Welcome!

Simone Campanoni
simonec@eecs.northwestern.edu
Who we are

Simone Campanoni
simonec@eecs.northwestern.edu

Ettore M. G. Trainiti
ettoretrainiti2022@u.northwestern.edu
What we are going to do

• Teach you **code analysis and transformation**

• What they do
  • What they could do

• What they can’t do
Who you are (or will be)

• An engineer

• A C++ developer
  (you don’t have to be an incredible coder)

• An enthusiastic learner

Compiler expert is not mentioned ;}
Software knowledge assumed

• You know how to write C++ code in Linux platforms (e.g., class, inheritance, method overloading, containers like a set)

• You know Makefile
  Makefile tutorial:
  http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor

• You know how to debug C++ code (e.g., gdb)
Machines to use for this class

You have access to the following machines, which are used to test your homework

• Wilkinson lab
gotham.ece.northwestern.edu, batman.ece.northwestern.edu, robin.ece.northwestern.edu, alfred.ece.northwestern.edu, gordon.ece.northwestern.edu, madhatter.ece.northwestern.edu, joker.ece.northwestern.edu, cobblepott.ece.northwestern.edu, bane.ece.northwestern.edu, nightwing.ece.northwestern.edu, selina.ece.northwestern.edu, ras.ece.northwestern.edu, poisonivy.ece.northwestern.edu, freeze.ece.northwestern.edu, scarecrow.ece.northwestern.edu, clayface.ece.northwestern.edu, harley.ece.northwestern.edu, killercroc.ece.northwestern.edu, huntress.ece.northwestern.edu, batgirl.ece.northwestern.edu, riddler.ece.northwestern.edu, hush.ece.northwestern.edu

• WOT systems
murphy.wot.ece.northwestern.edu, finagle.wot.ece.northwestern.edu, hanlon.wot.ece.northwestern.edu, moore.wot.ece.northwestern.edu
Outline of today’s CAT

• Structure of the course

• CAT and compilers

• CAT and computer architecture

• CAT and programming language
CS 323 CAT in a nutshell

• About: understanding and transforming code automatically
• Satisfy the system depth for CS major
• Tuesday/Thursday 5:00pm – 6:20pm

• Simone’s office hours: Wednesday noon – 1:00pm
• Ettore’s office hours: Monday 4:30pm – 5:30pm

• CAT is on Canvas
  • Materials/Assignments/Grades on Canvas
  • You’ll upload your assignments on Canvas
CAT materials

- Modern compiler implementation
- Slides and assigned papers
- LLVM documentation

http://llvm.org
CAT slides

- You can find last year slides from the [class website](#).
- We improve slides every year
  - based on problems we will observe during the next 10 weeks
  - as well as your feedbacks we will ask you at the end
  - Our goal: maximize how much you learn in 10 weeks

- We will upload to Canvas the new version of the slides just before each class.
The CAT structure

Today

Week

Tuesday

Thursday Homework

Topic & homework

12/8
The CAT grading

- Homework: 100 points
  - 10 points per assignment
  - The first 2 assignments are easy

- Extra points
  - Extra homework
  - Answering (correctly) special questions (I will emphasize them) during lectures
  - Best student so far: **114 points**!

<table>
<thead>
<tr>
<th>Grade</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>95 – 100</td>
</tr>
<tr>
<td>A -</td>
<td>90 – 94</td>
</tr>
<tr>
<td>B +</td>
<td>80 – 89</td>
</tr>
<tr>
<td>B</td>
<td>70 – 79</td>
</tr>
<tr>
<td>B -</td>
<td>61 – 69</td>
</tr>
<tr>
<td>C +</td>
<td>57 – 60</td>
</tr>
<tr>
<td>C</td>
<td>50 – 56</td>
</tr>
<tr>
<td>D</td>
<td>40 – 49</td>
</tr>
<tr>
<td>F</td>
<td>0 – 39</td>
</tr>
<tr>
<td>Pass</td>
<td></td>
</tr>
<tr>
<td>No pass</td>
<td></td>
</tr>
</tbody>
</table>
The CAT competition

• At the end, there will be a competition between your CATs

• The team that designed the best CATs
  • Get an A automatically (no matter how many points they have)
  • Their names go to the “hall of fame” of this class
Rules for homework

• You are encouraged (but not required) to work in pairs
  • Pair programming is *not* team programming
• No copying of code is allowed between pairs
• Tool, infrastructure help is allowed
  • First try it on your own
    (google and tool documentation are your friends)
• Avoid plagiarism
  [www.northwestern.edu/provost/policies/academic-integrity/how-to-avoid-plagiarism.html](http://www.northwestern.edu/provost/policies/academic-integrity/how-to-avoid-plagiarism.html)
• If you don’t know, please ask: simonec@eecs.northwestern.edu
Summary

• My duties
  • Teach you code analysis and transformation
  • And how to implement them in a production compiler (LLVM)

• Your duties
  • Learn code analysis and transformation
  • Implement a few of them in LLVM
    • Write code
    • Test your code
    • Then, think much harder about how to actually test your code
    • (Sometimes) Answer my questions about your code

No final exam
Structure & flexibility

• CAT is structured w/ topics

• Best way to learn is to be excited about a topic

• Interested in something?

   Speak

   I’ll do my best to include your topic on the fly
Week 1

Today
- Welcome/Structure
- Compiler/CAT

Thursday
LLVM

Today 12/8
The role of compilers

If there is no coffee, if I still have work to do,
I’ll keep working, I’ll go to the coffee shop.

Code analysis and transformation

001010101110010101010010101010110
Example of CAT

```python
varX = 5
...
...
...
...
print varX
...
```

What will it print?
Example of CAT

```plaintext
varX = 5
...
...
...
...
...
What will it print? print varX
...
...
print 5
```
Example of CAT

\[
\begin{align*}
\text{varX} &= 5 \\
\text{...} \quad \text{...} \\
\text{...} \quad \text{...} \\
\text{...} \quad \text{...} \\
\text{print 5} \quad \text{print \ varX} \\
\text{...} \quad \text{...}
\end{align*}
\]

Is it worth transforming this code?
Designing CATs

• Choose a goal
  • Performance, energy, identifying bugs, discovering code properties

• Design automatic analysis to obtain the required information

• Occasionally design the code transformation
Use of CATs

• Compilers
  • Increase performance
  • Decrease energy consumption
  • Decrease code size
  • Drive the code generation

• Developing tools (e.g., VIM, EMACS)
  • Understanding code (e.g., scopes, variables)

• Computer architecture
Structure of a compiler

Character stream (Source code)

Lexical analysis

Tokens

Syntactic & semantic analysis

AST

int main()

printf("Hello World!");

return 0;
Structure of a compiler

Character stream (Source code)

Lexical analysis

Tokens

Syntactic & semantic analysis

AST

```
int main ... |
```

```
INT SPACE STRING SPACE ... |
```

```
Function signature

Return type

Function name

INT

STRING
```
Structure of a compiler

Syntactic & semantic analysis

AST

IR code generation

IR

Function signature

Return type

Function name

INT

STRING

; Function Attrs: nounwind uwtable
define int @main() {

Structure of a compiler

Character stream (Source code)

Front-end

IR

Middle-end

IR

Back-end

Machine code

CS 322: Compiler Construction

; Function Attrs: nounwind uwtable
define int @main() {

int main ... ...

Code analysis and transformation

; Function Attrs: nounwind uwtable
define int @main() {

CS 322: Compiler Construction

010101110101010101
Structure of a compiler

Character stream (Source code) → Front-end → Middle-end → Back-end → Machine code

Character stream (Source code) → Front-end → Middle-end → Back-end → Machine code
Structure of a compiler

C → Front-end → IR → Middle-end → IR → Back-end → Machine code

C → Front-end → Middle-end → Back-end → Machine code
Structure of a compiler

- **Front-end**
  - IR
  - Middle-end
    - IR
    - Back-end
      - Machine code
- **Java**
  - Front-end
  - Middle-end
  - Back-end
  - Machine code
Structure of a compiler

Front-end

C

IR

Middle-end

Java

IR

Back-end

Machine code

FE

Front-end

Middle-end

Back-end

Java

M2

Machine code
Structure of a compiler

Front-end

Middle-end

Back-end

Machine code

Front-end

Middle-end

Back-end

Java

M2

C

IR

Java

FE

IR

BE

M2
Structure of a compiler

Front-end 1

Middle-end

Back-end A

Back-end B
Multiple IRs

• Abstract Syntax Tree

• Register-based representation (three-address code)
  \[ R1 = R2 \text{ add} R3 \]

• Stack-based representation
  \[ \text{push 5; push 3; add; pop ;} \]

IR needs to be easy
1) to produce
2) to translate into machine code
3) to transform/optimize
Example of LLVM IR

define i32 @main(i32 %argc, i8** %argv) {
  entry:
    %add = add i32 %argc, 1
    ret i32 %add
}
Multiple IRs used together

L1

Static compiler

IR1

Dynamic compiler FE

IR2

Dynamic compiler BE

Machine code
Multiple IRs used together

Java

Java compiler

Java bytecode

Java VM FE

IR2

Java VM BE

Machine code
CATs that we’ll focus on

• Semantics-preserving transformations
  • Correctness guaranteed

• Goal: performance

• Automatic

• Efficient
Evolution of CATs (hardware point of view)

- Simple hardware (few resources), simple CATs

---

```
Size
  La
Spec

Performance
1000
100
10
1

Time
```

"Moore's Law"

Processor-Memory Performance Gap: (grows 50% / year)

μProc 60%/yr.

DRAM 7%/yr.
Evolution of CATs (hardware point of view)

- Simple hardware (few resources), simple CATs
- More hardware resources available to compilers
- Opportunities to improve programs
- Challenging CATs

Compilers/CATs are developed in the processor-design stage!
Evolution of CATs (hardware point of view) (2)

1960 - ?: Complex instruction set computing (CISC)

1980 - ?: Reduced instruction set computer (RISC)
Evolution of CATs (hardware point of view) (3)

Superscalar

Inst 1
Inst 2
Inst 3
Inst 4
Inst 5
Inst 6
Inst 7
Inst 8

CATs

Very long instruction word (VLIW)

Inst 1  Inst 4  Inst 7  Inst 8

Inst 2  Inst 5  Inst 3  Inst 6
Evolution of CATs (PL point of view)

• First electronic computers appeared in the ’40s
• They were programmed in machine language

00101010111001010101001010101011010

• Low level operations only
  • Move data from one location to another
  • Add the contexts of two registers
  • Compare two values

• Programming: slow, tedious, and error prone
Evolution of CATs (PL point of view)

• Low level programming language, simple CATs
  • Not very productive

• More abstraction in programming language, more work for CATs to reduce their performance overhead
  • Macros -> Fortran, Cobol, Lisp -> C, C++, Java, C#, Python, PHP, SQL, ...

• CATs enable new programming languages
Evolution of CATs (PL point of view)

• Abstractions are great for productivity

• CATs remove their overhead

• But abstractions must be carefully evaluated considering CATs

• A simple abstraction in PL can generate challenges for CATs
  • CATs need to be understood
Evolution of CATs (PL point of view)(2)

PL without procedures

```c
void main (){ 
    Int v1,v2; 
    v1 = 1; 
    v2 = 2; 
    ... 
}
```
Evolution of CATs (PL point of view)(3)

Let’s add procedures to our PL

• Call-by-Value
  void proc1 (int a){...}  void proc1 (int *a){...}
  proc1(myVar1);  proc1(&myVar1);

• Call-by-Reference
  void proc1 (int a){...}
  proc1(myVar1);
void myProc (int *v1, int *v2)
{
    (*v1) = 1;
    (*v2) = 2;
}

What’s the problem for CATs? ... if v1 and v2 alias ...

Understanding if pointers alias: pointer alias analysis

This is one of the most challenging problem in CATs
Conclusion

• CATs used for multiple goals
  • Enable PLs
  • Enable hardware features

• CATs are effected by
  • Their input language
  • The target hardware

• When you design a PL or a new hardware platform, you need to understand what CATs **can** and **can’t** do
  • Often: a **can’t** becomes **can** thanks to research on CATs
Ideal CATs

• Proved to be correct

• Improve performance of many important programs

• Minor compilation time

• Negligible implementation efforts
As Linus Torvalds says ...

*Talk is cheap. Show me the code.*

Demo time