L1: Introduction to OPL

Rachata Ausavarungnirun

(rachata.a@tggs.kmutnb.ac.th)

January 7th, 2020

Architecture Research Group

SSE, TGGS

Administrative Stuff

Class Website

- Please sign-up on Canvas
  - Sign-up link: **https://canvas.instructure.com/enroll/FJMWBB**
    - Enrollment code **FJMWBB**

- This is where all the information from this class is posted
  - Class policy and syllabus
  - Class schedule
  - Announcement
  - Assignments

Class Policy

- **No plagiarism**
  - Everything will have to be from your own work
  - You need to put proper citations/references to your source
    - Max(grade) * number of times you got caught
- **5 late days total, 2 per assignment max**
- **Office hours:** I will be around after the lecture
- I encourage you to discuss material with your classmates and work together, **but each student must**
  - Write his/her own code
  - Clearly indicate who you have worked with

Grading Breakdowns

- Assignments 30%
- Project 20%
- In-class exercise 5%
- Quiz 20%
- Final 20%

- I can curve anything above to make sure everything is fair

Class Project

- Open-end
  - Build whatever you want, but they should utilize knowledge you learn from this class

- We will kick start this after the midterm
  - But you are all welcome to discuss your ideas as early as right after this lecture

Language Used in This Class

- We will use a few languages to show different concepts
  - Python
  - Standard ML
  - Scala
  - Rust

In-class Exercise

- Please bring a laptop
- There will be both lecture slides and coding exercises
- If there is not enough outlets, please let me know now

My Expectation

- There will be a lot of new way of coding
  - Functional programming will feel very different than imperative programming
  - Applies to both the assignments and the project
- Workload will be heavy
  - Start your assignment early is always a good idea
- You should have a good grasp of
  - Intro to programming (Python)
  - Intermediate programming (JAVA)
- You should have some basic on
  - Computer system
  - Computer hardware

What Will You Learn?

The Goal of This Course

- You should be able to:
  - Know essential concepts related to programming languages
  - Know the benefit of parallel programming
  - Know how to increase parallelism (more performance)

Historical Context

Dawn of Digital Computing

- Computer has been around for a long time
  - Mechanical calculator
  - Jacquard’s loom

© Wikipedia

ENIAC

© Rachata Ausavarungnirun
ENIAC
THE WORLD'S FIRST ELECTRONIC, LARGE SCALE
GENERAL-PURPOSE DIGITAL COMPUTER

- Eckert and Mauchly
  - Univ. of Penn
  - 18,000 Vacuum tubes
  - 30 tonnes, 80x8.5 feet
  - 20 decimal-digit words
  - Programmed using 3000 switches (all those plugs you see in the picture)

The 40s and the 50s

- Hardware advances
  - ABC (Atanasoff and Berry)
  - Z3, Z4 (Zuse)
  - Colossus (Turing)
  - ENIAC (Eckert and Mauchly)
  - EDVAC (von Neumann)
  - EDSAC (Wilkes) → First stored-program!
  - IAS (Bigelow)
- Emergence of software
  - Fortran in 1954

Modern Computers

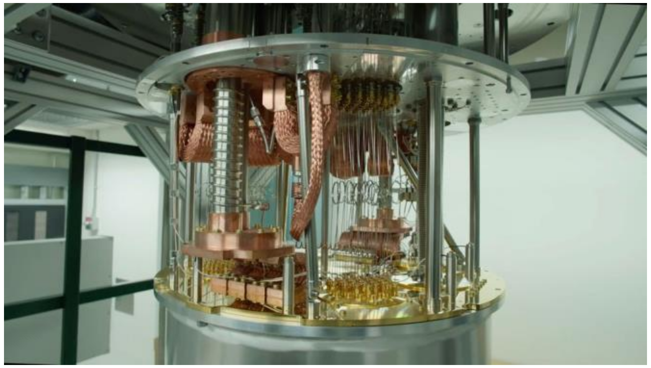
www.raspberrypi.org

www.nvidia.com

www.llnl.gov

www.apple.com

www.gopro.com



www.bloomberg.com

Designing a Programming Languages

Design Tradeoffs for Prog. Lang.

- Syntax and complexity of the code
- Semantics
- Paradigms that the language favors
- Type system and type rules
- Memory management
- Need a compiler?

Programming Languages Over Time

- Early day (1950s – 1960s)
  - Language mirrors hardware concepts
    - Compiler optimization is expensive and mostly impossible
  - Programmer is much cheaper compare to machines
    - Parts are costly
    - Programs has to be very efficient from the get-go
- Now
  - Language centers on design concepts
    - Includes things like objects, records, functions
  - Machine is cheap and will continue to be cheaper
    - Scripting and inefficient codes are(???) ok, quick to develop
  - Optimized for resource constraints and design goals
    - Low power
    - High throughput, high parallelism

Emergence of Parallelism

The von Neumann Model

- Stored-program computer
- Two key properties
  - Programs (instructions) are stored in a linear memory array
  - Memory is unified between instructions and data
    - Control signal interpret whether stored values are data or instructions
- Sequential instruction processing
  - One instruction at a time
    - Fetch → executed → complete
  - Program counter (PC) identify the current instruction
    - PC is also referred to as Instruction Pointer (IP)
  - Program counter advanced sequentially except for control transfer instruction (e.g., branches)

The von Neumann Model

- Is this the only model? No
- But this is one of the most dominant
- All major instruction set architectures (ISA) today use this model
    - x86, ARM, MIPS, SPARC, Alpha, POWER
- What is the alternative?

The Dataflow Model

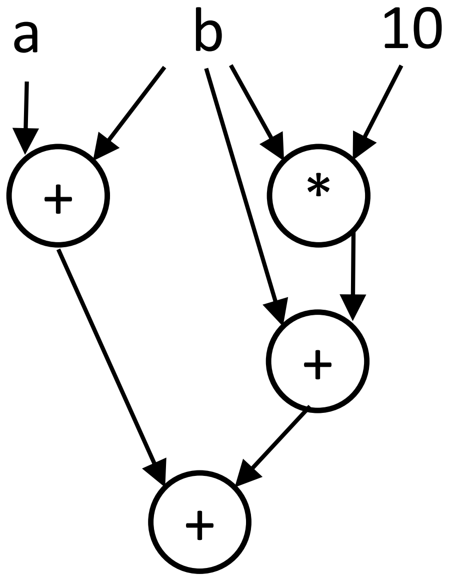
- Von Neuman: An instruction is fetched and executed in **control flow order**
    - Instruction pointer grabs the next instruction
    - Mostly sequential except control flow instructions
- Dataflow model: An instruction is fetched in the **data flow order**
    - Compute when operands are ready
    - No instruction pointer
    - Ordering is based on data flow dependence
        - Think of a math function
    - Many instruction can execute at the same time
        - **Parallelism** ☺

von Neumann vs. Data Flow

Sequential

C = A + B;
X = B * 10;
Y = B + X;
Z = C + Y;

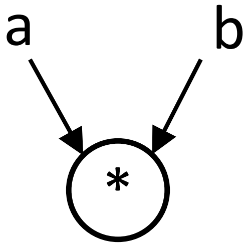
Dataflow



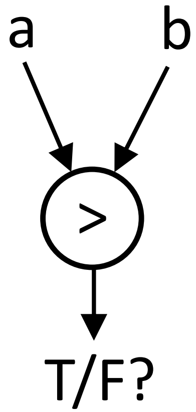
- Which is more natural as a programmer?

Types of Dataflow Nodes

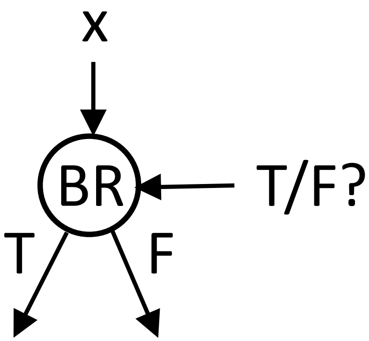
Computation



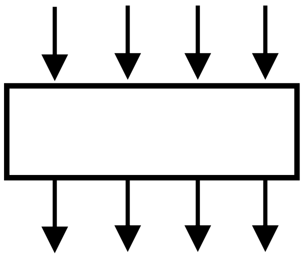
Relational



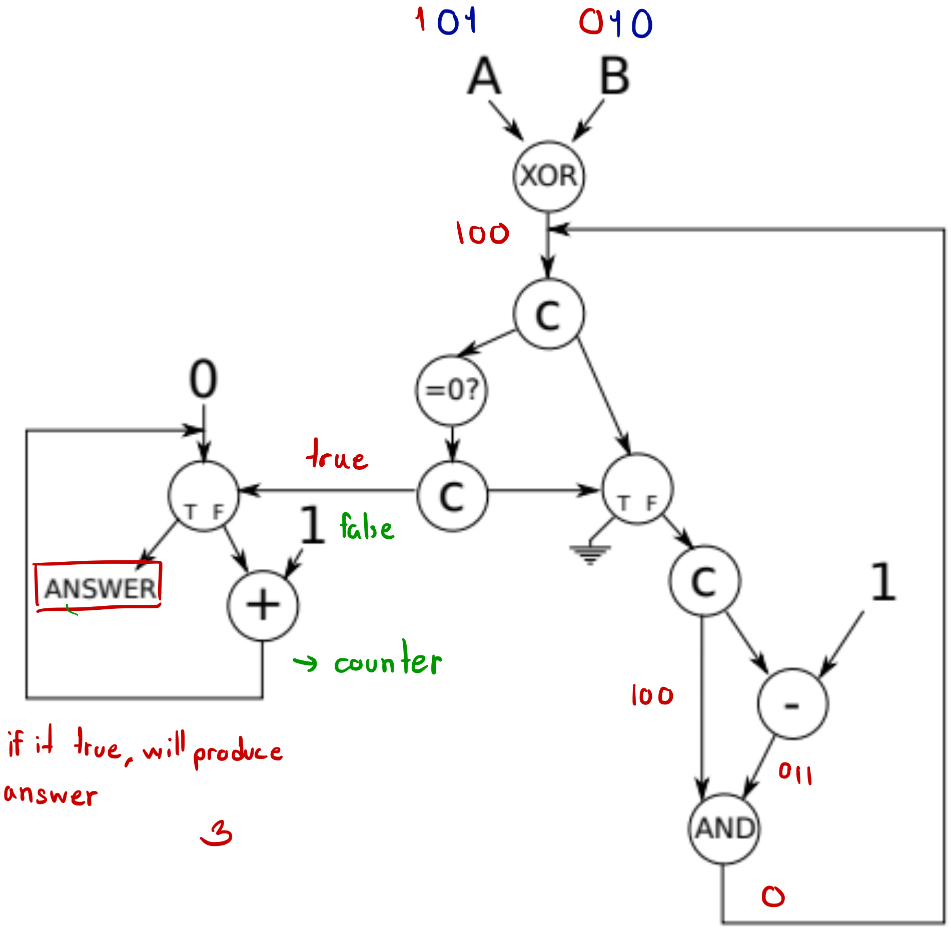
Conditional



Barrier/Synch



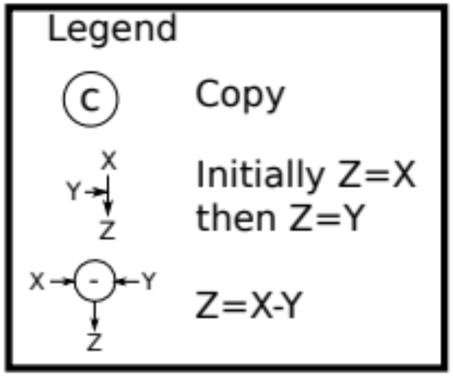
In-class Group Exercise



if it true, will produce answer

xor

A B	
0 0	0
0 1	1
1 0	1
1 1	0



Legend

Copy

Initially Z=X then Z=Y

Z=X-Y

If the XOR = 1 have to do 1 round or until it produces 0

- What does this dataflow program do?
  - Hint: do one side at a time

Depend on the value of A and B and the result of XOR