Deep Generative Models that Solve PDEs

Distributed Computing for Training Large Data-Free Models

Sergio Botelho[1], Ameya Joshi[3], **Biswajit Khara**[2], Vinay Rao[1], Soumik Sarkar[2], Chinmay Hegde[3], Santi Adavani[1], Baskar Ganapathysubramanian[2]

rocketML [1]

IOWA STATE UNIVERSITY [2]

NYU [3]

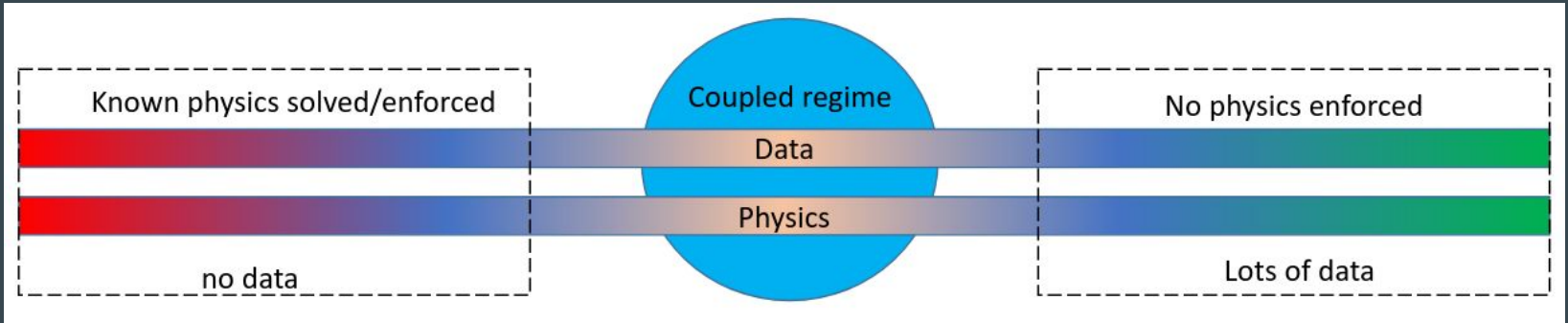
Solving PDEs with deep learning

- $\mathcal{N}(\mathbf{x}; u, \{\mathcal{D}u\}; \mathbf{s}(\boldsymbol{\omega})) = 0$ in $(\mathbf{x},\boldsymbol{\omega}) \in D \times \Omega$
- $\mathcal{B}(\mathbf{x}, \boldsymbol{\omega}, u) = 0$ on $\mathbf{x} \in \partial D \times \Omega$

- Analytical: $u = f(\mathbf{x};\boldsymbol{\omega})$
- Discrete: $\underline{U} = F_D(\underline{\mathbf{x};\boldsymbol{\omega}})$

- Numerical methods
  - Finding the "closest" function
  - Discretize → Calc derivatives → (optional) Integrate → Linear algebra problem: $\mathbf{Ax = b}$ → Solve
- Machine learning
  - Finding an approximate function / mapping
  - Setup an objective function: $\min_x ||\mathbf{Ax - b}||^p$ → Optimize
- Stochastic PDE
- Design
- Simulations can take very long time even on modern day supercomputers
- Can we leverage the machine learning advances to speed up the process of solving PDEs on supercomputers?

Deep learning constrained by physical laws

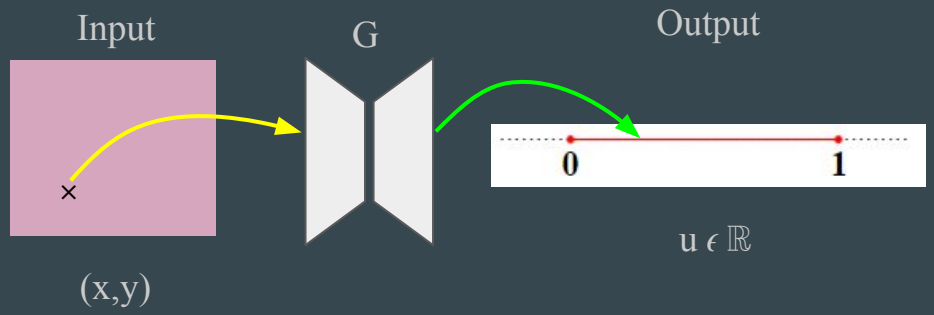


- Conventional applications of deep learning
  - Reliance on abundance of data
  - Lack of generalizability
- Application to areas where a physical law needs to be respected
  - Integrate physical law with the model
  - PDE residual models the loss function

Formulation approaches

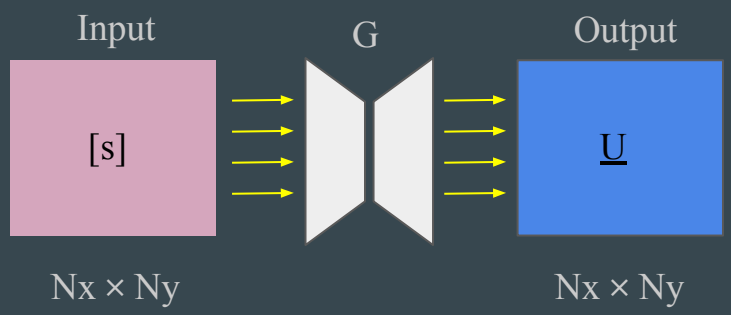
- PDE instance
  - $\mathcal{N}(\mathbf{x}; u, \{\mathcal{D}u\}; \mathbf{s}) = 0$ in $\mathbf{x} \in D$
  - $\mathcal{B}(\mathbf{x}, u) = 0$ on $\mathbf{x} \in \partial D$
- Pointwise predictions

$(x, y, t) \longrightarrow f_{NN}([s]) \longrightarrow u$



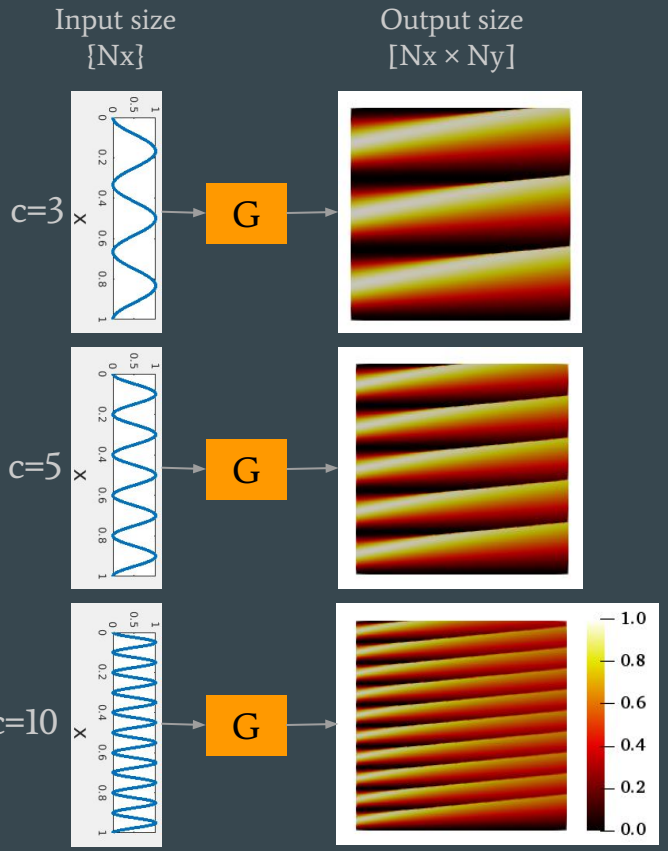
- PDE family solvers
  - $\mathcal{N}(x; u, \{\mathcal{D}u\}; s(\omega)) = 0$ in $(x,\omega) \in D \times \Omega$
  - $\mathcal{B}(x, \omega, u) = 0$ on $x \in \partial D \times \Omega$
- Full-field predictions

$[s]^I \longrightarrow G_{NN} \longrightarrow \underline{U}^I$



This work - *DiffNet*

- We attempt to solve a family of parametric PDEs
- A stochastic Burgers' equation:
  - $u_t + uu_x = 0$ in (x,t) in [0,1]×[0,⅛]
  - Boundary condition: u(x=0, t) = 0
  - Initial condition: u(x, t=0) = ½ (1 - *cos* 2π**c**x)
- Learned model:
  - IC → G_{NN} → $\underline{U}$
- Multiscale problems demand high resolution, locally or globally
- A full field approach, especially when aiming for space-time problems can be demanding in memory consumption



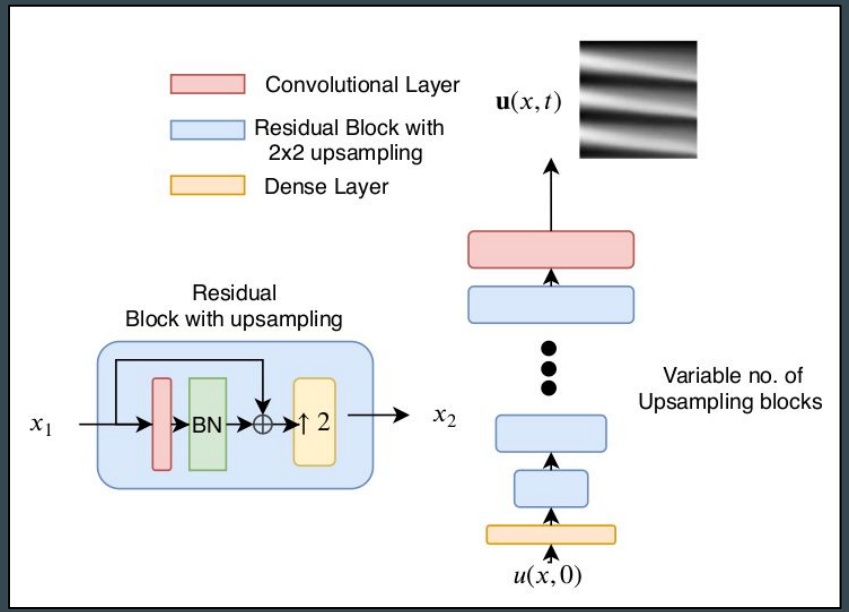
DiffNet problem formulation

Solving PDE ⇒ training a **2D convolutional generative neural network** G_θ

- **Input:** initial condition u(x, 0)
- **Output:** full-field solution u(x, t)
- **Loss function:** PDE residual + initial/boundary conditions

$$
\begin{aligned}
L &= L_p + \lambda L_b, \\
L_p(\theta) &= \mathbb{E}_{\mathbf{b},\nu}[\|\mathcal{A}_\nu(G_\theta(\mathbf{b},\nu)) - f\|_2^2], \\
L_b(\theta) &= \mathbb{E}_{\mathbf{b}}\|\mathcal{B}(G_\theta(\mathbf{b},\nu)) - \mathbf{b}\|_2^2].
\end{aligned}
$$

PDE: $\mathcal{A}_\nu(\mathbf{u}) = f, \quad \mathcal{B}(\mathbf{u}) = \mathbf{b}$



Implementation of forward model

- k^{th} order derivatives approximated by **convolutions** with finite-difference kernels:

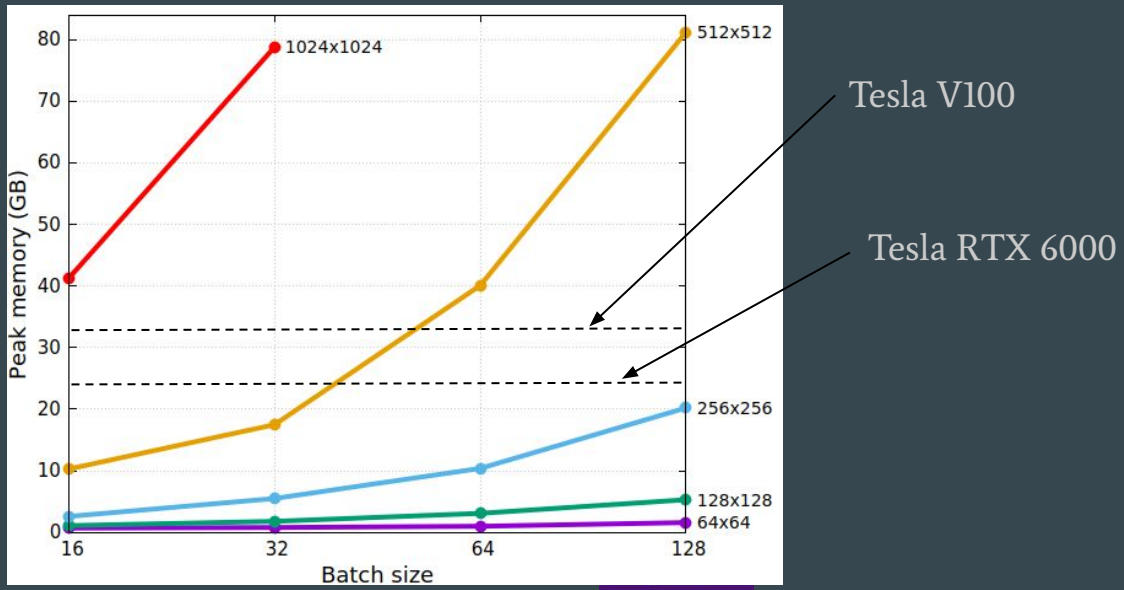
$$\nabla^k_{(x,t)} \approx u(x,t) \star S^k_{(x,t)}$$

- For 1^{st} order derivatives, we use 3×3 Sobel kernels:

$$S^1_x = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix} \quad , \quad S^1_t = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}$$

Challenges of training on GPUs

- GPUs are the most popular compute platform for training DNNs
- Known limitation: they have relatively small available memory
  - Training is usually done with very small mini-batches, which hinders convergence
  - Worse yet, often times training is done on single GPU, impacting productivity
- **Training DiffNet on domain sizes > 512 × 512 is not feasible on current GPUs!**



Introducing *DeepFusion*

Platform-agnostic software framework for large-scale distributed deep-learning:

- Extends memory capacity way beyond GPU limits while delivering excellent strong scaling
- **Strategy:** distributed training of DiffNet on CPU clusters
  - 5-10x more memory-per-node compared to GPUs
  - Multiple cores-per-node connected via high-end low-latency interconnects
  - Substantially cheaper price tag per node

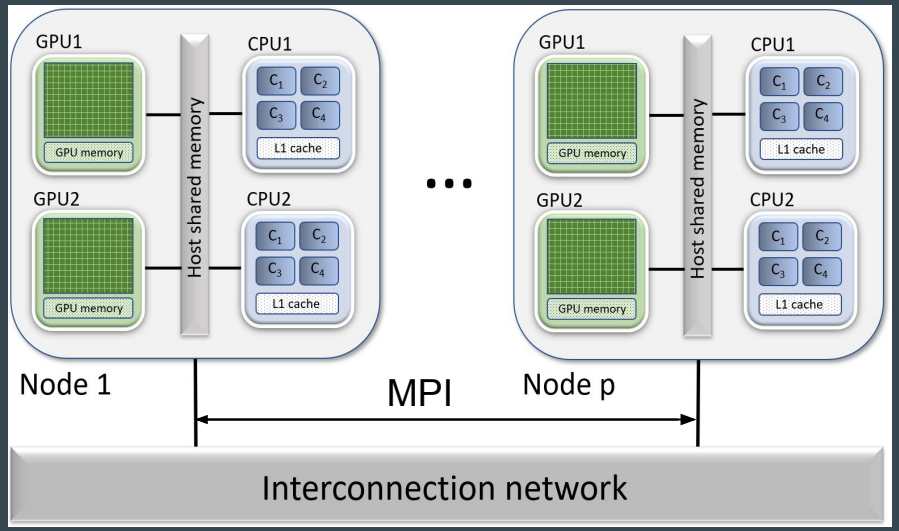
Specification	AWS	Azure	Stampede2
Type	Virtual Machine	Virtual Machine	Bare-Metal
CPU	Intel Xeon Platinum 8000	Intel Xeon Platinum 8168	Intel Xeon Platinum 8160
CPU cores	72	44	48
Memory (GB)	192	352	192
Interconnect	Elastic Fabric Adapter	EDR Infiniband	Intel Omni-Path
Bandwidth	100 Gb/sec	100 Gb/sec	100 Gb/sec
Topology	AWS Proprietary	Fat tree	Fat tree

Domain Size	Nodes	AWS	Azure	Stampede2
64x64	1	131.0	113.1	67.2
64x64	2	65.2	54.9	34.9
64x64	4	32.4	28.6	19.4
128x128	4	138.4	126.2	68.5
256x256	4	650.5	597.6	279.8

Per-epoch wall-clock times
(Batch size 1024; 4 processes per node, 8 threads per process)

DeepFusion: programming paradigm

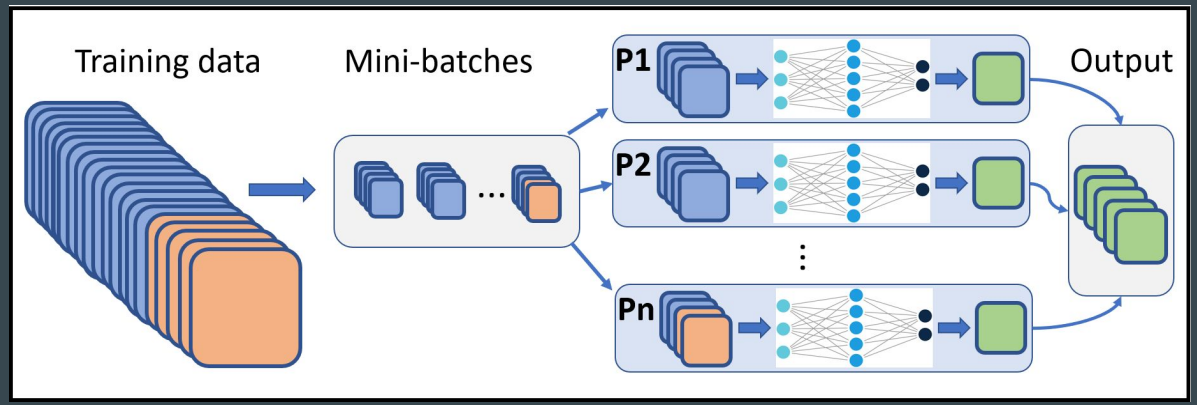
- Uses hybrid OpenMP + MPI programming for efficient intra/inter-node communication
- Designed with parallelization in mind ("scaling as a first-class citizen")
- Leverages Intel MKL-DNN for fast forward/backward propagation



DeepFusion: data-parallel strategy

Multiple replicas of model are simultaneously trained to optimize a single objective function:

- Mini-batches are equally split among available workers
- Forward & back-propagation are performed asynchronously; gradients are *MPI_AllReduce*'d

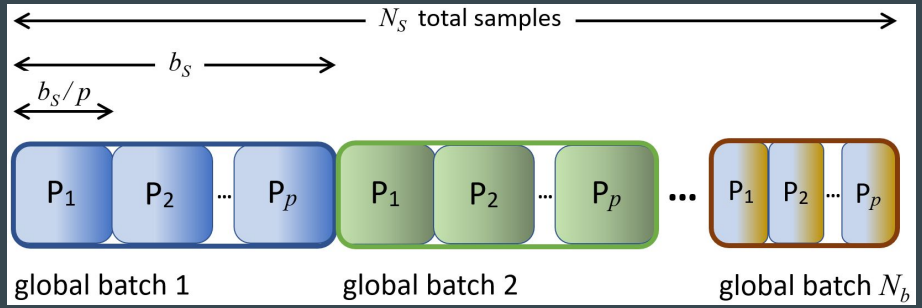


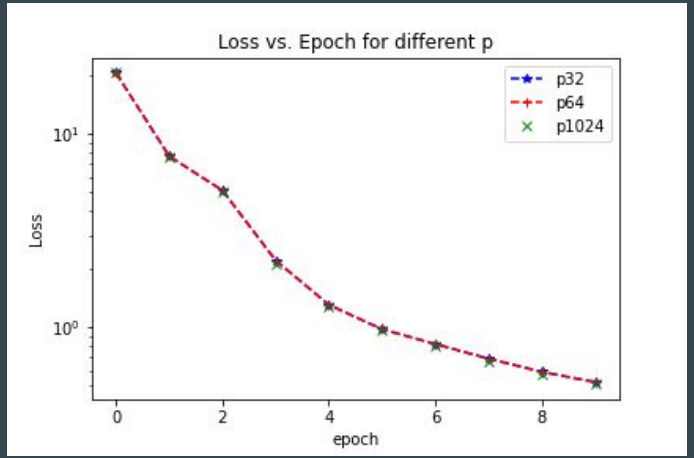
Data-parallel strategy (cont'd)

Solution robustness to parallelization: same problem is solved for any ***p***

- Local mini-batches are drawn sequentially from sample pool by each worker
- Equal-size mini-batches across workers guarantees optimal load-balancing

$$N_s^{loc} = \lceil N_s/p \rceil \,, \; b_s^{loc} = \lfloor b_s/p \rfloor$$



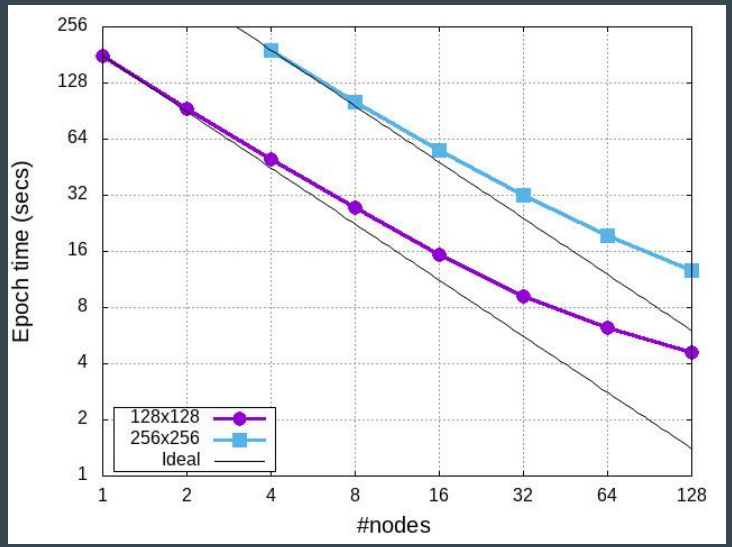


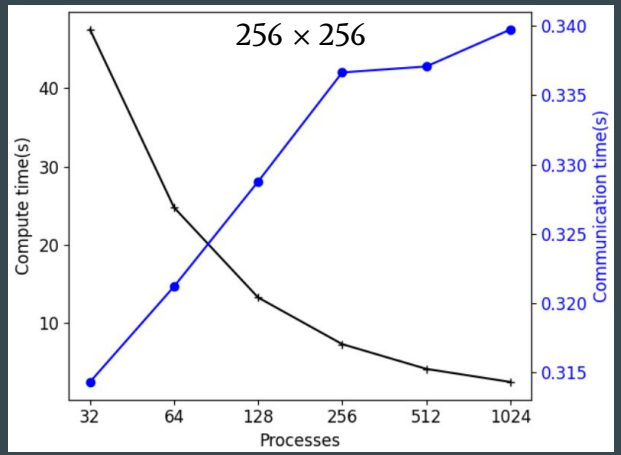
(DiffNet training on 256 × 256 domain size)

Scaling experiments

DiffNet training on 1 - 128 nodes (8 - 1024 processes) of **Stampede2**

- 8 processes per node, 12 threads per process (on 96 available hardware threads)
- Batch size 1024 (4096 training examples)





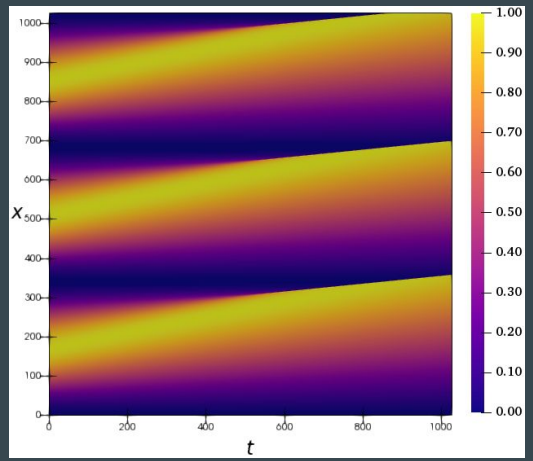
Communication complexity = $O(N_w + \log p), \ N_w \gg p.$

High-resolution DiffNet

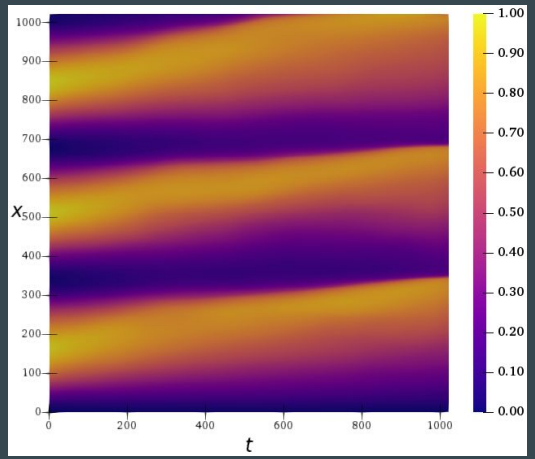
DiffNet on **1024 × 1024 domain size** (not previously done for such generative models):

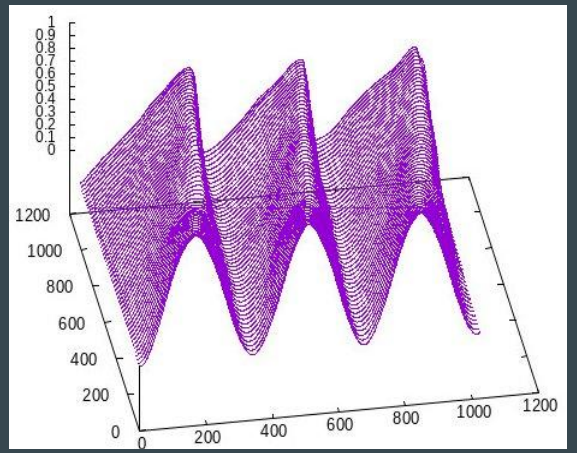
- Trained for a range of the initial condition parameter c ∈ [3, 6]
- 256 training examples; batch size 64
- 8 nodes of Stampede2 (8 processes-per-node)
- 2200 epochs until convergence (32hrs); Adam optimizer

FEM solver



DiffNet inference

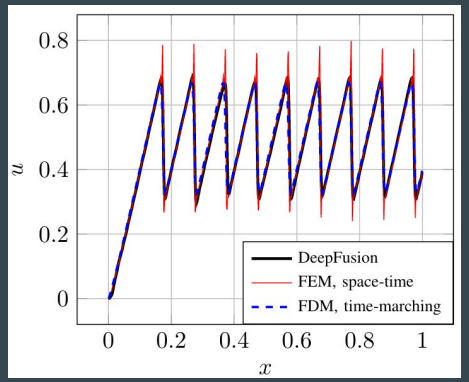




High-resolution DiffNet (cont'd)

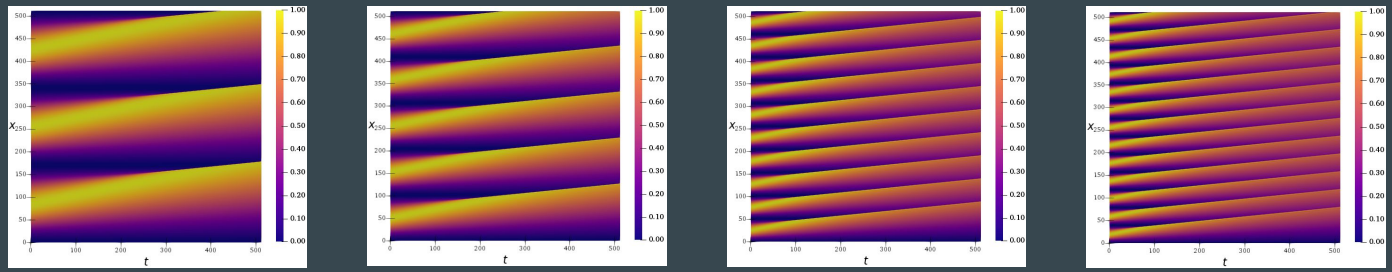
DiffNet on **512 × 512 domain size**:

- Larger distribution of initial conditions c ∈ [3, 16]
- 256 training examples; batch size 64
- 8 nodes of Stampede2 (8 processes-per-node)
- 4000 epochs until convergence (15hrs); Adam optimizer

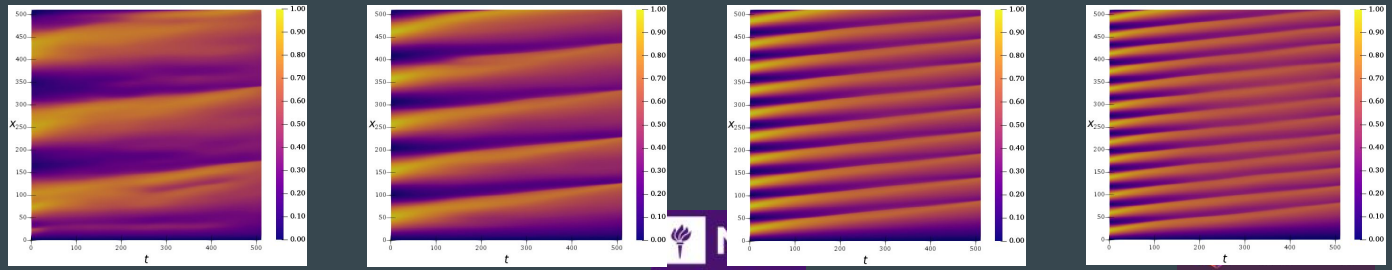


c = 3 | c = 5 | c = 10 | c = 13

FEM solver



DiffNet inference



rocketML

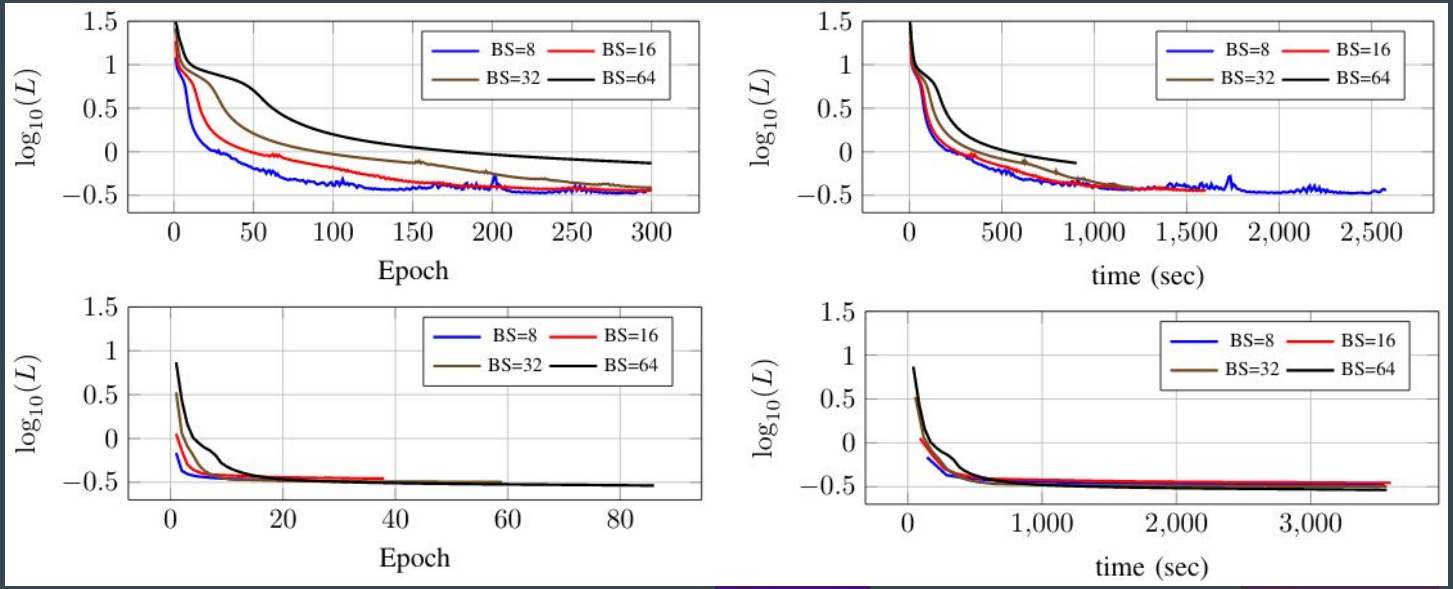
Second-order optimizer

Large-scale parallelism afforded by DeepFusion enables large batch sizes

- Higher-order optimization methods like L-BFGS benefit from large batches
- Larger memory required to evaluate the Hessian is gracefully accommodated by DeepFusion
- Training converges **2-3× faster** (15× fewer epochs) than SGD

SGD

L-BFGS



Conclusions and future work

- Proposed DiffNet, a data-free neural-network-based strategy to solve PDEs:
  - Applied to the solution of the inviscid Burgers' PDE with a parametric family of ICs

- Introduced DeepFusion, a software framework to train very large neural networks:
  - Proposed distributed training on CPU clusters to overcome GPU memory limitations
  - Demonstrated excellent scaling and accuracy on cloud-based and bare-metal infrastructures
  - Showed how 2nd-order optimizers can further improve convergence and training time

- Future work:
  - Other 3D PDEs (Navier-Stokes, wave eqs.)
  - Alternative loss functions (e.g. weighted losses)
  - Model-parallel strategy

- Acknowledgements:
  - Support from NSF XSEDE
  - ARPA-E DIFFERENTIATE program

rocketML IOWA STATE UNIVERSITY NYU SC20

Questions

BACKUP

DiffNet inference time

- Inference time is often very fast:
  - From practitioner perspective, time-to-solve **is** the time for inference
  - Training cost is large, but amortized over multiple users and instances

Domain Size	FEM (seconds)	DeepFusion (seconds)
512×512	23.2	3.6
1024×1024	395.6	9.8

DiffNet inference time vs. FEM solve time on single node

Effect of weighted loss function

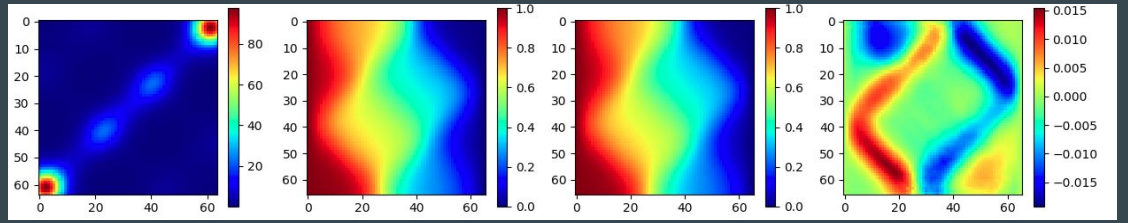
- Poisson equation with variable diffusivity

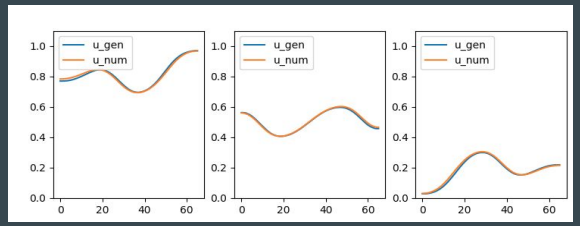
$$\begin{aligned} \underline{\nabla} \cdot (\tilde{v}(\underline{x})\underline{\nabla} u) &= 0 \text{ in } D \\ u(0, y) &= 1 \\ u(1, y) &= 0 \\ \frac{\partial u}{\partial n} &= 0 \text{ on other boundaries} \end{aligned}$$

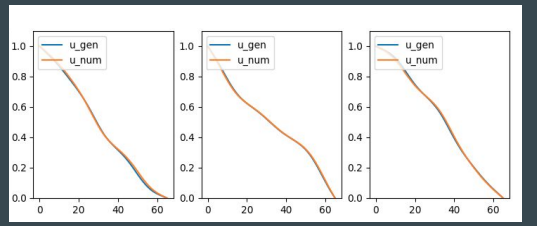
- Loss function

$$R = \int \tilde{v} |\underline{\nabla} u|^2 d\underline{x}$$

input (diffusivity ν) | output (solution u) | Numerical solution | pointwise difference







x = 0.2 | x = 0.5 | x = 0.8 | y = 0.2 | y = 0.5 | y = 0.8

Effect of weighted loss function

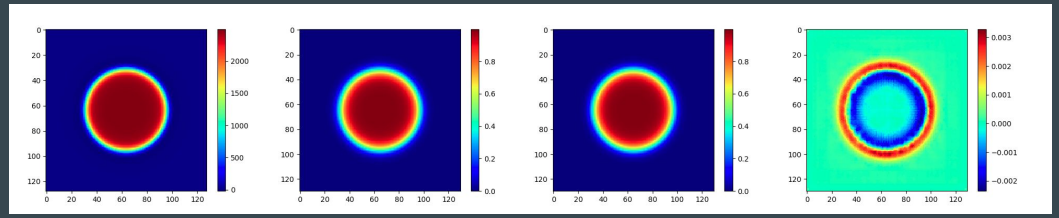
- Reaction diffusion equation

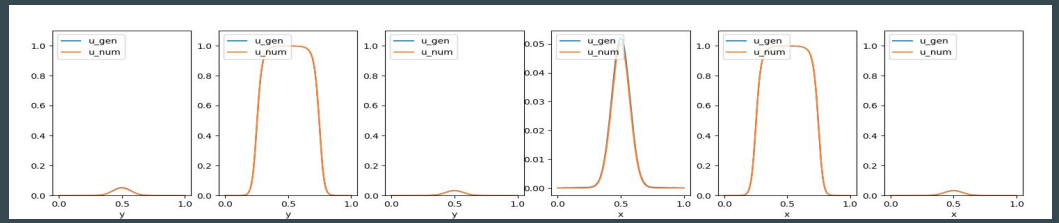
$$-\underline{\nabla} \cdot (\nu(\underline{x})\underline{\nabla} u) + \kappa^2 u = f \text{ in } D$$
$$u|_{\partial\Omega} = 0$$

- Loss function

$$R = \int \left[\nu|\underline{\nabla} u|^2 + \kappa^2 u^2 - fu\right] d\underline{x}$$

input (forcing) | output (solution u) | Numerical solution | pointwise difference





x = 0.2 | x = 0.5 | x = 0.8 | y = 0.2 | y = 0.5 | y = 0.8