McGill University
School of Computer Science
COMP 621

Optimizing a robust AspectMATLAB front-end

Report No. 2016-09

Hongji Chen

November 1, 2016

Contents

1 Introduction and Motivation

MATLAB is a widely used dynamic scientific programming language, and AspectMATLAB aims to introduce Aspect-oriented programming (AOP) into MATLAB. AOP allow users to match program points with pointcuts (patterns in AspectMATLAB) and apply user-defined pointcut advice (actions in AspectMATLAB). One major possible application of AspectMATLAB is programming profiling. Although current MATLAB has integrated with built-in profile features but is relative limited and difficult to customize. Like AspectJ and many other AOP implementation AspectMALAB provide user to capture program points using patterns and retrieve runtime information using content exposure selectors. But unlike most of AOP implementation, AspectMATLAB allow users to match program points not only by type and scope information but also by shape information. However, the dynamic features that are utilized by MATLAB pose several challenges for source code transformation and weaving. In order to match the user defined patterns, AspectMATLAB needs to insert runtime checks to collect the type and shape information about the program points, and such runtime checks are proven to be costly. This project is aimed to first create a robust transforming framework for AspectMATLAB and then implemented several static analysis to improve the performance of the weaved MATLAB source code.

2 Background and Related Work

The project is built on the success of Aslam's AspectMatlab compiler[5] , Bodzay's AspectMatlab++ compiler [1] , and summer research on "Creating a robust front-end for AspectMATLAB".

Aslam's AspectMatlab Compiler[2]

This is the first version of AspectMatlab compiler, it lays the foundation for later versions of AspectMatlab compiler. In the paper, eight patterns are purposed.

Call captures all call to function or scripts
Execution captures the execution of function bodies
Get captures array accesses
Set captures array sets
Loop captures execution of all loops
LoopHead captures the header of the loop
LoopBody captures the body of the loop
Within restricts the scope of matching

For each join point, AspectMatlab offers three type of weaving, **before**, **after** and **around**. The transforming strategy used by this version of AsepctMatlab compiler is relatively straight forward, and in most of the cases, optimization can be applied to improve the execution performance.

Bodzay's AspectMatlab++ compiler[3]

AspectMatlab++ compiler is built on the success of the original AspectMatlab compiler, and extended with **annotation**, **type** and **dimension** pattern. The **annotation** pattern allow user to define special formatted comment, and insert action calls at comment position. The **type** and **dimension** pattern behave similar with **within** pattern, but instead of restricting the matching by scope, it check the type and shape information of the join point. Despite the enhancement in the patterns, the AspectMatlab++ compiler utilizes the function inlining technique. Unlike the original AspectMatlab compiler which insert a function call at every join point, the AspectMatlab++ compiler will attempt to insert the action directly into the weaved source code. The experiment result shows that the function inlining can effectively boost the performance of the weaved code.

Summer research

However, in both existed version of AspectMatlab compiler, the pattern does not checked before weaved. Thus, it is possible that it accept some semantically invalid pattern, which could lead to unpredictable result for the generated weaved code. Thus, a more clear grammar and a set of semantic validation rules has been developed, such that the new compiler will reject the semantically invalid pattern before proceeding further in the compilation stages.

The transform strategy used in the existed version of AspectMatlab and AspectMatlab++ compiler has been out dated, and some weaved code cannot be correctly interpreted in the latest version of MATLAB (2016a). Thus, another part of the summer research is to develop a new set of transforming rules that are compatible with the latest version of MATLAB. But currently, the transforming strategy is straight forward and have no optimization applied. Thus, this project is aim to optimize the weaved source code generated by the new AspectMATLAB compiler.

3 Specific Problem Statement

In order to correctly transform the source code for action weaving with minimum development complexity, the transformer is designed to be straight forward, and it opens the opportunity for further optimization. The project is aimed to implement the following optimization of the generated weaved code:

1. data-flow analysis discussed in the lecture
2. a analysis to extract and minimize the number of dynamic checks required
3. a procedure to determine if a action can be directly inlined into the weaved source code

4 Solution Strategy

One approach to extract and minimize the number of dynamic checks is to design an analysis similar with common expression analysis that eliminate redundant dynamic checks. Another approach is to purpose a type and shape approximation analysis on AST, and then the compiler can statically decide whether a program point is matched by a pattern.

In order to determine if an action can be directly inlined into the weaved source code, the first step is to collects a set of conditions where the action must satisfied, and then purpose a set of analysis procedure to verify the conditions.

5 Experimental Framework

The new AsepctMATLAB is built on new McLab framework that modified during the summe. Also a new parser[4], developed by Samuel Suffos during the summer will also be used in the new AspectMATLAB compiler, this will solve most of the parsing errors in the existed AspectMATLAB compiler. The testing and benchmarking will be performed on the latest MATLAB environment (2016b), as the existed AspectMATLAB compiler is no longer compatible with the new version of MATLAB. Further, the new AspectMATLAB compiler will use Wu-wei benchmark toolkits for benchmarking and profiling.

6 Schedule of Activities

The first part of the project is to finish the implementation of the transforming framework. The design of the framework and transforming rules have been developed over the summer, and hopefully, the transforming framework can be finished shortly. The second part of the project is to implement several analysis discussed within the lectures, including constant propagation, and common expression analysis. The third part of the project to modify the common expression analysis such that it can correctly extract the dynamic query

expression inserted by the transform framework. The last part of the project is to develop the action inlining strategy. The first part and the second part is expected to be finished within short period of time, and the project is focus on the third and the fourth part.

7 Expected Results and Evaluation Method

The new AspectMATLAB compiler should be more robust than the existed one, and it can handle all the patterns correctly. Also, the new AspectMATLAB should perform relatively better than the existed version. One method to evaluate the result is to insert empty action into the source code, and measure the execution time with the benchmark toolkits. This will make sure that the change in execution time directly reflects the performance of the compiler.

References

[1] Laurie Hendren Andrew Bodzay. Aspectmatlab++: Annotations, types, and aspects for scientists. *MODULARITY'15*, 2015.

[2] Toheed Aslam. Aspectmatlab: An aspect-oriented scientific programming language. Master's thesis, McGill University, 2010.

[3] Andrew Bodzay. Aspectmatlab++: Developing an aspect-oriented language for scientists. Master's thesis, McGill University, 2014.

[4] Samuel Suffos. mclab-parser.

[5] Anton Dubrau Toheed Aslam, Jesse Doherty and Laurie Hendren. Aspectmatlab: An aspect-oriented scientific programming language. *AOSD'10*, 2010.