Delft University of Technology

HUMA
Heterogeneous, Ultra Low-Latency Model Accelerator for The Virtual Brain on a Versal Adaptive SoC

Movahedin, Amirreza; Landsmeer, Lennart P.L.; Strydis, Christos

DOI
10.1145/3706628.3708875

Licence
CC BY

Publication date
2025

Document Version
Final published version

Published in
FPGA 2025 - Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays

Citation (APA)
Movahedin, A., Landsmeer, L. P. L., & Strydis, C. (2025). HUMA: Heterogeneous, Ultra Low-Latency Model Accelerator for The Virtual Brain on a Versal Adaptive SoC. In *FPGA 2025 - Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (pp. 223-233). (FPGA 2025 - Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays). Association for Computing Machinery (ACM). https://doi.org/10.1145/3706628.3708875

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

HUMA: Heterogeneous, Ultra Low-Latency Model Accelerator for The Virtual Brain on a Versal Adaptive SoC

Amirreza Movahedin
Erasmus Medical Center
Rotterdam, The Netherlands
Delft University of Technology
Delft, The Netherlands

Lennart P. L. Landsmeer
Delft University of Technology
Delft, The Netherlands
Erasmus Medical Center
Rotterdam, The Netherlands

Christos Strydis
Erasmus Medical Center
Rotterdam, The Netherlands
Delft University of Technology
Delft, The Netherlands

Abstract

Brain modeling can occur at different levels of abstraction, each aimed at a different purpose. The Virtual Brain (TVB) is an open-source platform for constructing and simulating personalized brain-network models, favoring whole-brain macro-scales while reducing micro-level detail. Among other purposes, TVB is used to build patient-specific, digital, brain twins that can be used in different clinical settings, such as the study and treatment of epilepsy. However, fitting patient-specific TVB models requires a large number of successive and time-consuming simulations. By studying the internal structure of TVB, we observed heterogeneous computation needs in its models which could be leveraged to accelerate simulations. In this work, we designed and implemented *HUMA*, a heterogeneous, ultra low-latency, dataflow architecture on an AMD Versal Adaptive SoC to accelerate TVB fitting to different patient-brain makeups. Our heterogeneous solution runs about 27× faster compared to a modern-day, server-class, 32-core CPU while consuming a fraction of its power. Additionally, it delivers on average about 14× lower latency, 1.7× better power efficiency and an order-of-magnitude lower energy consumption when compared against the high-performance GPU version of TVB. The achieved latency savings reveal a significant potential in model-fitting for individual patients as well as in closed-loop biohybrid experiments.

CCS Concepts

• **Hardware → Hardware accelerators**; • **Computer systems organization → Data flow architectures**.

Keywords

The Virtual Brain, Acceleration, Ultra Low-latency, Versal, Heterogeneous

ACM Reference Format:
Amirreza Movahedin, Lennart P. L. Landsmeer, and Christos Strydis. 2025. HUMA: Heterogeneous, Ultra Low-Latency Model Accelerator for The Virtual Brain on a Versal Adaptive SoC. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '25), February 27–March 1, 2025, Monterey, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3706628.3708875

This work is licensed under a Creative Commons Attribution International 4.0 License.

FPGA '25, February 27–March 1, 2025, Monterey, CA, USA
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1396-5/25/02
https://doi.org/10.1145/3706628.3708875

1 Introduction

For centuries, understanding the brain and the way it works has been of great concern to scientists. The National Academy of Engineering of the United States has classified reverse engineering the brain as one of the grand challenges of the 21st century [20]. Due to the tremendous leaps in computing capabilities of late, brain simulation has become increasingly important in neuroscientific research. Brain simulation can be performed at different levels of abstraction for different purposes [21]. Nowadays, coarse-grained brain models are developed with the goal of clinical use in mind.

The Virtual Brain (TVB) is one of the leading tools in building and simulating large-scale brain models [22]. By reducing the complexity at the microscopic level, TVB can deliver an overview of the brain's macro-organization. With this modeling strategy, TVB can produce sufficiently accurate brain signals that can be useful in various clinical applications. TVB divides the entire brain into different regions known as *centers*. Each center models the activity of all the neurons in a given brain region, called a *neural mass*. These centers are typically *sparsely* interconnected, affecting each other's activity in a process called *coupling*. The couplings among centers bear different strengths (represented with a weight value) in addition to delays which are due to the limited speed of signal propagation in the brain.

To make the model more accurate for each patient, TVB encodes personal, brain-imaging data in the described base model. This means that the brain region that each center represents, in addition to the weights and delays associated with the centers' connectivity, are patient-specific. For each patient, the personal brain model must be tuned in a process called *model fitting*. One common fitting method is *Bayesian Inference* [7, 11] which requires a large number of successive simulations. One class of algorithms that can be used within the framework of Bayesian inference is Markov-Chain Monte Carlo (MCMC) [1, 10]. These fitting strategies are used to infer certain model parameters that are of interest within the setting in which TVB is being used [28]. Under MCMC, a large number of simulations is performed one after the other. This means that the output of each simulation has to be used as input to the next simulation.

Since such model-fitting strategies are typically very time-consuming, achieving low-latency execution of TVB models could greatly accelerate this process. Additionally, TVB models can potentially be used in brain-computer interface (BCI) systems [23, 29] or biohybrid experiments [15], which require interaction of biological with *in silico* neurons in their control loop. In this case, a low-latency platform that can iterate quickly over many, individual TVB model simulations would be advantageous.

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

Besides, a key observation is that center and coupling calculations are heterogeneous computations with different memory-access needs. For all aforementioned reasons, in this work, we present *HUMA*, a heterogeneous accelerator of TVB-model simulations implemented on a Versal Adaptive SoC [31]. HUMA delivers sustained lower latency compared to both multi-threaded CPU and high-performance GPU versions of TVB while consuming less power. The main contributions of this work are analysis of the TVB model and its needs in terms of computation and memory capacity, in addition to designing a heterogeneous accelerator for simulating TVB models, achieving on average 27× and 14× lower latency compared to the CPU and GPU versions of TVB, respectively.

The rest of the paper is structured as follows: Section 2 provides necessary background information on the problem at hand. Section 3 gives an overview of related works done regarding TVB acceleration. Section 4 dives into the details of the design and implementation of HUMA for TVB model simulation. Section 5 presents the performance results of HUMA and compares it to the related work. Finally, in Section 6 concluding remarks are presented.

2 Background

In this section, background information of the accelerated model and the heterogeneous-computing platform used is given.

2.1 The Virtual Brain Model

As shown in Figure 1, the TVB pipeline starts with MRI imaging to extract the brain regions in addition to the delay and strength of the connections between them [28]. To add dynamics to the model, neural masses are added to the information extracted from MRI imagings. This leads to the personalized TVB model. In order to make this model more accurate and patient-specific, it needs to be fitted. To perform the model fitting, the personalized TVB model is simulated, the output of the simulation is compared to the EEG signals taken from the patient, and the model parameters are updated accordingly. This process repeats until the model produces accurate enough outputs. Eventually, this fitted model can be used in different applications such as virtual simulation, virtual surgery [28], or real-time closed-loop systems [23, 29].

As mentioned earlier, TVB divides the brain into different regions called centers. These centers are interconnected via different patterns, with each connectivity (or coupling) having a certain strength (or weight) and delay associated with it. Each center contains internal neural activity called *local dynamics*, which comprise the neural-mass model of that center. In addition to local dynamics, each center is also affected by other centers which is referred to as the *coupling input* to the center. If we assume there are N centers in the model, the activity of each center $i \in [1, N]$ over time is described using a differential equation shown in (1).

$$\frac{d\vec{X}_i(t)}{dt} = F\left(\vec{X}_i(t),\ C_i(t)\right) + u(t) + \eta(t) \quad (1)$$

where $\vec{X}_i \in \mathbb{R}^M$ are the state variables of each center. The activity output of each center is one of these state variables. Function $F : \mathbb{R}^M \rightarrow \mathbb{R}^M$ describes the local dynamics of each center, while $u(t)$ and $\eta(t)$ are the external input to the center and the noise in the system, respectively. Although important for the model's accuracy, these two components are ignored for the rest of this work since they do not affect the design and implementation of the accelerator in a meaningful way.

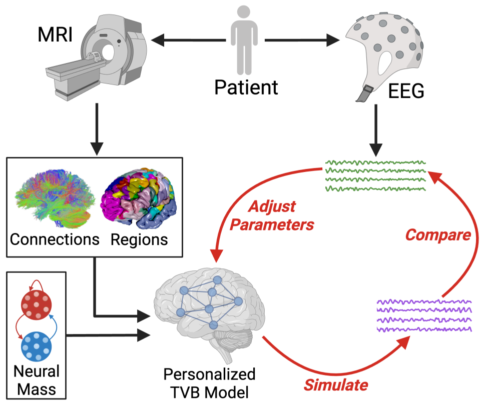


Figure 1: The Virtual Brain (TVB) model fitting pipeline [19]. A low-latency TVB simulation can reduce the fitting process time by speeding up the loop marked with red.

The coupling inputs to each center are represented by $C_i(t)$, which is added to one of the state variables of the center. The $C_i(t)$ for each center $i \in [1, N]$ is calculated as formulated in (2).

$$C_i(t) = K_{post}\left(\sum_{j=1}^{N} W_{ij} K_{pre}\left(X_j(t - d_{ij}), X_i(t)\right)\right) \quad (2)$$

where W_{ij} and d_{ij} represent the weight and delay associated with the connection from center j to center i respectively. Functions $K_{pre} \in (\mathbb{R}, \mathbb{R}) \rightarrow \mathbb{R}$ and $K_{post} \in \mathbb{R} \rightarrow \mathbb{R}$ are pre- and post-synapse functions respectively which scale the signals from other centers to more realistically represent their strength when reaching the receiving center [24]. It is worth noting that the connections between the centers in the model are usually sparse, meaning that a large number of weight values (W_{ij}) in the model are zero. This allows for utilizing sparse-storage and -calculation techniques [8] for the coupling calculation shown in (2).

When we talk about simulating the brain model, we mean solving differential equations (1) for all the centers in the model over a certain amount of timesteps. There are many methods to solve differential equations numerically with different accuracy levels and computational costs. The method we used in HUMA is the *Forward Euler (FE)* which provides the required accuracy with minimal computational cost [14, 24].

Since we want our implementation to be as general as possible and require no re-synthesis when different aspects of the model change, we replace the local-dynamics function F in (1) with a *Multi-Layer Perceptron (MLP)*. MLPs are *universal function approximators* [12], which can be leveraged in this system. By changing the parameters of the MLP, different local-dynamics functions can be realized in the model with no need for re-synthesizing the implementation. Additionally, MLPs that are trained to approximate a certain neural-mass model can be used to better infer unknown parameters that are of interest within the context in which TVB is being used [3] (for example regional excitability in epilepsy [14]).

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

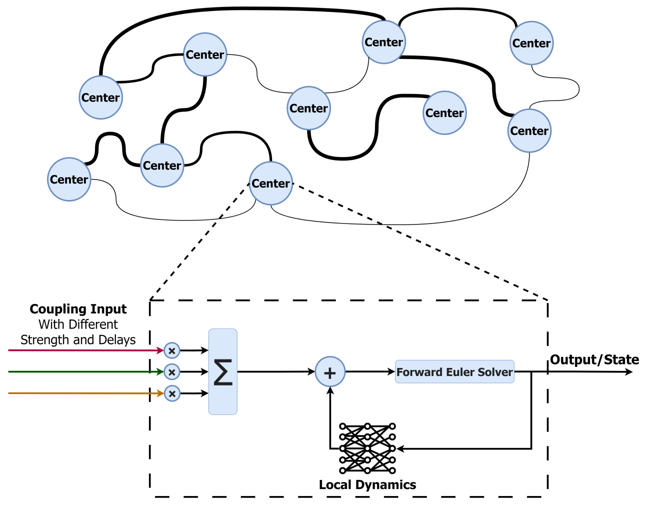


Figure 2: Illustration of the problem at hand. (Top) The interconnected network of centers with different strengths and distances, which results in different delays. (Bottom) The coupling calculation, MLP evaluation, and Forward Euler solver calculations within each center.

Equation (3) formulates the computation that must be performed for each center $i \in [1, N]$ of the model at each timestep of the simulation. h is the timestep size of the FE method, t is the timestep of the simulation, and $C_i(t)$ is calculated as shown in (2). Figure 2 demonstrates the network of centers and the processes that exist within each center. By comparing Equations (2) and (3), we can see the pre-existing heterogeneous computational need within the modified TVB algorithm which makes it a perfect fit for implementation on a heterogeneous platform such as the Versal Adaptive SoC.

$$\vec{X}_i(t+1) = \vec{X}_i(t) + h \cdot \left(MLP(\vec{X}_i(t)) + C_i(t) \right) \quad (3)$$

2.2 Versal Adaptive SoC

The Versal Adaptive SoC is a family of heterogeneous computing platforms developed by AMD. The Versal platform we used for HUMA, VC1902 (on the VCK190 evaluation board) combines the traditional FPGA fabric, referred to as the *Programmable Logic (PL)*, with intelligent engines, referred to as the *AI Engines (AIE)* [31]. The Versal VC1902 platform also includes a processing system (PS) which acts as the control unit of the device. The AIEs of the platform consist of 400 VLIW/SIMD processing units organized in a 2-dimensional interconnected array structure [2]. Each AIE processor has 32 KBytes of data memory, runs at 1.25 GHz, and can perform 8 single-precision, floating-point operations per cycle. We refer to the specific VC1902 device on the VCK190 board as the Versal platform for the rest of this paper.

3 Related Work

The original TVB [22] is a Python-based tool for large-scale brain modeling available to neuroscientists. The numerical algorithms behind the original TVB have been extracted from the main tool by the TVB team and are available to the public [30]. This code is also coded in Python and shows the basic algorithms and calculations performed in the TVB system. The original TVB runs on a CPU and does not utilize any parallelism (multithreading or SIMD). Additionally, the original TVB does not use sparse-calculation techniques for the coupling calculations and also does not include an MLP for local dynamics evaluation.

Some optimized versions of TVB on CPU such as *Fast TVB* [25] and *TVB C++* [17] have also been developed by the TVB team. Fast TVB is a specialized and optimized C implementation of TVB only for a specific neural-mass model (meaning specific connectivity and local dynamics for the large-scale model). Fast TVB trades the generality of the original TVB for the higher performance of a single model. TVB C++ is another, faster version of the original TVB which is more general than Fast TVB, providing the flexibility of the original TVB to some extent. Both of these implementations utilize multithreading and SIMD execution to accelerate the original TVB to mostly answer a specific research question. These implementations are not considered in this work any further due to their limited generality, the lack of MLP usage for the local dynamics approximation, as well as inaccessibility for performing performance evaluations.

The original TVB is also implemented as a JAX-based [4] Python package for parallelized execution on different platforms such as CPUs and GPUs called `vbjax` [6]. This JAX-based TVB implementation is capable of mapping the numerical calculations of the original TVB to the cores of CPU or GPU and, unlike the original TVB, it uses an MLP for local dynamics evaluation. However, this version of TVB does not perform sparse coupling calculations either. The main idea behind the acceleration performed in this version of TVB is *batching* many simulation instances and running them all at the same time, while parallelizing the calculations withing each simulation as well. To permit efficient batching, all simulations running in parallel are only different in the parameter values of the brain model (for example the values of the connection weights), and share all other aspects of the model. This means that all simulations running concurrently share the same control sequence in their calculation and differ only in the values used in those calculations, leading to optimal coalesced memory-access during the calculations, which is an important constraint when trying to quickly fit patient-specific models.

In this work we developed HUMA, the first heterogeneous accelerator which promotes simulation speed primarily by minimizing single-simulation latency, effectively permitting the rapid handling of diverse patient neural makeups.

4 HUMA Accelerator for TVB

In this section, initial problem analysis and design ideas, in addition to the implementation details of HUMA on the heterogeneous Versal fabric are discussed.

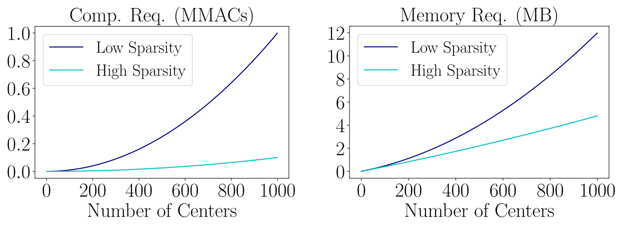
4.1 Problem Analysis

The TVB algorithm was provided to us as a Python script [30]. We ported this Python script to a single-threaded C++ program to use for profiling and implementation validation. Additionally, we added the MLP evaluation and sparse-coupling calculation support to our C++ port of the original TVB. The profiling of the C++-coded original TVB showed that when not using sparse calculations, *66%* of the

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

Table 1: Computation, memory capacity, and data-exchange dependency of each part of the problem on the number of centers (N) and state variables per center (M), number of MLP hidden layers (H) and neurons per hidden layer (L), and the maximum coupling delay with non-zero weight (d_{max}).

	MLPFE	CC
Computation	$O(NML + NHL^2)$	$O(N^2)$
Memory	$O(ML + HL^2 + MN)$	$O(N^2 + Nd_{max})$
Data Exchange	$O(N + M)$	$O(N)$



(a) Computation **(b) Memory**

Figure 3: CC requirements under different sparsity values

runtime of the simulation is spent on coupling calculations and the rest is spent on MLP evaluation. When using sparse calculations, around *26%* of the runtime is spent on coupling calculations and the rest is spent on MLP evaluation.

To analyze the problem and design the heterogeneous system, we divided the problem into two parts: 1) MLP evaluation and FE solver (MLPFE), and 2) Coupling Calculation (CC). These two parts of the problem have different computational needs. The MLPFE part consists of a series of dense-matrix multiplication workloads with organized, coalesced memory accesses, whereas the CC part requires unorganized, non-coalesced memory accesses to the state-variable history and performs sparse operations. The different computation needs inherent in the problem motivated us to implement The Virtual Brain on a heterogeneous platform.

Table 1 shows the dependency of computation, memory capacity, and data exchange of each part of the problem on the model's parameters using big-O notation. As seen in Table 1, the number of centers of the model has a quadratic effect on the computation and memory needs of the CC part of the problem. Furthermore, the memory needs of the CC process have a linear dependency on the largest coupling delay with non-zero weight (d_{max}), which indicates how much history of each center is required to be stored.

When looking at the MLPFE process, both the computation and memory needs of this part of the problem have a quadratic dependency on the number of neurons per hidden layer of the MLP (L). However, unlike the simulation parameters, the hyperparameters of the MLP are fixed throughout the operation of the system. Additionally, the MLPFE part produces $N * M$ values at each timestep which needs to be transferred to the CC part and the main memory. On the other hand, the CC part only produces N coupling values per timestep which has to be transferred to the MLPFE part. The required bandwidths for these value transactions are reflected in the data-exchange row in Table 1.

As mentioned in Section 2, the connections between the centers are sparse. This sparsity affects the computation and memory-capacity requirements of the CC part of the application. According to our analysis, if the sparsity of the connections is more than 50%, storing data in sparse format would save on memory. Figure 3 shows how the computation and memory-capacity requirements of the CC part change when the number of centers varies under two different sparsity values. As the figure reveals, with a higher sparsity, the computing needs still grow quadratically but at a slower rate. On the other hand, the memory requirements switch to a linear growth due to the sparse-storage optimization.

4.2 Design and Implementation

Design and implementation details of HUMA for each part of the problem are discussed first, followed by the overall architecture of the system.

4.2.1 *MLP Evaluation and FE Solver (MLPFE).*

As mentioned earlier, this part of the problem consists of MLP evaluation and Forward Euler solving for all of the centers in the model. We created *MLPFE Engines* that can perform the required computation for a group of centers. All the centers in the model are divided into different groups and the MLP and FE solving workload of all the centers in each group are assigned to one MLPFE engine. Since the MLP evaluation of each center only requires the past state of that center, the MLPFE engines do not require any communication with each other and can run in parallel independently. However, the FE solver requires the coupling input for each center which comes from the CC part of the problem, so the MLPFE engines have an input receiving the calculated couplings for the centers they are responsible for from the CC subsystem at each timestep.

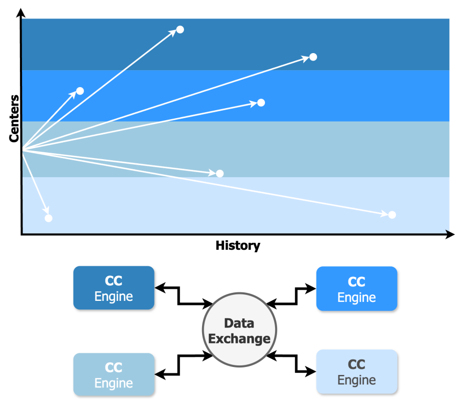
Since the majority of MLPFE engines' workload is the MLP evaluation (which are majority matrix multiplication), these engines were implemented on the AIE of the Versal platform. This choice was made because we found that the AIE performance [27] exceeds the pure FPGA implementations of DNN inference systems [9, 26] in terms of number of calculations per second. Each MLPFE engine time-multiplex the MLP evaluation of different centers that are assigned to it, receives the coupling inputs for each of those centers, and time-multiplex the FE solving step for each of the centers and outputs the result.

4.2.2 *Coupling Calculation (CC).*

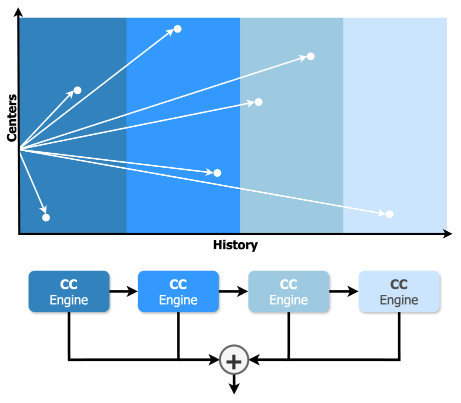
As mentioned in Section 2.1, each center for its coupling calculation needs to access past state variables of certain other centers. This can be seen in the history space diagrams shown in Figure 4. Therefore, the main question for designing the sub-system for this part of the problem becomes how to store and access the state variable history of the centers efficiently.

Center-based Approach The straightforward method is to store the state-variable history by dividing the centers into groups, and storing the data for each group in one memory block. A computation block resides next to each memory block that performs the coupling calculation for the centers of that group. The combination of the computation, memory, and necessary control blocks are referred

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.



(a) Center-based Approach where the history data is divided into groups based on the centers



(b) Delay-based Approach where the history data is divided into groups based on the delays

Figure 4: Diagram of different approaches regarding the coupling calculation

to as *CC Engines*. This way of approaching the problem, which we call a *center-based* approach, is depicted in Figure 4a.

In this design, due to inter-center dependencies mentioned earlier, the CC engines require to be able to access each others memory. Since these accesses are arbitrary and could change from simulation to simulation, a flexible *Data Exchange* block is required to facilitate inter-engine memory accesses. According to our analysis, the bandwidth that the data exchange block is needed to provide has a complexity of $O(NE)$, in which E is the number of CC engines. With this bandwidth requirement, the data exchange block can become a serious bottleneck in the system, in addition to limiting the scalability of the system where increasing the number of CC engines could worsen the performance.

Delay-based Approach In order to increase the data locality and movement, in addition to mitigating the bandwidth-requirement issue of the center-based approach, we designed and implemented a dataflow-style, systolic array architecture for this part of the problem. The main idea of this design is to change our perspective from a center-based approach, to a *delay-based* approach (depicted in Figure 4b) where we group the connectivities with the same delay and perform the calculations of the groups in parallel. In other words, This design uses the fact that the coupling calculation formula (2) can be rewritten as (4) and (5). E indicates the number of delay groups, and D_{eL} and D_{eH} show the lower and upper bounds of each delay group respectively.

$$C_i(t) = K_{post}\left(\sum_{e=1}^{E} C_{ie}(t)\right) \tag{4}$$

$$C_{ie}(t) = \sum_{j=1}^{N} W_{ij}K_{pre}\left(X_j(t-d_{ij}), X_i(t)\right) \quad \textbf{For } d_{ij} \in [D_{eL}, D_{eH}) \tag{5}$$

This rearrangement of equations presents a parallelization opportunity where data locality is increased, and data movement is performed with higher efficiency. For the coupling input of each center i at each timestep t ($C_i(t)$), its calculation is divided into different parts that need to be added together ($C_{ie}(t)$). These parts are coupling inputs that each has a delay in the range of D_{eL} to D_{eH}, specific to each group. In this way, the calculations of each part require no additional data from other parts, and data movement only happens when we move from one timestep to another. To get to an architecture based on this parallelization opportunity, and to increase data locality, we divide the entire range of coupling delays in the model into subgroups, and store the history of all the centers associated with each delay range in the memory of that delay group. This can be seen in Figure 4b.

There are multiple *CC Engines* in this design, and as can be seen in Figure 4b, the engines are chained together in a way that engine #1 can send data to engine #2, and engine #2 can send data to engine #3, and so on. Each engine calculates the couplings associated with a set of connection delays. We take all the unique connection delay values, sort them, and divide the calculation points associated with those delay values between the CC engines. At each timestep, all the calculated state variables enter the CC engine #1. Each state variable might or might not contribute to a coupling with the delay that engine #1 is responsible for. If it is, the coupling value is calculated in the engine using that value. As the simulation advances, the state variables are passed from engine to engine until they are no longer needed for coupling calculation and are absorbed by the last CC engine of the chain.

The data exchange block of the center-based design is replaced by an accumulation block in this approach (according to (5)). This is because all the calculated couplings in different CC engines must be added together to get the final coupling input for each center. This accumulation block is faster, more scalable and can be implemented efficiently in the PL of the Versal. Despite these benefits over having a centralized data exchange block, the accumulation process can quickly become a bottleneck when increasing the number of CC engines. This is because of the fact that values from more and more sources are needed to be added together. To mitigate the challenge, we implemented this block as a 2-level, pipelined tree. More levels can be added when increasing the number of CC engines.

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

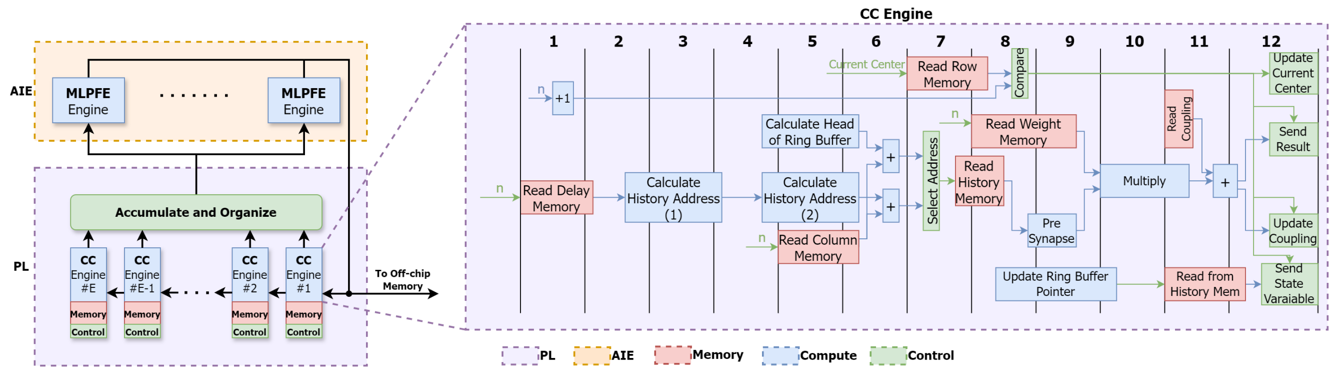


Figure 5: (Left) Block diagram of HUMA (Right) CC-Engine 12-stage calculation pipeline. n **is the pointer traversing the sparse arrays. Some operations span more than one clock cycle.**

However, unlike MLPFE engines, having more CC engines does not necessarily translate to better performance, regardless of the accumulation process. This will be discussed more in Section 5.2.

Within each CC engine, a 12-stage pipeline performs the calculations. Figure 5 shows the CC engine's pipeline. The main calculation only happens in stages 10 and 11, with the rest of the pipeline responsible for calculating the address of the memory accesses and transmitting the results to the accumulator. Additionally, the connectivity data are stored in Compressed Sparse Row (CSR) format which would reduce the required memory capacity for these data. Furthermore, the calculations done within the CC engines utilize the CSR storage format and does not decompress the connectivity data. Figure 5 confirms our analysis that the main challenge of the coupling calculation process is not the calculation, but the memory capacity and access.

All of the memory blocks (containing memory for connectivity and history data) in the calculation pipelines of the CC engines are implemented in the Vitis HLS as true 2-port URAMs. These URAM blocks are needed to be operating in 2-port configuration because of the pipelining with a minimum initiation interval of 1. At the beginning of each simulation, an initialization process populates the history and connectivity data memories from the main memory. This process can be bypassed if many simulations are running back to back with mostly similar connectivity data, only needing to update the changes from the main memory.

For each timestep of the simulation, the calculation pipeline shown in Figure 5 starts. During this process, all center calculations related to the CC engine are performed and sent to the accumulation block. Additionally, the history data that is needed to be passed to the next CC engine are also transmitted during this process. When the described calculation and transmission process are done, a receiving process starts that reads the new history values from the previous CC engine and updates its history ring buffer. Due to data dependencies within the simulation algorithm, the calculation and receiving processes are not overlapping and are performed one after the other. Additionally, for the same reason, each simulation timestep is calculated when the previous timestep is completely finished. Figure 6 illustrates the details of different parts of the CC engine and its connections to different parts of the system.

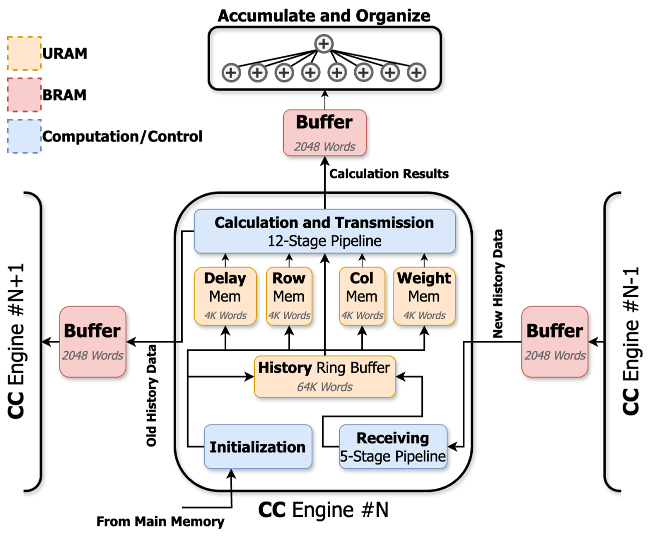


Figure 6: Details of different parts of the CC engine and its connections to the neighboring engines and the accumulation block. BRAM buffers are put between the connections to prevent communication stalls.

In the HUMA implementation shown in Figure 5, at each timestep, while the centers' coupling inputs are calculated in the CC engines, the MLPFE engines evaluate the local dynamic of all the centers and wait for the values from the CC engines. After receiving the calculated coupling values, the MLPFE engines perform the FE solving and produce the next step of the state variables. These newly calculated state variables are shifted into the first CC engine, as the state variable history values across different engines are shifted towards the last CC engine.

The delay-based, CC engine-chain architecture was initially implemented exclusively on the AIEs of the Versal alongside the MLPFE engines [18]. Our initial intuition was that the dataflow nature of the AIEs would make it a good candidate for this design. However, the inherent von Neumann architecture of the AIE tiles, in addition to the limited amount of memory available to each tile did not match the sparse calculation and large memory capacity

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

Table 2: TVB Datasets [5]

	TVB76	TVB192	TVB600[a]	TVB998
No. of Centers	76	192	600	998
Sparsity[b] (%)	72.99	90.42	94.80	96.41
No. of Connections	1,560	3,532	18,736	35,730

[a]A mock-up dataset for performance measurement purposes
[b]Percentage of zero-valued elements in connectivity matrix.

needs of the coupling calculations. Unlike the MLPFE engines which have organized calculations and coalesced memory accesses that can benefit greatly from the vector processors of the AIE tiles, the CC engines could not leverage this capability due to their sparse, arbitrary and unorganized calculations and memory accesses. The lower-than-expected performance of this implementation led us to leverage the heterogeneous computation and memory needs within the TVB algorithm, and move the CC engines to the PL of the Versal platform.

4.2.3 ***Full System***. Figure 5 shows the block diagram of HUMA. The structure of the MLPFE and CC engines was discussed earlier in this section. The accumulation process in the *Accumulate and Organize* block is implemented as a reduction tree. The organization part of this block distributes the calculated couplings among the MLPFE engines and gathers the newly calculated state variables from these engines.

Next to the PL and AIE, an ARM processor controls these two entities. A Python-based pre-processing script takes in the simulation inputs (such as the size of the model and the connectivity data) and generates a configuration file for the MLPFE and CC engines. The ARM processor reads this configuration file and configures each MLPFE and CC engine based on it. It then starts the simulation and reads the outputs when the simulation finishes.

5 Evaluation

In this section, the performance, power, and area of HUMA are evaluated. Additionally, the comparison between HUMA and the related works introduced in Section 3 is also presented in this section.

5.1 Methodology

HUMA and related works were benchmarked using datasets provided by TVB shown in Table 2. It is worth noting that, because of the large size of the TVB998 model, a subset of this dataset with 600 centers (TVB600) was also used for benchmarking. This subset is not accurate in terms of neuroscientific validity and is used for performance measurement purposes only.

For the results presented in this section, we used the state-variable number of 2 ($M = 2$) and an MLP architecture with one hidden layer of 64 neurons ($H = 1$ and $L = 64$) which was sufficient to accurately calculate the local dynamics for the chosen neural-mass model [3]. The MLP was trained to approximate the local dynamics of a Simple 2D Oscillator model [13]. Additionally, the reported numbers in this section are for simulations running for 3,000 timesteps with a 0.05-ms step size. All of the mentioned model

Table 3: Specifications of TVB evaluation platforms

TVB Platform	CPU	GPU	HUMA
Language	Python (JAX)	Python (JAX)	C++ Vitis HLS
Device	AMD Ryzen 3970X	Nvidia Quadro RTX6000	AMD Versal VC1902 (VCK190 Board)
Max Freq.	4.5 GHz	1.77 GHz	AIE: 1.25 GHz PL: 250 MHz
No. of Cores	32 Cores (64 Threads)	4608 CUDA Cores	400 AI Engines
Memory	L1: 2 MB L2: 16 MB L3: 128 MB DDR4: 128 GB	L1: 64 KB L2: 6 MB DDR6: 24 GB	AIE: 12.8 MB PL: 20.5 MB DDR4: 8 GB
RAM BW	95.4 GB/s	672.0 GB/s	136.5 GB/s
Power Cons.[a]	280 W	Single: 162 W Batched: 236 W	47 W
Process	7 nm	12 nm	7 nm
No. of Trans.	15.2 B	18.6 B	37 B

[a]Measured for GPU using `nvidia-smi` tool while running single and batched simulations. Extracted for HUMA by measuring the power of the entire board while running. TDP value reported for CPU.

parameters, including the number of centers and state variables per center, hyperparameters of the MLP, and FE solver input such as timestep size can be changed without the need to re-synthesize the system.

In terms of correctness, the acceleration performed in HUMA did not compromise the accuracy of the algorithm. We verified the output validity of our implementation by comparing it to the original TVB for different models. HUMA uses single-precision floating-point arithmetic, just like CPU and GPU versions, in addition to implementing the same exact formulas to ensure bit-accuracy.

Table 3 shows the specifications of different platforms where TVB has been implemented and are considered here. Comparing very different computing platforms is not a straightforward task. However, using different normalization metrics, we try to get an insight into the performance comparison of these systems. For both CPU and GPU systems, we used the JAX-based implementation of TVB (`vbjax`). The multi-threading and SIMD capabilities of the CPU, in addition to the parallel computation capacity of the GPU are fully utilized in `vbjax`. Although JAX-based implementations might compromise some performance for the abstraction they provide, it has been shown that the XLA compiler powering JAX is highly competitive or can even outperform hand-tuned parallelized or CUDA implementations for the types of workloads consisting of local dynamics and coupling calculations [16].

5.2 HUMA Results

Table 4 shows the latency, throughput, and power consumption of the heterogeneous HUMA implementation with 64 MLPFE and 32

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

Table 4: HUMA results (on Versal VC1902 and with 64 MLPFE and 32 CC engines)

	TVB76	TVB192	TVB600	TVB998
Latency[a] (ms)	4.1	7.5	21.1	33.8
Throughput ($\frac{\text{iters}}{\text{s}}$)	726,037	398,830	142,186	88,683
Power (Watts)	46.71			

[a]For a simulation with 3,000 timesteps.

CC engines on the Versal device. Additionally, Table 5 shows the resource utilization and power consumption breakdown of HUMA. As mentioned earlier, HUMA is implemented on a VC1902 device on the VCK190 evaluation board. The verification of HUMA, in addition to performance and power measurements were all done on the hardware itself. Version 2022.2 of Vivado and Vitis toolsets were used.

HUMA utilizes URAMs of the Versal PL to store the connectivity and history data. The BRAM resources of the PL are used as buffers between connections to avoid stalls in the transmitting modules. This division of memory resources helped with the acceleration of the place-and-route processes when implementing this design. Table 5 shows that HUMA implementation on Versal, as expected, saturates the memory resources of the PL before any of the logic and computation resources. In the AIEs of the Versal implementation, a communication kernel exists next to the 64 MLPFE engines. However, all of these engines require additional memory to store the parameters of the MLP. For this reason, each engine uses the memory bank of its neighboring AIE tile as additional memory to store the MLP parameters, which brings the utilization of the AIEs to 130 tiles.

More than 50% of the dynamic power consumption of HUMA is due to the AI Engines, as shown in Table 5. This is mostly due to the fact that in the AIEs of the Versal platform, 64 small vector processors (the MLPFE engines) are running at 1.25 GHz, performing the MLP evaluations. The Network-on-Chip (NoC) of the Versal platform consumes around 12.4% of the total power consumption. The NoC connects different parts of the platform, including the ARM processor and the CC engines that reside in the PL to the off-chip main memory controller.

The MLPFE and CC engines run in parallel, and as a result the overall performance of the system depends on which of these parts of the system is running slower. The system's bottleneck itself is determined by different simulation parameters such as the number of centers, the number of state variables per center, the MLP hyper-parameters, and also the sparsity of the connections. Furthermore, during our analysis, we realized that the distribution of values in the delay matrix also affects the performance of the CC part of the application, which in turn can shift the bottleneck of the system. Figure 7 shows the distribution of the non-zero values in the delay matrix of dataset TVB192. As can be seen, the concentration of values around 300 timesteps is very high. As a result, the workload assigned to the CC engines responsible for that delay range is high which would bottleneck the CC execution. It is worth noting that in order to reduce the maximum workload assigned to a single CC

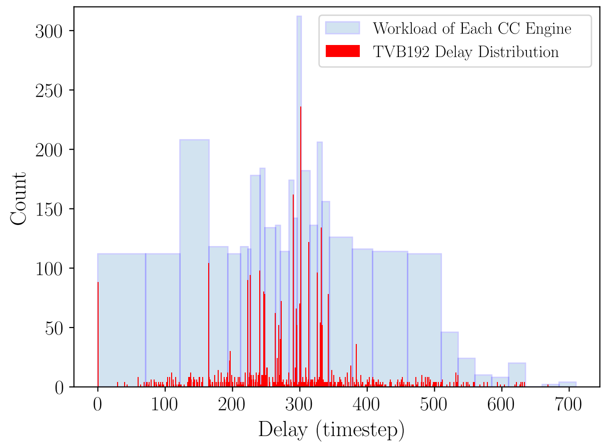


Figure 7: Delay value distribution of TVB192 dataset and the workload assigned to each of the 32 CC engines

engine, the pre-processor of the application assigns delay ranges in a way to balance the workloads, as can be seen in Figure 7. The way the performance of HUMA changes with regards to different simulation parameters depends on which part of the system is the bottleneck, and the formulas shown in Table 1.

5.3 Comparison to Related Work

In this section, HUMA is compared to the JAX-based CPU and GPU versions of TVB introduced in Section 3. For the CPU version, we used an AMD Ryzen Threadripper 3970X with 32 cores. This modern-day, server-class CPU has large cache memories and is optimized for multithreaded execution. For the GPU version of TVB, we used an Nvidia Quadro RTX6000 GPU, which has a relatively high performance per Watt. We benchmarked both these systems with a range of different models and configurations. Figures 8a and 8b show the latency and throughput results of the CPU and GPU version of TVB next to HUMA, respectively. For latency, the *single-simulation* configuration of the CPU and GPU versions of TVB was used, whereas in the throughput results the *batched-simulation* performance is reported. Additionally, Figures 8c and 8d show the energy consumption and performance efficiency of the benchmarked patforms, respectively.

Table 6 presents a comparison of the different TVB implementations as ported on the CPU, GPU, and Versal platforms. In order to make the comparison fairer and easier, reported numbers are normalized by the number of centers in the model. This is done by dividing the latency and multiplying the throughput results by the number of centers in the model, respectively. This normalization makes comparison easier across different models, and also fairer for implementations that experience under-utilization in certain model sizes.

In terms of latency, HUMA performs on average 27× and 14× faster compared to the CPU and GPU versions of TVB, respectively. Additionally, since it consumes less power compared to the benchmarked CPU and GPU, the energy efficiency of HUMA is around 85× and 52× higher compared to the TVB implementation on CPU and GPU, respectively.

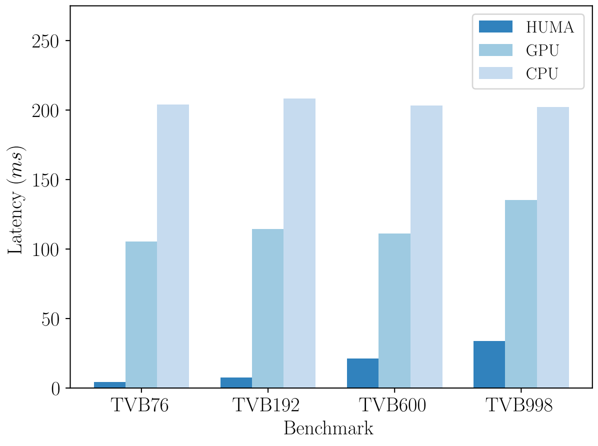
Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

Table 5: Resource utilization and power consumption breakdown of HUMA on the Versal platform

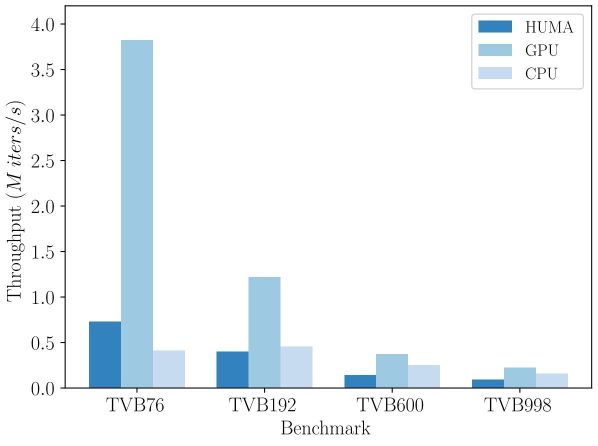
Resource		Utilization (–)	% of Total	Power Consumption (W)	% of Total
Programmable Logic				6.57	15.06 %
	LUT	131,255	14.59 %	0.78	1.78 %
	FF	182,167	10.12 %		
	DSP	191	9.71 %	0.21	0.48 %
	BRAM	466	48.19 %	1.15	2.64 %
	URAM	386	83.37 %	0.26	0.60 %
	Other			4.17	9.56 %
AI Engines		130[a]	32.50 %	15.64	35.85 %
Network-on-Chip				5.40	12.38 %
Device Static				14.67	33.62 %
Total				43.63[b]	100.0 %

[a]Half of the utilized AIE tiles only used for their memory.

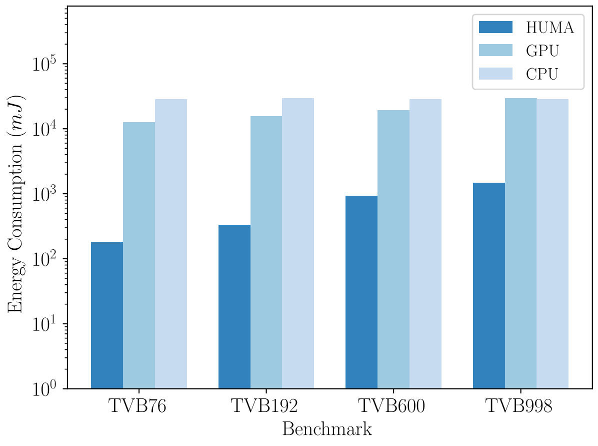
[b]This is a post-implementation estimation by the Vivado power tool. The actual power consumption is 46.71 W.



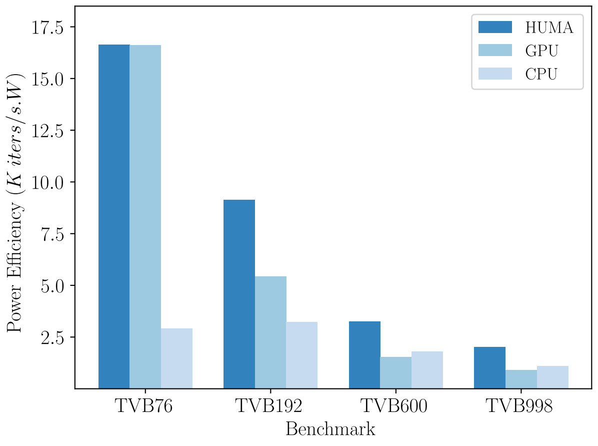
(a) Latency (as measured for a single simulation with 3,000 timesteps)



(b) Throughput (as measured for the CPU and GPU versions in the batched-simulation configuration)



(c) Energy Consumption (as measured for a single simulation with 3,000 timesteps)



(d) Performance Efficiency (as measured for the CPU and GPU versions in the batched-simulation configuration)

Figure 8: Performance comparison between HUMA (AMD VC1902 on VCK190 board), GPU (Nvidia Quadro RTX6000), and CPU (AMD Ryzen Threadripper 3970X) versions of TVB.

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

Table 6: Overview of the normalized (by the number of centers) results of different versions of The Virtual Brain

Implementation	TVB on CPU [6]	TVB on GPU [6]	TVB on Versal (HUMA)
Avg. Latency[a] **(ms)**	1.08	0.57	0.04
Avg. Throughput (M iters/s)	105.9	243.1	76.6
Avg. Energy Consumption[a,b] **(mJ)**	150.8	93.06	1.91
Avg. Throughput per Watt[b] **(K iters/s.Watt)**	756.2	1,030.0	1,626.4

[a]For a single simulation with 3000 timesteps.

[b]Half the reported TDP value is used for CPU calculations.

The low latency of HUMA means that it would fit very well within the model-fitting processes that utilize MCMC in their Bayesian inference framework. This means that, by utilizing HUMA, a patient-specific, TVB brain model can be *trained an order of magnitude faster* compared to CPU or GPU while *consuming a fraction of their power.* This would benefit the patients and doctors who use large-scale brain models such as TVB in the course of their treatment significantly.

In terms of throughput, since the GPU is capable of batching a large number of simulations and running them at the same time, the GPU version of TVB performs around 3× better compared to HUMA. This superiority of GPU over the Versal platform in terms of throughput was expected since the GPU, with its substantial parallel capacity, is designed for delivering very high throughputs. The CPU implementation also outperforms HUMA when it comes to throughput by around 1.4× on average. However, as can be seen in Figure 8b, HUMA has slightly higher throughput when running smaller model of 76 centers. Although both GPU and CPU versions outperform HUMA by the throughput metric, when we look at the normalized metrics, the comparison turns out differently.

When normalizing throughput results by power consumption, we can see that HUMA has on average around 1.7× and 2.3× higher throughput per Watt compared to the GPU and CPU versions of TVB, respectively. Although both the GPU and CPU implementations have higher throughput compared to HUMA, the application-specific, dataflow design of the heterogeneous system on the Versal platform consumes much less power which results in better performance efficiency. It is worth noting that for both energy consumption and performance efficiency calculation of the CPU version, we used half the reported TDP value of the CPU mentioned in Table 3 to maintain fairness in comparison. However, realistically, the CPU version uses more than half the TDP value when executing many simulations at the same time, utilizing all 32 cores of the device.

Finally, a word on area utilization. The Versal VC1902 SoC used in HUMA has 37 billion transistors. However, unlike the CPU and GPU versions of TVB which uses almost all the device capacity in a high-throughput, batched configuration, HUMA only utilizes a fraction of all the resources available in the Versal platform. Because of the heterogeneous nature of the Versal platform, calculating exactly what percentage of all the transistors are actually being used is not possible. Though there is no easy way of calculating that metric, by looking at the resource-utilization and power-consumption results of HUMA shown in Table 5 and the specifications of the CPU and GPU used (such as the high-bandwidth, on-board GDDR6 memory), we can claim that HUMA delivers at least equal if not higher *throughput per area* compared to the both the CPU and GPU versions of TVB.

6 Conclusion

In this work we presented HUMA, an ultra low-latency, power-efficient, heterogeneous accelerator of The Virtual Brain (TVB) on Versal VC1902 Adaptive SoC. The Virtual Brain is one of the leading, large-scale brain modeling tools that can build patient-specific, personalized digital brain twins used for different clinical purposes such as epilepsy. We designed a dataflow, systolic array architecture for the TVB algorithm which performed on average around 27× better compared to a multi-threaded CPU implementation of TVB in terms of latency, and more than 2× in terms of power efficiency. When compared to a high-performance GPU version of TVB, HUMA delivered on average 14× lower latency and 55× less energy consumption. In terms of throughput, although HUMA had lower throughput compared to the GPU version, it delivered around 1.7× higher throughput per Watt.

TVB models require training for each patient to be as accurate as possible. However, this training process could be very time-consuming, and make using these models challenging in practice. HUMA, the heterogeneous Versal implementation we developed, with its low latency and low power consumption, makes this process faster and more efficient for patients and doctors who benefit from large-scale brain modeling in the course of their treatment.

Acknowledgments

This paper is partially supported by the European-Union Horizon Europe R&I program through projects SEPTON (no. 101094901) and SECURED (no. 101095717) and through the NWO - Gravitation Programme DBI2 (no. 024.005.022). We thank AMD/Xilinx and NVIDIA corporations for their donations. We are grateful to Dr. Mario Negrello, Freddie Renyard, and Marmaduke Woodman for all their help and feedback throughout this work.

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.

References

[1] M. Faizan Ahmad, James Murphy, Deniz Vatansever, Emmanuel A. Stamatakis, and Simon J. Godsill. [n. d.]. Bayesian Inference of Task-Based Functional Brain Connectivity Using Markov Chain Monte Carlo Methods. 10, 7 ([n. d.]), 1150–1159. https://doi.org/10.1109/JSTSP.2016.2599010

[2] AMD. 2023. *Versal Adaptive SoC AI Engine Architecture Manual (AM009).* https://docs.xilinx.com/r/en-US/am009-versal-ai-engine

[3] Nina Baldy, Martin Breyton, Marmaduke M. Woodman, Viktor K. Jirsa, and Meysam Hashemi. 2024. Efficient Inference on a Network of Spiking Neurons using Deep Learning. *bioRxiv* (2024). https://doi.org/10.1101/2024.01.26.577077

[4] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. *JAX: composable transformations of Python+NumPy programs.* http://github.com/google/jax

[5] The Virtual Brain. 2019. *tvb-data.* https://zenodo.org/records/3474071#.XZmcU-cza_U

[6] The Virtual Brain. 2024. *vbjax.* https://github.com/ins-amu/vbjax/tree/main

[7] Shuo Chen, Yishi Xing, Jian Kang, Peter Kochunov, and L Elliot Hong. [n. d.]. Bayesian modeling of dependence in brain connectivity data. 21, 2 ([n. d.]), 269–286. https://doi.org/10.1093/biostatistics/kxy046

[8] Timothy A. Davis. 2006. *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2).* Society for Industrial and Applied Mathematics, USA.

[9] Kaiyuan Guo, Shulin Zeng, Jincheng Yu, Yu Wang, and Huazhong Yang. 2019. A Survey of FPGA-based Neural Network Inference Accelerators. 12, 1, Article 2 (3 2019), 26 pages. https://doi.org/10.1145/3289185

[10] Imali Hettiarachchi, Shady Mohamed, and Saeid Nahavandi. [n. d.]. A marginalised Markov Chain Monte Carlo approach for model based analysis of EEG data. In *2012 9th IEEE International Symposium on Biomedical Imaging (ISBI)* (2012-05). 1539–1542. https://doi.org/10.1109/ISBI.2012.6235866

[11] Max Hinne, Tom M. Heskes, Christian F. Beckmann, and Marcel van Gerven. 2013. Bayesian inference of structural brain networks. *NeuroImage* 66 (2013), 543–552. https://doi.org/10.1016/j.neuroimage.2012.09.068

[12] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2 (1989), 359–366. https://doi.org/10.1016/0893-6080%2889%2990020-8

[13] E.M. Izhikevich. [n. d.]. Which Model to Use for Cortical Spiking Neurons? 15, 5 ([n. d.]), 1063–1070. https://doi.org/10.1109/TNN.2004.832719

[14] V. K. Jirsa, T. Proix, D. Perdikis, M. M. Woodman, H. Wang, J. Gonzalez-Martinez, C. Bernard, C. Bénar, M. Guye, P. Chauvel, and F. Bartolomei. 2017. The Virtual Epileptic Patient: Individualized whole-brain models of epilepsy spread. 145 (2017), 377–388. https://doi.org/10.1016/j.neuroimage.2016.04.049

[15] Farad Khoyratee, Filippo Grassia, Sylvain Saïghi, and Timothée Levi. 2019. Optimized Real-Time Biomimetic Neural Network on FPGA for Bio-hybridization. *Frontiers in Neuroscience* 13 (2019). https://doi.org/10.3389/fnins.2019.00377

[16] Lennart P.L. Landsmeer, Max C.W. Engelen, Rene Miedema, and Christos Strydis. 2024. Tricking AI chips into simulating the human brain: A detailed performance analysis. *Neurocomputing* 598 (2024), 127953. https://doi.org/10.1016/j.neucom.2024.127953

[17] Ignacio Martin, Gorka Zamora, Jan Fousek, Michael Schirner, Petra Ritter, Viktor K. Jirsa, Gustavo Deco, and Gustavo Patow. 2024. TVB C++: A Fast and Flexible Back-End for The Virtual Brain. https://doi.org/10.48550/arXiv.2405.18788

[18] Amirreza Movahedin. 2024. Heterogeneous Acceleration of Neural-Mass Models towards Digital Brain Twins. http://resolver.tudelft.nl/uuid:4e1f0693-ae57-42d9-bbfe-8c8d0edd330a

[19] A. Movahedin. 2024. *TVB Pipeline. Created in BioRender.* https://BioRender.com/g61o515 Parts of the figure from *here* and *here.*

[20] National Academy of Engineering. 2008. *Grand Challenges.* https://www.engineeringchallenges.org/challenges.aspx

[21] Anagh Pathak, Dipanjan Roy, and Arpan Banerjee. [n. d.]. Whole-Brain Network Models: From Physics to Bedside. 16 ([n. d.]). https://doi.org/10.3389/fncom.2022.866517 Publisher: Frontiers.

[22] Leon Paula, Knock Stuart, Woodman M., Domide Lia, Mersmann Jochen, McIntosh Anthony, and Jirsa Viktor. 2013. The Virtual Brain: a simulator of primate brain network dynamics. *Frontiers in Neuroinformatics* 7, 10 (2013). https://doi.org/10.3389/fninf.2013.00010

[23] Dirk Ridder, Muhammad Ali Siddiqi, Justin Dauwels, Wouter Serdijn, and Christos Strydis. 2024. NeuroDots: From Single-Target to Brain-Network Modulation: Why and What Is Needed? *Neuromodulation: Technology at the Neural Interface* 27 (04 2024). https://doi.org/10.1016/j.neurom.2024.01.003

[24] Paula Sanz-Leon, Stuart A. Knock, Andreas Spiegler, and Viktor K. Jirsa. 2015. Mathematical framework for large-scale brain network modeling in The Virtual Brain. 111 (2015), 385–430. https://doi.org/10.1016/j.neuroimage.2015.01.002

[25] Michael Schirner, Lia Domide, Dionysios Perdikis, Paul Triebkorn, Leon Stefanovski, Roopa Kalsank Pai, Paula Prodan, Bogdan Valean, Jessica Palmer, Chloê Langford, André Blickensdörfer, Michiel A. van der Vlag, Sandra Díaz-Pier, Alexander Peyser, Wouter Klijn, Dirk Pleiter, Anne Nahm, Oliver Schmid, Marmaduke M. Woodman, Lyuba Zehl, Jan Fousek, Spase Petkoski, Lionel Kusch, Meysam Hashemi, Daniele Marinazzo, J. F. Mangin, Agnes Flöel, Simisola Akintoye, Bernd Carsten Stahl, Michael Cepic, Emily Johnson, Gustavo Deco, Anthony Randal Mcintosh, Claus C. Hilgetag, Marc Morgan, Bernd Schuller, Alex Upton, Colin McMurtrie, Timo Dickscheid, Jan G. Bjaalie, Katrin Amunts, Jochen Mersmann, Viktor Jirsa, and Petra Ritter. 2022. Brain simulation as a cloud service: The Virtual Brain on EBRAINS. *NeuroImage* 251 (2022). https://doi.org/10.1016/j.neuroimage.2022.118973

[26] Ahmad Shawahna, Sadiq M. Sait, and Aiman El-Maleh. 2019. FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review. *IEEE Access* 7 (2019), 7823–7859. https://doi.org/10.1109/ACCESS.2018.2890150

[27] Endri Taka, Aman Arora, Kai-Chiang Wu, and Diana Marculescu. 2023. MaxEVA: Maximizing the Efficiency of Matrix Multiplication on Versal AI Engine. In *2023 International Conference on Field Programmable Technology (ICFPT).* 96–105. https://doi.org/10.1109/ICFPT59805.2023.00016

[28] Huifang E Wang, Paul Triebkorn, Martin Breyton, Borana Dollomaja, Jean-Didier Lemarechal, Spase Petkoski, Pierpaolo Sorrentino, Damien Depannemaecker, Meysam Hashemi, and Viktor K Jirsa. 2024. Virtual brain twins: from basic neuroscience to clinical use. *National Science Review* 11, 5 (2024), nwae079. https://doi.org/10.1093/nsr/nwae079

[29] Junsong Wang, Ernst Niebur, Jinyu Hu, and Xiaoli Li. 2016. Suppressing epileptic activity in a neural mass model using a closed-loop proportional-integral controller. *Scientific Reports* 6 (2016). https://doi.org/10.1038/srep27344

[30] Marmaduke Woodman. 2020. *tvb-algo.* https://github.com/maedoc/tvb-algo

[31] Xilinx. 2020. *Versal: The First Adaptive Compute Acceleration Platform (ACAP).* https://docs.xilinx.com/v/u/en-US/wp505-versal-acap

Downloaded from the ACM Digital Library by TU Delft Library on April 7, 2025.