Bundle Adjustment on a Graph Processor

Joseph Ortiz[1], Mark Pupilli[2], Stefan Leutenegger[1], Andrew J. Davison[1]
[1]Imperial College London, Department of Computing, UK. [2]Graphcore.
j.ortiz@imperial.ac.uk

Abstract

Graph processors such as Graphcore's Intelligence Processing Unit (IPU) are part of the major new wave of novel computer architecture for AI, and have a general design with massively parallel computation, distributed on-chip memory and very high inter-core communication bandwidth which allows breakthrough performance for message passing algorithms on arbitrary graphs.

We show for the first time that the classical computer vision problem of bundle adjustment (BA) can be solved extremely fast on a graph processor using Gaussian Belief Propagation. Our simple but fully parallel implementation uses the 1216 cores on a single IPU chip to, for instance, solve a real BA problem with 125 keyframes and 1919 points in under 40ms, compared to 1450ms for the Ceres CPU library. Further code optimisation will surely increase this difference on static problems, but we argue that the real promise of graph processing is for flexible in-place optimisation of general, dynamically changing factor graphs representing Spatial AI problems. We give indications of this with experiments showing the ability of GBP to efficiently solve incremental SLAM problems, and deal with robust cost functions and different types of factors.

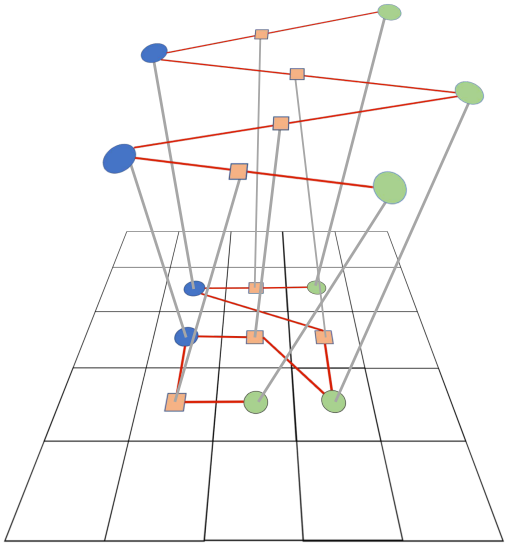


Figure 1: We map a bundle adjustment factor graph onto the tiles (cores) of Graphcore's IPU and show that Gaussian Belief Propagation can be used for rapid, distributed, in-place inference for large problems. Here we display the most simple mapping in which each node in the factor graph is mapped onto a single arbitrary tile. Keyframe nodes are blue, landmark nodes are green and measurement factor nodes are orange.

1. Introduction

Real-world applications which require a general real-time 'Spatial AI' capability from computer vision are becoming more prevalent in areas such as robotics, UAVs and AR headsets, but it is clear that a large gap still exists between the ideal performance required and what can be delivered within the constraints of real embodied products, such as low power usage. An increasingly important direction is the design of processor and sensor hardware specifically for vision and AI workloads to replace the general purpose CPUs, GPUs and frame-based video cameras which are currently prevalent [8, 24]. The space of AI and vision algorithm design continues to change rapidly and we believe that it is not the right time to make very specific decisions such as 'baking in' a particular SLAM algorithm to processor hardware, except perhaps for very specific use cases.

However, new architectures are emerging which have made quite general design choices about processing for AI workloads. Efficient and low power computation must be massively parallel and minimise data transfer. To this end, storage and processing should be distributed, and as much computation as possible should happen 'in place'. A key example is Graphcore's Intelligence Processing Unit (IPU) [1], which implements this concept within a single large chip which is composed of 1216 cores called *tiles*, each with local memory arranged in a fully connected graph structure. It is massively parallel like a GPU, but its tiles have a completely different interconnect structure. The IPU has break-

through performance for algorithms which have a sparse graph message passing character. The key early commercial use case for the IPU is as a flexible deep learning accelerator [19], primarily in the cloud, but we believe that it has much more general potential for Spatial AI computation.

In this paper we consider bundle adjustment (BA), a central element of 3D visual processing which is representative of many geometric estimation problems, and show that Gaussian Belief Propagation (GBP) can perform rapid optimisation of BA problems on a single IPU chip.

GBP is a special case of general loopy belief propagation, a well known technique in probabilistic estimation, but it has previously only been minimally used in geometric vision and robotics problems [9]. It is an algorithm which can be run on a CPU, but is not necessarily competitive there compared to alternative optimisation techniques which take global account of the structure of a problem. However, GBP can be mapped to a graph processor due to its fully distributed nature to take full advantage of the massively parallel capability of an IPU.

We present the first implementation of BA on a graph processor, with breakthrough optimisation speed for a variety of diverse sequences in which we record an average speed advantage 24x over the Ceres library on a CPU. Our implementation is simple and preliminary, implemented with only 1000 lines of Poplar$^{\text{TM}}$C++ code, and there is surely much room for future performance optimisation.

Positive characteristics of our GBP approach include: extremely fast local convergence, the ability to use robust cost functions to reject outlying measurements, and the ability to easily deal with dynamic addition of variables and data and rapidly re-optimise solutions. We highlight these aspects in our results, and argue as in [9] for the huge potential for graph processing and GBP in general incremental factor graph optimisation for Spatial AI. It would be straightforward and efficient to incorporate factors from additional priors and sensors into this framework, such as smoothness of scene regions due to recognition, and continue to optimise for global estimates with all computation and storage done in-place on a graph processor.

2. Related Work

Factor graphs are commonly used in geometric vision to represent the structure of constraints in estimation problems [6, 11, 12, 18, 20, 22]. In particular, for bundle adjustment [31] researchers have leveraged the global structure of these constraints to design efficient inference algorithms [4, 15].

Several works have taken the approach of converting the loopy factor graph into a tree [17, 25]. iSAM2 [17] uses variable elimination to convert the loopy factor graph to a Bayes tree while [25] uses a junction tree-like method which employs maximum likelihood projections to remove edges. This category of methods differs from our approach in that it requires periodic centralised computation to convert the loopy constraint graph into a tree.

More closely related to our work, [7] and [27] use Loopy Belief Propagation for geometric estimation problems, though with CPU implementation. [7] uses discrete BP to provide an initialisation for Levenberg-Marquardt refinement in BA, and Loopy SAM [27] uses GBP to solve a SLAM-like problem for a relatively small 2D scene.

In the domain of computer architecture, there has been substantial recent effort to design specific hardware for vision algorithms [29, 34]. This is particularly evident in industry, where we have seen development of chips such as the HoloLens' HPU and the Movidius VPU series, though the main accelerations achieved to date have been in vision front-ends such as feature matching.

Other related research has made use of parallelism on existing hardware to accelerate BA. Multicore BA [33] proposed an inexact but parallelisable implementation for CPUs or GPUs, while [14] advocated a hybrid GPU and CPU implementation. More generally, [10] accelerated non-linear least squares problems in graphics by automatically generating GPU solvers.

3. Preliminaries

3.1. Factor Graphs

Factor graphs are well known in geometric vision as a representation of the structure of estimation problems. A factor graph, $G = (V, F, E)$, is a bipartite graph composed of a set of variable nodes $V = \{\mathbf{v}_i\}_{i=1:N_v}$, a set of factor nodes $F = \{f_s\}_{s=1:N_f}$ and a set of edges E. Each factor node f_s represents a probabilistic constraint between a subset of variables $V_s \subset V$ which is described by an arbitrary function $f_s(V_s)$. The factorisation is explicitly represented in the graph by connecting factor nodes with the variable nodes they depend on. Probabilistically speaking, these factors are the independent terms that make up the joint distribution:

$$p(V) = \prod_{s=1}^{N_f} f_s(V_s) \, . \tag{1}$$

3.2. Belief Propagation

Belief propagation (BP) [26] is a well-known distributed inference algorithm for computing the marginal distribution for a set of variables from their joint distribution. The marginal for a single variable $\mathbf{v}_i$ is the integral of the joint distribution over all other variables:

$$p(\mathbf{v}_i) = \int p(V) \; d\mathbf{v}_1 \ldots d\mathbf{v}_{i-1} d\mathbf{v}_{i+1} \ldots d\mathbf{v}_{N_v} \, . \tag{2}$$

BP works by passing messages through the factor graph and is efficient as it leverages the fact that the topology

of the graph encodes the factorisation of the joint distribution. The marginals are computed using iterative local message passing which alternates between factor nodes sending messages to variable nodes and variable nodes sending messages to factor nodes. See [5] or [9] for a derivation of the message passing rules.

By design, belief propagation infers the marginals for tree graphs in one sweep of messages from the root node to the leaf nodes and then back up. For loopy graphs, the same BP message passing can be applied with a message passing schedule, and after many iterations estimates converge to the marginals. Loopy BP does not have convergence guarantees, however it is generally stable [23]. When the distributions are represented as Gaussians, Loopy Gaussian Belief Propagation converges to the correct marginal posterior means for all graph topologies [32].

Key to understanding why belief propagation is efficient is considering the least efficient way to compute the marginal distribution for a variable. The naive way would be to take a product of all of the factors to give the joint distribution and then marginalise over all other variables. This simultaneous marginalisation over all other variables is expensive; for example, in the discrete case, if each variable takes k discrete values then marginalising over all but one variable requires summing k^{N_v-1} terms. Belief propagation instead marginalises over minimal independent subsets of variables using the conditional dependency information which is encoded in the graph topology. Returning to the example of discrete variables, if we want to compute the marginal distribution for a tree graph containing only pairwise factors, belief propagation requires summing only $2N_f k^2$ terms.

4. The Bundle Adjustment Factor Graph

Bundle adjustment is the problem of jointly refining the set of variables $V = X \cup L$, where $X = \{\mathbf{x}_i\}_{i=1:N_k}$ is the set of keyframe poses and $L = \{\mathbf{l}_j\}_{j=1:N_l}$ is the set of landmark locations, subject to a set of constraints which define the error we want to minimise. Specifically, we include two types of error terms: reprojection errors and prior errors. The reprojection error penalises the distances between the projections of landmarks into the image plane of the keyframes that observe them and the set of measurements corresponding to these observations $Z = \{\mathbf{z}_{km}\}$. The prior error terms try to maximise the probability that the current variable values were drawn from the corresponding prior distribution $\{\mathcal{N}(\mathbf{x}_i; \mathbf{x}_{p_i}, \Sigma_{p,xi}),\ \mathcal{N}(\mathbf{l}_j; \mathbf{l}_{p_j}, \Sigma_{p,lj})\}_{i=1:N_k, j=1:N_l}$. The prior terms are required to set the overall scale for monocular problems and to condition the messages from the measurement factors which would otherwise only constrain 2 degrees of freedom. Given an initialisation point, the priors are automatically generated such that they are a factor of 100 weaker than the reprojection error terms in the objective. We formulate this using the Jacobians and the measurement model which define the strength of measurement constraints. An example factor graph for a small BA problem is shown in Figure 2.

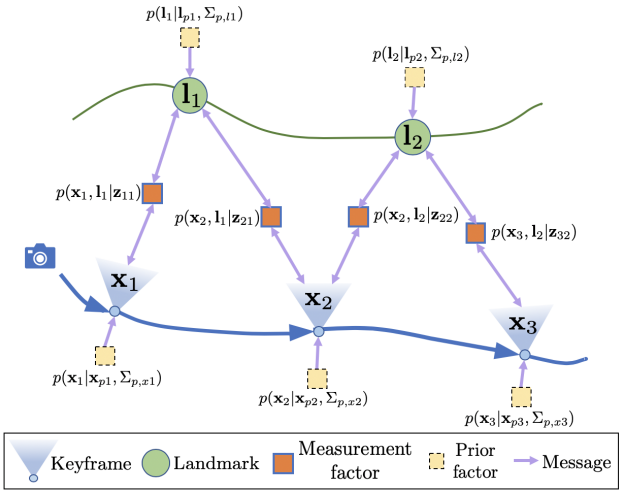


Figure 2: **Factor graph illustration.** Measurement factors connect keyframes and the landmarks they observe. Keyframes and landmarks are instantiated with an automatically generated weak prior factor. Messages are sent from all factors to adjacent keyframe and landmark nodes and from keyframe and landmark nodes to adjacent measurement factor nodes.

In bundle adjustment we want to perform maximum a posteriori (MAP) inference which computes the configuration of variables $\{X, L\}$ that maximises the joint probability $p(X, L|Z)$:

$$\{X^*, L^*\} = \arg\max_{\{X,L\}} p(X, L|Z) \quad (3)$$

$$= \arg\max_{\{X,L\}} p(Z|X, L)p(X, L)\ . \quad (4)$$

In the second line we have used Bayes theorem and dropped the denominator $p(Z)$ as measurements are given quantities and do not affect the MAP solution. This leads to the factorisation of the probability distribution that we want to maximise (which we will call $p_{\text{obj}}(X, L)$) into the product of the likelihood of the measurements given the variables $p(Z|X, L)$ and priors on the variables $p(X, L)$. As $\mathbf{x}_i$ and $\mathbf{x}_j$ are independent in our formulation, $\mathbf{l}_i$ and $\mathbf{l}_j$ are independent and $\mathbf{x}_i$ and $\mathbf{l}_j$ are only conditionally dependent given a measurement $\mathbf{z}_{ij}$, these terms can be further factorised:

$$p_{\text{obj}}(X, L) = \prod_{i=1}^{N_k} \phi_i(\mathbf{x}_i) \prod_{j=1}^{N_l} \theta_j(\mathbf{l}_j) \prod_{k=1}^{N_k} \prod_{m, \mathbf{l}_m \in L_k} \psi_{km}(\mathbf{x}_k, \mathbf{l}_m)\ , \quad (5)$$

where L_k is the set of landmarks observed by keyframe $\mathbf{x}_k$.

The set of factors $\{\phi_i, \theta_j, \psi_{km}\}_{i=1:N_k, j=1:N_l, km \in O}$ can be interpreted as prior constraints on the keyframe poses, prior constraints on the landmark positions and measurement reprojection constraints respectively. The prior constraints have the form of Gaussians over the variables $\{\mathbf{x}_i\}_{i=1:N_k}$ and $\{\mathbf{l}_j\}_{j=1:N_l}$:

$$\begin{aligned} \phi_i(\mathbf{x}_i) &= p(\mathbf{x}_i|\mathbf{x}_{p_i}, \Sigma_{p,xi}) && (6) \\ &\propto \exp\left(-\frac{1}{2} \parallel \mathbf{x}_i - \mathbf{x}_{p,i} \parallel^2_{\Sigma_{p,xi}}\right), && (7) \\ \theta_j(\mathbf{l}_j) &= p(\mathbf{l}_j|\mathbf{l}_{p_j}, \Sigma_{p,lj}) && (8) \\ &\propto \exp\left(-\frac{1}{2} \parallel \mathbf{l}_j - \mathbf{l}_{p,j} \parallel^2_{\Sigma_{p,lj}}\right). && (9) \end{aligned}$$

Assuming a Gaussian measurement model, $\mathbf{z}_{km} = \mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) + \eta$, with $\eta \sim \mathcal{N}(0, \Sigma_M)$ we can write out the form of the measurement factors:

$$\psi_{km}(\mathbf{x}_k, \mathbf{l}_m) = p(\mathbf{x}_k, \mathbf{l}_m|\mathbf{z}_{km}) \propto p(\mathbf{z}_{km}|\mathbf{x}_k, \mathbf{l}_m) \quad (10)$$

$$\propto \exp\left(-\frac{1}{2} \parallel \mathbf{z}_{km} - \mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) \parallel^2_{\Sigma_M}\right). \quad (11)$$

The measurement factor ψ_{km} is Gaussian in $\mathbf{z}_{km}$ but is Gaussian in the variables $\mathbf{x}_k$ and $\mathbf{l}_m$ only if the measurement function $\mathbf{h}(\mathbf{x}_k, \mathbf{l}_m)$ is linear. In our case, we have a nonlinear measurement function, $\mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) = \pi\left(R_k\, \mathbf{l}_m + \mathbf{t}_k\right)$, where π is the projection operator and R_k and $\mathbf{t}_k$ are the rotations and translations derived from $\mathbf{x}_k$. As a result, we must update the measurement factors by relinearising during optimisation.

After linearising about some fixed point $(\mathbf{x}_{k,0}, \mathbf{l}_{m,0})$, the measurement factors can be expressed as a Gaussian distribution using the information form which is parametrised by an information vector $\boldsymbol{\eta}$ and information matrix Λ:

$$\mathcal{N}^{-1}(\mathbf{x}; \boldsymbol{\eta}, \Lambda) \propto \exp\left(-\frac{1}{2}\mathbf{x}^\top \Lambda \mathbf{x} + \boldsymbol{\eta}^\top \mathbf{x}\right). \quad (12)$$

The information form is used as it can represent distributions with rank deficient covariances in which a variable is not constrained at all along a particular direction. With this at hand and after a small amount of work [9], we find that linearised measurement factors take the following form:

$$\psi_{km}(\mathbf{x}_k, \mathbf{l}_m) = \mathcal{N}^{-1}\left(\begin{bmatrix}\mathbf{x}_k \\ \mathbf{l}_m\end{bmatrix}; \boldsymbol{\eta}_{km}, \Lambda_{km}\right), \quad (13)$$

where,

$$\boldsymbol{\eta}_{km} = \mathtt{J}^\top \Sigma_M^{-1}\left(\mathtt{J} \begin{bmatrix}\mathbf{x}_{k,0} \\ \mathbf{l}_{m,0}\end{bmatrix} + \mathbf{z}_{km} - \mathbf{h}(\mathbf{x}_{k,0}, \mathbf{l}_{m,0})\right), \quad (14)$$

$$\Lambda_{km} = \mathtt{J}^\top \Sigma_M^{-1} \mathtt{J}, \quad (15)$$

and the 2×9 Jacobian $\mathtt{J} = \left[\frac{\partial \mathbf{h}}{\partial \mathbf{x}_k}, \frac{\partial \mathbf{h}}{\partial \mathbf{l}_m}\right]\Big|_{\mathbf{x}_k = \mathbf{x}_{k,0}, \mathbf{l}_m = \mathbf{l}_{m,0}}$.

Now that all of our constraints are in the Gaussian form, finding the MAP solution is equivalent to minimising the negative log likelihood which is a sum of squared residuals:

$$\begin{aligned} \{X^*, L^*\} = \underset{\{X,L\}}{\arg\min} \Bigg[& \sum_{i=1}^{N_k} \parallel \mathbf{x}_i - \mathbf{x}_{p,i} \parallel^2_{\Sigma_{p,xi}} + \\ & \sum_{j=1}^{N_l} \parallel \mathbf{l}_j - \mathbf{l}_{p,j} \parallel^2_{\Sigma_{p,lj}} + \\ & \sum_{k=1}^{N_k} \sum_{m, \mathbf{l}_m \in L_k} \parallel \mathbf{z}_{km} - \mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) \parallel^2_{\Sigma_M} \Bigg]. \end{aligned} \quad (16)$$

5. Gaussian Belief Propagation for Bundle Adjustment

GBP is a Bayesian algorithm that can be used to solve bundle adjustment problems by computing the marginal distribution, with mean equal to the MAP solution, for all variables. In contrast, classical bundle adjustment methods compute a point estimate of the MAP solution using the Levenberg-Marquardt algorithm.

As the bundle adjustment factor graph is loopy, GBP stores a belief distribution at each variable node which converges to the marginal distribution after sufficient iterations of message passing. To describe the message passing equations, we do not distinguish between keyframe and landmark variable nodes and denote a variable node from the set $V = X \cup L$ as $\mathbf{v}_i$ and the belief stored at this node at iteration t, $b_i^t(\mathbf{v}_i) = \mathcal{N}^{-1}(\mathbf{v}_i; \boldsymbol{\eta}_{b_i}^t, \Lambda_{b_i}^t)$.

Prior factors send the same message, $pr_i(\mathbf{v}_i) = \mathcal{N}^{-1}(\mathbf{v}_i; \boldsymbol{\eta}_{p_i}, \Lambda_{p_i})$, to the variable node they connect to at all iterations. To describe the messages from measurement factors, we must first divide up the parameters of the factor distribution:

$$\psi_{ij}\left(\begin{bmatrix}\mathbf{v}_i \\ \mathbf{v}_j\end{bmatrix}\right) = \mathcal{N}^{-1}\left(\begin{bmatrix}\mathbf{v}_i \\ \mathbf{v}_j\end{bmatrix}; \begin{bmatrix}\boldsymbol{\eta}_i^{ij} \\ \boldsymbol{\eta}_j^{ij}\end{bmatrix}, \begin{bmatrix}\Lambda_{ii}^{ij} & \Lambda_{ij}^{ij} \\ \Lambda_{ji}^{ij} & \Lambda_{jj}^{ij}\end{bmatrix}\right). \quad (17)$$

The message passing rules [5] dictate that a pairwise factor ψ_{ij} computes the message to variable node $\mathbf{v}_i$ by taking the product of its factor distribution and the message from variable node $\mathbf{v}_j$ before marginalising over $\mathbf{v}_j$. After this calculation, the message from measurement factor ψ_{ij} to $\mathbf{v}_i$ at iteration $t+1$, $\mu_{j\to i}^{t+1}(\mathbf{v}_i) = \mathcal{N}^{-1}(\mathbf{v}_i; \boldsymbol{\eta}_{j\to i}^{t+1}, \Lambda_{j\to i}^{t+1})$, has the form :

$$\boldsymbol{\eta}_{j\to i}^{t+1} = \boldsymbol{\eta}_i^{ij} - \Lambda_{ij}^{ij}\left(\Lambda_{jj}^{ij} + \Lambda_{b_j}^t - \Lambda_{i\to j}^t\right)^{-1}\left(\boldsymbol{\eta}_j^{ij} + \boldsymbol{\eta}_{b_j}^t - \boldsymbol{\eta}_{i\to j}^t\right), \quad (18)$$

$$\Lambda_{j\to i}^{t+1} = \Lambda_{ii}^{ij} - \Lambda_{ij}^{ij}\left(\Lambda_{jj}^{ij} + \Lambda_{b_j}^t - \Lambda_{i\to j}^t\right)^{-1}\Lambda_{ji}^{ij}. \quad (19)$$

Variable nodes update their belief by taking a product of incoming messages from their prior factor and all adjacent

measurement factors. The belief information vector and information matrix are updated as follows:

$$\boldsymbol{\eta}_{b_i}^{t+1} = \boldsymbol{\eta}_{p_i} + \sum_{j,\psi_{ij}\in n(\mathbf{v}_i)} \boldsymbol{\eta}_{j\to i}^{t} \,, \tag{20}$$

$$\Lambda_{b_i}^{t+1} = \Lambda_{p_i} + \sum_{j,\psi_{ij}\in n(\mathbf{v}_i)} \Lambda_{j\to i}^{t} \,, \tag{21}$$

where the function $n(.)$ returns the adjacent nodes. The beliefs are sent as messages from the variable nodes to the factor nodes as the true message can be recovered at the factor node using the previous factor to variable message.

We use a synchronous scheduling, in which, at each iteration, all factor nodes relinearise and send messages to adjacent variable nodes before all variable nodes update their belief and send back messages to adjacent factor nodes. In our framework, relinearisation is done in an entirely local manner and a measurement factor is relinearised when the distance between the current belief estimate and the linearisation point of the variables the factor connects to is greater than a threshold β.

After sufficient iterations of message passing and relinearisation, the belief distributions converge to the marginal distributions:

$$b_i^t(\mathbf{v}_i) \to p(\mathbf{v}_i) \,. \tag{22}$$

A final detail to note is that we use message damping which is commonly used to stabilise the convergence of Loopy GBP [21]. We damp the update in Equation 18, such that $\boldsymbol{\eta}_{j\to i}^{t+1}$ is replaced with $(1-d)\,\boldsymbol{\eta}_{j\to i}^{t+1} + d\,\boldsymbol{\eta}_{j\to i}^{t}$, where d is a damping factor.

6. Robust Factors

It is well understood that measurements from real sensors usually have a distribution with gross outliers which is better represented by a function with heavier tails than a pure Gaussian measurement model. We can straightforwardly use such a robust cost function in our measurement factors within GBP. We employ a Huber function, which transitions from the usual quadratic cost to a linear cost when the Mahalanobis distance $M_{km}(\mathbf{x}_k, \mathbf{l}_m) = \| \mathbf{z}_{km} - \mathbf{h}(\mathbf{x}_k, \mathbf{l}_m) \|_{\Sigma_M}$ exceeds a threshold N_σ.

In order to maintain the Gaussian form of the factors in the linear loss regime, following [9, 2] we rescale the covariance of the noise in the Gaussian measurement model such that the contribution to the objective is equivalent to the Huber loss at this value. This has the effect of down-weighting or reducing the information of messages outgoing from this measurement factor. A measurement factor ψ_{km} then takes the following form before linearisation [9]:

$$\psi_{km}(\mathbf{x}_l, \mathbf{l}_m) \propto \begin{cases} \exp\left(-\frac{1}{2}M_{km}^2\right) & , M_{km} \leq N_\sigma \\ \exp\left(-\frac{1}{2}M_{km}^2[\frac{2N_\sigma}{M_{km}} - \frac{N_\sigma^2}{M_{km}^2}]\right) & , M_{km} \geq N_\sigma \end{cases} . \tag{23}$$

7. IPU Implementation

An IPU chip is massively parallel, containing 1216 independent compute cores called *tiles*. Each tile has 256KB local memory and 6 hardware threads that can all execute independent programs. In contrast, a GPU has very limited cache on chip, all data must be fetched from off chip DRAM, and there is less flexibility for executing different programs on each thread. The IPU's distributed on-chip SRAM means that memory accesses consume approximately 1pJ per byte whereas external DRAM accesses on a GPU/CPU consume hundreds of pJ per byte. Embedded variants of the IPU will therefore have significant power advantages over existing processors [1].

To implement GBP on the IPU we must map each node in the factor graph onto a tile on the IPU. The tiles are connected all-to-all with similar latency between all pairs of tiles on a chip [16] meaning that nodes can be mapped to arbitrary tiles. The most simple mapping places exactly one factor or variable node per tile, as in Figure 1, but limits the size of the factor graph to 1216 nodes. Noting that variable and factor nodes alternate in compute and that there are 6 threads per tile, in all experiments we are able to map much larger graphs to a single chip by placing multiple nodes per tile without affecting speed.

In order to exploit this parallelism the IPU employs a *bulk synchronous parallel* execution model. In this model all tiles compute in parallel using their local memories. When each tile has finished computing it enters a waiting phase (idle). When all tiles are finished, there is a short synchronisation phase (sync) across all tiles before data is copied between tiles with extremely high bandwidth in a predetermined schedule (exchange). This process then repeats as all tiles re-enter the compute phase. The period between syncs is not fixed but determined by the time taken for the computation.

GBP has three compute phases and two exchange phases in a single iteration. As shown in the upper part of Figure 3, factor nodes first relinearise and then compute their messages which are sent to adjacent variable nodes before the variable nodes update their beliefs which are sent back to adjacent factor nodes. The lower part of Figure 3 shows that the total time for a single iteration of GBP is less than $125\mu s$ while factor relinearisation and message compute makes up the bulk of the total compute time.

8. GBP Implementation

In experiments, we set the relinearisation threshold $\beta = 0.01$ and allow a factor to relinearise at most every 10 iterations. The damping is set to $d = 0.4$ and messages from factors are undamped for 8 iterations after relinearisation. This damping schedule allows newly relinearised messages to propagate through the graph while also stabilising later

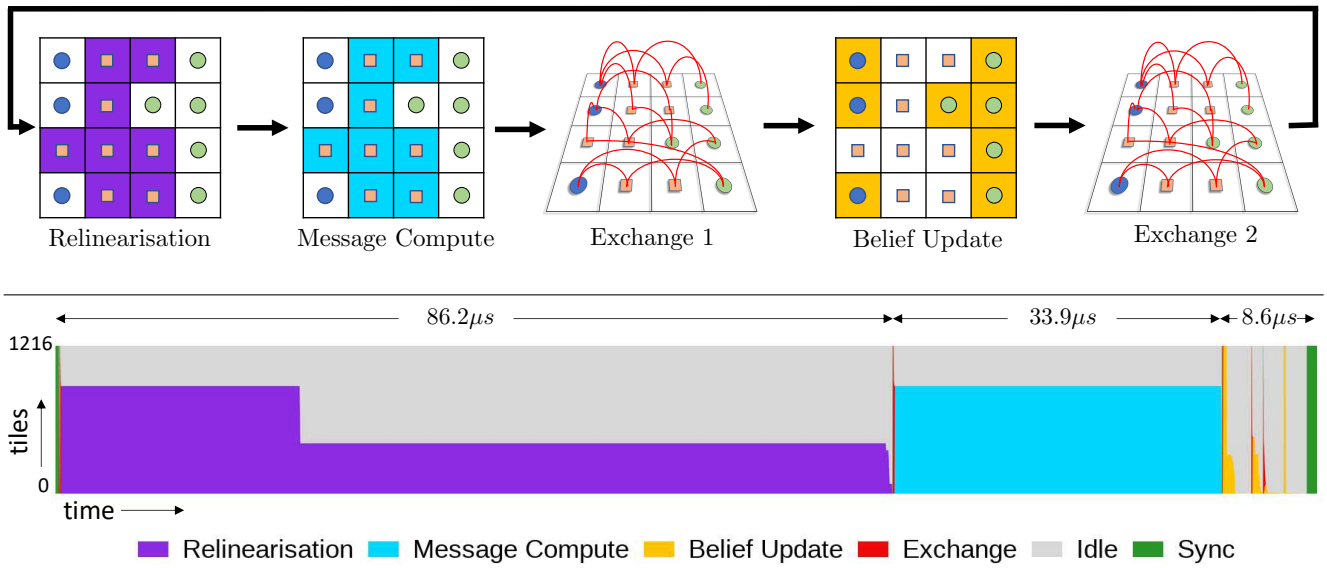


Figure 3: **IPU Phases.** *Above:* A schematic showing the compute on 16 tiles in a single iteration of GBP. Tiles are coloured when they are in a compute phase. In Exchange 1, factor nodes send messages to variable nodes and in Exchange 2 variable nodes send messages to factor nodes. Keyframe and landmark variable nodes are blue and green respectively and factor nodes are orange. *Below:* Plot shows the activity of each tile during a single iteration of GBP for a factor graph with 1216 nodes mapped 1-to-1 onto the tiles. In the Relinearisation phase, all 929 factors compute the distance of the adjacent beliefs from their linearisation point and a subset of these factors subsequently relinearise. The Belief Update is implemented with Graphcore's Poplibs™library and so is significantly faster and is indicative of the speed-ups possible with a more specific implementation using an optimised linear algebra library.

iterations. As the IPU handles halves and floats but not doubles, we found that it was necessary for numerical stability to use the Jacobians to automatically set prior constraints to initially have the same scale as the measurement constraints. These priors are then weakened to a hundredth of the strength gradually over 10 iterations. GBP is not sensitive to the mean of the prior and displays the same behaviour on convergence as when implemented on a CPU with doubles when the stronger priors are not required.

9. Experimental Evaluation

For evaluation we use sections of sequences from the TUM [30] and KITTI [13] data sets. We use ORBSLAM [22] as the front-end to select keyframes, generate ORB features [28] and handle correspondence. In all TUM experiments, landmarks are initialised at a depth of 1m from the first keyframe by which they are observed, while in KITTI experiments we initialise landmarks with Gaussian noise of standard deviation 0.5m.

We compare our implementation of GBP to Ceres [3], a non-linear least squares optimisation library often used for bundle adjustment. In all comparisons Ceres is run on a 6 core i7-8700K CPU with 18 threads (which we found experimentally to maximise performance) and uses Levenberg-Marquardt with Dense Schur and dense Cholesky on the reduced system, a Huber kernel and analytic derivatives.

9.1. Bundle Adjustment Speed Evaluation

First we present results to show that our implementation of GBP can rapidly solve large bundle adjustment problems.

We evaluate the optimisation speed by tracking the average reprojection error (ARE) over all measurements in the graph. Table 1 shows the time to converge to ARE < 1.5 pixels for 10 sequences with diverse camera motion and co-observation of landmarks in which keyframe positions are initialised with Gaussian noise of standard deviation 7cm. The corresponding ARE curves for 3 of the sequences are plotted on the left in Figure 4. GBP reaches convergence an average of 24x faster than Ceres over the 10 sequences. Typically GBP takes between 50-300 iterations to converge and Ceres takes between 10-40 steps, however, due to the rapid in-place computation on the IPU, which operates at 120W, GBP is significantly faster.

9.2. SLAM Speed Evaluation

In GBP, the confidence in the belief estimations grows over iterations as the beliefs tend towards the marginal distributions. This Bayesian property is an inherent advantage

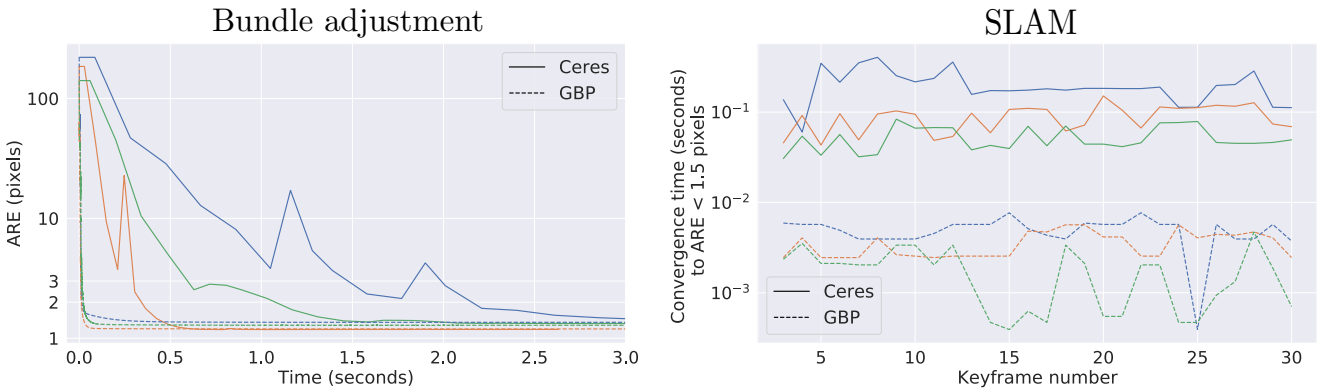


Figure 4: **Speed Comparison.** Note the logarithmic scale on the y axes. *Left:* **Bundle adjustment.** ARE for 3 sequences fr1desk, fr2desk, fr3teddy. fr1desk is more difficult as it has the most measurements and the camera moves a large distance. fr3teddy has 125 keyframes but is easier to solve as fewer landmarks are densely observed in object reconstruction. Similar results were observed for the other TUM sequences whose convergence times are described in Table 1. *Right:* **SLAM.** Time to converge to ARE < 1.5 pixels after a new keyframe is added and initialised with the pose of the most recent keyframe. Results are for the first 30 keyframes of the sequences fr1desk, fr2desk, fr3teddy.

Table 1: The final two columns give the time in milliseconds to converge to ARE < 1.5 pixels for 10 sequences from the TUM data set (two testing sequences, 4 handheld camera sequences, 2 robot mounted sequences, 2 object reconstruction sequences) and 2 from the KITTI data set. k is the number of keyframes, p landmarks, m measurements.

Sequence	k	p	m	GBP	Ceres
fr1xyz	42	2194	12908	**37.2**	1180
fr1rpy	34	1999	8920	**130.3**	1030
fr1desk	63	2913	13514	**77.3**	2850
fr1room	20	1467	5388	**31.7**	779
fr2desk	40	892	3995	**20.8**	425
fr3loh	36	1140	5065	**44.6**	470
fr2robot360	40	333	1745	**51.5**	212
fr2robot2	20	567	4036	**8.6**	345
fr1plant	40	1824	6818	**31.8**	1450
fr3teddy	125	1919	9032	**40.0**	1450
KITTI00	30	2745	16304	**14.2**	342
KITTI08	30	3053	10480	**14.8**	394

over batch methods that make point estimates in the SLAM setting. For GBP, new variables are quickly snapped into a state that is consistent with the current estimates given the new constraints, while for batch methods, the full solution must be recomputed to refine just a few variables.

We go towards validating this advantage in incremental SLAM by comparing the time taken to converge to ARE < 1.5 pixels after each new keyframe is added for 3 TUM sequences with 30 keyframes. New keyframes are initialised at the location of the most recent keyframe and new landmarks at a depth of 1m. To aid Ceres and mimic the

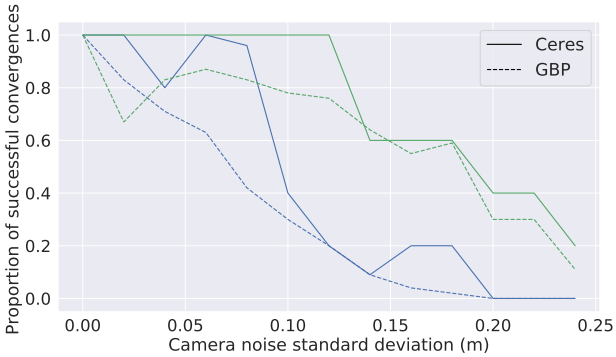


Figure 5: **Convergence basin comparison.** Proportion of successful convergences over 100 trials for different noise levels with the fr1desk and fr3teddy TUM 30-keyframe sequences. A successful convergence constitutes reaching ARE < 1.5 pixels.

Bayesian approach, we fix the landmarks for the first 3 steps of Levenberg-Marquardt optimisation. Results are shown in the right plot in Figure 4 for which on average, over the 90 keyframes added, GBP converges 36x faster than Ceres, often in fewer than 10 iterations.

9.3. Robustness Evaluation

We compare the robustness of GBP and Ceres in solving BA problems by varying the noise added to the keyframe initialisation and counting the proportion of successful convergences over 100 trials at each noise level. Figure 5 shows that GBP has a comparable convergence radius to Ceres for these two TUM sequences.

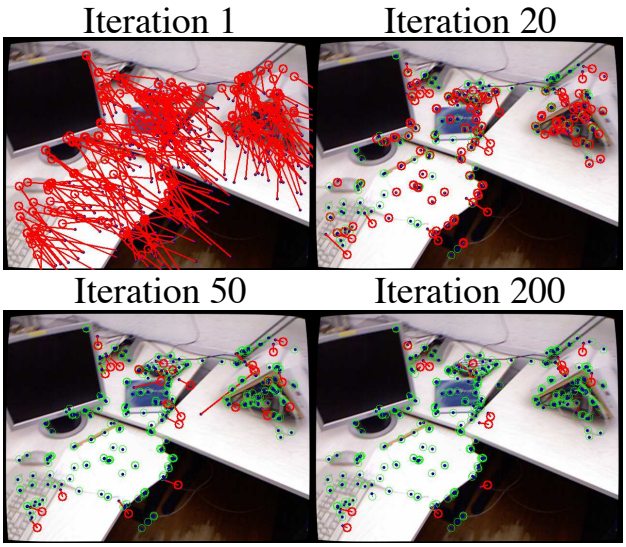
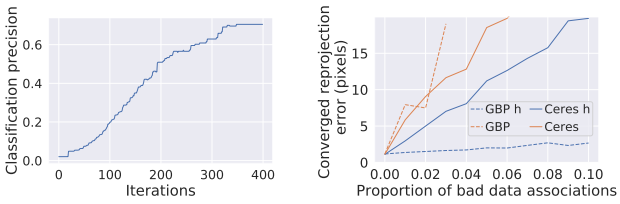


Figure 6: **GBP with Huber loss.** Landmark projections (blue points) and measurements (circles) are connected by lines. The lines and circles are red when the reprojection error exceeds the Huber threshold and the down-weighting of the message is proportional to the length of the red line.

9.4. Huber Loss Evaluation

The Huber loss function has the effect of down-weighting messages from factors that may contain outlying measurements. We demonstrate this effect in Figure 6 in which we visualise the reprojection errors at iterations 1, 20, 50 and 200 of GBP in a chosen keyframe for which 10% of measurements are artificially added outliers. All measurements begin in the outlier regime and after 20 iterations a large proportion of the measurements remain in this regime as GBP has not yet worked out which measurements are inliers. By iteration 200, only the erroneous measurements are in the outlier regime as GBP has determined that these measurements are least consistent with other constraints in the graph. This behaviour of gradually removing false positive outlier classifications can be observed in Figure 7a, for a sequence in which 3% of data associations are incorrect.

To validate quantitatively the benefits of the Huber loss with both GBP and Ceres, we conduct an ablation study on a sequence with incorrect data associations and measure the converged reprojection error. Figure 7b shows that for GBP, the Huber loss is necessary and effective in handling incorrect data associations. For Ceres however, the same Huber loss is unable to identify the outliers and Ceres cannot arrive at a low ARE solution. This indicates that GBP's local consideration of outliers may be more effective than the global consideration in LM.



(a) (b)

Figure 7: Results for a 20 keyframe sequence from fr1desk in which bad data associations are artificially added. *(a)* Measurements are classified as outliers if they are in the linear loss regime. The recall is 1 over all iterations. ARE converges to < 1.5 pixels after 268 iterations while the precision is still increasing. *(b)* h indicates Huber loss is used. For GBP, convergence is not reached without a Huber loss for more than 3% bad associations, while with a Huber loss GBP can down-weight the outliers and solve the bundle adjustment problem. For Ceres, the Huber loss improves the final ARE however it still cannot converge the solution.

10. Discussion / Conclusion

We have shown that with the emergence of new flexible computer architecture for AI, specifically Graph Processors like Graphcore's IPU, Gaussian Belief Propagation can be a flexible and efficient framework for inference in Spatial AI problems. By mapping the bundle adjustment factor graph onto the tiles of a single IPU, we demonstrated that GBP can rapidly solve a variety of bundle adjustment problems with a 24x speed advantage over Ceres. Additionally, we gave an indication of the framework's capacity to efficiently solve incremental SLAM problems and be robust to outlying measurements.

In the near term, we would like to apply GBP to very large bundle adjustment problems. Our framework scales arbitrarily to multiple chips, and Graphcore provide a custom interconnect for highly efficient inter-IPU message passing. An even more interesting direction which looks towards low power embedded Spatial AI would investigate how to fit large problems on a single chip by merging or replacing factors using a combination of network priors and marginalisation. We hope that our framework of flexible, in-place optimisation on a dynamically changing factor graph will be applied to a broad spectrum of AI tasks incorporating heterogeneous factors.

Acknowledgements

We thank Tristan Laidlow, Jan Czarnowski and Edgar Sucar for fruitful discussions.

References

[1] Graphcore. URL https://www.graphcore.ai/. 1, 5

[2] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard. Robust map optimization using dynamic covariance scaling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012. 5

[3] S. Agarwal, Mierle K., and Others. Ceres solver. http://ceres-solver.org. 6

[4] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building Rome in a Day. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2009. 2

[5] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, Inc., 2006. 3, 4

[6] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. J. Davison. CodeSLAM — learning a compact, optimisable representation for dense visual SLAM. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[7] D. Crandall, A. Owens, N. Snavely, and D. Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 2

[8] A. J. Davison. FutureMapping: The computational structure of Spatial AI systems. *arXiv preprint arXiv:arXiv:1803.11288*, 2018. 1

[9] A. J. Davison and J. Ortiz. FutureMapping 2: Gaussian Belief Propagation for Spatial AI. *arXiv preprint arXiv:arXiv:1910.14139*, 2019. 2, 3, 4, 5

[10] Z. DeVito, M. Mara, M. Zollhöfer, G. Bernstein, and J. Ragan-Kelley. Christian eobalt, pat hanrahan, ma hew fisher, and ma hias nießner. 2016. opt: A domain specific language for non-linear least squares optimization in graphics and imaging. In *ACM Transactions on Graphics (TOG)*, 2017. 2

[11] J. Engel, V. Koltun, and D. Cremers. Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2017. 2

[12] J. Folkesson and H. Christensen. Graphical SLAM — a self-correcting map. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004. 2

[13] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? The KITTI vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 6

[14] S. Gupta, S. Choudhary, and P.J.Narayanan. Practical time bundle adjustment for 3D reconstruction on GPU. In *ECCV Workshop on Computer Vision on GPUs*, 2010. 2

[15] Y. Jeong, D. Nister, D. Steedly, R. Szeliski, and I.S. Kweon. Pushing the envelope of modern methods for bundle adjustment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 2

[16] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza. Dissecting the Graphcore IPU architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413*, 2019. 5

[17] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree. *International Journal of Robotics Research (IJRR)*, 2012. To appear. 2

[18] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental Smoothing and Mapping. *IEEE Transactions on Robotics (T-RO)*, 24(6):1365–1378, 2008. 2

[19] D. Lacey. New Graphcore IPU Benchmarks. URL https://www.graphcore.ai/posts/new-graphcore-ipu-benchmarks, 2019. 2

[20] F. Lu and E. Milios. Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 4(4):333–349, 1997. 2

[21] D. M. Malioutov, J. K. Johnson, and A. S. Willsky. Walksums and belief propagation in Gaussian graphical models. *Journal of Machine Learning Research*, 7(Oct):2031–2064, 2006. 5

[22] R. Mur-Artal, J. M. M Montiel, and J. D. Tardós. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *IEEE Transactions on Robotics (T-RO)*, 31(5):1147–1163, 2015. 2, 6

[23] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 1999. 3

[24] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H.J. Kelly, A. J. Davison, M. Lujan, M. F.P. OBoyle, G. Riley, N. Topham, and S. Furber. Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015. 1

[25] M. A. Paskin. Thin Junction Tree Filters for Simultaneous Localization and Mapping. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2003. 2

[26] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 1988. 2

[27] A. Ranganathan, M. Kaess, and F. Dellaert. Loopy SAM. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2007. 2

[28] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *Proceedings of the International Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE, 2011. 6

[29] S. Saeedi, B. Bodin, H. Wagstaff, A. Nisbet, L. Nardi, J. Mawer, N. Melot, O. Palomar, E. Vespa, T. Spink, et al. Navigating the landscape for real-time localization and mapping for robotics and virtual and augmented reality. *Proceedings of the IEEE*, 2018. 2

[30] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A Benchmark for the Evaluation of RGB-D SLAM Systems. In *Proceedings of the IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, 2012. 6

[31] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle Adjustment — A Modern Synthesis. In *Proceedings of the International Workshop on Vision Algorithms, in association with ICCV*, 1999. 2

[32] Y. Weiss and W. T Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. In *Neural Information Processing Systems (NIPS)*, 2000. 3

[33] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz. Multicore Bundle Adjustment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 2

[34] Z. Zhang, A. A. Suleiman, L. Carlone, V. Sze, and S. Karaman. Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach. In *Proceedings of Robotics: Science and Systems (RSS)*, 2017. 2