Argonne NATIONAL LABORATORY

Vector Forward Mode Automatic Differentiation on SIMD/SIMT architectures

Jan Hückelheim, Michel Schanen,
Sri Hari Krishna Narayanan, Paul Hovland

Argonne National Laboratory

August 10, 2020

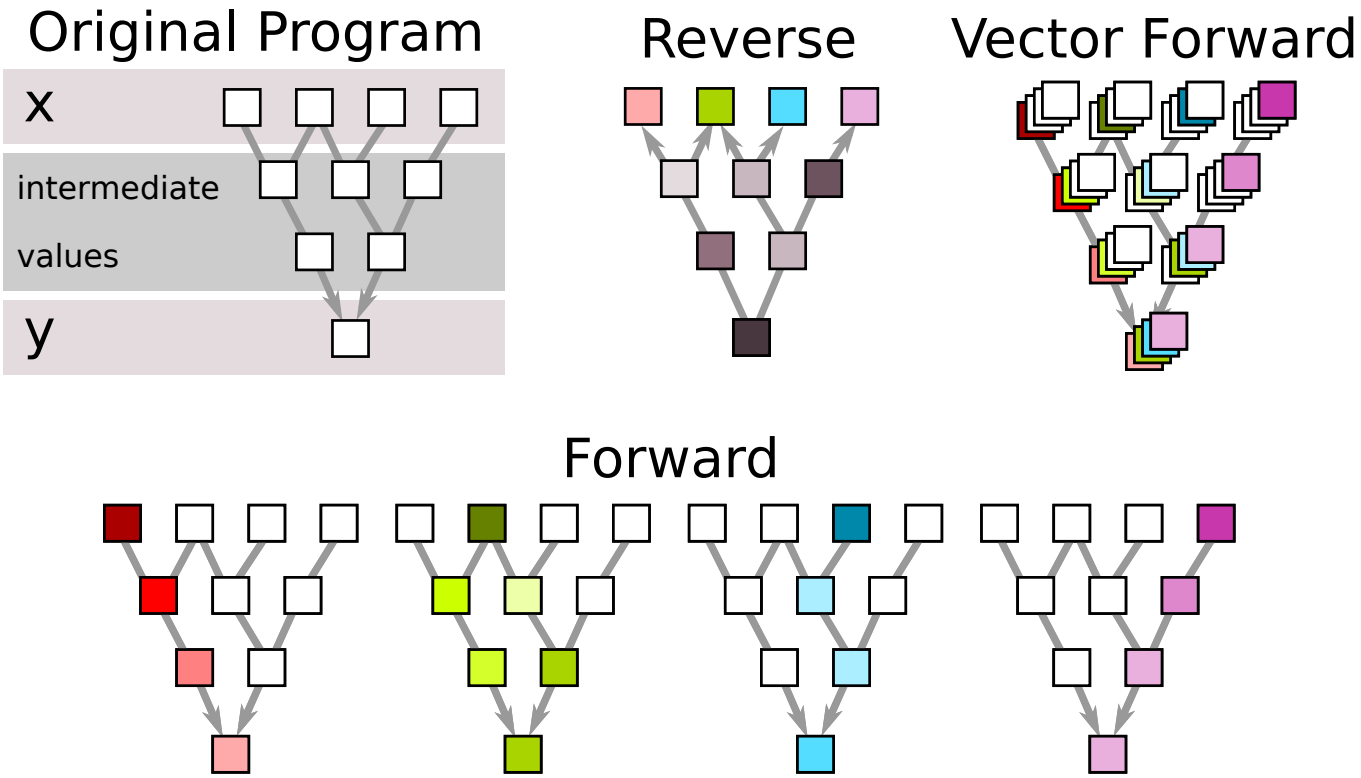
Outlines

Outline

Automatic Differentiation (AD)

- Given: a program that computes $y \leftarrow F(x)$
- AD produces program that computes derivatives of F.
- AD can be implemented using source-to-source compilers, run-time tracing, JIT, ...
- Many tools exist, for a variety of languages including Python, C/C++, Fortran, see `autodiff.org` community website for an overview
- AD is not ”numerical differentiation” or ”finite differences”: There is no finite step size, no truncation error
- AD can be seen as symbolic differentiation for real-world programs, including branches, loops, function calls, etc

Forward vs Reverse, scalar vs vector

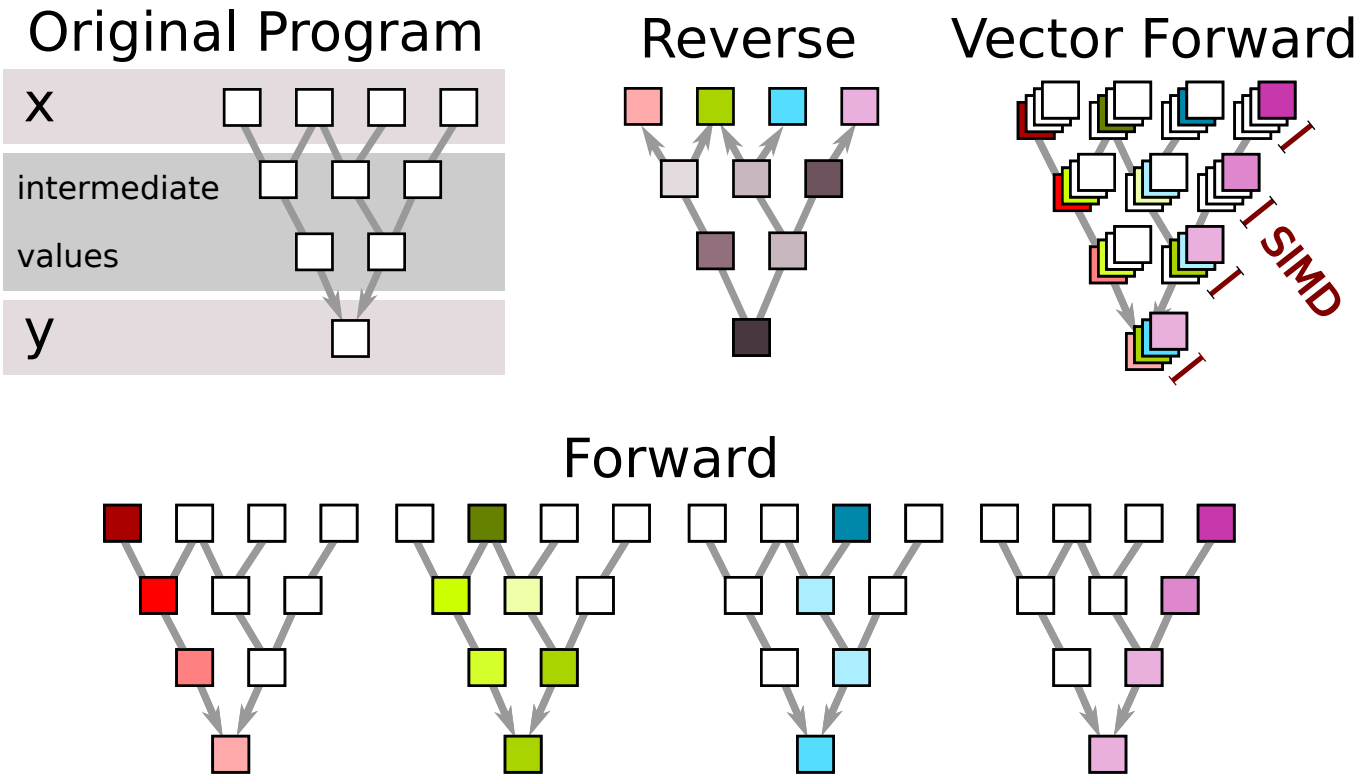


- Forward mode is simple to understand and implement, but: Need to re-run for every input.
- Reverse mode is cheaper for many inputs and few outputs (run once, get all directional derivatives).

Outline

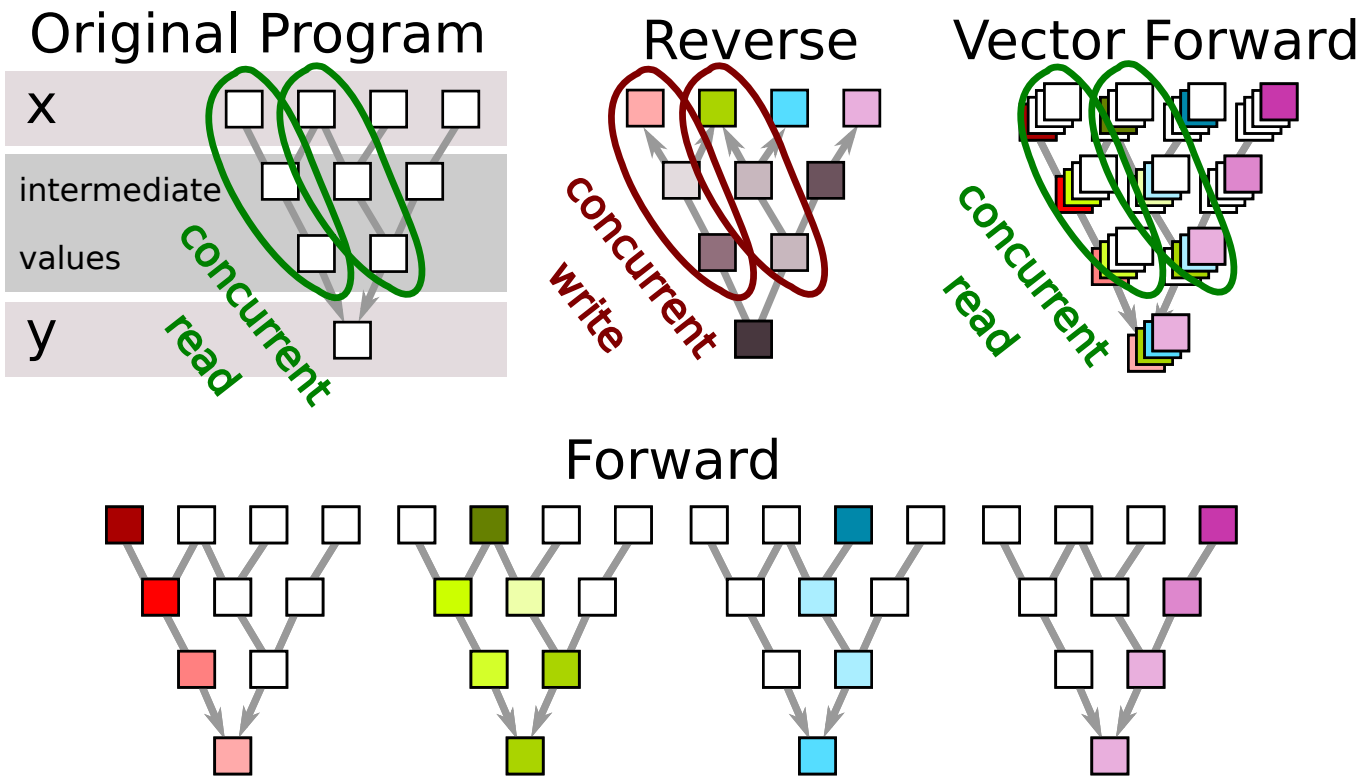
Parallelization challenges and opportunities

- Forward mode AD has the same data flow as original program. Can keep parallelism
- Vector-forward mode is another dimension of parallelism. Free of branch divergence, also good for SIMD/SIMT



Parallelization challenges and opportunities

- Reverse mode is cheaper for many inputs and few outputs (run once, get all directional derivatives)
- Reverse mode changes data flow, can cause new data races. Hard to analyze by an AD tool



Wait... But didn't back-propagation work in parallel?

- Some frameworks (e.g. TensorFlow, Pytorch, Halide) support parallel reverse mode, but:
- They operate on a higher level of abstraction, e.g.
  - composing manually-parallelized building blocks or
  - generating code while exploiting problem structure
- They are not general-purpose

What AD mode is best for my application?

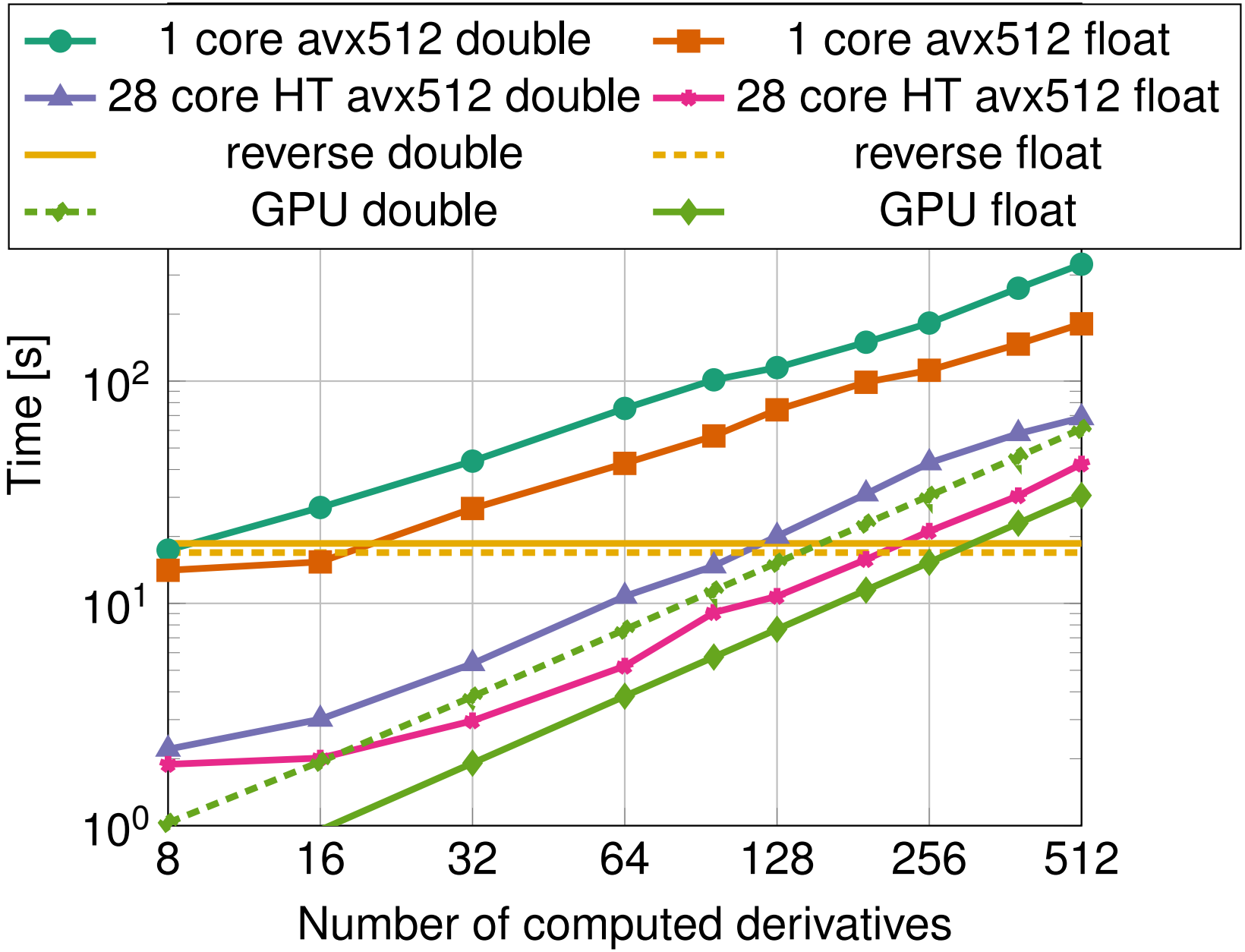
- This is commonly known in AD literature:
  - Forward mode is easy, low overhead, but gets costly for *many* inputs
  - Reverse mode is hard, high overhead, but it's worth it for *many* inputs
- But what is *many*?
  - Where is the break-even point on today's hardware?
  - What might it depend on?
  - We present a case study here.

Outline

Test Case 1 Description: Stencil

- Stencil computations are common in PDE solvers, convolutions, image processing
- Are easy to parallelize, but vectorize poorly due to misaligned data
- Reverse mode AD is difficult to parallelize (not supported by AD tools)
- We compare performance of primal, forward, and reverse mode, on CPU (Intel Skylake) and GPU (Nvidia Quadro GV100)
- Stencil is taken from Parboil benchmark suite, differentiated with Tapenade AD tool
- Vector-forward-mode is post-processed to insert OpenMP SIMD directives before direction loops
- GPU version is created by manual translation to Julia, then using Julia's built-in AD and GPU support

Test Case 1 Results: Stencil



Outline

How useful are O(100)-O(1000) inputs?

- Number of inputs != number of state variables. For example: CAD parameters in a simulation with many more state variables
- Coloring can reduce number of derivative directions. For example: Power Flow application with over half million inputs, due to sparsity, needs only 30 forward mode evaluations (see paper for details).

Conclusions

- Forward mode can be surprisingly competitive to reverse mode, for hundreds of inputs
- With current hardware trends (more parallelism, longer vectors, ...), this number may grow further
- We assume here that the reverse mode can not be auto-parallelized or auto-vectorized. Maybe (hopefully) this will change?
- We are happy to hear your questions and comments by email, jhueckelheim@anl.gov
- If you are watching this as part of ICPP20, please visit our Q & A session.

This work was funded in part by support from the U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.