Stabilization and Acceleration of CFD Simulation by Controlling Relaxation Factor Based on Residues: An SNN Based Approach

Mithilesh Maurya, Dighanchal Banerjee, Sounak Dey, Dilshad Ahmad
TCS Research, India
{mk.maurya|dighanchal.b|sounak.d|dilshad.ahmad}@tcs.com

Abstract

Computational Fluid Dynamics (CFD) simulation involves the solution of a sparse system of linear equations. Faster convergence to a physically meaningful CFD simulation result of steady-state physics depends largely on the choice of optimum value of the under-relaxation factor (URF) and continuous manual monitoring of simulation residues. In this paper, we present an algorithm for classifying simulation convergence (or divergence) based on the residues using a spiking neural network (SNN) and a control logic. This algorithm maintains optimum URF throughout the simulation process and ensure accelerated convergence of the simulation. The algorithm is also able to stabilize and bring back a diverging simulation to the converging range automatically without manual intervention. To the best of our knowledge, SNN is used for the first time to solve such complex classification problem and it achieves an accuracy of 92.4% to detect the divergent cases. When tested on two steady-state incompressible CFD problems, our solution is able to stabilize every diverging simulation and accelerate the simulation time by at least 10% compared to a constant value of URF.

1 Introduction

System of linear equations governing the model of solid-fluid interactions in Computational Fluid Dynamics (CFD) domain are usually solved using iterative methods [6, 19, 7] to reach to an acceptable numerical solution. This iterative process is time consuming, compute intensive, and the solution does not guarantee convergence to a physically meaningful solution for a given set of simulation parameters. This results into huge waste of man hours and computation resource. Though controlling factors such as *under relaxation factor* (URF) and convergence indicators such as *simulation residue history* (SRH) are tuned to ensure convergence, but it requires manual intervention to select suitable values. Thus, an automatic monitoring mechanism of residue history (to interpret convergence or divergence) and a subsequent control logic to auto-tune the URF would help in: (i) stabilising a diverging simulation, (ii) reaching faster convergence by accelerating converging simulation.

Fuzzy logic based approaches [15, 3] and Expression based methods [12, 20] have been applied to calculate & control URF but they tend to increase the computation time and are not applicable across all classes of CFD problems. RL based acceleration methods [13] for CFD simulations are computationally expensive and slow. Looking at the compute-intensive nature of simulation and to cater the requirement of early detection of divergence from as less data as possible, use of Spiking Neural Networks (SNN), a 3rd generation ML framework inspired from functionalities of mammalian brain, can be a good choice to reach to a useful solution [14]. SNNs are comparatively faster to learn from sparse data and are far less compute & power intensive. SNNs achieve this through asynchronous event handling and co-location of memory and computation [1, 4, 22].

Machine Learning and the Physical Sciences workshop, NeurIPS 2022.

In this paper, we propose an improved and robust solution to stabilize diverging CFD simulations and accelerate the converging CFD simulation by keeping the URF to its optimum value. The residual value over time is treated as a continuous time series. Current data of a fixed size window from this time series is passed to the SNN to be classified either as diverging or as converging. Depending on the classification, a control logic is used to change the URF. The classifier and control logic work together without interfering with the CFD simulation i.e. the process is non-invasive and works in parallel to the simulation. We demonstrate the efficacy of our approach via two steady-state incompressible CFD problems namely *Backward Facing Step* and *Flow inside Tundish*. We verified our findings by running the algorithm with OpenFOAM [18] and Ansys Fluent [11] simulators. We found that (i) the SNN is able to detect the divergent cases with 92.4% accuracy (with window size = 30), (ii) our solution is able to stabilize each and every diverging simulations and finally, (iii) it is also able to accelerate the simulation time by at least 10% compared to a fixed URF value.

2 Spiking Neural Network: Methodology & Architecture

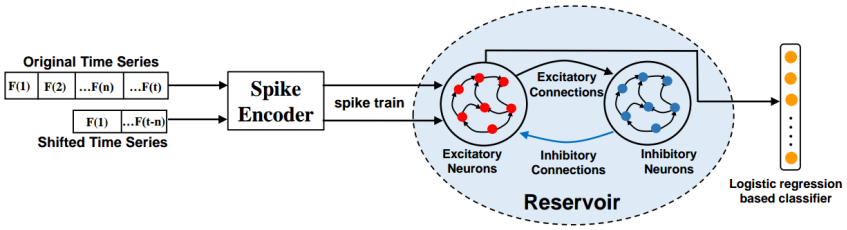


Figure 1: Architecture of the reservoir based SNN classifier

As shown in Fig. 1, proposed spiking neural network architecture comprises of three main blocks: (i) Spike encoder, (ii) Spiking reservoir and (iii) Classifier.

The Spike Encoder. Unlike classical ANNs, SNNs can work with spike inputs only. *Spike Encoder* block converts real-valued time series of residual values, denoted by F(t) in Fig. 1, into representative spike trains. Of the two popular soft spike encoding techniques namely *rate encoding* (information is encoded in terms of the number or rate of firing of a neuron) and *temporal* encoding (encodes information based on the temporal distance between spikes i.e. inter-spike interval) here we have used the latter as it is more efficient to encode temporal signal than the former.

The Spiking Reservoir. The encoded spike trains are fed into the second block of our architecture namely, a reservoir based SNN. The reservoir is a sparse and recurrently connected population of excitatory and inhibitory neurons, where each neuron is connected to a set of other neurons in the same population in a probabilistic fashion such that resulting dynamics of the network remains stable. Reservoir based SNN is best suited for extracting spatio-temporal features [10, 5, 21]. The sparse input and recurrent weights with directed cycles act as a non-linear random projection of the input feature space to a high dimensional spatio-temporal embedding space, where implicit temporal features become explicit. These embeddings are captured as neuronal traces of the reservoir neurons. In the reservoir, we have kept 400 excitatory and 100 inhibitory neurons, keeping in mind the 4:1 ratio in its biological counterpart. We have used *Leaky Integrate & Fire* (LIF) [8] neurons here with the membrane decay constant = 30. We have used a time-shifted version of $F(t)$, namely $F(t-n)$, and feed it into the reservoir after encoding, so that the activity of the reservoir always remain above an acceptable threshold. Here, n lies in range [5,10].

Classifier: The neuronal trace values of the excitatory reservoir neurons are fed into a *Logistic Regression* based classifier that is trained with corresponding class labels. Once training is done, the neuronal trace values corresponding to testing data are fed into this classifier to get the final classification result.

3 Control Logic and Simulation

Control Logic. The rule for changing the URF depends on the output of the classifier. For a diverging simulation, the URF is reduced in steps of 20% so that simulation stabilizes quickly. For converging simulation, the URF is increased in steps of 1% as we want to accelerate the simulation slowly.

Simulation Stabilization & Acceleration. The integrated view of classifier & control algorithm along with the simulation is shown in Fig. 2. The simulation for a defined CFD model starts with a

small URF value and is continuously monitored via the *SRH log file*. We discard first 5 iterations (as they might not be stable) and the SNN classifier takes SR values in a window of next 30 iterations as a time series to predict its convergence behaviour. The CFD model is updated with a new value of URF based on the aforementioned control logic and the same loop of iteration and classification goes on until the system reaches convergence. The activities in Green and Blue block are independent of each other.

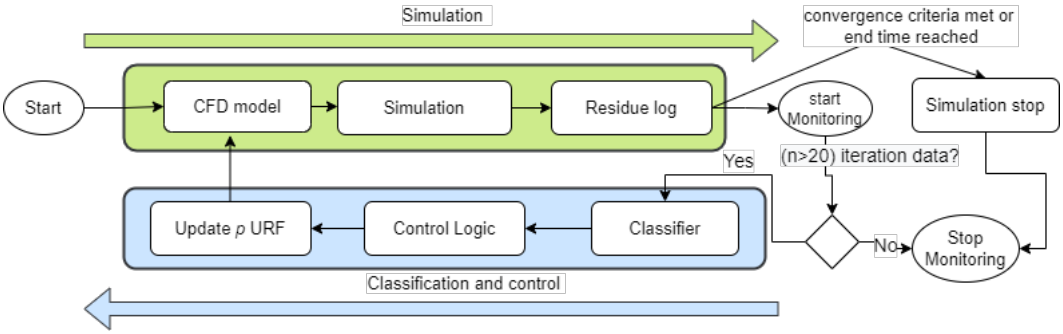
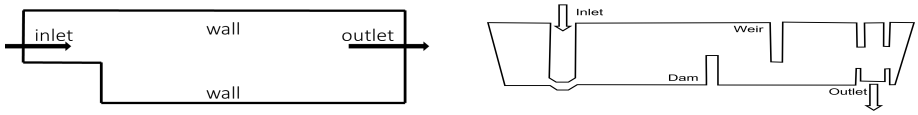


Figure 2: Integrated workflow of classifier and control (Blue block) with the simulation (Green block)

4 Experimental setup, Results & Discussion

Dataset Generation. Ten benchmark and validated cases of CFD are selected from OpenFOAM in order to generate training data. All the model variables corresponding to each of these CFD cases were kept constant except the URF of pressure variable which is changed in steps of 0.1 in the range [0.1, 1.8]. Each simulation terminal output is recorded in a log file and is parsed to get the SRH that has data for initial and final residue of the flow field variables. The recorded data is labelled as *"converging"* or *"diverging"* classes as per simulation outcome. A total of 140 converging and 140 diverging SRH data is generated. Number of iterations in each of this residue data vary from few hundred to thousands for diverging and converging simulations respectively. Each of these data files are split into smaller SRH data of 30 iterations, with the same labelling as that of the series. A total of 1200 such windowed data sets for each classes were taken for training and testing.

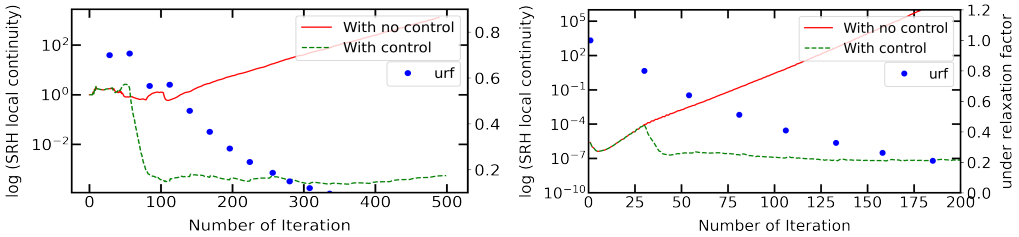
Training and Performance of SNN classifier. 900 samples of each class from the dataset were used for training and rest 300 were used for testing the SNN classifier. We have considered 9 features of CFD simulation namely, initial and final residue of pressure, velocity (x, y) components and local, global and cumulative continuity. Each sample time series from training set is segmented into windows of a particular length. To decide upon the optimal length of the window, we have trained the network with data of different window sizes and measured the corresponding testing accuracy. As shown in Fig 4f, window size 50 provides highest accuracy 94.9% but performance does not degrade much with reducing window size, till it reaches low values such as 25 or 20. Lower the window size, faster will be the recognition of divergent time series. We have taken *window size* = 30 as optimal for our experiments. *The network has achieved an accuracy of 92.4%*. Also the time for prediction of one label is around 8 seconds running on a 32 GB RAM CPU machine.



(a) (b)

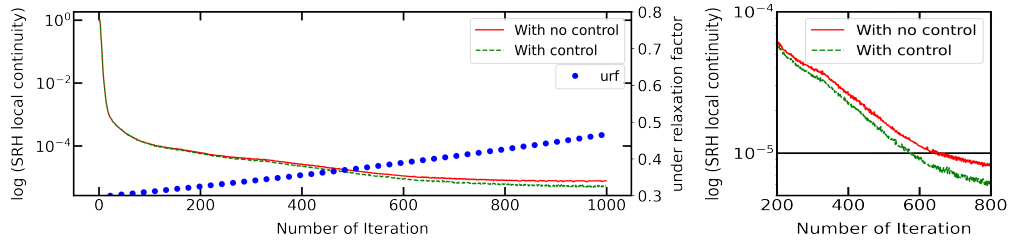
Figure 3: (a) *Backward facing step* geometry - a developed steady turbulent flow of air enters through the inlet and comes out through outlet. (b) *Tundish geometry* - molten steel enters through the inlet and comes out of the outlet, dam and weir help in removing impurities.

CFD Test Cases. We have tested the proposed system on two different steady-state incompressible CFD problems namely, *Backward-facing step* [17] and *Flow inside a Tundish* [16] implemented in OpenFOAM v7 [18] and Ansys Fluent v19 [11]. A *Backward Facing Step* is widely known for its application in the studies on turbulence in internal flows (refer Fig. 3a). A Tundish is a reservoir in continuous casting unit of steel(refer Fig. 3b).



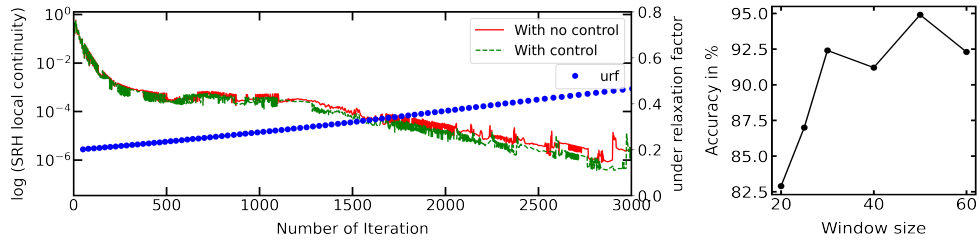
(a) Backward facing step problem solved using Fluent starting with $pressure$ URF=0.7

(b) Flow inside Tundish problem solved using OpenFOAM starting with $pressure$ URF=1



(c) Backward facing step problem solved using Fluent starting with $pressure$ URF=0.3

(d) Scaled plot of (c) to show the simulation acceleration



(e) Backward facing step problem solved using OpenFOAM starting with $pressure$ URF=0.2

(f) Variation of classifier accuracy with different window sizes.

Figure 4: CFD simulation performance with and without proposed model.

Results and Discussion: As shown in Fig. 4a, in the first step of simulation of *Backward Facing Step*, the URF value increases by 1% from 0.7 to 0.707 (following the control logic) as the classifier predicted convergence. At the next prediction, the classifier predicts divergence and the URF value is reduced by 20% to 0.5656. In this way, the classifier and control logic update the URF value at intervals of 30 iterations. Fig. 4b shows the simulation behaviour for the *Tundish*. The efficacy of our system is established by the Green and Red curves which represent the simulation behaviour with and without URF control. *Our solution is able to stabilize 100% of the diverging simulations.*

Another utility of our system is to accelerate a converging simulation. As shown in Fig. 4c, the URF value for a converging *Backward Facing Step* simulation is rising slowly from 0.3 till 0.5 in steps of 1%. A zoomed view of Fig. 4c near convergence, as shown in Fig. 4d, shows that the residue plot with control cuts the log of scaled residue value at 1e-5 much before residue plot without control. *Here, we have achieved 10% reduction in number of iterations*. In case of *Backward facing step*, a significant reduction in the number of iteration is observed (refer Fig. 4e). *Even if we start with low or high URF value, the model is observed to provide positive gains by accelerating simulation.*

Conclusion and Future Works. Our proposed solution is generic and can be integrated with any CFD tools. This automated process saves a lot of manual effort and time thereby making it beneficial for many industry scale CFD problems that run for days. Moreover, the SNN model can easily be retrained with additional data to improve the accuracy and to cater to different classes of problems. In future, we intend to improve the existing fixed control logic to an adaptive URF controller varying with the SR value. We also intend to test the performance and power consumption of the SNN by running it on a real neuromoprhic hardware such as Intel Loihi [2, 9] or Brainchip Akida [1].

References

[1] Brainchip. Akida akd 1000, 2022. URL https://brainchip.com/akida-neural-processor-soc/.

[2] M. Davies, N. Srinivasa, T.-H. Lin, G. N. Chinya, Y. Cao, S. H. Choday, G. D. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38:82–99, 2018.

[3] Z. Dragojlovic and D. A. Kaminski. A fuzzy logic algorithm for acceleration of convergence in solving turbulent flow and heat transfer problems. *Numerical Heat Transfer, Part B: Fundamentals*, 46(4):301–327, 2004.

[4] S. Furber. Large-scale neuromorphic computing systems. *Journal of neural engineering*, 13(5): 051001, 2016.

[5] A. M. George, D. Banerjee, S. Dey, and A. Mukherjee. A reservoir-based convolutional spiking neural network for gesture recognition from dvs input. *International Joint Conference on Neural Networks*, 2020.

[6] A. Gosman. Developments in cfd for industrial and environmental applications in wind engineering. *Journal of Wind Engineering and Industrial Aerodynamics*, 81(1-3):21–39, 1999.

[7] C. T. Kelley. *Iterative methods for linear and nonlinear equations*. SIAM, 1995.

[8] L. Lapicque. Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal de Physiologie et de Pathologie Generalej*, 9:620–635, 1907.

[9] C.-K. Lin, A. Wild, G. N. Chinya, Y. Cao, M. Davies, D. M. Lavery, and H. Wang. Programming spiking neural networks on intel's loihi. *Computer*, 51(3):52–61, 2018.

[10] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11): 2531–2560, 2002.

[11] U. Manual. Ansys fluent 12.0. *Theory Guide*, 2009.

[12] C. Min and W. Tao. An under-relaxation factor control method for accelerating the iteration convergence of flow field simulation. *Engineering Computations*, 2007.

[13] S. Pawar and R. Maulik. Distributed deep reinforcement learning for simulation control. *Machine Learning: Science and Technology*, 2(2):025029, 2021.

[14] F. Ponulak and A. Kasinski. Introduction to spiking neural networks: Information processing, learning and applications. *Acta neurobiologiae experimentalis*, 71(4):409–433, 2011.

[15] J. Ryoo, D. Kaminski, and Z. Dragojlovic. A residual-based fuzzy logic algorithm for control of convergence in a computational fluid dynamic simulation. 1999.

[16] P. Singh and D. Ahmad. Optimal tundish deisgn using openfoam. In *ASTFE Digital Library*. Begel House Inc., 2018.

[17] R. So, Y. Lai, B. Hwang, and G. Yoo. Low-reynolds-number modelling of flows over a backward-facing step. *Zeitschrift für angewandte Mathematik und Physik ZAMP*, 39(1):13–27, 1988.

[18] H. G. Weller, G. Tabor, H. Jasak, and C. Fureby. A tensorial approach to computational continuum mechanics using object-oriented techniques. *Computers in physics*, 12(6):620–631, 1998.

[19] P. Wesseling. *Principles of computational fluid dynamics*, volume 29. Springer Science & Business Media, 2009.

[20] W. You, Z.-Y. Li, and W.-Q. Tao. A general self-adaptive under-relaxation strategy for fast and robust convergence of iterative calculation of incompressible flow. *Numerical Heat Transfer, Part B: Fundamentals*, 77(4):299–310, 2020.

[21] A. Zhang, W. Zhu, and M. Liu. Self-organizing reservoir computing based on spiking-timing dependent plasticity and intrinsic plasticity mechanisms. In *2017 Chinese Automation Congress (CAC)*, pages 6189–6193, USA, 2017. IEEE.

[22] M. A. Zidan, J. P. Strachan, and W. D. Lu. The future of electronics based on memristive systems. *Nature Electronics*, 1(1):22, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]