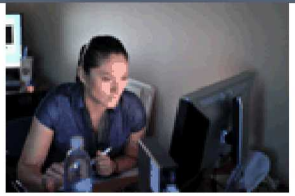
This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2019-15486C

Improved Neural Network Training: Layer-Parallelism, Least-squares and Initialization



Authors

Eric C. Cyr, Mamikon Gulian, Ravi Patel, Mauro Perego, Nat Trask, Denis Ridzal (SNL), Stefanie Guenther (LLNL), Lars Ruthotto (Emory), Jacob B. Schroder (UNM), Nico R. Gauger (TU Kaiserslautern)

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

What is a neural network?

- A single ResNet layer (for $l = 0 \dots L-1$):

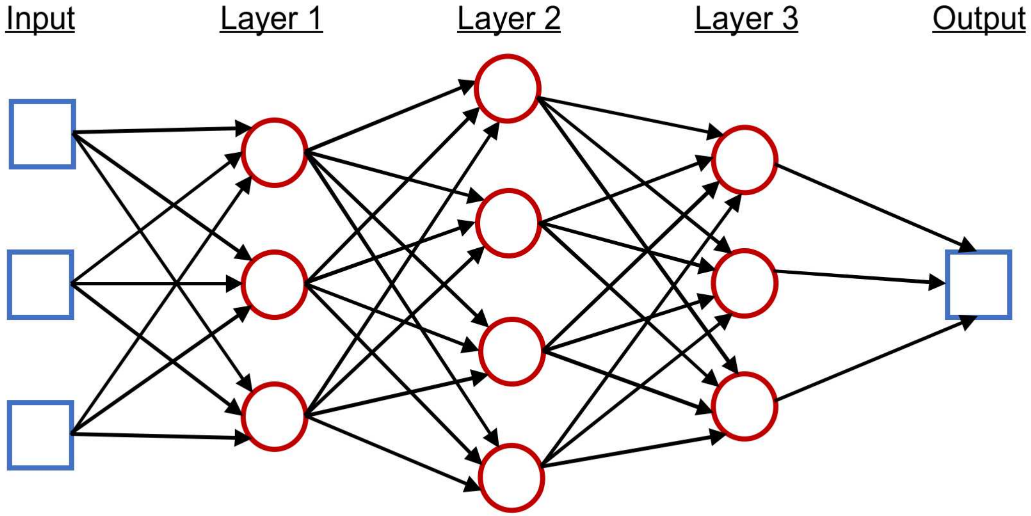
$$\mathbf{x_{k+1}} = \mathbf{x_l} + \sigma(\mathbf{W_l}\mathbf{x_l} + \mathbf{b_l})$$

- We also use ODE Networks:

$$\frac{\partial \boldsymbol{x}}{\partial t} = \sigma(\boldsymbol{W}(t)\, \boldsymbol{x}(t) + \boldsymbol{b}(t))$$

- Apply a NN to a data element $\boldsymbol{x}_0 = \boldsymbol{y}_d$:

$$\mathcal{NN}_\xi(\mathbf{x}_0) = \mathbf{x}_L \text{ where } \xi = \{\mathbf{W}_l, \mathbf{b}_l\}_{l=0}^{L-1}$$



Questions for this talk:

1. How do you select an initial set of weights and biases?
2. How do you accelerate the training with parallel computing?

Two Types of problems

Regression

Observations:

$$(\mathbf{y}_n, u_n) \in \mathbb{R}^d \times \mathbb{R}^1$$

$$n = 1 \dots N$$

$$\mathcal{X} = \{\mathbf{y}_n : n = 1 \dots N\}$$

Loss: Mean-Square Error

$$\operatorname*{argmin}_{\xi} \|u - \mathcal{N}\mathcal{N}_{\xi}\|^2_{\ell_2(\mathcal{X})}$$

Classification

Observations:

$$(\mathbf{y}_n, \mathbf{c}_n) \in \mathbb{R}^d \times \{0,1\}^K$$

$$n = 1 \dots N$$

Loss: Cross-Entropy

$$\operatorname*{argmin}_{\xi} \; -\sum_{d}^{D} \sum_{k=1}^{K} c_{d,k} \log(\mathcal{N}\mathcal{N}_{\xi}(y_d))$$

How to select an initial weights and biases?

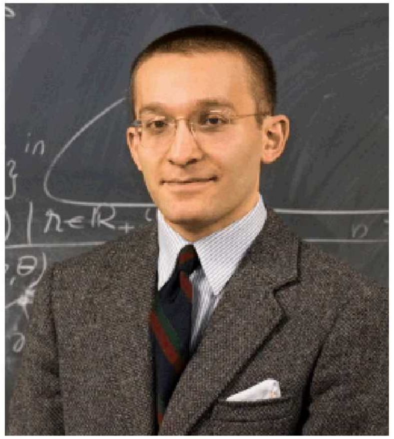
Problem statement:

1. With ReLU activation functions, does the initial choice of weights and biases have a strong impact on training? (yes)
2. Can you improve the training of the network by carefully selecting weights and biases? (yes)

Previous work (both in TensorFlow and pyTorch):

- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, 9:249–256, 13–15 May 2010.

Ravi Patel
SNL Postdoc



Mamikon Gulian
SNL JvN Postdoc

Nat Trask
SNL

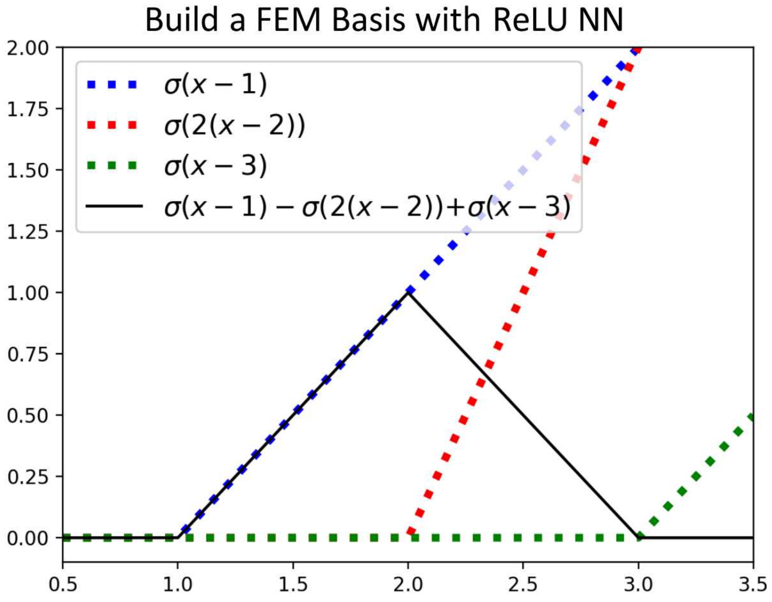
Mauro Perego
SNL

A View on ReLU Neural Networks

From: “J. He, L. Li, J. Xu, C. Zheng, ReLU deep neural networks and linear finite elements, arXiv preprint arXiv:1807.03973, 2018.”

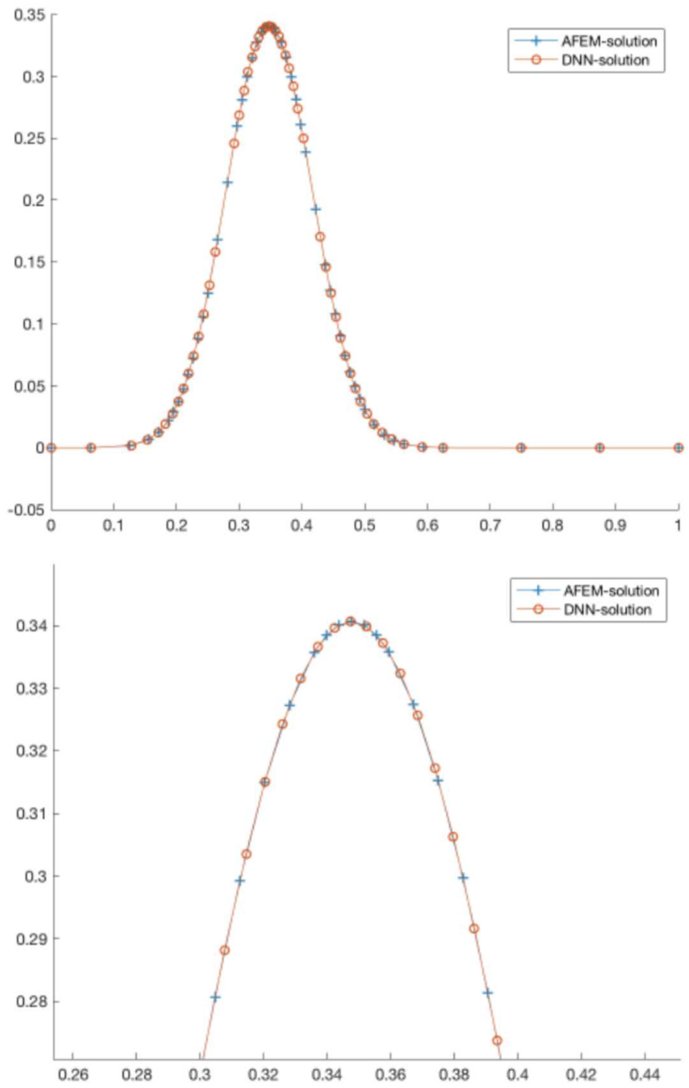
- ReLUs permit a continuous p-w linear approximation

$$\sigma(Wx+b) = max(0, \underbrace{Wx}_{\text{Slope}} + \underbrace{b}_{\text{0-Intersection}})$$



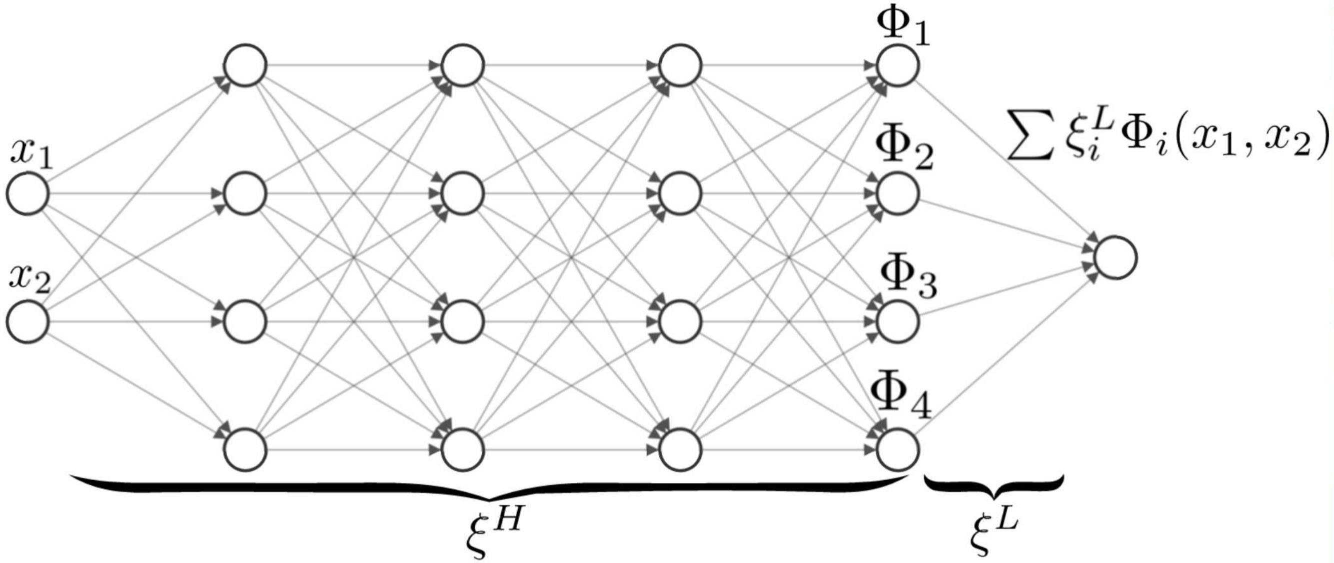
- Training ReLU NN yields an r-ref. like FE method

AFEM, versus DNN-FEM



Regression: An Adaptive Basis Viewpoint

$$\underset{\xi}{\operatorname{argmin}} \|u - \mathcal{NN}_{\xi}\|^2_{\ell_2(\mathcal{X})}$$



Neural network:

1. Width d: Input layer
2. Width w: hidden layers define adaptive basis
3. Linear coefficients: Output layer

- Adaptive viewpoint: references

$$\mathcal{NN}_{\boldsymbol{\xi}}(\boldsymbol{x}) = \sum_{i=1}^{w} \xi_i^{\mathrm{L}} \Phi_i(\boldsymbol{x}; \boldsymbol{\xi}^{\mathrm{H}})$$

To train we adopt a hybrid Least-squares/Gradient descent method (LSGD)

- Look at weight initialization

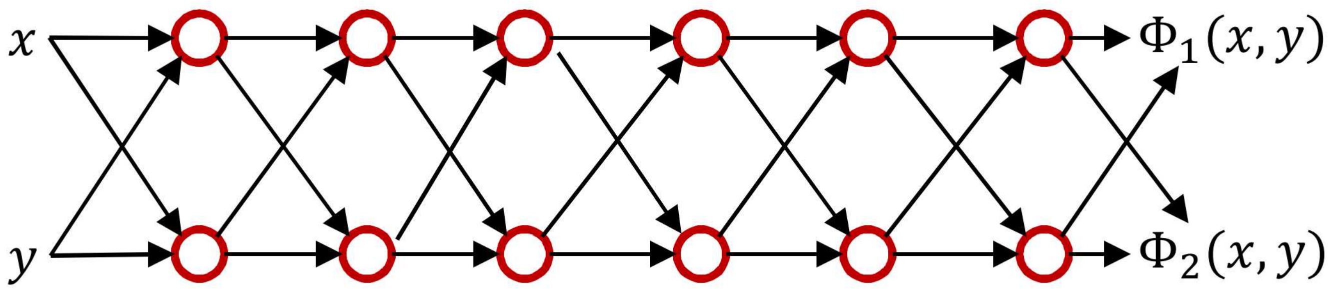
```
function LSGD($\boldsymbol{\xi}_0^H$)
    $\boldsymbol{\xi}^H = \boldsymbol{\xi}_0^H$        Initialize Weights
    $\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$
    for $i = 1 \ldots$ do
        $\boldsymbol{\xi}^H = GD(\boldsymbol{\xi})$
        $\boldsymbol{\xi}^L = LS(\boldsymbol{\xi}^H)$
    end for
end function
```

He Initialization: Plain Neural Networks

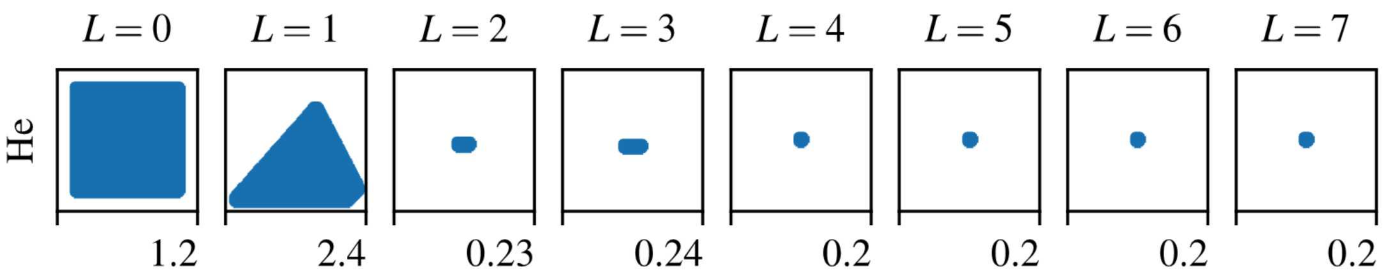
Use “He” initialization on a plain DNN

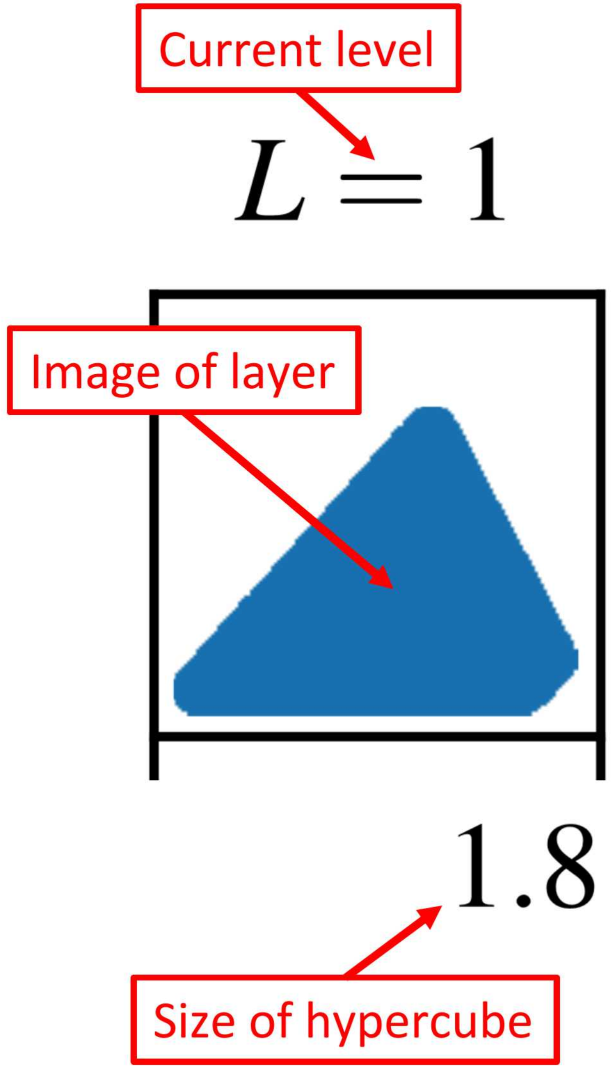
- He* is a (the) standard techniques (both pytorch and TensorFlow)
- Was designed for ReLU networks with batch normalization
- He does not modify the bias (sets it to zero)

Propagate $[0,1]^2$ through the neural network



Plot the image of $[0,1]^2$ through all layers





Current level

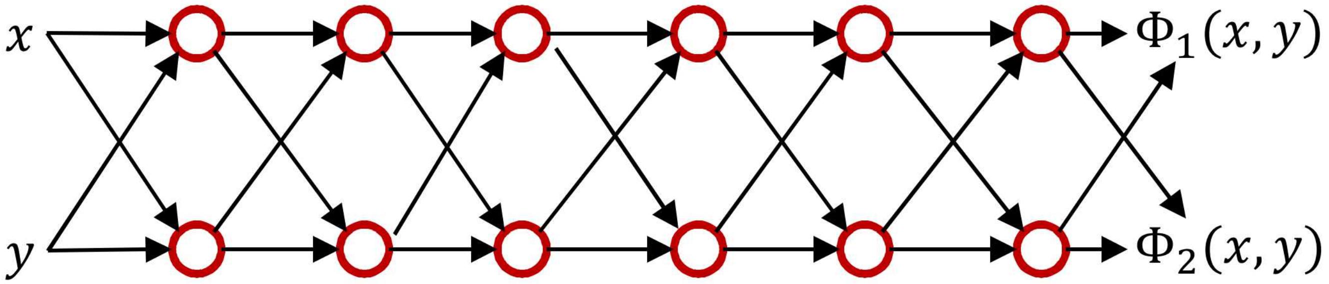
Image of layer

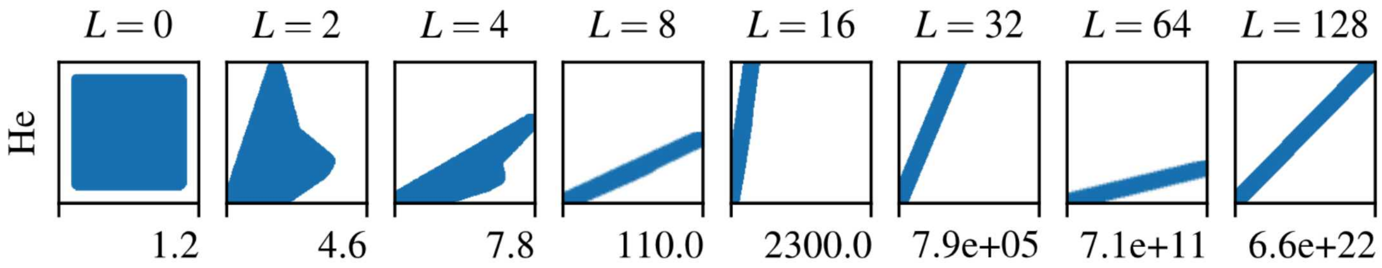
Size of hypercube

*He, K., Zhang, X., Ren, S., & Sun, J. (2015). In *Proceedings of the IEEE international conference on computer vision* (pp. 1026-1034).

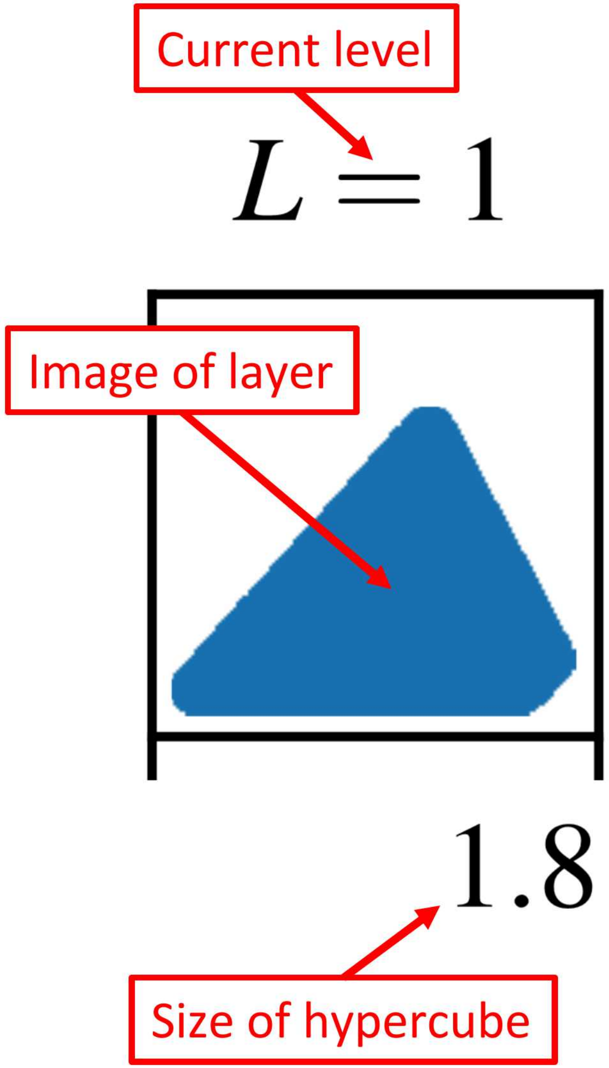
He Initialization: ResNets

Propagate $[0,1]^2$ through the neural network





- ResNets can get much deeper (128 layers)
- No collapse to a point, but it does collapse to a line
- Really large growth: 10^{22} (yikes!)



Current level

Image of layer

Size of hypercube

Our Approach: “Box” Initialization (ReLU-ResNets)

Goals:

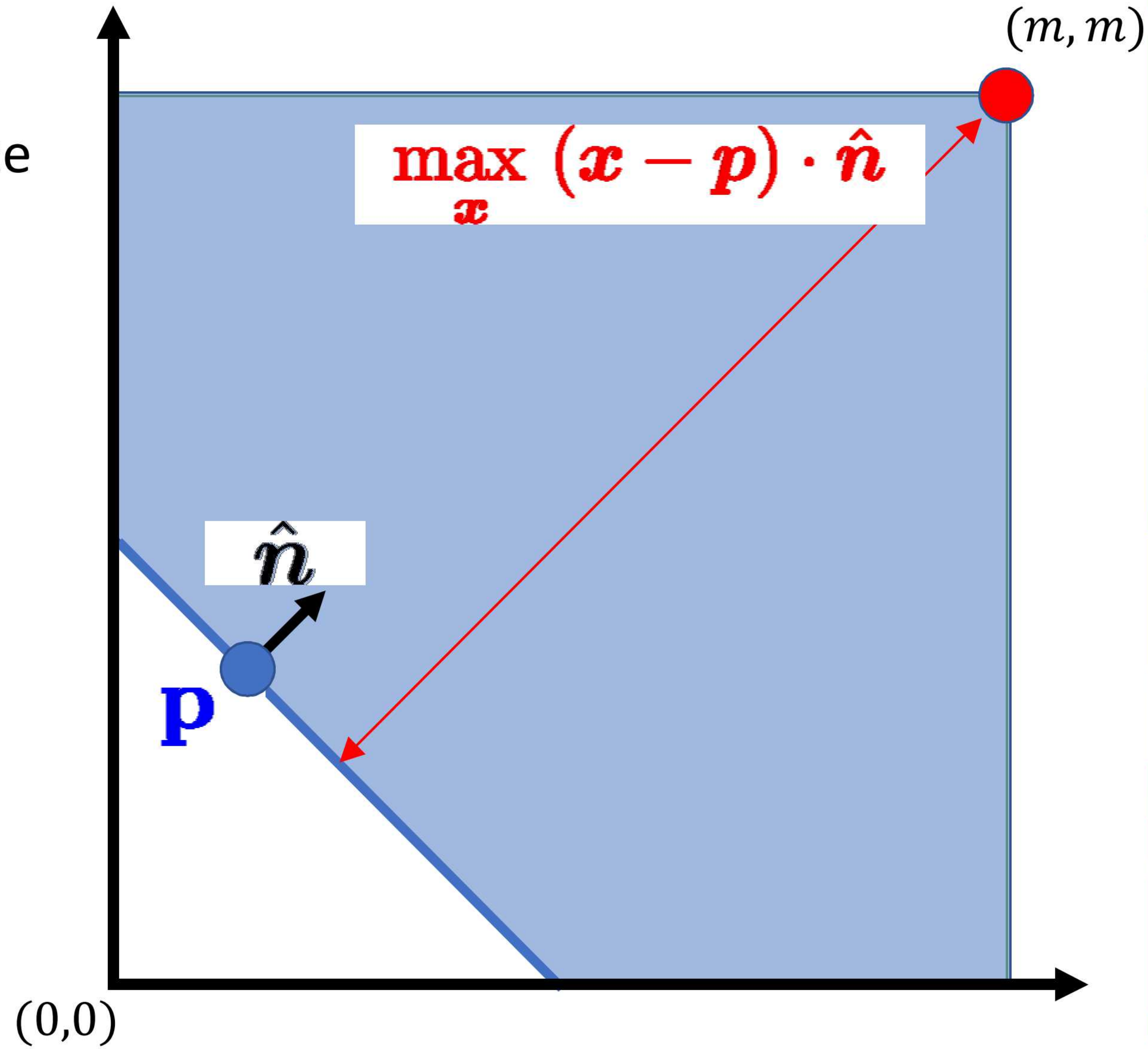
- Remain Bounded
- Don’t Collapse: Requires growth of cell size
- Keep cut-plane is in cell at each layer

For each row of a layer (l=1…L):

1. Choose a point $\boldsymbol{p} \in [0, m]^w$
2. Choose a random normal
3. Select a scaling k such that
$$\max_{x \in [0,m]^w} \sigma(k(\boldsymbol{x} - \boldsymbol{p}) \cdot \boldsymbol{n}) = \mathrm{m}L^{-1}$$

Affine trans defined row wise:

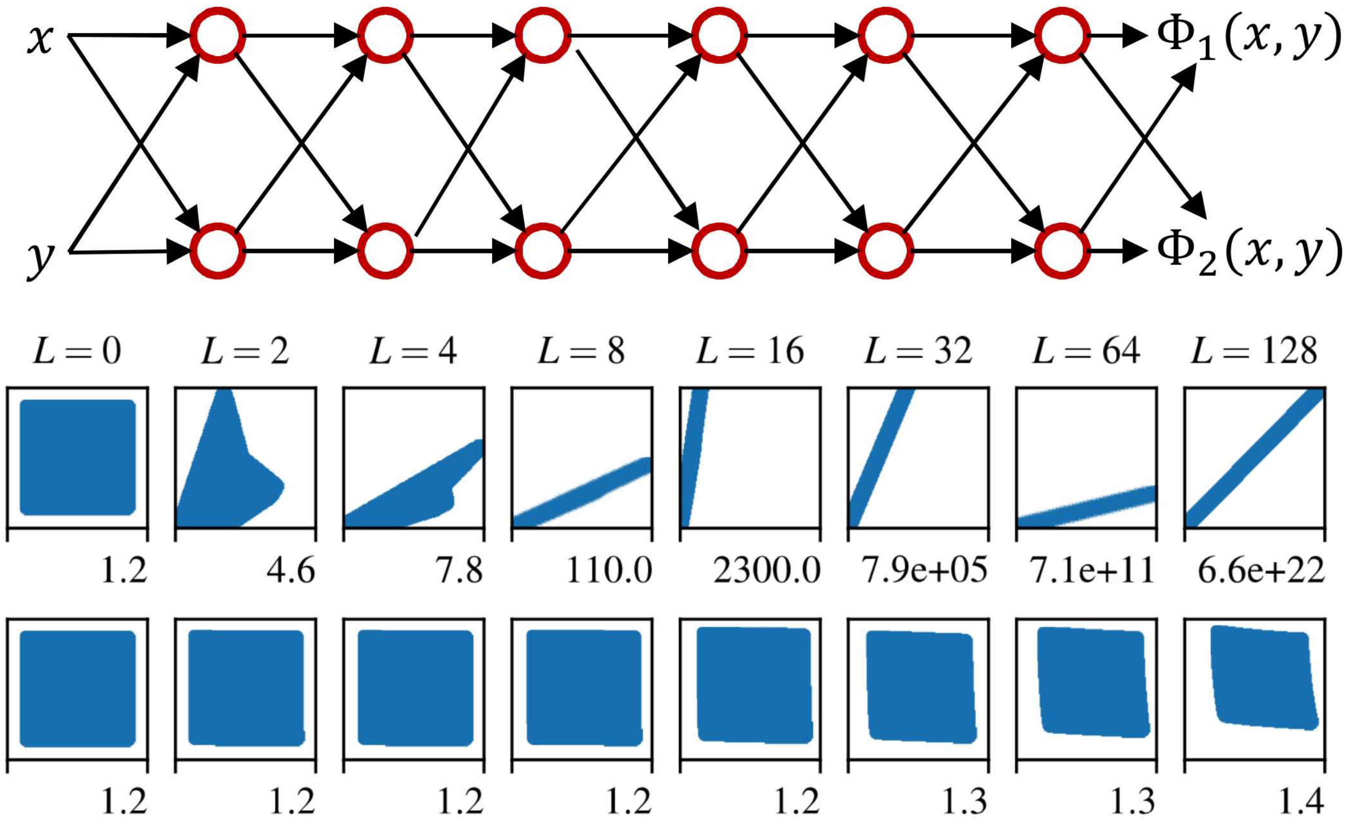
$$\boldsymbol{w}_i = k\boldsymbol{n}^T \text{ and } b_i = k\boldsymbol{p} \cdot \boldsymbol{n}$$



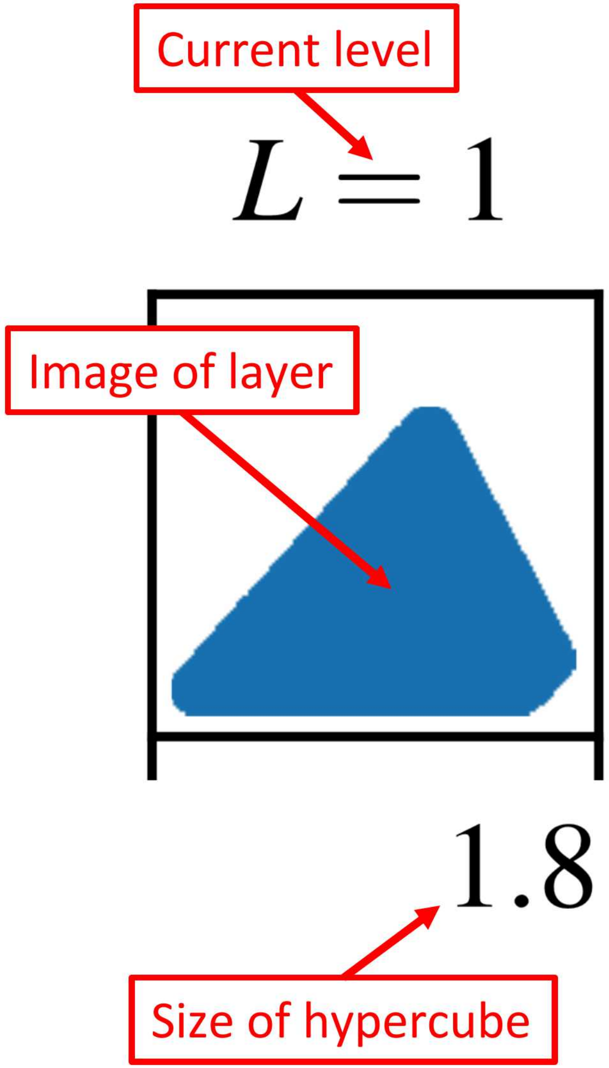
(m, m)
$\max_{\boldsymbol{x}} (\boldsymbol{x} - \boldsymbol{p}) \cdot \hat{\boldsymbol{n}}$
$\hat{\boldsymbol{n}}$
$\mathbf{p}$
$(0,0)$

Box Initialization: ResNets

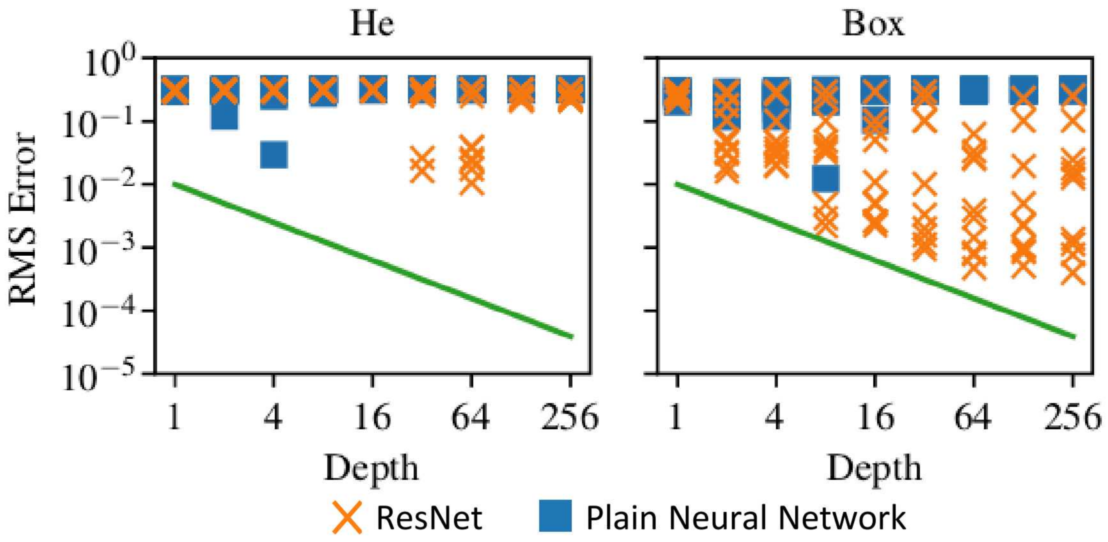
Propagate $[0,1]^2$ through the neural network

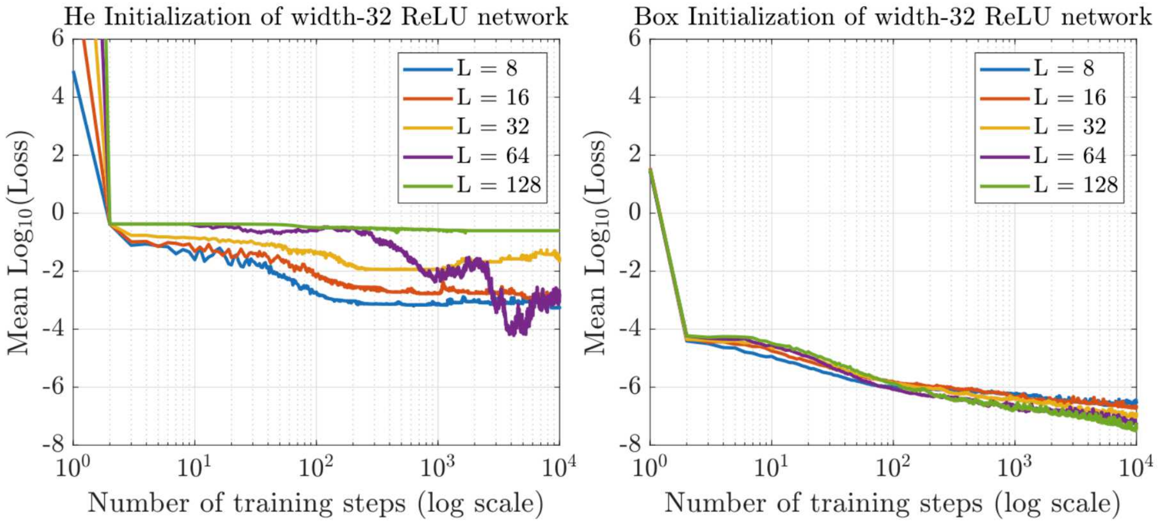


- “Box” prevents, collapse and exponential growth
- $[0,1]^2$ cube maps to nearly a cube after 128 layers



Experiments: Initialization with Box vs. He





Approximating a discontinuous function composed of two polynomials (network width is 2)

- Only Box with ResNet (orange crosses) works well
- Box does better over multiple samples, generally more robust achieving some convergence on average

Approximating $\sin(2\pi x)$

- Both He and Box work okay for small numbers of layers
- He suffers for large numbers of layers
- Box leads to smaller errors, with better performance for large numbers of layers

How to Accelerate Training With Parallelism?

Training neural networks can be costly (weeks)

- Loads of data to look at
- Lots of weights and features to optimize
- Nonlinear interactions to differentiate through
- Rough objective surface limits current applicability of optimizers; rely on gradient descent instead

Can parallel computing in general, and HPC specifically help here?

- Already multi-GPU codes are helping
- New optimization algorithms less sensitive to inaccurate gradients being developed

Our Goal: Develop a new dimension of parallelism to exploit!

Stefanie Guenther
LLNL Postdoc

Jacob Schroder
UNM

Lars Ruthotto
Emory

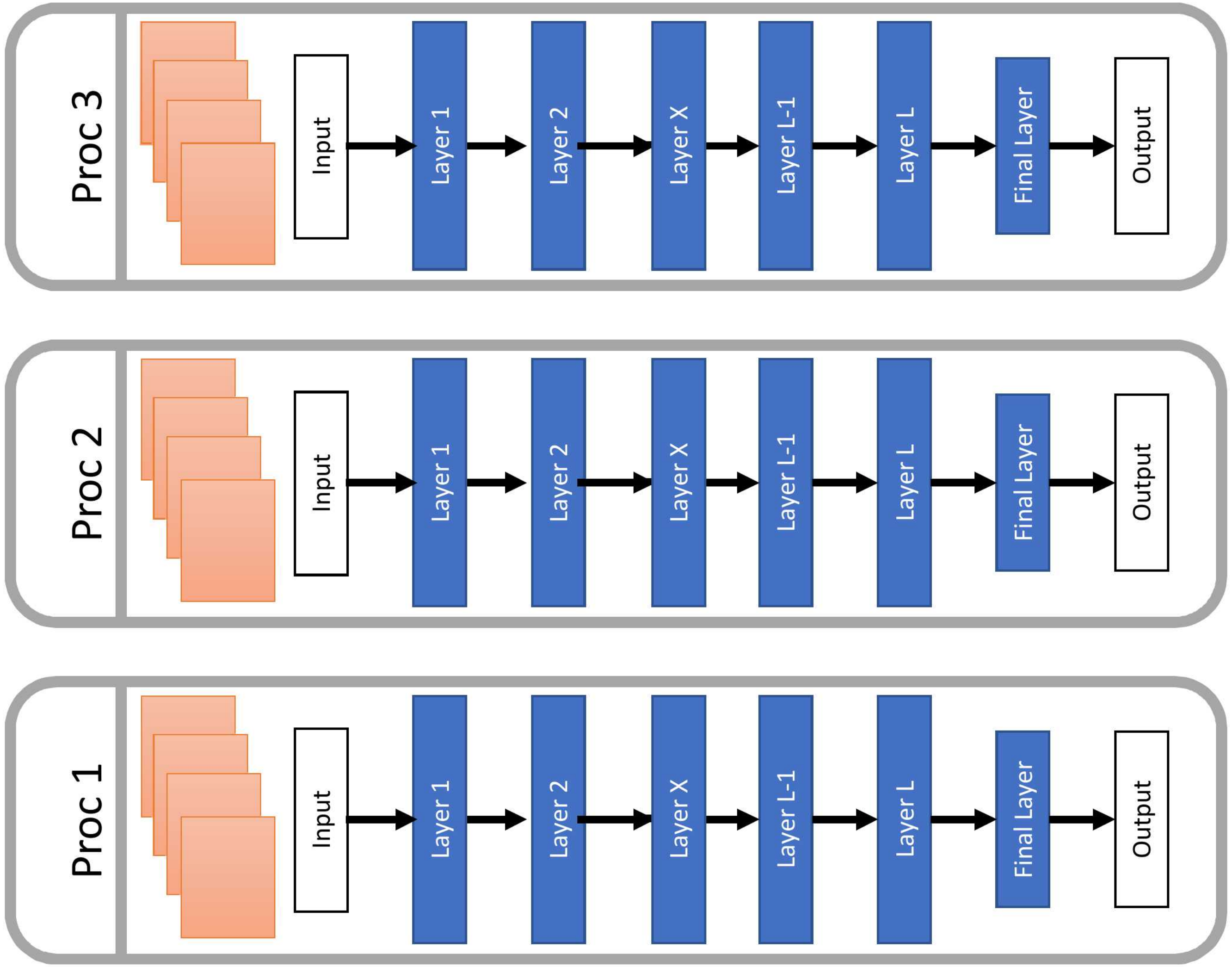
Nico Gauger
TU Kaiserslautern

Parallelization strategies: Data Parallel

Data Parallelism:

- Distribute a batch of samples over processors
- Replicate neural network across all processors

Problem: Stochastic gradient descent performance degrades with increased data size



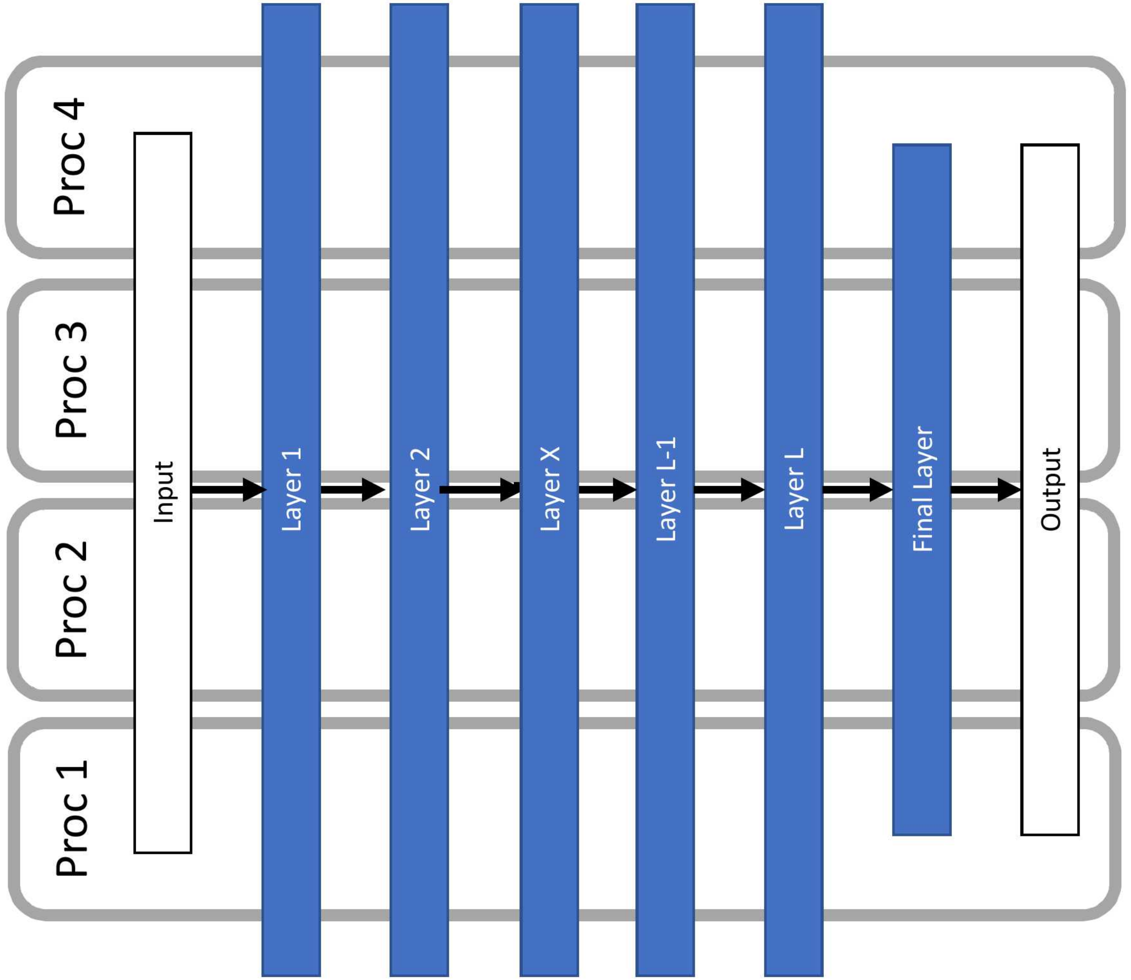
Parallelization strategies: Model Parallel

Model Parallelism:

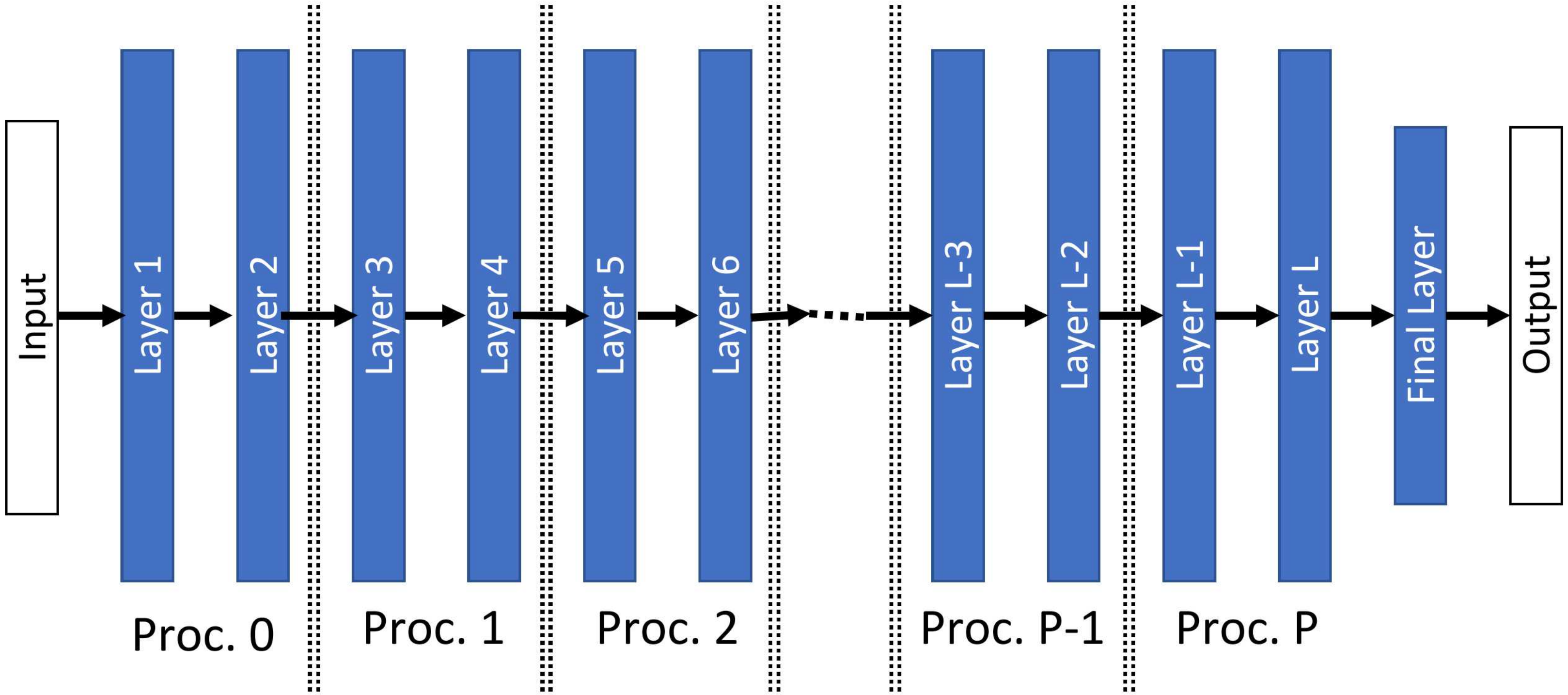
- Distribute network across processors
- Distribute data accordingly

Problem: Forward and backward propagation are serial bottlenecks. Increased depth leads unreasonable computation times

- Using a bigger computer will not solve this!



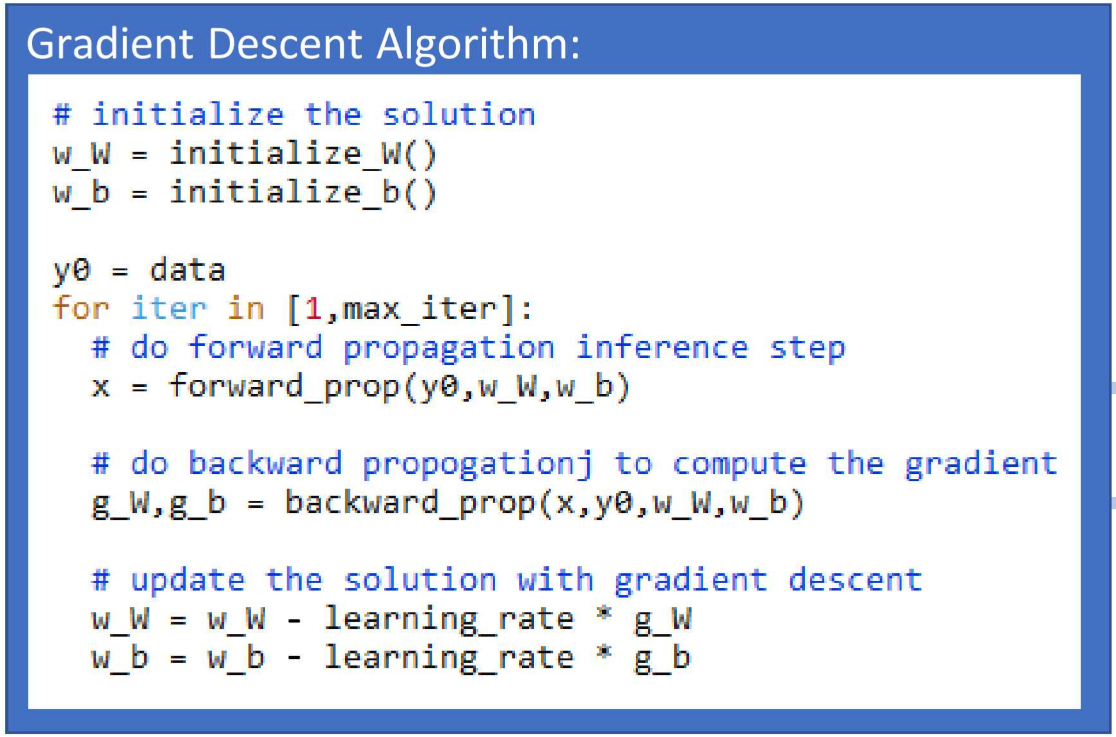
Our New Approach: Layer-Parallel Training



- Distribute layers across multiple processors (a new form of model parallelism)
- This approach is compatible with data and model parallelism
- Hint: Our approach to making this work is motivated by parallel-in-time

Wait, what? (Number one response)

Layer-Parallel makes no sense they say:



Gradient Descent Algorithm:

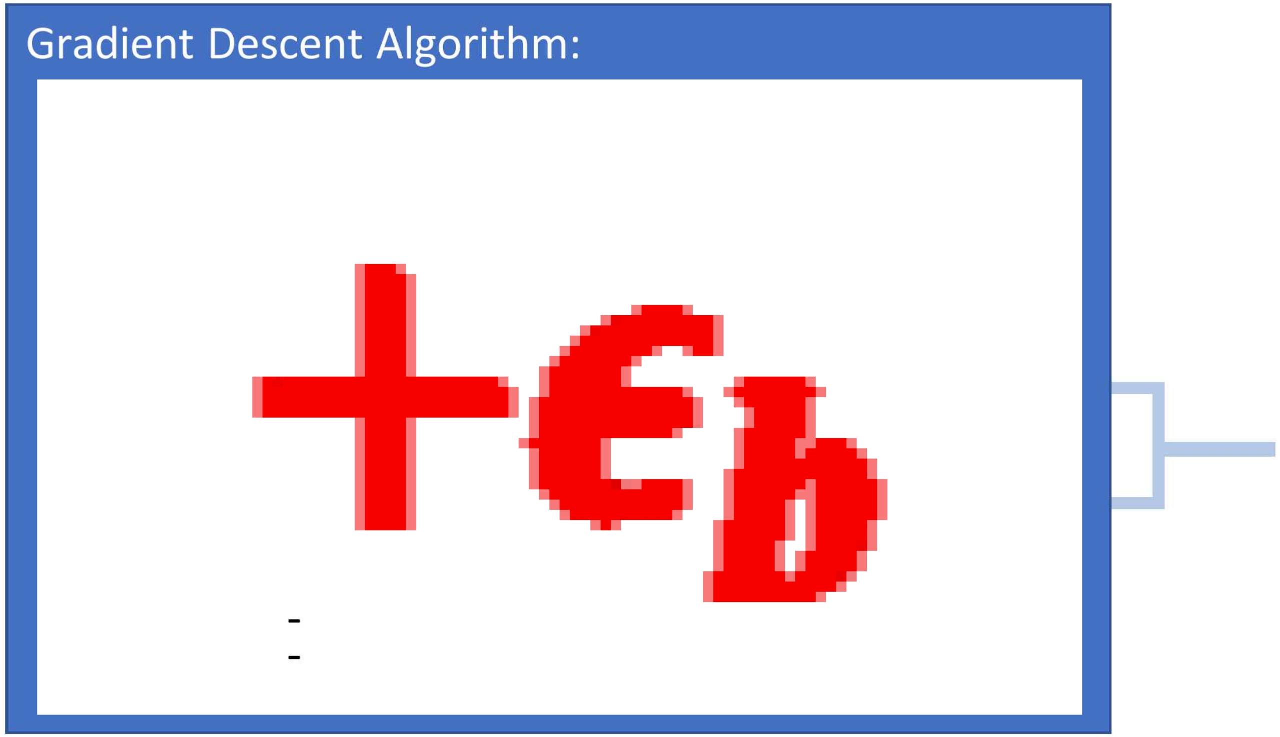
```
# initialize the solution
w_W = initialize_W()
w_b = initialize_b()

y0 = data
for iter in [1,max_iter]:
  # do forward propagation inference step
  x = forward_prop(y0,w_W,w_b)

  # do backward propogationj to compute the gradient
  g_W,g_b = backward_prop(x,y0,w_W,w_b)

  # update the solution with gradient descent
  w_W = w_W - learning_rate * g_W
  w_b = w_b - learning_rate * g_b
```

These serialize across the layers. A forward and then a backward sweep! How can you parallelize

- Forward and backward propagation are serial!
- Distributing the layers across processors still serializes!
- It doesn’t make a whole lot of sense does it?

Critical Assumption: Exactness of propagation

We can relax the exactness of propagation, and trade for parallelism!



Gradient Descent Algorithm:

$+\epsilon_b$

-

-

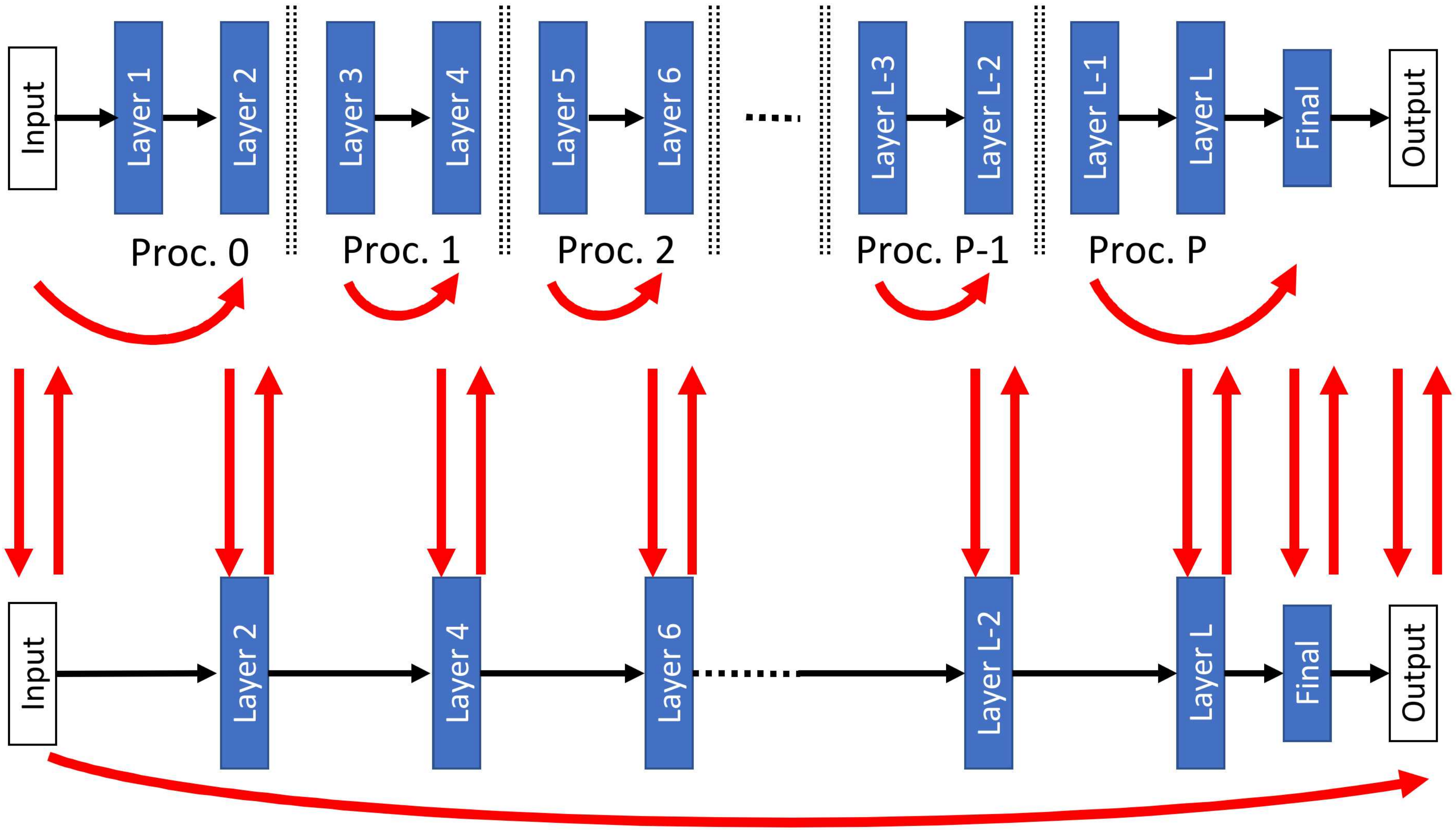
Introduce a small error

- If we can control the error we introduce, we can use it to get parallelism!
- We introduce this error through a multigrid algorithm, and get parallelism as a result

Layer-Parallel Algorithm

Takes advantage of recent advances in Multi-grid In Time (MGRIT*)

1. Relax on fine
2. Transfer to coarse
3. Coarse correction
4. Transfer to fine
5. Relax on fine



*R. D. Falgout, S. Friedhoff, Tz. V. Kolev, S. P. MacLachlan, and J. B. Schroder, Parallel Time Integration with Multigrid, SIAM J. Sci. Comput., 36 (2014), pp.C635-C661.

Layer-Parallel Algorithm: Details

① Uses ODE Networks (time=layers)

- Think ResNet as an ODE
- Theory from multigrid-in-time
- Questions about regularity required

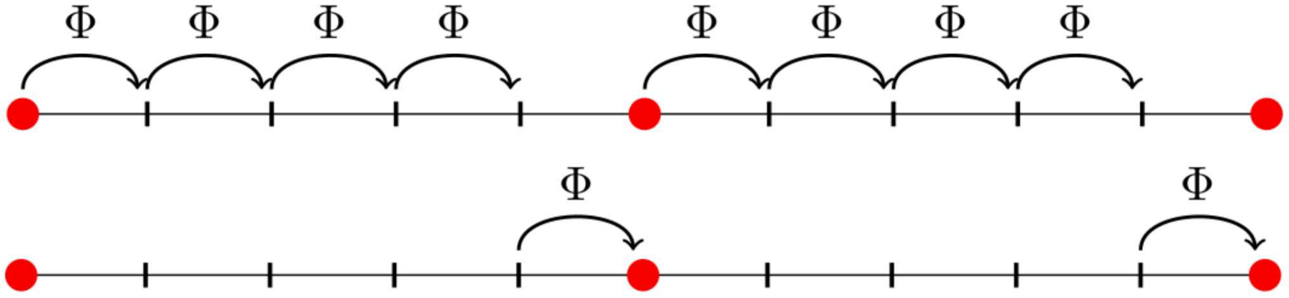
$$x_{k+1} = x_k + {\color{red}\Delta t}\,\sigma(W_k x_k + b_k)$$

$\Big\Uparrow$ Discretize

$$\partial_t x(t) = \sigma(W(t)x(t) + b(t))$$

② Fine-Coarse-Fine (FCF) relaxation with FAS multigrid:

1. Relax fine points
2. Relax on coarse points
3. Relax on fine points again



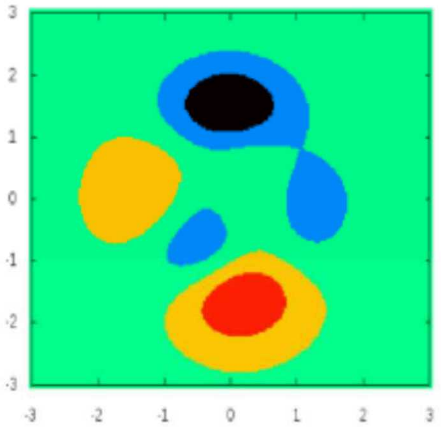
③ Using one-shot optimization

- No batching like SGD
- Probably suboptimal
- Using L-BFGS Hessian

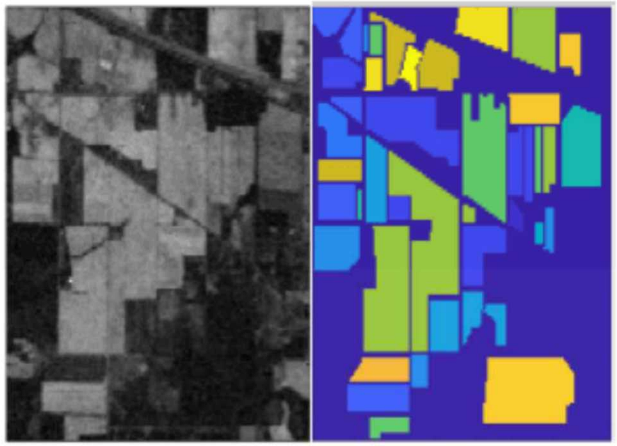
$$\checkmark^n \qquad \checkmark^n - \square{\color{red}H_k^{-1}} \overset{\checkmark}{} \frac{\mathrm{dLoss}_k}{\mathrm{d}\checkmark^n} \overset{\blacklozenge}{}$$

$$\text{with} \quad H \,\square\, @^2_{\checkmark}\left(L + \square k y - H(y, \checkmark)k^2\right)$$

Layer Parallel Scaling Results



(a) Peaks



(b) Indian Pines

(c) MNIST

Three different classification problems

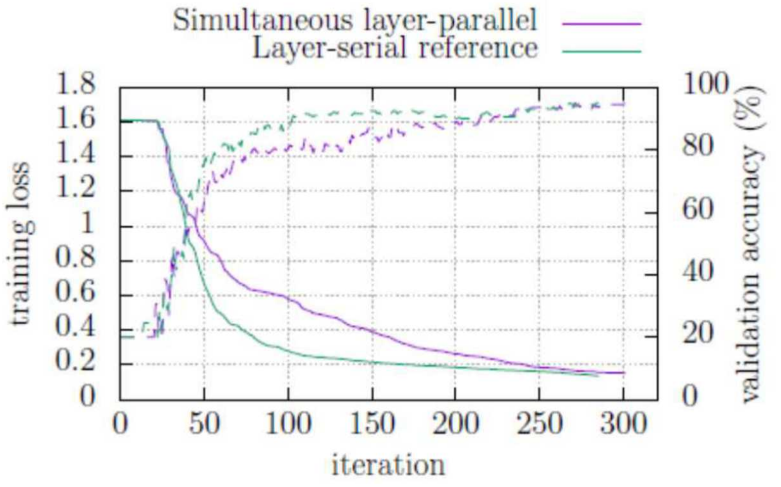
1. Peaks: Put particle position into one of 5 different classes
2. Indian Pines: Hyperspectral imaging, what crop? Soy, corn, etc…
3. MNIST: Handwritten digit classification

A comment on the code:

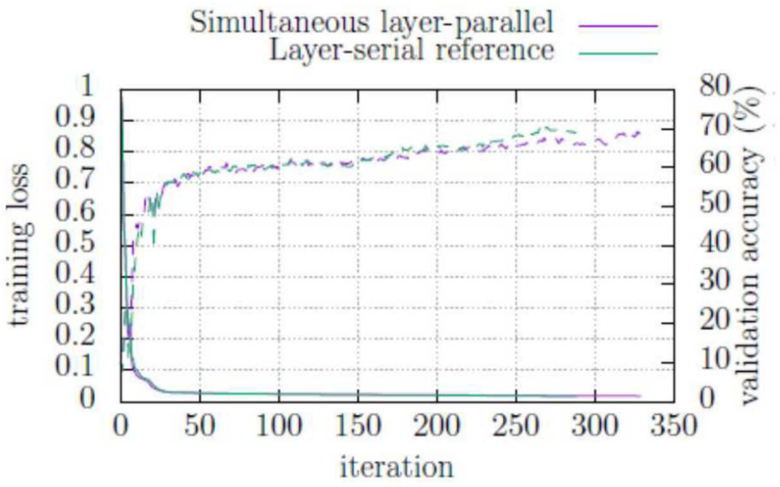
- Neural network code using Xbraid (LLNL) parallel-in-time library
- Code is not optimized: e.g. MNIST uses hand coded convolutions
- Neural networks architectures not optimized, simple ODENets

Layer Parallel Scaling Results

Peaks



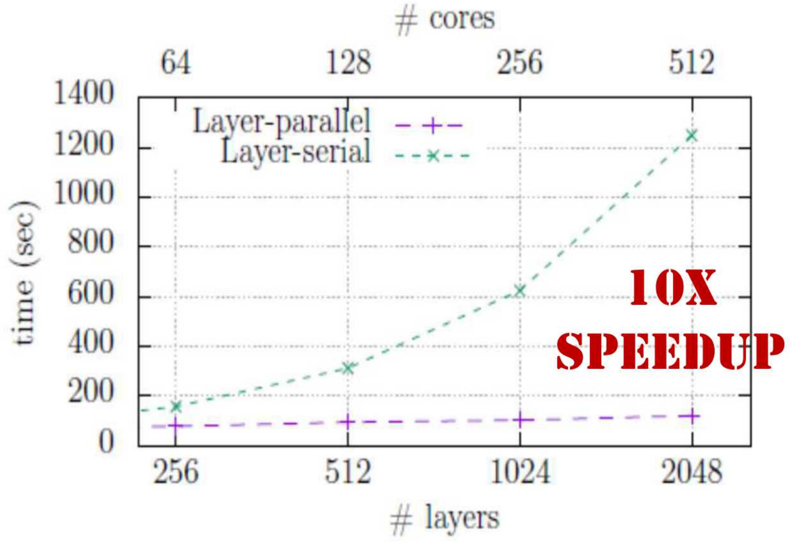
Indian Pines



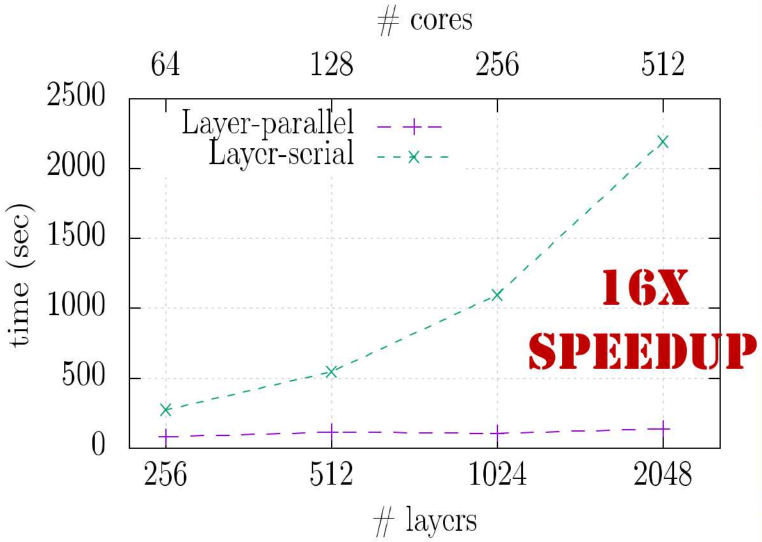
Indian Pines

MNIST

Weak Scaling

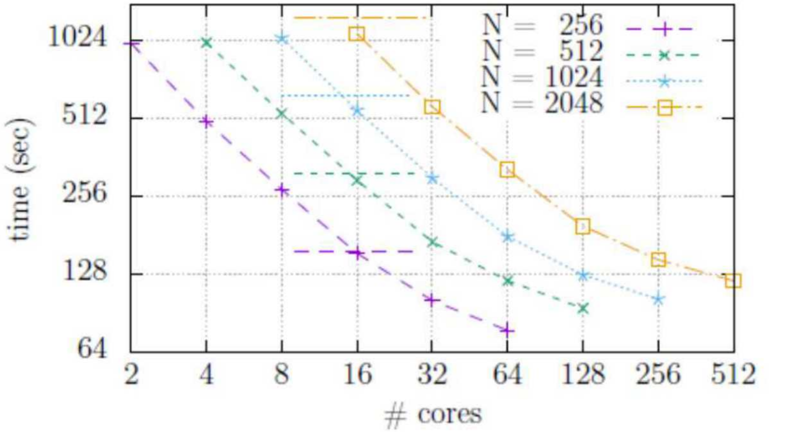


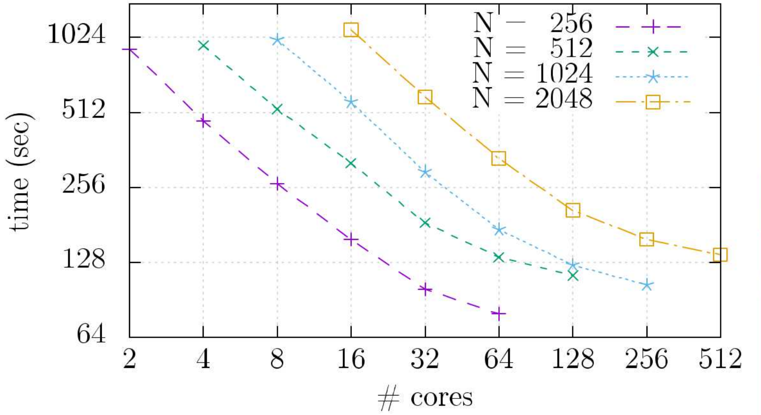
10X SPEEDUP



16X SPEEDUP

Strong Scaling





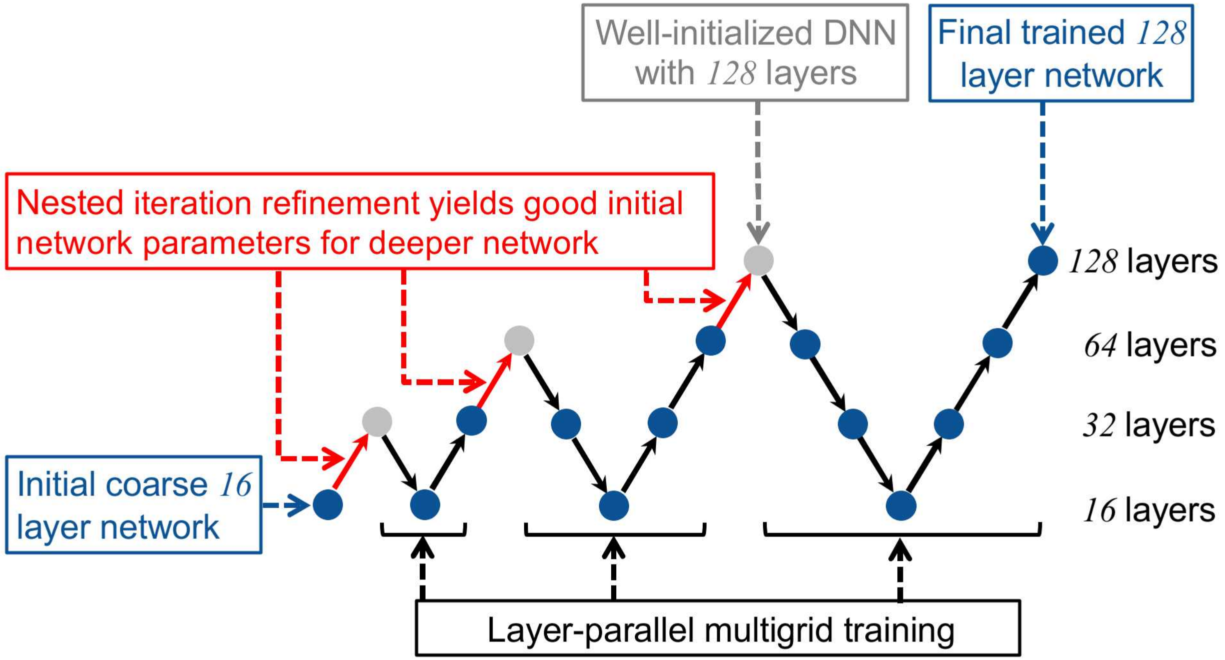
Layer-Parallel Initialization: Nested Iteration

Initialization of Layer-Parallel is complex

- Initialize weights and biases
- Initialize state and adjoint

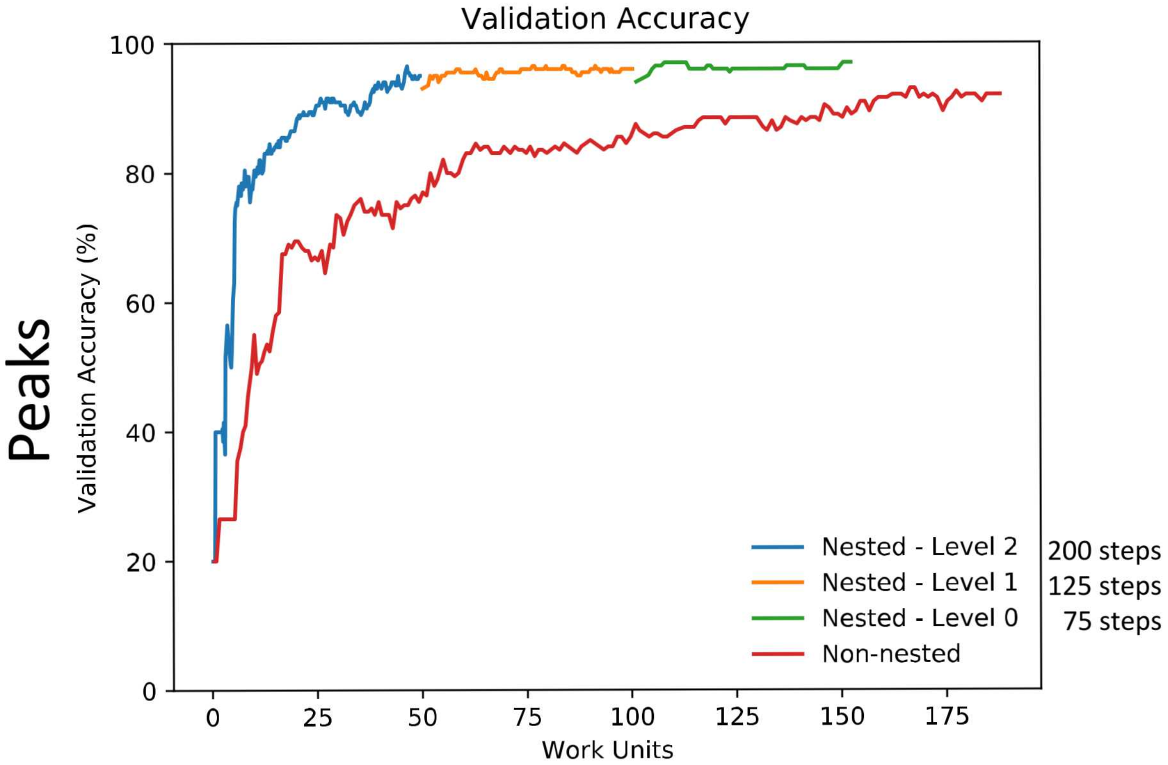
To overcome this, we have developed a nested iteration

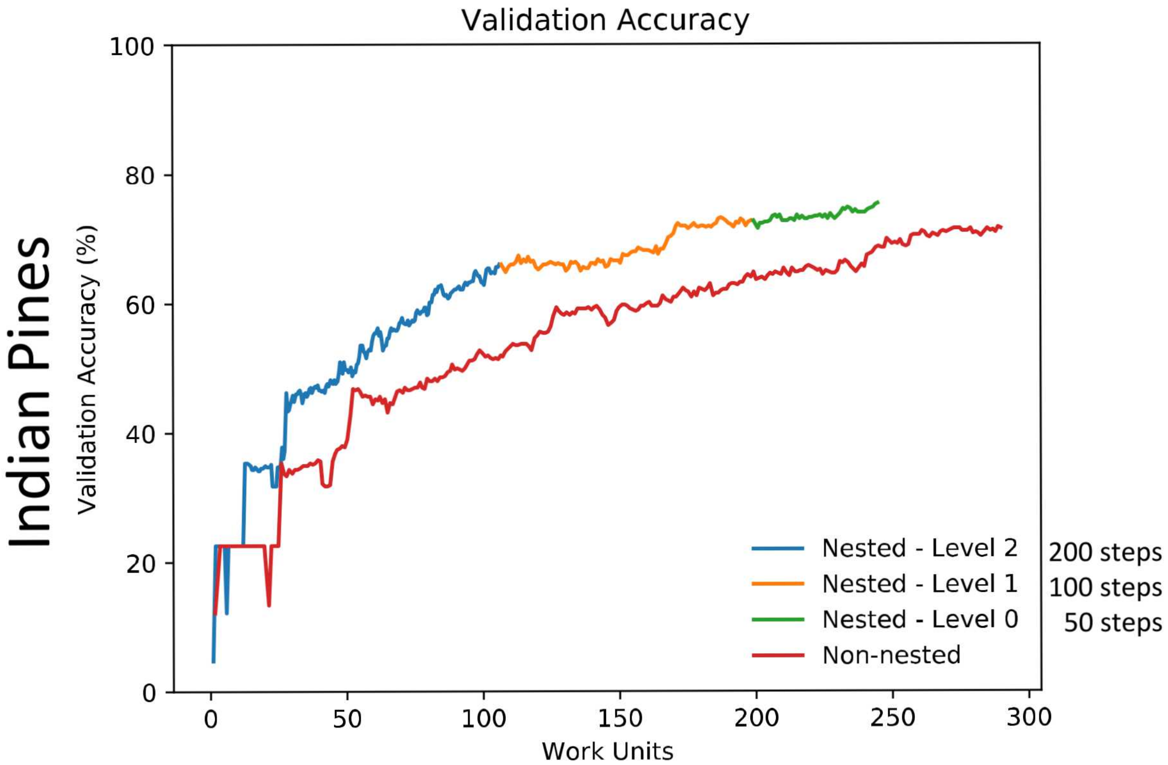
- Like full multigrid
- Train on the coarse network first, then upscale


Well-initialized DNN with 128 layers
Final trained 128 layer network
Nested iteration refinement yields good initial network parameters for deeper network
128 layers
64 layers
32 layers
16 layers
Initial coarse 16 layer network
Layer-parallel multigrid training

Nested Iteration: Indian Pines and Peaks

- 3 level example with Indian Pines and Peaks data sets
- Work Unit = Average Fine Level forward/adjoint gradient computation





Nested iteration yields better validation accuracy in less time

Nested Iteration: Regularization

To understand the regularization impact of nested iteration

- 4 different values for hyper parameters, chosen to give good results

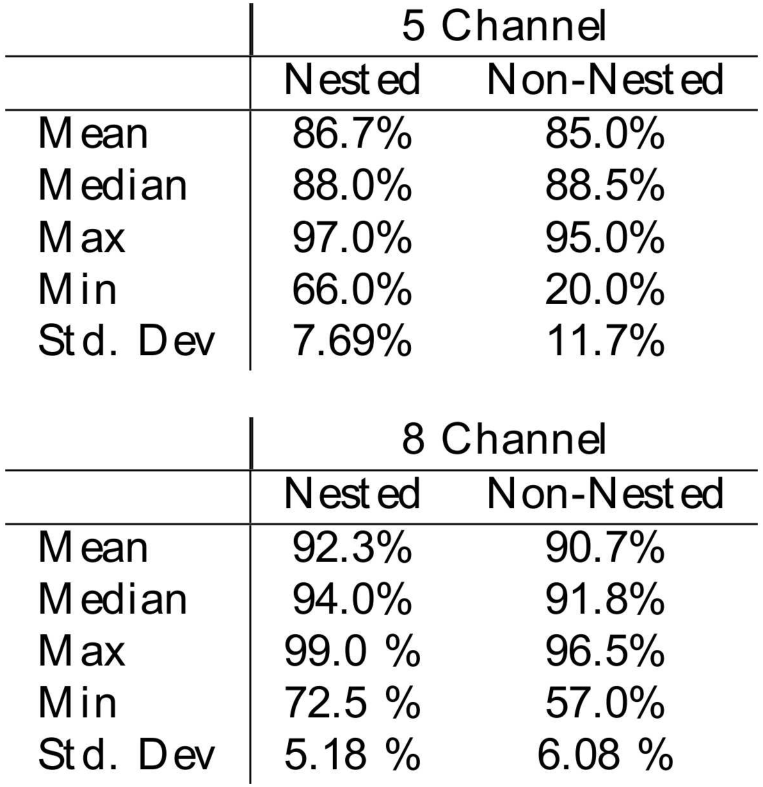
Tikanov Regularization	10^{-5}	10^{-7}
Initial Weights	0.0	10^{-6}

- 12 independent runs for each hyper parameterization (48 total runs)

Nested Iteration validation accuracy less sensitive than non-nested iteration

- Promising improvement to robustness (not definitive)
- **Hypothesis**: nested iteration applies implicit regularization

Peaks Validation Accuracy



	5 Channel	
	Nested	Non-Nested
Mean	86.7%	85.0%
Median	88.0%	88.5%
Max	97.0%	95.0%
Min	66.0%	20.0%
Std. Dev	7.69%	11.7%

	8 Channel	
	Nested	Non-Nested
Mean	92.3%	90.7%
Median	94.0%	91.8%
Max	99.0 %	96.5%
Min	72.5 %	57.0%
Std. Dev	5.18 %	6.08 %

Better with Layer-Parallel?

PDE constrained problem:

$$\begin{aligned} &\min_{z\in Z} \quad g(u,z) \\ &\text{s.t.} \;\; F(u,z)=0 \end{aligned}$$

- “g” is scalar objective function
- “F” is PDE problem in residual form
- “u” is a **state** variable (solution to PDE)
- “z” is a **control** variable

Denis Ridzal (SNL)

Some work in parallel-in-time optimization and applications

- Guenther, Gauger, Schroder, Opt. Methods and SW, 2018
- Götschel, Minion, preprint arXiv:1901.06850, 2019
- Ulbrich, Real-time PDE-constrained optimization, 2007
- Maday and Turinici, Proceedings of the 41st IEEE Conference on Decision and Control, 2002

PDE Constrained Optimization: The KKT System

The critical points of the Lagrangian

$$\mathcal{L}(u, z, \lambda) = g(u, z) + \lambda^T F(u, z)$$

are the 1st order necessary conditions:

$$\partial_u \mathcal{L} = \partial_u g(u, z) + \lambda^T \partial_u F(u, z) = 0$$

$$\partial_z \mathcal{L} = \partial_z g(u, z) + \lambda^T \partial_z F(u, z) = 0$$

$$\partial_z \mathcal{L} = F(u, z) = 0$$

Linearizing these conditions, gives us a matrix with the celebrated KKT structure

This structure, with minor variations, often appears in full space optimization algorithms

$$\begin{bmatrix} H_{11} & H_{12} & J_1^T \\ H_{21} & H_{22} & J_2^T \\ J_1 & J_2 & \end{bmatrix}$$

For the Inexact SQP Algorithm we are pursuing:

- $H_{11} = H_{22} = I$
- $H_{12} = H_{21} = 0$
- $J_1 = \partial_u F(u, z)$
- $J_2 = \partial_z F(u, z)$

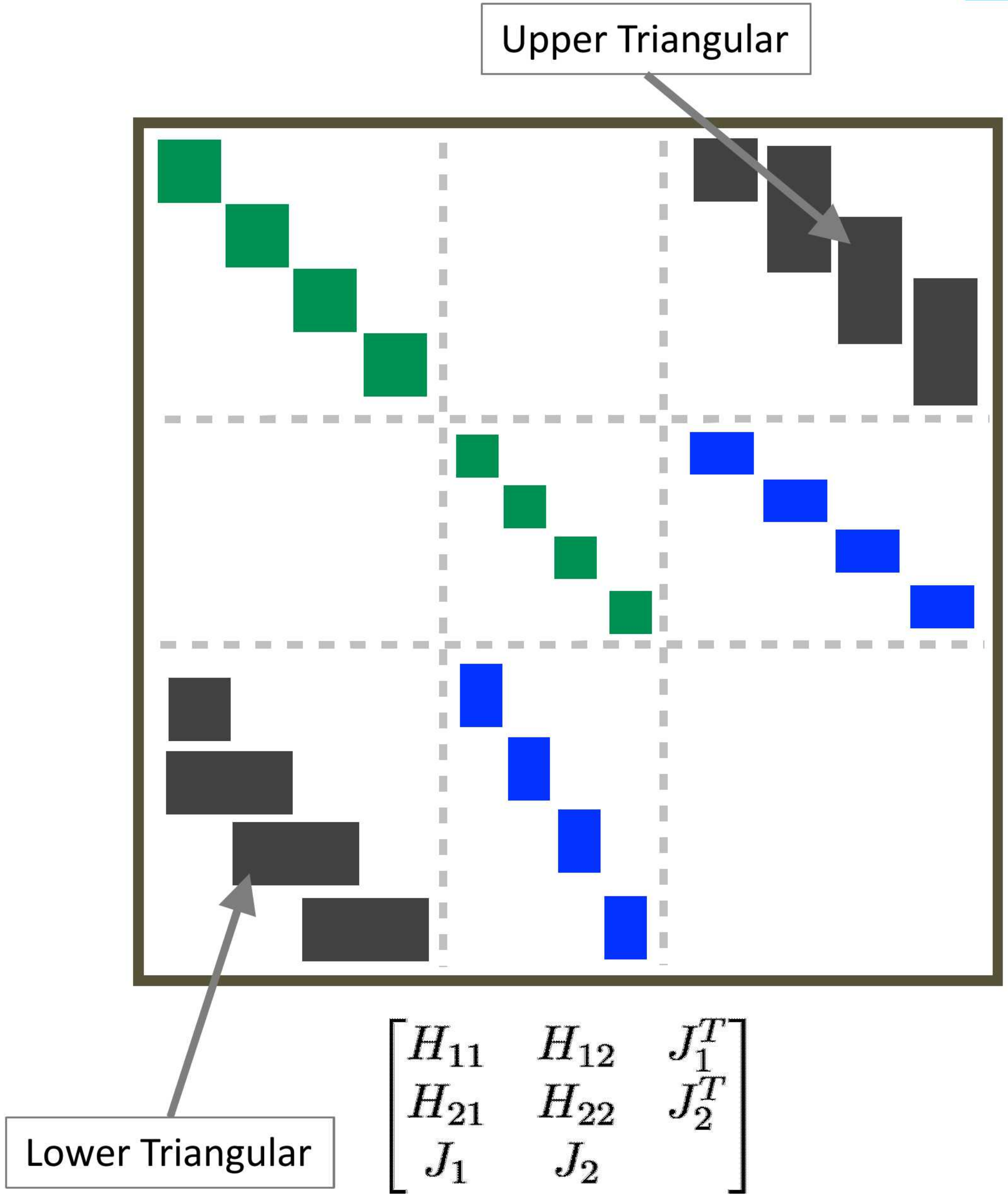
PDE Constrained Optimization: Transient KKT Systems

Assume a transient constraint

$$F(u, z) = \partial_t u + K(u, z)$$

Now J_1 is lower triangular, its transpose is upper triangular

- Implies a forward-time and backward-time solve
- The "adjoint problem" is backwards in time



Upper Triangular

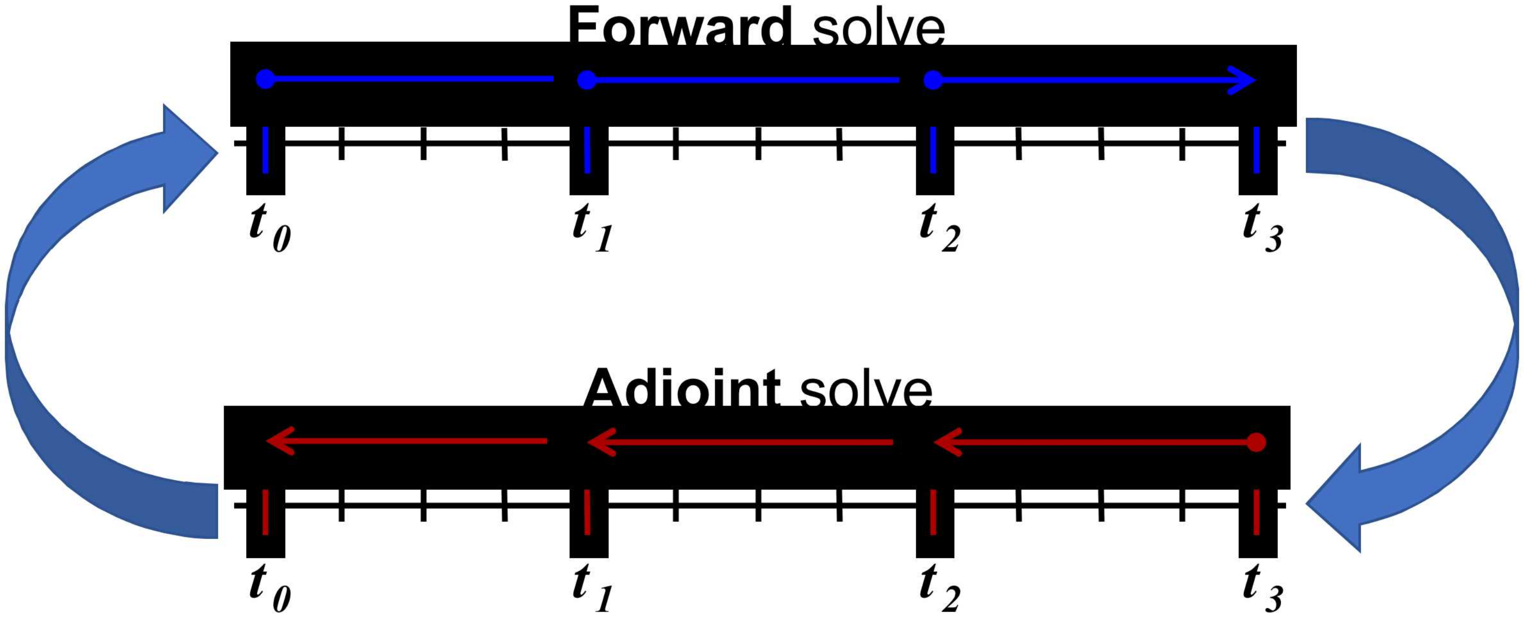
Lower Triangular

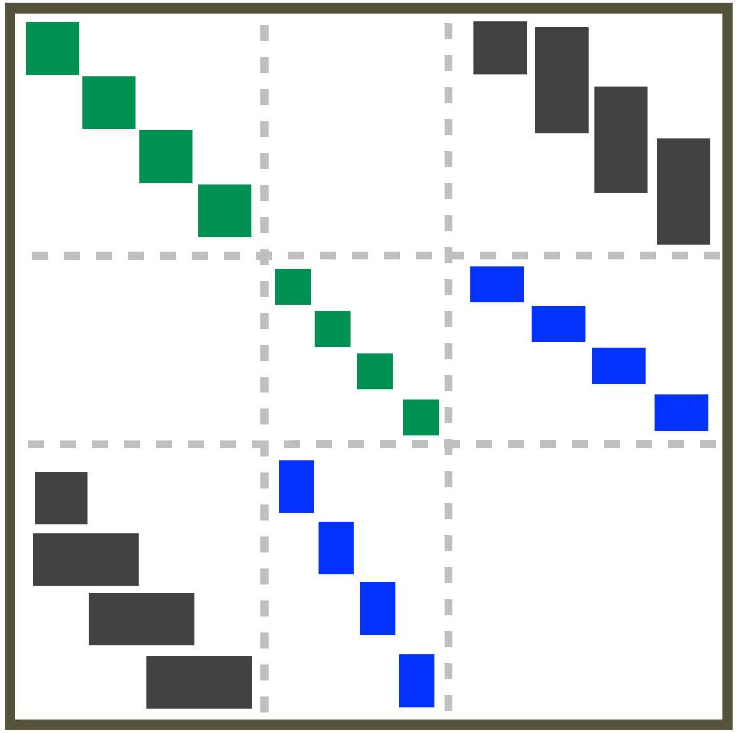
$$\begin{bmatrix} H_{11} & H_{12} & J_1^T \\ H_{21} & H_{22} & J_2^T \\ J_1 & J_2 & \end{bmatrix}$$

Forward and Adjoint Solve

Our solution to the serial challenge is to develop a method based on multigrid in time

- KKT system couples in time!
- The optimal solution **does not** couple in time
- **The path** to the optimal solution couples in time





Nonlinear optimization algorithms do repeated sequences of forward then adjoint solves

Examine the Optimization Problem

Solve the quadratic problem:

$$\min_{z} \ \frac{1}{2}\|u-\hat{u}\|^2 + \frac{1}{2}\|z\|^2$$
$$\text{s.t.} \ \frac{d}{dt}u = Ku + Gz$$

Optimality conditions for quadratic problem:

$$-\frac{d}{dt}w - K^T w + u - \hat{u} = 0$$
$$z - G^T w = 0$$
$$\frac{d}{dt}u - Ku - Gz = 0$$

Eliminating ‘u’ and ‘z’ yields the **elliptic in time** equation:

$$-\frac{d^2}{dt^2}w + (K - K^T)\frac{d}{dt}w + (KK^T + GG^T)w = \frac{d}{dt}\hat{u} - K\hat{u}$$

For related observations

- Lewis, Nash. SIAM Journal on Scientific Computing, *26*(6), 2005.
- Gander, Kwok. *Domain Decomposition Methods in Science and Engineering XXII*. 2016

Introduction of Coupling Constraints

We introduce coupling constraints between time steps

- Motivated directly by:
  - Heinkenschloss, J. Comp. Appl. Math., 2005.
  - Comas Ph.D. Thesis, Rice University, 2006.
- Similar to multiple shooting
- Thus the time coupling will be resolved by the nonlinear solver

For instance, a PDE constrained Burger's example:

$$\min_{u,z} \; \frac{1}{2}\int_0^T \int_0^1 (u(x,t) - \bar{u}(x,t))^2 + \alpha z(x,t)^2 \, dx dt$$

$$\begin{aligned} \text{subject to} \;\; \partial_t u(x,t) - \nu \partial_{xx} u(x,t) + \partial_x (u(x,t)^2) &= z(x,t) \\ u(0,t) = u(1,t) = 0, \; u(x,0) &= u_0(x) \end{aligned}$$

Next slide shows how we discretize and introduce coupling constraints

Introduction of Coupling Constraints: A Model Problem

Discretize with the theta method:

$$\min_{u,z} \sum_{i=1}^{N+1} \frac{\Delta t_{i-1} + \Delta t_i}{2} \left(\frac{1}{2} u_i^T M u_i + g(t_i)^T u_i \right) + \sum_{i=0}^{N+1} \frac{\Delta t_{i-1} + \Delta t_i}{2} \left(\frac{\alpha}{2} z_i^T Q z_i \right)$$

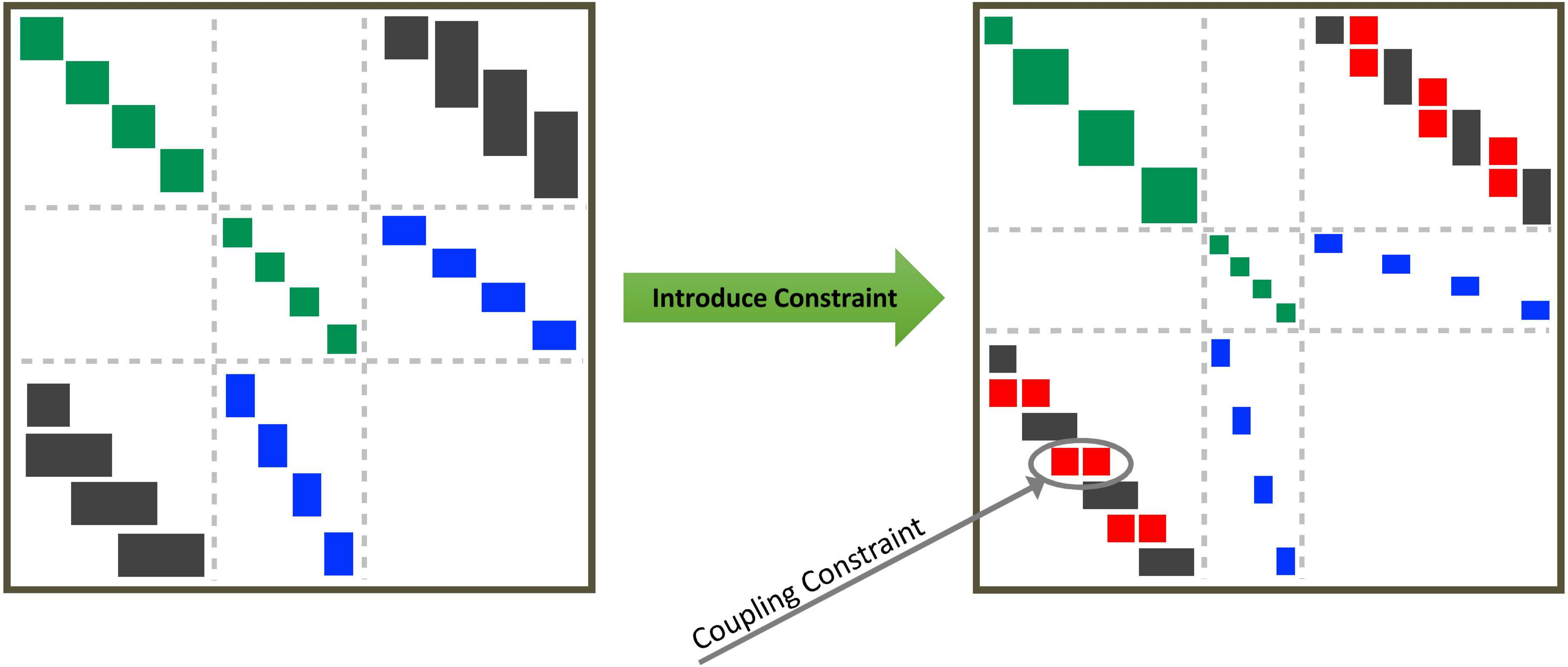
$$\begin{aligned} \text{subject to } \; (M + \theta \Delta t_i A) v_{i+1} + \theta \Delta t_i N(v_{i+1}) + (M + (1-\theta)\Delta t_i A) u_i & \\ + (1-\theta)\Delta t_i N(u_i) + \Delta_i B z_i &= 0 \\ u_i - v_i &= 0 \end{aligned}$$

Expose time continuity coupling constraint by introducing ”virtual” variables **v_i** into the optimization problem

Explicit exposure of these temporal constraints makes the development of a time domain decomposition approach straightforward.

Explicitly expose coupling in time

Introducing the coupling constraints changes the structure of the matrix



Introduce Constraint

Coupling Constraint

Ingredients to Multigrid

We have explained the structure of the operator:

- Introduced coupling constraints
- Depends on number of time steps

We want to develop solver that:

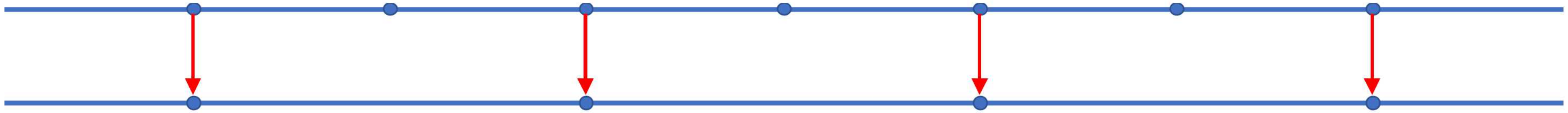
- Allows decomposition over time steps (and space)
- Can use a matrix free approach

We will develop a multigrid-in-time scheme to solve the linear problem

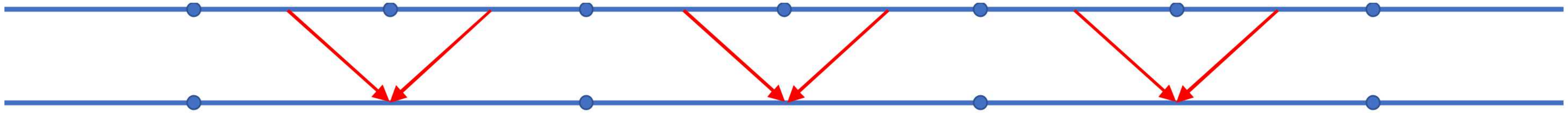
1. We need coarsening and restriction schemes in time
2. We need a scalable smoother

Restriction and Prolongation

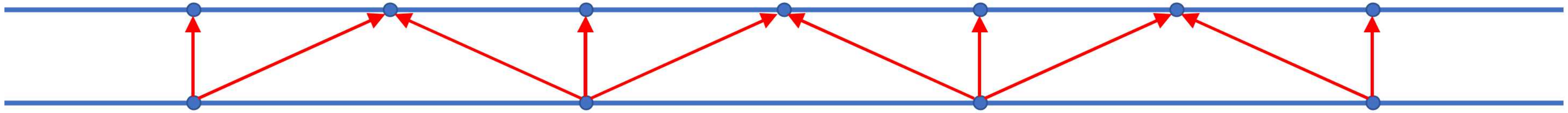
- For states and adjoints, we define restriction as point injection (copy).



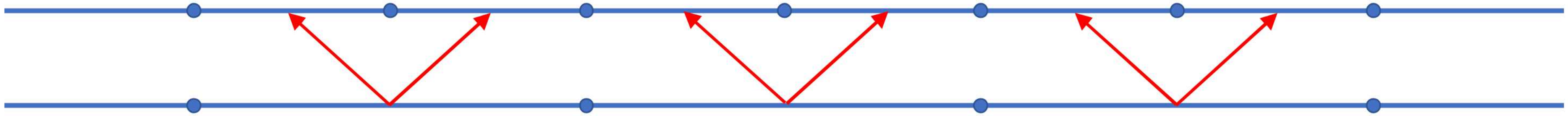
- For controls, we define restriction as a weighted 2-interval average.



- For states and adjoints, we define prolongation via linear interpolation.

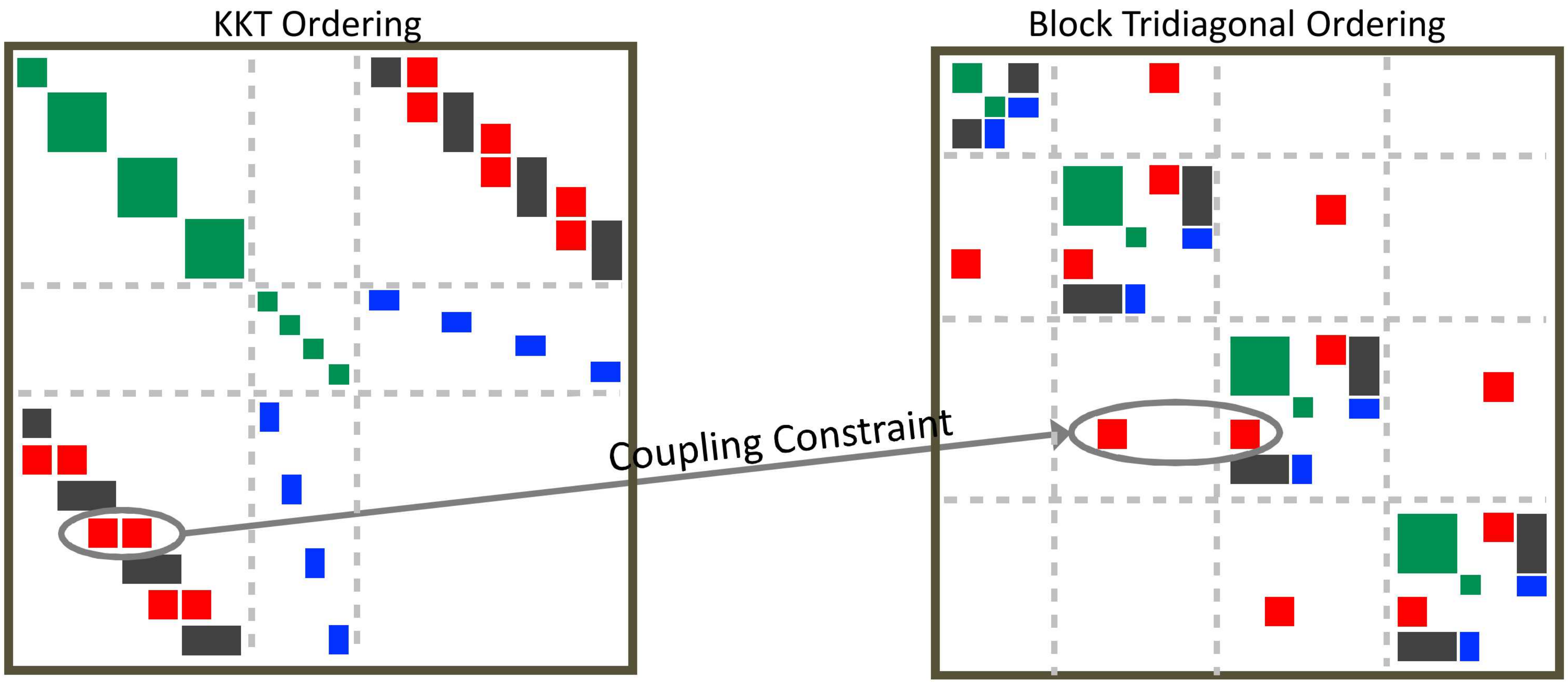


- For controls, we define prolongation as interval injection (copy).



Scalable Smoother

Reordering of unknowns creates a interesting structure:

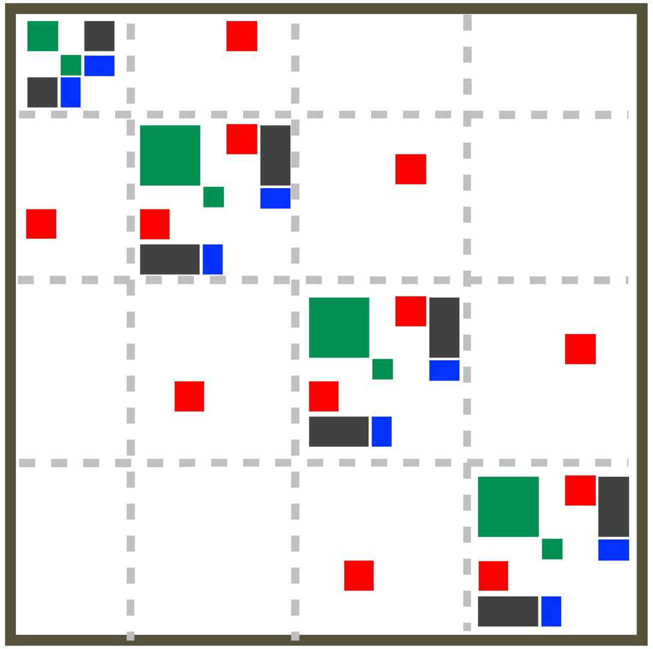


New structure has KKT systems for each time step on the block diagonal, with temporal continuity constraints on the off diagonals

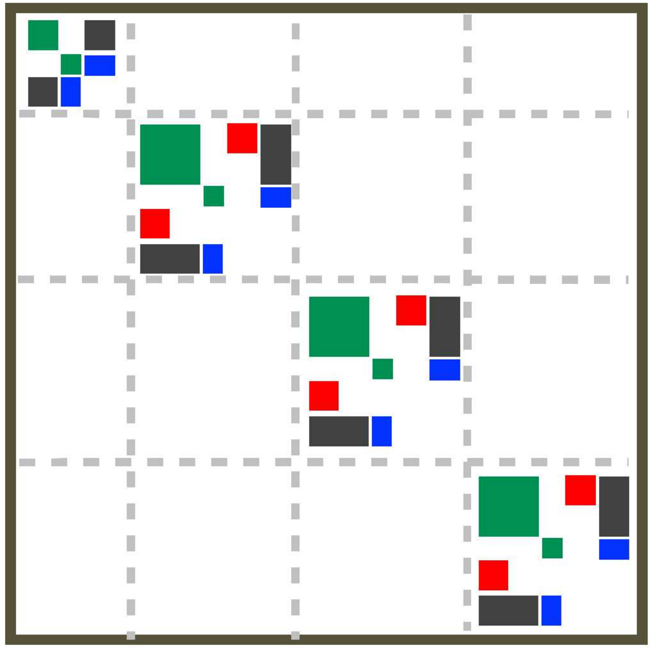
Scalable Smoother: Block Jacobi

Relax coupling between blocks by removing continuity condition:

$A =$



$M =$



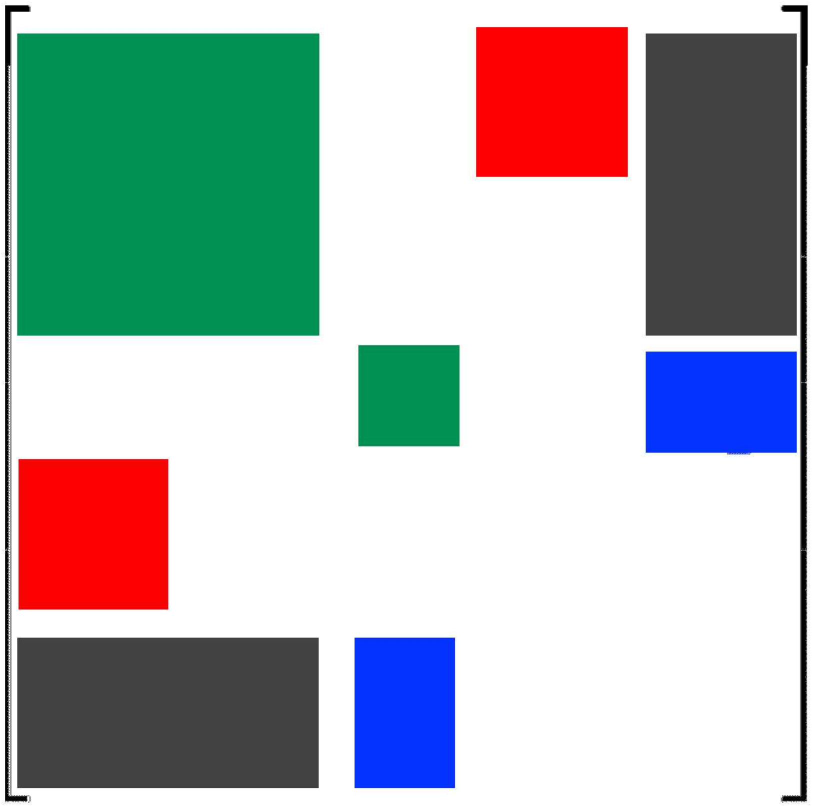
Relaxation scheme to solve Ax=b:

$$x_{i+1} = x_i + M^{-1}(b - Ax_i)$$

- Block Jacobi ”smoothing” over each time step
- Blocks are approximately inverted in parallel

Scalable Smoother: Solving the local KKT system

Each subdomain must solve a local KKT system:



- Following the work of Wathen and others*, we will use a block LU factorization
- Upper blocks are trivially invertible
- Schur complement of KKT must be approximated

$$\begin{bmatrix} I & J_{1o}^T \\ J_{1o} & P \end{bmatrix} \approx \begin{bmatrix} I & \\ J_{1o} & I \end{bmatrix} \begin{bmatrix} I & J_{1o}^T \\ & \hat{S} \end{bmatrix}$$

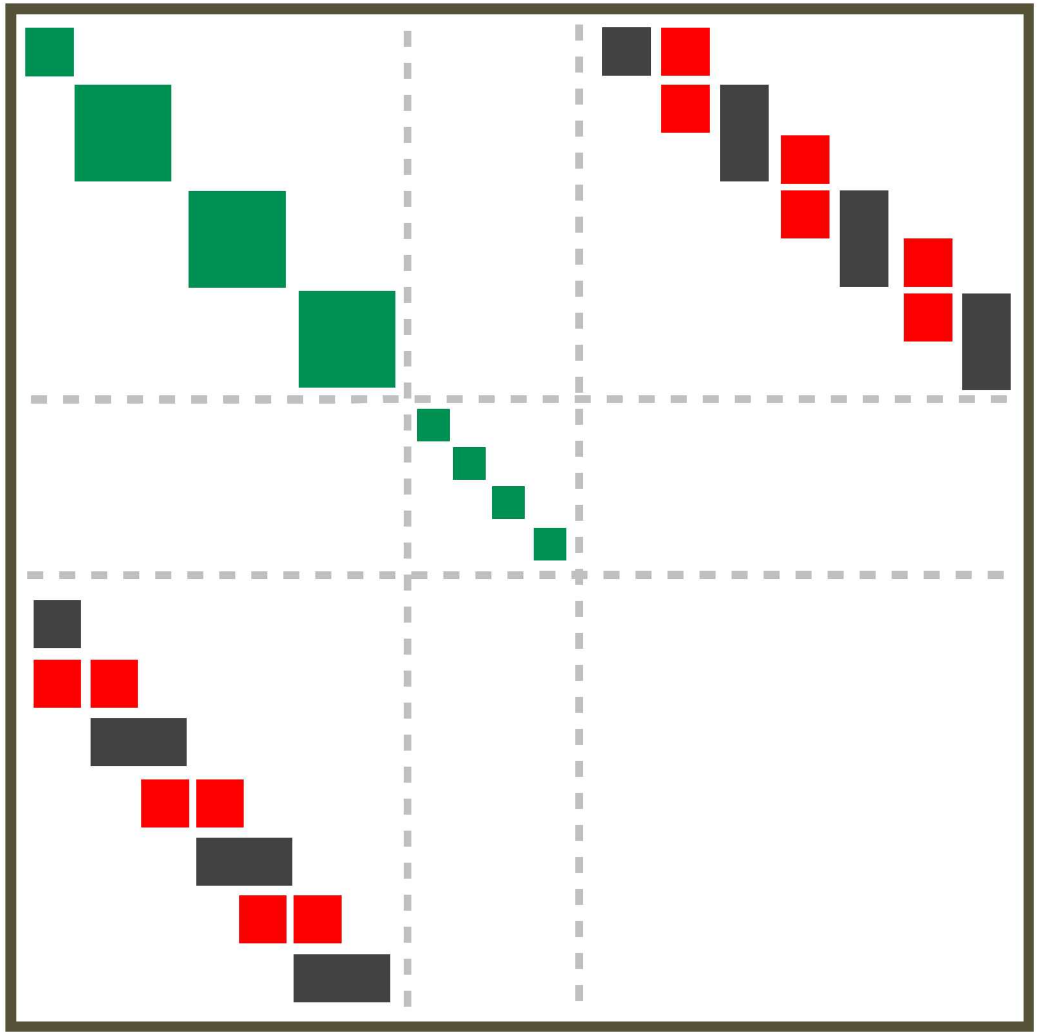
where

$$P = -J_{1o}J_{1o}^T - J_{1n}J_{1n}^T - J_2J_2^T$$
$$\hat{S} = -J_{1n}J_{1n}^T$$

- Applied as a smoother with residual correction

*T Rees, HS Dollar, and A Wathen. "Optimal solvers for PDE-constrained optimization." *SISC* 32, 2010.
M Stoll, and A Wathen. "All-at-once solution of time-dependent Stokes control." *Journal of Computational Physics* 232, 2013

Coarse Grid Correction



- On coarse grid we revert to the KKT form
- Assume control contributions are zero
- Solve this system again using Wathen style preconditioner
- Again use a residual correction, now on coarse grid
- This couples across time steps, and effectively serializes

Results: 1D Burgers Control

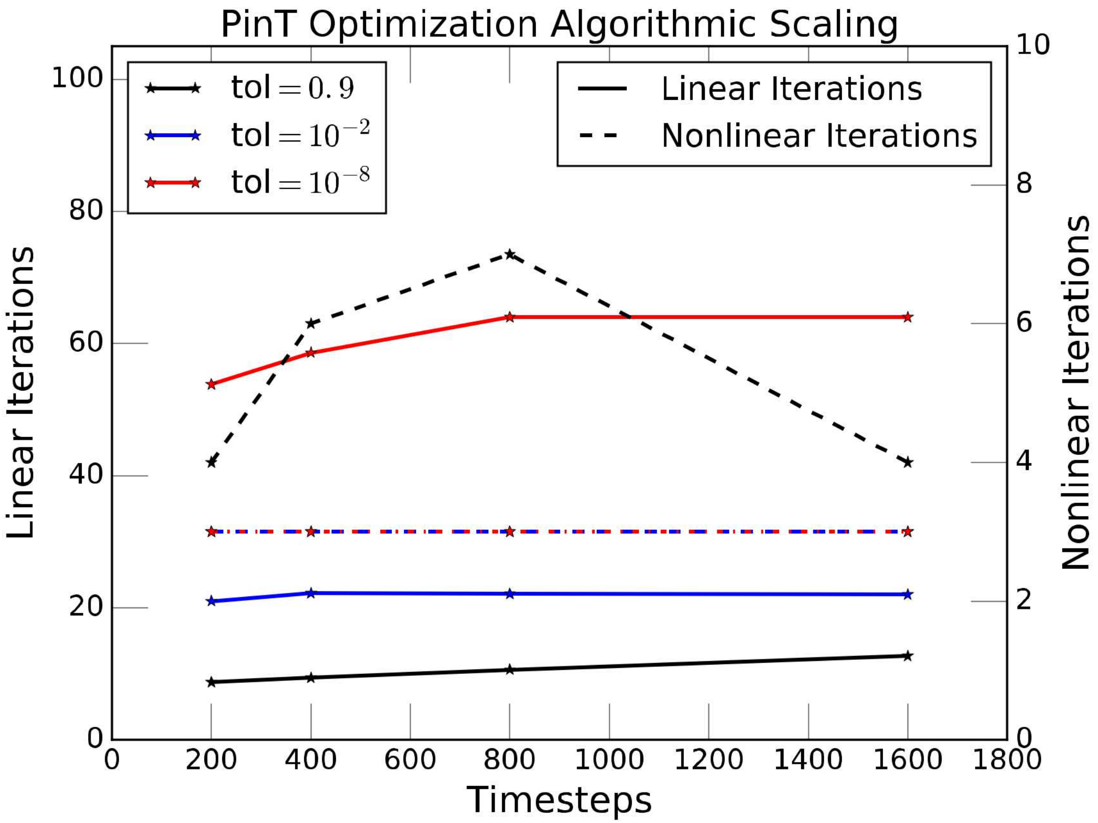
1D viscous Burgers control

$$\min_{u,z} \; \frac{1}{2}\int_0^T \int_0^1 (u(x,t) - \bar{u}(x,t))^2 + \alpha z(x,t)^2 \, dx dt$$

$$\begin{aligned} \text{subject to } \; & \partial_t u(x,t) - \nu \partial_{xx} u(x,t) + \partial_x (u(x,t)^2) = z(x,t) \\ & u(0,t) = u(1,t) = 0, \; u(x,0) = u_0(x) \end{aligned}$$

- Will use an inexact SQP algorithm, that requires KKT solves in the form discussed previously
- MATLAB implementation will demonstrate scalability
- We use “exact” KKT subdomain solves for this problem (not Wathen)

1D Burgers Control: SQP iterations



Flat linear iteration counts, combined with flat optimization iteration counts with respect to time step size leads to a scalable method*

*Caveat: This examples uses a direct solve for the KKT matrix, in general we are abusing the approximate block factorization preconditioner, as a smoother

Results: Control of the heat equation

Optimal control of the heat equation on a rectangular domain

- Finite element discretization in space: 60x20 mesh
- We focus on a single augmented system with appropriate right-hand side
- Serial baseline: GMRES with *Stoll, Wathen (2013)* approximate Schur preconditioner.
- Parallel results all use a 4-level multigrid solver

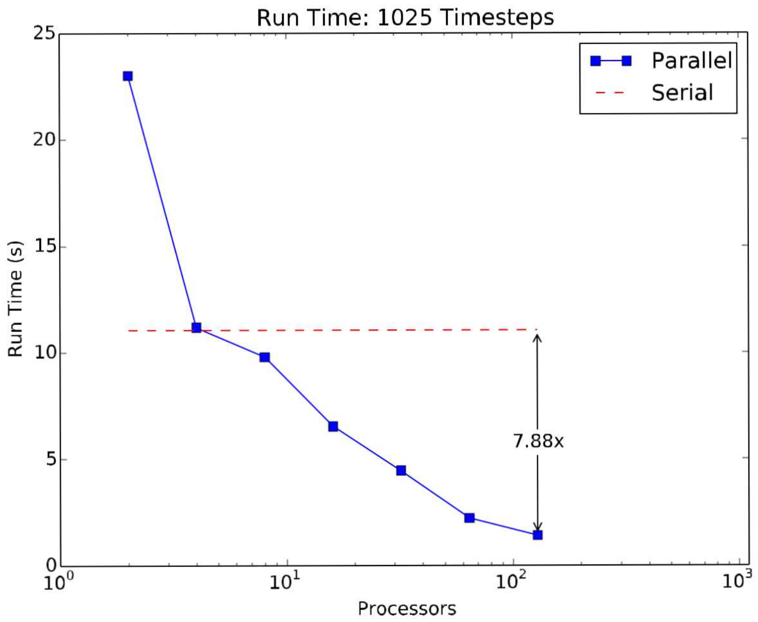
Implementation

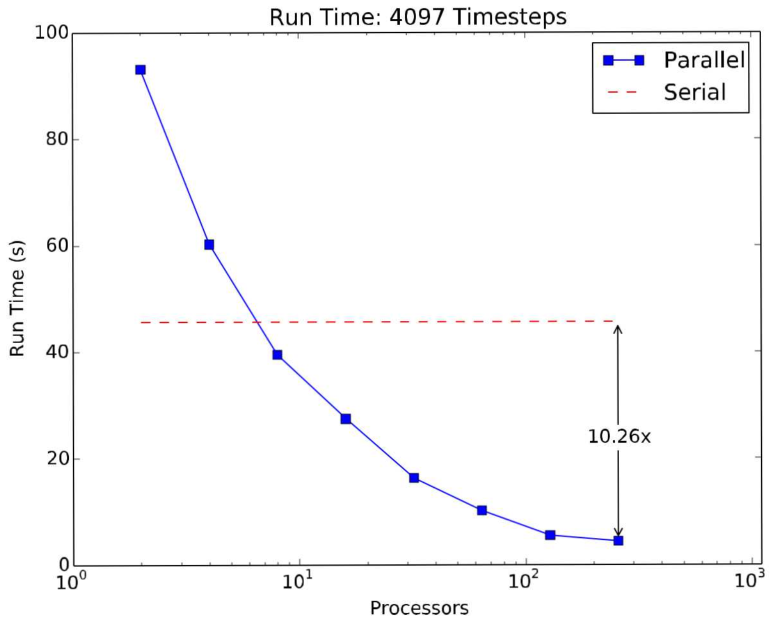
- Rapid Optimization Library (ROL) in Trilinos.
- Developed an interface for dynamic optimization
- Example implemented by Drew Kouri.

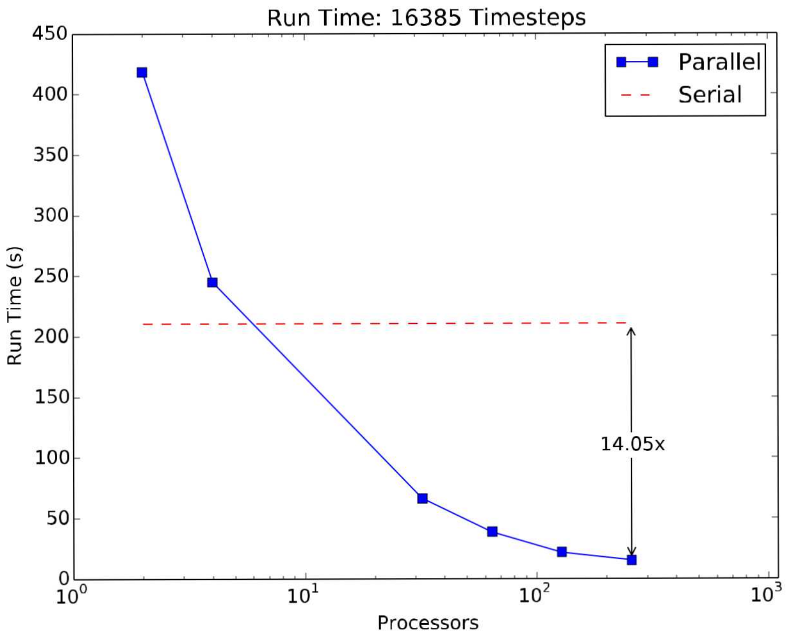
Optimal control of the heat equation

First the good news, real speedups!

- Need to improve parallel distribution to go to more processors





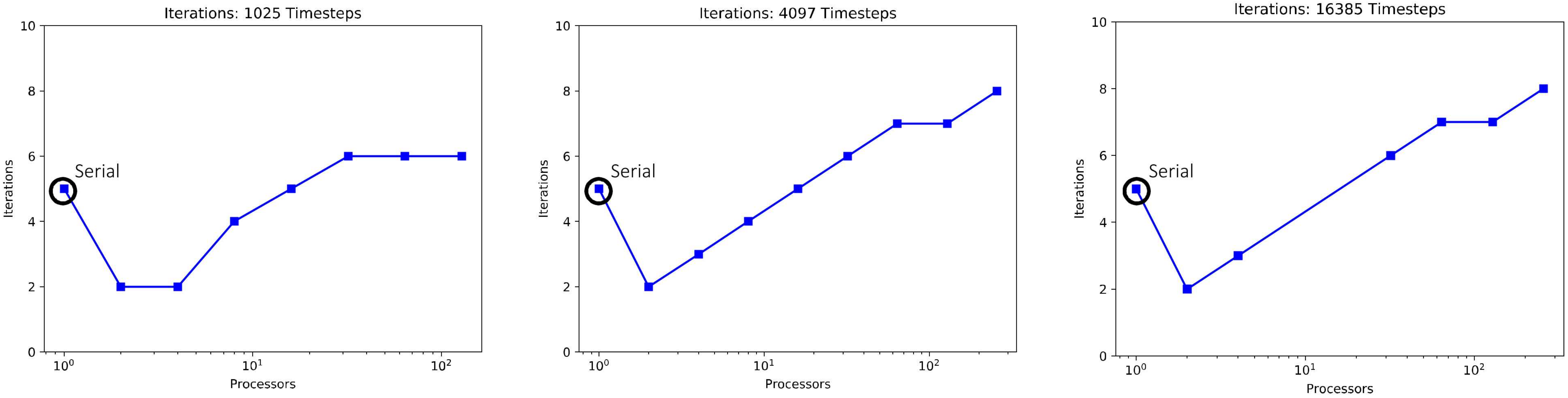


Note: We benefit heavily from the parallel distribution of the forward operator!

Optimal control of the heat equation

Now the bad news, iterations don’t scale with processor count

- *Glass half full perspective*: Opportunity for more speedups!
- Appears scalable with respect to number of time steps



Currently working on why this isn’t scaling

- Evidence from other problems suggests it is our smoother

Closing Thoughts

Developed a new “Box” initialization scheme

- Good initialization can improve the training algorithm
- Prevents collapse by allowing growth of feature space
- Limits growth to prevent blow up
- “Box” ReLU-ResNet models get convergence with depth

Developed a Layer-Parallel algorithm for training very deep NNs

- Parallelism is exposed by permitting inexact propagation
- We can take advantage of that with multigrid algorithms: achieve 10x speedup!
- Increases available parallelism and achieves 10x speedups
- More speedup possible, improve implementation, new multi-grid solvers (elliptic in time)

Papers:

- Guenther, Ruthotto, Schroder, Cyr, Gauger, Layer-Parallel Training of DNNs, Accepted to SIMODs, 2019
- Cyr, Guenther, Schroder, Nested Iteration Initialization of DNNs, Submitted to PinT Proceedings, 2019
- Cyr, Gulian, Patel, Perego, Trask, Training and Initializing DNNs, Submitted MSML, 2019