Introduction to Code Analysis and Optimization

Program Analysis and Optimization – DCC888

Fernando Magno Quintão Pereira

fernando@dcc.ufmg.br

Compiler Writers

The goal of a compiler writer is to bridge the gap between programming languages and the hardware; hence, making programmers more productive

---

*A compiler writer builds **bridges** between **people** and **machines**, and this task is each day more challenging.*

Software engineers want abstractions		Hardware engineers want efficiency

Goals of this Course

A Goal of this Course is to Explain the Student how to Transform a Program Automatically, while Preserving its Semantics, in such a way that the new Program is more Efficient According to a Well-Defined Metric.

- There are many ways to compare the performance of programs:
  - Time
  - Space
  - Energy consumption

We will be focusing mostly on runtime

Proebsting's Law

Compiler Advances Double Computing Power Every 18 Years.

"while hardware computing horsepower increases at roughly 60%/year, compiler optimizations contribute only 4%."

"Perhaps this means Programming Language Research should be concentrating on something other than optimizations [e.g., programmer productivity]."

Can you think on a way to verify this statement?

Todd A. Proebsting
The University of Arizona

Goals of this Course

ANOTHER GOAL OF THIS COURSE IS TO INTRODUCE STUDENTS TO TECHNIQUES THAT LET THEM UNDERSTAND A PROGRAM TO A LEVEL THAT WOULD BE HARDLY POSSIBLE WITHOUT THE HELP OF A MACHINE.

- We understand programs via static analysis techniques.
- These analyses are key to enable code optimizations.
- But they also have other uses:
  - They help us to prove correctness properties.
  - They help us to find bugs in programs.

Goal 1: Program Optimization

- One of the goals of the techniques that we will see in this course is to optimize programs.
- There are many, really many, different ways to optimize programs. We will see some of these techniques:
  - Copy elimination
  - Constant propagation
  - Lazy Code Motion
  - Register Allocation
  - Loop Unrolling
  - Value Numbering
  - Strength Reduction
  - Etc, etc, etc.

```
#include <stdio.h>
#define CUBE(x) (x)*(x)*(x)
int main() {
  int i = 0;
  int x = 2;
  int sum = 0;
  while (i++ < 100) {
    sum += CUBE(x);
  }
  printf("%d\n", sum);
}
```

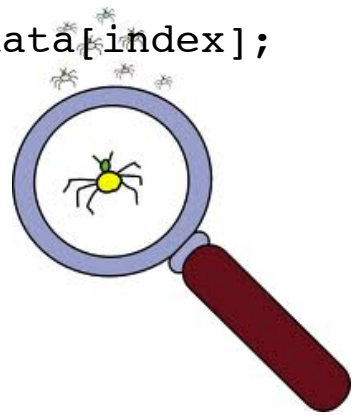
How can you optimize this program?

```
_main:
  pushl %ebp
  movl  %esp, %ebp
  pushl %ebx
  subl  $20, %esp
  call  L3
L3:
  popl  %ebx
  movl  $800, 4(%esp)
  movl  %eax, (%esp)
  call  _printf
  addl  $20, %esp
  popl  %ebx
  leave
  ret
```

Goal 2: Bug Finding

- Compiler analyses are very useful to find (and sometimes fix) bugs in programs.

```
1 void read_matrix(int* data,
       char w, char h) {
2   char buf_size = w * h;
3   if (buf_size < BUF_SIZE) {
4     int c0, c1;
5     int buf[BUF_SIZE];
6     for (c0 = 0; c0 < h; c0++) {
7       for (c1 = 0; c1 < w; c1++) {
8         int index = c0 * w + c1;
9         buf[index] = data[index];
10      }
11    }
12    process(buf);
13  }
14 }
```

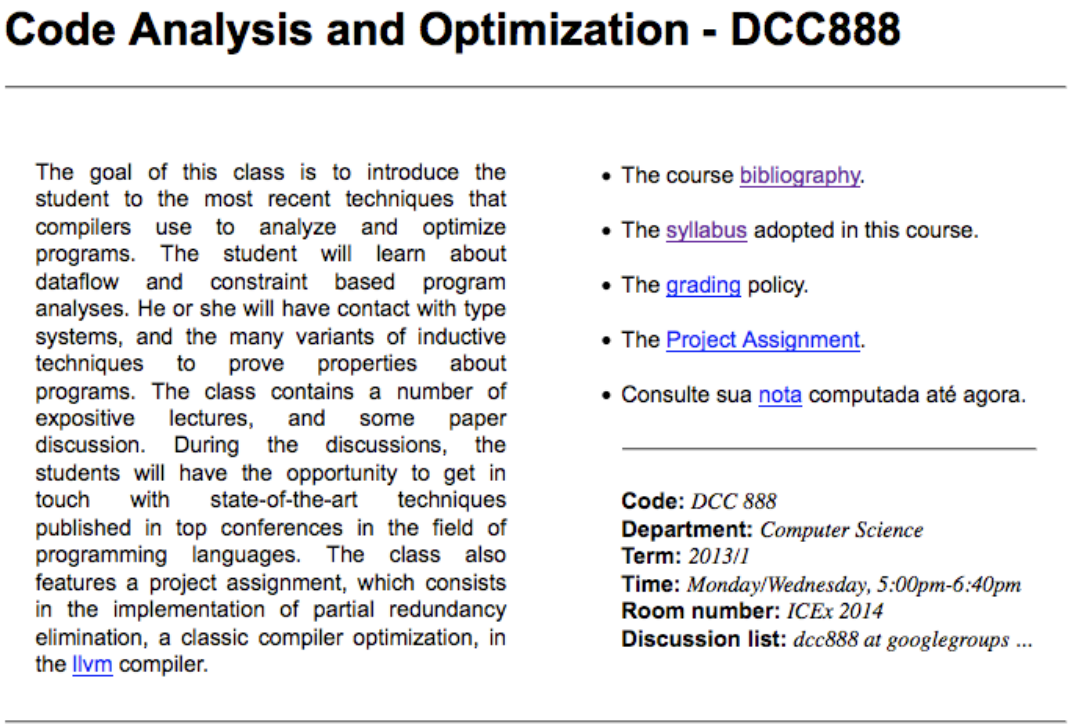


- Null pointer dereference
- Array out-of-bounds access
- Invalid Class Cast
- Tainted Flow Vulnerabilities
- Integer Overflows
- Information leaks

Can you spot a security bug in this program. **Be aware**: the bug is tricky.

The Contents of the Course

- All the material related to this course is available on-line, at http://www.dcc.ufmg.br/~fernando/classes/dcc888.
- This material includes:
  - Slides
  - Project assignment
  - Class Exercises
  - Useful links
- The page also contains the course bibliography, and a brief discussion about the grading policy



Code Analysis and Optimization - DCC888

The goal of this class is to introduce the student to the most recent techniques that compilers use to analyze and optimize programs. The student will learn about dataflow and constraint based program analyses. He or she will have contact with type systems, and the many variants of inductive techniques to prove properties about programs. The class contains a number of expositive lectures, and some paper discussion. During the discussions, the students will have the opportunity to get in touch with state-of-the-art techniques published in top conferences in the field of programming languages. The class also features a project assignment, which consists in the implementation of partial redundancy elimination, a classic compiler optimization, in the llvm compiler.

- The course bibliography.
- The syllabus adopted in this course.
- The grading policy.
- The Project Assignment.
- Consulte sua nota computada até agora.

Code: *DCC 888*
Department: *Computer Science*
Term: *2013/1*
Time: *Monday/Wednesday, 5:00pm-6:40pm*
Room number: *ICEx 2014*
Discussion list: *dcc888 at googlegroups ...*

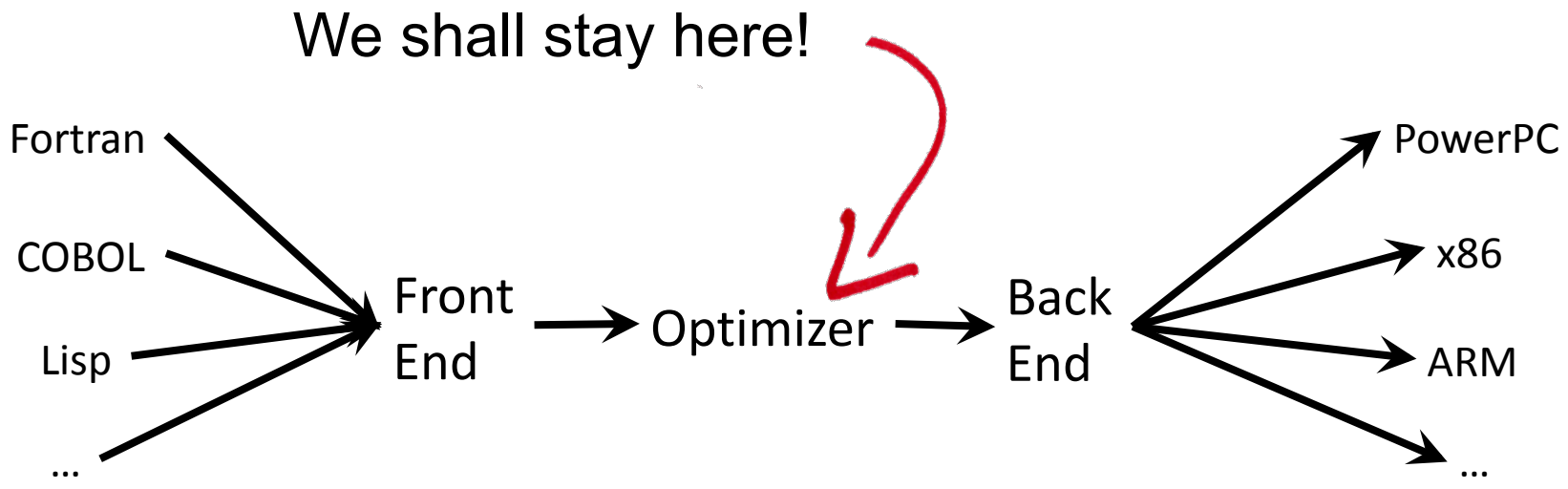
The Contents of the Course

- The course has 24 lectures. Slides for each lecture are available on-line:
  1. Introduction
  2. Control Flow Graphs
  3. Dataflow Analyses
  4. Worklist algorithms
  5. Lattices
  6. lazy Code Motion
  7. Constraint-Based Analysis
  8. Pointer Analysis
  9. Loop Optimizations
  10. Static Single Assignment
  11. Sparse Analyses
  12. Tainted flow analysis
  13. Range Analysis
  14. Program slicing
  15. Register Allocation
  16. SSA-Based Register Allocation
  17. Operational Semantics
  18. Type Systems
  19. Mechanical Proofs
  20. Type Inference
  21. Just-in-time compilation
  22. Correctness
  23. Divergence Analysis
  24. Automatic Parallelization

Where we will be

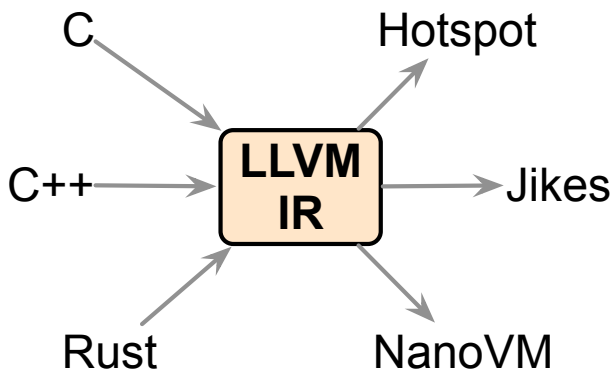
- A compiler has three main parts
  - The front-end is where parsing takes place.
  - The middle-end is where optimizations and analyses take place.
  - The back-end is where actual machine code is generated.

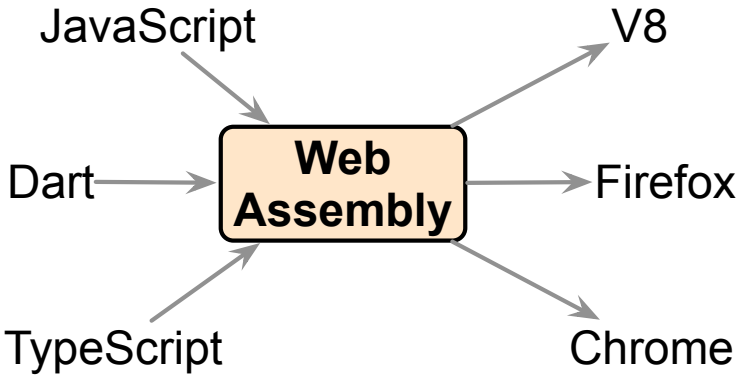
In which CS courses can we learn about the other phases?

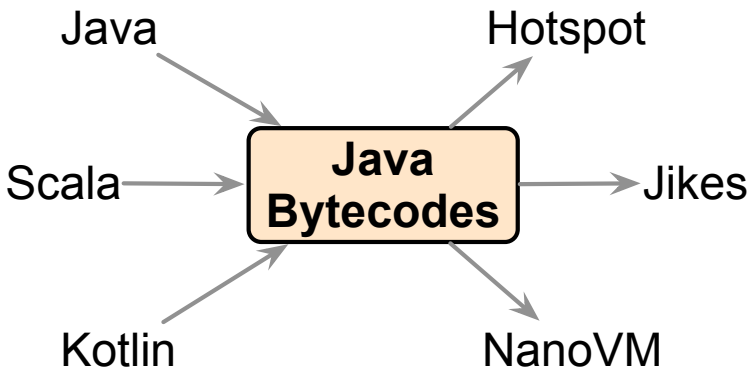


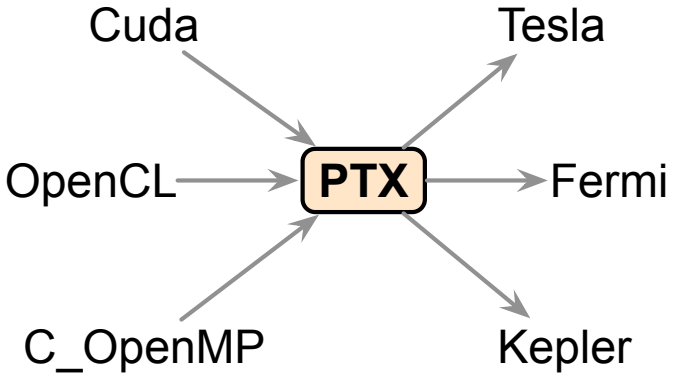
A Very General Pattern

Are you familiar with all these formats?









There is no Silver Bullet

1. It is impossible to build the perfect optimizing compiler.

- The perfect optimizing compiler transforms each program P in a program P_{opt} that is the smallest program with the same input/output behavior as P.

Can you prove the first statement?

There is no Silver Bullet

1. It is impossible to build the perfect optimizing compiler.

Let's assume that we are optimizing for size. We can reduce the problem of building the perfect compiler to an undecidable problem, such as the OUTPUT PROBLEM, e.g., "Does the program P output anything?"

The smallest program that does nothing and does not terminate is P_{least} = `L: goto L;` By definition, the perfect compiler, when fed with a program that does not generate output, and does not terminate, must produce P_{least}.

Thus, we have a decision procedure to solve the OUTPUT PROBLEM: given a program P, if the perfect compiler transforms it into P_{least}, then the answer to the problem is **No**, otherwise it is **Yes**.

The Full-Employment Theorem

1. If C is an optimizing compiler for a *Turing Complete Language*, then it is possible to build a better optimizing compiler than C.

- In other words, compiler writers will always have a job to do, as it is always possible to improve any existing compiler that compiles any meaningful programming language.

How could we prove this new theorem?

The Full-Employment Theorem

1. If C is an optimizing compiler for a *Turing Complete Language*, then it is possible to build a better optimizing compiler than C.

Let's assume that there exists the best compiler B. If P is a program, then let B(P) be its optimized version. There must be a program P_x that does not terminate, such that $B(P_x) \neq$ [L:goto L], otherwise B would be the perfect compiler. As we have seen, this is impossible.

Therefore, there exists a compiler B' that is better than B, as we can define it in the following way:

B'(P) = if P == P_x then [L:goto L] else B(P)

Compilers Laboratory

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSIDADE FEDERAL DE MINAS GERAIS
FEDERAL UNIVERSITY OF MINAS GERAIS, BRAZIL

WHY TO LEARN COMPILERS?

DCC 888

The Importance of Compilers

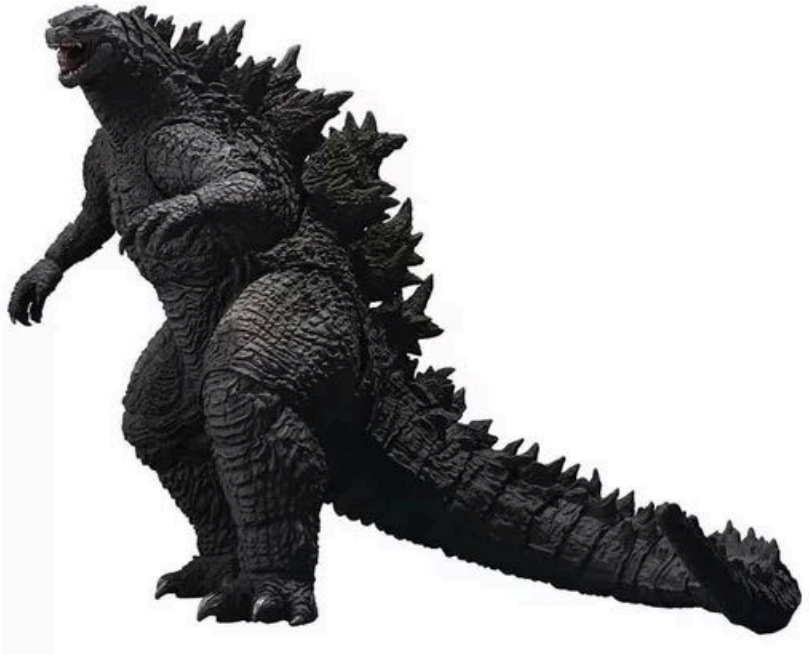
Robert Hundt is a leading compiler engineer working at Google. Read below what he says about the importance of compilers in general, and in that company in particular

- "At the scale of datacenters, every single performance percent matters! Just take a look at Google's (and other's) publicly available numbers on expenditures on datacenters. We are talking about billions of dollars. A single percent improvement can mean millions of dollars from more program features or improved utilization."
- "In order to deploy software at Google scale, engineers will touch a good dozen of programming and configuration languages, all with their own interesting optimization problems (from a compiler writer's point of view). Fundamental knowledge in language, compiler, and runtime implementation will help make better engineering decisions, everywhere."
- "Did you know that many of our first today's most celebrated engineers have compiler backgrounds? Jeff Dean, Sanjay Ghemawat, Urs Hoelzle, and many others. It's not a coincidence. Compiler optimization trains in bit-fiddling as well as algorithmic thinking, which are essential to success in today's world."

Industrial Compilers are Pretty Big...

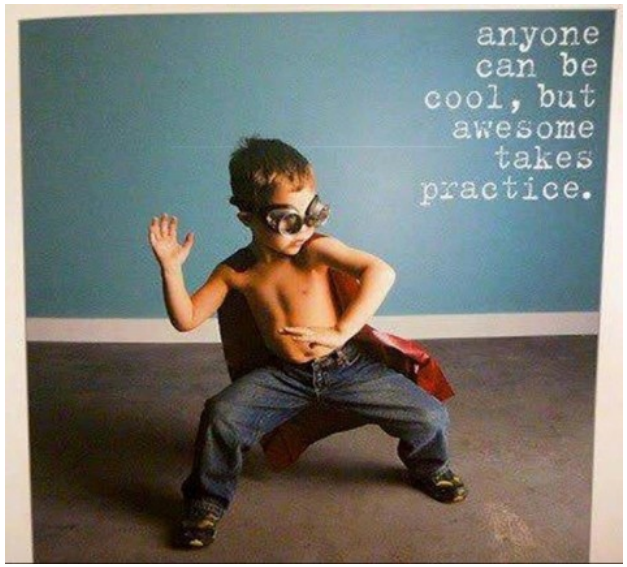
- Compilers are very complex systems. It really takes a lot of programming skills to write them.
  - Large, robust system, usually coded in a high-performance language, such as C or C++, with lots of interactions with the operating system and the hardware.

Can you guess the size of a compiler like LLVM or gcc, in lines of code?



... and mix together a lot of Computer Science!

- Compilers are backed up by a lot of computer science theory. It is not only programming.
  - Type systems, parsing theory, graph theory, algorithms, algebra, fixed point computations, etc

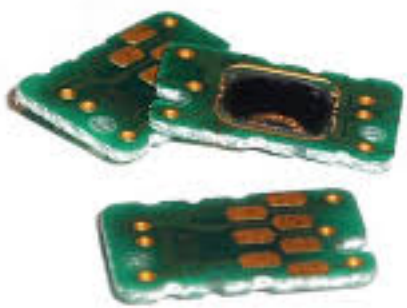


"The first reason Compiler Construction is such an important CS course is that it brings together, in a very concrete way, almost everything you learned before you took the course." Steve Yegge ♧

♧: Steve Yegge has worked at Amazon and Google, among others.

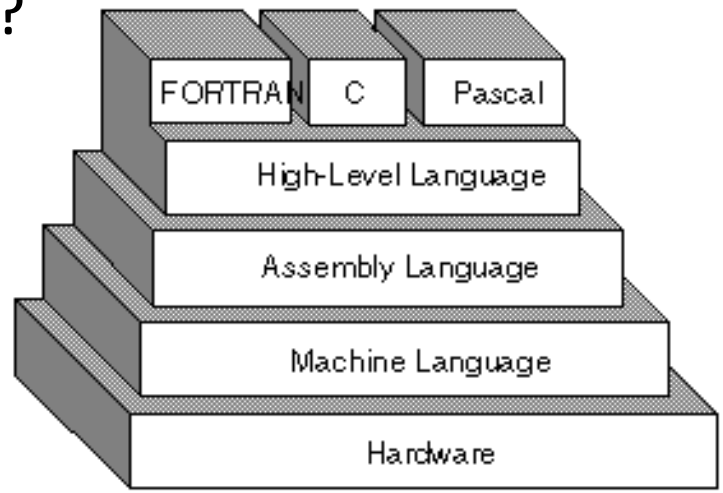
The Importance of Compilers

We have hundreds of different programming languages. How do you think they are executed?



But… can you think on why it would do good to you, from a personal point of view, to learn compilers?

And we have hundred of different hardware. How can we talk to all these different worlds?



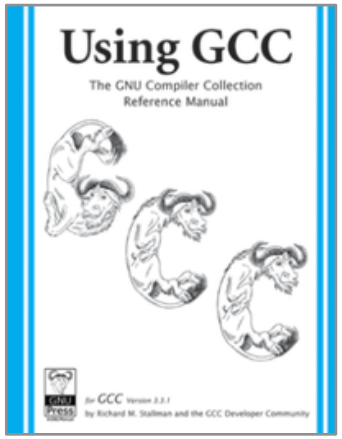
Why to Learn Compilers

1. We can become better **programmers** once we know more about compilation technology.
2. There are plenty of very good **job** opportunities in the field of compilers.
   - "*We do not need that many compiler guys, but those that we need, we need them badly.*"♤
3. We can become better computer **scientists** if we have the opportunity to learn compilation technology.
   - A microcosm crowded with different computer science subjects.
   - Lots of new things happening all the time.

Are there other reasons to learn about compilers?

♤: quote attributed to François Bodin, professor at the University of Rennes 1. Previously senior engineer at CAPS

Compilers Help us to be Better Programmers



Compilers usually have different optimization options. The iconic gcc –O1, for instance, runs these optimizations.

What does each of these things do?

```
fauto-inc-dec
fcompare-elim
fcprop-registers
fdce
fdefer-pop
fdelayed-branch
fdse
fguess-branch-probability
fif-conversion2
fif-conversion
fipa-pure-const
fipa-profile
fipa-reference
fmerge-constants
fsplit-wide-types
ftree-bit-ccp
ftree-builtin-call-dce
ftree-ccp
ftree-ch
ftree-copyrename
ftree-dce
ftree-dominator-opts
ftree-dse
ftree-forwprop
ftree-fre
ftree-phiprop
ftree-slsr
ftree-sra
ftree-pta
funit-at-a-time
```

We can even run these optimizations individually:

```
$> gcc -fdefer-pop -o test test.c
```

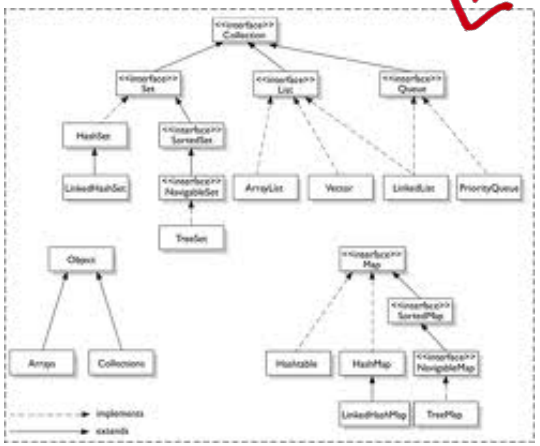
We can enable a few of them, and disable others, e.g.:

```
$> gcc -O1 -fno-defer-pop -o test test.c
```

Knowing Compilers will Fix some Misconceptions

"I am a conscious programmer, because I am reusing the name 'i'. In this way my program will consume less memory."

```
int i = read();
if (i != EOF)
    i = read();
    printf("%d", i);
```



"I will not use inheritance in Java, because it can make my method calls more expensive, as I have to find them in the class hierarchy."

"I will use macros, instead of functions, to have a more efficient code. In this way, there will be no time lost in function calls."

What are the fallacies in these reasonings?

```
#define MAX(X, Y) (X) > (Y) ? (X) : (Y)

int max(int i, int j) { return i > j ? i : j; }
```

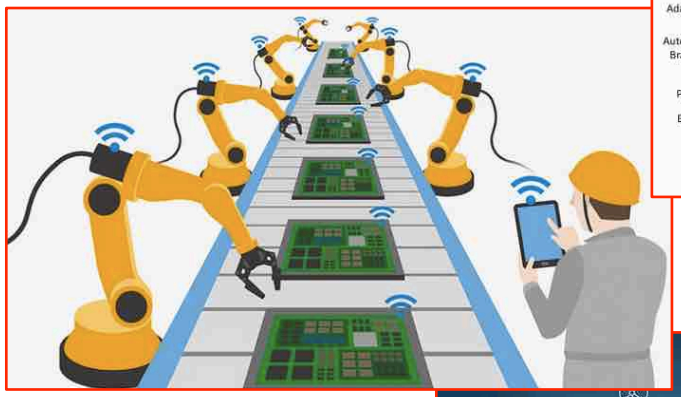
Lots of Job Opportunities

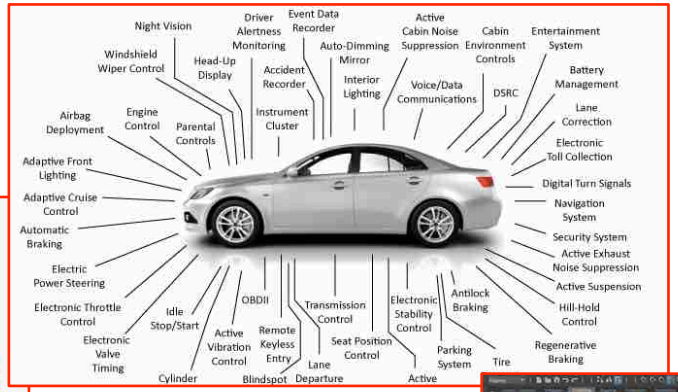
- Expert compiler writers find jobs in many large companies: Oracle, NVIDIA, Microsoft, Sony, Apple, Cray, Intel, Google, MathWorks, IBM, AMD, Mozilla, etc.
- These jobs usually ask for C/C++ expertise.
- Good knowledge of basic computer science.
- Advanced programming skills.
- Knowledge of compiler theory is a big plus!
- Papers published in the field will help a lot too!

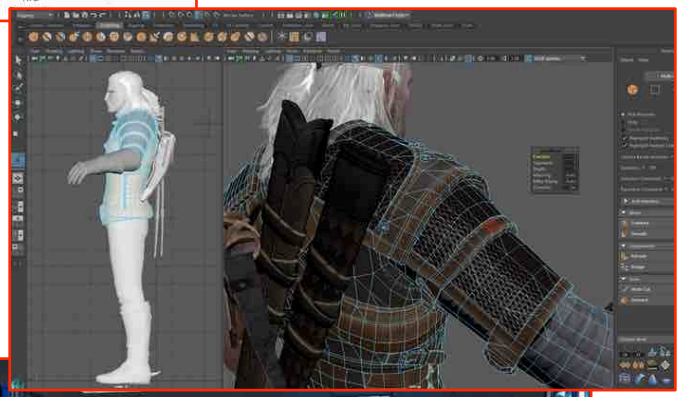
But, what do compiler engineers do?

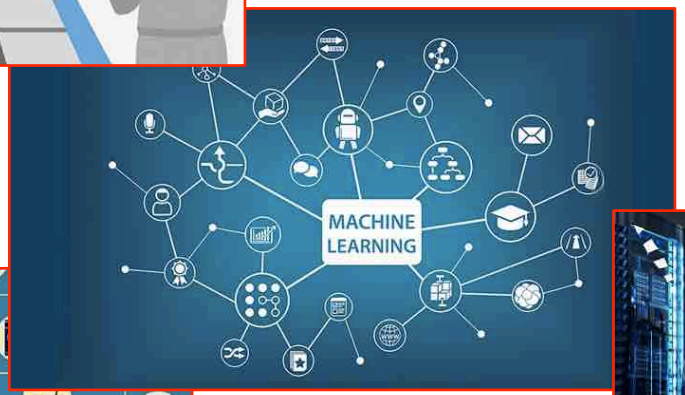
PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

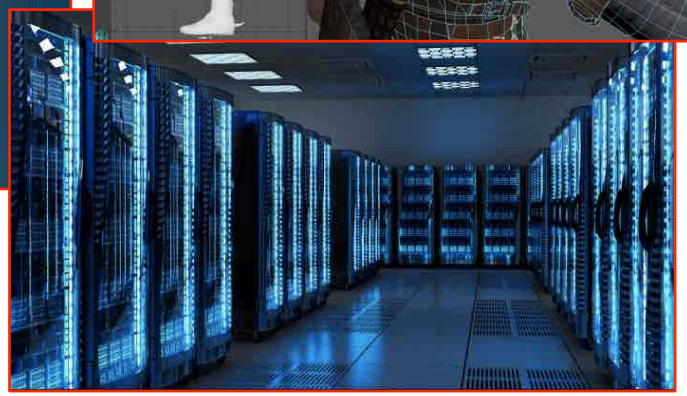
What do Compiler Engineers Do?

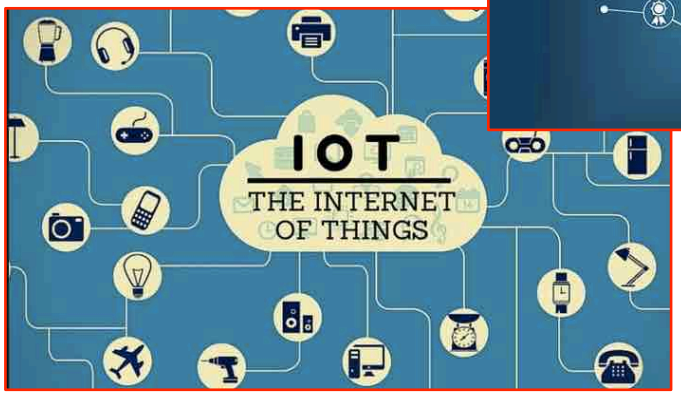












Compiler Writers in the Big Companies

Hardware companies need compilation technology.

Can you think about other examples?

Intel => icc

Mozilla => JaegerMonkey, IonMonkey, TraceMonkey

Apple => LLVM

NVIDIA => nvcc

Google => ART, V8

Microsoft => visual studio, .NET VM

STMicroelectronics => open 64

Examples of Specilized Companies

There are companies that sell mostly compilation technology, which can be used in several ways, e.g., to create a new back-end, to parse big data, to analyze programs for security vulnerabilities, etc.

CodePlay is a company that develops compilers for Systems on a Chip devices used in the automotive industry.

Coverity is a software vendor which develops static code analysis tools, for C, C++, Java and C#.

The Associated Compiler Experts (**ACE**) have developed compilers for over 100 industrial systems, ranging from 8-bit microcontrollers to CISC, RISC, DSP and 256-bit VLIW processor architectures.

PathScale Inc. is a company that develops a highly optimizing compiler for the x86-64 microprocessor architectures.

The Portland Group, Inc. (**PGI**) is a company that produces a set of commercially available Fortran, C and C++ compilers.

Green Hills produces compilers for C, C++, Fortran, and Ada that target the ARC, ARM, Blackfin and ColdFire platforms

Example: JetBrains

The company offers an extended family of integrated development environments (IDEs) for SQL and the programming languages Java, Kotlin, Ruby, Python, PHP, Objective-C, C++, C#, Go and JavaScript.

There is always love for CompGeeks

Hi Prof Pereira,

Long time don't see. I trust you are well.

Here at Apple, we are investing heavily in our compiler technologies. I was wondering if you might have recommendations for students who are interested in pursuing a career in industrial compiler work. I'd be interested in talking to them about opportunities here.

Regards,

Evan Cheng
Sr Manager, Compiler Technologies, Apple (March'13)

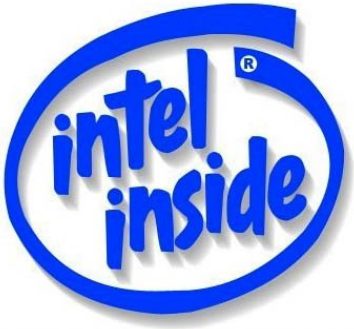
Lots of Love…

Dear Fernando,

I have read your paper in JIT specialization in JavaScript and particularly your work in the context of IonMonkey has caught my eye. I am working as a Research Scientist at Intel Labs on the River Trail project. We are collaborating with Mozilla on a Firefox implementation. Given your recent work on IonMonkey, maybe one of your students would be interested to join us in Santa Clara to work on this topic?

Regards,

Stephan (May'13)



Everywhere...

Hi Fernando,

The Compiler team at Arm (Cambridge, UK) will be hiring more than 10 people throughout this year to work in the Arm Compiler. As I have told you previously, the Arm Compiler is based on LLVM and is a product sold to companies that deploy Arm-based chips, i.e. pretty much *every single* big tech company you can think of. It is used to compile programs that run on virtually every smart device in the world.

No prior experience with compilers is necessary, although very much desired. Working in our team also involves contributing a lot to open-source LLVM, which is also a very motivating perk. In case you have anyone to recommend, please have them talk to me.

Victor (Feb'20)

ARM

Even in Brazil

Oi Fernando,

O pessoal da IBM de Campinas está procurando por pessoas da área de compiladores para montar um grupo de otimização por aqui. Devem contratar umas 10 pessoas. Você tem recomendação?

Rodolfo Azevedo (Feb'19)

And Even in Minas Gerais!

Oi Fernando,

tudo bom?

Preciso de uma ajuda sua. Estamos com uma vaga aberta no time de Compilation na Cadence. Eu gostaria de tentar entrevistar uma pessoa do Lac [...]

Obrigado,

Victor (Mar'18)

cadence

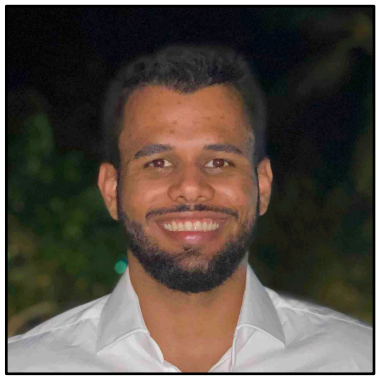
Compiler Engineers from UFMG

Hello all! My name is ***Victor Campos***. I worked as a Research Assistant in the Compilers Laboratory between 2011 and 2016, when I obtained my Master's degree under Fernando's supervision. After working at different jobs (luckily all in compilers), both in Brazil and abroad, last October I joined Arm, located in Cambridge, UK (where Alan Turing himself attended university, by the way).

At Arm I write compilation support for upcoming processor architectures that you will be carrying in your pockets in two years' time. My job is deliciously challenging because everything I write must be very efficient. After all, the compiler I develop is used to compile the programs that run on 95% of the world's smart devices, from entry-level Android phones in India to the latest iPhone in the US. It's quite satisfying to work on things that the entire world benefits from!

Compiler Engineers from UFMG

I am ***Caio Lima***. I finished my MSc at the Compilers Laboratory in 2019. During my graduation, I developed compiler optimizations for JavaScript. In parallel to that, I was an active open source contributor to JavaScriptCore, which is WebKit's JavaScript VM. This experience got me an internship at an Open Source company named Igalia. This is the company where I work now. At Igalia, I had the chance to work on new JavaScript features like BigInt or private class fields, from specification level until JIT optimizations. I feel very satisfied there. I can't imagine myself being in a better place, since the work is very challenging (every month I get segmentation faults where the stack is corrupted!) and also very impactful (I may be the person who broke your browser sometimes!), since JS is one of the most popular programming languages in the world.



Compiler Engineers from UFMG

I am ***João***, I've completed my Master's Degree at the Compilers Laboratory in 2021. During that time, I worked on generating parsers to detect and redact private information from infinite streams of semi structured text formats, such as database logs. During my Masters, I've got an internship opportunity at Microsoft, where I still work at. Currently, I work on the HLSL (High Level Shader Language) Compiler team. The work we do ranges from adding new built-in functions to the language and adding support for new language features, all the way to add support for new GPU hardware capabilities. We also maintain the HLSL backend for the XBox. The work we do has huge impact in the gaming industry. Working with compilers is challenging, but it also provides exciting opportunities for you to learn from and grow as a software developer.

Microsoft

Compiler Engineers from UFMG

I finished my PhD at the Compilers Lab in 2019. My project was on type inference for C (https://github.com/ltcmelo/psychec). Currently, I work at ShiftLeft (https://www.shiftleft.io), which offers products for application security; there, I am a compiler engineer responsible for our C# and Python frontends. Specifically, my main task is to analyze the source code of projects in those languages and put them in a common representation that is understood by our dataflow engine; I am also generally involved in the development of our technology as a whole. Nowadays, software security is going through an exciting momentum (in particular, application security). Thus, if you enjoy compiler construction, programming languages theory and program analysis, you have great chances of landing a good job. ***Leandro T. C. Melo*** - http://ltcmelo.com

ShiftLeft

Compiler Engineers from UFMG

I am ***Hugo Sousa***. I got my B.Sc. at UFMG in 2018, where I also had an internship in the Compilers Laboratory. In the lab my research topic was security: we designed defense mechanisms against Return Oriented Programming attacks. In practice, that meant we often had to analyze and instrument binary code. After graduating I became a software engineer at Cyral, a fast-growing startup that focuses on data security. In Cyral we create products to protect data, regardless of where it is located, or how it flows. Thus, we deal with several data endpoints, such as MySQL and MongoDB. Since most of them have their own query language, we typically use tools like yacc or ANTLR to generate parsers for query languages. With these parsers, we can, for example, provide security guarantees on queries. Some of the things I enjoy the most about my job is always being able to learn new things and technologies, and working on an exciting novel approach for data security.

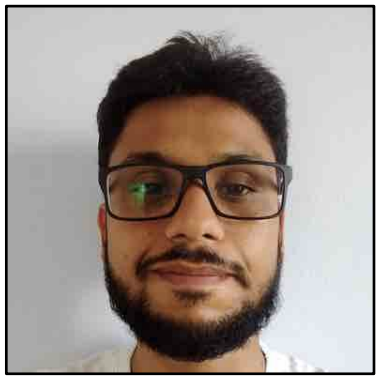
Cyral

Compiler Engineers from UFMG

I finished my MSc at the Compilers Lab on October 2019. In my project, I combined different static analyses to optimize arithmetic instructions. Also, I worked on two different projects related to "silent stores". Currently, I work at Quansight as a compiler engineer. Quansight is a company started by Travis Oliphant, the creator of NumPy. It has the goal of supporting open-source projects. At Quansight, I work with Numba, a compiler for high-performance python code and RBC, a compiler built on top of LLVM. Working as a compiler engineer has been a challenging yet exciting opportunity. Due to Machine Learning and Data-Science, Python is now going through an interesting direction for compiler folks, where different tools are being developed to make code run faster. –***Guilherme Leobas***

Compiler Engineers from UFMG

Hello! I am ***Pedro Caldeira*** and I got my MSc at UFMG on 2019. During the master, I developed a compiler that translates map-reduce patterns written in Java to FPGAs. Later that year, I got a position at IBM, where I presently work. Here I'm part of the Linux Technology Center. Among other things, we support the GNU Compiler Collection (GCC) for Linux on Power, enabling it to exploit new hardware features for each new generation of processors, and improving code generation for better performance. I think this job is amazing. We are proud to make many contributions to the open source community and to work on future technologies that will be available for everyone.



Compilers Laboratory

Department of Computer Science
Universidade Federal de Minas Gerais
Federal University of Minas Gerais, Brazil

What we will see in this course

DCC 888

Compilers – A Microcosm of Computer Science♧

- **Algorithms**: graphs everywhere, union-find, dynamic programming
- **Artificial intelligence**: greedy algorithms, machine learning
- **Automata Theory**: DFAs for scanning, parser generators, context free grammars.
- **Algebra**: lattices, fixed point theory, Galois Connections, Type Systems
- **Architecture**: pipeline management, memory hierarchy, instruction sets
- **Optimization**: operational research, load balancing, packing, scheduling

♧:Shamelessly taken from slides by *Krishna Nandivada,* professor at the Indian Institute of Technology (http://www.cse.iitm.ac.in/~krishna/cs3300/lecture1.pdf)

Static And Dynamic Analyses

- Compilers have two ways to understand programs:
  - Static Analysis
  - Dynamic Analysis
- Static analyses try to discover information about a program without running it.
- Dynamic analyses run the program, and collect information about the events that took place at runtime.

1) Can you give examples of dynamic analyses?

2) And can you give examples of static approaches?

3) What are the pros and cons of each approach?

Dynamic Analyses

- Dynamic analyses involve executing the program.
  - **Profiling**: we execute the program, and log the events that happened at runtime. Example: `gprof`.
  - **Test generation**: we try to generate tests that cover most of the program code, or that produce some event. Example: `Klee`.
  - **Emulation**: we execute the program in a virtual machine, that takes care of collecting and analyzing data. Example: `valgrind`, and `CFGGrind`.
  - **Instrumentation**: we augment the program with a meta-program, that monitors its behavior. Example: `AddressSanitizer`.

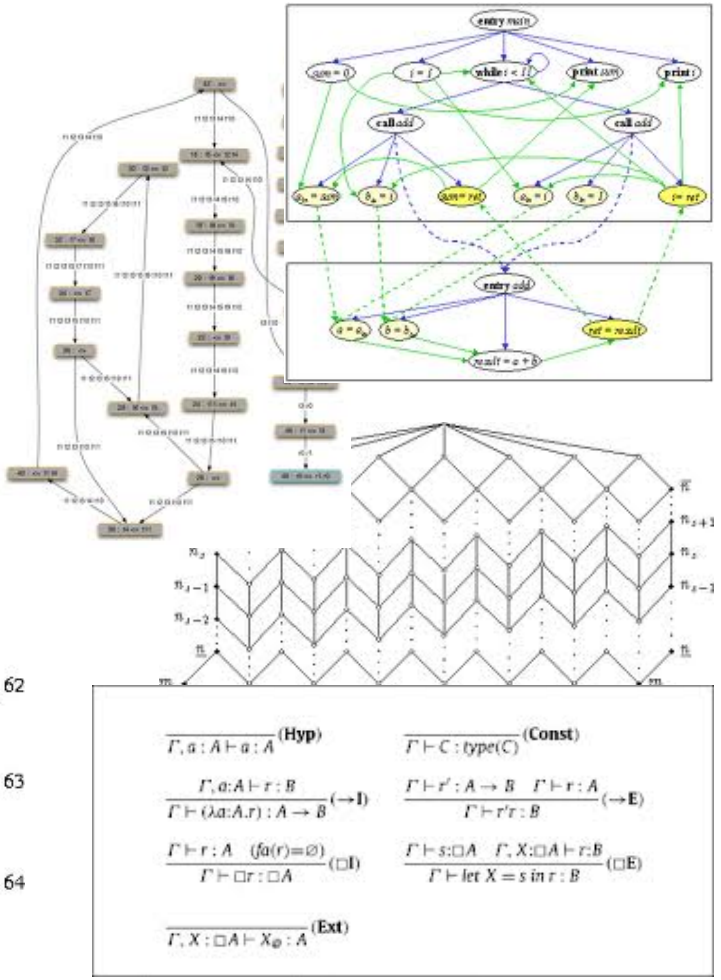
PPGCC
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DCC-UFMG

Static Analyses

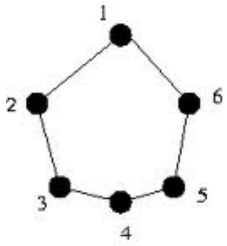
- In this course we will focus on static analyses.
- There are three main families of static analyses that we will be using:
  - **Dataflow analyses**: we propagate information based on the dependences between program elements, which are given by the syntax of the program.
  - **Constraint-Based analyses**: we derive constraints from the program. Relations between these constraints are not determined explicitly by the program syntax.
  - **Type analyses**: we propagate information as type annotations. This information lets us prove properties about the program, such as progress and preservation.

The Broad Theory

- The algorithms used in compiler optimization include many different techniques of computer science
  - Graphs are everywhere
  - Lattices and the Fixed point theory
  - Many different types of induction
  - Dynamic programming techniques
  - Type theory
  - Integer Linear Programming
  - etc, etc, etc



And all of this in industrial strength compilers!



J-i = 2
13, 24, 35, 46, 51, 62

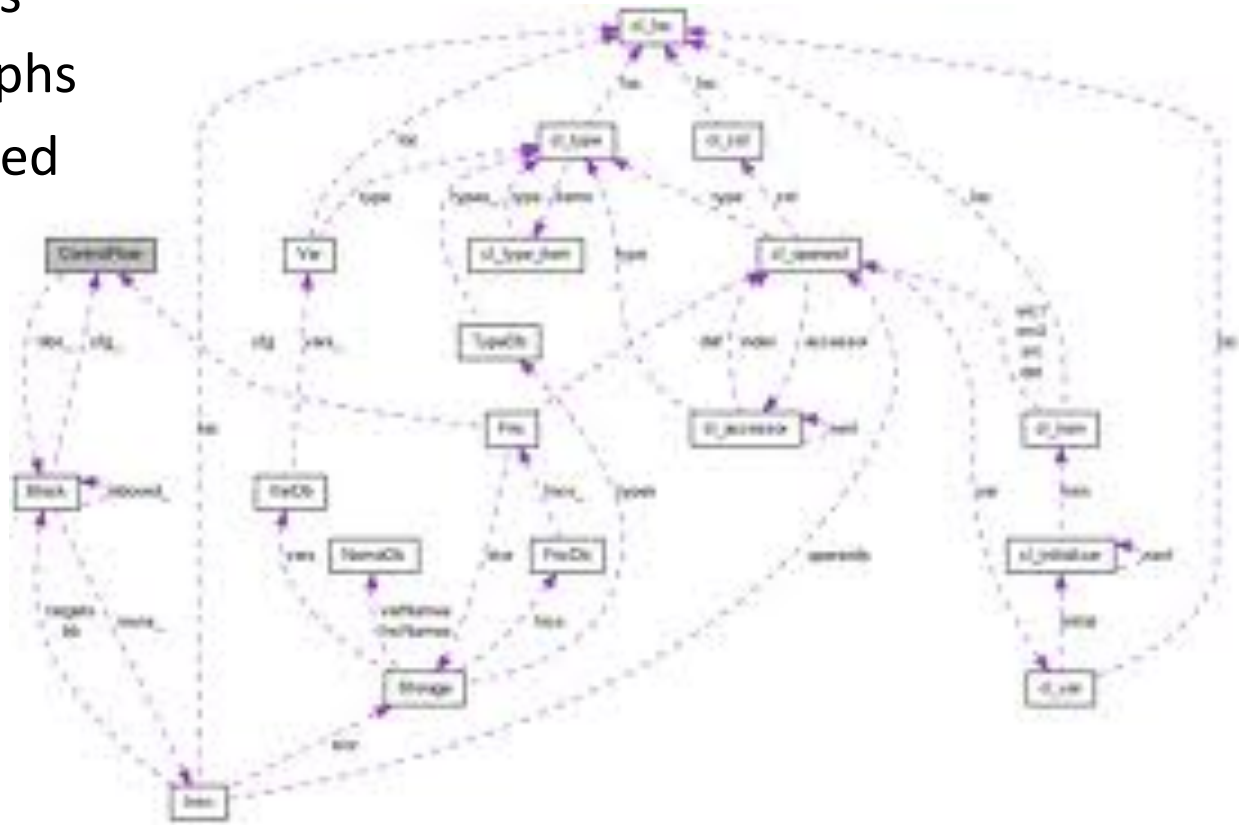
J-i = 3
14, 25, 36, 41, 52, 63

J-i = 4
15, 26, 31, 42, 53, 64
Finish with 16

Graphs

- Graphs are the core of computer science, and permeate code optimization theory.
  - Control Flow Graphs
  - Constraint Graphs
  - Dependence Graphs
  - Strongly Connected Components
  - Graph Coloring

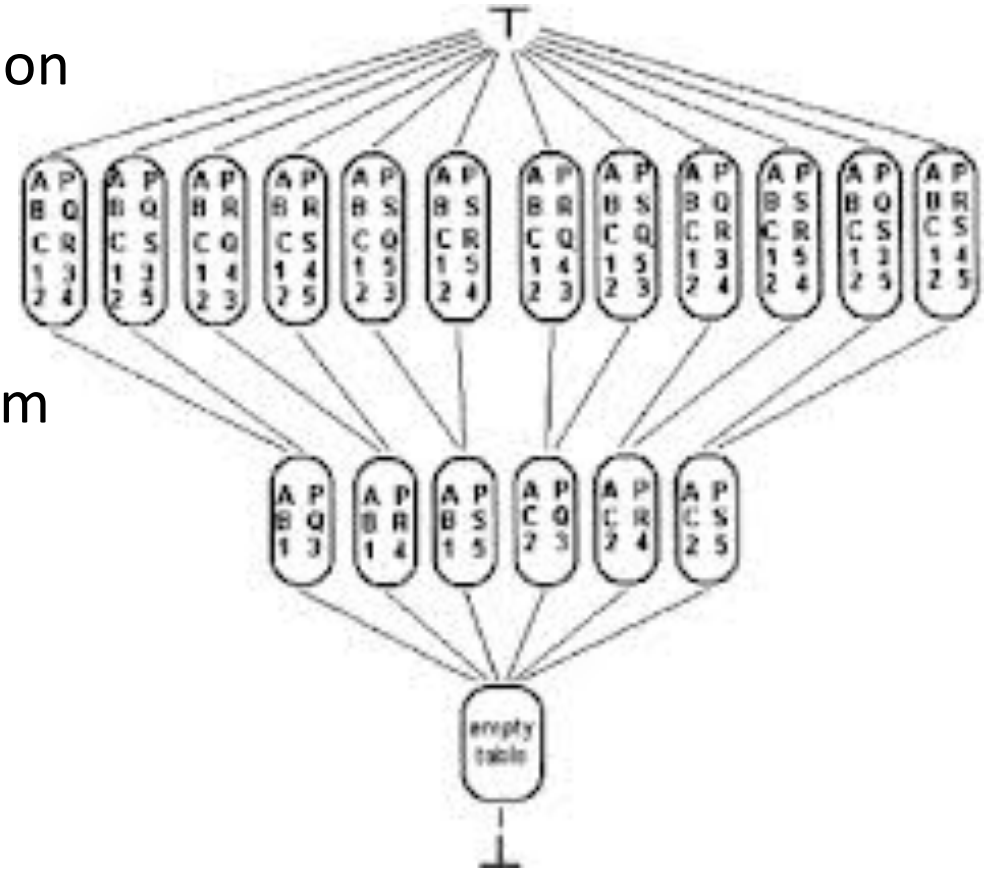
What are each of **these** graphs? Where are they used?



Fixed Point Theory

- Most of the algorithms that we see in code optimization are iterative. How can we prove that they terminate?

- If we always obtain more information after each iteration of our algorithm…
- and the total amount of information is finite…
- Then, eventually our algorithm must stabilize.



Induction all the way

- Most of our proofs are based on induction.
- We will be using mostly *structural induction*

POPL mark

1) Has anyone heard of structural induction?

2) Does anyone know one such tool?

The main advantage of proofs by induction is that we already have technology to verify if these proofs are correct mechanically.

The Program Representation Zoo

- Depending on how we represent a program, we may facilitate different static analyses.
  - Abstract Syntax Trees
  - Control Flow Graphs
    - Static Single Assignment form
  - Program Dependence Graphs
  - Constraint Systems

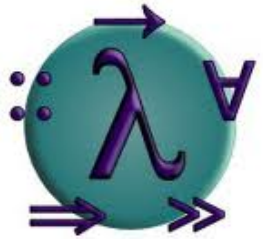
Open Source Community

- Many important compilers are currently open source.
- The Gcc toolkit has been, for many years, the most used compiler for C/C++ programs.

- LLVM is one of the most used compilers for research and in the industry.

- Mozilla's Monkey (SpiderMonkey, TraceMonkey, JagerMonkey, IonMonkey) family of compilers has a large community of users.

- Ocelot is used to optimize PTX, a program representation for graphics processing units.
- The Glasgow Haskell Compiler is widely used in functional programming research.



Conferences and Journals

- There are many important conferences that accept compiler related papers:
  - **PLDI**: Programming Languages Design and Implementation
  - **POPL**: Principles of Programming Languages
  - **ASPLOS**: Architectural Support for Programming Languages and Operating Systems
  - **CGO**: Code Generation and Optimization
  - **CC**: Compiler Construction
  - And there is a very important journal in this field: **TOPLAS** – ACM Transactions on Programming Languages and Systems.

CGO · *Code Generation and Optimization*

The International Symposium on Code Generation and Optimization (CGO) provides a premier venue to bring together researchers and practitioners working at the interface of hardware and software on a wide range of optimization and code generation techniques and related issues.

41 - Intel Corporation
22 - University of Edinburgh
18 - University of Illinois at Urbana-Champaign
16 - Google Inc.
13 - Princeton University
12 - IBM Thomas J. Watson Research Center
12 - University of New South Wales
11 - Pennsylvania State University
11 - Ohio State University
10 - University of Texas at Austin
10 - University of Virginia
10 - Swiss Federal Institute of Technology, Zurich
9 - Massachusetts Institute of Technology
9 - University Michigan Ann Arbor

8 - North Carolina State University
8 - Purdue University
8 - Indian Institute of Science, Bangalore
8 - The College of William and Mary
8 - Chinese Academy of Sciences
8 - Georgia Institute of Technology
8 - Microsoft Research
8 - University of California, Santa Barbara
7 - University of Michigan
7 - Rice University
7 - Stanford University
7 - INRIA
7 - Federal University of Minas Gerais
...
And 168 more institutions

Strong Participation of the Industry

Researchers affiliated with the following companies will be presenting papers at the **International Symposium on Code Generation and Optimization** (CGO 2024)

In total, 60 authors who contributed to 16 of the 37 papers accepted at CGO 2024 are researchers working in the industry

https://conf.researchr.org/home/cgo-2024

Compilers Laboratory

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSIDADE FEDERAL DE MINAS GERAIS
FEDERAL UNIVERSITY OF MINAS GERAIS, BRAZIL

A TIME OF EXCITEMENT

DCC 888

Tensor Compilers

Why this growth?

Tensor Compilers

Frontend & Backend

Image taken from https://huyenchip.com/2021/09/07/

Where is the challenge?

Tensor Compilers

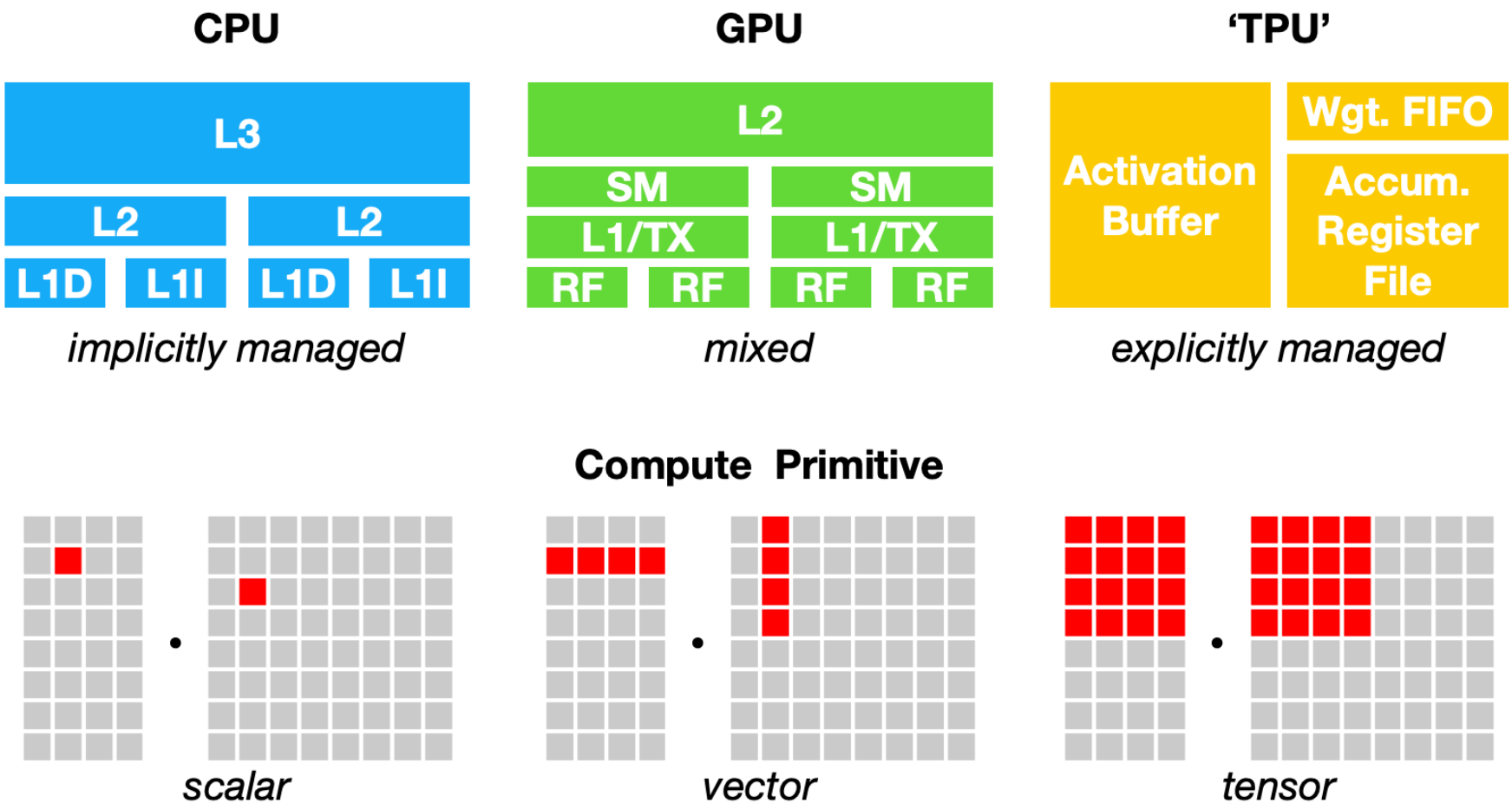
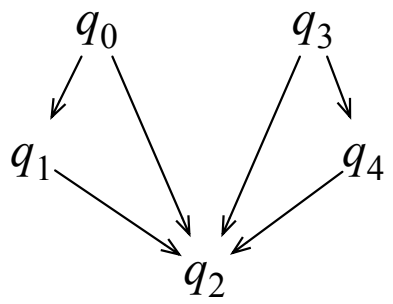


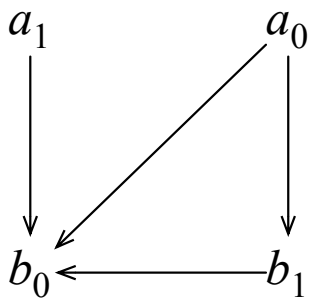
Image taken from https://huyenchip.com/2021/09/07/

Quantum Computers

Target Architecture

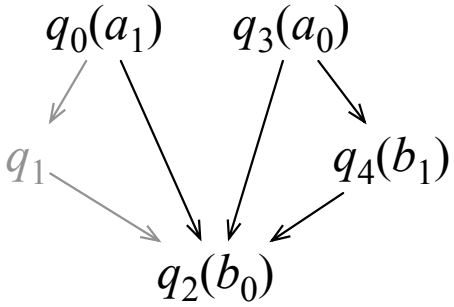


Logical circuit



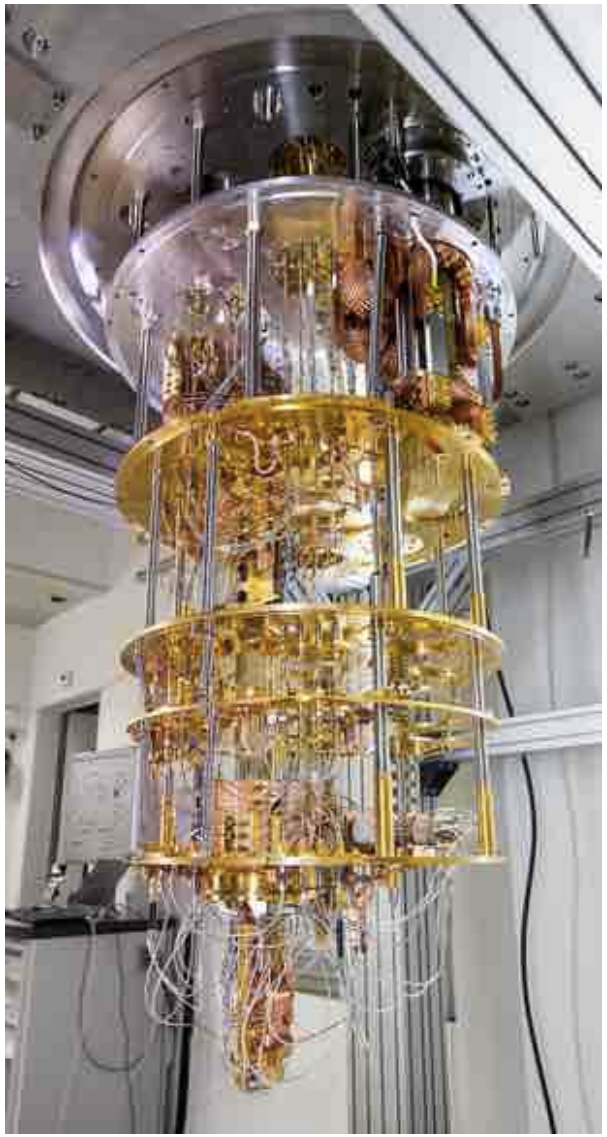
$+$

$=$



Running Program

IBM Research, Zürich



WebAssembly

WebAssembly became a World Wide Web Consortium recommendation on 5 December 2019 and, alongside HTML, CSS, and JavaScript, it is the fourth language to run natively in browsers.

```
local.get 0
i64.eqz
if (result i64)
    i64.const 1
else
    local.get 0
    local.get 0
    i64.const 1
    i64.sub
    call 0
    i64.mul
end
```

Compilers Laboratory

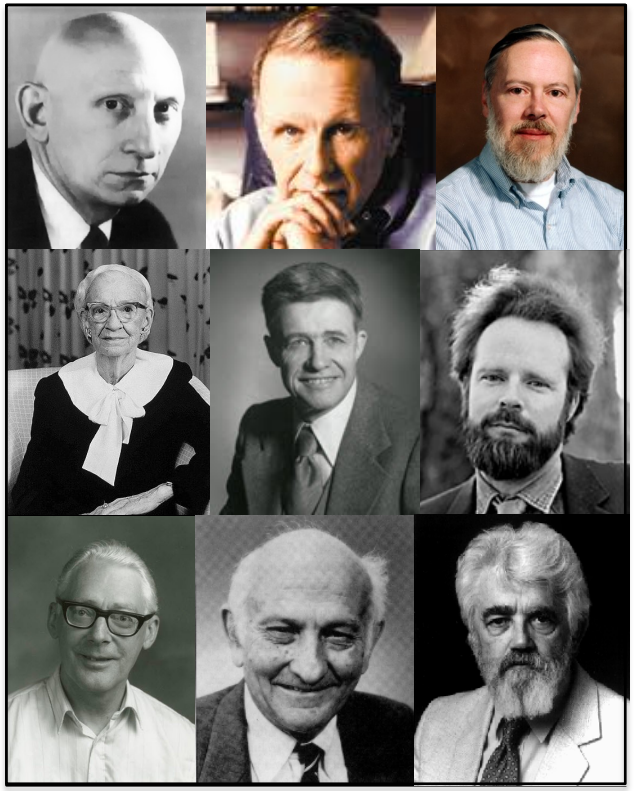
Department of Computer Science
Universidade Federal de Minas Gerais
Federal University of Minas Gerais, Brazil

A Brief History of Optimizing Compilers

DCC 888

Compiler Writers

The history of compilers is tightly bound with the history of computer science itself. It is fair to say that without compilation technology the progress of informatics would have been much slower, and probably we would not have internet, www, social networks, gene sequencing, fancy video games, or anything that requires some sort of non-trivial coding.



1) Who do you know in these photos?

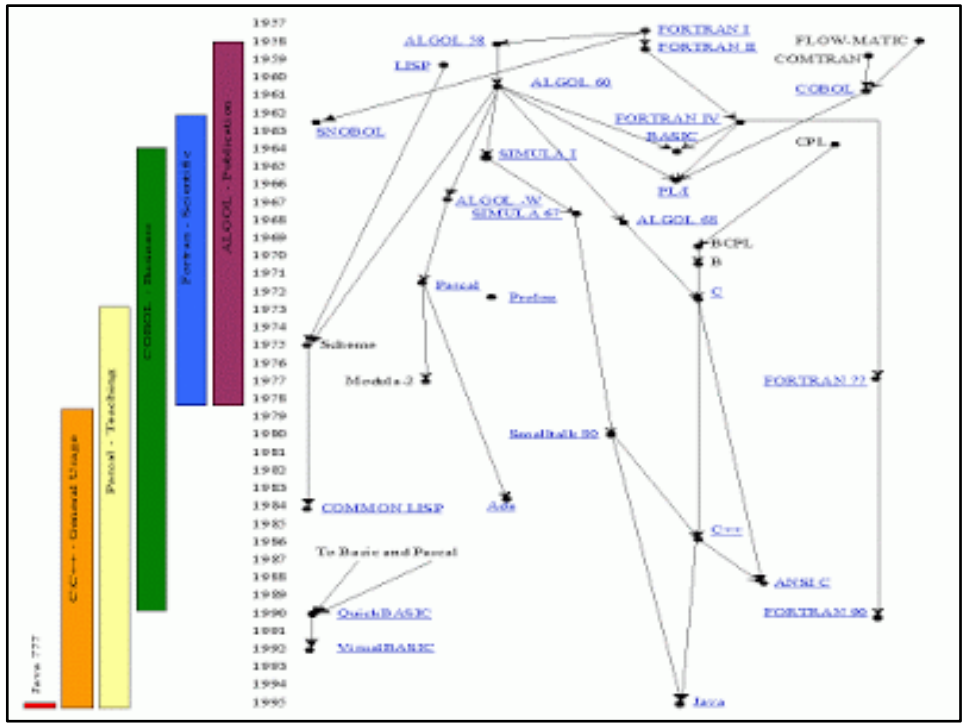
2) Can you point some milestone in the history of compilers?

The Long Road

1) What were the first compilers?

2) What were the “first compilation challenges”?

3) Who were the important people in compilation?



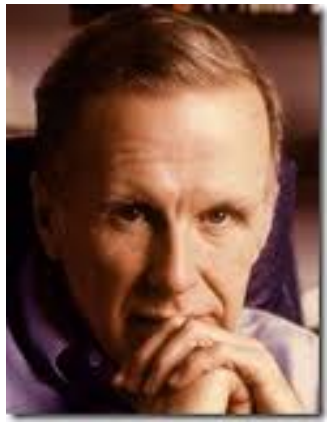
The First Documented Compiler Implementations

- Grace Hopper developed the A0-System, while working at Eckert-Mauchly Computer Corporation
  - Target: UNIVAC I
  - Years: 1951-52

> Automatic coding may, someday, replace the coder or release him to become a programmer. [...] Please, remember, however, that automatic programming does not imply that it is now possible to walk up to a computer, say "write my payrool checks", and push a button. Such efficiency is still in the science-fiction future.[§]

§: "Automatic Programming: The A 2 Compiler System - Part I". Computers and Automation. 4 (9): 25–29. Sep 1955.

The Dawn of the First Compilers



- Serious effort to move the task of generating code away from programmers started in the 50's.
- Fortran was one of the first programming languages, still in use today, to be compiled by an optimizing compiler.
- The developers of Fortran's compiler had to deal with two main problems:
  - Parsing
  - Code optimization
    - register allocation

Fortran was designed to be easily compiled. How so?

Early Code Optimizations

- Frances E. Allen, working alone or jointly with John Cocke, introduced many of the concepts for optimization:
  - Control flow graphs
  - Many dataflow analyses
  - A description of many different program optimizations
  - Interprocedural dataflow analyses
  - Worklist algorithms
- A lot of these inventions and discoveries have been made in the IBM labs.

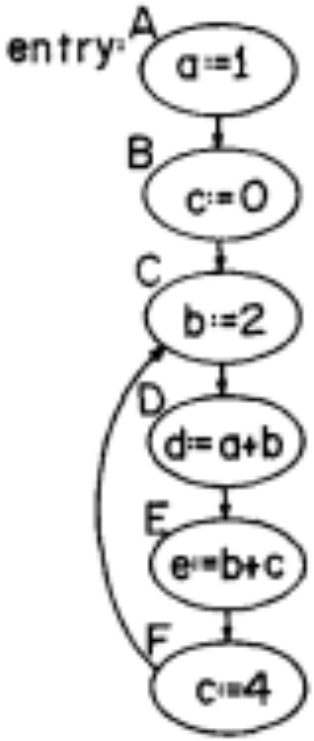
The Dataflow Monotone Framework

- Most of the compiler theory and technology in use today is based on the notion of the dataflow monotone framework.
  - Propagation of information
  - Iterative algorithms
  - Termination of fixed point computations
  - The meet over all paths solution to dataflow problems
- These ideas came, mostly, from the work of Gary Kildall, who is one of the fathers of the modern theory of code analysis and optimization.

The inventor of the BIOS system!

In addition of being paramount to the development of modern compiler theory, Gary Kildall used to host a talk show called "*The Computer Chronicles*". Nevertheless, he is mostly known for the deal with the Ms-DOS system that involved IBM and Bill Gates.

The Dataflow Monotone Framework

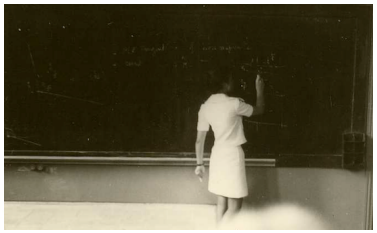


That's how the compiler would see a program in 1973!

- These ideas came, mostly, from the work of Gary Kildall, who is one of the fathers of the modern theory of code analysis and optimization.

Abstract Interpretation

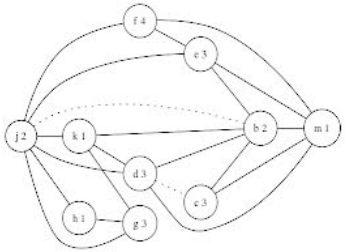
- Cousot and Cousot have published one of the most important papers in compiler research♤, giving origin to the technique that we call *Abstract Interpretation*.
- Abstract interpretation gives us information about the static behavior of a program.
- We could interpret a program to find out if a property about it is true.
  - But the program may not terminate, and even if it does, this approach could take too long.
  - So, we assign abstract states to the variables, plus an operator called widening, that ensures that our interpretation terminates.
  - This approach may be conservative, but it does terminate!

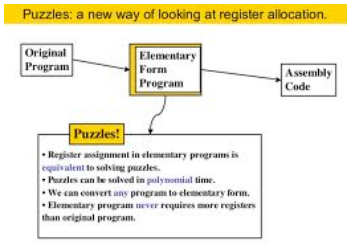


♤: Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints

Register Allocation

- Register allocation has been, since the early days of compilers, one of the most important code optimizations.
- Register allocation via graph-coloring was introduced by Gregory Chaitin in 1981.
- Linear scan, a register allocation algorithm normally used by JIT compilers, was introduced by Poletto and Sarkar in 1999.
- Linear scan and/or graph coloring are present in most of the modern compilers.





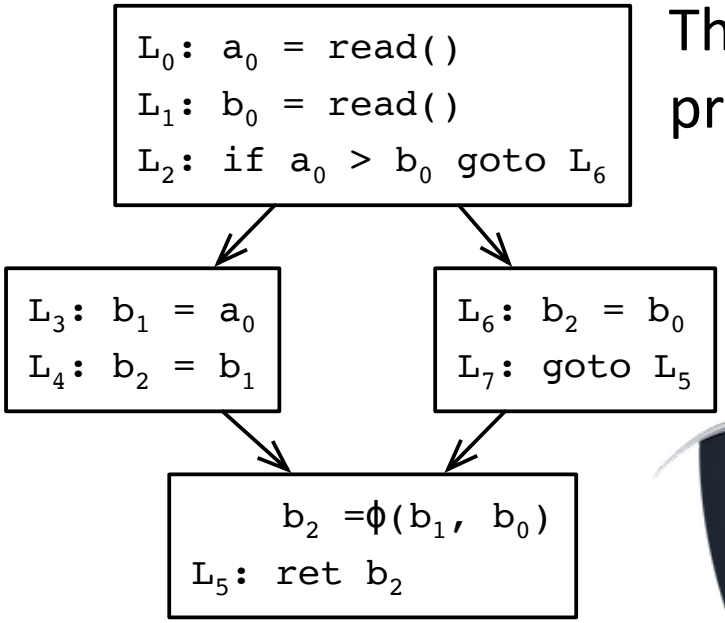
Static Single Assignment

- Once in a while we see smart ideas. Perhaps, the smartest idea in compiler optimization was the Static Single Assignment Form.
  - A program representation in which every variable has only one definition site.
- SSA form was introduced by Cytron *et al*., in the late eighties♡ in IBM.
- There were many improvements since then, such as pruned SSA form, SSA-based register allocation, etc.
- The idea took off very quickly. Today almost every compiler uses this intermediate representation.

SSA Seminar: celebrated the 20th anniversary of the Static Single Assignment form, April 27-30, Autrans, France

♡: An Efficient Method of Computing Static Single Assignment Form

Static Single Assignment

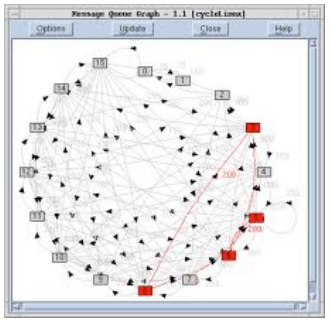


That's how the compiler would see a program in 2004, thanks to SSA form!

Constraint Based Analyses

- Control flow analysis was introduced by Olin Shivers in PLDI'88.
- Pointer analysis is the offspring of many fathers.
  - Lars O. Andersen launched the foundations of inclusion based-pointer analysis in 1994.
  - In 1996, Bjarne Steensgaard described a less precise pointer analysis that could be solved in almost linear time.
  - There is still much research in points-to analyses.

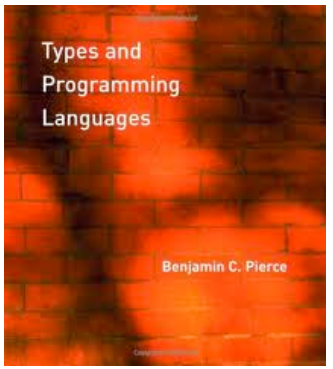
1) Which problems do we solve with control flow analysis? E.g., 0-CFA?
2) What is pointer analysis?
3) Which problems do we solve with pointer analyses?



Consolidation of Type Theory

- Types have been around for a long time.
  - Philosophers and logicians would rely on types to solve paradoxes in Set Theory.
- A lot of work has been done by researchers in the functional programming field, such as Philip Wadler.

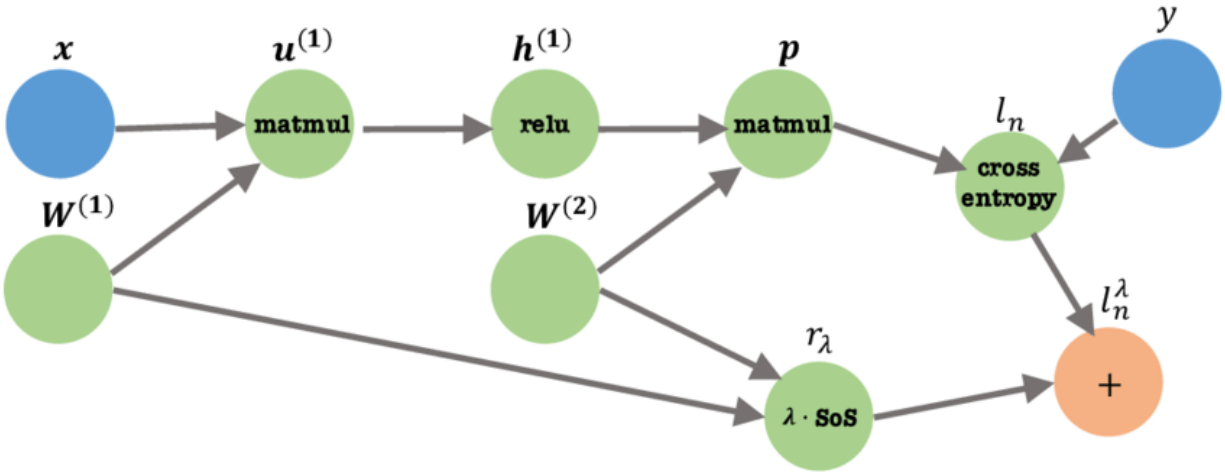
- Benjamin Pierce wrote a book, "Types and Programming Languages", in the early 2000's, that has been very influential in the field.
- A major boost in the mechanical verification of theorems is due to several independent groups, such as Xavier Leroy's (the CompCert compiler) and Frank Pfenning's (the Twelf system).



Multi-Level Intermediate Representation

- Fundational methodology to design new intermediate representations.
- Built onto the LLVM compiler.
- Easy to represent the computational graphs that implement neural networks.

Chris Lattner
(LLVM/MLIR)



MLIR

Multi-Level Intermediate Representation

That's how the compiler would see
"`y = ReLU(matmul(x, W) + b)`" in 2026!

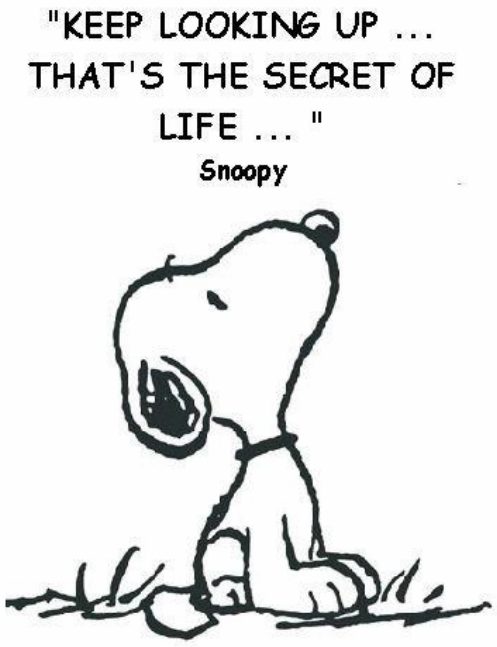
```
func @model(%x: tensor<1x128xf32>,
            %W: tensor<128x64xf32>,
            %b: tensor<64xf32>) -> tensor<1x64xf32> {

  %0 = linalg.matmul ins(%x, %W)
      : tensor<1x128xf32>, tensor<128x64xf32>
       -> tensor<1x64xf32>

  %1 = arith.addf %0, %b
      : tensor<1x64xf32>

  %2 = mhlo.relu %1
      : tensor<1x64xf32>

  return %2
}
```

MLIR

Compilers Laboratory

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSIDADE FEDERAL DE MINAS GERAIS
FEDERAL UNIVERSITY OF MINAS GERAIS, BRAZIL

THE FUTURE



DCC 888

The Ever Increasing Gap

The Programming Languages are always evolving, usually towards higher levels of abstraction. And the hardware is also always evolving, towards greater performance.

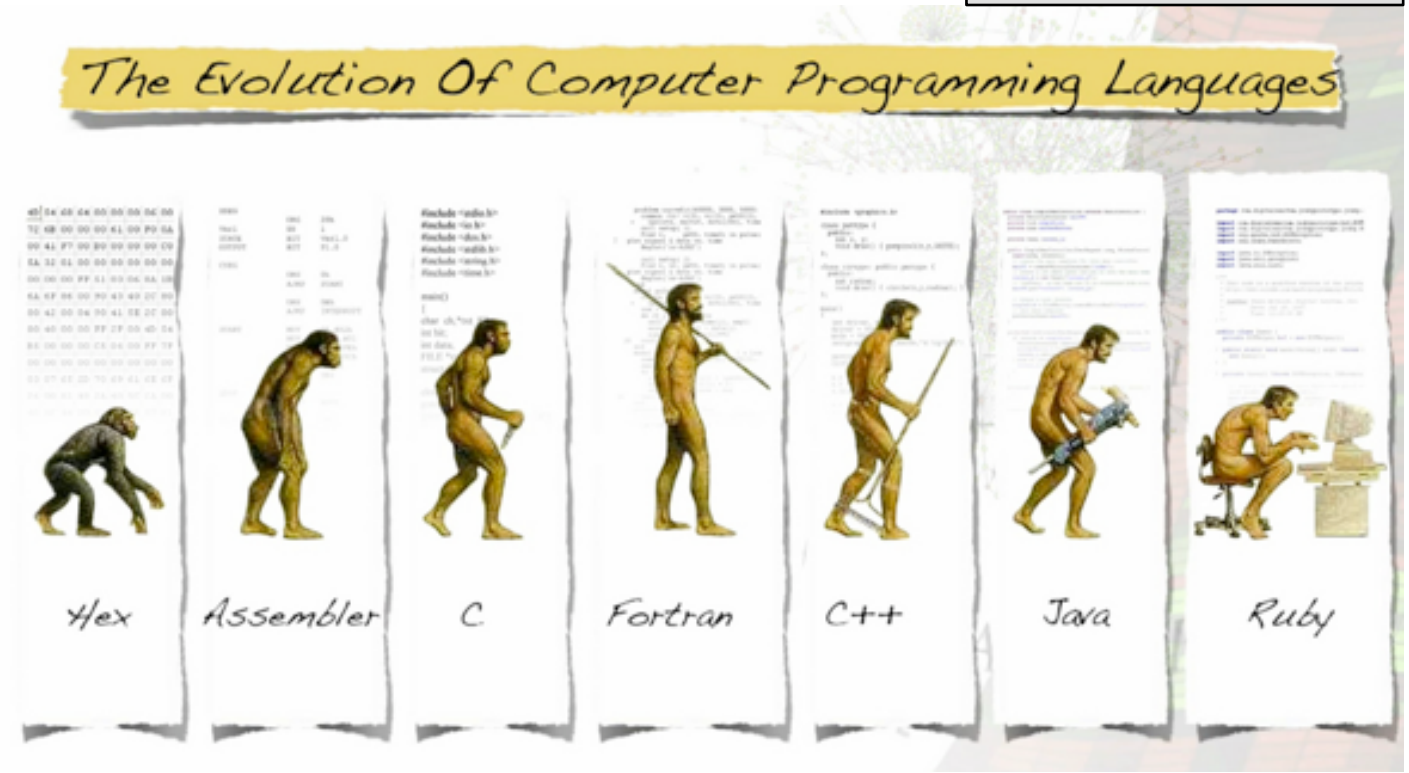
The compiler must bridge this ever increasing gap.

1) How will the languages of the future look like?

2) How will the hardware of the future look like?

3) Where is the research in compilers heading?

4) Is the compiler expert becoming more or less important?



The Challenges of Today

- Parallelism
- Dynamic Languages
- Correctness
- Security

1) Why are these problems an issue?
2) Who is trying to solve these problems?
3) What are the current solutions?
4) Which conference accepts results in these fields?
5) What are the unsolved questions?

The Future of Compiler Optimization

- The Full-Employment Theorem ensures that compiler writers will have a lot of work to do in the days ahead.
- Some influent scientists believe that research and development in the field will be directed towards two important paths in the coming years♤:
  - Automatic parallelization of programs.
  - Automatic detection of bugs.
- Given that everything happens so fast in computer science, we are likely to see these new achievements coming!
  - And they promise to be super fun ☺

♤: Hall, Padua e Pingali "Compiler Research: The Next Fifty Years" *Communications of the ACM 2009*

The Complete Text (by Snoopy)

It Was A Dark And Stormy Night

Part I

It was a dark and stormy night. Suddenly, a shot rang out! A door slammed. The maid screamed.

Suddenly, a pirate ship appeared on the horizon!

While millions of people were starving, the king lived in luxury. Meanwhile, on a small farm in Kansas, a boy was growing up.

Part II

A light snow was falling, and the little girl with the tattered shawl had not sold a violet all day.

At that very moment, a young intern at City Hospital was making an important discovery. The mysterious patient in Room 213 had finally awakened. She moaned softly.

Could it be that she was the sister of the boy in Kansas who loved the girl with the tattered shawl who was the daughter of the maid who had escaped from the pirates?

The intern frowned.

"Stampede!" the foreman shouted, and forty thousand head of cattle thundered down on the tiny camp. The two men rolled on the ground grappling beneath the murderous hooves. A left and a right. A left. Another left and right. An uppercut to the jaw. The fight was over. And so the ranch was saved.

The young intern sat by himself in one corner of the coffee shop. he had learned about medicine, but more importantly, he had learned something about life.

THE END