

Distributed bundle adjustment with block-based sparse matrix compression for super large scale datasets

Maoteng Zheng

China University of Geosciences(Wuhan)
 zhengmaoteng@cug.edu.cn

Nengcheng Chen

China University of Geosciences(Wuhan)
 chennengcheng@cug.edu.cn

Junfeng Zhu

Mirage3D Technology
 jf_zhu@smartmappingtek.com

Xiaoru Zeng

Mirage3D Technology
 xr_zeng@smartmappingtek.com

Huanbin Qiu

Jiantong Surveying
 mrqq18@hotmail.com

Yuyao Jiang

China University of Geosciences(Wuhan)
 yuyaojiang@cug.edu.cn

Xingyue Lu

China University of Geosciences(Wuhan)
 xingyuelu@cug.edu.cn

Hao Qu

Mirage3D Technology Inc.
 quhao@smartmappingtek.com

Abstract

We propose a distributed bundle adjustment (DBA) method using the exact Levenberg-Marquardt (LM) algorithm for super large-scale datasets. Most of the existing methods partition the global map to small ones and conduct bundle adjustment in the submaps. In order to fit the parallel framework, they use approximate solutions instead of the LM algorithm. However, those methods often give sub-optimal results. Different from them, we utilize the exact LM algorithm to conduct global bundle adjustment where the formation of the reduced camera system (RCS) is actually parallelized and executed in a distributed way. To store the large RCS, we compress it with a block-based sparse matrix compression format (BSMC), which fully exploits its block feature. The BSMC format also enables the distributed storage and updating of the global RCS. The proposed method is extensively evaluated and compared with the state-of-the-art pipelines using both synthetic and real datasets. Preliminary results demonstrate the efficient memory usage and vast scalability of the proposed method compared with the baselines. For the first time, we conducted parallel bundle adjustment using LM algorithm on a real datasets with 1.18 million images and a synthetic dataset with 10 million images (about 500 times that of the state-of-the-art LM-based BA) on a distributed computing system.

1. Introduction

Bundle adjustment (BA) is a significant step for 3D reconstruction in both computer vision and photogrammetry communities. Its main objective is to recover the camera poses as well as the 3D points by minimizing the square sum of the reprojection errors. This nonlinear optimization problem is an ancient subject, having been studied for decades. Classic solutions such as Steepest descent, Gauss-Newton, and Levenberg-Marquardt (LM) [16, 19] have been proposed and widely applied, and plenty of libraries have been open sourced, such as Ceres [26], g2o [11], SBA [17], PBA [27], MegBA [22], and so on. Among them, the LM algorithm is most widely supported. In a bundle adjustment, the nonlinear system is firstly linearized at an initial guess, which is a given prior, and then solved via direct inversion, Cholesky decomposition, or preconditioned conjugate gradient (PCG) [5, 6].

The Hessian matrix in a linear system is usually reduced by the Shur complement trick, forming a Reduced Camera System (RCS) that is much smaller than the Hessian. As the number of images increases, even the memory requirement for an RCS is a challenging issue. The formation, storage and inversion of the RCS gradually become the bottleneck of the bundle adjustment for large-scale datasets. Some scholars argue that BA is indivisible and hard to fit for parallel computing [30]. To solve this problem, they have turned to finding approximate methods that are more suitable for

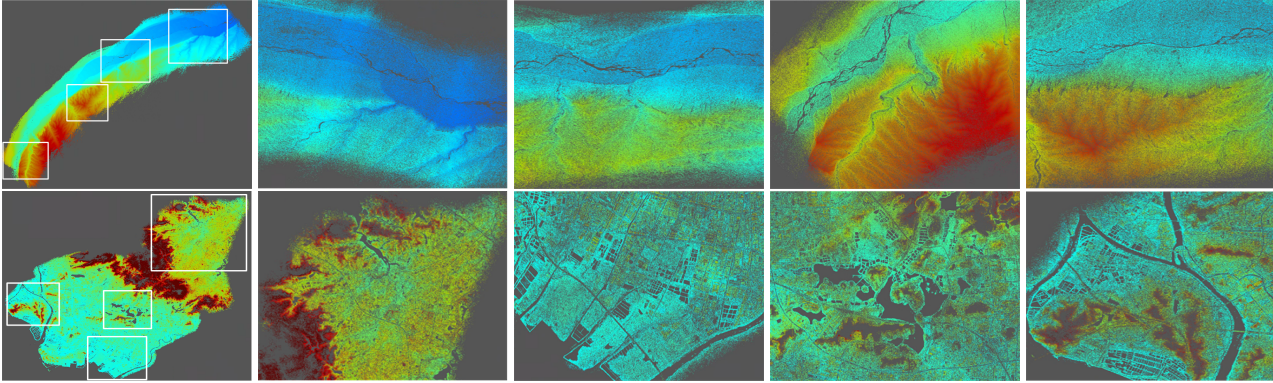


Figure 1. The sparse point cloud of a real dataset with 1.18 million images processed using the proposed method. The color from blue to red represents the height from low to high. The white rectangles on the left image are enlarged and shown on the right side

parallel computing [25, 8, 20, 28, 31, 7, 30, 13]. On the contrary, other works still utilize the LM algorithm, and attempt to design special parallel frameworks for it [27, 18, 22]. The former studies often give sub-optimal results, while the latter methods, despite running fast, are still limited to median-scale datasets (about 10k images).

2. Related works

For large-scale datasets at the city- or even world-level, some works just register images and give an approximate reconstruction [9, 15, 14, 24]. Heiny et al. [14] reported a method capable of processing a large dataset with 100 million images, and more than one million images were reconstructed. But the global bundle adjustment is never conducted due to the large memory requirement and heavy computation complexity. Some works try to optimize the global map, the most of them apply a divide-and-conquer strategy, partitioning the global map into submaps and then solve the submaps in parallel. The connection between submaps is either alternately optimized with submaps [21], or solved with alternating direction method of multipliers (ADMM) [8, 20, 28, 7, 30], global motion averaging [10, 12, 31] and domain decompositions [25, 13]. These methods, however, often give sub-optimal results because they are only approximations to the LM algorithm. As far as we know, the largest dataset processed by strict LM algorithm only includes about 20K images [22].

Another line of methods apply the traditional LM algorithm while using high-performance GPU hardware to accelerate the process. Agarwal et al. [3, 1, 2] first introduced a framework suitable for bundle adjustment with large-scale datasets. The framework is then implemented and accelerated on GPU hardware [27, 18]. More recently, Ren et al. [22] proposed a GPU-based distributed BA library, MegBA, and largely improved the efficiency of bundle adjustment using multiple GPUs. However, these methods consume a large amount of CPU/GPU memory, which limits their ap-

plication for larger datasets.

In this paper, to maintain high accuracy, we also utilize the LM algorithm. Different from the existing distributed methods, which partition the global map to small ones and conduct bundle adjustment in the submaps, we only perform a global bundle adjustment where the formation of the global RCS is actually parallelized and executed in a distributed fashion. The 3D points are firstly partitioned to different groups, each of which is submitted to a computing node along with the involved camera poses, and then sub-RCSs are generated at computing nodes. Finally, the global RCS is aggregated by summing all these sub-RCSs.

To store the large RCS, we compress it with a block-based sparse matrix compression format (BSMC) [29], which fully exploits the block feature of the RCS and subsequently achieves better performance than the widely used Compressed Sparse Row (CSR) format [23, 4] on both the compression ratio and accessing efficiency. Once the global RCS is formed, it is then solved by PCG, which is also implemented in parallel (multi-threads), and finally the unknowns are calculated and updated to all computing nodes for the next iteration.

The contributions of this work are as follows:

(1) We propose a distributed method to form the global RCS, which largely improves the efficiency and scalability.

(2) A block-based sparse compression (BSMC) format is applied to store the RCS and sub-RCS, and the compression ratio and the accessing efficiency of the compressed structure are largely improved.

(3) The proposed pipeline is compared with state-of-the-art methods. Numerous experimental results demonstrate the superiority of our proposed method over the baselines regarding memory usage and scalability.

(4) For the first time, we conduct parallel bundle adjustment using the exact LM algorithm on super-large datasets with up to 10 million images (500 times that of the state-of-the-art).

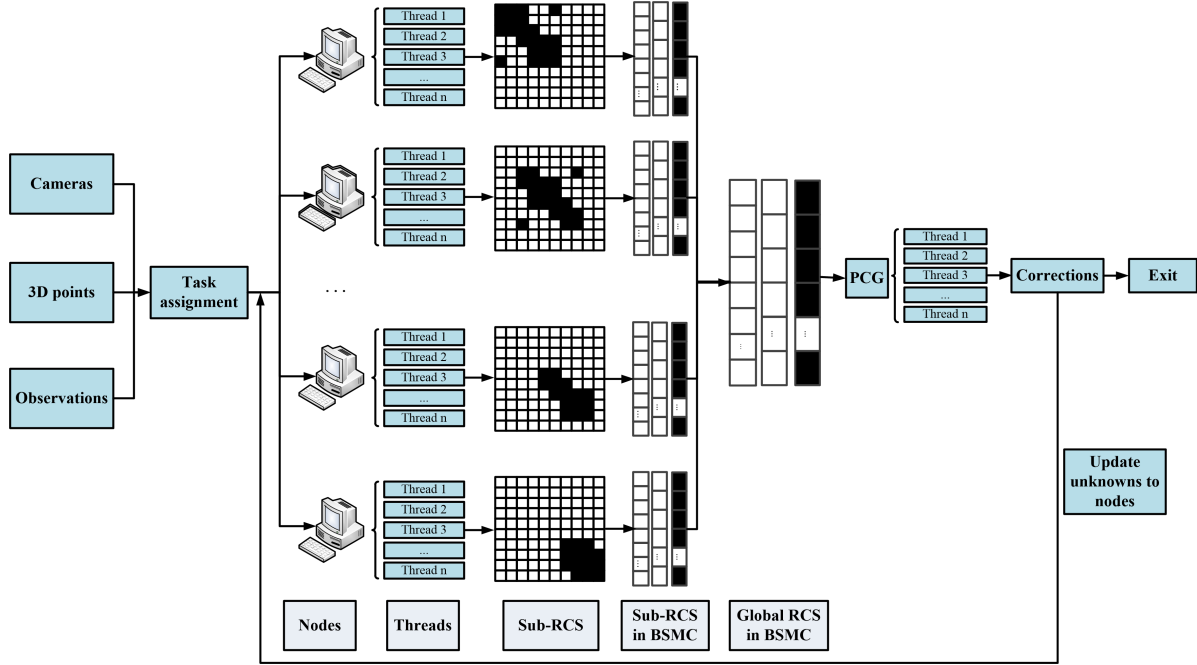


Figure 2. The pipeline of the proposed method. First, tie points are divided into groups and submitted to computing nodes along with the initial camera poses; then the sub-RCSs are generated and stored in BSMC format at the computing nodes; after that, the global RCS is obtained by aggregating all the sub-RCSs and then solved in parallel using PCG on the main node; finally, the camera unknowns are computed on the main node and updated on the computing nodes for the next iteration until the convergence is achieved.

3. Problem formation

3.1. The LM algorithm

Bundle adjustment optimizes the camera poses as well as the 3D points by minimizing the square sum of the re-projection errors, and has been widely studied for decades. First, we begin with the normal equation as follows.

$$(\mathbf{J}^T \mathbf{J} + \lambda \mathbf{D}^T \mathbf{D}) \Delta \mathbf{x} = -\mathbf{J}^T \mathbf{e} \quad (1)$$

Where \mathbf{J} is the Jacobian matrix, the damping matrix \mathbf{D} is usually a diagonal matrix extracted from the matrix $\mathbf{J}^T \mathbf{J}$, and the non-negative coefficient λ controls the damping strength. Vector $\Delta \mathbf{x}$ is the updating step of unknowns, and \mathbf{e} is the vector of reprojection errors.

The unknowns can be partitioned into camera part and ground-point part such that $\Delta \mathbf{x} = [\Delta \mathbf{x}_c \quad \Delta \mathbf{x}_p]$. Accordingly, the partitioning can be also performed to the Jacobian $\mathbf{J} = [\mathbf{J}_c \quad \mathbf{J}_p]$, and damping matrix $\mathbf{D} = [\mathbf{D}_c \quad \mathbf{D}_p]$ where the subscripts c and p denote the camera part and ground-point part, respectively. Then, Equation (1) can be rewritten as follows

$$\begin{bmatrix} \mathbf{U} & \mathbf{W} \\ \mathbf{W}^T & \mathbf{U} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}_c \\ \Delta \mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{l}_c \\ \mathbf{l}_p \end{bmatrix} \quad (2)$$

Where $\mathbf{U} = \mathbf{J}_c^T \mathbf{J}_c + \lambda \mathbf{D}_c^T \mathbf{D}_c$, $\mathbf{V} = \mathbf{J}_p^T \mathbf{J}_p + \lambda \mathbf{D}_p^T \mathbf{D}_p$, $\mathbf{W} = \mathbf{J}_c^T \mathbf{J}_p$, $\mathbf{l}_c = -\mathbf{J}_c^T \mathbf{e}$, and $\mathbf{l}_p = -\mathbf{J}_p^T \mathbf{e}$.

Matrices \mathbf{U} and \mathbf{V} are block diagonal, and matrix \mathbf{W} is a block-based sparse matrix. A block-wise Gauss elimination method is usually applied to eliminate the ground-point unknowns. Then, the number of unknowns in the adjustment is largely decreased because the number of cameras is usually much smaller than the number of ground points. After the elimination of ground points, we obtain:

$$(\mathbf{U} - \mathbf{WV}^{-1}\mathbf{W}^T) \Delta \mathbf{x}_c = \mathbf{l}_c - \mathbf{WV}^{-1}\mathbf{l}_p \quad (3)$$

$$\mathbf{V} \Delta \mathbf{x}_p = \mathbf{l}_p - \mathbf{W}^T \Delta \mathbf{x}_c \quad (4)$$

The camera unknown vector $\Delta \mathbf{x}_c$ is first solved according to Equation (3), and then the ground-point unknown vector $\Delta \mathbf{x}_p$ can be substituted according to Equation (4) or triangulated by camera poses. This is the so-called Schur complement trick. Equation (3) is the reduced camera system (RCS). In the following context, we refer the RCS to the matrix $\mathbf{U} - \mathbf{WV}^{-1}\mathbf{W}^T$ for simplicity.

3.2. Preconditioned conjugate gradient

The unknowns, although largely reduced by Shur complement, are still too much for direct inversion or Cholesky decomposition of the RCS. The conjugate gradient (CG) method is preferred. It is an iterative method to solve linear systems, whereby the direct inversion of the large matrix is avoided and only a matrix-vector product needs to be iteratively computed. In Equation (3), let $\mathbf{R} = \mathbf{U} -$

$\mathbf{W}\mathbf{V}^{-1}\mathbf{W}^T, \mathbf{y} = \Delta\mathbf{x}_c$, and $\mathbf{b} = \mathbf{l}_c - \mathbf{W}\mathbf{V}^{-1}\mathbf{l}_p$, and then it can be rewritten as follows

$$\mathbf{R}\mathbf{y} = \mathbf{b} \quad (5)$$

The above equation can be solved by CG. In general, a preconditioner \mathbf{M}^{-1} is applied to decrease the CG iteration times. The main task is now shifted to solving Equation (6).

$$\mathbf{M}^{-1}\mathbf{R}\mathbf{y} = \mathbf{M}^{-1}\mathbf{b} \quad (6)$$

This is the so-called preconditioned conjugate gradient (PCG). In this paper, a block diagonal preconditioner [3] is selected to conduct PCG. The detailed procedures for PCG can be found in numerous references; therefore, we omitted them here. The most time-consuming step in the PCG is a matrix-vector product. The others are only vector-vector products. Thus the parallel manipulation in Section 6.2 is only applied for the matrix-vector product

4. Block-based sparse matrix compression

The structure of an RCS is shown in Figure 4 (top). The non-zero blocks are determined by common points between cameras. If there are common points between camera i and j , then the block (i, j) is a non-zero block that consists of $c_i \times c_j$ elements, where c_i and c_j are the number of unknowns for camera i and j , respectively. Generally, the RCS is sparse; thus, it can be compressed by abandoning the zero elements. To this end, most of methods and libraries adopt the Compressed Sparse Row (CSR), which is a widely used sparse matrix compression format [23, 4].

The structure of CSR format consists of three arrays, the first array stores the non-zero elements, the second array stores the corresponding column identities, and the third stores the start column identities for each row (Figure 3 (bottom)). The sizes of the first and second arrays are the number of non-zero elements in the matrix, and the size of the third array is the row number of the matrix. However, this format doesn't take the advantage of the RCS's block characteristic (Figure 4 (top)). The Block-based Sparse Matrix Compression (BSMC) format is more suitable [29]. It takes the block as the minimal storage unit. Only the non-zero blocks in the upper triangle part, as well as their positions (column and row IDs) and sizes (width and height), are stored (Figure 4 (bottom)).

The BSMC format is more efficient than the CSR format as demonstrated in Figure 5 and Table 1. A good compression format should meet two requirements: the compression rate (the ratio of the compressed size and full size of the sparse matrix) should be as small as possible, and the accessing efficiency (decompression speed) should be as fast as possible. In this section, the BSMC is fully compared with the CSR format regarding these criteria.

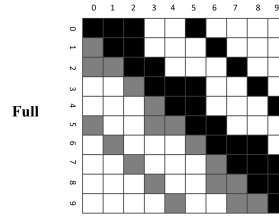
Full

1	3	0	7	0
0	4	0	5	0
0	8	5	0	6
9	0	0	7	0
0	3	0	0	2

CSR

Non-zero elements	1	3	7	4	5	8	5	6	9	7	3	2
Column Identities	0	1	3	2	3	1	2	4	0	3	1	4
Start and end positions	0	3	5	8	10	12						

Figure 3. A full matrix (top) and the corresponding compressed structure (bottom) with CSR. Find more details in the text.



BSMC

Blocks	[Shaded blocks]										...	[Shaded block]
Block column ID & Block size	0, c, r	1, c, r	2, c, r	5, c, r	1, c, r	2, c, r	6, c, r	2, c, r	3, c, r	7, c, r	...	9, c, r
Block start and end position	0	4	7	10	14	17	19	21	24	26		

Figure 4. The structure of an RCS with ten cameras in full matrix (top) and in BSMC (bottom). Find more details in the text.

4.1. Compression rate

The compression rate is defined as the ratio of the compressed size to the full size. The smaller the compression ratio is, the better the compressed format performs. Given a sparse matrix with n cameras, each of which contains c unknowns, and the sparsity (the ratio of the non-zero blocks and the total blocks) is α , the total number of non-zero blocks is αn^2 ; thus, the number of non-zero blocks in the upper triangle part is $\alpha n^2 - (\alpha n^2 - n)/2 = (\alpha n^2 + n)/2$. Generally, the data type for elements is double (8 bytes), and for the identities it is integer (4 bytes). Furthermore, the BSMC only stores the upper triangle part of the RCS. As such, the memory requirements of different compression formats can be calculated as follows:

$$Y_C = \alpha n^2 * c^2 * 8 + \alpha n^2 * c^2 * 4 + n * c * 4 \quad (7)$$

$$Y_B = (\alpha n^2 + n)/2 * c^2 * 8 + (\alpha n^2 + n)/2 * 3 * 4 + n * 4 \quad (8)$$

$$Y_C/Y_B = (6\alpha n c^2 + 2c)/(2\alpha n c^2 + 2c^2 + 3\alpha + 5) \quad (9)$$

Where Y_C and Y_B are the memory requirement for CSR and BSMC, respectively.

To compare the memory requirements of the two formats, the ratio of Y_C and Y_B is computed by Equation (9). To be specific, the number of camera parameters c is set to

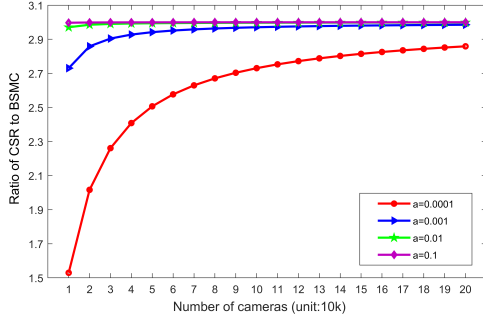


Figure 5. The relationship between the ratio of CSR to BSMC and the number of cameras for different sparsity a .

Format	First Operation	Second Operation
CSR	1 or $\log_2 nc$	$\sum_{i=0}^c \log_2 cs_i$
BSMC	1 or $\log_2 c$	$\log_2 s_i$

Table 1. Accessing efficiency of the BSMC and CSR formats

11 (6 for exterior and 5 for interior). We plotted the relationship between the ratio and the number of cameras ranging from 10k to 200k with different sparsities (Figure 5). As can be seen, the BSMC format is 1.5 to 3 times more efficient than the CSR format, especially for large-scale datasets.

4.2. Accessing efficiency

To evaluate the accessing efficiency of a compression format, we consider two operations. (1) Given an element or a block in the compressed structure, find its column and row identities in the full matrix; (2) Given the column and row identities of an element or a block in the full matrix, find the corresponding block in the compressed structure.

Table 1 shows the searching times of these two operations for the BSMC and CSR formats. Readers can find more details in the Supplementary Material. As shown in Table 1, if the elements or blocks in the compressed structure are accessed continuously, the first operation will be immediately performed for the two formats. In the real case, for matrix-vector product, the elements or blocks in the compressed structure are continuously accessed. However, for matrix updating or aggregation, the second operation is frequently applied. The accessing efficiency of BSMC for both the first and second operations is better than the CSR; thus, it's strongly recommended for the compression of block-based sparse matrices, such as the Hessian and RCS.

5. Formation of the RCS

The formation of the RCS is a major step in BA, dominating the most part of the computation complexity. In this section, we examine the details of the formation of an RCS and try to parallelize it. We find that the formation of the

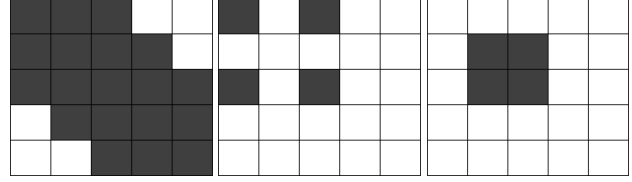


Figure 6. The formation of the RCS. A global RCS (left), consisting of 5 cameras, is aggregated by sub-RCSs generated by all 3D points. The right side shows two sub-RCSs. The middle is generated by a 3D point seen by camera 1 and 3, and the right is generated by a 3D point seen by camera 2 and 3, respectively.

RCS can be performed point by point as shown in Figure 8. The proof of this is omitted here due to the length limitations. Readers can find the details in the Supplementary Materials.

The formation of an RCS can be summarized as follows:

- (1) For each 3D point k , compute its Jacobian and Hessian, and then conduct the Shur complement to obtain a sub-RCS;
- (2) Update the RCS with the sub-RCS;
- (3) Stop until all the 3D points are completed.

According to the above procedures, the formation of the RCS can be divided into parallel tasks, with each consisting of a group of 3D points. Those tasks are independent to each other; therefore, they can be executed in a distributed way.

6. Parallelization

The time-consuming steps in BA include the formation of the RCS, which dominates most part of the computational complexity, the computation of PCG and the triangulation of 3D points. In this paper, the formation of the RCS is executed in a distributed way, and the computation of PCG and the triangulation of 3D points are both accelerated by multi-threading. However, the parallelization of triangulating 3D points is omitted here because it's simple.

6.1. Parallelizing the formation of the RCS

To parallelize the formation of the RCS, we first divide the 3D points to groups, with each submitted to a computing node along with the involved camera poses; and then generate sub-RCSs at computing nodes. Finally, we aggregate all the sub-RCSs to form a global RCS as shown in Figure 2. To be simple, we naively divide 3D points to groups with the same point number in this paper.

Note that, the sub-RCSs are first aggregated at the computing nodes to form a large sub-RCS, and then the large sub-RCS are aggregated to form a global RCS. We refer the large sub-RCS to sub-RCS for simplicity in the remainder of this paper. The sub-RCSs are directly stored in the main data center, which is connected with all computer nodes.

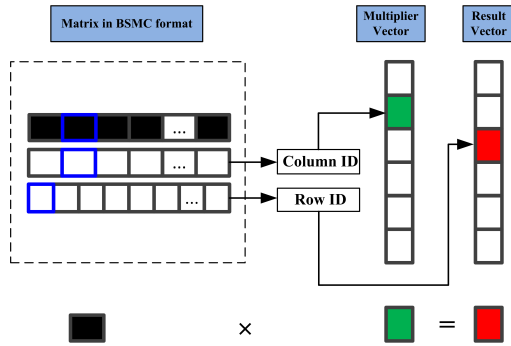


Figure 7. Multiplication of a matrix in BSMC format to a vector. Find details in the text.

The camera identities are reordered in the sub-RCS for efficient memory usage because they only involve a small part of the whole map as shown in Figure 2. The sub-RCSs and global RCS are all stored in BSMC structure. The column and row IDs of the blocks in the global RCS are stored in the second array of the BSMC structure for aggregating purpose as shown in Figure 4 (bottom). In the meantime, an extra array is added to store the local column and row IDs in the sub-RCS to update the sub-RCSs. The formation of the sub-RCS on each computing node is further parallelized by multi-threading, making the process even more efficient.

6.2. Parallelizing the PCG

Once the global RCS has been generated, the PCG algorithm is then applied to solve it. The most time-consuming step in the PCG is the multiplication of the RCS to the direction vector. Note that the RCS is stored in BSMC format. So, we first investigate the direct multiplication of a vector by a matrix in BSMC format. As demonstrated in Figure 4 (bottom) and Figure 7, the blocks are stored in the first array of the BSMC structure. We can successively read out the blocks and their column IDs in the full matrix, finding the row IDs in the full matrix by row counting according to the third array. The corresponding positions in the multiplier and result vectors can be determined by the block's column and row IDs in the full matrix, respectively. For each block, multiply it to the corresponding multiplier vector (green rectangle in Figure 7), and a component (red rectangle in Figure 7) of the result vector is generated. Finally, the result vector is obtained by aggregating all its components.

The multiplication of a BSMC-format matrix to a vector can be divided into the multiplication of blocks to the corresponding vectors as shown in Figure 7. Therefore, it is simple to parallelize this type of operation: First, divide the blocks into a number of groups, then submit them to multiple threads along with the corresponding multiplier vector, and finally collect all the outcome vectors and aggregate

them to obtain the result vector.

7. Experiments

The proposed method was implemented using C++ language. To setup the parallel environment, we used ten interconnected computers to form a Local Area Network (LAN). The main program was deployed on one of the computers called the main node, and the parallel tasks were executed on other computers called computing nodes. All the data were stored in the main data center, which was connected with all nodes. The communication speed between nodes and the data center can achieve 1GB/s. The computing nodes were equipped with Inter(R) Xeon(R) CPU E5-1650 v3 @3.5GHz, 64GB RAM and Windows 10 Operating System. The main node was equipped with the same CPU and operating system, but a larger 128GB RAM. A parallel task submission and management platform provided by Mirage3D Technology Inc. was used for parallel task assignment. The whole pipeline is also integrated in Mirage3D, which is a commercial 3D reconstruction software available at www.mirage3d.com. The code for BSMC is available at <https://github.com/MozartZheng/DistributedBA>, the public version of the whole pipeline will also be released when it is ready.

The proposed method is compared with the state-of-the-art bundle adjustment methods, such as Ceres [2], PBA [27], DeepLM [13] and MegBA [22]. These methods' experimental results on the public datasets were directly extracted from the papers of DeepLM and MegBA. To be as objective as possible, we didn't use the result of DeepLM method in its own paper; instead, we use the result provided by the paper of MegBA.

7.1. Datasets

To fully evaluate the proposed method, we test it with both synthetic and real datasets. Three public datasets, Ladybug, Venice, and Final were used to test and compare the proposed method with the state-of-the-art methods. The large-scale datasets collected by UAVs were used to test the scalability of the proposed method. The image numbers of those datasets range from 21K to 1.18M. Other methods, such as PBA, DeepLM, and MegBA were not tested with these datasets because they ran out of GPU memory. For Ceres, it encountered some kind of integer overflow error for these large datasets.

Moreover, to further test the proposed method's limit capacity, two synthetic datasets with 5 million and 10 million images are generated. To maintain simplicity, we intentionally generated the synthetic datasets with sparse features (about 300 features for each image) because the size of RCS is independent of image features. Random errors subject to normal distribution of (0.0, 1.0) pixels are added to these synthetic datasets. The statistics of the datasets are listed in

Dataset	Images	Cam.	3D pt.	Obs.	Spa.
Ladybug	1723	1723	0.16M	0.67M	0.040
Venice	1778	1778	0.99M	5.0M	0.082
Final	13682	13682	4.4M	29M	0.070
SW	21K	5	31M	166M	0.004
JLP	45K	5	44M	228M	0.002
YQC	46K	5	39M	254M	0.002
JZ	49K	5	41M	218M	0.001
XS	62K	95	55M	267M	0.001
DG	71K	5	87M	385M	9e-4
NM	85K	50	101M	638M	0.001
MCZ	86K	8	98M	588M	0.001
WQ	91K	60	93M	535M	8e-4
DaPu	133K	10	8M	61M	4e-4
HM	407K	35	352M	1790M	2e-4
AJH	442K	35	112M	1500M	3e-4
GDS	1.18M	114	1016M	5084M	8e-5
Syn1	5M	2	405M	2839M	3e-6
Syn2	10M	2	810M	5836M	1e-6

Table 2. The statistics of the datasets. The term Cam. represents cameras, pt. represents points, Obs. represents observations, and Spa. represents the sparsity.

Table 2. The sparsity is the ratio of non-zero blocks to total blocks in the RCS. The smaller the ratio, the sparser the RCS. The sparsities of the public datasets are the largest, possibly because of the large overlap. The real datasets are sparser because they are regularly collected by UAVs.

7.2. Memory usage and run time

The memory requirements of a dataset are mainly determined by the number of tie points (3D points and corresponding image points) and the size of the RCS. For the proposed method, the tie points were divided to groups, with each submitted to a computing node. Suppose ten computing nodes are applied, the memory requirement of the proposed method for tie points are theoretically 1/10 that of the serial method (such as Ceres). Meanwhile, the proposed method uses the BSMC format to store the RCS, which is 1.5 to 3 times more efficient than other methods applying the CSR format, as discussed in Section 4. The main node uses the most memory compared with other nodes because the global RCS is stored in the main node. Therefore, we define the main node’s memory usage as that of the proposed method.

To extensively evaluate the proposed method, we compared its memory usage, run time and accuracy with baselines using the public datasets. The results are listed in Table 4. As can be seen, the memory usage of the proposed method is the lowest since tie points are divided and stored in all nodes, and the memory-efficient format BSMC is ap-

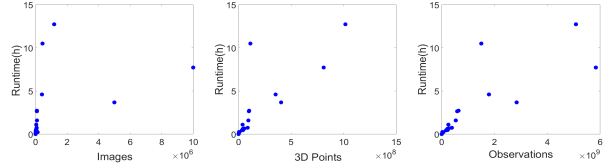


Figure 8. The runtime with respect to various parameters.

Data	Tie points	subRCSs	camera poses
Syn2	32	15	0.2

Table 3. Data transmission time for tie points (minutes), sub-RCS(seconds), and camera poses (seconds) among the network for the largest dataset Syn2

plied to store the RCS. The proposed method is faster than Ceres but slower than the GPU-based methods, MegBA, DeepLM, and PBA. MegBA runs fastest but requires multiple GPUs, as well as a large amount of memory. DeepLM is faster than PBA and its memory requirement is smaller; however, it might provide sub-optimal results. Although the GPU-based methods are faster than Ceres, they use almost the same memory. Therefore, for larger datasets, MegBA, DeepLM, and PBA all run out of memory. Owing to the distributed computing manner and memory-efficient format BSMC, the proposed method is superior to all the baselines regarding memory usage.

The super-large UAV and synthetic datasets are used to test the scalability of the proposed method. The memory usage, run time and accuracy are listed in Table 5. The memory requirements M_t and M_r are the theoretical memory required by the tie points and global RCS, respectively. The memory usage M_u is the actual memory consumption for the main node. The table shows that the actual memory usage is much smaller than the memory requirement because the tie points are stored in a distributed way, whereas the main node only stores 1/10 of the tie points. For the largest dataset with 10 million images and 5.8 billion observations, the memory usage of the proposed method is only 67.48 GB, which is affordable for an advanced modern computer.

About the communication overhead, there are three types of data need to be transferred among the nodes. First, the tie points are transferred to different computing nodes before LM iteration. Second, for each iteration, the sub-RCSs generated on the computing node should be transferred to the main node. Third, the camera poses for all images generated on the main node should be transferred back to the computing node for each iteration. As listed in Table 3, the time of data transmission for the largest dataset Syn2 is actually negligible compared to the total runtime which is 7.72 hours.

We plotted the run time with respect to various parameters (images, 3D points and observations) as shown in Fig-

Figure 8. The run time is affected by various factors, the image, tie point and iteration numbers. For an instance, the run time of the synthetic dataset with 5 million images is actually about 1/4 that of the real dataset GDS which includes only 1.18 million images. The reasons could be the following: First, the tie points of the synthetic dataset are actually about half that of the GDS dataset as shown in Table 2; second, the iteration number is also half that of the GDS. The same result can be found in the real dataset DaPu, which has more images but less tie points.

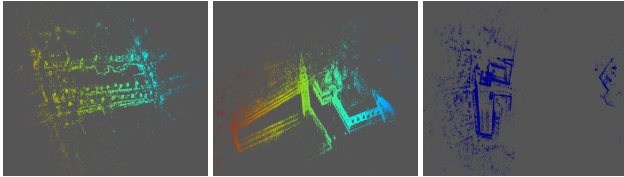


Figure 9. Sparse point cloud of the public datasets Ladybug, Venice and Final using the proposed method. The points in Final are all blue because there are some points far away from the main structure.

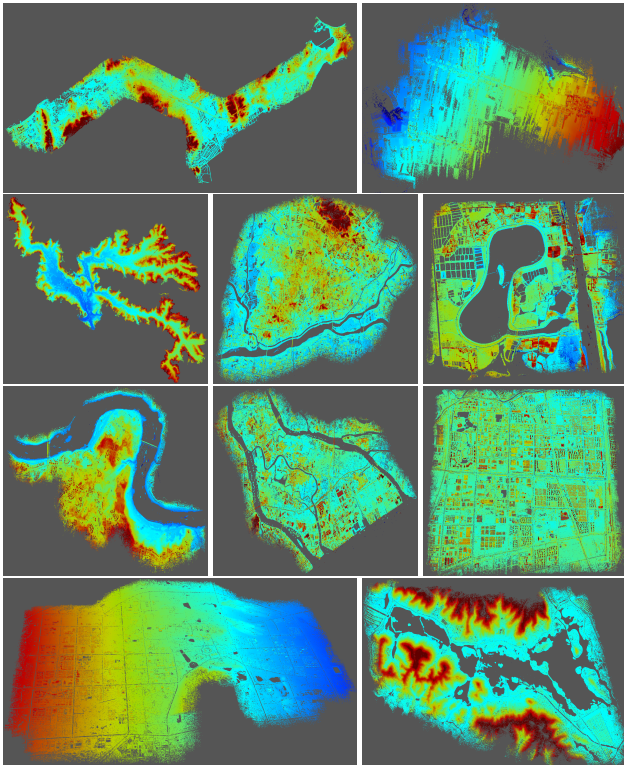


Figure 10. Visualization of the real datasets processed using the proposed method. From left to right, then top to bottom: SW, JLP, YQC, JZ, XS, DG, NM, MCZ, WQ, and DaPu, respectively.

7.3. Accuracy and scalability

As shown in Table 4, The accuracy of the proposed method is almost the same as the LM algorithm applied in Ceres, which is expected because the proposed method uses the exact LM algorithm. The difference is that the tasks of forming the RCS and computing the PCG are executed in a distributed way. MegBA achieves the best accuracy, which is unexpected since it also applies the exact LM algorithm and the difference is only that it is executed on GPU. The possible reason could be the strict outlier elimination strategy used in their method.

For the UAV datasets, the proposed method achieves about 1.2 pixels for most of datasets, and sub-pixel accuracy for two datasets. The sparse point clouds of the public datasets and the UAV datasets after bundle adjustment with the proposed method are shown in Figure 9 and 10, respectively. The proposed method successfully processed the super-large datasets (Syn1 and Syn2), and the accuracies are reasonable according to the errors added in the synthetic data, demonstrating the proposed method's vast scalability. To our own knowledge, datasets of this magnitude have never been handled by other LM-based methods. The proposed method overcomes the traditional LM algorithm's scalability bottleneck issue and relieves its limitations on image numbers by exploiting distributed computing and a memory-efficient compression format BSMC.

7.4. Ablation study for different parallel task assignments and node numbers

To investigate the efficiencies of different parallel task assignments and node numbers, we conduct ablation study on three datasets SW, DaPu and GDS using 2, 6 and 10 computing nodes, respectively. The GDS dataset is so large that we only test it on 10 computing nodes. The results are demonstrated in Figure 11. The parallel task number is neither the more the better, nor the fewer the better. For each dataset, there is a best parallel task number which achieves highest efficiency, but the best parallel task numbers various from different datasets. The best parallel task assignment scheme could be also related to the performance of the computing nodes. Nonetheless, the efficiency variation for different task assignment schemes is not big. However, it is certain that more computing nodes provide higher efficiency. The efficiency is generally linear to the number of computing nodes.

8. Conclusions

We parallelized the LM algorithm for bundle adjustment, which is thought to be indivisible. We found that the formation of the RCS can be divided by 3D points. So, the formation of the RCS can be executed in a distributed way using multiple computing nodes. We also introduced a block-

Dataset	Ceres-CG			PBA			DeepLM			MegBA			Ours		
	M	T	Acc.	M	T	Acc.	M	T	Acc.	M	T	Acc.	M	T	Acc.
Ladybug	0.52	46.7	1.14	0.3	12.3	2.22	2.1	3.9	1.12	1.6	0.77	0.56	0.06	15.6	1.12
Venice	3.68	1992	0.66	-	-	-	6.2	24.4	0.66	13.6	11.9	0.33	0.27	69.7	0.67
Final	16.8	3897	1.59	11.9	340	3.0	14.89	149	1.50	89.7	22.6	0.75	4.93	906	1.24

Table 4. Memory M(GB), run time T(s), and accuracy Acc.(pixels) of different methods.

Data	M_t	M_r	M_u	Iter.	T	Acc.
SW	6.3	0.53	1.21	6	0.44	1.29
JLP	8.7	1.21	2.22	4	0.48	1.37
YQC	9.1	1.37	2.44	7	1.10	0.89
JZ	8.3	0.69	1.60	6	0.62	1.29
XS	10.5	0.91	2.06	6	0.66	1.05
DG	15.5	1.39	3.10	5	0.74	1.24
NM	23.0	2.46	5.04	6	2.72	1.22
MCZ	21.6	2.22	4.63	7	2.65	0.69
WQ	19.9	2.00	4.22	5	1.59	1.24
DaPu	2.1	1.81	2.23	6	0.23	1.45
HM	69	9.72	17.73	6	4.60	1.27
AJH	46.3	16.18	22.64	6	10.50	1.28
GDS	197	28.22	51.15	6	12.73	1.23
Syn1	99.9	20.84	33.52	3	3.68	1.15
Syn2	204	41.70	67.48	3	7.72	1.17

Table 5. Memory(GB), run time T(h) and accuracy Acc.(pixels) of the proposed method for super-large datasets. The terms M_t and M_r show the theoretical memory that required by the tie points and the global RCS respectively. The memory usage M_u is the actual memory consumption on the main node. The term Iter. represents the iteration times.

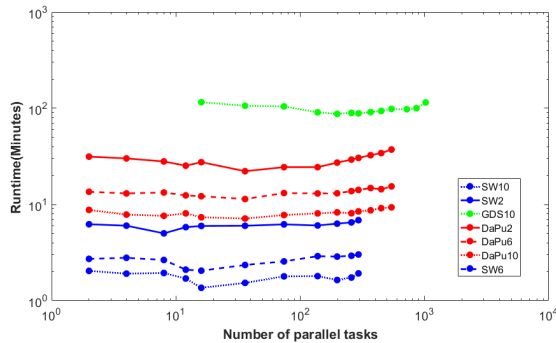


Figure 11. Ablation study for different parallel task assignments and node numbers. The blue, red and green lines represent SW, DaPu and GDS datasets respectively. The solid, long dashed and short dashed lines represent the results using 2, 6 and 10 computing nodes, respectively.

based sparse matrix compression format to store the global RCS and sub-RCSs. The BSMC format was compared with

the traditional sparse matrix compression format CSR, and our results demonstrate its superiority against the CSR on the compression rate and accessing efficiency. The other time-consuming steps in BA are also parallelized, such as the computation of PCG and the triangulation of 3D points. Based on the above, we proposed a parallel framework to conduct bundle adjustment with the exact LM algorithm in a distributed computing system. The proposed method is extensively evaluated and compared with the state-of-the-art methods using diverse datasets, including public internet datasets, real datasets collected by UAVs, and synthetic datasets. According to the experimental results and discussions, we conclude that

(1) The proposed method is memory efficient compared with the baselines due to the application of parallel computing and the memory-efficient compression format BSMC. The accuracy is the same as the Ceres since both methods apply the LM algorithm.

(2) The BSMC format can largely decrease the memory requirement of the RCS by 1.5 to 3.0 times compared with the traditional sparse matrix compression format CSR. The global RCS and sub-RCSs can be stored in a distributed manner based on the BSMC format.

(3) The vast scalability of the proposed method is verified by large-scale datasets with up to 10 million images. This magnitude of dataset has never been tested by other LM-based BA methods.

The parallel mode of the proposed method is suggested for large-scale datasets. However, the improvement on the small datasets (image number less than 2K) is not obvious. The serial mode of the proposed method is suitable for the small datasets since the BSMC format works for all datasets as long as they are sparse. A limitation of the proposed method is that the global RCS, though compressed by BSMC, is still stored on the main node, which means the proposed method will eventually fail as the dataset continues to scale up. We plan to not aggregate the global RCS on the main node at all, and direct conduct the PCG in a distributed way. This will be further studied in the following research.

Acknowledgement

This work is supported by the Special Fund of Hubei LuoJia Laboratory under grant 220100034.

References

- [1] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011. [2](#)
- [2] Sameer Agarwal and Keir Mierle. Ceres solver: Tutorial & reference. *Google Inc*, 2(72):8, 2012. [2](#), [6](#)
- [3] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part II 11*, pages 29–42. Springer, 2010. [2](#), [4](#)
- [4] Nathan Bell and Michael Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the conference on high performance computing networking, storage and analysis*, pages 1–11, 2009. [2](#), [4](#)
- [5] Martin Byröd and Kalle Åström. Bundle adjustment using conjugate gradients with multiscale preconditioning. In *BMVC*, pages 1–10. Citeseer, 2009. [1](#)
- [6] Martin Byröd and Kalle Åström. Conjugate gradient bundle adjustment. In *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5–11, 2010, Proceedings, Part II 11*, pages 114–127. Springer Berlin Heidelberg, 2010. [1](#)
- [7] Nikolaus Demmel, Maolin Gao, Emanuel Laude, Tao Wu, and Daniel Cremers. Distributed photometric bundle adjustment. In *2020 International Conference on 3D Vision (3DV)*, pages 140–149. IEEE, 2020. [2](#)
- [8] Anders Eriksson, John Bastian, Tat-Jun Chin, and Mats Isaksson. A consensus-based framework for distributed bundle adjustment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1754–1762, 2016. [2](#)
- [9] Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik, and Marc Pollefeys. Building rome on a cloudless day. In *European Conference on Computer Vision*, 2010. [2](#)
- [10] V. M. Govindu. Lie-algebraic averaging for globally consistent motion estimation. In *IEEE Computer Society Conference on Computer Vision & Pattern Recognition*, 2004. [2](#)
- [11] Giorgio Grisetti, Rainer Kümmerle, Hauke Strasdat, and Kurt Konolige. g2o: A general framework for (hyper) graph optimization. In *Proceedings of the IEEE international conference on robotics and automation (ICRA), Shanghai, China*, pages 9–13, 2011. [1](#)
- [12] Richard Hartley, Jochen Trunpf, Yuchao Dai, and Hongdong Li. Rotation averaging. *International journal of computer vision*, 103:267–305, 2013. [2](#)
- [13] Jingwei Huang, Shan Huang, and Mingwei Sun. Deeplm: Large-scale nonlinear least squares on deep learning frameworks using stochastic domain decomposition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10308–10317, 2021. [2](#), [6](#)
- [14] Heinly Jared, Johannes L Schonberger, Enrique Dunn, and Jan-Michael Frahm. Reconstructing the world in six days. In *CVPR*, 2015. [2](#)
- [15] Bryan Klingner, David Martin, and James Roseborough. Street view motion-from-structure-from-motion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 953–960, 2013. [2](#)
- [16] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of applied mathematics*, 2(2):164–168, 1944. [1](#)
- [17] Manolis IA Lourakis and Antonis A Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):1–30, 2009. [1](#)
- [18] Zheng Maoteng, Zhou Shunping, Xiong Xiaodong, and Zhu Junfeng. A new gpu bundle adjustment method for large-scale data. *Photogrammetric Engineering & Remote Sensing*, 83(9):633–641, 2017. [2](#)
- [19] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 11(2):431–441, 1963. [1](#)
- [20] Karthikeyan Natesan Ramamurthy, Chung-Ching Lin, Aleksandr Aravkin, Sharath Pankanti, and Raphael Viguier. Distributed bundle adjustment. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 2146–2154, 2017. [2](#)
- [21] Kai Ni, Drew Steedly, and Frank Dellaert. Out-of-core bundle adjustment for large-scale 3d reconstruction. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007. [2](#)
- [22] Jie Ren, Wenteng Liang, Ran Yan, Luo Mai, Shiwen Liu, and Xiao Liu. Megba: A gpu-based distributed library for large-scale bundle adjustment. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXXVII*, pages 715–731. Springer, 2022. [1](#), [2](#), [6](#)
- [23] Youcef Saad. Sparskit: a basic tool kit for sparse computations (version 2). *Computer Science Department, University of Minnesota*, 1994. [2](#), [4](#)
- [24] Tianwei Shen, Siyu Zhu, Tian Fang, Runze Zhang, and Long Quan. Graph-based consistent matching for structure-from-motion. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part III 14*, pages 139–155. Springer, 2016. [2](#)
- [25] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2004. [2](#)
- [26] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, pages 298–372. Springer, 2000. [1](#)
- [27] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *CVPR 2011*, pages 3057–3064. IEEE, 2011. [1](#), [2](#), [6](#)

- [28] Runze Zhang, Siyu Zhu, Tian Fang, and Long Quan. Distributed very large scale bundle adjustment by global camera consensus. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 29–38, 2017. [2](#)
- [29] Maoteng Zheng, Yongjun Zhang, Shunping Zhou, Junfeng Zhu, and Xiaodong Xiong. Bundle block adjustment of large-scale remote sensing data with block-based sparse matrix compression combined with preconditioned conjugate gradient. *Computers & Geosciences*, 92:70–78, 2016. [2](#), [4](#)
- [30] Lei Zhou, Zixin Luo, Mingmin Zhen, Tianwei Shen, Shiwei Li, Zhuofei Huang, Tian Fang, and Long Quan. Stochastic bundle adjustment for efficient and scalable 3d reconstruction. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XV*, pages 364–379. Springer, 2020. [1](#), [2](#)
- [31] Siyu Zhu, Tianwei Shen, Lei Zhou, Runze Zhang, Jinglu Wang, Tian Fang, and Long Quan. Parallel structure from motion from local increment to global averaging. *arXiv preprint arXiv:1702.08601*, 2017. [2](#)