Applied Intelligence (2022) 52:753–765
https://doi.org/10.1007/s10489-021-02349-8

A scalable parallel preconditioned conjugate gradient method for bundle adjustment

Jiaxin Peng[1,2] · Jie Liu[1,2] · Hua Wei[3]

Accepted: 11 March 2021 / Published online: 10 May 2021

Abstract
Bundle adjustment is a fundamental problem in computer vision, with important applications such as 3D structure reconstruction from 2D images. This paper focuses on large-scale bundle adjustment tasks, *e.g.*, city-wide 3D reconstruction, which require highly efficient solutions. For this purpose, it is common to apply the Levenberg-Marquardt algorithm, whose bottleneck lies in solving normal equations. The majority of recent methods focus on achieving scalability through modern hardware such as GPUs and distributed systems. On the other hand, the core of the solution, *i.e.*, the math underlying the optimizer for the normal equations, remains largely unimproved since the proposal of the classic parallel bundle adjustment (PBA) algorithm, which increasingly becomes a major limiting factor for the scalability of bundle adjustment.
This paper proposes parallel preconditioned conjugate gradient (PPCG) method, a novel parallel method for bundle adjustment based on preconditioned conjugate gradient, which achieves significantly higher efficiency and scalability than existing methods on the algorithmic level. The main idea is to exploit the sparsity of the Hessian matrix and reduce its structure parameters through an effective *parallel* Schur complement method; the result of this step is then fed into our carefully designed PPCG method which reduces matrix operations that are either expensive (*e.g.*, large matrix reverse or multiplications) or scales poorly to multi-processors (*e.g.*, parallel Reduce operators). Extensive experiments demonstrate that PPCG outperforms existing optimizers by large margins, on a wide range of datasets.

Keywords Structure from motion · Bundle adjustment · Preconditioned conjugate gradient

1 Introduction

Given a set of 2D projections of the same 3D structure, *e.g.*, each captured by a camera from a different viewpoint, bundle adjustment [1–3] simultaneously calculates the 3D world points and the camera view parameters that best fit these 2D projections. Bundle adjustment finds important applications in 3D reconstruction [4] from 2D images, such as structure from motion (SfM) [5, 6] and simultaneous localization and mapping (SLAM) [7]. This paper focuses on large-scale bundle adjustment, *e.g.*, 3D reconstruction of an entire city from a very large collection of 2D images. For such tasks, a key consideration is the efficiency and scalability of the bundle adjustment solution, which rules out computationally expensive solutions such as those based on deep neural networks, *e.g.*, [8–11]. Instead, in practice it is common to apply the Levenberg-Marquardt (LM) algorithm, which involves solving a large system of normal equations. Further, recent trends in the development of computational infrastructure indicate that it is critical to design a solution that well utilizes *parallel* computation resources. Hence, this paper focuses on building a scalable parallel optimizer that solves the normal equations in LM.

A classic solution for our problem is PBA [12, 13], which involves two key ideas to reduce the computational costs of the optimizer: Schur complement and preconditioned conjugate gradient. The latter (but not the former) is parallelized to utilize the capabilities of multicore CPUs

✉ Jie Liu
liujie@nudt.edu.cn

1 Laboratory of Software Engineering for Complex Systems, School of Computer Science, National University of Defense Technology, Changsha, 410073, Hunan, China

2 Parallel and Distributed Processing Laboratory, School of Computer Science, National University of Defense Technology, Changsha, 410073, Hunan, China

3 Water and Land Resources Research Center of the Middle Reaches of Yangtze River, School of Resources Environment Science and Engineering, Hubei University of Science and Technology, Xianning, 437100, Hubei, China

and GPUs. Note that in PBA, the parallelization is done on a low level, *i.e.*, on the level of matrix operations. To our knowledge, the core of this algorithm (*i.e.*, its underlying math) has remained largely unimproved since its proposal almost a decade ago. Further, as our experimental evaluation shows, on some data-sets, PBA scales poorly with the number of processors, whose performance peaks at around 12-20 processors. To improve the scalability of bundle adjustment tasks, recent approaches have instead focused on higher-level parallelism (*i.e.*, on the level of scenarios), which decompose a large bundle adjustment problem into smaller ones based on 3D world points, cameras, maps, and domains, and solve these smaller sub-programs in parallel [14, 15]. Meanwhile, several methods have addressed bundle adjustment computation in distributed settings with high communication costs (*e.g.*, across data centers).

For instance, [16] proposes a consensus-based framework for distributed bundle adjustment based on proximal splitting, and a distributed alternating direction method of multipliers (ADMM) framework is proposed for very large-scale bundle adjustment problems [17–19], where an over-relaxation technique and self-adaption schemes are employed to improve the convergence rate. These solutions, however, tend to be less efficient since they often have linear or even sub-linear convergence rate, leading to numerous iterations and, thus, high computation and communication overhead. Additionally, robust parallel bundle adjustment (RPBA) [20] extends the consensus based optimization methods with covariance information to get a better convergence bahavior. Though RPBA is implemented upon covariance information which is very effective, the naive ADMM has slow convergence rate and produces more communication overhead.

This paper presents PPCG, an efficient and scalable low-level parallel optimizer for solving the LM normal equations in large-scale bundle adjustment, which is a direct improvement over the PBA algorithm. Specifically, PPCG is based on a novel parallel Schur complement method that effectively decomposes the Hessian matrix, exploiting its special sparsity conditions to reduce structure parameters. Note that the Schur complement module in PPCG is fundamentally different from that in PBA (*i.e.*, the decomposition results can be considered as locally individual subsystems, which are different); meanwhile, we stress that unlike PBA, our Schur complement module runs in parallel on multiple processors, with a single Reduce operation to collect the results. Based on the decomposition results, PPCG then proceeds to perform preconditioned conjugate gradient, which is carefully designed to avoid expensive matrix operations and parallel Reduce steps. We formally prove the correctness of PPCG, and evaluate its performance through extensive experiments on the BAL benchmark datasets [2]. The evaluation results demonstrate that PPCG significantly outperforms PBA in terms of both acceleration and scalability.

2 Preliminaries on bundle adjustment

Before the application of bundle adjustment, the structure and camera parameters are roughly calculated by multi-view geometry methods. Since there are various distortions for uncalibrated camera models, the reprojection error is widely accepted to measure the accuracy. Generally considered as the back-end of SfM applications, bundle adjustment is an approach for estimating more accurate structure and camera parameters.

Let $\boldsymbol{x} \in \mathbb{R}^{m_s}$, $\boldsymbol{y} \in \mathbb{R}^{m_c}$ and $\boldsymbol{z} \in \mathbb{R}^{m_p}$ be the vectors of structure parameters, camera parameters and coordinates of feature points in the images respectively. If the 3D points (structure parameters) are reprojected into images through the same cameras, the new reprojected feature points $\boldsymbol{z}^* \in \mathbb{R}^{m_p}$ can be obtained by

$$\boldsymbol{z}^* = \Phi(\boldsymbol{x}, \boldsymbol{y}) \tag{1}$$

where $\Phi(\cdot, \cdot)$ is the reprojection function, including three-dimensional translations, rotations and projections. Due to optical distortions of cameras and truncation errors, the original feature points does not coincide with the reprojected feature points. Therefore, bundle adjustment can be represented by a least squares optimization problem, which is given by

$$\arg\min_{\boldsymbol{z}^*} \|\boldsymbol{z} - \boldsymbol{z}^*\|^2 = \arg\min_{\boldsymbol{\Delta x}, \boldsymbol{\Delta y}} \|\boldsymbol{z} - \Phi(\boldsymbol{x} + \boldsymbol{\Delta x}, \boldsymbol{y} + \boldsymbol{\Delta y})\|^2. \tag{2}$$

where $\Delta\boldsymbol{x} \in \mathbb{R}^{m_s}$, $\Delta\boldsymbol{y} \in \mathbb{R}^{m_c}$ are the increments of $\boldsymbol{x}$, $\boldsymbol{y}$ respectively. As long as $\boldsymbol{z}$ represents the original featured points which are all constant, we can adjust $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{y}$ iteratively to satisfy the minimum reprojection error.

Assuming that $f(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{z} - \boldsymbol{z}^*$, $J_x = \partial f(\boldsymbol{x}, \boldsymbol{y})/\partial \boldsymbol{x}$ and $J_y = \partial f(\boldsymbol{x}, \boldsymbol{y})/\partial \boldsymbol{y}$, The LM algorithm improves the original Gauss-Newton algorithm through gradient descent directions, then we can represent the above optimization problem approximately by

$$\begin{aligned} &\arg\min_{\boldsymbol{\Delta x}, \boldsymbol{\Delta y}} \left\| f(\boldsymbol{x}, \boldsymbol{y}) + \begin{bmatrix} J_x & J_y \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta x} \\ \boldsymbol{\Delta y} \end{bmatrix} \right\|^2 \\ &\quad +\lambda \left\| \operatorname{diag}\left(\begin{bmatrix} \sqrt{J_x^T J_x} \mathbf{0} & \\ \mathbf{0} & \sqrt{J_y^T J_y} \end{bmatrix} \right) \begin{bmatrix} \boldsymbol{\Delta x} \\ \boldsymbol{\Delta y} \end{bmatrix} \right\|^2 \end{aligned} \tag{3}$$

$$\begin{aligned} &= \arg\min_{\boldsymbol{\Delta x}, \boldsymbol{\Delta y}} \left\| f(\boldsymbol{x}, \boldsymbol{y}) + J_x \boldsymbol{\Delta x} + J_y \boldsymbol{\Delta y} \right\|^2 \\ &\quad +\lambda \operatorname{diag}\left(J_x^T J_x\right) \|\boldsymbol{\Delta x}\|^2 + \lambda \operatorname{diag}\left(J_y^T J_y\right) \|\boldsymbol{\Delta y}\|^2 \end{aligned} \tag{4}$$

where $\lambda \in \mathbb{R}$ is the *damping factor*. While setting the derivative of (3) to zero, we can obtain the following *normal equations*,

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \boldsymbol{\Delta x} \\ \boldsymbol{\Delta y} \end{bmatrix} = - \begin{bmatrix} J_{\boldsymbol{x}}^T f(\boldsymbol{x}, \boldsymbol{y}) \\ J_{\boldsymbol{y}}^T f(\boldsymbol{x}, \boldsymbol{y}) \end{bmatrix}, \tag{5}$$

where the sub-blocks $U = J_{\boldsymbol{x}}^T J_{\boldsymbol{x}} + \lambda \ \mathrm{diag}\left(J_{\boldsymbol{x}}^T J_{\boldsymbol{x}}\right)$, $V = J_{\boldsymbol{y}}^T J_{\boldsymbol{y}} + \lambda \,\mathrm{diag}(J_{\boldsymbol{y}}^T J_{\boldsymbol{y}})$, $W = J_{\boldsymbol{x}}^T J_{\boldsymbol{y}}$, and the coefficient matrix is the augmented Hessian matrix. Then U is an $m_s \times m_s$ block diagonal and positive definite matrix, V is an much smaller $m_c \times m_c$ block diagonal and positive definite matrix, and W is an $m_s \times m_c$ matrix.

Providing that the augmented Hessian matrix is highly sparse and the Hessian matrix U is block diagonal, the computation of the inverse of U is very cheap. Furthermore, the *Schur complement trick* [21] eliminates the structure parameters $\boldsymbol{\Delta x}$ through Gaussian Elimination. Therefore, the normal equations including only camera parameters, are highly reduced and become

$$(V - W^T U^{-1} W)\boldsymbol{\Delta y} = W^T U^{-1} J_{\boldsymbol{x}}^T f(\boldsymbol{x}, \boldsymbol{y}) - J_{\boldsymbol{y}}^T f(\boldsymbol{x}, \boldsymbol{y}).$$

Let $R = V - W^T U^{-1} W$ and $\boldsymbol{v} = W^T U^{-1} J_{\boldsymbol{x}}^T f(\boldsymbol{x}, \boldsymbol{y}) - J_{\boldsymbol{y}}^T f(\boldsymbol{x}, \boldsymbol{y})$, we then obtain the following system of linear equations,

$$R\boldsymbol{\Delta y} = \boldsymbol{v}. \tag{6}$$

The coefficient matrix R is the Schur complement called the Reduced Camera Matrix (RCM), which is much smaller and symmetrical and positive definite, and $\boldsymbol{v}$ is the Reduced Camera Vector (RCV). $\boldsymbol{\Delta y}$ can be solved through various iterative algorithms, then $\boldsymbol{\Delta x}$ can be obtained by

$$\boldsymbol{\Delta x} = -U^{-1}\left(J_{\boldsymbol{x}}^T f(\boldsymbol{x}, \boldsymbol{y}) + W\boldsymbol{\Delta y}\right) \tag{7}$$

3 Proposed solution

The bundle adjustment introduced in Section 2, contains great potential parallelization, and can be accelerated by parallel computing for large scale problems. This section firstly presents a parallel Schur complement method to eliminate the structure parameters by enhancing the matrix inverse and multiplication operations, and improves the classical preconditioned conjugate gradient (PCG) method through parallelization in the second place. Before discussing the parallel methods, we present the definitions of memory coherence and consistency.

Definition 1 (*Memory Coherence*) On a shared-memory parallel computing system with multiprocessors, there is only one processor (process or thread) can read from/write to a memory location such that the updated value can be observed by all the subsequent read operations of the corresponding memory location. We call this characteristic as *memory coherence*.

Definition 2 (*Memory Consistency*) On a distributed-memory parallel computing system where each node has its own memory space, if a memory location is written to, its copies on other nodes will be updated such that the updated values can be observed in their own local memory spaces. This characteristic is called memory consistency.

To prevent failures and errors caused by parallel computing systems, we assume that both of the following parallel methods are presented under the constraints of memory coherence and consistency.

3.1 Parallel Schur complement

The augmented Hessian matrix is approximately a block-diagonal arrow-head matrix, and the diagonal submatrices are independent from each other. Considering this particular structure, the augmented Hessian matrix can be divided into three parts depending on whether the local data are calculated from structure parameters, camera parameters or observations. The augmented Hessian and its divisions can be represented by

$$H^* = \begin{bmatrix} U_1^s & & & & & W_1 \\ & U_2^s & & & & W_2 \\ & & U_3^s & & & W_3 \\ & & & \ddots & & \vdots \\ & & & & U_n^s & W_n \\ W_1^T & W_2^T & W_3^T & \cdots & W_n^T & U^c \end{bmatrix}$$

$$\Rightarrow \begin{cases} U & = \mathrm{diag}\left[U_1^s, U_2^s, ..., U_n^s\right], \\ V & = U^c = \sum_{i=1}^n U_i^c, \\ W & = [W_1, W_2, ..., W_n]^T \end{cases} \tag{8}$$

where $U_1^s, U_2^s, ..., U_n^s$ are Hessian matrices of structure parameters for n 3D world points, U_i^c is the Hessian matrix of the i-th camera, whereas U^c is the Hessian matrix of all cameras, and $W_1, W_2, ..., W_n$ are derived from the observations between 3D world points and cameras. Sequentially, the RCM R can be presented as

$$R = V - W^T U^{-1} W = \sum_{i=1}^{n} (U_i^c - W_i^T (U_i^s)^{-1} W_i). \tag{9}$$

Given a parallel computing system with p concurrent computing units, we can distribute (5) into p subsystems, and the j-th subsystem can be summarized by

$$\begin{bmatrix} U_j^s & & & & W_j \\ & \ddots & & & \vdots \\ & & U_{lp+j}^s & & W_{lp+j} \\ & & & \ddots & \vdots \\ & & & U_{\phi(j)}^s & W_{\phi(j)} \\ W_j^T & \cdots & W_{lp+j}^T & \cdots\ W_{\phi(j)}^T & V(j) \end{bmatrix} \times \begin{bmatrix} \boldsymbol{\Delta x}_j \\ \vdots \\ \boldsymbol{\Delta x}_{lp+j} \\ \vdots \\ \boldsymbol{\Delta x}_{\phi(j)} \\ \boldsymbol{\Delta y} \end{bmatrix} = -\begin{bmatrix} J_{\boldsymbol{x}_j}^T f(\boldsymbol{x}_j, \boldsymbol{y}) \\ \vdots \\ J_{\boldsymbol{x}_{lp+j}}^T f(\boldsymbol{x}_{lp+j}, \boldsymbol{y}) \\ \vdots \\ J_{\boldsymbol{x}_{\phi(j)}}^T f(\boldsymbol{x}_{\phi(j)}, \boldsymbol{y}) \\ J_{\boldsymbol{y}}^T f(\boldsymbol{x}_{j:p:\phi(j)}, \boldsymbol{y}) \end{bmatrix} \tag{10}$$

where $V(j) = U_j^c + \cdots + U_{lp+j}^c + \cdots + U_{\phi(j)}^c (1 \leq l \leq \lfloor n/p \rfloor)$, $\phi(j) = \lfloor n/p \rfloor p$ when $j > n \mod p$ and $\phi(j) = (\lfloor n/p \rfloor)p + j$ when $j \leq n \mod p$. If we denote $U(j) = \operatorname{diag}\left[U_j^s, \cdots, U_{lp+j}^s, ..., U_{\phi(j)}^s\right]$ and $W(j) = [W_j, \cdots, W_{lp+j}, \cdots, W_{\phi(j)}]^T$, the related Schur complement is $V(j) - W^T(j)U^{-1}(j)W(j)$.

Theorem 1 *If the 3D world points of a bundle adjustment system are divided into p subsystems and these 3D world points are evenly observed by cameras, the distribution method achieves the best load balance for parallel Schur complement when the 3D points are distributed equally.*

Proof We use the function $\psi(\cdot)$ to map the global 3D world points to local 3D world points in subsystems, *i.e.* $\psi(1)$ is the first local 3D world point from the $\psi(1)$-th global 3D world point, $\psi(2)$ is the second local 3D world point from the $\psi(2)$-th global 3D world point, and et al.

Supposing that $n_j (1 \leq j \leq p)$ 3D world points are assigned to the j-th subsystem, the j-th divisions are denoted by $U'(j) = \operatorname{diag}\left[U_{\psi(1)}^s, U_{\psi(2)}^s, \cdots, U_{\psi(n_j)}^s\right]$, $W'(j) = \left[W_{\psi(1)}, W_{\psi(2)}, \cdots, W_{\psi(n_j)}\right]^T$, $V'(j) = U_{\psi(1)}^c + U_{\psi(2)}^c + \cdots + U_{\psi(n_j)}^c$, $\psi(1) \leq \psi(2) \leq \cdots \leq \psi(n_j)$. Then, the Schur complement R_j for the j-th subsystem can be inferred out like the form of (8), which implies

$$R_j = \sum_{i=\psi(1)}^{\psi(n_j)} \left(U_i^c - W_i^T (U_i^s)^{-1} W_i\right).$$

As these 3D world points are evenly observed by cameras, the sparsity of W_1 to W_n are considered to be equivalent such that $U_i^c - W_i^T (U_i^s)^{-1} W_i$ takes the same amount of time for all $i = 1, 2, \cdots, n$. Therefore, we can use the number of $U_i^c - W_i^T (U_i^s)^{-1} W_i$ to represent the load of the subsystems. Thus, the overall load for the j-th subsystem is determined by $\max_j n_j$. When $n_j = \lfloor n/p \rfloor$, we obtain the minimum overall load and achieves the best load balance. $\square$

Furthermore, without the assumption that 3D world points are evenly observed by cameras, $W'(j) = [W_{\psi(1)}, W_{\psi(2)}, \cdots, W_{\psi(n_j)}]^T$ are not equally distributed. The observations are always dense at one place, and sparse at another place, which is the locality of observations. Taking advantage of the locality, the distribution method presented in (10), achieves better load balance than other equal distribution methods.

Note that direct computation of the RCM still involves expensive operations such as matrix multiplications, parallel reductions, and sparse matrix subtractions. The operations that are all geometrical in time complexity, can be greatly reduced or avoided by associating with the following preconditioning approach.

3.2 Parallel PCG

When the number of images and cameras grows, the size of RCM increases quadratically. The sparsity of RCM mainly depends on the observations, and it turns to be much smaller and much denser after Schur complement, which becomes an important part with respect to overall performance. Before proposing the PPCG algorithm, we put forward the following theorem.

Theorem 2 *If a bundle adjustment system with the augmented Hessian matrix in the form of* (5)*, is divided into p concurrent subsystems with the augmented Hessian matrices in the form of* (10)*, then*

$$\sum_{i=1}^{n} W_i^T (U_i^s)^{-1} W_i = \sum_{j=1}^{p} W^T(j) U^{-1}(j) W(j). \tag{11}$$

Proof As the matrix $U^{-1}(j)$ is block diagonal and positive definite, the inverse of it can be easily determined by $U^{-1}(j) = \operatorname{diag}\left[\left(U_j^s\right)^{-1}, \cdots, \left(U_{lp+j}^s\right)^{-1}, ..., \left(U_{\phi(j)}^s\right)^{-1}\right]$, which is purely block diagonal. We expand $W^T(j)U^{-1}(j)W(j)$ of the right hand in Theorem 2 for each subsystems, which leads to

$$\sum_{j=1}^{p} W^T(j) U^{-1}(j) W(j)$$
$$= \sum_{j=1}^{p} \left[W_j^T \left(U_j^s\right)^{-1}, \cdots, W_{\phi(j)}^T \left(U_{\phi(j)}^s\right)^{-1}\right] W(j)$$

$$= \sum_{j=1}^{p} \left(W_j^T \left(U_j^s \right)^{-1} W_j + \cdots + W_{\phi(j)}^T \left(U_{\phi(j)}^s \right)^{-1} W_{\phi(j)} \right)$$
$$= \sum_{i=1}^{n} W_i^T \left(U_i^s \right)^{-1} W_i.$$

□

The PPCG approximates the accurate solution through updating the actual solution iteratively. Assuming that $\boldsymbol{\Delta y}_k \in \mathbb{R}^{m_c}$ is the solution vector for the k-th iteration, $\boldsymbol{r}_k \in \mathbb{R}^{m_c}$ is the residual vector, $\boldsymbol{t}_k \in \mathbb{R}^{m_c}$ is the temporary vector, $\boldsymbol{p}_k \in \mathbb{R}^{m_c}$ is the direction vector where $k = 0, 1, 2, \cdots$, and M is the preconditioned matrix, the initializations are summarized by

$$\boldsymbol{r}_0 = \boldsymbol{v} - R\boldsymbol{\Delta y}_0, \quad \boldsymbol{t}_0 = M^{-1}\boldsymbol{r}_0, \quad \boldsymbol{p}_0 = \boldsymbol{t}_0. \tag{12}$$

where $\boldsymbol{\Delta y}_0$ is zero or other reasonable vectors, and preconditioned matrix M is assigned to be the block diagonal of R. Since $m_c << m_s$, the communication overhead will be dominant (discussed in Section 4.3) if we distribute $\boldsymbol{\Delta y}_k$, $\boldsymbol{r}_k$, $\boldsymbol{t}_k$, $\boldsymbol{p}_k$ and M into all the subsystems. Thus, they are duplicated in every subsystem to avoid frequent communications.

Given that the direction vectors $\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_k$ are R-orthogonal, the step size α_k for the k-th iteration can be determined by

$$\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{t}_k}{\boldsymbol{p}_k^T R \boldsymbol{p}_k}. \tag{13}$$

As is demonstrated in Section 3.1, the RCM R is not explicitly calculated due to the expensive parallel reduce operations of sparse matrices. From the result of Theorem 2, we can calculate $R\boldsymbol{p}_k$ through local Hessian matrices $U(j)$, $V(j)$ and $W(j)$ instead of the RCM R itself according to the implicit Schur complement method [22], that is

$$\begin{aligned}
R\boldsymbol{p}_k &= \left(\sum_{i=1}^{n} \left(U_i^c - W_i^T \left(U_i^s \right)^{-1} W_i \right) \right) \boldsymbol{p}_k \\
&= \left(\sum_{i=1}^{n} U_i^c \right) \boldsymbol{p}_k - \left(\sum_{i=1}^{n} W_i^T \left(U_i^s \right)^{-1} W_i \right) \boldsymbol{p}_k \\
&= \left(\sum_{j=1}^{p} V(j) \right) \boldsymbol{p}_k - \left(\sum_{j=1}^{p} W^T(j) U^{-1}(j) W(j) \right) \boldsymbol{p}_k \\
&= \sum_{j=1}^{p} (V(j)\boldsymbol{p}_k) - \sum_{j=1}^{p} W^T(j)(U^{-1}(j)(W(j)\boldsymbol{p}_k)).
\end{aligned} \tag{14}$$

With the step size α_k and the result of $R\boldsymbol{p}_k$, the new solution vector $\boldsymbol{\Delta y}_{k+1}$ and the new residual vector $\boldsymbol{r}_{k+1}$ can be generated by

$$\boldsymbol{\Delta y}_{k+1} = \boldsymbol{\Delta y}_k + \alpha_k \boldsymbol{p}_k, \tag{15}$$
$$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k R \boldsymbol{p}_k \tag{16}$$

where $\boldsymbol{\Delta y}_{k+1}$ and $\boldsymbol{r}_{k+1}$ are updated in conjugate directions.

The residual vector $\boldsymbol{r}_{k+1}$ is applied to determine the exit conditions. Once $\|\boldsymbol{r}_{k+1}\|$ drops below a predefined threshold, we consider that $\boldsymbol{\Delta y}_{k+1}$ is a sufficiently accurate solution. Otherwise, we use the new residual vector to update the direction vector. The new temporary vector $\boldsymbol{t}_{k+1}$ can be obtained by

$$\boldsymbol{t}_{k+1} = M^{-1}\boldsymbol{r}_{k+1}. \tag{17}$$

Consequently, the Gram-Schmidt constant β_k for the k-th iteration is given by

$$\beta_k = \frac{\boldsymbol{t}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{t}_k^T \boldsymbol{r}_k}. \tag{18}$$

Finally, to ensure that the direction vectors $\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_{k+1}$ are R-orthogonal, the new direction vector $\boldsymbol{p}_{k+1}$ can be calculated with β_k by

$$\boldsymbol{p}_{k+1} = \boldsymbol{t}_{k+1} + \beta_k \boldsymbol{p}_k. \tag{19}$$

Until reaching the termination condition, each new iteration generates a new solution with $\boldsymbol{p}_{k+1}$. The overall parallel algorithm on a multi-process system for solving normal equations is described by Algorithm 1.

Algorithm 1 Parallel solution for normal equations.

```
Require:
    Reprojection error vector, $f(\boldsymbol{x}, \boldsymbol{y})$;
Ensure:
    solutions for the increments of camera parameters, $\boldsymbol{\Delta y}_{k+1}$;
1: Initialize $\lambda$ with a small positive number;
2: Distribute $f(\boldsymbol{x}, \boldsymbol{y})$ on $p$ processes, where $f(\boldsymbol{x}_{j:p:\phi(j)}, \boldsymbol{y})$, $1 \le j \le p$ is the $j$-th subvector for the $j$-th process;
3: Compute $J_j(\boldsymbol{x})$ and $J_j(\boldsymbol{y})$ parallelly;
4: $f_x^j \leftarrow J_j^T(\boldsymbol{x})\ f(\boldsymbol{x}_{j:p:\phi(j)}, \boldsymbol{y})$; $f_y^j \leftarrow J_j^T(\boldsymbol{y})\ f(\boldsymbol{x}_{j:p:\phi(j)}, \boldsymbol{y})$;
5: $U_j \leftarrow J_j^T(\boldsymbol{x})J_j(\boldsymbol{x}) + \lambda\ \mathrm{diag}\left(J_j^T(\boldsymbol{x})J_j(\boldsymbol{x})\right)$;
6: $V_j \leftarrow J_j^T(\boldsymbol{y})J_j(\boldsymbol{y}) + \lambda\ \mathrm{diag}\left(J_j^T(\boldsymbol{y})J_j(\boldsymbol{y})\right)$;
7: $W_j \leftarrow J_j^T(\boldsymbol{x})J_j(\boldsymbol{y})$;
8: Initialize $\boldsymbol{\Delta y}_0$, $\boldsymbol{r}_0$, $\boldsymbol{t}_0$, $\boldsymbol{p}_0$ and $M$ for all processes;
9: repeat
10:     /**Running parallelly for every concurrent process**/
11:     $\boldsymbol{v}_0^j \leftarrow W_j^T\left(U_j^{-1}(W_j\boldsymbol{p}_k)\right)$; /**$\boldsymbol{v}_0^j$ is a temporary vector for the $j$-th process**/
12:     $\boldsymbol{v}_1^j \leftarrow V_j\boldsymbol{p}_k$; /**$\boldsymbol{v}_1^j$ is a temporary vector for the $j$-th process**/
13:     $\boldsymbol{v}_2^j \leftarrow \boldsymbol{v}_0^j - \boldsymbol{v}_1^j$; /**$\boldsymbol{v}_2^j$ is a temporary vector for the $j$-th process**/
14:     Reduce $\boldsymbol{v}_2^j$ from $p$ processes: $\boldsymbol{v}_{Rp} \leftarrow \sum_{j=1}^{p}\boldsymbol{v}_2^j$; //$\boldsymbol{v}_{Rp} = R\boldsymbol{p}_k$
15:     $\alpha_k \leftarrow \left(\boldsymbol{r}_k^T\boldsymbol{t}_k\right) / \left(\boldsymbol{p}_k^T\boldsymbol{v}_{Rp}\right)$;
16:     $\boldsymbol{\Delta y}_{k+1} \leftarrow \boldsymbol{\Delta y}_k + \alpha_k\boldsymbol{p}_k$;   $\boldsymbol{r}_{k+1} \leftarrow \boldsymbol{r}_k - \alpha_k\boldsymbol{v}_{Rp}$;
17:     $\boldsymbol{t}_{k+1} \leftarrow M^{-1}\boldsymbol{r}_{k+1}$;
18:     $\beta_k \leftarrow (\boldsymbol{t}_{k+1}^T\boldsymbol{r}_{k+1}) / (\boldsymbol{t}_k^T\boldsymbol{r}_k)$;
19:     $\boldsymbol{p}_{k+1} \leftarrow \boldsymbol{t}_{k+1} + \beta_k\boldsymbol{p}_k$;
20:     $k \leftarrow k + 1$;
21: until $\|\boldsymbol{r}_{k+1}\| \le \epsilon$;
22: return $\boldsymbol{\Delta y}_{k+1}$;
```

3.3 Comparisons of parallel acceleration

Lemma 1 *(Amdahl's law)* [23] *For a parallel bundle adjustment system with p concurrent processors, the fraction that can be enhanced is ρ, and the fraction that can not be enhanced is* $1-\rho$*, then the overall speedup ratio can be determined by*

$$\gamma(p) = \frac{1}{(1-\rho)+\rho/p}. \tag{20}$$

The fraction ρ is completely determined by what is the aims of the task and how is the parallel algorithm designed. If $\rho = 1$, the system can be perfectly parallelized, but $\rho < 1$ in real applications. In this case, the speedup ratio is bounded by

$$\gamma(p) \le \lim_{p\to+\infty} \frac{1}{(1-\rho)+\rho/p} = \frac{1}{1-\rho}. \tag{21}$$

Let N be the matrix dimension of the RCM. The naive Cholesky decomposition solution incurs cubic time complexity, *i.e.*, $O(\frac{N^3}{3})$. Let π be the number of nonzero entries, and κ be the condition number of the RCM, the time complexity of single-thread preconditioned conjugate gradient (PCG) is then $O(\pi\sqrt{\kappa})$.

According to Lemma 1, ideally a parallel version of PCG would take $O(\frac{\pi}{\gamma(p)}\sqrt{\kappa})$ time with p processors. In practice, however, there is often additional scheduling and communication overhead, and different processors are usually not perfectly balanced. Thus, in a parallel environment whether by multiple processes or multiple threads, we can conclude the proposition of Lemma 1,

Theorem 3 *For a parallel bundle adjustment system with p concurrent processors considering overhead* $\mu(p)$*, the fraction that can be enhanced remains ρ, and the fraction that can not be enhanced is* $1-\rho$*, then the overall speedup ratio can be generated by*

$$\gamma'(p) = \frac{1}{(1-\rho)+\rho/p+\mu(p)}. \tag{22}$$

Intuitively, the overhead $\mu(p)$ is strictly increasing as the the number of processors increases, which affects the overall speedup. There is an equilibrium between $\gamma'(p)$ and the overhead $\mu(p)$, which is depicted by the the following experiments through different scales of datasets.

To study the time complexity of multi-process and multi-thread parallel methods, we compare the operations included in Algorithm 1 by Table 1. When loading data $f(\boldsymbol{x}, \boldsymbol{y})$, the startup overhead for multi-process is generally a bit larger than multi-thread on Linux platform, which is not dominant for high performance multi-node servers. However, the shared memory architecture of multi-thread will produce conflicts when reading or writing memory during the procedure of the whole algorithm. That is, the bandwidth of memory limits the concurrency of many processors for multi-thread, and the conflicts between threads will become the bottleneck as the number of processors increases.

To account for the overhead, we introduce two functions of p, $\gamma'_-(p)$ and $\gamma'_+(p)$ $(0 < \gamma'_-(p), \gamma'_+(p) < 1/(1-\rho))$, for the PBA and the proposed PPCG algorithm, respectively. When $p = 1$, multi-process and multi-thread will degenerate to single process, that is to say $\gamma'_-(p) = \gamma'_+(p)$. When very few processors (possibly 2 to 4 for our simulations, which depends on architectures and algorithms), the memory conflicts of multi-thread are comparable with multi-process, and $\gamma'_-(p) \approx \gamma'_+(p)$. When p grows much larger (5 or more), the memory conflicts are dominant as discussed above, which results to $\gamma'_-(p) < \gamma'_+(p)$. Then, the time complexity of PBA is $O\left(\frac{\pi}{\gamma'_-(p)}\sqrt{\kappa}\right)$, and that of PPCG is $O\left(\frac{\pi}{\gamma'_+(p)}\sqrt{\kappa}\right)$.

Additionally, RPBA distributes the whole block into a limited number of subblocks through consensus ADMM framework. Without considering the communication overheads of ADMM, the ideal time complexity of RPBA is $O\left(\frac{\pi}{p}\sqrt{\kappa}\right)$. Taking overheads into account, the actual ADMM iterates with sublinear convergence rate such that the time complexity of RPBA is $O\left(\frac{\pi}{\gamma'(p)}\sqrt{\kappa}\right)$ $(0 < \gamma'(p) < 1/(1-\rho))$ when we define a function, $\gamma'(p)$, to summarize it. Since the overhead of ADMM is very different from overheads produced by the OS and architectures, we can't compare the time complexity of RPBA with others in theory. The following experiments will compare the actual performance in real bundle adjustment applications. The time complexity of PPCG and the referenced algorithms are summarized by Table 2.

4 Experiments

We have implemented the proposed PPCG algorithm using C++11, and developed it on an Intel Core i5 8250 quad-core hyper-threading 1.6GHz personal computer running Linux Mint 19 Operating System with kernel 4.15.0-20-generic. The capacity of RAM is 8GB and the compiler is GCC7.3.0. A Message Passing Interface (MPI) based parallel framework is utilized for parallel implementation. State-of-the-art linear algebra software packages, BLAS and LAPACK, are employed for matrix computations. All experiments are performed with double-precision floating point numbers. The structure dimension is 3, and the camera dimension is 7 including 1 intrinsic parameter and 6 extrinsic parameters without considering distortion

Table 1 Operations of multi-process and multi-thread methods

Operation	Multi-Process	Multi-Thread
Loading Data	startup overhead and data transmission	startup overhead and data transmission
Reading Memory	exclusive memory, no conflict	shared memory, with conflicts
All_reduce	$O(N/p + \lg(p))$	$O(N/p + \lg(p))$, ideally for bipartition
Computation	Equivalent time	Equivalent time
Writing Memory	exclusive memory, no conflict	shared memory, with conflicts

parameters. Moreover, we also assume that the number of processes is not less than the number of processors to assure effective parallelization.

4.1 Dataset details

BAL benchmark datasets [2], *i.e.*, Ladybug-1723, Trafalgar-257, Dubrovnik-356, Venice-1776, Final-961 and Final-1936, are used in the experiments, as listed by Table 3. As well, the sizes and the sparsity of Hessian matrices built from these datasets are listed. The Hessian matrix are enormously large, but very sparse. Considering the memory limitation of SBA, we choose two subsets Final-961 and Final-1936 instead of the whole Final dataset.

Table 4 describes the characteristics of the six public data-sets. *Category* is how are the images are organized in the phase of feature extraction, and datasets with leaf images contains more observation information. Intuitively, datasets with leaf images contain more geometry information than skeletal sets, which is more efficient for reconstructions. *Camera number* are the number of cameras (or images) used while *Feature density* is the average number of feature points on one images. It is mentioned that datasets with larger feature density will produce more accurate point cloud.

The six public datasets are divided into 3 groups according to their characteristics, and we randomly choose one dataset from each of them for the following reconstructions, *i.e.* Final-1936, Ladybug-1723 and Trafalgar-257.

4.2 Reconstructions

With the applications of our algorithm for bundle adjustment, we perform the process of 3D reconstruction through the structure from motion method, and present the results of reconstructed point cloud. To verify the performance of bundle adjustment, we compare the reconstructed point cloud data before and after bundle adjustment. Figures 1, 2 and 3 depict the differences of the reconstructed point cloud for Final-1936, Ladybug-1723 and Trafalgar-257. All reconstructed point cloud data are observed in bird's-eye view.

Figure 1a and b present markedly the differences between reconstructions after and before bundle adjustment for Final-1936. It is shown that the reconstructed point cloud after bundle adjustment is optimized where the edges and figures are more clear. By contrast, the reconstructed point cloud without bundle adjustment is very indistinct, and nothing can be recognized except the contours. The feature density of Final-1936 is relatively dense. Every image has more features, which provides more geometry information for the nonlinear optimization procedure of bundle adjustment and produces more distinct point cloud.

Figure 1c and d present the differences between reconstructions after and before bundle adjustment for Ladybug-1723. There is significant improvements from our eye-sights where objects and edges are explicitly augmented after bundle adjustment. Scattering points are concentrated to their real positions, and we can observe bold trees and paths.

Figure 1e and f compare the point cloud after bundle adjustment with that before bundle adjustment for Trafalgar-257. Since the feature density of Trafalgar-257 is sparse, the point cloud without bundle adjustment is very sparse as well. The object points are not optimized and concentrated such that buildings and objects are extremely light. After bundle adjustment, the contours and edges of the buildings are augmented, the buildings and objects turn bold. As a result, more distinct point cloud can be observed with the optimization of bundle adjustment.

4.3 Convergence evaluation

The proposed PPCG algorithm is compared with the state-of-the-art algorithms, PBA [12] and RPBA [20]. Both single-thread and multi-thread algorithms are taken into consideration in our evaluation. For simplicity, single-thread PBA and RPBA are denoted by PBA-1 and RPBA-1 respectively, Two-thread PBA and RPBA are denoted by

Table 2 Time complexity analysis

Algorithm	SBA	PCG	PBA	PPCG	RPBA
Complexity	$O(\frac{N^3}{3})$	$O(\pi\sqrt{\kappa})$	$O(\frac{\pi}{\gamma'_-(p)}\sqrt{\kappa})$	$O(\frac{\pi}{\gamma'_+(p)}\sqrt{\kappa})$	$O(\frac{\pi}{\gamma'(p)}\sqrt{\kappa})$

Table 3 BAL datasets for simulations

Datasets	Object Points	Cameras	Observations	Hessian matrix	Sparsity
Ladybug-1723	156502	1723	678718	481567×481567	0.0068%
Trafalgar-257	65132	257	225911	197195×197195	0.014%
Dubrovnik-356	226730	356	1255268	682682×682682	0.0061%
Venice-1776	993909	1776	5001859	2994159×2994159	0.0013%
Final-961	187103	961	1692975	568036×568036	0.012%
Final-1936	649673	1936	5213733	1962571×1962571	0.0030%

Table 4 Dataset discriptions

Public Datasets		Category	Camera number	Feature density
Group #1	Final-961	with leaf images	many	dense
	Final-1936	with leaf images	many	dense
Group #2	Ladybug-1723	skeletal set	many	sparse
	Venice-1776	skeletal set	many	sparse
Group #3	Trafalgar-257	skeletal set	few	sparse
	Dubrovnik-356	skeletal set	few	sparse

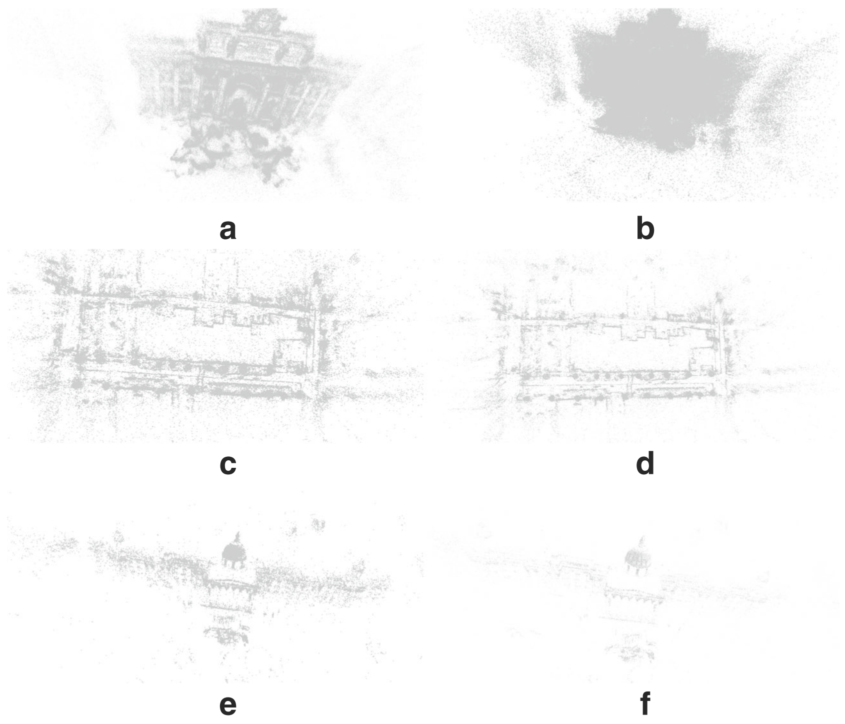

Fig. 1 Comparisons of the reconstructed point cloud for Final-1936, Ladybug-1723 and Trafalgar-257. **a**, **c** and **e** are reconstructed point cloud after bundle adjustment, **b**, **d** and **f** are reconstructed point cloud before bundle adjustment

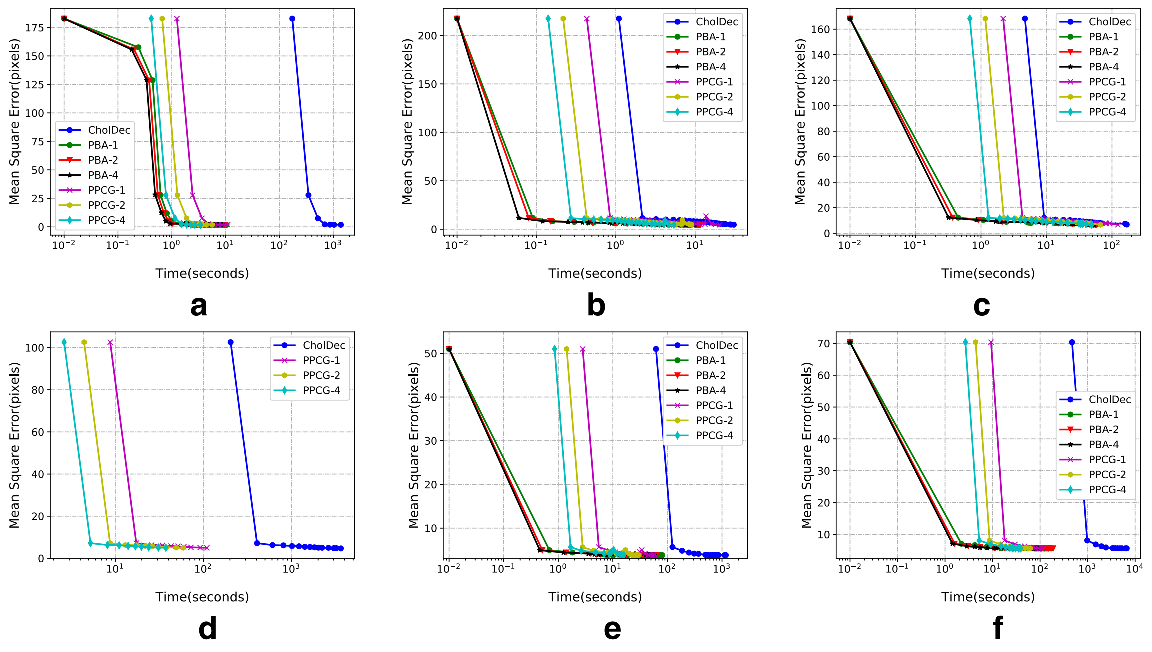

Fig. 2 Evaluations of convergence performance **a** Ladybug-1723 **b** Trafalgar-257 **c** Dubrovnik-356 **d** Venice-1776 **e** Final-961 **f** Final-1936

PBA-2 and RPBA-2, and et al. The convergence rates of CholDec, PBA-1, PBA-2, PBA-4, RPBA-1, RPBA-2, RPBA-4, PPCG-1, PPCG-2 and PPCG-4, are depicted by Table 5.

For each dataset, we run all algorithms until they converge to comparable MSE. In other words, throughout this section, all methods except RPBA, reach the same accuracy in all experiments. In a different way, RPBA uses the normalized weighted squared residuals (NWSR) to estimate global errors. The *Iterations* item summarizes iterations needed for an optimized solution for an algorithm, including successful LM iterations (the numerator) and total LM iterations (the denominator), the *Runtime* item is the time consumed for a whole bundle adjustment process while the *Speedup Ratio* item presents the speedup with respect to CholDec. Owing to the limitation that the source codes of

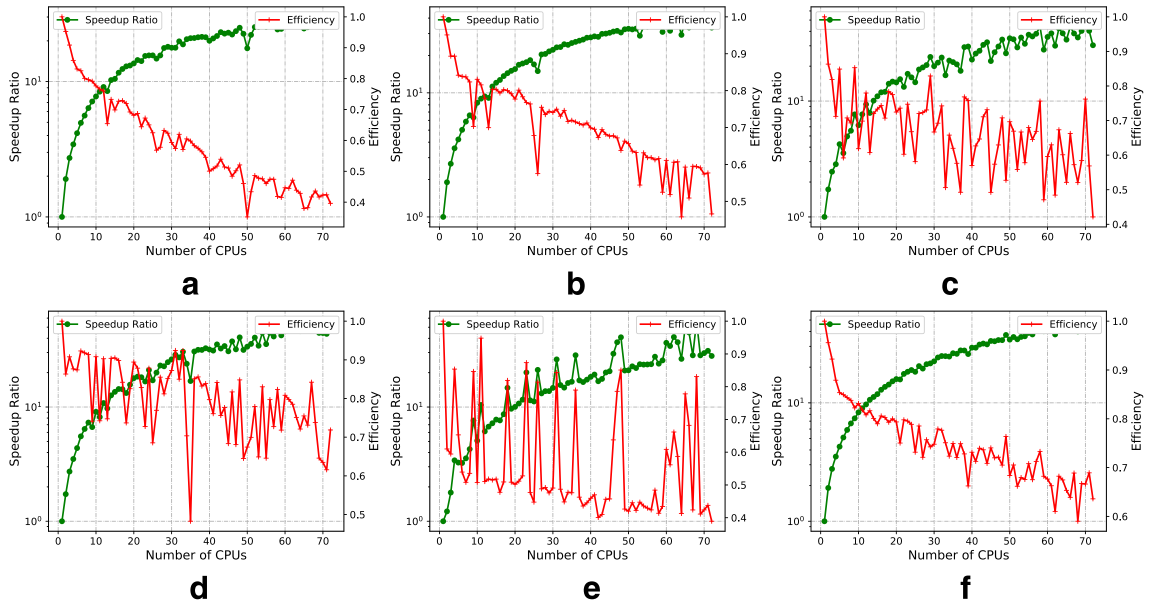

Fig. 3 Evaluations of parallel performance **a** Ladybug-1723 **b** Trafalgar-257 **c** Dubrovnik-356 **d** Venice-1776 **e** Final-961 **f** Final-1936

Table 5 Comparisons of convergence rate

Algorithms			Ladybug-1723	Trafalgar-257	Dubrovnik-356	Venice-1776	Final-961	Final-1936
Benchmark		Iterations	7/8	26/29	18/37	15/18	14/19	11/13
	CholDec	Runtime	1423.50s	30.88s	173.56s	3979.25s	1191.53s	5903.62s
		MSE	1.832	4.643	6.846	4.693	3.814	5.623
Single Thread	PBA-1†	Iterations	38/50	35/50	28/50	–	34/50	25/50
		Runtime	10.78s	11.27s	60.91s	–	81.54s	211.03s
		MSE	1.794	4.537	6.949	–	3.813	5.600
		Speedup	132.05×	2.74×	2.85×	–	14.61×	27.98×
RPBA-1	Iterations	2/2	2/2	2/2	2/2	2/2	2/2	
		Runtime	8.06s	6.97s	64.18s	114.77s	243.16s	689.74s
		NWSR	1.0*e*+10	1.0*e*+10	1.0*e*+10	1.0*e*+10	1.0*e*+10	1.0*e*+10
		Speedup	176.61×	4.43×	2.70×	34.67×	4.90×	8.56×
	PPCG-1	Iterations	9/11	22/24	18/24	10/10	10/15	11/11
		Runtime	10.94s	20.07s	122.79s	111.74s	58.62s	127.84s
		MSE	1.825	4.591	6.676	5.000	3.814	5.623
		Speedup	130.12×	1.54×	1.41×	35.61×	20.33×	46.18×
2 Threads	PBA-2†	Iterations	35/50	37/50	34/50	–	39/50	25/50
		Runtime	9.68s	12.06s	58.29s	–	63.15s	197.98s
		MSE	1.810	4.686	6.750	–	3.813	5.616
		Speedup	147.06×	2.56×	2.98×	–	18.87×	29.82×
	RPBA-2	Iterations	4/4	5/5	4/4	6/6	4/4	4/4
		Runtime	4.79s	1.91s	6.23s	72.84s	33.06s	108.43s
		NWSR	1.084	0.896	0.765	0.721	1.142	1.062
		Speedup	297.18×	16.17×	27.86×	54.63×	36.04×	54.45×
	PPCG-2	Iterations	8/9	22/25	18/25	10/10	11/14	11/11
		Runtime	5.68s	8.94s	64.26s	64.18s	28.75s	68.55s
		MSE	1.825	4.675	6.626	5.000	3.815	5.623
		Speedup	250.62×	3.45×	2.70×	62.00×	41.44×	86.12×
4 Threads	PBA-4†	Iterations	39/50	37/50	31/50	–	39/50	33/50
		Runtime	8.73s	10.35s	53.72s	–	57.71s	98.30s
		MSE	1.807	4.703	6.933	–	3.814	5.626
		Speedup	163.06×	2.98×	3.23×	–	20.65×	60.06×
	RPBA-4	Iterations	7/7	6/6	5/5	6/6	5/5	4/4
		Runtime	9.05s	1.19s	5.09s	42.12s	16.31s	45.92s
		NWSR	0.932	0.904	0.772	0.722	1.085	1.064
		Speedup	157.29×	**25.95**×	**34.09**×	94.47×	73.06×	128.56×
	PPCG-4	Iterations	8/9	22/23	18/31	10/10	9/12	10/10
		Runtime	3.40s	5.44s	47.90s	36.43s	15.34s	39.99s
		MSE	1.825	4.628	6.682	5.000	3.816	5.624
		Speedup	**418.68**×	5.68×	3.62×	**109.23**×	**77.67**×	**147.63**×

†PBA is failed on the Venice-1776 dataset

PBA and RPBA are optimized for shared memory architectures, the comparisons of convergence rate are only performed on a shared memory personal computer.

From the results of Table 5, it is demonstrated that PPCG-4 achieves absolutely best convergence performance among referred methods for most datasets. To measure the performance of the iterative methods, we choose CholDec as our baseline benchmark algorithm, and compares the run-time of PBA, RPBA and PPCG with it. CholDec converges much slower than other methods due to the fact that Cholesky decomposition is very complicated in time when the Hessian matrix grows. In contrast, PBA, RPBA and PPCG are globally faster than CholDec because each iteration of these iterative methods consume much less time for sparse problems, though more iterations are needed.

For datasets with more cameras, such as Ladybug-1723, Venice-1776, Final-961 and Final-1936, there is significant speedup for iterative algorithms over CholDec. Therefore, it can be concluded that the iterative methods, PBA, RPBA and PPCG, are highly effective for large problems, because the time complexity of CholDec (6) grows cubically for datasets with much more cameras and the iterative methods act more excellent than the direct method, CholDec when the RCM is very sparse. Without parallelization, PBA-1 gains speedup of 2.74 to 132.05, RPBA-1 gains speedup of 2.70 to 176.61 and PPCG-1 gains speedup of 1.54 to 130.12.

For parallel iterative methods, PBA, RPBA and PPCG with multiple threads obtain more significant speedup ratio than single-thread algorithms by reducing each iteration time through multi-threading techniques. PBA-4 achieves speedup of 2.98 to 163.06 times. PBA accelerates the traditional preconditioned conjugate gradient (PCG) [2] through multiple concurrent threads, but the efficiency is relatively low compared with the CPUs it occupies. RPBA-4 converges very fast with less iterations by improving the measurement with NWSR, and achieves speedup of 25.95 to 157.29 times. PPCG-4 achieves speedup of 3.62 to 418.68 times, and achieves speedup of 1.12 to 3.76 times against PBA-4, and obtains a speedup ratio of 1.06 to 2.66 over RPBA-4 for datasets excluding Trafalgar-257 and Dubrovnik-356 (Table 2).

Figure 2 describes the convergence curves of the algorithms, which is more clear for the trajectory of convergence. Since NWSR is a measurement which is very different from MSE, RPBA is not included in Fig. 2. The much steeper curve demonstrates much faster convergence for each iteration, and the much earlier ending point demonstrates much faster global convergence. For all of the datasets, the curve of PPCG-4 converges fastest and ends earliest among the referred algorithms excluding RPBA-1, RPBA-2 and RPBA-4.

4.4 Parallelization evaluation

We evaluate the parallel performance of our algorithm on a multi-node server. Each node in the server is equipped with 24 Intel IA-64 cores running at 1.2GHz without hyper-threading, 64GB shared RAM and 64bit Linux operating system for each computing node. PPCG is not constrained by share-memory architectures, and can be extended to multiple computing nodes, which is evaluated on the server to measure the parallel performance. The speedup ratio and the efficiency from 1 to 72 processors (3 computing nodes) are depicted by Fig. 3.

Based on the experimental results, we conclude that the speedup ratio grows as the number of processors grows, and approaches to a certain value due to the existences of the bound and overhead while the efficiency acts in the opposite way. It is shown that PPCG gains excellent parallelization whether in small or large scale problems. Moreover, the curves of speedup ratio and efficiency cross at one point, where the equilibrium of them is guaranteed. The real performance of computing nodes is fluctuating, which results to the non-smooth curves, but the trends demonstrate the deterministic or functional relationships.

4.5 Bound and overhead

Categories and descriptions for the overhead of a multi-node server are listed by Table 6, as well as functions under the MPI framework. The overhead of imbalanced workload can be eliminated by well-designed parallel algorithm while the others are determined by hardware architectures and parallel framework. When transmitting data, the nodes have to created connections to destination nodes. The overhead caused by this whole procedure is connection setup overhead, which is increased when the increasing processors enlarge routing time. And the communication overhead is the time consumed by data transmission after connections.

Theoretically, even more computing nodes or processors can be included to get much larger speedup ratio, but the bound of the speedup ratio (discussed in Section 3.3) will restrict the overall speedup ratio, as well as the overhead of startup and communications among different processes. Firstly, according to Lemma 1, the speedup ratio $\gamma'_+(p)$ will not increase as it approaches $1/(1-\rho)$ when adding more processors. This characteristic is owing to the algorithm or the task it self. Secondly, assuming that $\gamma'_+(p) << 1/(1-\rho)$, additional processors will speedup the whole task when *the computation burden* is greater than *related overhead* for each process. As adding more processors, the

Table 6 Overhead of multi-node computers

Category	Description	MPI Functions
Startup	process initiation, connection setup	MPI_Init, MPI_Send, MPI_Recv, and etc.
Communication	data transmission	MPI_Send, MPI_Recv, and etc.
Workload	barriers caused by imbalanced workload	optimized by algorithm design

computation burden decreases while overhead increases, which will results lower efficiency. As a result, for very large bundle adjustment problems, more computing nodes can be added for higher speedup ratio with acceptable efficiency.

5 Conclusion

This paper has proposed a PPCG algorithm to solve normal equations involved in bundle adjustment problems. A parallel Schur complement algorithm is devised to reduce the size of normal equations. Then, a parallel preconditioned conjugate gradient method is developed is to solve the reduced normal equations by parallelizing the original PCG algorithm. Taking the global performance into account, theoretical results ensuring the equivalence of matrix decomposition are presented with formal proof and the corresponding algorithm is developed to avoid expensive matrix computations. The experiments on a multi-node server show that PPCG achieves significant speedup ratios on several benchmark datasets compared to other algorithms, which reaching comparable accuracy. Moreover, the speedup ratio and the computational efficiency with respect to the number of processors are also evaluated and reported. Regarding future work, we plan to take PPCG as a local optimizer for consensus optimization based large-scale distributed bundle adjustment problems, and develop a distributed framework for further improvements with efficiency and scalability.

Acknowledgements This research was supported by the National Key Research and Development Program of China (2017YFB0202104).

Declarations

Conflict of Interests The authors declare that they have no conflict of interest.

References

1. Triggs B, McLauchlan P, Hartley R, Fitzgibbon A (2000) Bundle adjustment - a modern synthesis. In: Triggs B, Zisserman A, Szeliski R (eds) Vision algorithms: theory and practice. IWVA 1999 lecture notes in computer science, vol 1883. Springer, Berlin
2. Agarwal S, Snavely N, Seitz SM, Szeliski R (2010) Bundle Adjustment in the Large. In: European conference on computer vision. Springer, pp 29–42
3. Byröd M, Åström K (2010) Conjugate gradient bundle adjustment. In: European conference on computer vision. Springer, pp 114–127
4. Ma T, Kuang P, Tian W (2020) An improved recurrent neural networks for 3d object reconstruction. Appl Intell 50:905
5. Lhuillier M, Quan L (2005) A quasi-dense approach to surface reconstruction from uncalibrated images. IEEE Trans Pattern Anal Mach Intell 27(3):418
6. Klingner B, Martin D, Roseborough J (2013) Street view motion-from-structure-from-motion. In: Proceedings of the IEEE international conference on computer vision (ICCV). IEEE, pp 953–960
7. Engel J, Schöps T, Cremers D (2014) LSD-SLAM: Large-scale direct monocular SLAM. In: European conference on computer vision. Springer, pp 834–849
8. Vijayanarasimhan S, Ricco S, Schmid C, Sukthankar R, Fragkiadaki K (2017) Sfm-net: Learning of structure and motion from video
9. Zhou T, Brown M, Snavely N, Lowe DG (2017) Unsupervised learning of depth and ego-motion from video. In: Proceedings of the IEEE conference on computer vision and pattern recognition. IEEE, pp 1851–1858
10. Wang C, Miguel Buenaposada J, Zhu R, Lucey S (2018) Learning depth from monocular videos using direct methods. In: Proceedings of the IEEE conference on computer vision and pattern recognition. IEEE, pp 2022–2030
11. Clark R, Bloesch M, Czarnowski J, Leutenegger S, Davison AJ (2018) Learning to solve nonlinear least squares for monocular stereo. In: Proceedings of the European conference on computer vision (ECCV). Springer, pp 284–299
12. Wu C, Agarwal S, Curless B, Seitz SM (2011) Multicore Bundle Adjustment. In: Proceedings of the 2011 IEEE conference on computer vision and pattern recognition. IEEE, pp 3057–3064
13. Agarwal S, Snavely N, Simon I, Seitz SM, Szeliski R (2011) Building Rome in a Day. In: Proceedings of the 12th International Conference on Computer Vision. IEEE, pp 72–79
14. Ni K, Steedly D, Dellaert F (2007) Out-of-Core bundle adjustment for large-scale 3D reconstruction. In: Proceedings of the IEEE international conference on computer vision (ICCV). IEEE, pp 1–8
15. Ni K, Dellaert F (2012) HyperSfM. In: 2012, Second international conference on 3D imaging, modeling, processing, visualization and transmission. IEEE, pp 144–151
16. Eriksson A, Bastian J, Chin TJ, Isaksson M (2016) A consensus-based framework for distributed bundle adjustment. In: The IEEE Conference on computer vision and pattern recognition (CVPR). IEEE, pp 1754–1762
17. Zheng S, Kwok JT (2016) Fast-and-light stochastic ADMM. In: Proceedings of the twenty-fifth international joint conference on artificial intelligece (IJCAI-16). Morgan Kaufmann, pp 2407–2413
18. Huang F, Gao S, Chen S, Huang H (2019) Zeroth-order stochastic alternating direction method of multipliers for nonconvex nonsmooth optimization. In: Proceedings of the twenty-eighth international joint conference on artificial intelligece (IJCAI-19). Morgan Kaufmann, pp 2549–2555
19. Yu Y, Huang L (2017) Fast stochastic variance reduced ADMM for stochastic composition optimization. In: Proceedings of the twenty-sixth international joint conference on artificial Intelligece (IJCAI-17). Morgan Kaufmann, pp 3364–3370
20. Mayer H (2019) RPBA - robust parallel bundle adjustment based on covariance information
21. Lourakis MIA, Argyros AA (2009) SBA: A software package for generic sparse bundle adjustment. ACM Trans Math Softw 36(1):2:1
22. Jeong Y, Nistér D, Steedly D, Szeliski R, Kweons IS (2012) Pushing the envelope of modern methods for bundle adjustment. IEEE Trans Pattern Anal Mach Intell 34(8):1605
23. Hennessy JL, Patterson DA (2012) Computer architecture a quantitative approach (5th edition). Elsevier, New York

Jiaxin Peng received his B.S. and M.S. degree in electronical engineering from Chongqing University respectively in 2008 and 2013. He is currently a Ph.D candidate at the School of Computer Science, National University Of Defense Technology, Changsha, China. His research interest includes computer vision and parallel algorithm.

Jie Liu is a professor in Science and Technology on Parallel and Distributed Processing Laboratory at National University of Defense Technology, China. He received the Ph.D. degree in computer science and technology from National University of Defense Technology. His research interest includes parallel algorithm and high performance computing.

Hua Wei is a lecturer at Hubei University of Science and Technology. She received B.S. degree in biological technology at Southwest University in 2007, and Ph.D. degree in environmental science from Chinese Academy of Sciences in 2013 respectively. Her research interest includes aquatic plant ecology, ecological engineering of environment, purification and restoration ecology.