Crash course for the computational scientists

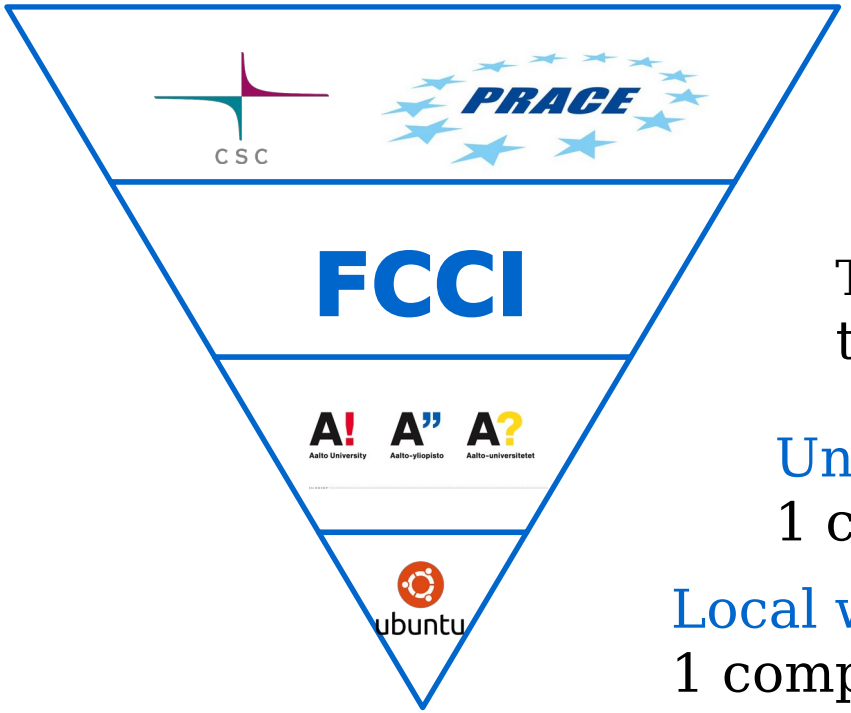
Summer Kickstart 2021

Ivan Degtyarenko, Simppa Äkäslompolo

Aalto University

June 7-9, 2021

Computing resources inverted pyramid



Tier-0 and -1: Supercomputing
massive CPU/GPU; storage

Tier-2: University clusters
thousands of CPUs; GPU; storage

University & department server(s)
1 computer, 8-12 processors

Local workstation
1 computer, 4-8 processors

Why supercomputing?

... why think of something that is outside of your desktop?

Primarily increase the problem scale, secondarily decrease the job runtime

Community built around the HPC resource: documentation, dedicated support, large number of user applications, homogeneous environment, ability to share project files

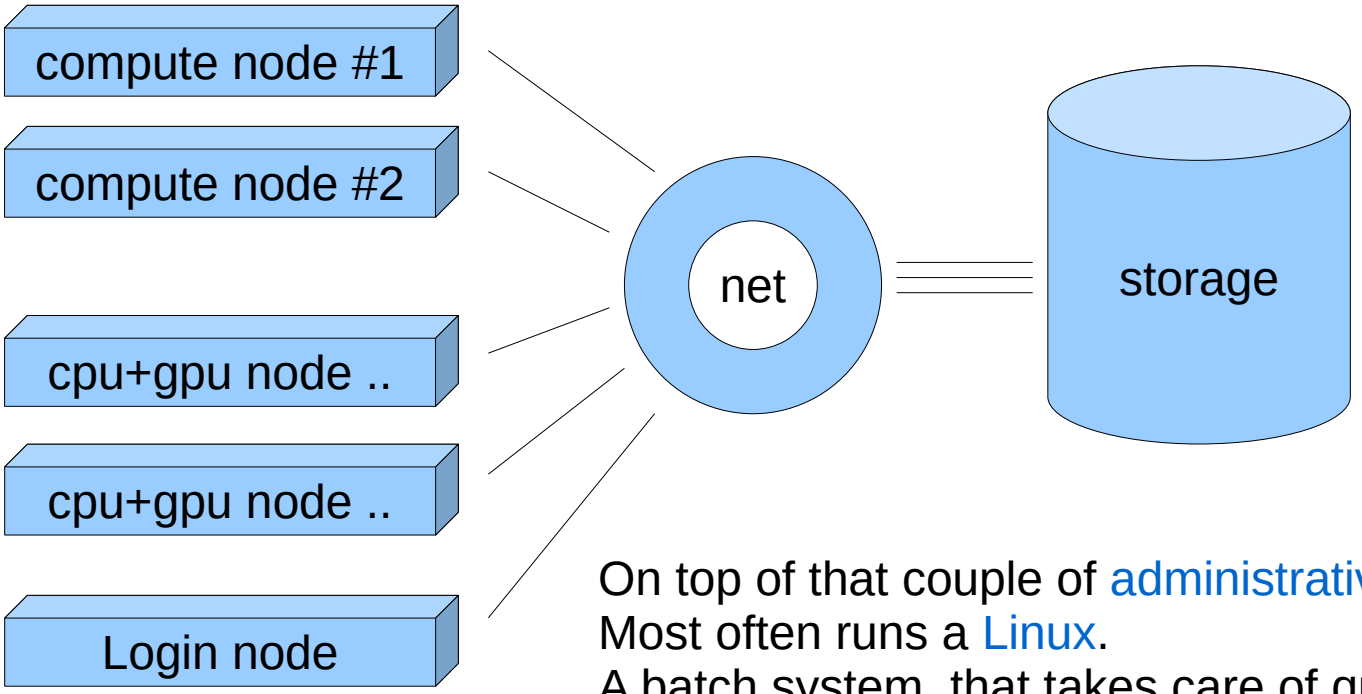
Hardware reasons:

- expensive and fast GPU cards, multiple
- multiple number of CPU sockets: not limited to *a single motherboard* capabilities
- large storage and fast interconnect

Supercomputing is often the only way to achieve specific computational goals at a given time

An HPC cluster

Bunch of powerful computers connected through fast network.
Some have add-on cards, co-processors, for acceleration
Common cross-mounted on all the nodes storage system.



On top of that couple of administrative servers
Most often runs a Linux.
A batch system, that takes care of queuing.

Reference: Computing Thesaurus

- Parallel Computing: means using more than one processing unit to solve a computational problem
- Embarrassingly Parallel: solving many similar, but independent, tasks; e.g. parameter sweeps. Often associated with Grid or Condor computing or array jobs on the cluster
- Distributed (Grid, HTCondor) Computing: solving a task by simultaneous use of multiple isolated, often physically distributed heterogeneous computer systems connected with a specific middleware like HTCondor or ARC
- High Performance Computing (HPC) or Supercomputing: use of the fastest and the biggest machines with fast interconnects and large storage capabilities to solve large computational problems
- GPU computing (or other accelerators): use of GPU cards (or other accelerators) to solve computational problems
- Computational cluster: machine for computing comprised of two or more nodes linked by interconnect
- Compute node: computational unit, has CPU, memory, accelerators
- Storage: at HPC a fast and reliable standalone device connected through cluster interconnect that stores data
- Interconnect: communication links between the compute nodes (Ethernet, Infiniband, etc.)

Real HPC example: K computer

RIKEN AICS K computer (June 2011 – August 2019)

- eight-core 64-bit Sparc VIIIfx, 2.0 GHz
- 11.28 PFlop/s peak, 10.51 PFlop/s Linpack
- 1.41 PByte memory; 705,024 CPU cores
- 864 racks; 88,128 nodes
- 6D interconnect (Tofu)
- Water cooling

Reference: flop/s

FLOPS (or flops or flop/s) – FLoating point OPerationS, a measure of a computer's performance

… a product of number of cores, cycles per second each core runs at, and number of double-precision (64 bit) FLOPS each core performs per cycle (depending on CPU could be a factor of 4/8/16/32)

Theoretical peak-performance of Triton, as of spring 2020 ~ 1.29 Pflop/s (CPU ~435 Tflop/s; GPU ~855 Tflop/s). LINPACK on full cluster never performed.

Top 10 of top500 are hundreds of petaflops, i.e. they perform 10^{17} floating point operations per second; your calculator is >10 flop/s, your quad core PC about 100-200 Gflop/s, single 40 cores Triton compute node is 3 Tflop/s, Nvidia Tesla V100 GPU computing processors around 7 Tflop/s

RIKEN AICS K computer (cont.)

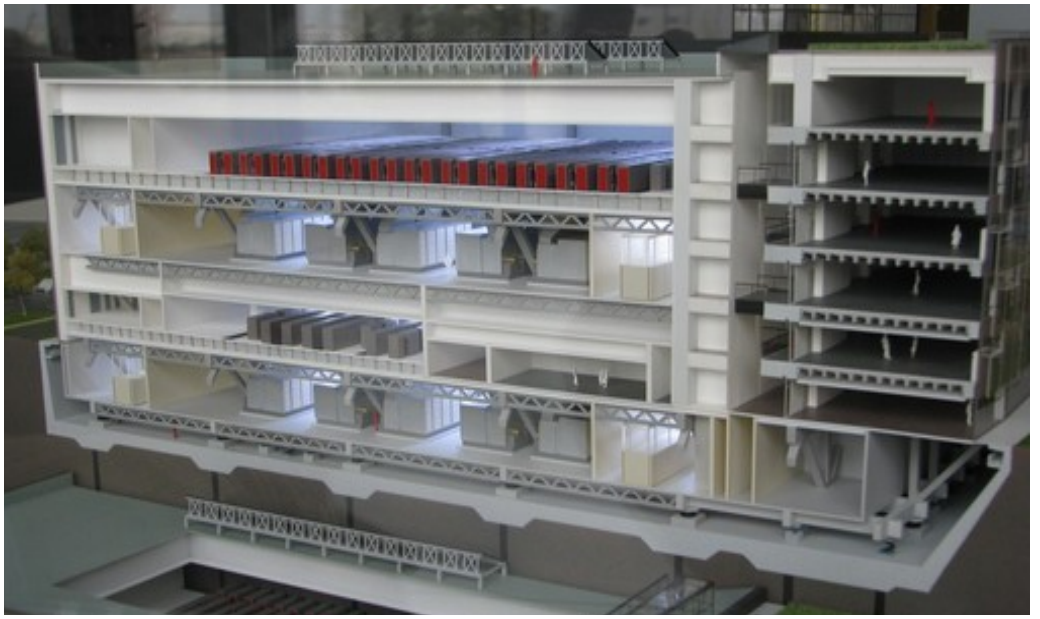
The computer with its own train station

4th floor: K computer

3rd floor: Computer cooling

2nd floor: Disks

1st floor: Disk cooling



K computer building Earth quake security

horizontal moves damper

Vertical moves damper

“wiggle” moves damper

flexible pipes

Local Examples

FGCI sites:

- **Aalto (Science IT project)**: Triton cluster with the Lustre storage http://scicomp.aalto.fi/triton
- **UH (IT for Science)**: Combination of Kale, Ukko2, Vorna, Carrington clusters with the Lustre storage; https://wiki.helsinki.fi/display/it4sci/Resources+for+Research
- **TUNI**: Narvi cluster https://wiki.eduuni.fi/display/tutsgn/TUT+Narvi+Cluster

CSC: Puhti – Atos Linux cluster and Mahti – AMD Rome supercomputer; data storage system Allas; LUMI...

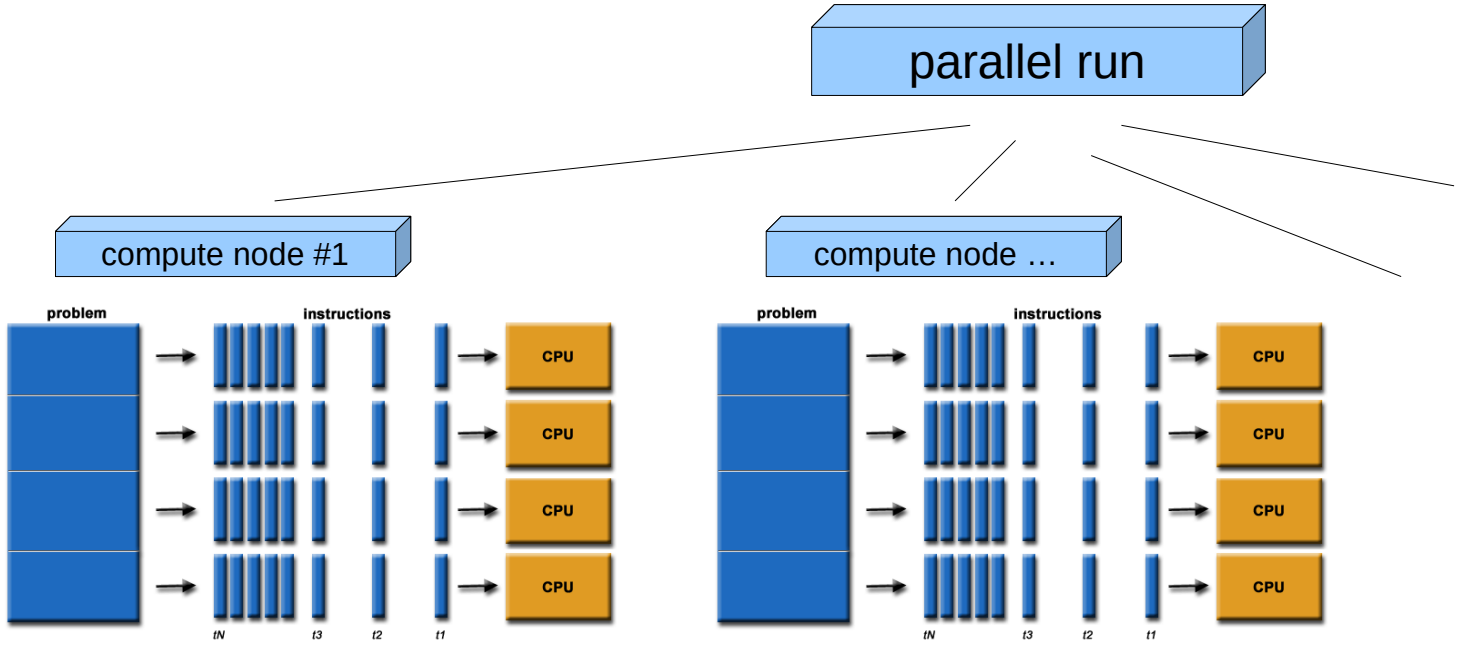
https://research.csc.fi/computing

What they can be used for?

- Large memory calculation: TB(s) of memory
- GPU computing
- Massive serial runs: thousands of runs as one job
- IO intensive runs with high storage requirements
- Storage capacities (TBs of disk space)
- Parallel runs (on more than one CPU)

Running in parallel on CPUs

Means using more than one processing unit to solve a computational problem. A problem is broken into parts that can be solved in parallel, each part is further broken down to a series of instructions, instructions from each part execute simultaneously on one node (or several different nodes) on different CPUs



Accelerators

- Something that makes your calculations run faster
- Common trend in the HPC world: computational nodes equipped with the Nvidia Tesla or AMD Radeon Instinct GPU cards
  - GPGPU stands for *General-Purpose computation on Graphics Processing Units*, a many core co-processor.
  - Initially external add-on cards (PCIe, NVLink)

- Massively parallel – thousands of cores (= thousand of threads are executed concurrently)
- High peak performance and performance per watt (one GPU as power as 10 x cpu-nodes)
- Fast memories high latency, high bandwidth device memory

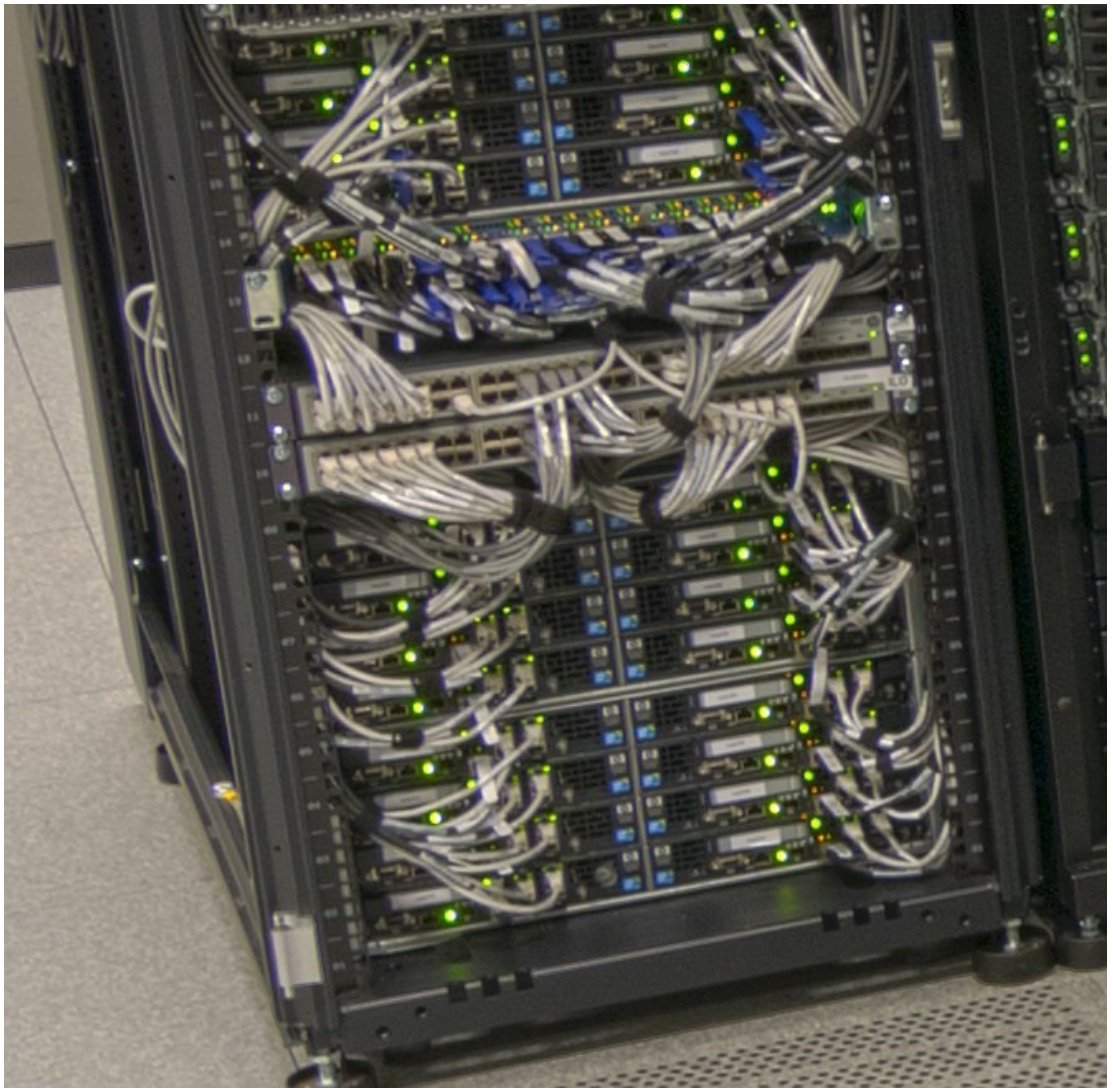
How to use GPU cards

- Tons of the existing software with the GPU support enabled:
  - Matlab, TensorFlow, PyTorch, Julia, … check your app
- Nvidia/AMD GPU-accelerated libraries:
  - https://developer.nvidia.com/gpu-accelerated-libraries
  - https://www.amd.com/en/graphics/servers-solutions-rocm-hpc
- Native GPU code: CUDA, HIP
  - Requires time for programming & debugging but brings most control and code performance
  - Can be combined with GPU libraries
- OpenCL / OpenMP frameworks supports both Nvidia and AMD GPUs

Interconnect

Every single node has at least one network connection, often two-three. Network backbone has hundreds of links of different type. While often there is only one uplink to outside world on the front-end.

The networking is the most critical component #2

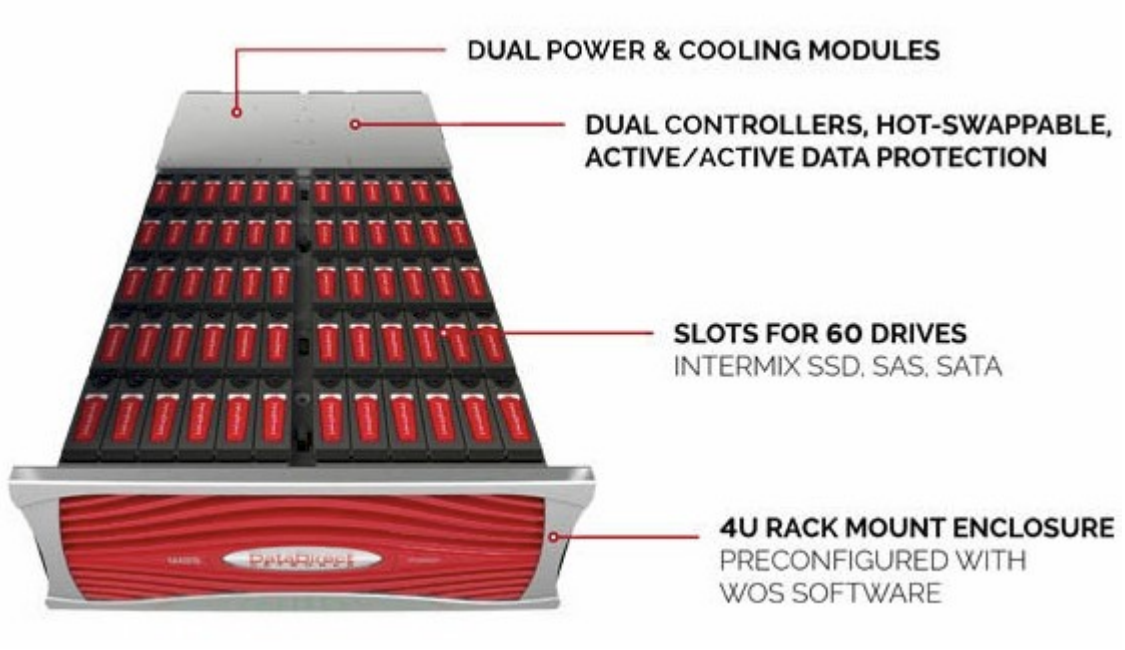


Interconnect

- Link in between the compute nodes
- Storage connectivity
- Distributed memory model, main characteristics latency and bandwidth
- Major players InfiniBand and Ethernet

	EDR IB	100GbE	1GbE
Throughput	100 Gb/s	100 Gb/s	1 Gb/s
Latency	0.6 us	>0.5 us	50-125 us

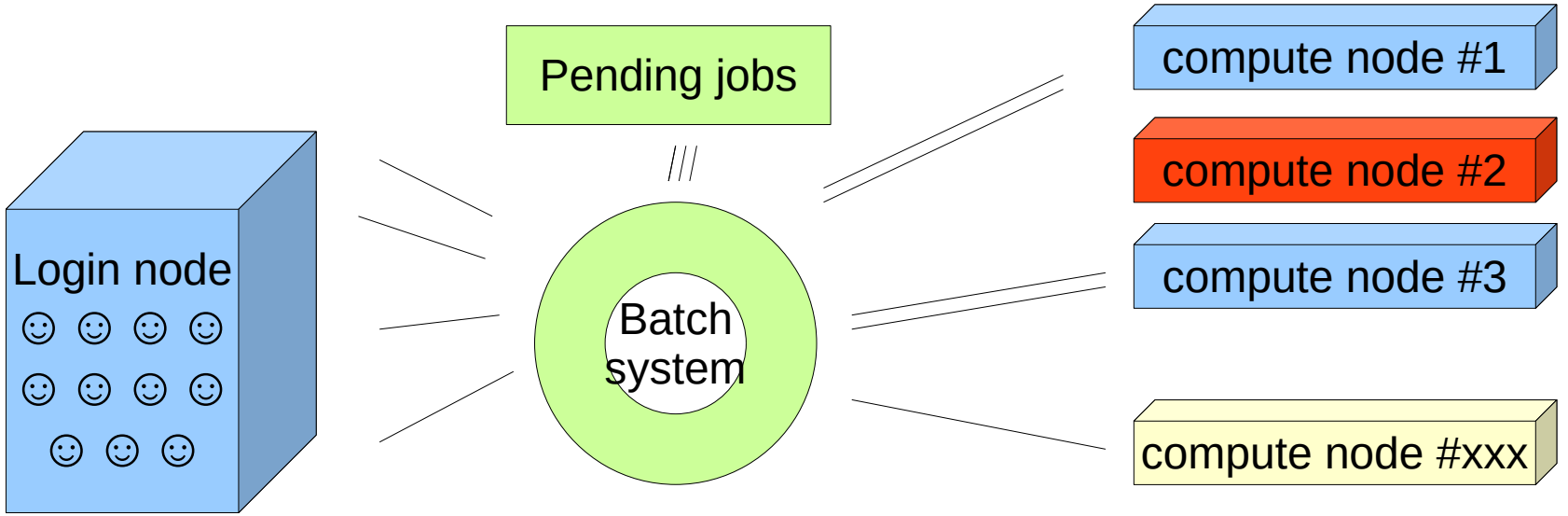
Storage



- **Storage – is the most critical components of any cluster**
- Hardware stack must provide reliability, speed, high throughput for hundreds of network clients, massive IOPs, ability to handle large volumes of data quickly
- **A parallel filesystem**

Managing users' jobs

A scheduler picks up jobs from the users, monitor available resources, decides which one to run and on which node



The rule of thumb: batch system is the "must use" system in order to run on a cluster.
Examples: SLURM (60% of TOP500), PBS, Grid Engine etc
Each system has a set of commands to manage jobs, monitor resources etc

Software

- Usually, cluster has a shared space for the pre-installed software, that includes: libraries, development kits like compilers and debuggers, profiling tools and actual applications.
- Most often managed with a tool that simplifies user environment setup, takes care of dependencies etc
- One of the examples “Environment Modules”, the first thing to try is `module avail` to see what you have got to run
- Good set of software with ready to copy/paste batch scripts is one of the keys for the effective cluster usage

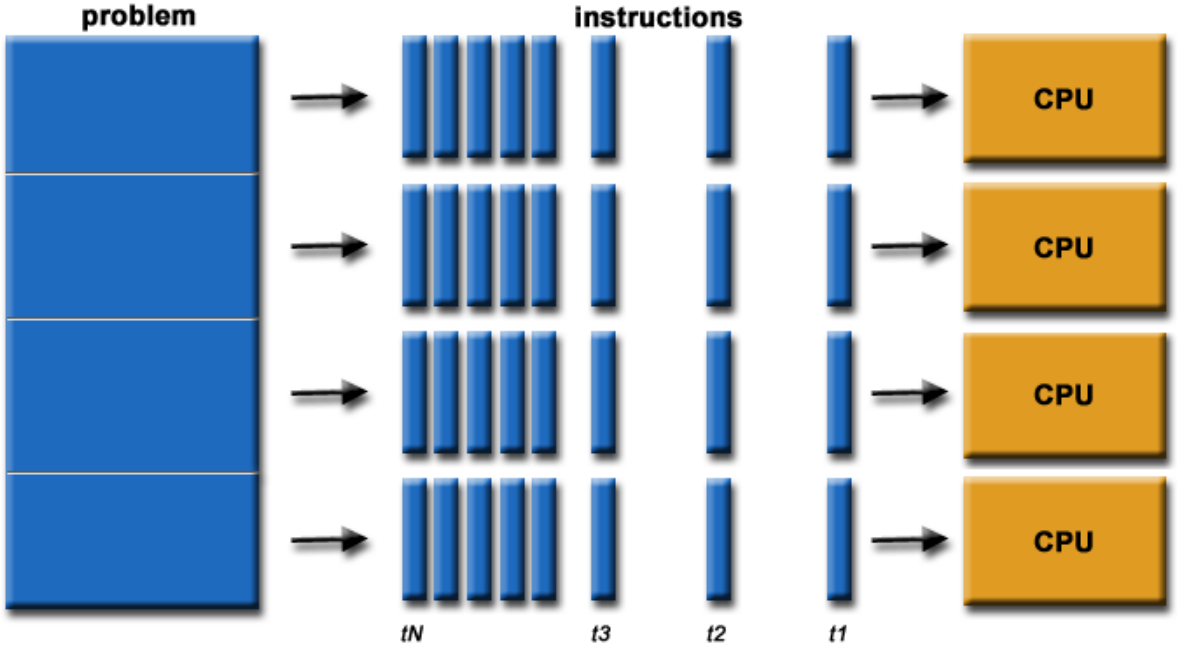
As a summary

The key components to play HPC game

- Infrastructure: space, cooling, networking
- Computational resources: CPU cores, memory, accelerators
- Storage
- Software: OS, batch system, dev tools, libraries, applications
- Support: tech staff, RSE, education, secured fundings

What is a parallel computing?

Means using more than one processing unit to solve a computational problem. A problem is broken into parts that can be solved in parallel, each part is further broken down to a series of instructions, instructions from each part execute simultaneously on different CPUs



Parallelization

No program can run more quickly than the longest chain of dependent calculations

Scalability – refers to ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors.

Factors that impact are hardware, algorithm, implementation, software stack.

```
1: function Dep(a,b)
2:     c := a·b
3:     d := 2·c
4: end function
```

```
1: function NoDep(a,b)
2:       c := a·b
3:       d := 2·b
4:       e := a+b
5: end function
```

Parallel Computer Architectures

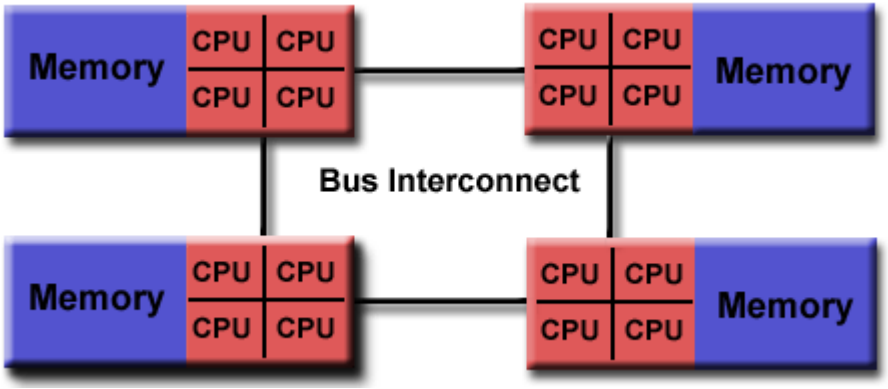
From the CPU point of view a common way to classify parallel computers is to distinguish them by the way how processors can access the system's main memory

- Shared memory
- Distributed memory
- Hybrid

On top of CPU comes computing on co-processors: GPU cards.

A programming model is always an abstraction on top of the available hardware.

Shared memory



Computing within one node

General Characteristics:

- ability for all processors to access all memory as global address space
- multiple processors can operate independently but share the same memory resources
- changes in a memory location effected by one processor are visible to all other processors.

Shared memory: pros and cons

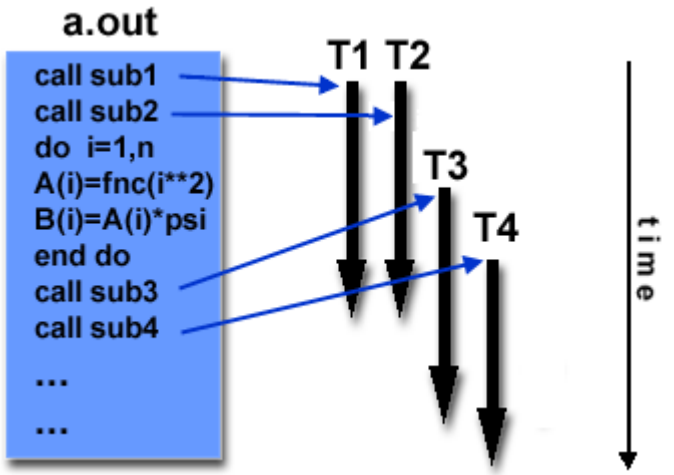
Advantages:

- Global address space provides a user-friendly programming perspective to memory
- Data sharing between tasks is both fast and uniform due to the proximity of memory to CPUs

Disadvantages:

- Primary disadvantage is the lack of scalability between memory and CPUs. Adding more CPUs can geometrically increases 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.
- Expense: it becomes increasingly difficult and expensive to design and produce shared memory machines with ever increasing numbers of processors

Programming/running on a shared memory system



- The main program `a.out` is scheduled normally as any sequential code, it performs some serial work, and then creates a number of tasks (threads) that can be scheduled and run by the operating system concurrently
- Each thread has local data, but also, shares the entire resources of `a.out` (memory, IO)

- Any thread can execute any part of program at the same time as other threads.
- Threads communicate with each other through global memory (updating address locations). This requires synchronization constructs to ensure that more than one thread is not updating the same global address at any time.
- Threads can come and go, but `a.out` process remains present to provide the necessary shared resources until the application has completed.

Reference: Pi as an example

It computes the value of π by numerical integration:

$$\pi = \int_0^1 f(x)\,dx, \quad with \quad f(x) = \frac{4}{1+x^2}$$

This integral can be approximated numerically by the midpoint rule

$$\pi = \frac{1}{n}\int_1^n f(x_i), \quad with \quad x_i = \frac{(i-0.5)}{n} \quad for \quad i = 1,\ldots,n$$

The larger *n*, gives more accurate approximation of π.

Practical implementations

- Threads' implementations commonly comprise:
  - A library of subroutines that are called from within the code
  - A set of compiler directives embedded in the code
- The programmer is responsible for determining all parallelism
- The most common implementation OpenMP
- The OpenMP code is a regular serial code with directives

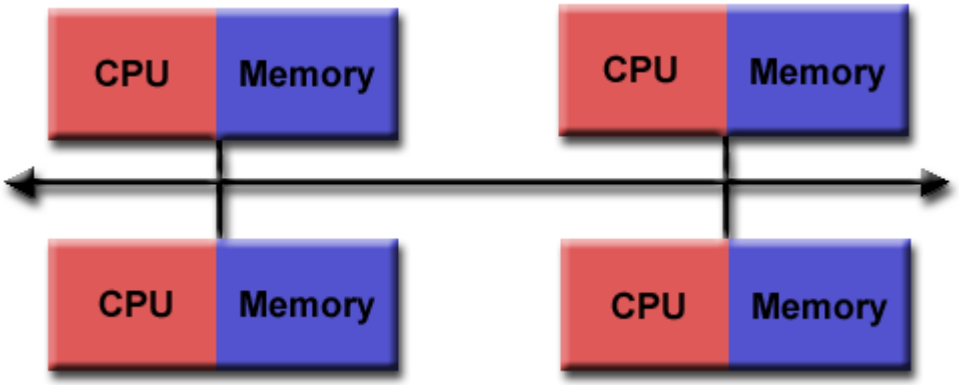
```
$ gfortran -fopenmp pi_openmp.f90 -o pi_openmp

$ export OMP_NUM_THREADS='12'

$ ./pi_openmp
```

```
program pi_openmp
implicit none
integer :: i
integer*8 :: n
double precision :: f, x, sum, pi, h
write(*,*) 'number of intervals?'
read(*,*) n
h = 1.0d0 / n
sum = 0.0d0
!$omp parallel do private(i,x)
reduction(+:sum)
do i = 1, n
  x = (i - 0.5d0) * h
  sum = sum + (4.d0 / (1.d0 + x*x))
end do
pi = h * sum
write(*,fmt="(A,F16.12)") "Value of
pi is", pi
end program
```

Reference: OpenMP directives

- **!$OMP PARALLEL DO** specifies a loop that can be executed in parallel
- **!$OMP PARALLEL SECTIONS** starts a set of sections that are each executed in parallel by a team of threads
- **!$OMP PARALLEL** introduces a code region that is executed redundantly by the threads
- **!$OMP DO / FOR** is a work sharing construct and may be used within a parallel region
- **!$OMP SECTIONS** is also a work sharing construct; allows the current team of threads executing the surrounding parallel region to cooperate in the execution of the parallel sections.
- **!$OMP TASK** greatly simplifies the parallelization on non-loop constructs by allowing to dynamically specify portions of the programs which can run independently

Distributed memory systems



… vary widely but share a common characteristic: consists of many nodes, requires a communication network to connect inter-process memory

- Processors have their own local memory. Memory addresses in one processor do not map to another processor: no global address space across all processors
- Each processor operates independently. Changes to its local memory have no effect on the memory of other processors. The concept of cache coherency does not apply.
- When needed, a processor needs access to data in another processor, it requests it through communication network
- It is the task of the programmer to explicitly define how and when data is communicated
- The network "fabric" used for data transfer varies widely, though it can can be as simple as Ethernet

Distributed memory: pros and cons

Advantages:

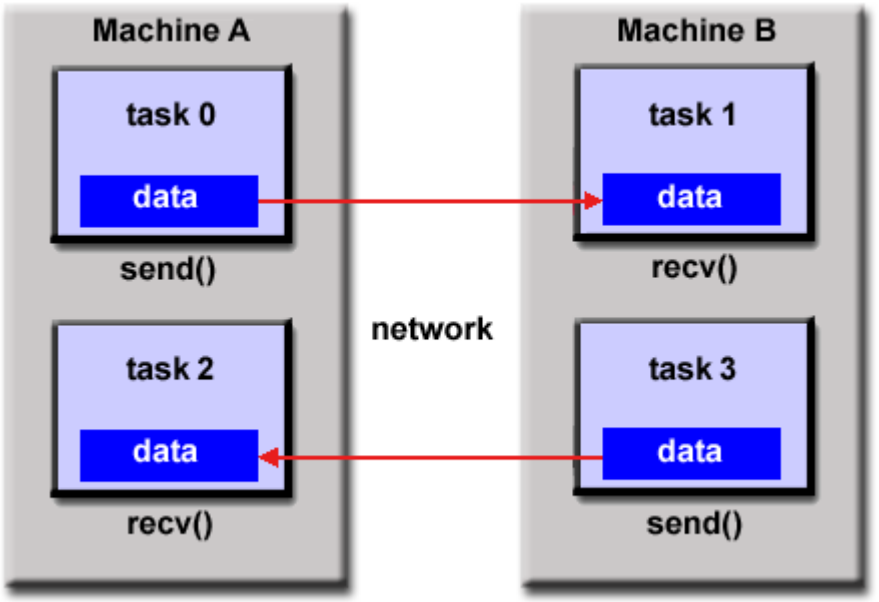
- Memory is scalable with number of processors. Increase the number of processors and the size of memory increases proportionately.
- Each processor can rapidly access its own memory without interference and without the overhead incurred with trying to maintain cache coherency.
- Cost effectiveness: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

Disadvantages:

- The programmer is responsible for many of the details associated with data communication between processors.
- It may be difficult to map existing data structures, based on global memory, to this memory organization.
- Non-uniform memory access (NUMA) times

Message Passing model

- A set of tasks that use their own local memory during computation. Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.
- Tasks exchange data through communications by sending and receiving messages.
- Natural to distributed memory archs, but can be used on shared memory machine



MPI implementation

- From a programmer or a user perspective: MPI is a library of subroutines that are embedded in source code. The programmer is responsible for determining all parallelism.
- MPI Forum: http://mpi-forum.org/docs/
- MPI is the "de facto" industry standard for message passing, provides Fortran 77/90, C/C++ language bindings.
- OpenMPI and MVAPICH are the most popular MPI flavors.

```
# export MPI environment
$ mpif90 pi_mpi -o pi_mpi
$ mpirun -np 12 ./pi_mpi
```

```
program pi_mpi
implicit none
include 'mpif.h'
integer :: i, ierr, myrank, numprocs
integer*8 :: n
double precision :: f, x, sum, pi, h, mypi
call MPI_Init(ierr)
call MPI_Comm_rank(MPI_COMM_WORLD, myrank, ierr)
call MPI_Comm_size(MPI_COMM_WORLD, numprocs, ierr)
if ( myrank == 0 ) then
  write(*,*) 'number of intervals?'
  read(*,*) n
end if
call MPI_Bcast(n, 1, MPI_INTEGER, 0, MPI_COMM_WORLD, ierr)
h = 1.0d0 / n
sum = 0.0d0
do i = myrank + 1, n, numprocs
  x = (i - 0.5d0) * h
  sum = sum + (4.d0 / (1.d0 + x*x))
end do
mypi = h * sum
call MPI_Reduce(mypi, pi, 1, MPI_DOUBLE_PRECISION, &
  MPI_SUM, 0, MPI_COMM_WORLD, ierr)
if (myrank == 0 ) then
  write(*,fmt="(A,F16.12)") "Value of pi is", pi
end if
call MPI_Finalize(ierr)
end program
```

Reference: MPI basic routines

MPI consists of more than 320 functions. But realistic programs can already be developed based on no more than six functions:

- **MPI_Init** initializes the library. It has to be called at the beginning of a parallel operation before any other MPI routines are executed.
- **MPI_Finalize** frees any resources used by the library and has to be called at the end of the program.
- **MPI_Comm_size** determines the number of processors executing the parallel program.
- **MPI_Comm_rank** returns the unique process identifier.
- **MPI_Send** transfers a message to a target process. This operation is a blocking send operation, i.e., it terminates when the message buffer can be reused either because the message was copied to a system buffer by the library or because the message was delivered to the target process.
- **MPI_Recv** receives a message. This routine terminates if a message was copied into the receive buffer.

Hybrid Distributed-Shared Memory

Most of the today's HPC installations are hybrid, i.e. smaller shared memory systems joint into a large distributed memory architecture

- The shared memory component is usually a 2-4 CPU sockets with several dozens of CPU cores all together. Processors on a given node can address that machine's memory as global.
- The distributed memory component is the networking of multiple shared memory node. Network communications are required to move data from one node to another.

Current trends seem to indicate that this type of memory architecture will continue to prevail and increase at the high end of computing for the foreseeable future.

Programming model is a pure MPI or a code that implements both threads and MPI