# Review of basic HPC computing terms/terminology

By
Clement Onime
(onime@ictp.it)
ICTP,
Trieste, Italy

# Contents

- Revision of concepts
- Parallelism
  - What, why and how
- Parallel Computing platforms
  - Multi-cores, many-cores to supercomputers
- Parallel programming paradigm
  - Shared memory - Threads (POSIX/OpenMP)
  - Message Passing

# Books & on-line reference materials

- Books
  - An Introduction to Parallel and Vector Scientific computing by Ronald W. Shonkwiler & Lew Lefton, Cambride texts in applied mathematics, 2006
  - Parallel Programming with MPI by Peter S. Pacheco, Morgan Kaufman Publishers, 1997
- On-line materials
  - Stanford University CS Education Library http://cslibrary.stanford.edu/
  - Lawrence Livermore National Laboratory https://computing.llnl.gov/?set=training&page=index
  - Durham University http://www.dur.ac.uk/resources/its/info/guides/

# REVISION

# Sectional Outline

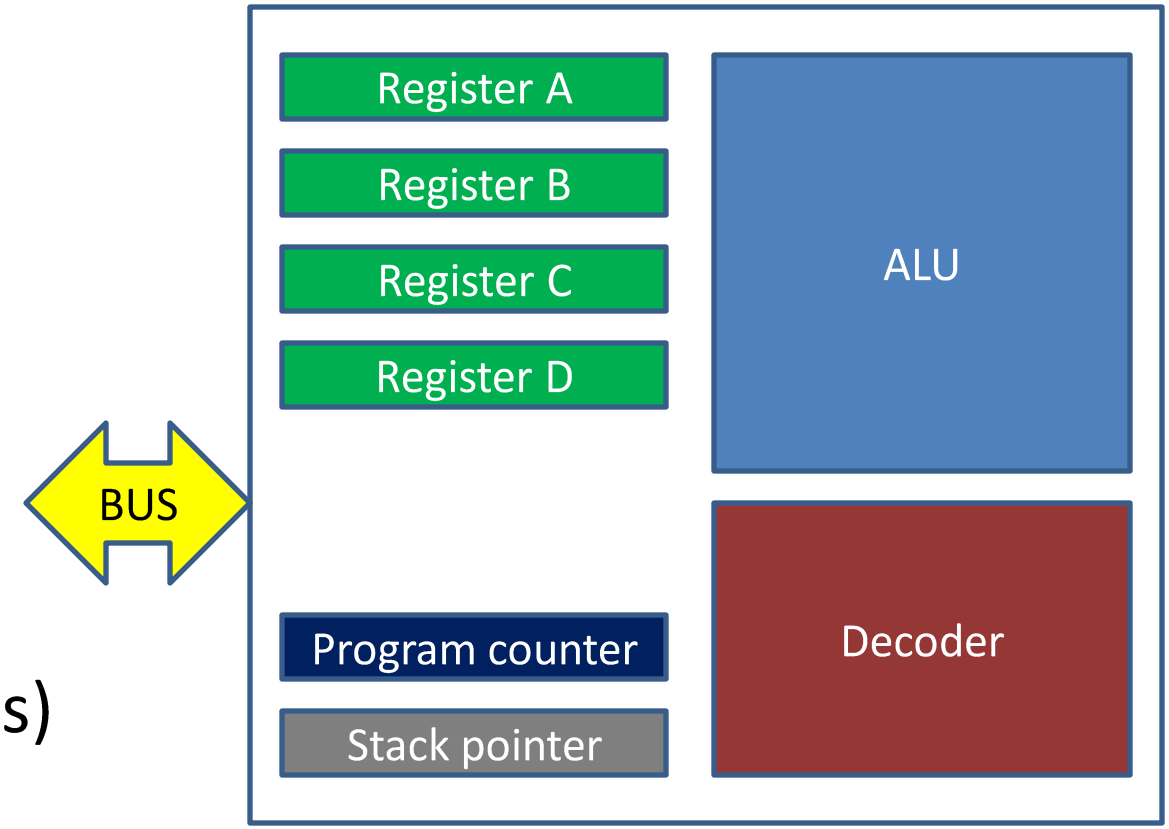
- Computer hardware
- Programming languages
- Tools for programming
- Data types (integers & floats)
- Operators
- Statements
- File operations

# Von Neumann Architecture

- General requirements for electronic computers (1945).
  - Main components: Memory, control unit, Arithmetic Logic Unit (ALU) and input/output unit
  - Classically used for designing all computers/CPUs
- Proposed stored/changeable programs executed sequentially.

# CPU Computer hardware

- Main parts
  - CPU
  - Memory
  - Data bus
  - Clock
- Bottleneck(s)




Overview of CPU

# Programming languages

- Native/executable computer/machine language
  - Assembly code
- Human readable languages
  - Interpreted (single executable with different program file as input)
    - Perl, python or tcl
  - Compiled (transforms readable input into executable code
    - Fortran & C
  - Java is semi compiled into byte-code

# Useful tools

- Creating and modifying program source code
  - Editors (may include syntax highlighting checking)
- Translating source code into machine or byte code
  - Compilers (identify & highlight errors in source code)
- Tracing execution or runtime code for errors
  - Debuggers
- Integrated Development Environments
  - Combine all above tools in one interface.

# Computer Data types - Integers

- Integer
  - Whole numbers, typically signed representations
  - Char (8 bit): -127 < x < 128
  - Short (16 bit): -32768 < x < 32768
  - Int : 16 or 32bits
  - Long (32 bit):  -2 billion < x < +2 billion
  - Long long (64 bits): …
  - Notes:
    - Unsigned types basically double the maximum possible positive value e.g unsigned short is 0 < x < 65536

# Computer Data types - Floats

- Float
  - Numbers having two parts separated by a decimal point
  - Single precision (32 bit, 6 digits after decimal)
  - Double precision (64 bit, 15 digits after decimal)
  - Long double  (even bigger)
  - Notes:
    - Standard is double, float(single) may be used to save memory
    - Always In-exact so equality test should be avoided

# Operators

- Arithmetic operators

  ( / * + - % )

- Relational

  ( == != > < <= >= )

- Assignment

  =

  *What about += and -=*

# Statements

- Executable
- Non-executable
- Control
- Input/output
- Built-in functions
- Functions
- Subroutines
- Logical

- *Classify the following*

  *X = 42*

  *break*

  *If ( )*

  *fopen*

  *printf*

  *Sqrt*

  *x < 5*

  *Char * memcpy()*

  *&&*

  *!*

# File operations

- Input
  - Sometimes can differentiate between text and binary files
  - Sequential or random
- Output
  - Create non existing
  - overwrite or clobber existing
  - Or append
- Fopen, printf, scanf, fgetc, fclose, fputc, ungetc

# Exercise 1

# CONCEPTS OF PARALLELISM

# Section outline

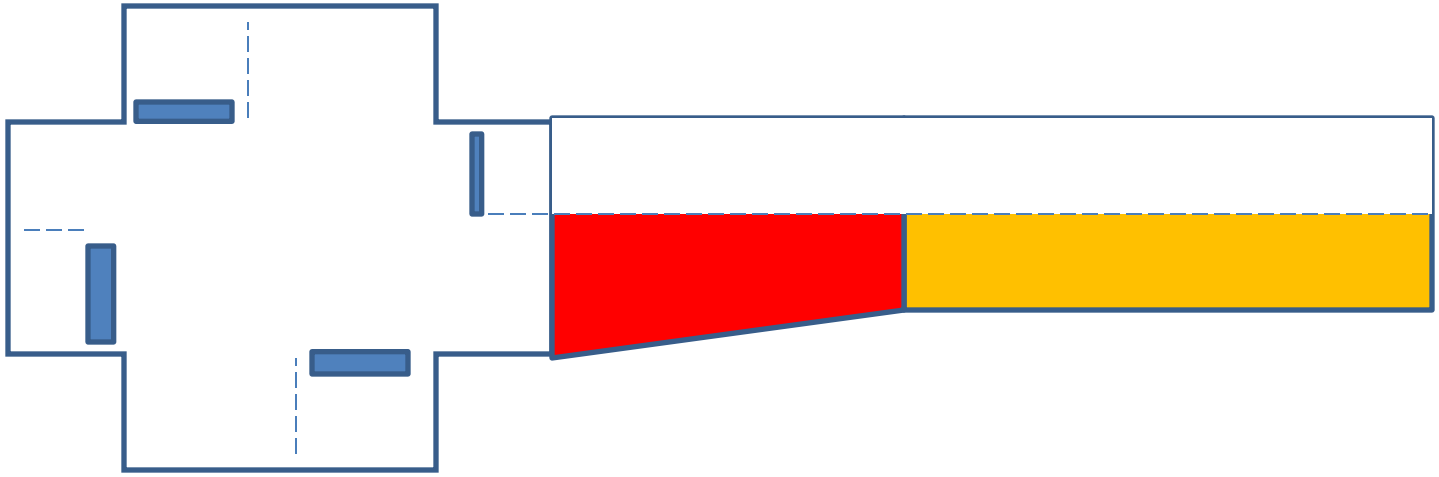
- Parallel computing – a definition
- Why is it important?
- Classification of computers
- Classification by memory architecture
- Programming models

# What is parallel computing

- Serial computing
  - Single program broken into parts, parts broken into single instructions, single instructions run one at a time on single CPU
- Parallel computing
  - Program broken into parts, each part broken into single instructions that maybe run concurrently on different CPU(s)
- *Simultaneous use of multiple compute resources to solve a computational problem*

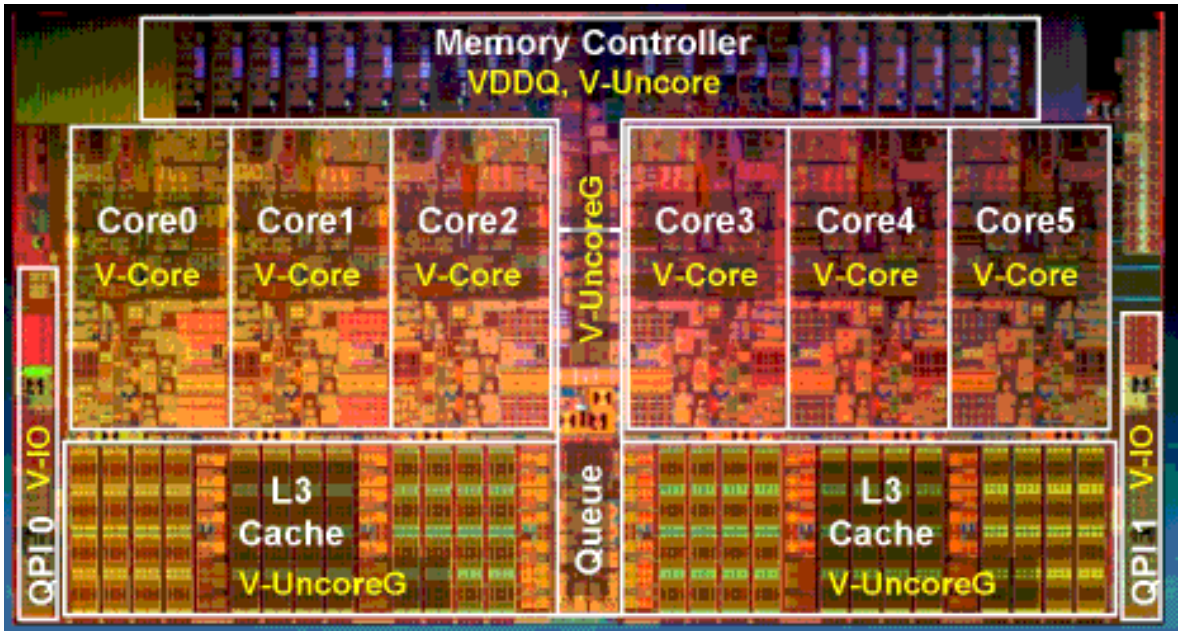
# Why Parallel computing modeling real world

- Useful as it can closely model the real world situations:- Many complex interrelated events happening at the same time within a temporal sequence. E.g: rush hour traffic at big junction, and rain…



# Why Parallel computing

- Only way to make optimal use of new/ evolving generations of CPU processors. E.g: dual core and beyond.

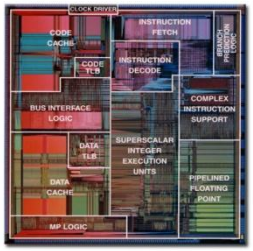


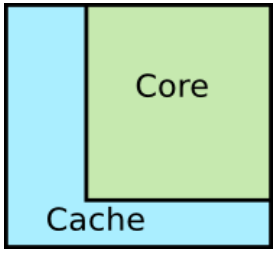

Intel Westmere CPU

# From single- to multi-core

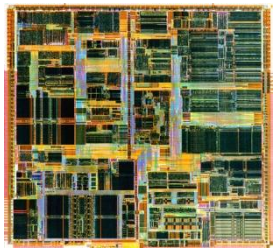
how and why CPU evolved

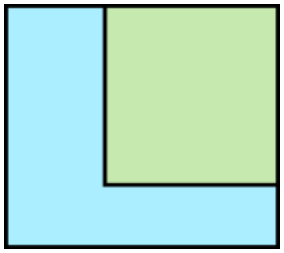
Pentium I



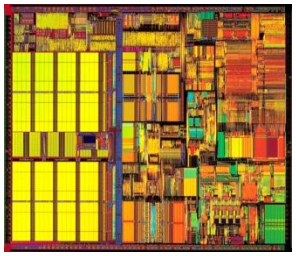



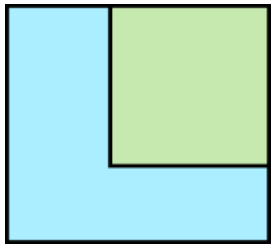

Pentium II



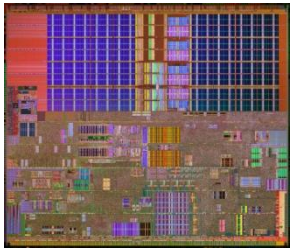


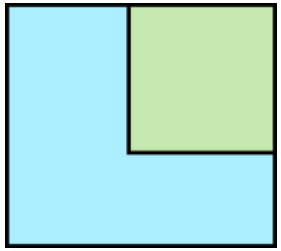
Pentium III



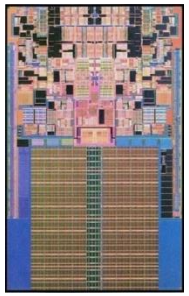


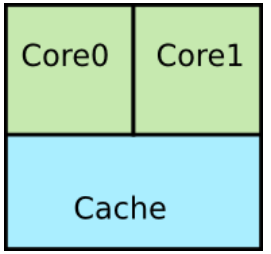
Pentium IV (Hyperthreading)



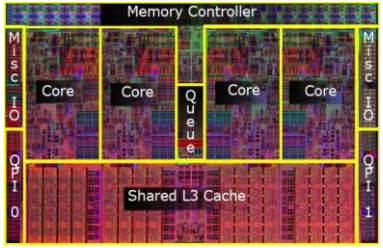


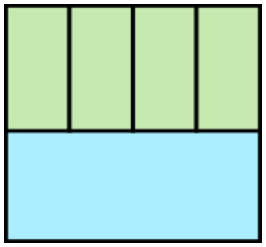
Core 2 (New HyperThreading)



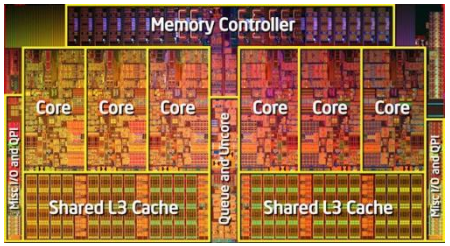


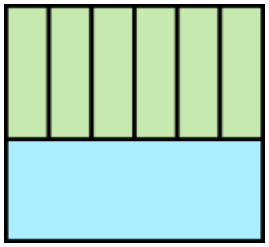

Core i7- 9xx (4 cores, 8 threads)



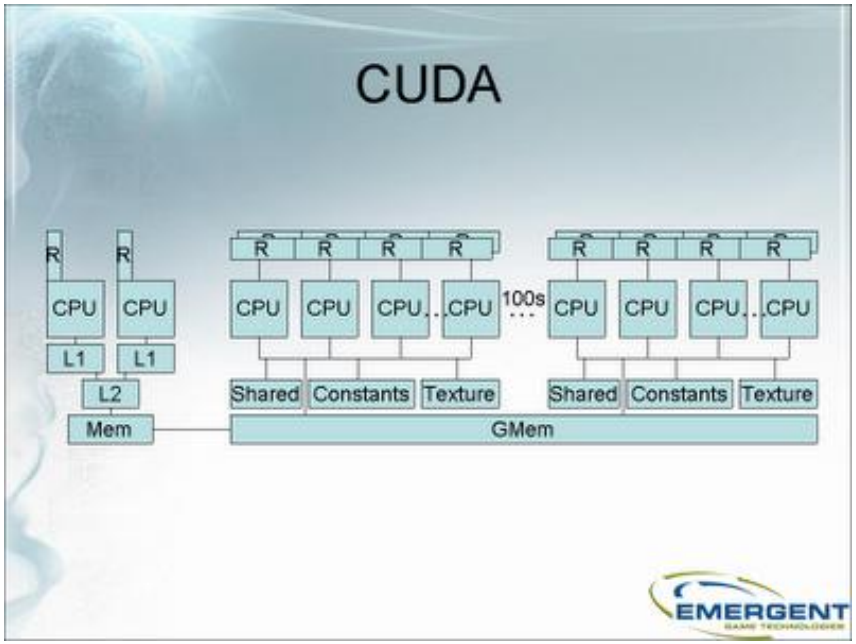



Westmere (6 cores, 12 threads)



# Now many-cores




- Single computer
  - 2 x 8 core CPU
  - 2 x 2483 Nvidia cores
- CUDA
  - Nvidia specific
- OpenCL
  - Same executable software can run on cpu and/or gpu
  - Distributed/networked systems

# Classification of computers

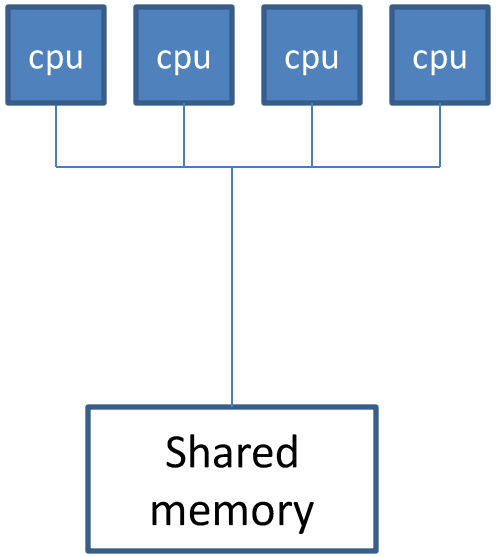
- Flynn Taxonomy from 1966
  - Single instruction, single data (SISD)
    - E.g: Serial computing
  - Single instruction, multiple data (SIMD)
    - E.g: classical GPU & CPUs
  - Multiple instruction, single data (MISD)
    - Few experimental examples
  - Multiple instruction, multiple data (MIMD)
    - E.g clusters, super-computers

# Classification by memory architecture

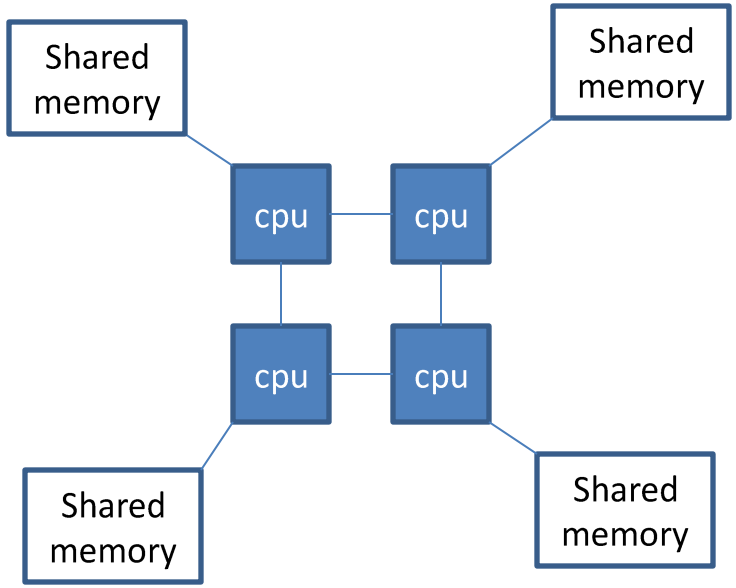
- Shared memory
  - UMA (SMP) & NUMA & ccNUMA
- Distributed memory
  - Interconnect between computers:
    - Gigabit Ethernet ~ 1gbs - 10gbs, Myrinet ~ 10 gbps, Infiniband ~ 10 gbps – 75 gbps
- Hybrid-distributed
  - Shared memory + distributed memory

# Classification by memory architecture

- Shared memory
  - Uniform Memory Access
  - Non Uniform Memory Access




UMA




NUMA

# Example of NUMA Super Computer

## Blacklight (Pittsburgh Supercomputing Center)



SGI UV 1000 ccNUMA

512 x quad-core Intel Xeon 7500 (2.27GHz) processors = 4,096 cores

2 x 16 TB of memory (each accessible as single memory space)

Processors & memory connected through NUMAlink-5
(15GB/s, 1µs latency)

US$ 2.8m

# Shared Memory system

pros &cons

ADVANTAGES:

- Global address space provides a user-friendly programming perspective to memory (such as with the OpenMP API)
- Data sharing between tasks is both fast and uniform due to the proximity of memory to the CPUs

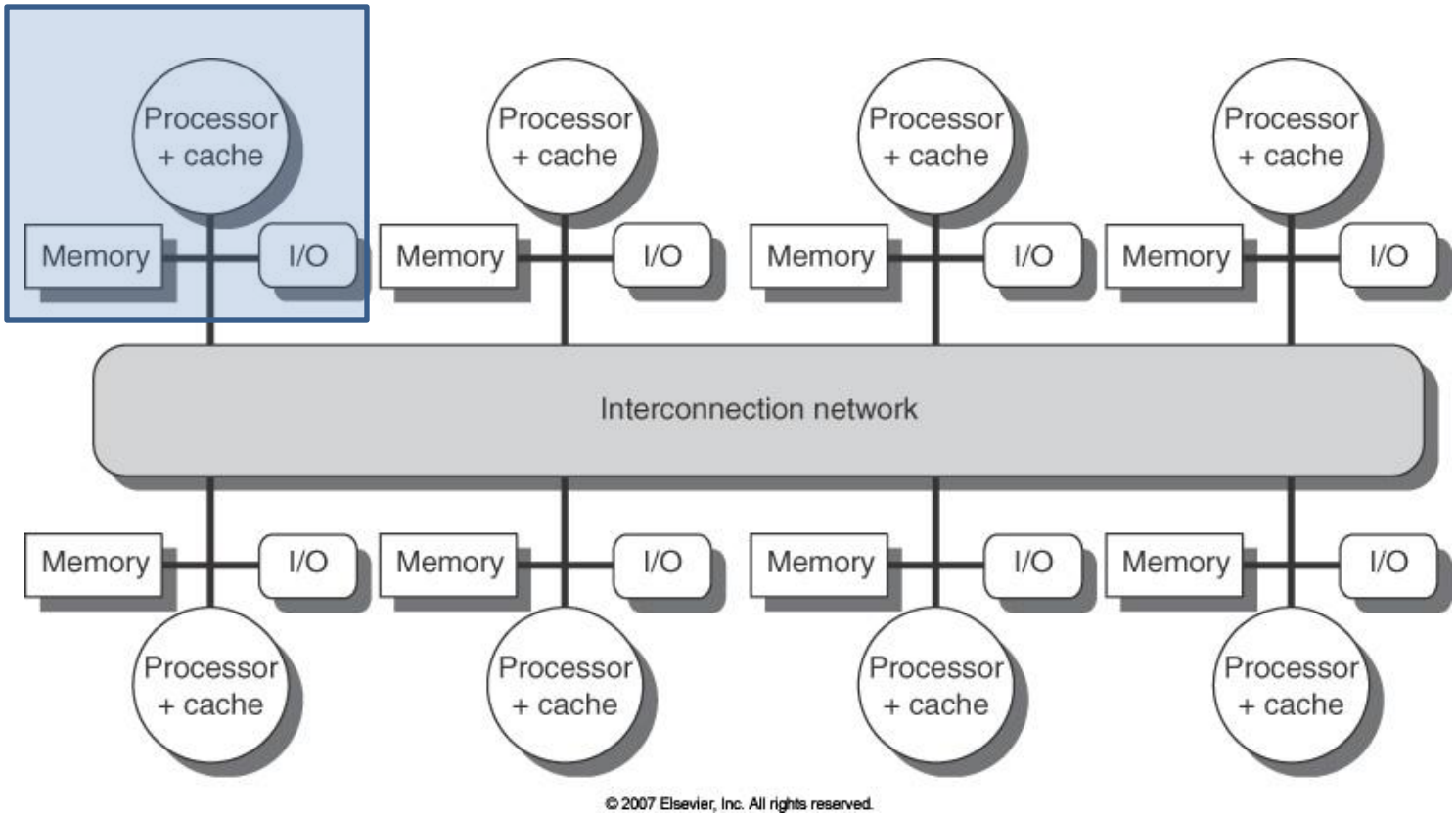
DISADVANTAGES:

- Need for cache-coherency
- Lack of scalability between memory and CPUs. Adding more CPUs can geometrically increase traffic on the shared memory-CPU path, and for cache coherent systems, geometrically increase traffic associated with cache/memory management.
- Programmer responsibility for synchronization constructs that ensure "correct" access of global memory (i.e. prevent race conditions)

*Slide from Introduction to HPC by ICHEC, Ireland*

# Distributed Memory system

schema






*Slide from Introduction to HPC by ICHEC, Ireland*

# Distributed Memory system

interconnection topologies

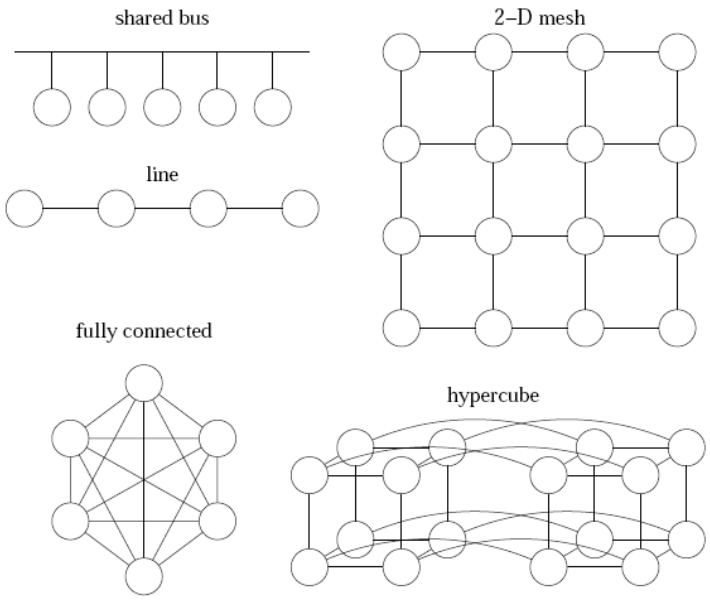



Figure 5.6: Five important interconnect network topologies.

The choice of the interconnection has a STRONG impacts of the communication performance

- Bandwidth
- Latency

| Topology | degree | diameter | avg. dist. | bisection | tot. BW (links) |
|---|---|---|---|---|---|
| Bus | 1 | 1 | 1 | 1 | 1 |
| Line | 2 | $N-1$ | $N/2$ | 1 | $N-1$ |
| 2D Mesh | 4 | $2(\sqrt{N}-1)$ | $2\sqrt{N}/3$ | $\sqrt{N}$ | $2N-2\sqrt{N}$ |
| Hypercube(2-cube) | $\log_2 N$ | $\log_2 N$ | $(\log_2 N)/2$ | $N/2$ | $(N\log_2 N)/2$ |
| Fully Connected | $N-1$ | 1 | 1 | $N^2/4$ | $N(N-1)/2$ |

Table 5.1: Some common interconnect topologies. $N$, number of total nodes.

*Slide from Introduction to HPC by ICHEC, Ireland*

# Distributed Memory system

pros &cons

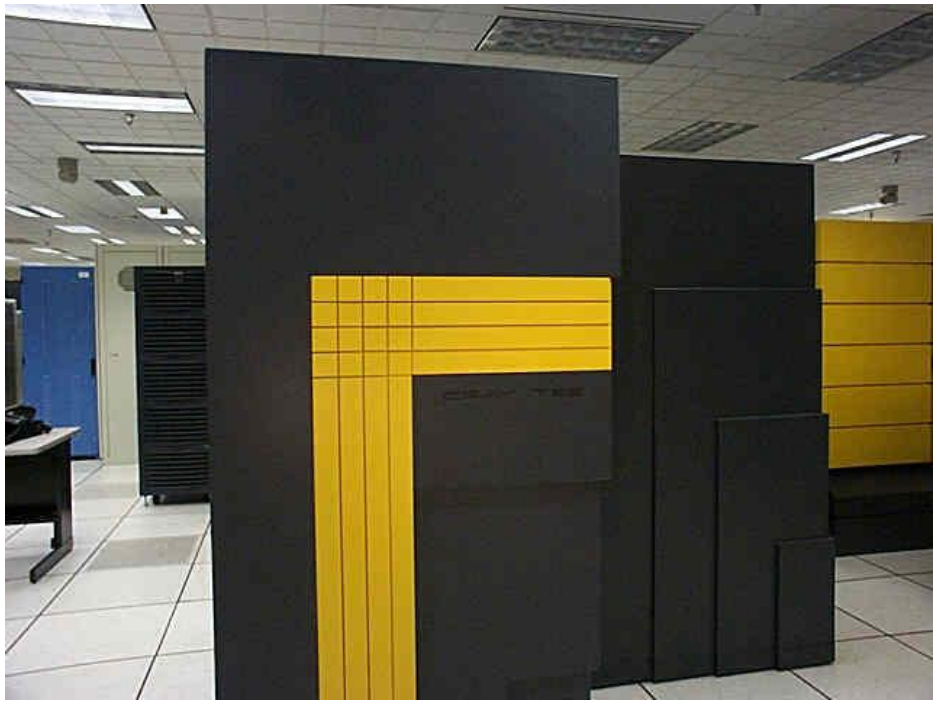
## ADVANTAGES:

- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness: can use commodity, off-the-shelf processors and networking.

## DISADVANTAGES:

- Programmer is responsible for mapping data structures across nodes
- Programmer is responsible for coordinating communication between nodes when remote data is required in a local computation (called message-passing)
- Access to remote memory is significantly slower than to local memory
- Currently, only low-level programming API's (such as MPI) are available to perform message-passing between nodes

# Super Computer Example



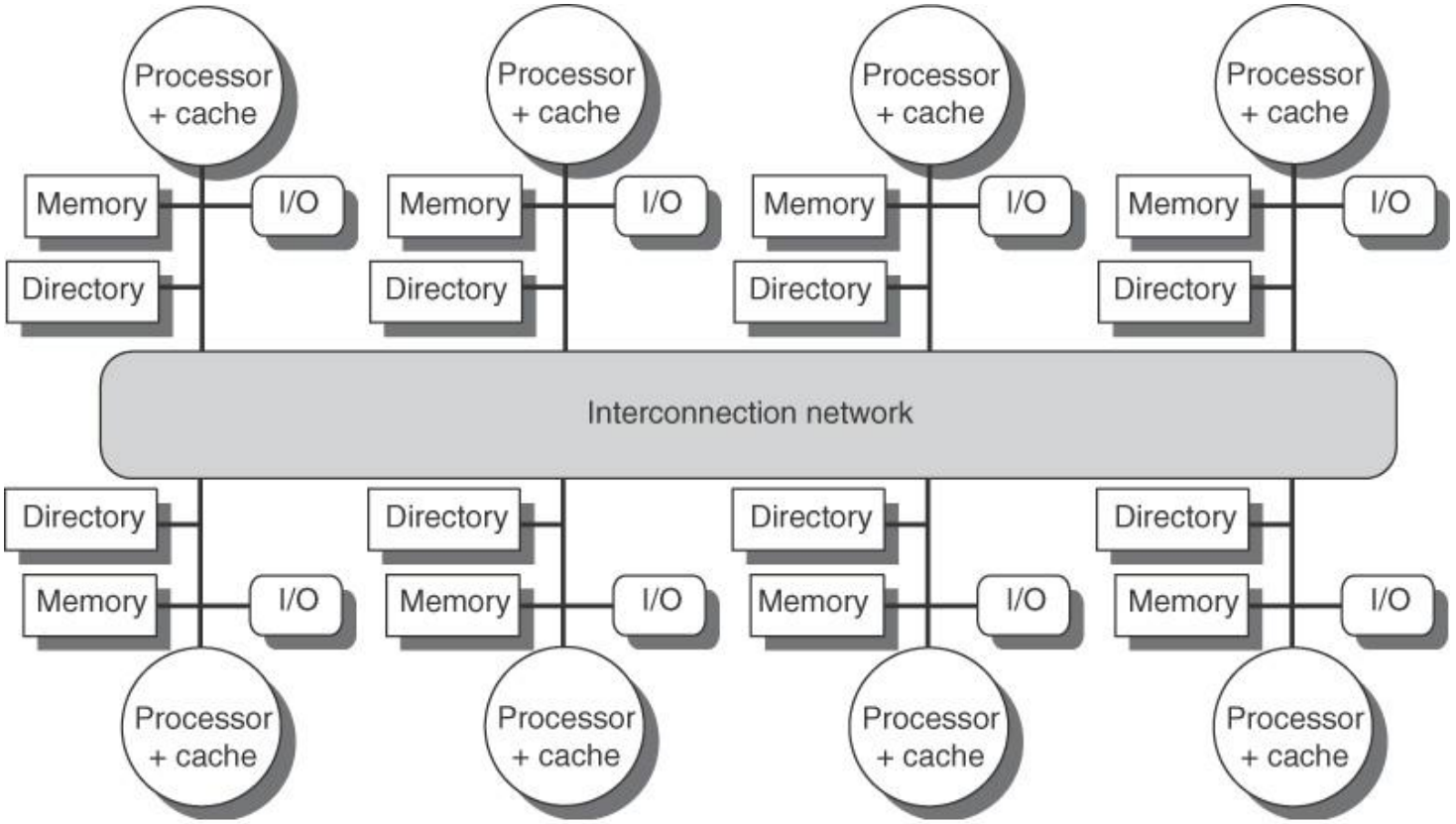
Cray T3E

A fully distributed memory machine designed to scale from 8 to 2176 nodes. Each node had between 64MB and 2GB or local RAM.

A 1480-processor T3E-1200 was the first supercomputer to achieve a performance of more than 1 teraflops running a computational science application, in 1998

# Distributed Shared Memory system

schema



# Distributed Shared Memory system

pros & cons

ADVANTAGES:

- Exploits the advantages of both shared and distributed-memory architectures at different levels.
- Can exploit both shared and distributed-memory programming paradigms (OpenMP and MPI) to solve difficult tasks
- Can be built from commodity processors and interconnects.

DISADVANTAGES:

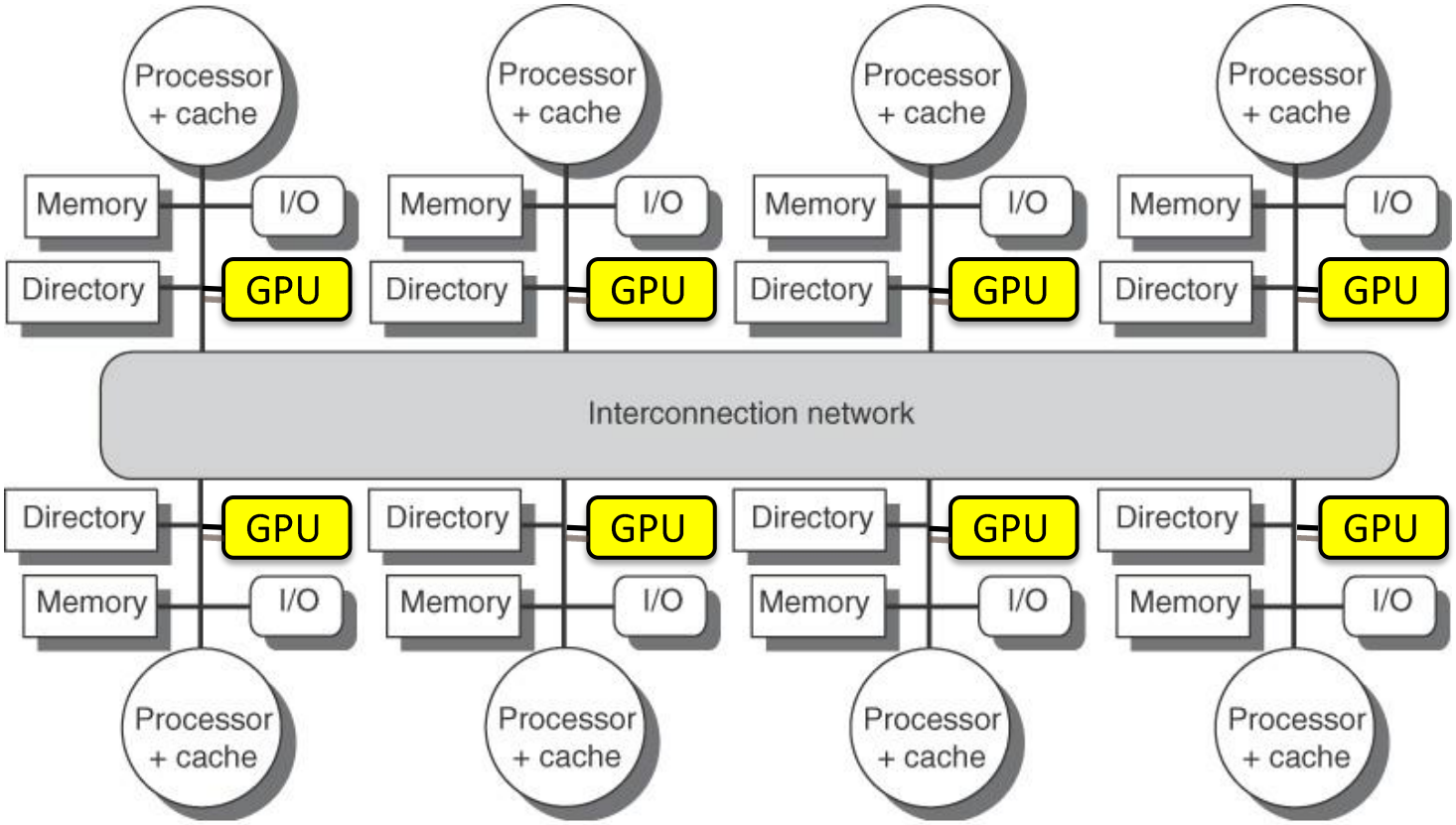
- The same disadvantaged of a pure Shared Memory system
- The same disadvantaged of a simple Distributed system

→ IDEA: explore new strategies like mixing parallelization

*Slide from Introduction to HPC by ICHEC, Ireland*

# A full hybrid system

standard hardware plus accelerators






*Slide from Introduction to HPC by ICHEC, Ireland*

# A full hybrid system
## pros & cons

ADVANTAGES:

- Accelerators (GPUs) can speed-up the calculation up to 100x times!

DISADVANTAGES:

- The same disadvantaged of a pure Shared Memory system;
- The same disadvantaged of a simple Distributed system;
- Accelerators are separated from the system (like a PCI card), they do not share the memory with the host system;
- Accelerators require their own programming environment.

→ A little bit of work for potentially huge advantages. Think about that.

*Slide from Introduction to HPC by ICHEC, Ireland*

# many core Super Computer

“Tianhe-1A"
National Supercomputing Center in Tianjin (China)


- 14,336 x Intel Xeon X5670 hex-core processors.
- 7,168 Nvidia Tesla M2050 GPUs.
- 2,048 NUDT FT1000 (home-grown CPUs).
- Total memory: 262 TB
- Custom proprietary interconnect – Arch – 160 Gbps (2x Infiniband).

4.7 petaflops

# Novel Architectures

"Roadrunner"
DOE/NNSA/LANL
(US)

Hybrid Design:

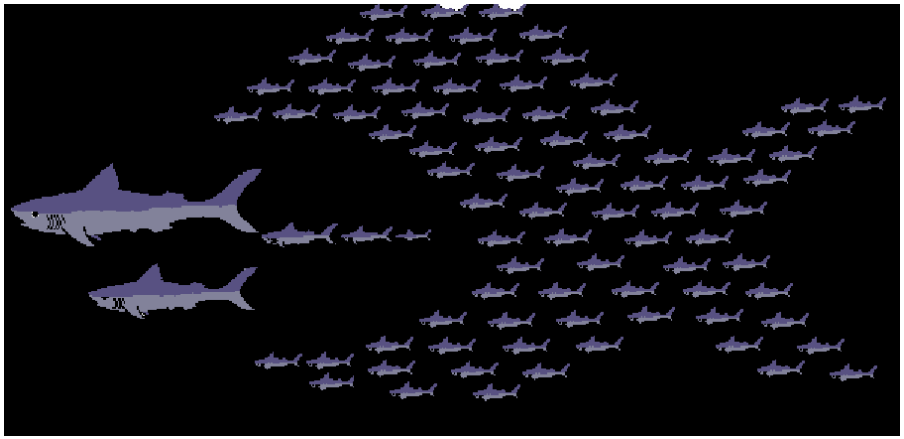
- 12,960 IBM PowerXCell 8i
  - 3.2 GHz
  - 1 general purpose core
  - 8 performance cores
- 6,480 AMD Opteron
  - Dual-core, 1.8 GHz

Node:

- Cell attached to Opteron core

1.37 petaflops

# Clusters

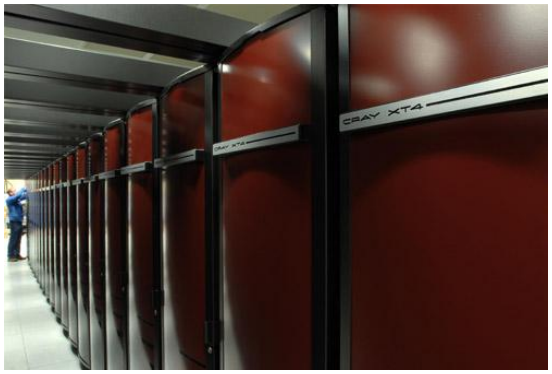


- Many computers (nodes) interconnected by high speed network.
- Commodity clusters
- Hybrid  supports both shared & distributed architecture & programming

# Advantages

- Exploits the advantages of both shared and distributed-memory architectures at different levels.
- Can exploit both shared and distributed-memory programming paradigms (OpenMP and MPI) to solve difficult tasks.
- Can be built from commodity processors and interconnects.

# Examples of Clusters




**HECToR (XT4 component), U.K**

- Cray XT4 supercomputer
  - Quad-core AMD Opteron CPUs
  - 3,072 nodes * 1 CPU * 4 cores
  - 8 GB memory / node
- 12,888 cores and 24.5TB of RAM
- Now only a PART of a bigger HECToR!

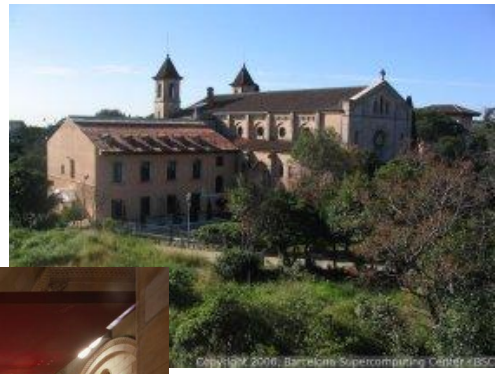

JuRoPA in Germany

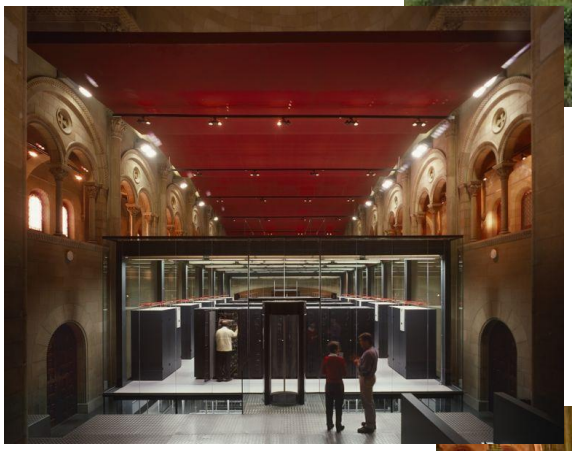
- Combination of Juropa-JSC & HPC-FF
- Intel Xeon X55xx (Nehalem-EP) quad-core processors
- 3,288 nodes * 2 CPUs * 4 cores
- 24 GB memory / node

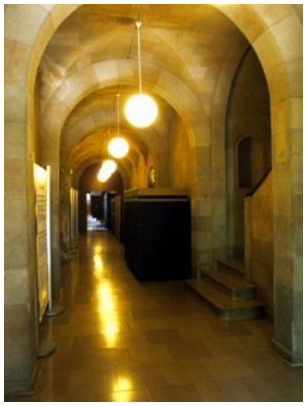
- Total of 26,304 cores, 79 TB main memory

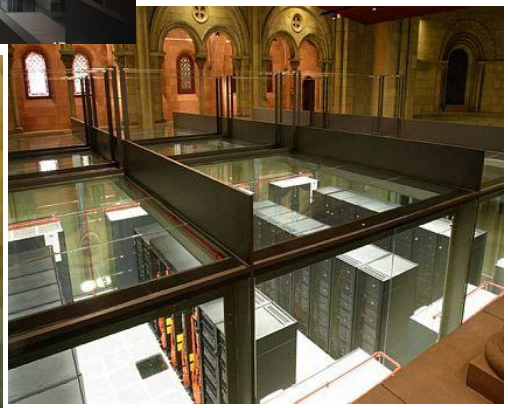
- 308 teraflops peak performance

# Examples









MareNostrum (Spain)

•2,560 computing nodes, each with dual IBM 64-bit PowerPC 970MP processors running at 2.3 GHz, 10,240 CPUs in total.

•It is capable of 63.83 teraflops and a peak performance of 94.21 teraflops.

•It occupies only 120 m² (less than half a basketball court).

•It was largely constructed in two months in Madrid and was installed in the Barcelona Supercomputing Center, Spain.

# BlueGene


JUGENE
Juelicher BlueGene/P
(Germany)

•294,912 cores made up from:
- 72 racks
- Each rack has 32 node cards
- Each node card has 32 compute cards
- Each compute card has 2 chips and 2GB of local memory
- Each chip has 2 cores

•3 parallel communication networks
•1 petaflop peak performance

# Programming parallel computers

- Models
  - Shared Memory (no threads)
  - Threads
  - Distributed memory
  - Data parallel
  - Hybrid model
  - Single Program Multiple Data
  - Multiple program multiple data
- **Programming models are independent of underlying hardware & memory architecture**

# Shared memory (no threads)

- Tasks share common address space
- Read & writes to common address space is asynchronous
- Simplified development but poor data locality
- May require locking and semaphores
- Many different running programs/executables
- Implementations
  - POSIX Shared Memory (shm)
  - Mapped memory (side effect)..

# Shared memory (no threads)

**POSIX shared Memory**

- Well defined standard
- Works across different executables
- Can survive across processes until O.S reboot

**Mapped memory**

- Side effect
- Multiple processes via fork() call
- Mapping must occur before forking.
- Destroyed when all processes exit

# Shared memory (threads)

- Single process with multiple execution paths
- Thread scheduling & execution is by O.S
- Threads may have private data but share common global address space as peers.
- Many threads can execute same subroutine/functions at same time (thread safe).
- Two main standards POSIX & OpenMP
- Portability issues

# Shared memory (threads)

- Thread API
  - Apropos pthreads
- Compiling threads(gcc & icc)
  - Use -pthread option
- Creating threads
  - Use pthread_create
  - Limits on number of threads by O.S
- Exiting
  - Use pthread_exit, pthread_cancel, pthread_join, pthread_detach
- Stacksize could be an issue

# Shared memory (threads)

**POSIX threads**

- May require some parallel coding
- C Language only
- Also known as pthreads
- Requires attention to details

**OpenMP threads**

- Can work in serial code as it is compiler directive based
- Available in C, C++ & Fortran
- Can be used to gradually parallelise a serial code
- Simple and easy to use

# OpenMP

- Code is divided into blocks and parallelisation is in blocks
- API
  - Compiler directives, runtime functions/routines & environmental variables
- Easy as parallelization by addition of just 3 or 4 lines/directives.

# OpenMP

| | Fortran | C/C++ |
|---|---|---|
| Gnu Compiler flags | -fopenmp | -fopenmp |
| Intel compiler flags | -openmp | -openmp |
| Compiler directives | !$OMP PARALLEL | #pragma omp parallel |
| Loop contruct | !$OMP DOSCHEDULE | #pragma  omp for schedule |
| Barrier | !$OMP BARRIER | #pragma omp barrier |
| Critical section | !$OMP ATOMIC | #pragma omp atomic |

# Distributed memory

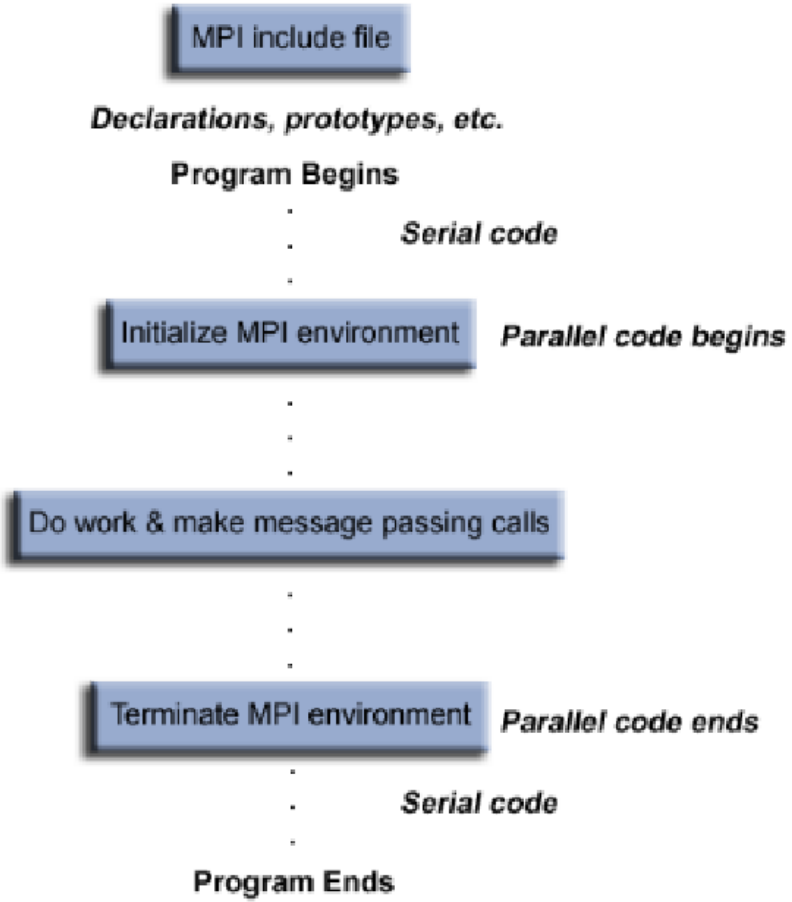
- Message Passing
  - Each task has it's own dedicated memory.
  - Task communicate by sending messages
  - Multiple tasks can run on same machine
  - Data transfer requires coordination between 2 or more tasks
  - Programmer is responsible for all parallelism
  - Implementations: MPICH, LAM & OpenMPI

# OpenMPI

- Available on all HPC/hardware platforms
  - Shared, distributed memory or hybrid

| | Fortran | C/C++ |
|---|---|---|
| Compiler wrapper | mpif77, mpif90 | mpicc, mpic++ |
| Runtime helper | Mpirun | Mpirun |
| Header file | mpif.h | mpi.h |
| Binding | CALL MPI_xxxx | MPI_xxxx |
| | | |

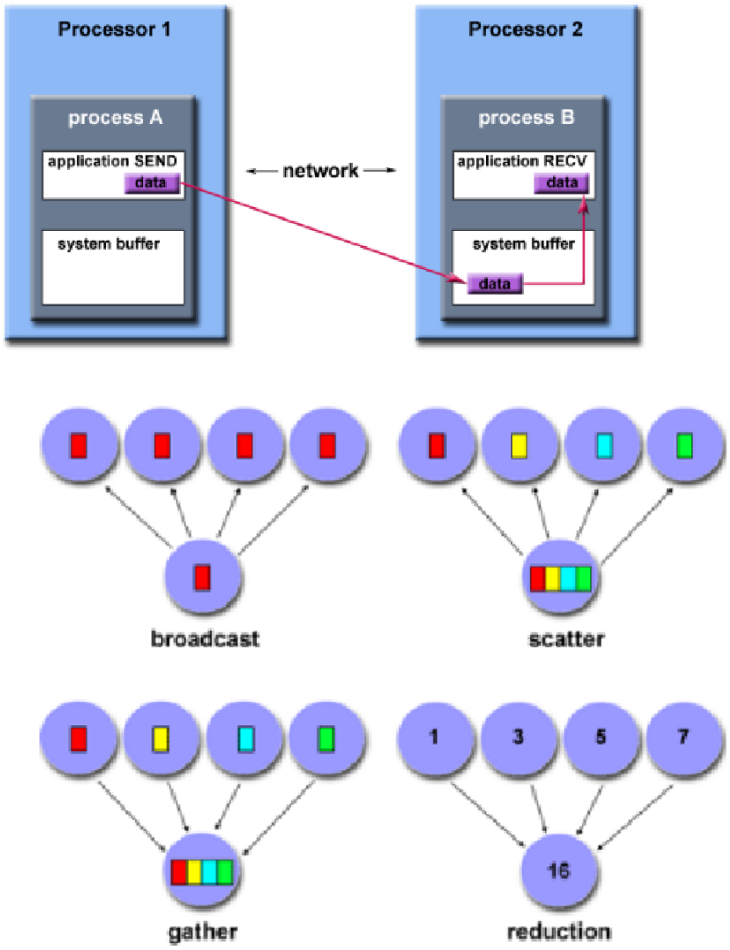
# OpenMPI




- Communicators
  - Global collection
  - Use MPI_COMM_WORLD
- Groups
  - set of processes that may communicate
- Ranks
  - Unique ID of each MPI process (0 to N-1)

# OpenMPI communication

- Point-to-Point
  - Types
    - Synchronous
    - Blocking
    - Non blocking
  - Order & fairness
- Collectives
  - All or none
  - Broadcast useful for data distribution
  - Blocking type



# Data Parallel

- Parallelism is focused around operations on data
- Data is organised in common structure such as array (1, 2 or 3D)
- Tasks work collectively on same data structure but each task has a different range or portion.
- All tasks perform same operation on data.
- Can be carried on shared or distributed memory architecture systems

# Data parallel implementation

- Program usually have to be written from the ground up as data parallel.
- Requires a data parallel compiler(or library)
  - Included in Fortran 95.
- Compiler directives used to specify distribution and alignment of data

# Hybrid Model

- Combines one or more of the previously described programming models
- Common is OpenMP + MPI
  - Openmp for thread computation using local on node data
- Also GPU + MPI
  - CUDA/OpenCL for computation on local intra-node data
- Good for multi-core/many cores..

# Single Program Multiple Data

- SPMD
  - single executable all starting together but each running different tasks or with different data
  - May include logic for different executables to only process individual tasks and not full program
  - Individual tasks maybe implemented using different parallel programming model.

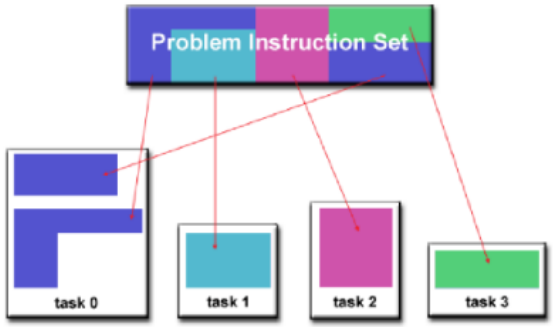
# Multiple Program Multiple Data

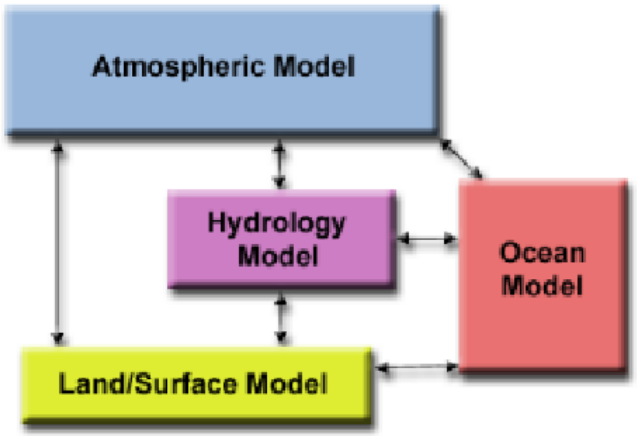
- MPMD is similar  to SPMD
  - Running tasks are different executables running at the same time
  - More useful for functional decomposition

# Designing a parallel program

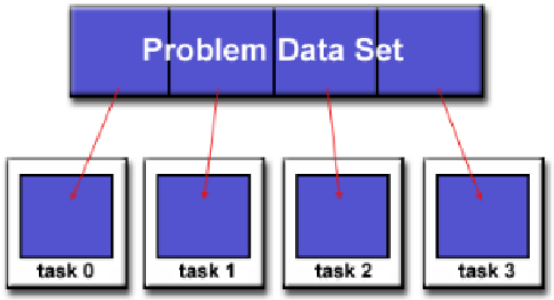
**Step 1: Decomposition**

- Breaking the problem into tasks (discrete chunks of work)
  - Functional decomposition is based on tasks to be done.
  - Domain decomposition is based on data partitioning

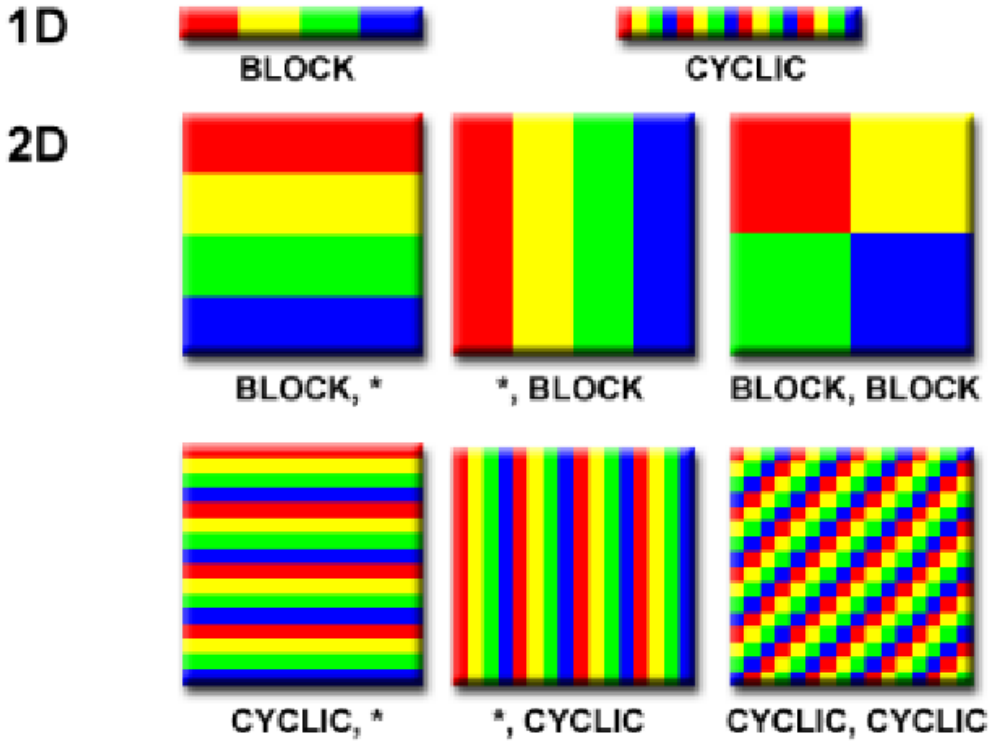


# Domain decomposition



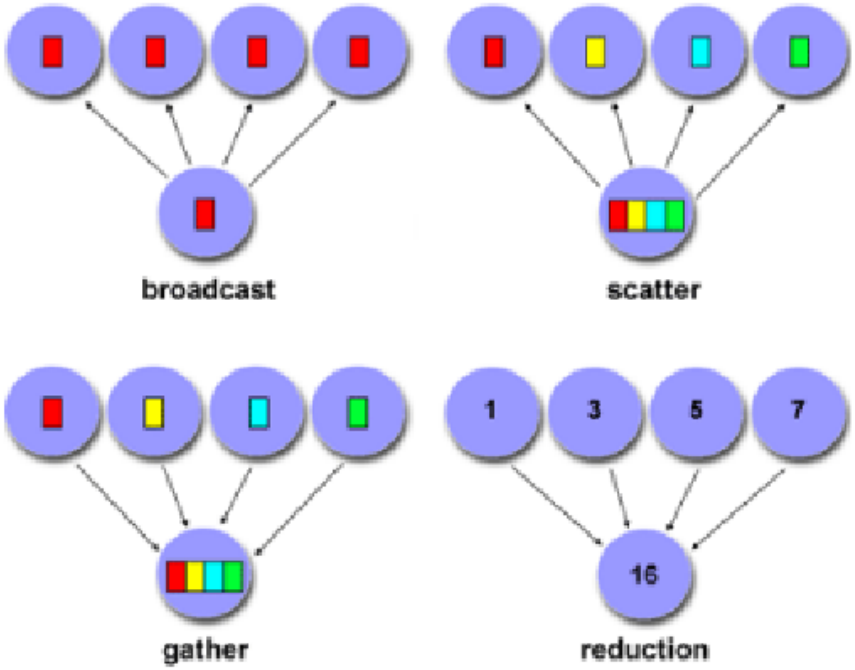

- There are different ways to partition data:



# Designing a parallel program

**STEP 2: Communication**

- Is communication required?
  - Embarrassingly parallel (no communication needed)
  - If yes then consider
    - cost, synchronous, latency/ bandwidth
    - scope



# Scalability & Amdahl's Lam

- Speedup = 1/(1-P)

- Speedup = 1/((P/N)+S)
  - *P = parallel fraction*
  - *N = no of processors*
  - *S = serial fraction*
- Scalability
  - Some problems do improve with increasing dimension.