Introduction

- Why study compilers?
- Administrative
- Structure of a Compiler
- Optimization Example

Reference: Muchnick 1.3-1.5

Moore’s Law

Imagine: Computers that
- Small enough to fit inside cells
- Cheap enough to be disposable
- Dense enough to embed a supercomputer
- Smart enough to assemble themselves

Computers from atomic scale components

Imagining it is hard enough, achieving it requires a rethink of the entire tool chain.
What is Behind Moore's Law?

- A lot of hard work!
- Two most important tools:
  - Parallelism
    - Bit-level
    - Pipeline
    - Function unit
    - Multi-core
  - Locality

Performance: Ops/Sec

Performance: Ops/Clk * Clks/Sec

SpecInt/Mhz
Another View of Moore's Law

The Computer System

The Memory Hierarchy

Compiler Writer's Job

- Improve locality
- Increase parallelism
- Tolerate latency
- Reduce power
Why study compilers

• They are really amazing
• Combines theory & practice
  - CS is about abstraction
    • Primary abstraction: programming language
    • Compiler lowers PL to ISA (or further!)
  - Compiler is a big system
• Crucial for performance
  - especially for modern processors
  - practically part of the architecture
• I bet: Everyone will write a compiler

--

What this course is about

High-level language
(E.g., C)

Source code
(E.g., SSA)

Optimizer

IR
(E.g., SSA)

Code Generator

IR

Low-level language
(E.g., x86)

ASM

Theory and practice of modern optimizing compilers

• No lexing or parsing
• Focus on IR, back-end, optimizations
• Internals of today's (and tomorrow's) compilers
• Working with a real compiler

--

Prerequisites

• 211 & 213 or the equivalent
• Parts of 411 or the equivalent
  - Basic compiler data structures
  - Frames, calling conventions, def-use chains, etc.
  - Don't really care about front-end
• Proficient in C/C++ programming
• Basic understanding of architecture
My Expectations

- You have the prerequisites
  - If not come see me asap
- 3 assignments + a project
- Class participation
  - THIS IS A MUST!
  - Read text/papers before class
  - Attendance is essentially mandatory

Grading

- Class participation  ~20%
  - Throughout the semester
  - During paper presentations
  - Project presentations
- assignments  ~20%
- Project  ~40%
- Midterm  ~20%

Assignments

- Intro to LLVM/Liveness
- Dependence analysis
- Locality/Parallel transformations

- All labs and the final project will be done in a state-of-the-art research compiler: LLVM

The Text

- No assigned text. There are some on reserve. Its really up to you.
- Muchnick, Advanced Compiler Design & Impl., 1997
- Allen, et.al., Optimizing Compilers for Modern Archs, 2001
- Copper, et.al., Engineering a compiler, 2003
- Aho, et.al., Compilers: ..., 2006

- Papers will be assigned
Before we get too bored

• More admin at the end, but first …

• What exactly is an optimizing compiler?
  - An optimizing compiler transforms a program into an equivalent, but “better” form.
  - What is equivalent?
  - What is better?

Full Employment Theorem

• No such thing as “The optimizing compiler”
  - Why not?

• There is always a better optimizing compiler, but …
  - Compiler must preserve correctness
  - On average improve \( X \), where \( X \) is:
    • Performance
    • Power
    • …
  - Finish in your lifetime

How might performance be improved?

\[
\text{execution time} = \sum \frac{\text{cycles per instruction}}{\text{instructions}}
\]

• Reduce the number of instructions
• Replace “expensive” instructions with “cheap” ones
• Reduce memory cost
  - Improve locality
  - Reduce # of memory operations
• Increase parallelism

Ingredients to a compiler opt

• Identify opportunity
  - Avail in many programs
  - Occurs in key areas (what are these?)
  - Amenable to “efficient” algorithm
• Formulate Problem
• Pick a Representation
• Develop an Analysis
  - Detect when legal
  - And desirable
• Implement Code Transformation
• Evaluate (and repeat!)
Examples of Optimizations

• Machine Independent
  - Algebraic simplification
  - Constant propagation
  - Constant folding
  - Common Sub-expression elimination
  - Dead Code elimination
  - Loop Invariant code motion
  - Induction variable elimination

• Machine Dependent
  - Jump optimization
  - Reg allocation
  - Scheduling
  - Strength reduction
  - Loop permutations

Really Powerful Opts we won’t do

• How to optimize:

Sumfrom1toN(int max) {
  sum = 0;
  for (i=1; i<=max; i++) sum+=i;
  return sum;
}

• What we should, but won’t do:

inline sumfrom1toN(int max) {
  return max > 0 ?
    ((max+max*max)>>1) : 0;
}

Algebraic Simplifications

a*1; \Rightarrow a

a/1; \Rightarrow a

a*0; \Rightarrow 0

a+0; \Rightarrow a

a-0; \Rightarrow a

c = a - 1 \Rightarrow c = b

a = b + 1

Use algebraic identities to simplify computations
Jump Optimizations

- Simplify jump and branch instructions.

Constant Propagation

- If the compiler can determine that the values of \( a \) and \( b \) are constants, then it can replace the variable uses with constant values.

Constant Folding

- The compiler evaluates an expression (at compile time) and inserts the result in the code.
- Can lead to further optimization opportunities; esp. constant propagation.

Common Subexpression Elimination (CSE)

- If the compiler can determine that:
  - an expression was previously computed
  - and that the values of its variables have not changed since the previous computation,
Then, the compiler can use the previously computed value.
Strength Reduction

- On some processors, the cost of an addition is less than the cost of multiplication.
- The compiler can replace expensive multiplication instructions by less expensive ones.

\[
\begin{align*}
c &= b \times 2; \\
c &= b + b; \\
c &= \text{lsh}(b);
\end{align*}
\]

\[
\begin{align*}
\text{move } d0, d0 \\
\text{muls } #2, d0 \\
\text{move } d0, $3000
\end{align*}
\]

\[
\begin{align*}
c &= -1 \times b; \\
c &= \text{negative}(b);
\end{align*}
\]

\[
\begin{align*}
\text{move } d0, d0 \\
\text{muls } #-1, d0 \\
\text{move } d0, $3000
\end{align*}
\]

Dead Code Elimination

debug = False;

if (debug) {
    
    a = f(b);
}

If the compiler can determine that code will never be executed or that the result of a computation will never be used, then it can eliminate the code or the computation.

Loop Invariant Code Motion

- Loop invariant: expression evaluates to the same value each iteration of the loop.
- Code motion: move loop invariant outside loop.
- Very important because inner-most loop executes most frequently.

\[
\begin{align*}
\text{for (i=0; i<100 ; ++i) { } for (j=0; j<100 ; ++j) { } for (k=0 ; k<100 ; ++k) { } a[i][j][k] = i*j*k; } \\
\text{for (i=0; i<100 ; ++i) { } for (j=0; j<100 ; ++j) { } t1 = a[i][j]; } \\
\text{t2 = i*j; } \\
\text{for (k=0 ; k<100 ; ++k) { } t1[k] = t2*k; } \\
\text{}}
\end{align*}
\]

\[
\begin{align*}
\text{int *a; } \\
\text{int n; } \\
\text{scanf("%d", &n); } \\
\text{for (i=0; i<n ; ++i) { } for (j=0; j<n ; ++j) { } for (k=0 ; k<n ; ++k) { } f = q/p; } \\
\text{a[i][j][k] = f*i*j*k; } \\
\text{}}
\end{align*}
\]

Oooops!!!!!
Cache Optimizations

Loop permutation changes the order of the loops to improve the spatial locality of a program.

```
for (j=0; j<n; ++j) {
    for (i=0; i<n; ++i) {
        x += a[i][j];
    }
}
```

Example

A program that sorts 4-byte elements in an n-element array of integers $A[1..n]$ using bubblesort.

```
for (i=n-1; i >= 1; --i) {
    for (j = 1; j <= i; ++j) {
        if (A[j] > A[j+1]) {
            temp = A[j];
            A[j] = A[j+1];
            A[j+1] = temp;
        }
    }
}
```

A Generated IR

```
for i
    S5: if i < 1 goto Exit
    t10 = j+1
    t11 = t10-1
    t12 = 4*t11
    t13 = [A+t12]
    t14 = t13-1
    t15 = 4+t14
    [A+t15] = t13
    t16 = j+1
    t17 = t16-1
    t18 = 4+t17
    [A+t18] = temp
    A[j+1] = temp
for j
    S4: if j > i goto S2
    i = n-1
    t1 = j-1
    t2 = 4*t1
    t3 = [A+t2]
    t4 = j+1
    t5 = t4-1
    t6 = 4*t5
    t7 = [A+t6]
    if t3 <= t7 goto S3
    A[j] = temp
    S3: j = j+1
    goto S4
    temp = t8
    t8 = j-1
    t9 = 4+t8
    temp = [A+t9]
    S2: i = i-1
    goto S5
    Exit:
```
Optimizations I - Algebraic Simplifications

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t4 = j+1 \]
\[ t5 = t4-1 \]
\[ t6 = 4*t5 \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 < t7 \text{ goto S3} \]
\[ t8 = t6 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ t10 = j+1 \]
\[ t11 = t10-1 \]
\[ t12 = 4*t11 \]
\[ t13 = [A+t12] \]

Optimizations II - CSE

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t14 = t1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*t8 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ t12 = 4*j \]
\[ t13 = [A+t6] \]

Optimizations III - Copy Propagation

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t14 = t1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t9 = 4*j \]
\[ \text{temp } = [A+t9] \]
\[ t12 = t6 \]
\[ t13 = [A+t6] \]
**Optimizations IV - CSE (2)**

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = \lfloor A+t2 \rfloor \]
\[ t6 = 4*t1 \]
\[ t7 = \lfloor A+t6 \rfloor \]
\[ t9 = 4*t1 \]
\[ \text{if } t3 = t7 \text{ goto S3} \]
\[ \text{temp} = \lfloor A+t9 \rfloor \]
\[ t13 = t7 \]
\[ t15 = 4*t1 \]
\[ A+t15 = t13 \]
\[ A+t6 = \text{temp} \]

**Optimizations V - Copy Propagation (2)**

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = \lfloor A+t2 \rfloor \]
\[ t6 = 4*j \]
\[ t7 = \lfloor A+t6 \rfloor \]
\[ \text{if } t3 = t7 \text{ goto S3} \]
\[ \text{temp} = \lfloor A+t9 \rfloor \]
\[ t13 = t7 \]
\[ t15 = t2 \]
\[ A+t15 = t13 \]
\[ A+t6 = \text{temp} \]

**Optimization VI - CSE (3)**

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = \lfloor A+t2 \rfloor \]
\[ t6 = 4*j \]
\[ t7 = \lfloor A+t6 \rfloor \]
\[ \text{if } t3 = t7 \text{ goto S3} \]
\[ \text{temp} = t3 \]

**Optimization VII - Copy Propagation (3)**

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = \lfloor A+t2 \rfloor \]
\[ t6 = 4*j \]
\[ t7 = \lfloor A+t6 \rfloor \]
\[ \text{if } t3 = t7 \text{ goto S3} \]
\[ \text{temp} = t3 \]
Optimizations VIII – IVE & Strength
Reduction

\[ i = n-1 \]

\[ S5: \text{if } i < 1 \text{ goto Exit} \]

\[ j = 1 \]

\[ S4: \text{if } j > i \text{ goto S2} \]

\[ t1 = j-1 \]

\[ t2 = 4*t1 \]

\[ t3 = [A+t2] \]

\[ t6 = 4*j \]

\[ t7 = [A+t6] \]

\[ \text{if } t3 <= t7 \text{ goto S3} \]

\[ [A+t2] = t7 \]

\[ [A+t6] = t3 \]

\[ S3: j = j+1 \]

\[ \text{goto S4} \]

\[ S2: i = i-1 \]

\[ \text{goto S5} \]

\[ \text{Exit:} \]

\[
\begin{align*}
\text{Loop Invariant} \\
\text{Code Motion...}
\end{align*}
\]
Done For Now.

\[ i = n-1 \]
\[ t19 = i<<2 \]
\[ \text{if } t19 < 4 \text{ goto Exit} \]

S5:  \[ t6 = 4 \]
\[ \text{if } t6 > t19 \text{ goto S2} \]

S4:  \[ t3 = \lceil A+t6-4 \rceil \]
\[ t7 = \lceil A+t6 \rceil \]
\[ \text{if } t3 \leq t7 \text{ goto S3} \]
\[ \lceil A+t6-4 \rceil = t7 \]
\[ \lceil A+t6 \rceil = t3 \]

S3:  \[ t6 = t6+4 \]
\[ \text{if } t6 \leq t19 \text{ goto s4} \]

S2:  \[ t19 = t19 - 4 \]
\[ \text{if } t19 \geq 4 \text{ goto s5} \]

Exit:

Inner loop: 7 instructions
\[ 4 \text{ mem ops} \]
\[ 2 \text{ branches} \]
\[ 1 \text{ addition} \]

Original inner loop: 25 instructions
\[ 6 \text{ mem ops} \]
\[ 3 \text{ branches} \]
\[ 10 \text{ addition} \]
\[ 6 \text{ multiplication} \]

Course Schedule

- [www.cs.cmu.edu/afs/cs/academic/class/15745-s09/www/](http://www.cs.cmu.edu/afs/cs/academic/class/15745-s09/www/)
- The Web site is a vital resource
- (And, of course me too.)

Course Staff

- Seth Goldstein
  seth@cmu.edu
- [www..../~seth](http://www..../~seth)
- Heather Carney
  hcarney@cs.cmu.edu

First Assignment

- Install llvm on your favorite machine
- Get familiar with llvm tools, IR, structure
- Lots of docs at [www.llvm.org](http://www.llvm.org)
- First part of assignment 1 will be posted Friday.