Neuromorphic Quadratic Programming for Efficient and Scalable Model Predictive Control

Towards Advancing Speed and Energy Efficiency in Robotic Control

By Ashish Rao Mangalore, Gabriel Andres Fonseca Guerra, Sumedh R. Risbud, Philipp Stratmann, and Andreas Wild

Applications in robotics or other size-, weight-, and power-constrained (SWaP) autonomous systems at the edge often require real-time and low-energy solutions to large optimization problems. Event-based and memory-integrated neuromorphic architectures promise to solve such optimization problems with superior energy efficiency and performance compared to conventional von Neumann architectures. Here, we present a method to solve convex continuous optimization problems with quadratic cost functions and linear constraints on Intel's scalable neuromorphic research chip Loihi 2. When applied to model predictive control (MPC) problems for the quadruped robotic platform ANYmal, this method achieves more than two orders of magnitude reduction in the combined energy-delay product (EDP) compared to the state-of-the-art solver, OSQP, on (edge) CPUs and GPUs with solution times under 10 ms for various problem sizes. These results demonstrate the benefit of non-von Neumann architectures for robotic control applications.

Digital Object Identifier 10.1109/MRA.2024.3415005

INTRODUCTION

Convex quadratic programming (QP) has been a topic of substantial research since the 1950s. The goal of this class of problems is to optimize a quadratic cost function subject to linear constraints. The convex nature of the problems ensures that iterative updates of the variables along the gradient of the cost function are guaranteed to converge to the optimal solution. Convex QP optimization problems are particularly attractive for edge applications like robotic MPC, in which a smooth closed-loop interaction with the environment requires solving the problems within millisecond latency [1]. In such embedded applications, energy consumption is also critical for long battery life. As control systems incorporate more degrees of freedom, the underlying optimization problems grow in terms of the number of variables and require more complex cost and constraint functions. Solving increasingly more difficult optimization problems drives the search for more efficient and scalable approaches beyond conventional CPUs or GPUs.

Brain-inspired neuromorphic architectures have demonstrated significant performance and energy gains over conventional architectures for a range of optimization problems with superior scalability up to hundreds of thousands of problem variables [2]. Neuromorphic architectures derive their advantage over conventional architectures from the integration of memory with compute units to minimize data movement, massive fine-grained parallelism, and a streamlined set of supported operations as well as architectural optimizations enabling sparse and event-based computation and communication only when necessary. As a result and similar to biological brains, these novel architectures have the potential to solve extremely complicated computational problems at low power and short response times on the order of Watts and milliseconds, respectively.

Among the algorithms that excel on neuromorphic architectures are solvers for constraint satisfaction problems [3], quadratic unconstrained binary optimization [4], and different optimization problems on graphs [5]. The development of the spiking locally competitive algorithm [6], [7]—to solve LASSO with neuromorphic hardware—was the first approach to solve unconstrained convex QPs as a spiking neural network with wide-ranging applications, such as in sparse coding or signal processing. Nonetheless, the problem of solving general convex optimization with constraints on neuromorphic hardware remained unaddressed.

In this article, we discuss a framework to solve general convex QPs on neuromorphic hardware and demonstrate the implementation of a QP solver that leverages the event-based, memory-integrated, and fine-granular parallel architecture of the Intel Loihi 2 research chip [8] using the Lava open source framework (https://lava-nc.org/). We further highlight its efficacy in solving large real-world QP problems arising in the context of the MPC of the ANYmal quadrupedal robot [9] (Figure 1). We explore the conditions under which neuromorphic architectures are more suitable hardware substrates to solve convex optimization problems than traditional von Neumann-based architectures.

Convex QP problems arising in SWaP systems are conventionally solved on CPUs, even though a few solutions have been developed to leverage the parallel compute capabilities of GPUs, field-programmable gate arrays (FPGAs), and application-specified integrated circuits (ASICs). A range of high-performance QP solvers exists for CPUs, such as GUROBI [10], MOSEK [11], SCS [12], and CVXOPT [13]. Lightweight CPU solvers specifically optimized for embedded systems include qpOASES [14], ECOS [15], and OSQP [16]; such solvers avoid any dependence on large external libraries, use only basic operations (e.g., avoiding division), minimize the steps to the solution, optimize the code for mobile processors, or often parallelize their code. Unfortunately, CPUs in general—and the ones in embedded systems in particular—often do not support the degree of parallelism needed to accelerate large optimization workloads.

While GPUs offer a high degree of parallelism for optimization algorithms [17], [18], they primarily achieve their efficiency through extremely wide data paths and deep pipelining, allowing them to stream batched data from off-chip memory to process many parallel threads. However, in the case of sparse problems, the GPU resources are massively underutilized, rendering them inefficient. Similar to sparse problems, the inefficiencies of using GPUs become apparent in real-time data processing applications, too (e.g., MPC), wherein the absence of batching (or batch size 1) leads to the inadequate usage of pipelining. In addition, both CPUs and GPUs suffer from high external memory access latencies that can hardly be hidden when solving iterative algorithms on a millisecond timescale in closed-loop control.

Similar to neuromorphic processors, more specialized solutions for solving QPs have been realized using FPGAs [19], [20] and ASICs [21]. While these approaches have their merits in terms of performance or energy consumption, they require a vastly higher development effort and may serve only a single purpose (ASICs) compared to highly efficient and programmable neuromorphic processors like the Intel Loihi that can be applied to many other problems apart from QP.

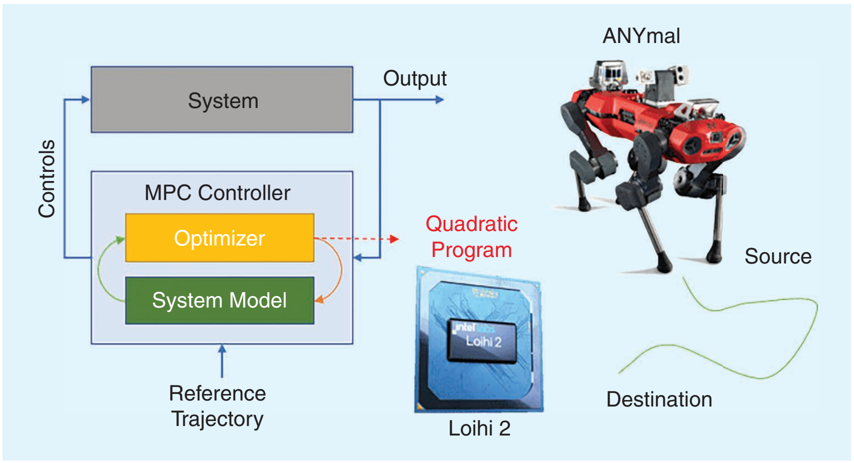

FIGURE 1. A class of convex optimization problems, namely, quadratic programs, is part of the MPC loop of the ANYmal. In this article, we explore running different sizes of these programs on Loihi 2, a neuromorphic research platform developed by Intel Labs, and see how they compare against conventional compute architectures in terms of performance and energy consumption. We show how the common motifs underlying many algorithms (such as gradient descent, primal-dual, or operator splitting methods) to solve convex-constrained optimization problems are framed as the dynamics of event-based recurrent neural networks whose steady states represent the solutions to the problems. The resulting network topology enables the implementation of diverse first-order algorithms for efficiently solving convex QP and LP problems on neuromorphic hardware.

SOLVING QPs AND LPs WITH DISTRIBUTED DISCRETE DYNAMICAL SYSTEMS

QP refers to

$$\text{minimize: } f(x) = \frac{1}{2}x^T Qx + p^T x \tag{1}$$

$$\text{subject to: } g(x) = Ax - k \le 0 \tag{2}$$

where $A \in \mathbb{R}^{M \times L}, x \in \mathbb{R}^L, p \in \mathbb{R}^L, k \in \mathbb{R}^M$, and $Q \in \mathbb{R}^{L \times L}$. This also covers the subproblem of linear programming (LP), wherein $Q = 0$. A quadratic program is convex when the feasible set is a convex set and Q is symmetric positive semidefinite $(Q \in S_+^L)$. In the following section (the "Iterative Solvers for Convex Quadratic Programs" section), we go over an iterative strategy to solve the QP from (1) and (2). In the "Neuromorphic Hardware Implementation" section, we present the implementation of our neuromorphic algorithm to solve convex QP such that it respects the constraints and strengths of Loihi 2, as determined in previous studies [7].

ITERATIVE SOLVERS FOR CONVEX QUADRATIC PROGRAMS

The unconstrained QP defined by (1) alone can be solved by first-order gradient descent. The convexity of the unconstrained problem guarantees the convergence to the global minimum

$$x_{t+1} = x_t - \alpha \cdot \nabla f(x) = (I - \alpha Q)x_t - \alpha p. \tag{3}$$

The constant $\alpha > 0$ determines the step size of the gradient descent.

When constraints are introduced, as defined by (2), pure gradient descent according to (3) may lead to constraint violations in the course of iterations. Mathematically, a constraint violation occurs if a hyperplane $A_{jx} = k_j$ is crossed (the hyperplane is defined by the normal vector A_j, which is the jth row of A [22]). To avoid such violations, one can deflect the gradient descent dynamics into the direction of the normal vector if a constraint boundary is crossed. The dynamics are explained in Figure 2(a).

To ensure that the gradient dynamics evolves in the feasible region, it is a sufficient condition to add this correction to our dynamical system

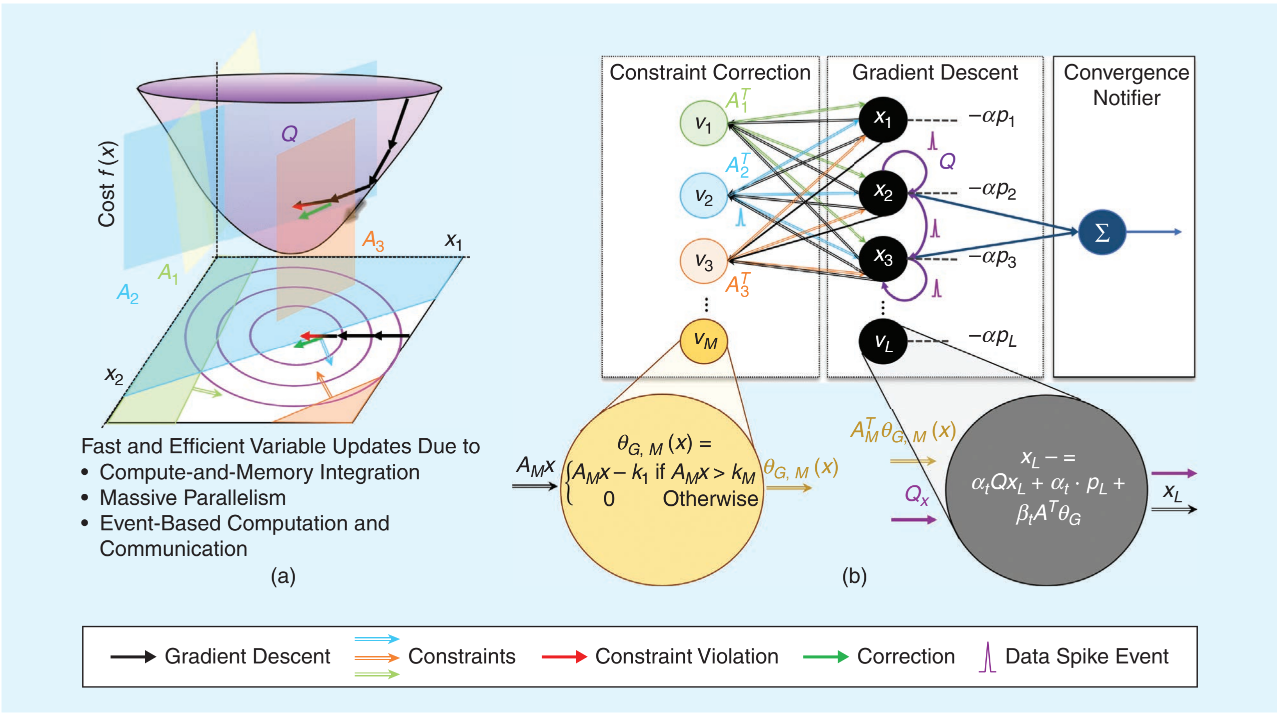

FIGURE 2. The QP solver can be formulated as a spiking neural network. (a) The 2D projection of a convex region defined by Q, as contour lines, alongside constraint planes defined by A as shaded regions. Black arrows represent the trajectory of gradient descent until it crosses the blue constraint plane, violating it by an amount proportional to the red arrow. At that point, the gradient descent is corrected by the light blue arrow contribution, which brings the network state back to the feasible region (final green arrow). (b) The gradient descent block shows the network encoding of a solver for unconstrained QP. Each black circle represents a neuron that performs gradient descent updates, dashed lines represent biases, and arrows represent synapses. For constrained QP, we add the constraint correction block, where colored circles are the neurons computing constraint violations and corresponding corrections for each constraining plane. The correcting terms are sent to the gradient descent neurons via channels indicated as colored arrows. An integration neuron (navy blue) tracks the cost of the current network state. In the case of the proportional-integral projected gradient method, each of the L gradient descent neurons encodes one scalar entry x_t^i of the variable vector x_t. It receives input x_t from the other gradient neurons via synapses with multiplicative weights defined by the vector Q_i. In addition, it receives input v_t from the constraint neurons via synapses of weight A_i^T. Its state is updated according to (6) and returned to the other gradient neurons. Each of the M constraint neurons maintains internal states w_t^j and v_t^j. It receives the states x_t from all gradient descent neurons via synapses with multiplicative synaptic weights A_j^T. It then updates its internal states w_t^j and v_t^j according to (7) and (8).

$$x_{t+1} = (I - \alpha_t Q)x_t - \alpha_t \cdot p - \beta_t A^T \theta_G(x_t) \tag{4}$$

$$\theta_G(x_t) = \begin{cases} Ax_t - k & \text{if } Ax_t > k \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $\theta_G(x_t)$ denotes the rectified linear unit function, indicating when x_{t+1} crosses a constraint hyperplane during gradient descent. Here, the hyperparameter α_t decays while β_t grows over time as the solver approaches the minimum of the state space.

The convergence rate can be accelerated by an additional integral term that accumulates the deviation of the state vector outside the feasible region, i.e., the constraint violation, similar to a proportional–integral controller. With this additional feature, one can revise the dynamics as [23]

$$x_{t+1} = \pi_{\mathbb{X}}(x_t - \alpha_t(Qx_t + p + A^T v_t)) \tag{6}$$

$$v_t = \theta_G(v_{t-1})(w_t + \beta_t(Ax_t - k)) \tag{7}$$

$$w_{t+1} = w_t + \beta_t(Ax_{t+1} - k) \tag{8}$$

where $\pi_{\mathbb{X}}$ refers to the projection of the corrected x_t into the feasible set $\mathbb{X}$. The hyperparameters α_t and β_t depend on the curvature of the cost function [23].

The dynamics described by (6) to (8) can be interpreted as that of a two-layer event-based recurrent neural network, as illustrated in Figure 2(b), which minimizes the cost function (1) using gradient descent. After algebraically eliminating w_t, the variables x_t and v_t can be identified as state variables of different types of neurons, while Q and A can be identified as matrices representing (sparse) synaptic connections between those neurons. Relying only on element-wise or matrix-vector arithmetic, this type of network can be efficiently implemented on neuromorphic hardware.

NEUROMORPHIC HARDWARE IMPLEMENTATION

Neuromorphic hardware architectures are designed to efficiently execute event-based neural networks. Example systems include SpiNNaker 1 and 2, Dynaps, BrainScaleS 1 and 2, TrueNorth, and Loihi 1 and 2 [24]. A shared characteristic of many of these architectures is a large number of parallel compute units operating out of local memory, executing either fixed-function or highly programmable neuron models. Typically, these architectures are optimized for sparse event-based computation and communication. The use of Loihi 2 for solving QP problems in this study has been motivated by prior work showing that the Loihi 2 architecture excels at solving iterative constraint optimization problems [7].

MAPPING THE QP SOLVER TO LOIHI 2

The QP solver corresponding to the dynamics of (6) to (8) was implemented on the second generation of the Intel Loihi research chip [8] (Figure 3). The massively parallel chip architecture consists of 128 independent asynchronous cores that communicate with each other by exchanging up to 24-b messages (graded spikes) using local on-chip routers. The innards of a core are shown in Figure 3. Ingress spikes from other cores are first buffered before passing through the synapse stage. This stage effectively performs a highly optimized dense or sparse matrix-vector multiplication with up to 8-b synaptic weights and up to 24-b spike activation. The resulting product can be read by the neuron stage from the dendritic accumulator stage. The neuron stage executes stateful parametrizable neural programs supporting basic arithmetic and bit-wise operations on variables up to 24 b as well as conditional logic. Programs can also generate egress messages, which are routed to other cores via the axon stage.

Variables x_t and v_t defined in (6) to (8) are encoded as 24-b state variables of two different types of neurons in the chip, i.e., the gradient descent and constraint check neurons, respectively. As seen in Figure 3, the spikes, in this case, carrying the values of states x_t and v_t, are multiplied with the synaptic weights, Q and A, and A^T, respectively, the results of which are fed into the gradient descent and constraint check neurons, respectively. The neuron programs then perform the remaining arithmetic operations involved to complete the dynamics. This colocation of memory and compute in the neuromorphic chip leads to gains in terms of energy and time to solution (TTS) for the algorithm. Note that different types of QP solvers can be implemented by merely changing the state update dynamics of the constraint correction or gradient descent neurons, which is possible due to Loihi 2's programmable neurons. The updated states are communicated only to the next neuron through synapses if necessary, in this case, when the value is nonzero.

According to Yue et al. [23], the hyperparameters, α and β, need to evolve at a certain rate for quicker convergence. This evolution, however, makes use of general-purpose division and floating-point operations, which are not supported on Loihi 2. We emulate the evolution on Loihi 2 by halving α and doubling β at a schedule, i.e., implementing simulated annealing for the hyperparameter α and geometric growth for β. The decay by halving is achieved using a bit right-shift operation, which is the same as division by two. The vector of values of the state variables of the gradient descent neurons after the convergence of the dynamics of the network is the solution to the QP. Note that to accelerate the convergence, QP problems are often first preconditioned. We choose the Ruiz preconditioner, which is known to work well for block-diagonal problems like the ones we deal with in this article. The preconditioning procedure is carried out on the host CPU in our experiments.

For QPs arising in MPC, the neuromorphic architecture of Loihi 2 already offers advantages over conventional architectures and artificial neural network accelerators because of the compute and memory colocation, which will be further discussed in the "QP for MPC" section. However, first-order type algorithms like those in (4) to (8) can be made more suitable for neuromorphic hardware by employing Σ-Δ coding, which can be implemented in Loihi 2 [8]. Doing so sparsifies the spiking activity at the cost of solution accuracy. This, in turn, improves the time and energy to solution (ETS) because

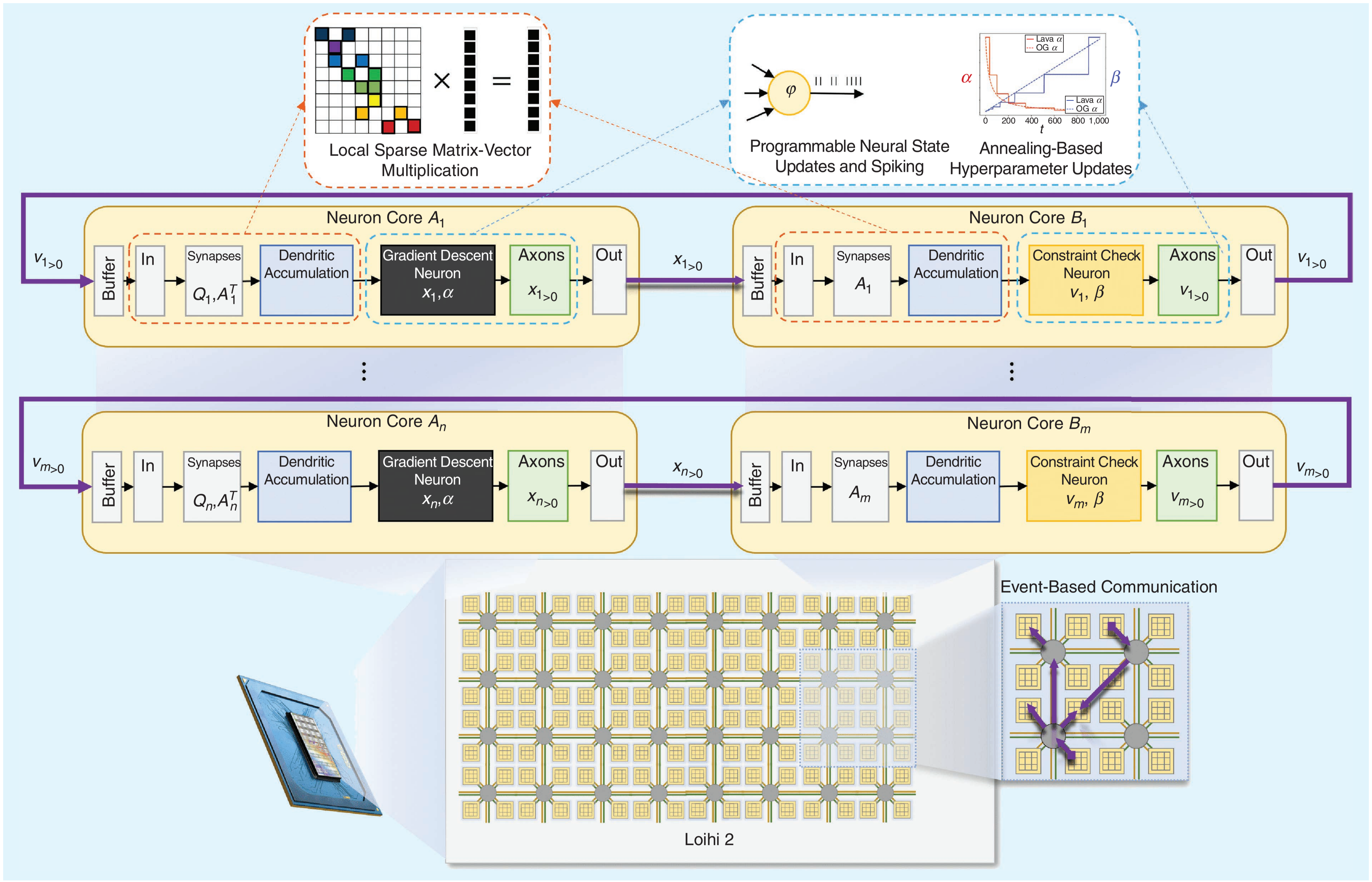

FIGURE 3. The mapping of the QP/LP solver to a neuromorphic substrate; an Intel Loihi 2 chip is shown here. Each Loihi 2 chip supports up to 1 million neurons and up to 120 million synapses depending on the model complexity, spread across 128 computing cores. Each neuron integrates synaptic-weighted input spikes from presynaptic neurons, can update local state variables via microcode programmable operations, and can send integer-valued message payloads to other postsynaptic neurons. Each core can house multiple neurons, and all these groups of neurons (and the synaptic connections corresponding to them) are spread across multiple cores. The recurrent dynamics $x_{t+1} = h(x_t)$ described by (6) to (8) combine gradient descent and constraint corrections and are mapped to the colorful and black neurons illustrated in Figure 2(b), respectively. All of these neurons are updated in parallel by different cores on the chip, shown here as yellow blocks. Multiple cores A_1 to A_n update the gradient descent neurons described by (6), while multiple cores B_1 to B_m update the constraint correction neurons described by (7) and (8). The network state evolves toward the minimum in the energy landscape [Figure 2(a)] by exchanging spikes (bottom right zoom-in). The final solution is then postprocessed to get the solution for the original nonpreconditioned problem, which is then used in the next stage of the ANYmal control pipeline.

of fewer messages/spikes being moved around and, as a consequence, fewer operations in general. However, this reduces operations within acceptable solution degradation only when the data types on chip are more precise (16-/32-b data types), which is currently not the case in Loihi 2. Therefore, in this article, we focus on results obtained without employing sigma-delta coding.

PROGRAMMING LOIHI 2 WITH LAVA

The QP solver is implemented on Loihi 2 using the open source software framework Lava for neuromorphic computing in the lava-optimization library. At its core, programming in Lava is based on asynchronous processes that communicate with each other via message passing over channels. Lava provides a cross-platform runtime and compiler to execute algorithms on different back ends, such as CPU/GPU and Loihi 2, but is also open to extension to other neuromorphic platforms. Lava optimization is one of several high-level algorithm libraries that build on top of Lava. [Lava is an open source software licensed under BSD 3-Clause and LGPL 2.1, and only proprietary modules required to run code on Loihi 2 are confidential to members of Intel's Neuromorphic Research Community (INRC). The proprietary code required to execute the solver with high performance on Loihi 2 can be accessed after joining the INRC.]

QP FOR MPC

To demonstrate the value of our neuromorphic QP solver, we apply it to solving the QPs arising in the MPC of a state-of-the-art quadrupedal ANYmal robot [9].

THE ANYMAL ROBOT

The ANYmal platform comprises a series of quadrupedal robots designed for different kinds of tasks, like inspection, surveillance, and search and rescue missions. The robots are equipped with a suite of sensors enabling autonomous navigation and perception of their surroundings as well as a comprehensive software ecosystem. The robot is commercially available and deployed in a range of industrial and commercial settings. However, the MPC for this robot has on the order of 10^3 variables. Due to this high dimensionality, each robot requires a separate Intel Core i7 processor to solve the MPC within acceptable time budgets. Further, as a mobile platform, bringing down power utilization would contribute to longer uptimes. Loihi could be used as a coprocessor to solve the MPC with very little additional burden on the battery. We therefore use data acquired from a physical ANYmal robot performing tasks provided by the ANYmal team.

PROBLEM DEFINITION

At the core of an MPC iteration is the mathematical optimization problem of minimizing the error between actual and goal trajectories, i.e., the tracking error. The optimization is subject to constraints, such as the limits posed by the rigid-body dynamics, joints, actuators, and environment. As such, the optimization problem is computationally expensive and nonlinear (and perhaps nonconvex). However, it is typically "linearized and quadratized" using techniques akin to Taylor series expansion, resulting in a convex QP problem from (1) and (2). MPC is a computationally intensive control scheme that must be executed under strict time budgets in real-time control loops. Solving the QP accounts for a major chunk of computational time in the MPC of ANYmal. The QP further increases in complexity with the larger time horizons covered by the MPC or more degrees of freedom of the robot. To meet these requirements, the MPC is solved on a dedicated laptop-class CPU. Running the QP in the MPC on Loihi would be the first step toward making computation for control more suitable for SWaP systems.

MPC uses a mathematical model of ANYmal to predict the temporal evolution of its state over a time horizon in the future. The prediction is based on the optimization of a task-specific objective function and the robot's current measured state (its position, orientation, linear and angular momenta, and joint angles [1]). From the predicted time series of the control variables, only a fraction in the beginning is used for actuating the locomotion of ANYmal. For example, a typical MPC horizon in the control loop of ANYmal of about 1 s is split into 100 time steps of 10 ms each, and only the first 20–30 ms worth of predictions are used to issue control commands. Once the robot moves, a new state measurement and sensory data are fed back to the model to generate the next set of predictions, thus completing an MPC iteration.

NEURAL AND SYNAPTIC SCALING

For an MPC horizon of $N = 100$, we require 7,248 neurons connected through 405, 504 weights at most. These resources are well within a single Loihi 2 chip's capacity of at least 64 kB of memory synaptic connections and 8 kB for neural programs per neuron core. Assuming 8-B neurons and 1-B weights, we get ~1 million neurons and ~15 million 8-b synapses for a single chip with ~123 cores. Note that the complexity of the neuron program can lead to higher memory requirements for it. For this article, we implemented programs that require 2 B per neuron, resulting in 500,000 available neurons in a single chip. This is well above the requirements for the QPs covered here.

To understand the scaling of neurons and synapses on Loihi for our QP formulation, we first calculate the number variables and weights involved in the QP problem for MPC. ANYmal's locomotion is captured by the temporal evolution of a 48-dimensional attribute vector formed by concatenating a 24-dimensional control vector (12 joint velocities and 12 contact forces) and a 24-dimensional state vector (six base poses, six momenta, and 12 joint angles). When we formulate the QP problem for an MPC iteration of the horizon with N time steps, we treat all N values of the attribute vector at every time step as a flattened vector. The causal dependence between the attribute vectors at successive time steps is captured in constructing the cost, Q, and constraints, A, matrices of the QP problem from (1) and (2). We need one neuron for each decision variable and one for each constraint variable. Therefore, the number of neurons required to map QPs of this type onto Loihi is given by

$$n_{\text{neurons}} = (N+1) * n_{\text{states}} + N * n_{\text{controls}} \tag{9}$$

where n_{states} is the size of the state vector (24 for ANYmal), and n_{control} is the size of the control vector (24 for ANYmal). Further, the maximum number of synapses required to map this QP onto Loihi is given by

$$\begin{aligned} n_{\text{synapses}} = {} & n_{\text{states}} * (2 * n_{\text{states}} + n_{\text{controls}}) * N \\ & + (n_{\text{states}} + n_{\text{controls}})^2 * N + n_{\text{states}}^2 . \end{aligned} \tag{10}$$

BENCHMARKING PROCEDURE AND DATASET

The efficiency, scalability, and speed of the Loihi QP solver were benchmarked against 1) CVXOPT, 2) a SCS solver running on a laptop CPU, and OSQP running on 3) a standard CPU, 4) an embedded CPU, and 5) against the CUDA-optimized version of it, cuOSQP, running on a GPU. The requirements are as follows:

- *Loihi 2:* This required an Oheo Gulch board running Lava v0.7.0 and Lava-Optimization v0.2.4 with an Intel Core i7-9700K CPU host with 32-GB RAM running Ubuntu 20.04.5 LTS.
- *OSQP:* OSQP v0.6.2.post9 ran on an Intel Core i7-9700K CPU @ 3.6GHz with 32-GB dynamic RAM (DRAM) running Ubuntu 20.04.5 LTS and an Nvidia Jetson Orin six-core Arm Cortex-A78AE v8.2 64-b CPU 1.5-MB L2 + 4-MB L3 CPU with 6 GB of shared RAM running Ubuntu 20.04.6 LTS.
- *cuOSQP:* cuOSQP ran on an Intel Core i7-9700K CPU @ 3.6GHz with 32-GB DRAM running Ubuntu 20.04.5 LTS with an Nvidia GeForce RTX 2070 Super GPU with 8 GB of RAM and cuda 10.2.

The evaluation metrics were TTS, ETS, and EDP for different problem sizes. OSQP is a CPU-based state-of-the-art solver for convex QP problems with linear inequality constraints [16]. We ultimately use the OSQP solver as a reference for our Loihi-based solver since it was the best-performing CPU solver.

The dataset used in this article consists of data from 2,173 individual time steps of the MPC of an ANYmal robot [25]. The dataset consists of the matrices obtained after the linearization and quadratization of a nonlinear objective function and penalizes deviation from a reference trajectory as well as the ANYmal dynamics. We tile these matrices in an appropriate manner to construct the Q matrix of the QP objective function from (1) and the A matrix of the constraints from (2). This tiling is demonstrated in Figure 4. The derivation and explanation of this tiling have been omitted here since they are beyond the scope of this article. It can be seen that these matrices are mostly block-diagonally populated and very sparse. This entails sparse matrix-vector multiplication, making a suitable candidate for Loihi 2. The vectors p and k from the cost and constraints, respectively, are constructed by stacking the vectors associated with the tracking cost and the dynamics in each stage of the MPC.

We investigate how well the performance of the solver scales with problem size by varying the number of variables. In general, the problem size is determined by the time horizon and the degrees of freedom of the robot. Here, we choose six horizon lengths, $N = 5, 50, 75, 100, 150, 175$ time steps, resulting in problems with $264, 2424, 3624, 4824, 7224, 8424$ variables, respectively. We use this as a proxy for a robot with higher degrees of freedom. Most QPs in the dataset are similar in difficulty (similar condition numbers). Therefore, for each horizon length, we have chosen 10 different QPs of varying difficulty that best represent all the problems in the dataset. In summary, we have a dataset containing a total of 60 representative QP problems of six different sizes from the ANYmal dataset, spanning from ~250 to ~8,500 variables.

CONVERGENCE, POWER, AND PERFORMANCE ANALYSIS

In the context of solving a type of QP problem, i.e., LASSO problems, on Loihi [7], [26], Loihi-based solvers rapidly converge to approximate solutions, but limited precision available on the chip, as explained in the "Neuromorphic Hardware Implementation" section, limits the convergence to a finite

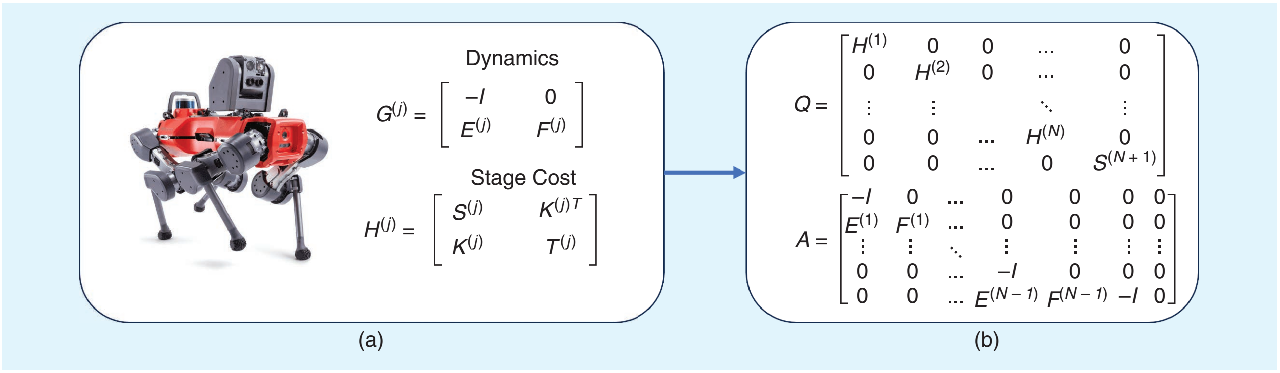
Dynamics
$G^{(j)} = \begin{bmatrix} -I & 0 \\ E^{(j)} & F^{(j)} \end{bmatrix}$
Stage Cost
$H^{(j)} = \begin{bmatrix} S^{(j)} & K^{(j)T} \\ K^{(j)} & T^{(j)} \end{bmatrix}$
$Q = \begin{bmatrix} H^{(1)} & 0 & 0 & \cdots & 0 \\ 0 & H^{(2)} & 0 & \cdots & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & H^{(N)} & 0 \\ 0 & 0 & \cdots & 0 & S^{(N+1)} \end{bmatrix}$
$A = \begin{bmatrix} -I & 0 & \cdots & 0 & 0 & 0 & 0 \\ E^{(1)} & F^{(1)} & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & -I & 0 & 0 & 0 \\ 0 & 0 & \cdots & E^{(N-1)} & F^{(N-1)} & -I & 0 \end{bmatrix}$
(a)
(b)

FIGURE 4. In (a), the matrices pertaining to stage costs, $H^{(j)}$, and dynamics, $G^{(j)}$, of ANYmal are associated with every time step of the MPC. $E^{(j)}$ and $F^{(j)}$ represent the dynamics of the robot at time step j. $S^{(j)}$, $K^{(j)}$, and $T^{(j)}$ are the stage costs at time step j. In (b), it is seen how these matrices are concatenated together to form large Q and A matrices for cost and constraints of the QP [(1) and (2)], respectively. Note that the matrices in (b) are sparsely populated, making these types of problems well-suited for Loihi.

optimality gap. The same applies to Loihi 2 as well. However, the CPU solver operates with 64-b floating-point precision. To make a fair comparison across all platforms, we run all solvers until they have converged up to the same accuracy of 8% from the true solution. This accuracy was chosen in alignment with the precision needed to make MPC robust for various control applications [27]. For MPC applications that need higher accuracy, we can in general increase the precision of the Loihi 2 solution. For this, Loihi 2 could represent each 64-b variable by allocating several of its 24-b states and represent each 64-b matrix weight by combining several of its 8-b synaptic weights. To verify that successive MPC iterations converge stably even with limited precision, we randomly perturbed Q and A matrices for each successive iteration and verified that the cost and constraint satisfaction keep converging with the warm starting of the QP solver. This suggests that approximate solutions are sufficient in closed-loop control applications.

Figure 5 demonstrates that the Loihi QP solver rapidly converges within ~55 iterations on average, where a solution is considered to have converged when the solver reaches within 8% or less of the OSQP reference solver. The solver continues to converge further, closer to optimality but with a slower convergence rate. Similar to the cost, constraint satisfaction continues to improve beyond our definition of convergence. In the subsequent performance comparison results, we have used the fact that the Loihi QP solver converges at 55 iterations and compared its performance with that of OSQP.

Figure 6(a) shows the TTS for the same set of QP problems of increasing problem size. As the MPC time horizon N—and thus the number of QP variables—increases, TTS increases roughly linearly for both OSQP and the Loihi QP solver. While the OSQP solver solves small problems faster than Loihi 2 in the submillisecond regime, the time it takes OSQP to find the optimal solution (black dashed line) grows rapidly beyond Loihi's TTS. Let us consider approximate solutions from Loihi for usage in iterative MPC with warm starts. In this case, OSQP's TTS grows less rapidly but still exceeds Loihi toward the largest problem size. We see that the solution time for a standard GPU is orders of magnitude higher than even a CPU for problems of this scale for reasons mentioned in the "Introduction" section. For the embedded CPU, the TTS follows the same trend as the laptop CPU albeit even slower.

Figure 6(b) shows the corresponding ETS for increasing problem complexity. The ETSs for both OSQP and the Loihi QP increase with problem size as more cores on Loihi are utilized. Loihi achieves a significant advantage compared to laptop-class CPUs in energy of more than 200× for all problem sizes. Energy consumption of the Loihi QP solver is composed of static and dynamic energy: $E = E_{\text{static}} + E_{\text{dyn}}$. Static energy is mostly governed by chip leakage power on Loihi $E_{\text{static}} = P_{\text{static}} \cdot TTS \cdot c_{\text{active}}/c_{\text{total}}$, where c is the number of active and total cores, respectively. Measurements for this analysis have been obtained from early silicon samples that still have highly variable leakage characteristics, leading to a high $P_{\text{static}} = 1.4$ W per chip at the lowest operating voltage, and thus are less relevant for this analysis. Dynamic energy is governed by the number of operations within Loihi cores to update neurons and route traffic between cores.

Further, Figure 6(c) shows a 520× EDP advantage for the largest problem size when using Loihi 2 in place of a laptop

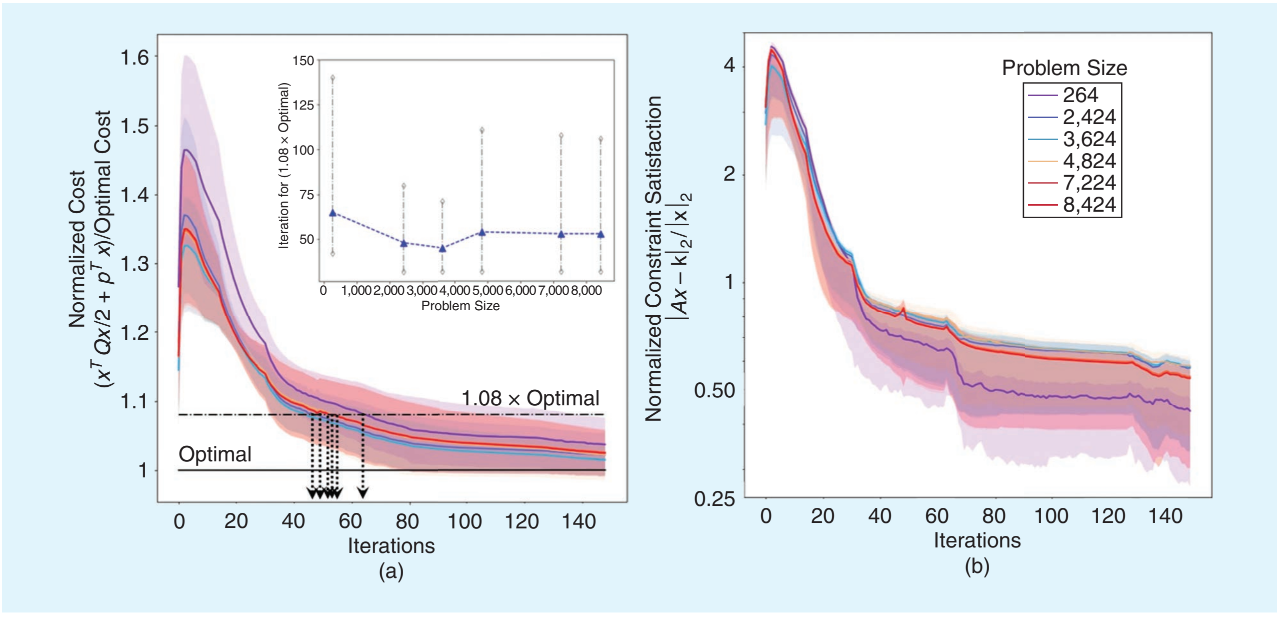

FIGURE 5. A convergence analysis of the Lava QP solver for 60 QPs in the MPC loops of ANYmal as measured by (a) the cost [(1)] normalized by OSQP's optimal cost and (b) the number of constraint violations [(2)] normalized by the L2-norm of the final solution is shown here. Solid lines represent the mean for cost and constraints respectively for 60 different problem sizes N = 5, 50, 75, 100, 150, 175, translating into 264–8,424 QP variables, as indicated by the legend. Shaded regions represent one standard deviation from the mean. The horizontal black lines correspond to the optimal cost. The inset in (a) depicts the number of iterations required for different problem sizes to reach an optimality gap of 8% with respect to OSQP.

CPU. It can be seen in Figure 6(c) that the EDP curves for CPUs (embedded and laptop) closely resemble each other. The laptop and embedded CPU thus trade the TTS against the ETS, but the combined EDP is characteristic of CPUs. For a GPU, both the ETS and EDP are orders of magnitude higher mainly.

By increasing Loihi's operating voltage from $V_{DD} = 0.6-0.8$ V, Loihi's TTS improves by about 47%. At the same time, dynamic energy increases by a factor of 1.9× for increasing the operating voltage for a problem of 4,824 variables. The total energy remains largely unchanged since the dominating leakage energy drops with lower TTS in this example. Figure 6(d) and (e) further illustrates how the level of multicore parallelism allows one to trade performance (TTS) for chip resources. As the number of neurons per core decreases and the number of cores increases proportionately, TTS improves from 17.5 to 2.6 ms for a problem size of 4,824 variables. The slightly less than linear decrease in TTS results from additional multicore synchronization and data traffic between parallel Loihi cores. Loihi's highly configurable nature allows users to trade off energy consumption (ETS), speed (TTS), and chip resource utilization, which can be useful in SWaP applications to select the optimal task-specific operating point.

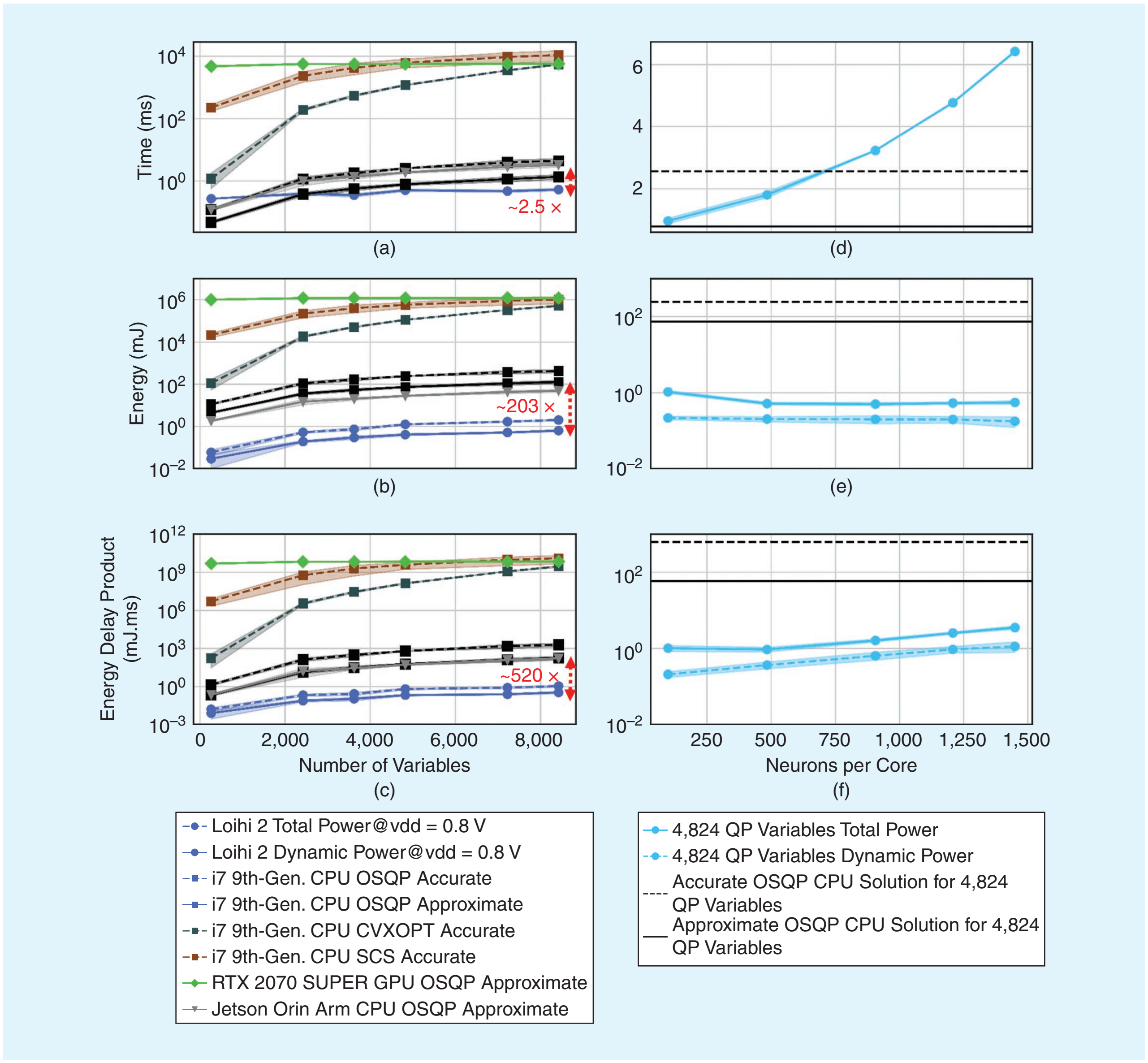

FIGURE 6. (a)–(c) Demonstrate the advantages of the Loihi 2-based solver over the CPU solver on a laptop-class (black, dark slate gray, and brown) and edge-level CPU (gray) and also over its GPU-based implementation (lime green) for increasing problems size on different metrics. OSQP was the best-performing solver and was therefore chosen as the reference. As seen from (d)–(f), there exists a parallelization-solution time tradeoff, but a slow solver beyond a certain point could lead to increased EDP. Note that the speed of execution can be increased by operating at higher voltages. (a) TTS. (b) ETS. (c) EDP to solution. (d) TTS on Loihi 2 versus parallelization. (e) ETS on Loihi 2 versus parallelization. (f) EDP to solution on Loihi 2 versus parallelization.

CONCLUSION

We have developed an efficient and scalable approach for solving convex quadratic constraint optimization problems with neuromorphic chips. The benefits of neuromorphic systems can be exploited for optimization algorithms by interpreting the structure of many iterative optimization approaches—such as gradient descent and primal-dual updates—as the dynamics of recurrent neural networks. To demonstrate the benefits of this framework, we implemented a QP solver on Intel's Loihi 2 research chip and made it publicly available as part of the Lava software framework. To illustrate its benefits, we applied the QP solver to tackle workloads that arose in the MPC pipeline of ANYmal during its normal operation. Our Loihi solver solved the workloads with a similar speed as the state-of-the-art solver OSQP on a modern CPU [28], which was the best von Neumann-based candidate for comparison, for approximate solutions while consuming about two orders of magnitude less energy. Most notably, the time and energy to a solution on the Loihi scale are better with increasing problem complexity than OSQP on CPU.

The performance, energy, and scaling advantages on Loihi result primarily from its massively parallel, event-based, and memory-integrated hardware architecture. Multicore parallelism allows one to quickly and independently update a large number of variables. The hardware is further optimized for arithmetic on sparse matrices, as present in most large real-world QP workloads [29]. Its event-based computation and communication further eliminate redundant data traffic routing and computation. Together, these features support scaling to large problem sizes with less overhead than on conventional computer architectures. In addition, the memory-integrated compute architecture minimizes the energy and latency required to access data for algorithmic iterations. With the observed gains in computational efficiency, interpretable model-based controllers could see increased adoption over more opaque model-free controllers like reinforcement learning, thus increasing the safety of applications.

The most significant limitation of the current approach is the limited bit precision of state variables and synaptic connections on neuromorphic architectures like Loihi. This can lead to lower solution optimality than solvers on conventional floating-point architectures. Nevertheless, our solver consistently achieved solutions that deviated by fewer than 8% from the true optimal solution for QPs extracted during a real-world operation of a physical ANYmal robot. For applications that require higher precision, the limited precision of the Loihi solver can be circumvented by allocating multiple low-bit variables to effectively achieve higher precision arithmetic or enabling higher precision arithmetic in general in future chip generations. Nonetheless, preliminary observations in closed-loop control simulations of ANYmal hint that fast approximate solutions are sufficient for iterative scenarios like MPC with sequential warm starts. We hypothesize that this is partly because the result of each MPC cycle is used only to determine motor commands for the next few time steps despite optimizing over time horizons of more than 100 time steps.

After controlling the next few steps, new sensory recordings are taken, and the next MPC cycle can iteratively correct any errors resulting from, e.g., the linearized mechanical robotic model, environmental perturbations, and limited bit precision. We invite robotics labs interested in assessing the performance of our fixed-point QP solver on their cyberphysical systems to become part of the INRC. By joining the INRC, labs will gain access to the proprietary code for running the solver on Loihi 2. They can then apply for borrowing a Loihi 2 board for closed-loop testing in their MPC applications. The implementation of our solver in Lava, a software framework built to support asynchronous message passing, bodes well for integration with the asynchronous publish and subscribe system of ROS. Further, with floating-point support, the current gains observed with compute-memory colocation can be bolstered by employing sigma-delta coding, which would reduce spiking activity in the chip and, consequently, the TTS and ETS.

In applications where faster solves are required, the massive energy advantage of neuromorphic architectures can be traded for additional speed by executing multiple solver instances in parallel with different initial conditions and selecting the best and fastest solution. In general, the customizability of neuromorphic architectures like Intel's Loihi 2 allows one to trade solution optimality, energy, speed, and chip resource utilization seamlessly against each other to achieve optimal operating conditions depending on the requirements of the task.

ACKNOWLEDGMENT

We would like to thank Farbod Farshidian, a former member of the Robotics Systems Lab, ETH Zürich, for providing the data used in this article and for his inputs on ANYmal. We appreciate the valuable feedback on the manuscript from Prof. Alin Albu-Schäffer from the German Aerospace Center (DLR) and TU Munich. We thank Yulia Sandamirskaya (ZHAW, Zurich) and Akshit Saradagi (Luleå University of Technology), who participated in discussions and also provided some useful suggestions for our work. Sumit Shrestha Bam and Leobardo E. Campos Macias from the Neuromorphic Computing Lab, Intel Labs provided support with the Loihi and Orin platforms. We also thank all the members of the Neuromorphic Computing Lab, Intel Labs in general for their support in developing features for Lava and for assistance with hardware-related issues. Ashish Rao Mangalore is the corresponding author.

AUTHORS

Ashish Rao Mangalore, School of Computation, Information, and Technology, Technische Universität München, 80333 Munich, Germany, and Neuromorphic Computing Lab, Intel Labs, 85579 Neubiberg, Germany. E-mail: ashish.rao.mangalore@intel.com.

Gabriel Andres Fonseca, Neuromorphic Computing Lab, Intel Labs, 85579 Neubiberg, Germany. E-mail: gabriel.fonseca.guerra@intel.com.

Sumedh R. Risbud, Neuromorphic Computing Lab, Intel Labs, Santa Clara, CA 95052 USA. E-mail: sumedh.risbud@intel.com.

Philipp Stratmann, Neuromorphic Computing Lab, Intel Labs, 85579 Neubiberg, Germany. E-mail: philipp.stratmann@intel.com.

Andreas Wild, Neuromorphic Computing Lab, Intel Labs, Hillsboro, OR 97124 USA. E-mail: andreas.wild@intel.com.

REFERENCES

[1] J.-P. Sleiman, F. Farshidian, M. V. Minniti, and M. Hutter, "A unified MPC framework for whole-body dynamic locomotion and manipulation," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4688–4695, Jul. 2021, doi: 10.1109/LRA.2021.3068908.

[2] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Comput. Sci.*, vol. 2, no. 1, pp. 10–19, Jan. 2022, doi: 10.1038/s43588-021-00184-y.

[3] G. A. Fonseca Guerra and S. B. Furber, "Using stochastic spiking neural networks on spinnaker to solve constraint satisfaction problems," *Frontiers Neurosci.*, vol. 11, Dec. 2017, Art. no. 714, doi: 10.3389/fnins.2017.00714.

[4] M. Z. Alom, B. Van Essen, A. T. Moody, D. P. Widemann, and T. M. Taha, "Quadratic unconstrained binary optimization (QUBO) on neuromorphic computing system," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN),* 2017, pp. 3922–3929, doi: 10.1109/IJCNN.2017.7966350.

[5] S. M. Mniszewski, "Graph partitioning as quadratic unconstrained binary optimization (QUBO) on spiking neuromorphic hardware," in *Proc. Int. Conf. Neuromorphic Syst. (ICONS)*, New York, NY, USA: ACM, 2019, pp. 1–5, doi: 10.1145/3354265.3354269.

[6] P. T. P. Tang, T.-H. Lin, and M. Davies, "Sparse coding by spiking neural networks: Convergence theory and computational results," 2017, *arXiv:1705.05475*.

[7] M. Davies et al., "Advancing neuromorphic computing with Loihi: A survey of results and outlook," *Proc. IEEE*, vol. 109, no. 5, pp. 911–934, May 2021, doi: 10.1109/JPROC.2021.3067593.

[8] G. Orchard et al., "Efficient neuromorphic signal processing with Loihi 2," in *Proc. IEEE Workshop Signal Process. Syst. (SiPS)*, 2021, pp. 254–259, doi: 10.1109/SiPS52927.2021.00053.

[9] M. Hutter et al., "Anymal-a highly mobile and dynamic quadrupedal robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Piscataway, NJ, USA: IEEE Press, 2016, pp. 38–44, doi: 10.1109/IROS.2016.7758092.

[10] "Gurobi optimizer reference manual." Gurobi Optimization, LLC. [Online]. Available: https://www.gurobi.com

[11] "The MOSEK optimization toolbox. Version 10.0." Mosek ApS. [Online]. Available: https://www.mosek.com/documentation/

[12] B. O'Donoghue, "Operator splitting for a homogeneous embedding of the linear complementarity problem," *SIAM J. Optim.*, vol. 31, pp. 1999–2023, Aug. 2021.

[13] M. S. Andersen, J. Dahl, and L. Vandenberghe. "CVXOPT: A Python package for convex optimization." CVX Research. Accessed: May 24, 2024. [Online]. Available: https://cvxopt.org/

[14] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Math. Program. Comput.*, vol. 6, no. 4, pp. 327–363, Dec. 2014, doi: 10.1007/s12532-014-0071-1.

[15] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An OSCP solver for embedded systems," in *Proc. Eur. Control Conf. (ECC)*, 2013, pp. 3071–3076, doi: 10.23919/ECC.2013.6669541.

[16] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Math. Program. Comput.*, vol. 12, no. 4, pp. 637–672, Dec. 2020, doi: 10.1007/s12532-020-00179-2.

[17] L. Yu, A. Goldsmith, and S. Di Cairano, "Efficient convex optimization on GPUs for embedded model predictive control," in *Proc. Gener. Purpose GPUs (GPGPU-10)*, New York, NY, USA: ACM, 2017, pp. 12–21, doi: 10.1145/3038228.3038234.

[18] M. Schubiger, G. Banjac, and J. Lygeros, "GPU acceleration of ADMM for large-scale quadratic programming," *J. Parallel Distrib. Comput.*, vol. 144, pp. 55–67, Oct. 2020, doi: 10.1016/j.jpdc.2020.05.021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731520303063

[19] I. McInerney, G. A. Constantinides, and E. C. Kerrigan, "A survey of the implementation of linear model predictive control on FPGAs," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 381–387, 2018, doi: 10.1016/j.ifacol.2018.11.063.

[20] S. Lucia, D. Navarro, Lucía, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE Trans. Ind. Informat.*, vol. 14, no. 1, pp. 137–145, Jan. 2018, doi: 10.1109/TII.2017.2719940.

[21] T. Skibik and A. A. Adegbege, "An architecture for analog VLSI implementation of embedded model predictive control," in *Proc. Annu. Amer. Control Conf. (ACC)*, 2018, pp. 4676–4681, doi: 10.23919/ACC.2018.8431320.

[22] A. Mancoo, S. Keemink, and C. K. Machens, "Understanding spiking networks through convex optimization," in *Proc. Adv. Neural Inform. Process. Syst.*, 2020, vol. 33, pp. 8824–8835.

[23] Y. Yu, P. Elango, and B. Aç Ikmeşe, "Proportional-integral projected gradient method for model predictive control," *IEEE Control Syst. Lett.*, vol. 5, no. 6, pp. 2174–2179, Dec. 2021, doi: 10.1109/LCSYS.2020.3044977.

[24] A. Shrestha, H. Fang, Z. Mei, D. P. Rider, Q. Wu, and Q. Qiu, "A survey on neuromorphic computing: Models and hardware," *IEEE Circuits Syst. Mag.*, vol. 22, no. 2, pp. 6–35, 2nd Quart. 2022, doi: 10.1109/MCAS.2022.3166331.

[25] F. Farshidian et al., "OCS2: An open source library for optimal control of switched systems." GitHub. [Online]. Available: https://github.com/leggedrobotics/ocs2

[26] M. Davies et al., "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018, doi: 10.1109/MM.2018.112130359.

[27] M. Diehl, R. Findeisen, and F. Allgöwer, "A stabilizing real-time implementation of nonlinear model predictive control," in *Proc. Real-Time PDE-Constrained Optim.*, Philadelphia, PA, USA: SIAM, 2007, pp. 25–52.

[28] Y. de Viragh, M. Bjelonic, C. D. Bellicoso, F. Jenelten, and M. Hutter, "Trajectory optimization for wheeled-legged quadrupedal robots using linearized ZMP constraints," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1633–1640, Apr. 2019, doi: 10.1109/LRA.2019.2896721.

[29] K. Cheshmi, D. M. Kaufman, S. Kamil, and M. M. Dehnavi, "NASOQ: Numerically accurate sparsity-oriented QP solver," *ACM Trans. Graph*, vol. 39, no. 4, Aug. 2020, doi: 10.1145/3386569.3392486.