

# COMP 621 Suggested Projects - Winter 2009

Due Dates:

- Project descriptions available and signups, Mon February 9 to Fri. Feb 13
- Project proposal due, Friday February 20
- Project update #1 due, Friday March 6
- Project update #2 due, Monday March 23
- Project presentations start: Monday March 30th
- Project reports due, Monday April 20

## 1 Introduction

The purpose of the final project is to give you a chance to focus on a specific problem, and to work out the solution to this problem in detail. Your project should give you the opportunity to review the current literature in your chosen area, to implement either the ideas you read about or to implement your own improvements on those ideas, and to evaluate the effectiveness of the ideas. You are encouraged to try to develop improvements over the current state-of-the-art. Even if your ideas do not turn out better, it is still worth trying (after all, that is what research is all about). It is worth noting that publishing novel work done in this course is not out of the question.

I have listed the various possible project areas below. Students should work individually, but up to two people can choose the same topic (they would produce different solutions).

The projects will be given out on a first-come, first-served basis. You may sign-up for your selected area on the sheet outside of my office, McConnell 228. If you are not usually at McGill, you can send me e-mail and I will sign you up. You are also free to design your own project, or to change a suggested project to more suit your interests. If you wish to modify or design your own project, please contact the instructor to discuss the project.

Your project proposal should be done using the document outline in:

<http://www.sable.mcgill.ca/hendren/621/Docs>

Latex documents are preferred, but not absolutely necessary. In any case, you should follow the overall organization given in <http://www.sable.mcgill.ca/hendren/621/Docs/prop621.tex>.

## 2 Soot

You have already used the Soot framework (<http://www.sable.mcgill.ca/soot>), a set of Java APIs which facilitate the analysis and transformation of Java classfiles. Many of the following projects were designed with the Soot framework explicitly in mind. Soot is freely available (licensed under the GNU Lesser General Public License (LGPL)), and was developed at McGill University by the Sable Research Group. The Soot framework is meant as a shared resource for compiler research. It would be very nice if your projects were robust enough to incorporate into the framework. You should join the Soot mailing lists if you do a Soot project.

abc is the AspectBench Compiler for AspectJ, developed by a team from Oxford, McGill and Aarhus (<http://www.aspectbench.org>). The front-end of the compiler is developed using Polyglot and the back-end uses Soot. Students working on projects relevant to abc should join the abc mailing lists.

## 3 Projects

### Projects proposed by instructor and mentors - available

#### 1. Space Optimization:

Java Virtual machines are being implemented in small devices with restricted memory and caches. In this context it becomes important to reduce the size of the code (rather than the speed of the code). In this project the purpose will be to reduce the size of the bytecode, reduce the number of local variables, and reduce the maximum stack height, rename strings to reduce constant pool size, remove useless fields and useless methods, reverse inlining, and so on. Static size measurements can be used to evaluate the effectiveness of the optimizations.

#### 2. Optimizing reflective access to thisJoinPoint:

In AspectJ programmers can ask for reflective information for thisJoinPoint. This information can refer to static information (stored in thisJoinPointStaticPart) or dynamic information. Most AspectJ compilers try to recognize when the the dynamic information is not needed and replace thisJoinPoint with thisJoinPointStaticPart when possible. However, when some dynamic information is needed, current compilers create all of the dynamic information, even if only part of it is needed.

Thus, the purpose of this project is to study the problem of optimizing reflective access to thisJoinPoint information. It involves looking at the code generation strategy (and the library implementation for the dynamic information), and designing and implementing techniques that optimize the performance. This could include both static analyses to detect what parts of the dynamic information are needed and tuning of the runtime library dealing with the dynamic information.

This project would be implemented in abc. This is a "Laurie's choice project".

#### 3. Obfuscation:

Decompilers try to convert bytecode to Java. The job of an obfuscator is to convert ordinary bytecode into semantically equivalent bytecode that is more difficult to decompile. This project will involve a literature search for other ofuscators and then design a new obfuscation technique(s) using Soot. You can do name obfuscation, type obsucation (by perhaps introducing spurious parameters on methods), control flow obfuscation (rewrite control flow to more complicated, but equivalent control flow) or data structure obfuscation (add extra fields), rearrange arrays, and so on.

Michael Batchelder has recently developed JBCO, a Java obfuscator based on Soot. In his recently submitted M.Sc. thesis he provides many new obfuscations and he studies the effect of those obfuscations on performance. There are two very interesting avenues of future work. First, one could develop new obfuscations and add them to JBCO. Second, one could develop some techniques for automatically determining the correct set of obfuscations which lead to maximum obfuscation, with minimum overhead (in terms of space and/or time).

See <http://www.sable.mcgill.ca/JBCO/> and Michael's thesis at <http://www.sable.mcgill.ca/publications/thesis/#michaelsMastersThesis>.

#### 4. Decompiling:

Soot contains a decompiler called Dava. Dava was originally designed to decompile arbitrary bytecode and it is quite good at doing this job. Nomair Naeem has recently finished his M.Sc. thesis on making Dava create more readable code. In his thesis he gives many interesting points to study further. See Chapter 11 of Nomair's thesis, found at <http://www.sable.mcgill.ca/publications/thesis/#nomairsMastersThesis>.

One very interesting project would be to use dataflow analysis and type information to come up with good identifier names. However, there are also other topics mentioned in Nomair's thesis.

#### 5. Faster Data Flow Analysis for Soot:

The design of the Soot data flow analysis framework focuses on flexibility and genericity rather than on speed and optimal memory use. This project would focus on a speed and/or memory optimised data flow analysis framework for Soot, perhaps even at the expense of flexibility.

Possible areas that might be examined are: (1) finding more efficient representations of flow facts, (2) different work-list algorithms specialized to specific kinds of dataflow problems, and (3) avoiding analysis or reanalysis of parts of the cfg not impacted by this analysis. Other ideas are also welcome.

## 6. String optimizations on Java bytecode

Modern versions of `javac` generate efficient calls to `StringBuffers` for computations involving string concatenation in one expression.

For example, the expression:

```
y = x+"h"+1+"l"+"jennifer"
```

would get generated as something like the following:

```
x = new StringBuffer
x.init
x.append(x)
x.append("h")
x.append(1)
...
y = x.toString()
```

However, if a method is computing a string via many different concatenations to a variable (as in a pretty printer or code generator), then not all of the string concatenations show up in a single expression - they are spread throughout the method body. Each smaller statement will result in a new `StringBuffer` and extraneous work.

The purpose of this project is to automatically detect and optimize `String` operations within a method body. Jennifer Lhotak's benchmark program from 2005 exhibits a large speedup when this optimization is done by hand and so would provide a good test for this optimization.

7. **Less Naive Jimple** Currently the Soot framework first produces very naive Jimple, and then applies a series of cleanup steps to produce tighter Jimple that is suitable for further optimizations. Although this is a very modular approach, it may be the case that it is more expensive to first produce very verbose Jimple than it would be to try and produce slightly better Jimple to begin with.

The purpose of this project is to analyze the costs involved in the process of going from bytecode to Jimple, and to suggest and implement improvements to this process.

Recently a Soot user reported some tuning that he felt helped speed up the process of producing Jimple, which included:

- Implemented parent and child pointers in the `TypeVariable` classes more efficiently
- Changed `SmartLocalDefs` to use `BitSets` rather than `HashSets` for representing flow facts.

These and other changes may make a significant improvement in the time it takes to produce good Jimple code. The mentor for this project is Laurie Hendren.

## 8. Loop Transformations

Scientific programs often use a collection of loops, which may be in sequence and/or nested. Often it is possible to improve the performance of a loop nest by restructuring it, which can include loop fusion, loop splitting, loop inversion, loop blocking and so on.

The purpose of this project would be to identify and implement a set of loop transformations, either for Java using Soot, or in for Matlab using the new McLab project (this is the preferred option).

The project should include a description of when it is legal to apply each transformation (although the static analysis needed for this need not be part of the project), and a set of example loop nests on which the transformations are demonstrated. Experiments should show the impact on performance of applying each loop transformation.

## 9. Tail recursion elimination

Recursion is often a very natural way of expressing a computation. However, recursion also entails overhead for method calls. In order to avoid this overhead, compiler writers have developed automatic techniques for rewriting a recursive computation to an equivalent iterative one. This project would involve researching the past work on tail recursion elimination (most usually done in the context of functional languages) and then developing some of the techniques either for Java, using Soot, or for Matlab, using McLab.

## 10. Shrink Wrapper

Java programmers often use large third-party libraries, such as the weka machine learning library or the Soot optimization library. In order to package their application the developers often have to either include the whole library or ask the end-user to obtain the whole library. However, another alternative is to “shrink wrap” the entire project, producing a standalone .jar file that contains only the required parts of the program and library. The purpose of this project would be to design and implement a Java shrink wrapping tool. Ideally the shrink wrapper would include only those classes, fields and methods that are needed for a specific entry point.

## 11. Optimizing bytecode from a source language other than Java

There are several compilers for other languages that produce Java bytecode. The bytecode produced by these compilers may not be optimal. The purpose of this project is to examine the generated bytecode to identify opportunities for optimization (method inlining, removal of spurious container objects, loop restructuring ...). Then, after the study on what optimizations are useful, to use Soot, both as a stand-alone tool, and to develop specialized optimizations, to produce smaller and/or faster code.

## 12. Better Fortran code from Matlab

As part of our new McLAB project we have been designing a back-end that produces Fortran code from Matlab. There are two motivations for producing Fortran code: 1) enable the use of the excellent Fortran optimizing compilers and 2) provide a way of migrating a project from Matlab to Fortran. In this second case we need to produce code that is readable and looks like code that a programmer would have written. This project would be to work with the current Fortran back-end in order to produce more readable Fortran code (which will certainly involve some analysis).

## Projects already proposed and taken

### 1. Efficient compilation of vector/matrix expressions.

Evaluation of matrix expressions typically involves creation of temporary storage for intermediate results. For example consider the following expression:

$$A = B * C + D * E - F$$

where A, B, C, D, E and F are matrices and the expression is well-formed; i.e. the dimensions of the matrices do not conflict.

If we convert the expression into 3-address IR, we obtain the following:

```
t1 <- B*C
t2 <- D*E
t3 <- t1 + t2
A <- t3 - F
```

This involves the creation of three temporary variables t1, t2 and t3, and different calls to BLAS library. However, as it is the case, in many vector/matrix expressions, these temporary variables are unnecessary and different calls to BLAS might be avoided. For example, given the dimensions of all the matrices above, by transforming the expression into loops, it is possible to compute A with fewer or no temporary variables.

This project develops a strategy for efficient compilation of vector/matrix expressions. This involves eliminating redundant intermediate results and computations; and a strategy for efficient mapping of expressions to BLAS, where appropriate. This will enable a JIT compiler to generate efficient machine code for the expressions.

## 2. Array Pointcuts

Scientific programmers heavily use arrays in any sort of computation, and arrays are the backbone of a scientific programming language such as Matlab. This aspects based project provides a way to cross-cut the concerns related to array; e.g. bound checking, type and shape, value range, etc and then optimize the array usage based on the information gathered. This analysis can also be helpful for the profiling purposes.

## **Propose a Project**

You may also propose a new project that is not listed here. If you choose something else, it should have some element of "program" analysis in it, although the "program" being analyzed could be something from a different domain such as games.