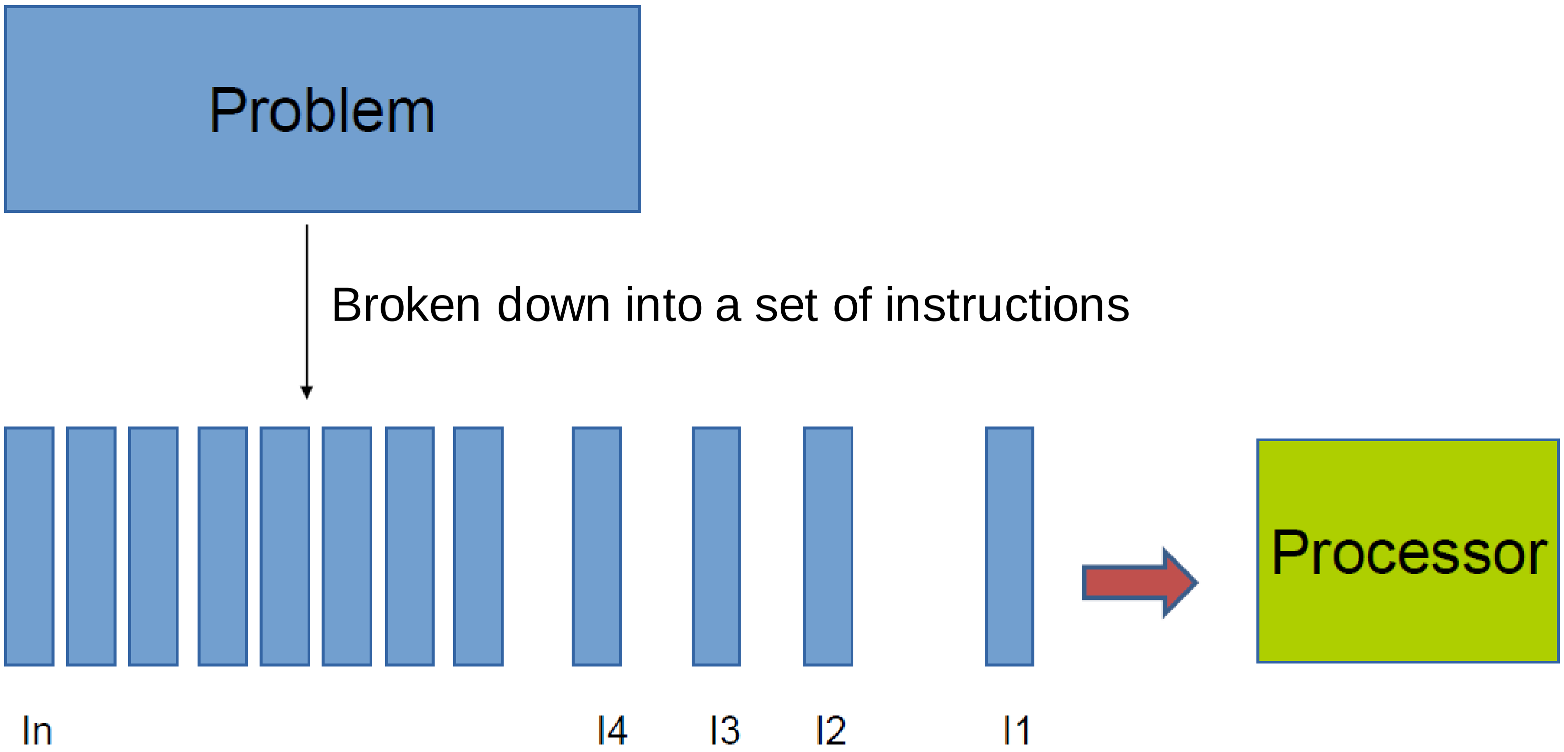
Topic: Parallel Computing/Programming

Objective

- Serial vs Parallel computing
- Parallel memory architecture
- Parallel programming models

Topic: Parallel Computing/Programming

Serial computing

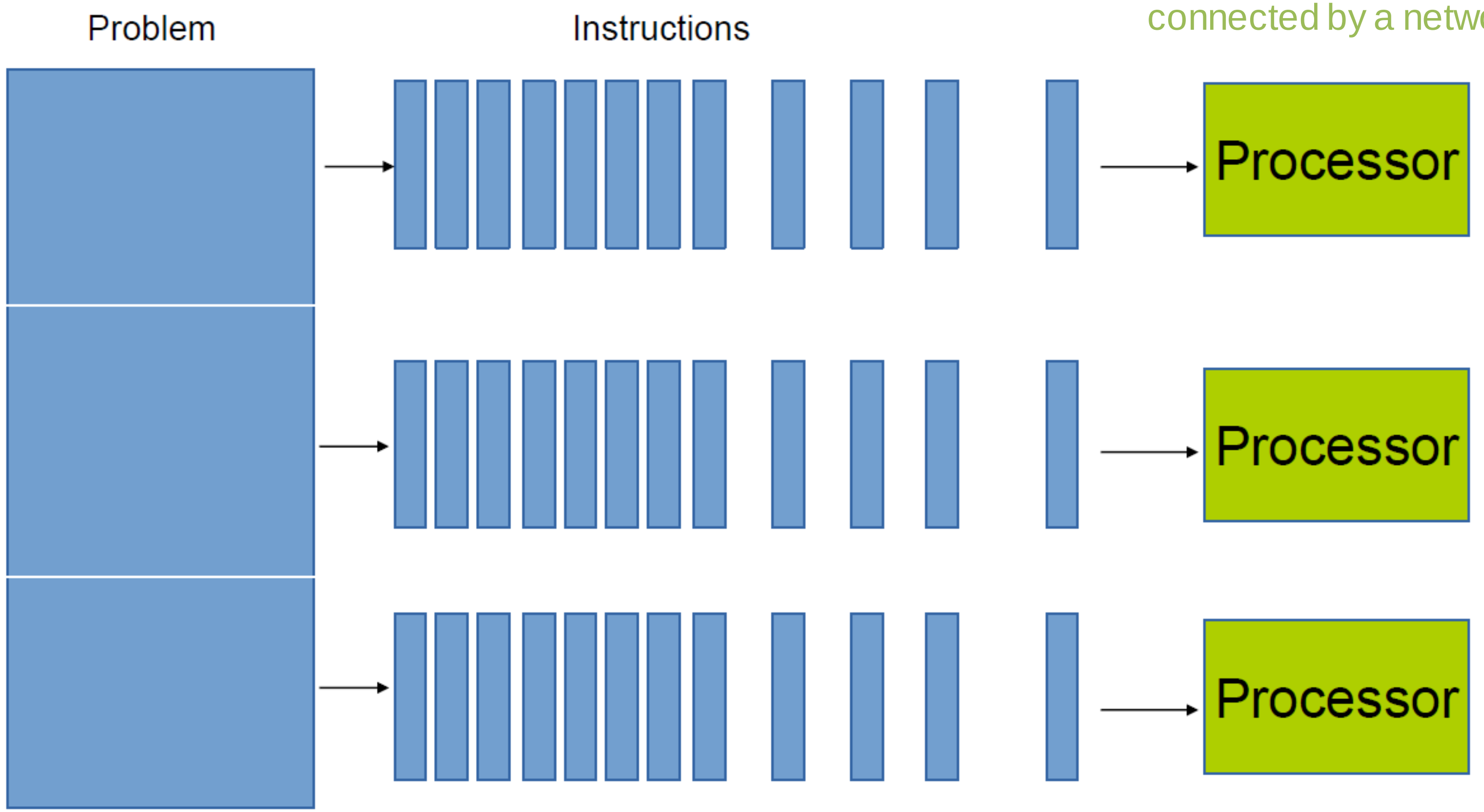


Instructions are executed sequentially one after another

Topic: Parallel Computing/Programming

Parallel computing

Processors can be from a single computer or computers connected by a network

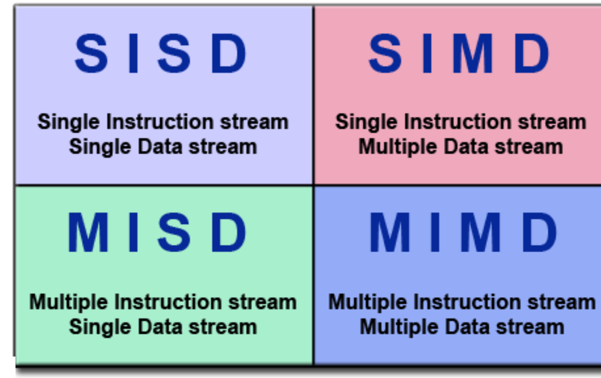


Instructions from each part execute simultaneously on different processors

Ex. MPI programming

Flynn's classification of parallel computers

distinguishes multi-processor computer architectures based on **Instruction Stream** and **Data Stream**



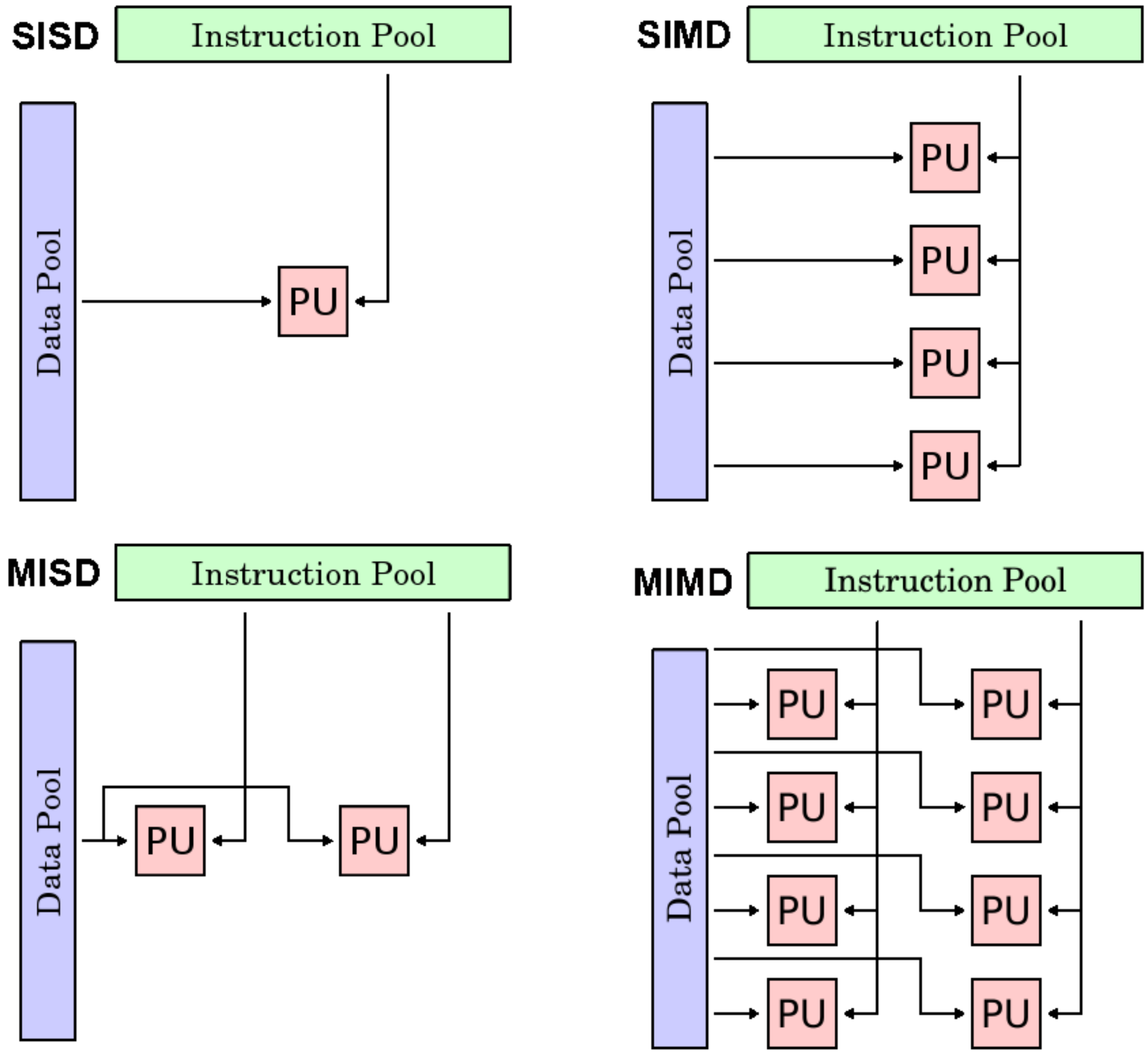
Instruction Streams

Data Streams

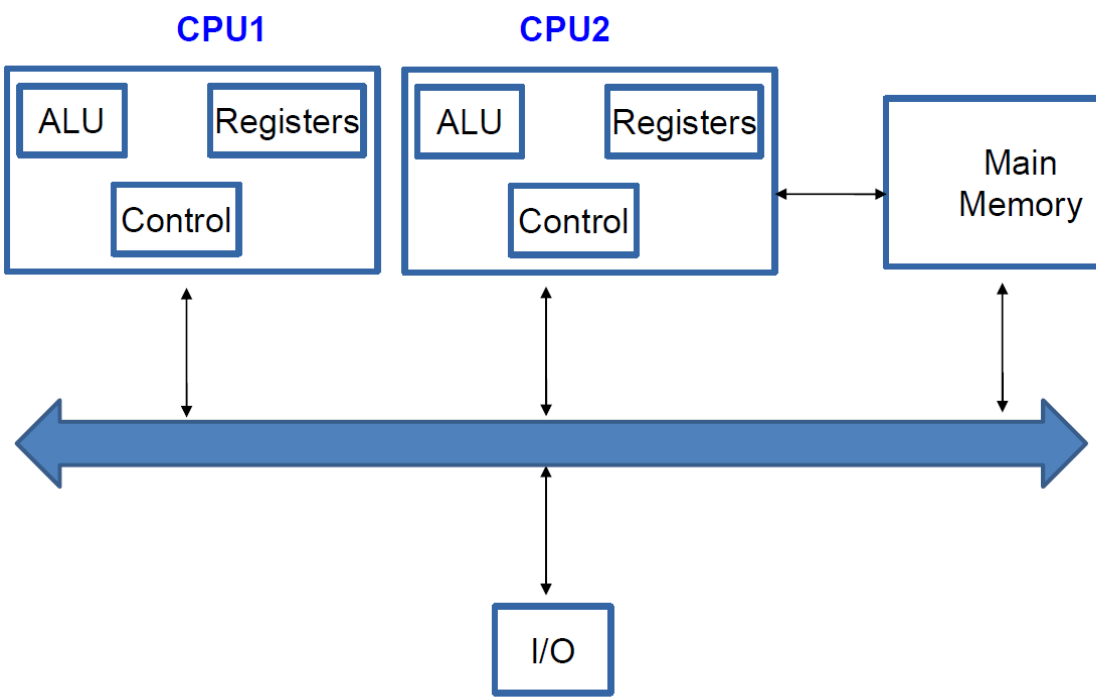
	one	many
one	**SISD** Traditional von Neumann single CPU computer	**MISD** May be pipelined computers
many	**SIMD** Vector processors fine grained data Parallel computers	**MIMD** Multi computers Multiprocessors

Topic: Parallel Computing/Programming

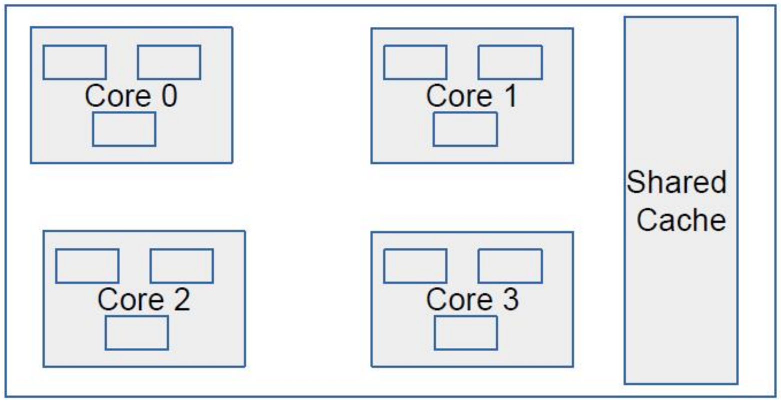
Flynn's classification of parallel computers

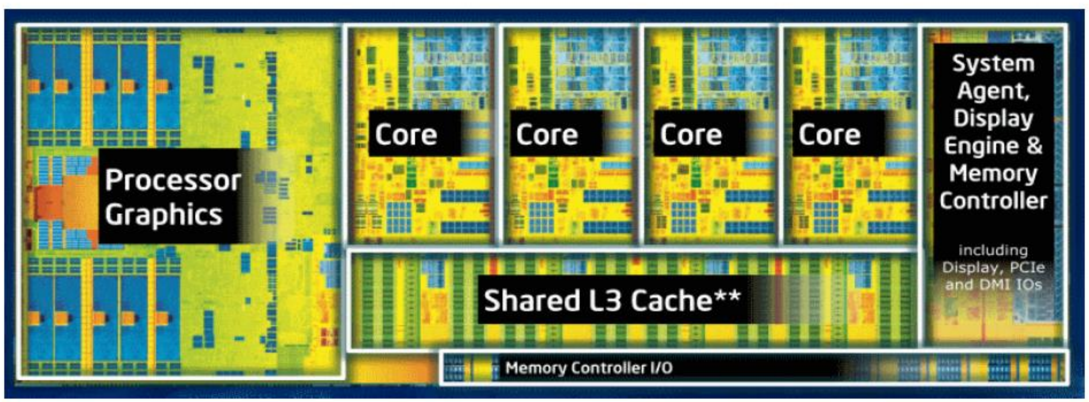


Parallel computer architecture

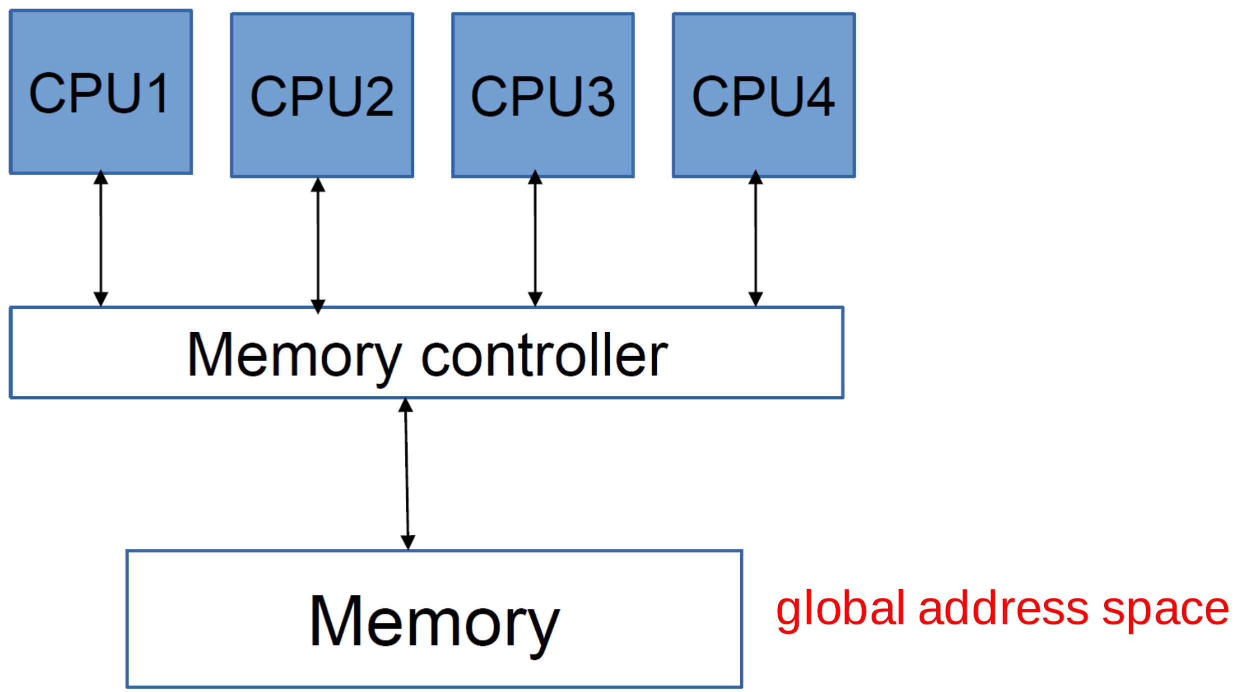


Multi-core processor



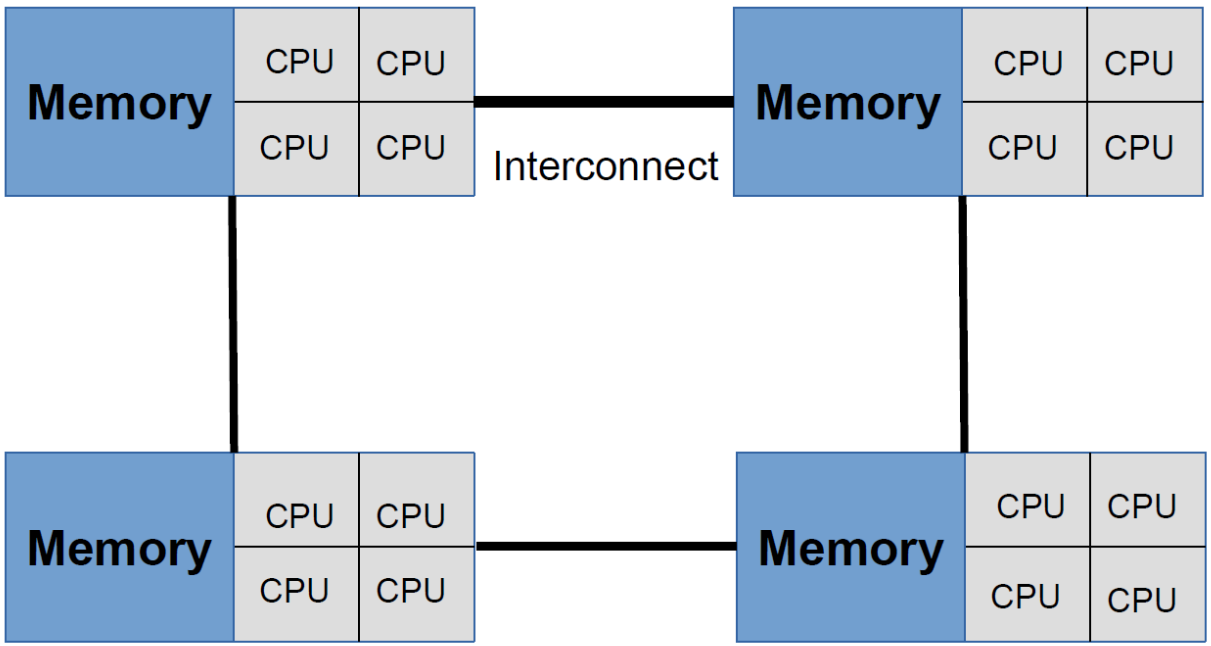


Parallel computer memory architectures -- Uniform memory access (UMA)



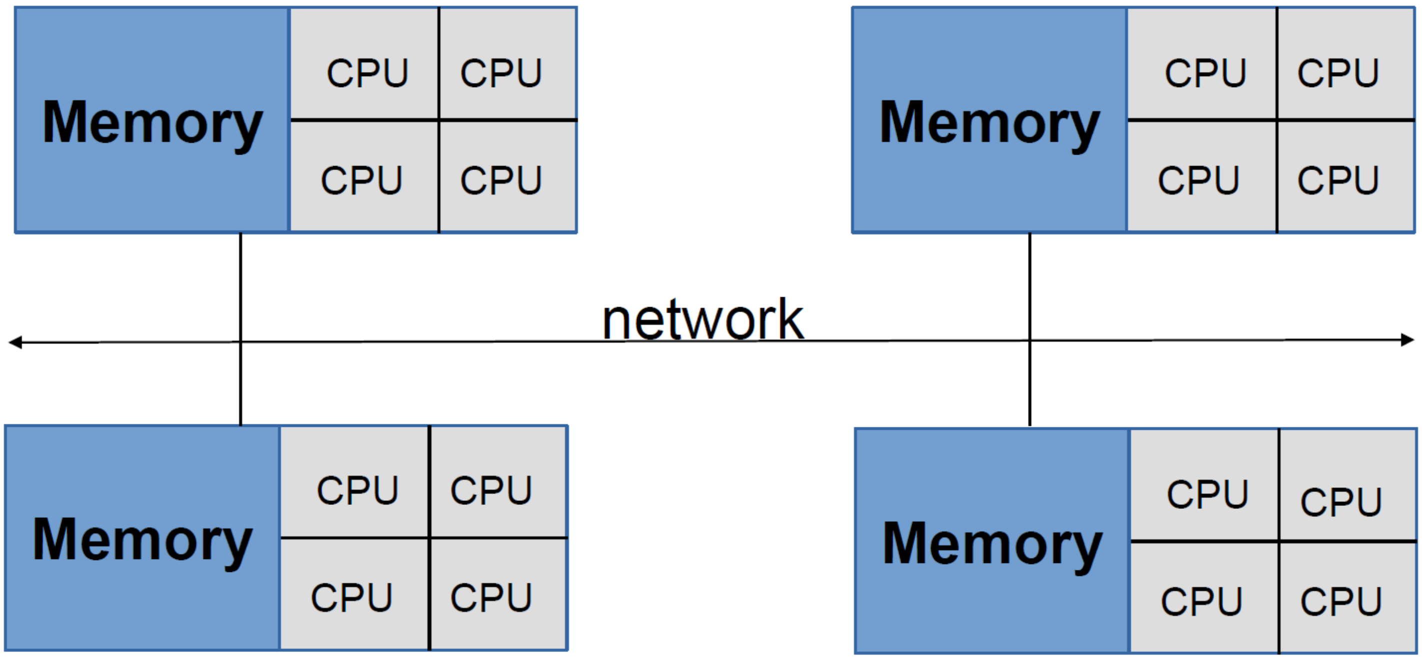
- Equal access and access times to memory
- Identical processors
- Cache coherent

Parallel computer memory architectures -- Non-uniform memory access (NUMA)



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

Parallel computer memory architectures -- Hybrid distributed-shared systems



All supercomputers use this memory architecture

- Memory is local to shared memory processor/GPU
- Cache coherence does not apply
- No global address space

Types of computing

- **Parallel**: multiple tasks cooperate closely to solve a problem. (speedup and efficiency)
- **Concurrent**: multiple tasks can be in progress at any instant, may or may not execute multiple executions at the same time, may or may not interact with each other. (convenience)
- **Distributed**: program may need to cooperate with other programs to solve a problem.

Parallel

- **Parallel**: multiple tasks cooperate closely to solve a problem. (speedup and efficiency)

```
If(rank==0) then
    work0
elseif(rank==1) then
    work1
elseif(rank==2) then
    work2
endif
```

Work0, work1, and work2 are independent of each other

Each process/thread/task can progress independently, and OS schedules them to execute parallelly

Parallelism is about doing lot of things at once -- Rob Pike

Concurrency

- **Concurrent**: multiple tasks can be in progress at any instant, may or may not execute multiple executions at the same time, may or may not interact with each other. (convenience)

```
If(rank==0) then
    work0
elseif(rank==1) then
    work1
endif
```

work0 depends on work1 and vice vers, i. e. share the variables.

Topic: Parallel Computing/Programming

Concurrency

```
If(rank==0) then
     If(no_tickets>0) then
          work --> book_ticket
           no_tickets = no_tickets - 1
elseif(rank==1) then
      If(no_tickets>0) then
          work --> book_ticket
           no_tickets = no_tickets - 1
endif
```

Example: booking a flight ticket.

Shared memory location: book_ticket

Concurrency is used when multiple processes/tasks/threads need to coordinate or when shared variables or memory locations need to be updated

Concurrency is about dealing with lot of things at once -- Rob Pike

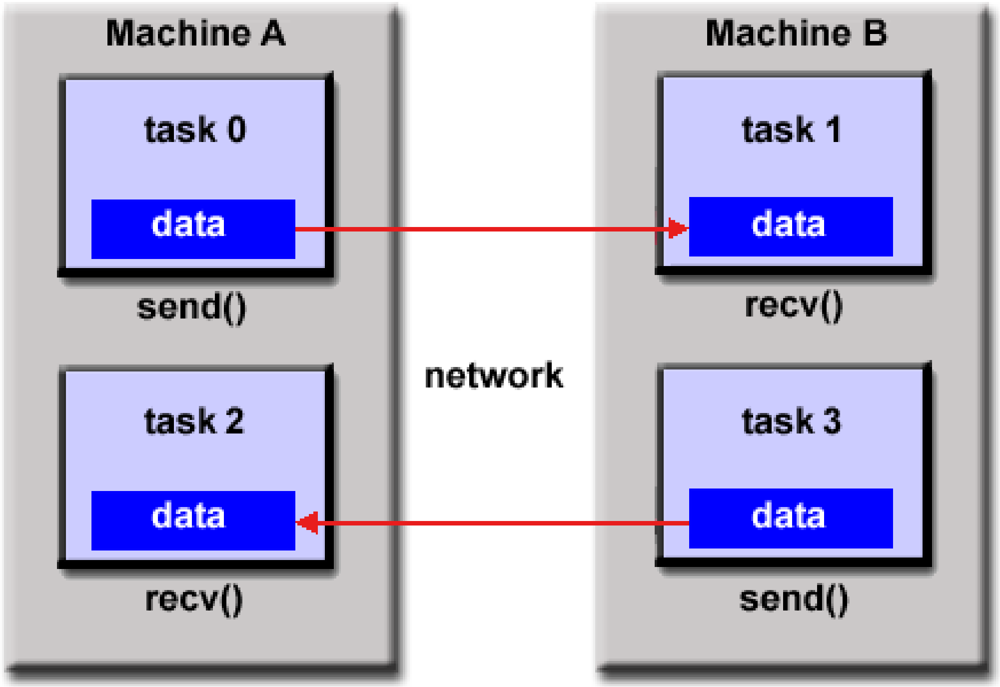
parallel programming models

- **Shared Memory**
- Threads
- **Distributed Memory / Message Passing, eg. MPI**
- Data Parallel
- **Hybrid**
- Single Program Multiple Data (SPMD)
- Multiple Program Multiple Data (MPMD), etc...

These models are NOT specific to a particular type of machine or memory architecture, i. e. shared memory model on distributed memory machine, distributed memory model on the shared memory machine, etc.

Distributed memory model or Message passing model

- A set of tasks that use their own local memory during computation.
- Multiple tasks can reside on the same physical machine as well across an arbitrary number of machines.
- Data exchange between tasks done usind send() and recv() functions



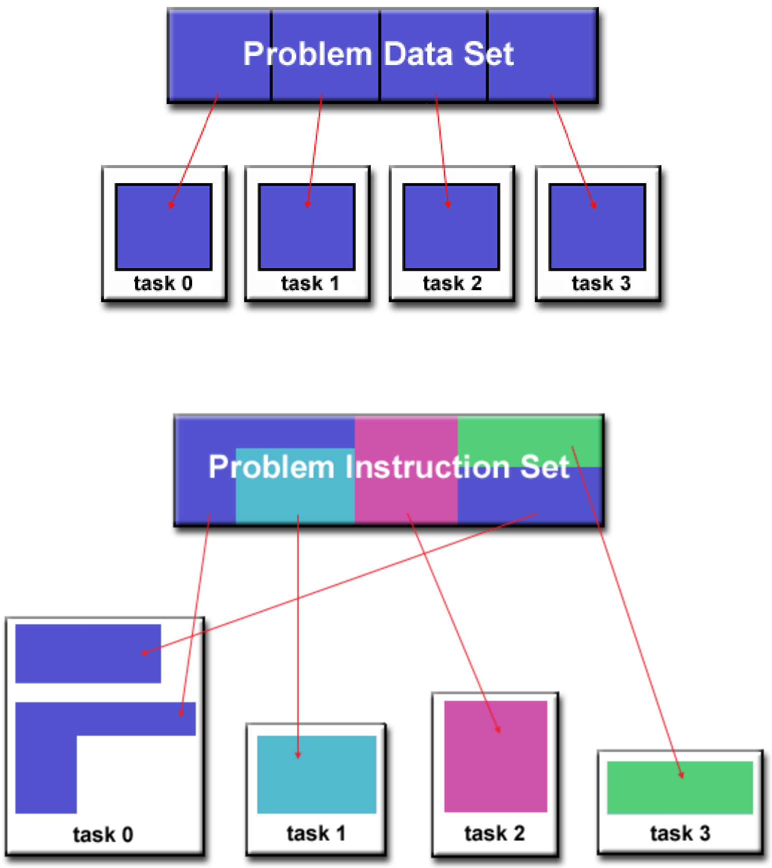
Shared memory model

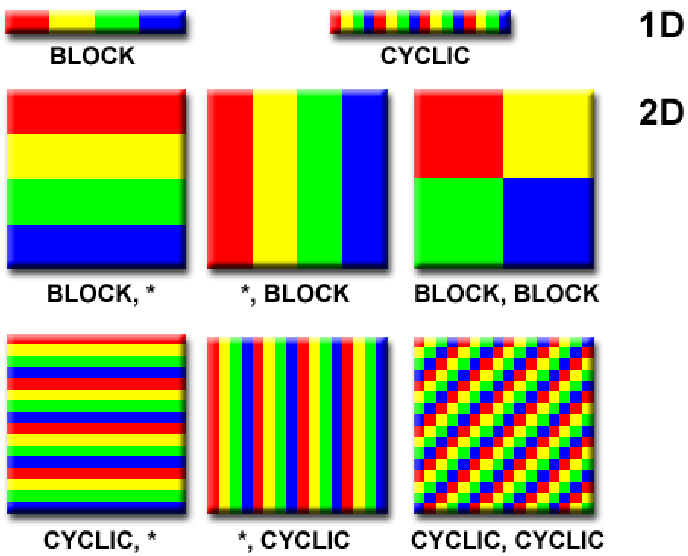
- Tasks share a common address space, which they read and write asynchronously.
- Various mechanisms such as locks may be used to control access to the shared memory.
- An advantage of this model from the programmer's point of view is that the notion of data "ownership" is lacking, so there is no need to specify explicitly the communication of data between tasks.

Factors which effects the Inter-task communication

Partitioning

- Domain decomposition and functional decomposition

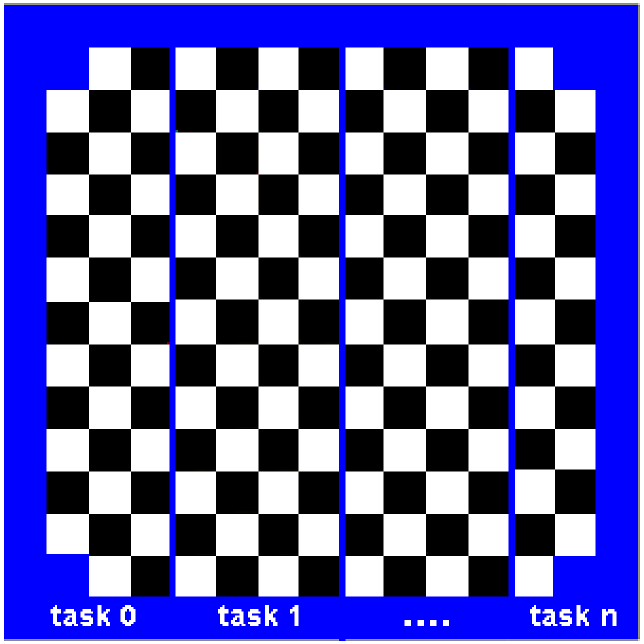




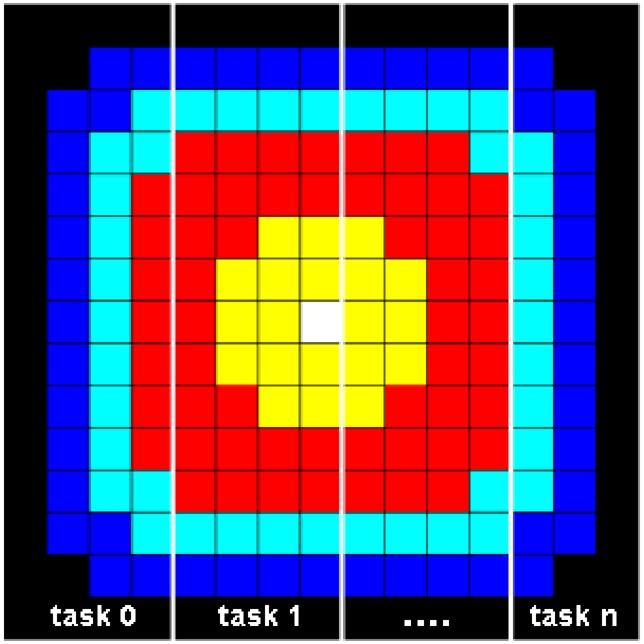
https://computing.llnl.gov/tutorials/parallel_comp/#Models

Topic: Parallel Computing/Programming

Coordination - Communication



Don't need communications

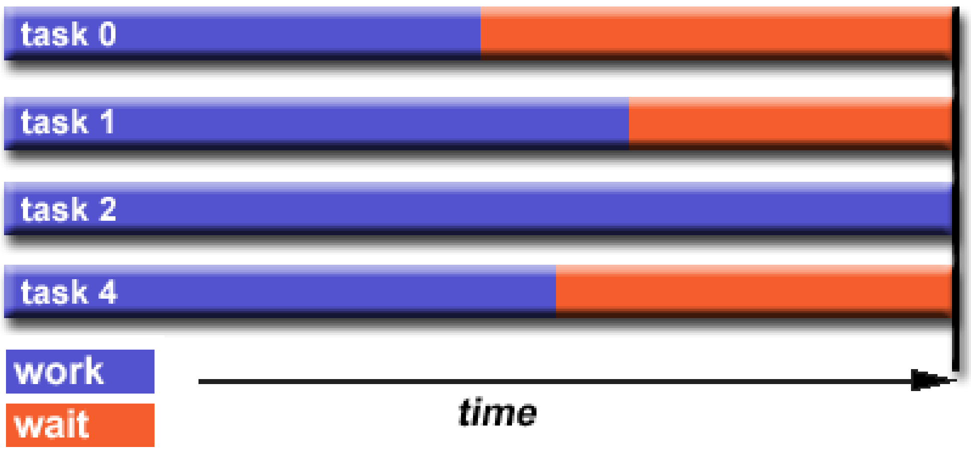


Need communications

MPI_Comm_World, MPI_Bcast, blocking and non-blocking, etc

https://computing.llnl.gov/tutorials/parallel_comp

Coordination - Load balancing



- the slowest task will determine the overall performance.
- Equally partition the work each task receives or use dynamic work assignment

https://computing.llnl.gov/tutorials/parallel_comp

Coordination - synchronization

- Synchronization among tasks is needed before communication happens
- Blocking and non-blocking

https://computing.llnl.gov/tutorials/parallel_comp

Speed up and efficiency

Speed up

$$S(n,P) = \frac{T(n,1)}{T(n,P)}$$

where $0 < S(n,P) \leq P$

Efficiency

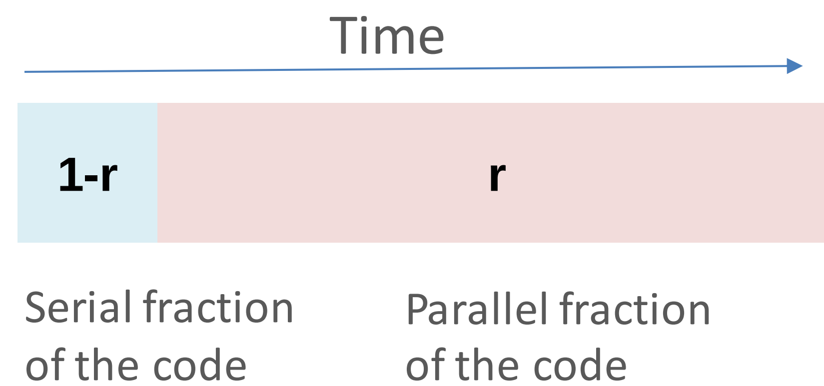
$$E(n,P) = \frac{S(n,P)}{P}$$

where $0 < E(n,P) \leq 1$

- run time of a parallel program depends on the number of processes P as well as the input size n.
- It is the main goal of any parallel program to achieve the best speedups.

Amdhal's law

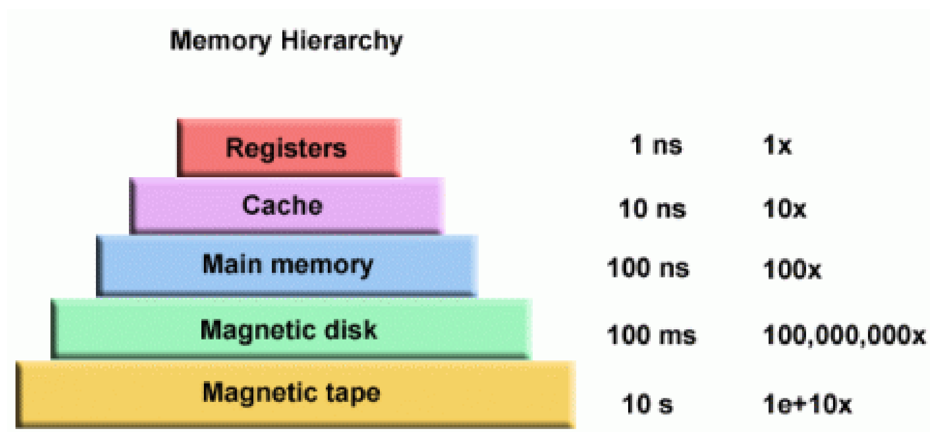
Provides an upper bound on the speedup that can be obtained by a parallel program.



$$S(n,P) = \frac{T(n,1)}{(1-r)T(n,1) + \frac{rT(n,1)}{P}}$$
$$= \frac{1}{(1-r) + \frac{r}{P}}.$$

Now as $P \to \infty$,

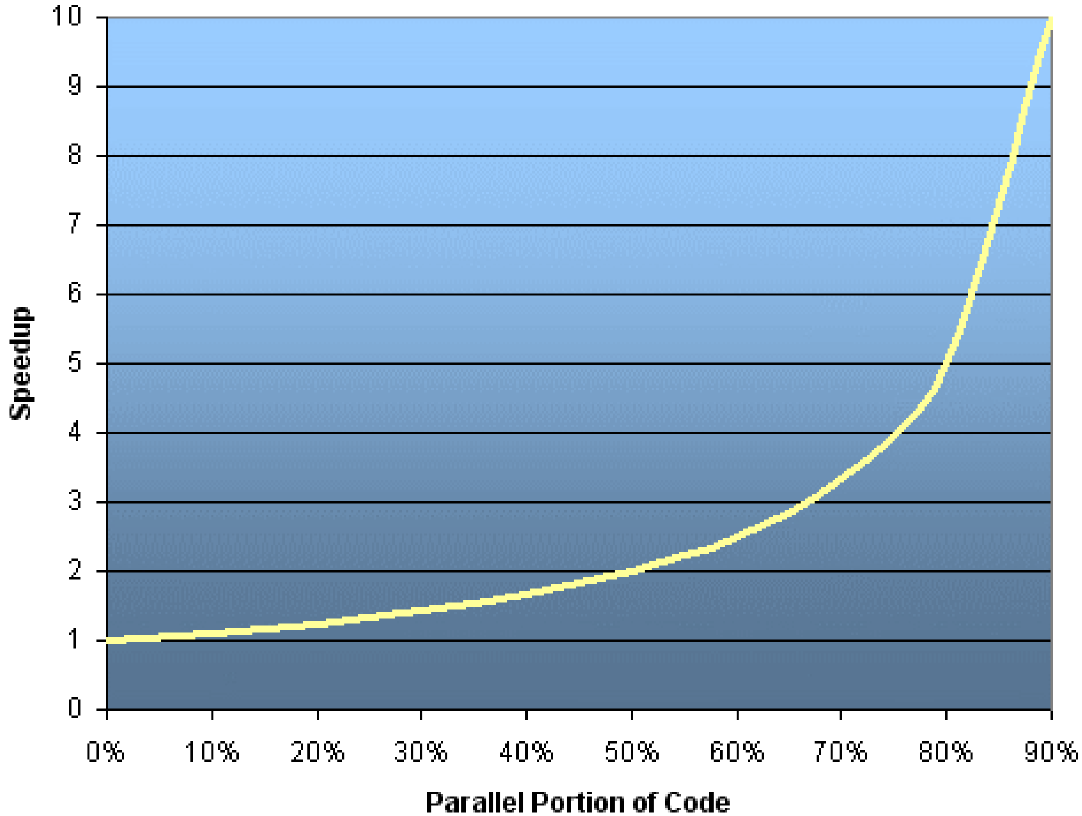
$$S(n,P) \to \frac{1}{1-r}$$



https://computing.llnl.gov/tutorials/parallel_comp

Topic: Parallel Computing/Programming

Amdhal's law



as $P \to \infty$,

$$S(n, P) \to \frac{1}{1 - r}$$

https://computing.llnl.gov/tutorials/parallel_comp

Amdhal's law

Speedup

	Speedup			
P	r = .50	r = .90	r = .95	r = .99
10	1.82	5.26	6.89	9.17
100	1.98	9.17	16.80	50.25
1000	1.99	9.91	19.62	90.99
10000	1.99	9.91	19.96	99.02
100000	1.99	9.99	19.99	99.90

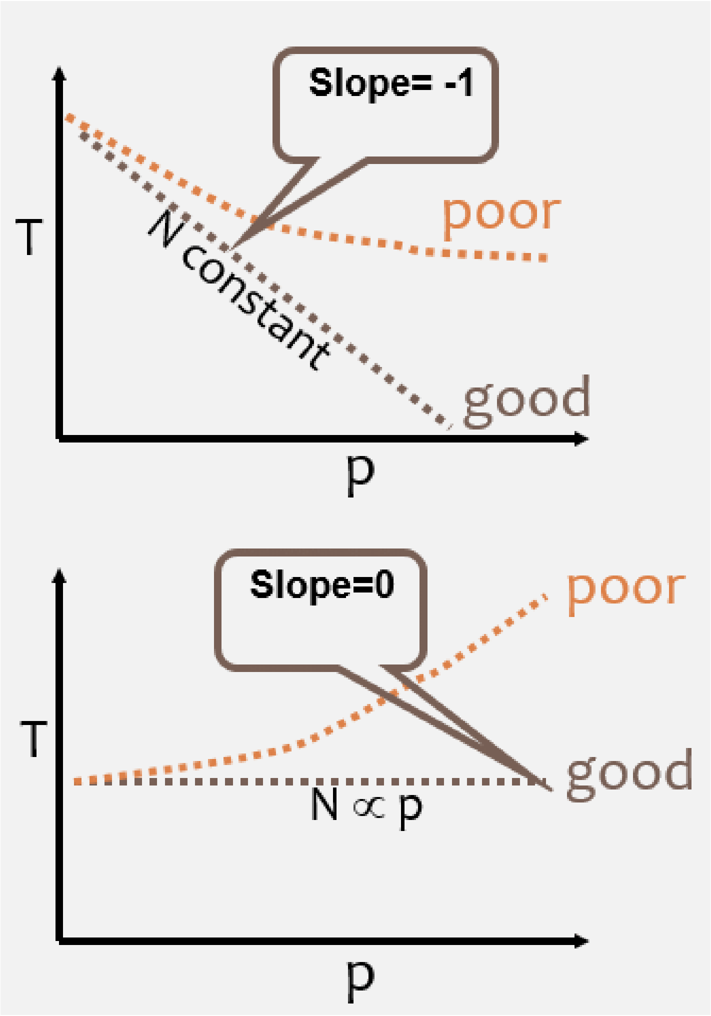
https://computing.llnl.gov/tutorials/parallel_comp

Topic: Parallel Computing/Programming

Scalability

Strong scaling: Keep problem size fixed and vary processor number

Weak scaling: vary both problem size and processor number



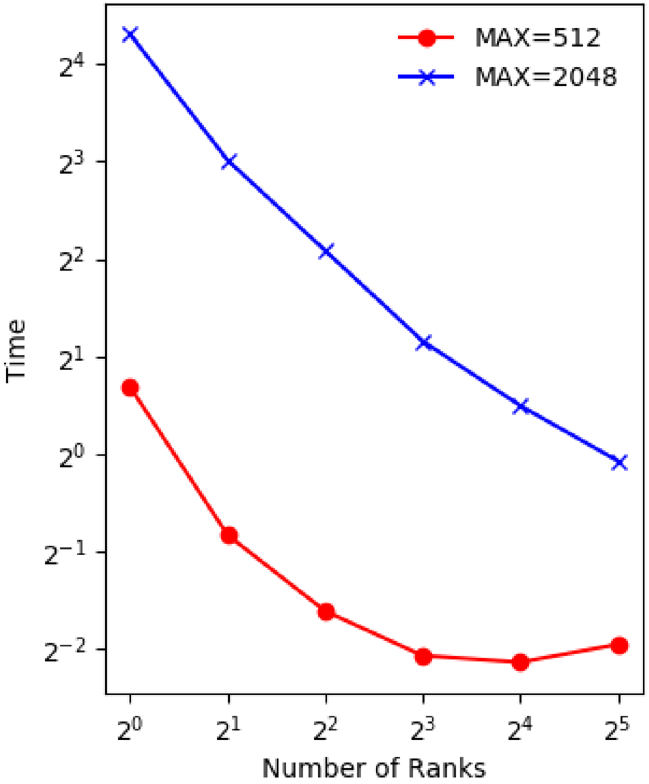
https://computing.llnl.gov/tutorials/parallel_comp

Topic: Parallel Computing/Programming

Poisson solver

Strong scaling:

- Grid-size=512 and Grid-size=2048
- Ran on two 20-core Intel Xeon Scalable CPUs



https://rantahar.github.io/introduction-to-mpi/index.html

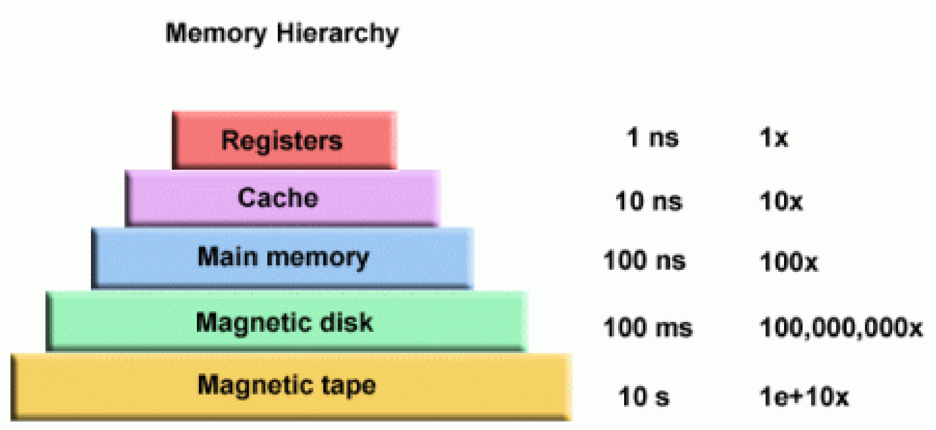
Parallel program design

- Parallelization is required when
  - The problem is too big to fit into the memory of one processor.
  - The problem takes too long to run on one processor.

- Know when to parallelize – not all parts of the program are parallelizable
- Identify the program's hotspots:
  - Know where most of the real work is being done. Use **Profilers** and **performance analysis tools** can help here
  - Focus on parallelizing the hotspots and ignore those sections of the program that account for little CPU usage.

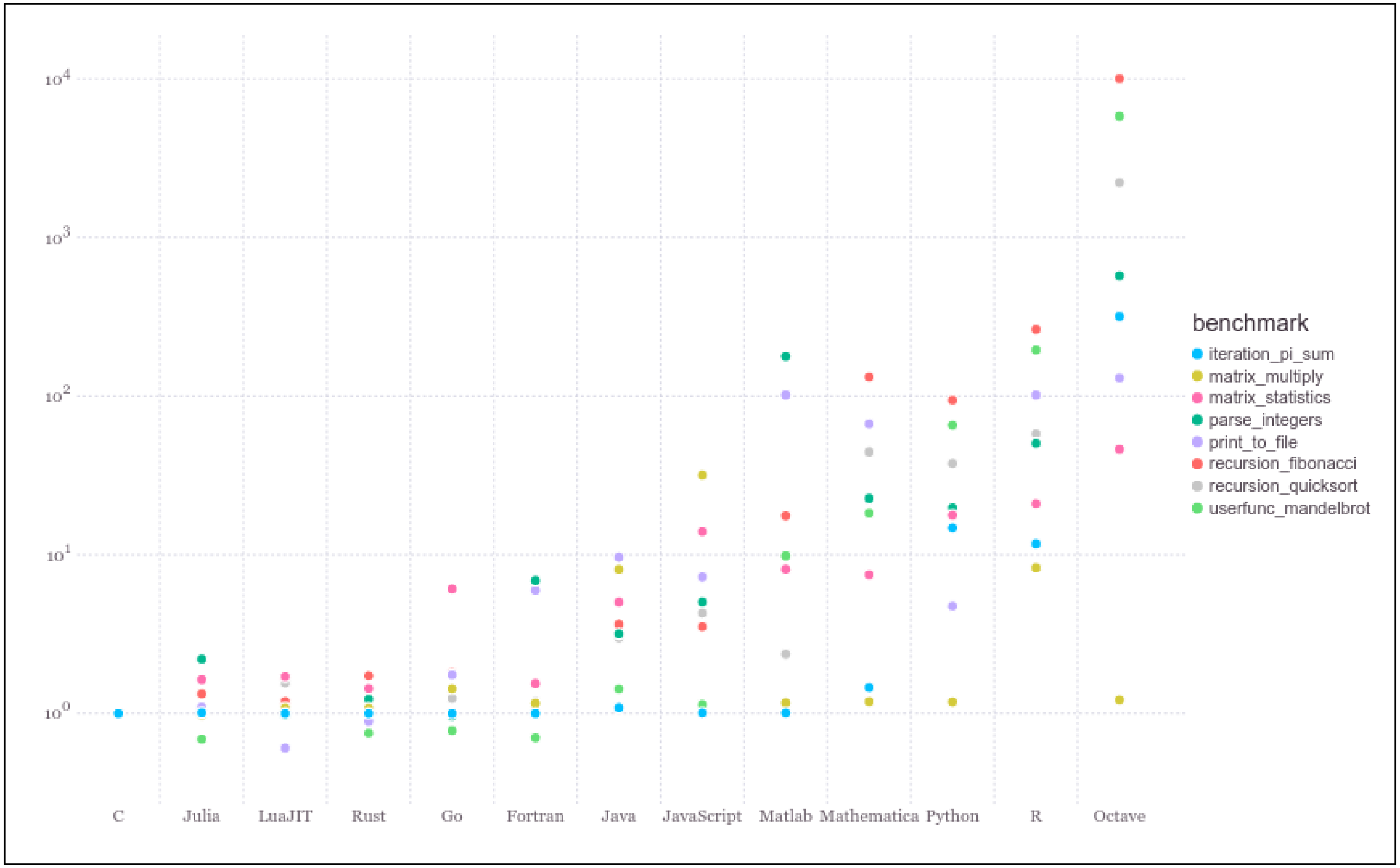
Parallel program design

- Identify bottlenecks in the program:
    - Are there areas that are disproportionately slow, or cause parallelizable work to halt or be deferred? For example, I/O
    - If possible, restructure the program or use a different algorithm to reduce or eliminate unnecessary slow areas
- Identify inhibitors to parallelism, Ex. Data dependency
- Take advantage of optimized third-party parallel software



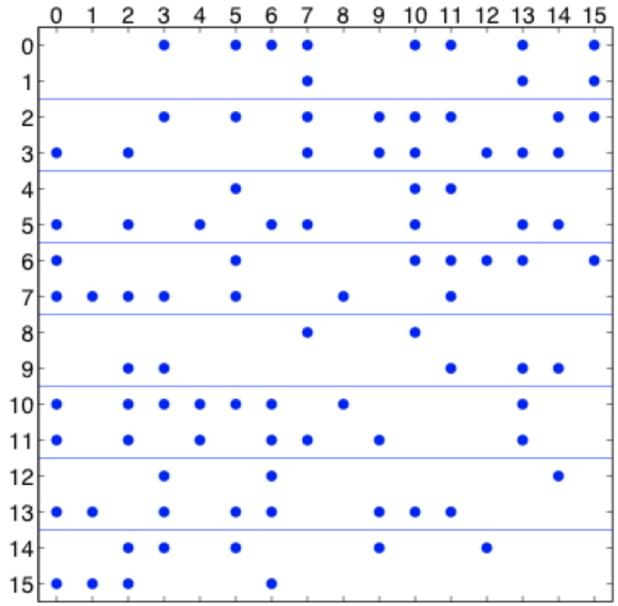
Topic: Parallel Computing/Programming

Parallel program design

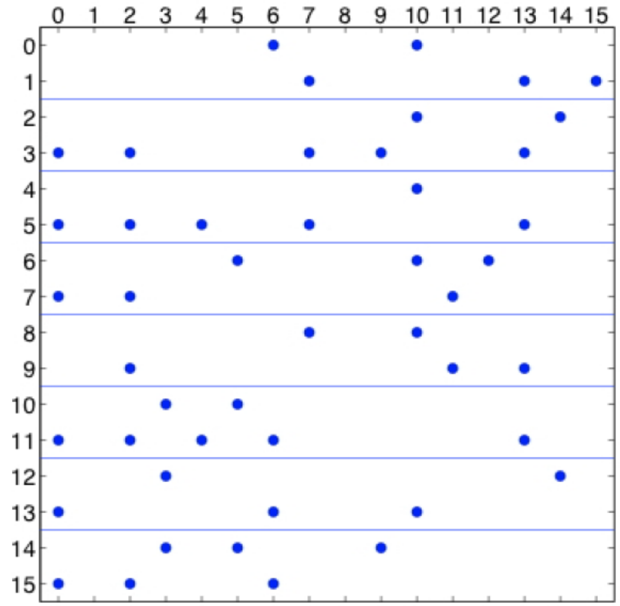


https://julialang.org/benchmarks/

Parallel Decompositions -- Atom decomposition

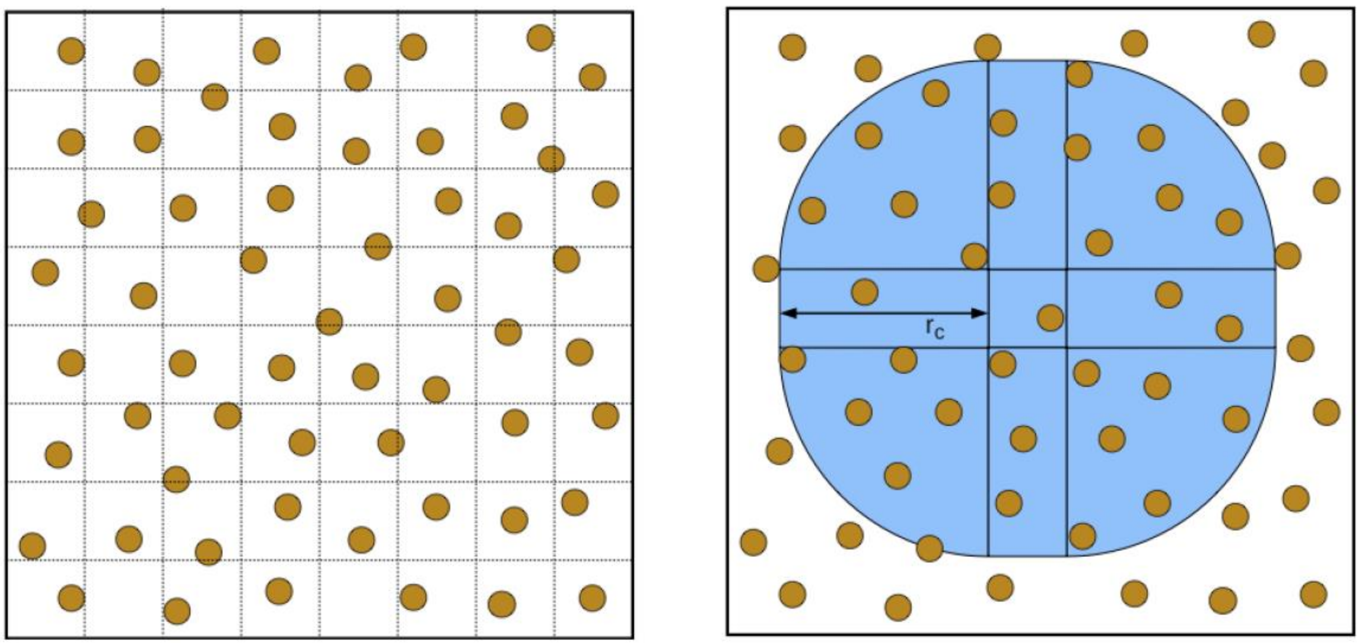


(a) Force matrix.



(b) Force matrix, redundancies removed.

Parallel Decompositions -- Spatial decomposition



(a) Decomposition into 64 cells.   (b) Import region for one cell.

Topic: Parallel Computing/Programming

Reading material

- Introduction to Parallel Computing, Blaise Barney https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial
- An Introduction to Parallel Programming, Peter Pacheco