Introduction, background, jargon

Jakub Yaghob

Literature

- T.G.Mattson, B.A.Sanders, B.L.Massingill: *Patterns for Parallel Programming*, Addison-Wesley, 2005, ISBN 978-0-321-22811-6
- M.McCool, A.D.Robinson, J.Reinders: *Structured Parallel Programming*, Morgan Kaufmann, 2012, ISBN 978-0-12-415993-8
- Threading Building Blocks, www.threadingbuildingblocks.org
- OpenMP, www.openmp.org
- MPI, www.mpi-forum.org
- OpenCL, www.khronos.org/opencl
- C++, en.cppreference.com/w/cpp/thread

Parallel programming

- Exploitable concurrency
- Why?
  - Solve problem in less time
  - Solve bigger problem than would be possible on a single CPU
- Programmer's task
  - Identify the concurrency in the problem
  - Structure the algorithm so that this concurrency can be exploited
  - Implement the solution using a suitable programming environment

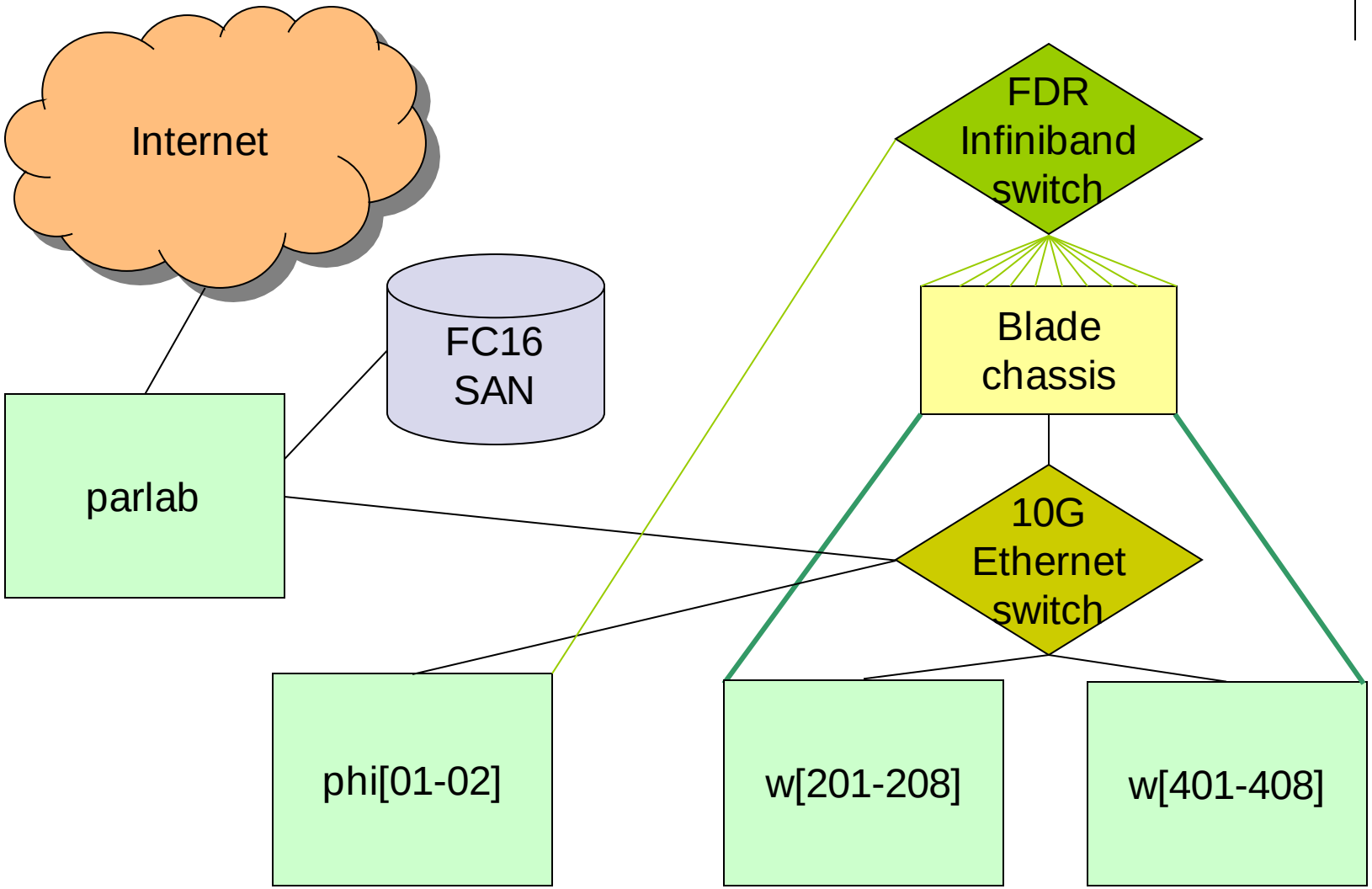
Goals

- Parallel programming in the real world
  - Not about algorithms using 2^n CPUs
- Practical using of different technologies for parallel programming
  - TBB, OpenMP, MPI, OpenCL, C++
- Application of design patterns for parallel programming

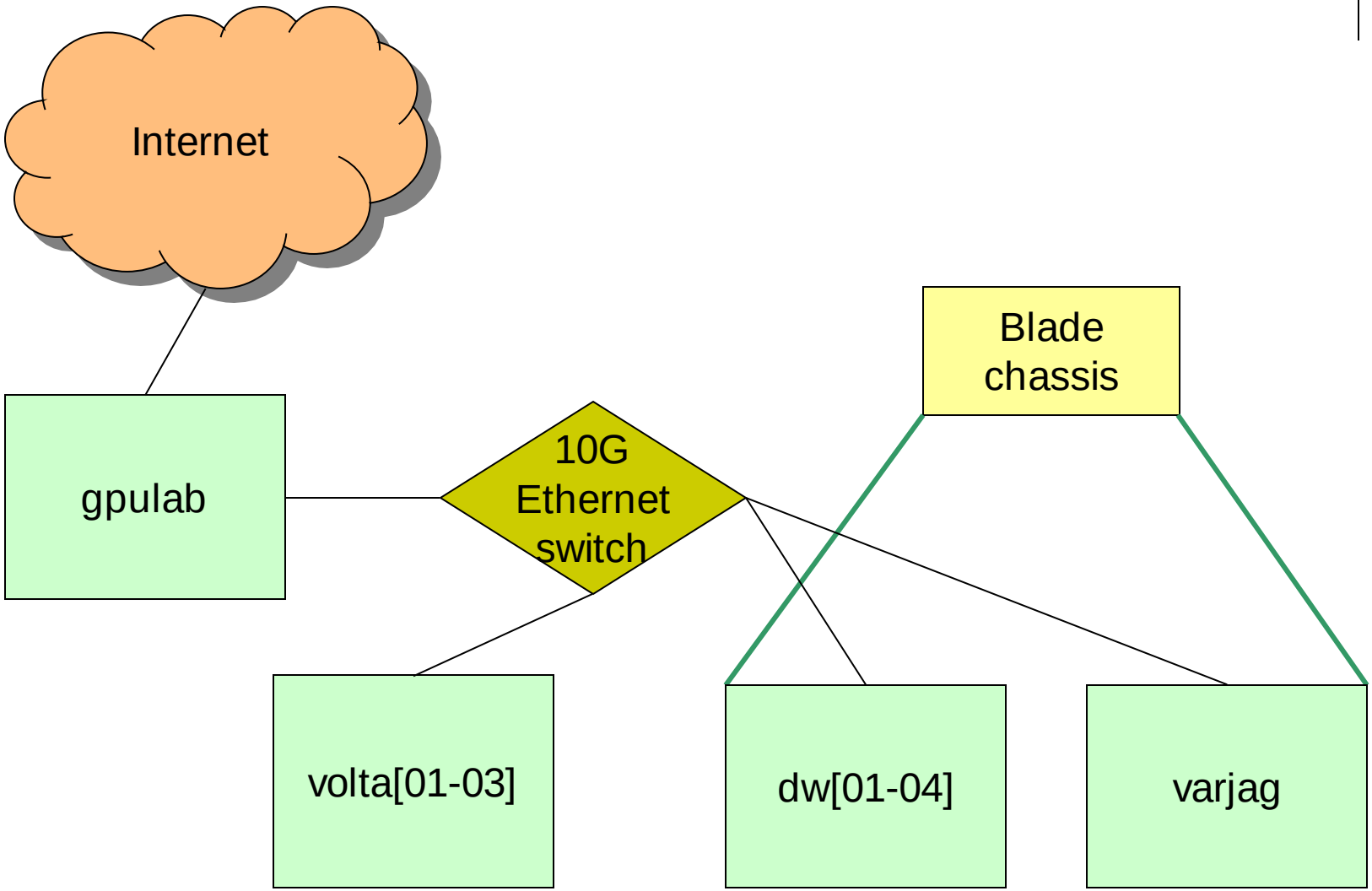
Laboratory – overview

- Two SLURM clusters
  - parlab – parallel and distributed computing
  - gpulab – parallel and GPU computing, Docker
- Common homes
  - NFS array mounted on /mnt/home

Parlab


Internet
FDR
Infiniband
switch
FC16
SAN
Blade
chassis
parlab
10G
Ethernet
switch
phi[01-02]
w[201-208]
w[401-408]

Gpulab


Internet
gpulab
10G
Ethernet
switch
Blade
chassis
volta[01-03]
dw[01-04]
varjag

Parlab – specification

- Blade servers
  - w[401-404]: 4x Octa-core HT CPU 2 GHz, 128 GB RAM, 2xSAS 146 GB 15k
  - w[201-208]: 2x 16-core HT CPU 2.1 GHz, 128 GB RAM, 2xSAS 120 GB SSD
- Xeon Phi servers
  - phi[01-02]: 1x 64-core HT-4 CPU 1.3 GHz, 96 GB RAM+16 GB MCDRAM, 2xSATA 200 GB SSD
- Ethernet
  - Default traffic
  - 10GbE
- InfiniBand
  - MPI traffic
  - FDR (56 Gb/s)

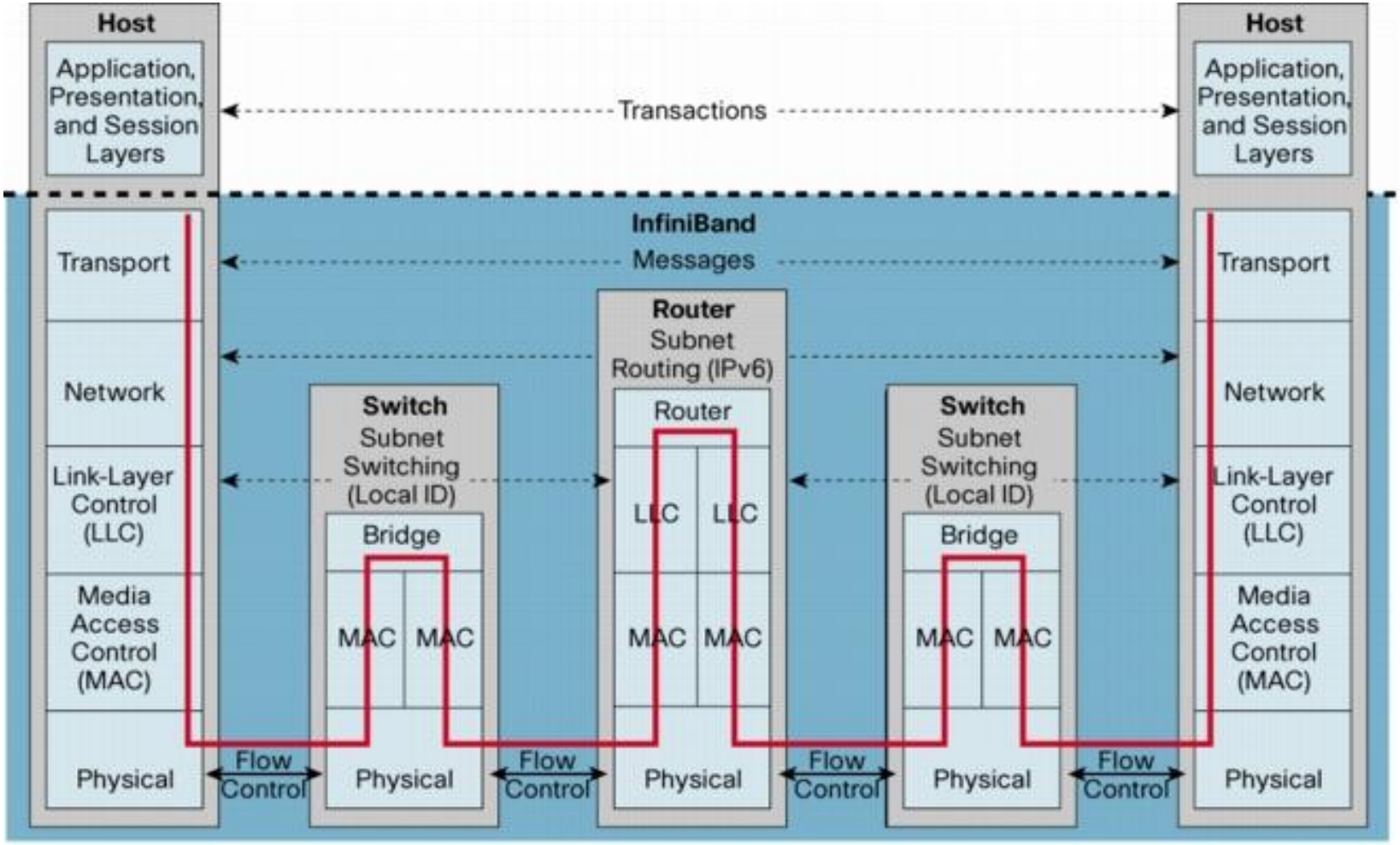
Gpulab – specification

- Blade servers
  - varjag: 4x Octa-core HT CPU 2.13 GHz, 256 GB RAM, 2xSAS 146 GB 15k
  - dw[01-02]: 2x Quad-core CPU 3 GHz, 32 GB RAM, 2x SAS 146 GB 15k
  - dw03: 2x Quad-core HT CPU 2.67 GHz, 96 GB RAM, 2x SAS 146 GB 15k
  - dw04: 2x 10-core HT CPU 2.2 GHz, 256 GB RAM, 2x SAS 300 GB 10k
- GPU servers
  - volta[01-02]: 2x 8-core HT CPU 2.1 GHz, 128 GB RAM, 2x SAS 120 GB SSD, 2x Tesla V100 16GB PCIe
  - volta03: 2x 8-core HT CPU 2.1 GHz, 196 GB RAM, 2x SAS 240 GB SSD, 2x Tesla V100 16GB PCIe
- Ethernet
  - 10GbE

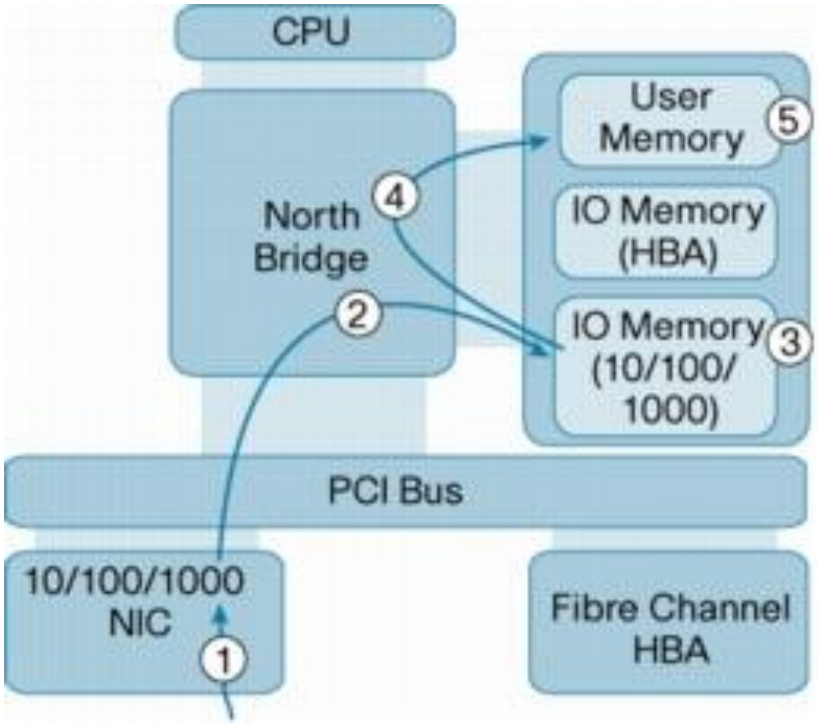
SLURM – crash course

- Resource manager
  - Commands from any node
  - Acquire resources before use
  - Cluster nodes divided to partitions
- Important commands
  - srun – acquire resources and immediately run a command, creates a job
  - sbatch – acquire resources and run a shell script , creates a job
  - scancel – cancel a job
  - sinfo – get info about partitions, nodes, …
  - squeue – get info about job queue

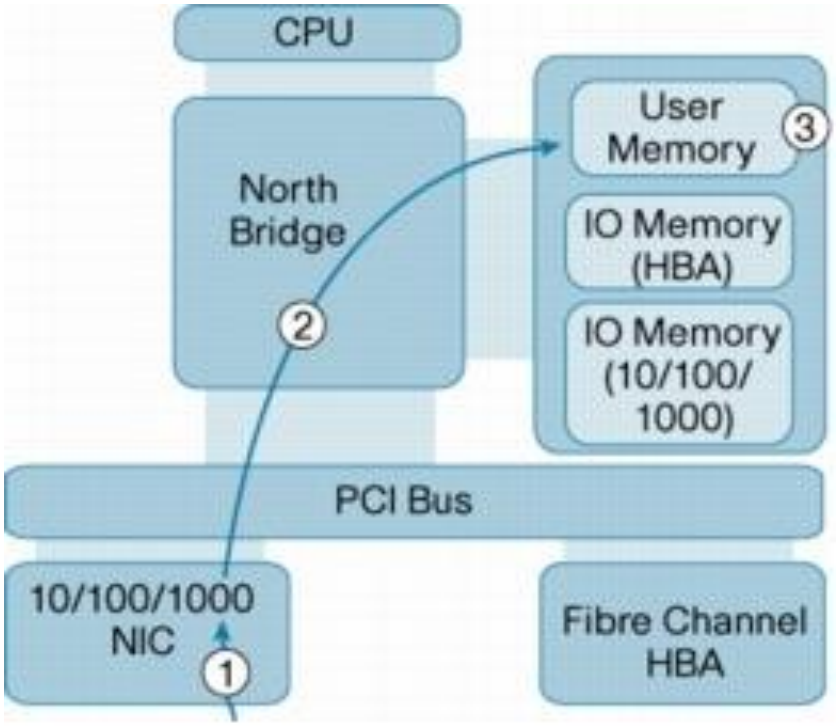
Infiniband protocol stack



InfiniBand – RDMA (Remote DMA)

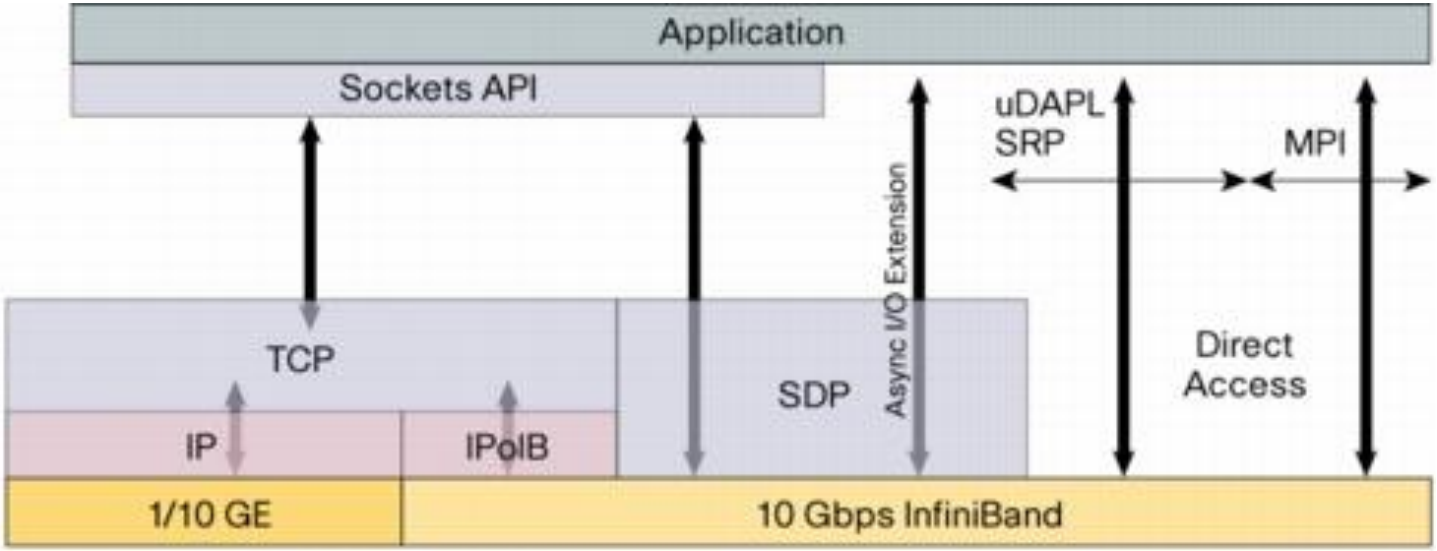


a) traditional



b) RDMA

InfiniBand – latency



Throughput	1 Gbps	3.9 Gbps	4.1 Gbps	4.5 Gbps	7.9 Gbps	8 Gbps	8 Gbs
Latency	40 usec	40 usec	20 usec	15 usec	15 usec	8 usec	4.5 usec
CPU Utilization	30%	80%	85%	25%	4%	1%	>1%

MPI: Message Passing Interface

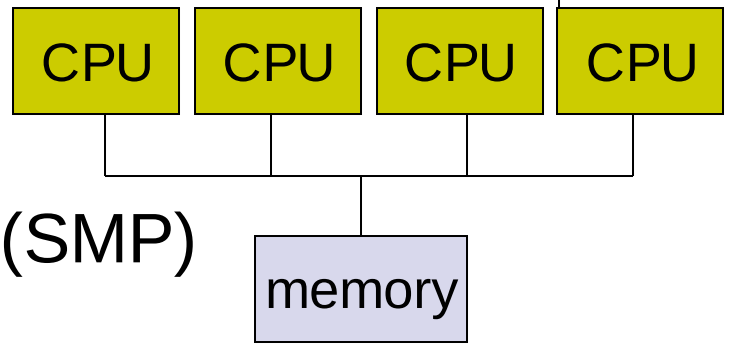
SRP: SCSI Remote Protocol

uDAPL: User-level Direct Access Programming Language

Flynn's taxonomy

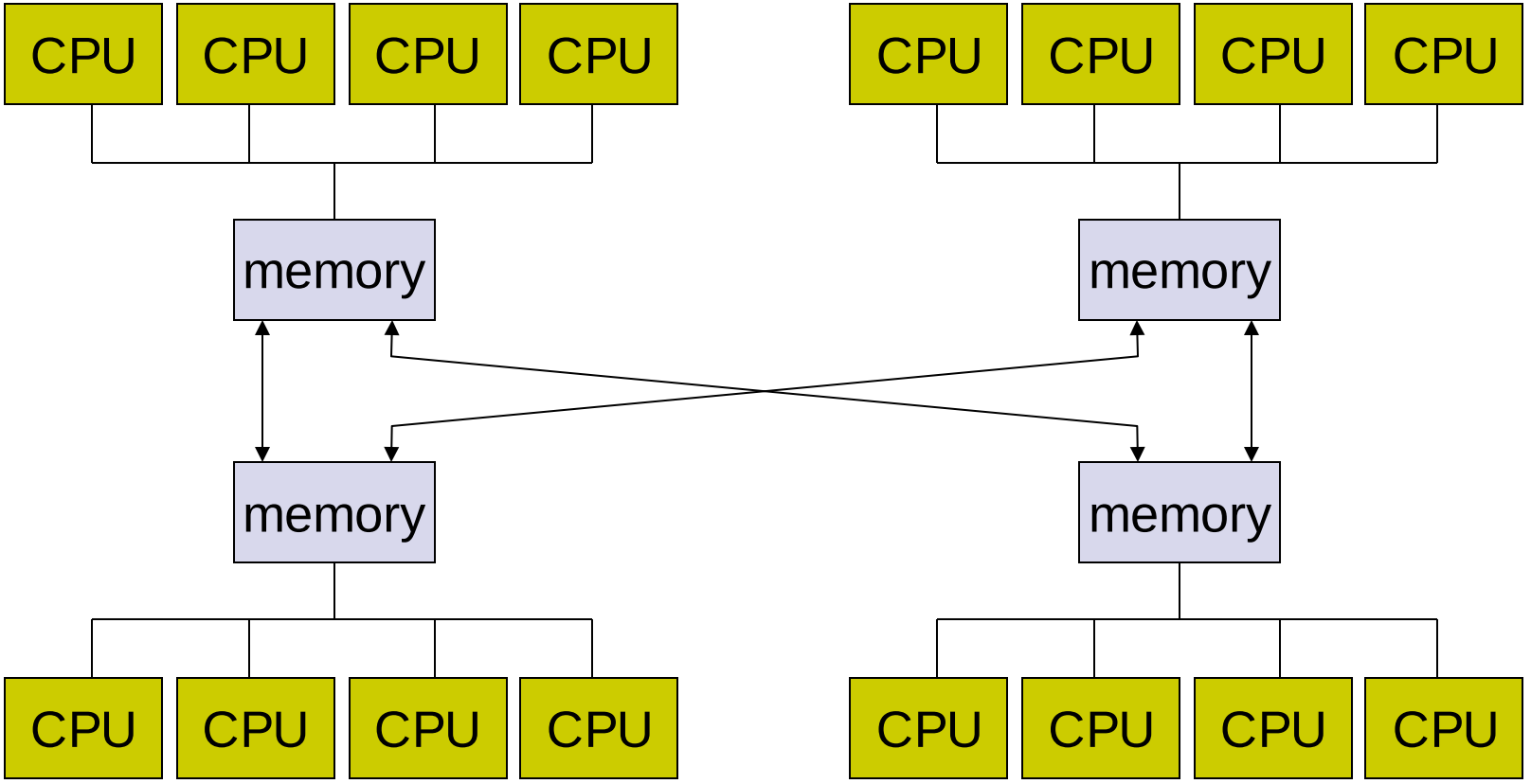
- Single Instruction, Single Data (SISD)
  - von Neumann
- Single Instruction, Multiple Data (SIMD)
  - Vector processors
- Multiple Instruction, Single Data (MISD)
  - No well-known systems
- Multiple Instruction, Multiple Data (MIMD)
  - All modern parallel systems

A further breakdown of MIMD – shared memory



- Shared memory
  - Symmetric multiprocessors (SMP)
    - Easiest to program
    - Do not scale well, small number of CPUs
  - Nonuniform memory access (NUMA)
    - Uniformly addressable from all CPUs
    - Some memory blocks may be physically more closely, the access time significantly varies
    - Each CPU has a cache, it mitigates the effect of NUMA – cache coherent NUMA (ccNUMA), nearly as SMP (locality issues, cache effects)

A further breakdown of MIMD – NUMA



A further breakdown of MIMD – distributed memory

- Distributed memory
  - Own address space, message passing
  - Explicitly program the communication, the distribution of data
  - Massively parallel processors (MPP)
    - CPUs and network tightly coupled
  - Cluster
    - Composed of off-the-shelf computers connected by an off-the-shelf network

A further breakdown of MIMD – clusters

- Clusters
  - Hybrid systems
    - Each node several CPUs with shared memory
  - Grids
    - Heterogeneous resources
    - Connected by Internet
    - Various resources in the grid do not have a common point of administration

Parallel programming environments

- OpenMP
  - Set of language extensions implemented as compiler directives (Fortran, C, C++)
  - Works well on SMP, less ideal for NUMA or distributed systems
- MPI
  - Set of library routines – process management, message passing, collective communication operations
  - Difficult to write – explicit data distribution, interprocess communication
  - Good choice of MPP
- Combination for hybrid systems

The jargon

- Task
  - Sequence of instructions that operate together as a group
  - Logical part of an algorithm
- Unit of execution (UE)
  - Process or thread
  - Tasks are mapped to UEs
- Processing element (PE)
  - HW element that executes a stream of instructions

The jargon

- Load balancing
  - How well the work is distributed among PEs
- Synchronization
  - Enforcing necessary ordering constraints
- Synchronous x asynchronous
  - Coupling two events in time
- Race condition
  - Outcome of a program depends on on the scheduling of UEs
- Deadlock

Performance modeling

- Total running time on one PE

$$T_{total}(1) = T_{setup} + T_{compute} + T_{finalization}$$

- Total running time on *P* PEs

$$T_{total}(P) = T_{setup} + \frac{T_{compute}(1)}{P} + T_{finalization}$$

- Speedup

$$S(P) = \frac{T_{total}(1)}{T_{total}(P)}$$

Performance modeling

- Efficiency

$$E(P) = \frac{S(P)}{P} = \frac{T_{total}(1)}{P\,T_{total}(P)}$$

- Perfect linear speedup: S(P)=P
- Serial fraction

$$\gamma = \frac{T_{setup} + T_{finalization}}{T_{total}(1)}$$

- Rewrite total running time for P PEs

$$T_{total}(P) = \gamma\, T_{total}(1) + \frac{(1-\gamma)\,T_{total}(1)}{P}$$

Amdahl's law

- Rewriting the speedup

$$S(P) = \frac{T_{total}(1)}{\left(\gamma + \frac{(1-\gamma)}{P}\right) T_{total}(1)} = \frac{1}{\gamma + \frac{(1-\gamma)}{P}}$$

- No overhead in the parallel part
- The same problem with varying number of CPUs
- Very large number of CPUs

$$\lim_{P \to \infty} S(P) = \lim_{P \to \infty} \frac{1}{\gamma + \frac{(1-\gamma)}{P}} = \frac{1}{\gamma}$$

Gustafson's law

- Rewrite total running time when executed on P PEs

$$T_{total}(1) = T_{setup} + P\,T_{compute}(P) + T_{finalization}$$

- Scaled serial fraction

$$\gamma_{scaled} = \frac{T_{setup} + T_{finalization}}{T_{total}(P)}$$

- Rewrite

$$T_{total}(1) = \gamma_{scaled} T_{total}(P) + P(1-\gamma_{scaled}) T_{total}(P)$$

- Rewrite speedup – scaled speedup

$$S(P) = P + (1-P)\,\gamma_{scaled}$$

- What happens, when the number of PEs is increased (γ_{scaled} is constant)?

Communication

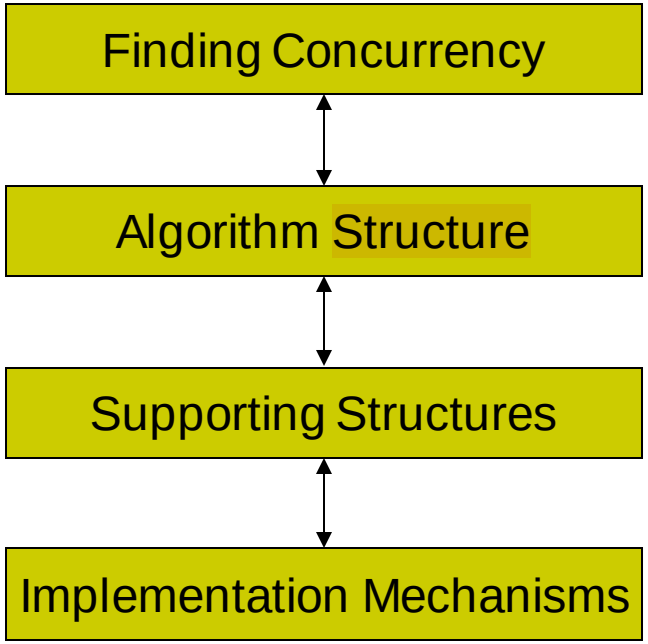
- The total time for message transfer (Bandwidth β, length N)
- Latency α

$$T_{message-transfer} = \alpha + \frac{N}{\beta}$$

  - Fixed cost
  - The time it takes to send an empty message to the time it is received
- Latency hiding
  - Overlapping communication
  - Mapping multiple UEs to each PE

Design patterns

- Organized into design spaces
- Guidance through the entire process of developing a parallel program



Design spaces

- Finding Concurrency
  - Structuring the problem to expose exploitable concurrency
  - High-level algorithmic issues
- Algorithm Structure
  - Structuring the algorithm to take advantage of potential concurrency
  - How to use concurrency exposed in Finding Concurrency
  - Overall strategies for exploiting concurrency

Design spaces

- Supporting Structure
  - Intermediate stage between Algorithm structure and Implementation mechanisms
  - Program-structuring approaches
  - Commonly used data structures
- Implementation mechanisms
  - How the patterns of the higher-level spaces are mapped into particular programming environments
  - Not patterns

Design process

- Design process starts in the problem domain
- The ultimate aim of the design process is software
  - At some point, the design elements change into ones relevant to program
  - Program domain
- Be careful!
  - After a problem has been mapped onto the program domain too early, it can be difficult to see opportunities to exploit concurrency