SuperNoVA: Algorithm-Hardware Co-Design for Resource-Aware SLAM

Seah Kim
University of California, Berkeley
Berkeley, CA, USA
seah@berkeley.edu

Roger Hsiao
University of California, Berkeley
Berkeley, CA, USA
roger_hsiao@berkeley.edu

Borivoje Nikolić
University of California, Berkeley
Berkeley, CA, USA
bora@berkeley.edu

James Demmel
University of California, Berkeley
Berkeley, CA, USA
demmel@berkeley.edu

Yakun Sophia Shao
University of California, Berkeley
Berkeley, CA, USA
ysshao@berkeley.edu

Abstract

Simultaneous Localization and Mapping (SLAM) plays a crucial role in robotics, autonomous systems, and augmented and virtual reality (AR/VR) applications by enabling devices to understand and map unknown environments. However, deploying SLAM in AR/VR applications poses significant challenges, including the demand for high accuracy, real-time processing, and efficient resource utilization, especially on compact and lightweight devices. To address these challenges, we propose SuperNoVA, which enables high-accuracy, real-time, large-scale SLAM in resource-constrained settings through a full-stack system, spanning from algorithm to hardware. In particular, SuperNoVA dynamically constructs a subgraph to meet the latency target while preserving accuracy, virtualizes hardware resources for efficient graph processing, and implements a novel hardware architecture to accelerate the SLAM backend efficiently. Evaluation results demonstrate that, for a large-scale AR dataset, SuperNoVA reduces full SLAM backend computation latency by 89.5% compared to the baseline out-of-order CPU and 78.6% compared to the baseline embedded GPU, and reduces the maximum pose error by 77% over existing SLAM solutions, while always meeting the latency target.

1 Introduction

Simultaneous Localization and Mapping (SLAM) is a critical workload in robotics [10, 47], autonomous systems [11, 48] and AR/VR [23, 25, 45], enabling navigation and map construction of unknown environments in real-time. It involves the integration of sensor data to concurrently estimate the position and construct a map of its surroundings. Specifically, the demand for SLAM in AR/VR is driven by a diverse range of applications, including gaming, architectural visualization, remote collaboration, and training simulations, where the seamless integration of virtual content with the real world is crucial for delivering compelling immersive experiences. In these critical applications, the accuracy and efficiency of SLAM algorithms are paramount to providing spatial understanding and seamless tracking.

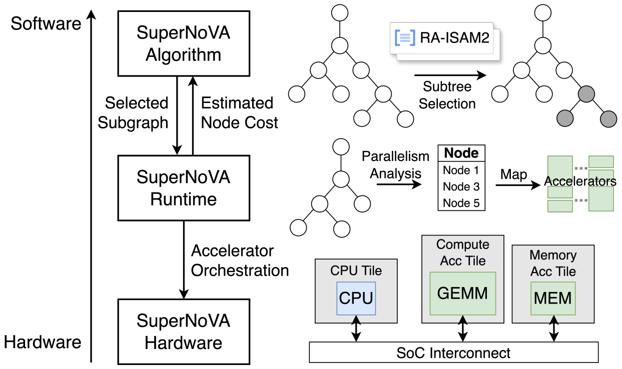


Figure 1. SuperNoVA is a novel full-stack system for co-designing algorithm-hardware for high-accuracy real-time SLAM solver on resource-constrained settings.

However, SLAM deployment is challenging in AR/VR applications due to several key constraints. First, seamlessly integrating virtual content into the physical world requires precise tracking of both user position and orientation. Achieving high accuracy relies on processing large volumes of sensor data and trajectories, which is computationally intensive. Second, SLAM must operate in real-time to provide immediate feedback to the system and to match the strict sensor input rate and frame update rate. Balancing real-time requirements while maintaining accuracy and efficiency is a significant challenge. Third, physical device constraints, such as stringent battery and form-factor constraints, impose limitations on available on-device computational resources, further adding to the challenge.

Existing SLAM solutions, both hardware and algorithms, fall short of these requirements, suffering from either high computational costs too high for real-time applications or accuracy degradation over time [31, 52]. Previous efforts to accelerate SLAM using multi-core CPUs [1] and GPUs [2, 13, 41] do not fully leverage SLAM-specific properties, leading to inefficient or power-hungry hardware that is unsuitable for battery- and area-constrained wearable devices. Attempts with specialized accelerators like FPGAs [17, 34] or ASICs

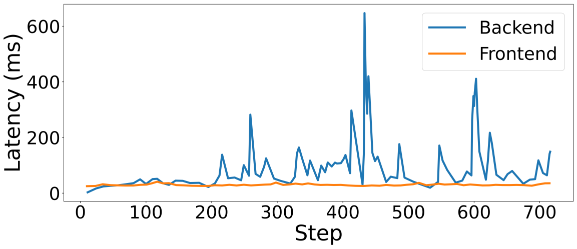


Figure 2. Performance breakdown of a typical SLAM system.

[32, 48] have focused on smaller-scale problems and lack the scalability and adaptability needed for broader AR/VR applications. Thus, there is a pressing demand for innovative approaches that can deliver superior performance while remaining flexible and scalable to meet the evolving requirements of AR/VR applications.

To address these challenges, this work presents SuperNoVA[1], a novel full-stack system that co-designs the algorithm and hardware to enable resource-aware SLAM backend. As shown in Figure 1, SuperNoVA consists of 1) an algorithm that computes a subproblem of **SuperNo**des to meet the latency requirement, 2) a dynamic runtime algorithm that **V**irtualizes hardware resources from application thread for efficient tree graph processing, and 3) a hardware **A**rchitecture that accelerates the SLAM backend efficiently. Our evaluation shows that SuperNoVA reduces the full SLAM backend computation latency by 89.5% compared to out-of-order CPU and 78.6% compared to GPU, and reduces the maximum pose error by 77% compared to existing SLAM solutions for a large-scale AR dataset, LaMAR [46], while meeting the target processing time. In summary, this paper makes the following contributions:

- We propose SuperNoVA, a novel full-stack system to enable a real-time, high-accuracy global SLAM solver in resource-constrained settings.
- We develop the SuperNoVA algorithm, a resource-aware incremental smoothing and mapping algorithm that achieves high accuracy while always meeting the target deadline.
- We implement the SuperNoVA runtime, which virtualizes accelerators for efficient run-time management of hardware resources for graph processing.
- We design the SuperNoVA hardware architecture to accelerate the SLAM backend, which delivers high and robust performance, and implement it in RTL.

2 Background and Motivation

This section discusses SLAM deployment challenges and prior works on the SLAM backend algorithm and its hardware acceleration.

Table 1. SLAM processing requirement for AR/VR.

Refresh Rate (Hz)	Absolute Error (cm)	Power (W)	Trajectory Length (m)
30	5	1	1000

2.1 SLAM Deployment and Requirements

SLAM is ubiquitous in the domain of autonomous vehicles, robotics, and AR/VR applications. Localization refers to estimating the system's current position, and mapping refers to constructing a representation of the surrounding environment. For example, an autonomous system uses SLAM to make decisions about its next action; an AR/VR application relies on SLAM in order to display realistic 3D scenes. The AR/VR use case is particularly challenging because it requires a low-latency, high-absolute-accuracy response for a large-scale problem as shown at Table 1 [25, 46]. A missed deadline or inaccurate localization may lead to frame skipping or inconsistent geometry in rendering, which induces motion sickness over long periods of use [36**?**].

A SLAM workload can be roughly split into two stages, the frontend and the backend. The frontend collects and processes various sensor measurements to create constraints between unknown states (poses, landmark locations, etc.). The backend performs a maximum-a-posteriori (MAP) estimation given all the constraints. In general, the frontend has a small, fixed computation, requiring little resources to meet the target refresh rate, while the backend latency depends heavily on the trajectory of the system and the environment. Figure 2 shows the performance breakdown of a SLAM system [45] running the EuRoC dataset [9] on a commercial Intel Xeon Gold 6354 Processor. Unlike the frontend, the backend latency varies drastically from iteration to iteration. This backend dynamicity is the major obstacle to designing an accurate, real-time SLAM solution.

2.2 SLAM Solver

The SLAM backend is a state estimation problem that consists of a linear least-squares (LLS) problem in the inner loop of the form Equation (2). There are many prior works with different formulations and library implementations to efficiently solve this LLS problem, as shown in Table 2. Local methods such as local bundle adjustment (LBA), visual-inertial odometry (VIO), and extended Kalman filter (EKF) methods provide a low-latency estimate of the recent trajectory or local map space by discarding states that are outside of a sliding window. The main drawback of local methods is that small errors can accumulate and cause a large drift over time. On the other hand, global methods such as full bundle adjustment (FBA) and pose graph optimization (PGO) are capable of handling loop closure (LC) events (when the system returns to a previous location) and correcting the drift. Thus, a common multi-level SLAM setup would have a local solver

[1] https://github.com/ucb-bar/SuperNoVA

Table 2. Comparison of SLAM backend solvers.

Backend solvers	Local	Global	Incremental	RA-ISAM2
Example libraries	ROVIO [8], G2O [19]	PyPose [49], G2O [19]	GTSAM [14]	**Ours**
Global consistency	✗	✓	✓	✓
Bounded latency	✓	✗	✗	✓
Loop closure	✗	✓	✓	✓
Resource-aware	✗	✗	✗	✓

running at sensor frequency and a global solver that handles LCs. An LC event may update a large portion of the map. During the long latency of solving the LC, the system must use the state estimate from the local solver, which may cause a sudden spike in pose estimation error that leads to serious consequences, such as inconsistent AR/VR rendering.

Incremental methods, such as incremental smoothing and mapping (ISAM2), are a separate class of solvers that is widely considered to be state-of-the-art in online SLAM [26]. Incremental methods update *only the affected sub-map*. In most cases, they achieve high accuracy with very low latency; however, upon an LC event, a large part of the map needs to be updated, leading to high processing latency. The variability in latency (such as that in Figure 2) makes ISAM2 unsuitable for latency-critical applications such as AR/VR.

Our proposed solution, SuperNoVA, is capable of handling LC events with high accuracy while ensuring that the target latency is met, by using a resource-aware incremental algorithm (RA-ISAM2).

2.3 SLAM Hardware Acceleration

Modern mobile SLAM applications commonly use CPU as the compute platform [7, 42], which is versatile and can handle a wide range of tasks. However, multi-core CPUs lack the parallel processing power required for real-time SLAM. To leverage the parallelism in compute-intensive SLAM tasks, GPU solutions have been widely proposed [2, 13, 41], but they suffer from high power consumption and large form factor, which is not suitable for wearable AR/VR devices. Custom accelerators for SLAM have also been proposed; however, prior works focus on smaller-scale problems, such as SoCs for VIO [48] or LBA [32] for drones, or FPGA-based acceleration [17, 21, 22, 34, 51].

SLAM for AR/VR applications is particularly challenging due to the large, dynamic problem size and stringent constraints on power, weight, and latency. This motivates an SoC-based approach. Prior works [17, 22, 32, 34, 48] in SLAM hardware acceleration have followed a common paradigm: they characterize the algorithm and operating conditions (e.g., power, latency, accuracy) and then design accelerators tailored to these specifications. While effective for static problem sizes up to tens of timesteps, these designs struggle to handle dynamic environmental changes such as LCs, where the sparse matrix structure is determined at runtime. Once generated, the fixed-functional units are physically limited to the predetermined problem sizes and cannot guarantee bounded latency for larger problems. This inflexibility makes prior solutions unsuitable for large-scale, real-time SLAM.

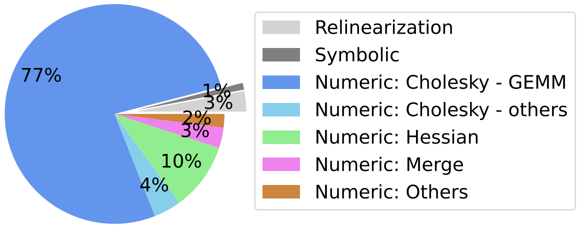


Figure 3. Representative SLAM backend latency breakdown.

In contrast, SuperNoVA takes a fundamentally different approach. It adapts in real-time to both environmental conditions and available hardware resources by changing the algorithm at runtime. The flexible orchestration of programmable, virtualized hardware accelerators supports variable-sized supernodes and exploits different levels of parallelism on-the-fly, which is the key to sparse matrix factorization. This adaptability allows it to scale to much larger problems, handling thousands of timesteps with latency guarantees.

Besides SLAM accelerators, there has been prior research on accelerating matrix factorization using systolic array-based accelerators for matrix multiplication (GEMM), which is a key operation in the SLAM backend (Figure 3). However, these works accelerate a static matrix factorization problem without the context of SLAM application's dynamicity, which leads to a lack of consideration for critical system operations such as dynamic memory management [16] and prevents them from taking advantage of the application-specific properties of the problem such as block sparsity [50].

Therefore, a systematic way of designing both algorithm and hardware is needed. To the best of our knowledge, SuperNoVA is the first full-stack approach to co-designing algorithm and hardware for large-scale, real-time SLAM on resource-constrained platforms.

3 SLAM Algorithm

This section discusses the mathematical formulation of the SLAM backend problem, and provides a background to the incremental smoothing and mapping (ISAM2) algorithm, which is the basis for RA-ISAM2.

3.1 Overview

The SLAM backend is a state estimation problem which can be written as the nonlinear least-squares (NLS) problem over

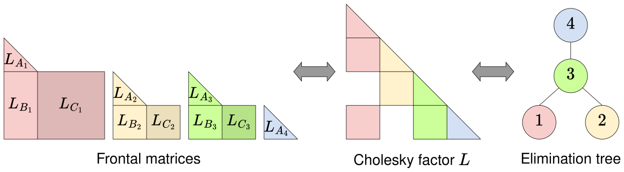


Figure 4. Elimination tree and frontal matrix representation. The L_Cs are discarded after the factorization.

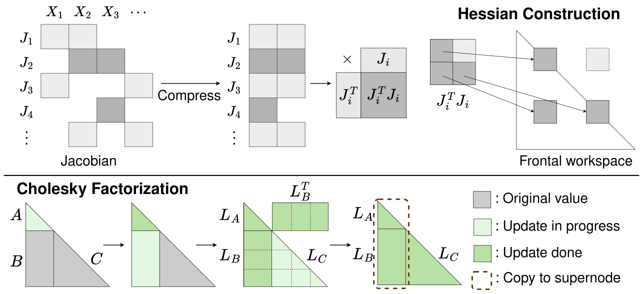


Figure 5. Hessian construction and Cholesky factorization.

a set of factors

$$X = \underset{X}{\operatorname{argmin}}\, \phi(X) = \underset{X}{\operatorname{argmin}} \sum_i \|\phi_i(X)\|_2^2 \tag{1}$$

A factor $\phi_i(X)$ represents the reprojection error of a sensor measurement. Each component X_j represents a variable to be estimated, such as a pose or a landmark. The Gauss-Newton's method of solving this NLS problem linearizes Equation (1) as the standard linear least-squares (LLS) problem

$$\underset{\Delta}{\operatorname{argmin}}\, \|J\Delta - f\|_2^2 \tag{2}$$

The solution of the LLS is used to obtain the next state estimation $X^{(k+1)} = X^{(k)} \oplus \Delta$, where $\oplus$ denotes the retraction operation over the optimization manifold.

The most common approach to solving Equation (2) is the normal equations, which solve the linear system $J^T J\Delta = J^T f$. J is the Jacobian matrix, $H = J^T J$ is the Hessian matrix, and $g = J^T f$ is the gradient vector. Since H is symmetric positive definite, we take the Cholesky factorization $H = LL^T$, for a lower triangular Cholesky factor L. Then the solution Δ can be obtained after two efficient triangle solves $Ly = g$ and $L^T\Delta = y$.

3.2 Partial Factorization

The Hessian matrix H is an unstructured sparse matrix, whose Cholesky factor L can be represented as an elimination tree [33]. For efficient computation, each vertex in the tree represents a *supernode*, a set of columns in L that have the same nonzero pattern. The Cholesky factorization of H can then be described as performing the partial factorization of each supernode from leaf to root [15]. For the remainder of this paper, we will use *node* and *supernode* interchangeably.

The frontal matrix representation of the node j is $H_j = \begin{bmatrix} A & B^T \\ B & C \end{bmatrix} \in \mathbb{R}^{(m+n)\times(m+n)}$. Because columns in the node all share the same row indices, H_j is stored as a dense matrix with an auxiliary row index vector. The high-level goal of the partial factorization is to compute $L_j = \begin{bmatrix} L_A & 0 \\ L_B & L_C \end{bmatrix}$. Only the first m columns strictly belong to the node, while L_C is a temporary matrix used to update subsequent columns. A common implementation in high-performance software is to perform the partial factorization of H_j in a temporary workspace (frontal workspace), and copy $\begin{bmatrix} L_A \\ L_B \end{bmatrix}$ to a separate memory location. Figure 4 shows the two equivalent representations (matrix, tree) of the Cholesky factor L.

Hessian Construction. Instead of materializing H in full, it is more efficient to construct H_j before factorizing node j. Let H'_j be the symmetric frontal matrix of node j before Hessian construction, whose lower triangular half is stored. We need to compute $H_j = H'_j + \sum_{i\in S_j} J_i^T J_i$, for all the J_i that belong to this node. J_i is the linearized form of the factor ϕ_i, and is typically a very sparse block-row in J. Hessian construction is therefore made up of a series of small GEMM and scatter-addition, shown in Figure 5 (top).

Cholesky Factorization. Computing L_j from H_j involves three steps, which is also described in Figure 5 (bottom).

1. Compute the dense Cholesky factor $A = L_A L_A^T$
2. Do a triangle solve on the subdiagonal block $L_B L_A^T = B$
3. Perform a symmetric rank-k update $L_C = C - L_B L_B^T$

Merge. Let node $j+1$ be the parent node of j. We update the parent frontal matrix with L_C. The row indices of L_C are a strict subset of the row indices of H'_{j+1}; therefore, the entries of L_C need to be scattered into H'_{j+1}.

3.3 Non-Numeric Operations

In Section 3.2, we describe the numeric operations for solving the LLS problem. For simplicity, we will refer to the operations required to set up the problem as *non-numeric operations* (regardless of the actual computation), which also contribute to the end-to-end latency.

Symbolic Factorization. Before the numeric factorization, the nonzero pattern of the Cholesky factor L is computed and the required memory for the matrix entries is allocated.

Relinearization. The Jacobian matrix J is the first derivative of ϕ evaluated at X. Therefore, a block-row i is computed as the linearization of the factor $J_i = J_{\phi_i}(X)$. Each factor is independent, thus it is trivially parallelizable.

Figure 3 shows the full backend run-time breakdown for a large-scale AR dataset on an out-of-order CPU. Since the numeric operations are dominant, SuperNoVA focuses on numeric acceleration, and runs non-numeric parts on CPU.

3.4 Incremental Smoothing and Mapping (ISAM2)

ISAM2 is an incremental NLS solver that achieves high accuracy at each step, usually with a very low amount of computation. This is enabled by a technique called *fluid relinearization*, which evaluates J at a linearization point (LP) Θ instead of at the current state estimate X. LP is the state estimate at which the Jacobian matrix is evaluated. A component Θ_j is only updated to X_j if the step size of that variable is large enough, i.e. $\|\Delta_j\|_\infty > \beta$ for some small constant β.

A factor ϕ_i is usually associated with a very small set of states. If any of these states updated their LPs, then J_i must be re-computed. All nodes associated with J_i must be re-factorized, along with all of their ancestors. On a non-loop-closure frame, most updates occur towards to root of the tree, leading to a small computation cost. Conversely, a loop closure frame updates a node deep in the tree, causing a large update. This is the main source of latency variability in incremental SLAM, which SuperNoVA tackles with its resource-aware algorithm.

4 SuperNoVA System

SuperNoVA is a complete full-stack system for the SLAM backend computation in which algorithm and hardware are co-designed to process large-scale SLAM workloads under resource constraints. As Figure 1 shows, SuperNoVA is composed of 1) a resource-aware algorithm that dynamically constructs the most relevant sub-problem that fits within the target deadline while maintaining accuracy, 2) a runtime system that allocates multi-accelerator compute resources to the supernodes of the tree graph by exploiting different types of parallelism, and 3) a hardware architecture that accelerates the real-time SLAM backend efficiently with programmable accelerators for sparse matrix computation and dynamic memory management.

This section discusses in the order of the SuperNoVA algorithm, hardware architecture, and the runtime system.

4.1 SuperNoVA Algorithm

We propose Resource-Aware Incremental Smoothing and Mapping (RA-ISAM2), a novel incremental SLAM algorithm that dynamically adjusts the algorithm complexity based on available system resources and target deadline. RA-ISAM2 builds upon ISAM2 by leveraging the incremental partial updates for resource-awareness. This approach addresses the variable latency issue in ISAM2 by modifying the relinearization conditions for variables. We define the *relevance score* of a variable j as the size of the update step $\|\Delta_j\|_\infty$, which represents the optimal update of the variable from its LP. A larger distance from LP implies that the Jacobian matrices associated with that variable have higher errors. Instead of checking for $\|\Delta_j\|_\infty > \beta$, we estimate the cost of relinearization and only update Θ_j if it does not violate the target deadline. A greedy algorithm is used so that variables with high relevance scores are considered first. The intuition is that variables that are farther away from their linearization points cause more inaccuracies in the state estimation. By only selecting the most relevant variables at each step, we can amortize the cost of loop closure over several steps, where each step has an acceptably high accuracy while meeting the latency target.

Algorithm 1 Compute relinearization cost of a variable

```
1:  function ComputeRelinCost(candidate_variable)
2:      affected_variables ← all variables that share a factor with candidate_variable
3:      relin_cost ← 0
4:      for variable in affected_variables do
5:          relin_cost += ComputePathCost(variable.node)
        return relin_cost
6:  function ComputePathCost(start_node)
    ▷ % Traverse up the tree and compute the latency cost of each node until a previously visited node %
7:      node ← start_node
8:      while node != root and !node.visited do
9:          node.visited = True
10:         node.cost = ComputeNodeCost(node)
11:         end_node = node
12:         node = node.parent
        ▷ % Traverse down and compute each path cost%
13:     end_node.path_cost = end_node.cost
14:     for node from end_node to start_node do
15:         node.path_cost = node.cost + parent.path_cost
        return start_node.path_cost
```

Algorithm 1 shows the procedure to estimate the cost of relinearizing a variable. Selecting a variable affects all the variables it shares a factor with (line 2), causing an update to the corresponding nodes and their paths to the root. Thus, the cost of relinearizing a variable is the sum of the cost of all updated paths. For each path, Lines 7-10 first compute the cost of each node individually, accounting for the latency of both numeric and non-numeric operations. Lines 13-15 traverse down the tree and sum up the path cost for each node, which is returned as the estimated relinearization cost. If it is less than the remaining processing time, the variable will be selected for relinearization and the cost is subtracted from the remaining time. The path cost of each node is cached so that the cost estimation step can be shortened if it encounters a repeated node. Thus, the overhead of the adaptive relinearization algorithm is at most two visits per node.

4.2 SuperNoVA Hardware

Figure 6 describes the SuperNoVA hardware SoC architecture. SuperNoVA hardware provides a scalable, multi-core, multi-accelerator architecture for large-scale SLAM problems. SoC components, including the accelerator configuration and the

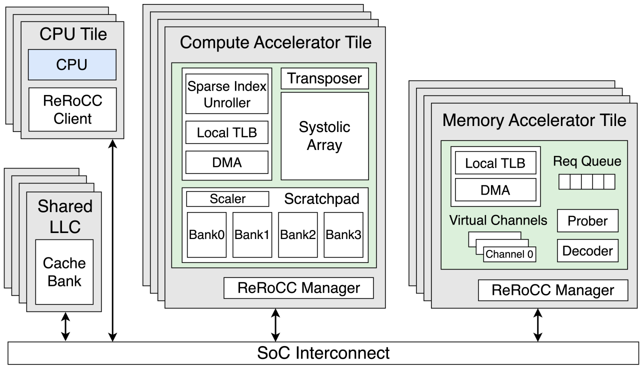


Figure 6. SuperNoVA hardware architecture.

number of accelerators and CPU tiles, are all configurable at design time. It consists of two types of programmable accelerators, the compute accelerator (COMP) to accelerate matrix operations in SLAM and the memory accelerator (MEM) to perform efficient memory operations for workspace-based algorithms.

In large-scale, real-time SLAM, the sparse matrix structure updates at each step, which incurs a large overhead in sparse index management and dynamic memory allocation. To address this, SuperNoVA adds a new sparse indexing unit to handle block-sparse matrix scatter to COMP. The MEM is dedicated to memory-management tasks, such as memset and memcpy to nodes, to facilitate the dynamic management of sparse matrices, crucial for dynamic large-scale SLAM.

The accelerators are shared across different operators and are applicable to other linear algebra algorithms. The accelerator instances are decoupled from the controller CPU and can be duplicated across the SoC interconnect, allowing the CPU to flexibly virtualize and orchestrate multiple accelerators (see Section 4.3). The disaggregated accelerators and CPU tiles share the last-level cache (LLC).

4.2.1 Compute accelerator.

As most of the numerical operations are GEMM (Figure 3), we build the compute accelerator (COMP) on top of a general matrix accelerator, depicted in Figure 6 as the Compute Accelerator Tile. The transposer allows either of the matrix operands to be transposed. The local scratchpad memory performs double-buffering to minimize load and store overhead. The programmable scaler units scale the operands when loading into the local scratchpad.

COMP also implements the Sparse Index Unroller (SIU) for efficient small block matrix scatter, used in Hessian constructions and merge. SIU calculates the operand and output sparse block addresses by referring to the sparse block indices. Calling the accelerator on each small block addition places a high overhead on the CPU, which may cause the operation to be bottlenecked by the instruction fetch bandwidth. SuperNoVA leverages the SIU to pack multiple block additions into a single instruction, which frees the CPU from computing addresses and fetching instructions for each small block operation sharing consecutive sparse row indices. For block matrix addition, COMP's partial sum accumulator logic is reused to minimize area overhead.

4.2.2 Memory accelerator.

The lightweight memory accelerator (MEM) is shown as the Memory Accelerator Tile in Figure 6. In online SLAM, the elimination tree structure is determined on the fly. As such, the memory space needs to be dynamically allocated and managed during run-time, requiring a substantial number of memory operations. Thus, MEM is crucial for offloading operations such as memcpy and memset from the CPU to solve dynamic problems efficiently. Memcpy is used to prefetch small factors for efficient Hessian construction and to copy the supernodes during Cholesky. The memset acceleration is used for resetting the frontal workspace for each node. The DMA can track multiple in-flight memory requests, out-of-order responses, and burst memory transactions for high memory bandwidth utilization. MEM implements multiple DMA virtual channels (VC) that effectively share the DMA logic and queue. This allows the CPU to offload different memory requests with varying memory access patterns concurrently by configuring each channel differently as if it has access to different DMAs with minimal logic overhead. The VC has configuration registers for source and destination strides, allowing flexible strided memory accesses. The MEM's custom ISA provides fine-grained control over the VC's configurations such as base addresses, strides, and dimensions. The decoder decodes the instruction type and either configures the VC registers or sends load and store requests to the request queue, which pops out when the DMA core is free for a new request. The prober checks the DMA for the status of each VC, so that the CPU can resolve dependencies before fetching dependent compute instructions.

4.2.3 Accelerator integration.

SuperNoVA integrates the accelerator using the remote RoCC (ReRoCC) accelerator interface, which disaggregates the accelerators from the core for scalable virtual integration while maintaining the illusion of core-accelerator tight-coupling to provide ease of programmability [29]. This integration methodology is suitable for SLAM acceleration, as it provides a low-latency interface for accelerator calls for small operations, such as small factor multiplication, and for flexible partitioning of virtualized accelerators to provide both inter- and intra-node parallelism. It is also a low overhead interface, beneficial for area-constrained design. Its microarchitecture blocks are shown as ReRoCC `Client` and `Manager` in Figure 6.

4.3 SuperNoVA Runtime

The SuperNoVA runtime manages multiple accelerators to enable a scalable system for large-scale SLAM. The runtime takes the elimination tree as an input from the SuperNoVA algorithm on each time step. It allocates accelerator resources

Algorithm 2 SuperNoVA accelerator allocation

```
    ▷ Functions: GetParent (gets parent node of the node)
1:      ChildrenDone (check if all child nodes are merged)
    ▷ Lists: Accels (accelerators in the system)
2:      ACQ (acquired virtualized accelerators each thread)
3:      NodeQueue (Supernodes ready to be computed)
4:      MEMspace (Workspace size usage of each thread)
    ▷ Inputs: $T$ (tree), $j$ (supernode), $\omega$ (this thread)
    ▷ Outputs: Decision of accelerator usage
5:  Function calc_space(node):
    ▷ %Computes the workspace usage of the node%
6:      H ← min(total_factor_size, H_workspace_size)
7:      F ← node_height$^2$
8:      next_F ← GetParent(node).node_height$^2$
9:      return (H + F + next_F)
10: while NodeQueue.len > 0 do
11:     $j$ ← NULL
12:     for q in len(NodeQueue) do
13:         $j$ ← NodeQueue[q]
            ▷ % Determine if the selected node fits in LLC%
14:         mem_usage ← sum(MEMSpace)+calc_space($j$)
15:         if mem_usage < Shared LLC size then
16:             NodeQueue.pop(q)
17:             break
18:     if $j$ is NULL then
        ▷ % Release accelerators so other threads can use %
19:         release(ACQ$_\omega$) && ACQ$_\omega$.pop()
20:         Continue
        ▷ % Acquire available accelerators %
21:     acquire(Accels.idle) && ACQ$_\omega$.push(Accels.idle)
22:     runNode($j$) && NodeQueue.popNode($j$)
23:     if ChildrenDone(j.Parent) then
24:         NodeQueue.push(j.Parent)
```

and calls them according to the identified parallelism, while abstracting hardware acceleration and programming complexity from the application thread through virtualization. In addition, it provides an accurate model to estimate the computational cost of each node, which the SuperNoVA algorithm uses to select an appropriate subtree to compute. In order to run in real-time, the runtime system is designed to be lightweight and operate in userspace.

4.3.1 Accelerator resource allocation. As the real-time sensor measurements are unpredictable, the structure of the elimination tree is determined at run-time; thus, computational resources need to be allocated dynamically. Given a subtree to compute, the runtime constructs a ready queue of supernodes on the fly. The runtime identifies and harnesses parallelism in the elimination tree, and allocates hardware resources accordingly. Algorithm 2 describes the procedure for a thread to acquire an available node from the node queue.

The elimination tree has many independent small nodes near the leaves, and less near the root, where branches converge. When processing the leaves, the runtime identifies inter-node parallelism across branches and allocates a different node to each thread to efficiently utilize thread-level parallelism. The runtime constructs a working NodeQueue that exposes a node for scheduling when all its dependent children nodes are processed and merged in. Inter-node parallelism is achieved by allowing each thread to access the NodeQueue for the next ready node (Line 13).

When the node size becomes larger, processing multiple nodes concurrently will cause the active workspace to exceed cacheable memory size. Pursuing maximum inter-node parallelism in this case causes cache thrashing. To avoid this, the runtime allows concurrent node processing up to the point where all the concurrent node's workspace can fit into the shared LLC (Lines 14-17). If there are no available nodes that fit into the remaining LLC space or a lack of parallelizable branches, then the runtime de-schedules the thread, and releases the accelerators to let other threads acquire its accelerators to process bigger nodes close to the root (Lines 18-20). SuperNoVA exploits inter-node parallelism efficiently by partitioning the operations across multiple accelerators.

4.3.2 Heterogeneous accelerator orchestration. The SuperNoVA runtime virtualizes and orchestrates heterogeneous accelerator resources. The memory operations such as workspace initialization and data movement can be exposed as performance overheads if they are not masked under compute operations. The runtime rearranges the accelerator operations strategically, by overlapping memory operations with compute tasks that are independent to the memory operation and offloading the memory tasks to MEM before calling COMP. The parallelism between heterogeneous accelerators perfectly hides the memory latency behind compute operations, increasing the hardware utilization and eliminating performance overhead due to workspace management.

4.3.3 Node cost computation. The SuperNoVA runtime provides an estimated processing time of a supernode, which allows the algorithm to select an appropriate subtree to update at each step, as it abstracts the hardware layer from the algorithm. As SuperNoVA offloads most of the numeric operation to accelerators, the runtime can estimate the node computational cost by considering multi-level memory hierarchy, number of PEs, and node dimensions similar to [28]. By providing insights into the compute overhead, the runtime empowers the algorithm to construct a subtree to meet the SLAM real-time requirement.

5 Methodology

This section details SuperNoVA's implementation, together with the workloads and metrics used for our evaluation.

Table 3. SoC configurations used in the evaluation.

Parameter	Value
# of COMP tiles	1-4
Systolic array dimension (per tile)	4x4
Scratchpad/Accumulator size (per tile)	32KB/16KB
# of MEM tiles	1-4
Virtual channels (per tile)	4
# of CPU tiles	1-4
ReRoCC L2 TLB entries	256
ReRoCC PTW cache size	2KB
Shared L2 (size / banks)	4MB, 8
DRAM bandwidth	64GB/s
Frequency	1GHz

5.1 SuperNoVA Implementation

We implement the SuperNoVA hardware architecture using the Chisel HDL [6]. We build COMP on top of the Gemmini [18] infrastructure, which is a systolic-array-based GEMM accelerator, by adding a sparse index unroller (SIU). To generate the SoC, we use the Chipyard [3] SoC framework, an open-source framework for designing and evaluating SoCs. We evaluate SuperNoVA's performance by running our SLAM backend workloads on FireSim, a cycle-exact, FPGA-accelerated, RTL simulator [27]. We also synthesize SuperNoVA hardware on a commercial 16nm technology process for area and power evaluation.

Table 3 shows the SoC configuration used in the evaluations. We demonstrate SuperNoVA with different numbers of accelerator sets (COMP+MEM), 1, 2, and 4. Each COMP and MEM accelerator is individually integrated to a ReRoCC `Manager`, and multiple sets of accelerators can be instantiated on the same SoC. Each COMP is equipped with a floating point 32-bit precision 4x4 weight-stationary systolic array. Each MEM has 4 DMA VCs, and it can track 8 in-flight burst transactions. All the tiles share the LLC memory subsystem.

The SuperNoVA algorithm and runtime are implemented in C/C++ and run on top of a full Linux stack. The runtime uses a lightweight software look-up table to track which thread is active and the current accelerator allocation. The runtime also implements the supernode queues, which track the supernodes that are ready to be processed.

5.2 Workloads

SuperNoVA is evaluated on large-scale pose graph datasets:

1. **M3500**: 3.5K steps, 5453 edges
2. **Sphere**: 2K steps, 3951 edges
3. **CAB1**: 464 steps, 2287 edges, 1800 m^2 range
4. **CAB2**: 3K steps, 15144 edges, 6000 m^2 range

The Sphere dataset (Figure 7a) and M3500 are synthetic pose graph datasets with many loop closures [12]. We choose large-scale synthetic datasets with different sparsity characteristics to show SuperNoVA's effectiveness across different environments. Sphere is a dense dataset with high rotational noise and larger supernodes, while M3500 is a sparse dataset with many small supernodes. The CAB datasets (Figure 7b) are part of the LaMAR, a large-scale dataset collected by AR devices in diverse environments [46]. CAB2 is constructed by concatenating multiple AR sessions to form an extremely long pose trajectory, where the factors between poses are determined by the covisibility of a common landmark. To simulate a system running online SLAM, a new pose is added at each step, along with all the associated factors.

5.3 Metrics

To showcase the resource- and target-aware execution of SuperNoVA, we measure the latency at each step of the workload to demonstrate that SuperNoVA meets real-time SLAM requirements. We also evaluate the accuracy of the estimated trajectory and show that SuperNoVA's is capable of maintaining accuracy over a long duration.

Latency. We measure the latency of the full SLAM backend including relinearization, symbolic factorization, and the subtree selection algorithm for RA-ISAM2, on top of the numeric factorization, and compare it to the target processing latency for each step. To match the camera frame rate, we set the target processing rate to 30FPS, which implies a latency target of 33.3ms.

Accuracy. We use the Python package evo to find the maximum translation error (MAX) and the root mean square error (RMSE) of an estimated trajectory X against a reference trajectory X_{ref} [20]. In online SLAM, it is important to measure the error at each timestep, not just over the entire trajectory. Therefore, we also measure the incremental RMSE (iRMSE) [35] , where

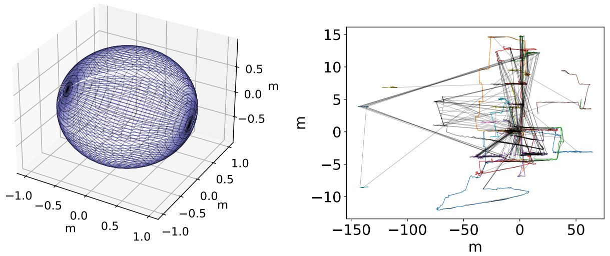
$$\text{iRMSE} = \sum_k \left[\frac{k}{\sum_k} \text{RMSE}(X^{(k)}, X_{ref}^{(k)}) \right] \tag{3}$$

The reference trajectories are obtained by optimizing reprojection error until convergence at each step.

5.4 Hardware Baselines

To evaluate the efficiency of SuperNoVA hardware acceleration, we compare its latency against six hardware baseline platforms running the same incremental SLAM (ISAM2).

1. **BOOM**: An out-of-order (OoO) superscalar RISC-V core with performance comparable to ARM Cortex A72-like cores [53]. The system setup is the same as SuperNoVA (memory subsystem, frequency, etc.).
2. **Mobile CPU**: ARM Cortex A72 core [4] (1.5GHz) on a commercial Raspberry Pi4 [43].
3. **Mobile DSP**: Neon SIMD unit [5] with Mobile CPU
4. **Server CPU**: Server class Intel Xeon E5-2643 processor (3.5GHz).



(a) Sphere

(b) CAB2

Figure 7. Groundtruth trajectories of our datasets. The thin black lines indicate factors between poses.

5. **Embedded GPU**: NVIDIA Maxwell GPU [40] on the commercial Jetson Nano [39], using the cuSparse and cuSolver libraries [37, 38] to run an incremental solver.
6. **Spatula** [16]: Prior work that accelerates static matrix factorization. Uses vanilla matrix accelerators. The system setup is the same as SuperNoVA (CPU, memory subsystem, frequency, etc.).

As shown in Section 6.4, one BOOM has a comparable area to two Rocket CPUs and two sets of SuperNoVA accelerators (COMP + MEM). Thus, we use 2 sets of SuperNoVA accelerators to show the efficiency of the SuperNoVA hardware compared to the baselines on Section 6.1.

To show how SuperNoVA's resource-aware algorithm accommodates different resource availabilities on Section 6.2, we compare the algorithm for different numbers of SuperNoVA accelerator sets, 1/2/4 sets, to the incremental baseline running on the same hardware and runtime.

5.5 Algorithm Baselines

To evaluate the efficiency of SuperNoVA's resource-aware algorithm, we compare against the following baselines:

1. **Local**: We use VIO, a fixed-lag smoother with a window size of 20 [24]. Factors outside of the sliding window are discarded. The oldest pose is marginalized.
2. **Local + Global**: This represents a multi-level SLAM system with a local solve and an LC solver. The LC solver only runs when a loop is detected. When the LC solver is done, its estimate is used to correct the local solver.
3. **Incremental**: ISAM2 backend with a fixed relinearization threshold. A well-known optimization for this is to take one Gauss-Newton step in each backend iteration, instead of iterating until convergence [44].

To evaluate the effectiveness of RA-ISAM2 and its ability to accommodate varying levels of resource availability, we compare the accuracy of the resource-aware algorithm (RA) across different configurations: using 1, 2, and 4 SuperNoVA accelerator sets (RA1S, RA2S, and RA4S). These results are compared against baseline configurations, as well as across the different RA setups, to demonstrate how the algorithm scales with available resources while maintaining accuracy.

Furthermore, we conduct an ablation study by replacing SuperNoVA hardware with a server CPU (RACPU). This allows us to isolate the performance impact of SuperNoVA's specialized accelerators and evaluate how the resource-aware algorithm performs when relying on general-purpose hardware, providing insight into the benefits of the hardware-accelerator co-design.

6 Evaluation

In this section, we evaluate the ability of SuperNoVA to process the full SLAM backend in real-time by comparing it to different compute platforms and existing SLAM methods. The first part of the evaluation shows that SuperNoVA hardware improves SLAM processing latency over existing compute platforms. The second part demonstrates that through co-designing algorithm and hardware, SuperNoVA achieves better accuracy than existing SLAM backends, while guaranteeing to meet target processing latency.

6.1 Latency Analysis

We first evaluate the latency to show SuperNoVA's efficacy in accelerating SLAM backend, by comparing its processing latency with the existing hardware platforms when processing the same incremental baseline. In this evaluation, SuperNoVA hardware refers to SuperNoVA hardware and runtime.

SuperNoVA hardware accelerates SLAM backend better than CPU baselines. Figure 8 demonstrates that SuperNoVA's acceleration outperforms the baseline compute platforms. Compared to BOOM, SuperNoVA hardware achieves 90.7%/54.4%/93.3%/89.5% (Sphere/M3500/CAB1/CAB2) latency reduction for end-to-end SLAM backend (Total). SuperNoVA consistently outperforms commercial Mobile CPU to a similar degree as well. Even compared to server-class CPU, SuperNoVA hardware achieves better processing latency. For the numeric part (Numeric), SuperNoVA hardware achieves 87.4%/1.8%/81.4%/76.8% latency reduction compared to the server CPU. The lower speedup on M3500 can be explained by the higher sparsity and smaller supernode sizes due to the graph structure of the dataset, leading to decreased accelerator utilization while a powerful server CPU is less impacted. In terms of end-to-end latency, SuperNoVA hardware shows 65.5% (Sphere), 63.9% (CAB1), 43.9% (CAB2) reduction, while for M3500, SuperNoVA's latency is 3× compared to server CPU. M3500 has a high number of relinearized variables per step, resulting in a higher relinearization cost on an in-order CPU. However, AR datasets, CAB1 and CAB2, and Sphere, which is a dense dataset with a lot of big LCs, show significant improvement across all class CPUs.

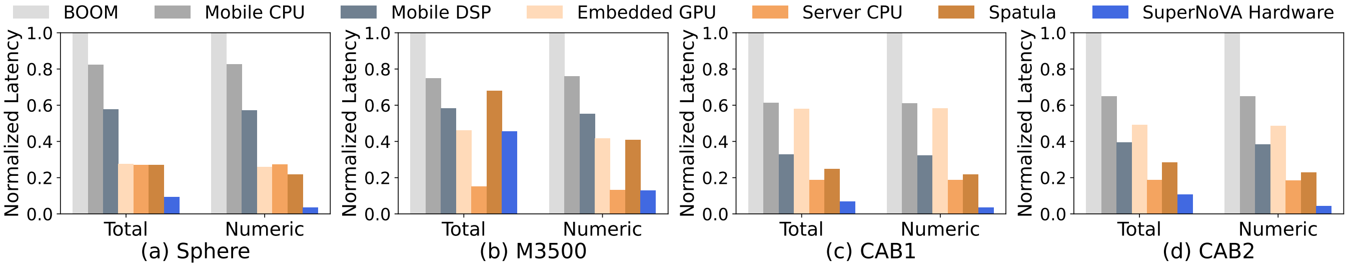


Figure 8. Latency comparison of 2 sets of SuperNoVA hardware with the baselines on 4 different benchmarks running incremental baseline (ISAM2). The Y-axis is normalized by BOOM's latency.

SuperNoVA hardware is more efficient in accelerating the numeric than other compute units. Compared to Mobile DSP, SuperNoVA hardware shows 83.9%/21.7%/79.5%/73.4% (Sphere/M3500/CAB1/CAB2) latency reduction in total, and 94%/76.9%/89.3%/88.8% for numeric. SuperNoVA's matrix-based compute engine is more powerful than SIMD when processing numeric operations as most of the operation can be mapped to BLAS-3. The efficiency is more pronounced in denser and larger problems, Sphere and CAB2. Compared to Embedded GPU, SuperNoVA achieves 66.2%/0.9%/88.4%/78.6% latency reduction in total, and 86.7%/69.4%/94.1%/91.2% for numeric. GPU particularly performs poorly on CAB1, in that it shows similar latency to Mobile CPU and 1.77× longer latency than Mobile DSP in total. This is due to the initial memory load cost being more pronounced in small problems, which is not the case in SuperNoVA hardware as its compute accelerator efficiently interleaves computational and memory load to its internal scratchpad memory. In addition, COMP's matrix-based computation unit is more efficient than GPU for highly GEMM-based Cholesky operation.

SuperNoVA's algorithm-aware hardware co-design is efficient. Compared to Spatula baseline, SuperNoVA hardware achieves 84.2% (Sphere), 68.6% (M3500), 84.2% (CAB1), 81.2% (CAB2) latency reduction for numeric. This improvement indicates the significance of algorithm-aware accelerator co-design. Spatula can accelerate the matrix factorization efficiently using its GEMM accelerator, but it is missing optimizations for other key components of the end-to-end dynamic operation, such as memory management operations and Hessian construction. As SLAM is a dynamic problem that the pose graph of each step is decided during run-time, each supernode's memory space and workspace needs to be dynamically managed. SuperNoVA addresses these challenges through its dedicated MEM for memory management tasks and the sparse indexing unit integrated to COMP. These specialized hardware components lead to substantial performance gains over Spatula's GEMM-only approach. The gain is more pronounced in the datasets with larger nodes, which require long latency memory management operation, thus showing higher improvement on Sphere and CAB benchmarks than M3500.

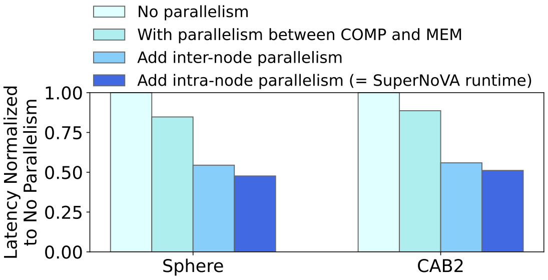


Figure 9. Latency improvement by enabling SuperNoVA runtime parallelism.

SuperNoVA runtime's parallelism optimizations enable scalable multi-accelerator approach. Figure 9 shows the numeric's latency improvement by gradually enabling SuperNoVA's runtime optimizations on the Sphere and CAB2 datasets. Enabling parallelism between different accelerators, COMP and MEM, achieves 15.3%/11.4% (Sphere/CAB2) reduction from single-threaded no parallelism case. The larger supernode sizes in Sphere allow more overlapping of the memory management overhead through heterogeneous accelerator orchestration. Enabling inter-node parallelism decreases the latency by 35.8%/37% (Sphere/CAB2), by processing the branches of the elimination tree in parallel. Finally, enabling intra-node parallelism brings in a further 12.5%/8.6% (Sphere/CAB2) latency reduction. The improvement comes from parallelizing within large nodes near the root of the tree where there are few branches. This speedup is more pronounced for Sphere due to the larger node sizes.

6.2 Real-time Analysis

We evaluate SuperNoVA's resource-aware relinearization variable selection, by comparing SuperNoVA's RA-ISAM2 and the incremental algorithm running on same SuperNoVA hardware and runtime. We use 3 different SoC configurations by varying the accelerator resources, 1/2/4 sets, to show SuperNoVA's adaptiveness under different resource constraints.

SuperNoVA's resource aware algorithm improves the target satisfaction rate of SLAM backend. Figure 10 demonstrates the latency comparison between SuperNoVA

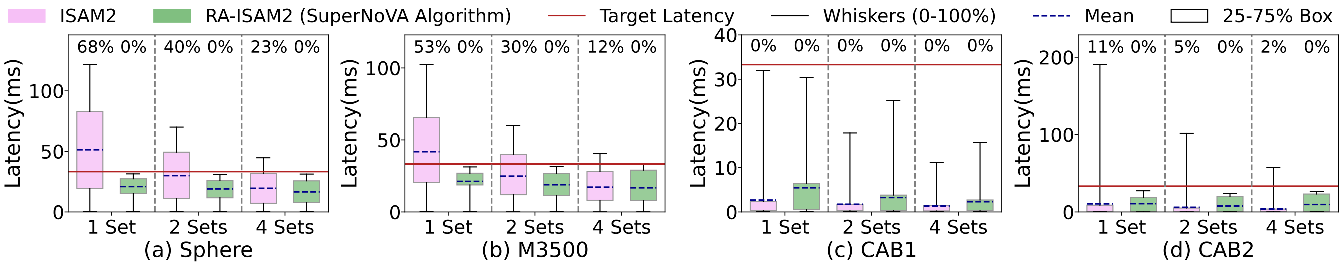
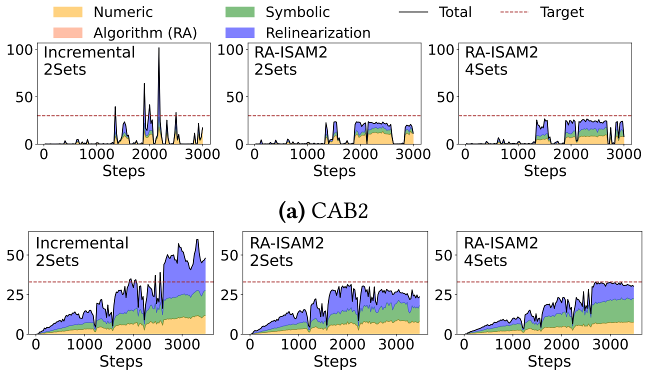


Figure 10. Latency comparison of incremental baseline (ISAM2) and SuperNoVA algorithm (RA-ISAM2) running on SuperNoVA hardware+runtime. "1/2/4 sets" stands for the number of accelerator sets. The percentage above the box means target miss rate.



(a) CAB2

(b) M3500

Figure 11. Execution time breakdown of end-to-end backend latency. (2/4Sets: 2/4 sets of SuperNoVA accelerator)

and the incremental baseline running on the same SuperNoVA SoC. All steps, including outliers, are included to provide a comprehensive view of performance variability. Resource-aware SuperNoVA consistently meets the 33.3ms target required for real-time processing under different resource availabilities across all benchmarks, whereas the incremental baseline fails to meet the deadline by 68%, 40%, 23% (1/2/4 sets) for Sphere, 53%, 30%, 12% for M3500 and 12%, 6%, 3% for CAB2. This demonstrates SuperNoVA algorithm's ability to select a subgraph to update that fits within the target deadline. Both incremental and SuperNoVA algorithms always meet the timing for CAB1. However, the average latency has increased for SuperNoVA as its algorithm will select more relinearization variables than the baseline when latency allows, which benefits the accuracy as Table 4 shows.

SuperNoVA effectively selects variables to relinearize based on available resources. Figure 11 breaks down the latency of the incremental baseline and SuperNoVA algorithm into relinearization, symbolic, numeric, and SuperNoVA algorithm overhead, for CAB2 and M3500. For CAB2, incremental shows high latency spikes in some of the steps that have LC events, which leads to relinearizing and refactorizing most of the elimination tree. However, SuperNoVA's resource-aware algorithm amortizes the LC cost across multiple steps, which helps meeting the target deadline while making sure the important updates are done in time. Similarly, for M3500, SuperNoVA algorithm dynamically selects the important variables to relinearize, which greatly reduces the relinearization cost. In both datasets, the increase from 2 accelerator sets to 4 increases the symbolic costs as the selected subtree size increases with more compute resources. This, in turn, boosts the accuracy (Section 6.3). Despite this increase in computational load, RA-ISAM2 maintains similar latency close to the target by dynamically adjusting the workload based on the available hardware resources. It always performs the maximum possible work within the target latency when hardware resources are constrained. Overall, SuperNoVA's variable selection algorithm imposes minimal overhead of 0.1%/0.9% (M3500/CAB2) on average which shows SuperNoVA algorithm is scalable to large-scale problems.

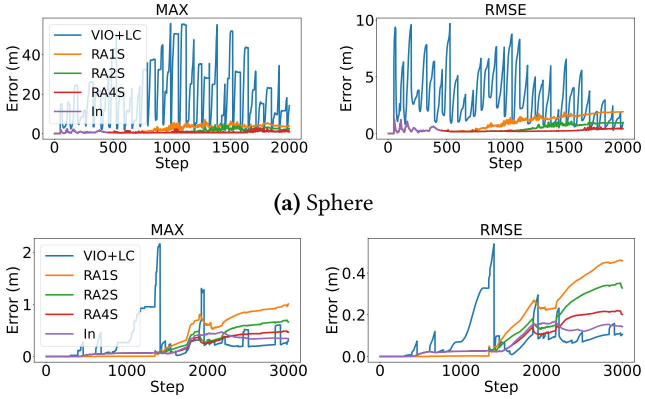
6.3 Accuracy Analysis

We run the dataset on each SLAM algorithm and measure the maximum pose error and RMSE at each step. We use 3 different existing SLAM configurations, Local, Local+Global, and incremental (In), and compare them against our proposed resource-aware algorithm (RA) with 3 SoC configurations, 1, 2, 4 SuperNoVA accelerator sets (RA1S, RA2S, RA4S). We also conduct a hardware ablation study with server CPU (RACPU). The summary of the results is shown in Table 4 while the errors for each step are shown in Figure 12. The incremental baseline at each step represents the idealized SuperNoVA algorithm with infinite compute.

The SuperNoVA algorithm achieves high accuracy despite the latency and resource constraints, demonstrating viability for large-scale AR applications. The resource-aware methods generally outperform the baseline SLAM methods for both MAX and iRMSE. Compared to the Local+Global baseline, RA4S reduces the MAX and the iRMSE by 95% and 89% for Sphere, 84% and 90% and 45% for M3500, 96% and 95% for CAB1. For CAB2, RA4S reduces the MAX by 77% and is very close to both Local+Global and the ideal incremental baseline for iRMSE. Figure 12 further illustrates

Table 4. Accuracy comparisons of the baseline algorithms (Local, Local+Global, In) and the resource-aware algorithms (RA) with different hardware configurations (CPU, 1/2/4S). Upper row is MAX, lower row is iRMSE. Error unit: meter

	Local	Local+ Global	RA CPU	RA1S (Ours)	RA2S (Ours)	RA4S (Ours)	In
Sphere	213.62	55.79	59.98	7.15	5.04	**2.86**	2.36
	95.64	2.72	10.48	1.28	0.61	**0.29**	0.37
M3500	97.05	18.46	1.87	1.98	1.87	**1.86**	0.22
	43.13	1.59	0.89	0.91	0.89	**0.87**	0.06
CAB1	0.27	6.11e-2	1.76e-3	**1.51e-3**	2.00e-3	2.25e-3	1.49e-2
	2.70e-2	4.31e-3	1.97e-4	**1.46e-4**	2.17e-4	2.04e-4	3.07e-3
CAB2	4.77	2.16	2.69	1.01	0.70	**0.49**	0.47
	1.36	**0.11**	0.72	0.25	0.18	0.13	0.10



(a) Sphere

(b) CAB2

Figure 12. The maximum and RMS error at each step compared to a fully optimized reference trajectory.

this effect. A sliding window Local will ignore loop closure (LC) events to maintain a fixed latency, inducing a large drift over time. Even though Global's LC solver can correct the error, the LC latency causes the correction to lag behind the error spike. By the time the Global unit fully optimizes the error, the system may have already entered an unrecoverable state of failure. Compared to the RA1S and RA2S, RA4S shows 77% and 52% reduction in iRMSE for Sphere (60%, 43% in MAX) and 48% and 28% reduction for CAB2 (52%, 30% in MAX), demonstrating the scalability of our approach. While SuperNoVA successfully maintains accuracy by amortizing LC costs, its scalability is not infinite. Section 7 further discusses this limitation.

SuperNoVA accelerator enhances accuracy, highlighting the importance of its algorithm-hardware co-design. We performed an ablation study comparing RACPU with SuperNoVA hardware to evaluate the impact of hardware acceleration on SLAM accuracy. While the resource-aware algorithm guarantees meeting the target deadline, the CPU's limited compute capabilities lead to lower accuracies. In contrast, SuperNoVA's powerful hardware acceleration allows for more updated variables within the target latency, resulting in significantly reduced errors. For the Sphere dataset, RACPU experiences a 11.9× and 21× increase in MAX, and a 17.2× and 36.1× increase in iRMSE compared to RA2S and RA4S respectively. Similarly, for CAB2, RACPU exhibits 3.8× and 5.5× higher MAX and 4× and 5.5× higher iRMSE than RA2S and RA4S respectively. M3500, however, is an exception as it is an edge-case benchmark with high relinearization costs. These comparisons highlight the crucial role of SuperNoVA hardware acceleration in achieving superior accuracy, demonstrating the necessity of SuperNoVA's algorithm-hardware co-design for large-scale SLAM tasks.

Table 5. Area breakdown of SuperNoVA hardware.

Component	**Area (μm^2)**	**% of Area**
Rocket CPU tile	151K	100%
COMP tile	301K	100%
ReRoCC Manager	20K	6.6%
Accelerator	281K	93.4%
Mesh	92K	30.6%
Scratchpad+Accumulator	86K	28.6%
Sparse Index Unit	9K	3.1%
MEM tile	51K	100%
ReRoCC Manager	20K	39.2%
Accelerator	31K	60.8%
Total (CPU tile+Accelerator tiles)	504K	40%
BOOM baseline	1262K	100%

6.4 Physical Design and Area Analysis

We synthesize SuperNoVA hardware, with each SuperNoVA accelerator integrated with ReRoCC `Manager` and Rocket CPU integrated with ReRoCC `Client` [30] using Cadence Genus with commercial 16nm process technology with the configuration in Table 3. As shown in Table 5, the total area, adding CPU, COMP and MEM tiles is 40% of BOOM's area. Thus, having two SuperNoVA accelerator sets with two CPUs equals 80% of a single BOOM. This shows that SuperNoVA hardware is an area-efficient solution for SLAM acceleration, in addition to the benefit of real-time latency performance.

6.5 Power Analysis

We evaluated the power consumption of SuperNoVA using Cadence Joules and observed that it consumes 114mW during its most power-intensive operation, the symmetric rank-k update, on the Intel16 process, running at 1GHz and 0.8V. This power consumption is significantly lower than the 5-10W consumed by embedded GPUs and the 2.5-5W consumed by FPGA-based accelerators [34]. These results highlight SuperNoVA's efficiency in performing SLAM computations while maintaining low power usage, making it highly suitable for resource-constrained environments such as AR/VR devices.

7 Future Work

Although SuperNoVA efficiently solves large-scale SLAM problems by distributing a large number of relinearization variables across multiple steps to amortize LC costs and preserve accuracy, its scalability is not infinite, as demonstrated in the later steps of Figure 12b. When the history size grows too large, updating variables deep in the history can lead to timing violations. When this happens, SuperNoVA is forced to exclude those variables, effectively "dropping" older sensor measurements to meet timing constraints, which results in a trade-off between accuracy and real-time performance. A potential direction to address this limitation would be to supplement SuperNoVA with an LC module running on a base station or as a background process to handle older variables, similar to the VIO+LC baseline. This hybrid approach would allow SuperNoVA to maintain its low latency and high accuracy for real-time tasks on-device while offloading deeper historical updates, thus further scaling the problem.

Additionally, this work does not yet optimize for power constraints. However, the SuperNoVA algorithm could be extended by integrating an energy cost model into the SuperNoVA runtime, enabling an energy-aware SLAM system.

Finally, SuperNoVA hardware is broadly applicable beyond SLAM, particularly for applications that require floating-point GEMM, such as those in robotics or vision processing. The resource-aware algorithm can be applied to any factor-graph-based optimization problem, including control systems and other domains requiring similar optimization approaches.

8 Conclusion

We introduce SuperNoVA, a full-stack algorithm-hardware co-designed SLAM system, to address the challenges of state estimation in AR/VR applications. By adaptively selecting updated subtrees, virtualizing hardware resources, and optimizing hardware architecture, SuperNoVA enables high-accuracy, real-time, large-scale SLAM processing in resource-constrained settings. Our evaluation on large-scale AR datasets shows that SuperNoVA achieves SLAM backend latency reduction by 89.5% for the CPU baseline and 78.6% for the GPU baseline, and reduce the maximum pose error by 77% over existing SLAM solutions, while guaranteed to meet the 33.3 ms latency target.

Acknowledgments

We thank our anonymous reviewers and shepherd, Sabrina Neuman, for the valuable feedback. This research was, in part, funded by the NSF Award CCF-2238346, NSF CCF-1955450, NSF CCRI-2016662 and NSF Award POSE 2303735, and in part by SLICE Lab industrial sponsors and affiliates. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

References

[1] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. Ceres Solver, 3 2022.

[2] Stefano Aldegheri, Nicola Bombieri, Domenico D. Bloisi, and Alessandro Farinelli. Data flow orb-slam for real-time performance on embedded gpu boards. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5370–5375, 2019.

[3] Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, Paul Rigge, Colin Schmidt, John Wright, Jerry Zhao, Yakun Sophia Shao, Krste Asanović, and Borivoje Nikolić. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4):10–21, 2020.

[4] ARM. *ARM Cortex-A72.* Accessed: 2023-12-01.

[5] ARM. *ARM NEON SIMD.* Accessed: 2023-12-01.

[6] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. Chisel: Constructing hardware in a scala embedded language. In *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, page 1216–1225, New York, NY, USA, 2012. Association for Computing Machinery.

[7] Armand Behroozi, Yuxiang Chen, Vlad Fruchter, Subramanian, Lavanya Subramanian, Sriseshan Srikanth, and Scott Mahlek. Slimslam: An adaptive runtime for visual-inertial simultaneous localization and mapping. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '24, 2024.

[8] Michael Bloesch, Sammy Omari, Marco Hutter, and Roland Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 298–304. IEEE, 2015.

[9] 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*, 2016.

[10] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *Trans. Rob.*, page 1309–1332, 2016.

[11] Carlos Campos, Richard Elvira, Juan J. Gómez Rodríguez, José M. M. 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.

[12] Luca Carlone, Roberto Tron, Kostas Daniilidis, and Frank Dellaert. Initialization techniques for 3d slam: A survey on rotation estimation and its use in pose graph optimization. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 4597–4604. IEEE, 2015.

[13] NVIDIA corporation. cuvslam, 2023.

[14] Frank Dellaert and GTSAM Contributors. borglab/gtsam, May 2022.

[15] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear. *ACM Trans. Math. Softw.*, 9(3):302–325, sep 1983.

[16] Axel Feldmann and Daniel Sanchez. Spatula: A hardware accelerator for sparse matrix factorization. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 91–104, 2023.

[17] Y. Gan, Y. Bo, B. Tian, L. Xu, W. Hu, S. Liu, Q. Liu, Y. Zhang, J. Tang, and Y. Zhu. Eudoxus: Characterizing and accelerating localization in autonomous machines industry track paper. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 827–840, 2021.

[18] Hasan Genc, Seah Kim, Alon Amid, Ameer Haj-Ali, Vighnesh Iyer, Pranav Prakash, Jerry Zhao, Daniel Grubb, Harrison Liew, Howard

Mao, Albert Ou, Colin Schmidt, Samuel Steffl, John Wright, Ion Stoica, Jonathan Ragan-Kelley, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration. In *Design Automation Conference*, pages 769–774, 2021.

[19] 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)*, pages 9–13, 2011.

[20] Michael Grupp. evo: Python package for the evaluation of odometry and slam. https://github.com/MichaelGrupp/evo, 2017.

[21] Ramyad Hadidi, Bahar Asgari, Sam Jijina, Adriana Amyette, Nima Shoghi, and Hyesoon Kim. Quantifying the design-space tradeoffs in autonomous drones. ASPLOS '21, page 661–673, 2021.

[22] Yuhui Hao, Yiming Gan, Bo Yu, Qiang Liu, Yinhe Han, Zishen Wan, and Shaoshan Liu. Orianna: An accelerator generation framework for optimization-based robotic applications. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, page 813–829, 2024.

[23] Joel Hesch, Anna Kozminski, and Oskar Linde. Powered by ai: Oculus insight, 2019.

[24] Guoquan P Huang, Anastasios I Mourikis, and Stergios I Roumeliotis. An observability-constrained sliding window filter for slam. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 65–72. IEEE, 2011.

[25] Muhammad Huzaifa, Rishi Desai, Samuel Grayson, Xutao Jiang, Ying Jing, Jae Lee, Fang Lu, Yihan Pang, Joseph Ravichandran, Finn Sinclair, Boyuan Tian, Hengzhi Yuan, Jeffrey Zhang, and Sarita V. Adve. Exploring extended reality with illixr: A new playground for architecture research, 2021.

[26] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

[27] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra, Q. Huang, K. Kovacs, B. Nikolic, R. Katz, J. Bachrach, and K. Asanovic. Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 29–42, 2018.

[28] Seah Kim, Hasan Genc, Vadim Vadimovich Nikiforov, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao. Moca: Memory-centric, adaptive execution for multi-tenant deep neural networks. In *International Symposium on High-Performance Computer Architecture (HPCA)*, pages 828–841, 2023.

[29] Seah Kim, Jerry Zhao, Krste Asanovic, Borivoje Nikolic, and Yakun Sophia Shao. Aurora: Virtualized accelerator orchestration for multi-tenant workloads. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '23, page 62–76, 2023.

[30] Seah Kim, Jerry Zhao, Krste Asanović, Borivoje Nikolić, and Yakun Sophia Shao. Aurora: A full-stack solution for scalable and virtualized accelerator integration. *IEEE Micro*, 44(4):97–105, 2024.

[31] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *IROS*, volume 3, pages 1442–1447, 1991.

[32] Ziyun Li, Yu Chen, Luyao Gong, Lu Liu, Dennis Sylvester, David Blaauw, and Hun-Seok Kim. An 879gops 243mw 80fps vga fully visual cnn-slam processor for wide-range autonomous exploration. In *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*, pages 134–136, 2019.

[33] Joseph W Liu. A compact row storage scheme for cholesky factors using elimination trees. *ACM Transactions on Mathematical Software (TOMS)*, 12(2):127–148, 1986.

[34] Weizhuang Liu, Bo Yu, Yiming Gan, Qiang Liu, Jie Tang, Shaoshan Liu, and Yuhao Zhu. Archytas: A framework for synthesizing and dynamically optimizing accelerators for robotic localization. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, page 479–493, 2021.

[35] Daniel McGann, John G. Rogers, and Michael Kaess. Robust incremental smoothing and mapping (risam). In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4157–4163, 2023.

[36] Daniel Medeiros, Eduardo Cordeiro, Daniel Mendes, Maurício Sousa, Alberto Raposo, Alfredo Ferreira, and Joaquim Jorge. Effects of speed and transitions on target-based travel techniques. In *Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology*, VRST '16, page 327–328, New York, NY, USA, 2016. Association for Computing Machinery.

[37] Nvidia. *cuSOLVER*. Accessed: 2023-12-01.

[38] Nvidia. *cuSPARSE*. Accessed: 2023-12-01.

[39] Nvidia. *Jetson Nano*. Accessed: 2023-12-01.

[40] Nvidia. *Nvidia Maxwell Architecture*. Accessed: 2023-12-01.

[41] NVIDIA-ISAAC-ROS. Isaac ros visual slam.

[42] Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. A 64-mw dnn-based visual navigation engine for autonomous nano-drones. *IEEE Internet of Things Journal*, pages 8357–8371, 2019.

[43] Raspberry Pi Foundation. *Raspberry Pi 4 Model B Specifications*. Accessed: 2023-12-01.

[44] David M Rosen, Michael Kaess, and John J Leonard. Rise: An incremental trust-region method for robust online sparse least-squares estimation. *IEEE Transactions on Robotics*, 30(5):1091–1108, 2014.

[45] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1689–1696. IEEE, 2020.

[46] 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 *ECCV*, 2022.

[47] Xuesong Shi, Dongjiang Li, Pengpeng Zhao, Qinbin Tian, Yuxin Tian, Qiwei Long, Chunhao Zhu, Jingwei Song, Fei Qiao, Le Song, Yangquan Guo, Zhigang Wang, Yimin Zhang, Baoxing Qin, Wei Yang, Fangshi Wang, Rosa H. M. Chan, and Qi She. Are we ready for service robots? the openloris-scene datasets for lifelong slam. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3139–3145, 2019.

[48] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. Navion: A fully integrated energy-efficient visual-inertial odometry accelerator for autonomous navigation of nano drones. In *2018 IEEE Symposium on VLSI Circuits*, pages 133–134, 2018.

[49] Chen Wang, Dasong Gao, Kuan Xu, Junyi Geng, Yaoyu Hu, Yuheng Qiu, Bowen Li, Fan Yang, Brady Moon, Abhinav Pandey, Aryan, Jiahe Xu, Tianhao Wu, Haonan He, Daning Huang, Zhongqiang Ren, Shibo Zhao, Taimeng Fu, Pranay Reddy, Xiao Lin, Wenshan Wang, Jingnan Shi, Rajat Talak, Kun Cao, Yi Du, Han Wang, Huai Yu, Shanzhao Wang, Siyu Chen, Ananth Kashyap, Rohan Bandaru, Karthik Dantu, Jiajun Wu, Lihua Xie, Luca Carlone, Marco Hutter, and Sebastian Scherer. PyPose: A library for robot learning with physics-based optimization. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.

[50] Jian Weng, Sihao Liu, Zhengrong Wang, Vidushi Dadu, and Tony Nowatzki. A hybrid systolic-dataflow architecture for inductive matrix algorithms. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 703–716, 2020.

[51] Zhengdong Zhang, Amr Suleiman, Luca Carlone, Vivienne Sze, and Sertac Karaman. Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach. In *Robotics: Science and Systems*,

2017.

[52] Zhengyou Zhang and Ying Shan. Incremental motion estimation through local bundle adjustment. 2001.

[53] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. Sonicboom: The 3rd generation berkeley out-of-order machine. May 2020.