CS 149 // Lec 2 // A modern multicore processor

→ Instruction level parallelism

  ↳ superscalar execution (instruction level parallelism → ILP)

    ↑ automatically finds independent instructions

→ reading values from DRAM destroys it.

→ cache : data cache & instruction cache

  ↑ in the processor

  → replacement policies include LRU

  → hit

  miss → cold miss
       → capacity miss
       → conflict miss

  → reduces latency of memory access

    ↑ also lower energy for lower level of cache

→ parallel processor

  ↳ duplicated with multiple instruction streams

    ↳ maybe 25% slower per "core" but overall it is ← $0.75 \times 2 = \underline{\underline{1.5x}}$ faster!

    ↑ that is a speedup

→ without changing the instruction stream, the execution on the new processors will be 25% slower

→ a thread is an instruction stream (each one)

→ number of threads doesn't matter, we can get as many as we can in the hardware

→ nvidia calls cores ← SMs (shared multiprocessors)
  ↳ apple has heterogenous cores (big + little)

→ data parallelism
  ↳ all threads do the same thing in terms of instructions
    ↳ increase the number of ALUs and registers

→ vector program ← using AVX instructions
  ↳ uses the multitude of ALUs
  ↳ can use a single instruction stream
    ↳ and then multiple processors can have multiple instruction streams.

→ SIMD ← single instruction, multiple data

→ ALUs are very cheap, FPUs are not as much

→ what about conditionals ?
  ↳ mask outputs of unneeded vector bits
    ↳ this is wasteful, we could get up to 1/8th performance
  ↳ instruction stream coherence v/s divergent

→ SIMD in GPUs
↳ implicit SIMD

at fetch/decode level

→ Parallel execution
↳ superscalar (ILP) ←
- branch predictions.
- out of order exec.
- memory pre-fetching.

↳ H/W
↳ no change in instructions

at ALU level

SIMD ← vector operations (data parallelism)

multi-core ← thread parallelism
↳ diff instruction streams for each processor

at processor core level.