solvers and derivatives

Jean Utke[1]

[1]University of Chicago and Argonne National Laboratory

Sisiphus Meeting
Feb. 16 2010

simple

context: apply automatic differentiation to models that use (linear) solvers.

- have $\boldsymbol{Ax} = \boldsymbol{b}$
- also have solver (source code) to do the mapping $\boldsymbol{b}[, \boldsymbol{A}] \mapsto \boldsymbol{x}$
- want forward derivatives $\dot{\boldsymbol{x}} = \boldsymbol{A}^{-1}(\dot{\boldsymbol{b}}[-\dot{\boldsymbol{A}}\boldsymbol{b}])$ (parameter sensitivities)
- want adjoints $\bar{\boldsymbol{b}} = \boldsymbol{A}^{-T}\bar{\boldsymbol{x}}$ [and $\bar{\boldsymbol{A}} = -\boldsymbol{A}^{-T}\bar{\boldsymbol{x}}\boldsymbol{x}^T = -\bar{\boldsymbol{b}}\boldsymbol{x}^T$]
  gradients for state estimates
- questions
  - are $\boldsymbol{A}, \boldsymbol{b}$ active ?
  - which solver is being used?
  - ignore the context and differentiate through with AD ?
  - efficiency/accuracy?

most models need an answer (not only climate research but also other subject areas, e.g. NE, economics)

what kind of solvers?

- direct / iterative
- reuse the factors / derivative convergence
- self-adjoint?
- home grown solvers / libraries (petcs,lapack,...slap)

for example - lapack

... because I tried this myself

- linear system solvers, also for least-squares solutions, eigen/singular value problems
- but lapack uses blas ...
- blas = basic linear algebra subprograms
  - scalar,vector,vector/vector, matrix/vector,matrix/matrix operations
  - variations on precision and real vs complex
  - total of 150 subroutines and functions in F77
  - F77 reference implementation (slow)
  - vendor specific implementations, ATLAS, Goto are optimized for performance

lapack ... contd.

similar situation here

- reference implementation on netlib
- again vendor implementations optimized for speed
- with type/precision variations 1.5k routines (400+ marked "auxiliary")

observations after experimenting with a nuclear physics code

- blas reference implementation is known to be slow
- contains some manual code optimizations that can mislead AD tools
- lapack to blas calls use a lot of difficult-to-analyze offsets into work arrays
- efficiency problems with combinations of matrix-vector and matrix/matrix ops $\Rightarrow$ inefficient derivatives
- one-shot implementations are not reusable across AD tools
- variants caused by different activity patterns

observations apply to libraries in general

solution

near term:

- use recipies of existing OpenAD capabilities for wrapping solver calls (PatricK)
- provide solutions for use of slap/petsc solvers in ice models

long term:

- treat blas/solver routines as high-level intrinsics
- generate derivative code & interfaces
- performance advantage from explicit derivative computations
- avoid pitfalls from brute force differentiations (for example problem with dgesvd from Bastani/Guerrieri)
- reusable solution

why automatic differentiation?

given: some numerical model $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) : \mathbb{R}^n \mapsto \mathbb{R}^m$ implemented as a (large / volatile) program

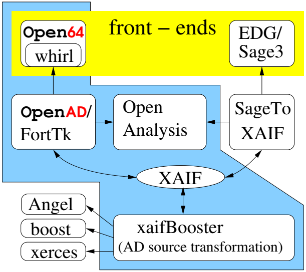
wanted: sensitivity analysis, optimization, parameter (state) estimation, higher-order approximation...

1. don't pretend we know nothing about the program
   (and take finite differences of an oracle)
2. get machine precision derivatives as $\boldsymbol{J}\dot{\boldsymbol{x}}$ or $\bar{\boldsymbol{y}}^T\boldsymbol{J}$ or ...
   (avoid approximation-versus-roundoff problem)
3. the reverse (aka adjoint) mode yields "cheap" gradients
4. if the program is large, so is the adjoint program, and
   so is the effort to do it manually ... easy to get wrong but hard to debug

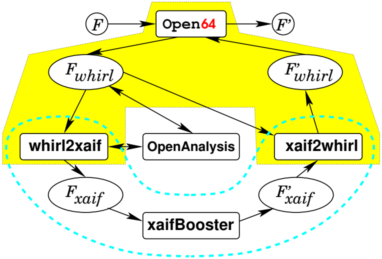
OpenAD overview - current

- www.mcs.anl.gov/OpenAD
- forward and **reverse**
- source transformation
- modular design
- aims at large problems
- language independent transformation
- researching combinatorial problems
- current Fortran front-end Open64 (Open64/SL branch at Rice U)
- uses *association by address*

  (i.e. has an active type)
- Rapsodia for higher-order derivatives via type change transformation

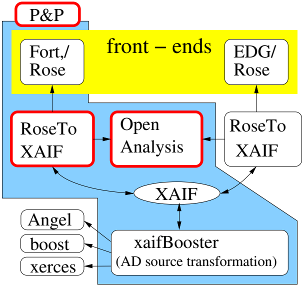


Fortran pipeline:

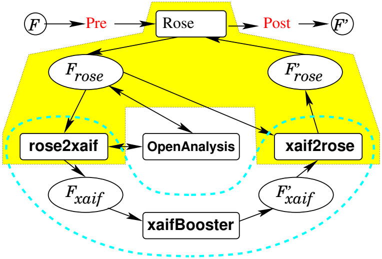


OpenAD overview - changes

- expanded language coverage (common blocks, equivalence, unstrucctured control flow, intrinsics,...)
- new pre- and postprocessor (python, MITgcm consequences)
- migration from Open64 to Rose (LLNL)



Fortran pipeline:



some research toopis

- adjoinable MPI
- optimal local preaccumulation (scarcity)
- additional parallelism from checkpointing
- higher order derivatives (in parallel)
- ...

some research toopis

- adjoinable MPI
- optimal local preaccumulation (scarcity)
- additional parallelism from checkpointing
- higher order derivatives (in parallel)
- ...
- make it work on code <*insert something here*> ...

some other applications

- suite of reactor models
  - old style Fortran
  - equivalence, unstructured control flow,...
- transport of nuclear materials (container safety)
  - Fortran 9X
  - dependecies via files
  - dynamic memory
- forthcoming: ice sheet models (NSF and DOE projects)

needs migration to Rose

for MITgcm

- installed on beagle (updated/recompiled nightly)
- w. Chris (use w/o intervention)
  - cost function change,
  - adding extra output
  - compiler optimization
  - computational cost
- w. Patrick 20 year 1x1 run on beagle
  - setup hurdle (find the right combination of modules for the sge run script)
  - bottleneck checkpointing via NFS (switch to local disk)
- usability: remove extra steps e.g. Common Block to Module conversion, some specific changes to non-transformed files. e.g. cost_final
- next step w. Chris: "high-res" run