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
Course Staff

• Faculty
  – John Whaley
  – Dror Maydan
  – Jeff Ullman

• TAs
  – Ross Daly
  – Yancheng Ou

• Website
  – https://cs243.stanford.edu

• Piazza
  – Sign up here: http://www.piazza.com/stanford/spring2022/cs243

• Gradiance
  – Sign up at http://www.gradiance.com/services
  – Class Token: A1F858D4
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

Programs
- Static statements
- Dynamic execution
- Generated code

Abstractions
- Graphs
- Fixed-point solutions
- Linear Integer programs
- Linear algebra
- Logic databases
- Binary Decision Diagrams (BDD)
- Neural networks

Solutions
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
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;
        }
```

Code Generated by the Front End

\[ \begin{align*}
i &:= n-2 \\
s5: \text{ if } i<0 \text{ goto s1} \\
j &:= 0 \\
s4: \text{ if } j>i \text{ goto s2} \\
t1 &= 4*j \\
t2 &= \&A \\
t3 &= t2+t1 \\
t4 &= *t3 \quad ;A[j] \\
t5 &= j+1 \\
t6 &= 4*t5 \\
t7 &= \&A \\
t8 &= t7+t6 \\
t9 &= *t8 \quad ;A[j+1] \\
\text{if } t4 \leq t9 \text{ goto s3} \\
t10 &= 4*j \\
t11 &= \&A \\
t12 &= t11+t10 \\
temp &= *t12 \quad ;\text{temp}=A[j] \\
\end{align*} \]

\[ \begin{align*}
t13 &= j+1 \\
t14 &= 4*t13 \\
t15 &= \&A \\
t16 &= t15+t14 \\
t17 &= *t16 \quad ;A[j+1] \\
t18 &= 4*j \\
t19 &= \&A \\
t20 &= t19+t18 \quad ;\&A[j] \\
t21 &= j+1 \\
t22 &= 4*t21 \\
t23 &= \&A \\
t24 &= t23+t22 \\
t25 &= \text{temp} \quad ;A[j+1]=\text{temp} \\
s3: \quad j &= j+1 \\
goto S4 \\
s2: \quad i &= i-1 \\
goto s5 \\
s1: \\
(t4=*t3 \text{ means read memory at address in } t3 \text{ and write to } t4, \\
*t20=t17 \text{ means store value of } t17 \text{ into memory at address in } t20) \]

J. Whaley
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!

```c
i = n-2
```
```
t27 = 4*i
t28 = &A
t29 = t27+t28
t30 = t28+4
```
```
s5: if t29 < t28 goto s1
```
```
t25 = t28
t26 = t30
```
```
s4: if t25 > t29 goto s2
```
```
t4 = *t25 ;A[j]
t9 = *t26 ;A[j+1]
if t4 <= t9 goto s3
```
```
temp = *t25 ;temp=A[j]
t17 = *t26 ;A[j+1]
*t25 = t17 ;A[j]=A[j+1]
*t26 = temp ;A[j+1]=temp
```
```
s3: 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>
</tbody>
</table>

---
2. High-Performance Computing (Machine Learning)

7 ExaFLOPS
60 Million Parameters

20 ExaFLOPS
300 Million Parameters

100 ExaFLOPS
8700 Million Parameters

Microsoft ResNet
Superhuman Image Recognition

2015

Baidu Deep Speech 2
Superhuman Voice Recognition

2016

Google Neural Machine Translation
Near Human Language Translation

2017

1 ExaFLOPS = $10^{18}$ FLOPS
Nvidia Hopper GPU Architecture

80B transistors
900 mm²
1650 MHz

132 Streaming Multiprocessors (SM)
528 Tensor cores
16,896 CUDA cores

https://www.techpowerup.com/gpu-specs/h100-pcie.c3899
In Each SM

128 FP32 cores
128 Int cores
64 FP64 cores
4 Tensor cores

Tensor Cores
D = A \times B + C; \ A, B, C, D \ are \ 4\times4 \ matrices
4 \times 4 \times 4 \ matrix \ processing \ array
1024 \ floating \ point \ ops \ / \ clock

FP64: 30 TFLOPS
FP32: 500 TFLOPS
FP16: 1000 TFLOPS
FP8/INT8 Tensor: 2000 TFLOPS

https://wccftech.com/nvidia-volta-tesla-v100-cards-detailed-150w-single-slot-300w-dual-slot-gv100-powered-pcie-accelerators/
Blocking for Matrix Multiplication

\[
\begin{align*}
&\begin{array}{ccc}
1000 & & \\
\end{array}
= \\
\begin{array}{ccc}
& & 1000 \\
\end{array} \\
1000 & & \\
& & \\
= \\
\begin{array}{ccc}
1000 & & \\
\end{array}
\times \\
\begin{array}{ccc}
& & \\
\end{array} \\
\begin{array}{ccc}
& & 1000 \\
\end{array}
\times \\
\begin{array}{ccc}
1000 & & \\
\end{array} \\
\begin{array}{ccc}
32 & & \\
\end{array}
= \\
\begin{array}{ccc}
& & \\
\end{array} \\
\begin{array}{ccc}
1000 & & \\
\end{array} \\
32 & & \\
& & \\
= \\
\begin{array}{ccc}
1000 & & \\
\end{array}
\times \\
\begin{array}{ccc}
& & 1000 \\
\end{array} \\
\begin{array}{ccc}
& & \\
\end{array} \\
\begin{array}{ccc}
& & \\
\end{array}
\end{align*}
\]

Data Accessed

- 1000 x 1000: 1002000
- 32 x 1000: 65024
Experimental Results

- With Blocking
- Without Blocking

Speedup vs. Processors
Blocking with Matrix Multiplication

- Original program
  ```
  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];
      }
    }
  }
  ```

- Stripmine 2 outer loops
  ```
  for (ii = 0; ii < n; ii = ii+B) {
    for (i = ii; i < min(n,ii+B); i++) {
      for (jj = 0; jj < n; jj = jj+B) {
        for (j = jj; j < min(n,jj+B); j++) {
          Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
        }
      }
    }
  }
  ```

- Permute loops
  ```
  for (ii = 0; ii < n; ii = ii+B) {
    for (jj = 0; jj < n; jj = jj+B) {
      for (i = ii; i < min(n,ii+B); i++) {
        for (j = jj; j < min(n,jj+B); j++) {
          Z[i,j] = Z[i,j] + X[i,k]*Y[k,j];
        }
      }
    }
  }
  ```
# 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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parallelism and locality optimizations</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>
</tbody>
</table>
3. 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

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:

\[
o = \text{req.getParameter}();
\]

\[
\text{stmt.executeQuery}(o);
\]

Statically:

\[
p_1 = \text{req.getParameter}();
\]

\[
\text{stmt.executeQuery}(p_2);
\]

\[p_1 \text{ and } p_2 \text{ point to same object?}\]

Pointer alias analysis
In Practice

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

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)

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></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: PQL
- Security analysis in 10 lines

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

1000s of lines
1 year tuning

Binary Decision Diagrams (BDD)
BDD: 10,000s-lines library

Domain specific language
Logic database programming language
Exponential state operations
## 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>
<tr>
<td></td>
<td>Course Introduction</td>
<td>Data Flow Optimizations</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>2</td>
<td>Data-flow analysis: introduction</td>
<td>Data-flow analysis: theoretic foundation</td>
</tr>
<tr>
<td>3</td>
<td>Optimization: constant propagation</td>
<td>Optimization: redundancy elimination</td>
</tr>
<tr>
<td>4</td>
<td>Pointer Alias Analysis</td>
<td>Algorithm</td>
</tr>
<tr>
<td>5</td>
<td>BDDs in pointer analysis</td>
<td>Machine Dependent Optimizations</td>
</tr>
<tr>
<td>6</td>
<td>Register allocation</td>
<td>Instruction scheduling</td>
</tr>
<tr>
<td>7</td>
<td>Software pipelining</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>8</td>
<td>Dependence Analysis</td>
<td>Neural network compilers</td>
</tr>
<tr>
<td>9</td>
<td>Loop transformations</td>
<td>Satisfiability Modulo Theories</td>
</tr>
<tr>
<td>10</td>
<td>Software pipelining</td>
<td>Introduction</td>
</tr>
<tr>
<td>11</td>
<td>Machine Learning</td>
<td>Runtime Systems</td>
</tr>
<tr>
<td>12</td>
<td>Neural network compilers</td>
<td>Dynamic (JIT) Compilation</td>
</tr>
</tbody>
</table>
Homework

• Read Chapter 9.1 for introduction of the optimizations

• Work out the example on page 10-12 in this handout.