Introduction

• Why study compilers?
• Administriva
• 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: \( \text{Ops/Clk} \times \text{Clks/Sec} \)
Another View of Moore’s Law
The Computer System

The Memory Hierarchy

Register | Cache | Memory | Disk Memory
---------|-------|--------|--------------
size: 200 B | 32 KB/4 MB | 128 MB | 20 GB
speed: 3 ns | 6 ns | 60 ns | 8 ms
$/Mbyte: $100/MB | $.30/MB | $0.005/MB
block size: 8 B | 32 B | 8 KB | larger, slower, cheaper
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
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

- 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
- Building a real compiler for embedded processor
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 tim or me asap
• 4 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

• Labs ~20%
• Project ~30%
• Midterm ~15%
• Final ~15%
Labs

- ADCE & CCP in Pegasus
- Global register allocation
- Global Code Scheduling
- Profile-based optimization

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

The Text


- 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” instructs 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:

```c
Sumfrom1toN(int max) {
    sum = 0;
    for (i=1; i<=max; i++) sum+=i;
    return sum;
}
```
Really Powerful Opts we won’t do

- How to optimize:
  ```c
  Sumfrom1toN(int max) {
    sum = 0;
    for (i=1; i<=max; i++) sum+=i;
    return sum;
  }
  ```

- What we should, but won’t do:
  ```c
  inline sumfrom1toN(int max) {
    return max > 0 ?
    ((max+max*max)>>1) : 0;
  }
  ```

Algebraic Simplifications

```
  a*1; ⇒ a
  a/1; ⇒ a
  a*0; ⇒ 0
  a+0; ⇒ a
  a-0; ⇒ a
  a = b + 1
  c = a - 1  ⇒  c = b
```

Use algebraic identities to simplify computations
Jump Optimizations

\[
\begin{align*}
\text{cmp } d0, d1 & \quad \text{cmp } d0, d1 \\
\text{beq } L1 & \quad \text{beq } L1 \\
\text{bra } L2 & \quad \text{bne } L2 \\
L1: & \quad \Rightarrow \\
L2: & \\
\end{align*}
\]

Simplify jump and branch instructions.

Constant Propagation

\[
\begin{align*}
a &= 5; \\
b &= 3; \\
\vdots & \\
n &= a + b; & \Rightarrow n &= 5 + 3; \\
\text{for } (i = 0; i < n; ++i) & \quad \text{for } (i = 0; i < n; ++i) \\
\{ & \quad \{
\vdots \\
\} & \quad \}
\end{align*}
\]

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.

```
n = 5 + 3;
for (i = 0 ; i < 8 ; ++i) {
    :
}
```

Common Subexpression Elimination (CSE)

```
a = c*d;
d = (c*d + t) * u
```

⇒

```
a = c*d;
d = (a + t) * u
```

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; \\
    \text{move} &\quad \text{d0} \\
    \text{muls} &\quad \text{d0} \\
    \text{move} &\quad d0, \: 3000 \\

    c &= b + b; \\
    \text{move} &\quad \text{d0} \\
    \text{add} &\quad d0, d0 \\
    \text{move} &\quad d0, \: 3000 \\

    c &= \text{lsh}(b); \\
    \text{move} &\quad \text{d0} \\
    \text{lsl} &\quad d0, 1 \\
    \text{move} &\quad d0, \: 3000 \\

    c &= -1 \times b; \\
    \text{move} &\quad \text{d0} \\
    \text{muls} &\quad -1, d0 \\
    \text{move} &\quad d0, \: 3000 \\

    c &= \text{negative}(b); \\
    \text{move} &\quad \text{d0} \\
    \text{neg} &\quad d0 \\
    \text{move} &\quad d0, \: 3000
\end{align*}
\]

Dead Code Elimination

```python
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.

```
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;
        }
    }
}
```

```
for (i=0; i<100; ++i) {
    for (j=0; j<100; ++j) {
        t1 = a[i][j];
        t2 = i*j;
        for (k=0; k<100; ++k) {
            t1[k] = t2*k;
        }
    }
}
```

Oooops!!!!!
Cache Optimizations

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

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

Cache Optimizations

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

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

// i and j are not used later

---

A Generated IR

```
for i
    i = n-1
    S5: if i < 1 goto Exit
    j = 1

for j
    S4: if j > i goto S2
    t1 = j-1
    t2 = 4*t1
    t3 = [A+t2]
    t4 = j+1
    t5 = t4-1
    t6 = 4*t5
    if t6 <= t7 goto S3
    A[j] = [A+t6]
    t7 = temp
    t8 = j-1
    t9 = 4*t8
    temp = [A+t9]

for j
    if t3 <= t7 goto S3
    t10 = j+1
    t11 = t10-1
    t12 = 4*t11
    t13 = [A+t12]
    t14 = j-1
    t15 = 4*t14
    [A+t15] = t13
    t16 = j+1
    t17 = t16-1
    t18 = 4*t17
    [A+t18] = temp
    A[j+1] = temp

S3: j = j+1 goto S4
for j

for i
    S5: if i < 1 goto Exit
    j = 1
```

---
Another generated IR

Optimizations I - Algebraic Simplifications

\[
i = n-1
\]
S5: if i < 1 goto Exit
 j = 1
S4: if j > i goto S2
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
t8 = j-1
t9 = 4*t8
\text{temp} = [A+t9]
\]
t10 = j+1
t11 = t10-1
t12 = 4*t11
t13 = [A+t12]
t14 = j-1
t15 = 4*t14
[A+t15] = t13
t16 = j+1
t17 = t16-1
t18 = 4*t17
[A+t18] = \text{temp}
\]
S3: j = j+1
 \text{goto S4}
S2: i = i-1
 \text{goto S5}
Exit:
Optimizations II - CSE

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 \leq t7 \text{ goto S3} \]
\[ t8 = j-1 \]
\[ t9 = 4*t8 \]
\[ \text{temp} = [A+t9] \]
\[ t12 = 4*j \]
\[ t13 = [A+t12] \]
\[ t14 = j-1 \]
\[ t15 = 4*t14 \]
\[ [A+t15] = t13 \]
\[ t18 = 4*j \]
\[ [A+t18] = \text{temp} \]
\[ \text{S3: } j = j+1 \]
\[ \text{goto S4} \]
\[ \text{S2: } i = i-1 \]
\[ \text{goto S5} \]
\[ \text{Exit:} \]
### Optimizations III - Copy Propagation

\[ i = n-1 \]

**S5:** if \( i < 1 \) goto Exit

\[ j = 1 \]

**S4:** if \( j > i \) goto S2

\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 \leq t7 \text{ goto S3} \]

\[ t8 = t1 \]
\[ t9 = 4*t8 \]
\[ \text{temp} = [A+t9] \]

**S3:** \( j = j+1 \)

**S2:** \( i = i-1 \)

**Exit:**

\[ t9 = 4*t1 \]
\[ [A+t6] = \text{temp} \]

### Optimizations IV - CSE (2)

\[ i = n-1 \]

**S5:** if \( i < 1 \) goto Exit

\[ j = 1 \]

**S4:** if \( j > i \) goto S2

\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 \leq t7 \text{ goto S3} \]

\[ t9 = t1 \]
\[ t9 = 4*t8 \]
\[ \text{temp} = [A+t9] \]

**S3:** \( j = j+1 \)

**S2:** \( i = i-1 \)

**Exit:**

\[ t9 = 4*t1 \]
\[ [A+t15] = t13 \]
\[ [A+t18] = \text{temp} \]

\[ [A+t6] = \text{temp} \]

\[ \text{temp} = [A+t9] \]
Optimizations V - Copy Propagation (2)

\[
i = n-1
\]

S5: if i < 1 goto Exit
\[
j = 1
\]

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

Optimization VI - CSE (3)

\[
i = n-1
\]

S5: if i < 1 goto Exit
\[
j = 1
\]

S4: if j > i goto S2
\[
t1 = j-1
\]
\[
t2 = 4*t1
\]
\[
t3 = [A+t2]
\]
\[
t6 = 4*j
\]
\[
t7 = [A+t6]
\]
\[
\text{if } t3 \leq t7 \text{ goto S3}
\]
\[
t9 = t2
\]
\[
temp = [A+t9]
\]
\[
\text{temp = [A+t2]}
\]
\[
[A+t2] = t7
\]
\[
[A+t6] = \text{temp}
\]
Optimization VII - Copy Propagation (3)

\[ i = n-1 \]

S5: if \( i < 1 \) goto Exit

\[ j = 1 \]

S4: if \( j > i \) goto S2

\[ t1 = j-1 \]

\[ t2 = 4*t1 \]

\[ t3 = [A+t2] \]

\[ t6 = 4*j \]

\[ t7 = [A+t6] \]

if \( t3 \leq t7 \) goto S3

\[ t6 = 4*j \]

\[ t7 = [A+t6] \]

if \( t3 \leq t7 \) goto S3

\[ temp = [t3] \]

\[ [A+t2] = t7 \]

\[ [A+t6] = temp \]

Optimizations VIII - IVE & Strength Reduction

\[ i = n-1 \]

S5: if \( i < 1 \) goto Exit

\[ j = 1 \]

S4: if \( j > i \) goto S2

\[ t1 = j-1 \]

\[ t2 = 4*t1 \]

\[ t3 = [A+t2] \]

\[ t6 = 4*j \]

\[ t7 = [A+t6] \]

if \( t3 \leq t7 \) goto S3

\[ [A+t2] = t7 \]

\[ [A+t6] = t3 \]

S3: j = j+1
goto S4

S2: i = i-1
goto S5

Exit:
Optimizations VIII - IVE & Strength Reduction

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ [A+t2] = t7 \]
\[ [A+t6] = t3 \]
\[ S3: j = j+1 \]
\[ \text{goto S4} \]
\[ S2: \text{if } t19 < 4 \text{ goto S2} \]
\[ \text{goto S4} \]
\[ \text{Exit:} \]

Loop Invariant Code Motion...

Done?

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ t2 = 0 \]
\[ t6 = 4 \]
\[ S4: \text{if } t19 < 4 \text{ goto S2} \]
\[ t19 = 4*i \]
\[ t2 = t2+4 \]
\[ t6 = t6+4 \]
\[ \text{goto S4} \]
\[ S3: \text{if } [A+t2] \text{ goto S3} \]
\[ t2 = t2+4 \]
\[ t6 = t6+4 \]
\[ \text{goto S4} \]
\[ S2: \text{if } t19 < 4 \text{ goto S2} \]
\[ t19 = t19-4 \]
\[ \text{goto S4} \]
\[ \text{Exit:} \]
Done?

\[
i = n-1 \\
t19 = i*4 \\
S5: \text{if } t19 < 4 \text{ goto Exit} \\
t6 = 4 \\
S4: \text{if } t6 > t19 \text{ goto S2} \\
t3 = [A+t6-4] \\
t7 = [A+t6] \\
\text{if } t3 \leq t7 \text{ goto S3} \\
[A+t6-4] = t7 \\
[A+t6] = t3 \\
S3: t6 = t6+4 \\
\text{goto S4} \\
S2: t19 = t19 - 4 \\
\text{goto S5} \\
\text{Exit:}
\]

Eliminate mult, Use double load (if aligned?)
Unroll?
Eliminate jmp

Done For Now.

\[
i = n-1 \\
t19 = i<<2 \\
\text{if } t19 < 4 \text{ goto Exit} \\
S5: t6 = 4 \\
\text{if } t6 > t19 \text{ goto S2} \\
S4: t3 = [A+t6-4] \\
t7 = [A+t6] \\
\text{if } t3 \leq t7 \text{ goto S3} \\
[A+t6-4] = t7 \\
[A+t6] = t3 \\
S3: t6 = t6+4 \\
\text{if } t6 \leq t19 \text{ goto S4} \\
S2: t19 = t19 - 4 \\
\text{if } t19 \geq 4 \text{ goto S5} \\
\text{Exit:}
\]

Inner loop: 7 instructions
4 mem ops
2 branches
1 addition

Original inner loop: 25 instructions
6 mem ops
3 branches
10 addition
6 multiplication
Course Schedule

- www.cs.cmu.edu/afs/cs/academic/class/15745-s07h/www/

- The Web site is a vital resource

- Also, class newsgroup

- (And, of course us too)

Course Staff

- Tim Callahan  www..../~tcal

- Seth Goldstein  www..../~seth

- Mahim Mishra  www..../~mishra

- Marilyn Walgora  mwalgora@cs.cmu.edu