CS 526

Advanced
Compiler
Construction

https://charithm.web.illinois.edu/cs526/sp2024/
(slides adapted from Sasa and Vikram)

Goals of the Course

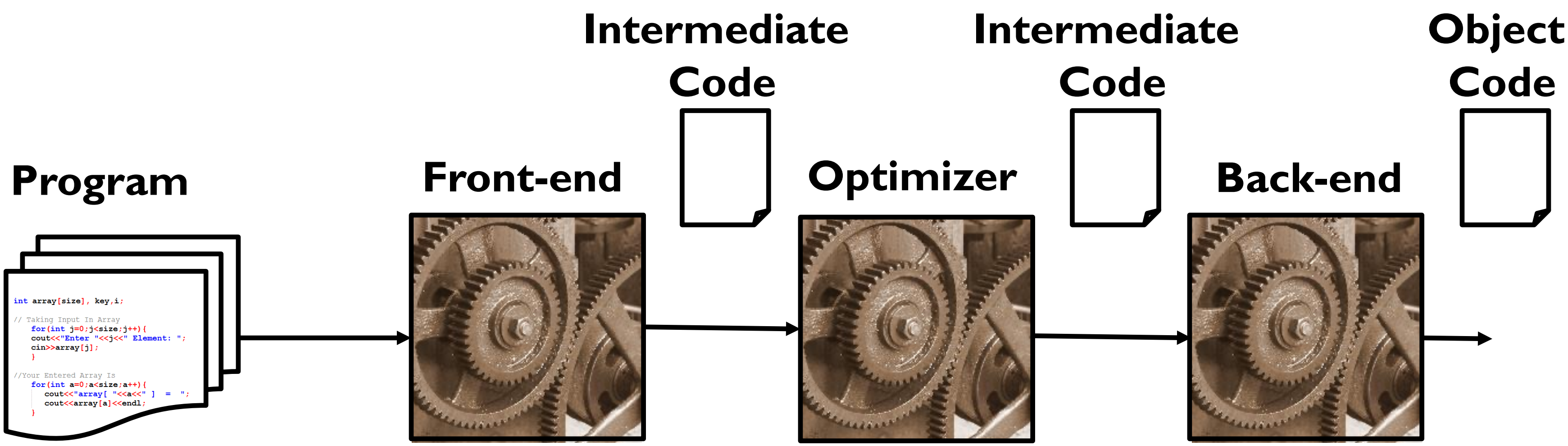
Develop a fundamental understanding of the major approaches to program analysis and optimization

Understand published research on various novel compiler techniques

Solve a significant compiler problem by reading the literature and implementing your solution in LLVM

Learn about current research in compiler technology

Compiler Overview



Optimizer Transformations

- Automatic Parallelization
- Vectorization
- Cache Management
- Performance Modeling

Code Generation

- Source Code Portability
- Back-end Optimizations
- Static Profiling
- Power Management

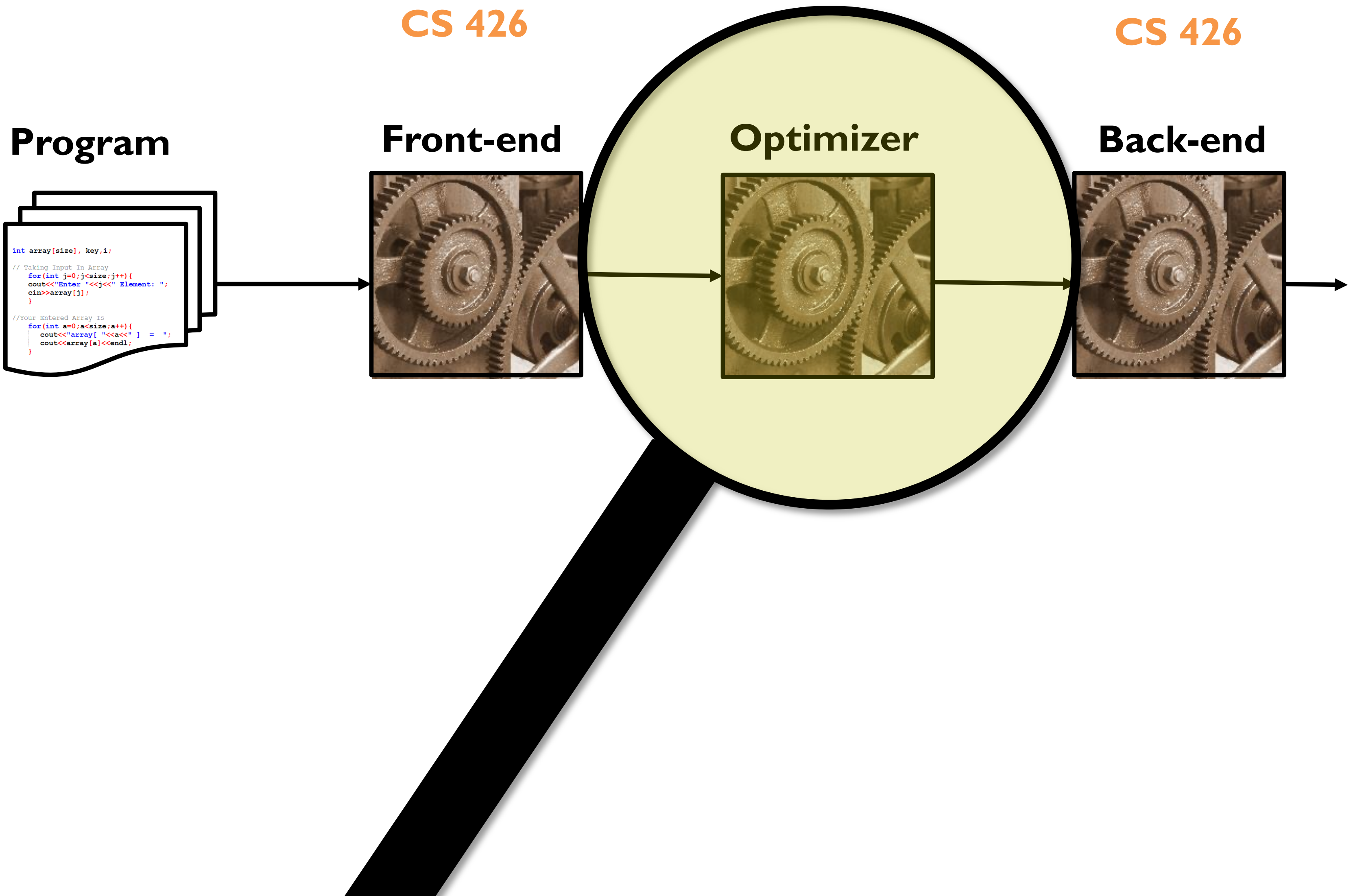
Linking/Loading

- Interprocedural optimization
- Load-time optimization
- Security checking

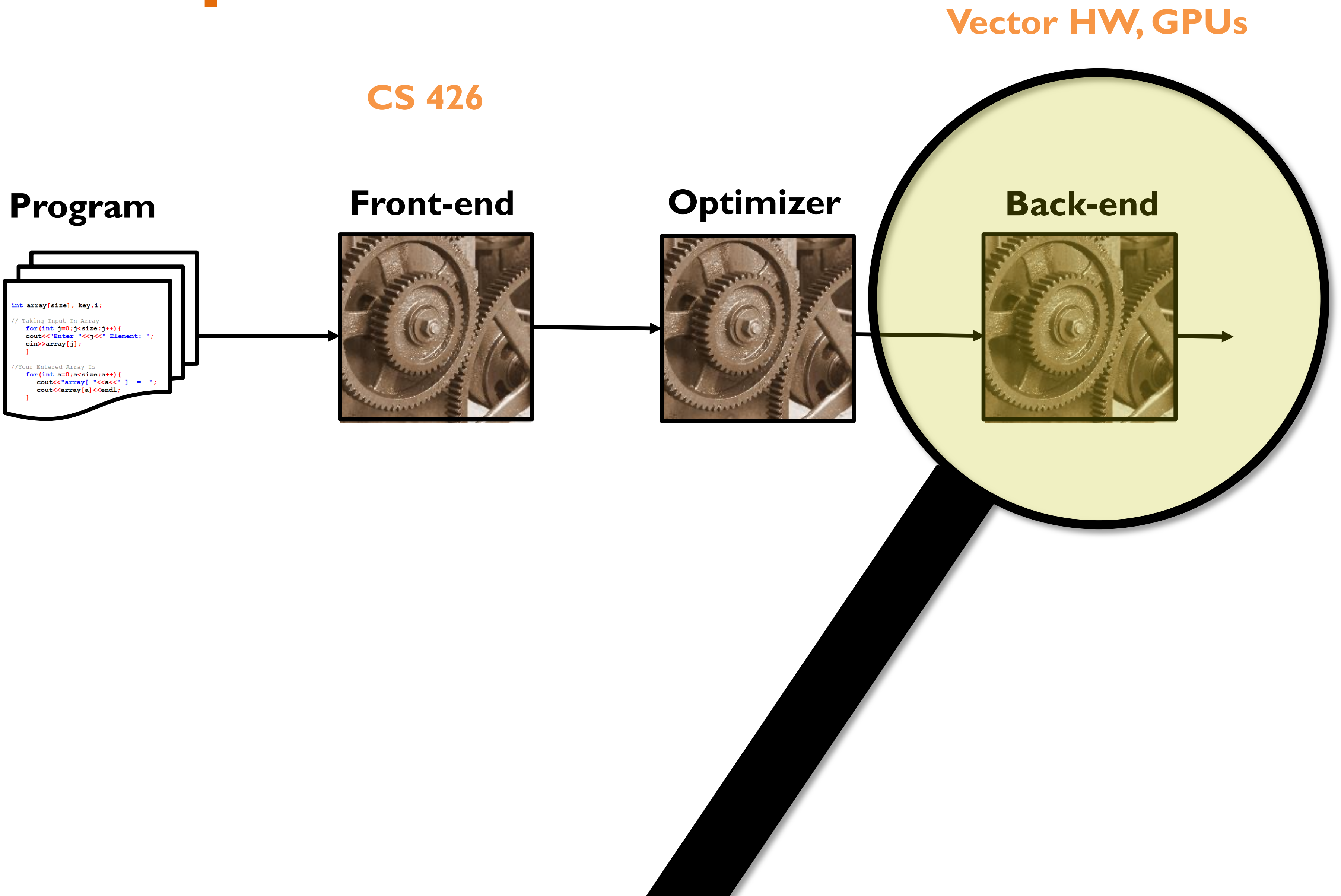
Runtime compilation

- JIT code generation
- Runtime optimization
- Fault tolerance

Compiler Overview



Compiler Overview



Program Analysis +
Program Transformation

Why is Optimization Important?

For source-level programming languages

Liberate programmer from machine-related issues and enable portable programming without unduly sacrificing performance.

John Backus on the first FORTRAN compiler:

> "It is our belief that if FORTRAN, during its first months, were to translate any reasonable scientific program **into an object program only half as fast as its hand-coded** counterpart, then acceptance of our system would be in serious danger."
>
> "To this day I believe that our **emphasis on object program efficiency rather than on language design was basically correct**. I believe that had we failed to produce efficient programs, the widespread use of languages like FORTRAN would have been seriously delayed."

-- John Backus, Fortran I, II and III, Annals of the History of Computing}, vol. 1, no. 1, July 1979

Why is Optimization Important?

For expressive language features

Allow programmer to focus on clean, easy-to-understand programs; avoid detailed hand-optimizations:

- **Expression simplification:** Constant folding, associativity, commutativity
- **Redundancy elimination:** Loop-invariant code motion, common subexpressions, equivalent subexpressions
- **Dead code elimination:** Unreachable code, unused computations
- **Control flow simplification:** Branch folding, branch elimination
- **Procedure call elimination:** Single-use functions, frequent function calls
- **Bounds check elimination:** Array expressions

Why is Optimization Important?

Because Moore's Law is Dead

Intelligent Machines

DARPA has an ambitious $1.5 billion plan to reinvent electronics

The US military agency is worried the country could lose its edge in semiconductor chips with the end of Moore's Law.

by Martin Giles July 30, 2018

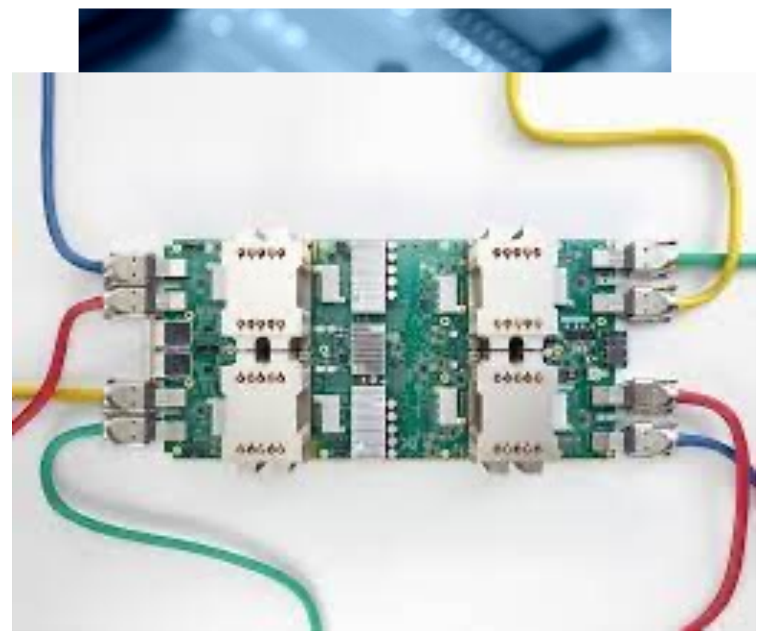
Last year, the Defense Advanced Research Projects Agency (DARPA), which funds a range of blue-sky research efforts relevant to the US military, launched a $1.5 billion, five-year program known as the Electronics Resurgence Initiative (ERI) to support work on advances in chip technology. The agency has just unveiled the first set of research teams selected to explore unproven but

How are we going to leverage new post-Moore architectures?

Why is Optimization Important?

For portable performance

Maintain performance across a wide range of computing devices that include CPUs, GPUs and various Domain Specific Accelerators

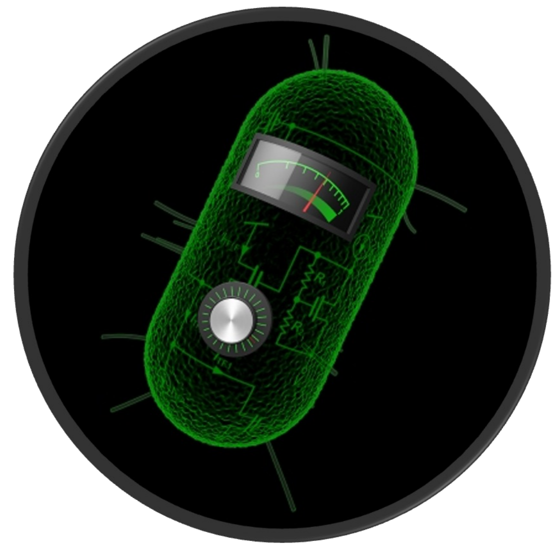


Why is Optimization Important?

For new applications

Wearable computing (e-textiles)

Analog nano-computing (Bio)



Self Driving Cars

Edge intelligence

Why is Optimization Important?

To Understand

In discussing any optimization, look for three properties:

Safety — Does it change the results of the program?
What opportunities exist that are safe?

Profitability — Is it expected to speed up execution?

Optimality — How can we find the best optimization?
Or come to the close it?

Why is Optimization Important?

To Understand

In discussing any optimization, look for three properties:

Safety — Transformation Space (Analysis)

Profitability — Cost Model (Analysis)

Optimality — Optimization Strategy (Transformation)

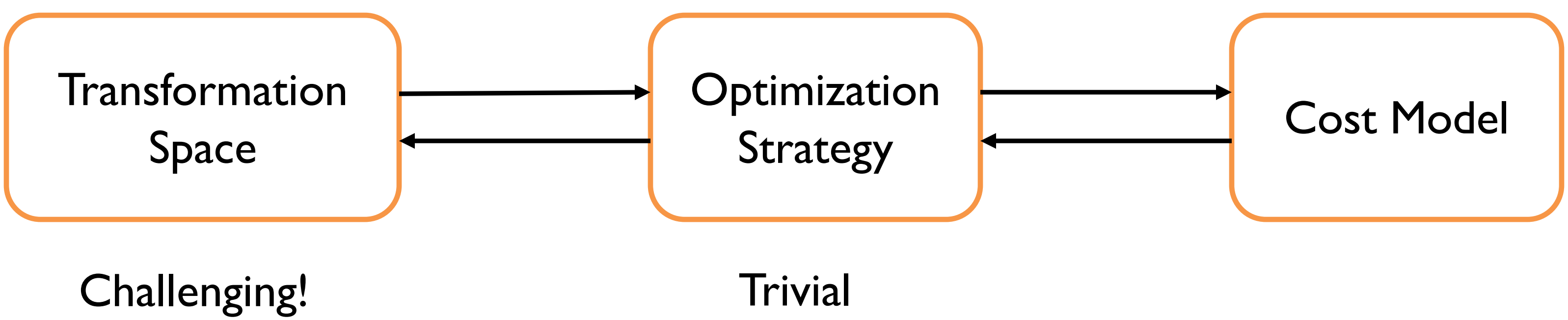
Type I Optimizations

Optimizations that are generally always profitable (e.g., Dead Code Elimination, Constant Propagation)

Safety — Transformation Space (Analysis)

Profitability — Cost Model (Analysis)

Optimality — Optimization Strategy (Transformation)



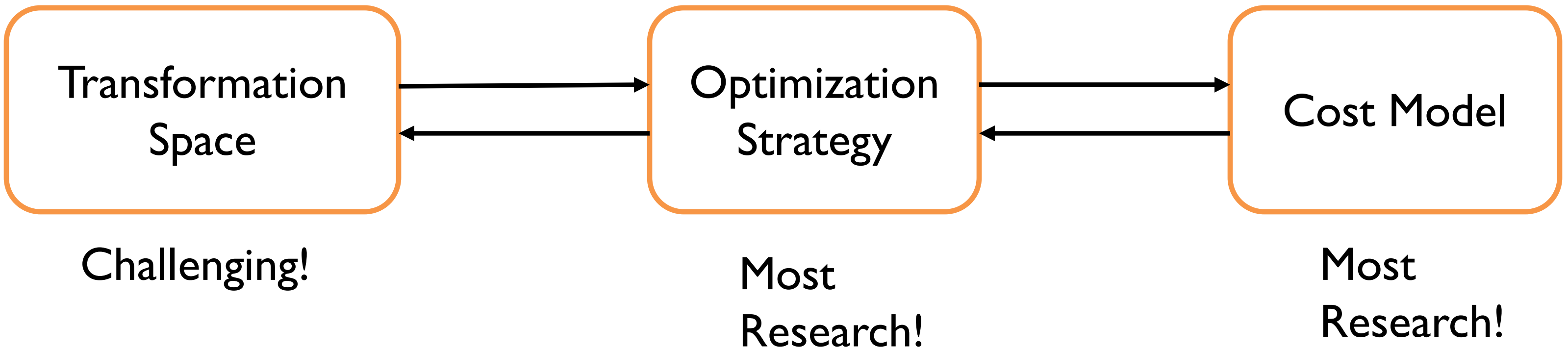
Type II Optimizations

Optimizations with mutually exclusive opportunities with varying profitability (e.g., Vectorization, Loop Transformations)

Safety — Transformation Space (Analysis)

Profitability — Cost Model (Analysis)

Optimality — Optimization Strategy (Transformation)



CS 526

COURSE TOPICS

List of Topics (Part 1)

The order of topics is subject to change

Static Program Analysis

- Natural loops, intervals, reducibility (refresher)
- Static single assignment (SSA)
- Dataflow analysis
- Pointer analysis
- Dependence analysis
- Interprocedural analysis

List of Topics (Part II)

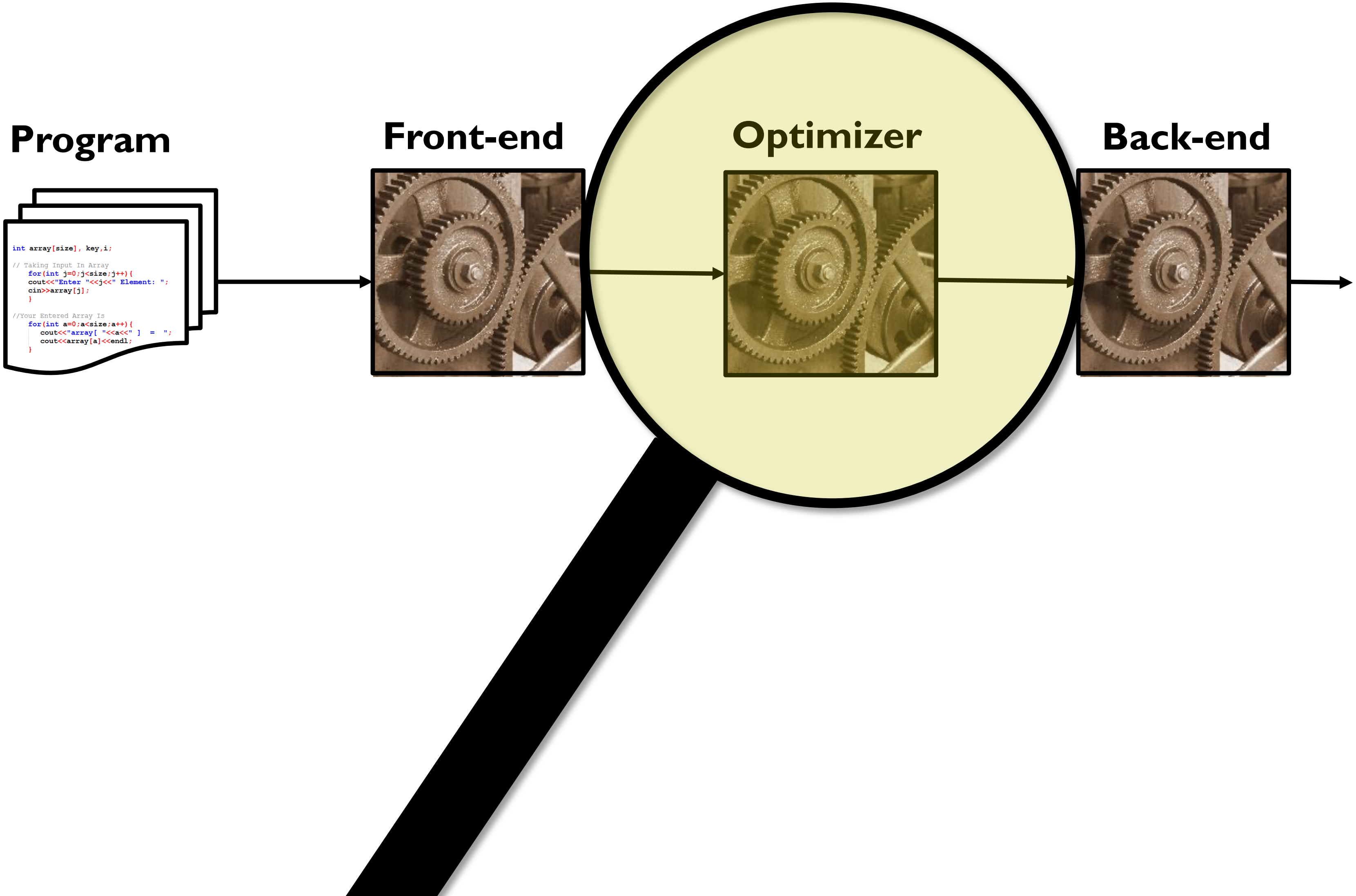
Optimizations

- Code motions and redundancy elimination
- Induction variable optimizations
- Loop transformations and memory hierarchy optimizations
- Basic interprocedural optimizations

Advanced topics (NEW!)

- Vectorization
- Tensor Compilation
- GPU Compilation
- ML in Compilers

Compiler Overview



Topics We Will Not Cover

- Back-end code generation, e.g., scheduling, allocation, software pipelining (CS 426)
- Automatic parallelization (CS 598dp)
- ML for Compilers and Architecture (CS 598cm)
- Program verification (CS 476, CS 477…)
- LLVM hacking (although we have the project ☺)

CS 526

COURSE LOGISTICS

Schedule

Twice a week – Tuesdays and Thursdays 12:30 pm-1:45 pm

All classes will be in person

Course Format

- Lectures – most of the weeks (maybe guest)
- Projects – two programming assignments (LLVM)
- Exams – midterm and final exams; both take home
- Mini-quizzes – before (almost) every lecture (starting from week 3)

Prerequisites

Helpful (I will assume you took it):

Basic **compilers** course (e.g., CS 426)

Also helpful:

Basic **programming languages** course (e.g., CS 421)

Basic **computer architecture** (e.g., CS 233)

Most important: commitment to learn as you go

Grading

Mini-quizzes	10%
Optimization Project	10%
Midterm Quiz	20%
Final Quiz	20%
Open-ended Project	40%

Miniquizzes

Test background knowledge

- **5 minutes** at the beginning of each class
- Concept from compiler theory, something that was covered in previous courses or lectures
- We will use CampusWire polls to conduct the quizzes
- We will discuss the solution immediately afterwards

Each miniquiz is **worth ~0.67%** (up to 10%).

- The main purpose is to bring everyone to the same page before we start the discussions
- In total ~20 quizzes; can miss 5 without penalty

Exams

First

- Take home (March 7; before the break)
- Focuses on analysis (SSA, dataflow, dependency)
- 75 minutes (within 24 hour time)

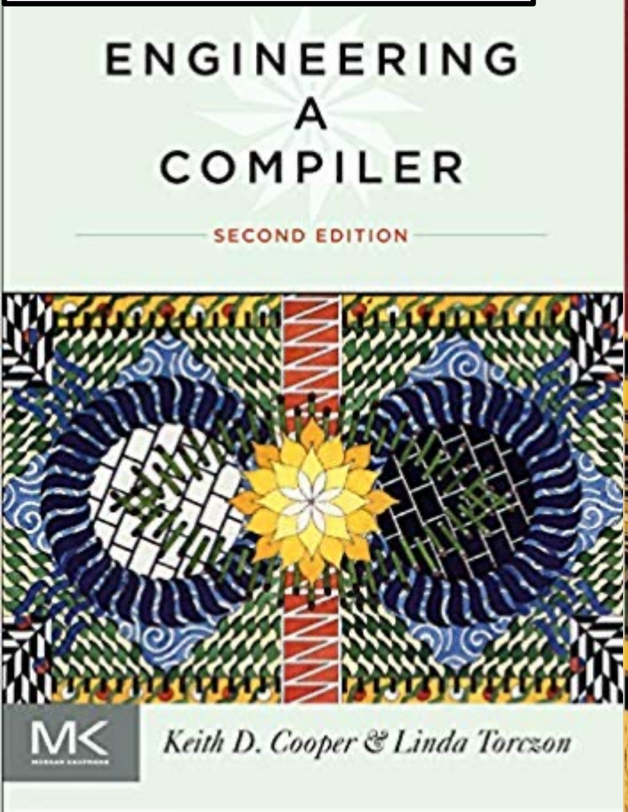
Second

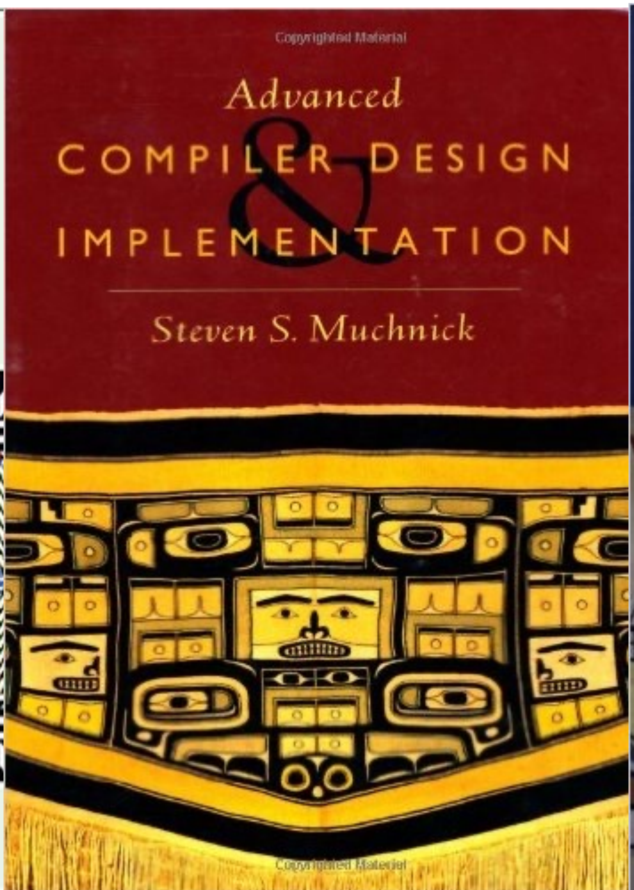
- Take home (April 30)
- Pointer analysis, optimization, and special topics
- It also includes the materials from the first one
- 90 minutes (within 24 hour time)

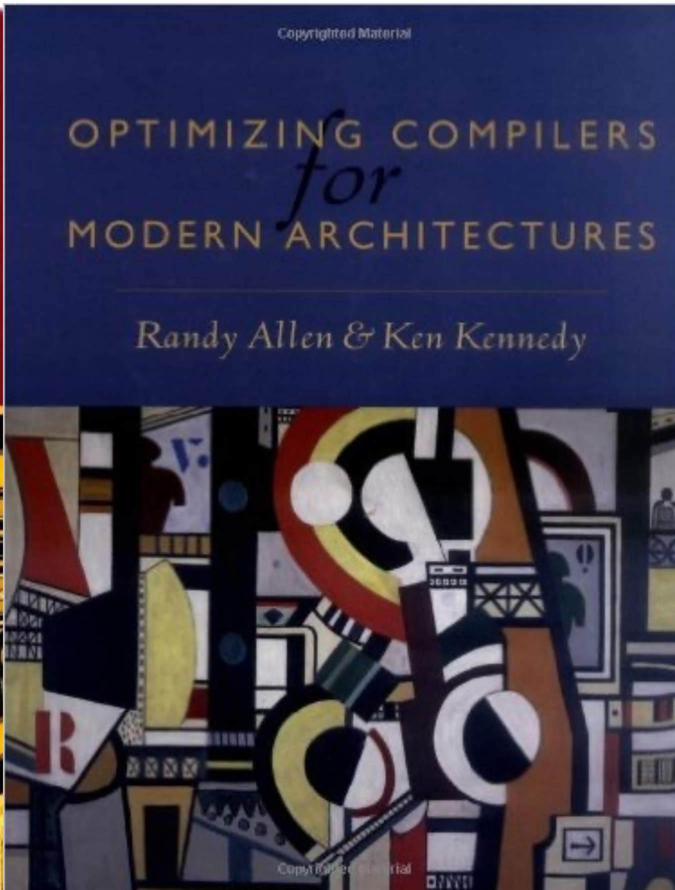
Books

No official book, but many times you will **need** to look into one of these:

Available online via
Illinois University Library



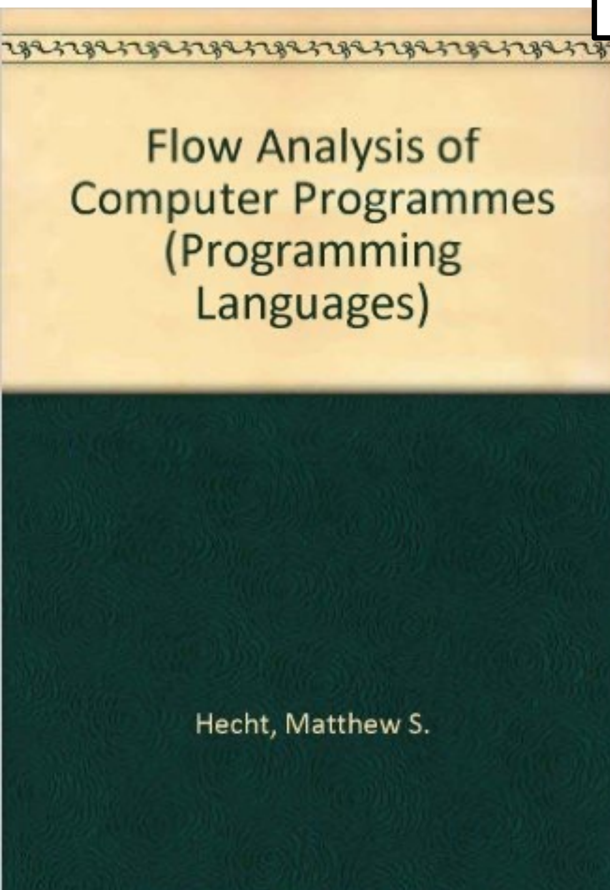


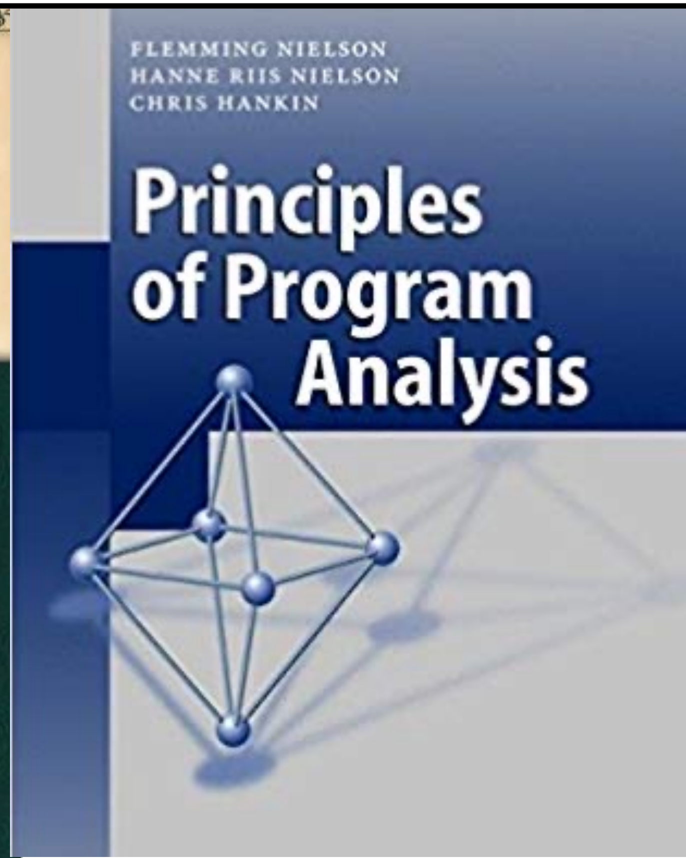


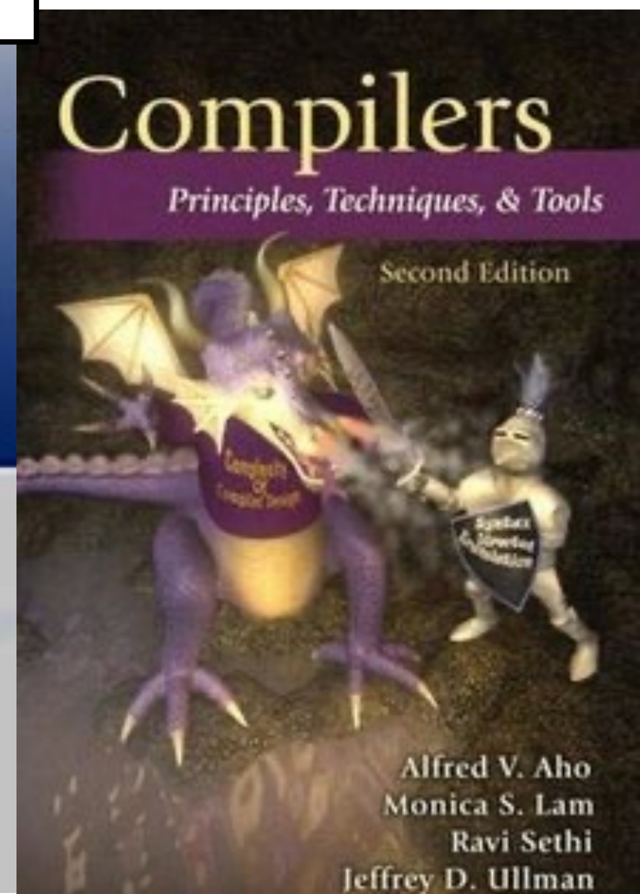
And More Books

No official book, but many times you will **need** to look into one of these:

Available online via Publisher







And More …

We will point our several classical papers that introduced the analysis and/or optimization techniques

To access the papers from ACM/IEEE prepend the link with the following:

```
http://www.library.illinois.edu/proxy/go.php?url=
```

Projects

Gain experience solving existing compiler problems

- Read the literature for the problems
- Find or develop a solution
- Implement the solution in a realistic compiler
- Test it on realistic benchmarks

Projects

P1 – Warm-up exercise:

- ***Individual*, ~**2 weeks but do it sooner
- Scalar replacement of aggregates via SSA (Muchnick, Chapter 12)
- Goal: become familiar with the infrastructure

P2 – Main problem

- ***Groups of two*, ~**12 weeks, also do it sooner!
- Choose and solve a harder problem (Suggestions coming soon)

Infrastructure

LLVM: Low Level Virtual Machine http://llvm.org

- Virtual instruction set: RISC-like, SSA-form
- Powerful link-time (interprocedural) optimization system
- Many front-ends: C/C++, D, Fortran, Julia, Haskell, Objective-C, OpenMP, OpenCL, Python, Swift, ...
- Software: 1.3M+ lines of C++
- Open source: In use at many universities and major companies

Infrastructure

Prepare for the project, during next week:

Read LLVM Documentation at http://llvm.org/docs:

Introduction to the LLVM Compiler Infrastructure

Follow instructions in the ***Getting Started*** and ***Writing an LLVM Pass*** guides to:

(a) Download LLVM, with Clang and test-suite

(b) Do a full build (no need to run "make install")

(c) Compile and run the "Hello" pass

Install on **your EWS or on Campus Cluster**: `ssh <netid>@linux.ews.illinois.edu`

Get in Touch

Email: charithm@illinois.edu

- Please include "[CS 526]" in the subject line

CampusWire: please register as most announcements will be using this.

Office: Siebel Center, office 4118

Office Hours:

- By appointment (send me an email)
- I am typically free right after the class
- We can organize dedicated office hours before the exams

CS 526

QUESTIONS SO FAR?