Low-Memory Tour Reversal in Directed Graphs

Uwe Naumann, Viktor Mosenkis, Elmar Peise
LuFG Informatik 12, RWTH Aachen University, Germany

1EuroAD 9, INRIA, Nov. 2009
Contents

- Motivation: Derivative Code Compilers (dcc)
- Motivating Example
- Control Flow Reversal Problem
- Tour Reversal Problem
- Loop Compression
- Offline Algorithm
- Online Algorithm
- Implementation
- Implementation
- Test Results
- Conclusion and Outlook
Motivation: Derivative Code Compilers (dcc)

\[ F : \mathbb{R}^n \to \mathbb{R}^m \Rightarrow \text{Tokens} \Rightarrow \text{AST} \Rightarrow \text{AAST} \Rightarrow F' \]

tangent-linear:

\[ \dot{F}(x, \dot{x}) \equiv \nabla F(x) \cdot \dot{x}, \quad \dot{x} \in \mathbb{R}^n \]

adjoint:

\[ \bar{F}(x, \bar{y}) \equiv \nabla F(x)^T \cdot \bar{y}, \quad \bar{y} \in \mathbb{R}^m \]

second-order tangent-linear:

\[ \tilde{F}(x, \dot{x}, \ddot{x}) \equiv \langle \nabla^2 F(x), \dot{x}, \ddot{x} \rangle, \quad \dot{x}, \ddot{x} \in \mathbb{R}^n \]

second-order adjoint:

\[ \hat{F}(x, \dot{x}, \bar{y}) \equiv \langle \nabla^2 F(x), \dot{x}, \bar{y} \rangle, \quad \dot{x} \in \mathbb{R}^n, \quad \bar{y} \in \mathbb{R}^m \]

... and higher-order tangent-linear and adjoint codes.
AD by source code transformation for subset of C
- started off as teaching tool
- higher-order derivative codes by reapplication
- joint call tree reversal
- additional overloading mode
- main component of AC-SAMMM (see outlook)
- there are others$^2$ ...

$^2$Tapenade, TAC, ADIC
Case Study (Perhaps Later ...)

Use \texttt{dcc} to generate

- TLC
- ADC
- SOTLC
- SOADC
  - FoR
  - RoF
  - RoR
- TOTLC (FoFoR)
- TOADC (FoFoR)

for

\[ y = f(x) = \prod_{i=0}^{n-1} x_i \quad \text{(Speelpenning)} \]
Motivating Example

\[ F : \mathbb{R}^n \rightarrow \mathbb{R} \quad y = F(x) = \prod_{i=0}^{n-1} x_i \]

```c
void F(int n, float *x, float &y) {
    int i = 0;
    while (i < n) {
        if (i == 1)
            y = x[i];
        else
            y = y * x[i];
        i = i + 1;
    }
}
```

E.g. \( n = 3 \)

```c
y = x[0];
y = y * x[1];
y = y * x[2];
xa[2] = y * ya;
ya = x[2] * ya;
xa[1] = y * ya;
ya = x[1] * ya;
xa[0] = ya;
```
Motivating Example (cont'd)

```c
void Fa(int n, float *x, float* xa, float &y, float ya)
{
    int i = 0;
    while (i < n) {
        if (i == 1)
            { push(cs, 1);
              push(fds, y);
              y = x[i];
            }
        else
            { push(cs, 2);
              push(fds, y);
              y = y * x[i];
            }
        push(cs, 3);
        push(ids, i);
        i = i + 1;
    }
    int bb;
    while (pop(cs, bb))
        switch (bb) {
            case 1: { y = pop(fds);
                       xa[i] = ya;
                       break; }
            case 2: { y = pop(fds);
                       xa[i] = y * ya;
                       ya = x[i] * ya;
                       break; }
            case 3: { i = pop(ids);
                       break; }
        }
}
```
Motivating Example (cont’d)

```c
int i = 0;
while (i < n) {
    if (i == 1)
        { push(cs, 1);
          push(fds, y);
          y = x[i];
        }
    else
        { push(cs, 2);
          push(fds, y);
          y = y * x[i];
        }
    push(cs, 3);
    push(ids, i);
    i = i + 1;
}
```

control stack (cs)

1, 2, 3, 2, 3, 2, 3, ... , 2, 3

float data stack (fds)

\( y^0, y^1, y^2, y^3, y^4, ... \)

integer data stack (ids)

0, 1, 2, 3, 4, ...

or

0, 1, 1, 1, 1, ...

if increments are stored
**Control Flow Reversal (CFR)**

**Given:** \( y = F(x) \) as conditional SAC:

\[
\text{for } (j = n; \ j < n + p + m; \ j++)
\]
\[
\text{if } (v_{j-1} > 0)
\]
\[
\quad v_j = \varphi_{j-n}(v_i)_{i<j}
\]

\( v_{-1} = x_0; \ v_0 = x_1 \)
\[
\text{if } (v_0 > 0) \quad v_1 = v_{-1} \cdot v_0 \text{ else } ...
\]
\[
\text{if } (v_1 > 0) \quad v_2 = \sin(v_1) \text{ else } ...
\]
\[
\text{if } (v_2 > 0) \quad v_3 = v_{-1} \cdot v_2 \text{ else } ...
\]
\[
\text{if } (v_3 > 0) \quad v_4 = v_3/v_0 \text{ else } ...
\]
\[
\text{if } (v_4 > 0) \quad v_5 = \cos(v_3) \text{ else } ...
\]

\( x_0 = v_4; \ x_1 = v_5 \)

**Wanted:** predecessors of all vertices in reverse order requiring evaluation of conditions in reverse order, i.e. the values of \( v_4, v_3, v_2, v_1, v_0 \)
CFR is NP-complete

... by reduction from **DAG Reversal**

... from **Fixed Cost DAG Reversal**

▶ unit cost for storage and recomputation
▶ fixed total cost $|V|$
▶ minimize storage

... from **Vertex Cover** enabling recomputation at unit cost from stored arguments.

**Tour Reversal**

1: ...  
   \[ \text{for } (i = 0; i < 2n, i++) \{ \]  
2: \[ \text{if } (i \% 2) \{ ... \} \]  
3: \[ \text{else } \{ ... \} \]  
}  
4: ...  

... by enumeration of basic blocks
1, 2, 3, ..., 2, 3, 4

... by counting loop traversals and flagging branches
0, 1, ..., 0, 1, 2n (reducible control flow only!)

**Tour Reversal Problem**  
Use minimal memory to reverse a tour \((i_1, i_2, \ldots, i_l)\) in a directed graph \(G = (V, E)\).
Loop Compression

Enumerate Basic Blocks

- uncompressed tour: $1, 2, 3, \ldots, 2, 3, 4$
  $2n + 2$ integers
- compressed tour: $1, 2, 3, [n, 2], 4$
  $6$ integers

Count loops and flag branches

- uncompressed tour: $0, 1, 0, 1, 0, 1, \ldots, 0, 1, 2n$
  $2n + 1$ integers
- compressed tour: $0, 1, [n, 2], 2n$
  $5$ integers

```plaintext
1:  ...  
   for (i = 0; i < 2n, i++) {
2:    if (i % 2) {...}
3:    else {...}
   }
4:  ...  
```
Offline Algorithm

Dynamic programming compresses according to

\[ f(d_{i,j}) = \begin{cases} 1 & i = j \\ \min_{i \leq s < j} f(d_{i,s} \circ d_{s+1,j}) & \text{otherwise} \end{cases} \]

where \( d_{i,j} \) is the optimally compressed subtour from \( i \)-th to \( j \)-th index and

\[ f(d_{i,j}) := |N \cap V| \quad \text{(loop elements not counted)} \]

Problems

- need to see whole tour in order to find an optimal solution
- slow \((O(n^3)\) for tour of length \( n \))
Online Algorithm

1, 2, 3, ..., 2, 3, 4 becomes

push(1): 1
push(2): 1, 2
push(3): 1, 2, 3
push(2): 1, 2, 3, 2
push(3): 1, 2, 3, [2, 2]
push(2): 1, 2, 3, [2, 2], 2
push(3): 1, 2, 3, [3, 2]
...
last push: 1, 2, 3, [n, 2], 4
pop: 1, 2, 3, [n, 2]
pop: 1, 2, 3, [n − 1, 2], 2

- push
  - pushes element to top of stack
  - compresses the loops on window of predefined size
- pop
  - unrolls one instance of last loop if needed
  - returns element from top of stack and removes it from stack
## Test Results

<table>
<thead>
<tr>
<th>Problem</th>
<th>uncompr. stack size</th>
<th>compr. stack size</th>
<th>compr. rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>burger</td>
<td>3432 MB</td>
<td>144 Byte</td>
<td>25013889</td>
</tr>
<tr>
<td>bratu</td>
<td>4576 MB</td>
<td>168 Byte</td>
<td>28572380</td>
</tr>
<tr>
<td>ljc</td>
<td>3692 MB</td>
<td>704 KByte</td>
<td>5500</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Problem</th>
<th>Window size</th>
<th>Runtime rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>burger</td>
<td>51</td>
<td>≈ 2.5</td>
</tr>
<tr>
<td>bratu</td>
<td>51</td>
<td>≈ 2.8</td>
</tr>
<tr>
<td>ljc</td>
<td>51</td>
<td>≈ 5.5</td>
</tr>
<tr>
<td>ljc</td>
<td>11</td>
<td>≈ 3.3</td>
</tr>
</tbody>
</table>
Implementation

- **C++ front-end**
  - distribution: `ccistack.{hpp,cpp}`
  - class `ccistack`
  - `void ccistack::push(int)` and `int ccistack::pop()`

- **C front-end**
  - distribution: `cistack.{h,c}`
  - `void init (struct cistack *k, int MaxStackSize, int WindowSize)` to allocate
  - `void finalize (struct cistack *k)` to free
  - `void push (struct cistack *k, int i)`
  - `int pop (struct cistack *k)` and `int empty (struct cistack *k)`

- **Fortran front-end**
  - uses C front-end
#include <fstream>
using namespace std;
#include "cistack.h"

int main(int argc, char* argv[]) {
    ifstream infile(argv[1]);
    ofstream outfile(argv[2]);
    cistack cs(100, 21); int v;
    while (!infile.eof()) {
        infile >> v;
        if (!infile.eof()) cs.push(v);
    }
    while (!cs.empty())
        outfile << cs.pop() << endl;
    return 0;
}
#include "cistack.h"

int main(int argc, char* argv[]) {
    struct cistack cs;
    FILE* infile=fopen(argv[1],"r+");
    FILE* outfile=fopen(argv[2],"w+");
    init(&cs,100,4);
    int v;
    while ( fscanf(infile,"%i", &v) > 0 ) push(&cs,v);
    while (!empty()) fprintf(outfile,"%i\n",pop(&cs));
    finalize(&cs);
    fclose(csout); fclose(outfile); fclose(infile);
}
Conclusion (I)

- powerful control flow reversal tool ready to use for developers of adjoint compiler technology and/or debugging/profiling tools
- paper (almost) ready for submission

Outlook

- compression of integer data stack
- compression of float data stack
Conclusion (II)

You need a (adjoint) derivative code compiler if

- finite differences cannot be trusted
- finite differences or exact forward sensitivities are too expensive
- you are unable to build and solve the adjoint system manually

You need to invest

\[ 3, 6, 18, 36 \] (wo)man months for sustained runtime of adjoint runtime of original simulation of

\[ 50, 20, < 10, < 4 \]
Further Ongoing Activities at STCE

- **CompAD-III** (J. Riehme and D. Gendler, with UHerts and NAG)
- Adjoint error correction in **ICON** (J. Riehme and K. Leppkes, with MPI-M)
- Model reliability analysis in **Sisyphe/Telemac** (J. Riehme, with BAW)
- Data assimilation in **JURASSIC** (E. Varnik and M. Förster, with ICG-I at FZJ)
- dcc and the AaChen platform for Structured Automatic Manipulation of Mathematical Models (**AC-SAMMM**) (M. Förster and B. Gendler, with AVT)
- Toward adjoint **MPI** (M. Schanen, with ANL and INRIA)
- Pushing **TBR** (with ANL and INRIA)
- Call tree reversal (H. Lakhdar and X. Jin)
Further Ongoing Activities at STCE (cont.)

- Elimination techniques in linearized DAGs (V. Mosenkis)
- Higher-order adjoints for uncertainty quantification (M. Beckers, with UHerts)
- Adjoint subgradients for McCormick relaxations (M. Beckers, V. Mosenkis, and M. Maier, with MechEng at MIT)
- What color is the non-constant part of your Jacobian? (E. Varnik and L. Razik)
- Shared-memory parallelism in tape-based adjoints (K. Leppkes and J. Riehme)

- Submitted: Hybrid AD for C/C++ (ADOL-C + dcc, D. Gendler, with UPaderborn)
The AaChen platform for Structured Automatic Manipulation of Mathematical Models

Standard (AC–SAMMM)

- Custom Derivative Models
- Mathematical Submodels in C– (imperative)
- Mathematical Model in C++ (descriptive)

Problem

- Mathematical Model in
  - Modelica
  - gPROMS
  - ...

Diversity (The World)

Solution

- Mexa
- DyOS

Model refinement

- tailored AD
- symbolic manipulation
- AD

i1

i2

i3
Global Optimization using McCormick Relaxations

(adjoint) subgradients by NAG Fortran compiler based on

(C. Corbett, M. Beckers, V. Mosenkis, M. Maier)
e.g. \( \min_{\mu_x} (\mu_y + \sqrt{\sigma^2_y}) \) where \( y = F(x) \) and w.l.o.g. \( F : \mathbb{R} \rightarrow \mathbb{R} \) using

- approximate mean

\[
\mu_y = F(\mu_x) + \frac{F''(\mu_x)}{2} \cdot \sigma_x^2
\]

- approximate variance

\[
\sigma_y^2 = F'(\mu_x)^2 \sigma_x^2 + F'(\mu_x) F''(\mu_x) S_x \sigma_x^3
\]
\[
+ \frac{1}{4} (F''(\mu_x))^2 (K_x - 1) \sigma_x^4
\]

for given initial mean \( \mu_x \), variance \( \sigma_x^2 \), skewness \( S_x \), and kurtosis \( K_x \) of \( x \in \mathbb{R} \). Second-order method (e.g. Newton) requires derivatives up to fourth order. (M. Beckers)
Adjoint (Total Model) Error Correction

... in ICON GCM: ICOsahedral Non-hydrostatic General Circulation Model developed by Max Planck Institute for Meteorology and Deutscher Wetterdienst.

(Discrete) Adjoint by NAG Fortran Compiler. (J. Riehme)

(References to) theoretical foundations e.g. in


Adjoint Error Correction (Linear Case)

Given: \( x_f = F(A, x_s) \) after solution of \( A \cdot x_f = x_s \) and value of linear objective \( J = J(g, x_f) = \langle g, x_f \rangle \).

Wanted: corrected objective \( J = \langle g, x_f^c \rangle - \langle \bar{x}_s^c, A \cdot x_f^c - x_s \rangle \).

Solution:

\[
A^T \cdot \bar{x}_s = \bar{x}_f = g
\]

NOT using discrete adjoint code

\[
x_f = A^{-1} \cdot x_s
J = \langle g, x_f \rangle 
\bar{x}_f = g \cdot \bar{J} = g
\bar{x}_s = A^{-T} \cdot \bar{x}_f
\]

produced by derivative code compiler.
Adjoint Error Correction (Nonlinear Case)

Given: \( x_f = F(A, x_s) \) after solution of \( N(x) = 0 \) and nonlinear objective \( J = J(x_f) \) with gradient \( g \equiv \frac{\partial J}{\partial x_f} \).

Wanted: corrected objective \( J = J(x^c_f) - < \bar{x}^c_s, N(x^c_f) > \).

Solution to the adjoint system preferably using discrete adjoint code

\[
\begin{align*}
  x_f &= F(x_s) \\
  J &= J(x_f) \\
  \bar{x}_f &= g \cdot \bar{J} = g \\
  \bar{x}_s &= F'(x_s)^T \cdot \bar{x}_f
\end{align*}
\]

produced by derivative code compiler.
Objective: Potential Energy wrt. Height Field

rotating earth; all water except at poles; objective: potential energy somewhere in middle of the Sahara ... (F. Rauser, P. Korn)

Test Case 3: Unsteady Solid Body Rotation from

Laeuter: *Unsteady analytical solutions of the spherical shallow water equations*. Journal of Computational Physics, 2005