Massively Parallel Causal Inference of Whole Brain Dynamics at Single Neuron Resolution

Wassapon Watanakeesuntorn*, Keichi Takahashi*, Kohei Ichikawa*, Joseph Park†, George Sugihara‡, Ryousei Takano§, Jason Haga§, Gerald M. Pao¶

* Nara Institute of Science and Technology, Nara, Japan,
{wassapon.watanakeesuntorn.wq0, keichi, ichikawa}@is.naist.jp
† U.S. Department of the Interior, Florida, USA,
josephpark@ieee.org
‡ University of California San Diego, California, USA,
gsugihara@ucsd.edu
§ National Institute of Advanced Industrial Science and Technology, Tsukuba, Japan,
{takano-ryousei, jh.haga}@aist.go.jp
¶ Salk Institute for Biological Studies, California, USA,
pao@salk.edu

***Abstract*—Empirical Dynamic Modeling (EDM) is a nonlinear time series causal inference framework. The latest implementation of EDM, cppEDM, has only been used for small datasets due to computational cost. With the growth of data collection capabilities, there is a great need to identify causal relationships in large datasets. We present mpEDM, a parallel distributed implementation of EDM optimized for modern GPU-centric supercomputers. We improve the original algorithm to reduce redundant computation and optimize the implementation to fully utilize hardware resources such as GPUs and SIMD units. As a use case, we run mpEDM on AI Bridging Cloud Infrastructure (ABCI) using datasets of an entire animal brain sampled at single neuron resolution to identify dynamical causation patterns across the brain. mpEDM is 1,530× faster than cppEDM and a dataset containing 101,729 neuron was analyzed in 199 seconds on 512 nodes. This is the largest EDM causal inference achieved to date.**

***Keywords*-Empirical Dynamic Modeling, Causal Inference, Parallel Distributed Computing, GPU, High-Performance Computing, Neuroscience**

I. Introduction

Reverse-engineering and building a digital reconstruction of the brain is one of the greatest scientific challenges of today. A recent study on the mouse cortex [1] showed that 97% of the possible connections between neurons exist. This result suggests that it is likely more informative to investigate the dynamic interactions between neurons rather than the static connectivity between them to fully understand the function of the brain. Based on this insight, we are building mathematical and computational tools to analyze the dynamic interactions between neurons based on Empirical Dynamic Modeling (EDM).

EDM is a nonlinear time series causal inference framework based on the generalized Takens' embedding theorem on state space reconstruction [2]. EDM is used to study and predict the behavior of nonlinear dynamical systems. Convergent Cross Mapping (CCM) is one of the EDM algorithms that allows to estimate the existence and strength of the causal strength between two time series in a dynamical system [3].

In this study, we utilize CCM to infer the causal relationships between every neuron in an entire brain and construct a causal map that describes the dynamic interactions among neurons. For this purpose, we have recorded the neural activity (*i.e.* firing rate) of an entire larval zebrafish brain at singe-neuron resolution by using light sheet fluorescence microscopy. The original implementation of EDM, cppEDM, has mostly been used for individual time series of relatively short length and and mostly small numbers of variables for its computational cost. Since a larval zebrafish brain contains approximately 10^5 neurons, a staggering number of 10^{10} cross mappings need to be performed in total. CCM of this enormous scale has never been achieved so far because of the sheer amount of computation required.

The goal of this paper is to develop a highly scalable and optimized implementation of EDM that is able to analyze the whole zebrafish brain dataset within a reasonable time. We present mpEDM[1], a parallel distributed implementation of EDM optimized for execution on modern GPU-centric supercomputers. We improve the original algorithm in cppEDM to reduce redundant computation and optimize the implementation to fully utilize hardware resources such as GPUs and SIMD units.

Our evaluation on AI Bridging Cloud Infrastructure (ABCI), Japan's most high performance supercomputer as of today, demonstrated the unprecedented performance of mpEDM. mpEDM was used to analyze a dataset containing the activity

[1]https://github.com/keichi/mpEDM

of 53,053 neurons in only 20 seconds using 512 ABCI nodes. In contrast, cppEDM took 8.5 hours to analyze the same dataset using the same number of nodes [4]. Furthermore, mpEDM analyzed a larger dataset containing 101,729 neurons in 199 seconds on 512 nodes. To our knowledge, this is the largest CCM calculation achieved to date. This result shows the potential for mpEDM and ABCI to analyze even larger datasets in the future.

The rest of this paper is structured as follows. Section II describes the background of this research and EDM algorithm. Section III explains our proposal to improve the algorithm of the EDM for parallelization and to support GPU architecture. Section IV evaluates the performance of mpEDM and presents the scientific outcomes obtained with mpEDM. Finally, section V concludes this paper and discusses future work.

II. Background

A. *Causal Map of the Zebrafish Brain at Single Neuron Resolution*

To understand the human brain activity dynamics with a complexity of 10^{11} neurons and 10^{15} synapses at single neuron resolution is currently a technically impossible task. Similarly a mouse brain with 7.6×10^7 neurons is not tractable because mammalian brains are opaque and it is impossible to image a complete mouse brain. With this in mind, the zebrafish embryo is an attractive model system with 120,000 neurons and transgenic technology as well as natural brain transparency. The zebrafish embryo is sufficiently complex to exhibit interesting behaviors and is technologically feasible to study to infer basic principles of systems neuroscience. Even in the case of the larval zebrafish with about 120,000 neurons we do not have the physical connectivity map, that is the connectome of the larval zebrafish, nor do we have the synaptic strengths which are pieces of information required to understand the brain starting from the physical connectivity.

Complicating this notion, recent work from the mouse brain shows that 97% of possible physical connections exist within the mouse cortex thus making it difficult to analyze. Given this difficulty, using an analogy of a city; to understand how a city works it will be easier to understand the city from the traffic patterns than from the street map. Thus, we wished to analyze the fish brain at single neuron resolution from a network activity dynamics perspective. Although imperfect, we used neural activity imaging data of an entire brain at single cell resolution in a behaving larval zebrafish (a transparent vertebrate) to extract all relationships in an intact vertebrate brain.

To achieve this, we recorded whole brain neural activity patterns in multiple animals experiencing hypoxia using a Selective Plane Illumination Microscope (SPIM) [5]. We obtained data from the entire 5-day-old larval brain (120,000 neurons) at 2 Hz in response to hypoxia for varying amounts of time typically ranging from 1,500 time steps to up to 8,000+ [6].

CCM allows the inference of causation from nonlinear time series even with substantial noise and complete absence of correlation [7], [8]. We used CCM and other tools from the EDM framework for the inference of existence, strength and sign of causal relationships within the neural activity network of the transparent larval fish brain [5]. CCM determines whether and how much causality exists between individual neurons. The adjacency in the network is determined by time delay cross mapping [8]. Predictive accuracy values give the interaction strength allow us to infer relationships within the neural network without observing the physical connectivity. As a test case, we have collected multiple data sets of lengths around 1600 time steps at 2 Hz which contain 50,000–80,000 active neurons in most cases. We have analyzed this data and show that the generated time series are suitable for causal network inference using the EDM framework and thus demonstrated a proof of principle of computational tractability.

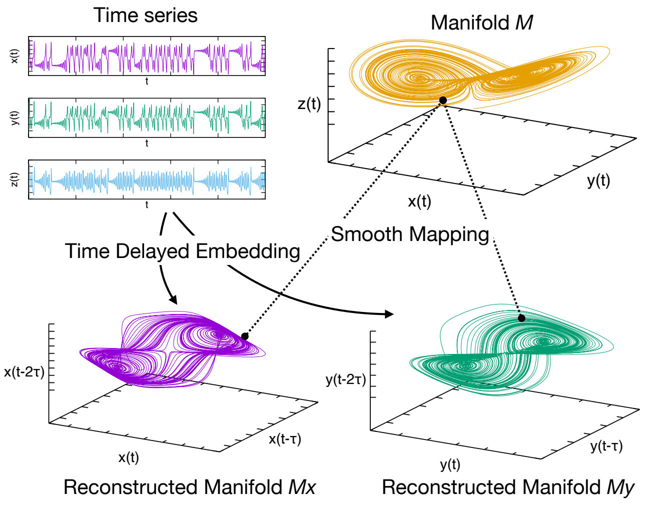


Fig. 1: Basic Idea Behind Empirical Dynamic Modeling

B. *Empirical Dynamic Modeling*

EDM is a mathematical framework designed for studying nonlinear dynamical systems. EDM is based upon the concept of state space reconstruction (SSR) [9]. Takens' theorem states that the attractor manifold of a multivariate dynamical system can be reconstructed from time lagged coordinates of a single time series variable [10]. Figure 1 illustrates the concept of state spaces reconstruction. In this example, three causally related time series variables $x(t)$, $y(t)$ and $z(t)$ that constitute a dynamical system form an attractor manifold M in the state space. A shadow manifold M_x can be reconstructed using the time delayed embeddings of x $(x(t), x(t-\tau), x(t-2\tau), \ldots)$, where τ denotes the time lag. In the same manner, lags of y form a shadow manifold M_y. Takens' theorem states that the reconstructed manifolds M_x and M_y preserve essential mathematical properties (such as the topology) of the true manifold M. In particular, there exist *smooth* mappings between M, M_x, and M_y, suggesting that neighbors in M_x are neighbors in M_y as well.

Simplex projection is a nonlinear forecasting algorithm often used for estimating the dimensionality of a dynamical system. In simplex projection, the input time series is split into two halves: library x and target y. Both halves are embedded into E-dimensional state space by using delayed embeddings. Given a point $\boldsymbol{y}(t_p) = (y(t_p), y(t_p - 1), \dots, y(t_p - E + 1))$ in the target state space, its $E + 1$ nearest neighbors (*i.e.* vertices of the simplex enclosing $\boldsymbol{y}(t_p)$) are searched from the embedded library. Suppose those neighbors are $\boldsymbol{x_1}(t_1), \boldsymbol{x_2}(t_2), \dots, \boldsymbol{x}_{E+1}(t_{E+1})$. A forecast $\boldsymbol{y}(t_p + 1)$ can be made by averaging the future of the neighbors in the library: $\boldsymbol{x_1}(t_1 + 1), \boldsymbol{x_2}(t_2 + 1), \dots, \boldsymbol{x}_{E+1}(t_{E+1} + 1)$. This prediction is performed for every point in $\boldsymbol{y}$ and the results are compared with the true y to evaluate the prediction accuracy. This entire procedure is repeated for different E values and the E that achieves the highest prediction accuracy is determined as the optimal embedding dimension of the dynamical system.

CCM determines the existence and strength of causality between two time series variables [11]. It works similar to simplex projection, but instead of predicting within a single time series, CCM predicts one time series from another. If y can be predicted from x with significant accuracy, we conclude that *y CCM causes x*.

There have been extensive studies on causal inference. Structural Causal Model (SCM) is one of the most popular causal models [12] based on statistical modeling of equilibrium systems. In contrast to SCM, EDM is based on the principle of state-space reconstruction shown in Takens' theorem of non-equilibrium systems. Granger causality is another causal inference technique based on statistical modeling [13]. Granger causality however as stated by Granger himself, only works with linear and stochastic systems and cannot be applied to a nonlinear dynamical system. Compared to these alternatives, EDM is better suited to find the causal relationships in a nonlinear dynamical system such as the brain. Tajima *et al.* [14] also applied embedding theorems in nonlinear state-space reconstruction to analyze a dynamic system. They also built on the causality inference method from Sugihara *et al.* [3] in their work.

EDM has been successfully applied to diverse research fields [15]. In neuroscience, CCM was applied to identify the effective connectivity between brain areas from magnetoencephalography (MEG) data [16]. In ecology, Grziwotz *et al.* found the causal relationships between the environment and mosquito abundance by using CCM [17]. Environmental factors, such as temperature, precipitation, dew point, air pressure, and mean tide level were identified to causally affect mosquito abundance. Ma *et al.* applied simplex projection to forecast wind generation [18]. In [19], an EDM algorithm called S-Map [20] was used to find the relationship between harvested and unharvested fish in terms of size, age, and others. Luo *et al.* applied CCM to estimate the causal relationships of user behavior in an online social network [21]. These use cases demonstrate the wide applicability of EDM to analyze nonlinear dynamical systems.

C. cppEDM

cppEDM [22] is the latest implementation of the EDM framework. cppEDM is a general purpose C++ library used as a backend by rEDM [23] and pyEDM [24], which are EDM implementations for the R and Python language, respectively.

We have identified two major issues in cppEDM that hinder large-scale analysis on HPC systems: redundant computation and lack of GPU support. Since cppEDM is a general purpose library, it provides a one-to-one cross mapping function to identify the causality between a selected combinations of time series variables. The all-to-all cross mapping function is implemented by reusing the one-to-one cross mapping function. This results in redundant computation. Additionally, cppEDM is a reference implementation of EDM; therefore, it is not optimized for a specific hardware architecture such as GPUs. Furthermore, cppEDM suffers from significant load imbalance among workers because it performs static decomposition of the problem. In fact, a performance evaluation in a previous work showed that the runtime of workers varied greatly from 5 hours to 8.5 hours [4].

III. mpEDM

In this section, we first outline the original causal inference algorithm in cppEDM. Then, we describe the algorithmic improvement and the design of the inter-node and intra-node parallelization in mpEDM.

A. Original Algorithm

Algorithm 1 outlines the causal inference algorithm in cppEDM. The input to the algorithm is an $L \times N$ array ts, where L is the number of time steps within a time series and N is the number of time series. In addition to the input dataset, maximum embedding dimension E_{max} and time lag τ need to be supplied. The output is an $N \times N$ casual map ρ. The algorithm consists of two phases: (1) simplex projection and (2) CCM. Simplex projection finds the optimal embedding dimension for each time series. CCM estimates the causal relationship between two time series using the optimal embedded dimension obtained in the first phase. Note that in the original definition of CCM, predictions are made multiple times using randomly subsampled library sets of different sizes and it is tested whether increasing the library set size improves the prediction accuracy. In this research, we excluded this step since the convergence test passes in most cases if the prediction using the full library set achieves high accuracy.

In the first phase, simplex projection (line 1–11) takes a time series in the dataset and splits into into $library$, the first half, and $target$, the second half (line 3–4). Next, both library and target are embedded into E-dimensional space using time delayed embeddings. A k-nearest neighbors (kNN) search is performed in the state space to find the $E + 1$ nearest target points from each library point (line 5). The search results are stored in two lookup tables $indices$ and $distances$, both of which are two-dimensional arrays of shape $L \times (E + 1)$. Element (i, j) in the indices array is the index of the j-th nearest target point from library point i, whereas element (i, j)

Algorithm 1: Causal Inference in cppEDM

```
Input: Dataset $ts$ ($N$ time series of length $L$),
       maximum embedding dimension $E_{max}$
Output: $N \times N$ causal map $\rho$
// Phase 1: Simplex projection
1  for $i \leftarrow 1$ to $N$ do
2      for $E \leftarrow 1$ to $E_{max}$ do
3          $library \leftarrow$ First half of $ts[i]$
4          $target \leftarrow$ Second half of $ts[i]$
5          $indices, distances \leftarrow$ kNN$(library, target, E)$
6          $distances \leftarrow$normalize$(distances)$
7          $prediction \leftarrow$
             lookup$(indices, distances, library, E)$
8          $\rho[E] \leftarrow$ corrcoef$(target, prediction)$
9      end
10     $optE[i] \leftarrow \underset{E}{\text{argmax}}\ \rho[E]$
11 end
// Phase 2: CCM
12 for $i \leftarrow 1$ to $N$ do
13     for $j \leftarrow 1$ to $N$ do
14         $indices, distances \leftarrow$
             kNN$(ts[i], ts[i], optE[j])$
15         $distances \leftarrow$normalize$(distances)$
16         $prediction \leftarrow$
             lookup$(indices, distances, ts[j], optE[j])$
17         $\rho[i, j] \leftarrow$ corrcoef$(ts[j], prediction)$
18     end
19 end
```

in the $distances$ array is the Euclidean distance between the library point i and its j-th nearest target point. The $distances$ array is then converted to exponential scale and each row is normalized (line 6). A one step ahead prediction of a target point is made by (1) obtaining the indices of its $E+1$ library neighbors from $indices$, (2) obtaining the one step ahead values of those library points from $library$ and (3) computing a weighted average of the future library points using $distances$ (line 7). Finally, Pearson's correlation coefficient is computed to evaluate the predictive skill of the simplex projection using the prediction results and real observed withheld values (line 8). This is repeated for every E ranging from 1 to E_{max} ($\leq$20 in practice). The E value that achieves the highest accuracy is determined to be the optimal embedding dimension for the time series and stored in $optE$ (line 10).

In the second phase, CCM (line 12–19) works similar to simplex projection but predicts between two different time series. A given $library$ time series is used to cross predict another $target$ time series in the dataset to evaluate whether the latter is the cause of the former. It computes and normalizes the kNN tables from the library time series (line 14–15) and uses the tables to predict the target time series (line 16). Note that simplex projection predicts within the same time series while CCM predicts across two different time series. Therefore, the kNN tables computed in the simplex projection phase cannot be reused in the CCM phase. The correlation between the predicted values and the actual values represents strength of causality (line 17). In this manner, causal inference is performed for all combinations of time series in the dataset.

We have profiled cppEDM and found out that over 97% of the total runtime is spent in the kNN search. In addition, we have discovered that the time delayed embedding in cppEDM replicates the time series $E+1$ times and causes significant memory overhead.

B. Improved Algorithm

The key observation behind our algorithmic improvement is that the kNN lookup table for CCM is constructed from the library time series only, and the target time series is not used. This suggests that once the kNN lookup table is computed for a particular library time series, we can reuse the precomputed table to make predictions for every target time series. This improvement is trivial if N is in the same order as E_{max}, which was the case in previous use cases of EDM. However, in our use case N is equal to the number of active neurons in a zebrafish brain, which is roughly 10^5. Therefore, the potential speedup becomes significantly large.

Algorithm 2 shows the pseudocode of the improved causal inference algorithm in mpEDM. The simplex projection algorithm is unchanged from cppEDM but its kNN and lookup functions are parallelized and optimized. The CCM algorithm in mpEDM is improved in the following manner. For each library time series, we first compute the kNN lookup tables for every embedding dimension ranging from 1 to E_{max} (line 4–7). Then, we iterate through all $target$ time series and use the precomputed lookup table for the optimal embedding dimension of the $target$ time series to predict the $target$ time series (line 9–10). Finally, we compute the correlation between the prediction and the actual $target$ to estimate the causality (line 11).

Algorithms 3 outlines the kNN function for CPU. We first calculate the all-to-all distances between every library and target point in the state space. Note that we do not explicitly create the time series embeddings on memory but we compute them on-the-fly to reduce memory footprint and increase cache hit. In addition, both $indices$ and $distances$ are stored in row-major format to match the access pattern. Then, each row in the $distances$ and $indices$ arrays is partially sorted in descending order using the distances as sort keys. We use heap sort to implement partial sort. After the sorting, both arrays are trimmed from $L \times L$ to $L \times (E+1)$ and returned. Algorithm 4 shows the kNN function for GPU. In the GPU version, we create time series embeddings on the host and transfer them to the device. The kNN search is executed on the GPU and the resulting kNN tables are returned to the host.

Algorithm 5 outlines the lookup function. It uses the kNN lookup tables $indices$ and $distances$ of the $library$ time series. For each target point, the indices of its $E+1$ neighbors are retrieved from the $indices$ table. Then, those neighbors are

Algorithm 2: Causal Inference in mpEDM

```
Input: Dataset $ts$ ($N$ time series of length $L$),
       maximum embedding dimension $E_{max}$
Output: $N \times N$ causal map $\rho$
// Phase 1: Simplex projection
1  for $i \leftarrow 1$ to $N$ do
       // Same as cppEDM (Algorithm 1)
2  end
// Phase 2: CCM
3  for $i \leftarrow 1$ to $N$ do
4      for $E \leftarrow 1$ to $E_{max}$ do
5          $indices[E], distances[E] \leftarrow$
             kNN($ts[i], ts[i], E$)
6          $distances \leftarrow$normalize($distances$)
7      end
8      for $j \leftarrow 1$ to $N$ do
9          $E_j \leftarrow optE[j]$
10         $prediction \leftarrow$
             lookup($indices[E_j], distances[E_j], ts[j], E_j$)
11         $\rho[i, j] \leftarrow$ corrcoef($ts[j], prediction$)
12     end
13 end
```

accumulated using the weights stored in the *distances* table. Finally, the function returns the predicted *target* time series.

The average time complexity of each algorithm is analyzed as follows. The time complexity of the kNN function in Algorithm 3 and 4 is $O(L^2E)$ because the all-to-all distance calculation is $O(L^2E)$ and the sorting is approximately $O(L^2 \log E)$. The time complexity of the lookup function in Algorithm 5 is $O(LE)$. By combining these results, the time complexity of simplex projection in mpEDM is $O(NL^2E)$, which is the same as cppEDM. The time complexity of CCM in mpEDM, on the other hand, is $O(NL^2E^2 + N^2LE)$. In cppEDM, the time complexity of CCM is $O(N^2L^2E)$. As a result, the time complexity of the whole causal inference algorithm in mpEDM is $O(NL^2E^2 + N^2LE)$.

C. Inter-Node Parallelism

To distribute the work across multiple compute nodes, we naturally choose the loops with the highest granularity. That is, the two outermost loops that iterate over the time series (line 1–2 and 3–13 in Algorithm 2). We implement a simple master-worker framework based on MPI to distribute these loops. To dynamically distribute work and mitigate load imbalance among workers, we adopt *self-scheduling* in our master-worker framework. In self-scheduling, the master accounts and dispatches tasks to workers. Each worker performs assigned tasks, and once it completes, the worker asks the master for a new task.

The high-level organization of the inter-node parallelism is as follows. First, the workers execute the embedding dimension phase. The optimal embedding dimension for each time series is reported back to the master. Once the first phase is complete, the master broadcasts *optE* to all workers. Subsequently, the workers execute the all-to-all CCM phase. The final results are written to the file system by each worker to alleviate the load on the master.

Both the input dataset and the inferred causal map are stored as HDF5 [25] files for easy integration with the pre/post processing workflow. The workers read the input HDF5 file in parallel and keep the entire dataset on memory during the execution. Every time a worker completes a cross map, the worker writes an element of the causal map asynchronously to the output HDF5 file. This small random write pattern, however, is known to be slow on parallel file systems. In fact,

Algorithm 3: kNN for CPU

```
Input: $library$ and $target$ time series, embedding
       dimension $E$, time lag $\tau$
Output: Arrays $diatances$ and $indices$ for lookup
// All-to-all distance calculation
1  for $i \leftarrow 1$ to $L$ do
2      for $k \leftarrow 1$ to $E$ do
3          $distances[i, :] \leftarrow 0$
4          for $j \leftarrow 1$ to $L$ do
5              $indices[i, j] \leftarrow j$
6              $distances[i, j] \leftarrow distances[i, j] +$
                 $(target[k\tau+i] - library[k\tau+j])^2$
7          end
8      end
9  end
// Sorting
10 $top\_k \leftarrow$ E+1
11 for $i \leftarrow 1$ to $L$ do
12     $indices[i, :] \leftarrow$
         partialSort($indices, distances, top\_k$)
13 end
```

Algorithm 4: kNN for GPU

```
Input: $library$ and $target$ time series, embedding
       dimension $E$, time lag $\tau$
Output: Arrays $diatances$ and $indices$ for lookup
// Embedding
1  for $i \leftarrow 1$ to $E$ do
2      for $j \leftarrow 1$ to $L$ do
3          $libraryBlock[i, j] \leftarrow library[i\tau+j]$
4          $targetBlock[i, j] \leftarrow target[i\tau+j]$
5      end
6  end
// All-to-all distance calculation
   and sorting
7  $top\_k \leftarrow$ E+1
8  Copy $libraryBlock$ and $targetBlock$ to device
9  $indices, distances \leftarrow$
     nearestNeighbour($libraryBlock, targetBlock, top\_k$)
10 Copy $indices$ and $distances$ to host
```

Algorithm 5: Lookup

```
Input: Array of $indices$ and $distances$, $target$ time
       series, embedding dimension $E$ of $target$
Output: Prediction of the time series $prediction$
1 for $i \leftarrow 1$ to $L$ do
2     $prediction[i] \leftarrow 0$
3     for $j \leftarrow 1$ to $E + 1$ do
4         $idx \leftarrow indices[i, j]$
5         $dist \leftarrow distances[i, j]$
6         $prediction[i] \leftarrow$
            $prediction[i] + target[idx] \cdot dist$
7     end
8 end
```

we observed that write I/O becomes a significant bottleneck of the application on GPFS. We therefore take advantage of BeeOND (BeeGFS On Demand) [26], the burst buffer deployed on ABCI. BeeOND combines local SSDs installed on the compute nodes and provides an on-demand parallel file system to a job. The workers write the results to BeeOND to minimize I/O overhead.

D. Intra-Node Parallelism

We focus our efforts to parallelize and optimize the kNN kernel since it is the primary bottleneck in cppEDM as discussed in section III-A. We design and implement kNN kernels for both CPU and GPU architecture to ensure that mpEDM can efficiently run on a wide variety of computing platforms. In the kNN kernel for CPU shown in Algorithm 3, the two loops that iterate over the time steps within a time series are parallelized using OpenMP (line 1–9 and 10–13 in Algorithm 3). We also utilize OpenMP 4.0 SIMD directives to vectorize the innermost loop explicitly. Note that the nested loops are ordered such that the memory accesses in the innermost loop are contiguous.

In the kNN kernel for GPU shown in Algorithm 4, we take advantage of ArrayFire [27], a highly optimized library for GPU-accelerated computing. ArrayFire provides backends for CUDA, OpenCL and CPU, but in this paper we only use the CUDA backend since ABCI is installed with Tesla V100 GPUs. The kNN algorithm implemented in ArrayFire is essentially the same as our CPU implementation. ArrayFire uses a block-wide parallel radix sort implementation in the CUDA UnBound (CUB) template library. Since each ABCI compute node is equipped with four GPUs, we also distribute the work across multiple GPUs. To achieve this, the loop that iterates over E (line 4–7 in Algorithm 2) is parallelized such that each GPU computes lookup tables for one or more E. We dynamically schedule this loop to ensure load balancing across GPUs because the runtime of the kNN kernel depends on E as discussed in section III-B.

For the lookup kernel shown in Algorithm 5, we currently only have a CPU version of this kernel. The time step loop is parallelized using OpenMP (line 1–8 in Algorithm 5). This kernel is heavily memory bandwidth bound since it requires random memory access.

IV. Evaluation

The computational performance of mpEDM was evaluated on ABCI. Furthermore, we present the scientific outcomes obtained using mpEDM.

A. Evaluation Environment

ABCI [28] is the world's first large-scale Open AI Computing Infrastructure, which is constructed and operated by the National Institute of Advanced Industrial Science and Technology (AIST). According to the latest TOP500 list published in November 2019 [29], ABCI is the most powerful supercomputer in Japan and the 8th in the world. ABCI has 1,088 compute nodes, each equipped with two 20-core Intel Xeon Gold 6148 CPUs, four NVIDIA Tesla V100 SXM2 (16GB) GPUs, 384GB of RAM and 1.6TB of local NVMe SSD. The parallel file system is based on GPFS with a total capacity of 22PB.

B. Performance Evaluation

We compared mpEDM with cppEDM from the following three aspects: total runtime, parallel scalability and impact of dataset size on the runtime. We used three real-world datasets recorded from larval zebrafish under different conditions. Table I shows the list of datasets used in the evaluation.

TABLE I: Datasets used in the evaluation

Dataset	# of Time Steps	# of Time Series	Size
Fish1_Normo	1,450	53,053	0.7 GB
Subject6	3,780	92,538	3.0 GB
Subject11	8,528	101,729	9.5 GB

1) Total Runtime: mpEDM shows significantly higher performance compared to cppEDM. Table II shows the performance comparison between cppEDM and mpEDM. cppEDM took 8.5 hours to analyze the Fish1_Normo dataset using 512 ABCI nodes [4], whereas mpEDM took only 20 seconds to analyze the same dataset using 512 ABCI nodes with GPU architecture. The result shows that mpEDM is 1,530× faster than cppEDM. Moreover, mpEDM finished the causal inference of two larger datasets: Subject6 in 101 seconds and Subject11 [6] in 199 seconds.

TABLE II: Performance comparison between cppEDM and mpEDM

	cppEDM	mpEDM	
Dataset	512 Nodes	1 Node	512 Nodes
Fish1_Normo	8.5h	1,973s	20s
Subject6	N/A	13,953s	101s
Subject11	N/A	39,572s	199s

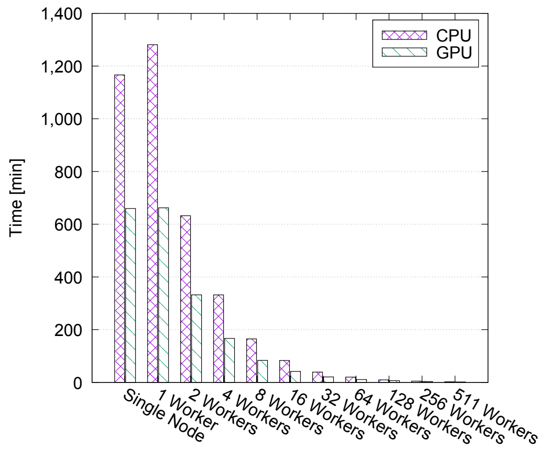


Fig. 2: Strong scaling performance (absolute runtime)

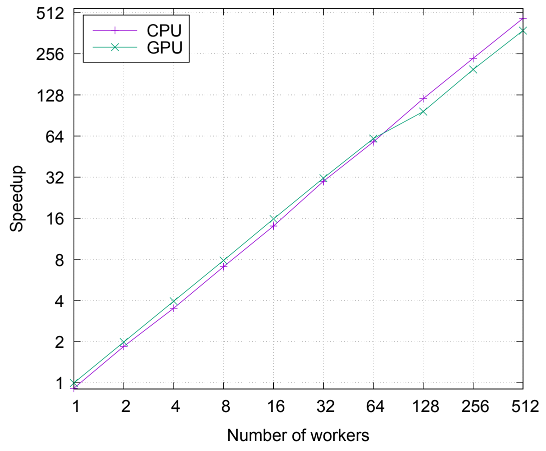


Fig. 3: Strong scaling performance (relative speedup)

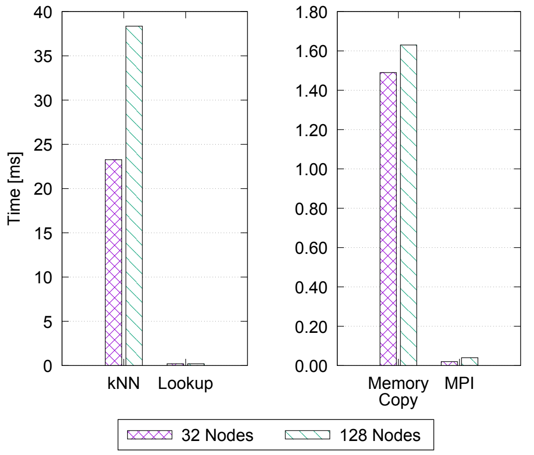


Fig. 4: Breakdown of simplex projection (average runtime per time series)

2) Parallel Scalability: We measured the parallel scalability of mpEDM by varying the number of workers and measuring the runtime of mpEDM with and without GPU. We used the largest Subject11 dataset in this evaluation.

Figure 2 shows the strong scaling performance of mpEDM. In the *Single Node* setup, mpEDM is executed on a single node without MPI. In the *X Workers* setup, mpEDM is executed with MPI using the specified number of workers. We measured up to 511 workers since ABCI allows a maximum of 512 nodes per job (except for jobs running under the ABCI grand challenge program, which can use the full 1,088 nodes). The result shows that the GPU version runs as twice as fast as the CPU version in every case. We noticed that the CPU version ran in the single worker setup 10% slower than the single node setup. We believe this slowdown is caused from the interference between the background tasks performed by the BeeOND daemon and the computation in mpEDM. This does not happen with the GPU version because the average CPU utilization is lower than the CPU version.

Figure 3 shows the relative speedup of the multi-node setup in relation to the single node setup. It reveals that the speedup is nearly linear with both GPU and CPU. However, the speedup of the GPU version drops when the number of nodes is 64 or more.

We measured the breakdown of each phase to investigate the cause behind the scalability decline. We compared 32 workers and 128 workers since the GPU version declines beyond 64 nodes. Figures 4 and 5 show the breakdown of average runtime for processing a single time series in simplex projection and CCM. The two figures clearly indicate that memory copy, MPI communication and I/O are not bottlenecks and do not significantly increase with the number of workers. However, the kNN function becomes slower when the number of workers increases. We found out that the kNN search for the first time series processed on a worker is significantly slower (ranging from 3.3 seconds to 16.4 seconds) than the subsequent ones. We believe this is caused by the initialization process of the GPUs.

To verify this, we created a simple program that initializes the GPUs and allocates some GPU memory on a single node. We submitted a job that run this program 100 times and measured the initialization time. The result revealed that the initialization time follows a long-tailed distribution: the median was 4.6 seconds while the maximum was 22.9 seconds. This suggests that a few stragglers impact the total runtime and degrade the scalability as the number of workers increases.

3) Impact of Dataset Size: We evaluated how the size of the dataset impacts the runtime of mpEDM using dummy datasets with different sizes. Furthermore, we measured the time spent in each function. We also measured the speedup of the GPU version over the CPU version with varying number of time steps.

Figures 6 and 7 show the runtime of mpEDM when increasing the number of time series and time steps, respectively. We confirmed that the increase of runtime is not bigger than the increase predicted from the time complexity. We also confirmed that CCM consumed the majority of the total runtime and other tasks including I/O and MPI communication

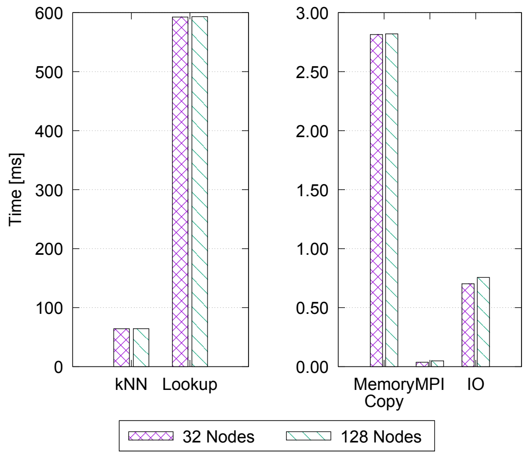


Fig. 5: Breakdown of cross mapping (average runtime per time series)

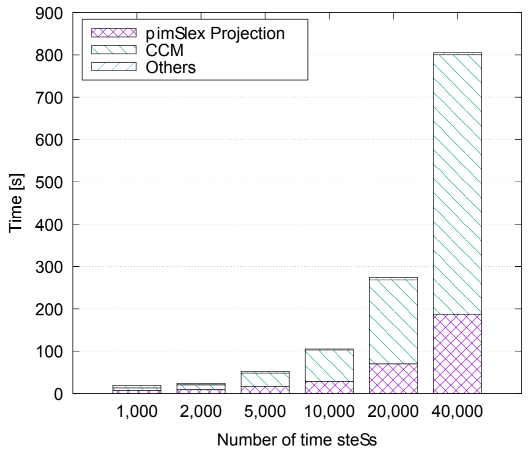


Fig. 7: Runtime with varying number of time steps (1,000 time series)

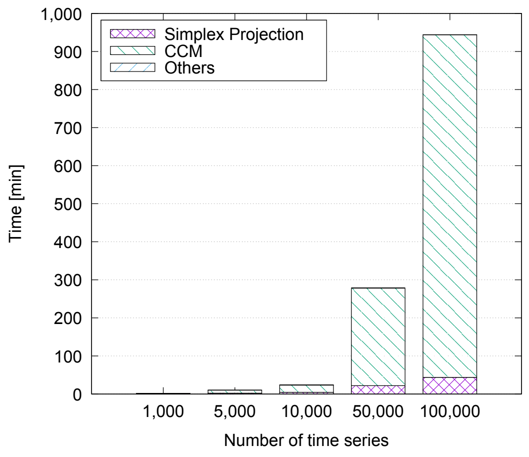
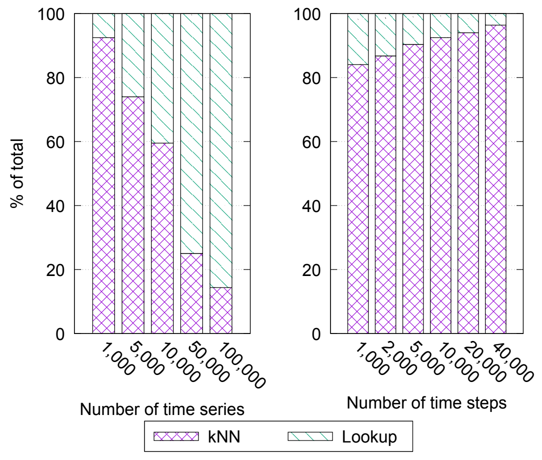


Fig. 6: Runtime with varying number of time series (10,000 time steps)



(a) Varying number of time series (10,000 time steps)

(b) Varying number of time steps (1,000 time series)

Fig. 8: Breakdown of CCM

are ignorable.

Figures 8a and 8b show the runtime breakdown of each function in CCM when increasing the number of time series and time steps, respectively. Figure 8a shows that the runtime of the lookup function becomes dominant when increasing the number of time series. On the other hand, Fig. 8b shows that the runtime of the kNN function becomes dominant when increasing the number of time steps. These trends can be explained from the time complexity analysis of each algorithm described in section III-B.

Figure 9 shows the speedup of the GPU version over the CPU version when varying number of time steps. We compared the performance between a single CPU socket and one or more GPUs to evaluate the GPU speedup. Evidently, the GPU speedup increases with the number of time steps. Single GPU is slower than the CPU if the number of time steps is 2,000 or less. This is because of the overhead inherent to offloading computation to the GPU. However, single GPU consistently surpasses the CPU if the number of time steps

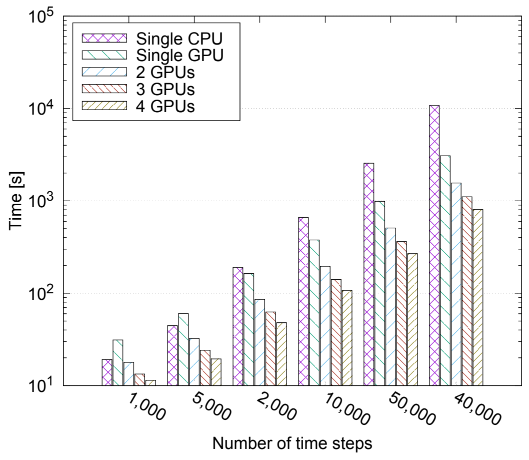


Fig. 9: GPU speedup with varying number of time steps (1,000 time series)

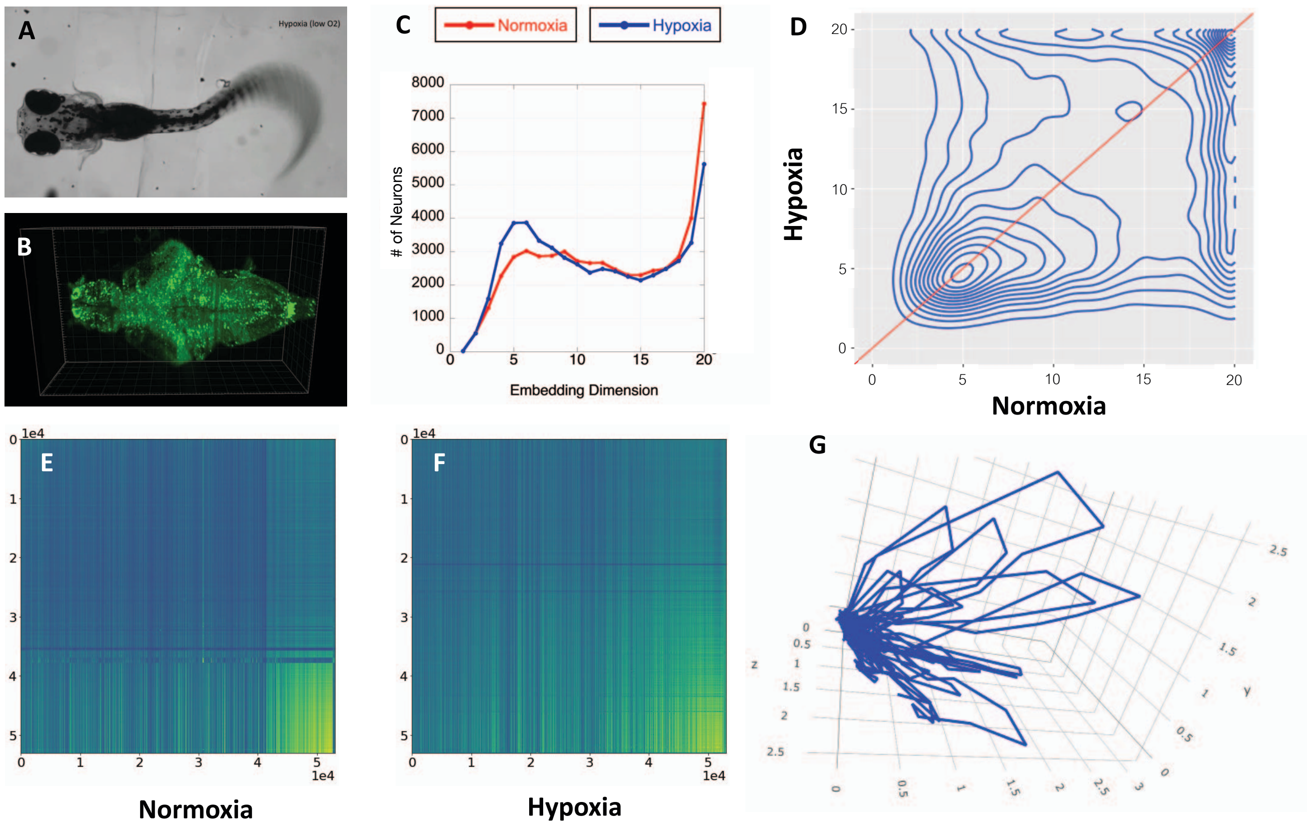


Fig. 10: Scientific results. (A) Zebrafish larvae were imaged to study their response to low oxygen (B) The day larvae were imaged using a SPIM lightsheet microscope and whole brain calcium activity was recorded at single cell resolution (C) Our calculation of the dimensionality of the neuronal populations show a decrease under low oxygen (hypoxia) as seen in the distribution. (D) Measured transitions between normal oxygen concentrations (normoxia) to hypoxia show a bias below to the right of the diagonal line showing that dimensionality decreases as oxygen decreases. (E, F) Whole brain CCM all vs all causal inference matrix of an all vs all neurons. Results show a more homogeneous map in hypoxia (F) than normoxia (E) indicating a simplification of behavior consistent with the above dimensionality drop. (G) An identified signal integration manifold capable of predicting turns of the fish at least 0.5 seconds (a single time step) ahead of time. Whenever the neural activity trajectory enters one of the loops of the manifold, the fish will turn.

if 5,000 or more. If the number of time steps is 40,000, the speedup of a single GPU is 3.5 times compared to CPU. When four GPUs are used, the speed up is 13.4 times.

C. *Scientific Outcomes*

Figure 10 shows the scientific outcomes obtained using mpEDM. Our results showed that we could determine the causal connectivity across the entire brain across two behaviors. This shows that depending on task, the network of relationships between individual neurons change and become more connected, homogeneous and simplified with a goal directed task. In the resulting network connectivity increased and became simpler. Furthermore, we were able identify individual neurons that integrate signals from multiple other neurons that contain decision making information. These neurons allow the prediction of fish turn behaviors while swimming and generate low dimensional manifold models based on data geometry that are able to predict the fish's behavior at least 0.5 seconds (a single time step) ahead. A three dimensional projection of one of these manifolds is shown in Fig. 10 (G), where entering the loop predicts turn behavior. Based on the combined activity of two neurons and information on prior states we are able to predict when the fish will turn. Beyond this, this is the first map of causal connectivity of any vertebrate animal at single neuron resolution.

V. Conclusion & Future Work

EDM is a nonlinear time series analysis framework proven its applicability in various fields. However, EDM has only been applied to small datasets due to its computational cost. In this paper, we designed and implemented mpEDM, a parallel distributed implementation of EDM optimized for execution on modern GPU-centric supercomputers. mpEDM improves the EDM algorithm to reduce redundant computation and optimizes the implementation to fully utilize hardware resources such as GPUs and SIMD units. mpEDM took only

20 seconds to finish the causal inference of a dataset containing the activity of 53,053 zebrafish neurons on 512 ABCI nodes. This is 1,530× faster than cppEDM, the current standard implementation of EDM. Moreover, mpEDM could analyze a 13× larger dataset in 199 seconds. This is the largest EDM causal inference achieved to date.

We will continue to optimize the performance of mpEDM. As discussed in section IV-B, we need to improve the performance of the lookup as it becomes the primary bottleneck when we scale up the number of time series further. We will also explore other efficient implementations of nearest neighbor search on GPUs. Currently, mpEDM uses the exact kNN search implementation provided by ArrayFire. There exist many studies on efficient Approximate Nearest Neighbor (ANN) search [30], [31]. However, it is unclear how ANN affects the accuracy of EDM predictions. Another well-known approach is to use spatial indices such as KD-trees and Ball-trees to accelerate kNN search [32], [33].

Additionally, EDM algorithms other than simplex projection and CCM will be implemented in mpEDM to expand mpEDM to a standard implementation of EDM on HPC systems. We will make this EDM library widely available to the community with a hope to assist scientists in need to analyze large-scale time series datasets of nonlinear dynamical systems.

Acknowledgment

This work was supported by JSPS KAKENHI Grant Number JP20K19808 (KT) and an Innovation grant by the Kavli Institute for Brain and Mind (GMP). Computational resources of the AI Bridging Cloud Infrastructure (ABCI) were provided by the National Institute of Advanced Industrial Science and Technology (AIST).

References

[1] R. Gămănuţ, H. Kennedy, Z. Toroczkai, M. Ercsey-Ravasz, D. C. Van Essen, K. Knoblauch, and A. Burkhalter, "The mouse cortical connectome, characterized by an ultra-dense cortical graph, maintains specificity by distinct connectivity profiles," *Neuron*, vol. 97, no. 3, pp. 698–715, 2018.

[2] E. R. Deyle and G. Sugihara, "Generalized theorems for nonlinear state space reconstruction," *PLoS One*, vol. 6, no. 3, 2011.

[3] G. Sugihara, R. May, H. Ye, C.-h. Hsieh, E. Deyle, M. Fogarty, and S. Munch, "Detecting causality in complex ecosystems," *Science*, vol. 338, no. 6106, pp. 496–500, 2012.

[4] J. Park, G. M. Pao, E. Saberski, C. Smith, J. Haga, R. Takano, C. M. Yeh, S. Chalasani, and G. Sugihara, "Massively parallel empirical dynamic cross mapping," in *37th Meeting of the Pacific Rim Applications and Grid Middleware Assembly (PRAGMA37)*, 2019.

[5] M. B. Ahrens, M. B. Orger, D. N. Robson, J. M. Li, and P. J. Keller, "Whole-brain functional imaging at cellular resolution using light-sheet microscopy," *Nature Methods*, vol. 10, no. 5, p. 413, 2013.

[6] X. Chen, Y. Mu, Y. Hu, A. T. Kuan, M. Nikitchenko, O. Randlett, A. B. Chen, J. P. Gavornik, H. Sompolinsky, F. Engert, *et al.*, "Brain-wide organization of neuronal activity and convergent sensorimotor transformations in larval zebrafish," *Neuron*, vol. 100, no. 4, pp. 876–890, 2018.

[7] A. T. Clark, H. Ye, F. Isbell, E. R. Deyle, J. Cowles, G. D. Tilman, and G. Sugihara, "Spatial convergent cross mapping to detect causal relationships from short time series," *Ecology*, vol. 96, no. 5, pp. 1174–1181, 2015.

[8] H. Ye, E. R. Deyle, L. J. Gilarranz, and G. Sugihara, "Distinguishing time-delayed causal interactions using convergent cross mapping," *Scientific Reports*, vol. 5, no. 14750, pp. 1–9, 2015.

[9] G. Sugihara and R. M. May, "Nonlinear forecasting as a way of distinguishing chaos from measurement error in time series," *Nature*, vol. 344, no. 6268, pp. 734–741, 1990.

[10] F. Takens, "Detecting strange attractors in turbulence," in *Dynamical systems and turbulence, Warwick 1980*, pp. 366–381, Springer, 1981.

[11] K. Schiecke, B. Pester, M. Feucht, L. Leistritz, and H. Witte, "Convergent cross mapping: Basic concept, influence of estimation parameters and practical application," in *37th Annual International Conference of the Engineering in Medicine and Biology Society (EMBC)*, pp. 7418–7421, 2015.

[12] J. Pearl, "Causal inference in statistics: An overview," *Statistics surveys*, vol. 3, pp. 96–146, 2009.

[13] C. W. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica: journal of the Econometric Society*, pp. 424–438, 1969.

[14] S. Tajima, T. Yanagawa, N. Fujii, and T. Toyoizumi, "Untangling brain-wide dynamics in consciousness by cross-embedding," *PLoS computational biology*, vol. 11, no. 11, 2015.

[15] C.-W. Chang, M. Ushio, and C.-h. Hsieh, "Empirical dynamic modeling for beginners," *Ecological Research*, vol. 32, no. 6, pp. 785–796, 2017.

[16] H. Natsukawa and K. Koyamada, "Visual analytics of brain effective connectivity using convergent cross mapping," in *SIGGRAPH Asia 2017 Symposium on Visualization*, pp. 1–9, 2017.

[17] F. Grziwotz, J. F. Strauß, C.-h. Hsieh, and A. Telschow, "Empirical dynamic modelling identifies different responses of Aedes Polynesiensis Subpopulations to Natural Environmental Variables," *Scientific Reports*, vol. 8, no. 1, pp. 1–10, 2018.

[18] J. Ma, M. Yang, X. Han, and Z. Li, "Ultra-short-term wind generation forecast based on multivariate empirical dynamic modeling," *IEEE Transactions on Industry Applications*, vol. 54, no. 2, pp. 1029–1038, 2017.

[19] C. N. Anderson, C.-h. Hsieh, S. A. Sandin, R. Hewitt, A. Hollowed, J. Beddington, R. M. May, and G. Sugihara, "Why fishing magnifies fluctuations in fish abundance," *Nature*, vol. 452, no. 7189, pp. 835–839, 2008.

[20] G. Sugihara, "Nonlinear forecasting for the classification of natural time series," *Philosophical Transactions of the Royal Society of London. Series A: Physical and Engineering Sciences*, vol. 348, no. 1688, pp. 477–495, 1994.

[21] C. Luo, X. Zheng, and D. Zeng, "Causal inference in social media using convergent cross mapping," in *2014 Joint Intelligence and Security Informatics Conference*, pp. 260–263, 2014.

[22] "cppEDM library." https://github.com/SugiharaLab/cppEDM.

[23] H. Ye, A. Clark, E. Deyle, and G. Sugihara, "rEDM: an R package for empirical dynamic modeling and convergent cross-mapping." https://github.com/SugiharaLab/rEDM, 2016.

[24] "pyEDM library." https://github.com/SugiharaLab/pyEDM.

[25] M. Folk, A. Cheng, and K. Yates, "HDF5: A file format and I/O library for high performance computing applications," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, vol. 99, pp. 5–33, 1999.

[26] "BeeOND: BeeGFS On Demand." https://www.beegfs.io/wiki/BeeOND.

[27] J. Malcolm, P. Yalamanchili, C. McClanahan, V. Venugopalakrishnan, K. Patel, and J. Melonakos, "ArrayFire: a GPU acceleration platform," in *Modeling and Simulation for Defense Systems and Applications VII*, vol. 8403, pp. 49–56, 2012.

[28] "ABCI official website." https://abci.ai/.

[29] "TOP500 list, November 2019." https://top500.org/lists/2019/11/.

[30] W. Chen, J. Chen, F. Zou, Y.-F. Li, P. Lu, and W. Zhao, "RobustiQ: A robust ANN search method for billion-scale similarity search on GPUs," in *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pp. 132–140, 2019.

[31] J. Pan and D. Manocha, "Fast GPU-based locality sensitive hashing for k-nearest neighbor computation," in *Proceedings of the 19th ACM SIGSPATIAL international conference on advances in geographic information systems*, pp. 211–220, 2011.

[32] V. Garcia, E. Debreuve, and M. Barlaud, "Fast k nearest neighbor search using GPU," in *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1–6, 2008.

[33] M. R. Abbasifard, B. Ghahremani, and H. Naderi, "A survey on nearest neighbor search methods," *International Journal of Computer Applications*, vol. 95, no. 25, pp. 39–52, 2014.