Evaluating the Impact of Spiking Neural Network Traffic on Extreme-Scale Hybrid Systems

Noah Wolfe, Mark Plagge and Christopher D. Carothers
Department of Computer Science
Rensselaer Polytechnic Institute
wolfen,plaggm@rpi.edu, chrisc@cs.rpi.edu

Misbah Mubarak and Robert B. Ross
Mathematics and Computer Science
Argonne National Laboratory
mmubarak@anl.gov, rross@mcs.anl.gov

***Abstract*—As we approach the limits of Moore's law, there is increasing interest in non-Von Neuman architectures such as neuromorphic computing to take advantage of improved compute and low power capabilities. Spiking neural network (SNN) applications have so far shown very promising results running on a number of processors, motivating the desire to scale to even larger systems having hundreds and even thousands of neuromorphic processors. Since these architectures currently do not exist in large configurations, we use simulation to scale real neuromorphic applications running on a single neuromorphic chip, to thousands of chips in an HPC class system. Furthermore, we use a novel simulation workflow to perform a full scale systems analysis of network performance and the interaction of neuromorphic workloads with traditional CPU workloads in a hybrid supercomputer environment. On average, we find Slim Fly, Fat-Tree, Dragonfly-1D, and Dragonfly-2D are 45%, 46%, 76%, and 83% respectively faster than the worst case performing topology for both convolutional and Hopfield NN workloads running alongside CPU workloads. Running in parallel with CPU workloads translates to an average slowdown of 21% for a Hopfield type workload and 184% for convolutional NN workloads across all HPC network topologies.**

***Index Terms*—Neuromorphic computing, Interconnection networks, Discrete-event simulation, large-scale**

I. Introduction

In recent years, new types of processor and network technologies have emerged that may be deployed in future supercomputer system designs. First, neuromorphic computing is a new class of processor that provides a brain-like computational model that enables complex neural network computations to be done using significantly less power than current processors. IBM has created an instance of a spiking neuromorphic processor, called TrueNorth, that has 4096 neurosynaptic cores with a total of 1 million spiking neurons and 256 million reconfigurable synapses which consumes only 63 milliwatts when executing a multi-object detection and classification program using real-time video input [1]. Several other advanced neuromorphic processor hardware architectures have been developed as well, featuring various hardware designs and features [2], [3], [4]. More recently, Intel has created the *Loihi* spiking neuromorphic processor which is able to perform on-chip learning [5].

Now, supercomputer network designs are also moving away from multi-dimensional torus networks used in Cray XT and IBM Blue Gene systems with the advent of high-radix routers enabling lower diameter interconnect topologies such as Dragonfly [6], [7], fat tree [8] and Slim Fly [9]. These networks promise either full bisection bandwidth as is the case with the fat tree or strike a balance between global and nearest neighbor traffic patterns using fewer network links and lower overall costs as with the Dragonfly and Slim Fly topologies.

Given these technology advances, the central question we will address is how might a node-level, neuromorphic processor accelerator impact overall system/network performance? This systems design question is driven by the DOE SEAB report [10] on high-performance computing which highlights the neuromorphic architecture as a key technology in future supercomputer systems as well as the DOE CORAL-2 RFP [11] which indicates that deep learning/neural network type workloads will be of key importance to next generation systems.

In this paper, representative workloads from leadership class supercomputer systems and neuromorphic applications are used to analyze the performance of a heterogeneous/hybrid system where a compute node has both CPU and neuromorphic chips. Our extensive simulation study examines the following questions: (1) How well can HPC network topologies handle neuromorphic communication workloads? (2) How much communication interference is caused by neuromorphic workloads that impacts conventional HPC workload performance and vice versa? and (3) How beneficial is destination-based spike aggregation for decreasing message transfers and improving HPC network performance? We explore the performance using modern high-radix interconnect topologies including Slim Fly, Dragonfly and fat tree. While running the CPU-based HPC and neuromorphic applications in parallel, we analyze the performance tradeoffs for both neuromorphic and HPC workloads.

To perform the detailed design space exploration, we use an end-to-end modeling and simulation workflow that answers metric-driven questions about the capabilities of potential hybrid CPU-neuromorphic supercomputer system designs. Key findings show CPU workloads only experience a mild slowdown in runtime of up to 16% running in the presence of neuromorphic workloads while neuromorphic applications leveraging convolutional neural networks like CIFAR and MNIST can experience up to 10x slowdown in runtime running in the presence of CPU workloads. Additionally, a Hopfield type neural network application performs equally

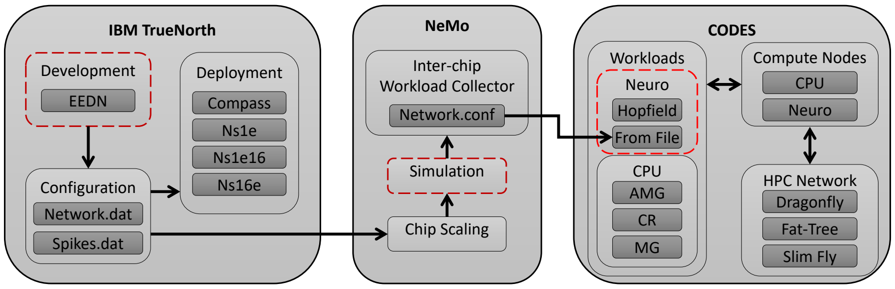


Fig. 1: Integration of IBM TrueNorth, NeMo, and CODES ecosystems to form the large-scale neuromorphic systems simulation workflow. The red dashed boxes indicate the various entry points for users to setup a large-scale neuromorphic simulation.

well or poorly (depending on your view) across similarly provisioned Fat-Tree, Slim Fly, Dragonfly-1D, and Dragonfly-2D network topologies because there is not much that can be done to improve the all-to-all communication pattern.

The major contributions presented in this paper are:

- A complete workflow for generating neuromorphic application trace workloads and simulating their execution in an HPC system alongside traditional CPU applications.
- A study predicting network performance of real convolutional neuromorphic applications simulated at scale in a HPC environment.
- A study quantifying the possible network interference effects of neuromorphic workloads on traditional CPU workloads and vice versa.
- Performance comparison of similarly provisioned Dragonfly, Fat-Tree, and Slim Fly network topologys for the above studies.

II. Related Work

The fields of neuromorphic computing and HPC simulation have already seen significant research contributions. Survey papers by Schuman et al. [12] and Brette et al. [13] provide a comprehensive overview of neuromorphic computing and spiking neural networks including hardware, applications, and algorithms. Projects such as Neuron [14] and Blue Brain [15] simulate neuromorphic architectures focussing on biologically relevant neuron behaviors but they don't study architectures that act as general purpose machine learning tools. They also do not study the construction of large-scale neuromorphic HPC systems. Conversely, the Structural Simulation Toolkit (SST) [16], Booksim [17], and IBM's Venus [18] are all simulation frameworks that provide HPC network simulation capabilities, however they all currently do not support neuromorphic simulations.

III. Neuro-NeMo-CODES Workflow

In this section, we describe the simulation framework being used for exploring the design space of HPC-neuromorphic systems. We present a complete workflow capable of taking predefined spiking neural networks, scaling them up beyond a single chip and simulating them on a large-scale neuromorphic system to study application and system performance. A high level overview of the workflow is shown in Figure 1. The workflow is composed of three loosely integrated components including the IBM TrueNorth, NeMo and CODES frameworks. The exploration process requires an ensemble of simulation runs to analyze the permutations of different parameters where both HPC and neuromorphic architectures are modeled at a detailed fidelity.

A. *TrueNorth*

The first component in the workflow is the IBM TrueNorth Neurosynaptic System, which provides an ecosystem for building real spiking neural network applications constrained and trained for the IBM TrueNorth neuromorphic processor [19]. The TrueNorth processor is a spiking neuron based architecture that groups 256 input axons with 256 output neurons into one neurosynaptic core. A weighted synaptic crossbar connects all input axons to all output neurons within a core. To form one TrueNorth chip, a total of 4,096 neurosynaptic cores are tiled together resulting in 1M neurons. Currently, deployable hardware solutions range from a one chip board to a 16 chip board with a 64 chip system [20] due in the near future. The neuromorphic applications can also be deployed to the IBM Compass simulator [21] however this approach also limits the number of simulated chips to the number of chips available in the modeler's available hardware solution.

Diving deeper into the neuromorphic chip architecture, its important to note that the TrueNorth hardware is synchronized to process spike data across all neurons following a 1ms unit of time called a tick. Within the 1ms tick window, all configured neurons integrate over their input spikes, generate a spike if their voltage reaches their set threshold, and send the generated spike to its destination axon across the network. The 1ms tick window is necessary as it provides enough of a buffer to guarantee all neurons perform their integration, spike generation, and spike transfer before the next round so data progresses through the neural network at the same rate. Thus, it is critical for any network topology to meet this 1 ms tick window guarantee for correct execution of any IBM TrueNorth neuromorphic application.

B. *NeMo*

The second component to the workflow is NeMo, a general purpose neuromorphic processor simulator. NeMo [22] is a simulation model leveraging the high-performance and massively parallel Rensselaer Optimistic Simulation System (ROSS) discrete-event simulation framework [23] and validated against the IBM Compass Simulator. Using ROSS, neurons are simulated as discrete logical processes with spike communication represented as events. NeMo consists of a modular neuron model, initially based on the IBM TrueNorth neurosynaptic neuron model, crossbar simulation, and neurosynaptic core models. NeMo allows for the simulation of arbitrary neuromorphic hardware designs with various configurations. NeMo can simulate various spiking neuron models, currently supporting the Leaky Integrate and Fire model as well as the IBM TrueNorth neuron model.

In addition to simulating the TrueNorth style neuromorphic processor, NeMo's file I/O provides the ability to read in and simulate applications developed within the IBM TrueNorth ecosystem. From there, both custom NeMo developed neuromorphic applications as well as TrueNorth applications can be scaled to run on thousands of chips utilizing NeMo's chip-scaling capability.

NeMo is able to generate partially synthetic multi-chip workloads for the purposes of benchmarking novel processor designs. These multi-chip benchmarks are created through interpolation of existing models. NeMo extracts the communication between cores to generate an approximation of a multi-chip neural network. If the number of desired virtual chips is greater than or equal to the number of physical neurosynaptic cores, NeMo simply splits the core-to-core communication across that many chips. For smaller numbers of simulated chips, NeMo will generate chip-to-chip communication based on the underlying core communications. For benchmarking network traffic, this is a reasonable approach, as the traffic of trained classification spiking neural networks tends to be extremely structured, as observed in other distributed spiking neural network research [24], [25]. This is in contrast to what is observed when simulating biological simulations of spiking networks, where traffic is generally homogenous and complex [26]. If the underlying model is constructed in NeMo, rather than imported from a trained TrueNorth network, NeMo's multi-chip simulation will produce complete chip-to-chip traffic using the underlying synthetic neuromorphic model.

A limitation of NeMo is that it lacks the ability to simulate inter-chip communication through an external network such is the case in large-scale HPC systems. When inter-chip spikes are detected, NeMo routes them instantaneously to their destination. We address this limitation by using the HPC network models provided by CODES, as explained in the next section. To allow for external processing of inter-chip communication, NeMo collects the chip-to-chip connections as well as the time dependent spiking information, computes the average number of spike transfers per connection per tick, and saves it all to file. The result is a neuromorphic many-chip communication workload representing the average number of spikes transferred between all chip connections during one tick of execution. This serves as an input to the CODES network simulation framework, which is described next.

C. CODES

The final component in the workflow is the CODES framework for HPC network simulations. CODES also leverages the discrete-event simulation capability of ROSS to provide models of high performance interconnects, network and I/O workloads, communication algorithms and storage devices into a unified parallel discrete-event simulation model. The interconnection network piece of CODES supports a packet level simulation of various high-radix network topologies using validated simulation models including Dragonfly [27], Fat-Tree [28] and Slim Fly [29]. Workloads from various sources can be replayed on these HPC interconnects and the simulation has the ability to replay multiple workloads executing in parallel. The different workloads include synthetic traffic patterns and communication traces from HPC applications [30]. CODES, also supports the ability to test various network configurations including multi-rail, multi-plane, and hybrid nodes.

Two new options for running neuromorphic workloads have been added to the CODES HPC network simulation toolkit. The first feature allows CODES to read in inter-chip communication data files to accurately replay real neuromorphic workloads in an HPC environment. In this approach, CODES leverages a NeMo generated input file to establish the network of neuromorphic chips, their connections, and their number of spikes transferred to each connected chip over the length of the simulation. When inter-chip spikes are observed, they are sent in an `MPI_Isend` and routed through the network to their destination chip. Alternatively, a second feature allows the modeler to run a fully synthetic neuromorphic workload based on a hopfield style neural network [31]. Following this type of neural network, in which all neurons are fully connected, results in all-to-all communication between chips. During each 1ms tick, each chip sends a user-configurable number of spikes to all chip destinations using the `MPI_Isend` operation.

D. Application Development

Currently, modelers have three different points of entry into the neuromorphic HPC workflow, each providing a different level of detail/accuracy. First, neural network applications can be generated using the IBM TrueNorth development platform. This level provides an easy adoption by TrueNorth developers. Entering at this level benefits from the data being generated from a real application source. However, this approach requires knowledge of and access to the TrueNorth specific hardware and software systems.

Modelers without any knowledge of TrueNorth can enter the workflow by generating a neural network workload within NeMo. Developing an application within NeMo allows for custom neuromorphic hardware constraints but requires learning the NeMo development API. Finally, a neuromorphic workload can be constructed at a much higher level of abstraction using the CODES framework. CODES specifically deals with only the inter-chip communication. Therefore, if the modeler understands their neural network application in terms of number of connections per chip, and number of spikes per

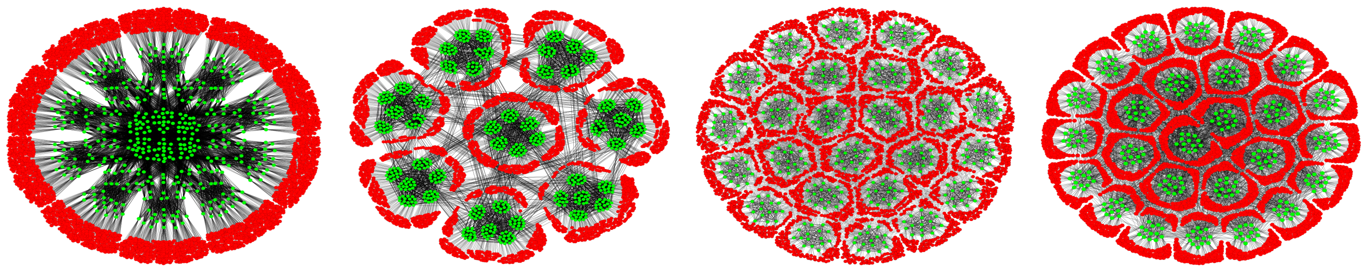


Fig. 2: Visualization of the networks utilized in this paper. Green nodes are routers and red nodes are compute nodes.

tick transferred over those connections, they can implement a strong application workload approximation at scale directly. Additionally, if the connections and spiking data is available, modelers can convert the data to the CODES neuromorphic input file format to be read and replayed over the network.

When mapping the virtual neuromorphic cores to simulated chips we use a simple linear placement approach. Sawada et. al. [19] describe the IBM heuristic placement algorithm which optimizes the placement of virtual neurons to the TrueNorth hardware in order to minimize message travel distance. The algorithm is optimized for placing virtual cores on hardware that uses a 2D connectivity of chips. However, we aren't using a 2D connectivity of chips. We are using, dragonfly, fat-tree and slimfly connectivity of chips and opt for a simple linear scaling and placement approach which is non-optimized but provides a consistent placement for many chips connected via Dragonfly, Fat-Tree, and Slim Fly topologies. Therefore, our results can be seen as a non-optimized or possibly even worst-case outcome.

IV. HPC System Configurations

In this work, we construct theoretical hybrid neuromorphic/CPU HPC systems with a total of 3,000 nodes and using Dragonfly, Fat-Tree, and Slim Fly interconnect topologies. Each simulated compute node consists of one neuromorphic processor representing the equivalent of one TrueNorth processor as well as one CPU processor. Both processors on a node share one network interface card.

A. Interconnection Network

The interconnect topology is one of the crucial factors in determining the performance of the HPC system. The HPC interconnects are modeled using the CODES network models, which provide validated packet level simulation of Dragonfly [32], Fat-Tree [33], and Slim Fly [29] network topologies. In this study we test two network configurations (Dragonfly-2D and Fat-Tree) inspired by currently deployed HPC systems and two other theoretical configurations (Dragonfly-1D and Slim Fly) that show potential as future neuromorphic HPC interconnects.

Table I lists and compares the network configurations. Each network is configured prioritizing similar compute node counts of roughly 3,000 nodes. Other parameters such as router radix are unrestrained to avoid placing design restraints which limit the specific benefits of the different topologies. All parameters independent of the network topology such as link speed, buffer space, and router latency are the same across all three networks to maintain a fair comparison. Also, buffer spaces are allocated per-vc and per-direction. The system diameter indicates the maximum number of links between any two compute nodes in the configured system. Max hop count indicates the maximum number of link traversals between any two compute nodes using the selected routing protocol. For adaptive routing, the max hop count is given from the worst-case non-minimal path. Table entries for local, global and compute node connections are per router. Router Radix Total describes the commodity type switch used for each configuration and Router Radix Used indicates how many ports are utilized. Injection bandwidth ($total\_terminal\_to\_router\_links * link\_speed$) and network bandwidth ($total\_router\_to\_router\_links * link\_speed$) calculations are provided for each configuration in Table I showing a simple high level comparison of bandwidth available for compute nodes to inject traffic into the network and bandwidth available for transferring the data throughout the network.

1) Fat-Tree: The Fat-Tree network is composed of multiple levels of switches or high-radix routers. Each switch has as many up links as down links whereas the top level uses half the number of switches only having down links. Due to the equal number of links in both directions, each pair of nodes is able to communicate via a unique network path, which allows the network to have a full bisection bandwidth. HPC systems such as Summit to be deployed at Oak Ridge National Laboratory uses a pruned Fat-Tree system to have 3,400 network node count. In this work, we use a system configuration that has a node count similar to the Summit system and has a single rail/single plane of the network [28]. We use static routing in our study, which is widely used in systems having Fat-Tree networks and detailed in [28].

2) Dragonfly: The Dragonfly network is a hierarchical, high-radix, and low diameter topology. In general, the Dragonfly consists of many groups of routers maintaining all-to-all global connections between the groups. Within each router group there are many options for intra-group connectivity. In this work we study two different Dragonfly configurations. The first configuration, which we call Dragonfly-2D, adopts the approach incorporated into the Cray XC30 and XC40 series of systems deployed at Lawrence Berkeley National Laboratory and Argonne National Laboratory. Following the Cray Cascade architecture [7], the network nodes within a

TABLE I: Network Configurations

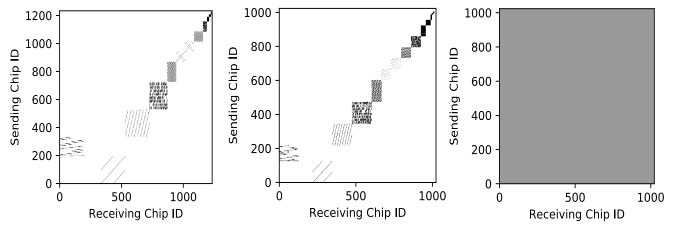
Metric	***Fat-Tree***	***Dragonfly 2D***	***Dragonfly 1D***	***Slim Fly***
Approximated System	Summit	Theta	n/a	n/a
Network Diameter	6	7	5	4
Max Hop Count	6	12	8	6
Levels	3	n/a	n/a	n/a
Pods/Groups	10	8	25	26
Routers per Pod/Group	36	96	16	13
Planes/Rails	1/1	1/1	1/1	1/1
Compute Nodes	3,240	3,072	3,200	3,042
Routers	468	768	400	338
Total Links	9,720	11,424	8,600	6,253
Router Radix Total	36	48	36	36
Router Radix Used	36	42	35	28
Router Latency	90ns	90ns	90ns	90ns
Routing Protocol	static	adaptive	adaptive	adaptive
Virtual Channels	1	8	6	4
Buffer Space per VC	64 KB	64 KB	64KB	64 KB
Local Connections	18	30	15	6
Global Connections	n/a	8	12	13
Compute Node Connections	18	4	8	9
Link Speed	100 Gbps	100 Gbps	100 Gbps	100 Gbps
Injection Bandwidth	316 Tbps	300 Tbps	313 Tbps	297 Tbps
Network Bandwidth	648 Tbps	835 Tbps	540 Tbps	321 Tbps

group are arranged in the form of a 16x6 "2D" matrix where routers in the same row have all-to-all connections and routers in the same column have all-to-all connections. The second configuration, which we term Dragonfly-1D has much smaller groups consisting of a 16x1 "1D" array of routers with all-to-all connections. For each configuration, we use adaptive routing, which dynamically chooses between minimal and non-minimal paths depending upon the network congestion.

3) Slim Fly: A new proposed topology is called the Slim Fly [11]. The Slim Fly is based on a specialized graph construction algorithm based on an McKay, Miller and Siran (MMS) graph that guarantees a maximum diameter of 2 (e.g., a high degree of node connectivity) and it has a very low cost in terms of the number of switches and links to construct. The major drawback of the Slim Fly network design is of implementation, as it can require an odd sized switch radix in order to obtain the guarantee on network diameter. Additionally, this network appears to not be easily divided such that parallel job network traffic will not interfere with each other, unlike the Fat-Tree or Torus. For consistency, we run Slim Fly using adaptive routing detailed in [29].

V. Workloads

The analysis of network performance benefits greatly from the existence of workloads that represent real system use cases. These representative workloads can be broken down into two categories, namely synthetic workloads and trace workloads. On the one hand, synthetic workloads are manually constructed by the user to replicate specific activity observed from applications running on the system. Trace workloads, on the other hand, are generated from an actual execution of an application and are therefore able to replicate the activity of that application on the system. In this work, we investigate synthetic and trace application workloads from both neuromorphic and CPU architectures.



(a) MNIST (b) CIFAR (c) Hopfield

Fig. 3: Communication pairing diagrams showing the level of communication between all neuromorphic chips in each neural network application. Sending chip IDs are on the left axis and receiving chip IDs are on the bottom axis.

TABLE II: Neuromorphic Workload Comparison

Workload	*Chips*	*Connectivity*	*Spikes/Tick*	*MB/Tick*	*Waits*
MNIST	1234	15 layers	577K	4.4MB	0
CIFAR	1024	15 layers	647K	4.9MB	0
Hopfield	1024	all-to-all	10.2M	80MB	0

A tick is 1ms of simulated time.

A. Neuromorphic

We created two trained neural network trace workloads and one synthetic workload to test the simulated distributed neuromorphic hardware systems. Using the IBM TrueNorth EEDN framework [34], we trained two convolutional neural networks based on common neural network datasets, MNIST and CIFAR. These applications were then imported into NeMo, scaled to thousands of chips, and simulated to obtain theoretical large-scale communication workloads. Starting from real TrueNorth neural network applications, we are able to generate approximations of traffic produced by potential large distributed neural networks. The following neuromorphic workloads are studied in this work.

1) MNIST: MNIST is a dataset containing 28x28 pixel images of handwritten digits 0 through 9 [35]. The MNIST workload used in this study is a convolutional network trained on the MNIST dataset to classify handwritten digits. This network workload is of interest because it leverages the common convolutional neural network structure. MNIST can also be considered a gold standard benchmark for neural networks as a number of different implementations and results have been very well documented. For the implementation, we are using a prebuilt MNIST configuration provided in the set of IBM examples. The network consists of 15 layers of neurons using 2,468 neurosynaptic cores which is roughly 60% of one TrueNorth chip. For the large-scale neuromorphic studies, we have scaled the workload from 1 chip to 1234 chips. Figure 3a presents a heatmap showing the layered connections between chips in the network.

2) CIFAR: CIFAR is a much larger and complex image dataset containing 32x32 pixel images of objects that belong to one of 100 different classes [36]. The classes include various animals, foods, and inanimate objects. The workload used in this study is again, a convolutional neural network implemented in the IBM TrueNorth ecosystem and provided as an IBM generated example. The network consists of 15 layers of neurons using 4,044 neurosynaptic cores, totalling

TABLE III: CPU Workload Comparison

Workload	*Processes*	*End Time*	*Msgs*	*Msg Size*	*Waits*
AMG	1,728	0.50ms	2.2M	0.79KB	101.1K
CR	1,000	258.48ms	39.9M	7.95KB	39.9M
MG	1,000	5.51ms	0.5M	9.30KB	0

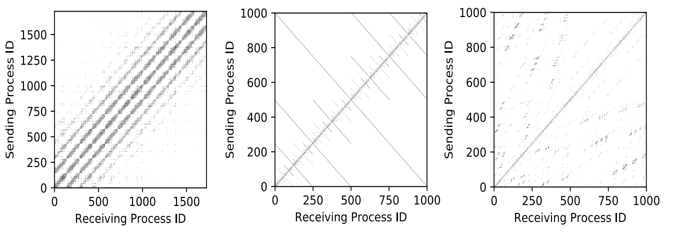
End time is the virtual time to replay the workload through the Fat-Tree configuration. Msg size is the average size of all messages transfered across all processes.

roughly 99% of one TrueNorth chip. The CIFAR workload again, represents the very popular convolutional network approach, but the increased size and complexity of the input data results in a more dense connectivity between layers as compared to MNIST. For the large-scale neuromorphic studies, we have scaled the workload from 1 chip to 1024 chips. Figure 3b shows a heatmap indicating the different layers and connections between chips in the network.

3) Hopfield: The third network workload is a synthetic implementation of a Hopfield network. We have chosen Hopfield because it represents another commonly used neural network approach posing a different communication workload from the TrueNorth implementations of MNIST and CIFAR. Unlike the layered connectivity of the CIFAR and MNIST convolutional networks, the Hopfield network consists of one layer of fully connected neurons [31]. The high degree of connectivity poses it's own concerns for large-scale HPC networking and therefore an interesting case to study. The Hopfield network is also a recurrent network, and unlike the convolutional networks of CIFAR and MNIST, TrueNorth does not currently support recurrent networks. Therefore, we implemented a synthetic representation of the Hopfield network workload at the HPC network level within the CODES framework. In Section VI, we configure the Hopfield workload to utilize all neuromorphic chips in each simulated HPC system configuration. For comparison with the MNIST and CIFAR workloads, a chip connectivity heatmap for the Hopfield workload running on 1024 chips is provided in Figure 3c along with workload details in Table II. While Hopfield has more data transfered per tick than CIFAR and MNIST workloads, its communication spans the entire network following all-to-all connections instead of tightly grouped layered connections as seen in the convolutional workloads. Note, the data values presented in Table II are aggregate totals for the entire workload and not per MPI rank.

B. Traditional CPU Jobs

To test the performance implications of neuromorphic applications running in parallel with traditional CPU jobs, we use three trace workloads from common HPC scale CPU applications. All three applications are part of the DOE Design Forward initiative. Conforming to the DUMPI trace format, each workload is read in from file by the CODES framework and replayed over the configured HPC network system alongside the neuromorphic workloads allowing network performance analysis studies. The three CPU workloads considered are Algebraic Multigrid, Crystal Router, and Multigrid each of which emphasize a different load on the HPC network.



(a) AMG (b) Crystal Router (c) Multigrid

Fig. 4: Communication pairing diagrams showing the level of communication between all CPU MPI processes in each CPU application. Sending process IDs are on the left axis and receiving process IDs are on the bottom axis.

1) AMG: The AMG application is a parallel Algebraic Multigrid solver used for unstructured grids developed at LLNL [37]. AMG decomposes a data grid into 3D chunks, resulting in a 3D nearest neighbor communication pattern among processes as shown in Figure 4a. Our application trace of AMG, collected for 13824 MPI processes, represents the communication in a single cycle of the grid sequence. MPI operations account for 53% of its run time.

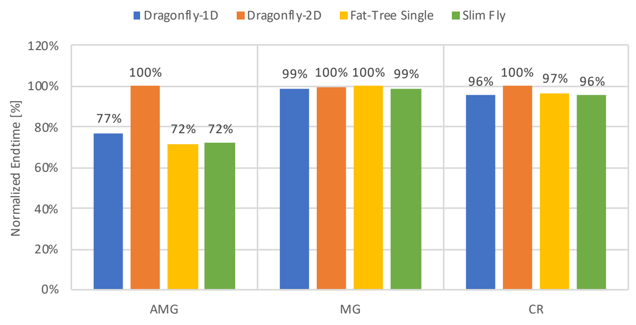
2) CR: Crystal Router represents the staged many-to-many communication pattern of the highly scalable Nek5000 spectral element code developed at ANL [38]. The MPI processes in Crystal Router perform large data transfers in the form of a n-dimensional hypercube as shown in Figure 4b. The trace for 1000 MPI processes shows an overall communicating time of 69% of the application's run time.

3) MG: The Geometric Multigrid miniapp implements a single production cycle of the linear solver used in BoxLib [39], an adaptive mesh refinement code. Multigrid processes communicate along the diagonals, which results in nearest-neighbor like communication and is shown in Figure 4c. The Multigrid communication trace used in the paper has been executed on 1000 MPI processes with roughly 4% of its run time spent in MPI communication.

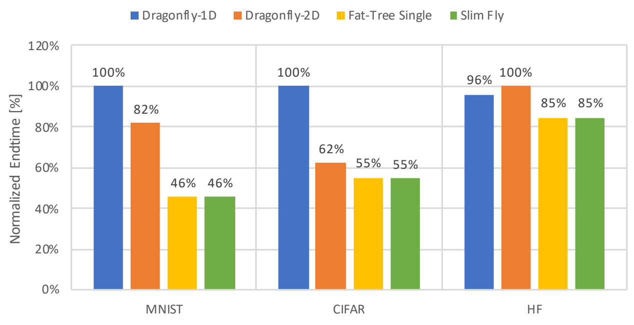
In summary, Table III shows Crystal Router is a highly synchronized application that consistently injects data over a longer period of time. The one-to-one ratio of message transfers to waits makes Crystal Router a latency sensitive workload. Multigrid, in comparison, can be classified as a bandwidth sensitive workload because it has no synchronization between MPI processes and sends a smaller number of larger messages. Finally, AMG is a network resource heavy application injecting relatively small messages in a short amount of time with a decent amount of synchronization.

VI. Evaluation

To test the impact of large-scale neuromorphic computing, we conducted a series of studies to quantify performance of these proposed systems. We start with predicting both CPU and neuromorphic application performance in single-job executions and then follow up with an interference study in a hybrid multi-job environment.



(a) CPU Performance



(b) Neuro Performance

Fig. 5: Simulated end time results for CPU and Neuromorphic applications running alone on the Dragonfly-1D, Dragonfly-1D, Fat-Tree, and Slim Fly HPC systems. End times are normalized within a workload to the slowest performing topology result. Lower is better.

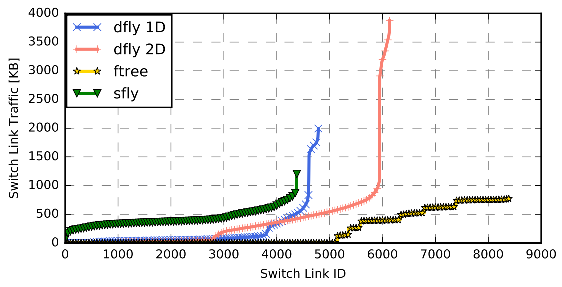
A. Data Visualization and Nomenclature

In this work, we investigate end time performance as well as multi-job interference, and we use observed link traffic to explain the differences between the network topologies. To study performance, we collect and analyze the reported virtual end time in which the simulated application workloads finish. The applications studied have widely varying end times, so we normalize each end time to clearly show how much faster each topology is in comparison to the worst-case performer. To study the interference of multi-job executions, we compute and analyze the slowdown. This is computed as the difference in end time of the application in the multi-job and single-job executions divided by the single-job end time. Performance and interference results are presented in the form of bar plots. The bars within each bar plot are grouped into clusters representing a single-job or multi-job execution. Multi-job executions are denoted by clusters with labels following the format ”X in pres. of Y.” The X refers to the workload for which the end time is collected and Y being the workload running concurrently on the system.

Link traffic data is presented in line plots sorted by increasing value. Each point on a switch traffic plot represents the aggregate traffic transmitted across a global link over the length of the simulation. In the case of Fat-Tree, global links are those connecting the core and aggregate level switches. For clarity, markers on switch traffic plots are shown every 50 points.

B. Single-Job Performance

In this study, we run each CPU and neuromorphic workload individually to generate a baseline performance on each of the four HPC network configurations. The CPU results are collected over the entire workload execution while the neuromorphic results are collected for only one tick of execution. Figure 5 shows the results for each network clustered by the application labeled on the x-axis. The end times for each network within a clustered group are normalized with respect to the worst-case end time within each clustered group of results.

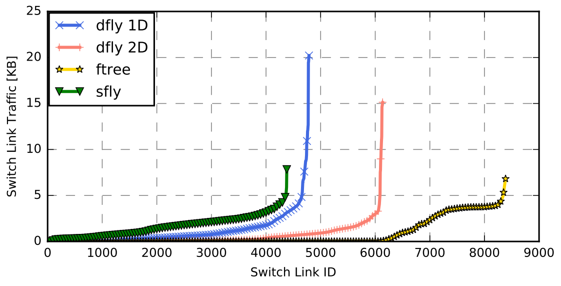


(a) Switch Traffic

Fig. 6: Aggregate global link traffic for Dragonfly-1D, Dragonfly-1D, Fat-Tree, and Slim Fly running the single-job AMG execution.

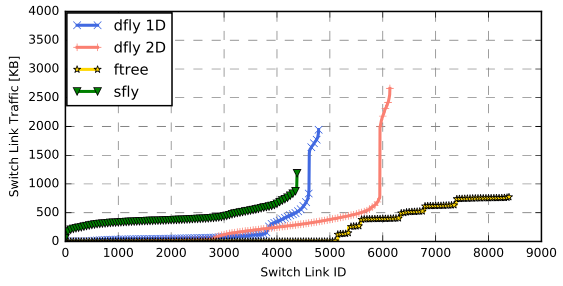
1) CPU: Focusing first on the CPU workloads, AMG shows the largest variance in end time between the network topologies. In this case, Dragonfly-1D is 23% faster than Dragonfly-2D, and best performance is observed on Slim Fly and Fat-Tree which are 28% faster than Dragonfly-2D. The traffic data presented in Figure 6, shows Dragonfly-2D has a small number of global links that are heavily saturated compared to the remaining links. High saturation of global links for Dragonfly indicates packets are traversing deeper into the network, taking additional hops, and increasing chances of congestion. The high utilization hot spots along the global link effect many compute nodes and result in the slowdown in run time performance.

Moving on to the remaining CPU workloads, we see MG and CR observe similar performance across all topologies. Note from Section V-B, MG and CR aren’t as communication intensive as AMG for a couple reasons. There are fewer processes, data is injected at a lower rate (quantity of data over time), and the communication patterns of MG and CR send more data directly along the diagonal. The combination of these workload characteristics results in less demand on the network than AMG and leads to all networks being able to finish the MG and CR workloads in roughly the same time when run individually.



(a) Switch Traffic

Fig. 7: Aggregate global link traffic for Dragonfly-1D, Dragonfly-1D, Fat-Tree, and Slim Fly running the single-job MNIST execution.



(a) Switch Traffic

Fig. 9: Aggregate global link traffic for Dragonfly-1D, Dragonfly-1D, Fat-Tree, and Slim Fly running the hybrid AMG-MNIST execution.

2) Neuro: Overall, the Hopfield workload achieves closer end times across all three HPC network topologies than the convolutional workloads of CIFAR and MNIST. In this case, Dragonfly-2D execution is 4% slower than Dragonfly-1D and both Fat-Tree and Slim Fly are 15% faster than the worst case Dragonfly-2D. While the Hopfield workload has more connections and sends more messages than MNIST and CIFAR, the all-to-all communication pattern helps to evenly distribute the messages across the entire HPC network and avoid hotspots. In this case, hop counts play a large role in end time as each extra hop incurs additional router traversal delays resulting in increased end-to-end message latency for the small 8 byte spike messages. Fat-Tree using static routing and Slim Fly using adaptive routing both have a max hop count of 6. Dragonfly-1D and Dragonfly-2D have larger max hop counts of 8 and 12 and end up 11% and 15% slower respectively than Fat-Tree and Slim Fly.

For MNIST and CIFAR, there is a significant difference in simulation end time between the HPC topologies as Fat-Tree and Slim Fly outperform worst-case performer Dragonfly-1D by 54% and Dragonfly-2D outperforms Dragonfly-1D by 18%. The layered connectivity and reduction in layer size of the convolutional workload focuses communication through a small subset of network links that can result in adversarial communication for the Dragonfly networks. Figure 7a shows Dragonfly-1D and 2D have global links that observe up to 2.7x and 2x more traffic than the worst case global links for Fat-Tree and Slim Fly. In this case, the specific convolutional network of MNIST translates to an adversarial communication mapping for the Dragonfly network resulting in adaptive routing moving traffic to global links to distribute the load.

CIFAR results are similar to MNIST with Fat-Tree and Slim Fly finishing in roughly half the time of Dragonfly-1D but in this case Dragonfly-2D also finishes much quicker at 38% less run time than Dragonfly-1D. CIFAR has the same number of layers as MNIST but the layers implement different numbers of connections and placements of connections again resulting in hotspots for Dragonfly-1D. Interestingly, it doesn't translate to as severe of hotspots for Dragonfly-2D. Running the Dragonfly topologies with random allocation of chips to nodes results in performance similar to Fat-Tree and Slim Fly and confirms the long end time is a result of an adversarial mapping of the convolutional network to the Dragonfly topologies. In summary, the performance for convolutional neural network applications can be highly dependent on the specific connections of each application.

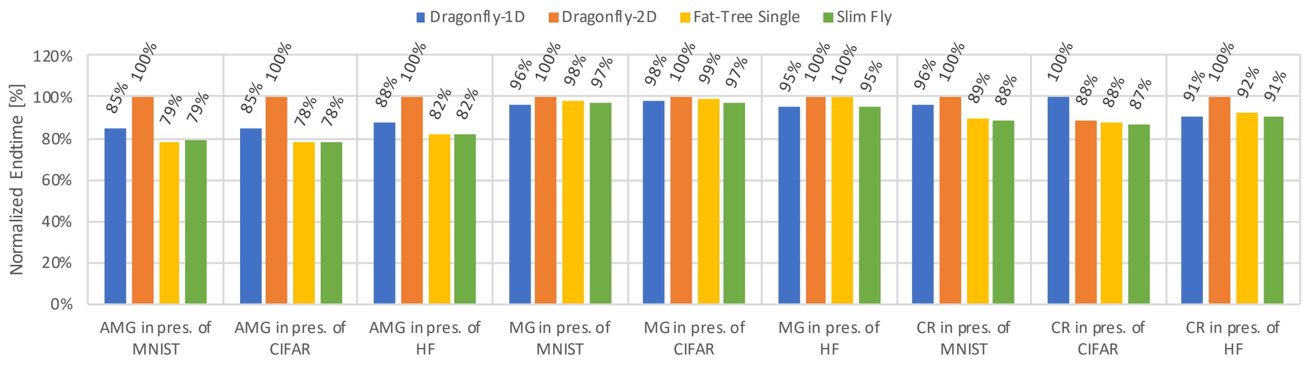
C. Hybrid Multi-Job Performance

To study the interference of neuromorphic and traditional CPU workloads running in a hybrid compute system, we execute the neuromorphic workloads in parallel with the three CPU application workloads on the four HPC network topologies. Each CPU and neuromorphic workload is again mapped linearly to compute nodes with one CPU and one Neuromorphic MPI process per compute node. We use the simulation end time to measure the high-level interference.

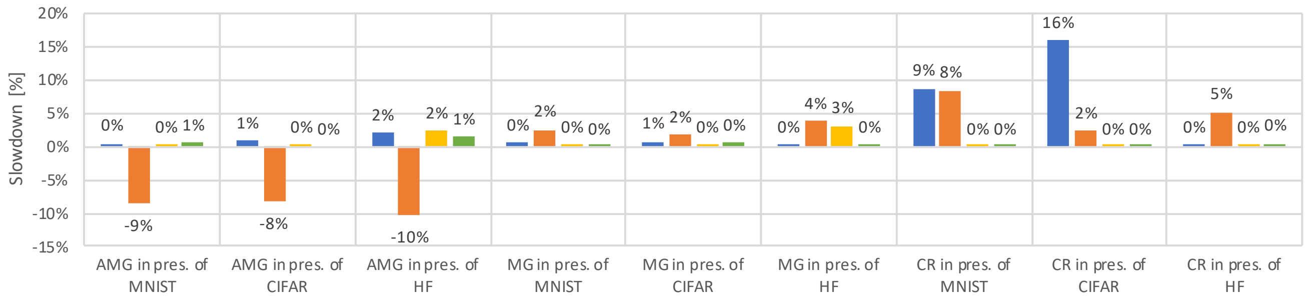
Measuring interference between the tick synchronized neuromorphic applications and unrestrained CPU architecture workloads poses a unique challenge. To study the effect of CPU applications on neuromorphic workloads, the neuromorphic workload is executed in parallel with the CPU application for the entire length of the CPU application. The longer the CPU workload end time, the more ticks of the neuromorphic workload are simulated. Since the progression of neuromorphic workloads is governed by the 1ms tick delay, and not simply by the availability of work, we compute the end time for the neuromorphic workloads, during multi-job executions, to be the average time to complete the work within each 1ms tick. This allows us to extract the true effect of the CPU workloads on the neuromorphic spike generation and sending process within each tick.

1) CPU in Presence of Neuro: Overall, end time performance and interferance results in Figure 8 show AMG, Multigrid and Crystal Router CPU workloads with little effect from running in the presence of neuromorphic workloads. Multigrid end time performance closely matches the trends observed for single-job execution with all four network topolgies finishing within a max of 5% of one another. Of the four neuromorphic workloads, Hopfield creates the most interference for Multigrid resulting in only 4% and 3% slowdowns for Multigrid.

The CPU workload Crystal Router, on the other hand, shows the largest negative effect observing up to 16% slower end time running in parallel with CIFAR. Dragonfly-1D sees more slowdown than others to make it the worst-case performer



(a) CPU Job Performance (in presence of Neuro)



(b) CPU Job Interference (in presence of Neuro)

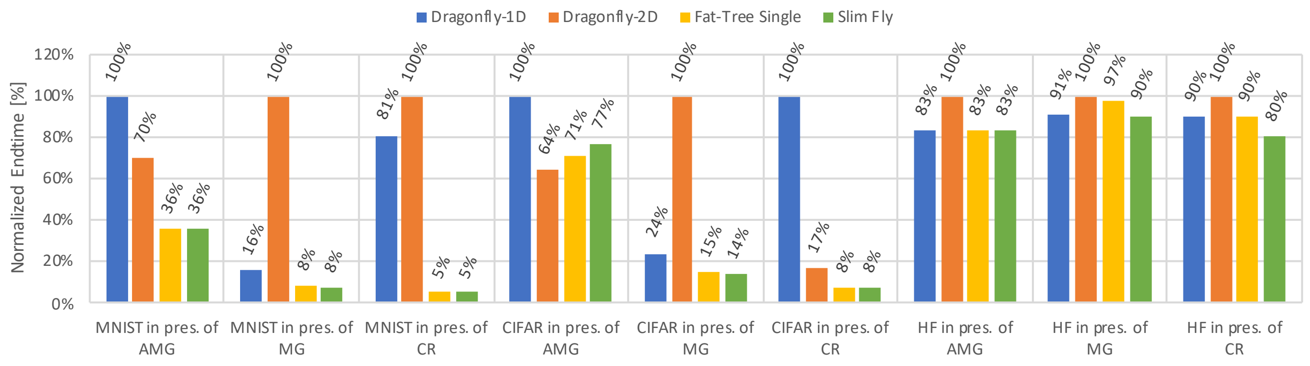
Fig. 8: The top subfigure presents simulated end time results for the CPU applications when running in the presence of neuromorphic applications on the Dragonfly-1D, Dragonfly-1D, Fat-Tree, and Slim Fly HPC systems. End times are normalized within each workload pairing to the slowest performing topology result. The bottom subfigure presents the net slowdown in end time performance for CPU workloads when running in the presence of neuromorphic workloads. In both subfigures, lower is better.

when running in the presence of MNIST and CIFAR. The convolutional neural networks present adversarial traffic creating hotspots along critical paths for Crystal Router. Crystal Router is a highly synchronized workload performing a wait operation after each send, making it a latency sensitive application. The global link hot spots seen from the MNIST and CIFAR convolutional NN type applications in Figure 7, force some traffic in CR to take longer paths through the network to avoid congestion resulting in delays for both Dragonfly configurations.

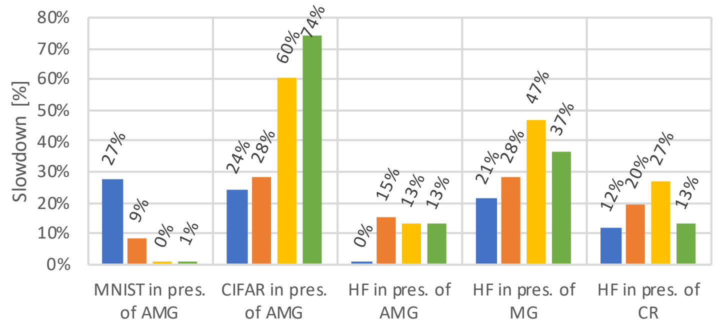
Finally, the AMG CPU workload shows some interesting results with Dragonfly-2D improving performance in the presence of all three neuromorphic workloads while the other network topologies see up to 2% slowdown. Dragonfly-2D sees improved performance benefiting from additional traffic on the network to increase network utilization and decrease congestion on global links. Switch traffic for the AMG workload running in the presence of MNIST is presented in Figure 9 and shows a small decrease in traffic for all Dragonfly-2D global links. Most notably, the max link traffic decreased by roughly 30% compared to the max link traffic observed in the single-job AMG execution.

2) Neuro in Presence of CPU: The results for neuromorphic application end time performance and interference in the presence of CPU workloads is presented in Figure 10. Overall, Neuromorphic workloads are highly susceptible to network interference from CPU workloads. Both Dragonfly-1D and Dragonfly-2D observe severe slowdowns for the convolutional NN workloads of MNIST and CIFAR when running in the presence of Multigrid and Crystal Router. Both Multigrid and Crystal Router have a large portion of their communication along the diagonal and that consistent nearest neighbor pattern can saturate local links. Adaptive routing, in response, pushes the traffic further into the network to global links to minimize congestion and resulting in overlap with the global link hotspots for the convolutional workloads. Fat-Tree, using static routing always take the same shortest path regardless of congestion. Adaptive routing for Slim Fly performs better largely because of the smaller hop counts for minimal and nonminimal paths as well as the ability of minimal routing to naturally distribute the workload throughout the network to utilize all available resources.

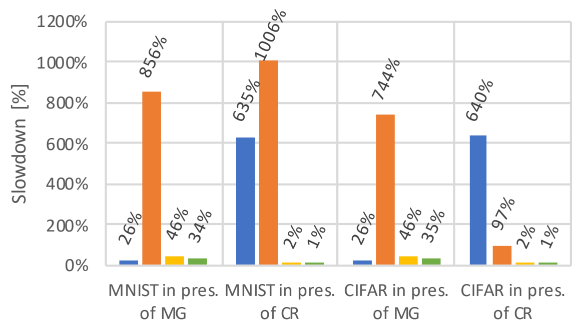
The Hopfield neuromorphic workload observes the least amount of slowdown running in the presence of CPU workloads. The average slowdown for Hopfield across all topologies and CPU workloads is 21% compared to 148% and 220% for CIFAR and MNIST. Additionally, end times are consistently close between the topologies for Hopfield further indicating the all-to-all communication pattern is equally adversarial for all topologies and with distributed communication



(a) Neuro Job Performance (in presence of CPU)



(b) Neuro Job Interference (in pres. of CPU) [Mild Cases]



(c) Interference (Neuro in pres. of CPU) [Severe Cases]

Fig. 10: The top subfigure presents simulated end time results for neuromorphic applications when running in the presence of CPU applications on the Dragonfly-1D, Dragonfly-1D, Fat-Tree, and Slim Fly HPC systems. End times are normalized within each workload pairing to the slowest performing topology result. The bottom subfigures present the net slowdown in end time performance for neuromorphic workloads when running in the presence of CPU workloads. In all subfigures, lower is better.

that isn't readily susceptible to traditional CPU workloads.

Neuromorphic workload performance also varies widely between the tested network topologies. On average, Slim Fly, Fat-Tree, Dragonfly-1D, and Dragonfly-2D are 45%, 46%, 76%, and 83% respectively faster than the worst case performing topology in each hybrid workload configuration. While the Dragonfly configurations many times have the worst-case end times, we do believe the causes (namely link traffic hot spots) can be significantly reduced and possibly even alleviated with approaches to better balance the communication workloads such as minimal path bias, an adaptive threshold, or randomly mapping processes to compute nodes.

VII. Conclusion

A new simulation workflow is presented allowing exploration of real, extreme-scale neuromorphic workloads and their interference with traditional CPU workloads when executed on a hybrid HPC system. We study single-job and mutli-job executions analyzing performance of systems configured with Dragonfly, Fat-Tree and Slim Fly network topologies.

Neuromorphic workloads representing convolutional neural network and Hopfield network applications pose little effect on traditional CPU applications when running in parallel in a multi-job hybrid HPC environment. In the worst-case, the Crystal Router CPU workload observed 16% slower end time performance. Furthermore, performance for the Dragonfly-1D, Dragonfly-2D, Fat-Tree, and Slim Fly topologies matched the trends of the single-job CPU end times with Fat-Tree and Slim Fly finishing faster than the Dragonfly configurations with some instances where Dragonfly-1D matched Fat-Tree and Slim Fly's performance.

The neuromorphic workloads, on the other hand, are largely susceptible to the network workloads generated by the traditional CPU applications. Convolutional NN workloads MNIST and CIFAR have been shown to observe up to 10x slowdown in end time running in parallel with CPU workloads. Slim Fly and Fat-Tree are roughly 40% faster on average than Dragonfly-1D and Dragonfly-2D for neuromorphic workloads in the presence of CPU workloads. However the Dragonfly configurations may still achieve the performance of Fat-Tree and Slim Fly with approaches such as minimal path bias, adaptive thresholds, or randomly mapping processes to compute nodes.

Acknowledgement

This work is supported by the Air Force Research Laboratory (AFRL), under contract FA8750-15-2-0078 and by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

References

[1] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014. [Online]. Available: http://science.sciencemag.org/content/345/6197/668

[2] M. M. Khan, D. R. Lester, L. A. Plana, A. Rast, X. Jin, E. Painkras, and S. B. Furber, "Spinnaker: Mapping neural networks onto a massively-parallel chip multiprocessor," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 2849–2856.

[3] K. Meier, "A mixed-signal universal neuromorphic computing system," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 4.6.1–4.6.4.

[4] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[5] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, January 2018.

[6] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable dragonfly topology," in *ACM SIGARCH Computer Architecture News*, vol. 36, no. 3. IEEE Computer Society, 2008, pp. 77–88.

[7] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray cascade: A scalable hpc system based on a dragonfly network," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 103:1–103:9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389136

[8] F. Petrini and M. Vanneschi, "k-ary n-trees: High performance networks for massively parallel architectures," in *Parallel Processing Symposium, 1997. Proceedings., 11th International*. IEEE, 1997, pp. 87–93.

[9] M. Besta and T. Hoefler, "Slim fly: A cost effective low-diameter network topology," in *High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for*. IEEE, 2014, pp. 348–359.

[10] Secretary of Energy Advisory Board, "*DoE Report on Task Force in High Performance Computing*." [Online]. Available: https://www.energy.gov/sites/prod/files/2015/02/f19/HPC%20Task%20Force%20Follow%20Up%20Response%20V7%20Final%2019-DEC-14_FINAL.pdf

[11] DoE Office of Science, "Draft Coral-2 Build Statement of Work," Tech. Rep. LLNL-TM-7390608, February 2018. [Online]. Available: http://procurement.ornl.gov/rfp/CORAL2/03_CORAL-2-SOW-DRAFT-v20.pdf

[12] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *CoRR*, vol. abs/1705.06963, 2017. [Online]. Available: http://arxiv.org/abs/1705.06963

[13] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J. M. Bower, M. Diesmann, A. Morrison, P. H. Goodman, F. C. Harris, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A. P. Davison, S. El Boustani, and A. Destexhe, "Simulation of networks of spiking neurons: A review of tools and strategies," *Journal of Computational Neuroscience*, vol. 23, no. 3, pp. 349–398, Dec 2007. [Online]. Available: https://doi.org/10.1007/s10827-007-0038-6

[14] M. Hines, *The Neuron Simulation Program*. Boston, MA: Springer US, 1994, pp. 147–163. [Online]. Available: https://doi.org/10.1007/978-1-4615-2736-7_8

[15] H. Markram, "The Blue Brain Project," *Nature Reviews Neuroscience*, vol. 7, no. 2, pp. 153 EP —-160, Feb. 2006.

[16] A. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Riesen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "The structural simulation toolkit," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 4, pp. 37–42, March 2011, http://doi.acm.org/10.1145/1964218.1964225.

[17] N. Jiang, J. Balfour, D. U. Becker, B. Towles, W. J. Dally, G. Michelogiannakis, and J. Kim, "A detailed and flexible cycle-accurate network-on-chip simulator," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, vol. 00, 04 2013, pp. 86–96. [Online]. Available: doi.ieeecomputersociety.org/10.1109/ISPASS.2013.6557149

[18] C. Minkenberg and G. Rodriguez, "Trace-driven co-simulation of high-performance computing systems using omnet++," in *Proceedings of the 2Nd International Conference on Simulation Tools and Techniques*, ser. Simutools '09. ICST, Brussels, Belgium, Belgium: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009, pp. 65:1–65:8. [Online]. Available: http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

[19] J. Sawada, F. Akopyan, A. S. Cassidy, B. Taba, M. V. Debole, P. Datta, R. Alvarez-Icaza, A. Amir, J. V. Arthur, A. Andreopoulos, R. Appuswamy, H. Baier, D. Barch, D. J. Berg, C. d. Nolfo, S. K. Esser, M. Flickner, T. A. Horvath, B. L. Jackson, J. Kusnitz, S. Lekuch, M. Mastro, T. Melano, P. A. Merolla, S. E. Millman, T. K. Nayak, N. Pass, H. E. Penner, W. P. Risk, K. Schleupen, B. Shaw, H. Wu, B. Giera, A. T. Moody, N. Mundhenk, B. C. Van Essen, E. X. Wang, D. P. Widemann, Q. Wu, W. E. Murphy, J. K. Infantolino, J. A. Ross, D. R. Shires, M. M. Vindiola, R. Namburu, and D. S. Modha, "Truenorth ecosystem for brain-inspired computing: Scalable systems, software, and applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 12:1–12:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014904.3014920

[20] IBM, "U.s. air force research lab taps ibm to build brain-inspired ai supercomputing system," *News Room*, Jun 2017. [Online]. Available: https://www-03.ibm.com/press/us/en/pressrelease/52657.wss

[21] R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A scalable simulator for an architecture for cognitive computing," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, Nov 2012, pp. 1–11.

[22] M. Plagge, C. D. Carothers, and E. Gonsiorowski, "Nemo: A massively parallel discrete-event simulation model for neuromorphic architectures," in *Proceedings of the 2016 Annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*, ser. SIGSIM-PADS '16. New York, NY, USA: ACM, 2016, pp. 233–244. [Online]. Available: http://doi.acm.org/10.1145/2901378.2901392

[23] C. D. Carothers, D. Bauer, and S. Pearce, "Ross: A high-performance, low-memory, modular time warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002.

[24] S. Carrillo, J. Harkin, L. J. McDaid, S. Pande, S. Cawley, B. McGinley, and F. Morgan, "Hierarchical Network-on-Chip and Traffic Compression for Spiking Neural Network Implementations," in *2012 Sixth IEEE/ACM International Symposium on Networks-on-Chip (NoCS)*. IEEE, 2012, pp. 83–90.

[25] S. Carrillo, J. Harkin, L. J. McDaid, F. Morgan, S. Pande, S. Cawley, and B. McGinley, "Scalable Hierarchical Network-on-Chip Architecture for Spiking Neural Network Hardware Implementations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2451–2461, 2013.

[26] 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*, vol. PP, no. 99, pp. 1–1, 2017.

[27] M. Mubarak, P. Carns, J. Jenkins, J. K. Li, N. Jain, S. Snyder, R. Ross, C. D. Carothers, A. Bhatele, and K.-L. Ma, "Quantifying i/o and communication traffic interference on dragonfly networks equipped with burst buffers," in *Cluster Computing (CLUSTER), 2017 IEEE International Conference on*. IEEE, 2017, pp. 204–215.

[28] N. Wolfe, M. Mubarak, N. Jain, J. Domke, A. Bhatele, C. D. Carothers, and R. B. Ross, "Preliminary performance analysis of multi-rail fat-tree networks," in *Proceedings of the 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE Press, 2017, pp. 258–261.

[29] N. Wolfe, C. D. Carothers, M. Mubarak, R. Ross, and P. Carns, "Modeling a million-node slim fly network using parallel discrete-event simulation," in *Proceedings of the 2016 annual ACM Conference on SIGSIM Principles of Advanced Discrete Simulation*. ACM, 2016, pp. 189–199.

[30] M. Mubarak, C. D. Carothers, R. B. Ross, and P. Carns, "Enabling parallel simulation of large-scale hpc network systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 1, pp. 87–100, 2017.

[31] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982. [Online]. Available: http://www.pnas.org/content/79/8/2554

[32] M. Mubarak, N. Jain, J. Domke, N. Wolfe, C. Ross, J. K. Li, A. Bhatele, C. D. Carothers, K.-L. Ma, and R. B. Ross, "Toward reliable validation of hpc network simulation models," *2017 Winter Simulation Conference (WSC)*, pp. 659–674, 2017.

[33] N. Jain, A. Bhatele, L. H. Howell, D. Böhme, I. Karlin, E. A. León, M. Mubarak, N. Wolfe, T. Gamblin, and M. L. Leininger, "Predicting the performance impact of different fat-tree configurations," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: ACM, 2017, pp. 50:1–50:13. [Online]. Available: http://doi.acm.org/10.1145/3126908.3126967

[34] S. K. Esser, P. A. Merolla, J. V. Arthur, A. S. Cassidy, R. Appuswamy, A. Andreopoulos, D. J. Berg, J. L. McKinstry, T. Melano, D. R. Barch, C. di Nolfo, P. Datta, A. Amir, B. Taba, M. D. Flickner, and D. S. Modha, "Convolutional networks for fast, energy-efficient neuromorphic computing," *Proceedings of the National Academy of Sciences*, vol. 113, no. 41, pp. 11 441–11 446, 2016. [Online]. Available: http://www.pnas.org/content/113/41/11441

[35] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[36] A. Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

[37] *Co-design at Lawrence Livermore National Laboratory*, "*Algebraic Multigrid Solver (AMG)*," (Accessed on: Apr. 19, 2015). [Online]. Available: https://codesign.llnl.gov/amg2013.php

[38] J. Shin, M. W. Hall, J. Chame, C. Chen, P. F. Fischer, and P. D. Hovland, "Speeding up nek5000 with autotuning and specialization," in *Proceedings of the 24th ACM International Conference for Supercomputing*. ACM, 2010, pp. 253–262.

[39] Department of Energy, "*AMR Box Lib*." [Online]. Available: https://ccse.lbl.gov/BoxLib/

VIII. Artifact Description Appendix

A. Abstract

This artifact describes the end to end process of installing, configuring, and executing the neuromorphic HPC simulations to reproduce results presented in our paper. All datasets and code repositories are provided along with hardware and software dependencies.

B. Description

1) Check-list (artifact meta information):

- **Algorithm:** Discrete-event simulation
- **Simulation Program:** MPI and C/C++
- **Analysis Program:** Python 2.7 and Matplotlib 2.2.2
- **Compilation:** MPI: MPICH 3.2, C/C++: GNU gcc 5.4
- **Binary:** MPI executable
- **Data set:** Publicly available application traces
- **Run-time environment:** Linux
- **Hardware:** Any single machine, cluster or supercomputer with x86_64 processors
- **Output:** CSV files with network statistics collected at MPI, compute node and router levels
- **Experiment workflow:** Git clone project, download application traces, run execution scripts, run post-process analysis scripts
- **Publicly available?:** Yes

2) How software can be obtained (if available): In this work, there are three required simulation frameworks, and one optional package of scripts for execution, post-processing and analyzing the experiments in the paper. The latest versions of all required material can cloned from publicly available online repositories. The following is a list of the packages and their location url.

- ROSS: https://github.com/carothersc/ROSS.git
- NeMo: https://github.com/markplagge/NeMo.git
- DUMPI: https://github.com/sstsimulator/sst-dumpi.git
- CODES: https://xgitlab.cels.anl.gov/codes/codes.git
- Scripts: Will be released after double blind review

3) Hardware dependencies: To run the NeMo chip scaling simulations, the compute system needs to have a minimum of 128GB. Simulations have been executed on Intel® Xeon® E5-2643v1, E5-2650v2, E5-2640v3, and E5-2640v4 processors.

4) Software dependencies: All simulation frameworks have been tested on Scientific Linux 6 and Ubuntu 16.04 and are expected to run on other Linux distributions.

5) Datasets: The two datasets consisting of both cpu application traces and neuromorphic application traces can be downloaded from publicly available websites or repositories. The following is a list of the packages and their location.

- CPU Traces: http://portal.nersc.gov/project/CAL/designforward.htm
- Neuromorphic Traces: Will be released after double blind review

The CPU application traces are in the DUMPI format. The neuromorphic workloads are available in two different formats. The first follows a custom architecture specific format used in the IBM TrueNorth ecosystem and contains a complete representation of the neural network as well as spiking input data. This format is needed to run the neuromorphic application on the IBM TrueNorth processor or in the NeMo simulator. The second format for the neuromorphic application traces is a representation of the neuromorphic application's communication workload after having been scaled in NeMo to run on many neuromorphic chips. The second format is used by CODES to replay the application's communication through the simulated HPC systems. Both neuromorphic formats are provided in the online repository but the communication format can also be generated from the architecture specific format using NeMo.

C. Installation

First, clone the simulation framework repositories listed in section VIII-B2 to the local system. Each framework can be manually installed and configured following their respective documentation. Alternatively, an automated approach for installing, configuring and setting up a complete working environment is provided by the `build.sh` shell script located in the "Scripts" repository. Simply edit the `src`, `build`, and `install` variables with the appropriate paths on the local machine and then execute the script.

D. Experiment workflow

After installing and configuring all simulation frameworks, the first step is running NeMo to generate a many-chip communication workload file from the input neuromorphic application traces. This is done with the following lines of code:

```
$ mpirun -n 2 ./NeMo --cores=4096 --extramem
    =155535000 --network --end=300 --synch=5
      --svouts --svm --svs --gvt-interval=4
```

Pregenerated communication workload files exist in the neuromorphic traces repo and allow for the previous step to be skipped if desired.

Next, the HPC network simulations are performed with the CODES framework using the supplied `execution.sh` script in the "Scripts" repository. The script allows selection of CPU and neuromorphic workloads using the `NEUROMORPHIC_JOB` and `CPU_JOB` variables. Network topology can also be selected using the `NET_MODEL`. Additional variables exist in the script allowing further control of the model parameters and execution of the simulator. Each variable is described in the script along with available options.

E. Evaluation and expected result

Each simulation execution will generate a number of output log files in CSV format containing network performance statistics including average hop counts, packet latencies, bytes transferred, and simulated end time. If using the execution script, all output files, input configuration files, and the standard output log will be placed in one folder labeled with the simulation metrics selected for the simulation.

Additional post processing and visualization scripts are provided in the "Scripts" repository to generate the figures presented in the paper. These scripts are written in Python 2.7 and utilize the matplotlib plotting framework, however any plotting utility can be used.