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;
        }
    }
}
```


**Code Generated by the Front End**

\[
\begin{align*}
\text{i := n-2} & & \text{t13 = j+1} \\
\text{S5: if i<0 goto s1} & & \text{t14 = 4*t13} \\
\text{j := 0} & & \text{t15 = &A} \\
\text{s4: if j>i goto s2} & & \text{t16 = t15+t14} \\
\text{t1 = 4*j} & & \text{t17 = *t16}; \text{A[j+1]} \\
\text{t2 = &A} & & \text{t18 = 4*j} \\
\text{t3 = t2+t1} & & \text{t19 = &A} \\
\text{t4 = *t3}; \text{A[j]} & & \text{t20 = t19+t18}; \text{&A[j]} \\
\text{t5 = j+1} & & \text{t21 = j+1} \\
\text{t6 = 4*t5} & & \text{t22 = 4*t21} \\
\text{t7 = &A} & & \text{t23 = &A} \\
\text{t8 = t7+t6} & & \text{t24 = t23+t22} \\
\text{t9 = *t8}; \text{A[j+1]} & & \text{*t24 = temp}; \text{A[j+1]=temp} \\
\text{if t4 <= t9 goto s3} & & \text{s3: j = j+1} \\
\text{t10 = 4*j} & & \text{goto S4} \\
\text{t11 = &A} & & \text{S2: i = i-1} \\
\text{t12 = t11+t10} & & \text{temp = *t12}; \text{temp=A[j]} \\
\text{temp = *t12}; \text{temp=A[j]} & & \text{goto s5} \\
\text{s1:} & & \text{s1:}
\end{align*}
\]

(t4=*t3 means read memory at address in t3 and write to t4:
* t20=t17: store value of t17 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!

\[
i = n-2
\]
\[
t27 = 4*i
\]
\[
t28 = &A
\]
\[
t29 = t27+t28
\]
\[
t30 = t28+4
\]
\[
S5: \text{if } t29 < t28 \text{ goto s1}
\]
\[
t25 = t28
\]
\[
t26 = t30
\]
\[
s4: \text{if } t25 > t29 \text{ goto s2}
\]
\[
t4 = *t25 \quad ;A[j]
\]
\[
t9 = *t26 \quad ;A[j+1]
\]
\[
\text{if } t4 <= t9 \text{ goto s3}
\]
\[
temp = *t25 \quad ;\text{temp}=A[j]
\]
\[
t17 = *t26 \quad ;A[j+1]
\]
\[
*t25 = t17 \quad ;A[j]=A[j+1]
\]
\[
*t26 = temp \quad ;A[j+1]=\text{temp}
\]
\[
s3: \ t25 = t25+4
\]
\[
t26 = t26+4
\]
\[
goto S4
\]
\[
S2: \ t29 = t29-4
\]
\[
goto s5
\]
\[
s1:
\]
## 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>

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

- 7 ExaFLOPS
  - 60 Million Parameters
  - 2015
  - Microsoft ResNet
  - Superhuman Image Recognition

- 20 ExaFLOPS
  - 300 Million Parameters
  - 2016
  - Baidu Deep Speech 2
  - Superhuman Voice Recognition

- 100 ExaFLOPS
  - 8700 Million Parameters
  - 2017
  - Google Neural Machine Translation
  - Near Human Language Translation

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

\[ D = A \times B + C; \ A, B, C, D \text{ are } 4 \times 4 \text{ matrices} \]

- 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/
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: N²
- Poor sequential / parallel performance without locality optimization
Optimizing for Single Core: Permuting Loops

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:

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);
        }
    }
}

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

Assume cache size < $N^2$
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 for different optimization levels. The graph plots the performance in GFlops against size. The lines represent Naive (no opt), Naive (std opt), Naive (full opt), Permutated, 1-d Tiled (std opt), and 1-d Tiled (full opt).]
Parallel scaling (matrix size 1500)
## 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 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
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:

\[
o = req.getParameter(); \\
stmt.executeQuery(o);
\]

Statically:

\[
p_1 = req.getParameter(); \\
stmt.executeQuery(p_2);
\]

\(p_1\) 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)

    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>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

10,000s
lines library

Binary Decision Diagrams (BDD)

PQL

Datalog

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</td>
<td>Integer linear programming, Linear algebra</td>
<td>Hide parallelism and locality from programmers</td>
</tr>
<tr>
<td>optimizations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pointer alias analysis</td>
<td>Program Query Language (Datalog), BDDs</td>
<td>Automate error-prone security inspection. Illustrate language abstraction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
Carnegie Mellon

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

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

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

environment
“when the ragweed pollen count is high and Bob is running, warn him”
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

“\textbf{When I use my inhaler, get my GPS location, if it is not home, write it to logfile in Box.}"

\begin{verbatim}
monitor @Inhaler-use(), => @GPS(), location <> "home" => @Box-write(file="logfile", data=location)
\end{verbatim}

Giovanni, Ramesh, Xu, 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>WHEN</th>
<th>Natural Language</th>
<th>API Signatures</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Stanford tweets</td>
<td>Monitor (@home_timeline(), ...) author==“Stanford”)</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>tweets matching “#Cardinal”</td>
<td>search(...), contains (hashtag, ...)</td>
</tr>
<tr>
<td>DO</td>
<td>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
**Paradigm Shift from Annotation to Data Synthesis**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Goal</th>
<th>Manual Annotation</th>
<th>Synthesis + Automatic Paraphrases</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>To understand anything the computer can do</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Compositionality</td>
<td>To understand new sentences</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Cost</td>
<td>Correct annotations at scale</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Naturalness &amp; variety</td>
<td>✓ Few-shot</td>
<td>✓ Pretrained-networks</td>
</tr>
</tbody>
</table>
Comparison with Commercial Assistants

Examples of Long-Tail Questions

<table>
<thead>
<tr>
<th>-query</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>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></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>Natural language programming</td>
<td>Neural networks, NLP training data synthesis</td>
<td>End-user programming. Tools to democratize voice interfaces.</td>
</tr>
</tbody>
</table>

# Tentative Course Schedule

<table>
<thead>
<tr>
<th></th>
<th>Course Introduction</th>
<th>Data Flow Optimizations</th>
<th>Machine Dependent Optimizations</th>
<th>Loop Transformations</th>
<th>Pointer Alias Analysis</th>
<th>Satisfiability Modulo Theories</th>
<th>Garbage Collection</th>
<th>Natural Language Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Course Introduction</td>
<td>Data-flow analysis: introduction</td>
<td>Register allocation</td>
<td>Parallelization</td>
<td>Algorithm</td>
<td>BDDs in pointer analysis</td>
<td>Advanced optimizations</td>
<td>Neural networks</td>
</tr>
<tr>
<td>2</td>
<td>Data Flow Optimizations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Data-flow analysis: theoretic foundation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Optimization: constant propagation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Optimization: redundancy elimination</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Machine Dependent Optimizations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Non-numerical code scheduling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Software pipelining</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Loop Transformations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>Loop transformations</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>Pipelined parallelism</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Pointer Alias Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>BDDs in pointer analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Satisfiability Modulo Theories</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Garbage Collection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Natural Language Programming</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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 11-13 in this handout.