ECP Julia Tutorial

A differentiable PDE solver example using the Burgers equation

DJ4Earth https://dj4earth.github.io/

CliMA CLIMATE MODELING ALLIANCE https://clima.caltech.edu/

ECP EXASCALE COMPUTING PROJECT

NNSA National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

DJ4Earth

- Differentiable programming in Julia for Earth system modeling

https://dj4earth.github.io/

- Automatic differentiation applied to CliMA https://clima.caltech.edu/
- NSF CSSI "Collaborative Research - Frameworks - Convergence of Bayesian inverse methods and scientific machine learning in Earth system models through universal differentiable programming“
- Checkpointing.jl

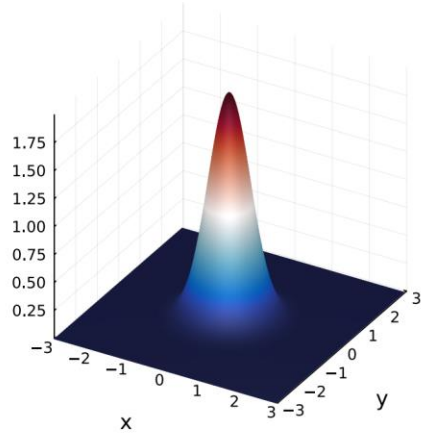
DiffDistPDE.jl

Burgers' equation

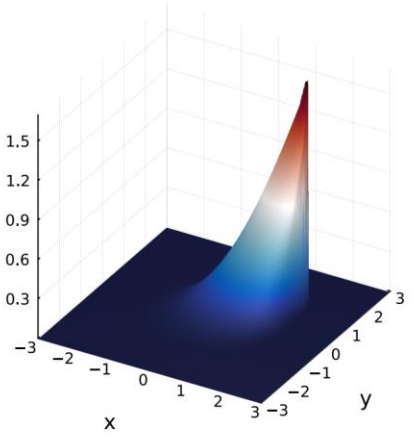
$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

Nonlinear time-dependent PDE

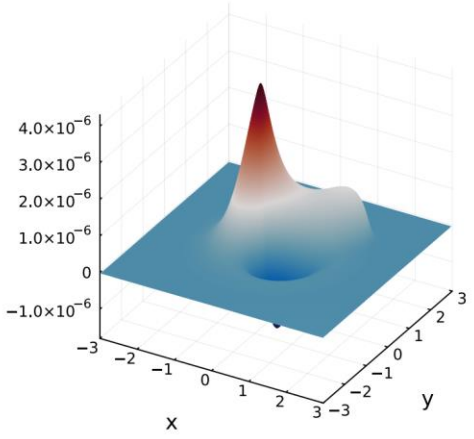
- Differentiable
- Distributed (MPI)
- Extensible beyond Burgers (stencil…



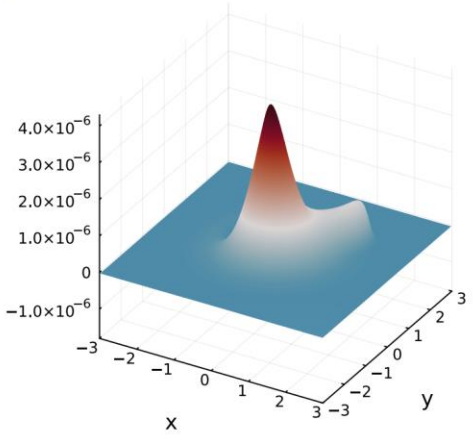
(a) Initial velocity magnitude with $v_0 = v(0, x, y)$



(b) Final velocity magnitude with $v(t_f, x, y)$



(c) Sensitivity $\bar{u}_0$ of the final energy with respect to the initial velocity u_0



(d) Sensitivity $\bar{v}_0$ of the final energy with respect to the initial velocity v_0

DiffDistPDE.jl

- User API
  - Stencil computation **`stencil!`**,
  - Set boundary conditions **`set_boundary_conditions!`**
  - Set initial conditions **`set_initial_conditions!`** for a given PDE
- DiffDistPDE.jl provides
  - The data partitioning
  - The halo exchange
  - Wiring for enabling automatic differentiation
- Done in 500 lines of code

DiffDistPDE.jl

APIs in Julia

- Specialization on types

```
abstract type AbstractPDE end
struct DistPDE{PT <: AbstractPDE}
    nextu::Matrix{Float64}
    nextv::Matrix{Float64}
    lastu::Matrix{Float64}
    lastv::Matrix{Float64}
    ...
end
```

```
struct Burgers <: AbstractPDE end
```

- Add methods to functions for specialized types

```
"""
    halo!(pde::DistPDE{AbstractPDE})

Exchange halo regions between neighboring processes.
"""
function halo!(pde::DistPDE{AbstractPDE})
    error("halo! not implemented for $(typeof(pde))")
end

"""
    stencil!(pde::DistPDE{AbstractPDE})

Apply the stencil to the PDE.
"""
function stencil!(pde::DistPDE{AbstractPDE})
    error("stencil! not implemented for $(typeof(pde))")
end
```

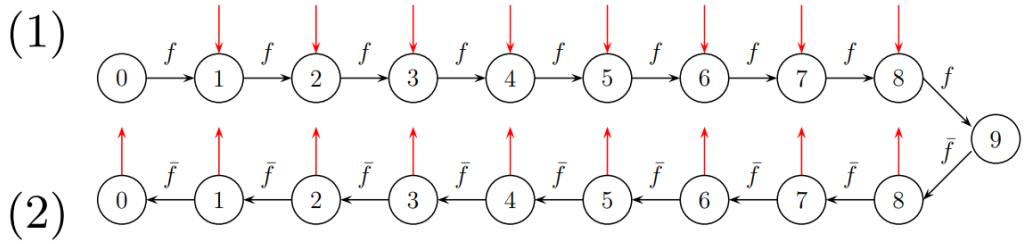
DiffDistPDE.jl

- Final energy for the velocity fields

```
function final_energy(
    pde::DistPDE{PT},
) where {PT}
    for i in 1:pde.tsteps
        advance!(pde)
        halo!(pde)
        copyto!(pde.lastu, pde.nextu)
        copyto!(pde.lastv, pde.nextv)
    end
    return energy(pde)
end
```

Differentiation

$$u_{t+1} = f(u_t)$$
$$\bar{u}_t = \bar{f}(u_t, \bar{u}_{t+1}) = \frac{\partial f(u_t)}{\partial u_t}\bar{u}_{t+1}$$



$$E = \frac{1}{2}u^2$$
Final energy

- Compute sensitivity $\frac{\partial E}{\partial u_0}$ of final energy E with respect to initial condition u_0
- Done like backprogation in machine learning, except in pure Julia. No domain specific language!
  - Enzyme.jl does the differentiation of the timestep,
  - Zygote.jl differentiates outer loop (computation of the energy E, and
  - Checkpointing.jl takes care of storing intermediate states u

Differentiation

```
burgers = DistPDE{Burgers}(Nx, Ny, μ, dx, dy, dt, tsteps)
set_boundary_conditions!(burgers)
set_initial_conditions!(burgers)
revolve = Revolve{DistPDE{Burgers}}(tsteps, snaps; verbose=1, storage=storage)

@time begin
    set_boundary_conditions!(burgers)
    set_initial_conditions!(burgers)
    Checkpointing.reset(revolve)
    dburgers = Zygote.gradient(final_energy, burgers, revolve)
end

dvel = (
    dburgers[1].lastu.^2 +
    dburgers[1].lastv.^2
)
```

DiffDistPDE.jl

- Play with the code: https://github.com/DJ4Earth/Burgers.jl

```
michel@sp8 ~/git
$ git clone git@github.com:DJ4Earth/Burgers.jl.git
Cloning into 'Burgers.jl'...
remote: Enumerating objects: 173, done.
remote: Counting objects: 100% (173/173), done.
remote: Compressing objects: 100% (119/119), done.
Receiving objects: 100% (173/173), 35.23 KiB | 8.81 MiB/s, done.
Resolving deltas: 100% (86/86), done.
remote: Total 173 (delta 86), reused 112 (delta 46), pack-reused 0
michel@sp8 ~/git
$ cd Burgers.jl
michel@sp8 ~/git/Burgers.jl <main>
$ julia --project
               _
   _       _ _(_)_     |  Documentation: https://docs.julialang.org
  (_)     | (_) (_)    |
   _ _   _| |_  __ _   |  Type "?" for help, "]?" for Pkg help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 1.8.5 (2023-01-08)
 _/ |\__'_|_|_|\__'_|  |  Official https://julialang.org/ release
|__/                   |

(DiffDistPDE) pkg> up
```