A MODERN MULTI-CORE PROCESSOR

× Forms of ‖ism & understanding latency + bandwidth

1. Quick review

  × single instruction stream = single list of instr. to execute
  × superscalar execution is max 4x.
  × speedup blocked by communication, shared resources, work distribution (load balancing).

2. Overview

  × two concepts about ‖ execution
  × two challenges of accessing memory

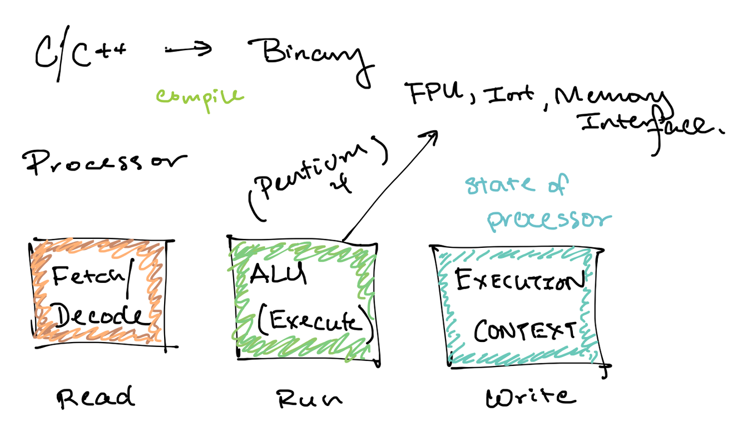
3. Parallel Execution

  × compute sin(x), for vector x, via Taylor expansion.

3.1 Compile

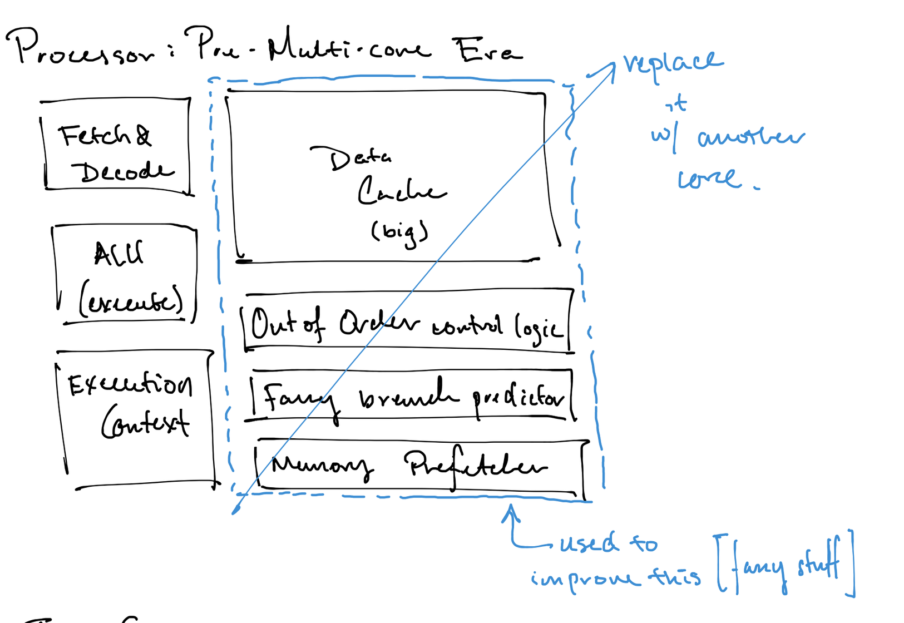
C/C++ → Binary (compile)

3.2 Processor

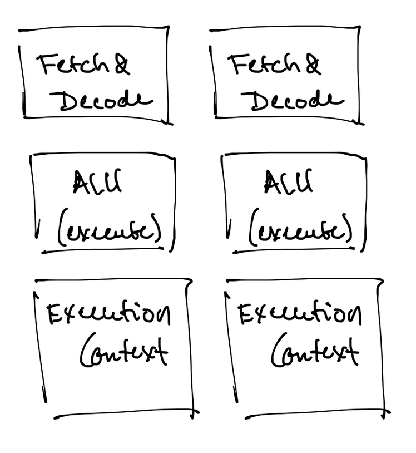


× instruction stream ⟺ thread. (ILP)
  ↳ not how many ALUs you have.

3.3 Processor: Pre-Multi-core Era



3.4 Two Cores



simpler cores = slower (25%).

How can we use our second core?

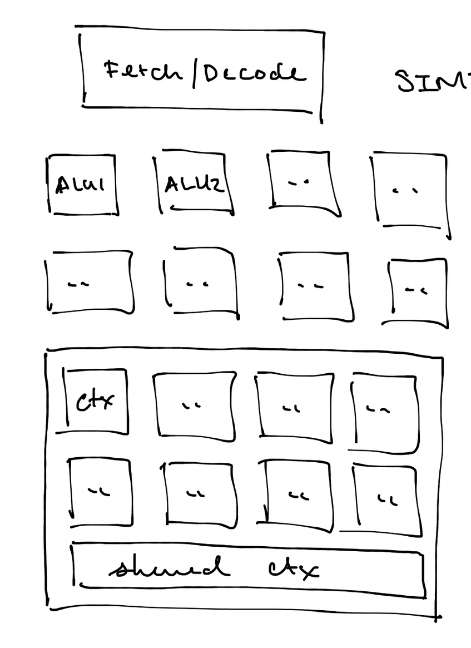
3.4.1 Expressing ‖ism using pthreads.

  × Use parent to compute ½ N.
  × Use pthread to compute second ½ N.

3.5 Data-parallel execution

  × "forall" ⟹ all iterations are data-independent
    - launch a pthread per core.
  × this is better b/c it is [# core independent]
  × also tells that all iterations do EXACTLY the same thing

3.6 SIMD Processing



REGISTERS:
  before: 32 bits wide.
  now: 8 × 32 bits wide.

SSE, AVX intrinsics

Scalar program → Vector program ← still a single instruction stream, but each instr. does 8 things at once.

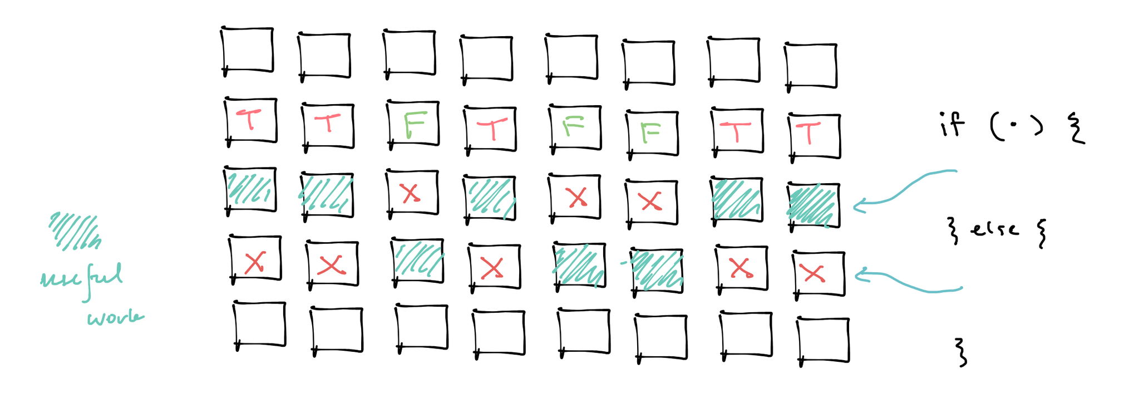
use same instruction on 8 bytes of data at a time

3.7 16 SIMD cores: 128 elts in ‖.
  (16 cores) × (8 data per time) = 128.

3.8 What about conditional execution?

Within a core, all the instr. need to be the same.
but diff cores can run different instr. streams.

Continue through, but don't execute certain ALUs in the SIMD core (or mask out unwanted).



worst case: 1/8 peak performance:

```
if (only 1 true) {
   // DO A LOT
} else {
   // DO A LITTLE
}.
```

× instruction stream coherence (ISC)
  × same instr. sequence applies to all elts operated on simultaneously
  × necessary for efficient use of SIMD.
  × NOT necessary for efficient parallelization across CORES
× "divergent" execution (lack of ISC)

3.9 Modern CPUs, SIMD

  × SSE 128 bit ops: 4 × 32 bits or 2 × 64 bits (4 wide float vec).
  × AVX 256 bit ops: 8 × 32 bits or 4 × 64 bits (8 wide float vec)

  × ‖ism conveyed using ‖ language semantics.
  × ‖ism explicitly requested by programmer by intrinsics.
  × def "explicit SIMD" — SIMD generated at compile time.

3.10 SIMD execution on GPUs.

  × "Implicit SIMD"
  × compiler generates scalar binary.
  × N instances of program always run together on the processor
  × GPU, SIMD width ranges from 8 to 32. (Nvidia: 32 wide)
    ⤳ divergence would mean 1/32 of peak performance!

3.11 Summary

  × several forms of ‖ execution
    - Multicore — thread-level ‖ism: different instructions, indep. instr. streams. [different programs]
    - SIMD — same instruction thread, different data — can construct "adversarial code" that limits performance. [same program]
      - lack of dependencies is known prior to execution.
      - if conditionals, want to chunk data into (SIMD width) blocks s.t. all data within a block evaluates the same way

4. Accessing Memory

4.1 Terminology

  def Memory latency. the amt of time for a memory request (e.g. load, store) from a processor to be serviced by the memory system

  def memory bandwidth. the rate at which a memory system can provide data to a processor.

$$\text{delay} = \text{latency} + \frac{\text{size of request}}{\text{bandwidth}}$$

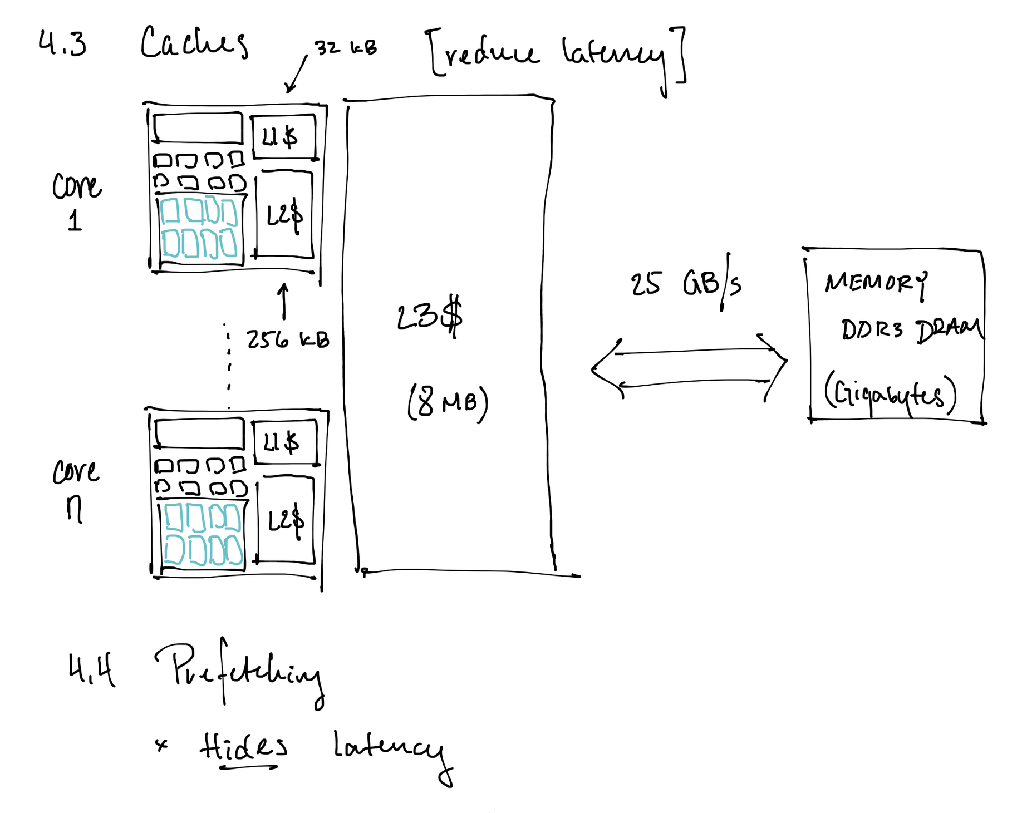
latency: cycles or [time]; size of request: a size.; bandwidth: a rate (size/sec)

4.2 Stalls

  × a processor "stalls" when the next instr. can't be run b/c of dependence on previous instruction

  × memory access time is measure of latency
  × accessing memory is major source of stalls.

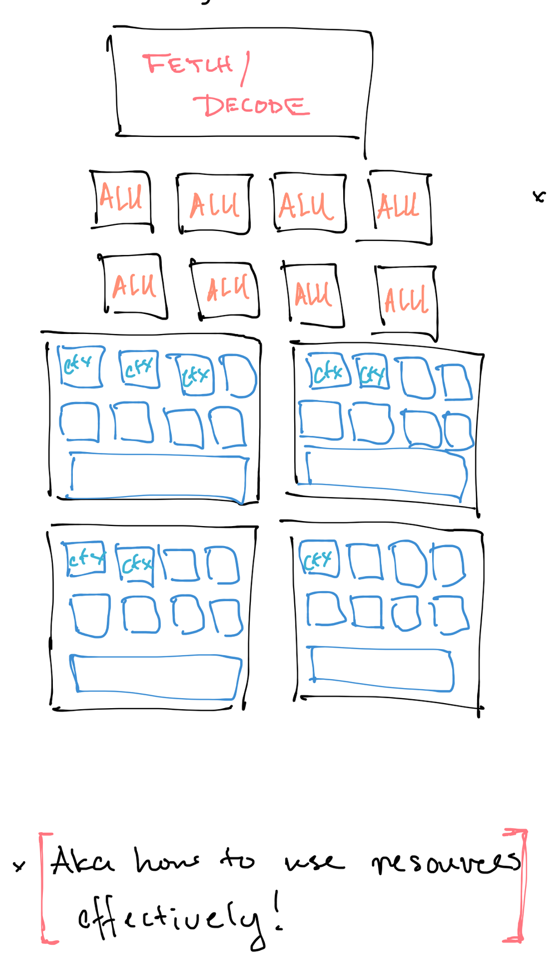
4.3 Caches [reduce latency]

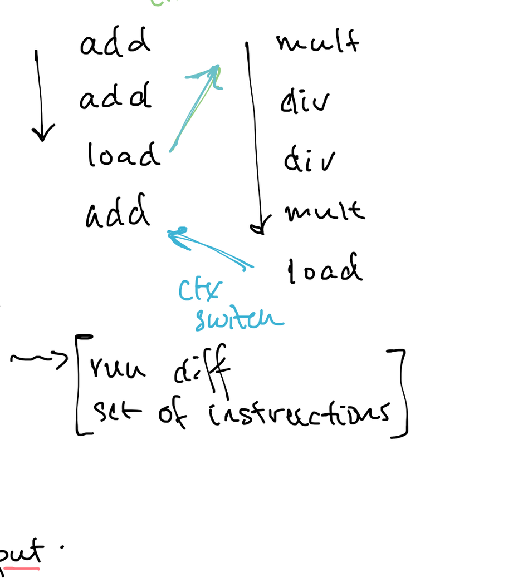


4.4 Prefetching

  × Hides latency

4.5 Hiding stalls w/ multithreading





  × whenever stall, switch to different execution context ⤳ [run diff set of instructions]
    aka a different "thread"

  × key idea of throughput-oriented systems: potentially incr. time to complete any single thread, in order to increase overall system throughput when running multiple threads.

  × [Aka how to use resources effectively!]

  × tradeoff more L1$ per thread, or more threads?

  × 4 instruction stream per core, running concurrently (not simultaneously)

    SIMULTANEOUS = # cores
    CONCURRENT = # cores × # ctxs per core

  × Nvidia GTX 480:

    15 cores, 32 wide SIMD per core, 48 execution ctxs per core.

    15 cores: 15 diff instruction streams SIMULTANEOUSLY
    32 wide SIMD: 32 per instruction
    48 execution ctxs: 48 diff inst. streams.

  × ratio of $\frac{\text{MATH}}{\text{MEMORY}}$ is important!