جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

CEMSE - Computer, Electrical, and
Mathematical Science and Engineering

Decomposition Methods for Coupled and Strongly Non-linear Problems in HPC

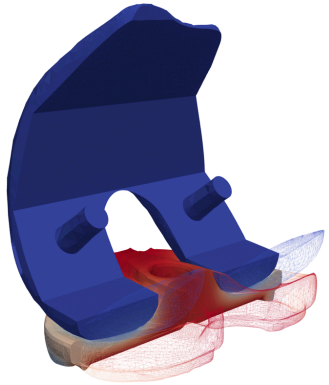
R. Krause, B. Capriqi, S. Likaj, S. Cruz, K. Trotti, C. Groß, P. Zulian

Rolf Krause, Simulation and Machine Learning in Science, Engineering, and Medicine
KAUST, Saudi Arabia
AMCS - CEMSE Division

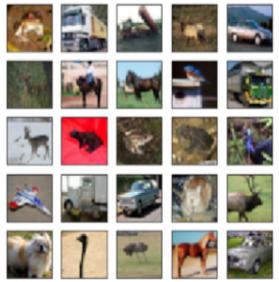
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

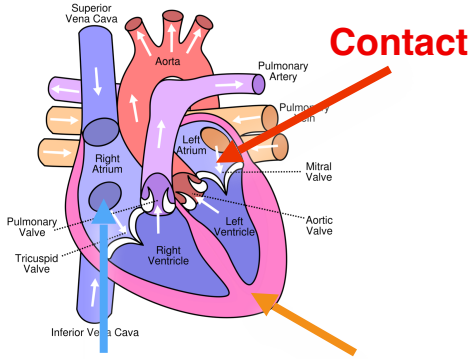
CEMSE - Computer, Electrical, and
Mathematical Science and Engineering

Computational Mechanics



Machine Learning Optimization





Contact

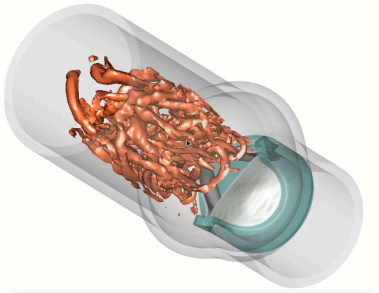
Fluid

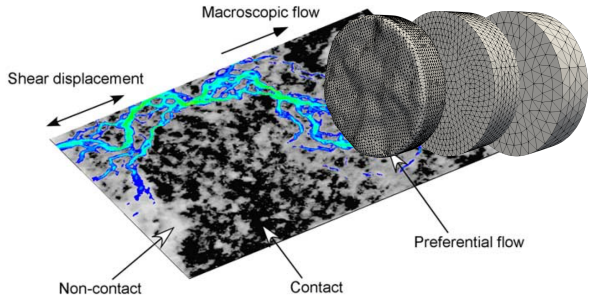
Elasticity/
Electrophysiology

HPC

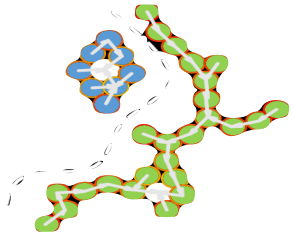
Abstraction
Flexibility
Application

Computational Medicine





Computational Geophysics



Data Driven Computing
Data Analysis

Fluid Structure Interaction

- $\mathcal{J}: \mathbf{H} \longrightarrow \mathbb{R}$ (non-)convex functional: stored energy function, loss function, error, ...
- constraints: $\mathbf{u} \in \mathcal{K}$: equality/inequality constraints

$$\mathcal{J}(\mathbf{u}) = \min_{\mathbf{v} \in \mathcal{K}} \mathcal{J}(\mathbf{v})$$

Direct minimization $J(\mathbf{u}^0) \geq J(\mathbf{u}^1) \geq J(\mathbf{u}^2) \geq \cdots \geq J(\mathbf{u}), \ \mathbf{u}_i \in \mathcal{K}$
gradient methods, sequentiell coordinate minimization, Newton-methods,...

First order necessary conditions: solve non–linear equation

$$\mathrm{F}(\mathrm{u}) = \nabla \mathrm{J}(\mathrm{u}) = 0 \Leftrightarrow \quad J'(\mathbf{u})(\mathbf{v}) = 0\,, \qquad \mathbf{v} \in H\,.$$

Newton-methods, interior points,penalty,...

Dual view

Either minimise J or solve F(u) = 0

Newton's method

$u^k \in H$ Newton's method replaces F by the linear model

$$F(u^k + c^k) \approx F(u^k) + F'(u^k)c^k = 0$$

leading to the the Newton correction

$$c^k = -F'(u^k)^{-1}F(u^k)$$

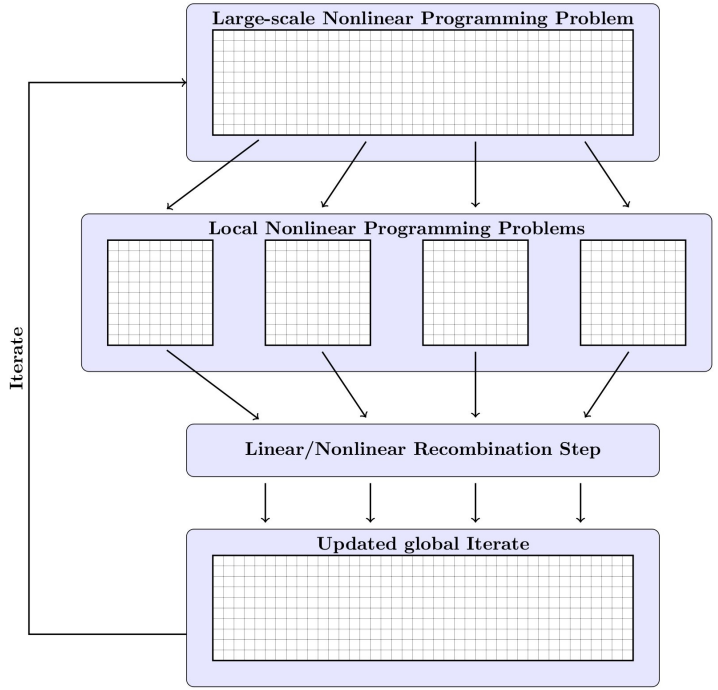
and the Newton update

$$c^{k+1} = c^k + \alpha^k c^k, \tag{1}$$

- $\alpha^k \geq 0$ damping or line-search parameter for the Newton correction.
- Invariance under affine transformations [Ortega '70, Deuflhard, Heindl '79, Deuflhard ..., '11]
- special case $F = \nabla J$
- Solve large linear systems in parallel: preconditioner, sparse linear algebra, iterative solvers

"Linear HPC"

Non-linear Domain Decomposition



- handling the nonlinearity
  - Newton first linearize, then decompose (multigrid, DD as inner solver)
  - nonlinear DD first decompose, then linearize - choice of sub-space/sub-domain model
- global communication - choice of "coarse space"
- convergence control

"Distributed Design: Nonlinear HPC"

Properties of the Newton Direction [Deuflhard '11]

General level set function $T(u|A) = \frac{1}{2}\|AF(u)\|_2^2$, A regular:

$$\nabla T(u|A) = (AF'(u))^T (AF(u))$$

The "natural" choice $A = F'(u)^{-1}$ leads to

$$-\nabla T(u|F'(u)^{-1}) = -F'(u)^{-1}F(u)$$

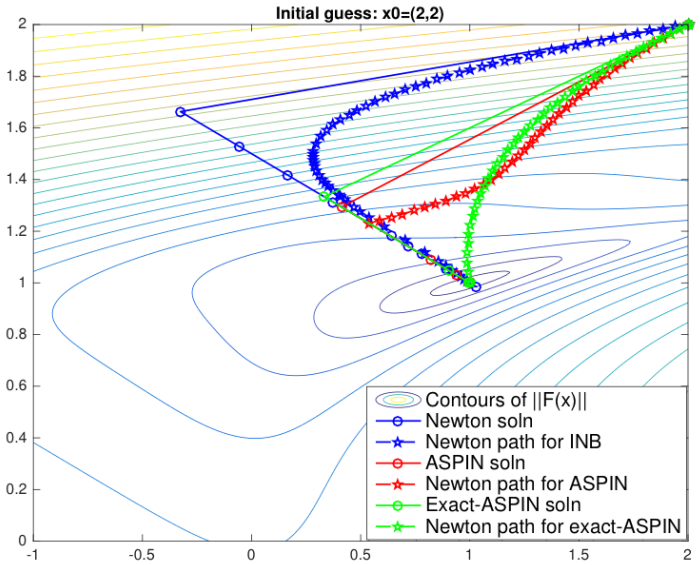
- The Newton correction is direction of steepest descent for $T(u|F'(u)^{-1})$
- Damping strategy for the exact Newton method can be derived using $T(\cdot|A)$

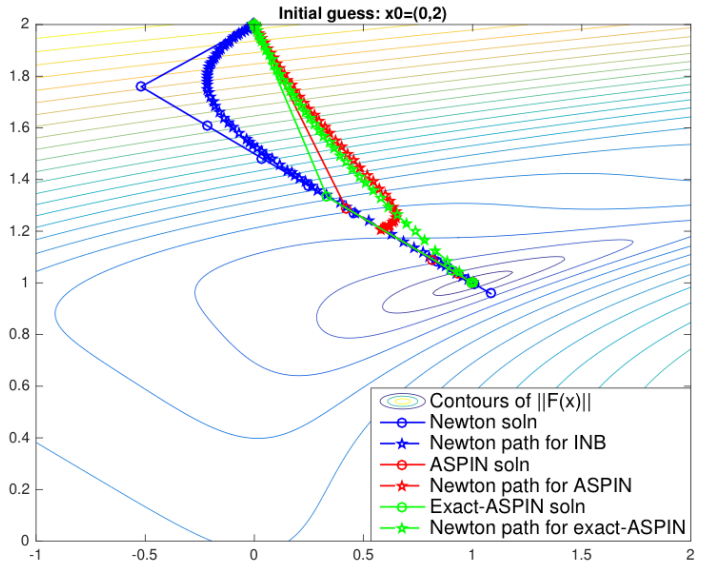
How to interpret search directions?

$$
\begin{aligned}
F_1(x_1, x_2) &= (x_1 - x_2^3 + 1)^3 - x_2^3, \\
F_2(x_1, x_2) &= x_1 + 2x_2 - 3.
\end{aligned}
$$

The exact solution is $u^* = [1, 1]^T$.

- INB: Inexact Newton with backtracking
- ASPIN: standard ASPIN
- Exact-ASPIN: ASPIN with analytical Jacobian for the preconditioned system





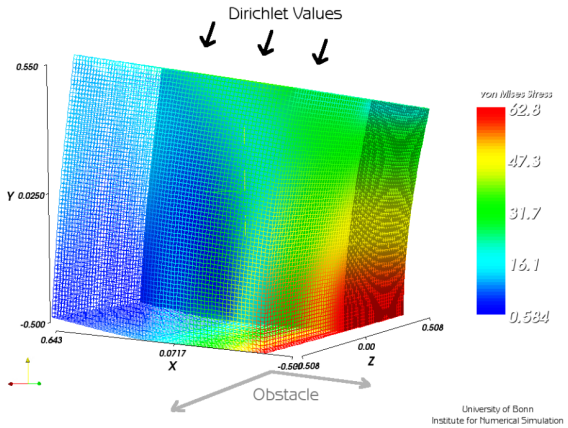
[Cai, Keyes; 2002; L. Liu, D. Keyes, K', 2018, Kothari, Kopanicakove, K', 2022]

Non-convex Model Problem and Applications

Solution of arbitrary non–linear problem

$$u \in \mathcal{B} \subset \mathbb{R}^n : J(u) = \min!$$

$\mathcal{B} = \{v \in \mathbb{R}^n : \underline{\phi} \leq v \leq \overline{\phi}\}$ a set of admissible solutions, J continuously differentiable objective function. $\underline{\phi}, \overline{\phi} \in \mathbb{R}^n$.



Contact problem with highly non–linear objective function

Applications:

- **Nonlinear Elasticity**
- Computer Vision
- Neuroinformatics

Solution is carried out employing a globalization strategy

- Trustregion Strategy
- Linesearch Strategy

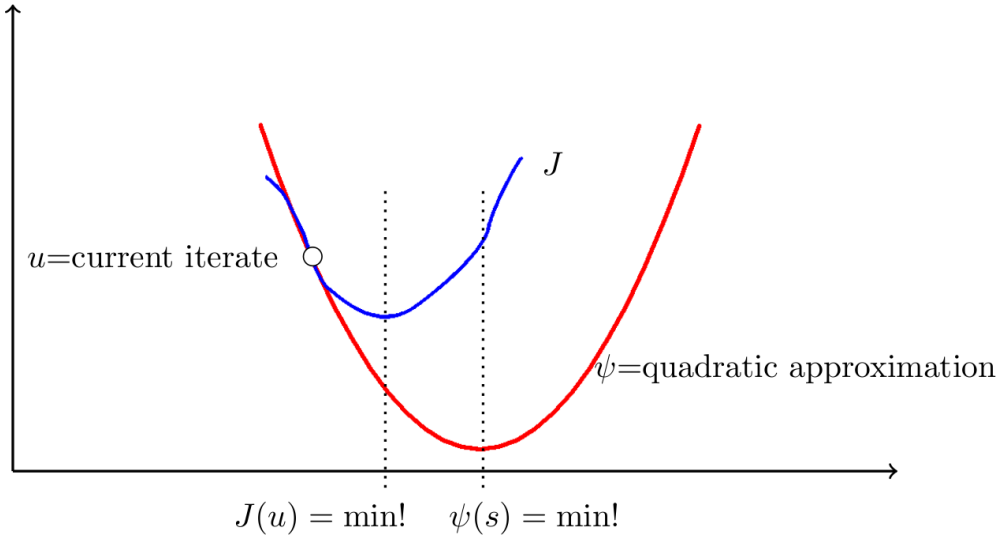
Trust Regiones Methods

Iterative Method, initial iterate can be chosen almost arbitrary

1. Newton-step: Solve

$$s \in \mathbb{R}^n : \psi(s) = \tfrac{1}{2}\langle s, Bs\rangle + \langle \nabla J(u), s\rangle = \min!$$
$$\text{such that } \|s\| \leq \Delta,\ u + s \in \mathcal{B}$$

where B is a symmetric approximation the Hessian(Quasi-Newton-Method)



Quadratic approximation to a nonlinear function

Trust Regiones Methods

Iterative Method, initial iterate can be chosen almost arbitrary

1. Newton-step: Solve

$$s \in \mathbb{R}^n : \psi(s) = \tfrac{1}{2}\langle s, Bs\rangle + \langle \nabla J(u), s\rangle = \text{min!}$$
$$\text{such that } \|s\| \leq \Delta,\ u + s \in \mathcal{B}$$

   where B is a symmetric approximation the Hessian(Quasi-Newton-Method)

2. Acceptance: $\rho = \frac{J(u+s)-J(u)}{\psi(s)} \geq \eta$ then: $u^{\text{new}} = u + s$, otherwise $u^{\text{new}} = u$, $\eta \in (0, 1)$.
3. Update of the Trust-Region: Δ by means of ρ. Iterate!

Theorem

If $\psi(s) = \text{min!}$ is solved accurately enough, the gradients and B are bounded on a compact set, then the method computes a globally converging sequence of iterates

The local Objective Function [Nash '00]

Choose the particular nonlinear, local objective function

$$H_k(u_k) = J_k(u_k) + \langle R_k \nabla J(u^G) - \nabla J_k(P_k u^G), u_k \rangle$$

- J_k is an a priori given nonlinear function (continuously differentiable)
- $R_k = (I_k)^T$

Properties of the coupling term

It holds $\nabla H_k(P_k u^G) = R_k \nabla J(u^G)$. This yields

$$\frac{J(u^G + \sum_k I_k s_k) - J(u^G)}{\sum_k (H_k(P_k u^G + s_k) - H_k(P_k u^G))} \to 1 \qquad \text{for } \|s_k\| \to 0$$

APTS Nonlinear Additively Preconditioned Globalization Strategies

1. Decompose $\mathbb{R}^n$ into N subsets D_k such that $\mathbb{R}^n = \bigcup_k I_k D_k \subset \mathbb{R}^n$.
2. Employ on each D_k a Trust-Region method to solve

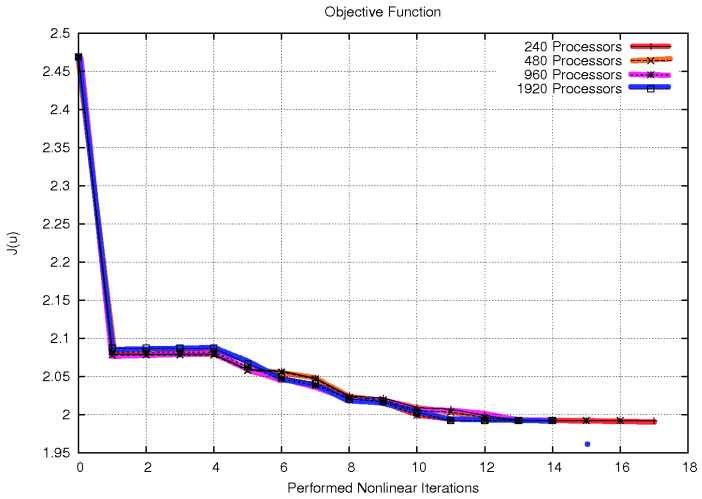
$$s_k \in \mathcal{B}_k : H_k(P_k u^G + s_k) < H_k(P_k u^G) \text{ such that } \|I_k s_k\| \leq \Delta^G$$

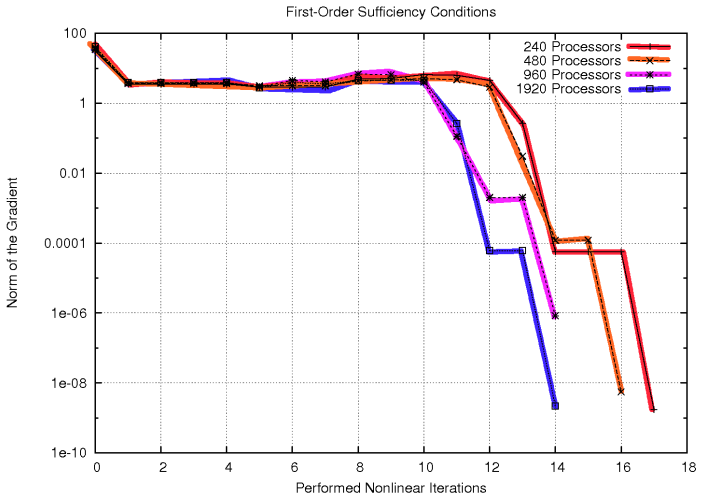
   where
   - $u^G \in \mathbb{R}^n$ is the current global iterate, Δ^G the current global Trust-Region radius,
   - $\mathcal{B}_k$ local admissible corrections,
   - $H_k : D_k \to \mathbb{R}$ a particular, local objective function,
   - $I_k : D_k \to \mathbb{R}^n$ (prolongation) and $P_k : \mathbb{R}^n \to D_k$ (Projection)
3. Combine s_k as follows

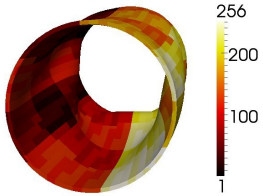
$$u^{G,\text{new}} = \begin{cases} u^G + \sum_k I_k s_k & \text{if } \rho_A = \frac{J(u^G) - J(u^G + \sum_k I_k s_k)}{\sum_k (H_k(P_k u^G) - H_k(P_k u^G + s_k))} \geq \eta \\ u^G & \text{otherwise} \end{cases}$$

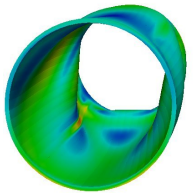
   where $I_k : D_k \to \mathbb{R}^n$. Update Δ^G by means of ρ_A.
4. Compute $\tilde{s}$ employing a Trust-Region method. $u^{G,\text{new}+1} = u^{G,\text{new}} + \tilde{s}$

Comparisons









Evolution of the objective function $J(u^i)$ and the norm of the gradient $\|g^i\|$ for globalized Aspin employing different numbers of processors

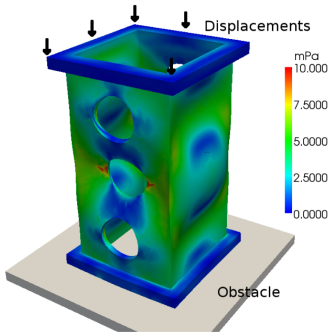
	240 cores	480 cores	960 cores	1920 cores
Overall Time	196.49	105.98	57.24	44.50
Solver global TR problem	70.72	40.43	25.25	22.26
Solver local QP Problem	4.43	1.82	0.43	0.30
Assembling	66.39	40.17	19.32	13.89
Nonlinear Iterations	17	16	14	14

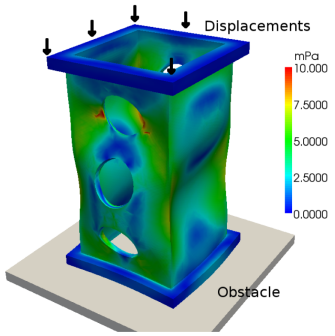
Computation time in seconds.

Cylinder Contact Problem

Performance of Trust-Region Methods

	Newton it.	parallel cg it.	Time
seq. Trust-Region	137	54,800	1.0
APTS	112	44,800	1.10
MPTS	73	29,200	0.61
AMPTS	45	18,000	0.50





- runtime comparison ($\mathcal{F} \mathrel{\widehat{=}} 4$ local Trust-Region steps on each D_k, 4 global Trust-Region steps in order to compute $\tilde{s}$)
- time is measured relatively to the sequential Trust-Region method

Contact in Linear Elasticity

$$
\begin{aligned}
J(\mathbf{u}) &= \frac{1}{2}a(\mathbf{u},\mathbf{u}) - f(\mathbf{u}) = \frac{1}{2}\int_\Omega \boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{u})\, d\Omega - \int_\Omega \mathbf{f}\mathbf{u}\, d\Omega \\
\mathcal{K} &= \{\mathbf{u} \in \mathbf{H} \,|\, \mathbf{u}\cdot\mathbf{n} \leq g \text{ on } \Gamma_C\}
\end{aligned}
$$

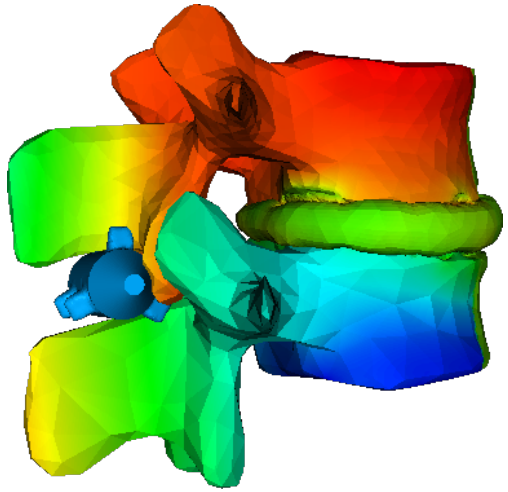
$\mathbf{n}$: outer normal g: normal distance to obstacle Γ_C: possible contact boundary

Linear Elasticity

$$
\begin{aligned}
-\sigma_{ij,j} &= f_i & &\text{equilibrium conditions} \\
u_i &= 0 & &\text{on } \Gamma_D \\
\sigma_{ij}\cdot\mathbf{n}_j &= p_i & &\text{on } \Gamma_N
\end{aligned}
$$

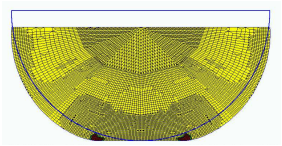
\+ Contact conditions on Γ_C

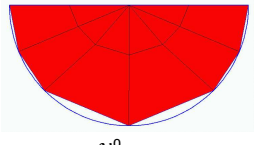
$$
\begin{aligned}
\sigma_T &= F(u_T) & &\text{friction law} \\
\sigma_n(\mathbf{u}\cdot\mathbf{n} - g) &= 0 & &\text{complementarity} \\
\mathbf{u}\cdot\mathbf{n} - g &\leq 0 & &\text{no penetration} \\
\sigma_n &\leq 0 & &\text{contact pressure}
\end{aligned}
$$



Multigrid

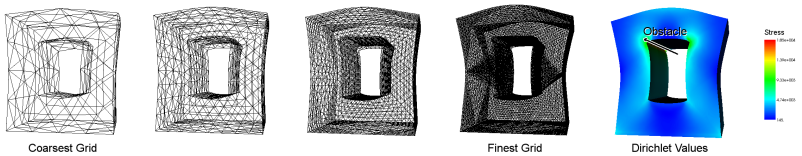
Replace Computation by Approximation





$$u_h = \underbrace{u_h - u_H}_{\text{fine level}} + \underbrace{u_H}_{\text{coarse level}} \qquad u_h \in X_h, \quad u_H \in X_H$$

$u_h - u_H$	can be computed easier than	u_h	smoothing resolve non-linearities
X_H	of smaller dimension than	X_h	approximation
J_H	can be computed easier than *or* can be different to	J_h	model for correction



RMTR [Gratton et al. 2008; Gratton et al. 2009; Groß, K' 2009]

1. **compute** m_1 pre–smoothing trust–region steps to approximately solve
   $H_k(u_k) < H_k(P_{k+1}u_{k+1})$ w.r.t $u_k \in \mathcal{B}_k, \|u_k\| \leq \Delta_k$
2. **if** (k is not coarsest level)
   - Compute $\mathcal{B}_{k-1}$, and H_{k-1}, $u_{k-1,0} = P_k u_{k,m_1}$
   - call RMTR on level $k-1$ and receive a correction s_{k-1}

$$u_{k,m_1+1} = \begin{cases} u_{k,m_1} + I_{k-1}s_{k-1} & \text{if } \rho_M = \frac{H_k(u_{k,m_1}) - H_k(u_{k,m_1} + I_{k-1}s_{k-1})}{H_{k-1}(P_k u_{k,m_1}) - H_{k-1}(P_k u_{k,m_1} + s_{k-1})} \geq \eta \\ u_{k,m_1} & \text{otherwise} \end{cases}$$

   - Update trust-region Δ_{k,m_1+1}
3. **compute** m_2 post–smoothing trust–region steps to approximately solve
   $H_k(u_k) < H(u_{k,m_1+1})$ w.r.t $u_k \in \mathcal{B}_k, \|u_k\| \leq \Delta_k$
4. **return** final iterate

Monotone Multigrid Methods [Kornhuber] ensure reduction of H by exploiting convexity and by designing appropriate coarse spaces and constraints.

Multigrid for Constrained Minimisation

Algorithm

Hierarchy of meshes
Transfer operators
Smoothing operators

Approximation View

Hierarchy of approximation spaces
Inverse inequality
Approximation property

Optimization View

Hierarchy of quadratic models
Multilevel minimisation
Level dependent constraints

- Dirichlet BC: equality constraints ⟷ contact conditions: inequality constraints
- Coarse level basis functions need to live in the kernel of the constraints
- Coarse level spaces need to be as rich as possible (approximation property)
- Active set a priori unknown ⟷ multilevel hierarchy unknown (nonlinear approximation)

Challenge: coarse level spaces as large as possible ⟷ incorporate contact constraints

Multigrid for Constrained Minimisation

(Non-smooth) Newton View

Outer non-linear iteration
Quadratic model for objective
Linearise constraints

Full Approximation Scheme

Non-linear problem on each level
No general convergence proof

Multilevel Optimisation

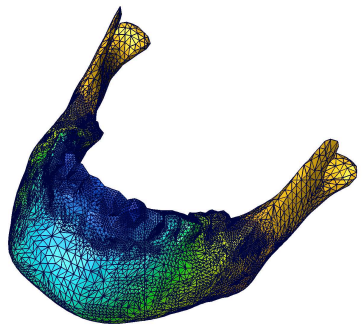
Multilevel basis as search directions
First order correction for consistency
Monotonicity for convergence

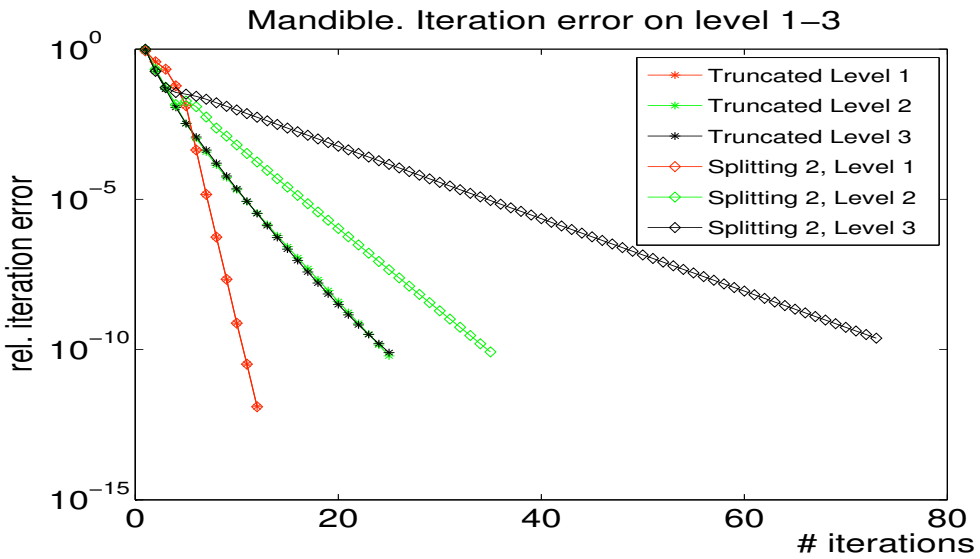
- Hessian might have negative Eigenvalues
- Coarse level basis functions become search directions

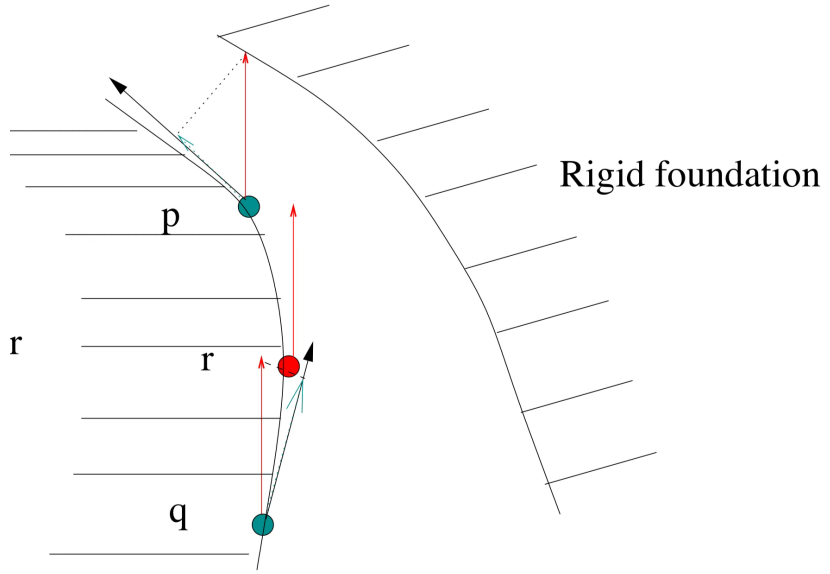
Optimisation view allows for non-linear corrections *inside* the multilevel hierarchy

Transfer Optimality of Multigrid to Non-linear and Non-smooth Problems

Direct Minimisation
Step 2: Do Multilevel Method



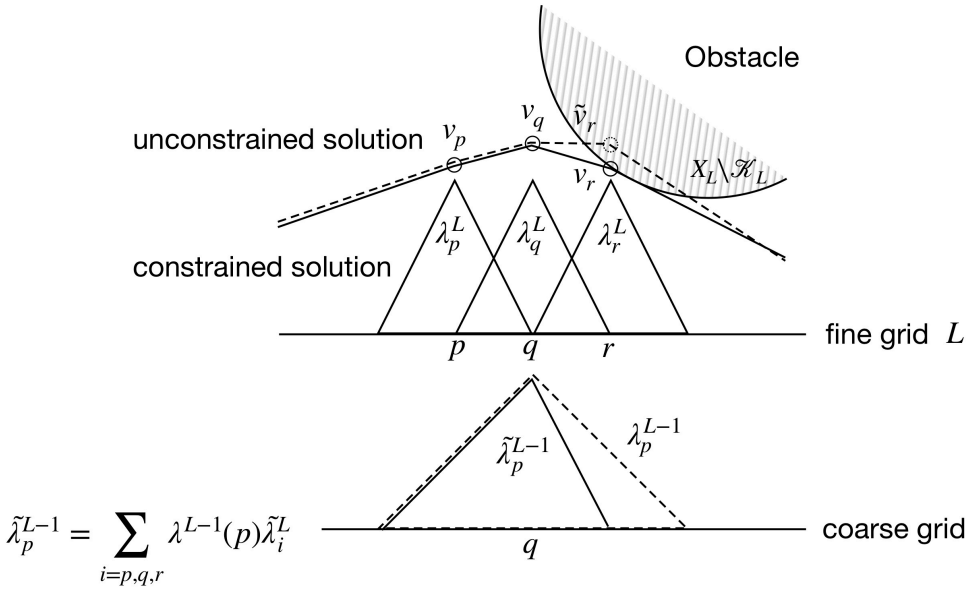




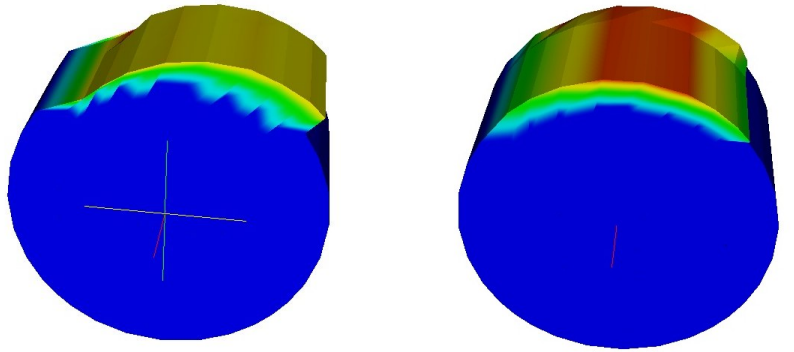
- Extended search directions need to respect constraints over their complete support
- Not possible for linear functions : use non-linear functions as search directions

Direct Minimisation
Step 3: Adapt Approximation Spaces

MG/DD for contact [Kornhuber, K' 99; K '01; Dickopf, K' 12; Kothari; K' 22]



coarse grid correction in tangential direction. One dof on the coarse grid



Left: standard multilevel-basis

Right: modified multilevel-basis

- Transform into normal/tangential coordinate system
- **Treat non-linearites localised inside the MG (non-linear smoother)**
- Coarse level basis incorporates constraints and cannot be evaluated on the coarsest mesh
- Inner approximation of the feasible
- Nonsmooth and globally convergent

Contact MG Hierarchy + Localised Non-linearities

Linear sparse solver pardiso [O. Schenk]

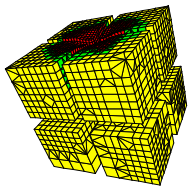
#dof	#nodes	decomp time (s)	peak memory
14.739	4.913	6,58	0,101GB
32.937	10.979	18,7	0,232GB
107.811	35.937	351,62	1,1GB
159.771	53.257	402,89	1,9GB

Linear multigrid for a linear problem
$\text{TOL} = 10^{-12}$

#dof	#nodes	solution time
14.739	4.913	3,05
107.811	35.937	29,0
823.875	274.625	238,3

non-linear multigrid for friction
$\mathcal{F} = 0.3$, $\text{TOL} = 10^{-12}$

- Friction resolved in smoother pointwise and excatly
- Constrained linearisation of friction law for sliding nodes
- Exact stick-slip detection
- Cost of about 1.5 linear solves - or 1.5 Newton steps
- Optimal complexity

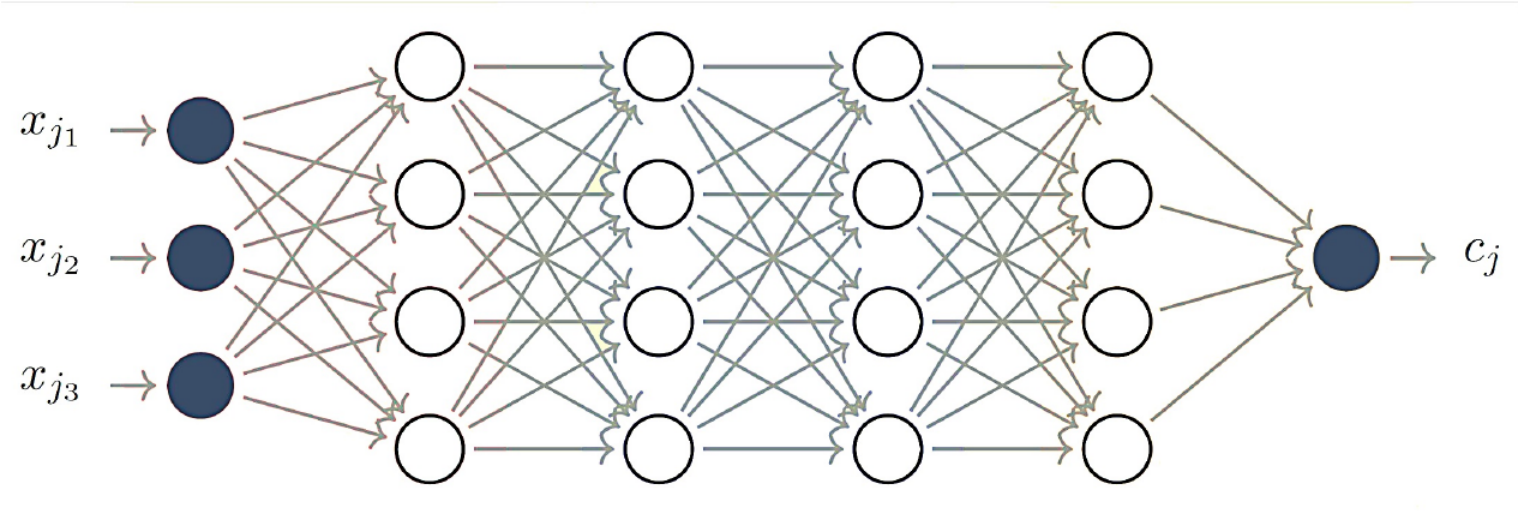


[Kornhuber, K' 99; K' 01]

Machine Learning

Neural Network

Fully connected NN with 1 input layer, 4 hidden layers and 1 output layer.



$\mathcal{N}(\boldsymbol{\theta}, x) = L_O(\ldots(L_2(L_1(x, \theta_1), \theta_2)\ldots), \theta_O)$, where $\theta = [\theta_1, \ldots, \theta_O]$ and

$$L_i(x, \theta_i) = \sigma(W_i x + b_i), \quad \theta_i = \begin{bmatrix} \mathrm{vec}(W_i) \\ b_i \end{bmatrix} \tag{1}$$

with σ being the non-linear activation function.

Machine Learning

Training as Optimization

- The loss function $\mathcal{L}: \mathbb{R}^n \longrightarrow \mathbb{R}^+$ is typically defined as

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{p} \sum_{i=1}^{p} \ell(\mathcal{N}(\boldsymbol{\theta}, \boldsymbol{x}_i), \boldsymbol{y}_i)$$

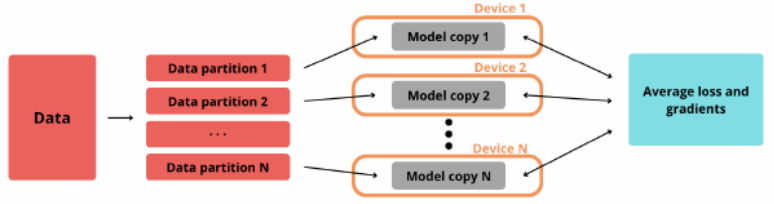
- $\ell(\cdot, \cdot)$ measures prediction error (e.g., cross-entropy, MSE).
- Training seeks parameters $\boldsymbol{\theta}^*$ for the neural network $\mathcal{N}$ such that the loss function $\mathcal{L}$ is minimized

$$\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta} \in \mathbb{R}^n} \mathcal{L}(\boldsymbol{\theta})$$

- In practice: high-dimensional, highly nonconvex objectives; ill-conditioned problem.
- First-order methods (SGD/Adam) dominate but can be sensitive to hyperparameters (such as learning rates).
- Parallelism improves throughput, but usually does not change the basic update logic.

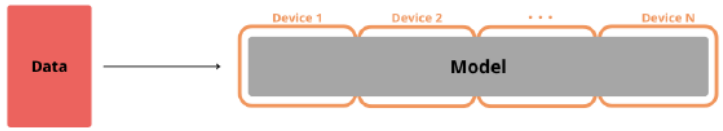
Parallelism

Model, data and model + data parallelism



- **Data parallel**:

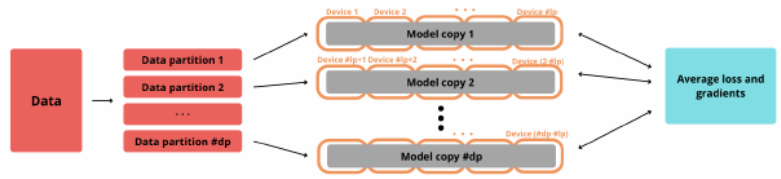
  The model and data partition are processed on each device



- **Model (layer) parallel**:

  Divides the model into different nodes/GPU devices

  A single forward and backward pass requires device communication



- **Hybrid parallelism**:

  Combines both approaches

Data-Parallel APTS for PINNs

Physics-Informed Neural Networks (PINNs)[1]

- Approximate the PDE solution with a neural surrogate $u_\theta(t,x)$ (typically an MLP).
- Enforce the physics by penalizing the PDE residual at interior collocation points.
- Train with a composite loss: interior residual + IC + BC.
- Practical difficulty: nonconvexity, loss-term imbalance, and *resampling-induced* gradient noise

$$r_\theta(t,x) = \mathscr{D}[u_\theta](t,x) - f(t,x)$$

$$L(\theta) = L_\Omega(\theta) + L_{\mathrm{IC}}(\theta) + L_{\mathrm{BC}}(\theta)$$

$$L_\Omega(\theta) = \sqrt{\frac{1}{|\mathscr{C}|}\sum_{(t_j,x_j)\in\mathscr{C}} \big(r_\theta(t_j,x_j)\big)^2}$$

$$L_{\mathrm{IC}}(\theta) = \sqrt{\frac{1}{|\mathscr{I}|}\sum_{x_j\in\mathscr{I}} \big(u_\theta(0,x_j) - u_0(x_j)\big)^2}$$

$$L_{\mathrm{BC}}(\theta) = \sqrt{\frac{1}{|\mathscr{B}|}\sum_{(t_j,x_j)\in\mathscr{B}} \big(u_\theta(t_j,x_j) - g(t_j,x_j)\big)^2}$$

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *J. Comput. Phys.*, 2019.

Data-Parallel APTS for PINN Optimization

- Adapt APTS to PINNs using a data-parallel **two-stage trust-region iteration**.
- Sample interior collocation points uniformly at random over the full space–time domain:

$$\mathscr{C}_k \sim \mathrm{Unif}(\Omega \times [0, T]).$$

- Define "subdomains" by randomly partitioning $\mathscr{C}_k$:

$$\mathscr{C}_k = \bigsqcup_{d=1}^{D} \mathscr{C}_k^d.$$

  Each partition still has **global coverage** of the domain, rather than representing a geometric subregion.
- IC/BC data are shared across all replicas.
- Each local problem is solved with a quadratic TR model using Steihaug truncated CG and autodiff Hessian-vector products.

Data-Parallel APTS for PINNs

Pseudo-Algorithm

- **Input:** initial radius Δ_0
- **(Re)sampling:** sample $(\mathscr{C}_k, \mathscr{I}_k, \mathscr{B}_k)$ and refresh every r outer iterations.
- **Stage I (local additive preconditioner):**
  - Partition interior points $\mathscr{C}_k = \bigsqcup_{d=1}^{D} \mathscr{C}_k^d$ and run D replicas in parallel. Each replica minimizes

$$L_k^d(\theta) = L_\Omega(\theta; \mathscr{C}_k^d) + L_{\mathrm{IC}}(\theta; \mathscr{I}_k) + L_{\mathrm{BC}}(\theta; \mathscr{B}_k).$$

    Take J local TR steps and sum them: $s_k^d = \sum_{j=1}^{J} s_{k,j}^d$ with a radius split Δ_k/J so $\|s_k^d\| \le \Delta_k$.
  - **Aggregate:** $\bar{s}_k = \frac{1}{D} \sum_{d=1}^{D} s_k^d$ and accept/reject preconditioned step.
- **Stage II (global TR):** compute a TR step for the full loss $L(\theta)$ at the accepted point; shrink/retry until accepted.

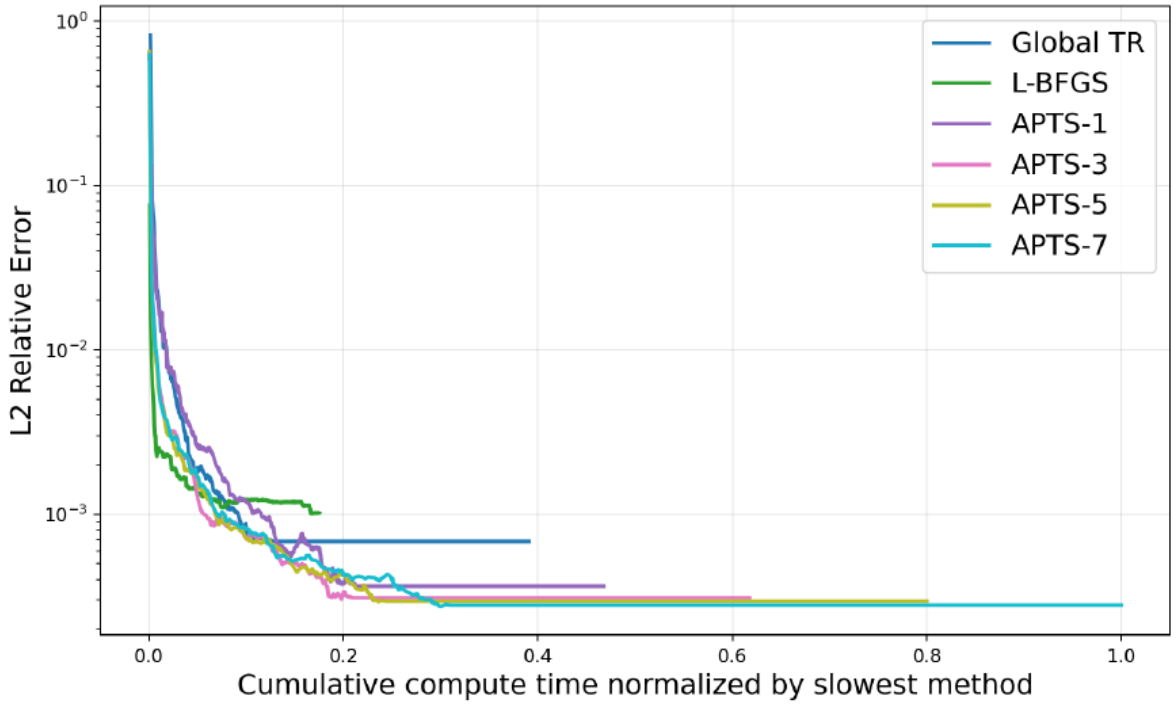
Data-Parallel APTS for PINNs

Experiments

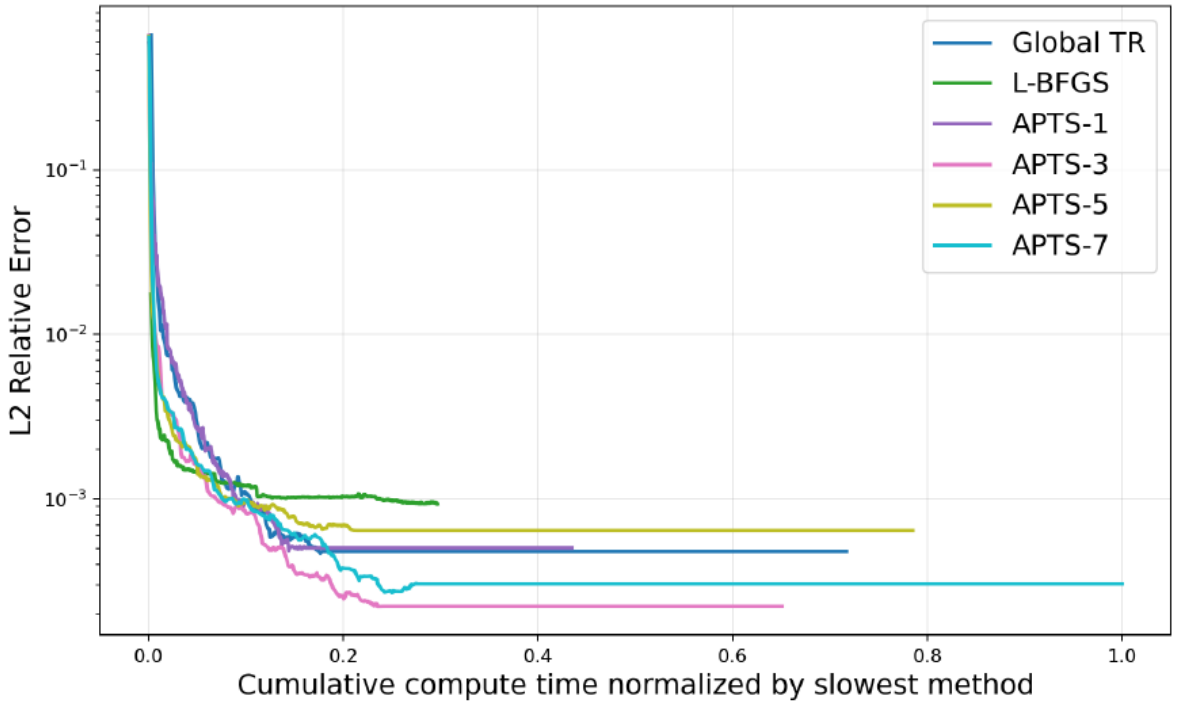
- Benchmarks: Heat, Burgers, Laplace.
- We compare LBFGS, Global TR and APTS-1,-3,-5 and -7, which refers to the number J of local TR steps.
- Training points per resample: 8000 interior (collocation), 1600 initial (IC), 1600 boundary (BC).
- Network: MLP with tanh, sizes $[2, 20, 20, 20, 1]$.
- Resampling every 10 iterations; results averaged from 3 random seeds.

New Results

L2 Relative Error vs Global Time (Burgers)



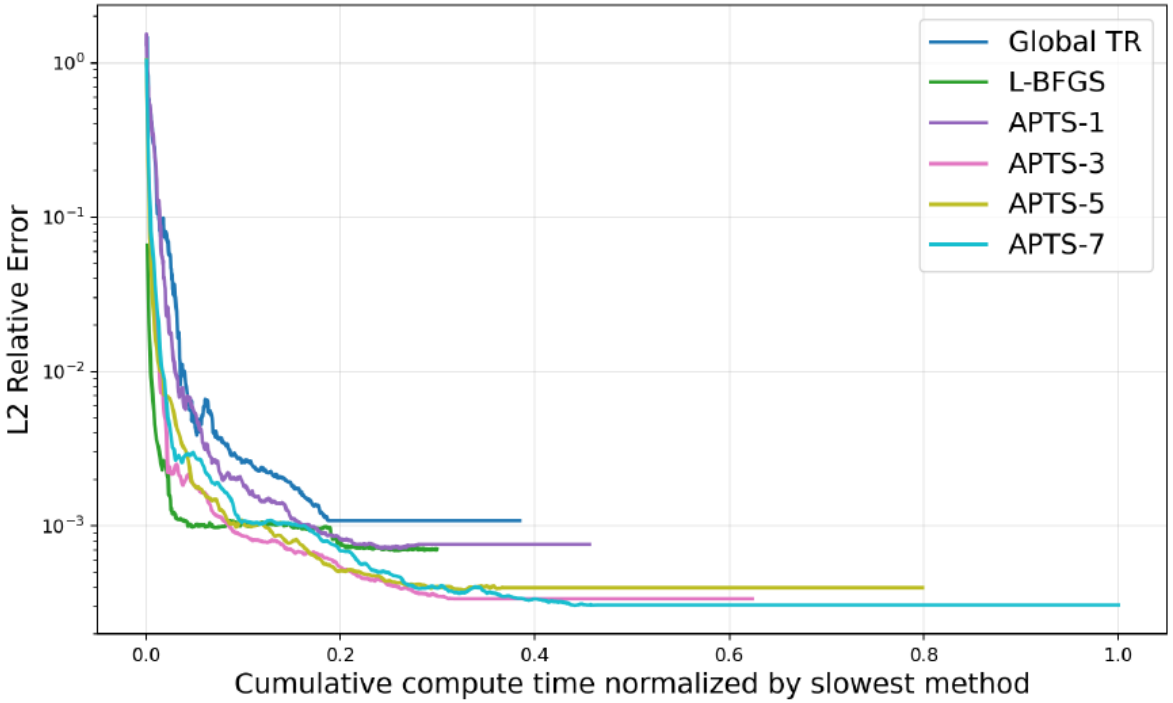
2 GPUs



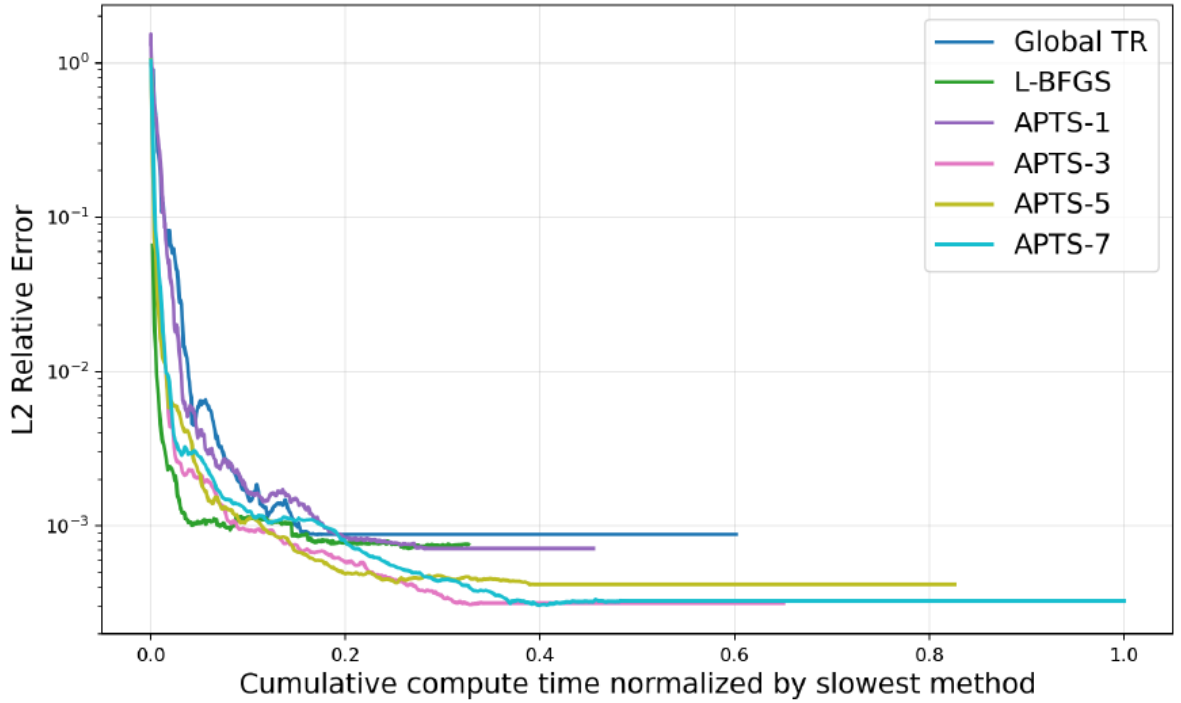
4 GPUs

New Results

L2 Relative Error vs Global Time (Heat)



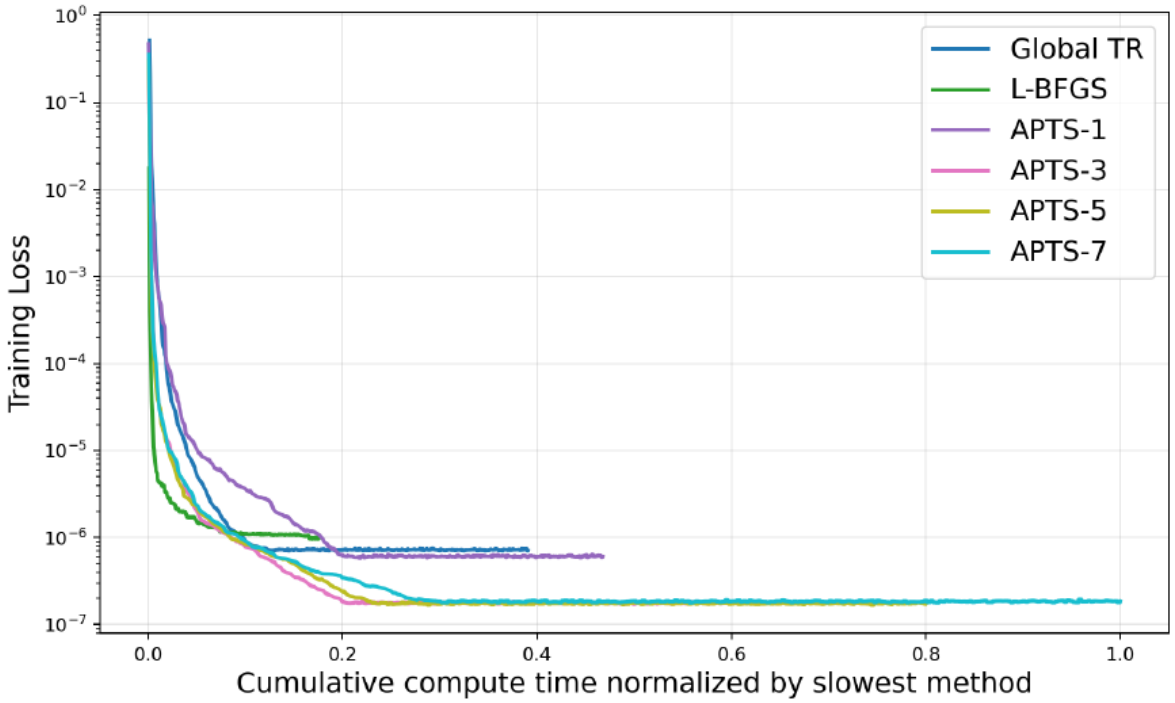
2 GPUs



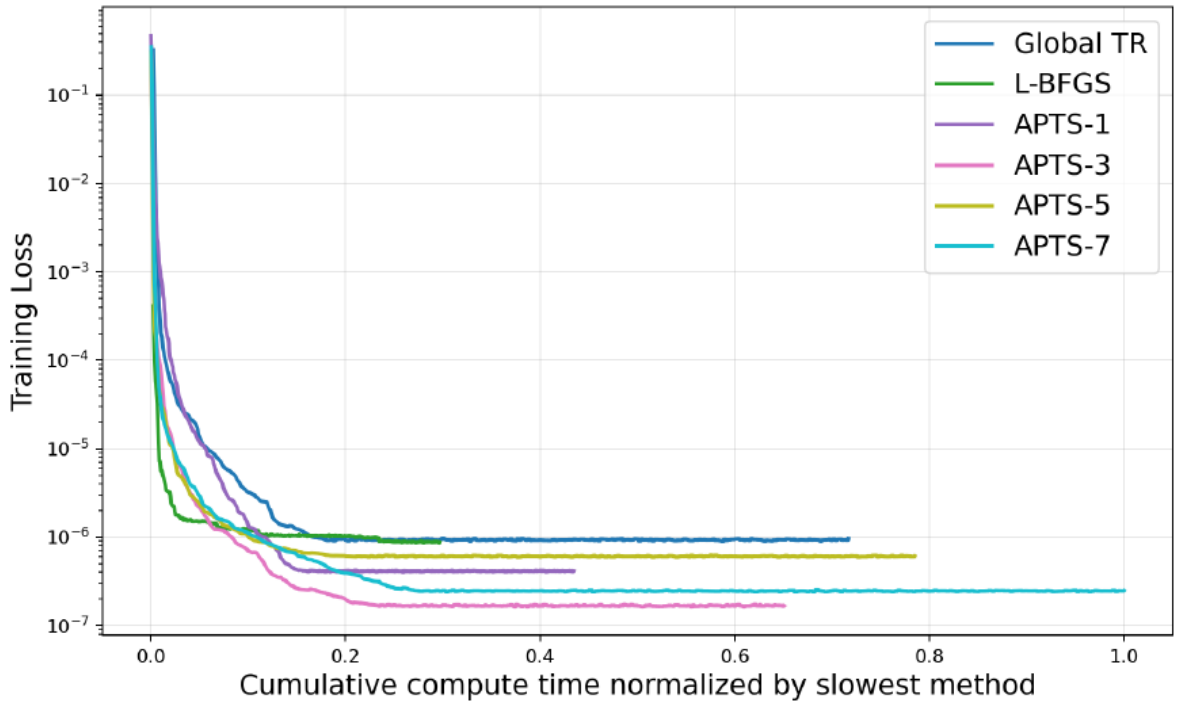
4 GPUs

New Results

Training Loss vs Global Time (Burgers)



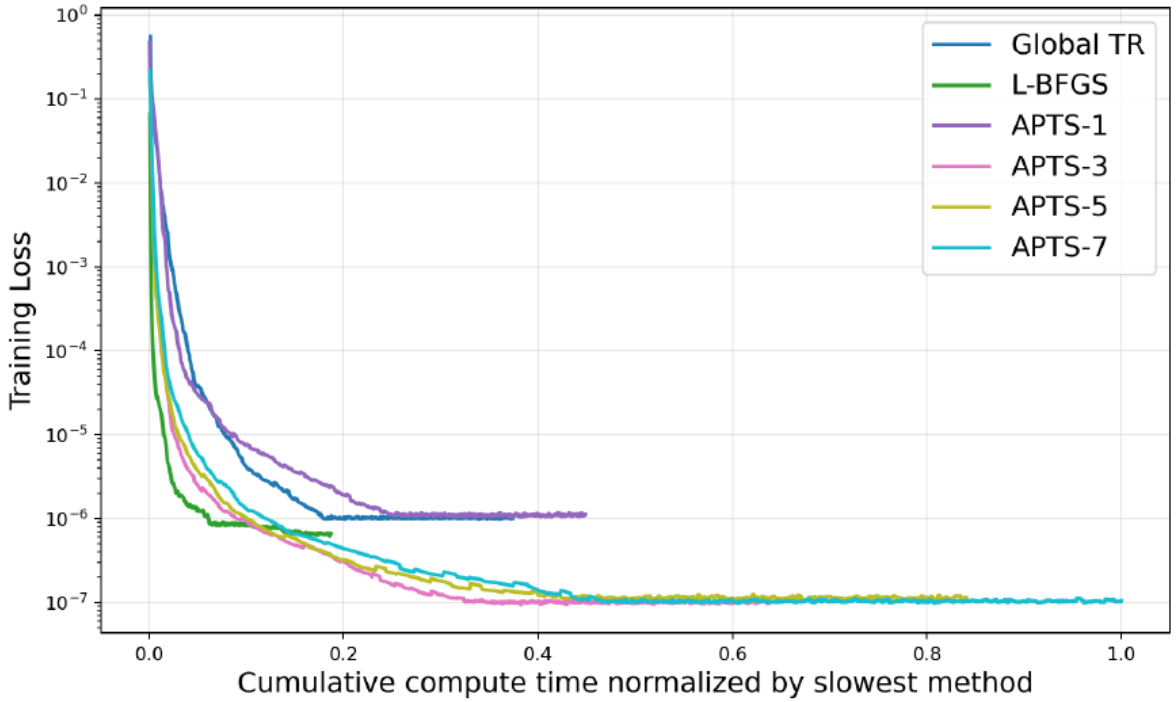
2 GPUs



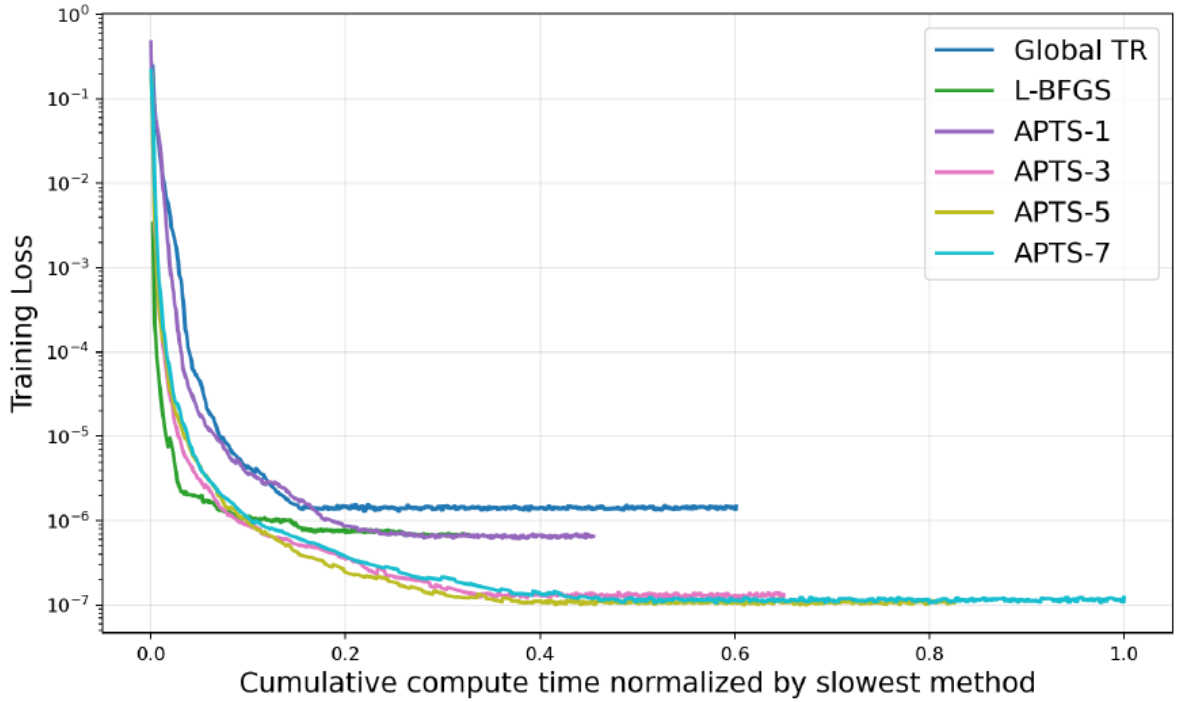
4 GPUs

New Results

Training Loss vs Global Time (Heat)



2 GPUs



4 GPUs

Nonlinear preconditioning framework[9,10]

- Recall: optmization problem

$$\boldsymbol{\theta}^* := \text{argmin}_{\boldsymbol{\theta}\in\boldsymbol{\Theta}}\ \mathcal{L}(\boldsymbol{\theta})$$

- Consider the framework of nonlinear system of equations

$$F(\boldsymbol{\theta}) := \nabla\mathcal{L}(\boldsymbol{\theta}) = 0$$

- Instead of solving $F(\boldsymbol{\theta}) = 0$, our goal is to construct and solve a nonlinearly preconditioned system of equations

$$\mathcal{H}(\boldsymbol{\theta}) = 0,$$

where

$$\mathcal{H}_L(\boldsymbol{\theta}) = G(F(\boldsymbol{\theta})) \quad \text{or} \quad \mathcal{H}_R(\boldsymbol{\theta}) = F(G(\boldsymbol{\theta}))$$

such that

  - Solution of $\mathcal{H}(\boldsymbol{\theta})$ should be identical to that of $F(\boldsymbol{\theta})$
  - Solving system $\mathcal{H}(\boldsymbol{\theta})$ should be easier than solving $F(\boldsymbol{\theta})$
  - $\mathcal{H}(\boldsymbol{\theta})$ should have more balanced nonlinearities
  - Computations involving $\mathcal{H}$ should be computationally feasible

[9] Cai, Keyes. Nonlinearly preconditioned inexact Newton algorithms. SIAM SISC. 2002.

[10] Dolean et al. Nonlinear Preconditioning: How to Use a Nonlinear Schwarz Method to Precondition Newton's Method. SIAM SISC. 2016.

Nonlinearly preconditioned training[11]

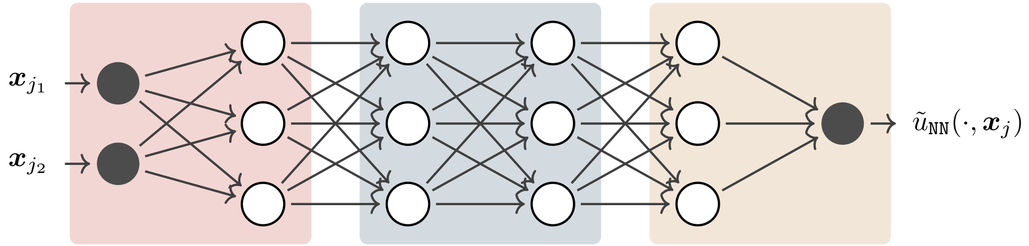
Using the right-preconditioning strategy, we construct a nonlinearly preconditioned system of equations as

$$\mathcal{H}_R(\boldsymbol{\theta}) := F(\underbrace{G_R(\boldsymbol{\theta})}_{=:\tilde{\boldsymbol{\theta}}}) = 0$$

- $\mathcal{H}_R(\boldsymbol{\theta})$ is constructed as composite multiplicative preconditioner
- $\tilde{\boldsymbol{\theta}} := G_R(\boldsymbol{\theta})$ is an outcome of an iterative process, e.g., a few iterations of some nonlinear iterative method (SGD, L-BFGS, Adam, AdaGrad, ...)
- $G_R(\boldsymbol{\theta})$ provides an improved initial iterate for the next optimization step

[11] **kopanicakova_pinnspqn_2023**.

Nonlinear Schwarz method for training NN



- Decompose parameter space $\boldsymbol{\Theta}$ into N_{sd} subnetworks: $\{\boldsymbol{\Theta}_i\}_{i=1}^{N_{sd}}$
- The restriction operators extracts the parameters associated with subnetworks i, i.e.,

$$\boldsymbol{R}_i : \boldsymbol{\Theta} \to \boldsymbol{\Theta}_i$$

- For a given initial guess $\boldsymbol{\theta}_i^{(0)} \in \boldsymbol{\Theta}_i$, where $\boldsymbol{\theta}_i^{(0)} \hookleftarrow \boldsymbol{R}_i \boldsymbol{\theta}^{(k)}$:

$$\text{Find } \boldsymbol{\theta}_i^* \in \boldsymbol{\Theta}_i \text{ that minimizes } \mathcal{L}_i(\boldsymbol{\theta}_i),$$

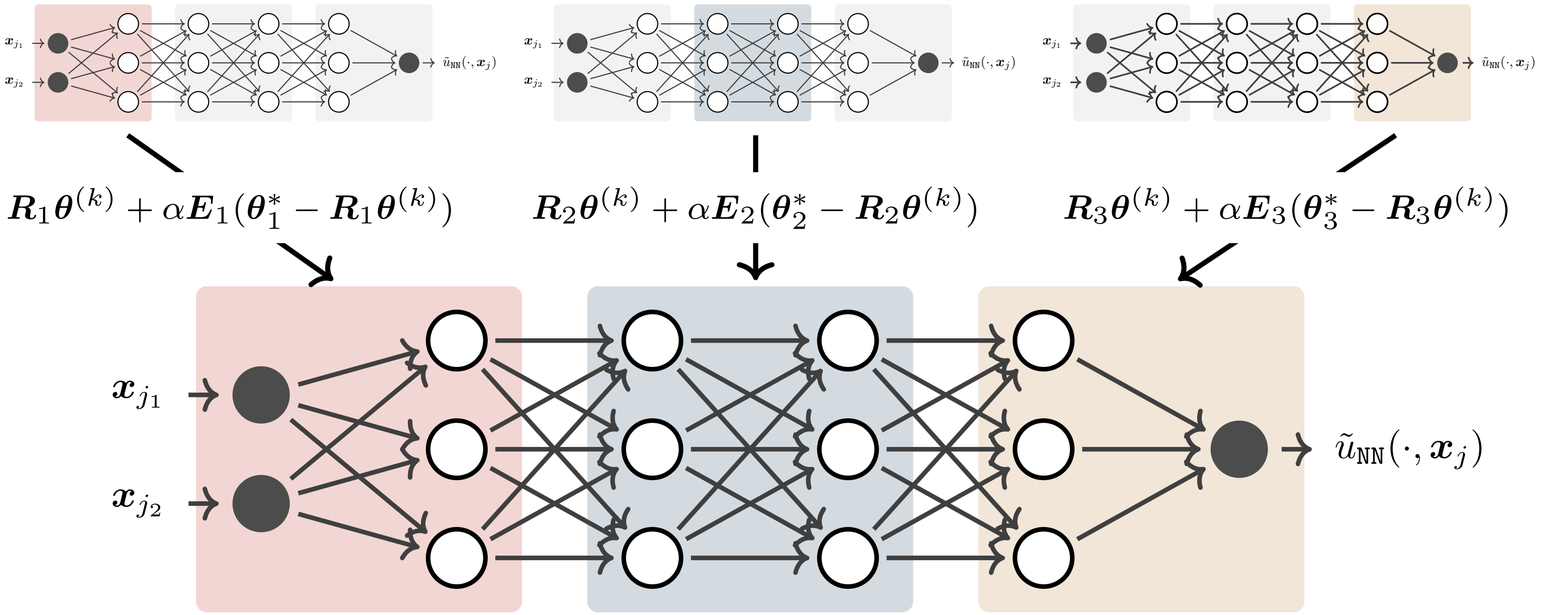
- The prolongation/extenstion operators extends quantities related to subnetworks i to the whole DNN, i.e.,

$$\boldsymbol{E}_i : \boldsymbol{\Theta}_i \to \boldsymbol{\Theta}$$

- The global iterate $\boldsymbol{\theta}^{(k)}$ is updated using the corrections associated with the subnetworks, as in

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \alpha^{(k)} \sum_{i=1}^{N_{sd}} \boldsymbol{E}_i(\boldsymbol{\theta}_i^* - \boldsymbol{R}_i \boldsymbol{\theta}^{(k)}),$$

Training of subnetworks



Additive (parallel):

- For $i = 1, \ldots, N_{sd}$

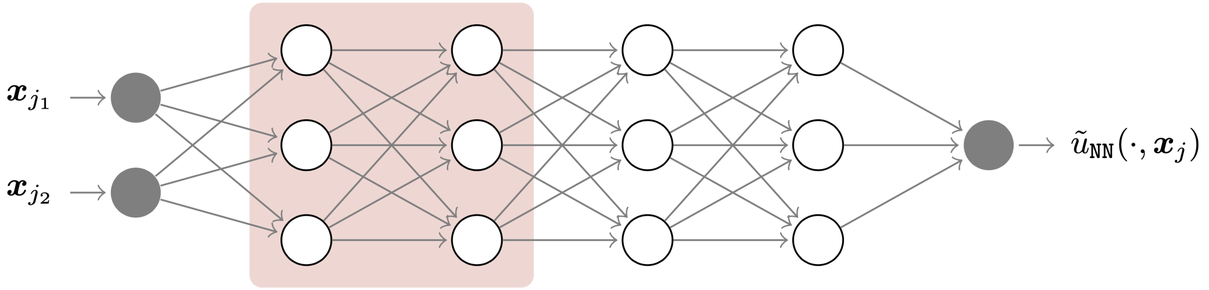
$$\theta_i^* = \mathrm{argmin}_{\theta_i} \mathcal{L}(\theta_1^k, \ldots, \theta_i^k, \ldots, \theta_{N_{sd}}^k)$$

Multiplicative (sequential):

- For $i = 1, \ldots, N_{sd}$

$$\theta_i^* = \mathrm{argmin}_{\theta_i} \mathcal{L}(\theta_1^*, \ldots, \theta_i^k, \ldots, \theta_{N_{sd}}^k)$$

Nonlinear preconditioning of the training



- The global iterate $\boldsymbol{\theta}^{(k)}$ is updated using the corrections associated with the subnetwork, as in

$$\boldsymbol{\theta}^{(k+1)} = \boldsymbol{\theta}^{(k)} + \alpha^{(k)} \sum_{i=1}^{N_{sd}} \boldsymbol{E}_i (\boldsymbol{\theta}_i^* - \boldsymbol{R}_i \boldsymbol{\theta}^{(k)}),$$

- The step size $\alpha^{(k)}$ is computed using a line-search strategy
- G_R: Nonlinear Schwarz method as an optimizer

$$F\big(G_R(\boldsymbol{\theta})\big) = F\Big(\boldsymbol{\theta}^{(k)} + \alpha^{(k)} \sum_{i=1}^{N_{sd}} \boldsymbol{E}_i (\boldsymbol{\theta}_i^* - \boldsymbol{R}_i \boldsymbol{\theta}^{(k)})\Big) = 0,$$

- Now, to solve $F\big(G_R(\boldsymbol{\theta})\big)$, we employ a quasi-Newton method.

Pseudo-algorithm

1. For a given $\boldsymbol{\theta}^{(k)}$, perform local training:
   - training on subnetworks in an *additive* manner (parallel)

$$\text{Find } \boldsymbol{\theta}_s^* = \text{argmin}_{\boldsymbol{\theta}_s} \mathcal{L}(\boldsymbol{\theta}_1^{(k)}, \ldots, \boldsymbol{\theta}_s, \ldots, \boldsymbol{\theta}_S^{(k)})$$

   - training on subnetworks in a *multiplicative* manner (sequential)

$$\text{Find } \boldsymbol{\theta}_s^* = \text{argmin}_{\boldsymbol{\theta}_s} \mathcal{L}(\boldsymbol{\theta}_1^*, \ldots, \boldsymbol{\theta}_{s-1}^*, \boldsymbol{\theta}_s, \boldsymbol{\theta}_{s+1}^{(k)}, \ldots, \boldsymbol{\theta}_S^{(k)})$$

2. Synchronization step $\quad \boldsymbol{\theta}^{(k+\frac{1}{2})} \leftarrow\!\shortmid \boldsymbol{\theta}^{(k)} + \alpha^{(k)} \sum_{i=1}^{N_{sd}} \boldsymbol{E}_i (\boldsymbol{R}_i \boldsymbol{\theta}^{(k)} - \boldsymbol{\theta}_i^*)$
3. Preconditioned quasi-Newton step $\quad \boldsymbol{p}^{(k+\frac{1}{2})} \leftarrow\!\shortmid -(\boldsymbol{B}^{(k+1)})^{-1} \nabla \mathcal{L}(\boldsymbol{\theta}^{(k+\frac{1}{2})})$
4. Momentum update $\quad \boldsymbol{v}^{(k+\frac{1}{2})} \leftarrow\!\shortmid (1-\mu)\boldsymbol{v}^{(k-\frac{1}{2})} + \mu \boldsymbol{p}^{(k+\frac{1}{2})}$
5. Global update using momentum step $\quad \boldsymbol{\theta}^{(k+1)} \leftarrow\!\shortmid \boldsymbol{\theta}^{(k+\frac{1}{2})} + \alpha^{(k+\frac{1}{2})} \boldsymbol{v}^{(k+\frac{1}{2})}$
6. Update $\boldsymbol{S}^{(k)}$ with $\quad \boldsymbol{s}^{(k)} \leftarrow\!\shortmid \boldsymbol{\theta}^{(k+1)} - \boldsymbol{\theta}^{(k+\frac{1}{2})}$
7. Update $\boldsymbol{Y}^{(k)}$ with $\quad \boldsymbol{y}^{(k)} \leftarrow\!\shortmid \nabla \mathcal{L}(\boldsymbol{\theta}^{(k+1)}) - \nabla \mathcal{L}(\boldsymbol{\theta}^{(k+\frac{1}{2})})$

Optimizer setup

State-of-the-art optimizers:

- Adam with fixed learning rate
- L-BFGS with momentum, backtracking line-search (strong Wolfe's conditions), and memory size $m = 3$

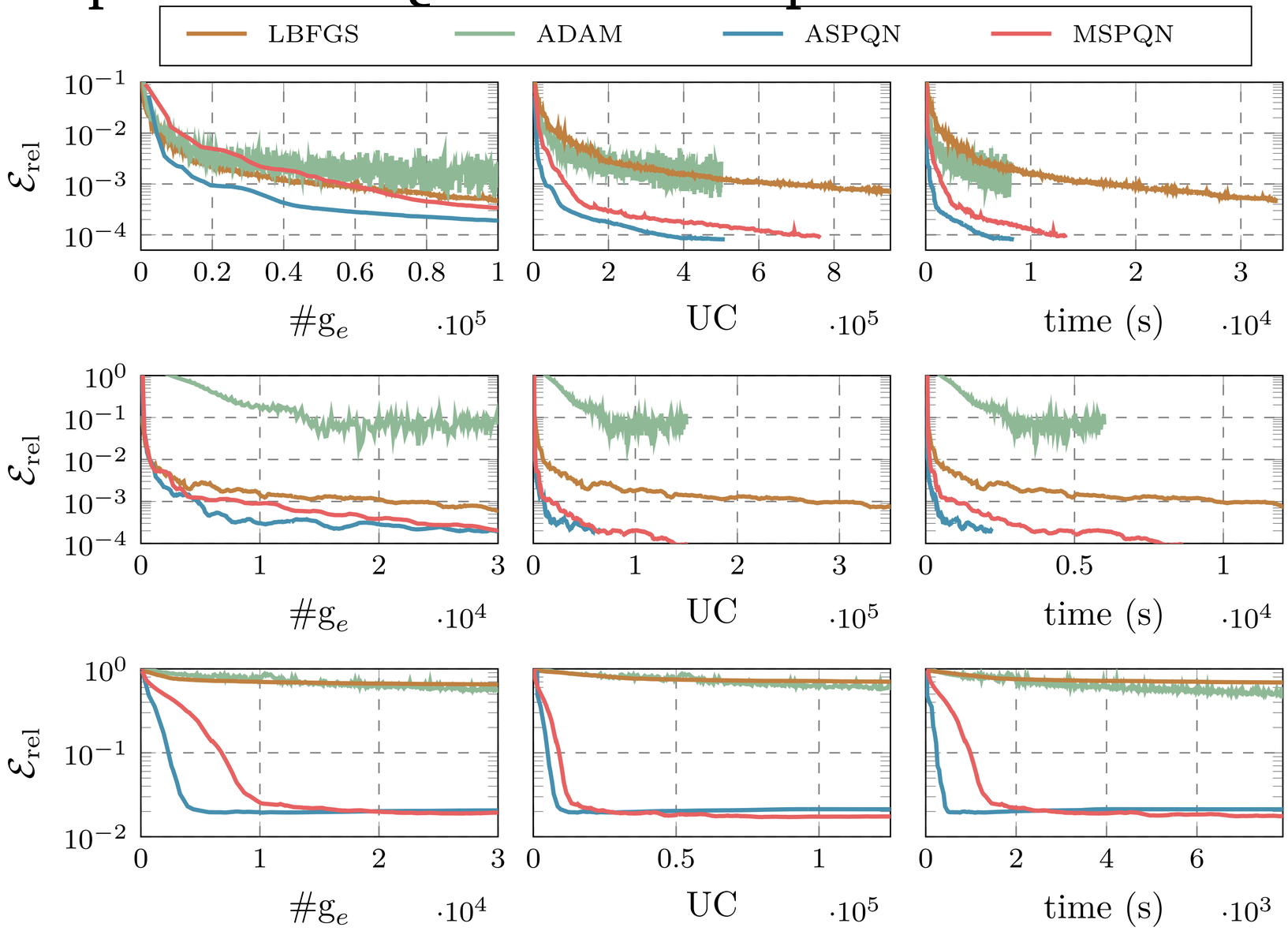
Right preconditioned L-BFGS setup:

- L-BFGS as a local optimizer with k_s steps
- Global step performed by preconditioned L-BFGS with backtracking line-search (strong Wolfe's conditions)
- Varying number of subdomains and memory size, $m = 3$

Implementation

- PyTorch library
- Nvidia's NCCL backend

Comparision SPQN vs. SOTA optimizers



- Burger's equation
- Klein-Gordon equation
- Diffusion-Transport equation

Computational time (SPQN vs. SOTA optimizers)

Example	$\mathcal{E}_{\text{rel}}$ (L-BFGS)	Time to solution (mins)				
		L-BFGS	Adam	ASPQN	(# GPUs)	MSPQN
Burgers'	4.6×10^{-4}	558.5	–	14.4	(8)	40.7
Klein-Gordon	6.1×10^{-4}	236.5	–	6.8	(6)	26.9

Minimum mean $\mathcal{E}_{\text{rel}}$ reached by L-BFGS optimizer and time to solution required to reach that error for all considered optimizers. All optimizers are executed on a single GPU, except ASPQN, which utilizes multiple GPU devices.

Conclusion

- Proposed novel **additive** and **multiplicative** preconditioning strategies for L-BFGS optimizer with a particular focus on PINNs
- Preconditioners constructed leveraging **layer-wise network decomposition**
- Conducted numerical experiments with PINNs benchmark problems to evaluate training methods
- Compared with Adam and L-BFGS optimizers, demonstrating **substantial reduction in training time SPQN methods**
- Achieved average **speedups** of approximately **10 for MSPQN method** and **28 for ASPQN method** compared to L-BFGS
- Demonstrated **significantly more accurate solutions for underlying PDEs** using SPQN methods