First steps towards micro-benchmarking the Lava-Loihi neuromorphic ecosystem

First steps towards micro-benchmarking the Lava-Loihi neuromorphic ecosystem

Walter Gallego Gomez, Andrea Pignata, Riccardo Pignari, Vittorio Fra, Enrico Macii and Gianvito Urgese
Politecnico di Torino, 10138, Torino (TO), Italy. Email: name.surname@polito.it

***Abstract*—Neuromorphic computing has proved to be capable of remarkable gains in energy efficiency over traditional architectures. With the continuous development of new software and hardware tools, benchmarking plays a crucial role in the measurement of technological advancement in the field. Over the last few years, some benchmarking efforts for neuromorphic computing have been proposed focusing on specific tasks like noise suppression or gesture recognition, while micro-benchmarking approaches have not been widely investigated. In this paper, we present our proposal to fill this gap: the Lava micro-benchmarking suite, a set of tests specifically designed for the Lava neuromorphic framework and the Loihi 2 neuromorphic architecture. Tests are divided into two broad categories: those aimed at evaluating Lava's message-passing implementation, that are partially inspired by MPI benchmarks, and those specifically designed to test the Loihi 2 architecture and the Lava compilation process. The suite is still a work in progress that needs to be extended to cover more functionalities of Lava and Loihi 2, but in its current state is functional and includes tests covering the three main hardware backends of interest: host CPU, Loihi 2 embedded CPU and Loihi 2 neuron cores. We present the general software design of the suite, the methods we have used to implement the tests and collect measurements and some examples of results obtained from running the tests. We expect that the suite can help Lava / Loihi 2 developers and users with meaningful insights that can be used to improve the state of this neuromorphic ecosystem.**

***Index Terms*—Neuromorphic, Lava, Loihi, Benchmarking**

I. Introduction

Neuromorphic computing straddles the boundary between Computer Science and Neuroscience, representing an alternative paradigm for computation based on the primitives implemented by biological nervous systems [1]. Spiking neural networks (SNNs) are the artificial counterpart of these latter [2], where information is processed by *artificial neurons* and transmitted through *artificial synapses* emulating the biological action potentials through discrete events referred to as *spikes*. As a consequence, neuromorphic computing inherently differs from the classical von Neumann architectures, for both structures and functions [3]. Furthermore, thanks to the sparse nature of the event-based SNNs, it can unlock orders of magnitude gains in energy efficiency over traditional architectures [4], which makes it an appealing candidate for on-edge applications [5], [6].

From the hardware perspective, neuromorphic computing implies and requires architectures and design principles directly inspired by the brain [7], resulting in huge efforts that have so far produced different device implementations and dedicated hardware [8]–[10]. Among them, a prominent representative is the *Loihi* chip by Intel, first presented in 2018 [11], which has already reached its second generation [12]. Together with *Loihi 2*, Intel presented the open-source framework *Lava* [13], [14], that allows users to write neuro-inspired applications and map them to both traditional and neuromorphic hardware.

With such advancements and the ongoing development of new technologies, benchmarking is indispensable as a tool for establishing a common goal and clear methodology to measure progress. Therefore, expanding the set of tools available for benchmarking in the neuromorphic computing domain is of great importance. Given that both the algorithmic and the system components are still under development [15], there is an opportunity for cross-stack advancement, by utilizing dual track benchmarking: 1) freezing the task and dataset, the algorithm is optimized; 2) freezing the data and the algorithm, the neuromorphic system is optimized. As algorithms mature, they can be incorporated into the system benchmark as reference challenges. As system innovations arise, they can offer new tools for algorithms to use [16]. In line with these ideas, Intel has recently issued the *"Intel Neuromorphic Deep Noise Suppression Challenge"* [17]. Other workloads for benchmarking neuromorphic systems were also discussed in [18], including spoken keywords and MNIST digits classification, Path Optimization in graphs and Constraint Satisfaction Problems (CSP) such as Latin Square and Map Coloring. Implementation of benchmarks for neuromorphic hardware includes: benchmark for spatio-temporal tactile pattern recognition and comparison of performance between Loihi and traditional hardware [6]; benchmarking using keyword spotting in Loihi and traditional hardware [19]; comparison of Loihi and SpiNNaker 2 performance for keyword spotting and adaptive robotic control tasks [20].

These types of benchmarks are appropriate for the specific areas they cover, but we believe that the development of a more general benchmarking approach, that can be applied to diverse areas is also required. Micro-benchmarking has been used as an alternative to application benchmarking and has proved to be useful for evaluating the performance of the Message Passing Interface (MPI) standard [21], [22] and to evaluate the performance of several architectures like GPUs [23], high-speed cluster interconnects [24], memories [25] and cloud applications [26]. Micro-benchmarking has also been used to evaluate the SpiNNaker neuromorphic architecture [27], [28].

In this paper we present our work in progress: Lava Micro-Benchmarking Suite, a set of tests that can be used to assess

the performance of Lava and Loihi 2. The proposed suite is delivered as an extension to Lava and supports a configurable set of parameters for each test category. The tests can be divided into two main groups 1) tests targeting the communication performance of Lava processes running on traditional hardware (host CPU and embedded Loihi 2 CPU); 2) tests targeting the communication performance and hardware utilization when deploying a network of neurons in Loihi 2 neuron cores. We expect that the results and insights obtained by running the suite will help in the improvement of the algorithmic and system layers of neuromorphic computing.

II. Methods

In the Lava Micro-Benchmarking Suite, we propose a set of single and well-defined common micro-operations that represent the building blocks of more complex tasks. This can help in the characterization of Intel's neuromorphic ecosystem at different levels: 1) Lava as a message-passing framework; 2) Lava compilation process to map networks to Loihi 2 hardware; 3) Loihi 2 hardware architecture in terms of communication between Neuron Cores (NCs); 4) hardware backends when executing Lava processes; 5) the network interconnection between different hardware backends.

Micro-benchmarking can be an excellent complement to the high-level benchmarking proposed in [6], [16]–[20]. High-level benchmarking can inform when a specific task or algorithm is not performing as well as expected in a given system, but it does not always give information about the specific aspect of the system that needs improvement. The results from running micro-benchmarks can fill this gap by providing the performance of individual components of the system, for example, the latency on the transmission of spikes as a function of the number of neurons.

Neuromorphic computing inherently implies sparse and asynchronous communication between elemental units designed to implement neuro-inspired functionalities, and the resulting gain in the energy-delay product when coupled with dedicated hardware [6] emphasizes the tacit role of assessing the communication performance within the system. Especially in view of on-edge and real-time applications, information passing between nodes has indeed to be investigated in order to evaluate, and whenever possible to reduce, the latency of the system in local communication. A way to do so is to leverage micro-operations to measure elemental contributions to higher-level activities involving data transmission.

Given that the Lava framework is based on the *Communicating Sequential Process* paradigm and uses channel-based message passing between asynchronous processes [14], it makes sense to take inspiration from micro-benchmarking implementations for the Message Passing Interface (MPI) standard. Specifically, we use as references the Intel MPI Benchmarks (IMB) [21] and the Ohio State University Micro-Benchmarks (OSU-MB) [22]. Even if Lava is not based on MPI, and it has features specific for neuromorphic execution, the concepts from MPI benchmarks to test the communication performance between processes sharing data through messages are still useful.

For the characterization of Loihi 2 hardware, we leverage the profiler tools already available in Lava, which allows us to obtain different execution metrics like the duration of each synchronization phase at each timestep. We focus our attention on the duration of the spiking phase and the hardware utilization ratio when deploying spiking neural networks of different sizes and topologies.

A. Intel's neuromorphic ecosystem

1) Lava framework: Lava is the open-source Python software framework designed by Intel to deploy neuro-inspired applications on heterogeneous hardware. We give an overview of the key features of Lava relevant to our suite:

a) Processes and models: Processes in Lava are abstract elements and only define their interface in terms of ports and internal variables. The corresponding concrete implementation is defined via *Process Models*. A single process can be defined by multiple models, each one with different characteristics and targeting different backends. Processes can be interconnected to create an application, and the framework will choose the appropriate model for each process, depending on the hardware configurations and preferences set by the user.

b) Types of models: As Lava supports multiple types of hardware backends, multiple types of models are also required. Three types of models are currently supported: `PyLoihiProcessModel` for Python models running on a host CPU, `CLoihiProcessModel` for `C` models running on a Loihi 2 embedded Lakemont (LMT) processor and `NcModel` for models running on Loihi 2 NCs. The user of the framework needs to define all the models corresponding to the hardware they aim to support.

c) Compiler: Process models need to be compiled so that they can be executed in the corresponding hardware backend. As different backends can have different compilation procedures, Lava can delegate the compilation job to other libraries such as *gcc* for *C* models, or *nxcore* for NC models.

d) Processes interconnection: The interconnection between processes is done through *Channels*, formed by connecting *ports* from different processes. A working interconnection between processes is guaranteed even if they run on different backends, as long as this type of connection is supported. Currently, Lava supports the following connections: CPU–CPU, CPU–LMT, LMT–NC and NC–NC.

e) Types of communication: Ports can be divided into two groups, that allow for two different types of communication:

- *Point-to-Point* using `InPorts` and `OutPorts`: An Input port of a process can be connected to the output port of another to form a communication channel. With this channel, the two processes can communicate via message-passing in a *point-to-point* fashion in which both processes are explicitly involved in the messages exchange, using the `send()` and `recv()` APIs.

- *One-sided* using `RefPorts` and `VarPorts`: Connecting a Reference port to a Variable port, a process can directly access an internal variable of another process, using the `read()` and `write()` APIs. As only the accessor process is explicitly involved, this communication is known as *one-sided*.

f) Data representation: The data to be sent can be represented as either *dense* or *sparse*. In the dense representation, the full array is transmitted as it is. In the sparse representation, only some elements of the array (usually non-zero ones) are sent, thus requiring two arrays: one for the data and another for the corresponding indices.

g) Synchronization protocol: Processes within a system or network operate in parallel and communicate asynchronously with each other through the exchange of message tokens. However, many use cases require synchronization among processes. For this purpose, Lava uses a *synchronization protocol* with *phases* that must be executed by all processes at each *timestep* before advancing to the next one. Lava defines the `LoihiProtocol`, which adheres to the phases of execution of Loihi 2. The main phase in this protocol is *Spiking*, in which neuron cores perform the actual computation and may generate a spike if firing conditions are met. The spiking phase is always required, unlike the other phases of this protocol (*Pre management*, *Learning*, *Post management* and *Host*).

For processes running on a host CPU, Lava also supports the `AsyncProtocol`, which does not define any phases for synchronization. With this protocol processes can run at varying speeds and message passing is possible at any time.

2) Loihi 2 architecture: The Loihi 2 chip consists of 6 embedded microprocessor cores (Lakemont x86) and 128 fully asynchronous neuron cores (NCs) connected by a network-on-chip. The NCs are optimized for neuromorphic workloads by implementing a group of spiking neurons and including all synapses connected to such neurons. All the communication between NCs is in the form of spike messages. Microprocessor cores are optimized for spike-based communication and execute standard `C` code to assist with data I/O as well as network configuration, management and monitoring. Some of the new functionalities added in this second version of the Loihi chip are the possibility of implementing custom neuron models using microcode instructions (assembly), the option to generate and transmit graded spikes, and support for three-factor learning rules. A single Loihi 2 chip supports up to 1 million neurons and 120 million synapses [13].

3) Resources: We use Intel's vLab server to run all the implemented tests. This server acts as our host CPU, and grants us access to neuromorphic boards, including Intel's Oheo Gulch, that we use for tests requiring a Loihi 2 backend. This board houses a single-socketed Loihi 2 chip, instrumented for characterization and debug [13].

We use Python (version 3.10.4) as our main programming language and the open-source Lava framework (version 0.6.0) which is available for download in the dedicated GitHub repository[1].

B. Micro-benchmarking suite architecture

The suite is developed in Python because it makes the interaction with Lava straightforward, it has excellent modules for data management and visualization (NumPy, Pandas, Matplotlib), and it supports object-oriented programming. We follow Lava's development guidelines [14] so that the micro-benchmarking suite can eventually be integrated into Lava's official repository. These guidelines include the use of PEP8 style, docstrings in NumPy format, the development of unit tests and the use of linting and type hinting.

The micro-benchmarking suite consists of two main components: 1) the benchmarking program that executes the tests requested by the user; 2) a tool for analyzing the obtained data and generating plots as requested by the user.

The benchmarking program follows a modular approach implemented using object-oriented programming. Basic components are implemented as classes which are then extended to fit the particularities of each test, through inheritance. When possible, the same basic components are reused in different tests.

To obtain statistically meaningful results, each test can be run multiple times. The user can choose how many repetitions to perform. For each category of test, there is *test maker* class, which executes the required test multiple times, according to the different combinations of parameters and repetitions requested. The maker then collects all of the obtained measurements and saves this (possibly multidimensional) data using basic Python structures (lists and dictionaries). This raw data is optionally written to disk.

The raw data is given to the data analyzer and plotter that transforms it into a Pandas DataFrame for easier manipulation, and process it via statistical methods to extract useful information from the measurements. This information, together with interactive plots, is used to generate an HTML report of the test execution.

C. MPI-inspired tests

Table I summarizes the proposed MPI-inspired tests, their configuration parameters and available variants. These variants are used to cover specific features of the Lava framework:

- Data representation in ports that can either be dense or sparse.
- Synchronization protocol that can either be the `LoihiProtocol` or the `AsyncProtocol`. This latter is not supported by LMT processes, so it is only applicable for the CPU–CPU connection.
- Communication type that can either be point-to-point or one-sided.

1) Single transfer tests: Single transfer tests are inspired by the *single transfer* benchmarks from the IMB [21] and the *Point-to-Point tests* from the OSU-MB [22]. They involve

[1] https://github.com/lava-nc/lava/

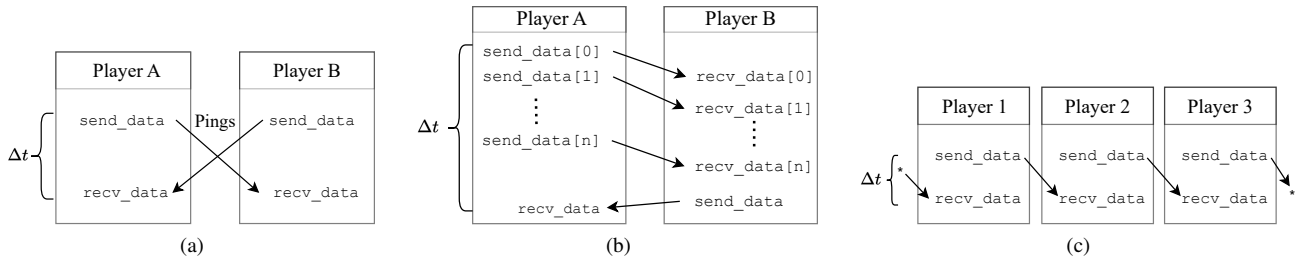

Fig. 1. (a) PingPing. (b) Unidirectional. (c) Periodic chain with three processes.

TABLE I
MPI-INSPIRED TESTS

Category	Tests	Parameters	Connection	Variants
Single transfer	- PingPong - PingPing	Message size	CPU–CPU CPU–LMT*	Dense or Sparse Loihi or Async+
Multiple transfer	- Unidirectional - Bidirectional	Message Size No. of messages		
Parallel single transfer	- Par. PingPing - Par. PingPong - Periodic chain - Bidirectional periodic chain	Message size No. of processes		
Parallel multiple transfer	- Parallel unidirectional	Message Size No. of messages No. of processes		P2P or One-sided
Collective transfer	- One to many - Many to one	Message size No. of processes		

*Only one process can be configured to run on the LMT processor.
+The Asynchronous protocol is only supported for CPU-CPU communication.

two active processes, known as *Player A* and *Player B* that communicate only with each other, using single messages.

The exchanged messages are arrays of 32-bit integers. The user can configure the size of the array (number of integers to send) and provide a list of sizes, in which case the test will be run for each provided size.

a) PingPong test: *Player A* sends a single message (*Ping*) to *Player B*. *Player B* waits for the message and replies with a message (*Pong*) of the same size. Time measurement is performed by *Player A*, from the moment the *Ping* is sent to the moment the *Pong* is received.

b) PingPing test: *Player A* sends a single message (*Ping 1*) to *Player B*. Simultaneously, *Player B* sends a single message (*Ping 2*) to *Player A*. Time measurement is performed by *Player A* from the moment *Ping 1* is sent to the moment *Ping 2* is received. (Fig. 1a).

2) Multiple transfer tests: Multiple transfer tests are inspired by the OSU-MB *Point-to-Point multiple transfer* benchmarks [22]. Two active processes are involved in the exchange of information. One or both processes sends multiple messages, sequentially.

Besides the message size, the user can also configure the size of the message stack to be transmitted, i.e., the number of messages that are sent one after the other, in a single run of the test. To execute the test with multiple configurations, the user can provide a list of integer pairs, each pair indicating the size of the messages and the number of messages.

a) Unidirectional: *Player A* sends a stack of messages, all of the same size, one after the other. *Player B* receives all of these messages, sequentially. Once all the messages are received, *Player B* will reply with one single message, of the same size. Time measurement is performed by *Player A* from the moment the first message in the stack is sent to the moment the reply is received. This test allows measuring the sustained data rate that can be achieved between two single processes (Fig. 1b).

b) Bidirectional: Both *Player A* and *Player B* send a stack of messages, all of the same size, sequentially. Each player receives the messages from the other, to then reply with one single message. Time measurement is performed by *Player A* from the moment the first message in the stack is sent to the moment the reply is received. With this test, we can measure the maximum sustainable aggregate bandwidth between two processes.

3) Parallel transfer tests: These tests are inspired by the *Parallel Transfer* benchmarks from the IMB [21] combined with *multi-pair tests* from the OSU-MB [22]. In this category, more than two processes are active at the same time, performing data transfer operations, in parallel. A configuration parameter controls the number of processes.

a) Parallel PingPing, PingPong and Unidirectional tests: The processes are divided into two ranks: *rank A* and *rank B*. Each process in *rank A* communicates with its counterpart in *rank B*. Each pair of processes performs the requested test (*PingPing*, *PingPong* or *Unidirectional*) independently and in parallel. Time measurement is performed independently for each pair.

b) Periodic chain: The processes are ordered to form a periodic communication chain. Each process sends a message to its neighbor to the right and receives a message from its neighbor to the left. Time measurement is performed inside each process individually from the instant it sends its message to the right to the instant it receives the message from the left (Fig. 1c).

c) Bidirectional periodic chain: The processes are ordered to form a periodic communication chain. Each process first sends a message to both of its neighbors and then waits for a message from both of them. Time measurement is performed

TABLE II
LOIHI 2 SPECIFIC TESTS

Category	Tests	Connection	Parameters	Variants
NC SNNs	- Fully connected	NC–NC	No. of layers No. of neurons Active neurons	Hcoded LIF Ucoded LIF Ucoded SDN
	- Convolutional - One-to-One	NC–NC	No. of layers No. of neurons kernel size	
SNNs with LMT	LMT as spiker	LMT–NC–NC	No. of layers No. of neurons	Hcoded LIF
	LMT as receiver	NC–NC–LMT		
	LMT as adapter	CPU–LMT–NC		

inside each process individually from the instant it sends its first message to the instant it receives the second message.

4) Collective transfer tests: In this category of tests, we make use of Lava's ability to split or merge channels when connecting processes in one-to-many or many-to-one configurations. The tests are inspired by the *Collective Benchmarks* from the IMB [21] and the OSU-MB [22].

a) One-to-many: One process, known as *Player A* sends a single message to n processes, which are known as $\{Player\ B_1, \ldots, Player\ B_n\}$. *Player A* does not need to send multiple messages: Lava allows the connection of one output port to multiple input ports. Each of the receiving processes replies with a single message to *Player A*. Time is measured by *Player A* from the instant the message is sent to the instant the last reply is received.

b) Many-to-one: n sender processes, known as $\{Player\ A_1, \ldots, Player\ A_n\}$, send a message to a receiver process, known as *Player B*. *Player B* receives the data in a single input port, and the incoming messages are combined by Lava using a point-wise addition. After receiving the data, *Player B* responds with a single message to $Player\ A_1$, which performs the time measurement, from the moment it sends its message to the moment it receives the reply.

D. Loihi 2 tests

To evaluate the performance of the Loihi 2 hardware, we designed some specific tests, summarized in Table II. The tests cover Loihi 2 neuron cores (NCs) and the Loihi 2 embedded Lakemont (LMT) processor. For these tests, we use Spiking Neural Networks (SNNs) of different topologies, sizes, and neuron models.

The Lava framework offers a set of tools known as *Profilers* that allow users to get performance metrics on the execution of a network configured on Lava and ran on a supported backend, like a Loihi 2 chip. The available data include execution time, resources and memory utilization, spiking activity and energy consumption. Measurements are performed by the Loihi 2 hardware itself, with little to no overhead, and the results are transmitted to the host PC by Lava, where we can access them from our suite.

We focused our attention on the execution time metric. In the Loihi synchronization protocol, execution is divided in *timesteps*. At each timestep, the enabled phases are executed one after the other, and the execution advances only when all processes in the network have finished the current phase. The duration of each timestep can be different and depends on the enabled phases for that timestep and on the network activity. We aim to verify how processes computation and spiking activity impact the duration of a timestep, so we measure the spiking phase, where computation is performed and spikes are transmitted. We have designed the tests so that we can easily understand, for each timestep, which neurons (if any) are generating spikes.

To obtain meaningful results, the parameters of the network are configured so that the number of timesteps with spikes is roughly the same as the number of timesteps without spikes. We then separate these two groups and calculate their average timestep duration:

- The timesteps without spikes are useful to understand how the execution performed by each node in the network impacts the timestep duration.
- The timesteps with spikes are useful to evaluate communication performance between nodes.

1) SNNs running on Loihi 2 neuron cores: We propose a set of tests where all the nodes in the network are run in Loihi 2 NCs. For this category of tests, besides execution time, we also analyze the allocation of resources. The Loihi 2 platform offers a maximum number of NCs, a maximum number of neurons per NC, and a maximum amount of memory per NC. As the networks used for our benchmarks can be tuned in the number of neurons and layers, we can increase these values to the limit allowed by the architecture and evaluate which resource is depleted first. We can also analyze how our networks are mapped by the compiler to the Loihi 2 hardware and evaluate if the result is optimal in terms of resource utilization.

As the neuron model used in each NC can impact the execution time and resource utilization, the tests can be configured to use any of the models present in Lava. Currently, Lava offers two implementations of the *Leaky Integrate and Fire (LIF)* neuron, one in hardware, known as *hard-coded*, and one in software, known as *micro-coded*, and one *micro-coded* implementation of the *Sigma-Delta Neuron (SDN)*. Executing the same tests with different models allows for a direct comparison between them.

The input to the network is generated by a Lava *Spiker* process, which outputs spikes at regular intervals and is modeled to run on one NC. The delay between the spikes is chosen so that a spike is propagated to the last layer before a new one is generated. The interconnection weights and neuron parameters are chosen so that each neuron does not generate an output spike until it receives a spike in all of its inputs. This guarantees that the spikes generated by the Spiker process travel from one layer to the next and reach the last layer.

a) Fully connected SNN: In this topology, all neurons from one layer are connected to all the neurons of the next layer, using a Lava *Dense* process for the interconnections.

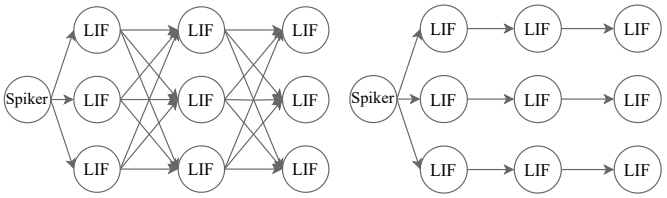

Fig. 2. (a) Fully connected network with 3 layers and 3 neurons per layer. (b) One-to-one network with 3 layers and 3 neurons per layer.

All layers have the same number of neurons. The user can personalize the number of layers and neurons per layer. An example with LIF neurons is represented in Fig. 2a.

With this configuration, we aim to evaluate the performance of the Loihi 2 hardware when the number of connections among neurons is high, so that the available synaptic memory (which is required for the connections) gets depleted before the available number of neurons inside each NC.

The suite allows reducing the percentage of active neurons in each layer, which by default is set to 100%, by setting to zero the corresponding weights of the Dense process. Running the tests with different percentages of active neurons is useful to investigate how the spiking activity affects communication performance.

b) Convolutional SNN: In this topology, the connection between layers is performed using a Lava *convolutional* process. This process allows us to configure the connection using a multi-dimensional matrix known as *filter* that contains the weights used during the convolutional operation.

With this configuration, the user can analyze the performance of the convolutional connection for different network and filter sizes.

c) One-to-One SNN: This test is a special case of the convolutional, using as filter a `1x1` matrix. The resulting topology is depicted in Fig 2b, where the i-th neuron from a layer is only connected to the i-th neuron of the next layer. With this configuration we reduce the number of connections between neurons, lowering the memory occupation required by each neuron. This allows us to verify the behavior of the platform when mapping the maximum allowed number of neurons in each NC.

2) SNNs with LMT: With this category, the aim is to evaluate the LMT processor when it makes part of an SNN. Code execution in the LMT, communication between LMT and NCs and between LMT and a host CPU are considered. Since the evaluation of resource utilization or neuron models running on the NCs is not performed here, for these tests we maintain a fixed topology and model by employing a fully connected network of LIFs.

a) LMT as Spiker: The LMT is configured to run *C* code that generates spikes periodically, and those spikes are propagated to the rest of the network. The performance can be compared to that of the network mapped only to NCs, to understand if the execution of the *C* code and the communication between the LMT and the first layer of neurons represent a bottleneck for the network.

b) LMT as Receiver: The LMT is connected at the end of the network and receives the spikes generated by the last layer. The LMT does not perform any computation on the received data, so this test allows us to examine the impact of the communication between NC cores and LMT in the duration of the spiking phase.

c) LMT as Bridge: This test targets spikes transfer between Python models running on the host CPU and NC cores, with the LMT serving as a bridge between them. We use the Lava processes *PyToNxAdapter* and *NxToPyAdapter*. The network is split, one half running on the CPU using Python models, and the other running on Loihi 2 NCs.

E. Time measurement

1) CPU–CPU: Lava uses `SharedMemory` objects from the `multiprocessing` Python module to implement the communication between CPU processes. To measure the transmission time between them we use the `perf_counter_ns()` function from the `time` Python module.

2) CPU–LMT: Communication between the Python process running on the CPU and the *C* process running on the LMT device is all handled by Lava, as long as the declared interfaces are compatible. Time measurement is performed with the same approach described for CPU–CPU tests, by the process(es) running on the host PC.

3) NC–NC and NC–LMT: These two connections reside entirely in the Loihi 2 hardware, so metrics are obtained using the Lava profiler tools.

4) LMT–LMT: Lava only supports the execution of one process targeting the LMT embedded processor[2], so it is not possible to implement single or multiple transfer tests with the two processes running on the LMT. This Lava limitation also impacts the parallel and collective MPI-inspired tests, as only one of the participating processes can be executed on the LMT.

5) CPU–NC: Lava does not support direct communication between the host CPU and Loihi 2 NCs.

III. Results and Discussion

To the best of our knowledge, ours is the first micro-benchmarking effort specifically designed for the Lava framework and the Loihi 2 hardware. Even though it is still a work in progress, we have taken considerable steps forward, as we completed the software architectural design of the micro-benchmarking suite and we implemented tests covering the main three hardware backends: host CPU; Loihi 2 embedded processor; Loihi 2 neuron cores. The delivered suite is modular and easily extendable with new tests or supported hardware. Users can interact with the suite from high-level Python APIs to choose the tests to run and generate HTML reports with the obtained data.

[2]As declared in the known issues in the Lava GitHub repository

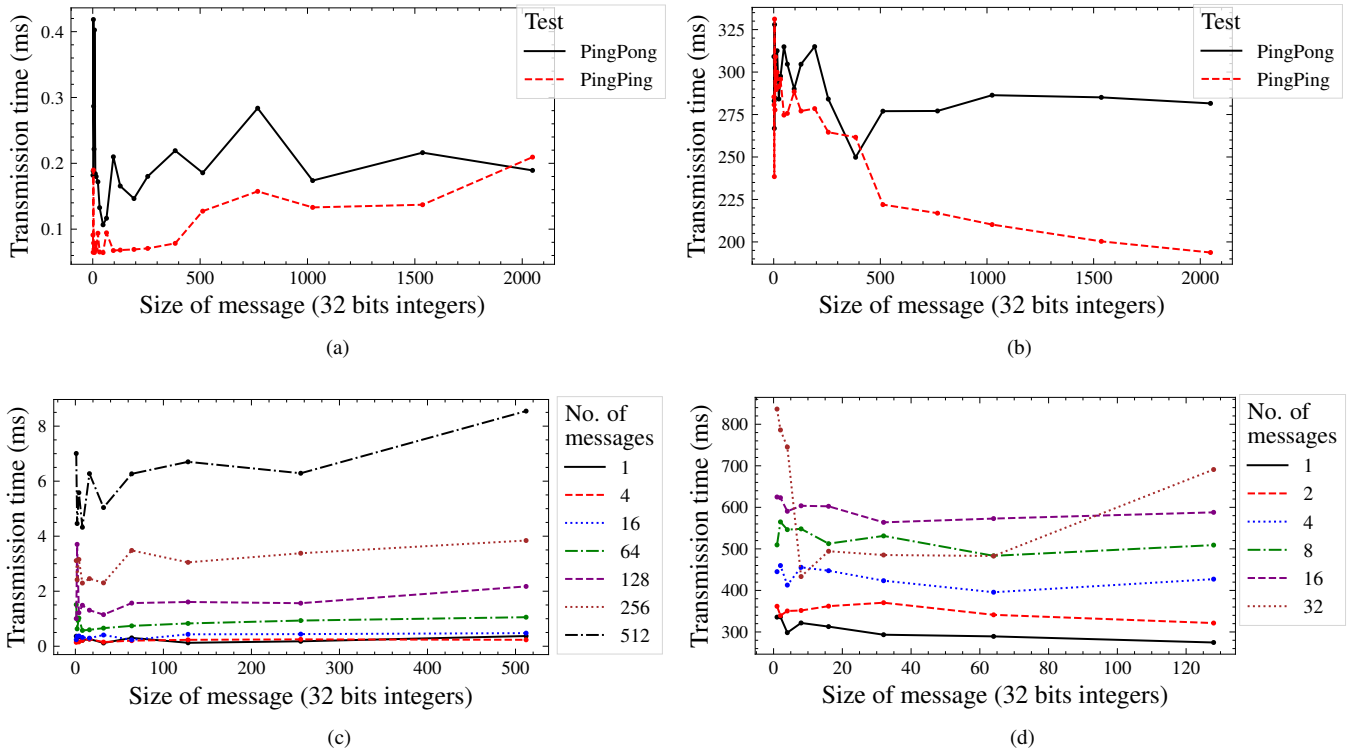

Fig. 3. Transmission time for: (a) CPU-CPU single transfer tests as a function of the message size; (b) CPU-LMT single transfer tests as a function of the message size; (c) CPU-CPU unidirectional multiple transfer test as a function of the message size, for several number of messages; (d) CPU-LMT unidirectional multiple transfer test as a function of the message size, for several number of messages.

A. *Execution example*

We present an example run of three tests: PingPing, PingPong and Unidirectional, using the host CPU and the Loihi 2 embedded processor, to demonstrate the performance measurement plots that can be currently generated with our suite. However, we refrain from performing any analysis on them, as it is out of the scope of this work. The analysis is indeed intended to be performed by the user of the suite according to the specific goal and characteristics of the executed tests.

1) Single transfer tests: Figs. 3a and 3b show the transmission time as a function of the message size for the CPU–CPU and CPU–LMT connections, respectively. We executed both tests with data sizes ranging from 1 to 2048, and for each size we performed 25 repetitions. From the results, we don't observe any clear correlation between the required transmission time and the message size, but there is a clear difference between both connection types, as CPU–LMT communication is around three orders of magnitude slower than the CPU–CPU counterpart.

2) Unidirectional test: Figs. 3c and 3d show the transmission time as a function of the message size, for different numbers of messages, for the CPU–CPU and CPU–LMT connections, respectively. For both tests, we used data sizes ranging from 1 to 512. For the CPU–CPU connection we configured the number of messages ranging from 1 to 512, while for the CPU–LMT we had to limit this range because, for this type of connection, Lava only supports a maximum of 32 messages. For each combination of number of messages and data size, we performed 25 repetitions. We observe a correlation between transmission time and the number of messages. There is however an exception to this correlation in the CPU-LMT connection when using 32 messages. As for the single transfer tests, the CPU–LMT connection is three orders of magnitude slower than the CPU–CPU version.

B. *Use case examples*

We briefly introduce two possible use cases of how the results from running the suite could be used.

1) Lava/NxCore developer: A Lava or NxCore developer may be interested in understanding how the Lava framework and the NxCore compiler behave when the number of neurons and layers to map into the Loihi 2 hardware change. By running the SNNs tests with different parameter values, the developer may find weak points in the compiler, for which the utilization of memory and cores is sub-optimal.

2) Lava user: An user of the Lava framework working on an application requiring real-time processing of data, may need to understand how the spiking time behaves as the size of the network increases, for different neuron models. Running the SNNs tests can help them find, for each neuron model, the largest network that is still capable of processing the data in the required time frame.

C. Future work

Even though the suite in its current state is functional and can be run for a set of tests, as a work in progress, there is still room for improvement. We have finished the design of all the tests described in Tables I and II. However, we have not completed the implementation and testing for all of them; this activity is currently in progress.

More tests should be designed to increase the coverage of Lava and Loihi 2 features like, for instance, the use of hierarchical Lava processes, virtual ports and data transformation, on-chip learning (learning Dense and LIF processes), usage of custom neurons described in micro-code and multi-chip connectivity.

The generated report could be improved as well. Currently, it contains plots as those shown on Figs. 3a to 3d for each of the configuration parameters of the executed tests. More informative plot types and statistical information might be added. Finally, we plan to offer more flexibility to the users in the plots to be generated through a configuration file.

IV. Conclusions

In this paper, we introduced the Lava Micro-Benchmarking Suite, a collection of tests specifically designed to assess the performance of the Lava framework and the Loihi 2 hardware. Our work fills the gap of a missing micro-benchmarking tool for Lava and Loihi 2. The suite is modular and can be easily extended to support other types of benchmarking procedures and neuromorphic platforms. It has been designed following the Lava developer guidelines so that it can be integrated as an extension of the Lava framework. We also presented some test examples to show the results the suite can provide.

The suite needs to be extended with more tests so that it can reach a high coverage of Lava and Loihi 2 features, and it should be updated when there are breaking changes in Lava, or when new hardware backends (e.g. GPUs) are supported.

We expect that the results obtained from running our suite can provide insights to Lava / Loihi 2 developers and users and that these insights can be used to advance the field of neuromorphic computing.

Acknowledgment

This research is funded by the Ebrains-Italy project CUP B51E22000150006, the 3A-ITALY project CUP E13C22001900001 and the Fluently project with Grant Agreement No. 101058680.

References

[1] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, 1990.

[2] W. Maass, "Networks of spiking neurons: The third generation of neural network models," *Neural Networks*, 1997.

[3] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, 2022.

[4] S. Furber, "Large-scale neuromorphic computing systems," *Journal of neural engineering*, 2016.

[5] V. Fra, E. Forno, R. Pignari, T. C. Stewart, E. Macii, and G. Urgese, "Human activity recognition: suitability of a neuromorphic approach for on-edge AIoT applications," *Neuromorphic Computing and Engineering*, 2022.

[6] S. F. Müller-Cleve *et al.*, "Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware," *Frontiers in Neuroscience*, 2022.

[7] G. Indiveri *et al.*, "Neuromorphic Silicon Neuron Circuits," *Frontiers in Neuroscience*, 2011.

[8] M. Davies *et al.*, "Advancing Neuromorphic Computing With Loihi: A Survey of Results and Outlook," *Proceedings of the IEEE*, 2021.

[9] S. Wang, T. H. Cheng, and M. H. Lim, "A hierarchical taxonomic survey of spiking neural networks," *Memetic Computing*, 2022.

[10] D. Ivanov, A. Chezhegov, M. Kiselev, A. Grunin, and D. Larionov, "Neuromorphic artificial intelligence systems," *Frontiers in Neuroscience*, 2022.

[11] M. Davies *et al.*, "Loihi: A Neuromorphic Manycore Processor with On-Chip Learning," *IEEE Micro*, 2018.

[12] G. Orchard *et al.*, "Efficient Neuromorphic Signal Processing with Loihi 2," in *IEEE Workshop on Signal Processing Systems (SiPS)*, 2021.

[13] "Technology Brief: Taking Neuromorphic Computing to the Next Level with Loihi 2," Intel Labs, 2021.

[14] Lava Software Framework. https://lava-nc.org/.

[15] G. Urgese, A. Rios-Navarro, A. Linares-Barranco, T. C. Stewart, and K. Michmizos, "Editorial: Powering the next-generation IoT applications: new tools and emerging technologies for the development of Neuromorphic System of Systems," *Frontiers in Neuroscience*, 2023.

[16] (2022) Benchmarks: Seeding a Collaborative Effort + An Audio Challenge! [Online]. Available: https://intel-ncl.atlassian.net/wiki/spaces/INRC/pages/1831895043/Benchmarks+Seeding+a+Collaborative+Effort+An+Audio+Challenge

[17] J. Timcheck *et al.*, "The Intel Neuromorphic DNS Challenge," *CoRR*, 2023.

[18] M. Davies, "Benchmarks for progress in neuromorphic computing," *Nature Machine Intelligence*, 2019.

[19] P. Blouw, X. Choo, E. Hunsberger, and C. Eliasmith, "Benchmarking keyword spotting efficiency on neuromorphic hardware," in *Proceedings of the 7th annual neuro-inspired computational elements workshop*, 2019.

[20] Y. Yan *et al.*, "Comparing Loihi with a SpiNNaker 2 prototype on low-latency keyword spotting and adaptive robotic control," *Neuromorphic Computing and Engineering*, 2021.

[21] (2021) Intel MPI Benchmarks. Intel. [Online]. Available: https://www.intel.com/content/www/us/en/docs/mpi-library/user-guide-benchmarks/2021-2/overview.html

[22] (2023) OSU Micro-Benchmarks. MVAPICH. [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/

[23] R. Taylor and X. Li, "A micro-benchmark suite for AMD GPUs," in *39th International Conference on Parallel Processing Workshops*, 2010.

[24] Jiu-xing *et al.*, "Micro-benchmark level performance comparison of high-speed cluster interconnects," in *Proceedings of 11th Symposium on High Performance Interconnects*, 2003.

[25] A. Ahmed and K. Skadron, "Hopscotch: a micro-benchmark suite for memory performance evaluation," in *Proceedings of the International Symposium on Memory Systems*, 2019.

[26] J. Scheuner and P. Leitner, "Estimating cloud application performance based on micro-benchmark profiling," in *IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018.

[27] G. Urgese, F. Barchi, and E. Macii, "Top-Down Profiling of Application Specific Many-core Neuromorphic Platforms," in *IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, 2015.

[28] G. Urgese, F. Barchi, E. Macii, and A. Acquaviva, "Optimizing Network Traffic for Spiking Neural Network Simulations on Densely Interconnected Many-Core Neuromorphic Platforms," *IEEE Transactions on Emerging Topics in Computing*, 2018.