Advanced Compiler Construction

Qing Yi

class web site:
www.cs.utsa.edu/~qingyi/cs6363
A little about myself

Qing Yi
- Ph.D. Rice University, USA.
- Assistant Professor, Department of Computer Science
- Office: SB 4.01.30
- Phone: 458-5671

Research Interests
- Compiler construction
- Program analysis and optimization for high-performance computing
- Programming languages
- Type systems, object-oriented design
- Software engineering
- Automatic code generation; systematic error-discovery and verification of software

Overall goal: develop tools to improve both the productivity and efficiency of programming
General Information

- Class website
  - www.cs.utsa.edu/~qingyi/cs6363
    - Check for class handouts and announcements
  - Office hours: MW 3:30-4pm 5:15-5:45pm; by appointment

- Textbook
  - Optimizing compilers for Modern Architectures: A dependence-based Approach
    - Ken Kennedy and Randy Allen, Morgan-Kauffman Publishers Inc.

- Requirements
  - Basic understanding of algorithms, programming languages, and compilers

- Grading
  - In class exercises and quizzes: 30%
  - Research paper presentation: 20%
  - Paper review: 10%
  - Research project: 40%
Program Optimization Projects

- In order to optimize a program, a tool must be built to
  - Understand (parse/unparse) a programming language
  - Analyze the program to determine which potential optimizations are possible (safety analysis)
  - Analyze the program to determine which transformations are profitable and how to apply the transformations (optimization configuration)

- Ways to resolve the problem
  - Build a small parser/unparser for a subset of the C language
    - Easy and flexible if publishing paper is all you want
  - Use existing infrastructures from open-source compilers and languages
    - We provide the ROSE C/C++ compiler and the POET language (an interpreted language for building ad-hoc optimizers/translators)
  - Your project can focus on one of the analysis or transformation aspects of performance optimizations
Acknowledgements

- Slides from the compiler optimization class (comp515) of Rice University
  - By Prof. Ken Kennedy
    - [www.cs.rice.edu/~ken/comp515](http://www.cs.rice.edu/~ken/comp515)
  - By prof. Vevek Sarkar
High performance computing on modern machines

- Applications must efficiently manage architectural components
  - Pipelining
  - Multiple execution units --- pipelined
  - Vector operations, multi-core
  - Parallel processing
    - Shared memory, distributed memory, message-passing
  - VLIW and Superscalar instruction issue
  - Registers
  - Cache hierarchy
  - Combinations of the above --- parallel-vector machines

- What are the compilation challenges?
Optimizing For High Performance

- Optimization means eliminating inefficiencies in programs
- Eliminate redundancy: if an operation has already been evaluated, don’t do it again
  - Especially if the operation is inside loops or part of a recursive evaluation
  - All optimizing compilers apply redundancy elimination, e.g., loop invariant code motion, value numbering, global RE, PRE
- Resource management: reorder operations and data to better map to the targeting machine
  - Reorder computation(operations)
    - parallelization, vectorization, pipelining, VLIW, memory reuse
    - Instruction scheduling and loop transformations
  - Re-organization of data
    - Register allocation, regrouping of arrays and data structures
Optimizing Compilers For Modern Architectures

- Sophisticated compiler optimizations beyond traditional redundancy elimination
  - Parallelization and vectorization
  - Memory hierarchy management
  - Instruction scheduling
  - Interprocedural (whole-program) optimizations
- Goal: reorder operations to better manage the targeting machine
- Most compilers focus on optimizing loops, why?
  - This is where the application spends most of its computing time
  - What about recursive function/procedural calls?
    - Extremely important, but often left unoptimized...
Compiler Technologies

- **Source-level Program Transformations**
  - Most architectural issues can be dealt with by restructuring transformations of program source
    - Vectorization, parallelization, cache reuse enhancement
  - Challenges:
    - Determining when transformations are legal
    - Selecting transformations based on profitability

- **Low level code generation**
  - Some issues must be dealt with at a lower level
    - Prefetch insertion
    - Instruction scheduling

- All require some understanding of the ways that instructions and statements depend on one another (share data)
Dependence-based Optimization

- Vectorization and Parallelization require a deeper analysis than optimization for scalar machines
  - Bernstein’s Conditions: it is safe to run two tasks R1 and R2 in parallel if none of the following holds:
    - R1 writes into a memory location that R2 reads
    - R2 writes into a memory location that R1 reads
    - Both R1 and R2 write to the same memory location
- Dependence is the theory that makes this possible
  - There is a dependence between two statements if they might access the same location, there is a path from one to the other, and one access is a write
- Dependence has other applications
  - Memory hierarchy management
    - Restructuring programs to make better use of cache and registers
    - Includes input dependences
  - Scheduling of instructions
Dependence --- a Static Program Analysis Technique

- Program analysis --- support software development and maintenance
  - Compilation --- identify errors without running program
    - Smart development environment (check simple errors as you type the code)
  - Program Optimization --- cannot not change the meaning of the program
    - Improve performance, reduce resource consumption, …
    - Code revision/re-factoring ==> reusability, maintainability,
  - Program correctness --- Is the program safe? Is it dependable?
    - Program verification --- is the program guaranteed to satisfy certain properties? Is the implementation safe, secure, and dependable?
    - Program integration --- are there any communication errors among different components of a collaborated project?
  - Program understanding --- extract high-level semantics from low-level implementations (reverse engineering)

- In contrast, if the program needs to be run to figure out information, it is called dynamic program analysis.

Dependence: models the re-ordering constraints between operations
Focus of this class

- Reorder computation to better map to the targeting machine
  - Focus on using dependence information to guide optimizations of loops
    - Determine what transformations are safe and profitable
  - Introduce other optimizations applied at the source level
    - Regrouping of data, prefetching techniques
  - Instruction scheduling and redundancy elimination are covered in cs5363 and not covered here

- Whole program analysis and optimizations
  - Interprocedural control-flow analysis, aliasing analysis, pointer analysis
  - Extend the application scope of optimizations

- Research experience
  - Study literature on cutting edge optimizations
    - Object-oriented programming, data layout optimizations, interprocedural optimizations, tuning of optimization parameters
  - Research project
    - Identify an important problem, solve the problem, evaluate the solution.
Introduction
- Compilation for parallel machines and automatic detection of parallelism.

Dependence Theory and Practice
- Types of dependences; Testing for dependence; Control Dependence. Types of branches. If conversion. Program dependence graph.

Preliminary Transformations
- Loop normalization, scalar data flow analysis, induction variable substitution, scalar renaming.

Parallel Code Generation
- Fine- and Coarse-Grained parallel code generation and loop transformations to enable parallelism.

Memory Hierarchy Management
- The use of dependence in scalar register allocation and management of the cache memory hierarchy.

Interprocedural Analysis and Optimization
- Management of interprocedural analysis and optimization.
Roadmap

- **Week1-8 --- Fundamental theories**
  - Materials from the textbook
  - Instructors giving lectures
  - Students select from a pool of papers and project ideas
  - Initial project plan due
    - Each project must resolves at least one non-trivial problem and evaluates the solution

- **Week9-13--- theory applied to solve real problems**
  - Materials from the research literature
  - Student paper presentations
  - Class discussion and paper reviews
  - Project intermediate and final report