A Programmable Hardware Accelerator for Simulating Dynamical Systems

Jaeha Kung Yun Long Duckhwan Kim Saibal Mukhopadhyay
Georgia Institute of Technology
{jhkung,yunlong,kimduckhwan,smukhopadhyay6}@gatech.edu

ABSTRACT

The fast and energy-efficient simulation of dynamical systems defined by coupled ordinary/partial differential equations has emerged as an important problem. The accelerated simulation of coupled ODE/PDE is critical for analysis of physical systems as well as computing with dynamical systems. This paper presents a fast and programmable accelerator for simulating dynamical systems. The computing model of the proposed platform is based on multilayer cellular nonlinear network (CeNN) augmented with nonlinear function evaluation engines. The platform can be programmed to accelerate wide classes of ODEs/PDEs by modulating the connectivity within the multilayer CeNN engine. An innovative hardware architecture including data reuse, memory hierarchy, and near-memory processing is designed to accelerate the augmented multilayer CeNN. A dataflow model is presented which is supported by optimized memory hierarchy for efficient function evaluation. The proposed solver is designed and synthesized in 15nm technology for the hardware analysis. The performance is evaluated and compared to GPU nodes when solving wide classes of differential equations and the power consumption is analyzed to show orders of magnitude improvement in energy efficiency.

CCS CONCEPTS

• **Computer systems organization** → **Cellular architectures**; • **Hardware** → **Emerging architectures**;

KEYWORDS

Cellular nonlinear network; Dynamical systems; Hardware accelerator; Scientific simulation

ACM Reference format:
Jaeha Kung Yun Long Duckhwan Kim Saibal Mukhopadhyay Georgia Institute of Technology . 2017. A Programmable Hardware Accelerator for Simulating Dynamical Systems. In *Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017,* 13 pages.
https://doi.org/10.1145/3079856.3080252

ISCA '17, June 24-28, 2017, Toronto, ON, Canada

ACM ISBN 978-1-4503-4892-8/17/06…$15.00
https://doi.org/10.1145/3079856.3080252

1 INTRODUCTION

The dynamical system, represented as coupled ordinary/partial differential (continuous time) or difference (discrete time) equations (ODE/PDE), is a mathematical model for describing complex processes. The solution of dynamical systems is an important computing problem. The scientific simulations, a key driver for high-performance computing, are defined by solution of coupled ODEs/PDEs. There are many real-time control problems such as bipedal robotic walking, UAV path planning, or aircraft control that necessitates real-time ODE/PDE solution for fast actions [13, 16, 24, 26].

More recently, there is a growing interest in computing with dynamical systems, e.g. reaction-diffusion equations or oscillatory networks [1, 9, 29, 32, 38]. The concept of building a Turing complete machine using reaction-diffusion equations has been proposed [1]. The coupled oscillators based dynamical systems are being explored as a platform for solving complex problems [28, 31, 33, 41]. Computing with biophysical neuron with dynamics modeled using coupled PDEs is another well-known example of dynamical system based computing [29, 32]. The dynamical system based computing is showing promise in solving complex problems in computer vision, graph theory, optimization, to name a few [29, 32, 33, 41].

There are three major trends in designing hardware platforms for accelerating solution of coupled ODE/PDE to support dynamical system analysis. First, the Graphical Processor Unit (GPU) helps accelerate ODE/PDE solution for scientific computation [3, 22]. The GPU-based platforms provide the advantage of programmability, but in general are more power hungry and less energy-efficient. Second, the FPGA-based coprocessors have also been developed to speed up the computation of conventional numerical analysis methods [2, 43, 46]. However, FPGAs only accelerate the numerical algorithms, do not provide a different computing model. Finally, there is significant effort in exploring ASICs and non-CMOS devices to accelerate specific dynamical systems, for example, simple first/second order equations, oscillatory networks, and spiking systems [25, 29, 31–33, 41]. The ASIC-based approach provides high energy-efficiency, but lack the capability of solving different types of dynamical systems in a single platform. Likewise, directly using the internal dynamics of a device to build a computing platform does not provide the ability to 'program' a required dynamical behavior.

A fundamentally different computing model for dynamical system analysis is Cellular Nonlinear Network (CeNN) [8]. The CeNN is composed of an array of cells where each cell follows an ODE based dynamics [8]. Each cell in CeNN is connected to (local) neighboring cells resulting in a system of coupled ODEs. The weight of local connections, referred to as the **templates**, defines the coupling and hence, the nature of the system of the equations. A multilayer CeNN can realize a system defined by multiple coupled PDEs, where each layer represents the 'first-order' equation. The advantage of

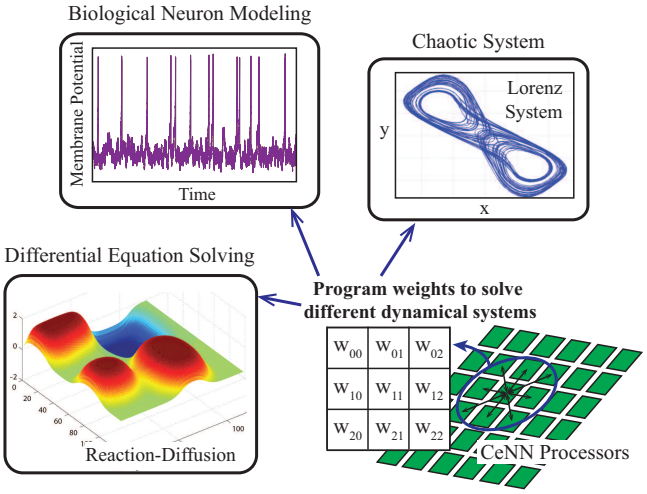

Figure 1: Programmable computing model by using Cellular Nonlinear Network (CeNN) with some dynamical systems that can be modeled by CeNN.

CeNN based computation originates from the inherent ODE-based cell dynamics, the high-degree of parallelism, local connectivity, and programmability to solve wide classes of 'system of equations' (Fig. 1).

Although the CeNN platform is most widely known for image processing, several past efforts have shown that a multilayer CeNN with linear and nonlinear templates can be used to solve different types of coupled differential equations [7, 10, 12, 37, 42, 47]. There are many studies on how to map a specific equation to CeNN algorithm [5, 12, 21, 42, 47]. Significant research efforts have also been directed to design digital and analog CeNN chips for image processing applications [20, 23, 36, 40]. However, the mapping algorithms developed in prior studies mostly focused on *specific equations* and/or *linear templates*. Likewise, the hardware accelerators were designed for image processing applications with *spatially* and/or *temporally invariant templates*. Therefore, the prior efforts are not sufficient to develop a generic dynamical system simulator with *nonlinear interactions* between equations leading to *space and time variant templates*.

This paper presents a generic and programmable hardware platform to solve coupled ODEs/PDEs and analyze dynamical systems. The contribution of the paper is to connect the versatility of the CeNN computing model with innovative 'fully digital' architecture to instigate orders of magnitude higher performance and energy-efficiency over generic purpose GPUs (GP-GPUs) in solving coupled differential equations. The main contributions can be summarized as follows:

1. We present a **CeNN based computing model and mathematical mapping** approach for solving dynamical systems. We map a dynamical system defined by multiple coupled PDEs to a multilayer CeNN, where each layer discretizes one equation in space, individual cell realizes the temporal dynamics, coupling between layers represents the interactions of different equations (nonlinear template). The mapping process identifies the linear (finite difference method for space discretization) and nonlinear templates to realize a given dynamical system.
2. We present **a memory-centric model** to enable spatially and temporally varying nonlinear templates that are updated dynamically during evolution of a dynamical system. We augment the multilayer CeNN with nonlinear function evaluation to enable real-time template update. A memory-based model composed of a hierarchy of look-up tables (LUT) and finite series expansion is proposed for fast function evaluation during operation.
3. We propose a **memory-centric architecture** with digital compute engine. The memory system is composed of global and local buffers for storing inputs, states, and template weights with LUTs for nonlinear function evaluation. The compute layer consists of an array of digital processing engines (PEs) that performs the convolution operation over template weights and current cell-states to generate the new states of the cells. We explore integration of computing platform with high-bandwidth and high-concurrency memory system, namely, hybrid memory cube (HMC).
4. We present **programming and execution model** of the dynamic system solver. The programming includes communicating number of layers, the template weights, and LUTs. The execution is managed by a dataflow model that pushes data from main memory to compute engines.

2 COMPUTATION MODEL

A CeNN is defined as a regular structure where cells are locally connected to their neighboring cells within a given radius. The dynamics of each cell in CeNN is defined by:

$$\frac{\partial x_{ij}(t)}{\partial t} = -x_{ij}(t) + \sum_{C(k,l)\in N_r(i,j)} \hat{A}_{kl}(t)\cdot x_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} A_{kl}\cdot y_{kl}(t) + \sum_{C(k,l)\in N_r(i,j)} B_{kl}\cdot u_{kl}(t) + z \quad (1)$$

$$y_{ij}(t) = f(x_{ij}(t)) = \begin{cases} -1 & \text{for} \quad x_{ij}(t) < -1 \\ x_{ij}(t) & \text{for} \quad |x_{ij}(t)| \le 1 \\ 1 & \text{for} \quad x_{ij}(t) > 1 \end{cases} \quad (2)$$

where i is the row index, j is the column index, $x_{ij}(t)$ is the state, $y_{ij}(t)$ is the output, $u_{ij}(t)$ is the input, z is the offset, $\hat{\mathbf{A}}$ is the state (feedback) template which is a function of time-varying state variables $x_{ij}(t)$ and $x_{kl}(t)$, $\mathbf{A}$ is the output (feedback) template, and $\mathbf{B}$ is the feedforward template for each cell $C(i,j)$. Here, $N_r(i,j)$ represents neighbors of a cell $C(i,j)$ within radius r where (k,l) is the index of those cells.

In multilayer CeNN, each 2D array defines one equation discretized in space, while the connections between nodes in different layers represent the coupling between different equations (Fig. 2). The prior efforts in exploring CeNN based differential equation solvers only focused on space-/time-invariant linear templates. We develop a more generic approach to map multiple coupled equations with nonlinear interactions by programming the template weights, ($\hat{A}_{kl}(t)$, A_{kl}, and B_{kl}).

The first step is to determine the number of layers as a function of the number of variables involved and the highest order of derivatives for each variable in the system. Second, we identify each first-order differential equations and map that to a layer within the multilayer

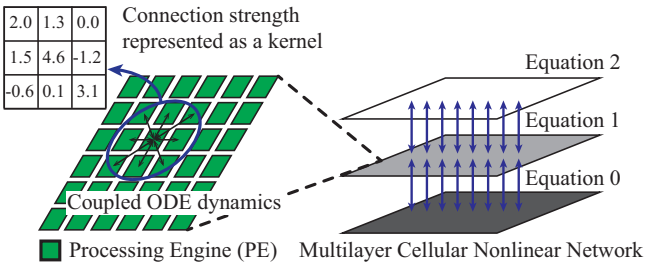

Figure 2: A 2-dimensional CeNN processing array having cell states locally coupled. This structure can be extended to a multilayer CeNN platform to handle coupled systems.

CeNN. Assume a dynamic system described by the following coupled differential equations:

$$\ddot{\omega} = f_1(\omega, \varphi) \text{ and } \dot{\varphi} = f_2(\omega, \varphi). \tag{3}$$

As CeNN cell dynamics is described by the first-order ODE, equation (3) is re-written as

$$\dot{\omega} = \chi \text{ and } \dot{\chi} = f_1(\omega, \varphi) \text{ and } \dot{\varphi} = f_2(\omega, \varphi). \tag{4}$$

Third, while mapping an equation to a layer in CeNN, we will identify whether the equation involves linear or nonlinear templates, and program the templates following the methods explained in Sections 2.1 and 2.2.

2.1 Mapping Linear Systems

A dynamical system can be described as a scalar ODE given as $\dot{\varphi} = g(x)$ where $g : \mathbb{R} \to \mathbb{R}$ is a continuous function. Let us first explain the mapping process when $g(x)$ is a linear equation, for example, heat equation given by:

$$\frac{\partial \varphi(x,y,t)}{\partial t} = \kappa \cdot \Delta\varphi(x,y,t) \tag{5}$$

where $\varphi(x,y,t)$ is temperature at location (x,y) at time t and κ is thermal conductivity. As CeNN cell dynamics in equation (1) is an ordinary differential equation (ODE), Laplace operator is discretized by Euler method. Then, equation (5) can be approximated as

$$\frac{\partial \varphi(x,y,t)}{\partial t} = \kappa \cdot \{\frac{\varphi(x+h,y,t) + \varphi(x-h,y,t) - 2 \cdot \varphi(x,y,t)}{h^2} + \frac{\varphi(x,y+h,t) + \varphi(x,y-h,t) - 2 \cdot \varphi(x,y,t)}{h^2}\} \tag{6}$$

where h is the step size in $\mathbb{R}^2$ Euclidean space. With an infinitesimal h, the approximation error goes to zero. Then, heat diffusion shown in equation (5) can be solved by CeNN model by setting parameters in equation (1) to

$$\hat{\mathbf{A}} = \kappa \cdot \begin{bmatrix} 0 & 1/h^2 & 0 \\ 1/h^2 & -4/h^2+1 & 1/h^2 \\ 0 & 1/h^2 & 0 \end{bmatrix}, \mathbf{A} = \mathbf{0}, \mathbf{B} = \mathbf{0}, z = 0. \tag{7}$$

Likewise, if the system of interest is described by a partial differential equation, we need to discretize the equation in space by using finite difference method to make them ODE [12, 30]. This discretization decides the linear part of state (feedback) template $\hat{\mathbf{A}}$. Note that in most physical systems, the output template $\mathbf{A}$ will be zero; it is used for applications like image processing or associative memory.

2.2 Mapping Nonlinear Systems

Our proposed approach for mapping equations with nonlinear templates is based on Taylor series based approximation for wide range of nonlinear functions (beyond polynomial). Assume that there is an additive nonlinear physical behavior observed on top of heat propagation in the system. The equation (5) converts to

$$\frac{\partial \varphi(x,y,t)}{\partial t} = \kappa \cdot \Delta\varphi(x,y,t) + l(\varphi(x,y,t)) \tag{8}$$

where $l(\cdot)$ is a continuous and infinitely differentiable nonlinear function. Using Taylor series expansion a nonlinear function can be approximated by a polynomial function at a specific point 'p'. By applying the Taylor series of degree three on function $l(\cdot)$ at p, equation (8) becomes

$$\frac{\partial \varphi(x,y,t)}{\partial t} \approx \kappa \cdot \Delta\varphi(x,y,t) + \{l(p) + l^{(1)}(p) \cdot (\varphi(x,y,t) - p) + l^{(2)}(p) \cdot (\varphi(x,y,t) - p)^2 + l^{(3)}(p) \cdot (\varphi(x,y,t) - p)^3\}. \tag{9}$$

With simple derivation from equation (9), the parameters in equation (1) becomes

$$\hat{\mathbf{A}} = \kappa \cdot \begin{bmatrix} 0 & 1/h^2 & 0 \\ 1/h^2 & -4/h^2+1 & 1/h^2 \\ 0 & 1/h^2 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{A} = 0, \mathbf{B} = 0, z = c_3 \tag{10}$$

$$\textit{where} \begin{cases} \alpha = c_0 + c_1 \cdot \varphi(x,y,t) + c_2 \cdot \varphi(x,y,t)^2 \\ c_0 = l^{(1)}(p) - 2p \cdot l^{(2)}(p) + 3p^2 \cdot l^{(3)}(p) \\ c_1 = l^{(2)}(p) - 3p \cdot l^{(3)}(p) \\ c_2 = l^{(3)}(p) \\ c_3 = l(p) - p \cdot l^{(1)}(p) + p^2 \cdot l^{(2)}(p) - p^3 \cdot l^{(3)}(p) \end{cases}$$

It implies that CeNN computing model requires real-time weight updates when a nonlinear function is involved in a given dynamic system. As shown in equation (10), the nonlinear template having α has to be recomputed with respect to the current cell state $\varphi(x,y,t)$. By providing c_0, c_1 and c_2 data to CeNN hardware platform, the state template $\hat{\mathbf{A}}$ can be updated with specialized hardware computing new α value. Note that $c_0 \sim c_3$ can be pre-computed with a given $l(\cdot)$ and stored as a look-up table (LUT) in off-chip memory. The degree of the polynomial function determines the accuracy of approximation.

3 OPERATION OF DE SOLVER

Before going into architecture details, the basic operation of the proposed CeNN-based differential equation (DE) solver is explained.

Set parameters: Fig. 3 illustrates the entire process of solving a coupled dynamical system described by the well-known reaction-diffusion (RD) equation. For a given equation, one can extract the required number of layers for the DE solver (refer to Section 2). As RD equation has two variables involved, $\mathbf{u} \in \mathbb{R}^2$ and $\mathbf{v} \in \mathbb{R}^2$, a two-layer CeNN model is used for simulating it. There is a nonlinear function involved in updating the activator $\mathbf{u}$, but only linear term is present when updating the inhibitor $\mathbf{v}$. The nonlinear function is

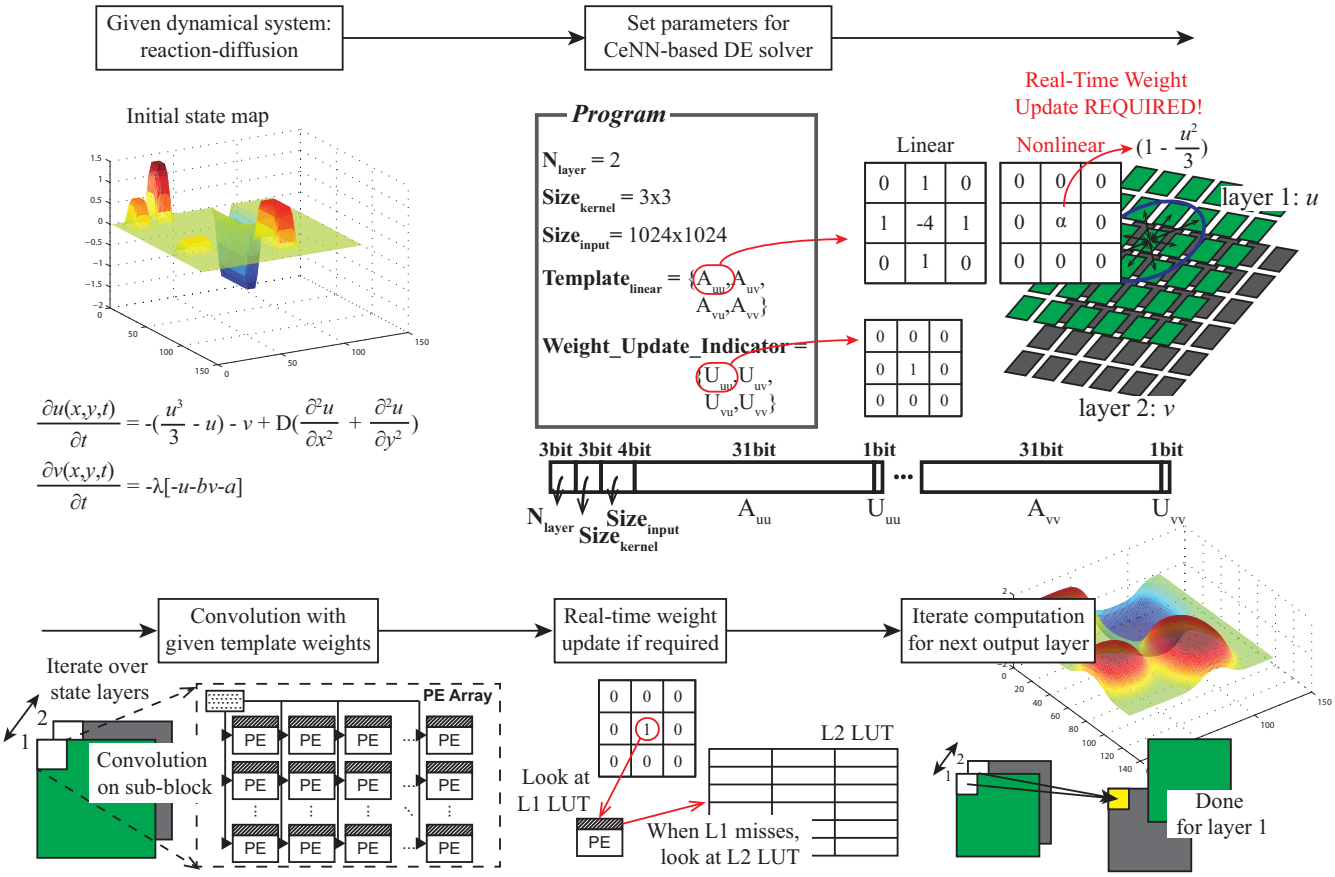

Figure 3: The overall operation of the proposed CeNN-based differential equation (DE) solver. With a given dynamical system, template weights and other parameters are set to program the DE solver (binary bit stream is used to program). Then, a sub-block in each state map (each layer) is fed into PE array to perform convolution on them. When real-time weight update is required, it looks at LUTs at different levels.

programmed to the DE solver by having state template $\hat{A}_{uu}$ (self-feedback template for layer **u**) with *real-time update* as the function depends on $\mathbf{u}(t)$.

As such a nonlinear function can be any function depending on the system of interest, we utilize LUTs to store sampled function values and coefficients of Taylor series to compute function values between sampled points (refer to Section 4.1 for details). By looking up LUTs, we can update the template weights at each cycle *if required*. Thus, we need an indicator to let PEs identify an equation that needs the real-time weight update. This update indicator flag is appended in the template data as shown in Fig. 3 (U_{uu} appended to A_{uu} making the total data bit-width 32bit).

Program DE solver: Accordingly, there are several parameters to be loaded to program the DE solver. They are input size (**Size$_{\text{input}}$**), kernel size (**Size$_{\text{kernel}}$**), number of layers (**N$_{\text{layer}}$**), space-invariant and linear templates (**Template$_{\text{linear}}$**), and binary indicator matrices for real-time weight update (**WUI**). This can be programmed by using a bit stream pushed to the DE solver. The bit size representing each parameter bounds the maximum value of that parameter. For instance, if 3 bits are used to represent N_{layer}, then a coupled dynamical system with up to 8 layers (equivalently, 8 equations) can be solved. The size of kernel or input is programmed by providing the side length; to program 3×3 kernel, 3 is given in the bit sequence. For the input size, the side length is constrained to be the power of 2. Thus, the exponent is programmed in the bit sequence; $1010_{(2)}$ to program 1024×1024 input size.

Other than state template ($\hat{\mathbf{A}}$), feedforward template (**B**) and offset (z) are also provided to the solver. They are appended to the bit sequence as well which adds another 148-Byte data at the end. For most cases, **B** and z do not require real-time update thus no weight update indicator is needed. In summary, a set of templates can be considered as a program for the DE solver to simulate a specific dynamical system.

Computation: After the DE solver is programmed, a PE array performs convolution on a sub-block in the state map of single variable (e.g. **u**). At each clock cycle, multiplication between the weight and the state value within a convolution is performed. For 3×3 convolution with 8×8 PEs, it takes nine cycles to complete convolutions for 64 cells of one state variable. Then, the PE array moves to a

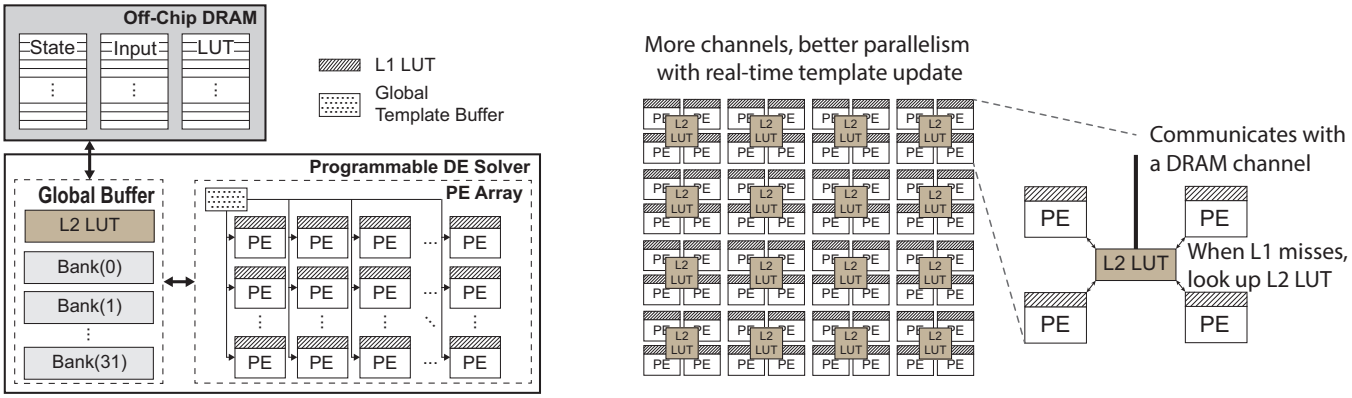

Figure 4: The overall architecture of CeNN-based DE solver. Look-up Tables (LUTs) allow nonlinear and real-time weight updates with complex functions.

sub-block at the same position but for the next variable (equivalently, layer) to be computed. As RD equation has two variables, convolution is done over 8×8 sub-blocks for two layers to get the updated sub-block of one layer. Then, the result is written back to off-chip memory for the computation in the next cycle.

Real-time weight update: During the multiplication, each PE checks *WUI* bit in the weight data and if the computation requires weight update it looks at a local (L1) LUT. If it misses, it looks at higher level (L2) LUT while setting PEs in idle mode. By having the intermediate level of shared LUT between off-chip memory and local LUT, we reduce the number of expensive accesses to DRAM. By retrieving the function value for the given state, PE becomes active again and completes the multiplication at next clock cycle. After the convolution for one output layer (update on one variable) is done, the computation moves to next layer to update next state variable. For RD equation, it needs to update two (output) variables; **u** and **v**.

LUT for [$l(x) = 0.5x^2 - 0.012x^3 + \sin(x)$]

111 95	64 63	48 47	32 31	16 15	0
-1.0					
0.0					
1.0	1.3295	-0.5057	1.9234	-0.6123	-0.8055
2.0	2.8133	5.7828	-2.1182	0.3441	-5.8461
3.0	4.3171	22.615	-7.6191	0.9180	-24.058
4.0	6.4752	18.939	-5.5109	0.5816	-24.806
5.0	10.041	-38.280	6.9339	-0.3557	62.513
6.0	15.129	-115.98	19.427	-1.0322	219.47
7.0					
p	l(p)	c0	c1	c2	c3

can vary the degree of expansion by trading-off throughput/power with accuracy

Figure 5: The data format stored in off-chip LUT of an exemplary nonlinear function required for real-time weight update.

4 SYSTEM ARCHITECTURE

Fig. 4 shows the overall architecture of the proposed generic DE solver. The system architecture is composed of memory system, the real-time template update, and the processing engine cluster.

4.1 Real-Time Template Weight Update

We propose a memory-centric approach using a hierarchy of look-up-tables (LUT) for real-time template update. We create an LUT to store the nonlinear function using Taylor-series expansion around different values as illustrated in Fig. 5. We store finite number of exact values for $l(x)$ in equation (9), i.e. $l(p)$ at points 'p', expand $l(x)$ in Taylor series around 'p', and store the coefficients ($c_0 \sim c_3$). Therefore, stored data corresponding to a point p is a tuple D_{LUT} = $\{l(p), c_0, c_1, c_2, c_3 - l(p)\}$. Note, as $l(p)$ is stored, we store c_3 in equation (10) without the term $l(p)$.

We propose a two-level cache hierarchy to manage the LUTs. The full LUTs are stored in the main memory. A part of the data is stored in a shared L2 LUT (one per memory channel of a chip), and multiple distributed L1 LUTs (one per processing engine). The performance of the solver depends on miss rates from the LUTs (mr_{L1} or mr_{L2}). As the LUT access is determined by the state of the CeNN cell, the miss rate depends on the size of on-chip LUTs and the distribution of states in the CeNN model.

As the number of LUT blocks is small in L1, the index is directly matched, multi-bit XNOR operation between higher 16 bits among 32bit of cell state and index in L1 LUT, to find the required value. For L2 LUT, as the size is much larger, direct matching is impossible. Therefore, a hash function utilizing modulo is being used as search index. The modulo by power-of-2 is used as the size of L2 LUT is 2^N and it is simple to design as hardware.

When there is miss at L1 LUT, required data is copied (also fetched to PE at the same time) from L2 LUT if the data is present. The L1 LUT has write pointer which increments by one (cyclic) to provide write address whenever L1 LUT misses. When L2 LUT misses, expensive DRAM access happens thus we get multiple data points whenever it happens. In the proposed DE solver, it fetches eight data points whenever L2 LUT misses. For instance, if data for p = 3.0 was required in Fig. 5 and both on-chip LUTs missed then the solver fetches data from p = 0.0 to p = 7.0 in DRAM. When

storing these data to L2 LUT, the same hash function is used to synchronize the data address with read operation.

After the function value $l(p)$ at point 'p' and coefficients (c_0, c_1, c_2, c_3) are loaded, each PE needs to decide whether to directly use the exact $l(p)$ or approximate function value using coefficients. This can be checked by looking at lower 16bit of state data as the off-chip LUT contains exact $l(p)$ with p represented by higher 16bit. Assume we have 32bit state (fixed-point) where the first half bits are integer and the rest are fractional. Then, 'p' will be an integer number for look-up index in LUTs. If the fractional part is non-zero, containing at least one high(1) bit in lower 16bit, we need to approximate a nonlinear template by computing α in equation (10). This is done by a specific hardware called, Template Update Module (TUM), attached to each PE (Fig. 6).

4.2 Storage of States, Inputs, and Templates

The state x, input u, template weights ($\hat{\mathbf{A}}, \mathbf{B}$), and values of nonlinear functions are stored in the off-chip memory. There are 32 data banks in our system and the half of them are dedicated to state x, the rest to input u of CeNN model. The data banks for each data type (x or u) are grouped into two (*primary* and *support*).

For each main memory channel, we consider a global buffer consisting of data banks, and one L2 LUT cache for nonlinear template update (Fig. 4). The off-chip memory communicates with the on-chip global buffer, and the global buffer pushes required data to a processing engine (PE) array which consists of a template buffer, L1 LUT cache for template updates, and PEs.

In the PE array, a template buffer is placed to store and broadcast template weight for the current convolution computation. The sequencing over this template decides which equation that PE array is currently solving (refer to Section 5). The data size filled in the template buffer is $N_{layer}^2 \cdot l_{kernel}^2$ for each template type (either feedback or feedforward); l_{kernel} is the length of a template. The finite state machine is used to address the template weight for each convolution operation.

Whenever a real-time weight update is required as a function of the current cell state, each PE searches for the value of that function or coefficients to perform series expansion from its local L1 LUT. If the required data is present, a PE computes its convolution without waiting. If not, the PE will wait for extra clock cycles to search the data from the connected L2 LUT.

4.3 Processing Engine Architecture

The detailed block diagram of PE array is shown in Fig. 7. There are N_{layer}^2 convolution templates in DE solver to handle and each convolution template has the size of $Size_{kernel}$. Each template weight is prefetched from shared template buffer by having two counters; one for layer indexing and the other for convolution indexing. Then, PEs start to compute convolution with the given weight and convolution index. By having proper data controller and muxes, the dedicated dataflow mode is used to move around the data within PEs as well as from data banks. The backup register in Fig. 7 helps PE promptly retrieve data when there is row change in template (refer to mode 2 in Fig. 10). The data prior to the horizontal shift is retrieved to move data to the upper PE of each PE (via x_V or u_V path). For mode 1 and mode 3, the data is moved to left within PE array; x_H or u_H path is for this purpose. This PE structure is simpler than the one presented in [11] as our PE only requires single-word register for state and input (no FIFO buffers needed).

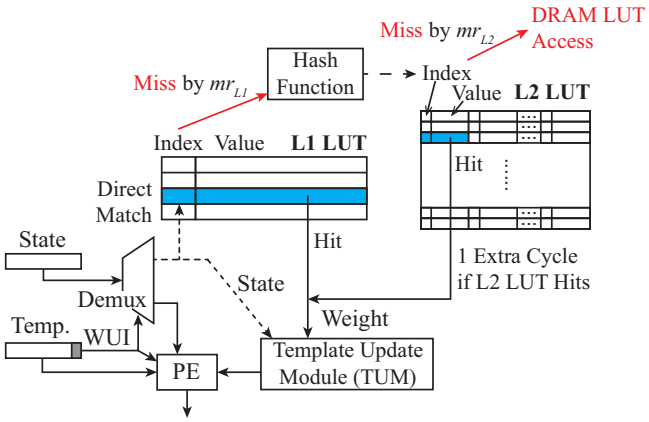

Figure 6: The operation of real-time weight update to compute state nonlinear template $\hat{\mathbf{A}}$.

In the following section, the proper dataflow method is explored suited for the proposed DE solver with real-time weight update. Also, the data movement in the memory system is explained in detail.

5 DATAFLOW IN PROPOSED DE SOLVER

5.1 Exploration of Different Dataflow Schemes

The main operation of CeNN is convolution between cell state (or external input) and programmed template kernel $\hat{\mathbf{A}}$ (or $\mathbf{B}$). Thanks to the improvement of convolutional neural network, various dataflow schemes exploiting data reuse are proposed and evaluated; optimized for the convolution computation [4, 6, 11, 14, 34, 45]. As CeNN-based DE solver requires the real-time weight update, the dataflow scheme that maximizes the throughput differs from the previous analyses.

In [6], various dataflow schemes are summarized and compared. The dataflow scheme can be grouped into four categories: no local reuse (NLR [45]), weight stationary (WS [4]), output stationary (OS [11, 14, 34]), and row stationary (RS [6]). According to the analysis in [6], RS dataflow showed the best energy-delay product by minimizing DRAM access and maximizing data reuse within memory units in lower hierarchy. However, one important thing to note is that other than OS dataflow different kernel (template) weights are given to PEs for convolution operation. In OS dataflow, one template weight is shared by all PEs in the processing array (refer to Fig. 8).

As shown in equation (4), some equations have a nonlinear function involved which is a function of the current state. Thus, a real-time weight update by accessing on-chip/off-chip LUTs is often needed for differential equation solving. By fetching coefficients from the LUT, we are able to update the template with complex functions. Depending on the size of on-chip LUTs, miss rate will vary. Whenever there is a LUT miss, DRAM needs to be accessed which is expensive in terms of delay and energy consumption.

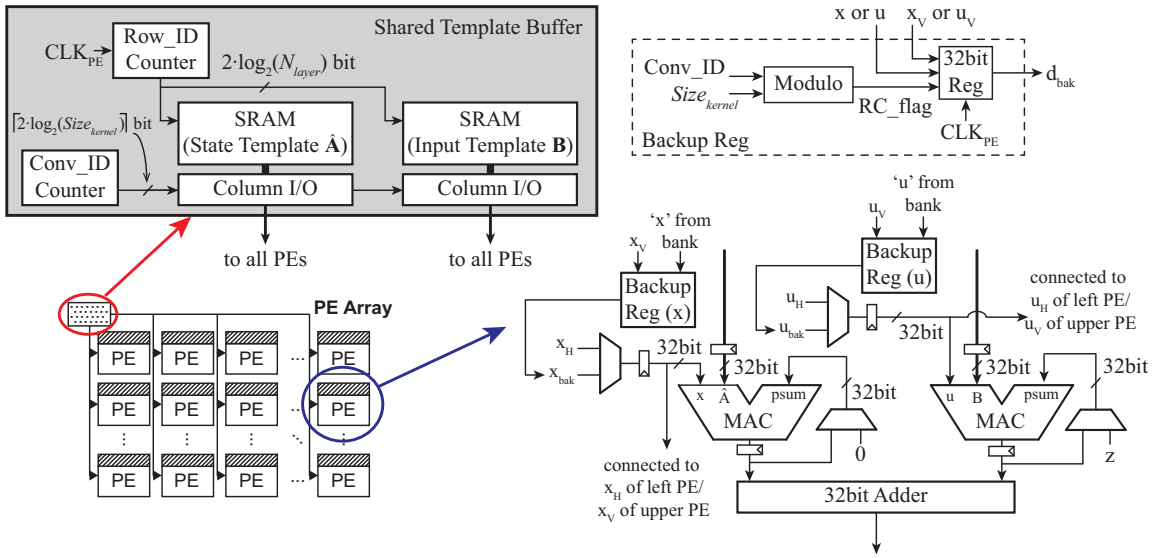

Figure 7: The detailed circuit block diagram of a global template buffer and a processing element.

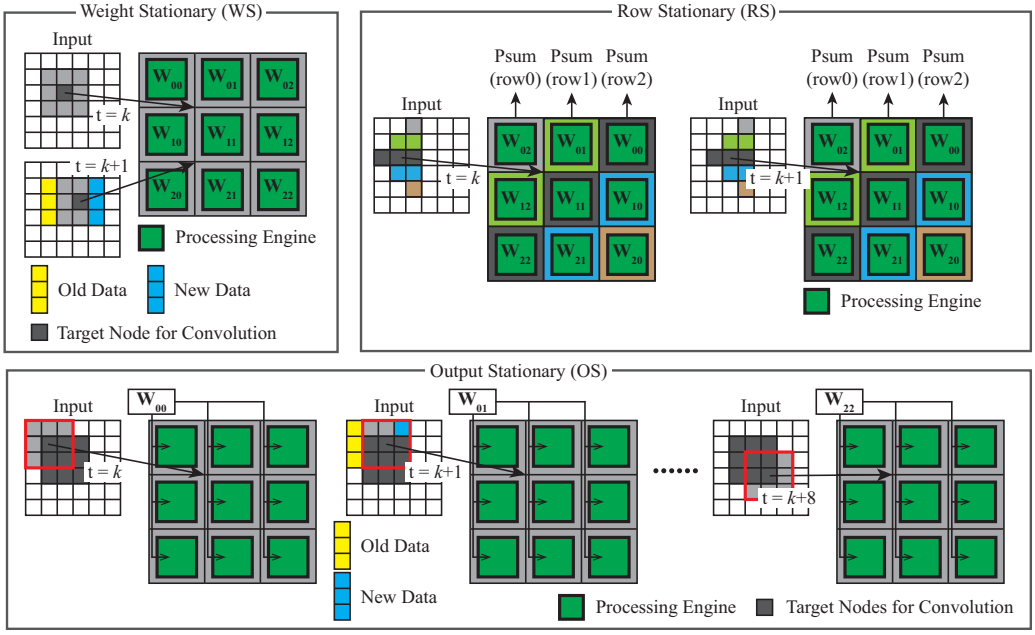

Figure 8: The comparison between different dataflow schemes for consecutive time steps with 3×3 template (kernel). Other than OS dataflow, entire template weights are used during the convolution operation.

For dataflows other than OS dataflow, DRAM will be accessed at a clock cycle when at least one weight value in the template requires the update and on-chip LUT misses. Then, the number of DRAM access for the proposed DE solver becomes

$$\#DRAM_{access} = (mr_{L1} \cdot mr_{L2}) \cdot Size_{input} \cdot N(\mathbf{U}_{ll^*} \neq \mathbf{0}) \quad (11)$$

where mr is the LUT miss rate and $N(\mathbf{U}_{ll^*} \neq \mathbf{0})$ is the number of templates requires weight update. For the example shown in Fig. 3, it requires 100K DRAM accesses if $(mr_{L1} \cdot mr_{L2}) = 0.1$ and $N(\mathbf{U}_{ll^*} \neq \mathbf{0}) = 1$.

By using OS dataflow, the number of DRAM access for real-time weight update is

$$\#DRAM_{access} = \frac{(mr_{L1} \cdot mr_{L2}) \cdot Size_{input} \cdot N(\mathbf{U}_{ll^*} \neq \mathbf{0})}{\#PEs}. \quad (12)$$

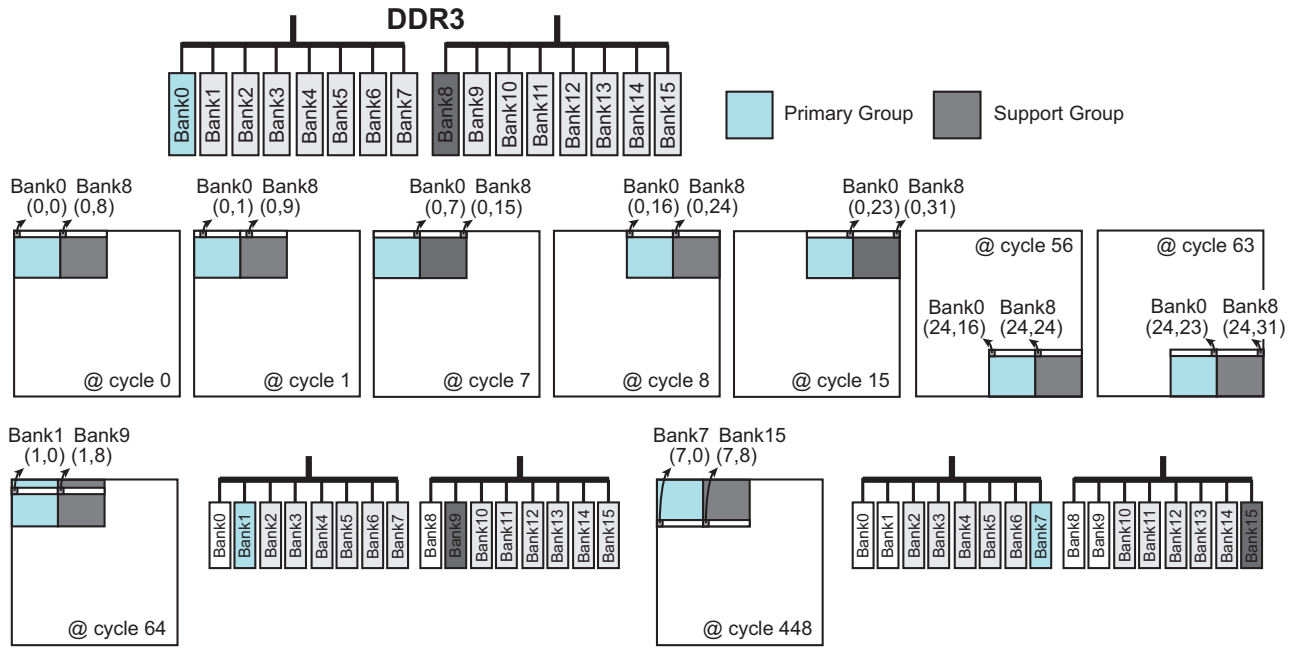

Figure 9: Data prefetching from off-chip DRAM to on-chip global buffer for 32×32 input. There are two bank groups (primary and support) to utilize intra-PE data transfer.

This is due to the weight sharing among all PEs, thus DRAM is accessed at that specific cycle when weight update is required. Then, the estimated number of DRAM access becomes 1.6K for the same scenario ($\#PEs\times$ less).

This analysis shows that OS dataflow is better than other dataflow schemes for the convolution where template needs to be updated over time. As CeNN state evolves over time, the advantage of utilizing OS dataflow piles up.

5.2 OS Dataflow in Proposed DE Solver

Before start computing, required data should be pushed from DRAM to the on-chip global buffer (Fig. 9). We have 16 state banks and 16 input banks in the global buffer for 8×8 PEs in the system. Among 16 data banks, 8 banks are in the primary group and the remaining banks are in the support group. This is to utilize intra-PE data transfer to reduce data delivery energy from banks to local registers in the PE array.

As the process of input data (u_{kl}) prefetching is identical to state data (x_{kl}) prefetching, only the dataflow for states of entire layers in CeNN model is discussed. Each row block in a data bank stores $nPE_x = 8$ words where there is a $[nPE_y \times nPE_x = 8 \times 8]$ PE array. The state map is divided into 8×8 sub-blocks where convolutions for those cells are handled by PE array at a time. Each bank in the primary group stores data for a row in the processing sub-block; bank $(k-1)$ has data for the k^{th} row in each sub-block. The support group stores state data in an interleaved fashion as shown in Fig. 9. Assuming external memory as DDR3 (2 channels), each channel handles data prefetching for each bank type (primary or support).

After the state and the input data are prefetched to the global buffer, template weights are pushed to the template buffer in PE array. The total number of weight data to be prefetched is $N^2_{layer} \times l^2_{kernel}$ (36 for the example in Fig. 3). After all data is prepared for DE solver, the convolution operation begins to solve a differential equation. There are four modes during the convolution and they are shown in Fig. 10. A dataflow mode is selected depending on which convolution operation is currently being handled by PE array:

(1) Mode 0: if $(conv\_id = 0)$
(2) Mode 1: if $(0 < conv\_id < l_{kernel})$
(3) Mode 2: if $(\lfloor \frac{conv\_id}{l_{kernel}} \rfloor > 0, mod(conv\_id, l_{kernel}) = 0)$
(4) Mode 3: if $(\lfloor \frac{conv\_id}{l_{kernel}} \rfloor > 0, mod(conv\_id, l_{kernel}) > 0)$

The proper mode selection for the convolution with 3×3 template (kernel) is shown in Fig. 10.

As we have 64 PEs in the system (Fig. 4), 64 convolutions with 3×3 template is completed in 9 clock cycles (w.r.t. CLK_{PE}) when there is no weight update. If there is the need for real-time weight update for any element in the template, then the required clock cycles increases accordingly.

6 SYSTEM ANALYSIS OF DE SOLVER

In this section, we present the performance and energy-efficiency of the proposed architecture with CeNN computing model when solving wide classes of differential equations.

6.1 Benchmark Differential Equations

To evaluate the efficiency of the proposed CeNN-based DE solver, we selected six differential equations describing various dynamical systems. For the simplest benchmark, heat diffusion is selected as it is described by single PDE in which only linear template is

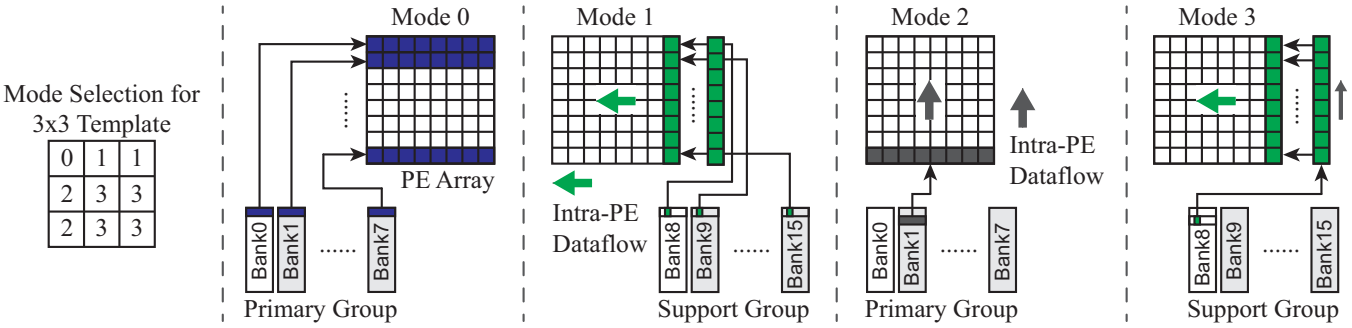

Figure 10: Dataflow modes from data banks to PE array during convolution operation. A proper mode selection for the convolution operation with 3×3 template is shown as an example.

	Heat	Fisher's	RD
Mean(error)	$4.09e^{-7}$	$8.01e^{-8}$	$6.48e^{-5}$
σ(error)	$1.03e^{-7}$	$3.51e^{-8}$	$5.55e^{-5}$

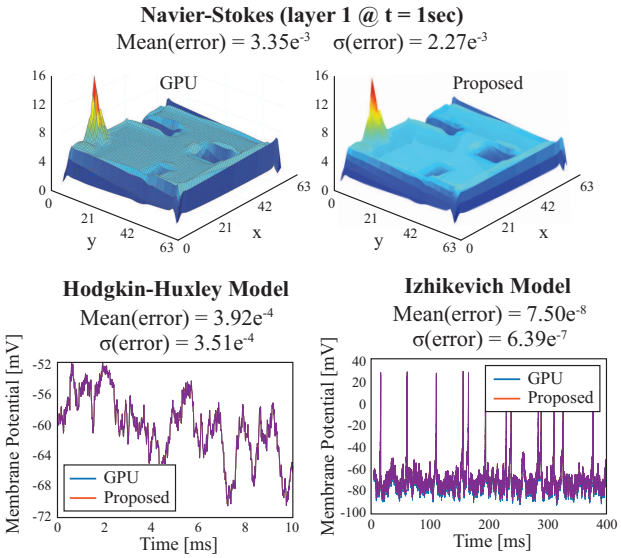

Figure 11: The accuracy comparison between GPU (32bit floating-point) and CeNN-based solver (32bit fixed-point).

present. The Navier-Stokes equation is used to simulate single PDE with nonlinear template. To simulate coupled systems, Fisher's and reaction-diffusion equations are selected. Among these, the RD equation can be used as another set of computing model, which is capable of simulating Turing machine [1]. Also, physical behaviors of cortical neurons can be modeled by coupled differential equations. They are called Hodgkin-Huxley (HH) model [15] and Izhikevich model [18]. These spiking models are candidates for a basic unit in neuromorphic computing engines. Thus, the CeNN-based DE solver not only finds the solution for a given differential equation, but it also accelerates the simulation of other computing models for faster development.

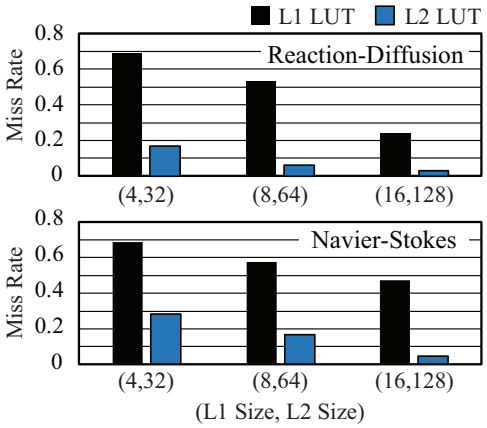

Figure 12: The miss rate depending on the size of on-chip LUTs for two different dynamical systems.

When designing the DE solver, the bit-precision of a system has to be carefully determined. If the system is designed to compute with floating-point, the accuracy will be high but the energy-efficiency degrades compared to fixed-point counterparts. For the power-constrained environment, such as mobile platforms or robotic controllers, the DE solver should perform fixed-point computation. Also, it becomes possible to run massive simulations with different conditions in parallel by utilizing mutiple (energy-efficient) DE solvers in finding a number of solutions to obtain near-optimal solution for a complex and large problem.

For six benchmark equations, we compared solutions with GPU (32bit floating-point) and the proposed DE solver (32bit fixed-point). The results on different dynamical systems are summarized in Fig. 11. The average and standard deviation of absolute error for three benchmarks (Navier-Stokes, HH model, and Izhikevich model) are shown with the solution of each example. The error values for other differential equations are also provided as a table in Fig. 11. For spiking models, spikes were well-matched with the GPU simulation. If we breakdown the error, the fixed-point computation always adds error. The LUT approximation error is negligible for linear (or low-order polynomial) interactions, but dominates total error for scientific functions (exp, sin, cos, tanh, ...). For example, in HH simulation

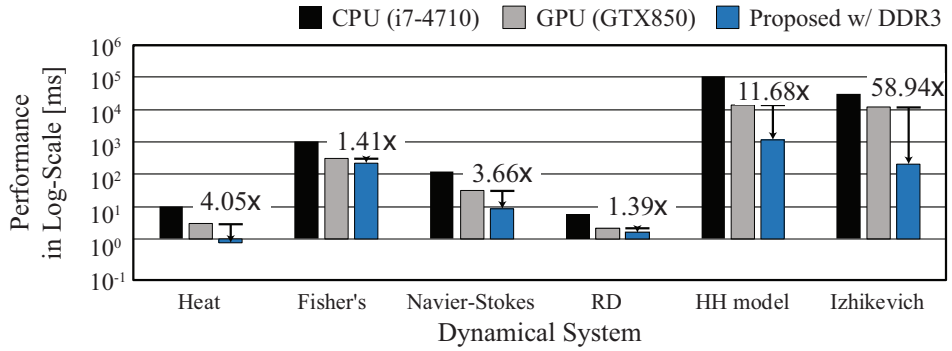

Figure 13: The performance comparison on six different benchmark differential equations. The speed-up using the proposed CeNN-based solver with DDR3 is shown compared to the performance using GPU (GTX 850).

|fixed_point_error| is $\sim$1.2e^{-7} while |LUT_error| can vary from $\sim$7.9e^{-8} to $\sim$5.4e^{-4}.

6.2 Miss Rate Analysis for Weight Update

As mentioned before, there exists trade-off between on-chip LUT size and miss rate. Several design choices are simulated and the results are shown in Fig. 12. They are simulated with two representative differential equations; one is reaction-diffusion and the other is Navier-Stokes equation. For both cases, miss rate of L1 LUT when there is only four blocks in the cache is about 0.7 which is quite high. The miss rate reduces by increasing the capacity, but with the support of larger L2 LUT the miss rate drops significantly (to 0.15-0.3) with one extra cycle. As the energy-efficiency is also crucial in some applications, we select the size of L1 and L2 LUT to 4 blocks and 32 blocks for the following simulations.

6.3 Performance Comparison

According to the analysis in Section 6.1, we designed our DE solver computing with 32bit fixed-point. To simulate the performance, we developed a cycle-level simulator including all architecture details explained in Section 4. The simulator takes parameters in Fig. 3 with a configuration file (memory type, $Size_{kernel}$, $Size_{input}$, N_{layer}, $Template_{linear}$, and WUI). In the simulator, memory specification (bandwidth, # of channels, bus-width, latency), global buffer (# of banks, bank size, mr_{L2}), shared template buffer (buffer size), and PE array (# of PEs, latency, mr_{L1}) are parameterized. The miss rates, mr_{L1} and mr_{L2}, are extracted from Matlab simulation and fed to the simulator to consider actual state distribution of each benchmark.

The data prefetching from DRAM is performed in burst mode and burst length is assumed to be 8. Thus, the data are pushed to global buffer for eight consecutive cycles and data controller waits for t_{CCD} for another burst to happen. For the real-time weight update, when PE array gets required data from data banks, it checks WUI-bit in weight data and either computes or checks LUTs for nonlinear template. The clock cycle of PE array is $1/4$ of DRAM (or L2 LUT) clock as four PEs are connected to one L2 LUT.

By assuming DDR3 as off-chip memory, the performance comparison between CPU, GPU, and our DE solver is summarized in Fig. 13. The performance depends on the number of layers, the number of nonlinear weight updates, and the number of grid cells to be computed. About 46.48$\times$ (13.52$\times$) performance improvement over CPU (GPU) on average is achieved by using the CeNN-based DE solver with DDR3. We assumed two channels for DDR3 and this limits the performance as each channel connects to 8 L2 LUTs (Fig. 4). In the worst case, these 8 LUTs will miss and request data from DRAM forming a long request queue. As there are 16 L2 LUTs in our DE solver, the system throughput maximizes by having 16 memory channels with concurrent access.

6.4 Integration with High-bandwidth Memory

The memory-centric nature of the proposed architecture suggests that a higher bandwidth and concurrent accesses between compute and memory can enhance the system performance. As illustrated in Fig. 4, having more memory channels, so that each channel connects to individual L2 LUT, improves system throughput as the PE array can handle concurrent L1 LUT misses with better parallelism. A higher degree of parallelism also reduces the time required to push the states, inputs, and weights into the compute layer. Therefore, we have explored integration of the CeNN based computing platforms with memory systems providing concurrent high bandwidth accesses, such as Hybrid Memory Cube (HMC) [17].

We evaluate the effect of high-bandwidth memory for different benchmarks (Fig. 14). Note that HMC provides two different interfaces: one externally connects to processors (HMC-EXT) and the other internally connects to processor-in-memory (HMC-INT). The simulations show that integration with HMC significantly improves the performance compared to the one with GPU (Fig. 14). By using HMC-INT (or HMC-EXT), the performance improves by 23.67$\times$ (or 77.37$\times$) on average. As the I/O clock frequency of HMC-EXT (10GHz) is much faster than that of HMC-INT (2.5GHz), the performance improvement is higher. However, this naturally leads to higher power consumption in the memory system and the processing array.

6.5 Power Consumption of DE Solver

To validate the power-efficiency of the proposed DE solver, the PE array is designed and synthesized using 15nm FinFET technology [27]. The power consumption of global buffer (L2 LUTs + data banks) is estimated by using PCACTI [39]. The power consumption is estimated assuming HMC-INT as an off-chip memory. As the maximum

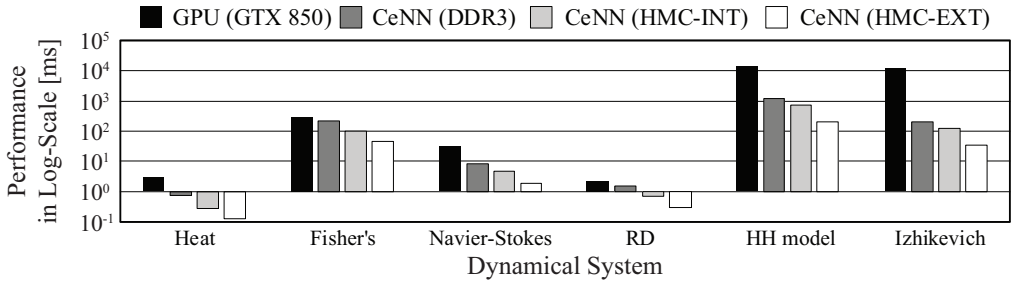

Figure 14: The performance improvement by using 3D memory stack with higher memory bandwidth.

Table 1: The power consumption of modules in PE array having 64 (PEs-L1 LUT) pairs.

PE Array		Power (mW)	Area (mm^2)
PE	TUM	1.20	0.00308
	ALU	1.12	0.00287
	TUM+ALU	2.32	0.00594
PEs		148.48	0.380
L1 LUTs		51.20	0.0698

Table 2: The overall power consumption including PE array and global buffer (data banks + shared template buffer).

System	Power (mW)	Area (mm^2)
PE Array	199.68	0.450
L2 LUT	63.61	0.00627
Global Buffer	260.16	0.625
Total	523.45	1.082

I/O clock frequency of HMC-INT is 2.5GHz, we synthesized the PE array to operate at 600MHz clock frequency. The L2 LUT runs at the same frequency as DRAM to maximize PE utilization at the worst case. The read operation of data banks or the shared template buffer runs at 600MHz which is synchronized to PE operation.

A PE array contains 64 PEs with 64 L1 LUTs. Each PE includes two backup registers, two MACs, one adder, and control logics (Fig. 7). This is denoted as ALU in Table 1. The template update module (TUM) is attached to ALU to form a PE. Table 1 shows the power estimation of PE array: 64 (PE-L1 LUT) pairs. The power consumption of 16 L2 LUTs or 32 banks is summarized in Table 2. Each L2 LUT has 16 lines of 64 Byte data (1KB); each line contains four look-up data. The global buffer has state data banks, input data banks, and shared template buffer where total size is about 2MB. As a result, the power consumption of the proposed DE solver (*on-chip components*) becomes $\sim$523mW and the area becomes $\sim 1.1 mm^2$. To compare the system power with GPU, the external memory power should be added to the on-chip system power reported in Table 2. The memory power depends on energy/bit and application-dependent activity ratio for DRAM accesses. For example, considering Izhikevich simulation (activity ratio = 0.22) the memory power estimates to $\sim$1.04W for HMC-Int (3.7pJ/bit) [19]. Then, the total system power of the proposed DE solver becomes 1.56W which is 32$\times$ less than GPU (40$\sim$50W).

7 RELATED WORK

There has been research focus on developing hardware platforms for implementing CeNN model [20, 23, 35, 36, 40, 44]. They are categorized by underlying circuit type: analog/ mixed-signal, FPGA, or fully digital platforms (Table 3). The analog CeNN hardware is fast and energy-efficient but it lacks accuracy (only 8bit precision), scalability, and programmability [20, 35, 36, 40]. There are emulated digital [44] and fully digital platforms [23] which give some level of programmability with higher accuracy. However, these works are only capable of dealing with linear time-invariant templates. Some emulated digital platforms provide specialized hardware units to handle nonlinear templates [21, 30]. Yet, they fail to provide architecture support to cover more general functions other than polynomial.

Our CeNN-based DE solver significantly improves the programmability compared to any previous platforms with proper architecture for real-time template update. This versatility is achieved along with high efficiency ($\sim$103 GOPS/W). Accordingly, the proposed DE solver will enable the efficient simulation of 'computing with dynamical systems'.

8 CONCLUSION

This paper presents the programmable differential equation (DE) solver based on CeNN computing model. The generic mapping of wide classes of dynamical systems to CeNN model is presented along with the architecture support to accelerate the computation. The proposed DE solver includes a spatial PE array with LUTs for real-time weight update which enables handling nonlinear functions involved in differential equations. This real-time weight update is the important feature to make the solver truly programmable. The performance significantly improves by $\sim$24$\times$ to $\sim$77$\times$ on average with the use of high bandwidth memory. The energy efficiency improves by three to four orders of magnitude by using CeNN-based DE solver.

REFERENCES

[1] Stefania Bandini, Giancarlo Mauri, Giulio Pavesi, and Carla Simone. 2004. Computing with a distributed reaction-diffusion model. In *International Conference on Machines, Computations, and Universality*. Springer, 93–103.

[2] Robert Baxter, Stephen Booth, Mark Bull, Geoff Cawood, James Perry, Mark Parsons, Alan Simpson, Arthur S. Trew, Andrew McCormick, Graham Smart, and others. 2007. Maxwell-a 64 FPGA supercomputer. *Adaptive Hardware Systems* 7 (2007), 287–294.

Table 3: The comparison of the proposed DE solver to previous hardware platforms for cellular nonlinear network.

	ACE16k [36]	Q-Eye [35]	GAPU [44]	VAE [23]	This Work
Type	Analog/ mixed-signal	Analog/ mixed-signal	FPGA	Digital	Digital
Technology	0.35um	0.18um	0.15um	0.13um	15nm
# PEs	16560	25344	1024	120	64
Power (W)	4.0	0.1	10.0	0.084	0.523
Area (mm^2)	92	25	-	4.5	1
Peak GOPS	330	0.1	1.3	22	54
GOPS/W	82.50	0.1	0.13	261.90	103.26
Nonlinear Weight Update					Yes

[3] William R. D. Boyd III. 2014. *Massively parallel algorithms for method of characteristics neutral particle transport on shared memory computer architectures*. Ph.D. Dissertation. Massachusetts Institute of Technology.

[4] Lukas Cavigelli, David Gschwend, Christoph Mayer, Samuel Willi, Beat Muheim, and Luca Benini. 2015. Origami: a convolutional network accelerator. In *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*. ACM, 199–204.

[5] Jean C. Chedjou and Kyandoghere Kyamakya. 2015. A universal concept based on cellular neural networks for ultrafast and flexible solving of differential equations. *IEEE Transactions on Neural Networks and Learning Systems* 26, 4 (Apr. 2015), 749–762.

[6] Yu-Hsin Chen, Joel Emer, and Vivienne Sze. 2016. Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 367–379. https://doi.org/10.1109/ISCA.2016.40

[7] Leon O. Chua, Martin Hasler, George S. Moschytz, and Jacques Neirynck. 1995. Autonomous cellular neural networks: a unified paradigm for pattern formation and active wave propagation. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 42, 10 (Oct. 1995), 559–577. https://doi.org/10.1109/81.473564

[8] Leon O. Chua and Lin Yang. 1988. Cellular neural network: Theory. *IEEE Transactions on Circuits and Systems* 35, 10 (Oct. 1988), 1257–1272. https://doi.org/10.1109/31.7600

[9] Joni Dambre, David Verstraeten, Benjamin Schrauwen, and Serge Massar. 2012. Information Processing Capacity of Dynamical Systems. *Scientific Reports* 2 (2012), 514 EP –. http://dx.doi.org/10.1038/srep00514

[10] Alexandre C. B. Delbem, Leonardo Garcia Correa, and Liang Zhao. 2009. Design of associative memories using cellular neural networks. *Neurocomputing* 72, 10–12 (June 2009), 2180–2188.

[11] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. 2015. ShiDianNao: shifting vision processing closer to the sensor. In *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*. ACM, 92–104. https://doi.org/10.1145/2749469.2750389

[12] Frank Gollas and Ronald Tetzlaff. 2005. Modeling complex systems by reaction-diffusion cellular nonlinear networks with polynomial weight-functions. In *2005 9th International Workshop on Cellular Neural Networks and Their Applications*. IEEE, 227–231.

[13] Jessy W. Grizzle, Christine Chevallereau, Ryan W. Sinnet, and Aaron D. Ames. 2014. Models, feedback control, and open problems of 3D bipedal robotic walking. *Automatica* 50 (2014), 1955–1988.

[14] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep learning with limited numerical precision. *CoRR, abs/1502.02551* 392 (2015).

[15] Allan L. Hodgkin and Andrew F. Huxley. 1952. The dual effect of membrane potential on sodium conductance in the giant axon of Loligo. *The Journal of Physiology* 116, 4 (Apr. 1952), 497–506.

[16] Yildirim Hurmuzlu, Frank Genot, and Bernard Brogliato. 2004. Modeling, stability and control of biped robots - a general framework. *Automatica* 40 (2004), 1647–1664.

[17] Hybrid Memory Cube Consortium. 2013. Hybrid Memory Cube Specification 1.0. (2013).

[18] Eugene M. Izhikevich. 2003. Simple model of spiking neurons. *IEEE Transactions on Neural Networks* 14, 6 (Nov. 2003), 1569–1572.

[19] Joe Jeddeloh and Brent Keeth. 2012. Hybrid memory cube new DRAM architecture increases density and performance. In *2012 Symposium on VLSI Technology (VLSIT)*. 87–88. https://doi.org/10.1109/VLSIT.2012.6242474

[20] Peter Kinget and Michel S. J. Steyaert. 1995. A programmable analog cellular neural network CMOS chip for high speed image processing. *IEEE Journal of Solid-State Circuits* 30, 3 (Mar. 1995), 235–243. https://doi.org/10.1109/4.364437

[21] Sándor Kocsárdi, Zoltán Nagy, Árpád Csík, and Péter Szolgay. 2008. Two-dimensional compressible flow simulation on emulated digital CNN-UM. In *2008 11th International Workshop on Cellular Neural Networks and Their Applications*. IEEE, 169–174.

[22] Jens Krüger and Rüdiger Westermann. 2003. Linear algebra operators for GPU implementation of numerical algorithms. *ACM Transactions on Graphics (TOG)* 22, 3 (2003), 908–916.

[23] Seungjin Lee, Minsu Kim, Kwanho Kim, Joo-Young Kim, and Hoi-Jun Yoo. 2011. 24-GOPS 4.5-Digital Cellular Neural Network for Rapid Visual Attention in an Object-Recognition SoC. *IEEE Transactions on Neural Networks* 22, 1 (Jan. 2011), 64–73.

[24] Bai Li and Zhijiang Shao. 2015. A unified motion planning method for parking an autonomous vehicle in the presence of irregularly placed obstacles. *Knowledge-Based Systems* 86, C (Sept. 2015), 11–20.

[25] W. D. Little and A. C. Soudack. 1965. On the analog computer solution of first-order partial differential equations. *Mathematics and Computers in Simulation* 7, 4 (Oct. 1965), 190–194.

[26] Teppo Luukkonen. 2011. *Modelling and control of quadcopter*. Technical Report. Aalto University.

[27] Mayler Martins, Jody M. Matos, Renato P. Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. 2015. Open Cell Library in 15nm FreePDK Technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design (ISPD'15)*. ACM, New York, NY, USA, 171–178. https://doi.org/10.1145/2717764.2717783

[28] Paul C. Matthews, Renato E. Mirollo, and Steven H. Strogatz. 1991. Dynamics of a large system of coupled nonlinear oscillators. *Physica D: Nonlinear Phenomena* 52, 2–3 (Sept. 1991), 293–331.

[29] Paul A. Merolla, John V. Arthur, Rodrigo Alvarez-Icaza, Andrew S. Cassidy, Jun Sawada, Filipp Akopyan, Bryan L. Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. 2014. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science* 345, 6197 (2014), 668–673. https://doi.org/10.1126/science.1254642

[30] Zoltán Nagy, Zsolt Vörösházi, and Péter Szolgay. 2006. Emulated digital CNN-UM solution of partial differential equations. *International Journal of Circuit Theory and Applications* 34, 4 (June 2006), 445–470.

[31] Dmitri E. Nikonov, Gyorgy Csaba, Wolfgang Porod, Tadashi Shibata, Danny Voils, Dan Hammerstrom, Ian A. Young, and George I. Bourianoff. 2015. Coupled-oscillator associative memory array operation for pattern recognition. *IEEE Journal on Exploratory Solid-State Computational Devices and Circuits* 1 (Nov. 2015), 85–93.

[32] Eustace Painkras, Luis A. Plana, Jim Garside, Steve Temple, Francesco Galluppi, Cameron Patterson, David R. Lester, Andrew D. Brown, and Steve B. Furber. 2013. SpiNNaker: A 1-W 18-Core System-on-Chip for Massively-Parallel Neural Network Simulation. *IEEE Journal of Solid-State Circuits* 48, 8 (Aug. 2013), 1943–1953. https://doi.org/10.1109/JSSC.2013.2259038

[33] Abhinav Parihar, Nikhil Shukla, Suman Datta, and Arijit Raychowdhury. 2016. Computing with dynamical systems in the post-CMOS era. In *2016 IEEE Photonics Society Summer Topical Meeting Series (SUM)*. 110–111. https://doi.org/10.1109/PHOSST.2016.7548777

[34] Maurice Peemen, Arnaud A. Setio, Bart Mesman, and Henk Corporaal. 2013. Memory-centric accelerator design for convolutional neural networks. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE, 13–19.

[35] Ángel Rodríguez-Vázquez, Rafael Domínguez-Castro, Francisco Jiménez-Garrido, Sergio Morillas, Juan Listán, Luis Alba, Cayetana Utrera, Servando Espejo, and Rafael Romay. 2008. The Eye-RIS CMOS vision system. In *Analog Circuit Design*. Springer, 15–32.

[36] Angel Rodríguez-Vázquez, Gustavo Liñán-Cembrano, L. Carranza, Elisenda Roca-Moreno, Ricardo Carmona-Galán, Francisco Jiménez-Garrido, Rafael Domínguez-Castro, and Servando E. Meana. 2004. ACE16k: the third generation of mixed-signal SIMD-CNN ACE chips toward VSoCs. *IEEE Transactions on Circuits and Systems I: Regular Papers* 51, 5 (May 2004), 851–863.

[37] Tamás Roska, Leon O. Chua, Dietrich Wolf, Tibor Kozek, Ronald Tetzlaff, and Frank Puffer. 1995. Simulating nonlinear waves and partial differential equations via CNN–part I: basic techniques. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 42, 10 (Oct 1995), 807–815. https://doi.org/10.1109/81.473590

[38] Fred Rothganger, Conrad D. James, and James B. Aimone. 2016. Computing with dynamical systems. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*. 1–3. https://doi.org/10.1109/ICRC.2016.7738701

[39] Alireza Shafaei, Yanzhi Wang, Xue Lin, and Massoud Pedram. 2014. Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices. In *2014 IEEE Computer Society Annual Symposium on VLSI*. IEEE, 290–295.

[40] Bertram E. Shi and Tao Luo. 2004. Spatial pattern formation via reaction-diffusion dynamics in 32×32×4 CNN chip. *IEEE Transactions on Circuits and Systems I: Regular Papers* 51, 5 (2004), 939–947. https://doi.org/10.1109/TCSI.2004.827628

[41] Nikhil Shukla, Abhinav Parihar, Matthew Cotter, Michael Barth, Xueqing Li, Nandhini Chandramoorthy, Hanjong Paik, Darrell G. Schlom, Vijaykrishnan Narayanan, Arijit Raychowdhury, and Suman Datta. 2014. Pairwise coupled hybrid vanadium dioxide-MOSFET (HVFET) oscillators for non-boolean associative computing. In *IEEE International Electron Devices Meeting*. 28.7.1–28.7.4. https://doi.org/10.1109/IEDM.2014.7047129

[42] Angela Slavova and Pietro Zecca. 2007. Complex behavior of polynomial FitzHugh–Nagumo cellular neural network model. *Nonlinear Analysis: Real World Applications* 8, 4 (Sept. 2007), 1331–1340.

[43] Olaf Storaasli. 2003. Computing faster without CPUs: scientific applications on a reconfigurable, FPGA-based hypercomputer. In *6th Military and Aerospace Programmable Logic Devices (MAPLD) Conference*.

[44] Zsolt Vörösházi, András Kiss, Zoltán Nagy, and Péter Szolgay. 2008. Implementation of embedded emulated-digital CNN-UM global analogic programming unit on FPGA and its application. *International Journal of Circuit Theory and Applications* 36, 5-6 (June 2008), 589–603.

[45] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*. ACM, 161–170.

[46] Ling Zhuo and Viktor K. Prasanna. 2005. High performance linear algebra operations on reconfigurable systems. In *Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*. IEEE Computer Society, 1–12.

[47] Bachar Zineddin, Zidong Wang, and Xiaohui Liu. 2011. Cellular neural networks, the Navier–Stokes equation, and microarray image reconstruction. *IEEE Transactions on Image Processing* 20, 11 (Nov. 2011), 3296–3301. https://doi.org/10.1109/TIP.2011.2159231