NIM: an HMC-based Machine for Neuron Computation

Geraldo F. Oliveira[1], Paulo C. Santos[1], Marco A. Z. Alves[2], and Luigi Carro[1]

[1] Informatics Institute – Federal University of Rio Grande do Sul
Porto Alegre, Brazil
{gfojunior, pcssjunior, carro}@inf.ufrgs.br
[2] Department of Informatics – Federal University of Paraná
Curitiba, Brazil
mazalves@inf.ufpr.br

Abstract. Neuron Network simulation has arrived as a methodology to help one solve computational problems by mirroring behavior. However, to achieve consistent simulation results, large sets of workloads need to be evaluated. In this work, we present a neural in-memory simulator capable of executing deep learning applications inside 3D-stacked memories. With the reduction of data movement and by including a simple accelerator layer near to memory, our system was able to overperform traditional multi-core devices, while reducing overall system energy consumption.

Keywords: Processing in Memory; Near-data processing; Neuron Simulator; Neural Networks; Hybrid Memory Cube; Vector operations;

1 Introduction

Neuron simulation has become a popular tool used to try to reproduce human brain's behavior, and a resource used to solve problems that require a learning capability from the system. For a given neuron in a Neural Network (NN), its Natural Time Step (NTS) defines the maximum time it has to read data from its neighbors, operate over input data, and output the resulted computation to subsequent neurons. Currently, the NTS for an Inferior-Olivary Nucleus (ION) neural arrangement is 50us [1]. To keep up with system constraints, today neural simulators aim to explore available application parallelism by using HPC devices, usually composed of a mix of multi-core processors [2], GPU devices [3], and accelerator units based on FPGAs [4]. However, those setting are highly expensive and not energy efficient. A significant part of system energy consumption comes from data movement throughout the whole system [5]. For a neuron, data from its neighbors travel throughout the entire memory system until it gets to the computational target core. Therefore, a neuron simulation system presents a small rate of memory reuse, since only data from a single layer would be useful for other neurons. This almost data-streaming behavior, intrinsic of neuron simulators, motives moving computational resources closer to the memory system.

Processing-in-Memory (PIM) aims to reduce system consumed energy and improve performance by including computational units inside or close to memory elements [6]. Several commercial 3D-stacked memories are available in the market nowadays, as Hybrid Memory Cube (HMC) [7], and High Bandwidth Memory (HBM) [8]. We have chosen to work with HMC memory because it has a concise public documentation, and also because it is technologically independent of any DRAM implementation. In the latest HMC specification [7], one device is composed of four high-speed serial links, a logic layer of 32 vault controllers, and four layers of DRAM memories connected via TSV through the vault controller. A single HMC device can provide a total bandwidth up to 320 GB/s.

In this work, we proposed a PIM reconfigurable accelerator implemented inside a HMC that can simulate biologically meaningful neural networks of considerable size. We highlight two distinct neuron's model, one proposed by Hodgkin-Huxley [9], and another by Izhikevich et. al. [10], since both works present a complete and well accepted neural model, yet being different in structure and complexity. The Neuron In-Memory (NIM) mechanism presented is capable of simulating up to 12288 neurons inside the NTS of 50us.

2 NIM: A Neuron in Memory Approach

In a generic NN architecture, each network layer is composed of several neurons, which are connected throughout a fixed number of layers. In each layer, a given neuron receives data from previous layers, and potentially from the external world. This structure exposes both the available parallelism between neurons from a single layer, as also the computational demand required for simulating about the number of neurons per layer. One can notice that all neuron's input parameters can be arranged in a vector structure, positioning each neuron parameter in sequential order. This arrangement enables to execute vector operations over NN data. Also, the vector structure can be exploited directly by HMC devices, both by taking advantage of its internal parallelism, as also by implementing a PIM module, which can provide acceleration to NN applications.

Figure 1 shows in black boxes our mechanism distributed among HMC vaults. Our work is based on the device presented in [11], which implements an HMC accelerator capable of vector operations over up to 8KB chunks of data, and it can also be reconfigured to work with different ranges of data as the work proposed by [12]. However, due to the particularity of NNs applications, minor changes in the [11] mechanism were necessary to accomplish the proposed tasks.

2.1 Computation: Minor Changes

The work presented in [11] provides plain Functional Units (FUs) capable of computing data directly from main memory. In our work, more complex FUs

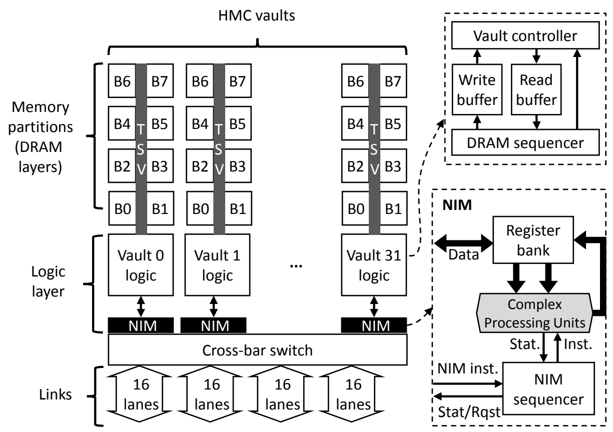

Fig. 1: NIM mechanism overview.

have been implemented to execute NN task-intensive operations, such as exponentiation and division, which can be reconfigurable at runtime. Our mechanism operates at a frequency of 1 GHz. It is composed of 2,048 functional units (integer + floating-point), and a small register file with 8 x 16 registers of 32 bits each per vault. The integer FUs have a latency of 1 cycle for ALU, three cycles for multiplication, 20 cycles for the division, and ten cycles for exponentiation. For the floating-point FUs, the latencies are five cycles for ALU, five cycles for multiplication, 20 cycles for the division, and 18 cycles for exponentiation.

We also included support to perform fast vector elements operation. [11] counts with up to 64 FUs per HMC vault. Thus, all its FUs could be accessed in parallel to execute a single vector addition. Nevertheless, the original register file does not allow such operation, since each process occurs between different registers. To avoid a slow execution that would be constituted by a sequence of *SHIFT* and *ADD* commands, [11] data path was modified to execute intra-register operations, and a new *SUM* instruction was added to [11] ISA. One single vector operation unit can have different ranges of elements, from 64B to 256B.

Also, to schedule a given NN into our device, we simply travel through the neuron parameters' vector, placing each element evenly between memory banks, in an interleaving fashion.

3 Experimental Methodology and Evaluation of NIM

This section describes all performed experiments and its following results. To better understand all presented results, it is important to notice that the total number of neurons simulated in a NN is equivalent to the product of the number of neurons per layer N/L by the total number of layers L.

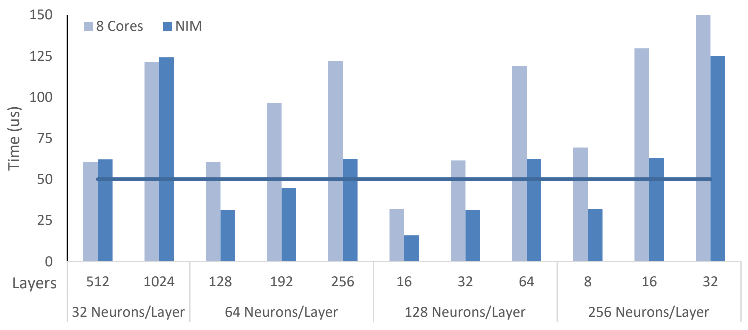

Fig. 2: Izhikevich Equations - 50us Results

3.1 Methodology

To evaluate our work, we have made use of a cycle-accurate HMC simulator [13]. We aimed to simulate the maximum number of neurons while respecting the 50us NTS. Besides, we investigated how many neurons our device was able to simulate in a more relaxed time window of 1ms. At both sets of experiments, we considered as the best configuration result the total number of neurons that could fit its simulation time window, while taking into account a tolerance factor of 3% for 1ms experiments, and 1% for 50us.

The baseline considered was inspired by Intel SandyBridge processor micro-architecture. The SandyBridge is configured with up to 8 cores and AVX2 instruction set capabilities (512 bits of operands per core), and in all cases, the main memory used was a HMC device.

3.2 Performance Results

Izhikevich Application: Figure 2 depicts the results for NNs using Izhikevich equations. As the amount of N/L increases, the number of connections between neurons at different layers grows, therefore requiring more computational power

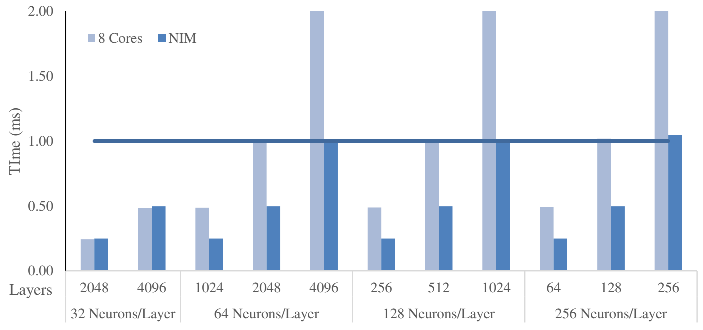

Fig. 3: Izhikevich Equations - 1ms Results

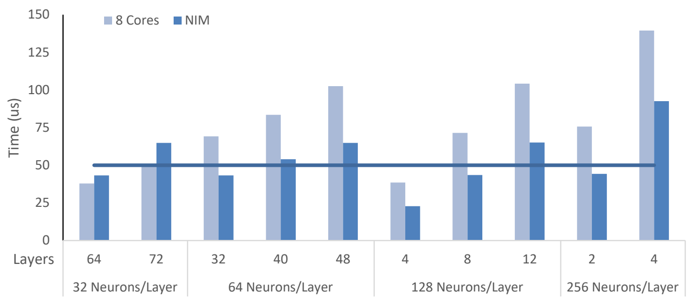

Fig. 4: Hodgkin-Huxley Equations - 50us Results

from the system. During simulation, our NIM mechanism was able to simulate up to 12288 neurons within the 50us NTS (64 N/L, 192 L). In contrast, for the same configuration, the baseline spent almost x2 more time than our NIM device. It is important to notice that for a small number of N/L, the baseline system performed better than our device. That happened because of two main factors. First, the baseline's CPU cores could execute instructions twice as fast as our NIM device. Second, and more important, the number of N/L is responsible for the amount of parallelism available. With more parallelism, a bigger array composed of neuron's input parameters can be sent to out device, thus providing data for more FUs to operate upon (an ideal array size would be of 8 KBytes, where all FUs would be operating).

Figure 3 shows the simulating results for the more relaxed scenario. When the time limit ranges to 1ms, the performance of the NIM mechanism showed the same behavior for N/L configured with up to 32 neurons. However, when the NN is configured with more than 64 N/L, the number of layers becomes less significant. The baseline can represent a maximum of 131072 neurons (64 N/L, 2048 L) while our NIM mechanism is capable of simulating the same amount of neurons at half the baseline time. For 1ms, the NIM simulated up to 262144 neurons in total (64 N/L, 4096 L).

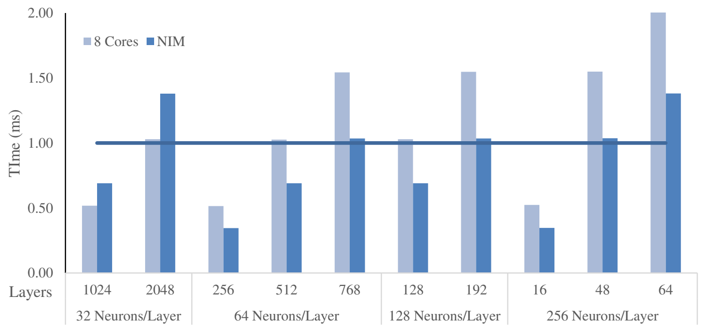

Fig. 5: Hodgkin-Huxley 1ms Results.

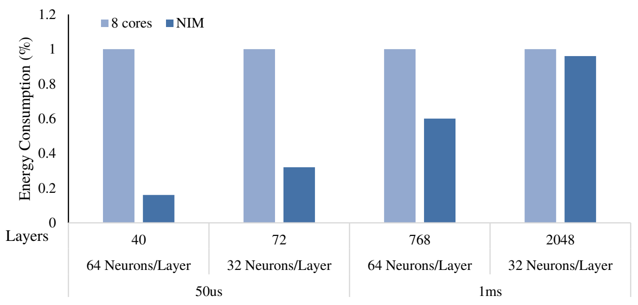

Fig. 6: Hodgkin-Huxley Energy Results

Hodgkin-Huxley (HH) Application: Figure 4 shows the results for the HH model with the time limit of 50us for both the baseline processor and for our mechanism. For a small number of N/L, the baseline showed a better performance than our device because of the little amount of parallelism available in the network. However, with more parallelism available, NIM achieves a better result. Within 50us NTS, the baseline can simulate up to 2304 neurons (32 N/L, 72 L). In contrast, within the same time, our device can simulate up to 2560 neurons (64 N/L, 40 L).

Figure 5 illustrates that the operational frequency of the baseline impacts the total number of neurons simulated. For the 1ms experiments, the baseline could simulate up to 65536 neurons (32 N/L, 2048 L), while at the best NIM configuration our device was able to simulate 49152 neurons (64 N/L, 768 L).

3.3 Energy Consumption

To measure system energy consumption, we used the McPat [14] tool, configured to use 32nm technology for both systems. We have chosen to estimate energy consumption for HH applications since their results showed a more heterogeneous scenario. We compared the baseline and NIM configurations that represented the maximum number of neurons simulated in each case.

Figure 6 depicts the percentage of energy consumed by our system when compared to the baseline. One can notice that the amount of N/L impacts the energy reduction our system can provide. For NNs with more N/L, our device mitigates unnecessary data movement from main memory to cache devices, since more N/L represent less data reuse. In contrast, increasing the number of layers reduces NIM impact over energy consumption, once that the number of hit access at cache memories will increase.

4 Related Work

In this section, we list several works that aim to simulate NNs. Each work targets distinct neuron models and networks topologies, making it not possible to compare the presented work directly with others. However, our evaluation metric (number of neurons in determined simulation time) can be used to approximate our gains over previous ones. We have classified the presented related works into four categories: GP-based, GPU-based, FPGA-based, and PIM-based.

In the first class, one could find works as [15] and [2]. Despite the large processing capability provided by these works, they both suffer from the same issue: neuron communication. In those cases, it is not possible to simulate NN within the natural time step.

[3] is an example of GPU-based neuron simulators. However, the timing constraint needed to represent biologically accurate NN on a large scale is a challenge for GPUs. Besides, GPUs are inefficient regarding energy and power.

In the third category, one could fit an extended number of works, as [4], [16], and [17]. Even though using dedicated hardware to simulate large NN is an effective approach, it is not as flexible as the other ones cited here.

Finally, similarly to our work, [18] aims to accelerate deep learning applications by exploiting PIM capabilities. In their work, the authors present an architecture composed of four HMC devices incremented with CPU and GPU modules at their logic layer. Even though [18] achieved good results, their module is computationally expensive, and it is not energy efficient as our device.

5 Conclusions

In this paper, we presented Neuron In-Memory (NIM), a computational mechanism able to simulate large Neural Networks (NNs). Our work is based on the vector processing capabilities extracted from NN applications that can be implemented directly in memory, taking advantages of the broad bandwidth available in modern 3D-stacked memory devices. To conclude, the presented NIM module is capable of simulating NN of significant sizes in an embedded environment. When compared with traditional multi-core environments, our mechanism provides system acceleration for large NN, while reducing overall energy consumption. In future works, we aim to extend our device to enable networks with layers of different sizes, thereby reducing data movement in small NN topologies.

References

1. J. R. De Gruijl, P. Bazzigaluppi, M. T. de Jeu, and C. I. De Zeeuw, "Climbing fiber burst size and olivary sub-threshold oscillations in a network setting," *PLoS Comput Biol*, vol. 8, no. 12, p. e1002814, 2012.
2. M. Hines, S. Kumar, and F. Schürmann, "Comparison of neuronal spike exchange methods on a blue gene/p supercomputer," *Frontiers in Computational Neuroscience*, vol. 5, no. 49, 2011.
3. M. Wang, B. Yan, J. Hu, and P. Li, "Simulation of large neuronal networks with biophysically accurate models on graphics processors," in *Neural Networks (IJCNN), The 2011 International Joint Conference on*, July 2011, pp. 3184–3193.

4. G. Smaragdos, S. Isaza, M. F. van Eijk, I. Sourdis, and C. Strydis, "Fpga-based biophysically-meaningful modeling of olivocerebellar neurons," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014, pp. 89–98.
5. F. Zenke and W. Gerstner, "Limits to high-speed simulations of spiking neural networks using general-purpose computers," *Frontiers in Neuroinformatics*, vol. 8, p. 76, 2014. [Online]. Available: http://journal.frontiersin.org/article/10.3389/fninf.2014.00076
6. R. Balasubramonian, J. Chang, T. Manning, J. H. Moreno, R. Murphy, R. Nair, and S. Swanson, "Near-data processing: Insights from a micro-46 workshop," *IEEE Micro*, vol. 34, no. 4, pp. 36–42, July 2014.
7. Hybrid Memory Cube Consortium, "Hybrid Memory Cube Specification Rev. 2.0," 2013, http://www.hybridmemorycube.org/.
8. D. U. L. et. al. S. Hong, "25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, Feb 2014, pp. 432–433.
9. A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *Bulletin of Mathematical Biology*, vol. 52, no. 1, pp. 25–71, 1990.
10. E. M. Izhikevich, "Simple model of spiking neurons," *Trans. Neur. Netw.*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003.
11. M. A. Z. Alves, M. Diener, P. C. Santos, and L. Carro, "Large vector extensions inside the hmc," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 1249–1254.
12. P. C. Santos, G. F. Oliveira, D. G. Tome, M. A. Z. Alves, E. C. Almeida, and L. Carro, "Operand size reconfiguration for big data processing in memory," in *2017 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2017.
13. M. A. Z. Alves, M. Diener, F. B. Moreira, C. Villavieja, and P. O. A. Navaux, "Sinuca: A validated micro-architecture simulator," in *High Performance Computation Conference*, 2015.
14. S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 10, no. 1, p. 5, 2013.
15. K. Sakai, P. Sajda, S.-C. Yen, and L. H. Finkel, "Coarse-grain parallel computing for very large scale neural simulations in the {NEXUS} simulation environment," *Computers in Biology and Medicine*, vol. 27, no. 4, pp. 257 – 266, 1997.
16. Y. Zhang, J. P. Mcgeehan, E. M. Regan, S. Kelly, and J. L. Nunez-Yanez, "Biophysically accurate foating point neuroprocessors for reconfigurable logic," *IEEE Transactions on Computers*, vol. 62, no. 3, pp. 599–608, March 2013.
17. M. Beuler, A. Tchaptchet, W. Bonath, S. Postnova, and H. A. Braun, *Artificial Neural Networks and Machine Learning – ICANN 2012: 22nd International Conference on Artificial Neural Networks, Lausanne, Switzerland, September 11-14, 2012, Proceedings, Part I*, 2012, ch. Real-Time Simulations of Synchronization in a Conductance-Based Neuronal Network with a Digital FPGA Hardware-Core.
18. L. Xu, D. P. Zhang, and N. Jayasena, "Scaling deep learning on multiple in-memory processors," *WoNDP: 3rd Workshop on Near-Data Processing*, 2015.