

# COMP 621 Program Analysis and Transformations Assignment #1

Profiling and Traditional Flow Analysis

Due: Thursday, February 2 – beginning of class

## Overview:

The purpose of this assignment is two-fold. The first purpose is to participate in the collection or creation of benchmarks that will be used later in the course for testing our optimization, parallelization, and analysis methods. The second purpose is to give you some experience with stating and solving dataflow problems.

## Question 1: *Benchmark Programs*

Benchmark programs form an important part of the development of new optimization, analysis, and parallelization techniques. Without some sort of quantitative measurement, we cannot really evaluate how well the new techniques work. In order to build up a comprehensive benchmark suite, each person will provide one program that can be incorporated into the suite.

This year, you will be using MATLAB and AspectMatlab. An introduction to these languages will be given in class on January 19. Further documentation can be found at:

- [http://www.mathworks.com/help/techdoc/matlab\\_prog/bqjgwp9.html](http://www.mathworks.com/help/techdoc/matlab_prog/bqjgwp9.html), for the documentation on Matlab basics.
- <http://www.sable.mcgill.ca/mclab/aspectmatlab/> for documentation and source code for AspectMatlab and some example aspects.
- <http://www.gnu.org/software/octave/docs.html> for documentation for Octave.

If you get stuck using the tools, please consult with the TAs and share your expertise with other class mates.

When choosing your benchmark, consider the following:

- The benchmark should compute something interesting. Real applications from different branches of Science and/or Engineering would be excellent. Applications solving toy CS problems are **not** interesting.
- You can write your own benchmark or you can use code that someone else has written, but we need to have permission for all of us to use it as a benchmark. It is even better if we are allowed to distribute it as part of our Matlab benchmark set. Something with an open source license is ideal.

- The benchmark should not rely on libraries that are not written in publicly-available MATLAB. We need to be able to access and optimize all of the code.
- The benchmark should spend most of its time computing (and not doing I/O).
- The benchmark should not use dynamic features that make the program hard to analyze. For example, it should not make use of functions like `evalin`.
- The benchmark should have deterministic output that can be checked for correctness. We need to be able to run optimized versions of the program and automatically check that the output matches the output of the original unoptimized version. The benchmark should not have a GUI interface that requires human interaction to run, nor require any terminal input (we need to be able to run the benchmarks from a script). If you need external files for input (i.e. if you compute on some large data set), please use ASCII files which contain one 2d array each, together with the matlab command `load`. Do not use `.mat` files, because it may make programs hard to analyze.
- Your benchmark needs to run for a significant amount of time. An execution time of at least 60 seconds using the production Matlab on a fast machine is recommended. If at all possible, provide several inputs to correspond to a short (around 10 seconds), medium (around 60 seconds) and long (around 5 minutes or longer) run-time.

## Where to look for a benchmark

There are many places to look. Here are a few possibilities:

- a research group at McGill using Matlab.
- <http://www.mathworks.com/matlabcentral/fileexchange/>
- [http://web.mst.edu/~gosavia/mrrl\\_website.html](http://web.mst.edu/~gosavia/mrrl_website.html)
- [https://github.com/languages/Matlab/most\\_watched](https://github.com/languages/Matlab/most_watched)

## What to hand in

- (a) Give a brief description of your benchmark which highlights the important or interesting characteristics of the program. Describe it both from the scientist/engineering point of view (why this is an interesting/important problem) and from the compiler optimizer point of view (what features make it interesting to optimize).
- (b) Describe how to run the benchmark, and describe the required formats for inputs and which different inputs you have provided. A template, `benchmark_setup.template`, is provided to help you in this regard. Organize your benchmark as outlined in the template.

- (c) Run your benchmark using the following two scenarios:
- Using Mathworks production version of MATLAB
  - Using Octave

For each scenario run the benchmark 10 times, throw out the slowest and fastest run and then report run-times including the min, max, average and standard deviation of the remaining 8 runs. You will probably want to write a script to do this task.

Report the versions of MATLAB and Octave that you used.

Report the architecture and OS on which you did the experiments.

- (d) Discuss the results you obtained in part (c).
- (e) Define an aspect using AspectMatlab which can be used to count or profile some feature of your benchmark. This could be a very simple aspect which counts the total number of array accesses and/or the total number of function calls. However, it could also be something more interesting. Give the source code for your aspect and explain its purpose.
- (f) Compile your original benchmark, along with your aspect using the AspectMatlab compiler. Report on any problems which could be improved by better static checks or better error messages.
- (g) Run the compiled/woven Matlab code using both the Mathworks MATLAB and Octave, reporting on run-times as in (c).
- (h) Discuss the results of (g). Did introducing your aspect slow down the program substantially? If so, why? Did your aspect provide you with interesting profiling results. If so, what?
- (i) Starting with the original benchmark code, profile the execution using the Matlab profile function as documented at <http://www.mathworks.com/help/techdoc/ref/profile.html>, or the IDE profiler as documented [http://www.mathworks.com/help/techdoc/learn\\_matlab/f1-26972.html#f1-30972](http://www.mathworks.com/help/techdoc/learn_matlab/f1-26972.html#f1-30972).

Summarize your profile results. Based on this profile discuss which are the most important parts of the program to optimize?

- (j) Based on your observations from (i), find some part of your program that you can attempt to optimize by hand (i.e. rewrite the Matlab code for that part to be faster). Attempt a rewriting and explain what you attempted to do. Using your hand-optimized code, rerun your timings for Mathworks MATLAB and Octave and report on the results and the speedup/slowdown as compared to the unoptimized version. Discuss your results.

## What to hand in

- A file called `yourLastName.tar.gz` file containing a directory called `yourLastName/`. Of course, replace “`yourLastName`” with with your real last name!
  - Inside this directory you should have a sub-directory called `orig/` which contains your source code (the original version, before you improved it), all the inputs, all the outputs, a README that describes how to compile and run your benchmark.
  - Also include a sub-directory called `aspect/` that contains the source code for your aspect, also with a README describing your aspect.
  - Finally, include a sub-directory called `improved/` that contains the source code of your hand-optimized benchmark, along with a README summarizing the differences between this version and the `orig/` version.

Thus, your file should be something like `hendren.tar.gz` and when unzipped it should create a directory structure like:

```
/hendren
  /orig
    README
    <otherfiles>
  /aspect
    README
    <otherfiles>
  /improved
    README
    <otherfiles>
```

Send this file as an attachment to `hendren@cs.mcgill.ca`. Indicate on the title of the message `cs621 Matlab Benchmark`. Please make sure to send *just one* copy but if you must update what you’ve already handed in, indicate this in the subject line with the word **UPDATE**.

- A hardcopy of your answers to all parts of this question. Please try to print in a paper-saving fashion. (You do not need to print your benchmark code)

## Question 2: *Putting analysis to practice*

Assume the following simplified C-like language.

```

<stmt_seq> ::= <empty_stmt> | <stmt> <stmt_seq>

<stmt> ::= <basic_stmt> ; |
          if ( <expr> ) <stmt> else <stmt> |
          while ( <expr> ) do <stmt> |
          { <stmt_seq> }

<basic_stmt> ::= <id> = <id_const> <bin_op> <id_const> |
                 <id> = <id_const> |
                 <id> = read() |
                 write(<id_const>) |
                 break

<expr> ::= <id_const> <relop> <id_const>

<id_const> ::= <id> | <const>

<bin_op> ::= + | * | - | / | mod

<rel_op> ::= < | > | <= | >= | == | !=

```

Use the following example program.

```

1 i = 1;
2 j = 1;
3 n = read();
4 sum = 0;
5 prod = 1;
6 while (i <= n)
7     { b = i mod 2;
8       if (b == 0)
9         sum = sum + i;
10      else
11        prod = prod * j;
12      if (sum == prod)
13        break;
14      i = i + 1;
15      write(i);
16    }
17 b = n mod 2;
18 if (b == 0)

```

```
19  write(sum);
20  write(prod);
```

- (a) Draw a typical abstract syntax tree (AST) that could be used to represent the statement sequence from lines 6-16 of the example program.
- (b) Draw a typical control flow graph (CFG) that could be used to represent the entire program fragment, putting the statements in basic blocks where possible.
- (c) Assuming that we can represent a definition by a pair  $(varname, lineno)$ , (i.e. the definition at line 3 would be  $(n,3)$ ), what are the set of definitions that **may** reach the input of lines 6, 7, 15 and 18 in the example program? As an example, the set of definitions reaching the input of line 2 is  $\{ (i,1) \}$ .
- (d) Give the data flow analysis for the **must** reaching definition problem. In this problem a definition reaches, only if it reaches along **all** paths. Follow all the steps outlined in class for defining the problem. Show the result of your must reaching definition analysis for lines 6, 7, 14, 16 and 18.
- (e) What variables are live at the input of lines 20, 18, 14, 9, 7, 5, 3 and 1 in the example program?
- (f) Some compilers warn the programmer when a use of a variable may not have a valid definition. In Java, the language definition says that each use must have a valid definition, and it issues a syntax error if there exists some path on which there is no definition of a use.

Formalize this problem for the language given in this problem. Give a flow analysis to solve the problem, and give example programs to show: (i) where your analysis correctly warns the programmer, and (ii) where your analysis warns the programmer unnecessarily (i.e. the analysis is conservative).

### What to hand in

- A hardcopy of your answers to all parts of this question.