

Automatic Differentiation for Parallel Programs

Laurent Hascoët
Paul Hovland
Jan Hückelheim
Sri Hari Krishna Narayanan

SIAM AN24 Student Days



U.S. DEPARTMENT OF
ENERGY

Office of
Science

Why do we need AD for parallel code?

- ▶ Most codes that care about performance exploit parallelism.
- ▶ **If the original program is parallel, can we recycle the parallelization for the derivative computation?**
- ▶ Short answer: Sometimes.
- ▶ The longer answer follows...

Forward Mode

- ▶ For distributed memory (e.g. MPI) in forward mode:
 - ▶ if a variable is sent, then its derivative must be sent.
 - ▶ if a variable is received, then its derivative must be received.

```
! Primal
call MPI_SEND( data, count, ... )
! Forward-mode AD
call TLS_MPI_SEND(data, datad, count, ... )
```

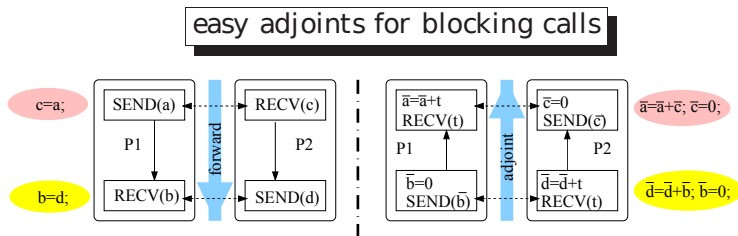
- ▶ For shared memory (e.g. OpenMP) in forward mode:
 - ▶ Derivative variables get the scope of primal counterparts.

```
! Primal
!$omp parallel do private(c) shared(data)
! Forward-mode AD
!$omp parallel do private(c) shared(data, datad)
```

- ▶ Tapenade supports MPI and OpenMP.

Reverse-Mode AD for MPI

- Communication flow is reversed.
- If the original code sends, the adjoint code must receive.
- If the original code receives, the adjoint code must send.
- Challenges around non-blocking communication, one-sided communication, wildcards, and other subtleties.
- Adjoint MPI: libraries are available, and used e.g. by Tapenade



Graphic: J. Utke, Adjoints of MPI programs, ECCO2 meeting slides, Argonne National Laboratory, 2008

Reverse-Mode AD for OpenMP

- ▶ Multiple threads run in parallel (e.g. on multi-core CPU)
- ▶ Memory visible to all threads, no explicit communication
- ▶ Parallel read-access is fine, parallel write access is a problem
- ▶ **Data-flow reversal potentially creates those problems!**



Breaking the parallelization barrier?

- ▶ Reverse AD for shared-memory parallel programs is hard, and will probably never work efficiently for *arbitrary* parallel input programs.
- ▶ For well-defined circumstances, much progress has been made, and there is promising ongoing research.
- ▶ Some tools (e.g. Tapenade) offer some support, new prototype tools and libraries are in development.
- ▶ In the meantime, if you need to differentiate a real-world parallel application *now* and can't wait for more research...

Going over the barrier instead?

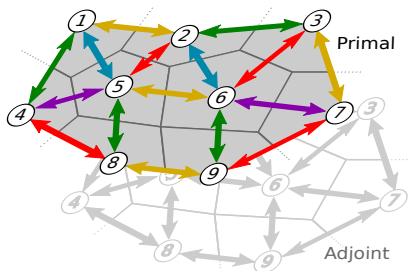
- ▶ There are (very successful) new high-level tools that operate on domain-specific languages, e.g. in PyTorch, TensorFlow, Halide, FireDrake
- ▶ Keep performance tweaks and parallelization "under the radar"
- ▶ Differentiate high-level code and express derivatives in terms of existing high-level building blocks.

```
! Primal
res = numpy.dot(a,b)
! Reverse-mode AD
ab = ab + numpy.multiply(b,resb)
bb = bb + numpy.multiply(a,resb)
```

- ▶ If the dot function is parallelized internally, we have parallel reverse-mode AD without doing any work.

Going under the barrier instead?

- ▶ Apply AD to individual iterations of parallel loops.
- ▶ Similarly: Apply AD to code blocks between MPI calls
- ▶ Manually take care of the communication reversal
- ▶ Sometimes, there are application properties that can be used to make life easier. Examples:
 - ▶ symmetry between read- and write-indices in parallel solvers means that parallelization can be safely re-used.
 - ▶ Halo / ghost cell exchange and domain decomposition often works similarly for the primal and reverse-AD code.



Parallel tool landscape

- ▶ Tapenade, ADOL-C, Adept support (all or some subset of) OpenMP and MPI in forward and reverse
- ▶ JAX, TensorFlow, PyTorch, Halide, ... go “over the barrier” and do high-level AD
- ▶ Many application-specific approaches exist for “under the barrier” parallel AD with hand-assembled parallelization
- ▶ Vector-forward and vector-reverse mode can be parallelized easily, even if the original program is sequential
- ▶ AD may *target* CUDA and parallel libraries for faster derivative computation, even if not fully supporting those languages as *input* programs
- ▶ Expect some potholes and road blocks when differentiating parallel programs.

Thank you

Questions?