Massively parallel inverse modeling on GPUs using the adjoint method

Ivan Utkin[12], Ludovic Rass[12]

[1]VAW, D-BAUG, ETH Zürich, Zürich, Switzerland
[2]WSL, Birmensdorf, Switzerland

Solid Earth and Geohazards in the Exascale Era
24th of May, 2023, Barcelona, Spain

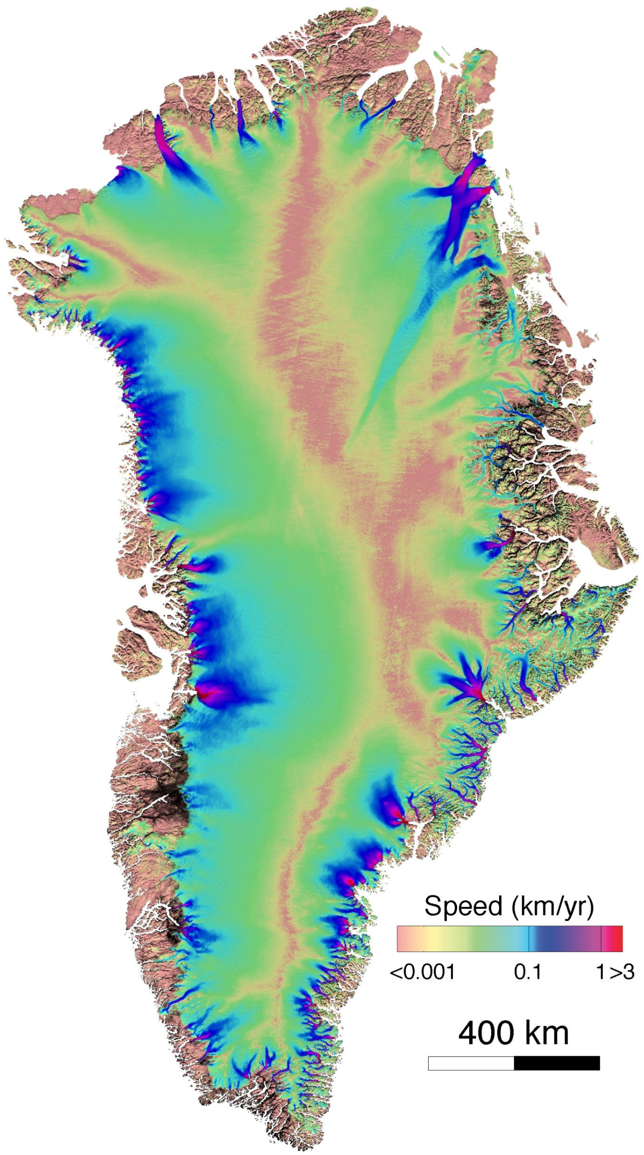
Motivation

- Large-scale computational models need to be calibrated against observational data
- Curse of dimensionality complicates the use of the standard stochastic methods like MCMC
- World’s most powerful supercomputers are GPU-accelerated, new scalable algorithms are needed to fully utilize the hardware

Motivation (#2)

- We develop a massively parallel ice flow model *FastIce*
- The goal of the project is to run the simulation of the ice flow over Greenland at 10m resolution
- We were awarded with 80 mio core-hours on LUMI, the fastest supercomputer in Europe (#3 in Top500)
- Uncertainty quantification is an important objective in computational glaciology and one of the goals in our project

Project website: https://ptsolvers.github.io/GPU4GEO/

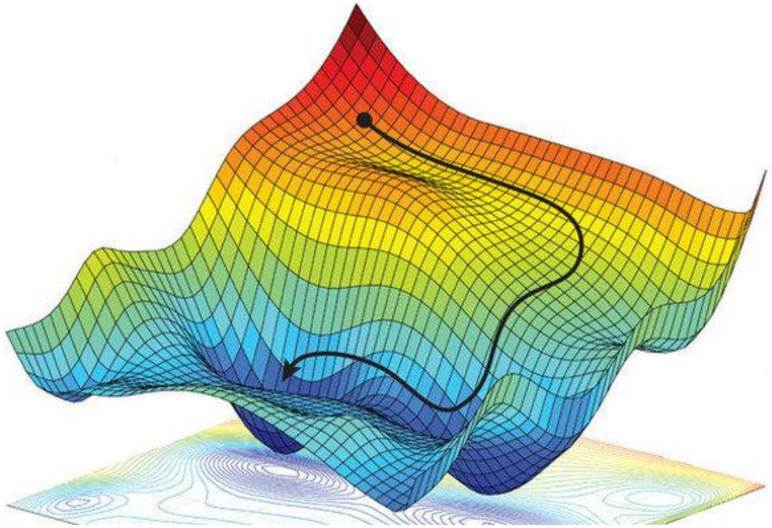


Adjoint method and inverse modelling

- Inverse problem: given a model $R(u) = 0$, find parameters λ minimizing the objective function $J = \int \|u - u_{obs}\|_2 \, \mathrm{d}x$
- Gradient-based approach:

$$\lambda^{n+1} = \lambda^n - \gamma \frac{\mathrm{d}J}{\mathrm{d}\lambda^n}$$

- Adjoint method:



$$\frac{\mathrm{d}J}{\mathrm{d}\lambda} = \underbrace{\frac{\partial J}{\partial u}}_{1\times N} \cdot \underbrace{\frac{\mathrm{d}u}{\mathrm{d}\lambda}}_{N\times M} = -\underbrace{\frac{\partial J}{\partial u}}_{1\times N} \cdot \underbrace{\left[\frac{\partial R}{\partial u}\right]^{-1}}_{N\times N} \cdot \underbrace{\frac{\partial R}{\partial \lambda}}_{N\times M}$$

Solve in 2 steps: $\left[\frac{\partial R}{\partial u}\right]^{\mathrm{T}} \Psi = \frac{\partial J}{\partial u}, \qquad \frac{dJ}{d\lambda} = -\Psi^T \frac{\partial R}{\partial \lambda}$

Can use automatic differentiation to solve the adjoint problem

Julia language

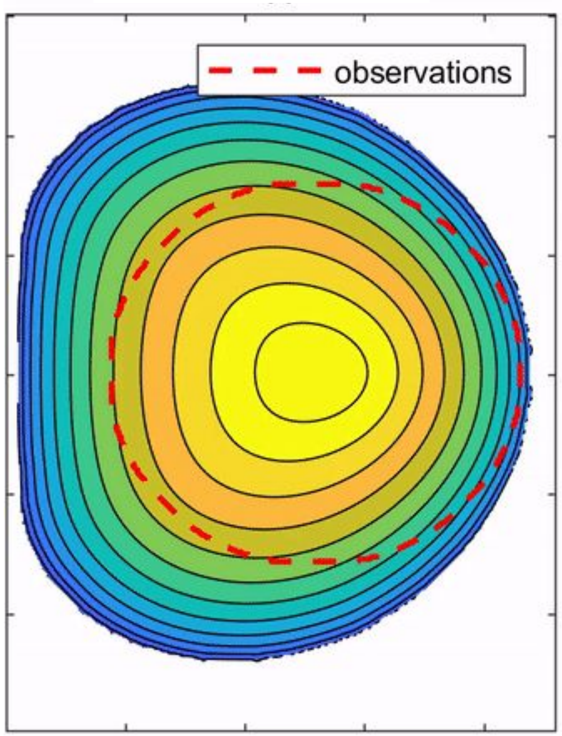
- Julia is a “fresh approach to scientific computing”, which solves the “two-language problem”
- Julia is a dynamically typed high-level language that runs just as fast as Fortran or C
- Has first-class support of GPU programming (Nvidia and AMDGPU)
- Includes capabilities for the differentiable programming on GPUs via Enzyme.jl
- Has a growing and friendly community

```
julia> using Enzyme

julia> f(ω,x) = sin(ω*x)
f (generic function with 1 method)

julia> ∇f(ω,x) = Enzyme.autodiff(Reverse,f,Active,Const(ω),Active(x))[1][2]
∇f (generic function with 1 method)

julia> @assert ∇f(π,1.0) ≈ π*cos(π)
```

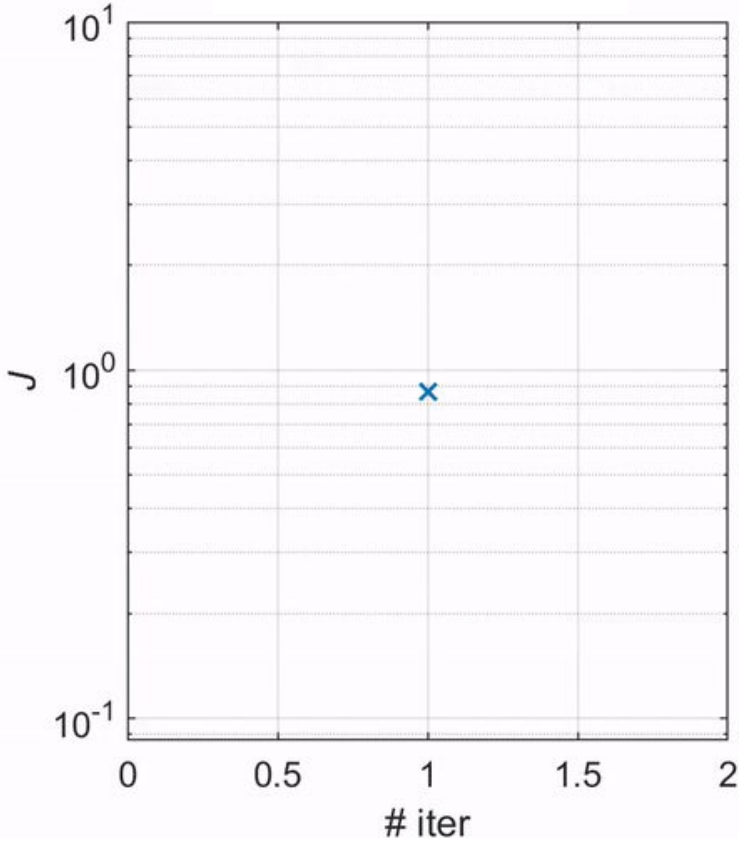
Gradient-based optimization

- We invert for the climate conditions to match tshe shape and volume of a glacier in a synthetic model setup
- We use a PDE-based depth-integrated forward model:
$$\nabla \cdot [D(H)\nabla S] = Q(S)$$
- We solve both forward and adjoint problems using a fixed-point iterative “pseudo-transient” method:
$$-\frac{\partial S}{\partial \tau} + \nabla \cdot [D(H)\nabla S] = Q(S)$$

Ice surface



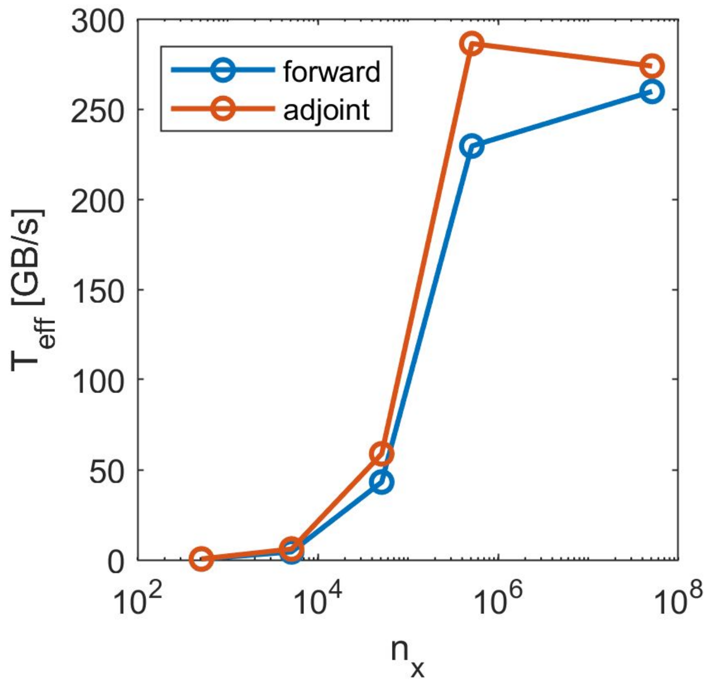
Loss function



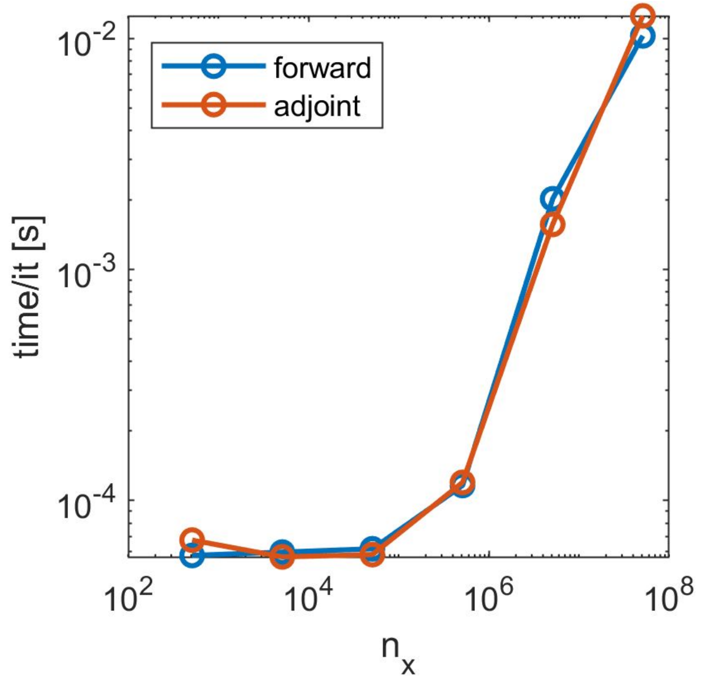
Performance

- The forward and inverse algorithm are memory-bound, we use the effective memory throughput for benchmarking performance
- The performance of the Enzyme-generated adjoint solve is similar to that of the forward solve
- The adjoint problem is linear, and usually takes only a fraction of iterations required for the nonlinear forward solve

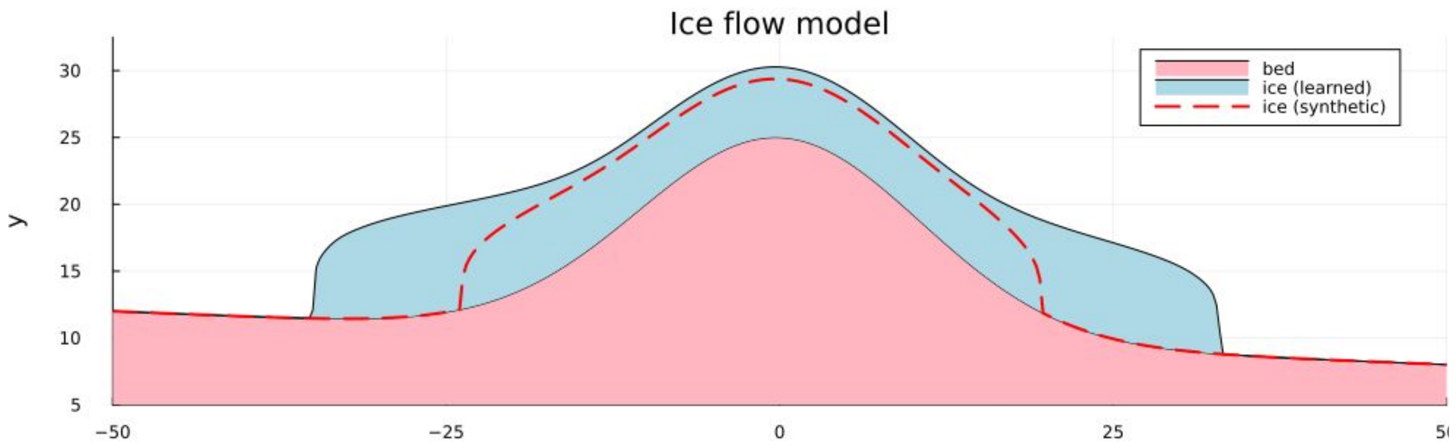
Memory throughput



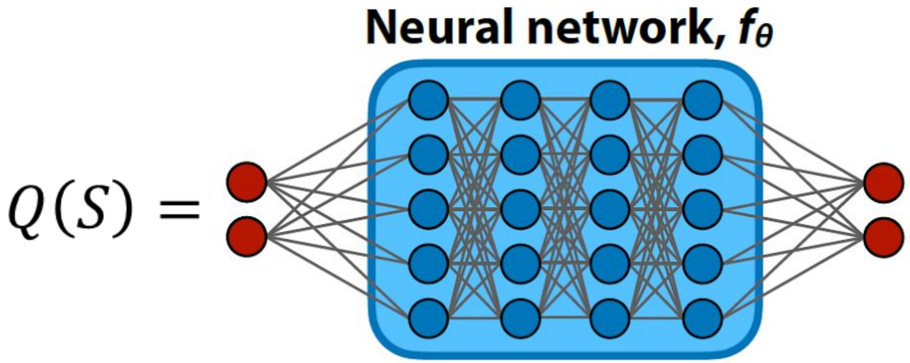
Time per iteration



Coupling physics and ML



- Differentiable programming allows combining physics-based approaches and data driven models, such as neural networks
- Here, we train the neural network-based climate forcing model to reproduce the shape and the volume of the glacier



Thank you!