Parallel computing platforms

Jeremy Iverson

College of Saint Benedict & Saint John's University

recap

- von Neumann architecture
  - central processing unit
  - memory
    - cache ($)
  - interconnection
- operating system
  - processes vs threads

- `lscpu`
- `cat /proc/cpuinfo` activity for CPU info
- `cat /sys/devices/system/cpu/cpu0/cache/` for cache info
- L1, L2, L3
  - L1 is usually split into data and instruction
  - L2 is usually not shared, i.e., each core has its own
  - L3 is usually shared by multiple cores
- multitasking operating system, runs many processes despite having only one or a “small” number of physical cores
  - context switches the processes at predefined time slices
  - `top` activity
- generally speaking, processes have their own address space, threads share an address space

cache performance

From Intel Performance Analysis Guide:

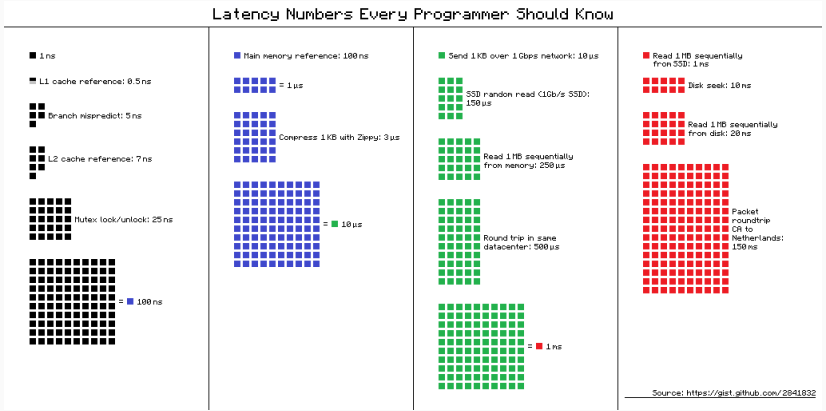
```
Core i7 Xeon 5500 Series Data Source Latency (approximate)              [Pg. 22]

local  L1 CACHE hit,                              ~4 cycles (   2.1 -  1.2 ns )
local  L2 CACHE hit,                             ~10 cycles (   5.3 -  3.0 ns )
local  L3 CACHE hit, line unshared               ~40 cycles (  21.4 - 12.0 ns )
local  L3 CACHE hit, shared line in another core ~65 cycles (  34.8 - 19.5 ns )
local  L3 CACHE hit, modified in another core    ~75 cycles (  40.2 - 22.5 ns )

remote L3 CACHE (Ref: Fig.1 [Pg. 5])       ~100-300 cycles ( 160.7 - 30.0 ns )

local  DRAM                                                    ~60 ns
remote DRAM                                                   ~100 ns
```

cache performance



Latency Numbers Every Programmer Should Know

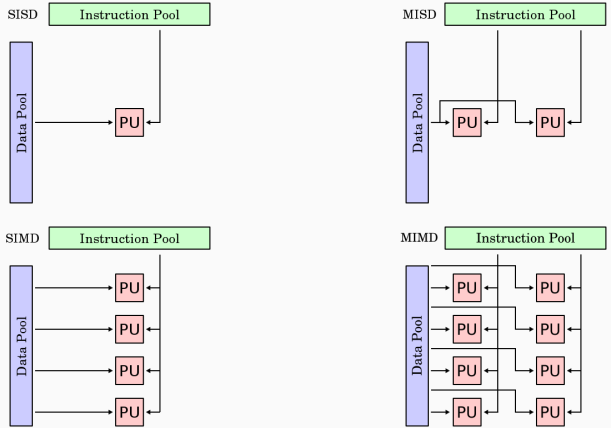
parallel computing platform

- logical organization
  - the user's view of the machine as it is being presented via its system software
- physical organization
  - the actual hardware architecture

- we need some vocabulary to be able to compare different types of parallel systems
  - we can describe a system based on its logical properties or its physical properties
- physical architecture is largely independent of the logical architecture
  - this slide deck deals with logical

flynn's taxonomy

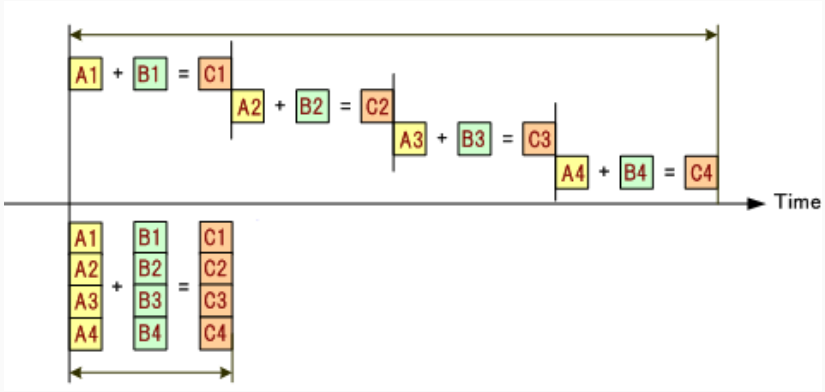
- based on the number of instruction streams and data streams available in the architecture



Flynn's taxonomy by Cburnett / CC BY 3.0 / presenting the four together

- ask students to give examples of each
  - SISD
    - serial computer
  - SIMD
    - gpu
    - some modern cpus have simd extensions
  - MISD
    - fault tolerance
  - MIMD (in many cases, this is further divided into a category commonly known as SPMD, single program multiple data)
    - most common type of parallel computer (multi-thread or multi-node)
- which of these might be interesting to us, i.e., which might make good parallel systems?

simd

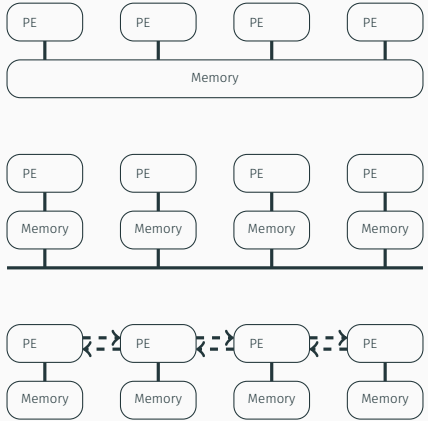


SIMD / cropped from original

- cuda / opencl are languages to express this type of parallelism, but we will not be studying them
- what sorts of problems might this be good for?
  - matrix-oriented scientific computing
  - media-oriented image and sound processors
- simd is simpler and more energy efficient than mimd, why?
  - only needs to fetch one instruction per “cycle”
- what would the mimd time line look like?
- so then what are the (dis)advantages of each of the classifications?
  - simd works in lock stop, so does not require synchronization, which can make it easier to reason about
  - mimd processing elements are autonomous, so can solve more complex problems

communication models

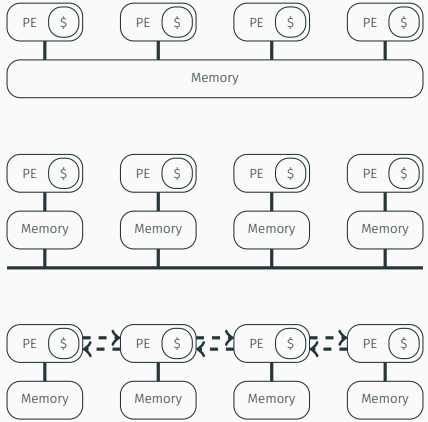
- shared-address space
  - UMA / NUMA / ccNUMA
- message-passing



- how do processing elements communicate with each other
  - shared-address space
    - everyone has access to everyone else's data
    - communicate through shared memory
  - message-passing
    - communicate through messages sent between PEs
    - build on two basic operations, *send* and *receive*
- why not just make everything uma?
- is there anything missing from this model?
  - what is a cache for?
- the existence of caches means that data can exist in multiple places
  - a cache coherence protocol is required to ensure proper semantics and correct program execution
  - have students think of a sequence of operations that would be incorrect without cache coherence

communication models

- shared-address space
  - UMA / NUMA / ccNUMA
- message-passing



- how do processing elements communicate with each other
  - shared-address space
    - everyone has access to everyone else's data
    - communicate through shared memory
  - message-passing
    - communicate through messages sent between PEs
    - build on two basic operations, *send* and *receive*
- why not just make everything uma?
- is there anything missing from this model?
  - what is a cache for?
- the existence of caches means that data can exist in multiple places
  - a cache coherence protocol is required to ensure proper semantics and correct program execution
  - have students think of a sequence of operations that would be incorrect without cache coherence

cache coherence

- update
  - increases communication on the bus
- invalidate
  - increases idling time

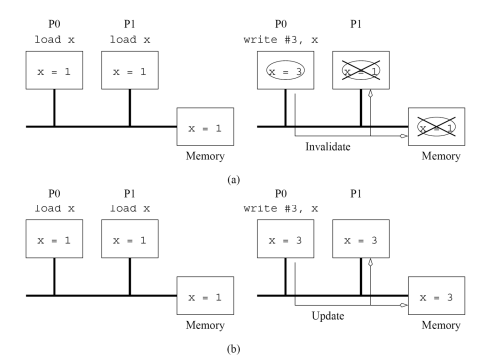
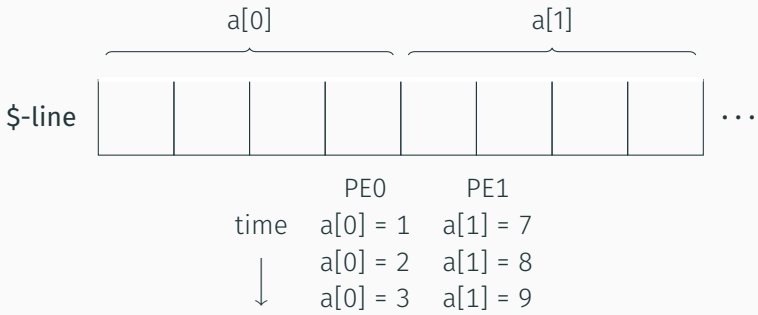


Figure 2.21 Cache coherence in multiprocessor systems: (a) Invalidate protocol; (b) Update protocol for shared variables.

- why is cache coherence important?
  - maintains serializability — just like ILP
- considerations for each of the two protocols
  - update increases communication because a message is sent every time that a PE updates a value that it has in cache — contrast this with invalidate which sends a single invalidate after the first update, but no subsequent messages
  - invalidate increases idle time because whenever a PE accesses a value that has been invalidated, it must wait while the updated value is fetched — contrast this with update, where the updated value would have been sent at the time of the update
- which do you think most processors use?
  - invalidate

false sharing



- a few more notes about caches
  - data is fetched from memory at the granularity of a cache line
  - cache lines are fixed size (typically 64 bytes)

- consider this example
  - what would happen if we were using invalidate protocol?
  - remember, the cache line is copied into the cache of PE0 *and* PE1
  - how can this be avoided?