Reducing the Computational Complexity of Adjoint Computations

William W. Symes

CAAM, Rice University, 2007

Agenda

- Discrete simulation, objective definition
- Adjoint state method
- Checkpointing
- Griewank's optimal schedule
- Implementation
- Examples
- Continuum adjoint state and adaptive time stepping
- Summary

Discrete Time Evolution

$$\mathbf{u}^{n+1} = \mathbf{H}^n[\mathbf{c}, \mathbf{u}^n],\ n = 0, 1, ..., N-1$$

Note:

- $\mathbf{u}^n \in U$ approximates state in *state space* U at $t = t_n, n = 0, ...N$;
- time-dependence of RHS accommodates possible time-dependence of control $\mathbf{c} \in C$, C = control space, and of other factors;
- control $\mathbf{c}$ can be initial data and/or material parameter fields or source term or actuator history or pump rate or ...
- RHS ($\mathbf{H}^n$) represents solution operator of discrete dynamics - can be either explicit or implicit. Originates in a continuum dynamics system (PDE), via finite element / difference / volume method.

Objective or Cost Functions

$\mathbf{u}[\mathbf{c}] = (\mathbf{u}^0, \mathbf{u}^1, ..., \mathbf{u}^N)^T \in U^N$ = time history of simulation - implicitly a function of $\mathbf{c}$.

$\mathbf{S} : U^N \to E$ = *sampling operator* - assume linear for simplicity,, though this is not really necessary. Sample space E may be U(Meyer control) or time series of projections of U (seismic traces, trajectory projections,...) or ...

$\mathbf{G} : E \to \mathbf{R}$ = *"goodness" function.*

Objective or *cost* function $J : C \to \mathbf{R}$ via

$$J[\mathbf{c}] = \mathbf{G}[\mathbf{S}[\mathbf{u}[\mathbf{c}]]]$$

Adjoint State Method

For computing the gradient of J:

- compute $\mathbf{u}[\mathbf{c}] = (\mathbf{u}^0, ..., \mathbf{u}^N)^T$;
- initialize $\mathbf{g} \in C$ and $\mathbf{w}^{N+1} \in U$ to zero. Then for $n = N-1, ...0$

$$\begin{aligned} \mathbf{w}^{n+1} &= D_u H^{n+2}[\mathbf{c}, \mathbf{u}^{n+1}]^T \mathbf{w}^{n+2} + [\mathbf{S}^T (\nabla \mathbf{G})[\mathbf{S}[\mathbf{u}[\mathbf{c}]]]]_{n+1} \\ \mathbf{g} &= \mathbf{g} + D_c H^n[\mathbf{c}, \mathbf{u}^n] \mathbf{w}^{n+1} \end{aligned}$$

- when $n = 0$ is reached, $\nabla J[\mathbf{c}] = \mathbf{g}$.

Observation: $\mathbf{u}$ evolves *forward* in step index, $\mathbf{w}$ *backward* in step index, but they are needed at indices $n, n+1$ respectively, $n = N-1, ...0$.

Computational Complexity

Strategies for simultaneous access to $\mathbf{u}^n, \mathbf{w}^{n+1}$ – in all cases $\mathbf{w}^{n+1}$ evolved backwards from $n = N$ to $n = 0$.

1. For each n, evolve $\mathbf{u}^n$ from $n = 0$, or
2. Compute $\mathbf{u}^0, ..., \mathbf{u}^N$, store all; For each n retrieve $\mathbf{u}^n$, or
3. Compute $\mathbf{u}^0, ...\mathbf{u}^N$, store every kth state, $k > 1$; for each n, interpolate n state from closest stored states, or
4. Compute $\mathbf{u}^0, ...\mathbf{u}^N$, evolve $\mathbf{u}^n$ backwards in time from $n = N$.

Computational Complexity

Cost: in units of simulation steps (flops) to compute $\mathbf{u}$, and number of state vectors stored:

1. working storage (1 state vector) for but $N^2/2$ steps - prohibitive;
2. N steps, N state vectors;
3. also N steps, N/k state vectors, but loss of accuracy due to use of interpolation rather than evolution;
4. $2N$ steps, 1 state vector, but only possible for conservative / time reversible problems.

Example: Reverse Time Migration

("RTM"): Adjoint state method applied to least squares residual seismogram. Theory $\Rightarrow$ in some instances, gradient of least squares residual is *image* of subsurface.

Increasingly popular because of its insensitivity to complexity of acoustic wavepaths (full session, 06 SEG).

3D RTM - typical state space dimension $\simeq 10^{13}$ w (= 10^9 space grid $\times$ 10^4 shots), N $\simeq 10^4$. Flops per space-time gridpoint for standard regular grid finite difference schemes $\simeq 10^2$.

$\Rightarrow$ cost per time step $\simeq 10^{15}$ flops. Storage per state vector $\simeq 10^9$ w (natural algorithms work per shot).

Example: Reverse Time Migration

Upshot:

- strategy 1 hopeless ($O(10^{38})$ flops);
- probably also strategy 2 ($10 - 100$TB storage);
- absorbing boundary conditions make wave equation time-irreversible, but some schemes admit variants of strategy 4 with considerable additional storage (not available with attenuation modeling).

Commercial 2D prototypes use strategy 3 with $k \simeq 10$. Even for 2D ($\times 10^{-3}$), application is I/O bound; for 3D, requires 1 - 10 TB.

Checkpointing

Alternative to strategies 1-4. Requires allocation of

- N_B buffers, each storing one state vector;
- $N_C >> N_B$ checkpoints = integers between 0 and N.

Forward sweep (n=0,...,N): solve forward evolution problem to compute $\mathbf{u}^0, \ldots, \mathbf{u}^N$; store N_B checkpoints in the buffers, including the first (always n=0) and last.

Backwards sweep (n=N-1,...,0): begin by using strategy 1, *starting at the last checkpoint*. When the n = last checkpoint, re-use its buffer to store another checkpoint. computing its state by application of strategy 1 starting from the previous stored checkpoint. Continue using strategy 1, starting from next-to-last checkpoint [this must be the replacement for the last checkpoint, unless it was previously stored]. Continue. At end of algorithm, buffers store some number of states starting with $n = 0$; finish using strategy 2.

Checkpointing

Example with $N = 15, N_B = 3, N_C = 6$

Meaning of colums:

- bufk records checkpoint stored in buffer k;
- *recomp* records the previously computed steps which are *recomputed* in each step of the backwards sweep, or *dash* if no recomputation necessary in step;
- *bold faced* checkpoints used as Cauchy data for strategy 1;
- *italic*: n for which $\mathbf{u}^n$ combined with $\mathbf{w}^{n+1}$ in evaluation of gradient update.

During forward sweep checkpoints 0, 6, 11 recorded in buffers 1, 2, and 3.

step	buf1	buf2	buf3	recomp
14	0	6	**11**	12,13,*14*
13	0	6	**11**	12, *13*
12	0	6	**11**	*12*
11	0	**6**	*11*	7, 8
10	0	6	**8**	9, *10*
9	0	6	**8**	*9*
8	0	6	*8*	-
7	0	**6**	8	*7*
6	0	*6*	8	-
5	**0**	1	3	1, 2, 3, 4, *5*
4	0	1	**3**	*4*
3	0	1	*3*	-
2	0	**1**	3	*2*
1	0	*1*	3	-
0	*0*	1	3	-

Griewank's Optimal Checkpoint Schedule

Big question: how do you choose checkpoints to

- minimize the amount of recomputation for given storage allocation (N_B), or
- minimize the amount of storage required for a given level of recomputation.

Solution by Griewank, *Opt. Meth. and Software*, 1992, published as Alg. 799, Griewank and Walther, *ACM TOMS* 2000, in terms of *recomputation ratio* = total number of forward steps required to compute adjoint / N.

Griewank's Optimal Checkpoint Schedule

Example, $N = 10000$:

buffers	3	5	10	15	20	25	30	35	40	60
ratio	27.9	11.3	5.8	4.5	3.8	3.6	3.4	3.1	2.9	2.8

Storage for 36 state vectors $\Leftrightarrow$ total cost of adjoint $\simeq$ 3 times forward simulation + 1.5 times for adjoint step ($\mathbf{w}^{n+1} \mapsto \mathbf{w}^n$) $\simeq$ **4.5 times simulation cost**.

Comparisons: with straight app of strategy 2, cost is 2.5 times simulation cost and *300 times as much storage!* Strategy 3 requires "only" 30 times as much storage but loses accuracy.

Example: for 3D RTM, use of opt. checkpointing drops requires storage to $O(100)$ GB, may eliminate disk i/o.

Implementation

Within **TSOpt** framework, adjoint step with optimal checkpointing implemented via three classes:

`RealFunction`: abstract interface specializing `LocalDataContainer`, representing a function of a real variable via a `set(Scalar t)` method.

`GriewankRealFunction`: implementation of `RealFunction` using a `Stencil` object to compute and store checkpointed state vectors, returns interpolation of nearest stored checkpoints to requested "time". *Uses TOMS Alg 799 code!!!*

`Dynamics`: base class for time stepping, includes `adjStep` method, which accepts a state vector arg of type `LocalDataContainer`.

Example

2D RTM using standard centered difference (2,4) schemes implemented in TSOpt.

Parallelization over shots (i.e. individual simulations) via parallel `DataContainer` subclass `MPI_PackageContainer`. [For 3D, parallelization of individual simulations will be required as well.]

Applied to Marmousi benchmark synthetic data: 240 shots, 3 s data 4 ms. Model is 826×2350 gridpoints ($4\text{m} \times 4\text{m}$), absorbing boundaries on all sides (PML). With internally computed time grid, $\simeq 8000$ time steps.

Time per simulation on AMD Opteron 275: 10 min. Time to simulate entire data set on Rice Cray XD-1 Opteron cluster using 120 cores: 20 min.

Time for adjoint state computation using 32 checkpoints, 120 cores (recomp ratio = 3): 90 min.

Continuum Adjoint State and Adaptive Time Stepping

With adaptive time stepping, grid for simulation **must** in general be independent of grid for linearized and adjoint simulation. [Trivial example: adaptive quadrature.]

Therefore must return to *continuum* adjoint state method for the differential equation

$$\frac{d\mathbf{u}}{dt} = \mathbf{H}[\mathbf{c}, \mathbf{u}, t]; \;\; \mathbf{u}(0) = \mathbf{u}_0$$

and the objective J defined as before,

$$\nabla J[\mathbf{c}] = D\mathbf{u}[\mathbf{c}]^T \mathbf{S}^T \nabla \mathbf{G}[\mathbf{S}[\mathbf{u}[\mathbf{c}]]]$$

$$= \int_0^T dt\, D_c \mathbf{H}[\mathbf{u}[\mathbf{c}](t), \mathbf{c}, t]^T \mathbf{w}(t),$$

Continuum Adjoint State and Adaptive Time Stepping

where the *continuum adjoint state* $\mathbf{w}$ satisfies

$$\frac{d\mathbf{w}}{dt}(t) + D_u H[\mathbf{u}[\mathbf{c}](t), \mathbf{c}, t]^T \mathbf{w}(t) = \mathbf{S}^T \nabla \mathbf{G}[\mathbf{S}[\mathbf{u}[\mathbf{c}](t)]]$$

How to approach this computation: use a comparably accurate scheme to solve this adjoint state equation, and a “real function” class like `GriewankRealFunction` to return the values of $\mathbf{u}[\mathbf{c}](t)$ required, as efficiently as possible for a given allocation of auxiliary storage. NB: these values with *never* be those computed in the computation of $\mathbf{u}[\mathbf{c}]$ by time stepping!

More details: stay tuned for *Marco Enriquez MA thesis*.

Summary

- Adjoint state method poses interesting computational complexity problem;
- Griewank solved it, and provided C and F77 realizations in ACM TOMS 799 (2000);
- This is enabling technology: it brings problems into reach which would otherwise be untouchable, and reduces the floating point and memory complexity of large-scale sim-driven opt problems (eg. 3D RTM) to manageable levels;
- TSOpt incorporates Griewank's optimal checkpointing scheme;
- Modification for adaptive gridding straightforward: since Griewank checkpointing does *discrete* backwards stepping optimally, it is also the optimal tool for extracting state at arbitrary times (augmented by interpolation).