	x_{22}	x_{23}
P_3	x_{31}	x_{32}	x_{33}
P_4	x_{41}	x_{42}	x_{43}

Fig. 1 A multiple view network model: (a) simple 3D reconstruction network model; (b) abstract data model

To facilitate the discussion below, the notations for the camera imaging model and the multiple view geometrical model are defined in Table 1. $\boldsymbol{a}_j$ and $\boldsymbol{b}_i$ denote the parameter vectors of the camera C_j and the 3D point P_i, respectively. $\boldsymbol{x}_{ij}$ and $\widehat{\boldsymbol{x}}_{ij}$ denote the measurement and measured vectors of the projection point x_{ij}, respectively.

2.2 Principles of sparse bundle adjustment using the Levenberg-Marquardt algorithm

The BA algorithm and the sparse bundle adjustment (SBA) algorithm are briefly described here. More details are available in Triggs et al. (1999) and Lourakis and Argyros (2009). Given a set of measured image feature locations $\boldsymbol{x}_{ij}$ and the initial parameter vector $\boldsymbol{p}$, i.e., $\boldsymbol{p}= (\boldsymbol{a}^{\mathrm{T}}, \boldsymbol{b}^{\mathrm{T}}) = (\boldsymbol{a}_1^{\mathrm{T}}, \boldsymbol{a}_2^{\mathrm{T}}, \cdots, \boldsymbol{a}_m^{\mathrm{T}}, \boldsymbol{b}_1^{\mathrm{T}}, \boldsymbol{b}_2^{\mathrm{T}}, \cdots, \boldsymbol{b}_m^{\mathrm{T}})$, the goal of the BA algorithm is to find the 3D point positions $\boldsymbol{b}_i$ and the camera parameters $\boldsymbol{a}_j$ that minimize the re-projection error (Triggs et al., 1999), which is denoted as

$$\min_{\boldsymbol{a}_i,\boldsymbol{b}_i} \sum_{i=1}^{n}\sum_{j=1}^{m} d\left(Q\left(\boldsymbol{a}_j,\ \boldsymbol{b}_i\right),\ \boldsymbol{x}_{ij}\right), \tag{1}$$

where $Q(\boldsymbol{a}_j, \boldsymbol{b}_i)$ is the predicted projection point of the i^{th} point on the j^{th} camera (where $\widehat{\boldsymbol{x}}_{ij} = Q(\boldsymbol{a}_j, \boldsymbol{b}_i)$, i=1, 2, $\cdots$, n, j=1, 2, $\cdots$, m), and $d(\boldsymbol{x}, \boldsymbol{y})$ denotes the Euclidean distance between the inhomogeneous image points represented by $\boldsymbol{x}$ and $\boldsymbol{y}$. The BA algorithm specifically minimizes the re-projection error with respect to all parameters of the 3D points and the cameras.

This optimization problem is usually formulated as a non-linear least squares problem, where the error is the squared L_2 norm of the difference between the observed feature location and the projection point of the corresponding 3D point on the image plane of the camera. The BA algorithm typically works iteratively, forming a linear sub-problem around the current solution, solving it, and repeating until it converges.

The BA algorithm keeps updating the parameter $\boldsymbol{p}$ by iteratively solving the following augmented normal equation:

$$\left(\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{J} + \mu\boldsymbol{I}\right)\boldsymbol{\delta}_{\boldsymbol{p}} = \boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{\epsilon}, \tag{2}$$

where $\boldsymbol{J} = \frac{\partial \hat{\boldsymbol{X}}}{\partial \boldsymbol{P}}$ is the Jacobian of Q, and $\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}$ is the covariance matrix expressing the uncertainty of the measured vector $\boldsymbol{x}$. $\boldsymbol{\delta}_{\boldsymbol{p}}$ is the sought update of the parameter vector $\boldsymbol{p}$, and $\boldsymbol{\epsilon} = \boldsymbol{x} - \hat{\boldsymbol{x}}$ over $\boldsymbol{p}$.

As there are a number of parameters involved, the BA algorithm incurs a high computation and memory cost. However, the lack of interaction among certain subgroups of the parameters makes the corresponding Jacobian sparse. Exploiting this features, SBA achieves considerable computational savings. Algorithm 1 shows the complete SBA algorithm.

From the steps given in Algorithm 1, it can be seen that SBA consists of the following stages. The first step is the initialization, which is followed by the computation of the Jacobian matrix. Then the computation of $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{J} + \mu\boldsymbol{I}$ and $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{\epsilon}$ is performed. They are followed by the computation of $\boldsymbol{S}$ and $\boldsymbol{e}_{\boldsymbol{a}}$. Subsequently, the linear equation $\boldsymbol{S}\boldsymbol{\delta}_{\boldsymbol{a}} = \boldsymbol{e}_{\boldsymbol{a}}$ is evaluated. Finally, the other computations are done. Considering a 3D model with 1936 cameras, Fig. 2 shows a distribution of the time consumption for each stage of SBA.

Fig. 2 indicates that the time consumption of the serial SBA algorithm is concentrated in the calculation of the Jacobian matrix, the calculation of $\boldsymbol{S}$ and $\boldsymbol{e}_{\boldsymbol{a}}$, and the solving of the equation set. The time complexities of these steps are $O(m^2 + n^2 + m \times n)$,

Table 1 Variable definitions of the serial SBA algorithm

Variable	Definition	Variable	Definition
m	Number of cameras	r	Number of nodes
n	Number of 3D points	l	The l^{th} node
cnp	Number of parameters defining a single camera	mnp	Number of parameters defining an image point (typically two)
pnp	Number of parameters defining a single 3D point	x_{ij}	Projection of the i^{th} 3D point on the j^{th} camera
$\boldsymbol{x}_{ij}$	Measured vector of x_{ij}	$\widehat{\boldsymbol{x}}_{ij}$	Measurement vector of x_{ij}
C_j	The j^{th} camera	$\boldsymbol{a}_j$	Parameter vectors of the j^{th} camera
P_j	The i^{th} 3D point	$\boldsymbol{b}_i$	Parameter vectors of the i^{th} 3D point

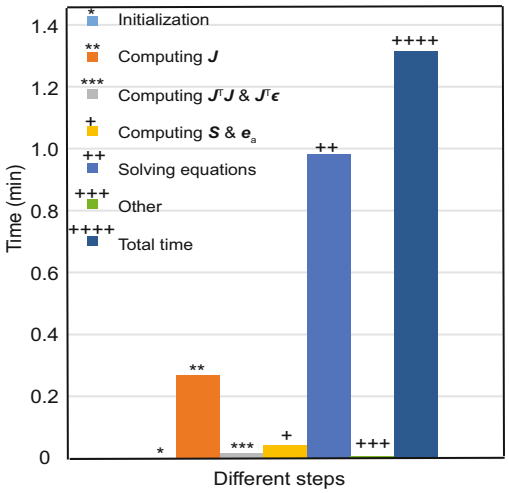


Fig. 2 Distribution of time consumption for each stage of sparse bundle adjustment

Algorithm 1 Serial SBA algorithm

Input:

vector of m initial camera parameters $\boldsymbol{a}_0$, n initial 3D point parameter vectors $\boldsymbol{b}_0$, function Q employing $\boldsymbol{a}_j$ and $\boldsymbol{b}_i$ to compute the predicted projection $\hat{\boldsymbol{x}}_{ij}$, measured projection $\boldsymbol{x}_{ij}$ damping term μ.

Output:

optimized camera parameter and 3D point parameter vectors $\boldsymbol{p}^{+}=\boldsymbol{p}_{k-1}+\boldsymbol{\delta}_k$.

1: Initialize measured value $\boldsymbol{x}$, $\boldsymbol{\epsilon}_0=\|\boldsymbol{x}-f(\boldsymbol{p}_0)\|$, $\boldsymbol{p}_0=\{\boldsymbol{a}_{01},\boldsymbol{a}_{02},\cdots,\boldsymbol{a}_{0m},\boldsymbol{b}_{01},\boldsymbol{b}_{02},\cdots,\boldsymbol{b}_{0m}\}$ and other threshold value

2: Compute the Jacobian matrix $\boldsymbol{J}=[A,B]$, where $A_{ij}=\frac{\partial\hat{x}_{ij}}{\partial a_j}=\frac{\partial Q(a_j,b_i)}{\partial a_j}$, $B_{ij}=\frac{\partial\hat{x}_{ij}}{\partial b_j}=\frac{\partial Q(a_j,b_i)}{\partial b_j}$

3: Compute

$$\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{J}+\mu\boldsymbol{I}=\begin{bmatrix}\boldsymbol{U}^{*} & \boldsymbol{W}\\ \boldsymbol{W}^{\mathrm{T}} & \boldsymbol{V}^{*}\end{bmatrix},\ \boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{\epsilon}=(\boldsymbol{\epsilon}_{\boldsymbol{a}},\boldsymbol{\epsilon}_{\boldsymbol{b}}).$$

Compute the following auxiliary variables:

$$\boldsymbol{U}_j:=\sum_{i=1}^{n}\boldsymbol{A}_{ij}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}}^{-1}\boldsymbol{A}_{ij},\ \boldsymbol{V}_i:=\sum_{j=1}^{m}\boldsymbol{B}_{ij}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}}^{-1}\boldsymbol{B}_{ij},$$

$$\boldsymbol{W}_{ij}:=\sum_{i=1,j=1}^{n,m}\boldsymbol{A}_{ij}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}}^{-1}\boldsymbol{B}_{ij},\ \boldsymbol{\epsilon}_{ij}=\boldsymbol{x}_{ij}-\hat{\boldsymbol{x}}_{ij},$$

$$\boldsymbol{\epsilon}_{\boldsymbol{a}_j}:=\sum_{i=1}^{n}{\boldsymbol{A}_{ij}}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}}^{-1}\boldsymbol{\epsilon}_{ij},\ \boldsymbol{\varepsilon}_{\boldsymbol{b}_i}:=\sum_{j=1}^{m}\boldsymbol{B}_{ij}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}}^{-1}\boldsymbol{\epsilon}_{ij},$$

$$\boldsymbol{U}_j^{*}\leftarrow\boldsymbol{U}_j+\mu\boldsymbol{I},\ \boldsymbol{V}_i^{*}\leftarrow\boldsymbol{V}_i+\mu\boldsymbol{I}$$

4: Compute $\boldsymbol{S}$ and $\boldsymbol{e}_{\boldsymbol{a}}$:

$$\boldsymbol{S}=\boldsymbol{U}^{*}-\boldsymbol{W}(\boldsymbol{V}^{*})^{-1}\boldsymbol{W}^{\mathrm{T}},\ \boldsymbol{e}_{\boldsymbol{a}}=\boldsymbol{\epsilon}_{\boldsymbol{a}}-\boldsymbol{W}(\boldsymbol{V}^{*})^{-1}\boldsymbol{\epsilon}_{\boldsymbol{b}}$$

5: Solve the linear equation $\boldsymbol{S}\boldsymbol{\delta}_{\boldsymbol{a}}=\boldsymbol{e}_{\boldsymbol{a}}$

6: Obtain $\boldsymbol{\delta}_{\boldsymbol{b}}$ from $\boldsymbol{V}\boldsymbol{\delta}_{\boldsymbol{b}}=\boldsymbol{\epsilon}_{\boldsymbol{b}}-\boldsymbol{W}^{\mathrm{T}}\boldsymbol{\delta}_{\boldsymbol{a}}$

7: If the termination condition is satisfied, update $\boldsymbol{p}$ with $\boldsymbol{p}+\boldsymbol{\delta}$, or jump to step 2

$O(m\times n\times(m+n))$, and $O(m^3)$, respectively. A distributed SBA (DSBA) algorithm is proposed in this study to improve the computational and storage efficiency of the SBA algorithm.

3 Overview of the DSBA algorithm

DSBA is composed mainly of distributed parameters of the 3D point and camera and distribution of both the point and camera parameters. In their research work, Ramamurthy et al. (2017) distributed both 3D points and camera parameters, while Eriksson et al. (2016) distributed only camera parameters. Our 3D point-based DSBA uses a 3D point-based segmentation scheme, which is simple, yet load balanced, and independent of the inherent relationships among the input parameters. Only a small amount of communication is required between each node in DSBA during the following steps: initialization, computation of $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{J}+\mu\boldsymbol{I}$, $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}}^{-1}\boldsymbol{e}$, $\boldsymbol{S}$, $\boldsymbol{e}_{\boldsymbol{a}}$, and a few others. Since the communication in the process of solving equations in DSBA is $(m\times\mathrm{cnp})^2\times(r-1)$, it has an exceptional time cost for a real-time system. This study proposes an A-DSBA which overlaps the communication and the solving of the equations when compared with the synchronous DSBA (S-DSBA). Some of the notations in Section 2 are improved to facilitate the representation of the data distribution in different nodes. Specifically, the superscript l denotes the data in the l^{th} node. Table 2 shows the new notations. The theoretical derivation regarding the data distribution and communication in DSBA is then discussed in detail.

4 Distributed scheme for SBA

4.1 Selection of the distributed storage method for D-SBA

Each camera in the 3D reconstruction model can see only some of the 3D points in its visual field. For simplicity, Fig. 3a shows a 3D reconstruction model with four cameras and six points. In Fig. 3a, each camera C_j can see the points P_i, P_{i+1}, and P_{i+2} forming a sparse structured graph. DSBA will first divide this graph up into r parts and assign the parts to r distributed nodes in the cluster. Suppose that there are two nodes ($r=2$) (Figs. 3b–3d). Then this graph can be distributed in three different methods.

Table 2 Definition of major parameters in the DSBA algorithm

Parameter	Definition	Parameter	Definition
l	Serial number of the subnodes, $l \in [1, r]$	n_l	The starting index of the 3D point in node l
m	Total number of cameras	m^l	Number of cameras in node l
n	Total number of 3D points	n^l	Number of 3D points in node l, and $n^l = n_{l+1} - n_l$
x_{ij}	Projection of the i^{th} 3D point in the j^{th} camera	x_{ij}^l	Projection of x_{ij} in node l
$\boldsymbol{x}_{ij}$	Measured vector of the projection point x_{ij}	$\boldsymbol{x}_{ij}^l$	Measured vector of the projection x_{ij}^l in node l
$\widehat{\boldsymbol{x}}_{ij}$	Measurement vector of the projection point x_{ij}	$\widehat{\boldsymbol{x}}_{ij}^l$	Measurement vector of the projection x_{ij}^l in node l
P_i	The i^{th} 3D point	P_i^l	The i^{th} 3D point in node l
$\boldsymbol{b}_i$	Parameter vector of the i^{th} 3D point	$\boldsymbol{b}_i^l$	Parameter vector of the i^{th} 3D point in node l
C_j	The j^{th} camera	C_j^l	The j^{th} camera in node l
$\boldsymbol{a}_j$	Parameter vector of the j^{th} camera	$\boldsymbol{a}_j^l$	Parameter vector of the j^{th} camera in node l

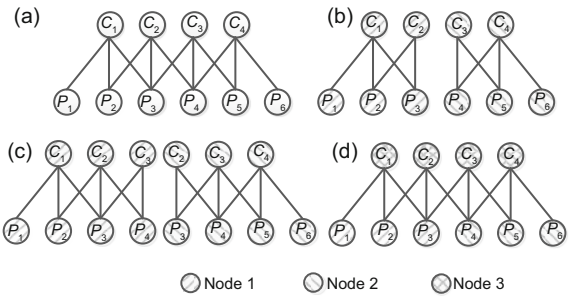


Fig. 3 Data/Task distribution methods for DSBA: (a) 3D reconstruction model; (b) the first distributed method; (c) the second distributed method; (d) the third distributed method

In Fig. 3b, each node has non-overlapping data. The projection points (shown as the lines C_2-P_4 and C_3-P_3) across the boundary are abandoned. In Fig. 3c, the parameters for the cameras (C_2, C_3), 3D points (P_3, P_4), and projection points corresponding to the boundary are copied to each of the relevant nodes. In Fig. 3d, each node has complete camera parameters, but the 3D points are uniformly assigned to two nodes, and the relevant projection points are distributed to the corresponding nodes. The first method could result in an incomplete model, because the data for some projection points is abandoned. The second method occasionally requires model alignment guarantee the consistency of the boundary data in each node. The amount of boundary data rapidly increases with the increase of the number of nodes. Thus, more storage space is needed for the duplicated boundary data, resulting in higher model complexity, more complex data dependence, and considerably higher computational overhead. For the third method, the number of 3D points is 2–4 orders of magnitude higher than the number of cameras. Compared with the second method, the increases in the memory space and communication overhead of the third method are relatively fixed and small, because they are mostly dependent on the number of camera parameters, which is much lower than the number of 3D points and projection points. In addition, the third method is easy to implement and it is independent of any specific BA problem.

Therefore, we adopt the third method for 3D point-based distributed storage in this study.

4.2 Data partition in DSBA

For a 3D model with n 3D points and m cameras, define a 3D point set $D_{\text{p}}=\{\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_n\}$, an actual projection point (true value) set $X=\{\boldsymbol{x}_{11}, \boldsymbol{x}_{12}, \cdots, \boldsymbol{x}_{1m}, \boldsymbol{x}_{21}, \boldsymbol{x}_{22}, \cdots, \boldsymbol{x}_{2m}, \cdots, \boldsymbol{x}_{n1}, \boldsymbol{x}_{n2}, \cdots, \boldsymbol{x}_{nm}\}$, and a camera parameter set $D_{\text{c}}=\{\boldsymbol{a}_1, \boldsymbol{a}_2, \cdots, \boldsymbol{a}_n\}$. According to the distributed method in Fig. 3d, the 3D point set is partitioned into r non-empty disjoint subsets $D_{\text{p}}^1, D_{\text{p}}^2, \cdots, D_{\text{p}}^r$, where $D_{\text{p}}^r=\{b_i^l | i=1, 2, \cdots, n^l\}$. Similarly, the projection point set X is partitioned into r subsets $X^1, X^2, \cdots, X^r$. Let D_{c}^l denote the set of camera data in the l^{th} node. From the distributed method depicted in Fig. 3d, it can be seen that all sets satisfy the following Eq. (3).

As the camera parameters are not partitioned, each node is set up with the same set of camera parameters. However, different nodes will obtain different camera parameter values during the subsequent

calculation; the superscript l is still used to distinguish the data on the different nodes.

$$\begin{aligned}
&\text{Number of cameras:}\\
&\quad m^1=m^2=\cdots=m^r=m.\\
&\text{Number of 3D points:}\\
&\quad n^1+n^2+\cdots+n^r=n,\\
&\quad n_{l+1}-n_l=n^l, l\in[1,r-1],\\
&\quad n_1=1.\\
&\text{Data set of 3D points:}\\
&\quad D_{\mathrm{p}}^1\cup D_{\mathrm{p}}^2\cup\cdots\cup D_{\mathrm{p}}^r=D_{\mathrm{p}},\\
&\quad D_{\mathrm{p}}^1\cap D_{\mathrm{p}}^2\cap\cdots\cap D_{\mathrm{p}}^r=\varnothing.\\
&\text{Data set of cameras:}\\
&\quad D_{\mathrm{c}}^1=D_{\mathrm{c}}^2=\cdots=D_{\mathrm{c}}^r=D_{\mathrm{c}}.\\
&\text{Data set of projections:}\\
&\quad X^1\cup X^2\cup\cdots\cup X^r=X,\\
&\quad X^1\cap X^2\cap\cdots\cap X^r=\varnothing.
\end{aligned}\tag{3}$$

5 Data distribution and communication during the key steps in DSBA

5.1 Initialization

Initializing DSBA involves two major steps. First, the master node reads the initial data from the 3D point parameters, camera parameters, projection point parameters, and the number of nodes. Eq. (3) is used to partition the dataset into r subsets, which are then distributed to each of the child nodes. Second, each of the child nodes is initialized after receiving the subsets.

After initialization, the initial data of node l includes the 3D point parameter vector $\boldsymbol{b}_i^l$, the initial value of the measured vector $\boldsymbol{x}_{ij}^l$, the camera parameter vector $\boldsymbol{a}_j^l$, the number of cameras m^l, the number of 3D points n^l, and the number of nodes r.

5.2 Computation of Jacobian matrix J

Let $\boldsymbol{A}_{ij}^l=\frac{\partial\hat{\boldsymbol{x}}_{ij}^l}{\partial\boldsymbol{p}_{\boldsymbol{a}_j^l}}$ denote the partial derivative of the $(ij)^{\text{th}}$ projection point in the l^{th} node with respect to the j^{th} camera, and $\boldsymbol{B}_{ij}^l=\frac{\partial\hat{\boldsymbol{x}}_{ij}^l}{\partial\boldsymbol{p}_{\boldsymbol{b}_i^l}}$ denote the partial derivative of the $(ij)^{\text{th}}$ projection point in the l^{th} node with respect to the i^{th} 3D point. Also, let $\boldsymbol{A}_i^l=\operatorname{diag}\left(\boldsymbol{A}_{i1}^l,\ \boldsymbol{A}_{i2}^l,\ \cdots,\ \boldsymbol{A}_{in}^l\right)$ denote the Jacobian matrix of the estimated value of the projection point corresponding to the i^{th} 3D point in the l^{th} node with respect to the camera parameters, where i=$n_l, n_l+1, \cdots, n_l+n^l-1$. Define $\boldsymbol{B}_i^l=\begin{pmatrix}\boldsymbol{0}&\boldsymbol{B}_{i1}^l&\boldsymbol{0}\\ \boldsymbol{0}&\boldsymbol{B}_{i2}^l&\boldsymbol{0}\\ \vdots&\vdots&\vdots\\ \boldsymbol{0}&\boldsymbol{B}_{im}^l&\boldsymbol{0}\end{pmatrix}$ as the Jacobian matrix of the estimated value of the projection point corresponding to the i^{th} 3D point in the l^{th} node with respect to the 3D point parameters, where i=$n_l, n_l+1, \cdots, n_l+n^l-1, j=1,2,\cdots,n^l$. Based on the matrix sparsity (Lourakis and Argyros, 2009) of the SBA algorithm, the Jacobian matrix $\boldsymbol{J}$ can be written as

$$\boldsymbol{J}=\frac{\partial\hat{\boldsymbol{X}}}{\partial\boldsymbol{P}}=\begin{pmatrix}
\begin{pmatrix}\boldsymbol{B}_1^1&\boldsymbol{A}_1^1\\ \boldsymbol{B}_2^1&\boldsymbol{A}_2^1\\ \vdots&\vdots\\ \boldsymbol{B}_{n_1+n^1-1}^1&\boldsymbol{A}_{n_1+n^1-1}^1\end{pmatrix}\\
\begin{pmatrix}\boldsymbol{B}_{n_2}^2&\boldsymbol{A}_{n_2}^2\\ \boldsymbol{B}_{n_2+1}^2&\boldsymbol{A}_{n_2+1}^2\\ \vdots&\vdots\\ \boldsymbol{B}_{n_2+n^2-1}^2&\boldsymbol{A}_{n_2+n^2-1}^2\end{pmatrix}\\
\vdots\\
\begin{pmatrix}\boldsymbol{B}_{n_r}^r&\boldsymbol{A}_{n_r}^r\\ \boldsymbol{B}_{n_r+1}^r&\boldsymbol{A}_{n_r+1}^r\\ \vdots&\vdots\\ \boldsymbol{B}_{n_r+n^r-1}^r&\boldsymbol{A}_{n_r+n^r-1}^r\end{pmatrix}
\end{pmatrix}.\tag{4}$$

Let

$$\boldsymbol{A}^l=\begin{pmatrix}\boldsymbol{A}_{n_l}^l\\ \boldsymbol{A}_{n_l+1}^l\\ \vdots\\ \boldsymbol{A}_{n_l+n^l-1}^l\end{pmatrix},\quad \boldsymbol{B}^l=\begin{pmatrix}\boldsymbol{B}_{n_l}^l\\ \boldsymbol{B}_{n_l+1}^l\\ \vdots\\ \boldsymbol{B}_{n_l+n^l-1}^l\end{pmatrix},$$

$\boldsymbol{J}^l=\begin{pmatrix}\boldsymbol{B}^l&\boldsymbol{A}^l\end{pmatrix}$. Then Eq. (4) can be simplified as

$$\boldsymbol{J}=\left(\boldsymbol{J}^{1\,\mathrm{T}},\boldsymbol{J}^{2\,\mathrm{T}},\cdots,\boldsymbol{J}^{r\,\mathrm{T}}\right)^{\mathrm{T}}.\tag{5}$$

In Eqs. (4) and (5), the Jacobian matrix $\boldsymbol{J}$ is divided into r block matrices. The term in the first bracket of Eq. (4) is the Jacobian matrix $\boldsymbol{J}^l$ of node l, which is the matrix that needs to be computed. In Section 4.2, it can be seen that the node l locally stores the data that is required to compute $\boldsymbol{J}^l$, including the estimated value of the

projection point $\boldsymbol{x}_{ij}^l$, the 3D point parameter vector $\boldsymbol{b}_i^l$, and the camera parameter vector $\boldsymbol{a}_j^l$, where $i \in [n_l,\ n_l + n^l - 1],\quad j \in [1,\ 2,\ \cdots,\ m]$. Therefore, $\boldsymbol{J}^l$ can be computed without any data from any other node, and there is no need to send any data to the other nodes either.

5.3 Computations of $\boldsymbol{J^{\mathrm{T}}\Sigma_x^{-1}J+\mu J}$ and $\boldsymbol{J^{\mathrm{T}}\Sigma_x^{-1}\epsilon}$

In Algorithm 1, the Jacobian matrices $\boldsymbol{A}^l$ and $\boldsymbol{B}^l$ computed in Section 4.2 for each child node are the input for the computation, and they involve three components, i.e., $\boldsymbol{U}^*$, $\boldsymbol{V}^*$, and $\boldsymbol{W}$.

5.3.1 Computation of $\boldsymbol{U}$

According to the data distribution scenario in Section 4.2, the matrix $\boldsymbol{U}_j$ in Algorithm 1 can be written as

$$\begin{aligned}\boldsymbol{U}_j &:= \sum_{i=1}^{n} \boldsymbol{A}_{ij}^{\mathrm{T}} \boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}}^{-1} \boldsymbol{A}_{ij} \\ &= \sum_{l=1}^{r} \sum_{i=n_l}^{n_l+n^l-1} {\boldsymbol{A}_{ij}^l}^{\mathrm{T}} \boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}^l}^{-1} \boldsymbol{A}_{ij}^l.\end{aligned} \tag{6}$$

Let the matrix, $\boldsymbol{U}_j^l := \sum_{i=n_l}^{n_l+n^l-1} {\boldsymbol{A}_{ij}^l}^{\mathrm{T}} \boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}^l}^{-1} \boldsymbol{A}_{ij}^l$, $(j=1, 2, \cdots, m)$, denote the matrix $\boldsymbol{U}$ computed in the l^{th} node. There is no need for data communication as the data required to compute $\boldsymbol{U}_j^l$ is locally available on node l. Substituting $\boldsymbol{U}_j^l$ into Eq. (6), we have

$$\boldsymbol{U}_j := \sum_{l=1}^{r} \boldsymbol{U}_j^l. \tag{7}$$

Thus, the matrix $\boldsymbol{U}_j$ of the j^{th} camera in the serial algorithm amounts to the sum of $\boldsymbol{U}_j^l$ in each node. Hence, computing the matrix $\boldsymbol{U}$ involves computing the array sum of reduction of $\boldsymbol{U}_j^l$ from all child nodes. The data from every child node should be transferred to the master node for summation.

5.3.2 Computations of $\boldsymbol{V}$ and $\boldsymbol{W}$

Algorithm 1 indicates that the expansion of $\boldsymbol{V}$ is

$$\boldsymbol{V} = \begin{pmatrix} \boldsymbol{V}_1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}_2 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{V}_n \end{pmatrix}, \tag{8}$$

where $\boldsymbol{V}_i = \sum_{j=1}^{m} \boldsymbol{B}_{ij}^{\mathrm{T}} \boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}}^{-1} \boldsymbol{B}_{ij}, i = 1, 2, \cdots, n.$

Define $\boldsymbol{V}^l$ as the matrix of node l. Thus, we have

$$\boldsymbol{V}^l = \begin{pmatrix} \boldsymbol{V}_{n_l} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}_{n_l+1} & \cdots & \boldsymbol{0} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{V}_{n_l+n^l-1} \end{pmatrix}. \tag{9}$$

Substituting Eq. (9) into Eq. (8), we have

$$\boldsymbol{V} = \begin{pmatrix} \boldsymbol{V}^1 & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{V}^2 & \cdots & \boldsymbol{0} \\ \vdots & \vdots & & \vdots \\ \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{V}^r \end{pmatrix}. \tag{10}$$

It can be seen from Section 4.2 that the computation of $\boldsymbol{V}^l$ involves the Jacobian matrix $\boldsymbol{B}_{ij}$, $j = 1, 2, \cdots, m$, $i \in [n_l, n_l+n^l-1]$, which is obtained from node l. From Algorithm 1, it can be seen that the block $\boldsymbol{V}^l$ is not used in the other nodes. Therefore, the matrix $\boldsymbol{V}$ can be computed without data from other nodes, and there is no need to transmit data to any other node. In Eq. (10), the calculation of matrix $\boldsymbol{V}$ is distributed to r nodes, each of which is responsible for computing a part of the diagonal matrix $\boldsymbol{V}$, thereby eliminating the data communication during the calculation.

Similarly, it can be seen that the distribution and the calculation of matrix $\boldsymbol{W}$ are the same for the matrix $\boldsymbol{V}$, as the local nodes suffice for the computation and there is no need for communication among the nodes. The distribution and calculation of matrix $\boldsymbol{e_a}$ are the same those for matrix $\boldsymbol{U}$; node l needs to perform partial summary operations on matrix $\boldsymbol{e_a}$ and transmit the result to the master node, where $\boldsymbol{e_a}$ is computed by summation.

From what has been discussed above, the computations of matrices $\boldsymbol{U}$ and $\boldsymbol{V}$ can be illustrated using Figs. 4a and 4c. In Figs. 4a and 4c, each of the child nodes computes matrix $\boldsymbol{U}^l$, which has the same size as $\boldsymbol{U}$. Then the master node obtains matrix $\boldsymbol{U}$ by computing the array sum of reduction of data from all the child nodes. For matrix $\boldsymbol{V}$, each of the child nodes is used to compute the matrix $\boldsymbol{V}^l$, and the matrix $\boldsymbol{V}$ is partitioned into r blocks which are distributed among the child nodes. The distribution and calculation mechanism of matrix $\boldsymbol{W}$ are the same as those of $\boldsymbol{V}$. Note that the size of $\boldsymbol{U}$ is determined by the number of cameras m, the size of $\boldsymbol{V}$ is determined by the number of 3D points n, and

the size of $\boldsymbol{W}$ is jointly determined by m and n. In the 3D model, the value of n is usually 2–4 orders of magnitude higher than the value of m. Therefore, the size of $\boldsymbol{U}$ is much smaller than that of $\boldsymbol{V}$ and $\boldsymbol{W}$. In the proposed algorithm, the larger matrices $\boldsymbol{V}$ and $\boldsymbol{W}$ are distributed to the different nodes to improve the spatial scalability and efficiency of the algorithm.

5.4 Computations of S and e_a

Since $\boldsymbol{S}=\boldsymbol{U}-\boldsymbol{W}\boldsymbol{V}^{-1}\boldsymbol{W}^{\mathrm{T}}$, this computation process consists of two steps: computing $\boldsymbol{Y}=\boldsymbol{W}\boldsymbol{V}^{-1}\boldsymbol{W}^{\mathrm{T}}$ and computing $\boldsymbol{U}-\boldsymbol{Y}$.

5.4.1 Computation of $\boldsymbol{Y}$

Let $\boldsymbol{Y}_j^l=\sum_{i=n_l}^{n_l+n^l-1}\boldsymbol{W}_{ij}^l(\boldsymbol{V}_i^l)^{-1}(\boldsymbol{W}_{ij}^l)^{\mathrm{T}}$, and we have

$$\boldsymbol{Y}_j=\sum_{i=1}^{n}\boldsymbol{W}_{ij}\left(\boldsymbol{V}_i^l\right)^{-1}\boldsymbol{W}_{ij}^{\mathrm{T}}=\sum_{l=1}^{r}\boldsymbol{Y}_j^l. \tag{11}$$

Substituting $\boldsymbol{W}_{ij}:=\sum_{i=1,j=1}^{n,m}\boldsymbol{A}_{ij}^{\mathrm{T}}\boldsymbol{\Sigma}_{\boldsymbol{x}_{ij}}^{-1}\boldsymbol{B}_{ij}$ into $\boldsymbol{Y}_j^l$,

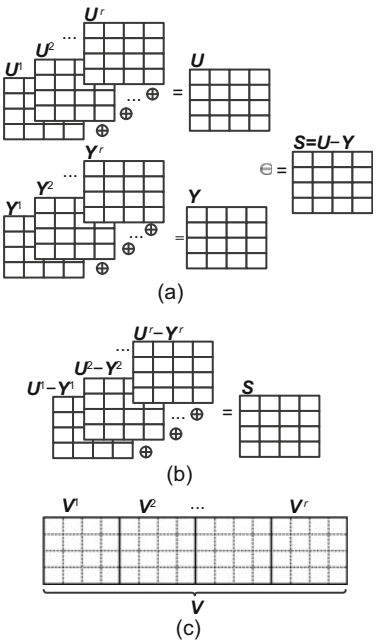


Fig. 4 Data distribution of U and V: (a) computation of matrix U; (b) computation of $S=U-Y$; (c) computation of matrix V

we have

$$\boldsymbol{Y}_j^l=\sum_{i=n_l}^{n_l+n^l-1}(\boldsymbol{A}_{ij}^l)^{\mathrm{T}}\boldsymbol{B}_{ij}^l(\boldsymbol{V}_i^l)^{-1}(\boldsymbol{B}_{ij}^l)^{\mathrm{T}}\boldsymbol{A}_{ij}^l. \tag{12}$$

From the discussions in Sections 4.2 and 5.3, it can be seen that the matrices $(\boldsymbol{A}_{ij}^l)^{\mathrm{T}}$, $\boldsymbol{B}_{ij}^l$, and $(\boldsymbol{V}_i^l)^{-1}$, which are needed to compute $\boldsymbol{Y}_j^l$ in Eq. (12), have been obtained in node l. Hence, $\boldsymbol{Y}_j^l$ can be computed without any communication with the other nodes.

Interestingly, the comparison of Eq. (11) with Eq. (7) reveals that the calculation and distribution mechanism of $\boldsymbol{Y}$ are the same as those of $\boldsymbol{U}$. It is required to compute the array sum of reduction of data in all the child nodes.

5.4.2 Computation of $\boldsymbol{S}=\boldsymbol{U}-\boldsymbol{Y}$

According to $\boldsymbol{S}=\boldsymbol{U}-\boldsymbol{Y}$, it can be seen that the matrices $\boldsymbol{S}$, $\boldsymbol{U}$, and $\boldsymbol{Y}$ are of the same size. According to Eqs. (7) and (11), we have

$$\boldsymbol{S}=\boldsymbol{U}-\boldsymbol{Y}=\sum_{l=1}^{r}\boldsymbol{U}_j^l-\boldsymbol{Y}_j^l. \tag{13}$$

Because the DSBA algorithm requires the original matrix $\boldsymbol{S}$ alone rather than the original matrices $\boldsymbol{U}$ and $\boldsymbol{Y}$, we can first compute $\boldsymbol{S}^l=\boldsymbol{U}^l-\boldsymbol{Y}^l$ using the local data, and subsequently obtain the matrix $\boldsymbol{S}$ by computing the sum of $\boldsymbol{S}^l$ in all the child nodes via Eq. (14):

$$\boldsymbol{S}=\sum_{l=1}^{r}\boldsymbol{S}^l. \tag{14}$$

This process is illustrated in Fig. 4b.

Similarly, to compute the error matrix $\boldsymbol{e_a}$, the array sum of reduction of the results from all child nodes is required. The data calculation and distribution process of $\boldsymbol{e_a}$ are identical to those of matrix $\boldsymbol{U}$.

In summary, there is no need for data communication to compute the matrices $\boldsymbol{J}$, $\boldsymbol{U}$, $\boldsymbol{V}$, and $\boldsymbol{W}$. However, to compute matrices $\boldsymbol{S}$ and $\boldsymbol{e_a}$, it is necessary to compute the array sum of reduction of the results from all the child nodes.

5.5 Computation of δ_a by solving $S\delta_a=e_a$

The analyses in Section 5.4 indicate that prior to reducing the matrices $\boldsymbol{S}$ and $\boldsymbol{e_a}$, each node has

a submatrix of the same size. The array sum of reductions of the two matrices is required to solve the augmented positive definite equation. There are two reduction methods to solve the equation set: synchronous reduction and asynchronous blocked cyclic-based reduction (Husbands and Yelick, 2007) .

Now we introduce how to solve the equation set using the synchronous reduction method.

After the child node obtains $\boldsymbol{S}^l$ and $\boldsymbol{e}_{\boldsymbol{a}}^l$, each node will be synchronized, and the array sum of reduction of matrices $\boldsymbol{S}$ and $\boldsymbol{e}_{\boldsymbol{a}}$ will be computed to solve the equation set.

The DSBA algorithm that solves the equation set using the synchronous reduction method is known as the synchronous DSBA (S-DSBA) method. In S-DSBA, reducing $\boldsymbol{S}$ and $\boldsymbol{e}_{\boldsymbol{a}}$ involves the transmission of double typed $(m \times \text{cnp})^2 \times (r-1)$ and $m \times \text{cnp}$, respectively, at each iteration. As matrix $\boldsymbol{e}_{\boldsymbol{a}}$ can be regarded as a vector of matrix $\boldsymbol{S}$, our attention is focused on the reduction of $\boldsymbol{S}$ instead. Consider an example of a 3D model, where $m = 500$ and $\text{cnp} = 10$. If there are six nodes in the cluster, then each round of reduction involves the transmisstion of 1 GB data. In Fig. 5, the amount of traffic increases with the increase of the scale of the problem and the number of nodes. The overhead for the communication to obtain the sum of reduction is proportional to the square of the number of cameras m and the number of nodes r. This communication overhead and time cost are too high to meet the real-time requirement of the system. Regarding this problem, in Section 6, a blocked cyclic based asynchronous reduction method is proposed to solve the equation set, and the corresponding DSBA is called 'asynchronous DSBA' (A-DSBA).

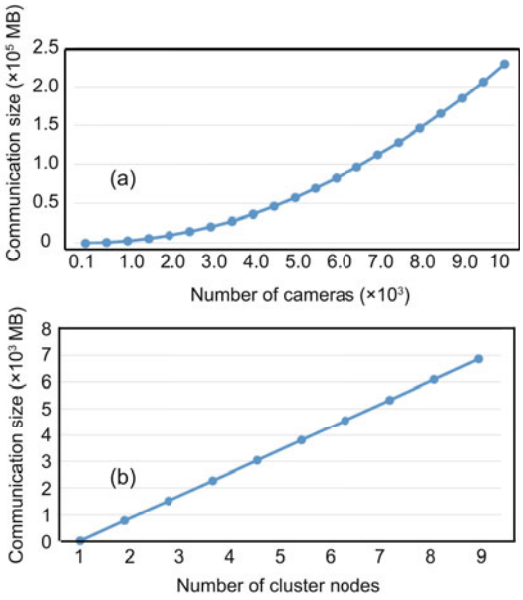


Fig. 5 Variation in the amount of traffic associated with the reduction of S: (a) r=4 and cnp=10; (b) m=1000 and cnp=10

6 Asynchronously distributed sparse bundle adjustment algorithm

The distributed method in Section 5 improves the speed and scalability of SBA. However, the analysis in Section 5.5 reveals that each iteration in S-DSBA involves reducing the matrix $\boldsymbol{S}$. With increases in the scale of the problem and the number of nodes, the communication and storage overheads become extensive. To solve this problem, A-DSBA is proposed. A-DSBA can be described in t iterations. Each of the iterations has two stages: matrix block reduction and calculation task. First, we introduce the general blocked cyclic based distribution method which A-DSBA adopts to solve the equation set. Second, we discuss the two stages of A-DSBA, and how to make asynchronous the reduction of matrices $\boldsymbol{S}$ and $\boldsymbol{e}_{\boldsymbol{a}}$, and also the solving of the equation set.

Before describing A-DSBA, the notations involved are defined in Table 3.

Table 3 Notation for A-DSBA

Notation	Definition
Goal	Solve equations $\boldsymbol{S\delta}=\boldsymbol{e}_{\boldsymbol{a}}$
r	Number of cluster nodes
N	Size of matrix $\boldsymbol{S}$
NB	Size of the blocks
P_n	Number of rows of the processor grid
Q_n	Number of columns of the processor grid
$\boldsymbol{S}_{ij}$	The block of $\boldsymbol{S}$ in the i^{th} row and j^{th} column
$\boldsymbol{S}_{ij}^l$	The block of $\boldsymbol{S}_{ij}$ in node l
$\lceil\rceil$	Rounding up

6.1 Solving the equation set using the blocked cyclic based distributed method

The blocked cyclic based distributed method is an iterative algorithm which divides the matrix $\boldsymbol{S}$ into several blocks of the same size. The blocks are then cyclically distributed to the grid of the processor. Then the distributed LU decomposition is

iteratively performed on these blocks. Finally, the solution to the equation set is obtained through two rounds of back substitution (Husbands and Yelick, 2007; Heinecke et al., 2013; Jo et al., 2015).

The partitioning and distribution of matrix $\boldsymbol{S}$ to the grid of the processor are as follows: the $N \times N$ matrix $\boldsymbol{S}$ is partitioned into $\lceil N/\mathrm{NB} \rceil \times \lceil N/\mathrm{NB} \rceil$ blocks, each of which is NB × NB in size (Fig. 6). Let $t = \lceil N/\mathrm{NB} \rceil$. The $t \times t$ blocks are cyclically distributed to the $P_n \times Q_n$ grid of the processor. For simplicity, a 2×2 processor grid (i.e., $P_n = Q_n = 2$) is taken as an example here.

In Fig. 6, different colors and background patterns are used to distinguish different nodes. After matrix $\boldsymbol{S}$ is partitioned into $t \times t$ blocks, the 2×2 block matrices in the upper left corner are used to cyclically distribute $\boldsymbol{S}$ into the 2×2 processor grid.

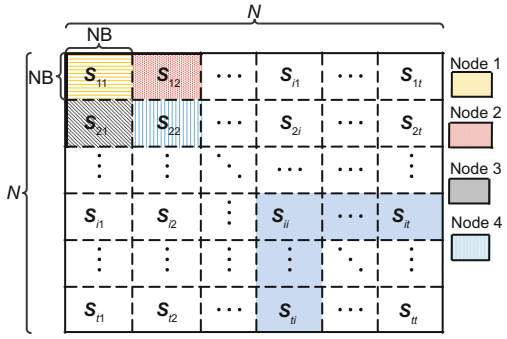


Fig. 6 The blocked cyclic based distributed method for solving the equation set

An analysis of the process in the blocked cyclic based method indicates that the i^{th} iteration involves only $\boldsymbol{S}_{ii}$, the right matrices from $\boldsymbol{S}_{i(i+1)}$ to $\boldsymbol{S}_{it}$, and the lower matrices from $\boldsymbol{S}_{(i+1)i}$ to $\boldsymbol{S}_{ti}$, while all the other matrices do not need to be reduced. This inspired us to propose the blocked cyclic based asynchronous reduction DSBA algorithm (A-DSBA) to solve the equation set. In the subsequent section, we describe A-DSBA in detail.

6.2 Data distribution and communication between nodes in A-DSBA

In A-DSBA, the reduction operation is asynchronous with the calculation of the lower-upper (LU) triangular matrices. According to Section 6.1, matrix $\boldsymbol{S}$ is separately reduced t times on different nodes. In i^{th} iteration, the corresponding nodes gather the matrices of $\boldsymbol{S}_{ii}$, $\boldsymbol{S}_{i(i+1)}, \cdots$, $\boldsymbol{S}_{it}$, $\boldsymbol{S}_{(i+1)i}, \cdots$, $\boldsymbol{S}_{ti}$ from other nodes. Define the process as the matrix block reduction stage. Then the node performs the calculation task, which computes mainly the LU decomposition of the reduced result obtained in the first stage. In the mean time of the decomposition, the matrix block reduction of the $(i+1)^{\mathrm{th}}$ iteration is produced on the corresponding node. Consequently, the reduced communication overlaps with the calculation so that the time consumption is reduced compared with that of S-DSBA.

Inspired by the blocked cyclic based distribution method, we assign reduction tasks to each node. Fig. 7 shows the $\boldsymbol{S}_{ii}$ reduction tasks distribution of the 9×9 block in the 2×2 processor grid, i.e., $t = 9$ and $P_n = Q_n = 2$. In Fig. 7, node 1 is responsible only for the reduction of the blocks whose row and column numbers are odd (i.e., $\boldsymbol{S}_{11}$, $\boldsymbol{S}_{13}$, etc.). Similarly, node 2 is responsible only for the blocks whose row numbers are odd and column numbers are even (i.e., $\boldsymbol{S}_{12}$, $\boldsymbol{S}_{34}$, etc.); node 3 is responsible only for the blocks whose row numbers are even and column numbers are odd (i.e., $\boldsymbol{S}_{21}$, $\boldsymbol{S}_{43}$, etc.); node 4 is responsible only for the blocks whose row and column numbers are even (i.e., $\boldsymbol{S}_{22}$, $\boldsymbol{S}_{24}$, etc.).

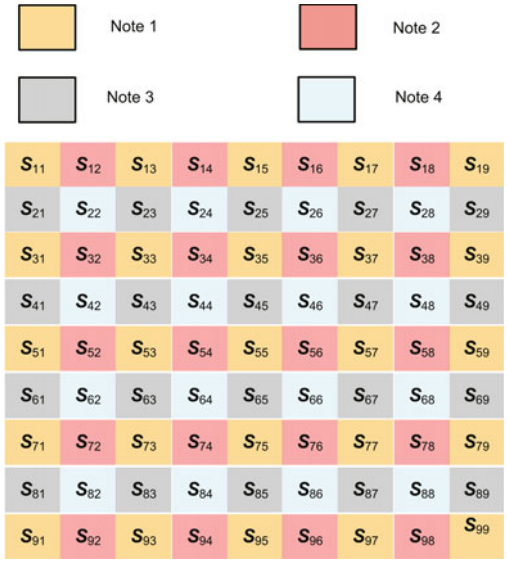


Fig. 7 Task allocation to different nodes during the reduction of matrix $\boldsymbol{S}$

In A-DSBA, the matrix $\boldsymbol{S}$ is divided into different parts allocated in the different nodes. Fig. 8 shows that each of the four nodes holds a part of matrix $\boldsymbol{S}_{ij}$ labeled as $\boldsymbol{S}_{ij}^{l}$, where l represents the index

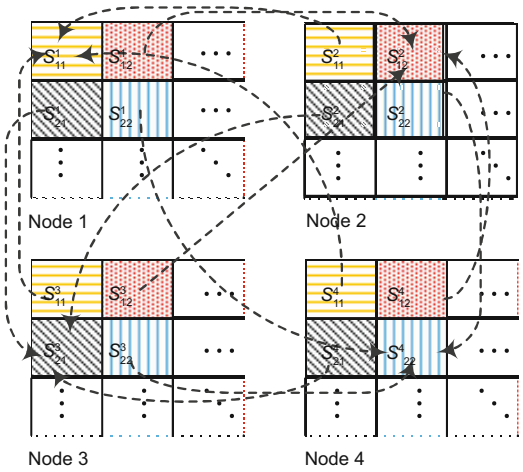


Fig. 8 Communication between nodes during the reduction of matrix $\boldsymbol{S}$

The arrowed dash line represents the direction of data transmission, the end of the line denotes the sender, and the arrow points to the receiver

of the node. Thus, before the reduction, all parts of $\boldsymbol{S}_{ij}$ need to be transferred to the node for the reduction task of $\boldsymbol{S}_{ij}$. The arrowed dash line represents the direction of data transmission, the end of the line denotes the sender, and the arrow points to the receiver. For instance, as the block $\boldsymbol{S}_{21}$, which is at the second row and first column of $\boldsymbol{S}$, is reduced at node 3, nodes 1, 2, and 4 need to send their corresponding blocks $\boldsymbol{S}_{21}^1$, $\boldsymbol{S}_{21}^2$, and $\boldsymbol{S}_{21}^4$ to node 3. After obtaining all the partial blocks, node 3 performs the array sum of reduction of $\boldsymbol{S}_{21}$. Note that each node is responsible for the reduction of a different matrix block from $\boldsymbol{S}_{11}$ to $\boldsymbol{S}_{22}$.

From Figs. 7 and 8, we can see that the reduction tasks of $\boldsymbol{S}_{ii}$ are asynchronous and load-balanced. Furthermore, before the $(i+1)^{\text{th}}$ LU decomposition, all the reduction results of the needed matrices are computed and transferred, at which time the i^{th} LU decomposition is being performed. Thus, the time spent on waiting for reduction is reduced by nearly 50%.

6.3 Sequential chart of asynchronous reduction in A-DSBA

In Fig. 9, the distribution of data in Fig. 7 is taken as an example to illustrate the relationship between reducing $\boldsymbol{S}$ and solving the equation set.

Due to the lack of space, the node index l for the notation $\boldsymbol{S}_{ij}^l$ in Fig. 9 is omitted. The notation '/' indicates that the node has no calculation or communication task during time t_i. The block with the blank background in time t_i denotes the process of the matrix block reduction of the i^{th} iteration, while the block with the gray background represents the calculation task of the $(i-1)^{\text{th}}$ iteration needed to solve the equation set.

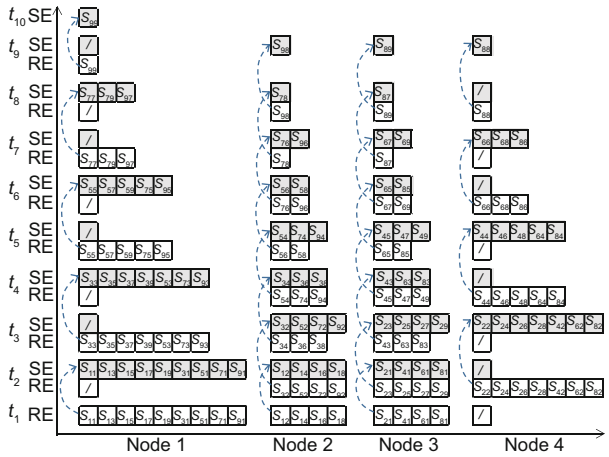


Fig. 9 Sequential chart showing how to solve the equation set based on asynchronous reduction

In Fig. 9, the distributed solving of the equation set based on asynchronous reduction has three characteristics: (1) The number of matrix blocks that need to be processed during each iteration decreases with time. Considering the example here, when the numbers of iterations increases, the numbers of matrix blocks that need to be processed by the cluster are 17, 15, 13, 11, 9, 7, 5, 3, and 1, respectively. This indicates that when the iteration proceeds, the work load associated with each iteration is decreased, until the size of the allocated block is smaller than the pre-set threshold. (2) Each node reduces the matrix blocks needed for the first iteration in the initial stage at time t_1. From time t_2 onward, each node should first initiate the reduction of matrix blocks needed in the current iteration. During the reduction, the equation set for the last iteration is solved at the same time. (3) It can be seen from the arrow in Fig. 9 that the matrix block obtained from the i^{th} array sum of reduction at time t_i will be used as the input to solve the $(i+1)^{\text{th}}$ equation set during the next time t_{i+1}.

6.4 Summary of A-DSBA

By reducing the matrix blocks in an asynchronous and time-sharing manner, A-DSBA hides

the communication time needed by DSBA to reduce the matrix $\boldsymbol{S}$, resulting in improved time performance. The algorithm steps are in Algorithm 2.

In Algorithm 2, the left part denotes the task of the master node, and the right part denotes the task of the child node. The dotted line represents the major procedures of communication and the arrow direction points to the receiver. In the first round of the communication, the parameters of the cameras, the 3D points, and the projection points are sent to each of the child nodes. This communication procedure is required only once during the initialization. While reducing matrix $\boldsymbol{S}$, each node in A-DSBA cyclically reduces their respective block matrices, and mutual communication between the nodes is needed. During the process of solving the equation set, data communication between the nodes is also necessary for broadcasting a small amount of data from the pivot element. After the equation set is solved, the $\boldsymbol{\delta}_a$ should be broadcasted to all child nodes so that they can compute the increment for the 3D point parameter vector $\boldsymbol{\delta}_b{}^l$. Finally, while checking the termination conditions, each of the child nodes needs to send the intermediate variables (i.e., several double-type variables) to the master node.

7 Experimental results

7.1 Experimental environment

To verify the performance of DSBA, experiments were performed using a cluster of eight nodes, each of which was equipped with a six-core Intel® Xeon® CPU E5-2640 0 @2.5 GHz, with 48 GB memory and Ubuntu 14.04 operating system (OS).

Five datasets in different scales in Table 4 were selected from the University of Washington GRAIL Lab (http://grail.cs.washington.edu/projects/bal/) to evaluate the efficiency and the scalability of the proposed algorithm.

Table 4 Dataset used in the experiment

Dataset	m	n	nproj
1	50	20 431	73 967
2	1031	110 968	500 265
3	1936	649 673	5 213 733
4	4585	1 324 582	9 125 125
5	13 682	4 456 117	28 987 644

7.2 Experimental method and results analysis

We performed experiments to evaluate the memory consumption and the load balance of DSBA

Algorithm 2 Serial SBA algorithm

Major node	Child node l
Read data: camera parameter vector **pa**, 3D point parameter vector **pb**, measured projections $\boldsymbol{x}$, projection function Q	
Divide the dataset into r parts	
Send $(\mathbf{pa},\mathbf{pb}^l,\boldsymbol{x}^l,Q)$ to node l	Receive $(\mathbf{pa},\mathbf{pb}^l,\boldsymbol{x}^l,Q)$ from major node
Initialize $\mathbf{pa}_0,\mathbf{pb}_0{}^1,\boldsymbol{x}_0{}^1$	Initialize $\mathbf{pa}_0,\mathbf{pb}_0{}^l,\boldsymbol{x}_0{}^l$
Compute the Jacobian matrix $\boldsymbol{J}^1$	Compute the Jacobian matrix $\boldsymbol{J}^l$
Compute $(\boldsymbol{U}^*)^1,(\boldsymbol{V}^{*1}),\boldsymbol{W}^1,\boldsymbol{\epsilon}_a^l$, and $\boldsymbol{\epsilon}_b^l$	Compute $(\boldsymbol{U}^*)^l,(\boldsymbol{V}^*)^l,\boldsymbol{W}^l,\boldsymbol{\epsilon}_a^l$, and $\boldsymbol{\epsilon}_b^l$
Compute $\boldsymbol{S}^1=\boldsymbol{U}^{*1}-\boldsymbol{W}^1(\boldsymbol{V}^{*1})^{-1}(\boldsymbol{W}^1)^{\mathrm{T}}$	Compute $\boldsymbol{S}^l=\boldsymbol{U}^{*l}-\boldsymbol{W}^l(\boldsymbol{V}^{*l})^{-1}(\boldsymbol{W}^l)^{\mathrm{T}}$
Send $(\boldsymbol{S}^l,\boldsymbol{e}^l)$ to node l	Send $(\boldsymbol{S}^1,\boldsymbol{e}^1)$ to the major node
Receive $(\boldsymbol{S}^1,\boldsymbol{e}^1)$ from node l	Send $(\boldsymbol{S}^l,\boldsymbol{e}^l)$ to node l
	Receive $(\boldsymbol{S}^l,\boldsymbol{e}^l)$ from node l
Solve equation $\boldsymbol{S}\boldsymbol{d}_a=\boldsymbol{e}_a$	Solve equation $\boldsymbol{S}\boldsymbol{d}_a=\boldsymbol{e}_a$
Broadcast $\boldsymbol{d}_a$ to node l	Receive $\boldsymbol{e}_a$ from the major node
Compute $\boldsymbol{d}_{b^l}=\left((\boldsymbol{V}^*)^l\right)^{-1}\left(\boldsymbol{\epsilon}_{b^l}^l-\sum_j\boldsymbol{W}^{l\mathrm{T}}\boldsymbol{d}_{a_j}\right)$	Compute $\boldsymbol{d}_{b^l}=\left((\boldsymbol{V}^*)^l\right)^{-1}(\boldsymbol{\epsilon}_{b^l}^l-\sum_j\boldsymbol{W}^{l\mathrm{T}}\boldsymbol{d}_{a_j})$
Collect termination variable from node l	Send termination variables to the major node
If the termination condition is satisfied, stop; else, $\boldsymbol{p}^1:=\boldsymbol{p}^1+\boldsymbol{\delta}_{p^l}$	If the termination condition is satisfied, stop; else, $\boldsymbol{p}^l:=\boldsymbol{p}^l+\boldsymbol{d}_{p^l}$

under different 3D models and computer clusters of various scales.

7.2.1 Memory consumption

The DSBA algorithm considerably reduces the memory consumption and improves the scalability by distributing the 3D reconstruction model's parameters to different nodes. Fig. 10 shows the variation in the memory consumption of the 3D model in different scales running on different numbers of nodes. Fig. 10a shows the variation in the number of 3D points in an individual node with the number of nodes. Fig. 10b shows the variation in the number of projection points in an individual node. For the 3D model in different scales in the DSBA algorithm, Fig. 10c presents the variation in the memory consumption of the Jacobian matrix with the number of nodes. Fig. 10 shows that the numbers of 3D points and projection points in an individual node decrease with an increase in the number of nodes. The memory consumption of the Jacobian matrix in an individual node also decreases linearly with the increase in the number of nodes. Therefore, the DSBA algorithm reduces the consumption of storage space for the SBA method and improves the scalability of SBA.

7.2.2 Load balance

The number of the projection points corresponding to each 3D point significantly differs during the 3D reconstruction process. Although the 3D points are evenly allocated to all nodes in sequence, the distribution of the projection points corresponding to the 3D points within each node is likely to cause a severe imbalance. To address the problem, the 3D points are allocated using a homogeneous and random distribution in practice. It is a very simple but effective approach for solving the load imbalance problem due to sequence allocation. Table 5 shows the number of projection points in each node using random uniform allocation and sequence allocation strategies, given a 3D model with $r = 4$, $m = 1936$, $n = 649\ 673$, and nproj $= 521\ 373$. It can be observed that for the case of sequence allocation, the number of projection points in node 1 is 1.7 times that in node 4, resulting in a serious load imbalance. However, the number of projection points shows no significant variation with the number of child nodes when using random allocation.

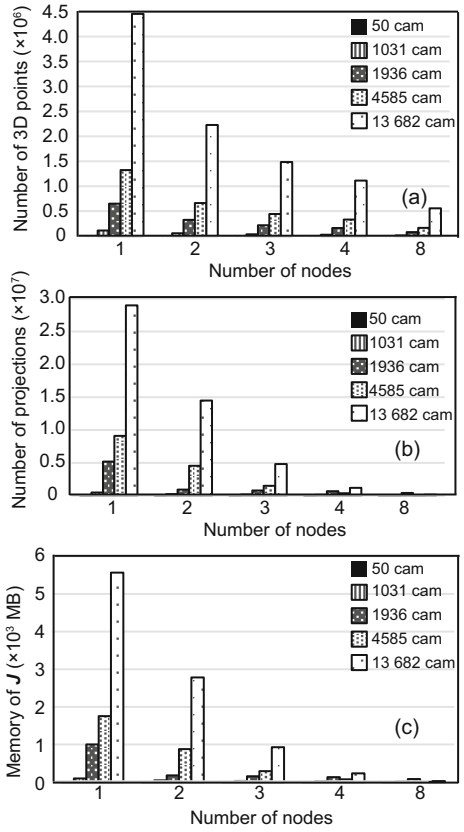


Fig. 10 Memory consumption of the DSBA algorithm: (a) variation in the number of 3D points; (b) variation in the number of projection points; (c) variation in the memory consumption of the Jacobian matrix

Table 5 Task allocation via serial or random strategies

Node ID	m	n^l	nproj (sequence)	nproj (random)
1	1936	162 419	1 707 806	1 315 597
2	1936	162 418	1 323 800	1 291 270
3	1936	162 418	1 138 712	1 281 072
4	1936	162 418	1 043 415	1 325 794

7.2.3 Performance analysis

1. Performance of the blocked cyclic based distributed method for solving the equation set

Table 6 shows the influence of the number of nodes on the time spent on solving the equation set in different scales. $r = 1$ indicates that there is one individual node for solving the equation set in a parallel manner. Fig. 11 shows the block diagram of

the three datasets in Table 6. Table 6 shows that the greater the number of nodes, the larger the margin by which the speedup ratio increases (compared with the single six-core node case, the same below). Considering the scenario with 1936 cameras, when the numbers of nodes are 2, 3, 4, and 8, the speedup ratios are 1.8x, 2.5x, 3.2x, and 6.1x, respectively. If there are 4548 cameras, the corresponding speedup ratios are 1.9x, 2.8x, 3.6x, and 7.09x, respectively. Table 6 also shows that if $m = 13\,682$ and cnp $= 10$, the memory consumption for solving the equation set is $(m \times \text{cnp})^2 \times 4/10^9 \approx 75$ GB. This means that the equation set cannot be solved in a 48 GB-memory node. However, by switching to a cluster of two or more nodes, the equations can be successfully solved and the speedup ratio is increased with the increase of the number of nodes.

Table 6 Time consumption for solving the equation set with a varying number of nodes

m	r				
	50	1031	1936	4585	13 682
1	0.15	12.65	64.59	1360.53	–
2	0.08	10.08	35.51	445.87	50 421.06
3	0.06	7.05	26.22	302.90	11 290.37
4	0.04	4.18	20.01	238.90	7 317.13
8	0.03	3.29	10.62	131.00	3 322.82

'–' indicates that the case is not executable, m denotes the number of cameras, and r represents the number of nodes in the cluster

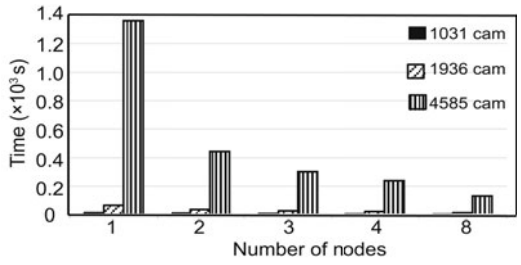


Fig. 11 Time consumption for solving the equation set

Fig. 12 shows the influence of the block size NB on the time spent on solving the equation set with 40 000 dimensions. It can be seen that the time consumption for solving the equation set is minimized and reaches the optimal when the block size is 128 or 256. Modular testing on the proposed algorithm indicates that the obtained speedup ratio is the most desirable when the number of processor grids P_n is close to the value of Q_n and when P_n is less than Q_n. The speedup ratio is optimal when 80% of the cluster memory is consumed. It is pertinent to mention that system performance begins to decrease if more memory is required. Consider a cluster with eight nodes, each of which is equipped with 48 GB memory and 64-bit OS. It is able to handle an equation set with a scale of up to $8 \times 48 \times 80\% \times 10^9/8 = 3.84 \times 10^{10}$, compared with the processing ability of 4.8×10^{10} for a single node.

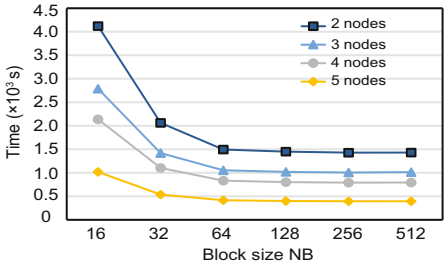


Fig. 12 Influence of block size on time consumption for solving the equation set

2. Performance analysis of key steps in the DSBA algorithm

Fig. 13 shows the variation in the DSBA key steps (calculation of $\boldsymbol{J}$, $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$, $\boldsymbol{S}$, and $\boldsymbol{e_a}$) with the number of nodes in a 3D model of 1936 cameras. It can be observed that the speedup ratio for the three time-consuming steps in the serial algorithm linearly increases with the increase of the number of nodes. This is because matrices $\boldsymbol{J}$ and $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$ can be computed without communication among nodes. Yet, the calculaton of $\boldsymbol{S}$ and $\boldsymbol{e_a}$ requires communication so that the speedup ratio is smaller than those in the other two steps. Compared with the single-node SBA algorithm, compution of $\boldsymbol{J}$, $\boldsymbol{J}^{\mathrm{T}}\boldsymbol{J}$, $\boldsymbol{S}$, and $\boldsymbol{e_a}$ achieves speedup ratios of 6.9x, 6.48x, 5.92x, and 5.8x, respectively, in the case of eight nodes.

3. Analysis of the general performance of the DSBA algorithm

Fig. 14 depicts the variation in the speedup ratio using synchronous or asynchronous DSBA methods, when the 3D model has 1936 or 4585 cameras. Consider the model with 1936 cameras and 2/3/4/8 nodes, the speedup ratios of S-DSBA are 1.3x, 1.7x, 2.2x, and 4.4x, while the speedup ratios of A-DSBA are 1.7x, 2.4x, 3.2x, and 6.4x. Hence, A-DSBA is faster than S-DSBA by 31%, 40%, 43%, and 46%, respectively. Compared with the serial single-core

SBA algorithm, the speedup ratios of A-DSBA are 10.3x, 14.9x, 19.2x, and 41.3x, respectively, given 2, 3, 4, and 8 nodes, respectively, each of which has six cores.

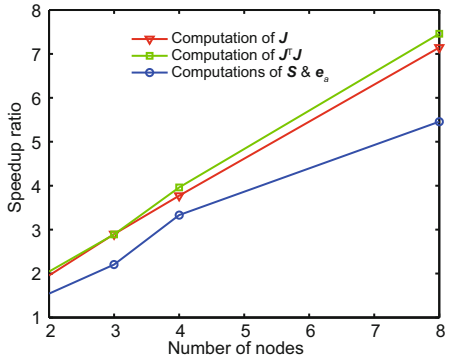


Fig. 13 Speedup ratio of the DSBA key steps

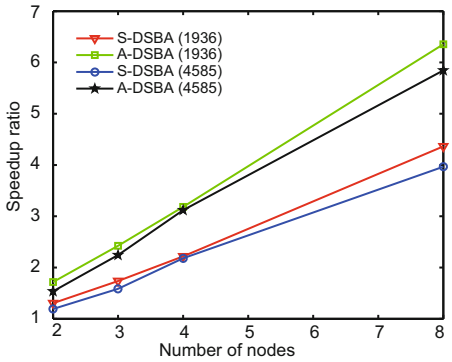


Fig. 14 Speedup ratio of DSBA

Modular analysis of the solving of the equation set reveals that the performance reaches the optimal when 80% of the memory in the entire cluster is consumed. However, if the method in Section 5.4 for solving the equation set is applied to the DSBA algorithm, a considerable amount of memory will be consumed by not only matrix $\boldsymbol{S}$, but also the 3D point parameters, the projection point parameters, and the Jacobian matrix. A statistical experiment indicates that the performance of DSBA reaches its optimal when 65% to 70% of the memory in the entire cluster is used to solve the equation set.

8 Conclusions and future work

Fortunately, in the present age we can easily create a small-scale cluster of computers due to drastic decrease in hardware cost. However, existing BA algorithms are slow and cannot address the problems incurred regarding scalability. Moreover, the performance of the algorithm is constrained by the scalability and heavily depends on the memory of individual nodes. In this regard, a distributed SBA algorithm, which is simple and easy to implement and independent of BA problems, is proposed in this paper. Compared with the serial SBA method, the proposed algorithm achieves remarkable speedup ratios without compromising on accuracy. What is more, a blocked cyclic based DSBA algorithm is proposed to improve the speed of serial SBA and the scalability of SBA. By adopting the blocked cyclic based asynchronous reduced DSBA algorithm (A-DSBA), the communication time associated with the reduction of $\boldsymbol{S}$ overlaps with the time taken to solve the equation set. Consequently, the calculation time of the A-DSBA algorithm is approximately 46% shorter than that of the S-DSBA algorithm. Memory consumption of A-DSBA is in proportion to the square of the number of cameras. As a future work, it is worth investigating how to further reduce the algorithm's spatial complexity by exploiting the characteristics of blocked cyclic.

References

Agarwal S, Furukawa Y, Snavely N, et al., 2011. Building Rome in a day. *Commun ACM*, 54(10):105-112. https://doi.org/10.1145/2001269.2001293

Choudhary S, Gupta S, Narayanan P, 2010. Practical time bundle adjustment for 3D reconstruction on the GPU. In: Kutulakos KN (Ed.), Trends and Topics in Computer Vision. Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/978-3-642-35740-4_33

Eriksson A, Bastian J, Chin T, et al., 2016. A consensus-based framework for distributed bundle adjustment. IEEE Conf on Computer Vision and Pattern Recognition, p.1754-1762. https://doi.org/10.1109/CVPR.2016.194

Frahm J, Fite-Georgel P, Gallup D, et al., 2010. Building Rome on a cloudless day. Proc 11th European Conf on Computer Vision, p.368-381. https://doi.org/10.1007/978-3-642-15561-1_27

Hänsch R, Drude I, Hellwich O, 2016. Modern methods of bundle adjustment on the GPU. *ISPRS Ann Photogr Remote Sens Spatial Inform Sci*, III-(3):43-50. https://doi.org/10.5194/isprs-annals-III-3-43-2016

Hartley R, Zisserman A, 2000. Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge, p.1865-1872.

Heinecke A, Vaidyanathan K, Smelyanskiy M, et al., 2013. Design and implementation of the linpack benchmark for single and multi-node systems based on Intel® Xeon Phi coprocessor. Proc 27th Int Symp on Parallel and Distributed Processing, p.126-137. https://doi.org/10.1109/IPDPS.2013.113

Husbands P, Yelick K, 2007. Multi-threading and one-sided communication in parallel LU factorization. ACM/IEEE Conf on Supercomputing, Article 31. https://doi.org/10.1145/1362622.1362664

Jo G, Nah J, Lee J, et al., 2015. Accelerating LINPACK with MPI-OpenCL on clusters of multi-GPU nodes. *IEEE Trans Parall Distrib Syst*, 26(7):1814-1825. https://doi.org/10.1109/TPDS.2014.2321742

Li X, Wu C, Zach C, et al., 2008. Modeling and recognition of landmark image collections using iconic scene graphs. Proc 10th European Conf on Computer Vision, p.427-440. https://doi.org/10.1007/978-3-540-88682-2_33

Liu X, Gao W, Hu Z, 2012. Hybrid parallel bundle adjustment for 3D scene reconstruction with massive points. *Comput Sci Technol*, 27(6):1269-1280. https://doi.org/10.1007/s11390-012-1303-3

Lourakis M, Argyros A, 2009. SBA: a software package for generic sparse bundle adjustment. *ACM Trans Math Soft*, 36(1), Article 2. https://doi.org/10.1145/1486525.1486527

Ni K, Steedly D, Dellaert F, 2007. Out-of-core bundle adjustment for large-scale 3D reconstruction. Proc 11th Int Conf on Computer Vision, p.1-8. https://doi.org/10.1109/ICCV.2007.4409085

Ramamurthy K, Lin C, Aravkin A, et al., 2017. Distributed bundle adjustment. IEEE Int Conf on Computer Vision Workshop, p.2146-2154. https://doi.org/10.1109/ICCVW.2017.251

Snavely N, Seitz S, Szeliski R, 2006. Photo tourism: exploring photo collections in 3D. *ACM Trans Graph*, 25(3):835-846. https://doi.org/10.1145/1141911.1141964

Snavely N, Seitz S, Szeliski R, 2008. Skeletal graphs for efficient structure from motion. IEEE Conf on Computer Vision and Pattern Recognition, p.1-8. https://doi.org/10.1109/CVPR.2008.4587678

Triggs B, McLauchlan P, Hartley R, et al., 1999. Bundle adjustment—modern synthesis. Int Workshop on Vision Algorithms: Theory and Practice, p.298-372. https://doi.org/10.1007/3-540-44480-7_21

Wu C, Agarwal S, Curless B, et al., 2011. Multicore bundle adjustment. IEEE Conf on Computer Vision and Pattern Recognition, p.3057-3064. https://doi.org/10.1109/CVPR.2011.5995552

Zheng E, Wu C, 2015. Structure from motion using structure-less resection. IEEE Int Conf on Computer Vision, p.2075-2083. https://doi.org/10.1109/ICCV.2015.240