Industrial partners:






University partners:


National
Technical
University of
Athens

# Unsteady discrete adjoint with source-transformed OpenMP 4.0

Jan Hückelheim*
Jens-Dominik Müller

Queen Mary, University of London
School of Engineering and Material Science

June 04, 2014

# Background

- OpenMP: language to parallelise code
- New version released 2013 with major new features
  - handing work to coprocessors such as GPU or Intel XeonPhi (MIC): special purpose devices that can do some tasks faster or more energy efficient
  - vectorisation (SIMD): most processors can perform the same operation at once on a whole vector of numbers

# Our goal

1. Parallelise primal with latest OMP features $\rightarrow$ show results
2. (Automatically?) parallelise adjoint[1] $\rightarrow$ show problems
3. New in this work: Differentiation of SIMD and coprocessor features in OMP

[1] M. Förster, U. Naumann, J. Utke: Toward Adjoint OpenMP, RWTH 2011

# Case study: Burgers equation

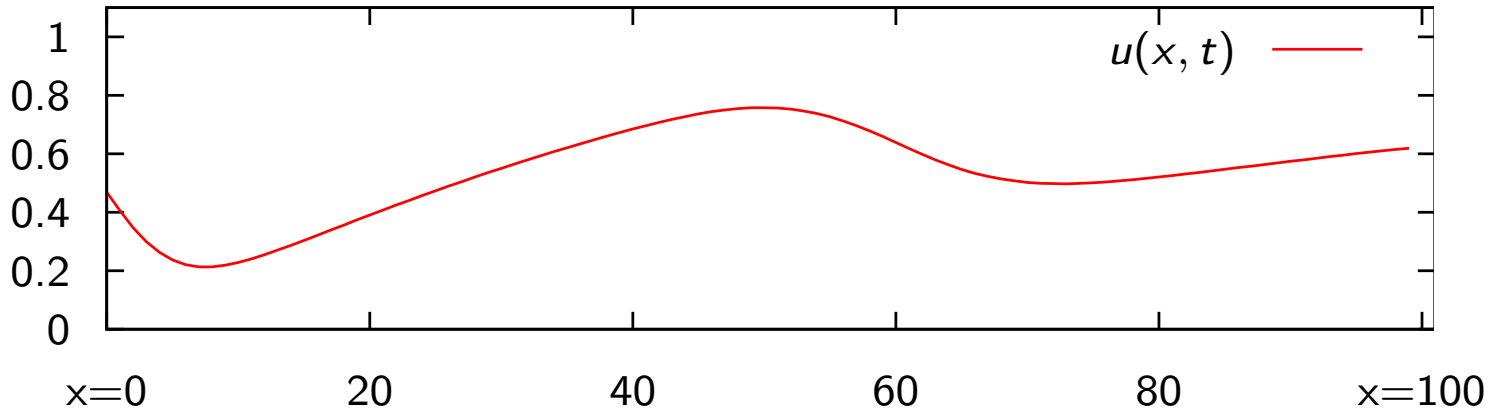
$$\frac{\partial}{\partial t}u + u \cdot \frac{\partial}{\partial x}u = \alpha \frac{\partial^2}{\partial x^2}u$$

- Scalar, 1D nonlinear 2nd order partial differential equation
- Implemented finite difference BDF2 dual timestepping code with explicit Euler inner iterations
- Differentiation with Tapenade [2]

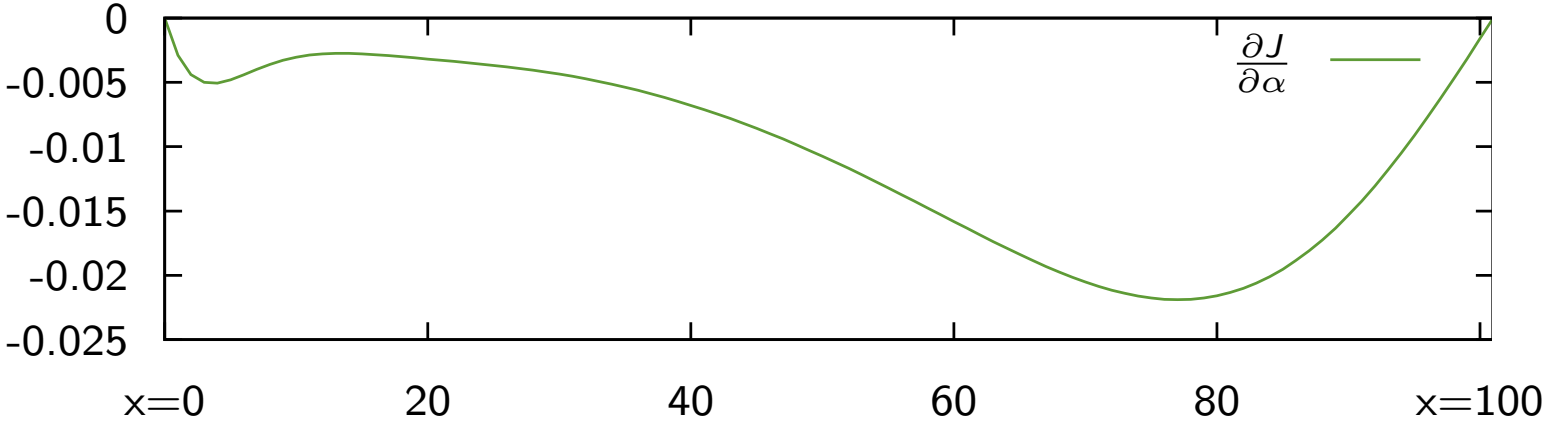
[2] L. Hascoët, V. Pascual: The Tapenade automatic differentiation tool: Principles, model, and specification, ACM Transactions on Mathematical Software Vol. 39 Issue 3, 2013

# Burgers equation

$u(0,t) \leftarrow \frac{1}{2} + \frac{1}{2} \cdot sin(t)$ $\quad u_t + u_x = -\alpha \cdot u_{xx}$ $\quad u(101,t) \leftarrow u(100,t)$




Sensitivity of cost function $J = \int_t u(100,t)^2 \mathrm{d}t$



# OpenMP example

**while** $i{=}1 \ldots n$ **do**
  $res_i \leftarrow \mathsf{calcres}(y_{i-1}, y_i, y_{i+1}, \alpha_i)$;
**end**

# OpenMP example

!$omp parallel do shared(res,y,$\alpha$) private(i); `// < 16x speedup`

**while** $i{=}1 \ldots n$ **do**
$\quad res_i \leftarrow \text{calcres}(y_{i-1}, y_i, y_{i+1}, \alpha_i)$;
**end**

# OpenMP example

!$omp parallel do shared(res,y,$\alpha$) private(i); `// < 16x speedup`
!$omp simd linear(i,1); `// < 4x speedup`
**while** $i{=}1 \ldots n$ **do**
  $res_i \leftarrow \text{calcres}(y_{i-1}, y_i, y_{i+1}, \alpha_i)$;
**end**

# OpenMP example

```
!$omp target map(to:alpha,y) map(from:res); // < 6x speedup
!$omp parallel do shared(res,y,α) private(i); // < 16x speedup
!$omp simd linear(i,1); // < 4x speedup
```

**while** $i{=}1 \ldots n$ **do**

$\quad res_i \leftarrow \text{calcres}(y_{i-1}, y_i, y_{i+1}, \alpha_i)$;

**end**

```
!$omp end target;
```

# From primal to reverse...

- Primal was easy
- Practical problems in automatic differentiation: Tapenade sees OpenMP directives as comments, dumps them in wrong places, resulting in wrong code. Tapenade library is not thread-safe
- Could Tapenade automatically produce good parallel adjoint code? Some problems on the next slides

# OpenMP reverse example

**while** $i{=}n\ldots1$ **do**

$[\bar{y}_{i-1}, \bar{y}_i, \bar{y}_{i+1}, \bar{\alpha}_i] \leftarrow$ calcres_b$(y_i, res_i)$;

**end**

# OpenMP reverse example

!$omp parallel do shared($\bar{res}$,$y$,$\bar{y}$,$\bar{\alpha}$) private(i);

**while** $i{=}n...1$ **do**
  !$omp atomic; // avoid conflicting writes to $\bar{y}$
  $[\bar{y}_{i-1}, \bar{y}_i, \bar{y}_{i+1}, \bar{\alpha}_i] \leftarrow$ calcres_b($y_i$, $res_i$);
**end**

- Atomic update: prevents concurrent access to the same entry. This is expensive.
- For good performance, custom treatment necessary

# OpenMP reverse example

!$omp parallel do shared($\bar{res}$,$y$,$\bar{y}$,$\bar{\alpha}$) private(i);
!$omp simd linear(i,1); `// negative stride forbidden`
**while** $i{=}1...n$ **do**
    !$omp atomic; `// avoid conflicting writes to` $\bar{y}$
    $[\bar{y}_{i-1}, \bar{y}_i, \bar{y}_{i+1}, \bar{\alpha}_i] \leftarrow$ calcres_b($y_i$, $res_i$);
**end**

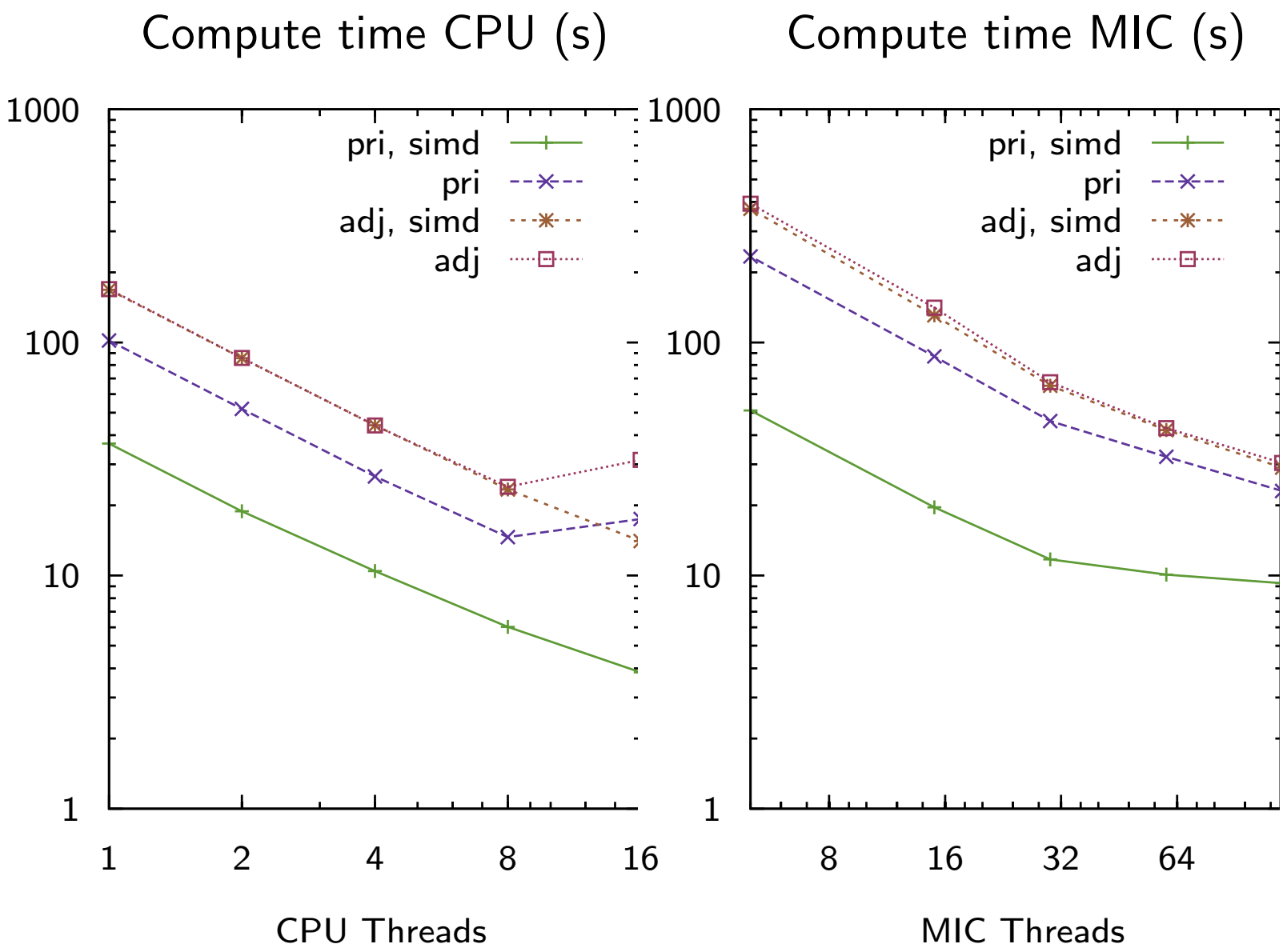
- Negative strides are not allowed in OpenMP SIMD
- Order of iterations can sometimes be chosen, but not always

# OpenMP reverse example

!$omp target map(to:res,y) map(from:$\bar{y}$,$\bar{\alpha}$);
!$omp parallel do shared($\bar{res}$,$y$,$\bar{y}$,$\bar{\alpha}$) private(i);
!$omp simd linear(i,1); `// negative stride forbidden`
**while** $i$=1...$n$ **do**
  !$omp atomic; `// avoid conflicting writes to` $\bar{y}$
  $[\bar{y}_{i-1}, \bar{y}_i, \bar{y}_{i+1}, \bar{\alpha}_i] \leftarrow$ calcres_b($y_i$, $res_i$);
**end**
!$omp end target;

- Transfer pattern changes.

# Speedup


Compute time CPU (s)
1000
100
10
1
pri, simd
pri
adj, simd
adj
1
2
4
8
16
CPU Threads
Compute time MIC (s)
1000
100
10
1
pri, simd
pri
adj, simd
adj
8
16
32
64
MIC Threads

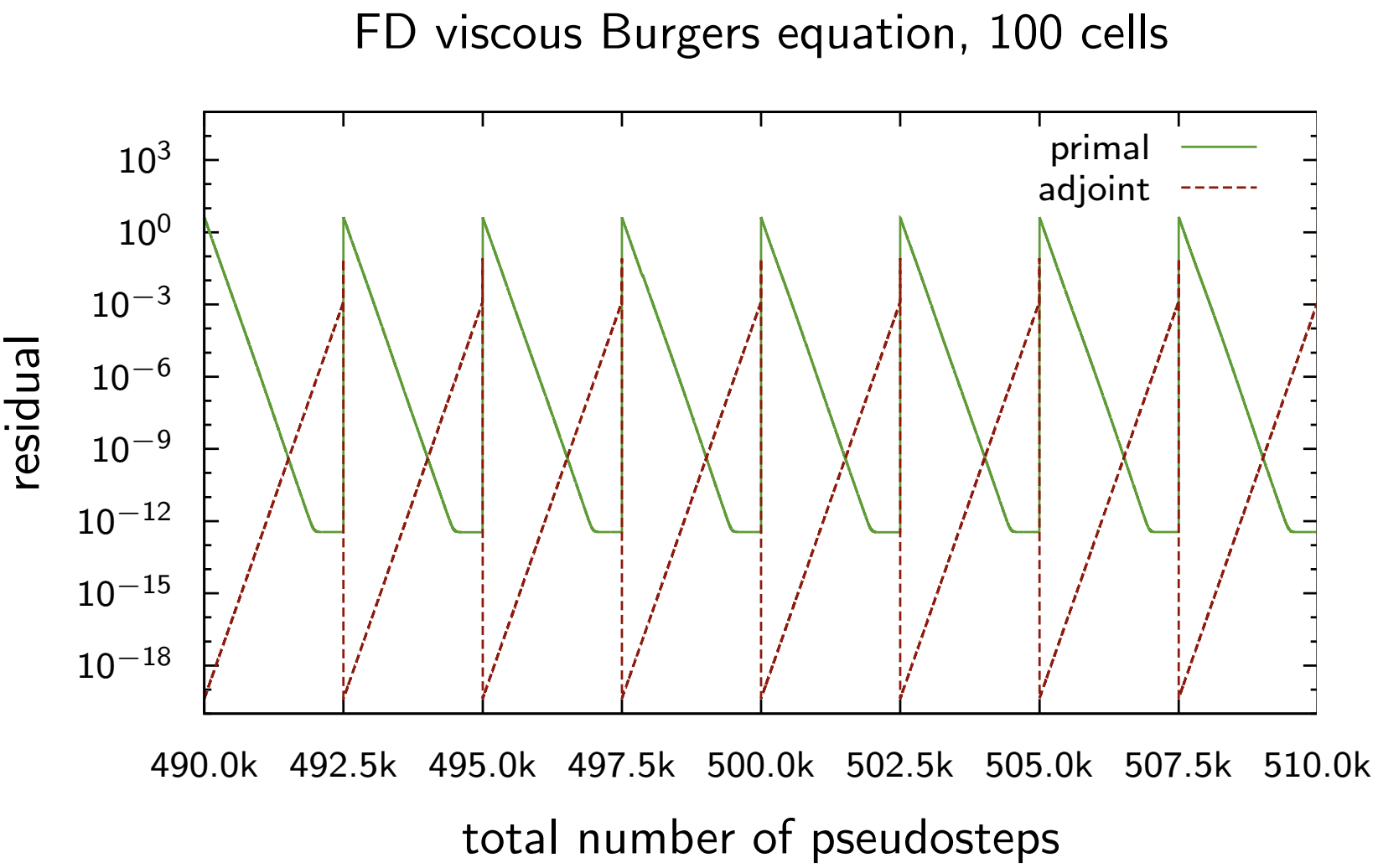
# Conclusion

- Differentiation of OpenMP is not trivial, problems are similar to parallel MPI
- Automatically guaranteeing correctness can be possible, but good performance will remain a manual job for the nearer future
- Need to port this to our 3D unstructured solver, expect more (bigger) challenges

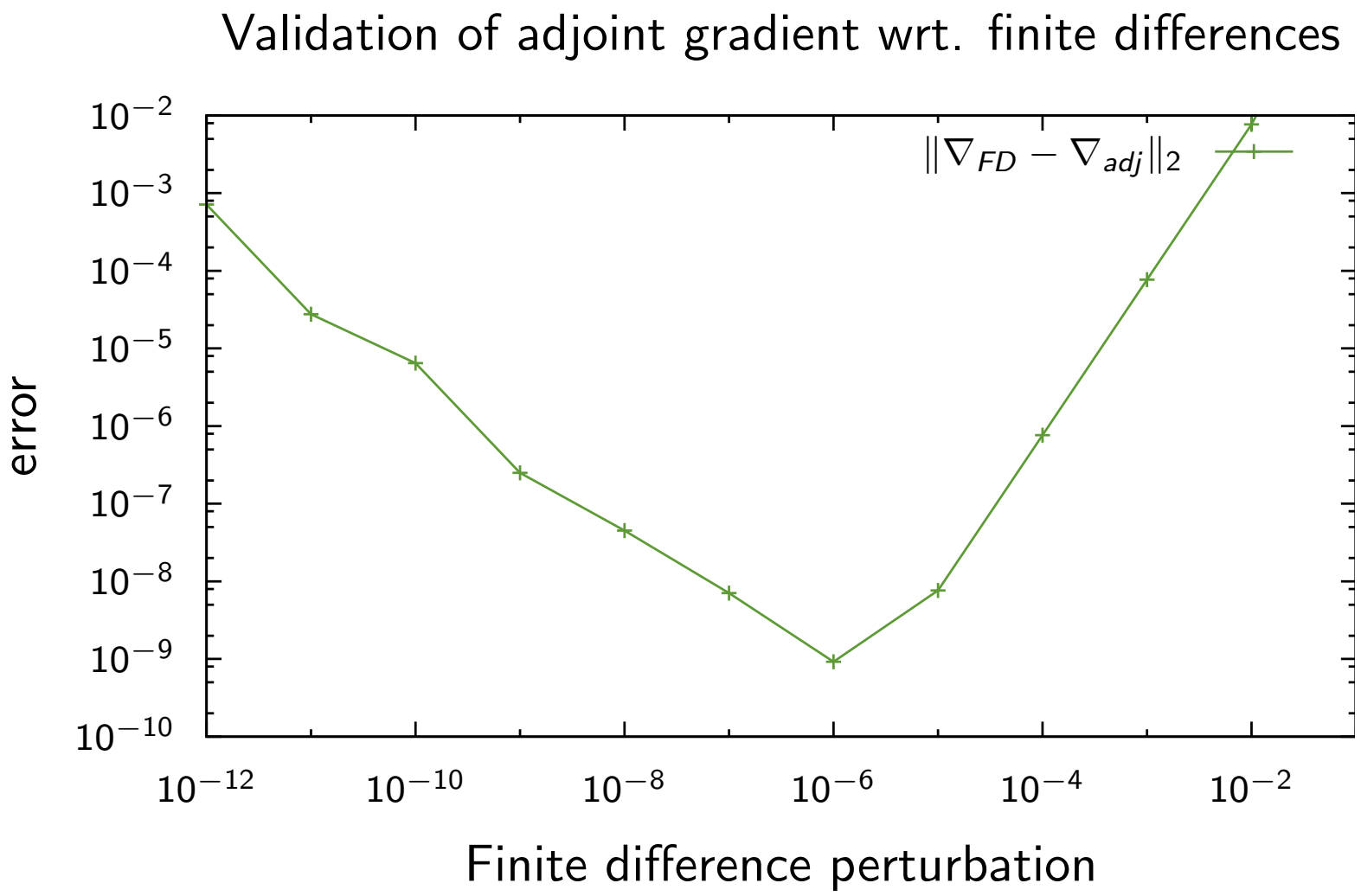
# Conclusion

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no [317006]

# Convergence, brute force



# Validation of gradients



# How to manage memory?

- Memory is always a problem in unsteady adjoint
- Coprocessors have less memory than hosts (up to 16GB, we have only 8GB)
- Revolve checkpointing[3] on coprocessor
- Store on host memory
- (Store on hard drive)

[3] A. Griewank, A. Walther: Algorithm 799: Revolve: An Implementation of Checkpointing for the Reverse or Adjoint Mode of Computational Differentiation, Technical University Dresden

# Cost of sending data back to Host

- In our case: 30 pseudo steps cost as much as transferring a checkpoint
- For implicit solvers, there are fewer steps which each cost more, so less transfer overhead
- We store only fully converged physical time snapshots