Automatic differentiation in Chapel

LUCA FERRANTI

AALTO UNIVERSITY, FINLAND

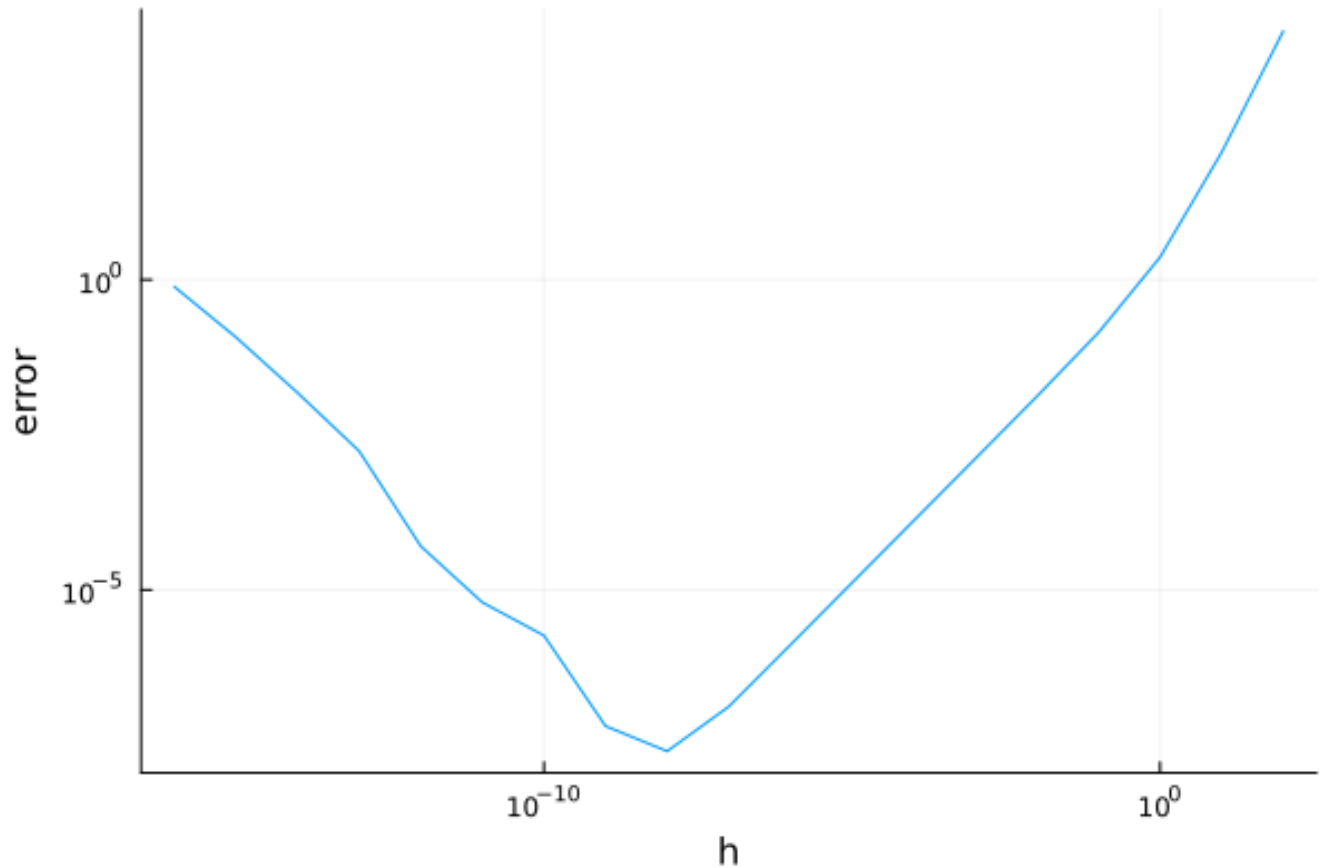
Agenda

- Brief motivation to automatic differentiation
- ForwardModeAD
- Chapel-Enzyme integration

How do we compute derivatives on a computer?

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \qquad f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

$$f(x) = x^3 + x^2 - x + 3$$



Enter automatic differentiation

- **Automatically differentiation**: compute derivative of code
  - "Compile code to its derivative"
  - No numerical error (other than normal floating point operations)
- Opposed to **symbolic differentiation**, dealing only with mathematical expressions

Types of automatic differentiation: 1st axis

$$y = f(g(h(x)))$$

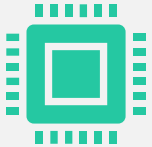
$$w_1 = h(x)$$
$$w_2 = g(w_1)$$
$$w_3 = f(w_2) = y$$

backward

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_2}\frac{\partial w_2}{\partial w_1}\frac{\partial w_1}{x}$$

forward

Types of automatic differentiation: 2nd axis



Operator overloading: Overload mathematical functions on custom types modeling derivatives

Source transformation: Automatically generate source code for derivative

ForwardModeAD

- Chapel library for forward-mode automatic differentiation based on operator overloading
- Design principles:
  - Easy to use
  - Compose with other functions and code
- Supports derivatives, gradients, jacobians, directional derivatives, JVP
- Inspired by ForwardDiff.jl

Mathematics behind it

$$a + b\epsilon, \epsilon^2 = 0$$

$$F + G = (f + g) + (f' + g')\epsilon$$
$$F \cdot G = fg + (f'g + fg')\epsilon$$
$$\frac{F}{G} = \frac{f}{g} + \frac{f'g - fg'}{g^2}\epsilon$$

$$f(a + a'\epsilon) = f(a) + f'(a)a'\epsilon$$

Implementation in chapel: define dual type

```
record dual {

  /* primal part of the dual number */
  var primalPart : real;

  /* dual part of the dual number */
  var dualPart : real;
}
```

Implementation in Chapel

```
operator +(a, b) where isEitherDualType(a.type, b.type) {
    var f = primalPart(a) + primalPart(b);
    var df = dualPart(a) + dualPart(b);
    return todual(f, df);
}
```

```
proc sin(a) where isDualType(a.type) {
    var f = sin(primalPart(a)),
        df = cos(primalPart(a)) * dualPart(a);
    return todual(f, df);
}
```

Example usage: Newton method

```
use ForwardModeAD;

proc f(x) {
    return exp(-x) * sin(x) - log(x);
}

var tol = 1e-6, // tolerance to find the root
    cnt = 0, // to count number of iterations
    x0 = initdual(0.5), // initial guess
    valder = f(x0); // initial function value and derivative

while abs(value(valder)) > tol {
    x0 -= value(valder) / derivative(valder);
    valder = f(x0);
    cnt += 1;
    writeln("Iteration ", cnt, " x = ", value(x0), " residual = ", value(valder));
}
```

Limitations, future work

- Dual numbers should be parametric types
  - e.g. work with MPFR wrappers
  - Custom number types, like double-double arithmetic
- Better support for lambda functions
- Interface not fixed, if you don't like the syntax, open an issue!

Enzyme

- Library for automatic differentiation **at LLVM level**
- Supports both forward and backward mode
- Designed for high-performance automatic differentiation

Why is LLVM the right abstraction level?

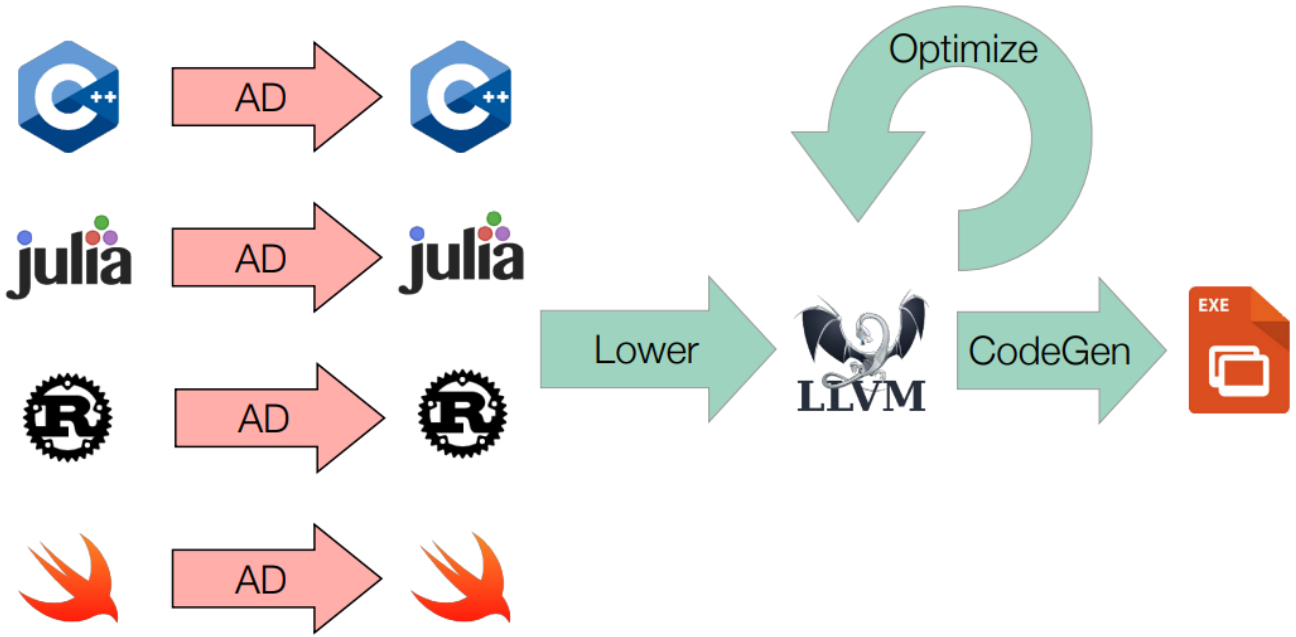
```
//Compute magnitude in O(n)
double mag(double[] x);

//Compute norm in O(n^2)
void norm(double[] out, double[] in) {

  for (int i=0; i<n; i++) {
    out[i] = in[i] / mag(in);
  }
}
```

Picture taken from
: https://indico.cern.ch/event/1145124/contributions/4994088/attachments/2508821/4311554/enzyme-mode.pdf

Why is LLVM the right abstraction level?



Early experiments: Chapel – Enzyme integration

Compile Chapel to LLVM

AD the generated LLVM with Enzyme

Resume compilation

Chapel – Enzyme working example

```
use CTypes;

extern {
  double __enzyme_autodiff(void*, ...);
}

proc square(x: real) {
  return x * x;
}

proc dsquare(x: real): real {

  return __enzyme_autodiff(c_ptrTo(square): c_ptr(void), x);
}


for i in 1..4 {
  writeln("x = ", i, " f(x) = ", square(i), " f'(x) = ", dsquare(i));
}
```

Status and future work

- Works with scalar functions in forward and backward mode
  - Backward though is dummy, because it works computing one input at the time
- Relies on C – Chapel interoperability

Future work

- Support arrays, structs and more advanced language features
- Benchmarks
- What is a good interface?

Thank you!

- ForwardModeAD: https://github.com/lucaferranti/ForwardModeAD
- Chapel – Enzyme integration: https://github.com/lucaferranti/chapel-enzyme
- Me on linkedin: https://www.linkedin.com/in/luca-ferranti/