Automatic Source Transformation and Performance Evaluation for Optimized Utilization of the Matlab Runtime System

Alexander Hück
Contact: alexander.hueck@sc.tu-darmstadt.de

15th European Workshop on Automatic Differentiation
ADiMat

- **ADiMat**
  - Automatic **Differentiation for Matlab** [1]
    - First Order Derivatives: Forward & Reverse Mode
    - Second Order Derivatives: Hessian
  - Current developer Johannes Willkomm
  - Homepage:
    - [http://www.adimat.de](http://www.adimat.de)
  - Interactive web interface and download:
    - [https://adimat.sc.informatik.tu-darmstadt.de/](https://adimat.sc.informatik.tu-darmstadt.de/)
ADiMat Overview

ADiMat uses **source transformation & operator overloading**

- Generation of XML-based AST, transformations with XSLT

- **Source Transformation** for *code differentiation*
  - Optimizations of code
  - Activity analysis: Which variables are affected by the differentiation?

- **Operator overloading** for *vectorized* derivative evaluation (**DE**)
  - Derivative container to hold multiple derivative directions

```plaintext
class AD
    ndd (number)  deriv (array)
```

![Diagram](Vector Mode)
ADiMat First Order Differentiation Mode (AD)

\[ z = F(a, b) \]
\[ z = a \times b; \]

**Derivative classes**
- Encapsulate multiple directional derivatives
- Operator overloading

**Functions only**
- No derivative class
- d_* variable is double array

\[ \begin{align*}
\text{function } [g_z, z] &= g_F(g_a, a, g_b, b) \\
g_z &= g_a \times b + a \times g_b; \\
z &= a \times b;
\end{align*} \]

\[ \begin{align*}
\text{function } [d_z, z] &= d_F(d_a, a, d_b, b) \\
d_z &= \text{opdiff_emult}(d_a, a, d_b, b); \\
z &= a \times b;
\end{align*} \]
Derivative class \textit{adderiv}

\begin{verbatim}
function [g_z, z] = g_F(g_a, a, g_b, b)
g_z = g_a * b + a * g_b;
z = a * b;
\end{verbatim}

\textbf{class: adderiv}

\begin{itemize}
  \item \textbf{ndd (number)}
  \item \textbf{derivs (cell)}
\end{itemize}
Performance Weakness with Derivative Class in Forward Mode

Motivation
- Slow derivative calculation with derivative class(es) of ADiMat

Where?
- Our initial assumption: Operator Overloading/Property Access

Solution?
- Remove operator overloading

function \[g_z, z\] = g_F(g_a, a, g_b, b)
\[g_z = g_a \times b + a \times g_b;\]
\[z = a \times b;\]
Idea to Improve Performance

- Additional Source Transformation Step

\[ \text{Derivative} \rightarrow \text{Transformation 'removes' class} \rightarrow \text{Derivative without} \]
  
  - Operator Overloading
  - Method invocations

- Replace derivative class by function calls
Transformation: Derivative Function

- 'Front-end' transformation
  - Derivative function $g_F$
    - Remove Operator Overloading
      - Arithmetic Operators, Indexing etc.
    - Method calls

```matlab
function [g_z, z] = g_F(g_a, a, g_b, b)
g_z = g_a * b + a * g_b;
...
```

```matlab
function [g_z, z] = g_F(g_a, a, g_b, b)
g_z = ad_plus(ad_mtimes(g_a, b), ad_mtimes(a, g_b));
...
```

- $ad\_plus$, $ad\_mtimes$
  - Originally operator methods $plus/mtimes$
Transformation: Derivative Class

- **'Back-end' transformation**
  - Functionality/operators is provided by class
    - Need same functionality
  - **Transformation**
    1. Class properties are encapsulated in *struct*
    2. Methods converted to a set of functions working on new *struct*

```
class: adderiv
    ndd (number)
    derivs (cell)
    plus (+)
    mtimes (*)
    ...

struct
    ndd (number)
    derivs (cell)
    ad_plus
    ad_mtimes
    ...
```
Removal of Operator Overloading?

- What is the usefulness of removing operator overloading?

- Benchmarks are required
  - **General Matlab Benchmark**
    - Many (micro) benchmarks relevant to object oriented programming vs. functions
  - Gist of it
    - Cell/Struct access should be avoided with Matlab class system
      - `adderiv` uses cell to store derivatives
      - Functions profit much more from JIT than methods in Matlab

- **ADiMat derivative class** (`adderiv`)
  - Compare to function based derivative calculation
ADiMat/adderiv Benchmark

- Polynomial
  
  \[ z = \sum_{i=0}^{n} c_i \cdot x^i \]

- Manually transformed
  - Only function calls
  - Also classdef version of adderiv

- Results
  - Function vs. adderiv class
    - major speedup (4+)
  - Profiling shows
    - Substantial overhead with cell access in operators (+/*)
Automated Source Transformation Process

Derivative → Transformation 'removes' class → Derivative without
- Operator Overloading
- Method invocations

- ADiMat generates XML-based AST
  - Rule-based transformation with 17 XSLT stylesheets
  - Functional programming style

- AST modification
  - Back-end and Front-end transformation have different requirements
Detailed View of the Source Transformation Process

- AST normalization
  - 3 stylesheets

- Mark Elements

  - Identify expressions with adderiv
    - 2 stylesheets
  - Back-end: 5 additional stylesheets
  - Identified expressions transformed to function calls
    - 4 stylesheets

- Modify Elements

- Postprocessing

  - Rename operations
    - 3 stylesheets
Derivative Function: Identification of Class Usage

- Matlab has no type information during 'compile time'

- Rule for static type inference necessary
  - Variable with prefix ('g_')
    - Derivative class
    - else: assumed to be double

- Prefix identification
  - Works for automatically generated derivatives of ADiMat
  - Can be generalized to arbitrary prefixes
General rules for the identification

- Expression is of type AD-class, if
  - A leaf of the current (AST) subtree starts with the prefix (`g_`)
  - Path to the leaf has no call returning different type (`double`)
    - Method call – double or AD-class?
      - Lookup Table necessary

- Expressions with at least one type AD-class need to be rewritten

AST: \( g_a \ast b \)  
AST: \( a + g_a \ast b \)  
AST: \( a + \text{size}(g_a) \)
Benchmark: Burgers' Equation

- 1D PDE Solver
  \[ \frac{\partial u}{\partial t} + \frac{1}{2} \frac{\partial u^2}{\partial x} = 0 \]
  \( u(x, 0) = u_0(x) \)

- Directional derivatives
  \( ndd = \{50, 100, 200\} \)

- 3 ADiMat classes
  - Speedup
    - adderiv
      - \(~1.70\)
    - arrdercont/madderiv
      - <1.10
  - Use Matrix data structure for derivative storage

![Speedup After The Transformation (R2012a)](chart)

6 functions, 57 statements, 169 LOC
Summary

- **Class system of Matlab**
  - Avoid **cell** or **struct** properties
    - Especially using `classdef` syntax
  - Method invocations slightly more expensive

- **Source Transformation**
  - Automatic for derived functions *(Front-end)*
    - (semi) automatic for derivative class itself *(Back-end)*
  - Performance gains
    - Very good for cell based `adderiv` class
    - Matrix based classes profit less
      - Arrdercont ~10% gain
Reference

"Combining Source Transformation and Operator Overloading Techniques to Compute Derivatives for MATLAB Programs"