CS 526 Advanced Topics in Compiler Construction

Course Organization

Instructor:

David Padua
3-4223
padua@uiuc.edu
Office hours: By appointment

Course material:

Textbook: Randy Allen and Ken Kennedy
Optimizing Compilers for Modern Architectures

Web papers and Photocopies (oldies)

Website

polaris.cs.uiuc.edu/~padua/cs526

Grading

25% Homeworks

25% Presentations

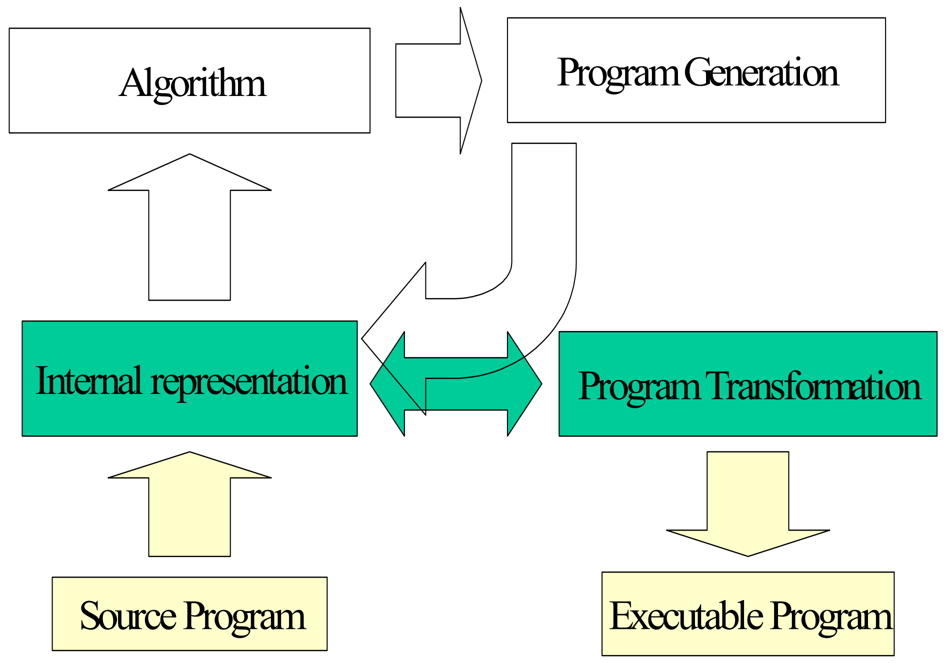
25% Midterm Exam (take home exam due on the Ides of March)

25% Final Exam (take home exam due the first day of final exams week)

Course subject

- **Optimizing** compiler technology and some notions of program generation.
- Topics in program analysis, and program transformation for sequential performance, parallelism, and locality.
- The study of compilers involve both theoretical and engineering issues.

Block diagram of compilers and program generators



Importance of optimizing compilers

- Compilers facilitate programming by
  - Presenting a high-level interface
  - Enabling portability/Machine independence
- For computational intensive applications, the acceptance of a programming language largely depends on the availability of effective compilers.
- For some classes of applications performance is not that important (e.g. web applications in which all program does is to wait.)

- Optimizing compilers liberate the programmer from concerns about machine-related issues and in this way make programs easy to port without performance loss.
- Compilers enable the programmer to focus on the development of clean (easy to understand and debug) programs. However, programmers still need to choose a good algorithm. And, usually, more efficient algorithms are more complicated.

A difficult problem

“Like most of the early hardware and software systems, Fortran was late in delivery, and didn’t really work when it was delivered. At first people thought it would never be done. Then when it was in field test, with many bugs, and with some of the most important parts unfinished, many thought it would never work. It gradually got to the point where a program in Fortran had a reasonable expectancy of compiling all the way through and maybe even running. This gradual change of status from an experiment to a working system was true of most compilers. It is stressed here in the case of Fortran only because Fortran is now almost taken for granted, as it were built into the computer hardware.”

Saul Rosen

Programming Languages and Systems

McGraw Hill 1967

- Question: How far we can go with program transformation?

Performance and compilers

"It was our belief that if FORTRAN, during its first months, were to translate any reasonable "scientific" source program into an object program only half as fast as its hand coded counterpart, then acceptance of our system would be in serious danger. This belief caused us to regard the design of the translator as the real challenge, not the simple task of designing the language."...
"To this day I believe that our emphasis on object program efficiency rather than on language design was basically correct. I believe that has we failed to produce efficient programs, the widespread use of language like FORTRAN would have been seriously delayed.

John Backus
FORTRAN I, II, and III
Annals of the History of Computing
Vol. 1, No 1, July 1979

Is performance equally important today ?

What drives optimizing compiler technology?

- High-level language, applications, and computer architecture define the character of the optimizing compiler technology that connects them.
- **High-level languages:**
  - General purpose language evolve slowly. Most of the constructs popular today were available in the 1960s.
  - Nevertheless they evolve. Object orientation is an example of evolution. Two future directions: parallel programming and higher levels of abstraction.
  -

- **Applications**
  - Applications determine the programming patterns that guide optimization strategies.
- **Architecture**
  - Parallelism in the form of global address space will dominate
  - Need to develop compiler technology for this. Compiling parallel programming languages. Automatic parallelization.