Great Reality #1:
Ints are not integers, Floats are not Reals.

40000 × 40000 → 1600000000

(gdb) ← (tool in ubuntu)

Computer Arithmetic

(1) Does not generate random values;
Arithmetic operations have important mathematical properties.

(2) Cannot assume all "usual" mathematical properties
(1) Due to finiteness of representations
(2) Integer operations satisfy "ring" properties
commutativity, associativity, distributivity.
(3) Floating point operations satisfy "ordering" properties

(3) Observation:
(1) Need to understand which abstractions apply in which contexts
Important issues for compiler writers and serious
application programmers.

(17:21)

Great Reality #2:

You've Got to Know Assembly:

① Chances are, you'll never write programs in assembly.
Compilers are much better & more patient than you are.
But: Understanding assembly is key to machine-level execution model.
(1) Behavior of programs in presence of bugs.
High-level language models break down.
(2) Tuning program performance.
(1) Understand optimizations done/not done by the compiler.
Understanding sources of program inefficiency.
(3) Implementing system software.
(1) Compiler has machine code as target.
(2) Operating system must manage process state.
(4) Creating/fighting malware:
x86 assembly is the language of choice.)

Great Reality #3: Memory Matters.

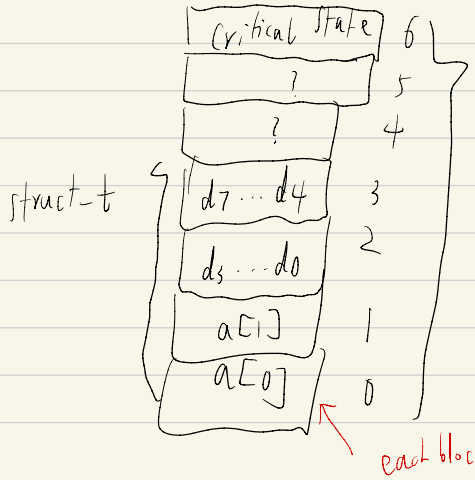
Random Access Memory Is an Unphysical abstraction.

Memory Is not unbounded
1. It must be allocated and managed.
2. Many applications are memory dominated.

Memory referencing bugs esp. pernicious.
Effects are distant in both time and space.

Memory performance is not Uniform.
Cache and virtual memory effects can greatly affect program performance.

Adapting program to characteristics of memory system can lead to major speed improvements.



Location accessed by fun(i).

It doesn't do any bound checking so a race it will happily let you reference.

(2 min)

Memory Referencing Errors

C and C++ do not provide any memory protection:

- Out of bounds array references.
- Invalid pointer values
- Abuses of malloc/free.

Can lead to nasty bugs.

(1) Whether or not bug has any effect depends on system and compiler.

(2) Action at a distance

Corrupted object logically unrelated to one being accessed

Effect of bug may be first observed long after it is generated.

How can I deal with this?

(1) Program in Java, Ruby, Python, ML, ...

(2) Understand what possible interactions may occur.

(3) Use or develop tools to detect referencing errors (e.g. Valgrind)

Great Reality #4: There's more to performance than asymptotic complexity.

① Constant factor matter too!

② And even exact op count does not predict performance.

easily see 10:1 performance range depending on how code written

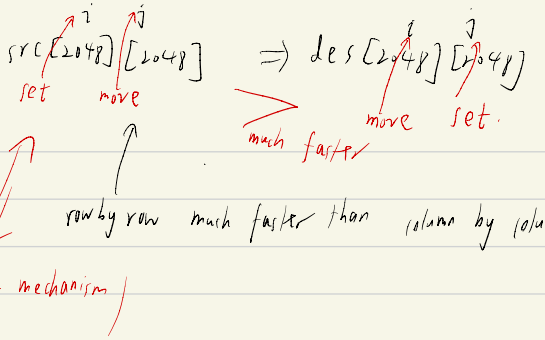
Must optimize at multiple levels: algorithm, data representations, procedures and loops.

Must understand system to optimize performance.

① How programs compiled and executed

② How to measure program performance and identify bottlenecks

③ How to improve performance without destroying code modularity and generality


src[2048][2048] ⇒ des[2048][2048]
i
j
set
move
much faster
move
set.
relation.
row by row much faster than column by column.
(cache mechanism)

Great Reality #3:

Computers do more than execute programs

They need to get data in and out.

I/O system critical to program reliability and performance.

They communicate with each other over networks.

Many system-level issues arise in presence of network.

Concurrent operations by autonomous processes,

Coping with unreliable media,

Cross platform compatibility.

Complex performance issues.

Course perrpective:

(-) Most Systems courses are Builder-Centric.

(1) Computer Architecture

(1)(i) Design pipelined processor in Verilog (a hardware description language)

(2) Operating Systems.

Implement sample portions of operating system.

(3) Compilers.

write compiler for simple language.

(4) Net working:

Implement and simulate network protocols.

Glossary:

instruction pipelining is a technique for implementing instruction_level parallelism within a single processor.

Instruction-level parallelism (ILP) is a measure of how many of the instructions in a computer program can be executed simultaneously.