Lecture 1

Advanced Compilers Course Introduction

I. Why Study Compilers?
II. Mathematical Abstractions: with Examples
III. Course Syllabus

Chapters 1.1-1.5, 8.4, 8.5, 9.1

Why Study Compilers?

Impact!
Techniques in compilers help all programmers
Compiler Technology: Key Programming Tool

Bridge the semantic gap between programmers and machines

Concepts in programming languages
- High-level programming languages
- Domain-specific languages
- Natural language

Concepts in computer architecture
- RISC vs CISC
- Locality: Caches, memory hierarchy
- Parallelism:
  - Instruction-level parallelism
  - Multi-processors

Programmers

Programming Language

Compilers

Machine

Programming Tools
- Security audits
- Binary translations

Compiler Study Trains Good Developers

• Reasoning about programs makes better programmers
• Tool building: there are programmers and there are tool builders ...
• Excellent software engineering case study: Compilers are hard to build
  – Input: all programs
  – Objectives:
• Methodology for solving complex real-life problems
  – Build upon mathematical / programming abstractions
Compilers: Where theory meets practice

- Desired solutions are often NP-complete / undecidable
- Key to success: Formulate the right abstraction / approximation
  - Can't be solved by just pure hacking
    - theory aids generality and correctness
  - Can't be solved by just theory
    - experimentation validates & provides feedback to problem formulation
- Tradeoffs: Generality, power, simplicity, and efficiency

Why Study Compilers?

Impact!
Techniques in compilers help all programmers

Better Programmer
Reasoning about programs
Mathematical Abstractions
Course Emphasis

- Methodology: apply the methodology to other real life problems
  - Problem statement
    - Which problem to solve?
  - Theory and Algorithm
    - Theoretical frameworks
    - Algorithms
  - Experimentation: Hands-on experience
    (Weekly programming/written homeworks)

- Compiler knowledge:
  - Non-goal: how to build a complete optimizing compiler
  - Important algorithms
  - Exposure to new ideas
  - Background to learn existing techniques

NOTE

- These slides supplement lectures
- They are not self contained!
- May miss main points to be emphasized in class!
The Rest of this Lecture

• Goal
  – Overview the course
  – Explain why I chose the topics
  – Emphasize abstraction methodology
• For each topic:
  – Motivate its importance
  – Show an example to illustrate the complexity
  – Describe the abstraction
  – Impact

1. Optimizing Compilers for High-Level Programming Languages

• Example:
  Bubblesort program that sorts array A allocated in static storage

```c
for (i = n-2; i >= 0; i--)
    for (j = 0; j <= i; j++)
        if (A[j] > A[j+1])
            temp = A[j];
            A[j] = A[j+1];
            A[j+1] = temp;
}
```

M. Lam
Code Generated by the Front End

```
S5:  if i<0 goto s1
  j := 0
t13 = j+1
t14 = 4*t13
S5:  if t29 < t28 goto s1
  t25 = t28
t26 = t30
  t25 = t28
t26 = t30
t27 = 4*i
t28 = 4A
t29 = t27+t28
t30 = t28+4
  t4 = t29
t5 = j+1
t6 = 4*t5
t7 = t2+t1
t8 = t7+t6
t9 = t8
t10 = 4*j
s3:  j = j+1
t11 = 6A
goto S4
t12 = t11+t10
s2:  i = i-1
temp = *t12
go to s5
s1:
```

After Optimization

Result of applying:
- global common subexpression
- loop invariant code motion
- induction variable code motion
- dead-code elimination

to all the scalar and temp. variables

```
i = n-2
t27 = 4*i
t28 = 6A
t29 = t27+t28
t30 = t28+4
S5:  if t29 < t28 goto s1
t25 = t28
t26 = t30
s4:  if t25 > t29 goto s2
t4 = t25
t9 = t26
if t4 <= t9 goto s3
temp = *t25
t17 = *t26
t1 = 4*j
t2 = 6A
t3 = t2+t1
t4 = t3
t5 = j+1
t6 = 4*t5
t7 = 6A
t8 = t7+t6
t9 = t8
if t4 <= t9 goto s3
  t4 = t25
t9 = t26
  temp = *t25
t17 = *t26
  *t25 = t17
  *t26 = temp
t25 = t25+4
t26 = t26+4
goto S4
s2:  t29 = t29-4
goto S5
s1:
```
Summary

<table>
<thead>
<tr>
<th>Topic</th>
<th>Abstraction</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data flow optimizations</td>
<td>Graphs</td>
<td>High-level programming without loss of efficiency</td>
</tr>
<tr>
<td></td>
<td>Recurrent equations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fixed-point</td>
<td></td>
</tr>
<tr>
<td>Parallelism and locality</td>
<td>Integer linear programming</td>
<td>Hide parallelism and locality from programmers</td>
</tr>
<tr>
<td></td>
<td>Linear algebra</td>
<td></td>
</tr>
<tr>
<td>Pointer alias analysis</td>
<td>Program Query Language</td>
<td>Automate error-prone security inspection.</td>
</tr>
<tr>
<td></td>
<td>Logic database (Datalog)</td>
<td>Illustrates language abstraction</td>
</tr>
<tr>
<td>Natural language programming</td>
<td>Neural networks</td>
<td>Consumers can automate personal and professional</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tasks themselves, eliminating dependence on coders.</td>
</tr>
</tbody>
</table>
Nvidia Volta GV100 GPU

- 218 transistors
- 815 mm²
- 80 Stream Multiprocessors (SM)
- 1455 MHz

In Each SM

- 64 FP32 cores
- 64 int cores
- 32 FP64 cores
- 8 Tensor cores

Tensor Cores:
\[ D = A \times B + C; \] A, B, C, D are 4x4 matrices
4x4 matrix processing array
1024 floating point ops / clock

FP32: 15 TFLOPS
FP64: 7.5 TFLOPS
Tensor: 120 TFLOPS
Parallelism and Locality

- Can programmers focus on high-level programming & get performance?
- **Example:** matrix multiply: core kernel in neural networks

```c
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        for (k = 0; k < N; k++) {
            m3(i, j) += m1(i, k) * m2(k, j);
        }
    }
}
```

- Lots of parallelism in the program: $N^2$
- Poor sequential / parallel performance without locality optimization

---

Optimizing for Single Core: Permuting Loops

```c
for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
        for (k = 0; k < N; k++) {
            m3(i, j) += m1(i, k) * m2(k, j);
        }
    }
}
```

Permute loop to make data access contiguous for vectorization:

```c
for (k = 0; k < N; k++) {
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            m3(i, j) += m1(i, k) * m2(k, j);
        }
    }
}
```
# Tiling: to Increase Reuse

for (k = 0; k < N; k++) {
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            m3(i, j) += m1(i, k) * m2(k, j);
        }
    }
}

**Assume cache size < N^2**

Tile the outermost loop

for (k1 = 0; k1 < N; k1 += B) {
    for (i = 0; i < N; i++) {
        for (k2 = k1; k2 < k1 + B; k2++) {
            for (j = 0; j < N; j++) {
                m3(i, j) += m1(i, k2) * m2(k2, j);
            }
        }
    }
}

**Assume N is divisible by B**

# Experiment

- Square float32 matrix of various sizes
- Initialized with random (0, 1) normal
- Average of 10 iterations
- Intel i7-4770HQ CPU @ 2.20GHz (Haswell), no turbo
  - Number of cores: 4
  - Number of threads: 8
  - SSE4.2 and AVX2: 256 bit SIMD instructions
- 32k L1 cache, 256k L2, 6M L3, 132M L4 cache (LLC, GPU shared)
- Compiled with g++ 7.2.1 20170915, as provided in Fedora 27
- Common options: --std=c++14 -Wall -g
- (The production version of clang does not support loop optimizations)
**Sequential Performance**

![Graph showing sequential performance](image)

**Parallel scaling (matrix size 1500)**

![Graph showing parallel scaling](image)
### Summary

<table>
<thead>
<tr>
<th>Topic</th>
<th>Abstraction</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data flow optimizations</td>
<td>Graphs</td>
<td>High-level programming without loss of efficiency</td>
</tr>
<tr>
<td></td>
<td>Recurrent equations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fixed-point</td>
<td></td>
</tr>
<tr>
<td>Parallelism and locality</td>
<td>Integer linear programming</td>
<td>Hide parallelism and locality from programmers</td>
</tr>
<tr>
<td>optimizations</td>
<td>Linear algebra</td>
<td></td>
</tr>
</tbody>
</table>

#### 3. Security of Web Applications

![Diagram of Security of Web Applications]

- **Hacker**
- **Browser**
- **Web App**
- **Database**

- **Evil Input**
- **Confidential information leak**
**SQL Injection Errors**

Hacker -> Browser -> Web App -> Database

Give me Bob's credit card#
Delete all records

**Happy-go-lucky SQL Query**

User supplies: name, password

```
"SELECT UserID, Creditcard FROM CCRec WHERE Name = "
  name " AND PW = "
  password "
```
**Fun with SQL**

“—”: “the rest are comments” in Oracle SQL

```sql
SELECT UserID, CreditCard FROM CCRec
WHERE:
Name = bob AND PW = foo
Name = bob— AND PW = x
Name = bob or 1=1— AND PW = x
Name = bob: DROP CCRec— AND PW = x
```

---

**Dynamic vs. Static Pattern**

Dynamically:

```java
| o = req.getParameter ( ); |
| stmt.executeQuery (o); |
```

Statically:

```java
| p1 = req.getParameter ( ); |
| stmt.executeQuery (p2); |
```

$p_1$ and $p_2$ point to same object?

Pointer alias analysis
In Practice

ParameterParser.java:586
String session.ParameterParser.getRawParameter(String name)

```java
public String getRawParameter(String name)
    throws ParameterNotFoundException {
    String[] values = request.getParameterValues(name);
    if (values == null) {
        throw new ParameterNotFoundException(name + " not found");
    } else if (values[0].length() == 0) {
        throw new ParameterNotFoundException(name + " was empty");
    }
    return values[0];
}
```

ParameterParser.java:570
String session.ParameterParser.getRawParameter(String name, String def)

```java
public String getRawParameter(String name, String def) {
    try {
        return getRawParameter(name);
    } catch (Exception e) {
        return def;
    }
}
```

In Practice (II)

ChallengeScreen.java:194
Element lessons.ChallengeScreen.doStage2(WebSession s)

```java
String user = s.getParameter().getRawParameter("USER", "");
StringBuffer tmp = new StringBuffer();
tmp.append("SELECT cc_type, cc_number from user_data
WHERE userid = ");
tmp.append(user);
tmp.append(";
query = tmp.toString();
Vector v = new Vector();
try
{
    ResultSet results = statement3.executeQuery(query);
    ...
```
Why is Pointer Alias Analysis Hard?

- Unbounded number of dynamically allocated objects
- An indirect write via an unknown pointer can write to all possible locations of the same type.
- Must analyze across procedures
- Must keep track of the calling contexts (exponential)

Vulnerabilities Found in 9 Programs

<table>
<thead>
<tr>
<th></th>
<th>SQL injection</th>
<th>HTTP splitting</th>
<th>Cross-site scripting</th>
<th>Path traversal</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header</td>
<td>0</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>Parameter</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>Cookie</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Non-Web</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9</strong></td>
<td><strong>11</strong></td>
<td><strong>5</strong></td>
<td><strong>5</strong></td>
<td><strong>30</strong></td>
</tr>
</tbody>
</table>
**Automatic Analysis Generation**

- **Programmer:** Security analysis in 10 lines
  - PQL

- **Compiler Writer:**
  - Flow-insensitive
  - Context-sensitive
  - Ptr analysis in 10 lines
  - Datalog

- **Binary Decision Diagrams (BDD):**
  - 1000s of lines
  - 1 year tuning
  - BDD: 10,000s-lines library

**Summary**

<table>
<thead>
<tr>
<th>Topic</th>
<th>Abstraction</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data flow optimizations</td>
<td>Graphs Recurrent equations Fixed-point</td>
<td>High-level programming without loss of efficiency</td>
</tr>
<tr>
<td>Parallelism and locality optimizations</td>
<td>Integer linear programming Linear algebra</td>
<td>Hide parallelism and locality from programmers</td>
</tr>
<tr>
<td>Pointer alias analysis</td>
<td>Program Query Language Logic database (Datalog) Binary decision diagrams (BDDs)</td>
<td>Automate error-prone security inspection. Illustrates language abstraction</td>
</tr>
</tbody>
</table>

**Natural language programming**

Consumers can automate personal and professional tasks themselves, eliminating dependence on coders.
4. Programming in Natural Language

- Today’s software
  - All possible combinations are hardcoded
  - Users choose from a menu of choices
  - Limited choices to keep the interface manageable

- Can consumers code in the highest programming language?
  - Natural language!

- What kind of programs?
  - Not C, Java, Python
  - Many useful APIs: virtual assistants
  - Our target: connect virtual assistant primitives

---

Asthma Patient

- "If Bob’s peak flow-meter drops below 180L/min, notify me."
- "When the ragweed pollen count is high and Bob is running, warn him."
- "Log where I am when I use my inhaler."
- "Let my Dad know if I am at the hospital."

---

CS243: Introduction 42 M. Lom
**Natural Language Programming**

*When I use my inhaler, get my GPS location, if it is not home, write it to logfile in Box.*

- Event-driven program
- Multiple function calls
- Parameter passing
- Filters on values

---

**Almond: 1st Programmable Virtual Assistant**

Natural Language Commands

```
monitor @inhaler-use(),
get my GPS location, if it is not home, => @GPS(), location <> "home"
write it to logfile in Box.
```

Semantic parsing

LUInet

Linguistic User Interface Neural Network

Formal target language

ThingTalk

Thingpedia

Giovanni, Ramesh, Xi, Fischer, Lam, WWW 2017
Thingpedia: Encyclopedia of Things

- Interoperability
  - API signatures + corresponding NL
- Open repository
  - Available to all assistants

> 60 devices / 200 functions

<table>
<thead>
<tr>
<th>Natural Language</th>
<th>API Signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHEN @Stanford tweets</td>
<td>Monitor (@home_timeline(), ...) author==&quot;Stanford&quot;)</td>
</tr>
<tr>
<td>GET tweets matching “#Cardinal”</td>
<td>search(...), contains (hashtag, ...)</td>
</tr>
<tr>
<td>DO tweet “Stanford won!”</td>
<td>post (status)</td>
</tr>
</tbody>
</table>

Real Natural Language Input

When I tweet, share the text on LinkedIn

Share my tweets on my LinkedIn
Whenever I tweet, post the same message on LinkedIn
Post all my tweets on LinkedIn
Experimental Results

- Off the shelf DecaNLP model [McCann et al, 2018]
- 62% on real data
  - Compositional sentences
  - No training with real data
- Accuracy can be improved with more templates + paraphrases

Summary

<table>
<thead>
<tr>
<th>Topic</th>
<th>Abstraction</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data flow optimizations</td>
<td>Graphs</td>
<td>High-level programming without loss of efficiency</td>
</tr>
<tr>
<td></td>
<td>Recurrent equations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Fixed-point</td>
<td></td>
</tr>
<tr>
<td>Parallelism and locality</td>
<td>Integer linear programming</td>
<td>Hide parallelism and locality from programmers</td>
</tr>
<tr>
<td>optimizations</td>
<td>Linear algebra</td>
<td></td>
</tr>
<tr>
<td>Pointer alias analysis</td>
<td>Program Query Language Logic database</td>
<td>Automate error-prone security inspection. Illustrates language abstraction</td>
</tr>
<tr>
<td></td>
<td>(Datalog)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Binary decision diagrams (BDDs)</td>
<td></td>
</tr>
<tr>
<td>Natural language programming</td>
<td>Neural networks</td>
<td>Consumers can automate personal and professional tasks themselves, eliminating dependence on coders.</td>
</tr>
</tbody>
</table>

CS243: Introduction
## Tentative Course Schedule

<table>
<thead>
<tr>
<th></th>
<th>Course introduction</th>
<th>Data flow optimization: introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Data flow optimization</td>
<td>Data-flow analysis: theoretic foundation</td>
</tr>
<tr>
<td>4</td>
<td>Optimization: constant propagation</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Optimization: redundancy elimination</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Register allocation</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Instruction Scheduling</td>
<td>Non-numerical code scheduling</td>
</tr>
<tr>
<td>9</td>
<td>Software pipelining</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Loop Transformation</td>
<td>Parallelization</td>
</tr>
<tr>
<td>11</td>
<td>Loop transformations</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Pipelined Parallelism</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Pointer alias analysis</td>
<td>Algorithm</td>
</tr>
<tr>
<td>14</td>
<td>BDDs in pointer analysis</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Satisfiability modulo theories</td>
<td>Introduction</td>
</tr>
<tr>
<td>16</td>
<td>Natural language programming</td>
<td>Neural network application</td>
</tr>
<tr>
<td>17</td>
<td>Garbage Collection</td>
<td>Algorithms</td>
</tr>
</tbody>
</table>

## Homework

- **Due Wednesday (no need to hand in)**
- Read Chapter 9.1 for introduction of the optimizations
- Work out the example on page 10-12 in this handout.