Talking about…

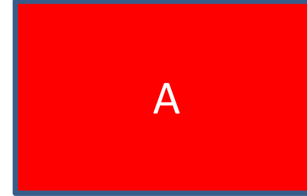
- Multi-core processors/chips
- CMP
- MPSoCs

Why parallel architectures

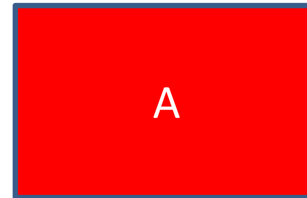
- Concurrency in applications
- Limits to frequency:
  - The memory wall
  - The ILP wall
  - the power wall
- Multicore can reduce power

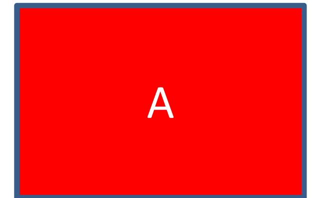
Example

- Core A
- Execution time = 10 sec
- P=10 W
- E=100 J

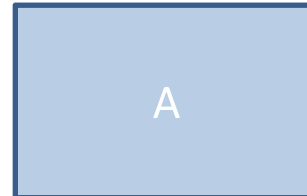


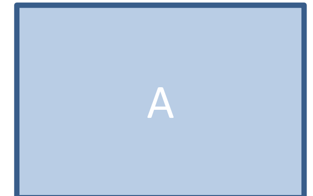
- 2 Core A
- Execution time = 5 sec
- P=2*10W = 20W
- E=100 J





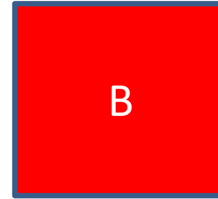
- 2 Core A
- Execution time = 10 sec
- P=2*5W = 10W (no saving)
- E=100 J



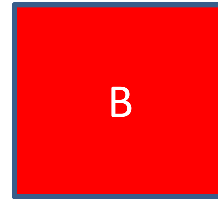


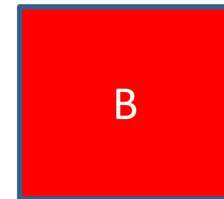
Example

- Core B
- Execution time = 15 sec
- P=5 W
- E=100 J



- 2 Core B
- Execution time = 7.5 sec
- P=2*5W = 10W
- E=75 J





Why (multiple) slower cores are power-efficient

- P_{switch} = Activity * Frequency * V^2
  rule of thumb $\propto F^2$

- Less pipeline stages are needed (lower number of flip flops)

- Synthesizing using a higher frequency constraints leads to more power hungry netlists

- In a multi-core cores can be specialized

Ahmdal's law

- **Amdahl's Law** states that potential program speedup is defined by the fraction of code (P) that can be parallelized:

$$\textbf{speedup} = 1/1 - P$$

- Introducing the number of processors performing the parallel fraction of work, the relationship can be modeled by:

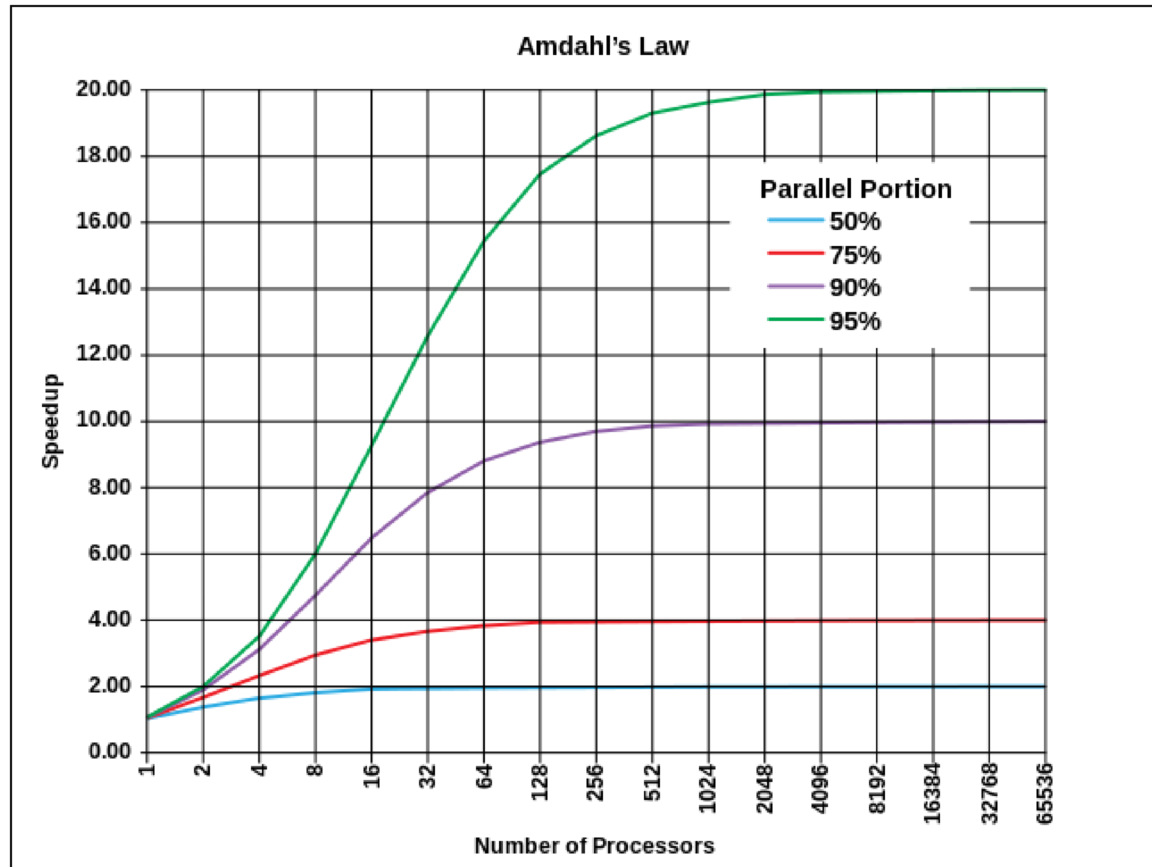
$$\textbf{speedup} = 1/ (S+ P/N)$$

where

P = parallel fraction,

N = number of processors and

S = serial fraction.



Limitation and costs

- Complexity
- Portability
- Resource requirement
- Scalability

MPSoCs, a stack view

Application

APIs

APIs

OS

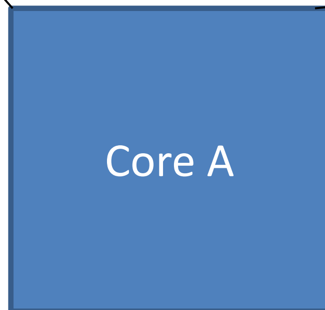
APIs

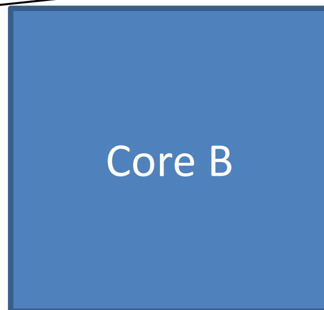
Communication

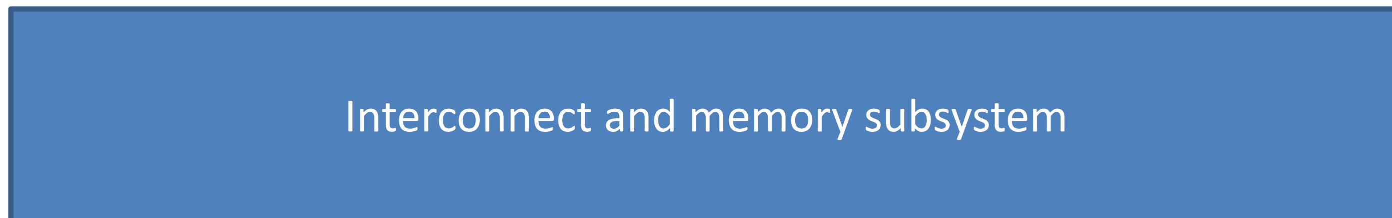
APIs

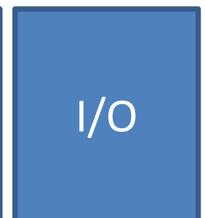
APIs

HAL



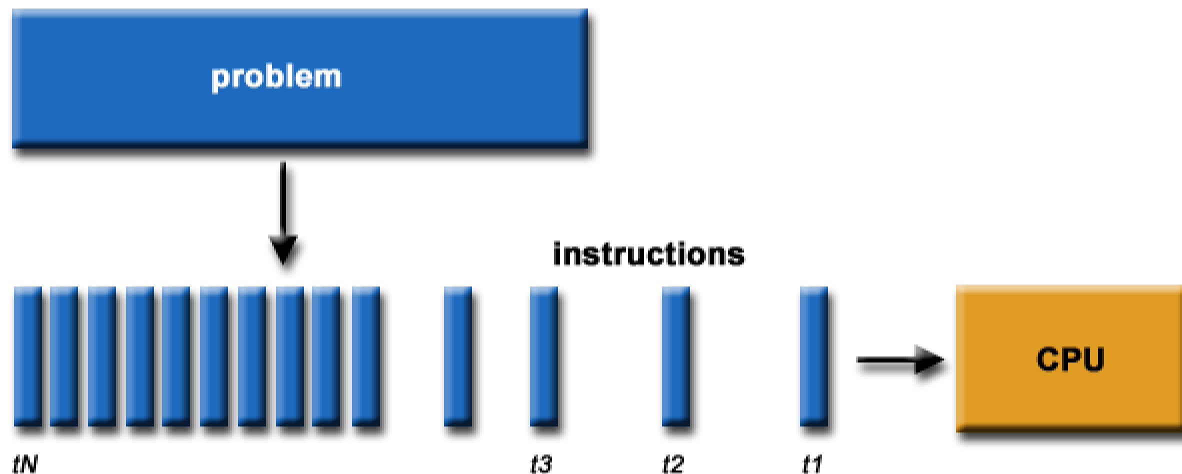






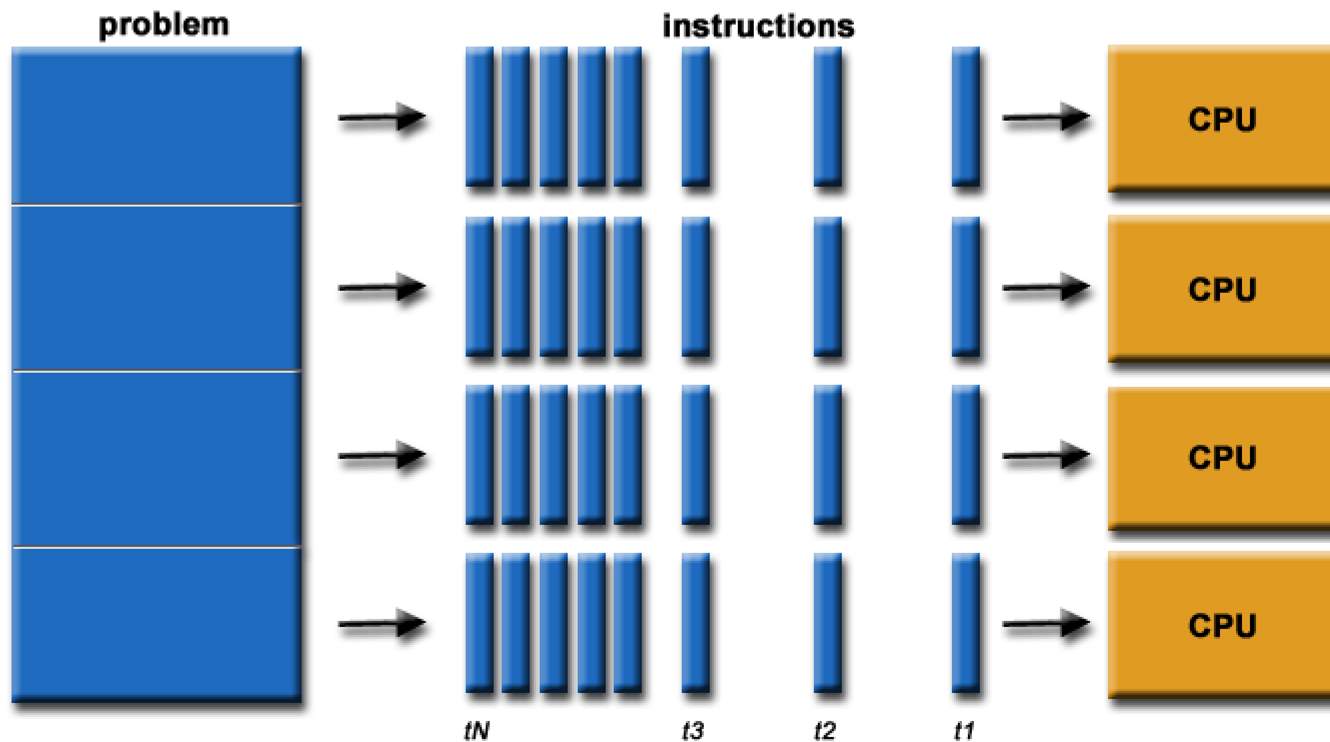
Serial computation

- Traditionally software has been written for serial computation:
  - run on a single computer
  - instructions are run one after another
  - only one instruction executed at a time



Parallel Computing

- Simultaneous use of multiple compute sources to solve a single problem



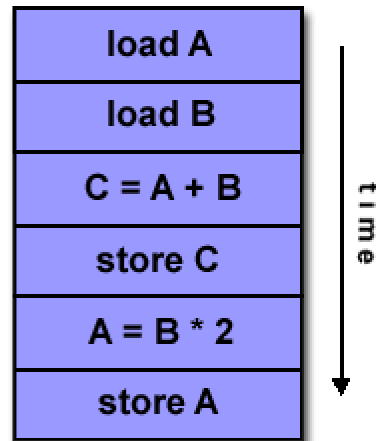
Concepts and Terminology

- Flynn's Taxonomy (1966)

S I S D	S I M D
Single Instruction, Single Data	Single Instruction, Multiple Data
M I S D	**M I M D**
Multiple Instruction, Single Data	Multiple Instruction, Multiple Data

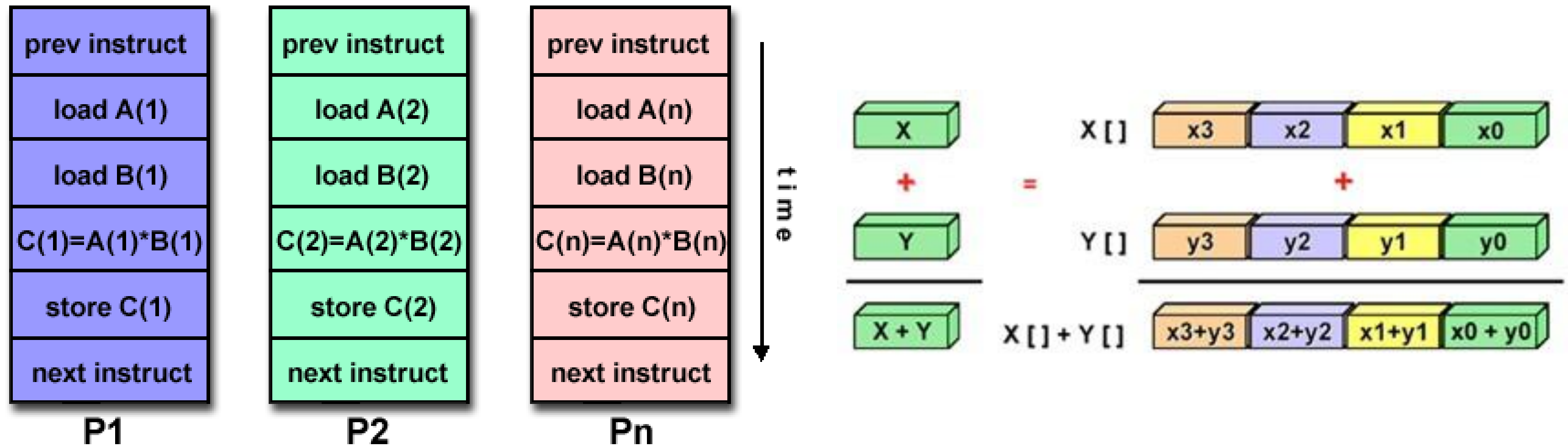
SISD

- Serial computer
- Deterministic execution
- Examples: older generation main frames, work stations, PCs



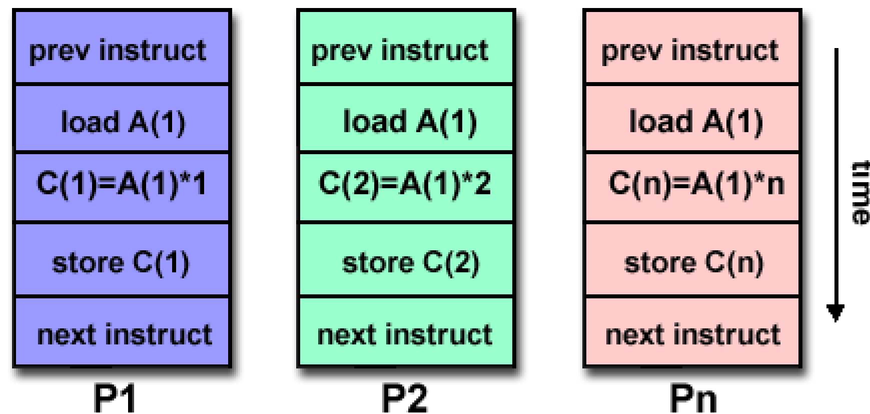
SIMD

- A type of parallel computer
- All processing units execute the same instruction at any given clock cycle
- Each processing unit can operate on a different data element
- Two varieties: Processor Arrays and Vector Pipelines
- Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.



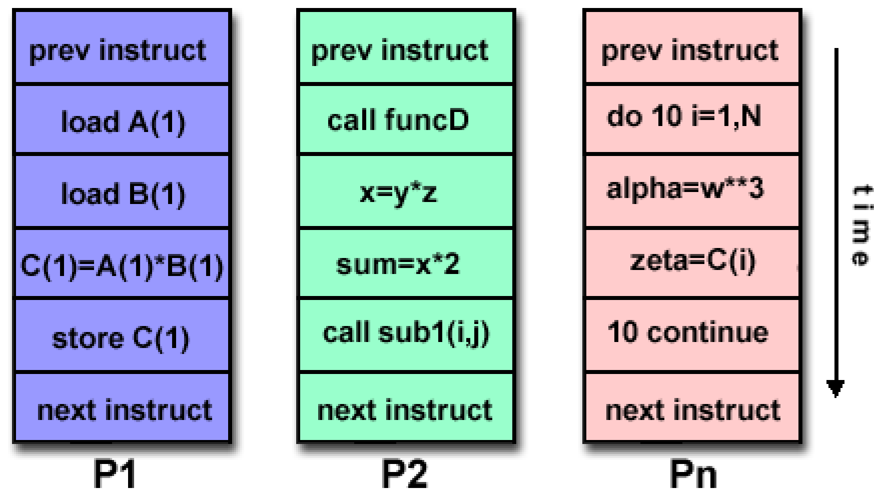
MISD

- A single data stream is fed into multiple processing units.
- Each processing unit operates on the data independently via independent instruction streams.
- Few actual examples : Carnegie-Mellon C.mmp computer (1971).



MIMD

- Currently, most common type of parallel computer
- Every processor may be executing a different instruction stream Every
- processor may be working with a different data stream Execution can
- be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.
- Note: many MIMD architectures also include SIMD execution sub-components



Parallel Computer Architectures

- Shared memory:

  all processors can access the

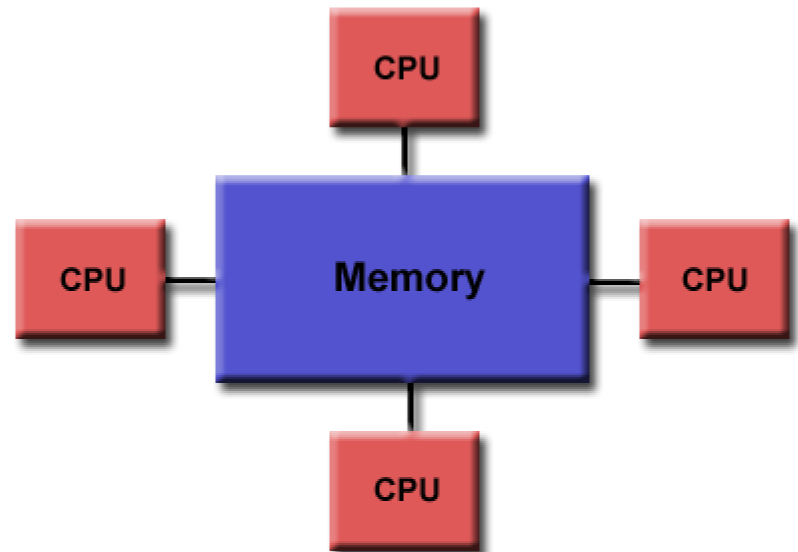
  same memory

- Uniform memory access (UMA):

  - identical processors

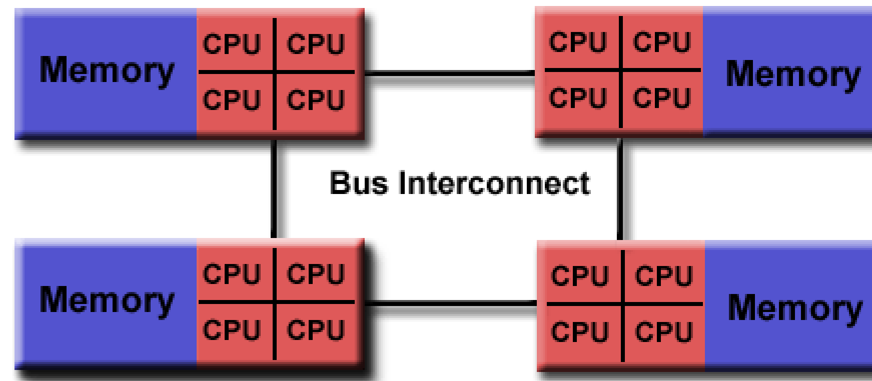
  - equal access and access

    times to memory



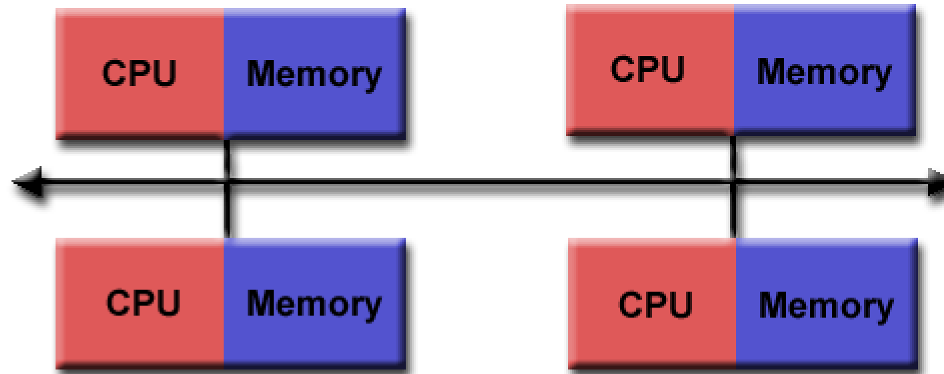
Non-uniform memory access (NUMA)

- Not all processors have equal access to all memories
- Memory access across link is slower



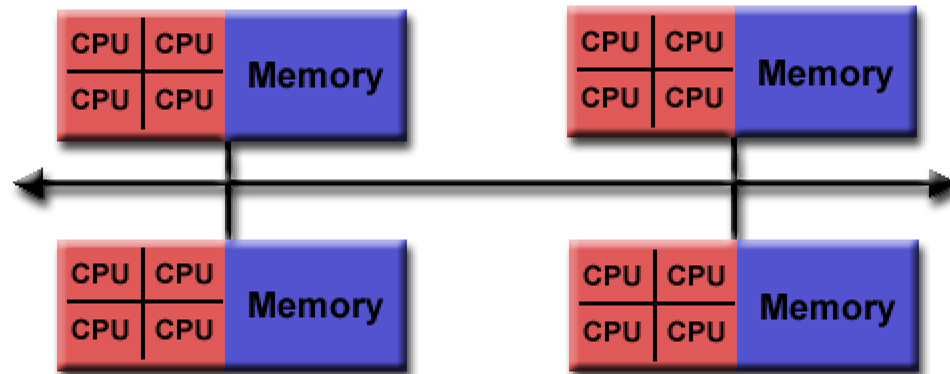
- Advantages:
  - user-friendly programming perspective to memory
  - fast and uniform data sharing due to the proximity of memory to CPUs
- Disadvantages:
  - lack of scalability between memory and CPUs.
  - Programmer responsible to ensure "correct" access of global
  - Expense

Distributed memory



- Distributed memory systems require a communication network to connect inter-processor memory.
- Advantages:
  - Memory is scalable with number of processors.
  - No memory interference or overhead for trying to keep cache coherency.
  - Cost effective
- Disadvantages:
  - programmer responsible for data communication between processors.
  - difficult to map existing data structures to this memory organization.

Hybrid distributed-shared memory



- Generally used for the currently largest and fastest computers
- Has a mixture of previously mentioned advantages and disadvantages

Parallel programming models

- Shared memory
- Threads
- Message Passing
- Data Parallel
- Hybrid

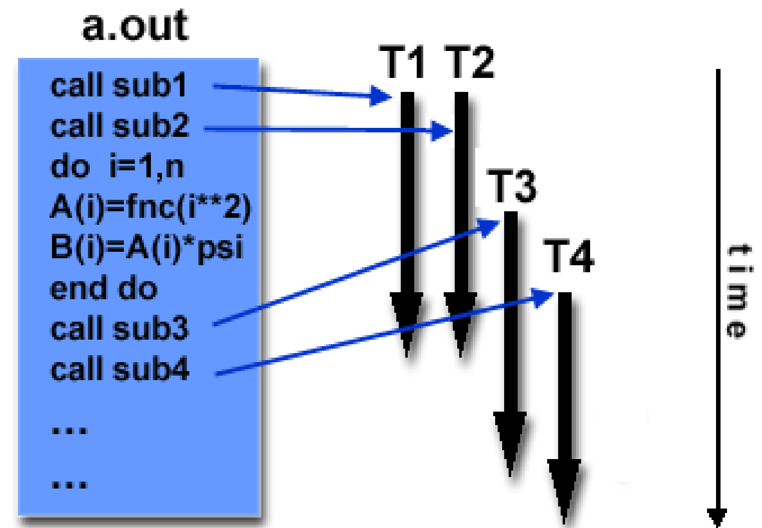
All of these can be implemented on any architecture.

Shared memory

- tasks share a common address space, which they read and write asynchronously.
- Various mechanisms such as locks / semaphores may be used to control access to the shared memory.
- Advantage: no need to explicitly communicate of data between tasks -> simplified programming
- Disadvantages:
  - Need to take care when managing memory, avoid synchronization conflicts
  - Harder to control data locality

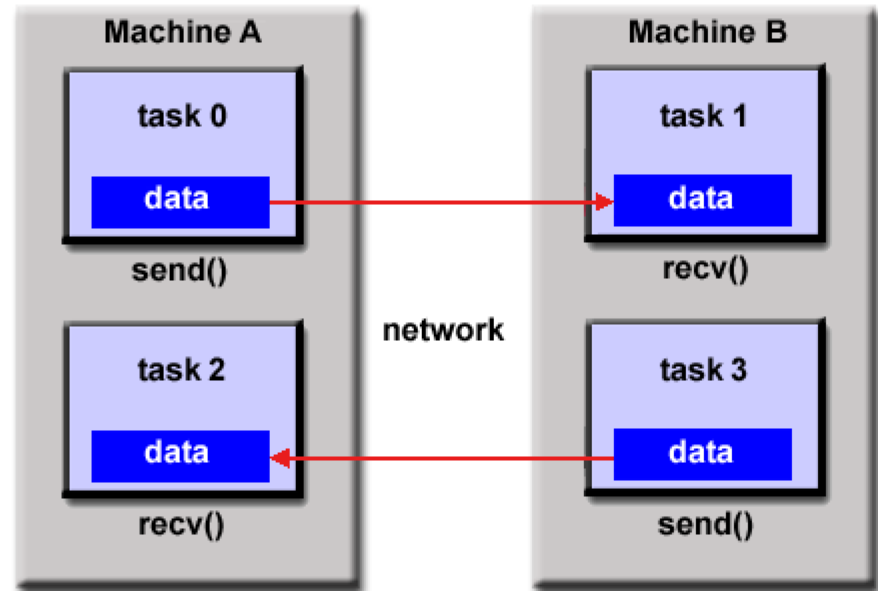
Threads

- A thread can be considered as a subroutine in the main program
- Threads communicate with each other through the global memory
- commonly associated with shared memory architectures and operating systems
- Posix Threads or pthreads
- OpenMP



Message Passing

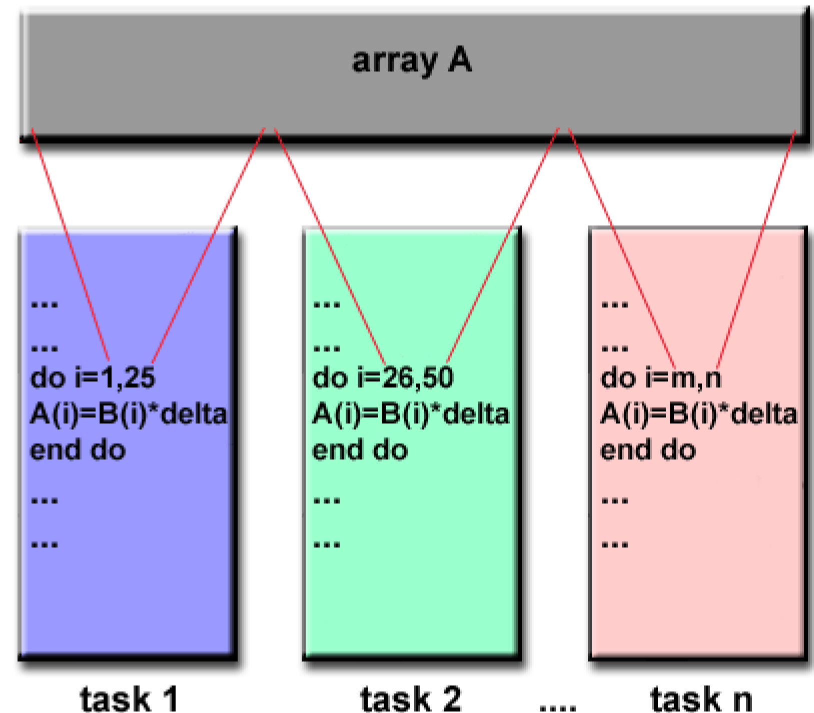
- A set of tasks that use their own local memory during computation.
- Data exchange through sending and receiving messages.
- Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation.



- MPI (released in 1994)
- MPI-2 (released in 1996)

Data Parallel

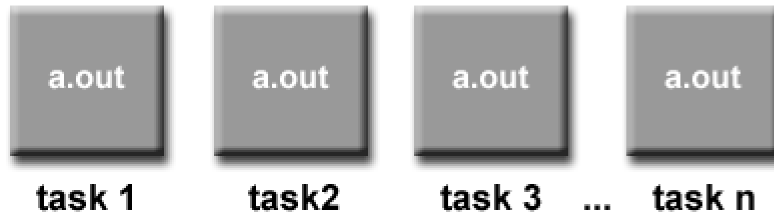
- The data parallel model demonstrates the following characteristics:
  - Most of the parallel work performs operations on a data set, organized into a common structure, such as an array
  - A set of tasks works collectively on the same data structure, with each task working on a different partition
  - Tasks perform the same operation on their partition



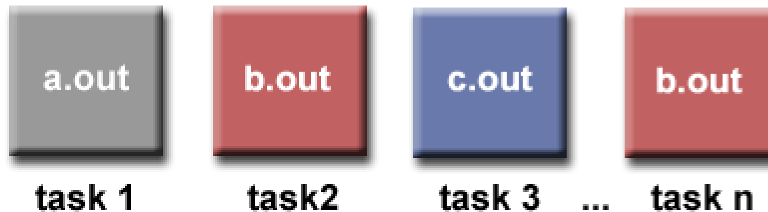
- On shared memory architectures, all tasks may have access to the data structure through global memory. On distributed memory architectures the data structure is split up and resides as "chunks" in the local memory of each task.

Other Models

- Hybrid
  - combines various models, e.g. MPI/OpenMP
- Single Program Multiple Data (SPMD)
  - A single program is executed by all tasks simultaneously



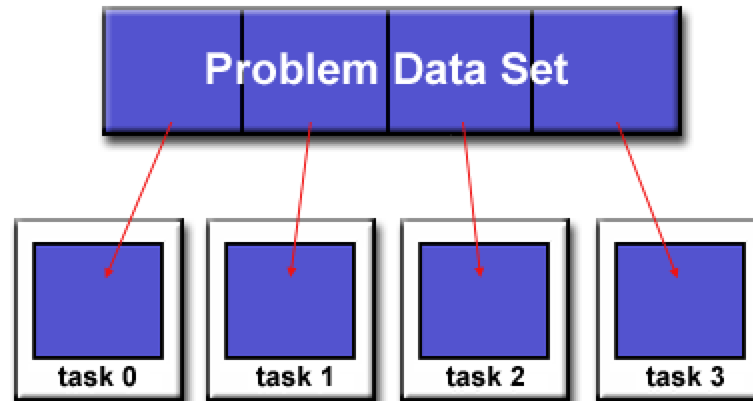
- Multiple Program Multiple Data (MPMD)
  - An MPMD application has multiple executables. Each task can execute the same or different program as other tasks.



Designing Parallel Programs

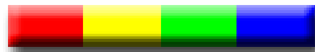
- Examine problem:
  - Can the problem be parallelized?
  - Are there data dependencies?
  - where is most of the work done?
  - identify bottlenecks (e.g. I/O)

- Partitioning
  - How should the data be decomposed?

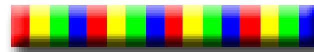


Various partitionings

1D



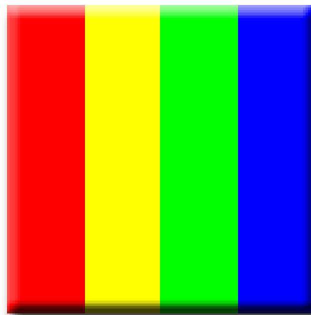
BLOCK



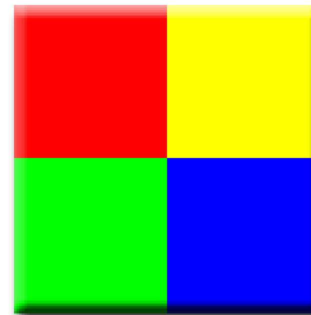
CYCLIC

2D

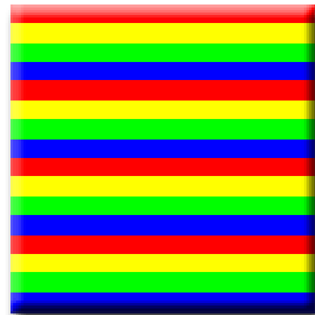
BLOCK, *



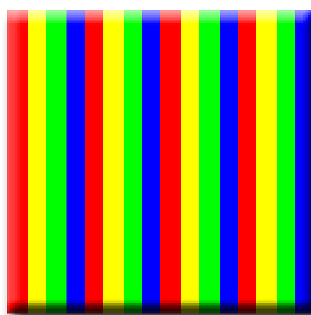
*, BLOCK



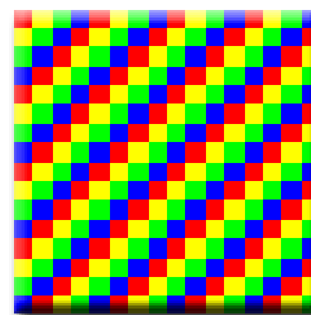
BLOCK, BLOCK



CYCLIC, *

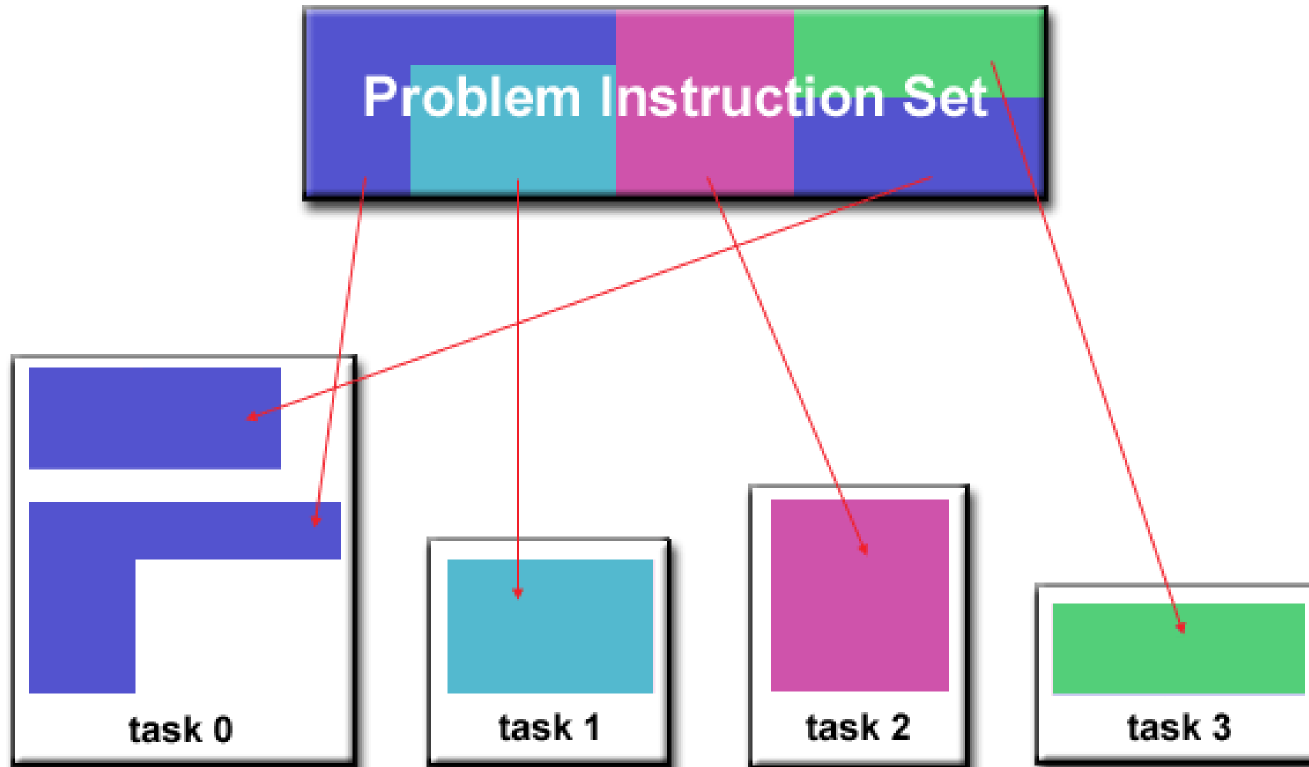


*, CYCLIC



CYCLIC, CYCLIC

How should the algorithm be decomposed

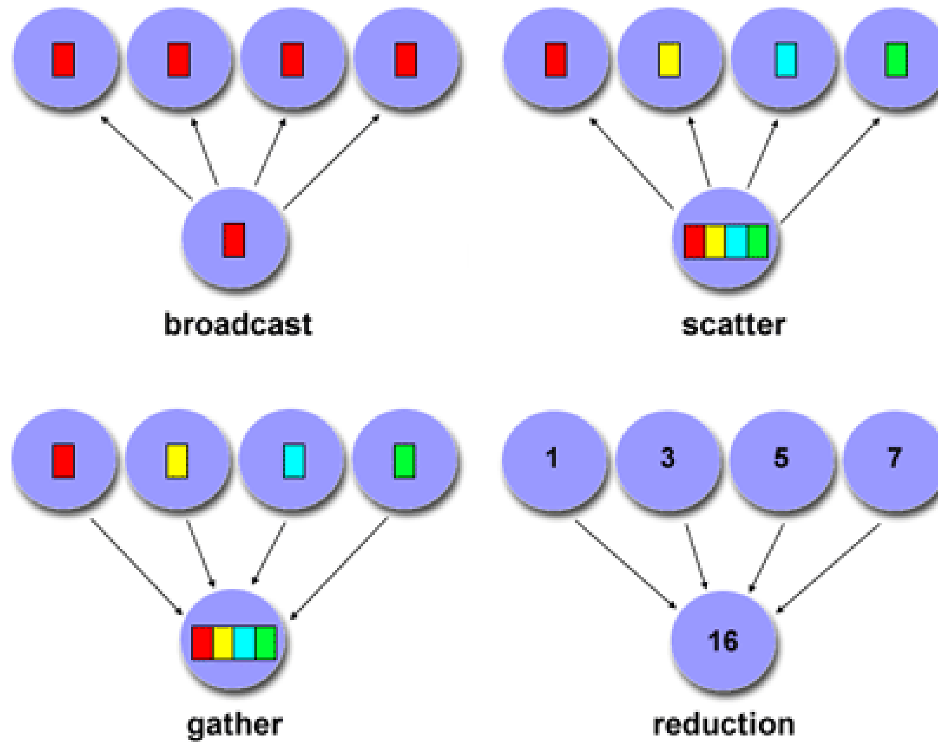


Communications

- Types of communication:

  - point-to-point

  - collective

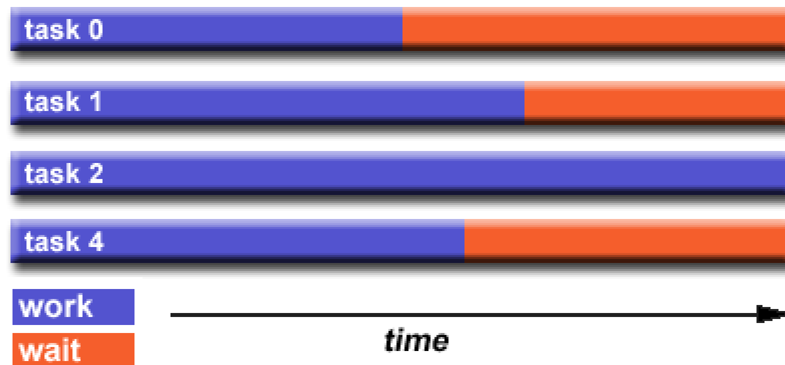


Synchronization types

- **Barrier**
  - Each task performs its work until it reaches the barrier. It then stops, or "blocks".
  - When the last task reaches the barrier, all tasks are synchronized.
- **Lock / semaphore**
  - The first task to acquire the lock "sets" it. This task can then safely (serially) access the protected data or code.
  - Other tasks can attempt to acquire the lock but must wait until the task that owns the lock releases it.
  - Can be blocking or non-blocking

Load balancing

- Keep ***all*** tasks busy ***all*** of the time. Minimize idle time.
- The slowest task will determine the overall performance.



Granularity

- In parallel computing, granularity is a qualitative measure of the ratio of computation to communication.
- Fine-grain Parallelism:
  - Low computation to communication ratio
  - Facilitates load balancing
  - Implies high communication overhead and less opportunity for performance enhancement
- Coarse-grain Parallelism:
  - High computation to communication ratio
  - Implies more opportunity for performance increase
  - Harder to load balance efficiently