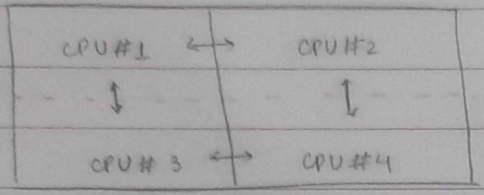
day / date:

What is parallel and distributed computing?

- use of multiple processors or machines working together on a common task.
- each processor/machine works on its section of the problem, they may exchange information.



* CPU must exchange data at their shared boundaries to maintain consistency.
* reduces computational time

- parallelism provides solving large problems in same time
- grain of parallelism: bits, instructions, blocks, loop, iteration, procedures.
- examples of parallel machines
  - CMP: multiple processor coves are integrated onto a single chip, these coves work together to execute tasks in parallel.
- a distributed computer contains multiple machines combined together with a network such as cluster, grid, cloud.

Why do parallel computing?

1. Technology Push
2. Application Pull

1. single processor system → more number of transistors → size got smaller → faster processor → more performance.

MOORE'S LAW: number of transistors doubles approximately after 2 years leading to more parallelism

implicit parallelism: managed by system, hidden from programmer, automatically detects and executes task in parallel without requiring explicit instructions.

explicit parallelism: programmer defines how task should be divided and executed in parallel, more effort to manage synchronization, workload and data dependencies. use of threads, message passing, shared memory.

Problems with single faster CPU

1. power consumption
   - with growing clock-frequencies, power consumption increases exponentially.
   - could have required a whole nuclear vector to fulfil energy needs.

day / date:

2. Heat Dissipation

↳ with growing clock frequency, more power is consumed → more heat dissipated → requires better cooling solution

3. Smaller size of transistor

↳ fabrication of processors become more difficult.

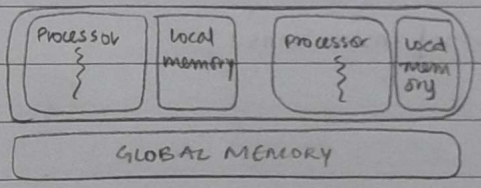
4. Limited memory size

↳ single processor has limited internal memory, main memory.

Multi-Core Era

NEW MOORE LAW: Computers no longer get faster, just wider

- data parallel computing is most scalable solution
- handful of processors, each supporting 1 hardware threads. → lots of on-chip memory near the processors for local computation. → shared global memory space which the processors can communicate through



* multicore

- many processor each supporting many hardware threads.
- each processor is paired with a on-chip memory, but per thread memory resources are smaller.
- serves as a convenient venue for thread communication.
- shared global memory space serves as a forum for inter-core communication (threads in different processes)

2. refers to the process of requesting, retrieving or pulling an application from a source.

day / date:

Parallelizing Applications

1. Traditional
   - written for serial computation
   - runs on a single computer with single CPU
   - problem is broken into a discrete series of instructions that are executed one after another.
   - slower processing time for large datasets
   - memory management is simpler, as no-coordination access between threads.

   example: reading a file line by line, and processing result back.

2. Parallel
   - simultaneous use of multiple compute resources to solve a computational problem.
   - run using multiple CPU on same or different machine.
   - problem broken in discrete part that can be solved concurrently.
   - each part further broken down to a series of instructions.

Parallelization strategy

1. Problem Understanding
   - understand the existing code.
   - whether or not problem is suitable for parallelization?
   - if tasks are highly dependent on each other, parallelization may be challenging.
   - code with high dependency, communication and synchronization decreases suitability to parallelize it.

   (a). Identify hotspots
   - check where most of the work is done and ignore those section of the program that account for little CPU usage.
   - input/output slows down the program.
   - solution: restructure, use different algorithm, overlapping communication with computation meaning while one part is performing computation, another part can handle sending or receiving data, hence overall execution time reduces.

day / date:

2. Partitioning and Decomposition

- breaking down a large, complex problem into smaller, manageable sub problems that can be solved concurrently.

a) Domain decomposition

- dividing data into smaller chunks, each computational task operates on a specific portion of the data
- block decomposition
  - ↳ dataset divided into equal-sized (large) blocks, and each assigned to a processing unit.
- cyclic decomposition
  - ↳ data distributed in a round robin fashion to balance workload.

b) Functional decomposition

- divides a problem based on different functions or operations that are to be performed.
- each task may operate on the same data but perform a different function.

3. Assignment

- refers to allocating tasks, sub-tasks or data chunks to available processing unit.
- ensure efficient use of resources and avoid bottlenecks.
- assigning tasks in a way that balances the load and minimizes communication overhead.
- ensuring each processor is assigned a fair share of the work.

4-5. Orchestration and Mapping

- refers to the management of tasks and resources during the execution of parallelized application.
- involves co-ordinating when and how tasks are executed.

a) communication

- ↳ some problems can be decomposed without any communication.
- ↳ these problems are called "Embarrasingly Parallel".
- ↳ some parallel application require tasks to share data with each other.
- ↳ CPU time and resources are used to package and transmit data.
- ↳ synchronization between tasks result in waiting time.
- ↳ latency is the time it takes to send a message from A to B.
- ↳ bandwidth is the amount of data that can be communicated per unit time
- ↳ efficient approach is to package small messages into larger ones to maximize bandwith usage and reduce overhead.

day / date:

↳ distributed memory ; communication is explicit

↳ shared memory mode ; communication is implicit.

↳ synchronous communication ; handshaking , other task must wait until the communications have completed.

↳ asynchronous communication ; allow task to transfer data independently from one another, non-blocking

↳ scope of communication

(a) point to point : involves two task , sender and receiver

(b) collective : data sharing between more than two tasks , which are in a common group.

(b) synchronization

↳ co-ordination and sequencing of parallel tasks.

↳ requires serialization of program.

(a) barrier

- all tasks are involved
- either stops or blocks
- all tasks resume , when last task reaches barrier.

(b) lock / semaphore

- involves number of tasks
- only one task at a time may use own lock / semaphore
- others wait for lock to be released.

(c) synchronous communication operation

- involves those task executing a communication operation.
- before initiating communication , acknowledgement must be received.

day / date:

Flynn's Taxonomy

1. Single Instruction, Single Data
   - execute one instruction at a time
   - uniprocessor
2. Single Instruction, Multiple Data
   - parallel processor
   - single instruction → all processing unit ~~can~~ execute same instruction at any given clock cycle
   - multiple data → each processing unit can operate on a different data element.
   - speeds up matrix operations
   - without SIMD, CPU adds one pair of number at a time; with SIMD, CPU adds multiple pairs at the same time.
3. Multiple Instruction, Single Data
   - sequence of data
   - each processor executes different instruction sequence to same data at the same time.
   - it is rare but valuable in systems that require redundancy and error checking.
4. Multiple Instruction, Multiple Data
   - different processors to work independently on different tasks.
   - true parallel computing
   - highly scalable
   - multi-cores, clusters, grid, cloud.
   - classified as shared memory or distributed memory.
     - ↓ SMP and NUMA
     - ↓ cluster, grid, cloud computing.

Symmetric Multiprocessor (SMP)

- where multiprocessors share the same memory and operating system.
- each processor has equal access to system resources.
- improves performance → multiple processors work together on tasks.
- shared memory → all processors access same RAM.
- scalability → increases computing power when processors added.
- instruction stream divided into smaller streams called "threads"
- has practical limit to number of processors

Non-Uniform Memory Access

- each processor has its own local memory but can also process/access memory from other processors.
- works well for high performance computing with many processors
- different processors access different region of memory at different speeds.
- NUMA retains SMP flavour while giving large scale multiprocessing.

(a) cache-coherent NUMA

- each processor has own L1 and L2 cache
- each node has own main memory.
- nodes are connected.
- cache coherence ensures that when one processor updates data, other processors see the latest version.
- L1 cache → L2 cache (local to processor)
- main memory → (local to node)
- remote memory →
- Disadvantage: performance can breakdown if too much access to remote memory

Distributed memory / message passing / Hybrid system

- each processor has its own memory.
- data transfer between processors is explicit using message passing.

supercomputers and data centres where node communicate over a network.

- hybrid models are a combination of shared memory and distributed memory models.

Cluster computing

- multiple independent nodes interconnected to act as a single system.
- communication happens via high speed networks
- no single shared memory
- high performance, high availability, scalability
- it provides multiprocessor support

day / date:

Grid computing

- connects computers worldwide to share processing power and storage
- applications executed at several locations
- uses standard protocol for distributed computing.
- roles include → user, resource broker, grid resource

★ SMP and NUMA are good for local parallelism

★ clusters and grid handle large scale distributed computing

Cloud computing

- network based computing model that provides on demand resources.
- instead of running applications locally, users access computing power over the internet
- hides complexity and detail of underlying infrastructure
- elastic scalability → resources scaled up or down

IaaS, PaaS, SaaS

★ cloud computing provides on demand scalability

SuperComputers

- fastest computers in terms of processing capacity, speed of calculation
- speed is measured in FLOPs

Performance

- how well a computer, processor, or algorithm performs
- performance metric includes: time, FLOPs, MIPs and Benchmark.
- components of execution time
  - (a) user CPU time → time executing the programs instruction
  - (b) system CPU time → time executing OS routines
  - (c) waiting time → delays caused by I/o operations or multitasking

$$\text{Execution Time} = \frac{\text{Total Instructions} \times \text{clock cycles}}{\text{clock speed (Hz)}}$$

- factors affecting computer performance
  - (a) processor speed: faster CPU cycles improve execution speed.
  - (b) data bus width: wider bus allows more data to be transferred.
  - (c) cache size: larger cache reduce memory access time.
  - (d) memory speed: faster RAM reduces memory bottlenecks
  - (e) processor cores: more cores for parallel execution.

1c) $S = 1$ → speedup factor - FLOPs (new) - 5.50×10^{15}

day / date:

Measuring Performance

- every CPU has a clock that ticks at fixed rate
- each tick represents one clock cycle and each instruction takes one or more cycles to execute

$$\text{Clock cycle time} = \frac{1}{\text{clock speed}}$$

- higher clock speeds mean faster execution

Millions of Instruction Per seconds (MIPS)

- better comparision
- affected by → clock speed, speed of the buses, speed of memory access

$$\text{MIPS} = \frac{\text{Instruction Count}}{\text{User CPU time} \times 10^6}$$

$$\text{MIPS} = \frac{\text{clock rate of processor (recycle)}}{\text{clock cycles per instruction} \times 10^6}$$

- faster processors lead to larger MIPS

Floating Point Operations Per second

$$\text{FLOPS} = \frac{\text{number of floating point}}{\text{execution time}}$$

Benchmarks

- to compare different systems
- measure performance improvement after upgrades
- (a) microbenchmarks
  - measure one performance dimension or aspect
  - cache, memory, procedure call, FP
- (b) - macrobenchmark
  - measure overall system performance

Limitations of memory system

- multicore increases computational capacity but reduces performance because of hardware capabilities and programming bottleneck.

day / date:

Amdahl's law

1. max speedup = $\frac{1}{1-p}$ or $\frac{1}{f}$

p = parallelizable section

f = serial fraction

2. $S_{latency} = \frac{1}{(1-p) + \frac{p}{s}}$

s = speedup

* when overall speedup is to be calculated.

3. speedup = $\frac{1}{\frac{p_1}{s} + \frac{p_2}{s} + \frac{p_3}{s}}$

4. speedup = $\frac{ts}{tp}$

ts = execution time on single procc.

tp = execution time using multi

5. speedup = $\frac{\text{number of computational 1 processor}}{\text{" " parallel " with p processors.}}$

6. speedup = $\frac{ts}{f\,ts + (1-f)\frac{ts}{p}}$

ts = serial time + parallel

f_s = fraction of code i.e. serial x ti

$(1-f)ts$ = parallel section x time

p = no. of processors

OR

$\frac{p}{1 + (p-1)f}$

7. speedup = $\frac{1}{fs + \frac{fp}{\text{proccessors.}}}$

fs = serial fp = parallel

8. efficiency = $\frac{\text{speedup}}{\text{no. of processors}}$

9. speedup = $\frac{\text{FLOPs (new)}}{\text{FLOPs (current)}}$

CV 10

(c) $S_{\text{latency}} = 1$ → speedup factor FLOPs (new) $5.80 \times 10^{1.5}$

day / date:

gustafsons speedup = $\frac{\text{sequential execution time}}{\text{parallel execution time}}$

speedup = $p + (1-p)f$

f = fraction of serial

p = no. of processors.

(a)

$$\frac{1}{\frac{(0.08)(172800) + (0.92)(172800)}{2048}}$$

hours = $48 \times 60 \times 60 = 172800 = 12.4$

(b) efficiency = $\frac{12.5}{2048} = 6.1 \times 10^{-3}$

communication overhead →

1000 → 5.1. → 95.1. efficiency $E = \frac{S}{P} \rightarrow 0.95 = \frac{S}{1000}$

1048 → 15.1. → 85.1. efficiency $E = \frac{S}{P} \rightarrow 0.85 = \frac{S}{1048}$ } + = 1840.8

$$S_{\text{latency}} = \frac{1}{(1-0.92) + \frac{0.92}{1840.8}} = 12.4$$

$S = \frac{ts}{tp}$ → $tp = \frac{ts}{S}$ → $tp = \frac{172800}{12.4} = \frac{13935.5}{60 \times 60} = 3.87$ hours.

(c) $40 = \frac{1}{\frac{0.08 + 0.92}{n}}$ → $40\left(0.08 + \frac{0.92}{n}\right) = 1$ → $3.2 + \frac{36.8}{n} = 1$

$\frac{36.8}{n} = -2.2$

$n = -16.7$

not possible

day / date:

(a) overhead reduces by 50 % →

~~#2~~ Exercise 4.1

P1 → clock rate = 4 GHz — 3 compilers

P2 → clock rate = 2 GHz

Class	CPI for P1	CPI for P2	C1 (%)	C2 (%)	C3 (%)
A	4	2	30		
B	6	4	50		
C	8	3	20		

$$\text{cycle time} = \frac{1}{4 \times 10^9} = 0.25 \times 10^{-10} \text{ ns}$$

$$\text{cycle time} = \frac{1}{2 \times 10^9} = 0.5 \text{ ns}$$

$$CPI_{P1} = (0.30 \times 4) + (0.50 \times 6) + (0.20 \times 8)$$

$$CPI_{P2} = (0.30 \times 2) + (0.50 \times 4) + (0.20 \times 3)$$

$$T_{P1} = \frac{I \times 5.8}{4 \times 10^9} \rightarrow \frac{\text{inst count}}{\text{user CPU time}} = \frac{\text{clock rate}}{CPI}$$

$$T_{P2} = \frac{I \times 3.2}{2 \times 10^9} \rightarrow \frac{T_{P2}}{T_{P1}} \rightarrow \frac{3.2}{2} \times \frac{4}{5.8}$$

Home Task Q1

Q1.

Class	CPI
I_1	1
I_2	2
I_3	3

X → clock rate = 2.2×10^{-9} cycle time = 0.5 ns

for A1 → exec time = $\frac{\text{inst count} \times \text{clock cycles}}{\text{clock speed}} = \frac{(5 \times 10^9 \times 1) + (1 \times 10^9 \times 2) + (1 \times 10^9 \times 3) \times 0.5 \times 10^{-9}}{1 \times 10^{10}} = 5$ seconds

A2 = $1.5 \times 10^{10} \times 0.5 = 7.5$ seconds.

(c) So = 1 → speedup factor FLOPs (new) = 5.50×10^{15}

Amdahls law Q1 day / date:

serial program → ts = 632 seconds

parallelizable = 0.95 serial = 0.05

processors = 16

$$\text{speedup} = \frac{1}{(1-P)+\frac{P}{n}} = \frac{1}{(1-0.95)+\frac{0.95}{16}} = 9.14$$

$$\text{efficiency} = \frac{9.14}{16} = 0.57 \times 100$$

Q2 $P = \frac{3}{4} = 0.75$

$$\text{speedup} = \frac{1}{(1-P)+\frac{P}{n}} \rightarrow 3.555 = \frac{1}{(1-0.75)+\frac{0.75}{n}} \rightarrow 3.555\left(0.25+\frac{0.75}{n}\right) = 1$$

$$= \frac{711}{800} + \frac{2.66625}{n} = 1$$

Q3 speedup = 7.804 processors = 32

percentage of parallelizable code?

$$\rightarrow 0.11125 = \frac{2.66625}{n} = \frac{23.96}{24 \text{ processor}}$$

$$\text{speedup} = \frac{1}{(1-P)+\frac{P}{n}} \rightarrow 7.804 = \frac{1}{(1-P)+\frac{P}{32}} \rightarrow 7.804 = \frac{32}{32(1-P)+P} \rightarrow 32-32P+P = \frac{32}{7.804}$$

$$\rightarrow -31P = -27.89$$

$$= +0.89 \times 100$$

$$= 89\%$$

- if none of the code can be parallelized P=0, and speedup = 1
- limits to scalability of parallelism
- linear speedup is usually with p processors
- to get super linear speedup (processors greater)
- by using multiprocessor, s(P) gives increase in speed.

Gustafson's law

- fixed time to do execution
- hypothetically assume that problem size will also increase as number of processors increase.
- Amdahl fixes the problem size.

Scalability

- a problem is scalable if it can handle ever increasing problem size.

(a) strongly scalable

- increasing no of processors without increasing problem size

(b) weakly scalable

- increasing no of processors with increasing problem size