CSE 501: Implementation of Programming Languages (Winter '97)

Instructor: Craig Chambers

- office hours: Tu 11:00-12:00, Th 12:30-1:30 (Sieg 309)

1/2 TA: Dave Grove

- office hours: W 1:30-2:30, F 2:30-3:30 (Chateau 110)

Goals:

- Understand how languages get implemented efficiently
- Understand what can and can't be done, with how much effort
- Understand state of current research in efficient language implementation

Prerequisites:

- CSE 401 or equivalent
- CSE 505 or equivalent

Readings:

- Dragon book
- important papers from the literature

Grading:

- Midterm: 25%
- Final: 25%
- Homework: 25%
- Project: 25%

Course Outline

Week	Topic
1/6	Intro; structure of compilers; standard optimizations Standard intermediate representations; control flow, data flow; dependence Dataflow analysis; reaching constants, constant propagation
1/13	Lattice-theoretic data flow analysis framework; integer range analysis Data flow analyzer generators, frameworks
1/20	***Holiday*** Advanced intermediate representations: def/use chains, SSA form, VDG CSE; loop-invariant code motion
1/27	Inlining Interprocedural analysis Procedure specialization, partial evaluation
2/3	Alias and pointer analysis Dependence analysis; automatic parallelization
2/10	Global register allocation Calling conventions; callee-save vs. caller-save Link-time register allocation
2/17	***Holiday*** Instruction scheduling
2/19	Garbage collection
2/26	Implementing functional languages
3/5	Implementing object-oriented languages

Course Project Options

There are two flavors of course projects:

- **paper-based project** for pairs of students
- **implementation-based project** for teams of students

The paper-based project is the default for people registering for CSE501. Pairs of students will select some topic related to the course material and produce a written (and possibly oral) presentation. The implementation-based project is for those students who want a deeper, more concrete understanding of compilers and optimizations, where they will actually design, implement, and assess some optimization in the context of a real optimizing compiler. The implementation-based project offers two extra credits in the form of CSE 590 CP.

Students interested in compilers as well as fields related to compilers, such as architecture, operating systems, or programming languages, are strongly encouraged to select the implementation-based project option.

The Paper-Based Project

Description

In the paper-based project, pairs of students will pick some topic that relates to the course material and get it approved by me. They will research their topic and write a paper on it, 10-15 pages in length. I may ask for a short (e.g. 15-minute) oral presentation in class during the last week of the quarter, depending on time.

Time Schedule

You should form a team and negotiate a reasonable topic with me by **February 21**. Written reports are due **March 10**.

Sample Project Topics

- Design and specify some new optimization.
- Study and report on source-level debugging of optimized code.
- Study and report on interesting profiling tools and their interactions with compilers.
- Study and report on some aspect of automatic parallelization work.
- Study and report on instruction scheduling, register allocation, or other optimizations for interesting architectures.
- Study and report on optimizations for some other family of interesting languages, such as parallel languages, lazy functional languages, logic languages, constraint languages, visual languages, etc.
- Study and report on techniques for efficient interpretation.

The Implementation-Based Project: CSE 590 CP

Registration

Register for CSE 590 CP, for two quarters credit, in addition to CSE 501.

Description

You will form your own teams of 2-4 people and each team will negotiate a suitable course project with me. Each project should include some sort of analysis component (something the compiler needs to extract from the program) and a transformation component (an improvement the compiler makes to the program based on the extracted information). Each project will include a written report describing carefully the analyses performed, the transformations made, the generality of the optimizations, and the bottom-line improvement in program performance made by the optimizations on some specified suite of programs. Some sample projects are listed on the next page.

Several different compiler infrastructures can be built upon in the course project. I recommend the Vortex optimizing compiler which my research group built; this compiler was designed to be relatively easy to extend with new optimizations, by being written in a heavily-object-oriented style and making heavy use of first-class functions and a large standard library. Vortex can compile programs written in C, C++, Java, Modula-3, and Cecil. The default output of Vortex is C++ code, and Cecil input programs can alternatively be compiled into SPARC assembly code. On the down side, Vortex is written in the Cecil language which you're not like to know as well as some other language (this can be viewed as a feature rather than a bug...), Vortex is pretty big and not as well documented as some other compilers, and Vortex may be buggier for languages other than Cecil and Java than other compilers. We will be giving tutorials during the first half of the quarter on Cecil and Vortex, to help overcome some of these disadvantages. Other compiler infrastructures that are possible candidates for a project are the ZPL compiler, SUIF, Multiflow, and (gasp) GCC, but we cannot provide any support for projects in these compilers; I'd recommend that you use Vortex unless you already know some other compiler infrastructure.

Time Schedule

Over the course of January, you should have formed project teams and negotiated with me about the details of the project. By **February 3**, at the start of class, you should turn in a formal project proposal, which is essentially a detailed outline of your final report, including a description of your analysis & transformations and a plan for what measurements you expect to take. By **February 19**, you should turn in a draft of your project report with the section describing your analysis and transformations completely written in a precise style. By **March 3**, you should turn in another draft of this report, including preliminary results, and make your current implementation available to us. The final version of the project report and your implementation is due **March 14**. The final report can be no longer than 5 pages per team member.

Some Sample Projects

- Design and implement some standard optimization(s), such as loop-invariant code motion, induction variable elimination, or partial redundancy elimination.
- Design and implement an interprocedural optimization, perhaps by extending some intraprocedural optimization to work interprocedurally.
- Implement an alternative global (or interprocedural) register allocation algorithm, and compare it to the one already implemented. This might lead to a cool quals project.
- Implement an effective instruction scheduler, for some modern pipelined architecture(s). Again, a comparative study of scheduling algorithms might make a cool paper.
- Design and implement some sort of pointer analysis and one or more optimizations using the results of the analysis.
- Implement support for dynamic compilation. E.g., the compiler reasons about variables whose values will be constant at run-time, performs optimizations under the assumption that the variable is an (unknown) constant, and generates simple directives to patch in the real constants at run-time. Some papers have been written describing the design of this algorithm.
- Adapt Jeff Dean's inlining database to support effective automatic inlining decisions.