CS 61C:
Great Ideas in Computer Architecture
Introduction to C, Part I

Instructors:
Krste Asanovic
Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/F12

Agenda

- Review
- Compile vs. Interpret
- Python vs. Java vs. C
- Administrivia
- Quick Start Introduction to C
- Technology Break
- More C
- Summary

Review

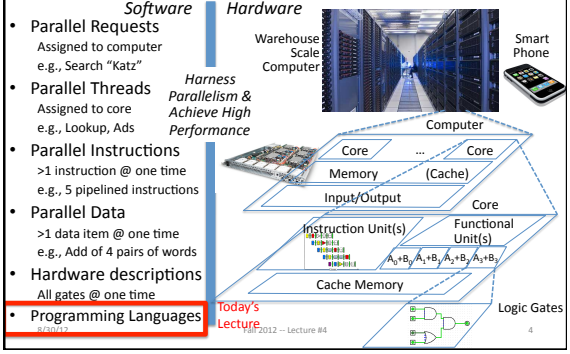
- Request-Level Parallelism
  - High request volume, each largely independent of other
  - Use replication for better request throughput, availability
- MapReduce Data Parallelism
  - Map: Divide large data set into pieces for independent parallel processing
  - Reduce: Combine and process intermediate results to obtain final result

New-School Machine Structures (It's a bit more complicated!)

Software *Hardware*

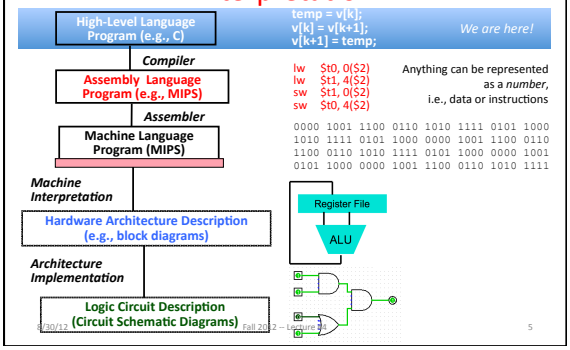
- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance



Today's Lecture

Big Idea #1: Levels of Representation/ Interpretation



Introduction to C "The Universal Assembly Language"

- "Some" experience is required before CS61C
  C++ or Java OK

- Class pre-req included classes teaching Java
- Java used in two labs and one project
- C used for everything else

Flash Card Language Poll!

Please raise card for *first* one of following you can say yes to

- ☐ I have programmed in C,C++,C#, or Objective-C
- ☐ I have programmed in Java
- ☐ I have programmed in FORTRAN, Cobol, Algol-68, Ada, Pascal, or Basic
- ☐ None of the above

7

Disclaimer

- You will not learn how to fully code in C in these lectures! Can only learn a language by using it!
- You'll need your C reference for this course
  - K&R is a must-have
    - Check online for more sources
  - "Java in a Nutshell," O'Reilly
    - Chapter 2, "How Java Differs from C"
    - http://oreilly.com/catalog/javanut/excerpt/index.html
  - Brian Harvey's helpful transition notes
    - On CS61C class website: pages 3-19
    - http://inst.eecs.berkeley.edu/~cs61c/resources/HarveyNotesC1-3.pdf
- Key C concepts: Pointers, Arrays, Implications for Memory management

Intro to C

- *C is not a "very high level" language, nor a "big" one, and is not specialized to any particular area of application. But its absence of restrictions and its generality make it more convenient and effective for many tasks than supposedly more powerful languages.*

  – Kernighan and Ritchie
- Enabled first operating system not written in assembly language: *UNIX* - A portable OS!
- C and derivatives (C++/Obj-C/C#) still one of the most popular application programming languages after >40 years!

TIOBE Index of Language Popularity

Position Aug 2012	Position Aug 2011	Delta in Position	Programming Language	Ratings Aug 2012	Delta Aug 2011	Status
1	2	⬆	C	18.937%	+1.55%	A
2	1	⬇	Java	16.352%	-3.06%	A
3	6	⬆⬆⬆	Objective-C	9.540%	+4.05%	A
4	3	⬇	C++	9.333%	+0.90%	A
5	5	=	C#	6.590%	+0.55%	A
6	4	⬇⬇	PHP	5.524%	-0.61%	A
7	7	=	(Visual) Basic	5.334%	+0.32%	A
8	8	=	Python	3.876%	+0.46%	A
9	9	=	Perl	2.273%	-0.04%	A
10	12	⬆⬆	Ruby	1.691%	+0.36%	A
11	10	⬇	JavaScript	1.365%	-0.19%	A
12	13	⬆	Delphi/Object Pascal	1.012%	-0.06%	A

Basic C Concepts

Compiler	Creates useable programs from C source
Typed variables	Kind of data that a variable contains
Typed functions	The kind of data returned from a function
Header files (.h)	Declare functions and variables in a separate file
Structs	Groups of related values
Enums	Lists of predefined values
Pointers	Aliases to other variables

These concepts distinguish C from other languages you may know

Integers: Python vs. Java vs. C

Language	sizeof(int)
Python	>=32 bits (plain ints), infinite (long ints)
Java	32 bits
C	Depends on computer; 16 or 32 or 64

- C: `int` should be integer type that target processor is most efficient working with
- Only guarantee: sizeof(`long long`) ≥ sizeof(`long`) ≥ sizeof(`int`) ≥ sizeof(`short`)
  - All could be 64 bits

C vs. Java

	C	Java
Type of Language	Function Oriented	Object Oriented
Program-ming Unit	Function	Class = Abstract Data Type
Compilation	gcc hello.c creates machine language code	javac Hello.java creates Java virtual machine language bytecode
Execution	a.out loads and executes program	java Hello interprets bytecodes

hello, world	`#include<stdio.h>`
	`int main(void) {`
	`  printf("Hello\n");`
	`  return 0;`
	`}`
		`public class HelloWorld {`
		`  public static void main (String[] args) {`
		`  System.out.printl("Hello");`
		`  }`
		`}`

Storage	Manual (**`malloc`**, **`free`**)	Automatic (garbage collection)

From http://www.cs.princeton.edu/introcs/faq/c2java.html

C vs. Java

	C	Java
Comments	`/* … */`	`/* … */` or `//` … end of line
Constants	`const, #define`	`final`
Preprocessor	Yes	No
Variable declaration	At beginning of a block	Before you use it
Variable naming conventions	`sum_of_squares`	`sumOfSquares`
Accessing a library	`#include <stdio.h>`	`import java.io.File;`

From http://www.cs.princeton.edu/introcs/faq/c2java.html

Compilation: Overview

- C *compilers* map C programs into architecture-specific machine code (string of 1s and 0s)
  - Unlike *Java*, which converts to architecture-independent *bytecode*
  - Unlike *Python* environments, which *interpret* the code
  - These differ mainly in exactly when your program is converted to low-level machine instructions ("levels of interpretation")
  - For C, generally a two part process of compiling .c files to .o files, then linking the .o files into executables;
  - Assembling is also done (but is hidden, i.e., done automatically, by default); we'll talk about that later

Compilation: Advantages

- Excellent run-time performance: generally much faster than Scheme or Java for comparable code (because it optimizes for a given architecture)
- Fair compilation time: enhancements in compilation procedure (Makefiles) allow only modified files to be recompiled
- Why C?: *we can write programs that allow us to exploit underlying features of the architecture – memory management, special instructions, parallelism*

Compilation: Disadvantages

- Compiled files, including the executable, are architecture-specific, depending on processor type and the operating system
- Executable must be rebuilt on each new system
  - I.e., "porting your code" to a new architecture
- "Change → Compile → Run [repeat]" iteration cycle can be slow, during the development cycle

Typed Variables in C

```
int   variable1  = 2;
float variable2  = 1.618;
char  variable3  = 'A';
```

- Must declare the type of data a variable will hold
  - Types can't change

Type	Description	Examples
int	integer numbers, including negatives	0, 78, -1400
unsigned int	integer numbers (no negatives)	0, 46, 900
float	floating-point numbers	0.0, 1.618, -1.4
char	single text character or symbol	'a', 'D', '?'
double	greater precision/big FP number	10E100
long	larger signed integer	6,000,000,000

Typed Functions in C

```
int number_of_people ()
{
  return 3;
}

float dollars_and_cents ()
{
  return 10.33;
}

char first_letter ()
{
  return 'A';
}
```

- You have to *declare* the type of data you plan to return from a function
- Return type can be any C variable type, and is placed to the left of the function name
- You can also specify the return type as `void`
  - Just think of this as saying that no value will be returned
- Also necessary to declare types for values passed into a function
- Variables and functions MUST be declared before they are used

Structs in C

- Structs are structured groups of variables, e.g.,

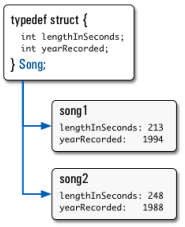
```
typedef struct {
  int length_in_seconds;
  int yearRecorded;
} Song;

Song song1;

Song1.length_in_seconds=  213;
song1.yearRecorded    = 1994;

Song song2;

Song2.length_in_seconds =  248;
song2.yearRecorded    = 1988;
```



Dot notation: `x.y = value`

Consts and Enums in C

- Constant is assigned a value once in the declaration; value can't change during entire execution of program

```
const float golden_ratio = 1.618;
const int days_in_week = 7;
```

- You can have a constant version of any of the standard C variable types
- Enums: a group of related integer constants used to parameterize libraries:

```
enum cardsuit {CLUBS,DIAMONDS,HEARTS,SPADES};
```


Question: Which statement is TRUE regarding C and Java?

- ☐ `short`, `int`, and `long` are in both languages and they have the same meaning
- ☐ As Java was derived from C, it has the same names of data types
- ☐ C programs use compilers to produce executable code but Java does not
- ☐ C has a preprocessor that allows conditional compilation, but Java does not

Administrivia

- CS61c is relentless!
  - Next week: Lab #2, HW #2
  - Lab #2, Amazon EC2
  - HW #2 will soon be posted
- Monday is Labor Day Holiday – no lecture!
- Wonderful to see the valuable discussion and help going on in Piazza!

Agenda

- Review
- Compile vs. Interpret
- Scheme vs. Java vs. C
- Administrivia
- Quick Start Introduction to C
- Technology Break
- More C
- Summary

A First C Program: Hello World

Original C:

```
main()
{
  printf("\nHello World\n");
}
```

ANSI Standard C:

```
#include <stdio.h>

int main(void)
{
  printf("\nHello World\n");
  return (0);
}
```


C Syntax: `main`

- When C program starts,
  - 1st runs initialization code to set up process for your program
  - Then calls your procedure named `main()`
- To get arguments to the main function, use:
  - `int main (int argc, char *argv[])`
- What does this mean?
  - `argc` contains the number of strings on the command line (the executable counts as one, plus one for each argument). Here `argc` is 2:
    unix% sort myFile
  - `argv` is a *pointer* to an array containing the arguments as strings (more on *pointers* later)

Example

- `foo hello 87`
- `argc = 3 /* number arguments */`
- `argv[0] = "foo",`
  `argv[1] = "hello",`
  `argv[2] = "87"`
  - Array of pointers to strings (cover later)

A Second C Program: Compute Table of Sines

```
#include <stdio.h>
#include <math.h>

int main(void)
{
    int    angle_degree;
    double angle_radian, pi, value;
    /* Print a header */
    printf("\nCompute a table of the
   sine function\n\n");

    /* obtain pi once for all       */
    /* or just use pi = M_PI, where */
    /* M_PI is defined in math.h    */
    pi = 4.0*atan(1.0);
    printf("Value of PI = %f \n\n",
   pi);

    printf("angle     Sine \n");

    angle_degree = 0;
    /* initial angle value */
    /* scan over angle     */
    while (angle_degree <= 360)
    /* loop until angle_degree > 360 */
       {
          angle_radian = pi*angle_degree/180.0;
          value = sin(angle_radian);
          printf (" %3d      %f \n ",
                    angle_degree, value);
          angle_degree = angle_degree + 10;
          /* increment the loop index */
       }
}
```


Second C Program Sample Output

```
Compute a table of the sine
   function

Value of PI = 3.141593

angle      Sine
   0      0.000000
  10      0.173648
  20      0.342020
  30      0.500000
  40      0.642788
  50      0.766044
  60      0.866025
  70      0.939693
  80      0.984808
  90      1.000000
 100      0.984808
 110      0.939693
 120      0.866025
 130      0.766044
 140      0.642788
 150      0.500000
 160      0.342020
 170      0.173648
 180      0.000000
 190     -0.173648
 200     -0.342020
 210     -0.500000
 220     -0.642788
 230     -0.766044
 240     -0.866025
 250     -0.939693
 260     -0.984808
 270     -1.000000
 280     -0.984808
 290     -0.939693
 300     -0.866025
 310     -0.766044
 320     -0.642788
 330     -0.500000
 340     -0.342020
 350     -0.173648
 360     -0.000000
```


C Syntax: Variable Declarations

- Similar to Java, but with a few minor but important differences
- *All* variable declarations must appear before they are used (e.g., at the beginning of the block)
- A variable may be initialized in its declaration; if not, it holds garbage!
- Examples of declarations:
  - Correct:
    ```
    {
      int a = 0, b = 10;
      ...
    ```
  - Incorrect:
    ```
      for (int i = 0; i < 10; i++)
    }
    ```

C Syntax : Control Flow (1/2)

- Within a function, remarkably close to Java constructs (shows Java's legacy) in terms of control flow
  - **if-else**
    - `if (expression) statement`
    - `if (expression) statement1 else statement2`
  - **while**
    - `while (expression) statement`
    - `do statement while (expression);`

C Syntax : Control Flow (2/2)

- **for**
  - `for (initialize; check; update) statement`
- **switch**
  - `switch (expression){ case const1: statements case const2: statements default: statements }`
  - **break**

C Syntax: True or False

- What evaluates to FALSE in C?
  - 0 (integer)
  - NULL (a special kind of *pointer*: more on this later)
  - *No explicit Boolean type*
- What evaluates to TRUE in C?
  - Anything that isn't false is true
  - Same idea as in Python: only 0s or empty sequences are false, anything else is true!

C and Java operators nearly identical

- arithmetic: +, -, *, /, %
- assignment: =
- augmented assignment: +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=
- bitwise logic: ~, &, |, ^
- bitwise shifts: <<, >>
- boolean logic: !, &&, ||
- equality testing: ==, !=
- subexpression grouping: ( )
- order relations: <, <=, >, >=
- increment and decrement: ++ and --
- member selection: ., ->
- conditional evaluation: ? :

And In Conclusion, …

- All data C is an efficient compiled language, widely used for systems programming (operating systems) and application development
- C successors (C++, Objective-C, C#) are also popular
- Java syntax based on C, due to C's popularity
- BUT C can be more difficult to use than more modern programming languages