High Performance Computing in C/C++
The C Abstract Machine Model

David Chisnall

February 14, 2011

Course Aims

- Introduce C programming
- Describe the differences between C, C++, and Java
- Explain programming styles appropriate to HPC

Why Should You Care About... C?

- Widely used in industry
- Many other languages inherit syntax and semantics
- 40 years of legacy code...

Why Should You Care About... HPC?

High-performance computing is a small (well-paid) niche, but 'High performance' is relative.

Machine	Performance	Architecture	Note
Cray-1	250MFLOPS	SIMD	First supercomputer
IBM XT	<1MFLOP	SISD	First PC
Cortex A8	10MFLOPS	SISD + SIMD	Smartphone CPU
nVidia Fermi	1GFLOPS	SIMD + MIMD	Modern GPU

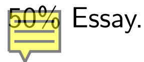
Many 'HPC' techniques eventually become useful elsewhere.

Assessment

50% Programming:

- 1 easy assignment: 15%
- 1 moderate assignment: 25%
- 1 hard assignment: 10%

50% Essay.



Why was C Created?

- UNIX was originally written in assembly language
- Porting it to a new architecture meant rewriting it
- C was created to be slightly more abstract than assembly, but almost as fast

The C Abstract Model

- Very close to how a real machine works.
- Some simple abstractions
- 'Close to the metal'

Every C language construct maps to a very short sequence of machine instructions.

Contrast: The Java Abstract Model

Contains many things that are not in the real hardware model:

- Automatic memory management
- Bounds-checked arrays
- Objects and classes
- Dynamic method lookup
- Typed memory and reflection

Why C is Fast

- A simple compiler can produce good code from C
- Easy for the programmer to understand performance
- CPU features are usually exposed directly, via laguage extensions

C Memory Model

- Flat array of bytes
- Each byte is addressable

Contrast with Java:

- Split into objects
- Objects accessed by (opaque) reference
- Access control on object fields

Pointers Vs Java References

C/C++ Pointers	Java References
Numbers indicating an address in memory	Opaque type with only assign (copy) operation defined.
Can be created as the result of arithmetic	Can only be created from valid objects.
Can point to invalid locations	Can only point to valid objects or `NULL`.
Point to memory. User must know type of destination.	Point to objects which know their own type.

Java implements a variant of Smalltalk's *object memory model*. C exposes a *flat memory model*.

No Garbage Collection

Memory must be explicitly allocated and deallocated. Potential problems include:

- Memory leaks (don't free after last use)
- Dangling pointers (use after free)
- Buffer overflows (access out of bounds)

Code is Data

1. C code is compiled to machine code
2. Machine code is loaded into memory
3. Everything in memory is accessible to C code
4. Pointers can point to functions as well as data

Example Memory Layout

Address	Region	Address
0xffffffff	stack Memory	0xffffffff
	Heap Memory	0xf0000000
	Library Code	0x00140000
		0x00100000
	Static Data	0x000fe000
	Program Code	0x0000f000
0x00000000		0x00001000

C Pointers

- Contain a memory address
- Can point anywhere in memory
- Do *not* store the type of the data pointed to

Example: Casting

Listing 1: Incorrect Casting in Java

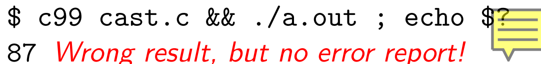
```java
class Cast
{
    public static void main(String[] args)
    {
        String str = "12345";
        Object obj = str;
        Number num = (Number)obj;
    }
}
```

```
$ javac Cast.java && java Cast
Exception in thread "main"
java.lang.ClassCastException:
java.lang.String cannot be cast to java.lang.Number
at Cast.main(Cast.java:7)
```

Example: Casting

Listing 2: Incorrect Casting in C

```c
int main(void)
{
    int a = 12;
    int *ptr = &a;
    float *ptr2 = (float*)ptr;
    *ptr2 = 1123e12;
    return a;
}
```



```
$ c99 cast.c && ./a.out ; echo $?
87
```

Wrong result, but no error report!

Example: Pointers are Numbers

```
#include <stdio.h>

int main(void)
{
    int a = 12;
    int b = (int)&a;
    printf("a: 0x%x, b: 0x%x\n", a, b);
    return 0;
}
```

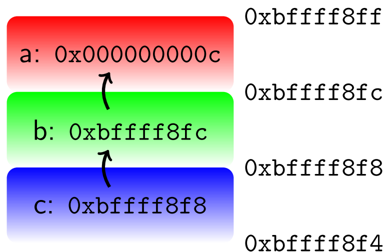
```
$ c99 ptr.c && ./a.out
a: 0xc, b: 0xbffff8fc
```

Pointers are Low Level

- Machine memory is indexed by a numerical address
- CPUs have load and store instructions for accessing memory
- This abstraction is exposed in C

Pointer Example

```
// Integer
int a = 12;
// Pointer to integer
int *b = &a;
// Pointer to pointer
   to integer
int **c = &b;
```



What Does `int a` Really Mean?

- Allocate enough space for an integer (2, 4, or 8 bytes, depending on the platform)
- Keep track of where this memory is
- Replace references to `a` with this address

`a = 12` means store the value `12` in memory at the address reserved for the variable `a`.

What Are Pointers For?

- Connecting data structures
- Aliasing data (two pointers to the same data)
- Anything you use object references for in Java