Matrix-Free Shared Intrinsics Bundle Adjustment

Daniel Safari
Sony Semiconductor Solutions

Abstract

Research on accelerating bundle adjustment has focused on photo collections where each image is accompanied by its own set of camera parameters. However, real-world applications overwhelmingly call for shared intrinsics bundle adjustment (SI-BA) where camera parameters are shared across multiple images. Utilizing overlooked optimization opportunities specific to SI-BA, most notably matrix-free computation, we present a solver that is eight times faster than alternatives while consuming a tenth of the memory. Additionally, we examine factors contributing to BA instability under single-precision computation and propose mitigations.

1. Introduction

Structure from Motion (SfM)[37] is a cornerstone of modern 3D reconstruction and view synthesis methods. Novel downstream work such as Multi-View Stereo[34], Neural Radiance Fields[19] and Gaussian Splatting[15] has been facilitated by the existence of SfM pipelines.

Bundle Adjustment (BA) – the joint refinement of image poses, landmark positions and camera intrinsics – is a core aspect of SfM. While ideally applied frequently, the associated computational cost unfortunately leads SfM pipelines to employ it conservatively.

Yet despite the existence of a large body of work dedicated exclusively to accelerating BA [7, 27, 38–41], major open-source SfM software such as COLMAP[33], TheiaSfM[35], OpenMVG[20] and GLOMAP[23] all use the same general-purpose non-linear least-squares solver, Ceres[2], as their BA backend.

A contributing factor to this curious development is the prominent status of the BAL[1] collection of SfM models as the de facto standard for BA benchmarking. Its models are generated from internet photo collections and thus share the distinctive property of every image having its own set of camera parameters. As a consequence, nearly all open-source BA implementations from academia – regardless of whether the underlying algorithms theoretically necessitate it or not – depend on aforementioned property to function at all.

Most SfM use cases do not involve photo collections. Capturing hundreds to thousands of images from just a few cameras is ubiquitous in practice. Refining SfM models stemming from the common case configuration calls for shared intrinsics bundle adjustment (SI-BA) – a BA variant seldom addressed by the literature.

This work seeks to remedy the current disparity between academic focus and practical application through the following contributions:

- Detailing a method of representing the SI-BA system suitable for exploiting its inherent structure (Secs. 4 and 5).
- Demonstrating that SI-BA facilitates matrix-free methods outperforming conventional approaches (Secs. 6 and 8).
- Examining numerical issues arising when performing BA in single-precision and proposing mitigations (Sec. 7).
- Sharing our solvers with the community.

2. Related Work

Background and basic building blocks of modern BA methods, such as the Preconditioned Conjugate Gradient (PCG) method and the Reduced Camera System (RCS), are detailed in the seminal work of [36].

Data structures and parallelization strategies for large-scale bundle adjustment are discussed in depth by [39]. To ease parallelization they bypass the augmented Hessian by implementing relevant matrix-vector products through the Jacobian instead. Matrix-free methods are introduced but degrade performance on CPUs. Our work is similar in theme yet takes the opposite approach. We circumvent the system Jacobian and focus instead on the augmented Hessian.

A method of efficiently constructing the angular BA normal equations by means of a compact linearization is presented by [40]. Notably, they discuss cache hit rates when constructing the RCS. Our work covers instead cache utilization when fetching image poses and landmark positions in the context of matrix-free matrix-vector products by the normal equations.

Discussion of single-precision computation is limited throughout the literature. Heuristics for improving numerical stability are given by [39]. Nullspace marginalization is shown by [7] to be a well-conditioned alternative to explicitly generating the RCS. To our knowledge, this paper is the first attempt at explaining the underlying causes of single-precision BA instability.

Power Bundle Adjustment by [38] is an alternative to PCG. Their method uses the augmented Hessian in a manner similar to PCG and therefore benefits equally from our work on accelerating matrix-vector products.

Apero[25] contains an SI-BA implementation; it is non-viable for large problem sizes due to solving the RCS through sparse Cholesky factorization [1, 38, 39].

Orthogonal to efforts on optimizing core computations, research exploring efficient BA preconditioners (e.g. [6, 14, 17]) and distributed computing through decomposition (e.g. [27, 41, 42]) should be mentioned.

3. Bundle Adjustment Abridged

As prior work describes the photo collection case, we must briefly introduce the shared intrinsics variant of BA in order to facilitate the rest of this paper.

BA revolves around a collection of images each observing a subset of 3D points called landmarks. Images contain keypoints specifying expected projections of observed landmarks. Camera parameters shared by one or more images affect the projection. The goal is to minimize the distance between landmark projections and keypoints by perturbing image poses, landmark positions and optionally camera parameters.

3.1. Normal Equations

The BA objective is $\min \frac{1}{2}\|r\|^2$ where r is a vector containing all reprojection errors. This is a nonlinear least-squares problem customarily solved by the Levenberg-Marquardt method, which in turn entails repeatedly solving the linear least-squares problem:

$$\min_x \|Jx + r\|^2 + \lambda \|x\|_D^2 \tag{1}$$

where x is a perturbation of the system state and J is the Jacobian of r w.r.t. x. The damping factor λ and associated scaling matrix $D = \operatorname{diag}(J^T J)$ stabilizes the system. The normal equations solving Eq. (1) are:

$$\begin{pmatrix} H_{cc} & H_{cp} & H_{cl} \\ H_{pc} & H_{pp} & H_{pl} \\ H_{lc} & H_{lp} & H_{ll} \end{pmatrix} \begin{pmatrix} x_c \\ x_p \\ x_l \end{pmatrix} = - \begin{pmatrix} b_c \\ b_p \\ b_l \end{pmatrix} \tag{2}$$

where $b = J^T r$ is the gradient of $\frac{1}{2}\|r\|^2$ w.r.t. x and $H = J^T J + \lambda D$ is referred to as the augmented Hessian. Camera, image and landmark components are denoted by c, p, and l respectively. Matrix blocks H_{cc}, H_{cp} and H_{cl} and associated transposes are unique to SI-BA.

3.2. Reduced Camera System

The Schur complement trick is used to generate the reduced camera system (RCS) $H_S \left(\begin{smallmatrix} x_c \\ x_p \end{smallmatrix}\right) = -b_S$ where:

$$H_S = \begin{pmatrix} H_{cc} & H_{cp} \\ H_{pc} & H_{pp} \end{pmatrix} - \begin{pmatrix} H_{cl} \\ H_{pl} \end{pmatrix} H_{ll}^{-1} \begin{pmatrix} H_{lc} & H_{lp} \end{pmatrix} \tag{3}$$

$$b_S = \begin{pmatrix} b_c \\ b_p \end{pmatrix} - \begin{pmatrix} H_{cl} \\ H_{pl} \end{pmatrix} H_{ll}^{-1}\, b_l \tag{4}$$

The RCS is smaller and better conditioned than Eq. (2) yet yields identical camera and image updates [1, 36]. Back-substitution gives the landmark update:

$$x_l = -H_{ll}^{-1}\left(b_l + H_{lc} x_c + H_{lp} x_p\right) \tag{5}$$

3.3. Preconditioned Conjugate Gradient Method

Solving the RCS for nontrivial problem sizes is done through the preconditioned conjugate gradient (PCG) method [36]. PCG is an inexact iterative method of solving positive semi-definite linear systems [22]. While PCG theory is complex, it is sufficient for the uninitiated reader to understand that the computationally demanding tasks of each PCG iteration consist of:

- Executing the matrix-vector product $H_S\, y$.
- Solving the preconditioning system $M^{-1} y$.

Explicitly constructing the RCS is intractable for large systems. Instead, the *implicit Schur*[1, 39] method may be used to compute matrix-vector products involving the RCS without having access to the matrix itself:

$$\begin{aligned} H_S \begin{pmatrix} y_c \\ y_p \end{pmatrix} &= \begin{pmatrix} H_{cc} \\ H_{pc} \end{pmatrix} y_c + \begin{pmatrix} H_{cp} \\ H_{pp} \end{pmatrix} y_p - \begin{pmatrix} H_{cl} \\ H_{pl} \end{pmatrix} y_l^* \\ y_l^* &= H_{ll}^{-1}\left(H_{lc}\, y_c + H_{lp}\, y_p\right) \end{aligned} \tag{6}$$

where y_c and y_p are components of an arbitrary vector relating to camera and image blocks respectively.

We utilize the preconditioner $M = \operatorname{diag}(H_{cc}, H_{pp})$ recommended by [39] for performing PCG on the RCS.

4. System Representation

Data structure selection is paramount to BA performance. This section details a simple system representation facilitating both canonical PCG on explicitly stored normal equations (Sec. 5) as well as our proposed matrix-free computation scheme (Sec. 6).

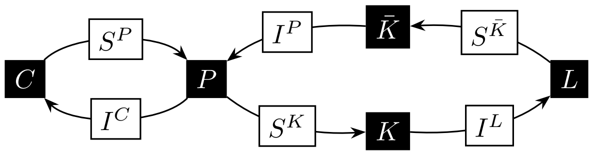

Figure 1. Linkage between state and connectivity of Tab. 1.

	Symbol	Shape	Description
State	C	$n_c \times n_i$	Camera intrinsics
	P	$n_p \times 7$	Image poses grouped by C
	L	$n_l \times 3$	Landmark positions
	K	$n_o \times 2$	Keypoints grouped by P
	$\bar{K}$	$n_o \times 2$	Keypoints grouped by L
Connectivity	S^P	n_c+1	Slices of P from C
	S^K	n_p+1	Slices of K from P
	$S^{\bar{K}}$	n_l+1	Slices of $\bar{K}$ from L
	I^C	n_p	Indices into C from P
	I^L	n_o	Indices into L from K
	I^P	n_o	Indices into P from $\bar{K}$

Table 1. Flat representation of all data required for BA. Cameras are assumed to use the same projection model. Poses have length 7 due to using quaternions for orientation.

State and associated connectivity is stored in flat buffers as outlined in Tab. 1 and illustrated in Fig. 1. Symmetry achieved through duplicated keypoint storage (K and $\bar{K}$) simplifies computations of Secs. 5 and 6. Memory allocation can be done up front knowing only the quantity of cameras n_c (each with n_i parameters), images n_p, landmarks n_l, and observations n_o.

Connectivity consists of slicing and indexing arrays. Readers familiar with the compressed sparse row (CSR)[28] matrix format may draw parallels to its row and column indexing arrays. To exemplify, assume one seeks the camera index k indicating where in C the intrinsics used by image i are located. The relevant camera index can be found at $k = I^C_i$.

Slicing arrays designate ranges instead of entries. As an example, for any camera index k all affiliated image indices are given by the interval $i \in \left[S^P_k, S^P_{k+1} - 1\right]$. Consequently, it follows that $S^P_0 = 0$ and $S^P_{n_c} = n_p$.

5. Canonical Computation

This section details the execution of all products needed for a single PCG iteration when using precomputed normal equations. Sparse matrix storage and matrix-vector product synthesis will be addressed. The connectivity representation described in Sec. 4 will be re-used for this task, thereby simplifying implementation and reducing memory consumption. See Tab. 2 for an overview of all matrices relevant to this section.

Matrices H_{cc}, H_{pp}, H_{ll} and their associated inverses are symmetric block-diagonal matrices. We store them contiguously block by block. Symmetry is exploited by only storing the upper triangular parts of H_{pp} and H_{ll}. Matrix-vector products by these matrices are trivially parallelized due to their block-diagonal structure.

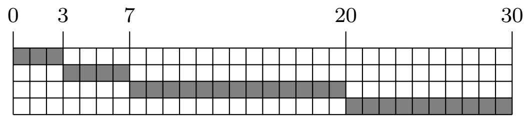

Figure 2. Sparsity of H_{cp} for $S^P = (0, 3, 7, 20, 30)$.

The matrix H_{cp} has the step-like sparsity structure depicted in Fig. 2. Blocks are stored contiguously and connectivity is fully defined by S^P alone. Matrix-vector products are realized through n_c dispatches of batch-reduce GEMM[11] kernels. Products by the transposed variant H_{pc} are done by strided-batch GEMM[8] kernels on the same underlying data.

H_{pl} is a block sparse row (BSR) matrix whose row and column indexing arrays are S^K and I^L respectively. Blocks are stored contiguously. Products are computed through a canonical multi-threaded sparse matrix-vector multiplication (SpMV) kernel as described in [9]. Handling of H_{lp} is identical except for using indexing matrices $S^{\bar{K}}$ and I^P instead.

Suitable representation of H_{cl} is challenging. Unlike all other matrices of Tab. 2, its sparsity cannot be determined purely from n_c, n_p, n_l and n_o. Sparsity varies by dataset but always lies within the range $[0, 1 - 1/n_c]$, meaning it is fully dense for single-camera datasets. We store it as a dense block matrix and accept the associated overhead on multi-camera datasets. Products are implemented using batch-reduce GEMM kernels as with H_{cp}. Products by H_{lc} are done through strided-batch GEMM invocations on the same data.

A BA implementation using the methodology just described was synthesized and profiled. Time spent on each matrix-vector product is provided in Tab. 2.

Matrix Product	Non-Zero Blocks	Block Size	Time (%)
H_{pl}	n_o	18	45.7
H_{lp}	n_o	18	44.9
H_{lc}	?	$3n_i$	2.3
H_{ll}^{-1}	n_l	6	1.6
H_{pp}	n_p	21	1.4
H_{pp}^{-1}	n_p	21	1.4
H_{cl}	?	$3n_i$	1.4
H_{cp}	n_p	$6n_i$	0.6
H_{pc}	n_p	$6n_i$	0.6
H_{cc}	n_c	${n_i}^2$	0.1
H_{cc}^{-1}	n_c	${n_i}^2$	0.1

Table 2. Relative time spent on each matrix-vector product over 50 PCG iterations on EuRoC of Sec. 8.1.

System	Time (ms)	
	NOP	SpMV
Laptop	2.24	2.36
Desktop	3.77	4.12
Server	5.03	5.45
Jetson	7.74	10.19

Table 3. Time spent on a product by H_{pl} (146 MB). A canonical kernel (SpMV) is compared against an altered copy of itself performing no arithmetic operations (NOP).

6. Matrix-Free Computation

Table 2 shows that products involving matrices H_{pl} and H_{lp} take up the majority of every PCG iteration. This is expected since both scale w.r.t. observation count and have large block sizes. This section focuses on accelerating these products via matrix-free methods.

High sparsity causes the products to be memory bound. Table 3 shows that omitting floating point operations entirely only marginally improves throughput. We therefore propose to bypass storing the matrices H_{pl} and H_{lp} by instead computing relevant matrix-vector products on the fly. As an example, rows of the product $H_{pl}\, y$ can be computed directly as follows:

$$(H_{pl}\, y)_i = \sum_{k=S_i^K}^{S_{i+1}^K - 1} h_{ijk}\, y_j \quad \text{where} \quad j = I_k^L \tag{7}$$

$$h_{ijk} = -w_k \begin{pmatrix} s_i\, I_{3\times3} \\ [p_j - t_i]_\times \end{pmatrix} J_{ij}^T\, J_{ij}\, R_i^T \tag{8}$$

where t_i, R_i and s_i designate position, orientation and numerical scale (see Sec. 7.2) of image i and p_j is the position of landmark j. The optional IRLS weight attached to observation k is w_k. The 2×3 matrix J_{ij} is the projection Jacobian w.r.t. landmark j and depends on the camera parameters associated with image i. Note that Eq. (8) assumes pose perturbations are applied in the local frame of each image. Explicit construction of the 6×3 matrix h_{ijk} is not necessary as relevant products can be computed on the fly.

It may seem counterintuitive that on-the-fly computation can outperform precomputed sparse matrices. Even ignoring all arithmetic, replicating a single block product requires fetching camera intrinsics, image pose and scale, IRLS weight and a landmark position.

Key to understanding this is realizing that camera intrinsics and image pose are constant for each row of H_{pl} and therefore rarely need to be fetched during computation. Additionally, while landmarks do vary per block, the relevant landmark position is often present in cache. This will be elaborated in Sec. 6.2.

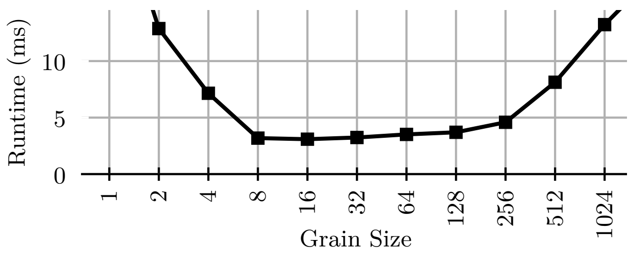

Figure 3. H_{lp} execution time by grain size on EuRoC.

6.1. Parallelization Strategy

Arithmetic pressure is substantially increased by on-the-fly computation schemes. Initial experiments using a naive multi-threaded scalar implementation showed worse performance than using precomputed matrices. Fully realizing the FLOPS potential of modern processors requires synthesis using SIMD instructions.

We therefore implement relevant kernels using ISPC[24] and CUDA[21]. We describe the CUDA implementation since more readers are familiar with the terminology. The ISPC version is essentially identical.

When computing products by H_{pl} each row of the output (relating to one image) is delegated to a thread block performing a grid stride loop over pertinent observations. Threads handle one observation at a time. Block reduction is performed on loop conclusion.

The above strategy is unsuited to H_{lp} due to landmarks generally having fewer observations than the warp size. Each thread is instead assigned to an entire landmark and loops over all associated observations. This implies that adjacently stored landmarks with differing observation counts induce performance-degrading warp divergence (see [21]). Partially mitigating data ordering strategies are introduced in Sec. 6.2.

As ISPC kernels are single-threaded, we realize multi-threading through parallelized for-loops. Grain size selection was guided by graphs akin to Fig. 3.

6.2. Ordering Implications

The order in which images and landmarks are laid out in memory has significant performance implications.

The kernels of Sec. 6.1 access the same data, e.g. landmark positions, numerous times. When sequentially stored images observe overlapping landmark sets, many of these accesses will result in performance-boosting cache hits. Images and landmarks should ideally be sorted by covisibility to maximize this effect.

Cache utilization is not the only factor that must be considered when ordering our data. Recall that our parallelization strategy for H_{lp} suffers from warp divergence when adjacent landmarks have different observation counts. Reducing this to a minimum is paramount.

Ordering	On-the-Fly H_{pl}	On-the-Fly H_{lp}	Precomputed H_{pl}	Precomputed H_{lp}
Random	1.39	8.57	2.62	2.66
Idx	1.20	6.65	2.60	2.66
Count	1.28	1.87	2.61	2.68
Count & Idx	1.19	1.88	2.59	2.66

Table 4. Computation times in milliseconds for different landmark orderings. Precomputing yields ordering invariance. Idx sorts by the minimum observing image ID and Count by observation count. Count & Idx sorts by observation count and breaks ties by minimum observing image ID. Random is self-explanatory. Dataset is EuRoC of Sec. 8.1.

Time spent on sorting itself must also be taken into account. Increased preprocessing times associated with complex ordering schemes may nullify their benefits.

We present a heuristic compromise addressing all three concerns. First, group images by camera and sort images within each group by their image ID in the underlying SfM model. This is a reasonable proxy for sorting by covisibility when the model stems from incremental SfM [40]. Even global SfM exhibits natural covisibility grouping by virtue of starting from pairwise reconstructions [23]. Secondly, sort landmarks by observation count, breaking ties by the lowest observing image ID. This ordering is easily realized and improves throughput substantially as seen in Tab. 4.

7. Single-Precision Synthesis

Using single-precision floating-point arithmetic is desirable from a performance perspective. Memory consumption is halved while arithmetic throughput and effective cache size are doubled. Implementations targeting consumer-grade GPUs seldom have a choice as such systems typically have limited double-precision support. This section presents solutions to issues encountered during our foray into single-precision BA.

7.1. Avoiding Catastrophic Cancellation

The absolute precision of IEEE-754 floats halves with every doubling of magnitude [12]. We encountered *catastrophic cancellation* when subtracting image and landmark positions (the first step of projection) far from the origin. Shifting the coordinate system origin to the midrange of all image positions remedied this.

Near-zero depth landmark observations also induce catastrophic cancellation. As preprocessing we project all landmarks in both single- and double-precision, then disconnect observations where the difference exceeds 0.005 pixels. Inspection revealed such observations were, almost without exception, erroneous anyway.

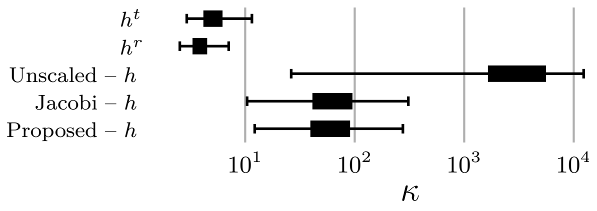

Figure 4. Condition numbers of full (h) blocks of H_{pp} compared to translation (h^t) and rotation (h^r) sub-blocks in isolation. Our variable scaling scheme improves conditioning as effectively as Jacobi scaling. Dataset is EuRoC.

7.2. Balancing Translation and Rotation

Blocks of H_{pp} are ill-conditioned when no variable scaling is performed. This causes H_{pp}^{-1} (used for preconditioning) to be inaccurately computed, which in turn leads to numerical failure. This issue is usually mitigated through Jacobi scaling [1, 7, 39], where each optimization variable is assigned a distinct numerical scaling factor derived from the augmented Hessian.

Jacobi scaling is unfortunately detrimental to matrix-free methods. Every fetch of an image pose from memory described in Sec. 6 would need to include all six associated scaling factors, thereby reducing effective cache size. A compact alternative to the general-purpose Jacobi scaling strategy is therefore necessary.

Note initially that every 6×6 block of H_{pp} can be further subdivided into four 3×3 blocks:

$$h_i = \begin{pmatrix} h_i^t & h_i^{tr} \\ h_i^{rt} & h_i^r \end{pmatrix} \tag{9}$$

where t and r signify translation and rotation components of the block h_i associated with image i.

Figure 4 shows that sub-blocks h^t and h^r are well-conditioned when considered in isolation, indicating that numerical scale differences between translation and rotation components cause blocks of H_{pp} to be ill-conditioned. Uniformly applied global scaling does not address this, as the imbalance varies by image.

Motivated by these observations, we introduce a single numerical scaling factor s_i unique to each image i:

$$s_i = \sqrt{\frac{\operatorname{tr}(h_i^r)}{\operatorname{tr}(h_i^t)}} \tag{10}$$

Jacobian entries related to translation are scaled equally by this factor. No variable scaling of rotation is performed. Inspecting Fig. 4 reveals that the proposed variable scaling scheme improves conditioning as effectively as Jacobi scaling despite its simplicity.

Scaling factors are stored together with image poses to aid alignment and improve caching.

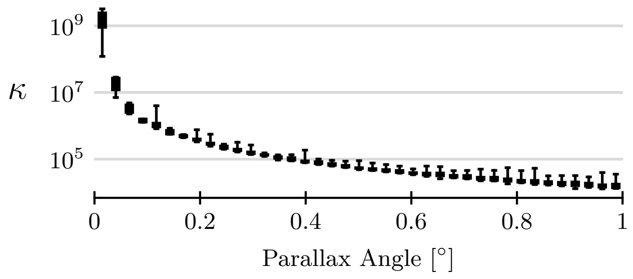

Figure 5. Condition numbers of H_{ll} blocks as a function of associated landmark parallax angle. Dataset is EuRoC.

7.3. Handling Low-Parallax Landmarks

The Schur complement requires computing the inverse of the block diagonal matrix H_{ll}. Blocks corresponding to low-parallax landmarks are ill-conditioned (Fig. 5) and cannot be reliably inverted under single-precision. While such landmarks are customarily removed during preprocessing, we observed instances of numerical failure stemming from parallax decreasing drastically during the BA process itself. Low-parallax landmarks must therefore be handled properly inside the BA loop.

Utilizing the pseudo-inverse H_{ll}^{+} resolves the issue by preventing perturbation in the degenerate depth direction of troublesome landmarks, but is unfortunately too computationally costly to apply to all blocks of H_{ll}.

We propose a compromise inspired by the pseudo-inverse implementation of the popular linear algebra library Numpy[13], which from the SVD $A = U\Sigma V^T$ computes the pseudo-inverse as $A^{+} = V\Sigma^{+}U^T$ where:

$$\Sigma_i^{+} = \begin{cases} 1/\Sigma_i & \text{if} \quad \Sigma_i > \tau \max \Sigma \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

The constant τ is determined from machine precision.

Equation (11) implies that the pseudo-inverse is only necessary for blocks where $\kappa(A) > \frac{1}{\tau}$. Frustratingly, exact condition number calculation is a computationally heavy task as well. In order to circumvent this, we employ the easily obtained upper bound for symmetric positive-definite matrices discovered by [18] instead:

$$\kappa(A) \leq \frac{1 + \sqrt{1 - (n \,/\, \operatorname{tr} A)^n \det A}}{1 - \sqrt{1 - (n \,/\, \operatorname{tr} A)^n \det A}} \tag{12}$$

where $n = 3$ is the matrix size. Although this substitution causes the pseudo-inverse to be unnecessarily applied to a few well-conditioned landmark blocks, the overhead associated with this is dwarfed by savings from avoiding exact condition number computation.

Note that the presented approach does not assume shared intrinsics, meaning it may be readily integrated into any existing single-precision BA pipeline.

8. Experiments

We evaluate the impact of ideas presented within this paper by synthesizing and comparing three matrix-free solvers against the industry standard (and virtually only option) for shared intrinsics BA: Ceres[2].

Our CPU solvers CPU-f64 and CPU-f32 compute in double- and single-precision respectively. Applicability to GPUs is demonstrated through GPU-f32 implemented in CUDA. All solvers have their Levenberg-Marquardt loops meticulously modeled after Ceres'.

8.1. Datasets

As no equivalent to BAL[1] for the SI-BA case exists to our knowledge, we generate and share COLMAP[33] models from datasets known by the wider community:

- `Tanks & Temples (Courthouse)`[16]:
  Used for benchmarking 3D reconstruction and view synthesis – most recently for Gaussian Splatting[15].
- `EuRoC (V1-01)`[3]:
  A mainstay of the visual SLAM community which figures in well-known works such as [4] and [26].
- `Aachen (Day-Night)`[31]:
  Prominent in visual localization research [32][29]. We use the SfM model released by the authors composed of three overlapping incremental sequences (and a photo collection component which we omit).
- `KITTI (00)`[10]:
  A classic of the autonomous vehicle community [5]. All images within stem from car-mounted cameras.
- `LaMAR (LIN)`[30]:
  Collection focused on large-scale AR. We were unable to construct a non-degenerate SfM model of the entire LIN dataset and settled for a subset of the mapping sequence reliably reconstructed.

See Tab. 5 for summary statistics. All SfM models use the COLMAP RADIAL camera model, except Aachen which uses SIMPLE_RADIAL and LaMAR which uses PINHOLE. Gaussian noise is added to initial image and landmark positions to increase BA difficulty.

	Cameras	Images	Landmarks	Observations
T&T	1	1106	223 290	1 294 841
Aachen	3	2369	889 342	5 022 411
EuRoC	1	2812	68 163	1 907 064
KITTI	1	4541	526 740	2 826 328
LaMAR	20	22 144	2 211 368	9 252 244

Table 5. Summary statistics of each dataset.

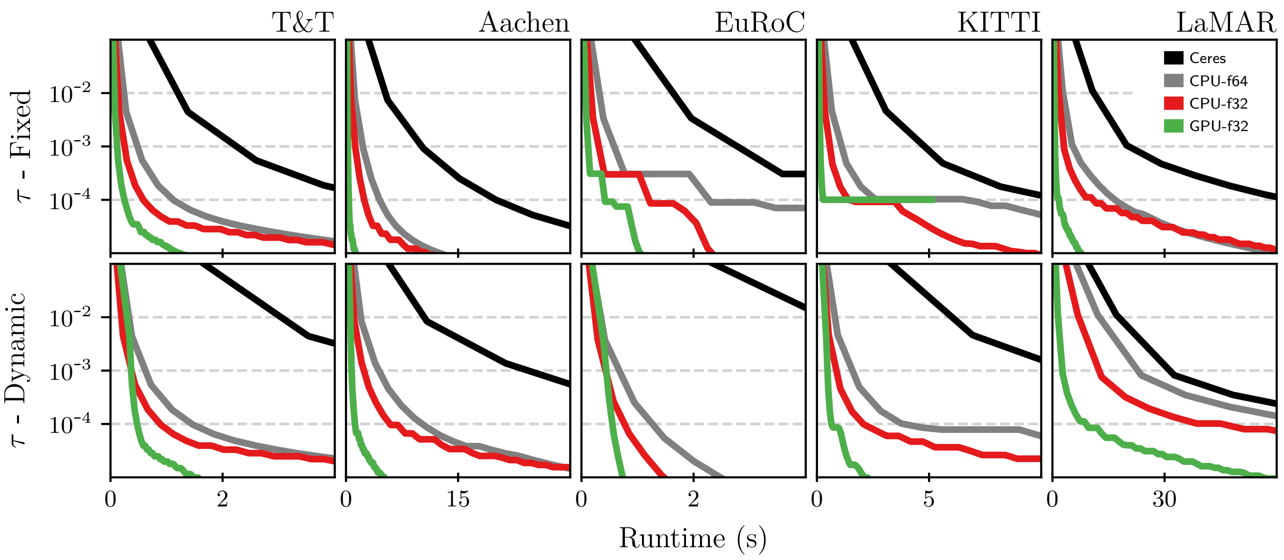

Figure 6. Achieved tolerance as a function of runtime. Tolerances tighter than $\tau = 10^{-4}$ are not achievable by GPU-f32 on KITTI due to our rather rudimentary implementation lacking fundamental floating point error accumulation mitigations.

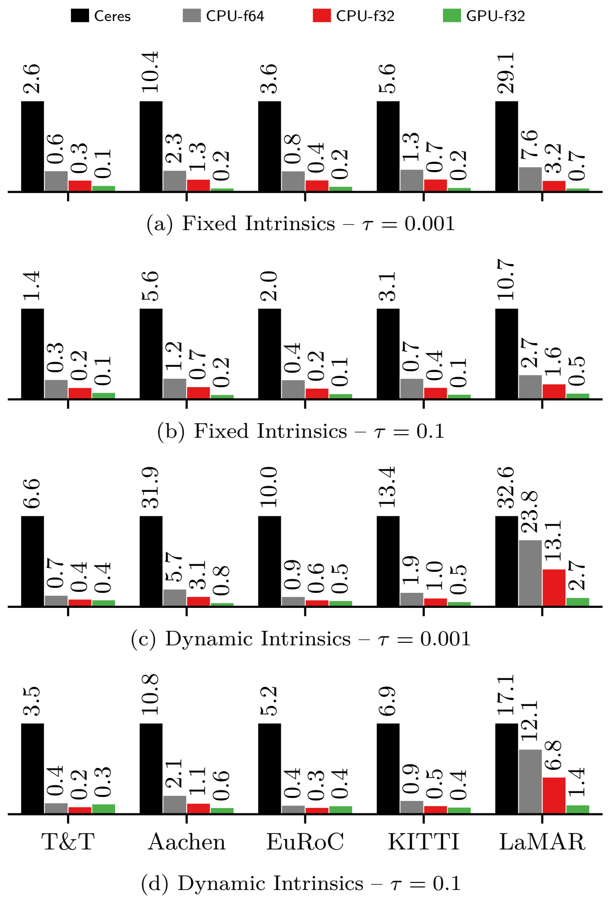

Figure 7. Time in seconds to reach τ.

8.2. Settings and Setup

Both dynamic (intrinsics vary freely) and fixed (intrinsics are kept constant) configurations are evaluated.

Levenberg-Marquardt iterations are capped at 100. PCG iterations are fixed at 50 to minimize pseudo-random fluctuations from forcing sequence thresholds. A 1 pixel Huber loss is applied. Block Jacobi preconditioning is used as recommended by [39].

The implicit Schur method was used for Ceres (explicit RCS generation worsened performance). Preprocessing time is removed from Ceres timings to make comparison fair, as it includes error checking and bookkeeping which our specialized solvers don't require. Timings of our solvers do include all preprocessing.

A Windows 10 laptop with 32GB of RAM, a 14 core Intel i7-12800H processor and an NVIDIA RTX 3080 (mobile) GPU constitutes the experimental test bench.

8.3. Results

8.3.1. Runtime

Following precedence[7, 17, 38] we measure the time to reach a target cost $E_\tau = E^* + \tau(E^0 - E^*)$ where E^0 is the initial cost before BA, E^* the lowest cost achieved by any solver (a substitute for the theoretical minimum cost) and E_τ the cost associated with the tolerance τ.

Figure 6 shows that our solvers provide significant runtime reductions at any tolerance. Detailed results for $\tau = 0.1$ and $\tau = 0.001$ (see Fig. 7) reveal speedups of roughly 4x (CPU-f64), 8x (CPU-f32) and 20x (GPU-f32) over Ceres. Reduced performance on LaMAR when not keeping intrinsics constant will be discussed in Sec. 8.4.

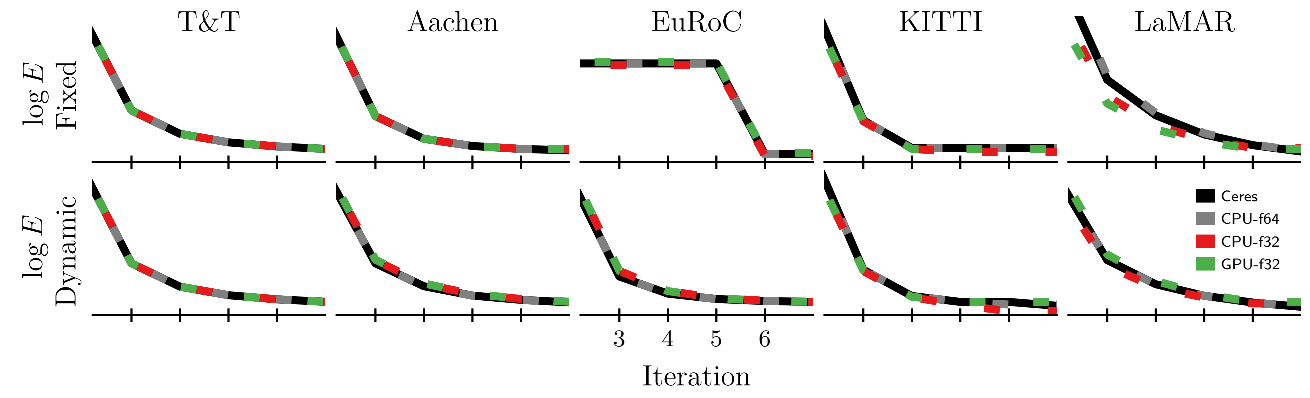

Figure 8. Convergence curves for fixed and dynamic intrinsics.

		T&T	Aachen	EuRoC	KITTI	LaMAR
Dyn	Ceres	1102	4333	1303	2548	8715
	CPU-f64	125	571	100	300	3022
	CPU-f32	81	358	74	192	1652
Fix	Ceres	1029	4118	1192	2381	8325
	CPU-f64	110	446	95	263	983
	CPU-f32	73	296	71	173	632

Table 6. Peak memory consumption in megabytes.

8.3.2. Memory Consumption

Peak memory consumption is given in Tab. 6. Proposed data structures and matrix-free methods lead to an order of magnitude reduction in memory consumption compared to Ceres. Note however that this is not the case when estimating intrinsics on LaMAR.

8.3.3. Convergence Characteristics

Cost reduction per iteration is seen in Fig. 8 to be virtually indistinguishable between CPU-f64 and Ceres. Single-precision solvers CPU-f32 and GPU-f32 deviate slightly yet exhibit rather remarkable robustness. Runtime improvements noted in Sec. 8.3.1 stem therefore not from superior convergence characteristics, but instead from efforts on accelerating core computations.

8.4. Discussion

Runtime and memory consumption of presented BA solvers suffers when refining intrinsics on LaMAR. Choosing to represent H_{cl} as a dense matrix is the root cause, as high camera counts lead to increased sparsity of H_{cl} (see Sec. 5). Extending the matrix-free computation scheme of Sec. 6 to include H_{cl} could resolve this issue, albeit at the cost of implementation complexity. We leave this challenge as potential future work.

Another matter warranting commentary is the surprising 4x speedup observed even when computing under double-precision, since extrapolation from Tables 2 and 4 suggests a twofold increase in throughput at best.

The remainder is addressed by noting that Ceres, in line with [39], executes products with the augmented Hessian H through the Jacobian J and its transpose. While elegant and allowing for easy integration of additional terms like pose priors, this slows down PCG iterations considerably as J scales with observation count in a manner similar to H. The authors of [39] chose this strategy to circumvent the considerable cost and complexity associated with generating H_{pl} and H_{lp}.

As our matrix-free approach never explicitly creates these matrices, it reaps the benefits of using the augmented Hessian without paying the customary matrix construction overhead. Indeed, this is the true benefit of our proposed method. The fact that matrix-free products are faster than their precomputed variants, while significant in its own right, is an ancillary result.

9. Conclusion

Algorithms and data structures tailored to SI-BA were presented. Experiments demonstrated remarkable reductions in runtime and memory consumption. Crucially, it was shown that SI-BA based on explicitly generated normal equations is memory bound and that our proposed matrix-free approach boosts performance beyond limits set by the memory subsystem. Applicability to GPUs was proven through synthesizing a solver outperforming CPU variants. Additionally, analysis of single-precision instability yielded two novelties: a compact alternative to Jacobi scaling and a performant method of handling low-parallax landmarks.

While BA on photo collections has been exhaustively studied, this exploratory paper has unearthed optimizations intrinsic to SI-BA. We hope to inspire further work on this issue of great practical concern.

References

[1] 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. 1, 2, 5, 6

[2] Sameer Agarwal, Keir Mierle, et al. Ceres solver: Tutorial & reference. *Google Inc*, 2(72):8, 2012. 1, 6

[3] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016. 6

[4] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021. 6

[5] Long Chen, Yuchen Li, Chao Huang, Bai Li, Yang Xing, Daxin Tian, Li Li, Zhongxu Hu, Xiaoxiang Na, Zixuan Li, et al. Milestones in autonomous driving and intelligent vehicles: Survey of surveys. *IEEE Transactions on Intelligent Vehicles*, 8(2):1046–1056, 2022. 6

[6] Shrutimoy Das, Siddhant Katyan, and Pawan Kumar. A deflation based fast and robust preconditioner for bundle adjustment. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1782–1789, 2021. 2

[7] Nikolaus Demmel, Christiane Sommer, Daniel Cremers, and Vladyslav Usenko. Square root bundle adjustment for large-scale reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11723–11732, 2021. 1, 2, 5, 7

[8] Jack Dongarra, Sven Hammarling, Nicholas J Higham, Samuel D Relton, Pedro Valero-Lara, and Mawussi Zounon. The design and performance of batched blas on modern high-performance computing systems. *Procedia Computer Science*, 108:495–504, 2017. 3

[9] Ryan Eberhardt and Mark Hoemmen. Optimization of block sparse matrix-vector multiplication on shared-memory parallel architectures. In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 663–672. IEEE, 2016. 3

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. 6

[11] Evangelos Georganas, Kunal Banerjee, Dhiraj Kalamkar, Sasikanth Avancha, Anand Venkat, Michael Anderson, Greg Henry, Hans Pabst, and Alexander Heinecke. Harnessing deep learning via a single building block. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 222–233. IEEE, 2020. 3

[12] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM computing surveys (CSUR)*, 23(1):5–48, 1991. 5

[13] Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020. 6

[14] Siddhant Katyan, Shrutimoy Das, and Pawan Kumar. Two-grid preconditioned solver for bundle adjustment. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3599–3606, 2020. 2

[15] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 1, 6

[16] Arno Knapitsch, Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Tanks and temples: Benchmarking large-scale scene reconstruction. *ACM Transactions on Graphics (ToG)*, 36(4):1–13, 2017. 6

[17] 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. 2, 7

[18] Jorma Kaarlo Merikoski, Uoti Urpala, Ari Virtanen, Tin-Yau Tam, and Frank Uhlig. A best upper bound for the 2-norm condition number of a matrix. *Linear algebra and its applications*, 254(1-3):355–365, 1997. 6

[19] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1): 99–106, 2021. 1

[20] Pierre Moulon, Pascal Monasse, Romuald Perrot, and Renaud Marlet. OpenMVG: Open multiple view geometry. In *International Workshop on Reproducible Research in Pattern Recognition*, pages 60–74. Springer, 2016. 1

[21] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2):40–53, 2008. 4

[22] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999. 2

[23] Linfei Pan, Daniel Barath, Marc Pollefeys, and Johannes Lutz Schönberger. Global Structure-from-Motion Revisited. In *European Conference on Computer Vision (ECCV)*, 2024. 1, 5

[24] Matt Pharr and William R Mark. ispc: A spmd compiler for high-performance cpu programming. In *2012 Innovative Parallel Computing (InPar)*, pages 1–13. IEEE, 2012. 4

[25] Marc Pierrot Deseilligny and Isabelle Cléry. Apero, an open source bundle adjusment software for automatic calibration and orientation of set of images. *The International Archives of the Photogrammetry, Remote*

Sensing and Spatial Information Sciences, 38:269–276, 2012. 2

[26] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE transactions on robotics*, 34(4):1004–1020, 2018. 6

[27] 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 *European Conference on Computer Vision*, pages 715–731. Springer, 2022. 1, 2

[28] Yousef Saad. *Iterative methods for sparse linear systems*. SIAM, 2003. 3

[29] Paul-Edouard Sarlin, Cesar Cadena, Roland Siegwart, and Marcin Dymczyk. From coarse to fine: Robust hierarchical localization at large scale. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12716–12725, 2019. 6

[30] Paul-Edouard Sarlin, Mihai Dusmanu, Johannes L Schönberger, Pablo Speciale, Lukas Gruber, Viktor Larsson, Ondrej Miksik, and Marc Pollefeys. Lamar: Benchmarking localization and mapping for augmented reality. In *European Conference on Computer Vision*, pages 686–704. Springer, 2022. 6

[31] Torsten Sattler, Tobias Weyand, Bastian Leibe, and Leif Kobbelt. Image retrieval for image-based localization revisited. In *BMVC*, page 4, 2012. 6

[32] Torsten Sattler, Will Maddern, Carl Toft, Akihiko Torii, Lars Hammarstrand, Erik Stenborg, Daniel Safari, Masatoshi Okutomi, Marc Pollefeys, Josef Sivic, et al. Benchmarking 6dof outdoor visual localization in changing conditions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8601–8610, 2018. 6

[33] 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. 1, 6

[34] Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, pages 501–518. Springer, 2016. 1

[35] Christopher Sweeney, Tobias Hollerer, and Matthew Turk. Theia: A fast and scalable structure-from-motion library. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 693–696, 2015. 1

[36] 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, 2

[37] Shimon Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153):405–426, 1979. 1

[38] Simon Weber, Nikolaus Demmel, Tin Chon Chan, and Daniel Cremers. Power bundle adjustment for large-scale 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 281–289, 2023. 1, 2, 7

[39] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M Seitz. Multicore bundle adjustment. In *CVPR 2011*, pages 3057–3064. IEEE, 2011. 1, 2, 5, 7, 8

[40] Zhichao Ye, Guanglin Li, Haomin Liu, Zhaopeng Cui, Hujun Bao, and Guofeng Zhang. Coli-ba: Compact linearization based solver for bundle adjustment. *IEEE Transactions on Visualization and Computer Graphics*, 28(11):3727–3736, 2022. 1, 5

[41] 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

[42] 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 *European Conference on Computer Vision*, pages 364–379. Springer, 2020. 2