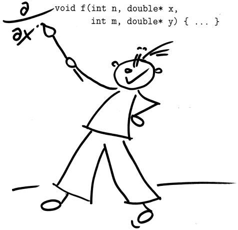


Mixed Integer Programming for Call Tree Reversal

J. Lotz, U. Naumann, S. Mitra

LuFG Informatik 12: Software and Tools for Computational Engineering (STCE)
RWTH Aachen University
and
Chemical Engineering, Carnegie Mellon University

[CSC16, Albuquerque, NM, Oct. 10–13, 2016]

Outline

Outline

Motivation

Call Tree Reversal

Conclusion and Outlook

Consider solution $x^\tau = x(\tau)$ of initial value problem (IVP)

$$\frac{dx}{dt} = f(t,x), \quad t \geq 0,\ x = x(t) \in \mathbb{R}^n,\ x(0) = x^0$$

at target time $\tau > 0$.

Gradient-based calibration of initial condition x^0 benefits from adjoint

$$\bar{x}^0 = \frac{dx^\tau}{dx^0}^T \cdot \bar{x}^\tau \in \mathbb{R}^n,$$

where $\bar{x}^\tau \in \mathbb{R}^n$ is gradient of, e.g, least-squares objective matching solution of x^τ to given observations.

Computation of $\bar{x}^0$ amounts to solution of adjoint IVP

$$-\frac{d\bar{x}}{dt} = \frac{df(t,x)}{dx}^T \cdot \bar{x}, \quad \tau \geq t \geq 0,\ \bar{x} = \bar{x}(t) \in \mathbb{R}^n,\ \bar{x}(\tau) = \bar{x}^\tau.$$

W.l.o.g, explicit Euler time stepping yields for $\Delta t = \tau/m$

- primal
$$x^{i+1} = x^i + \Delta t \cdot f(x^i), \quad i = 0, \ldots, m-1$$
- algorithmic adjoint
$$\bar{x}^i = \bar{x}^{i+1} + \Delta t \cdot \frac{df}{dx}^T (x^i) \cdot \bar{x}^{i+1}, \quad i = m-1, \ldots, 0$$
- symbolic adjoint
$$\bar{x}^i = \bar{x}^{i+1} + \Delta t \cdot \frac{df}{dx}^T (x^{i+1}) \cdot \bar{x}^{i+1}, \quad i = m-1, \ldots, 0$$

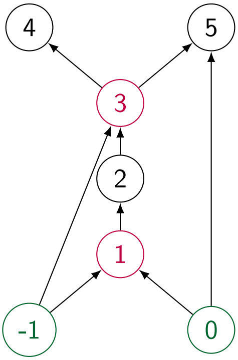
Note use of primal iterates in reverse order!

▶ A. Griewank: Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation, Opt. Meth. Softw. 1, 35-54 (1992).

Problem: Recovery of $|V| = n + q$ non-persistent (vertex) values in reverse order $(v_5, \ldots, v_{-1})$; extreme cases: store-all, recompute-all

Objective: Minimization of Primal Reevaluation Cost (PRC) for given upper bound M on Persistent Memory Requirement (PMR)

Complexity: Fixed Cost $(n+q)$ Minimum Memory Data Flow Reversal by reduction from Vertex Cover solvable by $O(n+q)$ instances of Fixed Memory Minimum Cost Data Flow Reversal



$G = (V, E)$

$n = 2; \ q = 5$

- U.N.: DAG Reversal is NP-Complete, J. Disc. Alg. 7(4), 402-410 (2009).

Outline

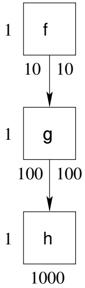
Motivation

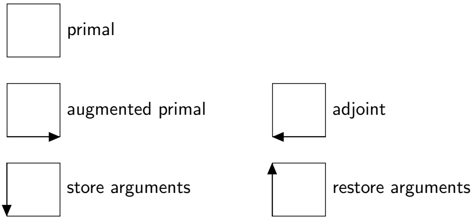
Call Tree Reversal

Conclusion and Outlook

Call Tree Reversal

Problem Description and Computational Complexity





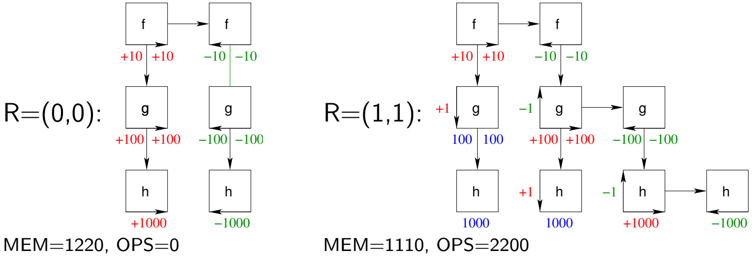
Objective: Reversal scheme R $: E \rightarrow \{0,1\}^{|E|}$ minimizing PRC for upper bound M on PMR and given annotated call tree $T = (V, E)$

Extreme Cases: R $= \mathbf{0}$ (fully split; checkpoint none); R $= \mathbf{1}$ (fully joint; checkpoint all)

- U.N.: Call Tree Reversal is NP-Complete, LNCSE 64, 13-22 (2008).

Call Tree Reversal

Example: Let $\overline{\text{MEM}} = 1110$ ...



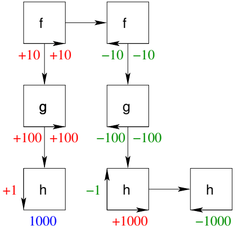
Notation: Set S of subprogram calls; Number $n[i]$ of subprogram calls in $i \in S$; Set $\chi(i)$ of callees; PMR $m[i] = \sum_{j=0}^{n[i]} \mathbf{m}[i]_j$; Reversal scheme R $= (r[i])_{i \in S}$; PMR $\overrightarrow{M}[i]$ after augmented primal run; PMR $\overleftarrow{M}[i]$ prior to adjoint run; PMR $\downarrow M[i]$ of argument checkpoint

Call Tree Reversal

Example: Let $\overline{\mathsf{MEM}} = 1110$ ... Greedy Heuristics

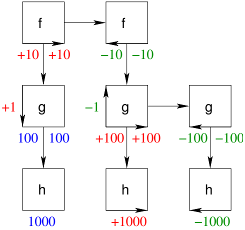
Smallest Memory Increase starts from R = $\mathbf{1}$ and yields . . .
Largest Memory Decrease (LMD) starts from R = $\mathbf{0}$ and yields . . .

R=(0,1):



MEM=1110, OPS=1000

R=(1,0):



MEM=1120, OPS=1200

Largest Memory Increase (LMI) remains at R = $\mathbf{1}$ as R = $(1, 0)$ infeasible

We aim to balance PRC $\sum_{f\in S} r[f]\cdot\bar{c}[f]$, cost induced by writing/reading checkpoints (memory traffic) $\tilde{c}\cdot\sum_{f\in S} r[f]$ and implementation effort due to number of checkpointed routines $\hat{c}\cdot\sum_{f\in S}\hat{r}[f]$

$$c(\mathsf{R}) = \sum_{f\in S} r[f]\cdot\bar{c}[f] + \tilde{c}\cdot\sum_{f\in S} r[f] + \hat{c}\cdot\sum_{f\in S}\hat{r}[f],$$

where $\hat{r}[f]\in\{0,1\}$ vanishes unless at least one call of f is checkpointed, $\bar{c}[f]$ denotes the PRC of f and $\tilde{c}$ and $\hat{c}$ are used to weigh impacts due to memory traffic and implementation effort, respectively.

Mixed Integer Programming Formulation (Constraints)

We define PMR for each subprogram s after execution of its augmented primal

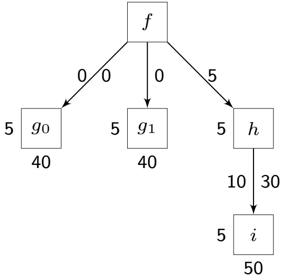
$$\overrightarrow{M}[g] = m[g] + \sum_{h \in \chi[g]} \left[(1 - r[h]) \cdot \overrightarrow{M}[h] + r[h] \cdot \downarrow M[h] \right].$$

and prior to execution of its adjoint, e.g,

$$\overleftarrow{M}[g]_{n[g]} = \overleftarrow{M}[f_i] - \mathbf{m}[f]_i + r[g] \left(\overrightarrow{M}[g] - \downarrow M[g] \right)$$

The maximum PMR is reached prior to execution of adjoint subprograms, hence, $\max_{g \in S} \overleftarrow{M}[g]_{n[g]}$ must not exceed the upper bound M.

- J. Lotz, U.N., S. Mitra: Mixed Integer Programming for Call Tree Reversal, SIAM CSC (2016).



For $M = 125$ we get

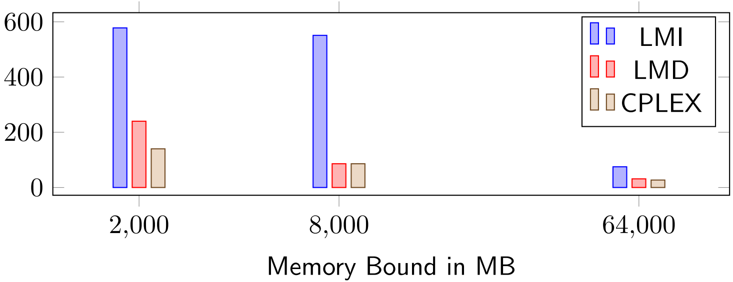
	MIP	LMI	LMD
R	(1,0,0,1)	(0,0,1,0)	(1,1,0,0)
PMR	95	90	125
c(R)	70	120	80

$\bar{c}[f] = 200$, $\bar{c}[h] = 120$, $\bar{c}[i] = 30$, and $\bar{c}[g] = 40$. The cost $\bar{c}[h]$ is chosen to be much higher than the corresponding memory requirement to mimic some time consuming operation in h (e.g, file I/O) without effect on PMR.

Numerical Results (Towards Reality)

We consider the solution of an elliptic boundary value problem with PETSc v3.3. A discrete adjoint version was generated using dco/c++ and AMPI. Bars visualize the overhead due to PRC.

Statistics: 1500 source files, 2717 subprograms instrumented based on dco/c++ count mode, 443k subprogram calls, PMR for R $= \mathbf{0}$ equal to 94GB, runtime of MIP analysis/optimization equal to 40s



▶ J. Lotz, U.N., M. Schanen: Discrete Adjoints of PETSc through dco/c++ and Adjoint MPI, Euro-Par 2013 Parallel Processing, 497-507.

Outline

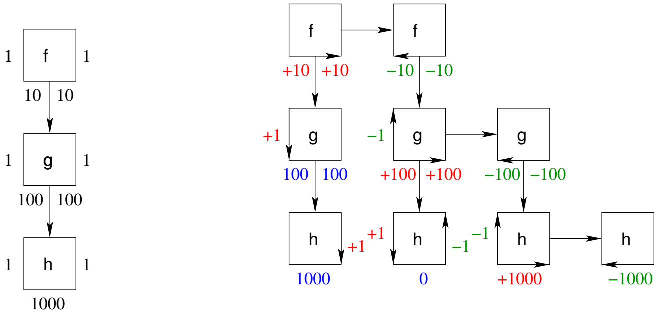
Conclusion

Technical Challenges

- automated instrumentation of large code bases ($\rightarrow$ clang)
- application to call graphs requires conservatism and abstract interpretation for annotation
- combination with static data flow analysis desirable
- definition of corresponding adjoint code design patterns is work in progress

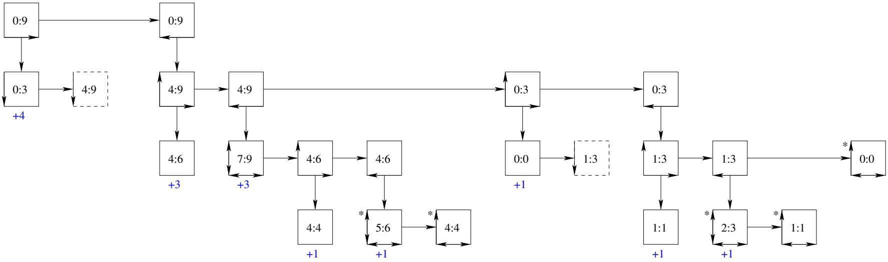
- U.N.: Adjoint Code Design Patterns, AD2016.
- L. Hascoët, U.N., V. Pascual: "To Be Recorded" Analysis in Reverse-Mode Automatic Differentiation, Future Generation Computer Systems 21(8):1401–1417, Elsevier (2005).

Result Checkpointing



MEM=1110, OPS=1200

- U.N.: The Art of Differentiating Computer Programs, SIAM (2012).



- recursive bisection (dynamic programming)
- local recompute all
- repeated accesses to checkpoints

▶ A. Griewank, A. Walther: Algorithm 799: revolve: an implementation of checkpointing for the reverse or adjoint mode of computational differentiation, ACM Transactions on Mathematical Software 26(1):19–45, ACM (2000).