

REBUTTAL:

METAMIZER: A VERSATILE NEURAL OPTIMIZER FOR FAST AND ACCURATE PHYSICS SIMULATIONS

Anonymous authors

Paper under double-blind review

1 COMPARISON TO GPU SOLVERS

To make a better comparison, we run Metamizer against various GPU based solvers of the CuPy package on the same NVidia Geforce RTX 4090. Here, Metamizer shows a similar convergence speed for the Laplace equation on a 100×100 grid (see Figure 1) and about 2 times faster convergence on a 400×400 grid (see Figure 2) compared to conjugate gradient and minres.

This is remarkable because Metamizer is a more general optimizer that relies only on local gradients and can be used to solve for example nonlinear PDEs or cloth simulations as well. Conjugate gradient methods on the other hand require full knowledge of the underlying (symmetric) matrix A to orthogonalize update steps and compute appropriate step sizes α .

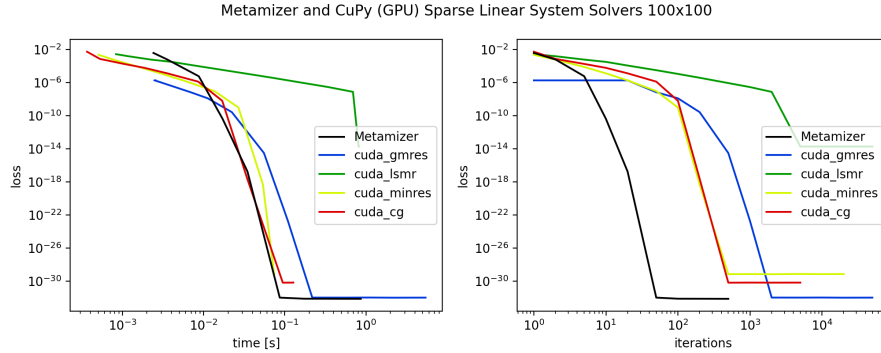


Figure 1: Performance comparison of Metamizer with various GPU based solvers of the CuPy package on a 100×100 grid. The loss corresponds to the mean squared residuals of the Laplace equation.

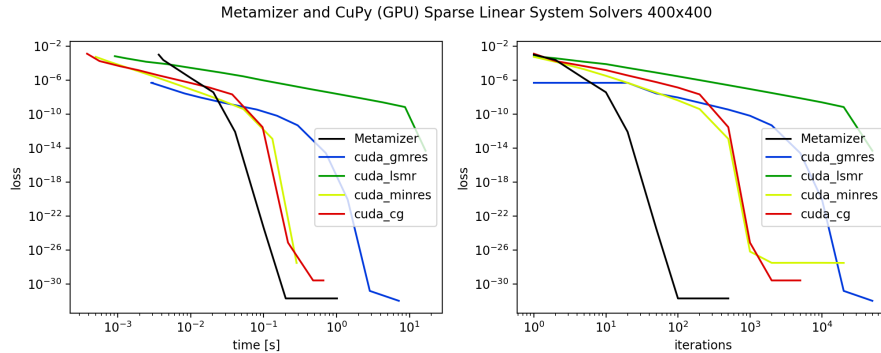


Figure 2: Performance comparison of Metamizer with various GPU based solvers of the CuPy package on a 400×400 grid. Metamizer is about $2\times$ faster than CG on the same GPU.

2 COMPARISON TO CPU SOLVERS

Since our original implementation of the Dirichlet boundary conditions resulted in an asymmetric matrix A , conjugate gradient methods did not converge properly. While Metamizer as well as GMRES and GCROT do not have this limitation, we implemented a symmetric matrix version of A to allow for comparisons with CG. On a 100×100 grid, this led to faster convergence (see Figure 3). However, on a 400×400 grid, GPU parallelization pays off resulting in $10\times$ faster convergence speed of Metamizer (see Figure 4).

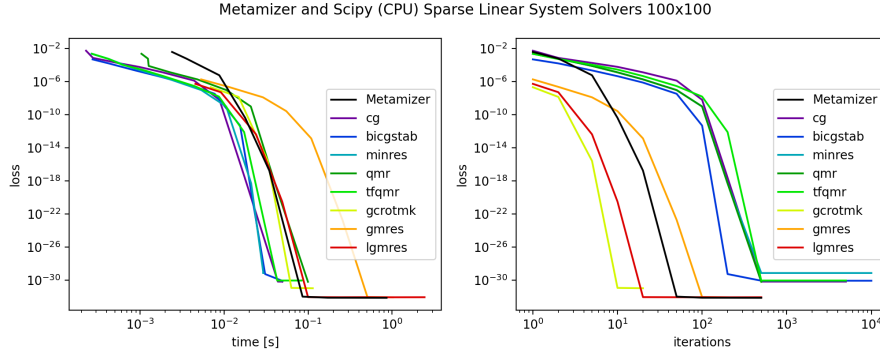


Figure 3: Performance comparison of Metamizer with various CPU based solvers of the SciPy package on a 100×100 grid.

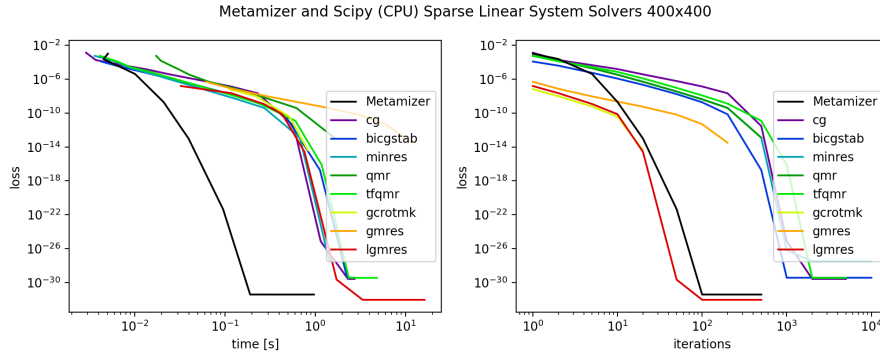


Figure 4: Performance comparison of Metamizer with various CPU based solvers of the SciPy package on a 400×400 grid. Metamizer is about $10\times$ faster than CG on a CPU.

3 COMPARISON TO NEWTON-CG, NONLINEAR CG AND L-BFGS-B

We added further comparisons for optimizers that, like Metamizer, rely only on local gradients of the loss. As can be seen in Figure 5, optimizers like Newton-CG, Nonlinear CG or L-BFGS exhibit much slower convergence.

4 COMPARISON OF DIFFERENT LEARNING RATES

We compared various different learning rates for all gradient descent solvers and only reported results for learning rates that resulted in good trade-offs between convergence speed and accuracy. Figure 6 shows a performance comparison of Metamizer with Adagrad, Adam and AdamW at different learning rates.

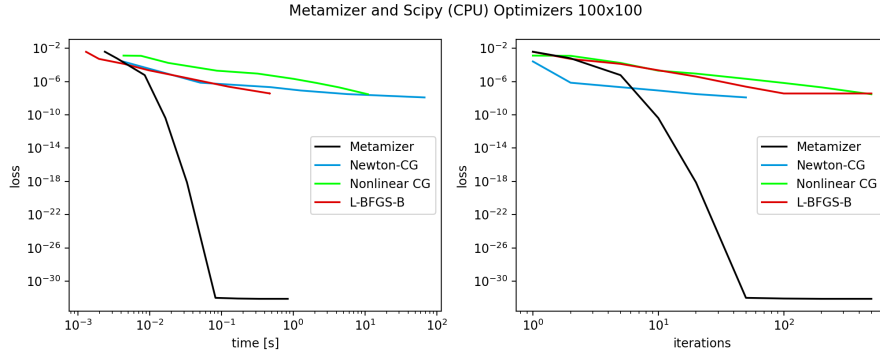


Figure 5: Performance comparison of Metamizer with various optimizers that, like Metamizer, rely only on local gradients of the loss. Here, we make use of the SciPy package on a 100×100 grid on a CPU.

5 COMPARISON TO GROUND TRUTH

We compared the mean squared errors of Metamizer and various GPU based linear system solvers to analytical ground truth values of the Laplace equation. Figure 7 shows fast convergence to highly accurate results as already indicated by the mean residual errors in Figure 2.

6 TURBULENT FLOW SIMULATION AT $Re = 2000$

We trained Metamizer on a 400×400 grid to enable fluid simulations at higher Reynolds numbers. Figure 8 demonstrates that Metamizer is able to simulate complex turbulent fluid dynamics at $Re = 2000$.

7 PRECONDITIONERS

7.1 INCOMPLETE LU (ILU) AND ALGEBRAIC MULTI GRID (AMG) PRECONDITIONERS

We compared Metamizer with preconditioned conjugated gradient methods based on incomplete LU factorization and algebraic multigrids. To this end, we used the pyAMG library. AMG preconditioning resulted in a significant speed-up over non-preconditioned solvers on the CPU (see Figure 9). A GPU based algebraic multigrid implementation will most likely outperform Metamizer on the Poisson equation, however, Metamizer is a more general approach that only requires local gradients and can be applied for example to non-symmetric Matrices, nonlinear PDEs or cloth simulations as well.

7.2 JACOBI PRECONDITIONER

We tested the diagonal Jacobi preconditioner for CG, GMRES and MINRES. Since the diagonal of the Laplace operator on a regular grid corresponds closely to the identity matrix, this preconditioner did not help to significantly reduce the number of iterations but resulted in a slight computational overhead and thus a slowdown of convergence (see Figure 10).

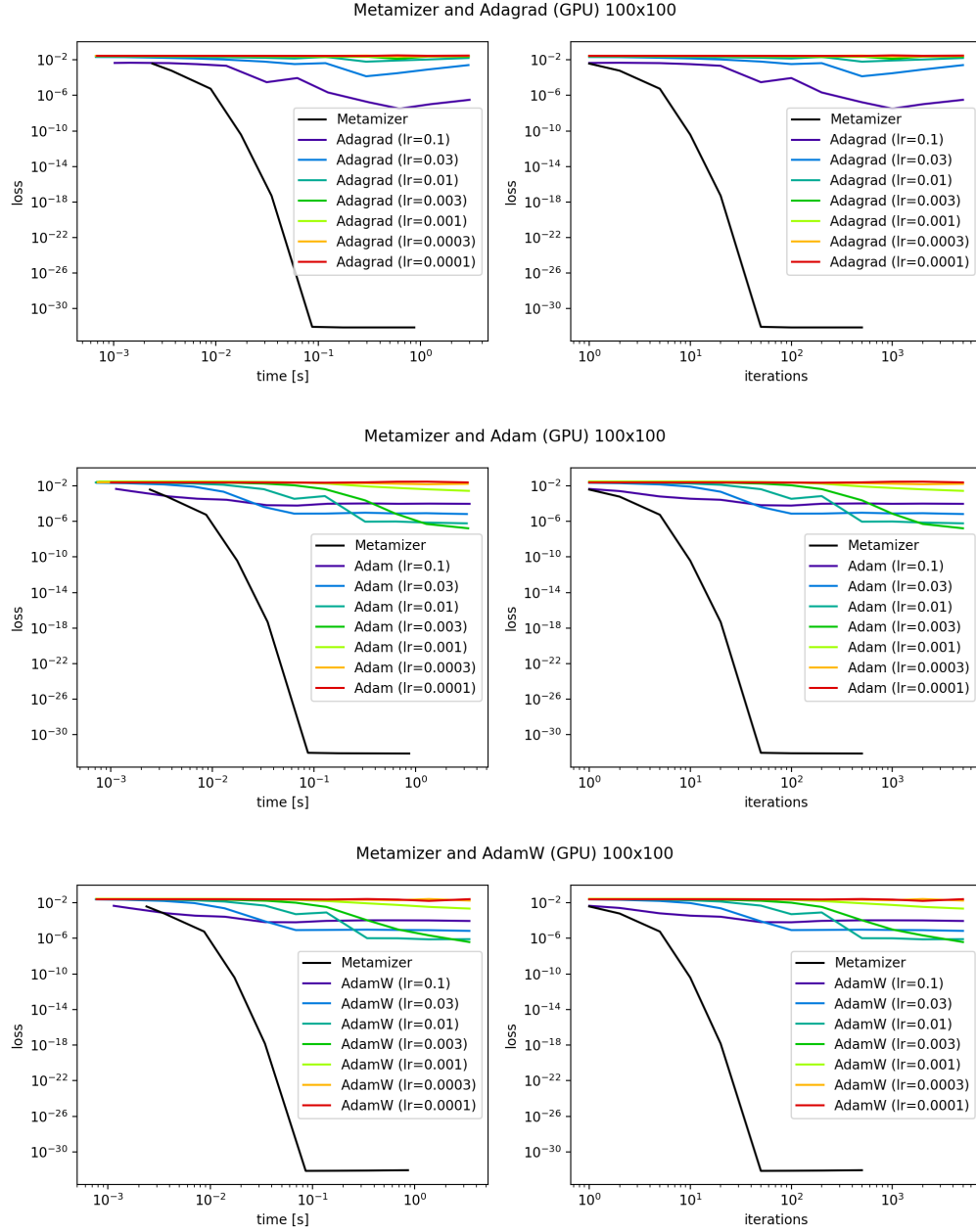


Figure 6: Performance comparison of Metamizer with various GPU based Gradient Descent Methods at different learning rates on a 100×100 grid.

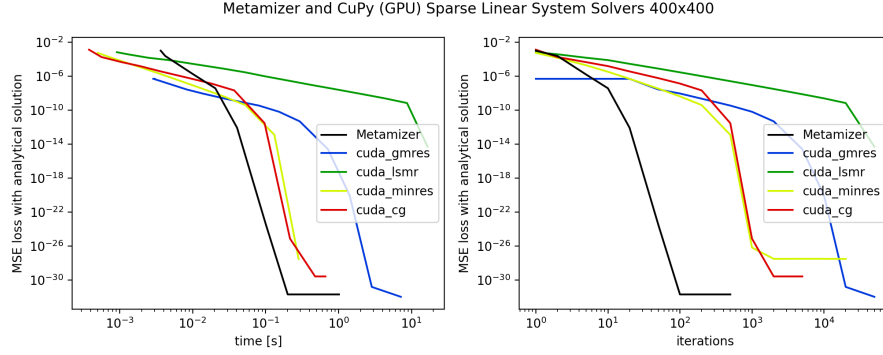


Figure 7: Mean squared errors of Metamizer and various GPU based solvers of the CuPy package on a 400×400 grid compared to analytical ground truth values of the Laplace equation.

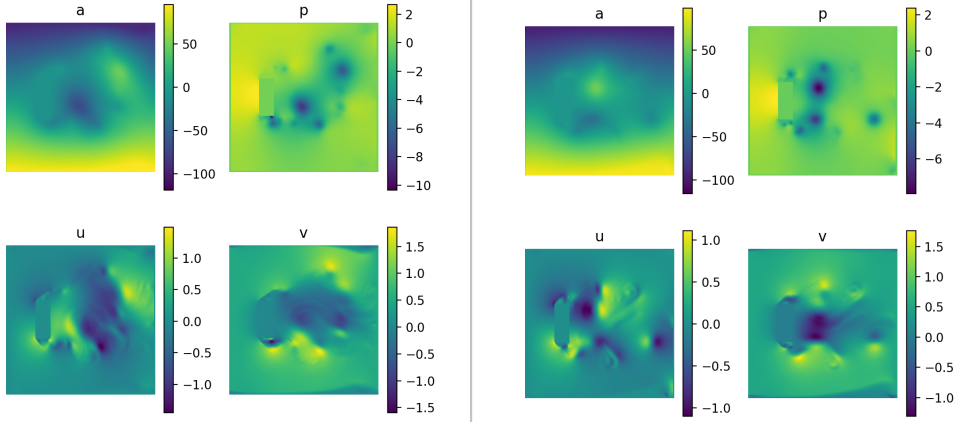


Figure 8: Simulation of turbulent fluid dynamics at $Re = 2000$ ($\mu = 0.1, \rho = 4, D = 100, ||\vec{v}|| = 0.5$) performed by Metamizer on a 400×400 grid.

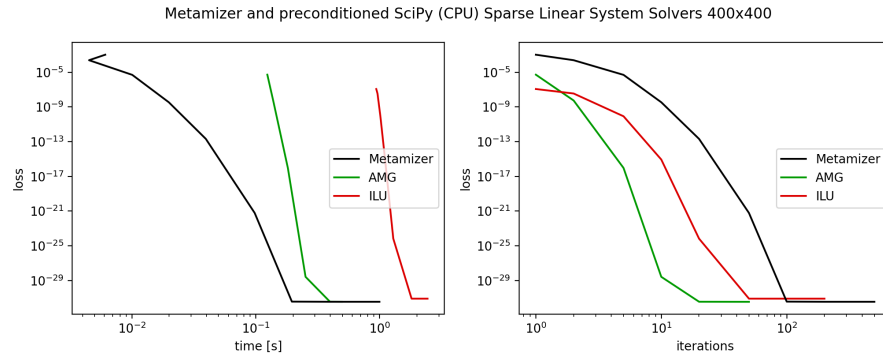


Figure 9: Performance comparison of Metamizer with conjugated gradients preconditioned on the incomplete LU factorization and the algebraic multigrid method (pyAMG) on a 400×400 grid.

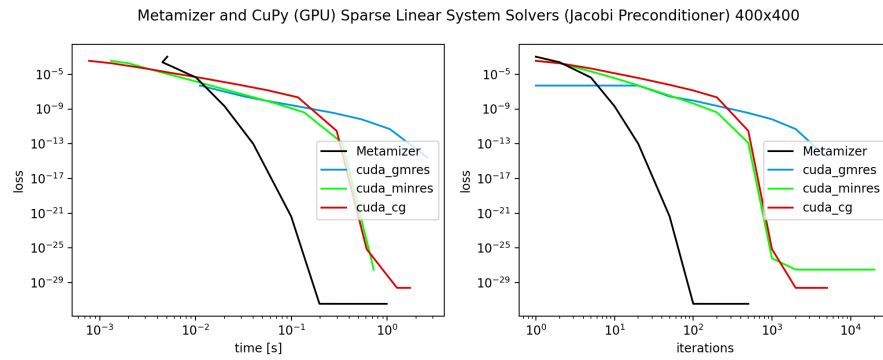


Figure 10: Performance comparison of Metamizer with various GPU based solvers of the CuPy package that make use of a Jacobi Preconditioner on a 400x400 grid. In comparison to non-preconditioned solvers (Figure 2) this resulted in a slight slowdown of convergence.