NVIDIA SimNet™: An AI-Accelerated Multi-Physics Simulation Framework

Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry

Nvidia

Abstract. We present SimNet, an AI-driven multi-physics simulation framework, to accelerate simulations across a wide range of disciplines in science and engineering. Compared to traditional numerical solvers, SimNet addresses a wide range of use cases - coupled forward simulations without any training data, inverse and data assimilation problems. SimNet offers fast turnaround time by enabling parameterized system representation that solves for multiple configurations simultaneously, as opposed to the traditional solvers that solve for one configuration at a time. SimNet is integrated with parameterized constructive solid geometry as well as STL modules to generate point clouds. Furthermore, it is customizable with APIs that enable user extensions to geometry, physics and network architecture. It has advanced network architectures that are optimized for high-performance GPU computing, and offers scalable performance for multi-GPU and multi-Node implementation with accelerated linear algebra as well as FP32, FP64 and TF32 computations. In this paper we review the neural network solver methodology, the SimNet architecture, and the various features that are needed for effective solution of the PDEs. We present real-world use cases that range from challenging forward multi-physics simulations with turbulence and complex 3D geometries, to industrial design optimization and inverse problems that are not addressed efficiently by the traditional solvers. Extensive comparisons of SimNet results with open source and commercial solvers show good correlation. The SimNet source code is available at https://developer.nvidia.com/simnet.

1 Introduction

Simulations are pervasive in every domain of science and engineering. However, they become computationally expensive as more geometry details are included and as model size, the complexity of physics or the number of design evaluations increases. Although deep learning offers a path to overcome this constraint, supervised learning techniques are used most often in the form of traditional data driven neural networks (e.g., [1, 2]). However, generating data can be an expensive and time consuming process. Furthermore, these models may not obey the governing physics of the problem, involve extrapolation and generalization errors, and provide unreliable results.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

In comparison with the traditional solvers, neural network solvers [3–5] can not only do parameterized simulations in a single run, but also address problems not solvable using traditional solvers, such as inverse or data assimilation problems and real time simulation. They can also be embedded in the traditional solvers to improve the predictive capability of the solvers. Training of neural network forward solvers can be supervised only based on the governing laws of physics, and thus, unlike the data-driven deep learning models, neural network solvers do not require any training data. However, for data assimilation or inverse problems, data constraints are introduced in the loss function.

Rapid evolution of GPU architecture suited for AI and HPC, as well as introduction of open source frameworks like Tensorflow have motivated researchers to develop novel algorithms for solving PDEs (e.g., [3, 5–8]). Recently, a number of neural network solver libraries are being developed (e.g., TensorFlow-based DeepXDE [9], Keras-based SciANN [10], and Julia-based NeuralPDE.jl [11]), aiming at making these solvers more accessible. Although the existing research studies and libraries played a crucial role in advancing the neural network solvers, the attempted examples are mostly limited to simple 1D or 2D domains with straightforward governing physics, and the neural network solvers in their current form still struggle to solve real-world applications that involve complex 3D geometries and multi-physics systems. In this paper we present SimNet, that aims to address the current computational challenges with neural network solvers. As an example, SimNet enables design optimization of a FPGA heat sink (see Figure 1) through a single network training without any training data. In contrast, the traditional solvers are not capable of simulating geometries with several design parameters in a single run.

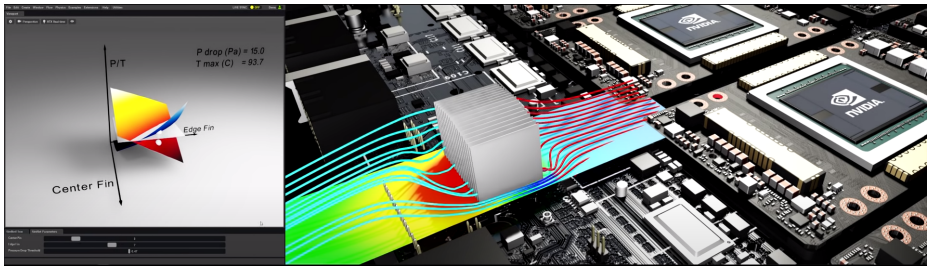


Fig. 1: Design optimization of an FPGA heat sink using SimNet. The center and side fin heights are the two design variables.

Our Contributions: Several research studies have recently been published demonstrating solution of PDEs using neural networks. However, our experience has shown that they do not converge well when used as forward solvers for industrial problems due to the gradients, singularities and discontinuities introduced by complex geometries or physics. Our main contributions in this paper are to offer several novel features to address these challenges - Signed Distance Functions

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

(SDFs) for loss weighting, integral continuity planes for flow simulation, advanced neural network architectures, point cloud generation for real world geometries using constructive geometry module as well as STL module and finally parameterization of both geometry and physics. Additionally, for the first time to our knowledge, we solve high Reynolds number flows (by adopting the RANS equations and the zero-equation turbulence model [12]) in industrial applications without using any data.

2 Neural Network Solvers

A neural network solver approximates the solution to a given PDE and a set of boundary and initial constraints using a feed-forward fully-connected neural network. The model is trained by constructing a loss function for how well the neural network is satisfying the PDE and constraints. A schematic of the structure of a neural network solver is shown in Figure 2.

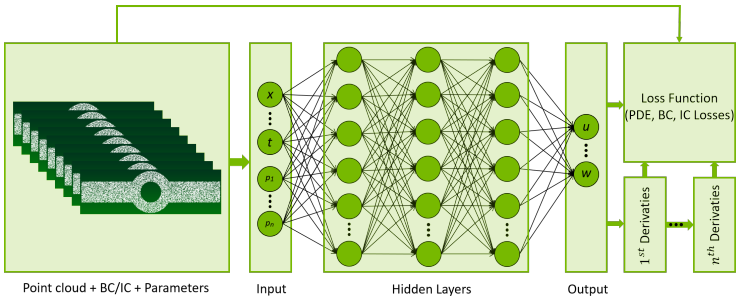


Fig. 2: A schematic of the structure of a neural network solver.

Let us consider the following general form of a PDE:

$$\begin{aligned} \mathcal{N}_i[u](\mathbf{x}) &= f_i(\mathbf{x}), \quad \forall i \in \{1, \cdots, N_{\mathcal{N}}\}, \mathbf{x} \in \mathcal{D}, \\ \mathcal{C}_j[u](\mathbf{x}) &= g_j(\mathbf{x}), \quad \forall j \in \{1, \cdots, N_{\mathcal{C}}\}, \mathbf{x} \in \partial\mathcal{D}, \end{aligned} \tag{1}$$

where $\mathcal{N}_i$'s are general differential operators, $\mathbf{x}$ is the set of independent variables defined over a bounded continuous domain $\mathcal{D} \subseteq \mathbb{R}^D, D \in \{1, 2, 3, \cdots\}$, and $u(\mathbf{x})$ is the solution to the PDE. $\mathcal{C}_j$'s denote the constraint operators and usually cover the boundary and initial conditions. $\partial\mathcal{D}$ also denotes a subset of the domain boundary that is required for defining the constraints. We seek to approximate the solution $u(\mathbf{x})$ by a neural network $u_{net}(\mathbf{x})$ that, in it's most simple form, takes the following form:

$$u_{net}(\mathbf{x}; \theta) = \mathbf{W}_n\big\{\phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_1 \circ \phi_E\big\}(\mathbf{x}) + \mathbf{b}_n, \quad \phi_i(\mathbf{x}_i) = \sigma(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i), \tag{2}$$

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

where $\mathbf{x} \in \mathbb{R}^{d_0}$ is the input to network, $\phi_i \in \mathbb{R}^{d_i}$ is the i^{th} layer of the network, $\mathbf{W}_i \in \mathbb{R}^{d_i \times d_{i-1}}, \mathbf{b}_i \in \mathbb{R}^{d_i}$ are the weight and bias of the i^{th} layer, θ denotes the set of network's trainable parameters, i.e., $\theta = \{\mathbf{W}_1, \mathbf{b}_1, \cdots, \mathbf{b}_n, \mathbf{W}_n\}$, n is the number of layers, and σ is the activation function. We suppose that this neural network is infinitely differentiable, i.e. $u_{net} \in C^\infty$. ϕ_E is an input encoding layer. More advanced architectures will be introduced in Section 3.3.

In order to train this neural network, we construct a loss function that penalizes over the divergence of the approximate solution $u_{net}(\theta)$ from the PDE in equation 1, and such that the constraints are encoded as penalty terms. To this end, we define the following residuals:

$$\begin{aligned} r_{\mathcal{N}}^{(i)}\left(\mathbf{x}; u_{net}(\theta)\right) &= \mathcal{N}_i[u_{net}(\theta)]\left(\mathbf{x}\right) - f_i\left(\mathbf{x}\right), \\ r_{\mathcal{C}}^{(j)}\left(\mathbf{x}; u_{net}(\theta)\right) &= \mathcal{C}_j[u_{net}(\theta)]\left(\mathbf{x}\right) - g_j\left(\mathbf{x}\right), \end{aligned} \tag{3}$$

where $r_{\mathcal{N}}^{(i)}$ and $r_{\mathcal{C}}^{(j)}$ are the PDE and constraint residuals, respectively. The loss function then takes the following form:

$$\mathcal{L}_{res}(\theta) = \sum_{i=1}^{N_{\mathcal{N}}} \int_{\mathcal{D}} \lambda_{\mathcal{N}}^{(i)}(\mathbf{x}) \left\| r_{\mathcal{N}}^{(i)}\left(\mathbf{x}; u_{net}(\theta)\right) \right\|_p d\mathbf{x} + \sum_{j=1}^{N_{\mathcal{C}}} \int_{\partial\mathcal{D}} \lambda_{\mathcal{C}}^{(j)}(\mathbf{x}) \left\| r_{\mathcal{C}}^{(j)}\left(\mathbf{x}; u_{net}(\theta)\right) \right\|_p d\mathbf{x}, \tag{4}$$

where $\|\cdot\|_p$ denotes the p-norm, and $\lambda_{\mathcal{N}}^{(i)}, \lambda_{\mathcal{C}}^{(j)}$ are weight functions that control the loss interplay between within and across different terms. The network parameters θ are optimized iteratively using variants of the stochastic gradient descent method. At each iteration, the integral terms in the loss function are approximated using a regular or Quasi-Monte Carlo method, and using a batch of samples from the independent variables $\mathbf{x}$. Automatic differentiation is commonly used to compute the required gradients in $\nabla\mathcal{L}_{res}(\theta)$.

3 SimNet Overview

SimNet is a Tensorflow based neural network solver and offers various APIs that enable the user to leverage the existing functionality to build their own applications on the existing modules. An overview of SimNet architecture is presented in Figure 3. The geometry modules, PDE module, and data are used to fully specify the physical system. The user also specifies the network architecture, optimizer and learning rate schedule. SimNet then constructs the neural network solver, forms the loss function, and unrolls the graph efficiently to compute the gradients. The SimNet solver then starts the training or inference procedure using TensorFlow's built-in functions on a single or cluster of GPUs. The outputs are saved in the form of CSV or VTK files and can be visualized using TensorBoard and ParaView.

3.1 Geometry modules

SimNet contains Constructive Solid Geometry (CSG) and Tessellated Geometry (TG) modules. With SimNet's CSG module, constructive geometry primitives

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

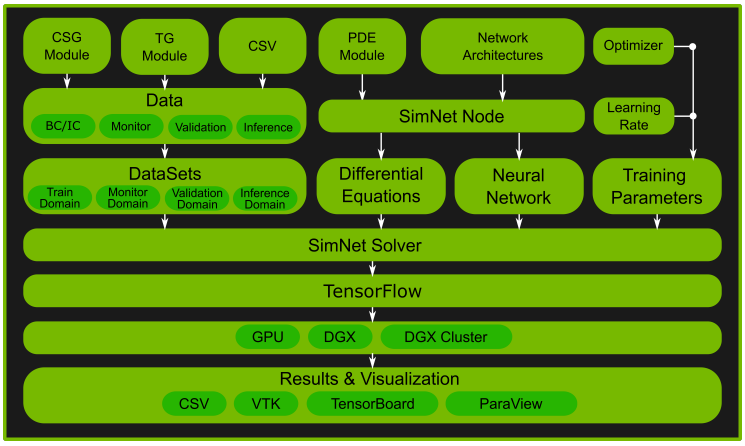


Fig. 3: SimNet structure.

can be defined and Boolean operations performed. This allows the creation and parameterization of a wide range of geometries. The TG module uses tesselated geometries in the form of STL or OBJ files to work with complex geometries. One area of considerable interest is how to weight the loss terms in the overall loss function. SimNet offers spatial loss weighting, where each weight parameter can be a function of the spatial inputs. In many cases we use the Signed Distance Function (SDF) for this weighting. Assuming $\mathcal{D}_x$ is the spatial subset of the input domain $\mathcal{D}$ with boundaries $\partial\mathcal{D}_x$, the SDF-based weight function is defined as

$$\lambda(\mathbf{x_s}) = \begin{cases} d(\mathbf{x_s}, \partial\mathcal{D}_x) & \mathbf{x_s} \in \mathcal{D}_x, \\ -d(\mathbf{x_s}, \partial\mathcal{D}_x) & \mathbf{x_s} \in \mathcal{D}_x^c. \end{cases} \tag{5}$$

Here, $\mathbf{x}_s$ is the spatial inputs, and $d(\mathbf{x_s}, \partial\mathcal{D}_x)$ represents the Euclidean distance between $\mathbf{x}_s$ and it's nearest neighbor on $\mathcal{D}_x$. If the geometry has sharp corners this often results in sharp gradients in the solution of the PDE. Weighting by the SDF tends to mitigate the deleterious effects of sharp local gradients, and often results in an improvement in convergence speed and accuracy. Both of the SimNet geometry modules allow for the SDF and its spatial derivatives to be computed. CSG uses SDF functions to implicitly define the geometry. To accelerate the computation of the SDF on tessellated meshes of complex geometries, we developed a custom library that leverages NVIDIA's OptiX for both inside/outside (sign) testing and distance computation. The sign test uses ray intersection and triangle normal alignment (via dot product). The distance testing is done by using the bounded volume hierarchy (BVH) interface provided by OptiX, which yields excellent performance and accuracy for distance computations.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

3.2 PDE module

The PDE module in SimNet consists of a variety of differential equations including the Navier-Stokes, diffusion, advection-diffusion, wave, and elasticity equations. To make this module extensible for the user to easily define their own PDEs, SimNet uses symbolic mathematics enabled by SymPy [13]. A novel contribution of SimNet is the adoption of the zero-equation turbulence model [12], and this is the first time a neural network solver is made capable of simulating flows with high Reynolds numbers, as shown in the next section. Moreover, for fluid flow simulation, we propose the use of integral continuity planes. For some problems involving channel flow, we found that, in addition to solving the Navier-Stokes equations in differential form, specifying the mass flow (for compressible flows) or volumetric flow rate (for incompressible flows) through some of the planes in the domain helps in satisfying the continuity equation better and faster and improving the accuracy further. Assuming there is no leakage of flow, we can guarantee that the flow exiting the system must be equal to the flow entering the system, and also equal to the flow passing from any plane parallel to the inlet plane throughout the channel.

3.3 Network architectures

In addition to the standard fully connected networks, SimNet offers more advanced architectures, including the Fourier feature and Modified Fourier feature networks, and Sinusoidal Representation Networks (SiReNs) [14] to alleviate the spectral bias [15] in neural networks and improve convergence. The Fourier feature network in SimNet is a variation of the one proposed in [16] with trainable encoding, and takes the form in equation 7 with the following encoding

$$\phi_E = \left[\sin\left(2\pi \mathbf{f} \times \mathbf{x}\right); \cos\left(2\pi \mathbf{f} \times \mathbf{x}\right) \right]^T, \tag{6}$$

where $\mathbf{f} \in \mathbb{R}^{n_f \times d_0}$ is the trainable frequency matrix and n_f is the number of frequency sets. The modified Fourier feature network is SimNet's novel architecture, where two transformation layers are introduced to project the Fourier features to another learned feature space, and are then used to update the hidden layers through element-wise multiplications, similar to its standard fully connected counterpart in [8]. It is shown in the next section that this multiplicative interaction between the Fourier features and hidden layers can improve the training convergence and accuracy. The hidden layers in this architecture take the following form

$$\phi_i(\mathbf{x}_i) = \left(1 - \sigma\left(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i\right)\right) \odot \sigma\left(\mathbf{W}_{T_1} \phi_E + \mathbf{b}_{T_1}\right) + \sigma\left(\mathbf{W}_i \mathbf{x}_i + \mathbf{b}_i\right) \odot \sigma\left(\mathbf{W}_{T_2} \phi_E + \mathbf{b}_{T_2}\right), \tag{7}$$

where $i > 1$ and $\{\mathbf{W}_{T_1}, \mathbf{b}_{T_1}\}, \{\mathbf{W}_{T_2}, \mathbf{b}_{T_2}\}$ are the parameters for the two transformation layers.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

4 Use Cases

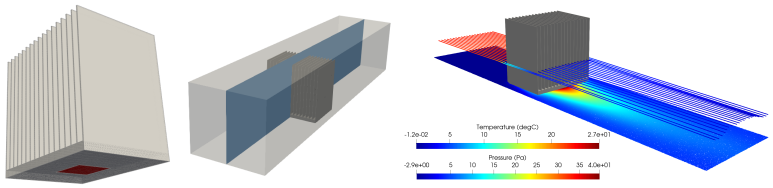
In this section, we present four use cases for SimNet to illustrate its capabilities. Although SimNet is capable of simulating transient flows using the continuous-time sampling approach [3], the first three use cases are time-independent. A more efficient and accurate approach based on the convolutional LSTMs for transient simulations as well as integration of two-equation turbulence models for turbulent simulations are under development. For the entire networks in this section, the architectures consist of 6 layers, each with 512 units. Swish [17] nonlinearities are used in the fully connected, Fourier feature, and modified Fourier feature networks (except for the Fourier layers). For the simulations presented in Sections 4.2 to 4.4, the standard fully connected architecture is used. Adam optimizer with an initial learning rate of 10^{-4} and an exponential decay is used. We use Monte Carlo integration for computing the loss function in equation 4. Moreover, we use integral continuity planes for channel flows. For the simulations in use cases 4.1 to 4.3, we use the SDF for weighting the PDE residuals. It must be noted that use cases 4.1 to 4.3 are solved in the forward manner without using any training data. Please refer to the SimNet user guide for details of the problem setup.

4.1 Turbulent and multi-physics simulations

Using an FPGA heat sink example, we demonstrate the SimNet's capability in accurately solving multi-physics problems involving high Reynolds number flows. The heat sink geometry placed inside a channel is depicted in Figures 4a, 4b. This particular geometry is challenging to simulate due to thin fin spacing that causes sharp gradients that are difficult to learn for a neural network solver. Using the zero-equation turbulence model, we solve a conjugate heat transfer problem with a flow at $Re = 13,239$. Generally, simulation of high-Re flows are particularly difficult due to the chaotic fluctuations of the flow field properties that are caused by instabilities in the shear layer. Due to the one-way coupling between the heat and incompressible flow equations, two separate neural networks are trained for flow (trained first) and the temperature (trained next) fields. This approach is useful for one-way coupled multi-physics problems to achieve significant speed-up.

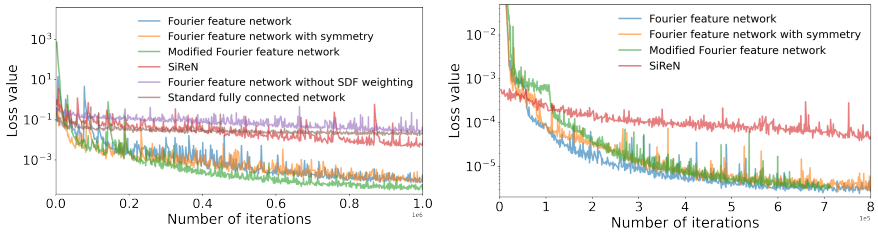
We simulate this conjugate heat transfer problem with different architectures and also with symmetry boundary conditions. Loss curves are shown in Figure 5. This figure also includes the flow convergence results for a Fourier feature model without SDF loss weighting and a standard fully connected model, showing that they fail to provide a reasonable convergence and highlighting the importance of SDF loss weighting and advanced architectures. The streamlines and temperature profile obtained from the modified Fourier feature model are shown in Figure 4c. A comparison between the SimNet and OpenFoam results for flow and temperature fields is also presented in Figure 6. Results for the pressure drop and peak temperature are presented in Table 1. The OpenFoam simulation was performed using a conjugate heat solver based on the SIMPLE algorithm and the differences between the commercial solver and OpenFoam peak temperatures are likely due to the differences in the solvers and the schemes used in these two simulations.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36



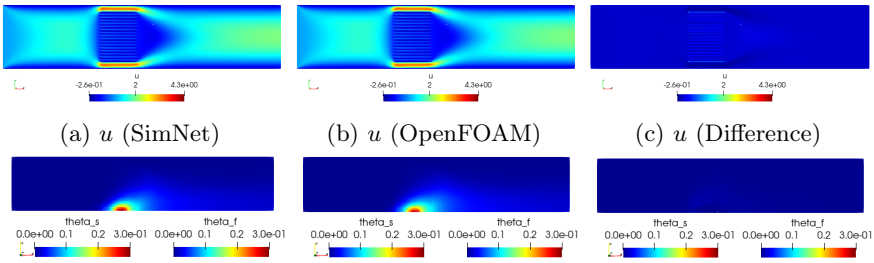
(a) (b) (c)

Fig. 4: FPGA heat sink example. (a) heat sink geometry; (b) Simulation domain (with symmetry plane); (c) SimNet results for streamlines and temperature.



(a) FPGA flow training (b) FPGA heat training

Fig. 5: Loss curves for FPGA training using different architectures.



(a) u (SimNet) (b) u (OpenFOAM) (c) u (Difference)

(d) T (SimNet) (e) T (OpenFOAM) (f) T (Difference)

Fig. 6: A comparison between the SimNet (with modified Fourier feature network) and OpenFoam results for FPGA on a 2D slice of the domain.

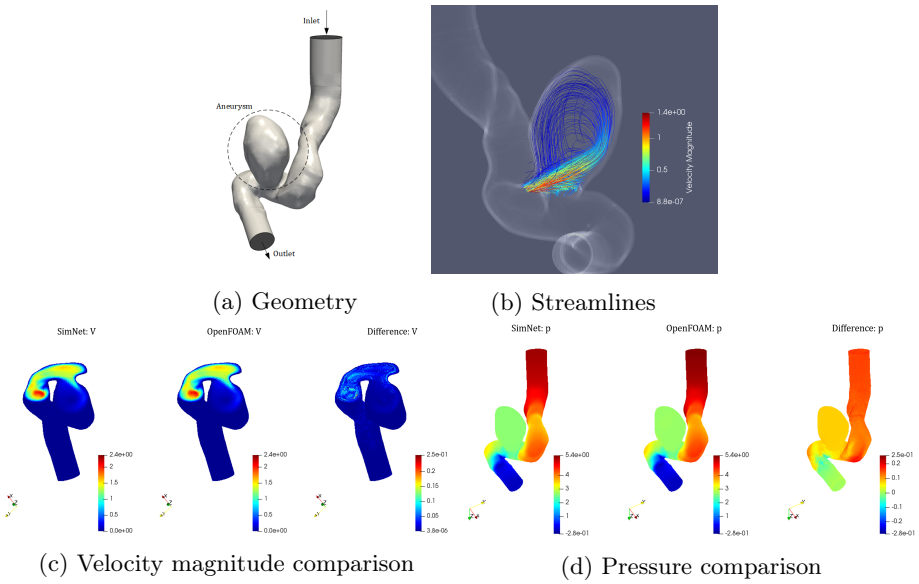
Table 1: FPGA pressure drop and peak temperature from various models.

Case Description	P_{drop} (Pa)	T_{peak} $(^{\circ}C)$
SimNet: Fourier network (axis spectrum)	25.47	73.01
SimNet: Fourier network (partial spectrum) with symmetry	29.03	72.36
SimNet: Modified Fourier network	29.17	72.52
SimNet: SiReN	29.70	72.00
OpenFOAM Solver	27.82	56.54
Commercial Solver	24.04	72.44

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

4.2 Blood flow in an Intracranial Aneurysm

We demonstrate the ability of SimNet to work with STL geometries from a CAD system. Using the SimNet's TG module, we simulate the flow inside a patient specific geometry of an aneurysm depicted in Figure 7a. The SimNet results for the distribution of velocity magnitude and pressure developed inside the aneurysm are shown in Figures 7c and 7d, respectively. Using the same geometry, the authors in [18] solve this as an inverse problem using concentration data from the spectral/hp-element solver Nektar. We solve this problem as a forward problem without any data. When solving the forward CFD problem with non-trivial geometries, one of the key challenges is getting the flow to develop correctly, especially inside the aneurysm sac. The streamline plot in Figure 7b shows that SimNet successfully captures the flow field very accurately.



(a) Geometry (b) Streamlines

(c) Velocity magnitude comparison (d) Pressure comparison

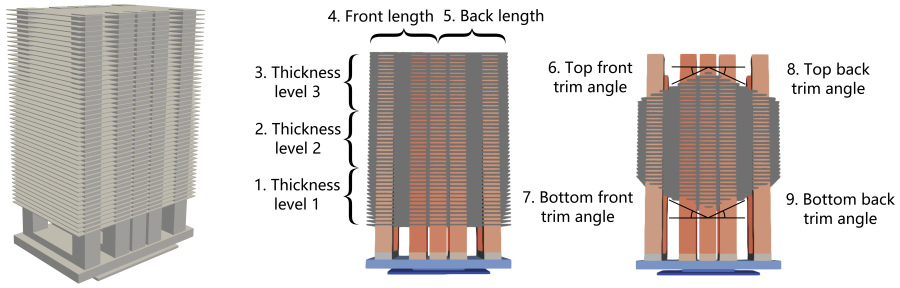
Fig. 7: SimNet results for the aneurysm problem, and a comparison between the SimNet and OpenFOAM results for the velocity magnitude and pressure.

4.3 Design optimization for multi-physics industrial systems

SimNet can solve several, simultaneous design configurations in a multi-physics, design space exploration problem much more efficiently than traditional solvers. This is possible because unlike a traditional solver, a neural network trains with multiple design parameters in a single training run. Once the training is complete, several geometry or physical parameter combinations can be evaluated using

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

inference as a post-processing step, without solving the forward problem again. Such throughput enables more efficient design optimization and design space exploration tasks for complex systems in science and engineering. Here, we train a conjugate heat transfer problem over the Nvidia's NVSwitch heat sink whose fin geometry is variable, as shown in Figure 8 (nine geometry variables in total). Details on the problem setup and training can be found in SimNet user guide. Forward solution of parameterized, complex geometry with turbulent fluid flow between thinly spaced fins and no training data makes this problem extremely challenging for the neural networks. Following the training, we perform a design optimization to find out the most optimal fin configuration that minimizes the peak temperature while satisfying a maximum pressure drop constraint. The fluid and heat neural networks in this example consist of 12 variables, i.e. three spatial variables and nine geometry parameter variables. Using SimNet, we train these two parameterized neural networks, and then use the trained models to compute the pressure drops and peak temperatures corresponding to 4 million random geometry realizations. Figure 9 shows the streamlines and temperature profile for the optimal NVSwitch geometry.



(a) Base geometry (b) Design parameters

Fig. 8: NVSwitch base geometry and design parameters.

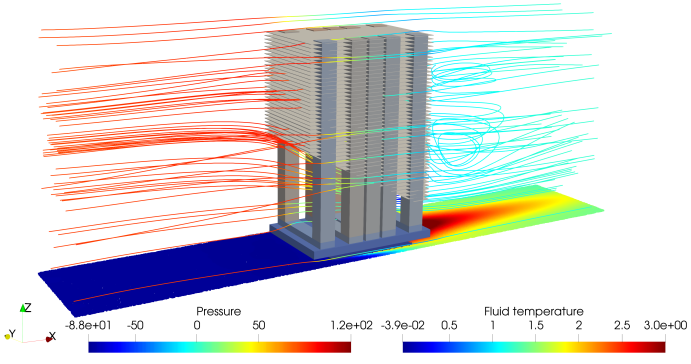


Fig. 9: SimNet results for the optimal NVSwitch geometry.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

By parameterizing the geometry, SimNet accelerates this design optimization task by several orders of magnitude when compared to traditional solvers, which are limited to single geometry simulations. This also suggests that SimNet can provide significant time savings when other design optimization methods, such as gradient-based design optimization, are used. The total compute time required by OpenFOAM, a commercial solver, and SimNet (including the training time) for this design optimization task is reported in Table 2. The OpenFOAM and commercial solver runs are run on 22 CPU processors, and the SimNet runs are on 8 V100 GPUs. To confirm the accuracy of the SimNet parameterized model, we take the NVSwitch base geometry and compare the SimNet results (obtained from the parameterized model) for pressure drop and peak temperature with the OpenFOAM and commercial solver results, reported in Table 3.

Table 2: Total compute time for the NVSwitch heat sink design optimization.

Solver	OpenFOAM	Commercial Solver	SimNet
Compute Time (x 1000 hrs.)	405935	137494	3

Table 3: A comparison for the solver and SimNet results for NVSwitch pressure drop and peak temperature.

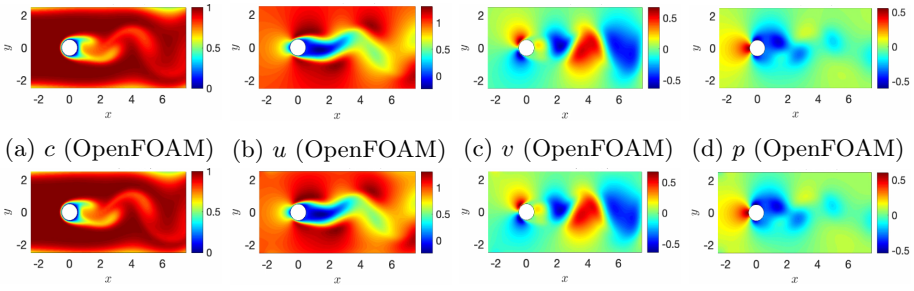
Property	OpenFOAM Single Run	Commercial Solver Single Run	SimNet Parameterized Run
Pressure Drop (Pa)	133.96	128.30	109.53
Peak Temperature ($°C$)	41.55	43.57	39.33

4.4 Inverse problems

Many applications in science and engineering involve inferring unknown system characteristics given measured data from sensors or imaging for certain dependent variables describing the behavior of the system. Such problems usually involve solving for the latent physics using the PDEs as well as the data. This is done in SimNet by combining the data with PDEs to decipher the underlying physics.

Here, we demonstrate the ability of SimNet to solve data assimilation and inverse problems on a transient flow past a 2D cylinder example. This example is adopted from [19]. Given the data consisting of the scattered concentration of a passive scalar in the flow domain at different times, the task is to infer the flow velocity and pressure fields as well as the entire concentration field of the passive scalar. In reality, the data is collected using measurements but for the purpose of this example, synthetic data generated by OpenFOAM is used. We construct a model with a hybrid data and physics-driven loss function. Specifically, we require the neural network prediction for the passive scalar concentration to fit to the measurements, and also satisfy the governing laws of the system that includes the transient Navier-Stokes and advection-diffusion equations. Here, the quantities of interest are also modeled as trainable variables, and are inferred by minimizing the hybrid loss function. A comparison between the SimNet results and the ground truth for a snapshot of the flow velocity, pressure, and passive scalar concentration fields is presented in Figure 10.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36



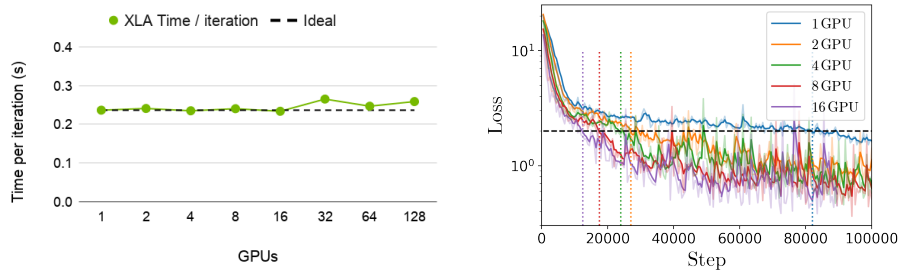
(a) c (OpenFOAM) (b) u (OpenFOAM) (c) v (OpenFOAM) (d) p (OpenFOAM)

(e) c (SimNet) (f) u (SimNet) (g) v (SimNet) (h) p (SimNet)

Fig. 10: A comparison between the SimNet and OpenFOAM results for a snapshot of the flow velocity, pressure, and passive scalar concentration fields. This example is adopted from [19].

5 Performance Upgrades and Multi-GPU Training

SimNet supports multi-GPU/multi-node scaling to enable larger batch sizes while time per iteration remains nearly constant, as shown in Figure 11a. Therefore, the total time to convergence can be reduced by scaling the learning rate linearly with the number of GPUs, as suggested in [20]. Doing so without a warmup would cause the model to diverge since the initial learning rate can be very large. Figure 11b shows the Limerock results for large batch training acceleration on A100 using learning rate scaling. SimNet also supports TensorFloat-32 (TF32), a new math mode available on NVIDIA A100 GPUs. Based on our experiments on the FPGA problem, using TF32 provides up to 1.6x and 3.1x speed-up over FP32 on A100 and V100 GPUs, respectively. Moreover, SimNet supports kernel fusion using XLA that, based on our experiments, can accelerate a single training iteration in SimNet by up to 3.3x.



(a) Time per iteration weak scaling (b) Large batch training acceleration

Fig. 11: SimNet's scaling performance results.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

6 Conclusion

SimNet is an end-to-end AI-driven simulation framework with unique, state-of-art architectures that enables rapid training of forward, inverse, and data assimilation problems for real world geometries and multiple physics types with or without any training data. SDF is used for loss weighting, which is shown to significantly improve the convergence in cases where the geometry has sharp corners and results in sharp solution gradients. SimNet's TG module enables the import tessellated geometries from CAD programs. For channel flow problems, continuity is imposed globally and locally to further improve the convergence and accuracy. SimNet enables the neural network solvers to simulate high Reynolds number flows for industrial applications. To the authors knowledge, this is the first such application of neural network solvers for RANS simulation of turbulent flows.

SimNet is designed to be flexible so that users can leverage the functionality in the existing toolkit and focus on solving their problem well rather than re-creating the tools. There are various APIs that enable the user to implement their own equations to simulate the physics, their own geometry primitives or importing complex tessellated geometries, or a variety of domains/boundary conditions. The geometry parameterization in the CSG module allows the neural network to address the entire range of all given parameters in a single training, as opposed to the traditional simulations that run one at a time. The inference for any design configuration can then be completed in real time. This accelerates the simulation with neural network solvers by orders of magnitude.

Acknowledgments

We would like to thank Doris Pan, Anshuman Bhat, Rekha Mukund, Pat Brooks, Gunter Roth, Ingo Wald, Maziar Raissi, Jose del Aguila Ferrandis, and Sukirt Thakur for their assistance and feedback in SimNet development. We also acknowledge Peter Messemer, Mathias Hummel, Tim Biedert and Kees Van Kooten for integration with Omniverse.

References

1. Xiaoxiao Guo, Wei Li, and Francesco Iorio. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 481–490, 2016.
2. Oliver Hennigh. Lat-net: compressing lattice boltzmann flow simulations using deep neural networks. *arXiv preprint arXiv:1705.09036*, 2017.
3. Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.
4. Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36

5. Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.
6. Ehsan Kharazmi, Zhongqiang Zhang, and George Em Karniadakis. hp-vpinns: Variational physics-informed neural networks with domain decomposition. *arXiv preprint arXiv:2003.05385*, 2020.
7. Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.
8. Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks. *arXiv preprint arXiv:2001.04536*, 2020.
9. Lu Lu, Xuhui Meng, Zhiping Mao, and George E Karniadakis. Deepxde: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.
10. Ehsan Haghighat and Ruben Juanes. Sciann: A keras wrapper for scientific computations and physics-informed deep learning using artificial neural networks. *arXiv preprint arXiv:2005.08803*, 2020.
11. Christopher Rackauckas and Qing Nie. Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia. *The Journal of Open Research Software*, 5(1), 2017. Exported from https://app.dimensions.ai on 2019/05/05.
12. David C Wilcox et al. *Turbulence modeling for CFD*, volume 2. DCW industries La Canada, CA, 1998.
13. Aaron Meurer, Christopher P Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K Moore, Sartaj Singh, et al. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103, 2017.
14. Vincent Sitzmann, Julien NP Martel, Alexander W Bergman, David B Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *arXiv preprint arXiv:2006.09661*, 2020.
15. Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred Hamprecht, Yoshua Bengio, and Aaron Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310, 2019.
16. Matthew Tancik, Pratul P Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains. *arXiv preprint arXiv:2006.10739*, 2020.
17. Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.
18. Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.
19. Maziar Raissi, Zhicheng Wang, Michael S Triantafyllou, and George Em Karniadakis. Deep learning of vortex-induced vibrations. *Journal of Fluid Mechanics*, 861:119–137, 2019.
20. Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

ICCS Camera Ready Version 2021
To cite this paper please use the final published version:
DOI: 10.1007/978-3-030-77977-1_36