Parallelized Multi-Agent Bayesian Optimization in Lava

1st Shay Snyder
Electrical and Computer Engineering
George Mason University
Fairfax, USA
ssnyde9@gmu.edu

2nd Derek Gobin
Electrical and Computer Engineering
George Mason University
Fairfax, USA
dgobin@gmu.edu

3rd Victoria Clerico
Electrical and Computer Engineering
George Mason University
Fairfax, USA
mclerico@gmu.edu

4th Sumedh R. Risbud
Neuromorphic Computing Lab
Intel Labs
Santa Clara, USA
sumedh.risbud@intel.com

5th Maryam Parsa
Electrical and Computer Engineering
George Mason University
Fairfax, USA
mparsa@gmu.edu

2024 International Conference on Neuromorphic Systems (ICONS) | 979-8-3503-6865-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICONS62911.2024.00051

***Abstract*—In parallel with the continuously increasing parameter space dimensionality, search and optimization algorithms should support distributed parameter evaluations to reduce cumulative runtime. Intel's neuromorphic optimization library, Lava-Optimization, was introduced as an abstract optimization system compatible with neuromorphic systems developed in the broader Lava software framework. In this work, we introduce Lava Multi-Agent Optimization (LMAO) with native support for distributed parameter evaluations communicating with a central Bayesian optimization system. LMAO provides an abstract framework for deploying distributed optimization and search algorithms within the Lava software framework. Moreover, LMAO introduces support for random and grid search along with process connections across multiple levels of mathematical precision. We evaluate the algorithmic performance of LMAO with a traditional non-convex optimization problem, a fixed-precision transductive spiking graph neural network for citation graph classification, and a neuromorphic satellite scheduling problem. Our results highlight LMAO's efficient scaling to multiple processes, reducing cumulative runtime and minimizing the likelihood of converging to local optima.**

I. Introduction

Many of today's most interesting problems require solutions to high dimensional and non-linear systems that determine the optimal parameter configuration. Multiple areas such as autonomous robotics [1], graph neural networks [2], evolutionary algorithms [3], and physics-informed neural networks [4] are limited by the time expenditure from individual parameter evaluations. Rather than traditional procedural approaches like random search [5] or grid search [6], modern techniques employ heuristic algorithms making informed decisions from prior knowledge, such as Bayesian optimization (BO) [7] with roots in Bayesian statistics [8]. While BO reduces the quantity of problem evaluations by orders of magnitude, many problems still face runtime issues where the reduced number of synchronous evaluations is not enough to compensate for the immense time required by individual evaluations [9].

Intel's neuromorphic software framework, Lava, was introduced as an abstract software framework for developing neuromorphic systems. In this work, we introduce **L**ava **M**ulti-**A**gent **O**ptimization (LMAO), a novel framework for evaluating parameter configurations across multiple asynchronous processes whose results are aggregated into a single optimizer or search algorithm. This framework is completely open-sourced through GitHub[1]. We evaluate the performance improvements and operational characteristics of LMAO with the Ackley function [10], a fixed-precision transductive spiking neural network for citation graph classification [11], and a satellite scheduling problem using quadratic unconstrained binary optimization [12].

In summary, the major contributions of this paper are:

- We introduce Lava Multi-Agent Optimization (LMAO) with support for distributed optimization.
- We demonstrate the performance of LMAO with the Ackley function [10], a transductive spiking graph neural network [11], and a QUBO optimization problem for satellite scheduling [12].

II. Architecture of LMAO within the Lava Software Framework

Intel's neuromorphic software framework, Lava [12], provides an abstract interface for building interconnected systems of event-based computational elements. The lowest-level building blocks are *Processes* which provide a blueprint of inputs, outputs, and internal variables. Lava provides a base library of ports allowing inter-process communication. *In-Ports* receive information from other processes whereas *Out-Ports* transmit information to other processes. Individual *Process* functionality is defined within *Process Models*[2]. Moreover,

[1]Code available at https://github.com/Parsa-Research-Laboratory/lmao.

[2]See http://lava-nc.org for details about Lava concepts like *Processes* and *Process Models*

979-8-3503-6865-9/24/$31.00 ©2024 IEEE
DOI 10.1109/ICONS62911.2024.00051

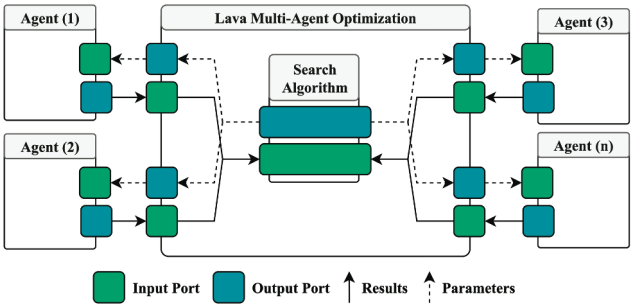

Fig. 1. Multiple independently operating agents processing different hyper-parameters from the central search algorithm with LMAO.

Process Models are architecture specific so the same *Process* can have multiple *Process Models* for execution on different hardware platforms such as central processing units or Loihi 2 neurocores.

Lava **M**ulti-**A**gent **O**ptimization (LMAO) introduces the general-purpose *Solver*. Serving as the single point of entry into the LMAO framework, the *Solver* is a contract between users and developers. This utility provides an abstract interface where users define various parameters such as number of iterations, number of initial points, parameter search spaces, and optimization algorithm types.

Rather than being limited to sequential parameter evaluations [13], LMAO introduces support for multiple, parallel agents communicating with a central optimization or search algorithm. A high-level flowchart of this process is presented in Figure 1. Controlled by the *numAgents* parameter, users can distribute evaluations to multiple asynchronous agents and increase the effective number of evaluations per time step. The LMAO backend supports this functionality by dynamically creating pairs of *In-Ports* and *Out-Ports* for each process and using the Lava runtime framework to distribute agents across multiple processes.

As shown in Algorithm 1, the system is initialized by sending a unique initial point to each agent. With agents executing asynchronously [14], they evaluate the received parameters and return the corresponding values on stochastic time intervals. Simultaneously, the search algorithm probes each *In-Port* from each agent process. If the port has received an evaluated parameter combination, the data is decoded and used to update the search algorithm. This process is repeated until all initial points have been evaluated.

With all initial points evaluated, LMAO uses learned knowledge to heuristically explore the parameter space. As shown in Algorithm 2 and using the constant liar strategy [15], unique, unknown points are sampled and transmitted to each agent for parallelized evaluation. Upon completion, the evaluated points are used to update the model. This process continues until the desired number of iterations is reached wherein all processes are stopped and results are returned to the user.

Algorithm 1 Agent Initialization & Initial Point Sampling

Require: $\text{numIps} = \{\text{numIps} \in \mathbb{N} | \text{numIps} \geq 1\}$
Require: $\text{numIps} \geq \text{numIterations}$
Require: $\text{numAgents} \in \mathbb{N}$
Require: $\text{numAgents} \geq 1$
  opt ← getOptimizer()
  ipQueue ← opt.getInitialPoints(numIps)
  completedIters ← 0
  for $i = 0$ **to** numAgents - 1 **do**
    outPort ← getOutPort(i)
    nextPoint ← ipQueue.pop()
    outPort.send(nextPoint)
  end for
  repeat
    for $i = 0$ **to** $numAgents - 1$ **do**
      inPort ← getInPort(i)
      if inPort.probe() is **false then**
        continue
      end if
      point ← inPort.recv()
      opt.update(point)
      completedIters = completedIters + 1
      if ipQueue.nonEmpty() **then**
        outPort ← getOutport(i)
        nextPoint ← ipQueue.pop()
        outPort.send(nextPoint)
      end if
    end for
  until completedIters ≥ numIps

III. Results & Discussion

We evaluate the performance of Lava Multi-Agent Optimization with a traditional non-convex optimization problem [10], a spiking graph neural network for citation graph classification [11], and a quadratic unconstrained binary optimization problem in the Lava-Optimization library [12]. All experiments were conducted with a desktop computer equipped with an AMD Ryzen 7 3700x processor and 64GB of quad-channel DDR4 memory.

A. *Traditional Non-Convex Optimization*

The Ackley [10] function is a classic non-convex optimization problem with widespread usage. We evaluate this function with Bayesian Optimization (BO) across a varying quantity of agents. The search space is continuous, real values across the range of each problem dimension. For a total of 50 optimization iterations, we configure the optimizer to sample 10 initial points before using BO to intelligently explore based on prior knowledge. BO is able to successfully learn each function and converge arbitrarily close to the global minima. As shown in Figure 2A, we perform the same experiment across a varying quantity of agents from 1 to 10. Given the low computational complexity of individual function evaluations,

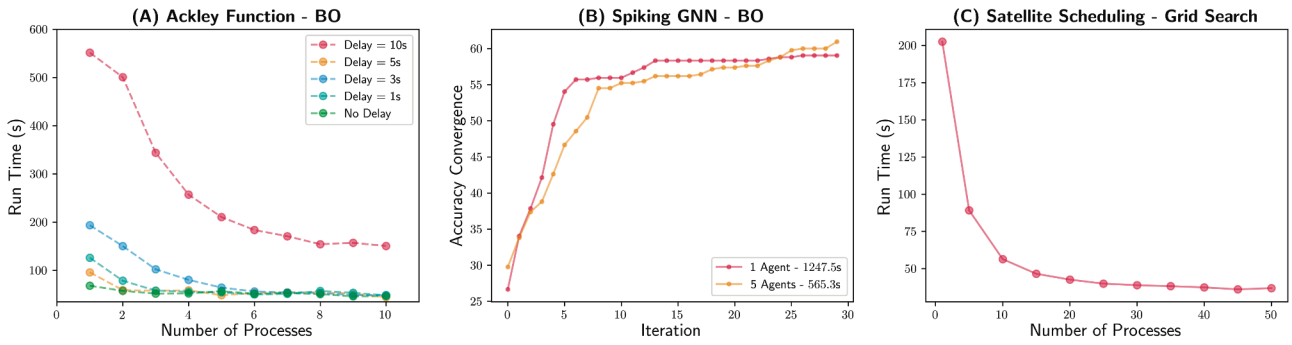

Fig. 2. (A) The runtime latency of LMAO using BO on the Ackley function [10] with varying amounts of manually induced delay. (B) The accuracy convergence of single and multi-agent BO for citation graph classification [11]. (C) Grid search execution times with satellite scheduling [12] across different numbers of processes.

Algorithm 2 Heuristic Search

Require: numIps $= \{\text{numIps} \in \mathbb{N} | \text{numIps} \geq 1\}$
Require: numIter $= \{\text{numIter} \in \mathbb{N} | \text{numIter} > \text{numIps}\}$
Require: numAgents $\in \mathbb{N}$
Require: numAgents ≥ 1
  opt $\leftarrow$ getOptimizer()
  completedIters $\leftarrow$ numIps
  repeat
    numPoints $\leftarrow$ min(numIter - completedIters, numAgents)
    if numPoints ≤ 0 **then**
      return
    end if
    unknownPoints $\leftarrow$ opt.ask(numPoints)
    for $i = 0$ **to** numPoints - 1 **do**
      outPort $\leftarrow$ getOutPort(i)
      nextPoint $\leftarrow$ unknownPoints.pop()
      outPort.send(nextPoint)
    end for
    numComplete $\leftarrow 0$
    repeat
      inPort $\leftarrow$ getInPort(i)
      if inPort.probe() is **true then**
        point $\leftarrow$ inPort.recv()
        opt.update(point)
        completedIters $=$ completedIters $+ 1$
      end if
    until numComplete $\geq$ numPoints
  until completedIters $\geq$ numIter

the overhead of spawning multiple processes and the latency of updating the Gaussian model outweigh the benefits of multiple agents and doesn't reduce the overall runtime. To evaluate the necessary evaluation latency for LMAO's multi-agent capabilities to be effective, we manually add delay to each function evaluation. As shown in Figure 2A, we iterate over multiple delay values: 1s, 3s, 5s, and 10s. These results highlight the positive correlation between the performance benefits from multiple agents and the latency of individual functional evaluations.

TABLE I
THE PARAMETER SEARCH SPACE FOR OPTIMIZING THE FIXED-POINT SPIKING GRAPH NEURAL NETWORK [11] WITH LMAO.

Parameter	*Options*
Paper to Paper Weight	{100, 101, ..., 500}
Train to Topic Weight	{1, 2, ..., 10}
Val. to Topic τ_+ & τ_-	{20, 21, ..., 60}
Simulation Steps	{5, 7, ..., 13}

B. Transductive Spiking Graph Neural Networks

In the second experiment, we demonstrate the performance of LMAO with a fixed-precision spiking graph neural network. Introduced in [2], citation graph classification is performed with transductive learning where the spiking neural network structure is designed based on the citation graph itself. More recent works such as [9] and [11] demonstrate the capability of this approach with Bayesian optimization (BO) while intra-network computations are limited to integer precision compatible with Loihi [16].

Using LMAO, our goal is to reduce the total optimization time required by BO to select the optimal parameter set. As shown in Table I, our search space consists of 4 variables: paper to paper weight, train to topic weight, val to topic $\tau_{+/-}$ and number of simulation steps. We perform two BO experiments with 1 and 5 agents, with the results being averaged over 3 repetitions and 3 random seeds. Both experiments generate and evaluate 10 random points to initialize the underlying Gaussian process (GP). For the experiment with one agent, the acquisition function is used to select a point to evaluate next. This point is evaluated with the results incorporated into the GP. This process is repeated 20 times for 30 total iterations. Conversely, the experiment with 5 agents takes the initialized model and selects 5 unknown points using the constant liar strategy [15]. These 5 points are evaluated in parallel with the results returned and used to update the GP. Distributing the

TABLE II
THE PARAMETER SEARCH SPACE FOR OPTIMIZING THE NEUROMORPHIC SATELLITE SCHEDULING PROBLEM [12] WITH LMAO. THE OVERALL SPACE CONTAINS 270 PARAMETER COMBINATIONS.

Parameter	*Options*
Turning Rate	{1.0, 1.25, ..., 3.0}
View Height	{0.25, 0.50, ..., 1.5}
Number of Satellites	{2, 3, 4, 5, 6}

optimization across multiple agents reduces the total number of GP model updates and expands the variety of evaluated points. As shown in Figure 2B, this allows the experiment with 5 agents to explore a wider area of the search space and avoid converging to local optima as in the case with 1 agent. Moreover, expanding the search across multiple agents reduces the overall optimization time by 2.2x.

C. Satellite Scheduling with Quadratic Unconstrained Binary Optimization

In our last experiment, we highlight the performance impact of multi-agent optimization for traditional grid search applied to a novel satellite scheduling algorithm within the Lava-Optimization library [12]. Using the 270 parameter search space shown in Table II, we perform grid search with varying numbers of agents, ranging from 1 to 50. As shown in Figure 2C, LMAO's multi-agent architecture efficiently scales where there is an inverse correlation between the number of agents and total search time. Specifically, increasing the number of agents from 1 to 50 reduced cumulative runtime by 5.57x.

IV. CONCLUSION

In this work, we introduce **L**ava **M**ulti-**A**gent **O**ptimization (LMAO), a novel framework for parallelized optimization and search algorithms within the Lava software framework. Our results demonstrate the scalability of this system applied to a variety of application spaces with multiple optimization and search algorithms. Using the abstract framework provided by LMAO, we are planning to include more algorithms such as: evolutionary algorithms [17], hyperdimensional Gaussian process regression [18] and distributed Bayesian search [19]. Moreover, we will expand the application space for LMAO in areas such as automated neural network design [3] and robotic control [20].

V. ACKNOWLEDGEMENTS

The research was funded in part by National Science Foundation through award CCF2319619 and a gift from Intel Corporation.

REFERENCES

[1] R. Patton, C. Schuman, S. Kulkarni, M. Parsa, J. P. Mitchell, N. Q. Haas, C. Stahl, S. Paulissen, P. Date, T. Potok *et al.*, "Neuromorphic computing for autonomous racing," in *International conference on neuromorphic systems 2021*, 2021, pp. 1–5.

[2] G. Cong, S.-H. Lim, S. Kulkarni, P. Date, T. Potok, S. Snyder, M. Parsa, and C. Schuman, "Semi-supervised graph structure learning on neuromorphic computers," in *Proceedings of the International Conference on Neuromorphic Systems 2022*, 2022, pp. 1–4.

[3] M. Parsa, C. Schuman, N. Rathi, A. Ziabari, D. Rose, J. P. Mitchell, J. T. Johnston, B. Kay, S. Young, and K. Roy, "Accurate and accelerated neuromorphic network design leveraging a bayesian hyperparameter pareto optimization approach," in *International Conference on Neuromorphic Systems 2021*, 2021, pp. 1–8.

[4] C. Scharzenberger and J. Hays, "Learning to estimate regions of attraction of autonomous dynamical systems using physics-informed neural networks," *arXiv preprint arXiv:2111.09930*, 2021.

[5] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. null, p. 281–305, feb 2012.

[6] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: A big comparison for nas," 2019.

[7] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*, ser. Adaptative computation and machine learning series. University Press Group Limited, 2006. [Online]. Available: https://books.google.com/books?id=vWtwQgAACAAJ

[8] T. Bayes, "Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s," *Philosophical transactions of the Royal Society of London*, no. 53, pp. 370–418, 1763.

[9] G. Cong, S. Kulkarni, S.-H. Lim, P. Date, S. Snyder, M. Parsa, D. Kennedy, and C. Schuman, "Hyperparameter optimization and feature inclusion in graph neural networks for spiking implementation," in *2023 International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2023, pp. 1541–1546.

[10] D. H. Ackley, "The model," in *A Connectionist Machine for Genetic Hillclimbing*. Springer, 1987, pp. 29–70.

[11] S. Snyder, V. Clerico, G. Cong, S. Kulkarni, C. Schuman, S. Risbud, and M. Parsa, "Transductive spiking graph neural networks for loihi," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, ser. GLSVLSI '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 608–613. [Online]. Available: https://doi.org/10.1145/3649476.3660366

[12] I. Labs, "Lava software framework," 2024, accessed: 2024-3-16. [Online]. Available: https://lava-nc.org

[13] S. Snyder, S. R. Risbud, and M. Parsa, "Neuromorphic bayesian optimization in lava," in *Proceedings of the 2023 International Conference on Neuromorphic Systems*, 2023, pp. 1–5.

[14] S. Snyder, S. Risbud, and M. Parsa, "Asynchronous neuromorphic optimization in lava," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, ser. GLSVLSI '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 776–778. [Online]. Available: https://doi.org/10.1145/3649476.3660383

[15] C. Chevalier and D. Ginsbourger, "Fast computation of the multi-points expected improvement with applications in batch selection," in *Learning and Intelligent Optimization*, G. Nicosia and P. Pardalos, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 59–69.

[16] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[17] K. A. De Jong, *Evolutionary computation*. Cambridge, MA: Bradford Books, Mar. 2016.

[18] P. M. Furlong, T. C. Stewart, and C. Eliasmith, "Fractional binding in vector symbolic representations for efficient mutual information exploration."

[19] M. T. Young, J. D. Hinkle, R. Kannan, and A. Ramanathan, "Distributed bayesian optimization of deep reinforcement learning algorithms," *J. Parallel Distrib. Comput.*, vol. 139, no. C, p. 43–52, may 2020. [Online]. Available: https://doi.org/10.1016/j.jpdc.2019.07.008

[20] C. Schuman, R. Patton, S. Kulkarni, M. Parsa, C. Stahl, N. Q. Haas, J. P. Mitchell, S. Snyder, A. Nagle, A. Shanafield *et al.*, "Evolutionary vs imitation learning for neuromorphic control at the edge," *Neuromorphic Computing and Engineering*, vol. 2, no. 1, p. 014002, 2022.