Introduction

- Why study compilers?
- Administra
- 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: \( \text{Ops/Clk} \times \text{Clks/Sec} \)

SpecInt/Mhz

Another View of Moore’s Law
**The Computer System**

- Processor
- Cache
- Memory-I/O bus
  - I/O controller
  - I/O controller
  - I/O controller
- Memory
- Disk
- Display
- Network

**The Memory Hierarchy**

- CPU
  - 8 B
- Cache
  - 32 B
- Memory
  - 8 KB
- Disk

<table>
<thead>
<tr>
<th>Register</th>
<th>Cache</th>
<th>Memory</th>
<th>Disk</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>size:</td>
<td>200 B</td>
<td>32 KB</td>
<td>128 MB</td>
<td>20 GB</td>
</tr>
<tr>
<td>speed:</td>
<td>3 ns</td>
<td>6 ns</td>
<td>60 ns</td>
<td>8 ms</td>
</tr>
<tr>
<td>$/Mbyte:</td>
<td>$100/MB</td>
<td>$0.30/MB</td>
<td>$0.005/MB</td>
<td></td>
</tr>
<tr>
<td>block size:</td>
<td>8 B</td>
<td>32 B</td>
<td>8 KB</td>
<td></td>
</tr>
</tbody>
</table>

**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

**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

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
• 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?

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

- 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 Opt. 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 = b
- c = a - 1 ⇒ c = b

Use algebraic identities to simplify computations.

Jump Optimizations

- Simplify jump and branch instructions.
Constant Propagation

\[
a = 5; \\
b = 3; \\
\text{\vdots} \\
n = a + b; \\
\text{\vdots} \\
\text{for (i = 0; i<n; ++i) \{} \\
\text{\vdots} \\
\text{\}}
\]

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.

\[
\begin{align*}
\text{a} &= 5; \\
\text{b} &= 3; \\
\text{\vdots} \\
n &= 5 + 3; \\
\text{for (i = 0; i < 8; ++i) \{} \\
\text{\vdots} \\
\text{\}}
\end{align*}
\]

Common Subexpression Elimination (CSE)

\[
a = c*d; \\
\text{\vdots} \\
d = (c*d + t) * u \\
\]

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.

\[
\begin{align*}
c &= b \times 2; \\
&\text{move } $2000, d0 \\
&\text{mul } #2, d0 \\
&\text{move } d0, $3000 \\
c &= b + b; \\
&\text{move } $2000, d0 \\
&\text{add } d0, d0 \\
&\text{lsl } #1, d0 \\
&\text{move } d0, $3000 \\
c &= -1*b; \\
&\text{move } $2000, d0 \\
&\text{mul } #-1, d0 \\
&\text{neg } d0 \\
&\text{move } d0, $3000
\end{align*}
\]
**Dead Code Elimination**

```java
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**

```java
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;
```

- 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.

---

**Cache Optimizations**

```java
int *a;
int n;
for (i=0; i<n; ++i)
    for (j=0; j<n; ++j)
        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
    for j
        temp = A[j]
            temp = A[j];
            A[j] = A[j+1];
            A[j+1] = temp;
        }
    }
for i
    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
    if t3 <= t6 goto S3
    t8 = j-1
    t9 = 4*t8
    temp = [A+t9]
    t10 = j+1
    t11 = t10-1
    t12 = 4*t11
    t13 = [A+t12]
    t14 = j-1
    t15 = 4*t14
    t16 = j+1
    t17 = t16-1
    t18 = 4*t17
A[j+1] = t6
A[j+1] = t3
    if t3 <= t7 goto S3
    t7 = [A+t6]
    A[j+1] = temp
    A[j] = temp
S3: j = j+1
    goto S4
S2: i = i-1
    goto S5
Exit:
```

Another generated IR

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

Optimizations I - Algebraic Simplifications

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

```
Optimizations II - CSE

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 < t7 \text{ goto S3} \]
\[ t8 = j-1 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ t12 = 4*j \]
\[ t13 = [A+t12] \]
\[ t14 = j-1 \]
\[ t15 = 4*t14 \]
\[ A+t15 = t13 \]
\[ t16 = 4*j \]
\[ t17 = [A+t16] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ t8 = t1 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ S3: \text{j } = j+1 \]
\[ \text{goto S4} \]
\[ S2: \text{i } = i-1 \]
\[ \text{goto S5} \]
\[ \text{Exit:} \]

Optimizations III - Copy Propagation

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ t8 = j-1 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ t12 = 4*j \]
\[ t13 = [A+t12] \]
\[ t14 = j-1 \]
\[ t15 = 4*t15 \]
\[ A+t15 = t13 \]
\[ t16 = 4*j \]
\[ t17 = [A+t16] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ t8 = t1 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ S3: \text{j } = j+1 \]
\[ \text{goto S4} \]
\[ S2: \text{i } = i-1 \]
\[ \text{goto S5} \]
\[ \text{Exit:} \]

Optimizations IV - CSE (2)

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ t8 = j-1 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ t13 = t7 \]
\[ t15 = 4*t15 \]
\[ A+t15 = t13 \]
\[ t16 = 4*j \]
\[ t17 = [A+t16] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ t8 = t1 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ S3: \text{j } = j+1 \]
\[ \text{goto S4} \]
\[ S2: \text{i } = i-1 \]
\[ \text{goto S5} \]
\[ \text{Exit:} \]

Optimizations V - Copy Propagation (2)

\[ i = n-1 \]
\[ S5: \text{if } i < 1 \text{ goto Exit} \]
\[ j = 1 \]
\[ S4: \text{if } j > i \text{ goto S2} \]
\[ t1 = j-1 \]
\[ t2 = 4*t1 \]
\[ t3 = [A+t2] \]
\[ t6 = 4*j \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ t8 = j-1 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ t13 = t7 \]
\[ t15 = t2 \]
\[ [A+t2] = t7 \]
\[ [A+t15] = t13 \]
\[ t16 = 4*j \]
\[ t17 = [A+t16] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ t8 = t1 \]
\[ t9 = 4*t8 \]
\[ \text{temp } = [A+t9] \]
\[ S3: \text{j } = j+1 \]
\[ \text{goto S4} \]
\[ S2: \text{i } = i-1 \]
\[ \text{goto S5} \]
\[ \text{Exit:} \]

\[ temp = [A+t2] \]
Optimization VI - CSE (3)

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

\[
[A+t2] = t7 \\
[A+t6] = \text{ temp}
\]

Optimization VII - Copy Propagation (3)

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

Optimizations VIII - IVE & Strength Reduction

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

\[
t2 = t2+4 \\
t6 = t6+4 \\
goto S4 \\
S2: \text{ i = i-1} \\
goto S5 \\
\text{ Exit:}
\]

\[
[A+t2] = t7 \\
[A+t6] = \text{ temp}
\]

Done?

\[
i = n-1 \\
S5: \text{if } i < 1 \text{ goto Exit} \\
t2 = 0 \\
t6 = 4 \\
S4: t19 = 4*i \\
\text{ if } t6 > t19 \text{ goto S2} \\
t3 = [A+t2] \\
t7 = [A+t6] \\
\text{ if } t3 \leq t7 \text{ goto S3} \\
[A+t2] = t7 \\
[A+t6] = t3 \\
S3: t2 = t2+4 \\
t6 = t6+4 \\
goto S4 \\
S2: \text{ i = i-1} \\
goto S5 \\
\text{ Exit:}
\]

\[
[A-4+t6] \\
t19 = i*4 \\
t19 < 4 \\
[A-4+t6] \\
t19 = t19-4 \\
\]
Done?

\[ i = n-1 \]
\[ t19 = i*4 \]

\[ S5: \text{if } t19 < 4 \text{ goto Exit} \]
\[ t6 = 4 \]

\[ S4: \text{if } t6 > t19 \text{ goto S2} \]
\[ t3 = [A+t6-4] \]
\[ t7 = [A+t6] \]
\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ [A+t6-4] = t7 \]
\[ [A+t6] = t3 \]

\[ S3: t6 = t6+4 \]
\[ \text{goto S4} \]

\[ S2: t19 = t19 - 4 \]
\[ \text{Use double load (if aligned?)} \]
\[ \text{goto S5} \]

Exit:

Unroll?

Eliminate jmp

---

Done For Now.

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

\[ S5: t6 = 4 \]

\[ \text{if } t6 > t19 \text{ goto S2} \]

\[ S4: t3 = [A+t6-4] \]
\[ t7 = [A+t6] \]

\[ \text{if } t3 <= t7 \text{ goto S3} \]
\[ [A+t6-4] = t7 \]
\[ [A+t6] = t3 \]

\[ S3: t6 = t6+4 \]
\[ \text{if } t6 <= t19 \text{ goto S4} \]

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

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-s05/www/](http://www.cs.cmu.edu/afs/cs/academic/class/15745-s05/www/)
- The Web site is a vital resource
- Also, class newsgroup
- (And, of course us too)

---

Course Staff

- Seth Goldstein
  [www.cs.cmu.edu/~seth](http://www.cs.cmu.edu/~seth)
  412-268-3828
  7122 Wean Hall
- Tim Callahan
  [www.cs.cmu.edu/~tcal](http://www.cs.cmu.edu/~tcal)
  8211 Wean Hall
- Marilyn Walgora
  mwalgora@cs.cmu.edu