# RoboShape: Using Topology Patterns to Scalably and Flexibly Deploy Accelerators Across Robots

Sabrina M. Neuman*
sneuman@seas.harvard.edu
Harvard University
Massachusetts, USA

Radhika Ghosal*
rghosal@g.harvard.edu
Harvard University
Massachusetts, USA

Thomas Bourgeat
bthom@csail.mit.edu
Massachusetts Institute of Technology
Massachusetts, USA

Brian Plancher
bplancher@barnard.edu
Barnard College, Columbia University
New York, USA

Vijay Janapa Reddi
vj@eecs.harvard.edu
Harvard University
Massachusetts, USA

## ABSTRACT

A key challenge for hardware acceleration of robotics applications is the enormous diversity of possible deployment scenarios. To create efficient accelerators while minimizing non-recurring engineering costs, it is essential to identify high-level computational patterns that are prescribed by the physical characteristics of the deployed robot system and directly embed these domain-specific insights into the accelerator design process. To address this challenge, we present RoboShape, an accelerator framework that leverages two topology-based computational patterns that scale with robot size: (1) topology traversals, and (2) large topology-based matrices. Using these patterns and building on prior work, we expose opportunities to directly use robot topology to inform architectural mechanisms including task scheduling and allocation, data placement, block matrix operations, and sparse I/O data. Designing architectures according to topology-based patterns enables flexible, scalable, optimized accelerator deployment across the nonlinear design space of robot shape and computing resources. With this insight, we establish a systematic framework to generate accelerators, and use it to implement three accelerators for three different robots, achieving speedups over state-of-the-art CPU and GPU solutions. For the topologically-diverse iiwa manipulator, HyQ quadruped, and Baxter torso robots, RoboShape accelerators on an FPGA provide a 4.0× to 4.4× speedup in compute latency over CPU and a 8.0× to 15.1× speedup over GPU for the dynamics gradients, a key bottleneck preventing online execution of nonlinear optimal motion control for legged robots. Taking a broader view, for topology-based applications, RoboShape enables analysis of performance and resource utilization tradeoffs that will be critical to managing resources across accelerators in future full robotics domain-specific SoCs.

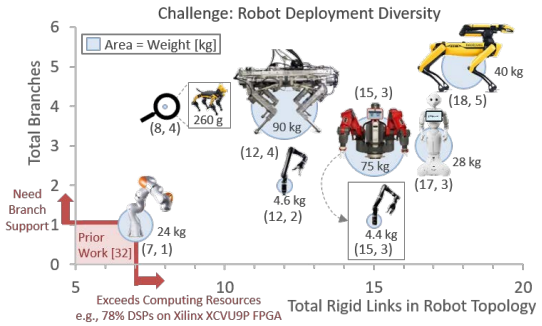


**Figure 1: RoboShape leverages topology-based computational patterns that scale with robot shape and size to flexibly deploy accelerators across diverse robots (e.g., [4, 5, 14, 25, 40, 42, 49]) and computing resource constraints.**

## CCS CONCEPTS

• **Hardware** → **Hardware accelerators**; • **Computer systems organization** → **Robotics**.

## KEYWORDS

robotics, hardware accelerators, dynamics, motion planning

**ACM Reference Format:**
Sabrina M. Neuman, Radhika Ghosal, Thomas Bourgeat, Brian Plancher, and Vijay Janapa Reddi. 2023. RoboShape: Using Topology Patterns to Scalably and Flexibly Deploy Accelerators Across Robots. In *Proceedings of the 50th Annual International Symposium on Computer Architecture (ISCA '23), June 17–21, 2023, Orlando, FL, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3579371.3589104

## 1 INTRODUCTION

In domains that interact heavily with the physical world, such as robotics, the internet of things (IoT), augmented and virtual reality systems, wearable devices, and healthcare monitoring technologies, a key challenge to domain-specific hardware acceleration is the enormous diversity of possible deployment scenarios. Hardware acceleration is a promising solution to provide high-performance and energy-efficient computation, but design requirements vary with the physical characteristics of the target electrical, biological, or mechanical system (e.g., the physical configuration and structure of a robot's limbs), as well as the performance demands of the task-level application (e.g., fast control for dynamic motion through

*Both authors contributed equally to this research.


*ISCA '23, June 17–21, 2023, Orlando, FL, USA*

ACM ISBN 979-8-4007-0095-8/23/06…$15.00
https://doi.org/10.1145/3579371.3589104

cluttered environments), and the constraints of the size, weight, area, and power allocated to the onboard computing platform.

To navigate this large, diverse design space, it is necessary to identify common computational patterns that are impacted by the physical characteristics of the deployment scenario, and *encode this domain-specific information into systematic hardware design flows*. Such domain-informed design exposes a natural set of *domain-specific hyperparameters* in hardware for emerging applications, making it straightforward to implement accelerators for new deployment scenarios by tuning those values (much like the optimization of hyperparameters in neural network accelerators, e.g., weight compression, MAC array size [38]). This enables agile, principled design of accelerators across deployment scenarios without the intervention of domain experts or hardware engineers.

Robotics is an emerging area of interest for hardware accelerator design [1, 3, 16, 17, 22, 26, 27, 31, 32, 48, 51], and some promising work has begun mapping out design flows [1, 16, 22, 26, 32, 48]. In particular, *robomorphic computing* [32] introduced *the use of physical robot characteristics as parameters for accelerator design* (e.g., sparse functional units based on joints, and naive parallelization based on robot limbs and links). However, this prior work had severe scalability limitations: there was no branching support, and it did not address scenarios where robot size (i.e., total rigid links) exceeds the total processing elements that fit on a computing platform (Fig. 1). To fully realize flexible deployment of accelerators, it is necessary to manage tradeoffs across the diverse nonlinear design space of robot shapes and sizes, and computing resource constraints.

To address this challenge, we present *RoboShape*: an accelerator design framework that expands the scope of prior work, *leveraging topology-based computational patterns that scale with robot shape and size*, which are common across a broad class of critical computations that underpin robotics applications [7, 11, 12, 29, 35, 52] (Table 1). We expose opportunities to systematically use these scalable computational patterns to design architectural mechanisms with topology-based domain-specific parameters, in order to *flexibly* implement optimized accelerators *scalably* across different robots and computational resource constraints.

In this work, we identify two key computational patterns that scale with robot size and shape: ① *topology traversal* patterns and ② *large topology-based matrices*. Topology traversals propagate computations down the tree of robot links and joints, e.g., to calculate robot physics, and can scale with the total number of robot links, $N$, or as $N^2$. Sparse, large topology-based matrices, such as the robot *mass matrix* [12], can scale as $N \times N$, and display sparsity patterns determined by independent robot limbs. These two patterns can be used to systematically implement architectural mechanisms parameterized by robot topology, including: task mapping, scheduling, and allocation; task data placement; sparse block matrix operations; and sparse I/O data patterns. Because topology-based patterns scale with robot size and shape, using them to directly tune an architecture that is natively parameterized according to these patterns enables the systematic design of accelerators that can be optimized and flexibly deployed across different combinations of robots and computing resource constraints.

To demonstrate the use of these key topology-based computational patterns, our motivating example in this work is acceleration of the first-order gradients of dynamics [7], a key computation that traverses robot topology. In motion planning and control for rigid body robots, prior work demonstrates that dynamics gradients can take up to 30% to 90% of total runtime for some state-of-the-art approaches (e.g., nonlinear optimal control [7, 32, 33, 39, 43]). Dynamics gradients are a bottleneck preventing nonlinear optimal motion control with accurate whole-body modeling from being performed online for complex robots, e.g., quadrupeds [4, 49] (Fig. 1).

We create an *automated design framework* and use it to implement accelerators for the dynamics gradients for multiple robots, e.g., the iiwa manipulator [25], HyQ quadruped [49], and Baxter torso [14], on a Xilinx XCVU9P FPGA. While this FPGA can fit enough processing elements to completely parallelize the 7-link iiwa robot, RoboShape's flexible allocation is *necessary to enable deployment* of accelerators on this platform for the larger 12-link HyQ and 15-link Baxter (Fig. 1). For iiwa, HyQ, and Baxter, RoboShape-generated dynamics gradient accelerators on an FPGA give a 4× to 4.4× speedup in compute latency over state-of-the-art CPU solutions, and a 8× to 15.1× speedup over GPU solutions.

Using our framework, we also perform a novel study of the influence of robot topology on the parallelism, performance, and resource utilization of our dynamics gradient accelerators. Future work towards a full robotics domain-specific system on chip (SoC) will require navigation of this design space along with flexible accelerator co-generation. To assist such efforts, we have packaged RoboShape into an automated hardware generation flow, to be released as an open-source project [2].

Key contributions of this work are that:

- We identify and leverage two computational patterns that scale with robot size and shape, (1) topology traversals and (2) large topology-based matrices, to expose opportunities to flexibly implement architectural mechanisms (e.g., task scheduling, sparse block matrix multiplication) for topology-based robotics computations, unlocking a previously unexplored tradeoff space necessary for robotics accelerators;
- We present *a novel robotics design framework* for architects to automatically generate accelerators using scalable mechanisms that adapt to robot topology, and package the framework as an automated open-source tool; and
- We evaluate *automatically generated accelerators* for three topologically-diverse robots (manipulator, quadruped, torso) on a Xilinx XCVU9P FPGA, demonstrating compute speedups of 4× to 4.4× over state-of-the-art CPU solutions and 8× to 15.1× over GPU solutions for dynamics gradients: a key bottleneck preventing online nonlinear optimal motion control for legged robots, which remain an elusive real-world deliverable due to their computational bottlenecks.

## 2 BACKGROUND

**Robotics Computational Pipeline.** Core robot functionality can be grouped into a three-stage pipeline: perception, mapping and localization, and motion planning and control. In perception, a robot uses sensors to capture data about its surroundings, and performs processing such as semantic segmentation, computer vision, and sensor fusion to parse that data. In mapping and localization, a robot constructs or updates a map of its environment, and determines its location with respect to surrounding objects and agents. Finally, in

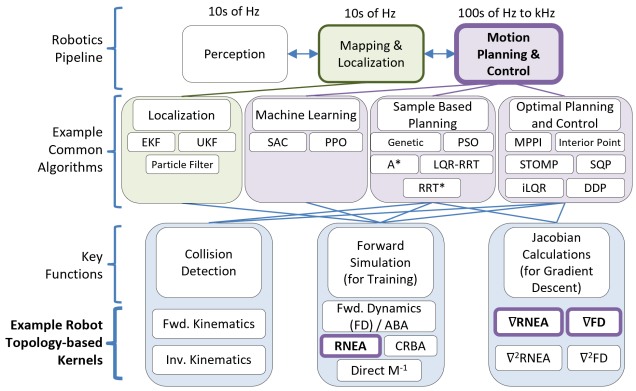


**Figure 2: Robot topology-based kernels are key computational building blocks for many common robotics algorithms. The motivating example in this work is forward dynamics gradients (∇FD) using RNEA and ∇RNEA [7], which can be used for motion planning and control (highlighted above).**

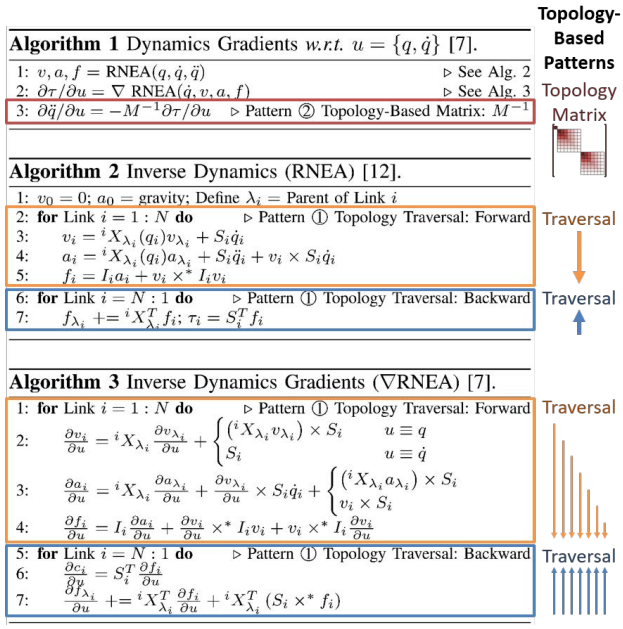


**Figure 3: Topology-based patterns appear in some important families of robotics algorithms, e.g., the dynamics gradients (Alg. 1, using Alg. 2 and Alg. 3), and are bottlenecks in state-of-the-art optimal robot motion control approaches.**

motion planning and control, a robot determines and executes a collision-free path of motion through its environment.

**Rigid Body Dynamics & Gradients.** Common topology-based building block computations, such as rigid body dynamics and gradients, underlie many algorithmic approaches to the robot pipeline stages (Table 1). Rigid body dynamics algorithms are functions of a robot's joint positions, velocities, accelerations, and torques $(q, \dot{q}, \ddot{q}, \tau)$. They propagate velocities, accelerations, and forces $(v, a, f)$ along the topology of *rigid body robots* according to link inertias $(I)$ and transformations along links and across joints $(X(q), S)$. Examples of such algorithms are shown in Alg. 1, Alg. 2, and Alg. 3, and are described in detail in prior work [12, 44]. A broad group

**Table 1: Common Topology-Based Patterns in Robotics: ① Forward & Backward Traversals, ② Topology Matrices**

| | Fwd. Kinematics [12] | Inv. Kinematics [12] | CRBA [52] | Direct $M^{-1}$ [7] | RNEA [12, 29] | ABA [11] | ∇RNEA [7] | ∇Dynamics [7] | ∇²RNEA [35] | ∇²Dynamics [35] |
|---|---|---|---|---|---|---|---|---|---|---|
| Fwd. Traversal | X | | X | X | X | X | X | X | X | X |
| Bwd. Traversal | | X | X | X | X | X | X | X | X | X |
| Topology Matrix | | | | | | | | X | | X |

of real robots can be described as trees of rigid links connected by joints, e.g., manipulators, quadrupeds, torsos (Fig. 1), as well as some approximations of flexible "soft" robots [19, 47]. In Sec. 5, we evaluate accelerators for robots with different topologies.

**Bottlenecks.** The kinematics, dynamics, and dynamics gradients of these rigid body robots are necessary calculations for many motion planning approaches. In particular, Sec. 3, Sec. 4, and Sec. 5 use *forward dynamics gradients* [7] (Alg. 1) as a motivating example example to demonstrate accelerator implementation using our framework because this is a *critical bottleneck calculation* that can take 30% to 90% of the total runtime of state-of-the-art optimal control approaches to motion planning and control [7, 32, 33, 39, 43].

Along with dynamics, several key kernels are bottlenecks in other motion planning approaches, e.g., collision detection for sampling-based planning [31] (Fig. 2). To holistically accelerate robotics workloads, it is critical to establish design frameworks that can be methodically extended and can provide common interfaces across algorithms and approaches, enabling designers to combine flexible hardware IP in future end-to-end robotics SoCs.

**Prior Work: Constant Sizes & Naive Parallelism.** The initial paper on robomorphic computing [32] (i.e., hardware designed based on robot physical parameters) introduced several mechanisms to use robot structure to guide accelerator design for physically-based computations like rigid body dynamics gradients, however these mechanisms focused on sparsity patterns in small constant-sized matrices ($6 \times 6$ elements), and on naive parallelization across processing elements that was directly based on the total number of links, $N$, in the robot. This approach has a constant growth with the number of links in the robot, and *does not scale with robot size.*

## 3 ROBOSHAPE: ARCHITECTURAL INSIGHTS

RoboShape's architectural insights are based on two computational patterns that scale with robot topology: ① *topology traversal* and ② *topology-based matrix operations* (Fig. 4). These patterns appear across robotics kernels (Table 1), e.g., those related to physics that we explore in this work (Fig. 3). We use these patterns to identify mechanisms to enable the *scalable*, *flexible* accelerator deployment.

**Scalable Computational Patterns from Robot Topology.** This work addresses computational patterns from robot topology that *scale with robot size* in order to *parallelize* in ways that intelligently make use of robot topology information (i.e., the branching structure of the robot's limbs and links); and *leverage sparsity patterns* in data structures that grow with the size of the robot ($N \times N$ links). Using these scalable topology-based computational patterns enables us to flexibly design optimized accelerators for different robots, delivering improved performance over previous

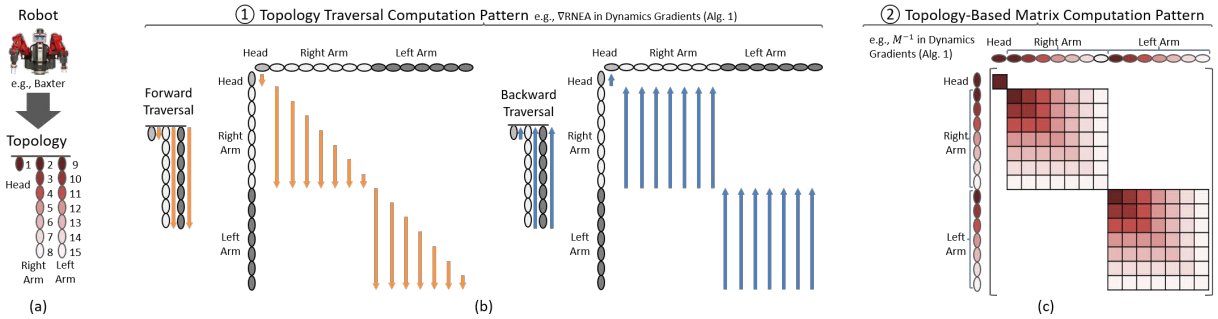


**Figure 4: We identify two common computational patterns based on (a) robot topology: ① topology traversal and ② topology-based matrix operations. For, e.g., dynamics gradients (Alg. 1) on Baxter [14], both (b) forward and backward topology traversals and (c) topology-based matrices reveal opportunities to leverage parallelism and sparsity exposed by the independence of robot limbs. Such patterns scale with robot size and shape, enabling deployment flexibility.**

work and other baselines (Sec. 5). This flexibility also unlocks the ability to customize accelerator resource utilization in order to fit accelerators on a given platform, or to co-optimize accelerator sizes, e.g., for the design of full robotics systems-on-chip (SoCs).

**Impact & Scope of Robot Topology Patterns.** There are broad classes of algorithms that rely on the propagation of physics values through a robot's body. While largely focused in motion planning and control, they also appear in other robotics workloads, e.g., localization with an extended Kalman filter (EKF) (Fig. 2). RoboShape addresses the hardware implications of topology-based computational patterns shared across these families of algorithms (e.g., kinematics, rigid body dynamics), and it is *complementary* to acceleration frameworks for other robotics algorithms to enable flexible combination of computational elements into future robotics SoCs.

**Motivating Example.** We describe these common computational patterns in more detail below. We use *forward dynamics gradients* (Alg. 1) [7] as our motivating example as it encompasses other topology-based functions (RNEA and ∇RNEA, Alg. 2 and Alg. 3) as subroutines. Our example robot is the Baxter torso [14], which has 15 links: two 7-link arms, and a 1-link head (Fig. 4a).

## 3.1 Pattern ①: Topology Traversals

There are many algorithms that propagate values up and down the robot topology tree, making forward and backward passes along the links of the robot. For example, in the dynamics gradient, a forward and backward traversal of the topology tree is made for the inverse dynamics (Alg. 2). The outputs of each per-link step of this traversal are then used to seed forward and backward passes calculating the partial derivatives of the inverse dynamics with respect to each link (∇RNEA). For the 1-link head and two 7-link arms in the Baxter robot, this results in per-link task patterns like those pictured in Fig. 4b. These task patterns can be used to easily determine efficient strategies for task data placement and task mapping, scheduling, and allocation (see Sec. 3.3), and also to tune the parameters of these strategies to deploy the same computational task graph under different computing resource constraint scenarios.

## 3.2 Pattern ②: Topology-Based $N \times N$ Matrices

There are also key robotics algorithms that require linear algebra operations performed on large, often sparse topology-based matrices that scale with the size of the robot (Table 1). For example, the final step of the dynamics gradient algorithm requires two multiplications of dense partial derivative matrices with the inverse of the *mass matrix* [12], a characteristic matrix with values related to the robot's joint space inertial properties. Sparsity patterns in the mass matrix are determined by the a robot's topology. For example, Fig. 4c shows Baxter's mass matrix. The head and two arms are independent branches of the robot, so they populate only the diagonal of the matrix, along the entries that correspond to the links within each of these three separate limbs. Robots with entirely independent limbs follow this block diagonal pattern, whereas robots with branches that share a large number of parent links have nonzero off-diagonal entries. As the inverse of a block diagonal matrix (e.g., used in dynamics gradients) is block diagonal, this enables block matrix operations on deterministic sparsity patterns (Sec. 3.3).

## 3.3 Using Topology-Based Architectural Mechanisms for Accelerator Design

Using computational patterns that scale with robot size (Sec. 3.1 and Sec. 3.2), we identify several mechanisms to leverage topology information for scalable accelerator design.

**Task Mapping, Scheduling, & Allocation.** We use robot topology traversal patterns (Sec. 3.1) to map out task graphs for robotics applications, and then perform scheduling and allocation of processing elements for robotics processors, e.g., robomorphic sparse $6 \times 6$ linear algebra processing elements based on robot links and joints. To achieve scalable solutions, in this work we evaluate topology-based strategies and show that they leverage the branching structure of the robot to expose parallelism (Sec. 4) and deliver improved performance over baselines, while conserving resources (Sec. 5).

Furthermore, using the parallelism of the robot topology and tying that to performance and resource utilization tradeoffs exposes architectural knobs (Sec. 5) that can be tuned to fit an accelerator design onto platforms of different sizes, or to re-scale a design to fit within area limits alongside other accelerators in a larger SoC. These knobs also enable future work in dynamic tuning, e.g., doing power gating of processing elements to manage Dark Silicon power wall constraints; or to dynamically co-schedule different types of kernels simultaneously on processing elements.

**Task Data Placement.** Studying topology traversal computational patterns also exposes opportunities for topology-informed data placement, to minimize the costs of data movement. Depending

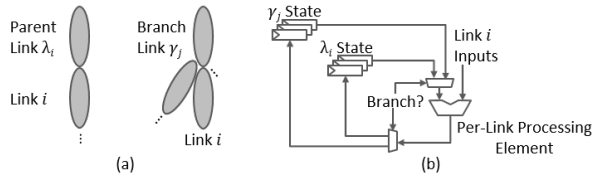


**Figure 5: Rigid body robot topologies are trees of (a) connected links and branches. Topology information can be used at design-time to determine data placement, e.g., (b) near-compute storage for parent links and branch links.**

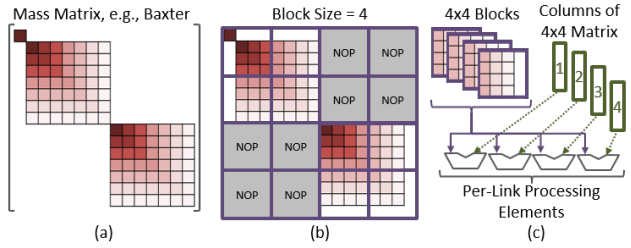


**Figure 6: Robots with independent limbs have a (a) block diagonal sparse characteristic matrices that scale with robot size (e.g., the mass matrix of a torso robot). Using this sparsity pattern we can (b) tune block size to skip zero entries when (c) scheduling and parallelizing linear algebra in hardware.**

on the memory hierarchy and size available on the computational substrate, it can make sense to choose a dataflow-style architecture (like those popularized by neural network accelerators) and then use the topology pattern to inform the design of near-processor element storage for intermediate state, e.g., state saved wherever branching occurs, so that computation can continue down the tree and then resume back at the source of the branch. Fig. 5b illustrates a dataflow-style processing element augmented with branch state storage and parent link storage, which we implement in our accelerator framework in Sec. 4. This approach is tractable for many common types of robots that have 1s to 10s of links (Fig. 1), but future work could also extend this branch checkpoint locality approach to cache data placement to increase storage density for robots with very large (100s to 1000s) numbers of links, e.g., hyper-redundant and continuum robots, and rigid body approximations of flexible "soft" robots [19, 47].

**Sparse Block Matrix Operations.** The topology-based matrix patterns from Sec. 3.2 expose limb-based sparsity that can lead to large patches of zeros in the (inverse) mass matrix of some robots. This presents an opportunity to perform block matrix operations, scaling the block size in order to skip large sections of zeros, reducing the total amount of work. For example, the $15 \times 15$ mass matrix for the Baxter robot (Fig. 6a) can be divided into $4 \times 4$ blocks (Fig. 6b), including blocks of all zeros, which can be skipped in matrix multiplication (labeled "NOP"). A dynamics gradient accelerator might have per-link processing elements (PEs) that can perform matrix vector multiplications, so one way to parallelize this operation in hardware would be to feed nonzero blocks into parallel per-link PEs (Fig. 6c) along with the columns of another matrix block.

**Sparse I/O Data.** Sparse topology-based matrix patterns (Sec. 3.2) enable data packet compression by skipping zeros when sending matrices (or their multiplication products) over communication channels. For example, Baxter's $15 \times 15$ mass matrix (Fig. 6a) has 99 nonzero elements, making it 56% sparse. We elaborate in Sec. 5.2.

# 4 ROBOSHAPE: ACCELERATOR FRAMEWORK

In this section, we present an automated accelerator design framework based on the computational patterns and insights explored in Sec. 3. The RoboShape framework can flexibly implement accelerators for a broad class of robotics computations (Table 1), across different robotics deployment scenarios.

**Overview.** Fig. 7 gives an overview of the RoboShape framework. RoboShape takes as *inputs* a standard robot description file and a constraint on compute resources, given as the maximum processing elements for forward and backward traversals (Fig. 4b), and maximum block size for topology-based matrix operations (Fig. 4c). With this, RoboShape:

- Parses the robot into a topology graph (Sec. 4.1);
- Uses ① robot topology traversal patterns in the application to generates task graphs and transforms them into schedules of tasks for processing elements (Sec. 4.2);
- Uses ② topology-based matrix patterns in the application to adjust linear algebra block sizes to minimize operating on zero values (Sec. 4.3); and
- Lowers computational patterns ① and ② to hardware by tuning a template architecture that has been factored according to these topology-based patterns (Sec. 4.4).

RoboShape *outputs* a streamlined topology-based accelerator design, flexibly tailored to the robotic deployment. The following sections provide details, using the dynamics gradient [7] and the Baxter robot [14] as motivating examples.

## 4.1 Robot Parsing to Extract Topology

We parse a standard URDF robot description file input into a topology that is a tree of articulated links connected by joints. URDF is an XML-based file format that describes the physical characteristics of a specific robot model. These files are provided by robot manufacturers to end users, and are leveraged by most common robot simulators. An example is shown in Fig. 4a for the Baxter robot: a torso with a head, and two arms that are each a chain of seven rigid links connected by rotating joints. Note that robot topologies with more complicated branching patterns (e.g., manipulators with gripper fingers [5]) force more dependencies between links (Fig. 14). For kernels involving derivatives, e.g., of dynamics (Alg. 1), this increases partial derivative calculations.

## 4.2 Using ① Topology Traversal Patterns for Task Allocation & Scheduling

We take the robot topology and use one or more of the ① topology traversal computational patterns (Sec. 3.1) in the application to generate graphs of work tasks based on calculations performed on the robot's rigid links. We can then allocate processing elements (PEs) to execute the tasks based on the maximum number of computing resources available within our input constraints, and generate a schedule that indicates which PE works on which task at each

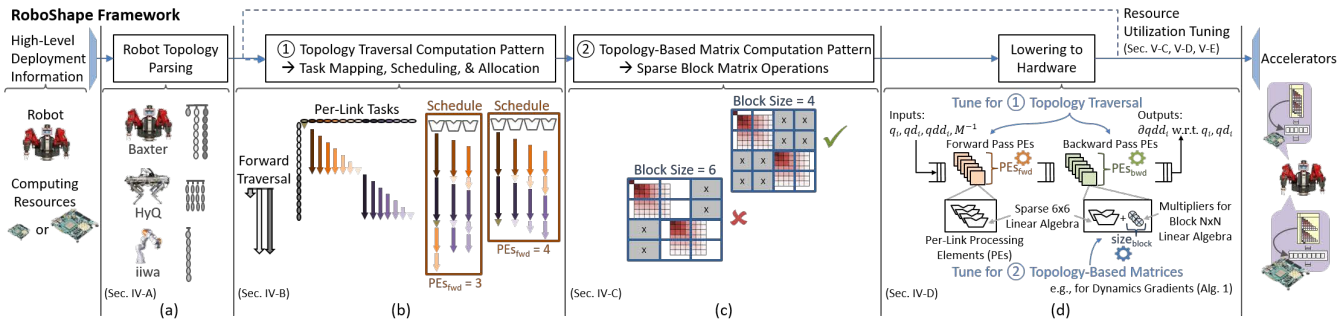


**Figure 7: The RoboShape framework takes in standard robot description XML files and computing resource constraints, and produces accelerators. It (a) parses robot topology (Sec. 4.1); (b) uses ① topology traversal patterns to generate task schedules for different hardware allocation operating points (Sec. 4.2); (c) uses ② topology-based matrix patterns to adjust block size to minimize operating on zeros (Sec. 4.3); and (d) lowers high-level robot topology-based decisions to generate accelerator hardware (in Verilog) by tuning a template architecture that has been factored according to topology-based patterns (Sec. 4.4).**

time step. For example, Fig. 7b shows two schedules for a forward traversal of Baxter's dynamics gradients, for 3 PEs and for 4 PEs.

Forward and backward topology traversal patterns can be scheduled onto PEs using a modified depth-first search: given a total number of PEs to allocate, for each step in the schedule and for each PE, we look for the longest sequential thread of tasks. We check that the next task in that thread has all prerequisite inputs ready (e.g., RNEA's inputs to ∇RNEA's partial derivatives), and if so, we schedule that task thread. When we hit a branch node in the topology, we save and restore its state appropriately when we process the next limb.

## 4.3 Using ② Topology-Based Matrix Patterns for Blocking & Task Scheduling

Multiplication by the mass matrix can be scheduled onto PEs as a blocked matrix multiplication, adjusting the block size for good coverage of dense regions of the mass matrix (e.g., $4 \times 4$ blocks in Fig. 6) and to minimize zero padding. For example, Fig. 7c shows a $6 \times 6$ block size for Baxter with wasteful zero padding, alongside the superior $4 \times 4$ block size. Block matrix multiplication tasks can be parallelized across per-link processing element hardware as shown in Fig. 6c, and those per-PE tasks can be scheduled in a similar manner to the topology-traversal tasks in the previous section.

## 4.4 Lowering to Hardware: Topology-Templated Accelerator Architecture

To demonstrate flexibly supporting scalable topology-based computation at the microarchitectural level, we generalize a dynamics gradient accelerator design introduced in prior work [32], creating a new architecture (see Fig. 8) that is deliberately factored and templated according to the robot-scalable computational patterns introduced in Sec. 3: ① topology traversals and ② topology-based matrices. This example architecture has three knobs that can be tuned according to those patterns to *flexibly deploy the hardware template* for different robot and computing constraint deployment scenarios: total forward and backward traversal PEs ($\text{PEs}_{fwd,bwd}$), and matrix multiplication block size ($\text{size}_{block}$). In Sec. 5.4 and Sec. 5.5, we analyze the impact of robot shape on accelerator resource utilization by tuning these parameters.

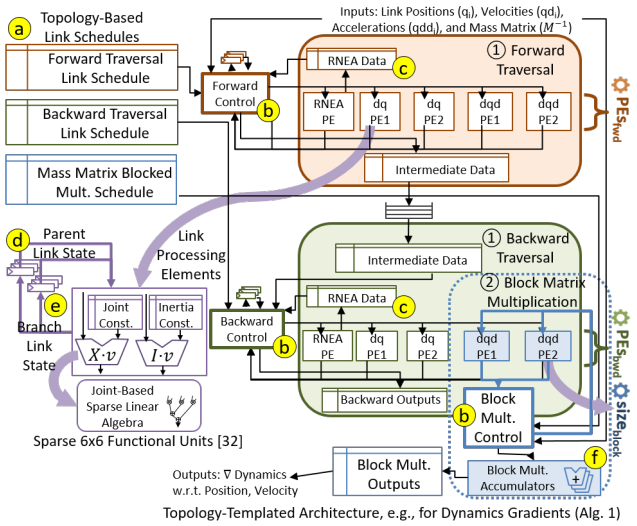


**Figure 8: Architecture for dynamics gradients, templated by topology-based parameters ($\text{PEs}_{fwd,bwd}$, $\text{size}_{block}$). Microarchitectural support includes: (a) storage for link schedules (Sec. 4.2, Sec. 4.3); (b) control state machines for PEs; (c) storage for RNEA outputs (Alg. 1); (d) storage to feed parent link values back into PEs; (e) storage for branch link state, restored when returning to branches; and (f) accumulators for block mass matrix multiplication.**

We introduce several architectural features to support per-link PE schedules (Sec. 4.2 and Sec. 4.3). We provide (a) dedicated schedule storage structures. To execute the schedules, there are (b) control state machines that step through the schedules and marshal robot link data to the correct PEs. Because the dynamics gradient example has data dependencies passing from the RNEA to the ∇RNEA (see Alg. 1), we add (c) intermediate storage structures for the outputs of the RNEA step, as it is an input to the partial derivative tasks.

During execution of a thread of tasks along a limb (e.g., forward traversal partial derivatives in Fig. 7b), and in between each schedule step, partial results from the current link are stored in (d) intermediate state, with mechanisms to flush the data when starting on a new limb. Similarly, when a PE hits a branching node in

**Table 2: Resource Utilization of RoboShape Designs**

| FPGA Resources (Xilinx XCVU9P) | iiwa [25] | HyQ [49] | Baxter [14] |
|---|---|---|---|
| LUTs (1182k Total) | 514552 (43.5%) | 507158 (42.9%) | 873805 (73.9%) |
| DSPs (6840 Total) | 5448 (79.6%) | 3008 (44.0%) | 3342 (48.9%) |

the robot topology that splits into several limbs, there are (e) checkpoint registers that save the state at the current link, so that after the current limb is completed, state can be restored.

Finally, to perform blocked multiplication (e.g., of the $M^{-1}$ matrix (Alg. 1), we introduce (f) accumulators near the matrix data structures to accumulate and store partial results.

## 5 EVALUATION

We evaluate the compute latency of RoboShape-generated accelerators on an FPGA target platform for three different robots (Sec. 5.1, Sec. 5.2), and analyze the resource utilization vs. performance design space across six different deployments (Sec. 5.3, Sec. 5.4, Sec. 5.5).

**Hardware Evaluation Platforms.** As baselines, we use the state-of-the-art Pinocchio [8] CPU and GRiD [45] GPU dynamics libraries. We use an eight-core 3.8GHz Intel Core i7-10700K CPU running Ubuntu 20.04 and a 1.7GHz NVIDIA GeForce RTX 3080 GPU with 8704 CUDA cores. These are comparable to platforms onboard real robots: e.g., robots in the DARPA Robotics Challenge [23, 24, 46, 50] had 4-core Intel i7 CPUs, and Boston Dynamics' Spot [4] has a 4-core Intel i5 CPU and an NVIDIA P5000 with 2560 CUDA cores. We disabled TurboBoost on the CPU and fixed all cores at maximum frequency. HyperThreading was enabled for threadpool use. We implemented accelerators in Verilog and Bluespec System Verilog on a Xilinx VCU-118 with a XCVU9P FPGA, which offers many digital signal processing units (6840 DSPs) for our linear algebra-heavy workload.

**Methodology.** In Sec. 5.1, we measure compute latency for a single computation averaged over one million trials. For CPU and GPU results, time was measured with `clock_gettime()`, using `CLOCK_MONOTONIC`. For FPGA accelerator compute latency, our goal in Sec. 5.1 was to isolate compute time without measuring I/O transfer time. (By contrast, in Sec. 5.2 we give full measured FPGA timing results, including I/O overheads.) We leverage the deterministic runtime (in clock cycles) of our design, multiplying it by our synthesized clock frequency to isolate pure compute latency through the accelerator. In Fig. 9, for *No Pipelining*, the latency of the accelerator stages (see Fig. 8) are added. For *Avg. w/ Pipelining*, we include pipelining between accelerator stages (Fig. 8).

**Baselines & Parallelism.** The state-of-the-art CPU software library [8] only parallelizes across multiple computations (see Sec. 5.2), and relies on vector operations within each computation. The state-of-the-art GPU library [45], is both parallelized across CUDA cores for a single computation (Sec. 5.1), and across streaming multiprocessors (SM) for multiple computations (Sec. 5.2).

### 5.1 Computation-Only Latency

We use RoboShape to implement dynamics gradient accelerators on a Xilinx XCVU9P FPGA for three different real robots: iiwa [25], HyQ [49], and Baxter [14], closing timing at 18ns, 18ns, and 22ns, respectively, enabling clock speeds of 55.6-45.5MHz. The critical path was through the input data marshalling logic set by the forward pass schedule, so clock speed scaled with that schedule's length.

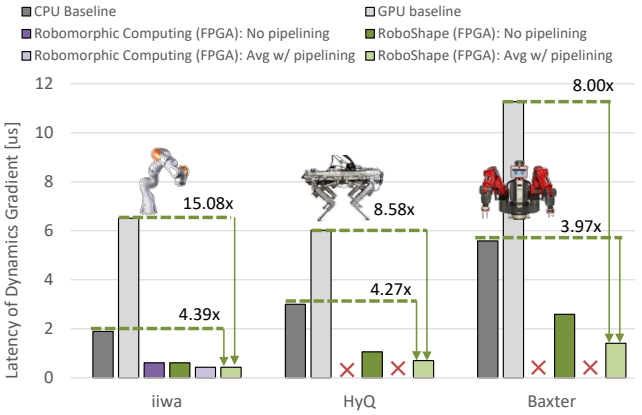


**Figure 9: Across a variety of real robots [14, 25, 49], RoboShape-generated accelerators on FPGA leverage low-overhead, streamlined parallel processing of robot links to achieve speedups of 4.0× to 4.4× and 8.0× to 15.1× over state-of-the-art CPU and GPU baselines [8, 45] for the latency of a dynamics gradient computation. RoboShape gives identical latency to the prior Robomorphic Computing (RC) [32] accelerator for iiwa, but unlike prior work, RoboShape designs are *able scale to larger multi-limb robots* (HyQ, Baxter).**

**RoboShape Generator Knob Settings.** We sized our iiwa accelerator for direct comparison to prior work [32]: $\text{PEs}_{fwd,bwd}$, $\text{size}_{block}$ = 7, 7. Because closing timing requires repeated iteration of FPGA synthesis, for HyQ and Baxter, we chose conservative values for the parameters with reasonably low resource utilization (Table 2) to reduce place-and-route complexity: for HyQ, $\text{PEs}_{fwd,bwd}$, $\text{size}_{block}$ = 3, 6; and for Baxter, $\text{PEs}_{fwd,bwd}$, $\text{size}_{block}$ = 4, 4.

**Comparison to CPU & GPU Baselines.** Fig. 9 compares the single computation latency on our target FPGA against CPU and GPU baseline solutions [8, 45]. Even with conservative parameter tuning and significantly lower clock speeds, RoboShape accelerators achieve speedups of 4.0x to 4.4x and 8.0x to 15.1x compared to our state-of-the-art baselines.

For CPU and FPGA, compute latency scales roughly with the total number of links in the robots (7, 12, and 15 for iiwa, HyQ, and Baxter). On the CPU this is caused by the computation being done by one thread with only limited parallelism avaiable via vector operations. For the FPGA, our accelerators provide sufficient parallelism to handle the longest limb length (7, 3, and 7 for iiwa, HyQ, and Baxter) by allocating $\text{PEs}_{fwd,bwd}$ = 7 and 3 for iiwa and HyQ. For Baxter, $\text{PEs}_{fwd,bwd}$ = 4 and the limb length = 7, but the schedules are densely packed, so latency scaling stays roughly close to $N$, despite the smaller number of PEs.

Like with the CPU, the GPU uses a single SM for each dynamics gradient calculation. As such, despite still having many available threads on each SM, GPU latency is significantly higher than both the CPU and FPGA accelerators as GPU pipelines are optimized for throughput rather than latency, penalizing sequential operations. In fact, GPU latency is similar between iiwa and HyQ because iiwa is entirely sequential (a pathological case for the GPU), while the larger HyQ robot has parallel limbs with short sequential chains.

**Comparison to Prior Work: Robomorphic Computing.** In addition to CPU and GPU baselines, Fig. 9 compares the *single*

Robomorphic Computing (RC) [32] accelerator (iiwa robot) against three RoboShape accelerators (iiwa, HyQ, and Baxter robots). RC only demonstrated a *single* accelerator for a robot with a *single* limb (iiwa robot), not addressing the patterns of branching multi-limb robot topologies. Instead, RC applies static parallelization fixed to total robot links, and does not use the branching topology structure of the robot– this leads to a resource blow-up that prevents RC from fitting robots with large numbers of links within the constraints of different compute platforms. By contrast, RoboShape uses robot topology-based architectural insights to address (i) the *scalability* challenge of deploying large robots within the resource constraints of different compute platforms, and (ii) the *flexibility* challenge of generating accelerators for diverse branching robot topologies (e.g., quadrupeds, torsos). For example, even for the large Xilinx XCVU9P FPGA, RC cannot scale beyond robots bigger than the 7-link iiwa arm (which consumes 77.5% of DSP resources; beyond this, it becomes difficult to close timing), whereas by using RoboShape's topology-based scheduling and allocation, we can automatically implement FPGA-based accelerators for the iiwa (with identical latency to the RC design), as well as *two other topologically-diverse robots* of up to 15 links (Fig. 9). RoboShape's novelty is that it adds a layer of domain-informed architectural flexibility that enables accelerators for topology-based applications (like RC) to *scale* to larger robots and to *flexibly* re-configure designs to support complex branching robots on diverse compute platforms.

**Overheads of RoboShape vs. Custom Design.** We compared resource utilization on a Xilinx XCVU9P FPGA against prior work [32] that specifically targeted the iiwa manipulator [25] with custom design (note that by contrast, the RoboShape framework supports multiple robots– see Fig. 9). For iiwa, our microarchitectural changes enabling generalizability across robots and computing constraints (see Sec. 4) introduce minimal overheads: 5.5% decrease in total LUTs (43.5%, versus 49.0% in prior work); and 2.2% increase in digital signal processing (DSP) blocks (79.6%, versus 77.5% in prior work). Overheads increase with robot size and complexity, but this design generalization enables RoboShape to generate accelerators for large multi-limb robots that flexibly fit within computing resource constraints, unlike prior work.

## 5.2 Coprocessor Roundtrip Latency with I/O

To analyze coprocessor roundtrip latency for the FPGA accelerator, we deployed it as a coprocessor to a host CPU for a batch of multiple dynamics gradients calculations. This computational pattern is often seen across time steps in a motion trajectory for nonlinear optimal control approaches to motion planning. Reasonable time steps can range from 1-10 [6, 9, 13, 28], so we demonstrate an intermediate value, 4, in this experiment. Note that some systems use very large numbers of time steps, into the 100s [15, 30, 34]. This greatly increases total I/O that must be transferred to a coprocessor, so later in this section we propose I/O optimizations that would help our system scale in future work.

**Methodology.** We run 4 time step computations, following the methodology of Section 5.1. For the FPGA, we give full measured results including I/O overheads (Fig. 10: *Roundtrip Including I/O*), as well as the total compute-only latency, extracted using clock periods and cycle counts, as in Sec. 5.1 (Fig. 10: *Compute Only*), to analyze I/O and pipeline stall latency. For these multiple computation experiments, the CPU library parallelizes between time step computations, so because we use 4 time steps, 4 threads are created. Within each thread, parallelism is leveraged through vectorized operations. The GPU library paralleizes across streaming multiprocessor (SM) cores for the 4 time step computations, and within each computation, multiple parallel CUDA core threads are created.

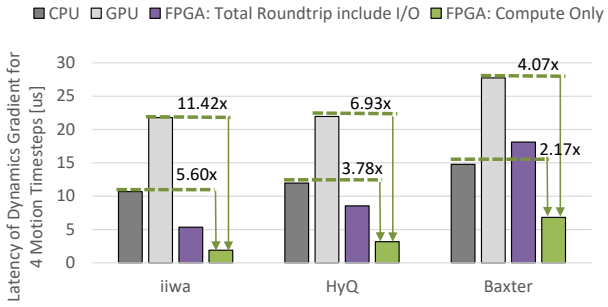


**Figure 10: Deployed as a coprocessor, accelerator computation-only latency achieves 2.2x to 5.6x and 4.1x to 11.4x speedups over CPU and GPU. However, high I/O overheads reduce roundtrip latency, especially for the largest robot (Baxter) which is slightly slower than CPU. Future I/O optimizations, e.g., leveraging mass matrix sparsity patterns to avoid sending zero values (Sec. 5.2), can reduce I/O data size by 3x for HyQ and 2x for Baxter.**

**Coprocessor Latency Results.** Fig. 10 gives latency results for the multiple-calculation coprocessor experiments. The *Compute Only* accelerator latency achieves 2.2x to 5.6x and 4.1x to 11.4x speedups over CPU and GPU. However, the *Roundtrip Including I/O* latency is slowed by the addition of I/O overheads, reducing these speedups for the iiwa and HyQ robots to 2.0x and 1.4x over CPU, and 4.1x and 2.6x over GPU. For the largest robot (Baxter, with 15 links), while the accelerator maintains a 1.5x speedup over GPU, it experiences an 18% slowdown compared to the CPU. I/O overheads increase with $N$, the total number of links in the robot, since position, velocity, and acceleration inputs scale per-link, and the mass matrix and the output partial derivative matrices all scale as $N^2$.

Reducing these I/O overheads can help approach the compute-only performance. One way is to improve inefficiencies in the Connectal [20] tool used for PCIe communication in order to approach the speed of PCIe Gen 3 (used by our GPU baseline), which is around 3x faster than the PCIe Gen 1-level connection used by our FPGA in Fig. 10. Another improvement is to leverage topology-based sparsity patterns in the I/O data to avoid spending time receiving and sending zeros (described in Sec. 3.3). We can skip zeros related to mass matrix sparsity that are inputs to the coprocessor when receiving the mass matrix itself, and are outputs when sending back the partial derivative matrices, which share these sparsity patterns. As robots increase in total number of links $N$, the mass matrix and partial derivative matrices (which scale as $N^2$) become the dominant portion of I/O packets, making up 84%, 90%, and 92% of the total I/O bits for the iiwa, HyQ, and Baxter robots, respectively. While iiwa's mass matrix is fully dense, for robots with multiple limbs such as HyQ and Baxter, we can skip zeros based on known sparsity patterns in the mass matrix, which is, e.g., 75% sparse for HyQ and 56% sparse for Baxter. This results in expected I/O data size

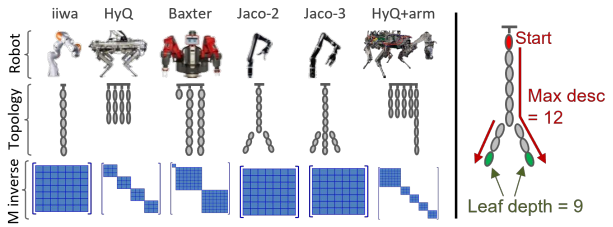


**Figure 11: *Left*: Robots with different topologies [14, 21, 25, 49] as motivating examples to analyze resource utilization in RoboShape accelerators. *Right*: Robot metrics in Table 3, e.g., maximum descendants (*Max desc*).**

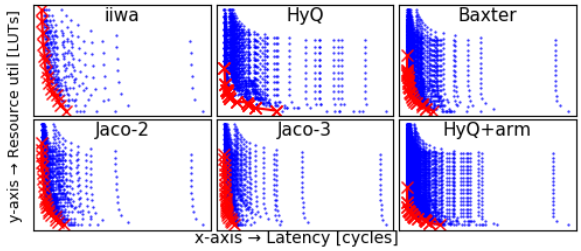


**Figure 12: Parameterizing accelerators based on robot topology patterns exposes varied, but tractable (1000s of design points) design spaces for different robots. Pareto frontier of optimal co-design points annotated with red $X$s. LUTs and latencies normalized to maximum for each robot. Maximum latencies are 829-7230 cycles; maximum LUTs are 507k-2600k.**

reductions of 3.1x for HyQ and 2.1x for Baxter. These I/O speedups will enable coprocessor-based accelerators to scale to even larger robots, and to greater numbers of batched motion time steps.

**Parallelism Tradeoffs vs. GPU.** RoboShape extracts maximal parallelism from within each individual rigid body dynamics computation (Fig. 14), while GPUs are best at extracting parallelism across multiple time steps of the same dynamics computation. This leads to the FPGA accelerator having far lower latency than the GPU (Fig. 9) for a single time step, but GPUs potentially having higher throughput when used for multiple time steps. This throughput limitation can be addressed with further I/O optimization (Sec. 5.2), or by instantiating multiple RoboShape cores in an ASIC.

## 5.3 Robot Shape Implications for Accelerators

While Sec. 5.1 and Sec. 5.2 demonstrated speedups for single design points produced with RoboShape, in the following sections we broaden our scope to consider the diverse design space of robot deployments, tying robot shape back to architectural insights to guide accelerator design.

**Diverse Robot Shapes.** Robotics deployments are diverse, and it is important for hardware designers to be able to analytically understand how performance and hardware resource utilization will change based on design choices under different deployment conditions (e.g., robot shape and size, computing platform choices). We characterize a diverse set of example robot deployments (see Fig. 11) and analyze how robot shape impacts resource and performance tradeoffs across RoboShape accelerators, according to the computational patterns introduced in Sec. 3. The size and shape

**Table 3: Topology Metrics for Robots in Fig. 11.**

| Topology Metric | iiwa | HyQ | Baxter | Jaco-2 | Jaco-3 | HyQ+arm |
|---|---|---|---|---|---|---|
| Total Links | 7 | 12 | 15 | 7 | 12 | 19 |
| Max Leaf Depth | 7 | 3 | 7 | 9 | 9 | 7 |
| Avg. Leaf Depth | 7 | 3 | 5 | 9 | 9 | 3.8 |
| Max Descendants | 7 | 3 | 7 | 12 | 15 | 7 |
| Leaf Depth StDev | 0 | 0 | 2.3 | 0 | 0 | 1.6 |

of the robots in Fig. 11 can be broken down by total number of links, the longest chain of links in the topology tree, the average leaf depth, the max number of descendants of any given parent link in the topology, and the standard deviation of leaf depth (Table 3). For example, the Baxter robot has 15 total links. The longest chain of links in Baxter is 7 links, through one of its arms. The average leaf depth in Baxter is 5, because of its single-link head and two 7-link arms. The max descendants of any link is 7, again through an arm. Finally, the standard deviation of leaf depth is 2.3 because of the asymmetry between Baxter's head and arms.

**Resource-Performance Design Tradeoffs.** RoboShape identifies computational patterns that scale with the robot shape and size, which enables us to perform analytical tuning of the parameters of architectural mechanisms that use those patterns. This unlocks the ability to analytically sweep parameters and study the impact of deployment scenarios (i.e., robot and computing platform) on hardware metrics of RoboShape accelerators (i.e., latency and resource utilization). In this section, we evaluate different sweep studies.

Without a principled approach to accelerator design, the space of system parameters can be enormous and intractable. By contrast, designing accelerators that are parameterized by physical properties provides an easy way to define knobs to tune that are meaningfully tied to the deployment scenario. This makes it straightforward to create tractable design spaces that can be navigated to select desired optimized design points. For example, Fig. 12 shows design spaces for RoboShape accelerators for the six robots in Fig. 11. The following sections analyze design spaces for resource utilization versus latency tradeoffs for RoboShape accelerators for different robot and resource constraint operating points.

**Key Insights: Guiding Accelerator Design with Robot Topology.** The following sections analyze the relationship between robot topology patterns and the performance and resource allocation of RoboShape accelerators. We find three key insights: (1) traversal patterns have topology-based bottlenecks and parallelism; (2) topology-based matrix resource allocation leads to nonlinear performance; and (3) topology-based tuning beats maximum resource allocation.

## 5.4 Robot Shape Impacts Resource Utilization

In this section, we establish physically-grounded intuition for which features of robot topology contribute to efficient resource allocation strategies for minimum latency in topology-based accelerators. We find that the resource allocation and latency of topology traversal patterns in dynamics gradients accelerators is affected by robot limb symmetry, and that parallelism differs between forward and backward traversals. We also demonstrate that robot shape has a nonlinear effect on optimized resource allocation for minimum latency for topology-based matrix operations.

**Methodology.** To analyze the impact of topology traversal pattern computations, we use our RoboShape scheduling algorithms

(Sec. 4) and a model of resource utilization (in LUTs and DSPs) for processing elements (PEs) on the XCVU9P FPGA, extracted from hardware synthesis reports for different resource allocation strategies in which the total numbers of forward and backward traversal PEs instantiated ($PEs_{fwd,bwd}$) are set to values based on the robot topology metrics in Table 3: *Total Links*, which corresponds to naive parallelism used in prior work [32]; *Average Leaf Depth*; *Maximum Leaf Depth*; *Maximum Descendants*; a *Hybrid* approach, allocating PEs for performing forward topology traversals set to the maximum leaf depth, and PEs for backward traversals set to the maximum descendants; and finally, *Optimal Minimum Latency* values from an exhaustive search of the design space. For all allocation strategies in this section we are targeting a minimum latency design point without constraints on overall resources (we perform a resource-constrained analysis in Sec. 5.5).

**Insight #1: Traversal Patterns Have Topology-Based Bottlenecks & Parallelism.** While Fig. 13a shows that nearly all strategies achieve minimum latency across the robots (with several exceptions discussed below), Fig. 13b demonstrates that they do so by allocating very different amounts of resources. Resource utilization based on *Total Links* grows rapidly for larger robots, while other strategies achieve the same latency for the same or fewer resources. This implies that naively scaling resources with the number of links vastly over-provisions resources. By contrast, allocation by average leaf depth underprovisions resources and gives poor latency on all robots except for iiwa and HyQ, where it happens to equal the maximum leaf depths and number of descendants because they are symmetric with no branches in their limbs.

The strategies that approach optimal allocation while meeting minimum latency are *Max Leaf Depth*, *Max Descendants*, and the *Hybrid* of those approaches. Fig. 14 illustrates how these strategies correspond with physical intuition about traversing robot topologies with an example of the forward and backward traversals patterns for the ∇RNEA subcomputation (Alg. 1). Allocating by *Max Leaf Depth* is a reasonable approach for forward traversals because there are narrow bottlenecks at the top of the tree, shaping the dependencies of computations like partial derivatives so that the main chain of parent link prerequisite values coming from the top of the tree follow along the depth of the tree. For branching topologies (e.g., Jaco-2 and Jaco-3), however, this approach underprovisions the backward traversal and gives poor latency (Fig. 13), because the wide tree bottom means that more parallel threads of execution can launch based on parent link prerequisite values from the bottom of the tree. Instead, allocation by *Max Descendants* does well in these cases, by scaling with the wide bottom of such trees.

A *Hybrid* using *Max Leaf Depth* for the forward and *Max Descendants* for the backward traversal improves on both strategies, consistently meeting minimum latency. For symmetric robots with *Leaf Depth StDev* = 0, the resource utilization of the *Hybrid* heuristic is on par with the optimal allocation as the upper bounds on $PEs_{fwd}$ and $PEs_{bwd}$ are maximum leaf depth and maximum number of descendants, respectively. Because all the limbs in symmetric robots are of equal length, the optimal resource split equals these upper bounds, so the design point found by the heuristic matches the optimal one. For asymmetric robots, e.g., Baxter and HyQ+arm with *Leaf Depth StDev* > 0, the *Hybrid* heuristic overprovisions resources compared to the optimal allocation as the RoboShape

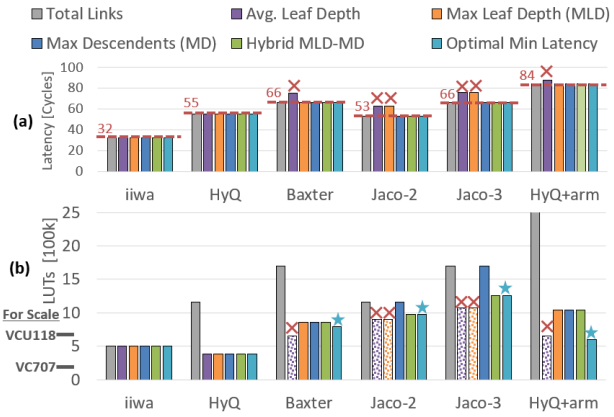


**Figure 13: *Optimal Minimum Latency* finds a minimum latency design point with lowest resource utilization, using exhaustive search. Other strategies allocate resources based on topology metrics (Table 3). Most achieve minimum latency, but with varying resource utilization. Red *X*s mark non-minimum latency configurations and blue stars (★) mark the optimal configurations.**

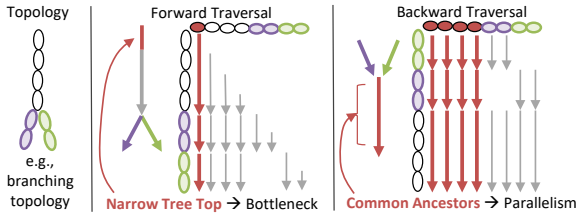


**Figure 14: The degree of parallelism leveraged by traversal computational patterns depends on robot topology. Forward pass: parallel threads launched scales with number of independent limbs. Backward pass: parallel threads scale with number of common ancestors for leaf links.**

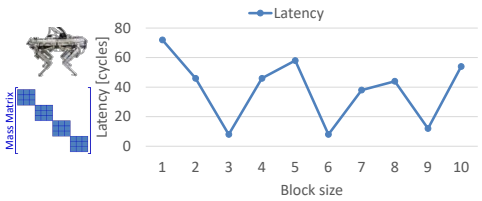


**Figure 15: The design space of sparse block matrix multiply is non-linear. An increase in block size can result in a *decrease* in performance. Here we sweep block size from 1 to 10, for 3 block matrix-vector multiply units (Fig. 8).**

scheduler takes advantage of the variation in limb length. In particular, it fits short threads of work into spaces in the task pattern graph left empty by large threads, and squeezes the number of PEs to below the upper bound set by metric-based heuristics.

**Insight #2: Topology-Based Matrix Resource Allocation Leads to Nonlinear Performance.** The relationship between resources allocated for the blocked matrix multiply and its latency is jagged and nonlinear, with increased resources sometimes resulting in a *decreased* performance. Fig. 15 shows the latency of the blocked mass matrix multiplication for different block sizes using HyQ

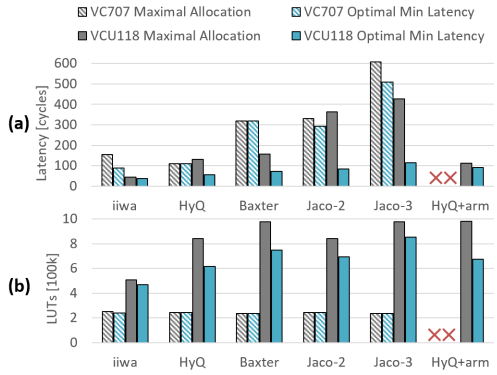


**Figure 16: Under resource constraints (e.g., Xilinx VCU118 and VC707 FPGA platforms), allocating more resources can *reduce* performance due to the nonlinear design space. Maximally-allocated designs often fail to match minimum possible latency, while using more resources. Note: no design point within the VC707 constraints exists for HyQ+arm.**

as an example. For block sizes 3, 6, and 9, the blocks cover the non-zero entries in the mass matrix sparsity pattern well without much additional zero padding. For other block sizes, the blocks are misaligned with the non-zero portions of the matrix, leading to additional zero padding and wasted cycles of work that add latency.

## 5.5 Resource Constraints Impact Performance

Putting together both the topology traversal and the topology-based matrix computational patterns, we now broaden our scope to examine latency and resource utilization for different robots across different resource constraint operating points imposed by two different computing platforms. This analysis models the full dynamics gradient accelerator architecture evaluated in Sec. 5.1 and Sec. 5.2, however we note that the patterns studied here and in Sec. 5.4 appear across a class of topology-based algorithms (Table 1).

**Methodology.** We use the Xilinx VCU118 (1182000 LUTs, 6840 DSPs) and VC707 (303600 LUTs, 2800 DSPs). We generate robot design spaces (Fig. 12), and threshold utilization by the LUT and DSP constraints. We set the threshold to 80% of total resources.

**Insight #3: Topology-Based Tuning Beats Maximum Allocation.** Figs. 16a and 16b show latency and utilization of the maximally-allocated design point and the minimum latency design point for the VCU118 and the VC707 FPGAs. We observe that the latency of the maximally allocated design point often fails to match the minimum latency possible; in fact the minimum latency design points do so by using *fewer* resources than the maximal allocation. For the accelerator architectures we evaluate, this resource over-provisioning effect is dominated by the non-linear behavior of the blocked matrix multiply, where allocating a larger block size can counterintuitively reduce performance (recall Fig. 15).

# 6 RELATED WORK AND DISCUSSION

**Relationship to Other Robotics Hardware Acceleration.** RoboShape and other robotics acceleration frameworks [26, 32, 48] are *complementary*: RoboShape uses robot topology to inform architectural mechanisms, and by doing so, flexibly deploying scalable accelerators for different resource constraints and robots. This principle can be interfaced with other acceleration frameworks. There has been recent work in hardware acceleration for the three core robotics pipeline tasks: perception [27], mapping and localization [1, 16, 26, 51], and motion planning and control [3, 31, 32, 48]. Promising work has introduced deployment constraints into the design process [1, 16, 22, 26, 32, 48]. The RoboX [48] framework maps model predictive control (MPC) optimization problems to compute hardware. Pisces [1] co-optimizes performance and power for simultaneous localization and mapping (SLAM), and the Archytas [26] flow tunes SLAM accelerators to meet power, latency, and resource specifications. Robomorphic Computing [32] lays out a pathway to design efficient functional units for dynamics based on robot joint type. Work in autonomous unmanned aerial vehicles [16, 22] explores co-designing hardware acceleration with drone weight. For distributed computing, HiveMind [41] provides a framework for coordinating hardware and software across drone swarms.

**Relationship to Sparse Tensor Algebra Accelerators.** RoboShape complements existing sparse matrix-vector multiplication (SpMV) work for large sparse matrices by addressing *two* classes of topology-defined small-to-moderate sized and sparse robotics matrices: small 6x6 joint/inertia matrices that are 40-60% sparse; and moderately-sized NxN topology-based matrices (e.g., mass matrix), where N (number of robot links) might range from 7-18 (Fig. 1) and where the matrices are either fully-dense or 50-75% sparse (Fig. 11). By contrast, most sparse tensor accelerators focus on large, very sparse matrices (e.g., neural networks with 1000s of elements that are up to 99.9% sparse) [32]. Approaches (e.g., compressed sparse row encoding) that work well for such matrices incur unsuitable overheads for RoboShape's moderately sized and sparse matrices.

**Hardware Design Frameworks and Tools.** To address emerging domains, it is essential to employ languages, tools, and frameworks to keep hardware design agile as applications evolve [18]. Hardware compiler frameworks and high-level synthesis (HLS) tools [10, 36, 37, 53] are promising approaches to automate hardware design; and can be enriched by integration with frameworks like RoboShape that encode domain-specific insights into designs.

**Beyond Rigid Body Dynamics.** There are bottleneck kernels in other motion planning techniques, e.g., collision detection for sampling-based approaches [31]. To accelerate motion planning as a whole, it is important to establish design methodologies across *all* potential bottlenecks, so designers can compose flexible hardware IP to accelerate diverse robotics applications. In this pursuit, Roboshape's principles can be integrated with existing accelerators.

# 7 CONCLUSION

RoboShape uses robot-topology-based architectural insights to address (i) the *scalability* challenge of deploying large robots within the resource constraints of different compute platforms, and (ii) the *flexibility* challenge of generating accelerators for diverse branching robot topologies. By encoding domain-specific insights, RoboShape is a pathway towards future robotics system on chip (SoC) design.

# ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation (CIFellows Grant 2030859, GRFP). All views expressed do not necessarily reflect the views of the funding organization.

# REFERENCES


[1] Bahar Asgari, Ramyad Hadidi, Nima Shoghi Ghaleshahi, and Hyesoon Kim. 2020. Pisces: power-aware implementation of slam by customizing efficient sparse algebra. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[2] Authors of RoboShape. Accessed in 2023. RoboShape Open-Source Repository. github.com/robot-acceleration/roboshape

[3] Mohammad Bakhshalipour, Seyed Borna Ehsani, Mohamad Qadri, Dominic Guri, Maxim Likhachev, and Phillip B. Gibbons. 2022. RACOD: Algorithm/Hardware Co-Design for Mobile Robot Path Planning. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. ACM, 597–609.

[4] Boston Dynamics. Accessed in 2022. Spot | Boston Dynamics. bostondynamics.com/spot

[5] Alexandre Campeau-Lecours, Hugo Lamontagne, Simon Latour, Philippe Fauteux, Véronique Maheu, François Boucher, Charles Deguire, and Louis-Joseph Caron L'Ecuyer. 2019. Kinova modular robot arms for service robotics applications. In *Rapid Automation: Concepts, Methodologies, Tools, and Applications*. IGI global, 693–719.

[6] Vinicius Cardoso, Josias Oliveira, Thomas Teixeira, Claudine Badue, Filipe Mutz, Thiago Oliveira-Santos, Lucas Veronese, and Alberto F De Souza. 2017. A model-predictive motion planner for the iara autonomous car. In *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 225–230.

[7] Justin Carpentier and Nicolas Mansard. 2018. Analytical derivatives of rigid body dynamics algorithms. In *Robotics: Science and systems (RSS 2018)*.

[8] Justin Carpentier, Guilhem Saurel, Gabriele Buondonno, Joseph Mirabel, Florent Lamiraux, Olivier Stasse, and Nicolas Mansard. 2019. The Pinocchio C++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 614–619.

[9] Robin Deits, Twan Koolen, and Russ Tedrake. 2019. LVIS: Learning from value function intervals for contact-aware robot controllers. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 7762–7768.

[10] Schuyler Eldridge, Prithayan Barua, Aliaksei Chapyzhenka, Adam Izraelevitz, Jack Koenig, Chris Lattner, Andrew Lenharth, George Leontiev, Fabian Schuiki, Ram Sunder, et al. 2021. MLIR as Hardware Compiler Infrastructure. In *Workshop on Open-Source EDA Technology (WOSET)*.

[11] Roy Featherstone. 1983. The calculation of robot dynamics using articulated-body inertias. *The international journal of robotics research* 2, 1 (1983), 13–30.

[12] Roy Featherstone. 2008. *Rigid body dynamics algorithms*. Springer.

[13] Siyuan Feng, Eric Whitman, X Xinjilefu, and Christopher G Atkeson. 2014. Optimization based full body control for the atlas robot. In *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE, 120–127.

[14] Cliff Fitzgerald. 2013. Developing baxter. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. IEEE, 1–6.

[15] Markus Giftthaler, Michael Neunert, Markus Stäuble, Jonas Buchli, and Moritz Diehl. 2018. A family of iterative gauss-newton shooting methods for nonlinear optimal control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 1–9.

[16] Ramyad Hadidi, Bahar Asgari, Sam Jijina, Adriana Amyette, Nima Shoghi, and Hyesoon Kim. 2021. Quantifying the design-space tradeoffs in autonomous drones. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 661–673.

[17] Yinhe Han, Yuxin Yang, Xiaoming Chen, and Shiqi Lian. 2020. DaDu series-fast and efficient robot accelerators. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–8.

[18] John L Hennessy and David A Patterson. 2019. A new golden age for computer architecture. *Commun. ACM* (2019).

[19] Robert K Katzschmann, Cosimo Della Santina, Yasunori Toshimitsu, Antonio Bicchi, and Daniela Rus. 2019. Dynamic motion control of multi-segment soft robots using piecewise constant curvature matched with an augmented rigid body model. In *2019 2nd IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE, 454–461.

[20] Myron King, Jamey Hicks, and John Ankcorn. 2015. Software-driven hardware development. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 13–22.

[21] Kinova. Accessed 2022. Jaco Robotic Arm. assistive.kinovarobotics.com/product/jaco-robotic-arm

[22] Srivatsan Krishnan, Zishen Wan, Kshitij Bhardwaj, Paul Whatmough, Aleksandra Faust, Sabrina M. Neuman, Gu-Yeon Wei, David Brooks, and Vijay Janapa Reddi. 2022. Automatic Domain-Specific SoC Design for Autonomous Unmanned Aerial Vehicles. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 300–317.

[23] Eric Krotkov, Douglas Hackett, Larry Jackel, Michael Perschbacher, James Pippine, Jesse Strauss, Gill Pratt, and Christopher Orlowski. 2017. The DARPA robotics challenge finals: results and perspectives. *Journal of Field Robotics* 34, 2 (2017), 229–240.

[24] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. 2016. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots* 40, 3 (2016), 429–455.

[25] KUKA AG. Accessed in 2022. LBR iiwa, KUKA AG. kuka.com/products/robotics-systems/industrial-robots/lbr-iiwa

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

[27] Yanqi Liu, Giuseppe Calderoni, and Ruth Iris Bahar. 2020. Hardware Acceleration of Monte-Carlo Sampling for Energy Efficient Robust Robot Manipulation. In *2020 30th International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 284–290.

[28] Yeting Liu, Junjie Shen, Jingwen Zhang, Xiaoguang Zhang, Taoyuanmin Zhu, and Dennis Hong. 2022. Design and control of a miniature bipedal robot with proprioceptive actuation for dynamic behaviors. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 8547–8553.

[29] JYSM Luh, M Walker, and R Paul. 1980. Resolved-acceleration control of mechanical manipulators. *IEEE Trans. Automat. Control* 25, 3 (1980), 468–474.

[30] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. 2020. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2536–2542.

[31] Sean Murray, William Floyd-Jones, Ying Qi, George Konidaris, and Daniel J Sorin. 2016. The microarchitecture of a real-time robot motion planning accelerator. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–12.

[32] Sabrina M Neuman, Brian Plancher, Thomas Bourgeat, Thierry Tambe, Srinivas Devadas, and Vijay Janapa Reddi. 2021. Robomorphic computing: a design methodology for domain-specific accelerators parameterized by robot morphology. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 674–686.

[33] Michael Neunert, Cédric De Crousaz, Fadri Furrer, Mina Kamel, Farbod Farshidian, Roland Siegwart, and Jonas Buchli. 2016. Fast nonlinear model predictive control for unified trajectory optimization and tracking. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 1398–1404.

[34] Michael Neunert, Markus Stäuble, Markus Giftthaler, Carmine D Bellicoso, Jan Carius, Christian Gehring, Marco Hutter, and Jonas Buchli. 2018. Whole-body nonlinear model predictive control through contacts for quadrupeds. *IEEE Robotics and Automation Letters* 3, 3 (2018), 1458–1465.

[35] John N Nganga and Patrick M Wensing. 2021. Accelerating second-order differential dynamic programming for rigid-body systems. *IEEE Robotics and Automation Letters* 6, 4 (2021), 7659–7666.

[36] Rachit Nigam, Samuel Thomas, Zhijing Li, and Adrian Sampson. 2021. A compiler infrastructure for accelerator generators. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. 804–817.

[37] Rishiyur Nikhil. 2004. Bluespec System Verilog: efficient, correct RTL from high level specifications. In *Proceedings. Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. MEMOCODE'04.* IEEE, 69–70.

[38] NVIDIA. Accessed in 2022. NVIDIA Deep Learning Accelerator (NVDLA). nvdla.org

[39] Zherong Pan, Bo Ren, and Dinesh Manocha. 2019. GPU-based contact-aware trajectory optimization using a smooth force model. In *Proceedings of the 18th annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 4.

[40] Amit Kumar Pandey and Rodolphe Gelin. 2018. A mass-produced sociable humanoid robot: Pepper: The first machine of its kind. *IEEE Robotics & Automation Magazine* 25, 3 (2018), 40–48.

[41] Liam Patterson, David Pigorovsky, Brian Dempsey, Nikita Lazarev, Aditya Shah, Clara Steinhoff, Ariana Bruno, Justin Hu, and Christina Delimitrou. 2022. HiveMind: a hardware-software system stack for serverless edge swarms. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 800–816.

[42] Petoi. Accessed 2022. Bittle. petoi.com/bittle

[43] Brian Plancher and Scott Kuindersma. 2018. A Performance Analysis of Parallel Differential Dynamic Programming on a GPU. In *International Workshop on the Algorithmic Foundations of Robotics (WAFR)*.

[44] B. Plancher, S. M. Neuman, T. Bourgeat, S. Kuindersma, S. Devadas, and V. Janapa Reddi. 2021. Accelerating Robot Dynamics Gradients on a CPU, GPU, and FPGA. *IEEE Robotics and Automation Letters (RA-L)* 6, 2 (2021), 2335–2342. https://doi.org/10.1109/LRA.2021.3057845

[45] Brian Plancher, Sabrina M Neuman, Radhika Ghosal, Scott Kuindersma, and Vijay Janapa Reddi. 2022. Grid: Gpu-accelerated rigid body dynamics with analytical gradients. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 6253–6260.

[46] Nicolaus A Radford, Philip Strawser, Kimberly Hambuchen, Joshua S Mehling, William K Verdeyen, A Stuart Donnan, James Holley, Jairo Sanchez, Vienny

Nguyen, Lyndon Bridgwater, Reginald Berka, Robert Ambrose, Christopher McQuin, John D. Yamokoski, Stephen Hart, Raymond Guo, Adam Parsons, Brian Wightman, Paul Dinh, Barrett Ames, Charles Blakely, Courtney Edmonson, Brett Sommers, Rochelle Rea, Chad Tobler, Heather Bibby, Brice Howard, Lei Nui, Andrew Lee, Michael Conover, Lily Truong, David Chesney, Robert Platt Jr., Gwendolyn Johnson, Chien-Liang Fok, Nicholas Paine, Luis Sentis, Eric Cousineau, Ryan Sinnet, Jordan Lack, Matthew Powell, Benjamin Morris, and Aaron Ames. 2015. Valkyrie: NASA's first bipedal humanoid robot. *Journal of Field Robotics* 32, 3 (2015), 397–419.

[47] Federico Renda, Frédéric Boyer, Jorge Dias, and Lakmal Seneviratne. 2018. Discrete cosserat approach for multisection soft manipulator dynamics. *IEEE Transactions on Robotics* 34, 6 (2018), 1518–1533.

[48] Jacob Sacks, Divya Mahajan, Richard C Lawson, and Hadi Esmaeilzadeh. 2018. Robox: an end-to-end solution to accelerate autonomous control in robotics. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 479–490.

[49] Claudio Semini, Nikos G Tsagarakis, Emanuele Guglielmino, Michele Focchi, Ferdinando Cannella, and Darwin G Caldwell. 2011. Design of HyQ–a hydraulically and electrically actuated quadruped robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 225, 6 (2011), 831–849.

[50] Anthony Stentz, Herman Herman, Alonzo Kelly, Eric Meyhofer, G Clark Haynes, David Stager, Brian Zajac, J Andrew Bagnell, Jordan Brindza, Christopher Dellin, Michael George, Jose Gonzalez-Mora, Sean Hyde, Morgan Jones, Michel Laverne, Maxim Likhachev, Levi Lister, Matt Powers, Oscar Ramos, Justin Ray, David Rice, Justin Scheifflee, Raumi Sidki, Siddhartha Srinivasa, Kyle Strabala, Jean-Philippe Tardif, Jean-Sebastien Valois, J. Michael Vande Weghe, Michael Wagner, and Carl Wellington. 2015. CHIMP, the CMU highly intelligent mobile platform. *Journal of Field Robotics* 32, 2 (2015), 209–228.

[51] Amr Suleiman, Zhengdong Zhang, Luca Carlone, Sertac Karaman, and Vivienne Sze. 2019. Navion: A 2-mw fully integrated real-time visual-inertial odometry accelerator for autonomous navigation of nano drones. *IEEE Journal of Solid-State Circuits* 54, 4 (2019), 1106–1119.

[52] Michael W Walker and David E Orin. 1982. Efficient dynamic computer simulation of robotic mechanisms. *Trans. ASME J. Dynamic Systems, Measurement, and Control* 104, 3 (1982), 205.

[53] Felix Winterstein, Samuel Bayliss, and George A Constantinides. 2013. High-level synthesis of dynamic data structures: A case study using Vivado HLS. In *2013 International conference on field-programmable technology (FPT)*. IEEE, 362–365.