Program Analysis and Transformations - COMP 621

Laurie Hendren

Fall 2016

Brief Overview

Interested in learning about how program analysis and compiler optimization and transformation techniques can be used to: make programs run faster; build program understanding tools, decompilers, obfuscators; and help verify correctness of programs?

This course provides an introduction to the tools and techniques are used in optimizing compilers and the old title of the course was “Optimizing Compilers”. However, these tools and techniques can actually be used in a wide variety of situations including compilers, software engineering tools and tools for domain-specific languages. Hence, the course was recently renamed “Program Analysis and Transformations”.

This year we will look at analysis and transformation of imperative languages such as C, object-oriented langauges such as Java, and dynamic scientific languages such as MATLAB. The course introduces foundations including standard flow analysis techniques and program transformations. This year we will work with the McLAB framework, a framework for analyzing and optmizing MATLAB currently being developed at McGill.

A previous course in compilers is highly recommended, although not strictly necessary, but students should be willing to do some background reading if they have not taken an introductory compiler course. Students should have a familiarity with Java, C# and/or C++ and the ability to work with large applications. The course will require a large project which includes substantial software development, as well as a written project report and an oral presentation. Thus, students must be fluent in writing reports and must be capable of presenting clear oral presentations.

General Information

Course Web Page: `http://www.sable.mcgill.ca/~hendren/621`

Time: Tuesday and Thursday, 10:00-11:30am

Place: McConnell 103

Instructor:

Laurie Hendren
Office: McConnell 228
Office Hours: Tuesday 11:30-12:30, or by appointment
E-mail: hendren@cs.mcgill.ca

Suggested Prerequisites:

- An introductory course in compiler writing is quite useful, but not strictly necessary. If you do not have this background, some background reading can be done to familiarize yourself with the front-end issues of compilers. As a start you should read the first two chapters of the Aho, Sethi and Ullman book entitled *Compilers: Principles, Techniques and Tools*. This is also widely known as the "dragon book" .

  Other online resources, available free online via a McGill IP or McGill VPN are:

  - Modern compiler design, Dick Grune, `http://mcgill.worldcat.org/title/modern-compiler-de` `oclc/801809592&referer=brief_results`
  - Modern compiler implementation in Java, Appel and Palsberg, `http://mcgill.worldcat.org/title/modern-compiler-implementation-in-java/oclc/56796736?referer=br&ht=edition`
  - Compiler Design. Virtual Machines: Wilhem and Seidl, `http://mcgill.worldcat.org/title/compiler-design-virtual-machines/oclc/693779246?referer=di&ht=edition`
- Excellent programming skills, preferably in Java, C#, and/or C++.
- A familiarity with some assembly or bytecode language.
- Ability to work independently on a large research project.
- Ability to write reports and give oral presentations.

Course Description

As computer architectures become more complex, the design of good optimizing compilers becomes more critical. For high-performance architectures, for parallel architectures (including multi-core processors), and for effectively utilizing GPUs, compilers must also be able to detect and utilize fine-grain and/or coarse-grain parallelism.

Computer languages are also evolving and in many cases, such as scripting languages, they are also becoming more dynamic. Languages that are dynamic tend to have dynamic types and dynamic loading. Both of these features present challenges for the compiler.

Techniques used for optimizing compilers are also commonly used in program understanding tools, software engineering and software verification.

This course is designed to study the important components of such optimizing compilers. The major areas covered include: (1) intermediate representations, (2) flow analysis and optimization, (3) register allocation, and (4) optimizing for object-oriented and dynamic scientific languages.

The course projects will focus on techniques for the dynamic scientific programming language MATLAB. A wide range of projects will be available. Students may also suggest projects in related fields.

The course will provide a background for those students interested in further work in areas such as compiler design, flow analysis, optimization techniques, architecture design, programming language design, software engineering applications such as program understanding, decompilers, obfuscators, slicers or concern tools.

The course consists of lectures on general content by Professor Hendren, one special topic lecture by each student, 3 assignments, and a course project. The assignments combine practical experience with more theoretical development and analysis of compiler optimizations and transformations. The course project allows the students to complete their own small research project in an advanced area. Many possible projects will be suggested, and students are also free to propose their own topic which may apply to their research interests. Students will be expected to give a presentation and prepare a technical report on their selected projects. There are no examinations.

Taking Notes

Many lectures will be given using the blackboard and will contain content that is not in the reference material. Students are expected to take notes.

Reference Material

A complete list of reference material will be listed on the course web page. Please refer to this page often to see the appropriate reading.

The course will use a course pack that combines chapters from three recent compiler textbooks:

- *Modern compiler implementation in C*, by Appel.
  See also: `http://www.cs.princeton.edu/~appel/modern/index.html` and `http://books.google.ca/books?id=A3yqQuLW5RsC&dq=Compiler+Appel`.
- *Compiler design*, by Wilhelm.
  See also: `http://portal.acm.org/citation.cfm?id=546015`
- *Advanced compiler design and implementation*, by Muchnick.
  See also: `http://books.google.ca/books?id=Pq7pHwG1_OkC&dq=Advanced+Compiler+Muchnick`

These readings are available in a coursepack, made available through the McGill Bookstore. Refer to the course web page for more details on the exact readings for each week.

Evaluation

Your performance in the course will be evaluated as a combination of a written/programming assignments, and a course project. The final grade is calculated as follows:

```
Assignments       -  3 x 10%                           - 30%
Special Topic Lecture                                  - 10%

Course Project -  Project Proposal                     - 10%
                  Project Status Reports and
                  Meeting project deadlines/milestones   -  5%
                  Class Presentation                   - 10%
                  Presentation Evaluations             - 5%
                  Project Report and Implementation    - 30%
                                                    -----------
                                                        100%
```

Written and Programming Assignments: There will be three assignments. Assignments are a combination of written and programming questions. The purpose of the written questions is to allow you to master concepts that are not directly integrated with the project.

The purpose of the programming questions is to introduce you to the software platforms we will be using and to to demonstrate some practical applications of the theoretical techniques in the framework of a real compiler.

Special Topic Lecture The purpose of the special topic lecture is to introduce a variety of current research topics into the class. Ideally the topics illustrate the use of program analysis and compiler techniques to a wide variety of application areas. Students will be expected to give a 40 minute lecture, followed by 5 minutes of questions to the class about the content. Students will evaluate the lectures and feedback from this evaluation will be used by the presenter to help improve their final project presentations. The 45-minute special topic lectures will be combined with 45-minute regular lectures within one class lecture slot.

Course Project Outline and Progress Reports: The course project outline and progress report will consist of a written document and, if necessary, a meeting with the instructor and/or the project leader. Your progress report should be clearly written and it should show that you have a clear understanding of your chosen project and a reasonable plan for the completion of the project with well defined milestones.

Course Project: The purpose of the project is to allow the student to do some more concentrated work on a specific advanced topic in compilers. The project will involve choosing a topic related to the course, researching this topic, and then implementing a component of a compiler that illustrates or experiments with this topic. A list of potential topics will be suggested by the instructor. Students should also feel free to suggest topics related to their specific interests. A written project report will be required.

Class Presentation: As part of the project, each student will give a class presentationh. Each student will also evaluate all other presentations, and the quality of the evaluation will count for 5% of the mark.

Homework Assignment Policy:

Assignments should be handed in at the beginning of class on the day they are due. Work must be done individually, all material used in preparation of assignments and the project must be attributed to the appropriate sources. Late assignments, project proposals or project milestones will result in a lower mark for the meeting of deadlines/milestones. Lateness of more than one week is not normally allowed and no credit will be given for work submitted more than one week late, unless special arrangements have been made in advance of the deadline.

Academic Integrity at McGill University:

McGill University values academic integrity. Therefore all students must understand the meaning and consequences of cheating, plagiarism and other academic offences under the Code of Student Conduct and Disciplinary Procedures
(see `http://www.mcgill.ca/students/srr/honest/` for more information).

In terms of this course, part of your responsibility is to ensure that you put the name of the author on all code that is submitted. By putting your name on the code you are indicating that it is completely your own work. If you use some third-party code you must have permission to use it and you must clearly indicate the source of the code.

Other required statements:

In accord with McGill's Charter of Students' Rights, students in this course have the right to submit in English or in French any written work that is to be graded.