

CS553 Compiler Construction

Instructor:

Michelle Strout

mstrout@cs.colostate.edu

Computer Science Building 342

Office hours: decide in class

URL:

<http://www.cs.colostate.edu/~cs553>

Plan for Today

Introductions

Motivation

- Why study compilers?

Issues

- Look at some sample program optimizations and assorted issues

Administrivia

- Course details

Motivation

What is a compiler?

- A translator that converts a source program into an target program

What is an optimizing compiler?

- A translator that *somehow* improves the program

Why study compilers?

- They are specifically important:
Compilers provide a bridge between applications and architectures
- They are generally important:
Compilers encapsulate techniques for reasoning about programs and their behavior
- They are cool:
First major computer application



Prelude

Q: Who wrote the first compiler, when, and for what language?

A: Admiral Grace Murray Hopper in 1952

Q: What language did it compile?

A: A-0 (similar to 3-address code) for the UNIVAC I at Eckert-Mauchly Computer Corporation

Q: What other interesting things did Admiral Hopper accomplish?

A: Helped develop COBOL for the UNIVAC

A: In 1969, awarded the first ever Computer Science “Man-of-the-Year” Award from the Data Processing Management Association.

A: Rear Admiral in the Navy (highest rank for a woman)

A: In 1986, at 80, oldest active duty officer in the US.

A: In 1991, the National Medal of Technology (first woman to win)

Quote: “It’s easier to ask forgiveness than it is to get permission.”



Traditional View of Compilers

Compiling down

- Translate high-level language to machine code

High-level programming languages

- Increase programmer productivity
- Improve program maintenance
- Improve portability

Low-level architectural details

- Instruction set
- Addressing modes
- Pipelines
- Registers, cache, and the rest of the memory hierarchy
- Instruction-level parallelism

Isn't Compilation A Solved Problem?

“Optimization for scalar machines is a problem that was solved ten years ago”

-- David Kuck, 1990

Applications keep changing

- Interactive, real-time, mobile, secure

Machines keep changing

- New features present new problems (e.g., MMX, EPIC, profiling support, multicore)
- Changing costs lead to different concerns (e.g., loads)

Some apps always want more

- More precision
- Simulate larger systems

Languages keep changing

- Wacky ideas (e.g., OOP and GC) have gone mainstream

Goals keep changing

- Correctness
- Run-time performance
- Code size
- Compile-time performance
- Power
- Security

Modern View of Compilers

Analysis and translation are useful everywhere

- Analysis and transformations can be performed at run time and link time, not just at “compile time”
- Optimization can be applied to OS as well as applications
- Analysis can be used to improve security by finding bugs
- Analysis can be used in software engineering
 - Program understanding, reverse engineering, refactoring
 - Debugging and testing
- Increased interaction between hardware and compilers can improve performance
- **Bottom line**
 - Analysis and transformation play essential roles in computer systems
 - Computation important \Rightarrow *understanding* computation important

Some Exciting Current Research in PLDI Research

PLDI

- Programming language design and implementation
- Premier conference for dissemination of compiler and programming languages research

Parallel Programming Languages

- Most common: C/C++ or Fortran 90+ combined with MPI and/or OpenMP
 - How do you do data-flow analysis for MPI programs?
- Up and coming languages and programming models
 - DARPA HPCS languages: Cray’s Chapel, IBM’s X10, Sun’s Fortress
 - PGAS languages like UPC and CoArray FORTRAN
 - CUDA and OpenCL for programming GPUs
 - Concurrent Collections: Intel and Rice University collaboration
 - Alphaz for expressing programs as equations, CSU project

Yes, but can it help me get a job?

Summer internships in past 4 years

- LLNL with ROSE compiler (2)
- Cray with Chapel group
- NCAR looking at optimizing look up tables in Fortran 90 code
- Intel working on hand-parallelization based on compiler feedback

Check out compilerjobs.com

Government labs often looking for research programmers who know about compilers.

Remember all of those new languages being developed ...

Types of Optimizations

Definition

- An *optimization* is a transformation that is expected to improve the program in some way; often consists of *analysis* and *transformation* *e.g.*, decreasing the running time or decreasing memory requirements

Machine-independent optimizations

- Eliminate redundant computation
- Move computation to less frequently executed place
- Specialize some general purpose code
- Remove useless code

Types of Optimizations (cont)

Machine-dependent optimizations


- Replace costly operation with cheaper one
- Replace sequence of operations with cheaper one
- Hide latency
- Improve locality
- Exploit machine parallelism
- Reduce power consumption

Enabling transformations

- Expose opportunities for other optimizations
- Help structure optimizations



Sample Optimizations

Arithmetic simplification


- Constant folding
e.g., $x = 8/2;$  $x = 4;$

- Strength reduction
e.g., $x = y * 4;$  $x = y \ll 2;$

Constant propagation

- *e.g.*, $x = 3;$
 $y = 4+x;$  $x = 3;$
 $y = 4+3;$  $x = 3;$
 $y = 7;$

Copy propagation

- *e.g.*, $x = z;$
 $y = 4+x;$  $x = z;$
 $y = 4+z;$

More examples: Loop Permutation for Improved Locality

Sample code: Assume Fortran's Column Major Order array layout

```

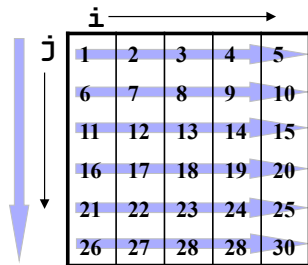
do j = 1, 6
do i = 1, 5
  A(j,i) = A(j,i)+1
enddo
enddo

```

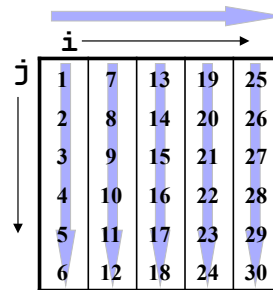
```

do i = 1, 5
do j = 1, 6
  A(j,i) = A(j,i)+1
enddo
enddo

```



poor cache locality



good cache locality

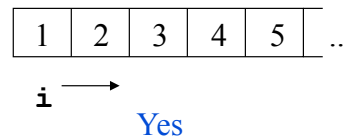
More examples: Parallelization

Can we parallelize the following loops?

```

do i = 1, 100
A(i) = A(i)+1
enddo

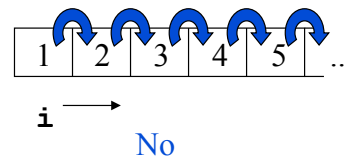
```



```

do i = 1, 100
A(i) = A(i-1)+1
enddo

```



Is an Optimization Worthwhile?

Criteria for evaluating optimizations

- **Safety**: does it preserve behavior?
- **Profitability**: does it actually improve the code?
- **Opportunity**: is it widely applicable?
- **Cost (compilation time)**: can it be practically performed?
- **Cost (complexity)**: can it be practically implemented?

Scope of Analysis/Optimizations

Peephole

- Consider a small window of instructions
- Usually machine specific

Local

- Consider blocks of straight line code (no control flow)
- Simple to analyze

Global (intraprocedural)

- Consider entire procedures
- Must consider branches, loops, merging of control flow
- Use data-flow analysis
- Make simplifying assumptions at procedure calls

Whole program (interprocedural)

- Consider multiple procedures
- Analysis even more complex (calls, returns)
- Hard with separate compilation

Limits of Compiler Optimizations

Fully Optimizing Compiler (FOC)

- $\text{FOC}(P) = P_{\text{opt}}$
- P_{opt} is the *smallest* program with same I/O behavior as P

Observe

- If program Q produces no output and never halts, $\text{FOC}(Q) =$
L: goto L

Aha!

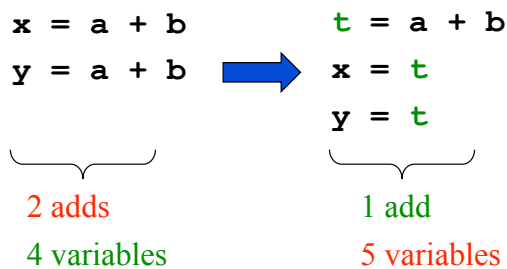
- We've solved the halting problem?!

Moral

- Cannot build FOC
- Can always build a better optimizing compiler
(*full employment theorem* for compiler writers!)

Optimizations Don't Always Help

Common Subexpression Elimination



Optimizations Don't Always Help (cont)

Fusion and Contraction

```
for i = 1 to n
  T[i] = A[i] + B[i]
for i = 1 to n
  C[i] = D[i] + T[i]
```



```
for i = 1 to n
  t = A[i] + B[i]
  C[i] = D[i] + t
```

t fits in a register, so no loads or stores in this loop.

Huge win on most machines.

Degrades performance on machines with hardware managed stream buffers.

Optimizations Don't Always Help (cont)

Backpatching

`o.foo();` } In Java, the address of `foo()` is often not known until runtime (due to dynamic class loading), so the method call requires a **table lookup**.

After the first execution of this statement, **backpatching** replaces the table lookup with a direct call to the proper function.

Q: How could this optimization ever hurt?

A: The Pentium 4 has a trace cache, when any instruction is modified, the entire trace cache has to be flushed.

Phase Ordering Problem

In what order should optimizations be performed?

Simple dependences

- One optimization creates opportunity for another
e.g., copy propagation and dead code elimination

Cyclic dependences

- *e.g.*, constant folding and constant propagation

Adverse interactions

- *e.g.*, common subexpression elimination and register allocation
e.g., register allocation and instruction scheduling

Engineering Issues

Building a compiler is an engineering activity

Balance multiple goals

- Benefit for *typical* programs
- Complexity of implementation
- Compilation speed

Overall Goal

- Identify a small set of general analyses and optimization
- Easier said than done: just one more...

Beyond Optimization

Security and Correctness

- Can we check whether pointers and addresses are valid?
- Can we detect when untrusted code accesses a sensitive part of a system?
- Can we detect whether locks are used properly?
- Can we use compilers to certify that code is correct?
- Can we use compilers to obfuscate code?

Administrative Matters

Turn to your syllabus

Discuss PA0 and PA1

Expectations

DO

- Expect to spend more time on this course than on a challenging undergraduate course.
- Write more than one draft for your project reports. Spelling mistakes will be penalized. Correct grammar is also expected (for help use Word or even better the Writing Center).
- Make decisions when the project is underspecified. Describe the reasoning for your decisions in the project report.
- Read assigned reading. Much of it will take more than two readings and anything in the readings might be on the midterm or final.
- Implement your projects in small pieces, thus making them easier to debug.
- Use a debugger BEFORE coming to me about problems in your implementation.
- Ask questions and come to office hours sooner rather than later.

Thinking is important and should be done frequently.

Next Time

Reading

- Chapters 1 and 2 in purple dragon book

Projects

- Take a look at project 0 (no turn in required)
- Find a partner for project 1 and get started

Lecture

- Undergrad compiler review

Concepts

Language implementation is interesting

Optimal in name only

Optimization scope

- Peephole, local, global, whole program

Example Program Optimizations

- Arithmetic simplification (constant folding, strength reduction)
- Constant/copy propagation
- Common subexpression elimination
- Dead assignment/code elimination
- Loop-invariant code motion
- Induction variable elimination
- Loop unrolling, permutation, and parallelization

Phase ordering problem