3 PARALLEL PROGRAMMING ABSTRACTIONS

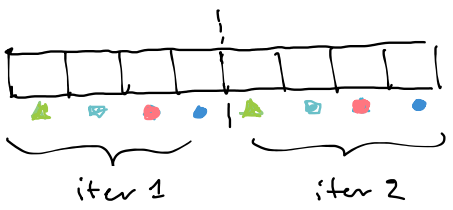
1. Review

* pthread # agnostic
* SIMD ; multiple ALUs for same instructions
  > implicit SIMD = scalar program deployed to all ALUs
* multiple execution ctx hides latency (always use CPU)
  ↳ the core is running two threads concurrently, but not simultaneously.
* thread = instruction stream = "a program"

2. Abstractions vs. Implementation

2.1 Programming w/ ISPC

* SPMD programming abstraction (ABSN).
  - call to func = spawn gang of ISPC program instances.
* sinx.ispc
  → runs sequential logic, every 8th datapoint in the array
* ISPC will implement by generating SIMD instructions.
* "interleaved" is better b/c cache — the data that is loaded is close together per iteration.



* ISPC ← for-each abstraction
  * assign iterations to program instances in the gang
  * ∥ loop iterations.
  * gang should cooperatively perform the loop iterations.

  → implementation SIMD instr. on a single thread.

  → foreach ≐ the loop iterations should be independent!

3. Parallel Programming Models [Abstractions]

* abstraction: a "thread"
* implementation: pthread_create() library implementation

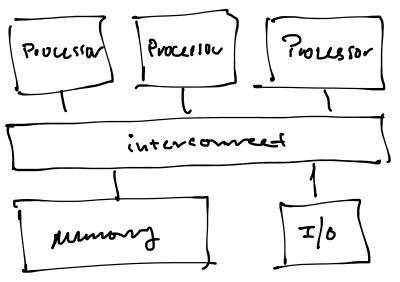
3.1 Models of Communication

* Shared Address Space
* Message Passing
* Data parallel

3.2 Shared Address Space Model of Computation

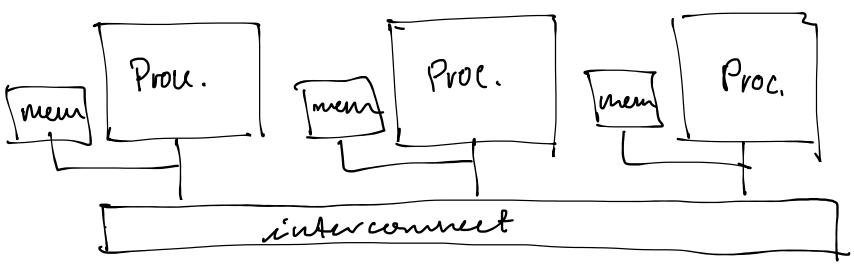
* communicate via "global" variables.
* synchronization is also shared variables.

– HW implementation



"dancehall"
or crossbar, bus, etc.

– HW implementation: Non-Uniform Mem Access (NUMA)



→ higher cost for greater distance

– "FAT TREE" topology.

3.3 Message Passing Model

* threads have private addr. space.
* explicit "SEND" & "RECV" messages.
* msgs are only way to comm. between threads.
* popular library: MPI [MESSAGE PASSING INTERFACE]
  > commodity clusters.