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, Systolic arrays
- 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
Interactive Instruction

• Compilers are not about memorizing facts
  – Open-book examinations
• Goal: teach how to derive the concepts
  – So you can apply to new problems
  – Lectures are interactive
    – Please come to class
    – The slides may miss main points to be emphasized in class!
      • These slides supplement lectures
      • They are not self contained!
      • They may contain mistakes, corrected 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

• Redundancy elimination
  – High-level programming languages introduce a lot of redundancies in programs that programmers are not aware of.

• 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;
        }
    }
}
```

Quiz: What is the most important optimization for this example?
Code Generated by the Front End

\[
i := n - 2
\]

\[
S5: \text{if } i < 0 \text{ goto s1}
\]

\[
j := 0
\]

\[
s4: \text{if } j > i \text{ goto s2}
\]

\[
t1 = 4 * j
\]

\[
t2 = &A
\]

\[
t3 = t2 + t1
\]

\[
t4 = *t3 \quad ;A[j]
\]

\[
t5 = j + 1
\]

\[
t6 = 4 * t5
\]

\[
t7 = &A
\]

\[
t8 = t7 + t6
\]

\[
t9 = *t8 \quad ;A[j+1]
\]

\[
\text{if } t4 \leq t9 \text{ goto s3}
\]

\[
t10 = 4 * j
\]

\[
t11 = &A
\]

\[
t12 = t11 + t10
\]

\[
temp = *t12 \quad ;temp=A[j]
\]

\[
t13 = j + 1
\]

\[
t14 = 4 * t13
\]

\[
t15 = &A
\]

\[
t16 = t15 + t14
\]

\[
t17 = *t16 \quad ;A[j+1]
\]

\[
t18 = 4 * j
\]

\[
t19 = &A
\]

\[
t20 = t19 + t18 \quad ;&A[j]
\]

\[
*\text{t20} = t17 \quad ;A[j] = A[j+1]
\]

\[
t21 = j + 1
\]

\[
t22 = 4 * t21
\]

\[
t23 = &A
\]

\[
t24 = t23 + t22
\]

\[
*\text{t24} = \text{temp} \quad ;A[j+1] = \text{temp}
\]

\[
s3: \quad j = j + 1
\]

\[
s2: \quad i = i - 1
\]

\[
s1: 
\]

\[
(t4 = *t3 \text{ means read memory at address in } t3 \text{ and write to } t4:
\]

\[
*\text{t20} = t17: \text{store value of } t17 \text{ into memory at address in } t20)
\]
After Optimization

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

to all the scalar and temp. variables

These traditional optimizations can make a big difference!

\[
\begin{align*}
i &= n-2 \\
t27 &= 4*i \\
t28 &= &A \\
t29 &= t27 + t28 \\
t30 &= t28 + 4 \\
S5: &i f t29 < t28 \text{ goto } s1 \\
& t25 = t28 \\
& t26 = t30 \\
s4: &i f t25 > t29 \text{ goto } s2 \\
& t4 = *t25 \quad \text{ ;A[j]} \\
& t9 = *t26 \quad \text{ ;A[j+1]} \\
& i f t4 \leq t9 \text{ goto } s3 \\
& \text{ temp } = *t25 \quad \text{ ;temp=A[j]} \\
& t17 = *t26 \quad \text{ ;A[j+1]} \\
& *t25 = t17 \quad \text{ ;A[j]=A[j+1]} \\
& *t26 = \text{ temp } \quad \text{ ;A[j+1]=temp} \\
s3: & t25 = t25 + 4 \\
& t26 = t26 + 4 \\
& \text{ goto S4} \\
S2: & t29 = t29 - 4 \\
& \text{ goto s5} \\
s1: &
\end{align*}
\]
## 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</td>
<td>High-level programming without loss of efficiency</td>
</tr>
<tr>
<td></td>
<td>Fixed-point</td>
<td></td>
</tr>
</tbody>
</table>

Consumers can automate personal and professional tasks themselves, eliminating dependence on coders.
2. High-Performance Computing (Machine Learning)

- 7 ExaFLOPS = 10^{18} FLOPS
  - 60 Million Parameters
  - Microsoft ResNet Superhuman Image Recognition
  - 2015

- 20 ExaFLOPS = 10^{18} FLOPS
  - 300 Million Parameters
  - Baidu Deep Speech 2 Superhuman Voice Recognition
  - 2016

- 100 ExaFLOPS = 10^{18} FLOPS
  - 8700 Million Parameters
  - Google Neural Machine Translation Near Human Language Translation
  - 2017

1 ExaFLOPS = 10^{18} FLOPS
Nvidia Volta GV100 GPU

21B transistors
815 mm²
1455 Mhz

80 Stream Multiprocessors (SM)

https://wccftech.com/nvidia-volta-tesla-v100-cards-detailed-150w-single-slot-300w-dual-slot-gv100-powered-pcie-accelerators/
In Each SM

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

Tensor Cores
D = A x B + C; A, B, C, D are 4x4 matrices
4 x 4 x 4 matrix processing array
1024 floating point ops / clock

FP32: 15 TFLOPS
FP64: 7.5 TFLOPS
Tensor: 120 TFLOPS

https://wccftech.com/nvidia-volta-tesla-v100-cards-detailed-150w-single-slot-300w-dual-slot-gv100-powered-pcie-accelerators/
Google TPU-v4 chips, 2022

TPU v4 chip

Matrix multiplication unit: 128 x 128 multiply/accumulators in a systolic array

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak compute per chip</td>
<td>275 teraflops</td>
</tr>
<tr>
<td>Min/mean/max power</td>
<td>90/170/192 W</td>
</tr>
<tr>
<td>TPU pod size</td>
<td>4096 chips</td>
</tr>
<tr>
<td>Peak compute per pod</td>
<td>1.1 exaflops</td>
</tr>
</tbody>
</table>

https://cloud.google.com/tpu/docs/system-architecture-tpu-vm
Google TPU-v4 System

Google PaLM: 540B parameters in large language models
Using 6144 TPU v4 chips (1.7 exaflops)
Compilers and Architectures for Numerical Applications

- Numerical applications have very regular structures, lots of parallelism
- Minimizing data communication is key

- Matrix Multiplication

```c
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        for (k = 0; k < n; k++) {
            Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
        }
    }
}
```

- Lots of data reuse: \( n^3 \) computation, \( 3n^2 \) data points
Blocking for Matrix Multiplication

\[
\begin{pmatrix}
\text{1000} & \text{1000} & \text{1000}
\end{pmatrix}
\begin{pmatrix}
\text{32}
\end{pmatrix}
\text{Data Accessed: 1002000}
\]

\[
\begin{pmatrix}
\text{32}
\end{pmatrix}
\begin{pmatrix}
\text{1000}
\end{pmatrix}
\text{Data Accessed: 65024}
\]
Experimental Results

With Blocking
Without Blocking

Speedup

Processors
Geometric Program Representation

- Model the data flow geometrically
  - 3D loop nest → 3D cube

- Matrix Multiplication

```java
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        for (k = 0; k < n; k++) {
            Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
        }
    }
}
```

- Abstraction used in systolic arrays and locality/parallelism optimizations
## 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, Systolic arrays</td>
</tr>
</tbody>
</table>
3. Non-Numerical Apps: Understanding Pointers

Application: Security of Web Applications

Hacker \rightarrow Browser \rightarrow Web App \rightarrow 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

```
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;
    }
}
```
String user = s.getParser().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>Type</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>Total</td>
<td>9</td>
<td>11</td>
<td>5</td>
<td>5</td>
<td>30</td>
</tr>
</tbody>
</table>
Automatic Analysis Generation

Programmer: Security analysis in 10 lines

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

1000s of lines
1 year tuning

Domain specific language

Logic database programming language

Exponential state operations

PQL

Datalog

Binary Decision Diagrams (BDD)

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</td>
<td>Integer linear programming, Linear algebra</td>
<td>Hide parallelism and locality from programmers</td>
</tr>
<tr>
<td>optimizations</td>
<td></td>
<td>Systolic arrays</td>
</tr>
<tr>
<td>Pointer alias analysis</td>
<td>Program Query Language, Logic database (Datalog),</td>
<td>Automate error-prone security inspection.</td>
</tr>
<tr>
<td></td>
<td>Binary decision diagrams (BDDs)</td>
<td>Illustrate language abstraction</td>
</tr>
</tbody>
</table>
4. Programming in Natural Language

- Highest level programming language: natural language
- Purposes of natural language programming?
4. Programming in Natural Language

• Highest level programming language: natural language

• Purposes of natural language programming
  – Improve the productivity of programmers
    • OpenAI Codex, Github Co-pilot
    • Issues: precision, subtle errors, plagiarism
  – Give consumers the ability to program!
    • e.g. database queries, simple scripts
Example of Consumer Programming: Asthma Patient

people
“if Bob’s peak flow-meter drops below 180L/min notify me”

devices
“log where I am when I use my inhaler”

location
“Let my Dad know if I am at the hospital”

environment
“when the ragweed pollen count is high and Bob is running, warn him”
Large Language Models (LLMs)

GPT-3 in 2020
- Trained on 45 TB of texts
- 175 Billion parameters
- College essays

ChatGPT, Dec 2022
- Fine-tuned GPT3
- Supervised learning
- User reinforcement
User: I'd like to buy a plane ticket to New York.

Agent: *Sure, I can help you with that.*

*Can you tell me what day and time you would like to fly?*

User: July 7th, 9am

Agent: *I can book a flight for you departing at 9am on July 7th.*

*Would you like me to book that for you?*

User: What is the flight number?

Agent: *The flight number is UA123.*

Fact check:

United UA 123

London to Washington
Semantic Parsing ➔ Executable Code

- **ThingTalk** is a programming language
- **Complete:**
  - Anything the virtual assistant can do can be represented in ThingTalk
  - Beyond queries (APIs, control flow, etc)
- **Simple:** Execute the code to get the results

**User Utterance:**
Tell me the top 3 restaurants either near the airport or the Fisherman’s Wharf

**Neural Semantic Parser (Codex)**

**Meaning**

**ThingTalk**

```plaintext
sort(desc rating of @com.yelp()
filter location == new Location("airport")
| | location == new Location("fisherman’s wharf"))[1:3]
```
Hello, I am Genie, your assistant.

I’d recommend Chez TJ in Mountain View. The distance to Chez TJ is 7.5 miles.
The distance to Left Bank is 2.8 miles.

Which day would you like to book Left Bank?
The price is moderate.

Is it expensive?

OK

Which day would you like to book Left Bank?

November 2, 7pm, for 2 people.

Could you confirm that you want a table for 2 people at Left Bank on November 2 at 7pm?

Yes.

I booked Left Bank for 2 on November 2 at 7pm.
Cost of Data Collection

- Machine Learning requires training data
- Annotation of conversations is expensive, difficult, & error-prone
- A more cost effective approach:
  - Synthesize (conversational) data
  - Add a few-shot of training data
  - Fine-tune a language model
Synthesis Sounds Really Hard!

One database software $\rightarrow$ one conversational synthesizer software
Based only on the schema, applicable to any agent in the domain!
Comparison with Commercial Assistants

Examples of Long-Tail Questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Alexa</th>
<th>Google</th>
<th>Siri</th>
<th>Genie</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show me restaurants rated at least 4 stars with at least 100 reviews</td>
<td>✔</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Show restaurants in San Francisco rated higher than 4.5</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>What is the highest rated Chinese restaurant near Stanford?</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>How far is the closest 4 star restaurant?</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Who works for W3C and went to Oxford?</td>
<td></td>
<td></td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Who worked for Google and lives in Palo Alto?</td>
<td></td>
<td>✔</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Who graduated from Stanford and won a Nobel prize?</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Who worked for at least 3 companies?</td>
<td></td>
<td></td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Show me hotels with checkout time later than 12PM</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Which hotel has a pool in this area?</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>
## 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>1970-1980s</td>
<td></td>
<td></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>1980-1990s</td>
<td></td>
<td>Systolic arrays.</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. Illustartive of language abstraction.</td>
</tr>
<tr>
<td>1990-2000s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Natural language programming</td>
<td>Neural networks, Large language models, NLP training data synthesis</td>
<td>End-user programming. More robust NL interface.</td>
</tr>
<tr>
<td>2010-2020s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Best of time-tested concepts in compilers!
## Tentative Course Schedule

<table>
<thead>
<tr>
<th></th>
<th>Course Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Data Flow Optimizations</td>
</tr>
<tr>
<td></td>
<td>Data-flow analysis: introduction</td>
</tr>
<tr>
<td>3</td>
<td>Data-flow analysis: theoretic foundation</td>
</tr>
<tr>
<td>4</td>
<td>Optimization: constant propagation</td>
</tr>
<tr>
<td>5</td>
<td>Optimization: redundancy elimination</td>
</tr>
<tr>
<td>6</td>
<td>Machine Dependent Optimizations</td>
</tr>
<tr>
<td></td>
<td>Register allocation</td>
</tr>
<tr>
<td>7</td>
<td>Non-numerical code scheduling</td>
</tr>
<tr>
<td>8</td>
<td>Software pipelining</td>
</tr>
<tr>
<td>9</td>
<td>Loop Transformations</td>
</tr>
<tr>
<td></td>
<td>Parallelization</td>
</tr>
<tr>
<td>10</td>
<td>Loop transformations</td>
</tr>
<tr>
<td>11</td>
<td>Pipelined parallelism</td>
</tr>
<tr>
<td>12</td>
<td>Pointer Alias Analysis</td>
</tr>
<tr>
<td></td>
<td>Algorithm</td>
</tr>
<tr>
<td>13</td>
<td>BDDs in pointer analysis</td>
</tr>
<tr>
<td>14</td>
<td>Satisfiability Modulo Theories</td>
</tr>
<tr>
<td></td>
<td>Introduction</td>
</tr>
<tr>
<td>15</td>
<td>Garbage Collection</td>
</tr>
<tr>
<td></td>
<td>Advanced optimizations</td>
</tr>
<tr>
<td>16</td>
<td>Natural Language Programming</td>
</tr>
<tr>
<td></td>
<td>Neural networks</td>
</tr>
</tbody>
</table>

---

*CS243: Introduction*  
48  
M. Lam
Homework

• **Due Wednesday (no need to hand in)**

• Read Chapter 9.1 for introduction of the optimizations

• Work out the example on page 11-13 in this handout.