DeepLM: Large-scale Nonlinear Least Squares on Deep Learning Frameworks using Stochastic Domain Decomposition

Jingwei Huang[1], Shan Huang[1], and Mingwei Sun[1,2]

[1]Rieman Lab, Huawei Technologies. [2]Wuhan University

Abstract

We propose a novel approach for large-scale nonlinear least squares problems based on deep learning frameworks. Nonlinear least squares are commonly solved with the Levenberg-Marquardt (LM) algorithm for fast convergence. We implement a general and efficient LM solver on a deep learning framework by designing a new backward jacobian network to enable automatic sparse jacobian matrix computation. Furthermore, we introduce a stochastic domain decomposition approach that enables batched optimization and preserves convergence for large problems. We evaluate our method by solving bundle adjustment as a fundamental problem. Experiments show that our optimizer significantly outperforms the state-of-the-art solutions and existing deep learning solvers considering quality, efficiency, and memory. Our stochastic domain decomposition enables distributed optimization, consumes little memory and time, and achieves similar quality compared to a global solver. As a result, our solver effectively solves nonlinear least squares on an extremely large scale. Our code will be available based on Pytorch[1] and Mindspore[2].

1. Introduction

Numerical optimization is an important stage in different problems across science and engineering. In computer vision and graphics, many problems are related to model fitting and can be formulated as nonlinear least squares including mesh processing [16, 43, 40, 34], motion and model reconstruction [46, 61, 47, 25, 19], and bundle adjustment [49, 36, 3, 54, 35, 57, 58] as a fundamental problem in 3D reconstruction. Nonlinear least squares are widely solved using Levenberg-Marquardt (LM) algorithm [30, 33] for its robustness and fast convergence. General solvers have been developed for academia and industry use like G2o [21] and Ceres [2]. However, they do not fit the increasing demand to efficiently handle extremely large problems with the availability of advanced hardware like GPUs.

Figure 1. Large-scale bundle adjustment on a deep learning framework using stochastic domain decomposition. We solve problems on a very large scale with high convergence quality.

Typical algorithms are specifically designed for efficiency or scalability. PBA [54] uses GPUs to accelerate the optimization for bundle adjustment. [19] proposes local schur complement for facial performance capture. [57] segments the graph and achieves camera consensus with ADMM [7]. STBA [58] accelerates the step computation by solving smaller problems with a correction stage. However, these methods cannot handle problems with billions of residuals, potentially sacrifice quality, or do not handle general nonlinear least squares problems.

Recently, deep learning frameworks like Tensorflow [1] and Pytorch [37] opens the potential to efficiently solve large-scale problems. It makes full use of computation resources like GPUs, provides an automatic backpropagation system to ease the user programming for general problems, and implements effective solvers for stochastic optimization. Unfortunately, they are not ideal for solving nonlinear least squares since gradient descent based optimizers

[1] https://github.com/hjwdzh/DeepLM

[2] https://gitee.com/mindspore/mindspore/tree/master/model_zoo/research/3d/DeepLM

converge much slower than commonly used LM algorithm. Therefore, we target at developing a stochastic LM optimizer inside deep learning frameworks to solve general and large nonlinear least squares with dramatic improvement in efficiency and scalability. Specifically, we mainly address two challenges for developing such an optimizer.

First, backpropagation cannot be directly used to compute sparse jacobians required by the LM algorithm. While backpropagation provides the partial derivative of a scalar function to each variable, jacobian matrix require derivatives of all residuals to related variables. It is impractical to call backpropagation for each residual since it yields a dense jacobian matrix and consumes a huge amount of memory and time. We find that indices of variables for residuals specify an one-to-one correspondence between nonzero entries of the jacobian and the derivatives of a scalar function. Based on it, We design a novel backward jacobian network to compute this scalar function, call a single backpropagation to collect derivatives and recover the jacobian.

Second, while batch-based stochastic optimization handles large problems, it easily causes LM solver to diverge since LM steps are usually large and require accurate jacobians computation from all data. We find that domain decomposition [9] can avoid the divergence. This approach segments variables into different domains and alternatively optimizes each domain by fixing variables inside other domains. From the training perspective, we view domains as batches and alternative optimization as batched optimization with multiple epochs. However, such an approach reduces convergence quality especially for boundary variables between different domains. To achieve high convergence quality, we propose a novel stochastic domain decomposition approach. Different from traditional domain decomposition, we derive different segmentation of domains using stochastic clustering so that variables at the boundaries of domains are changing for each epoch. This dramatically enhances the convergence quality for the optimization. We further provide an optional global reinitialization step at the beginning of each epoch. We abstract each domain with a descriptor preserving intra-domain residuals and globally optimize all descriptors for different domains.

In our experiments, we mainly study a fundamental problem as bundle adjustment. Figure 1 shows a large-scale bundle adjustment problem solved using our stochastic domain decomposition. Our LM solver significantly outperforms existing solutions considering quality, efficiency, and memory. Experiments show that our stochastic approach converges as fast as a global LM solver, and significantly reduces the consumption of memory and time. Ablation studies show that the key contribution to quality comes from stochastic decomposition. We provide a general, efficient, and scalable stochastic LM solver on deep learning frameworks for large-scale nonlinear least squares.

2. Related Works

Nonlinear Least Squares Many problems [16, 43, 11, 24] in computer vision and graphics require solving nonlinear least squares. Since these problems are usually considered sparse where each residual function takes a small number of variables, jacobian matrices can be efficiently evaluated and Levenberg-Marquardt (LM) algorithm [30, 33] is practically one of the best solution. Here, the core challenge is to save time and memory to handle large-scale problems efficiently. Fast and dedicated LM solvers can be implemented in GPUs [54] for specific problems or on regular domains created by a domain-specific language like Opt [14]. Fast and simple relaxation methods are used to further accelerate the speed [50, 51] sacrificing convergence quality. Local Schur complement [19] can further accelerate problems that can be decomposed into clusters with a small number of boundary variables. To save memory, matrix-free approaches [55, 61] are proposed by deferring the evaluation of residual derivatives. However, these methods are still not suitable for handling extremely large scale problems [57] with limited memory. Our deep solver not only utilizes GPU acceleration but also incorporates a stochastic batch-optimization approach to handle a large scale.

Stochastic Solvers Deep learning frameworks are popularly used for training deep neural networks. They provide powerful stochastic solvers with automatic differentiation [32] so that users can focus on network function design. For example, Tensorflow [1] and Pytorch [37] supports first-order gradient-based solvers including SGD [6], RMSProp [23], AdaGrad [15] and Adam [27]. Besides, these gradient-based solvers are perfect candidates to handle extremely large data via stochastic optimization. However, available deep solvers are not as good as the LM algorithm for nonlinear least squares problems considering convergence speed. We design a backward jacobian computation network and enable the LM solver in deep learning frameworks to optimize nonlinear least squares.

Domain Decomposition Domain decomposition [9, 44, 48] solves partial differential equations by iteratively solving subproblems in subdomains. Therefore, it reduces computation complexity and enhances parallelism and is a good candidate for large scale problems. Several recent methods tackle uncertainty across different domains by solving stochastic differential equations [39, 5, 56]. Different from their scenarios, our problem is nonlinear least squares without randomness but we introduce a stochastic clustering process during each epoch for high convergence quality.

Bundle Adjustment Bundle adjustment (BA) [49] is a fundamental problem in structure from motion [53, 41] that

requires solving nonlinear least squares. Recent works for this specific problem aim to achieve efficiency or a large scale. SBA [31] reduced the problem as pure a camera system using Schur complement. Sparse cholesky factorization approach is applied to the camera system with variable ordering [4, 13, 38]. Inexact solvers based on Conjugate Gradient (CG) [22] save memory and achieve better efficiency with the support of several different preconditioners [3, 26, 29]. SSBA [28] and PBA [54] designed efficient parallelized algorithms to accelerate the bundle adjustment. To handle a large scale, the problem is decomposed into clusters and solved in a divide-and-conquer fashion. [59, 60, 18] optimizes intra-cluster variables and merge clusters globally. [36, 35] optimizes global separators and base nodes followed by intra-cluster optimization. Motion averaging [10] can deliver suboptimal results for extremely large-scale problems that do not require subpixel accuracy. Recent methods [17, 57] exploits global optimization with distributed system using ADMM [7]. However, it requires careful hyper-parameter tuning and its inner iterations are time-consuming according to [58]. STBA [58] addressed point consensus inside the linear solver via constraint relaxation and correction. However, the correction step can still be inaccurate and prevent perfect convergence. Our solver exploits parallelism using deep learning frameworks and addresses the large scale with robustness and accuracy using novel stochastic domain decomposition.

3. Approach

We begin by discussing some background knowledge in Section 3.1. We show our backward jacobian network in Section 3.2 and introduce our formulation of nonlinear least squares using stochastic domain decomposition in Section 3.3 and Figure 2. In Section 3.4, we describe the details for stochastic variable graph clustering.

3.1. Background

Nonlinear Least squares problems can be formulated as energy minimization in Equation 1:

$$E(\mathbf{x}) = \frac{1}{2}\sum_{i}^{n} r_i(x_{s_{i,1}}, ..., x_{s_{i,k}})^2. \tag{1}$$

$\mathbf{x} = (x_1, ..., x_m)$ is an m-dimensional variable to be optimized. $\mathbf{r} = (r_1, ...r_n)$ is an n-dimensional residual function depending on $\mathbf{x}$. $1 \leq s_{i,j} \leq m$ is the index of variable for the j-th argument of i-th residual term r_i. Additional requirements is made that $s_{i,j}$ is unique for each residual. Our goal is to minimize the squared norm of $\mathbf{r}$. Newton step locally linearizes the residual function at $\mathbf{x}$ and find the optimal solution via

$$\min_{\mathbf{x}^*} ||\mathbf{r}(\mathbf{x}) + \mathbf{J_r}(\mathbf{x})(\mathbf{x}^* - \mathbf{x})||_2^2.$$

As a result, $\mathbf{x}$ is updated at each step as

$$\mathbf{x}^* = \mathbf{x} - \mathbf{H_r}(\mathbf{x})^{-1}\mathbf{J_r}^T(\mathbf{x})\mathbf{r}(\mathbf{x}).$$

Since hessian matrix $\mathbf{H_r}(\mathbf{x})$ is expensive to compute, it is approximated with $\mathbf{J_r}^T(\mathbf{x})\mathbf{J_r}(\mathbf{x})$ in Gauss-Newton method, which converges when final residual is small or nearly affine around the optimal location.

However, the linear approximation is inaccurate when a step size is big, and Gauss-Newton update does not guarantee to reduce the residual norm. Levenberg-Marquardt (LM) method adjusts the step size using a damping factor similar to Tikhonov regularization:

$$\min_{\mathbf{x}^*} ||\mathbf{r}(\mathbf{x}) + \mathbf{J_r}(\mathbf{x})(\mathbf{x}^* - \mathbf{x})||_2^2 + \lambda||\mathbf{D}(\mathbf{x}^* - \mathbf{x})||_2^2.$$

As a result, a LM step can be computed by solving

$$(\mathbf{J_r}^T(\mathbf{x})\mathbf{J_r}(\mathbf{x}) + \lambda\mathbf{D}^T\mathbf{D})\Delta\mathbf{x} = -\mathbf{J_r}^T(\mathbf{x})\mathbf{r}(\mathbf{x}) \tag{2}$$

where $\mathbf{D} = \sqrt{\text{diag}(\mathbf{J_r}^T(\mathbf{x})\mathbf{J_r}(\mathbf{x}))}$. λ is used to control the step size and can be adjusted using the trust-region method [8].

3.2. Backward Jacobian Network

Solving Equation 2 with a general solver requires an automatic evaluation of the jacobian matrix. A common solution is forward differentiation with multi-dimensional dual numbers [20]. However, it requires users to explicitly specify variable dimensions and is non-trivial to integrate into a standard deep learning framework. On the other hand, automatic differentiation with backpropagation cannot directly apply to jacobian. While it computes derivatives of a scalar function. we require the derivatives of each residual term to related variables. Therefore, the number of required backpropagation equals the residual dimension n. Additionally, backpropagation stores gradients for all variables and a dense jacobian matrix with dimension $n \times m$ will be created. This is impractical considering time and memory.

Our intuition is that derivatives of different residuals to non-overlapping variables can be evaluated with a single backpropagation from the sum of residuals. However, it is still challenging to segment residuals to non-overlapping groups and collect derivatives as a compact linked-list representation [19].

We define a new variable $\mathbf{y}$ such that $y_{i,j} = x_{s_{i,j}}$. We call $\mathbf{y}$ as "indexed variables" since it represents $\mathbf{x}$ indexed by $s_{i,j}$ in Equation 1. Our key idea is to collect derivatives of residuals to $\mathbf{y}$ instead of $\mathbf{x}$ to avoid overlapping issues. Since $s_{i,j}$ are unique for each residual r_i, jacobians to the original variables can be computed as shown in Equation 3.

$$\mathbf{J_r}(\mathbf{x})_{(i,s_{i,j})} = \mathbf{J_r}(\mathbf{y})_{(i,j)} = \nabla_{\mathbf{y}}(\sum_i r_i(\mathbf{y}))_{(i,j)} \tag{3}$$

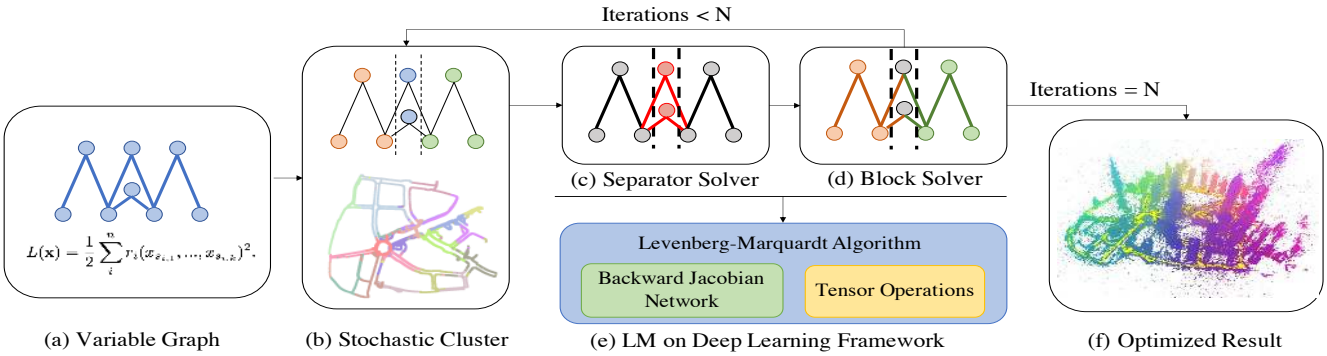

Figure 2. In each epoch, we stochastically recluster the variables into $C-1$ blocks and one separators cluster. We optimize separators and then variables in different blocks in parallel. With our backward jacobian network, we develop a LM solver and solve each optimization in a deep learning framework. We run multiple epochs to solve the problem.

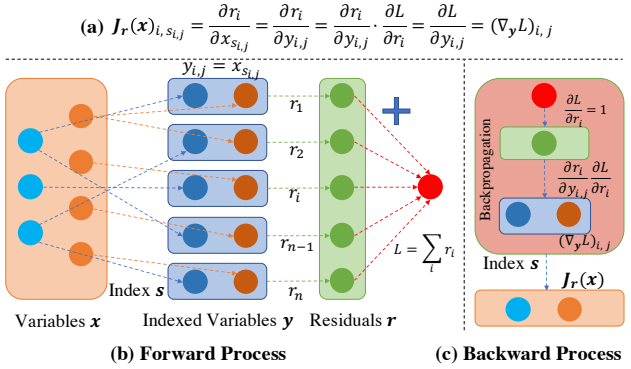

Figure 3. We treat the indexed variables as independent variables and collect derivatives of residual sums to these variables with a single backpropagation. Derivatives and variable indices form a triplet representation of the sparse jacobian.

We treat $\mathbf{y}$ as independent variables, and evaluate $\nabla_{\mathbf{y}}(\sum_i r_i(\mathbf{y}))$ with a single backpropagation. Afterwards, we use $< i, s_{i,j}, \nabla_{\mathbf{Y}}(\sum_i r_i(\mathbf{Y}))_{(i,j)} >$ to form a triplet representation for jacobian matrix.

Detailed mathematical derivation is shown in Figure 3(a). Based on it, we design a backward jacobian network in Figure 3 (b). The input to the network is original variables $\mathbf{x}$. We reindex the variables based on s to form indexed variables $\mathbf{y}$. $\mathbf{y}$ are sent to residual functions provided by users to evaluate residuals $\mathbf{r}$. Final loss L is computed as $\sum_i r_i$. During backpropagation, we follow Figure 3(c) and propagate derivatives to $\mathbf{y}$. Since each dimension of $\mathbf{x}$ corresponds to multiple r_i with multiple derivatives, our design allows to store them at multiple locations $\mathbf{y}$ duplicated from $\mathbf{x}$, and use $(i,j) \rightarrow (i, s_{i,j})$ to map derivatives at $\mathbf{y}$ to the jacobian. Our method computes the jacobian with a single backpropagation. Assuming residual dimension is n and each residual is related to k variables, our time and space complexity as $O(nk)$, while complexity of dual numbers [20] used by Ceres [2] is $O(nk^2)$. Rather than a neural network for training, our jacobian network is an architecture for structuring derivatives as a jacobian. Supplemental material provides implementation details for the LM solver.

3.3. Stochastic Domain Decomposition

For large scale problems, memory cannot afford the storage of jacobian or even all residuals. While batched-optimization can handle it via stochastic solvers, it easily causes the LM algorithm to diverge. We incorporate the idea of domain decomposition [9] to handle large-scale problems.

First, we build a variable graph $G =< V, E >$ where $V = \{x_1, ...x_m\}$ and E contains all pairs of variables that appear in at least one common residual. Such a graph is shown in Figure 2(a). In Figure 2(b), we segment the variables into C clusters and assign a cluster label $l_i \leq C$ for each variable x_i. We additionally ensure that variables with different labels won't appear in the same residual function unless either of them belongs to the first cluster. We call the first cluster the separators and other clusters the blocks. Instead of solving the global problem (Equation 1), we adopt the domain decomposition approach. We treat each cluster as a domain and solve variables in the same clusters alternatively by enumerating c from 1 to C in Equation 4.

$$\min_{\{x_i | l_i = c\}} \quad \frac{1}{2} \sum_i^n r_i(x_{s_{i,1}}, ..., x_{s_{i,k}})^2 \tag{4}$$

Since separators ensures that different blocks do not share residuals, it enables us to optimize separators (Figure 2(c)) followed by other blocks (Figure 2(d)) in parallel. We run multiple epochs to optimize the whole problem shown in Figure 2(b-d). For each optimization, we use our backward jacobian network to compute jacobian and run the LM algorithm with tensor operations on a deep learning framework.

One limitation in domain decomposition is that separator variables converges slowly. Therefore, we recluster the variables stochastically (details discussed in Section 3.4) so that the separators cluster is changing for each epoch.

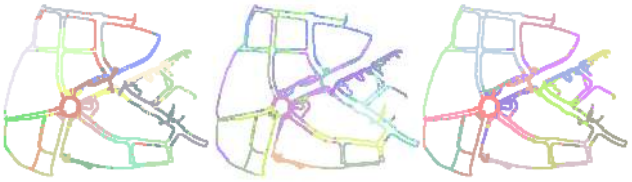

Figure 4. Stochastic segmentation from an example of a bundle adjustment problem. We segment cameras into different clusters, each labeled with a different color. Three different segmentation results are obtained by our algorithm for different epochs.

In addition, we optionally support user-specified global descriptors operated on blocks as a whole, requiring that such operations do not change intra-block residuals. We jointly optimize global descriptors together with separators in Figure 2(b), following Equation 5.

$$\min_{\{x_i, \mathbf{t}_j | l_i = 1\}} \frac{1}{2} \sum_i r_i(\mathbf{t}_{l_{s_{i,1}}} \circ x_{s_{i,1}}, ..., \mathbf{t}_{l_{s_{i,1}}} \circ x_{s_{i,k}})^2 \quad (5)$$

$\mathbf{t}_j$ is the global descriptor for cluster j that operates on all variables in j-th cluster, and $\circ$ is an operator invariant to intra-block residuals specified by the user. For example, in bundle adjustment or as rigid as possible deformation problems, $\circ$ can be defined as a rigid transformation operator and $\mathbf{t}_j$ represents rigid transformation with six parameters. Such global descriptors optimization leads to an effective global reinitialization over the whole graph structure. If such a global descriptor is not clear, we fix $\mathbf{t}_j \circ$ as identity operations so that the problem is reduced to Equation 4.

The concepts of separators, blocks, and global descriptor optimizations are extended from HyperSFM [35] for general nonlinear least squares with a general graph structure. Different from HyperSFM [35] who executes a global descriptor optimization followed by internal block optimization with a single epoch, we view it as a domain decomposition problem, execute multiple epochs, and rebuild domains by stochastic clustering every iteration. Furthermore, we only execute a single LM step for Equation 4 or 5 to save computation time for inner loops. Such a simplification achieves much better convergence than HyperSFM [35] with our stochastic domain decomposition.

3.4. Stochastic Variable Graph

For clustering, we aim at segmenting the variables evenly to achieve maximum parallelism. Second, we want to minimize the size of separators. Finally, we hope to recluster the problem stochastically so that the separators are changing for each epoch. The graph clustering problem has standard solutions including normalized cuts [42] used by [36] or spectral clustering [45] used by [19]. However, these methods produce fixed clusters and are inefficient for handling very large data.

Dataset	Method	# Cameras	# Blocks	# Separator
Ladybug	STBA	106	18	131209
	Ours-B	106	**16**	**60898**
Final	STBA	427	37	4156614
	Ours-B	427	**32**	**3177896**

Table 1. Stochastic clustering statistics. Using different modularity, we produce less separators and blocks than STBA [58] under the same constraint of maximum block size.

We adopt the solution provided by [58] for stochastic graph clustering. Specifically, each variable is initialized as a cluster, and clusters are greedily merged to maximize modularity with certain probabilistic distribution [12]. Different from [58], we minimize the size of separators to ensure fast convergence. Therefore, we prioritize to merge pairs of clusters with the maximum number of edges connecting them so that the number of edges accross different clusters are minimized. Accordingly, we define modularity as the number of edges in our variable graph internal to the same clusters. Clusters are merged following [58] (Equation 16) where Q is redefined using our modularity. We stop merging pairs if the number of variables in the merged cluster is larger than a threshold. After this step, we build the separator cluster so that blocks do not share residuals. In detail, for any edge in the variable graph that connects different blocks, we randomly pick one vertex of the edge and move it to the separator cluster.

Figure 4 illustrates segmentation in an example of a bundle adjustment problem. We segment cameras into different clusters, each labeled with a different color. Three different segmentation results are obtained by our algorithm for different epochs. According to the statistics in Table 1, the number of blocks and elements in the separator set from our formulation is less than the original formulation in [58] under the same constraint of maximum block size.

4. Evaluation

4.1. Results

Based on our backward jacobian network, we implement the LM solver on Pytorch where the linear equation is solved with preconditioned conjugate gradient [29]. We perform experiments on the bundle adjustment problems provided by datasets including 1DSFM [52] and BAL [3]. We use initialized bundle adjustment problems from BAL [3], and obtain the initialized problems for 1DSFM using Colmap [41]. We run experiments on machines with a Quadro P5000 GPU and a 16-core 3.7GHz CPU. We compare our methods with Ceres [2] using exact sparse linear solver (Ceres-S) and conjugate gradient (Ceres-CG). We additionally compare our methods with PBA [54] known as one of the most efficient solvers for bundle adjustment, and H-SFM [35] and STBA [58] as two state-of-the-art block-wise solvers for large-scale problems.

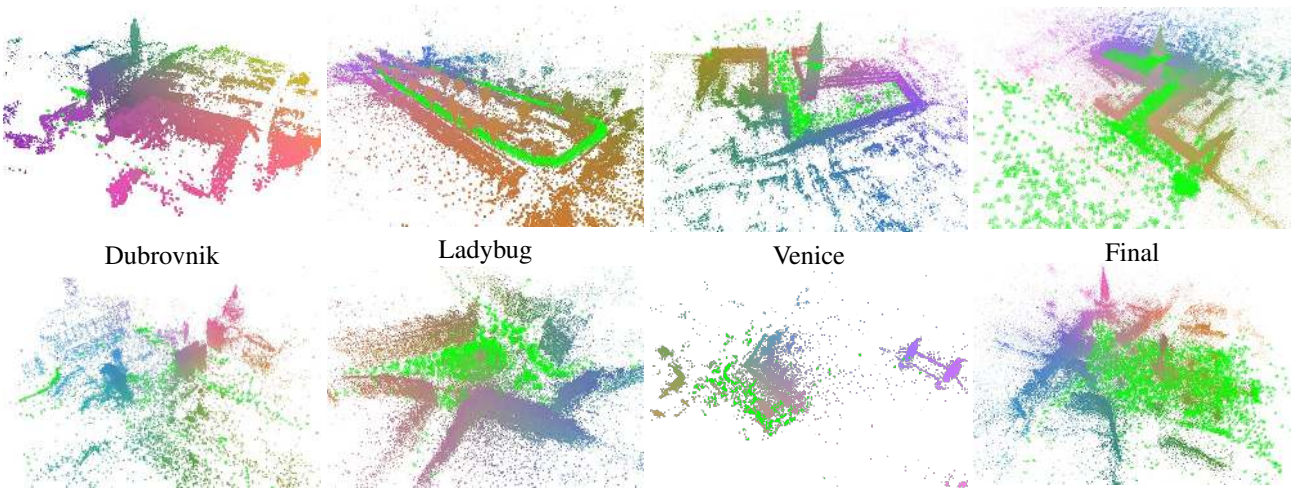

Figure 5. Visualization of problems from BAL [3] (first row) and 1DSFM [52] (second row) solved using our full pipeline "Ours-BG". Green frames represent cameras poses and point colors represent their 3D coordinates.

Dataset	# Cameras	# Points	# Projections	Ceres-S	Ceres-CG	PBA	H-SFM	STBA	Ours-G	Ours-BG	Ours-B
Trafalgar	257	65132	225911	**0.855**	0.862	1.913	3.423	1.020	0.858	0.861	0.892
Ladybug	1723	156502	678718	1.143	1.223	2.229	2.372	1.307	1.121	**1.102**	1.135
Dubrovnik	356	226730	1255268	**0.787**	0.788	1.899	2.325	0.963	**0.787**	0.788	0.813
Venice	1778	993923	5001946	**0.660**	0.664	-	2.158	0.775	0.662	0.664	0.701
Final	13682	4456117	28987644	-	1.587	3.004	3.394	-	**1.501**	1.505	1.524
Union Square	647	13368	108083	3.000	1.015	1.781	2.635	3.268	**0.826**	0.827	0.831
P. del Popolo	404	14128	128940	**0.957**	**0.959**	1.863	3.541	1.072	0.959	0.960	0.963
Ellis Island	367	20355	137450	**0.942**	**0.942**	2.300	3.338	1.105	**0.942**	0.944	0.949
NYC Library	491	21396	149720	**1.007**	1.010	2.363	3.902	-	1.009	1.011	1.013
M. N. Dame	547	33830	272575	1.058	**1.057**	2.553	4.001	1.176	1.058	1.059	1.065
Gen. markt	905	43620	280971	0.881	0.883	2.192	3.102	1.017	**0.880**	0.883	0.886
Alamo	741	31203	308147	0.971	0.970	2.138	3.081	1.025	**0.968**	0.971	0.972
Yorkminster	910	50947	333902	0.856	0.861	1.782	3.269	0.988	**0.840**	0.843	0.848
Roman Forum	1368	61008	435531	0.909	**0.906**	1.952	3.065	1.042	**0.906**	**0.906**	0.907
V. Cathedral	1016	56266	459014	0.913	0.895	2.168	3.043	1.033	**0.893**	0.895	0.899
M. Metropolis	524	86364	594848	0.459	**0.436**	0.981	1.331	0.523	0.438	0.438	0.440
Piccadily	2918	104027	823221	0.963	0.951	2.224	3.842	7.896	**0.950**	**0.950**	0.954
T. of London	814	185579	1512167	0.321	0.335	0.670	1.079	0.443	0.305	**0.303**	0.308
Trafalgar	7792	216650	1924901	1.377	1.392	3.267	5.192	1.641	**1.368**	1.370	1.374

Table 2. Statistics and mean squared errors produced by different methods for scenes in BAL [3] and 1DSFM [52] dataset.

For our methods, we report "Ours-G" as our Pytorch-based LM solver that optimizes the problem globally as a whole. "Ours-BG" and "Ours-B" represent stochastic domain composition using our whole pipeline with and without global reinitialization (Equation 5). For all scenes, we segment the global problem into 17 clusters and report a single run with 20 epochs. Problems solved by "Ours-BG" are visualized in Figure 5, where green frames represent cameras poses and point colors represent their 3D coordinates.

Quality We report mean squared error (MSE) after optimization to measure the optimization quality of different methods. We list the statistics of problems and MSE obtained from different methods in Table 2. Cells marked as "-" if optimization fails or running time is over 2 hours. Ceres-S is global optimization with an exact linear solver and usually delivers the best quality. Compared with other inexact solvers or blocked solvers, our global solver (Ours-G) achieves the best quality (marked in bold). It yields quite close or even better results compared to Ceres-S for several problems. Our stochastic solver "Ours-BG" and "Ours-B" achieves quite close quality compared to "Ours-G", and outperforms other state-of-the-art block solvers including H-SFM [35] and STBA [58].

Efficiency and Memory We report the time (second) and memory (GB) usage in Table 3 and Table 4 for BAL [3] dataset. Additional statistics for 1DSFM [52] are pro-

Dataset	Ceres-CG	PBA	STBA	Ours-G	Ours-BG	Ours-B
Trafalgar	65.1	16.8	59.6	3.44	2.14	**1.81**
Ladybug	46.7	12.3	89.4	5.87	5.53	**5.00**
Dubrovnik	320	25.5	213	13.1	3.97	**3.14**
Venice	1992	-	835	53.0	16.5	**14.2**
Final	3897	340	-	243	33.9	**25.4**

Table 3. Time (second) takes for different methods to optimize the problems in BAL [3] dataset.

Dataset	Ceres-CG	PBA	STBA	Ours-G	Ours-BG	Ours-B
Trafalgar	0.19	0.09	0.25	0.08	0.03	**0.01**
Ladybug	0.52	0.32	0.64	0.24	0.06	**0.03**
Dubrovnik	0.90	0.54	1.68	0.43	0.13	**0.04**
Venice	3.68	-	5.67	1.74	0.65	**0.24**
Final	16.8	11.9	-	9.73	3.52	**1.24**

Table 4. Memory (GB) used by different methods to optimize the problems in BAL [3] dataset.

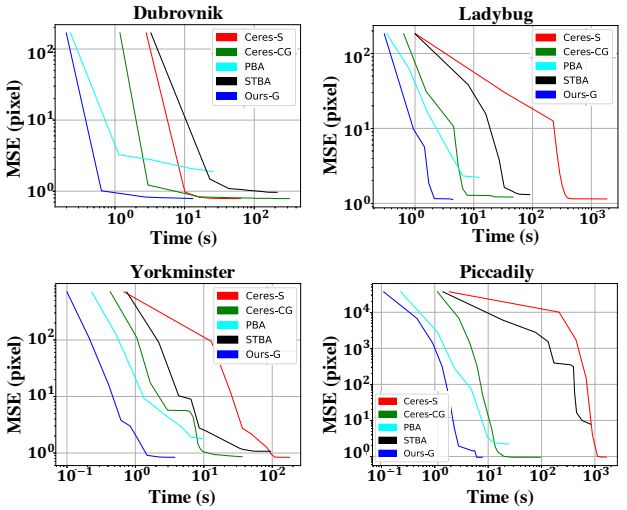

Figure 6. Convergence curves for different scenes in 1DSFM [52] and BAL [3].

vided in the supplemental material. Among existing global solvers, PBA [54] on GPU is faster than other algorithms run in multicore CPU implementation. Ours-G is even faster than PBA. We find that the major difference comes from accumulative summation inside $\mathbf{J}^T\mathbf{r}$. We directly call an efficient "index_add_" in Pytorch while PBA assign a thread to each variable x_i to collect $\sum_k \mathbf{J}_{k,i} r_k$, which is less efficient due to additional memory loading and uneven distribution of number of residuals related to different variables. As a result, our implementation can be more than 2x faster than PBA's kernel. Figure 6 plots the convergence curve for several problems in BAL [3] and 1DSFM [52] datasets. "Ours-G" is our solver shown in blue, which converges faster than other state-of-the-art methods.

With careful implementation, "Ours-G" uses less memory than other existing solvers. Typically, we use less memory to compute jacobians with our backward jacobian network. By solving subproblems in parallel for "Ours-BG" and "Ours-B", we significantly reduce the computation time and maximum memory usage for each sub-problem. Therefore, our stochastic solver supports the optimization of problems on a very large scale. "Ours-B" is more efficient than "Ours-BG" considering time and memory since it does not perform global reinitialization by slightly sacrificing the quality. In practice, either of them can be used depending on whether the application prefers quality or speed.

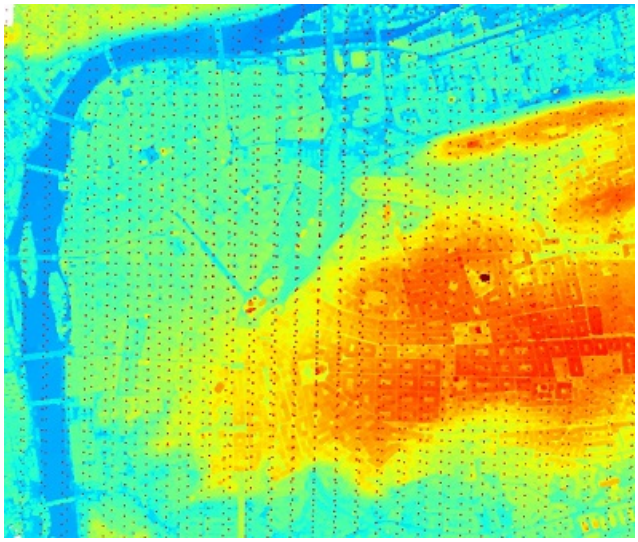

Figure 7. Visualization of city-scale bundle adjustment results processed using "Ours-B". Colors encode the z-values of 3D points.

# Points	# Cameras	# Projections	# Blocks
334.8M	0.28M	1236M	36
Memory	Time	Initial MSE	Final MSE
12.4G	403s	3.41	0.05

Table 5. City-scale dataset. We are able to process more than one billion residual terms by segmenting the data to 36 blocks and finalize the optimization with around 6 minutes.

Large Scale Figure 7 shows a large bundle adjustment problem with over one hundred thousand images solved with "Ours-B". Detailed statistics are reported in Table 5. We are able to process more than one billion residual terms in our self-collected dataset by segmenting the data to 36 blocks and finalize the optimization with around 6 minutes.

Deep Learning Solvers We compare our LM solver with other gradient-based optimizers in deep learning frameworks as shown in Figure 8. LM solvers converge much faster with better quality compared to SGD [6], RMSProp [23], and Adam [27].

4.2. Ablation Study

Ball Experiment To illustrate our intuition of why stochastic domain decomposition helps optimization, we consider a simple ball experiment in Equation 6 with its physical meaning illustrated in Figure 9(a) and (b). We have

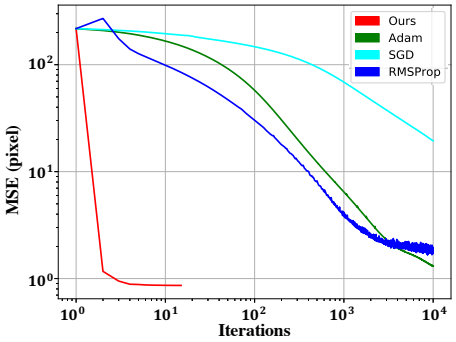

Figure 8. Convergence curve on Trafalgar dataset provided by BAL [3] using our method and deep learning solvers including SGD [6], RMSProp [23], and Adam [27].

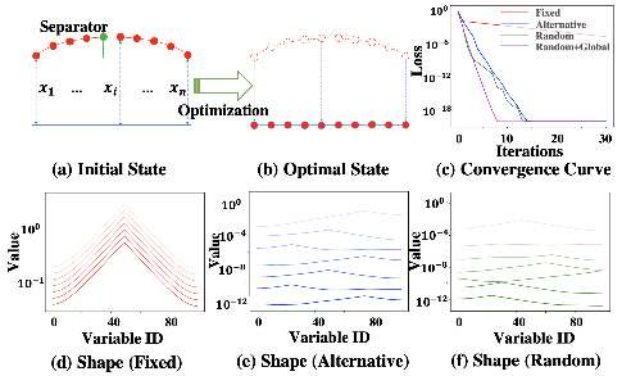

Figure 9. Illustration of the ball experiment. (a) We have N balls whose heights to the ground are x_i. (b) We optimize their heights so that each ball is close to both the ground and the mid point of the neighboring balls. (c) is the convergence curve of the loss to the number of iterations. (d)-(f) are shapes from the first 7 iterations over the optimization by different methods.

$N = 100$ balls whose heights to the ground are x_i. We optimize their heights so that each ball is close to both the ground and the mid point of its two neighbors.

$$E(\mathbf{x}) = \sum_{i=2}^{n-1} x_i^2 + (x_i - \frac{x_{i-1} + x_{i+1}}{2})^2 \qquad (6)$$

For traditional domain decomposition, we separate variables into two domains by cutting them in the middle and optimize both domains alternatively. The loss over iterations is shown as the red curve in Figure 9(c). We additionally visualize the results at the first seven iterations in Figure 9(d) (opaque value increases with the number of iterations). The convergence is slow especially for the separator in the middle. To change the separator, we split the domain at $25 \times (i \bmod 3+1)$ for the i-th iteration. The convergence curve and optimization states for the first seven iterations are shown in Figure 9(c) and (e) as the blue line. As a result, optimization converges much faster when we change the separator every epoch. By randomly pick the separator

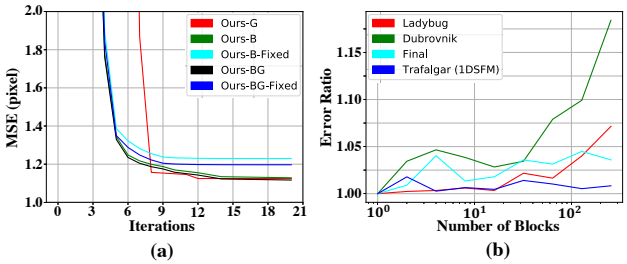

Figure 10. (a) Convergence curves of different variants of our methods for Bundle adjustment. Stochastic solvers ("Ours-BG" and "Ours-B") outperform fixed domain decomposition and achieve quite similar quality comparing to our global optimization. (b) Error ratio to the global solver using "Ours-B" with different numbers of blocks.

for each iteration, we derive the green curves in Figure 9(c) and (f), which shows similar convergence quality. Finally, with our global reinitialization as shown in the purple curve in Figure 9(c), convergence can be further enhanced.

Bundle Adjustment We plot the convergence curve for different variants of our methods in Figure 10(a) for the Ladybug scene in BAL [3] dataset. "Ours-G", "Ours-BG" and "Ours-B" are discussed in Section 4.1 as the global optimization, stochastic domain decomposition with and without global reinitialization. "Ours-BG-Fixed" and "Ours-B-Fixed" represent traditional domain decomposition where clusters are fixed over epochs with and without global reinitialization. "Ours-BG" and "Ours-B" converge quite close to our global optimization, while convergence is worse with traditional domain decomposition.

We plot the ratio of loss to the global optimization using "Ours-B" with different numbers of clusters for several bundle adjustment problems in Figure 10(b). The optimization quality is not sensitive to the number of clusters. We observe that for "Ladybug" and "Dubrovnik", optimization loss increases when the number of blocks is set to be 128 and 256. We believe the reason is due to too few cameras inside each cluster. For example, by segmenting the graph into 256 blocks, each block contains at most 2 cameras for the Dubrovnik scene. Therefore, the loss increase does not appear in larger scenes like "Final" and "Trafalgar".

5. Conclusion

We develop an efficient solution for large-scale nonlinear least squares problems based on deep learning frameworks. Our backward jacobian network enables our implementation of LM solver in Pytorch, and our stochastic domain decomposition method helps us robustly solve very large-scale problems with fast convergence.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, pages 265–283, 2016. 1, 2

[2] Sameer Agarwal, Keir Mierle, et al. Ceres solver. 2012. 1, 4, 5

[3] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. In *European conference on computer vision*, pages 29–42. Springer, 2010. 1, 3, 5, 6, 7, 8

[4] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905, 1996. 3

[5] Alexander Bihlo, Ronald D Haynes, and Emily J Walsh. Stochastic domain decomposition for time dependent adaptive mesh generation. *arXiv preprint arXiv:1504.00084*, 2015. 2

[6] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010. 2, 7, 8

[7] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011. 1, 3

[8] Richard H Byrd, Jean Charles Gilbert, and Jorge Nocedal. A trust region method based on interior point techniques for nolinear programming. 89(1):149–185, 2000. 3

[9] Tony F Chan, Tarek P Mathew, et al. Domain decomposition algorithms. *Acta numerica*, 3(1):61–143, 1994. 2, 4

[10] Avishek Chatterjee and Venu Madhav Govindu. Efficient and robust large-scale rotation averaging. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 521–528, 2013. 3

[11] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5556–5565, 2015. 2

[12] Yves Darmaillac and Sébastien Loustau. Mcmc louvain for online community detection. *arXiv preprint arXiv:1612.01489*, 2016. 5

[13] Timothy A Davis, John R Gilbert, Stefan I Larimore, and Esmond G Ng. A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):353–376, 2004. 3

[14] Zachary DeVito, Michael Mara, Michael Zollhöfer, Gilbert Bernstein, Jonathan Ragan-Kelley, Christian Theobalt, Pat Hanrahan, Matthew Fisher, and Matthias Niessner. Opt: A domain specific language for non-linear least squares optimization in graphics and imaging. *ACM Transactions on Graphics (TOG)*, 36(5):1–27, 2017. 2

[15] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7), 2011. 2

[16] Marvin Eisenberger, Zorah Lähner, and Daniel Cremers. Divergence-free shape correspondence by deformation. In *Computer Graphics Forum*, volume 38, pages 1–12. Wiley Online Library, 2019. 1, 2

[17] 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. 3

[18] Meiling Fang, Thomas Pollok, and Qum Chengchao. Mergesfm: Merging partial reconstructions. In *BMVC*, page 29, 2019. 3

[19] Marco Fratarcangeli, Derek Bradley, Aurel Gruber, Gaspard Zoss, and Thabo Beeler. Fast nonlinear least squares optimization of large-scale semi-sparse problems. In *Computer Graphics Forum*, volume 39, pages 247–259. Wiley Online Library, 2020. 1, 2, 3, 5

[20] Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008. 3, 4

[21] 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

[22] Magnus R Hestenes, Eduard Stiefel, et al. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49(6):409–436, 1952. 3

[23] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, 14(8), 2012. 2, 7, 8

[24] Jingwei Huang, Zhili Chen, Duygu Ceylan, and Hailin Jin. 6-dof vr videos with a single 360-camera. In *2017 IEEE Virtual Reality (VR)*, pages 37–44. IEEE, 2017. 2

[25] Jingwei Huang, Angela Dai, Leonidas J Guibas, and Matthias Nießner. 3dlite: towards commodity 3d scanning for content creation. *ACM Trans. Graph.*, 36(6):203–1, 2017. 1

[26] Yekeun Jeong, David Nister, Drew Steedly, Richard Szeliski, and In-So Kweon. Pushing the envelope of modern methods for bundle adjustment. *IEEE transactions on pattern analysis and machine intelligence*, 34(8):1605–1617, 2011. 3

[27] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2, 7, 8

[28] Kurt Konolige and Willow Garage. Sparse sparse bundle adjustment. In *BMVC*, volume 10, pages 102–1, 2010. 3

[29] Avanish Kushal and Sameer Agarwal. Visibility based preconditioning for bundle adjustment. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1442–1449. IEEE, 2012. 3, 5

[30] 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, 2

[31] 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. 3

[32] David J. C. Mackay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 1992. 2

[33] 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, 2

[34] Przemyslaw Musialski, Christian Hafner, Florian Rist, Michael Birsak, Michael Wimmer, and Leif Kobbelt. Nonlinear shape optimization using local subspace projections. *ACM Transactions on Graphics (TOG)*, 35(4):1–13, 2016. 1

[35] Kai Ni and Frank Dellaert. Hypersfm. Georgia Institute of Technology, 2012. 1, 3, 5, 6

[36] 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. 1, 3, 5

[37] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019. 1, 2

[38] Vladimir Rotkin and Sivan Toledo. The design and implementation of a new out-of-core sparse cholesky factorization method. *ACM Transactions on Mathematical Software (TOMS)*, 30(1):19–46, 2004. 3

[39] Abhijit Sarkar, Nabil Benabbou, and Roger Ghanem. Domain decomposition of stochastic pdes: theoretical formulations. *International Journal for Numerical Methods in Engineering*, 77(5):689–701, 2009. 2

[40] Kirk Schloegel, George Karypis, and Vipin Kumar. *Graph partitioning for high performance scientific simulations*. Army High Performance Computing Research Center, 2000. 1

[41] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4104–4113, 2016. 2, 5

[42] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8):888–905, 2000. 5

[43] Anna Shtengel, Roi Poranne, Olga Sorkine-Hornung, Shahar Z Kovalsky, and Yaron Lipman. Geometric optimization via composite majorization. *ACM Trans. Graph.*, 36(4):38–1, 2017. 1, 2

[44] Barry Smith, Petter Bjorstad, and William Gropp. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge university press, 2004. 2

[45] X Yu Stella and Jianbo Shi. Multiclass spectral clustering. In *null*, page 313. IEEE, 2003. 5

[46] Tomomichi Sugihara. Solvability-unconcerned inverse kinematics by the levenberg–marquardt method. *IEEE Transactions on Robotics*, 27(5):984–991, 2011. 1

[47] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2387–2395, 2016. 1

[48] Andrea Toselli and Olof Widlund. *Domain decomposition methods-algorithms and theory*, volume 34. Springer Science & Business Media, 2006. 2

[49] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999. 1, 2

[50] Huamin Wang. A chebyshev semi-iterative approach for accelerating projective and position-based dynamics. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015. 2

[51] Huamin Wang and Yin Yang. Descent methods for elastic body simulation on the gpu. *ACM Transactions on Graphics (TOG)*, 35(6):1–10, 2016. 2

[52] Kyle Wilson and Noah Snavely. Robust global translations with 1dsfm. In *European Conference on Computer Vision*, pages 61–75. Springer, 2014. 5, 6, 7

[53] Changchang Wu. Towards linear-time incremental structure from motion. In *2013 International Conference on 3D Vision-3DV 2013*, pages 127–134. IEEE, 2013. 2

[54] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *CVPR 2011*, pages 3057–3064. IEEE, 2011. 1, 2, 3, 5, 7

[55] Chenglei Wu, Michael Zollhöfer, Matthias Nießner, Marc Stamminger, Shahram Izadi, and Christian Theobalt. Real-time shading-based refinement for consumer depth cameras. *ACM Transactions on Graphics (ToG)*, 33(6):1–10, 2014. 2

[56] Dongkun Zhang, Hessam Babaee, and George Em Karniadakis. Stochastic domain decomposition via moment minimization. *SIAM Journal on Scientific Computing*, 40(4):A2152–A2173, 2018. 2

[57] 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. 1, 2, 3

[58] 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. *arXiv preprint arXiv:2008.00446*, 2020. 1, 3, 5, 6

[59] Lei Zhou, Siyu Zhu, Tianwei Shen, Jinglu Wang, Tian Fang, and Long Quan. Progressive large scale-invariant image matching in scale space. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2362–2371, 2017. 3

[60] 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. 3

[61] Michael Zollhöfer, Matthias Nießner, Shahram Izadi, Christoph Rehmann, Christopher Zach, Matthew Fisher, Chenglei Wu, Andrew Fitzgibbon, Charles Loop, Christian Theobalt, et al. Real-time non-rigid reconstruction using an rgb-d camera. *ACM Transactions on Graphics (ToG)*, 33(4):1–12, 2014. 1, 2