SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

# Parallel computation methods on CPU architectures

Claudia Blaas-Schenner

VSC Research Center, TU Wien

VIENNA SCIENTIFIC CLUSTER

06/2021

Univerza v Ljubljani

TECHNISCHE UNIVERSITÄT WIEN

CINECA consorzio interuniversitario

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

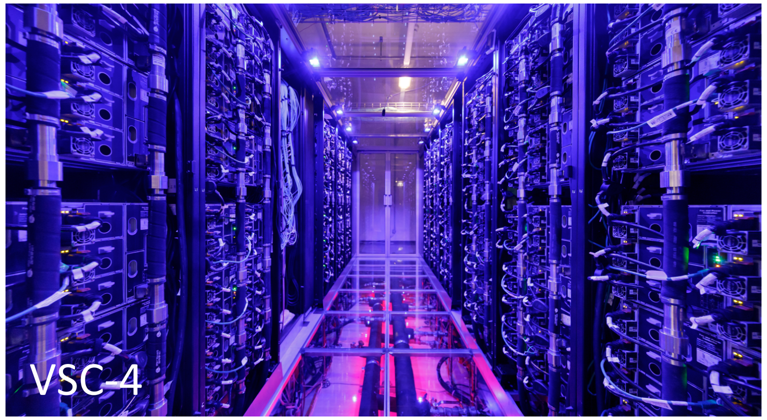
Co-funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission.
This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

# HPC systems

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

- VSC – joint high performance computing (HPC) facility of Austrian universities



VSC-5

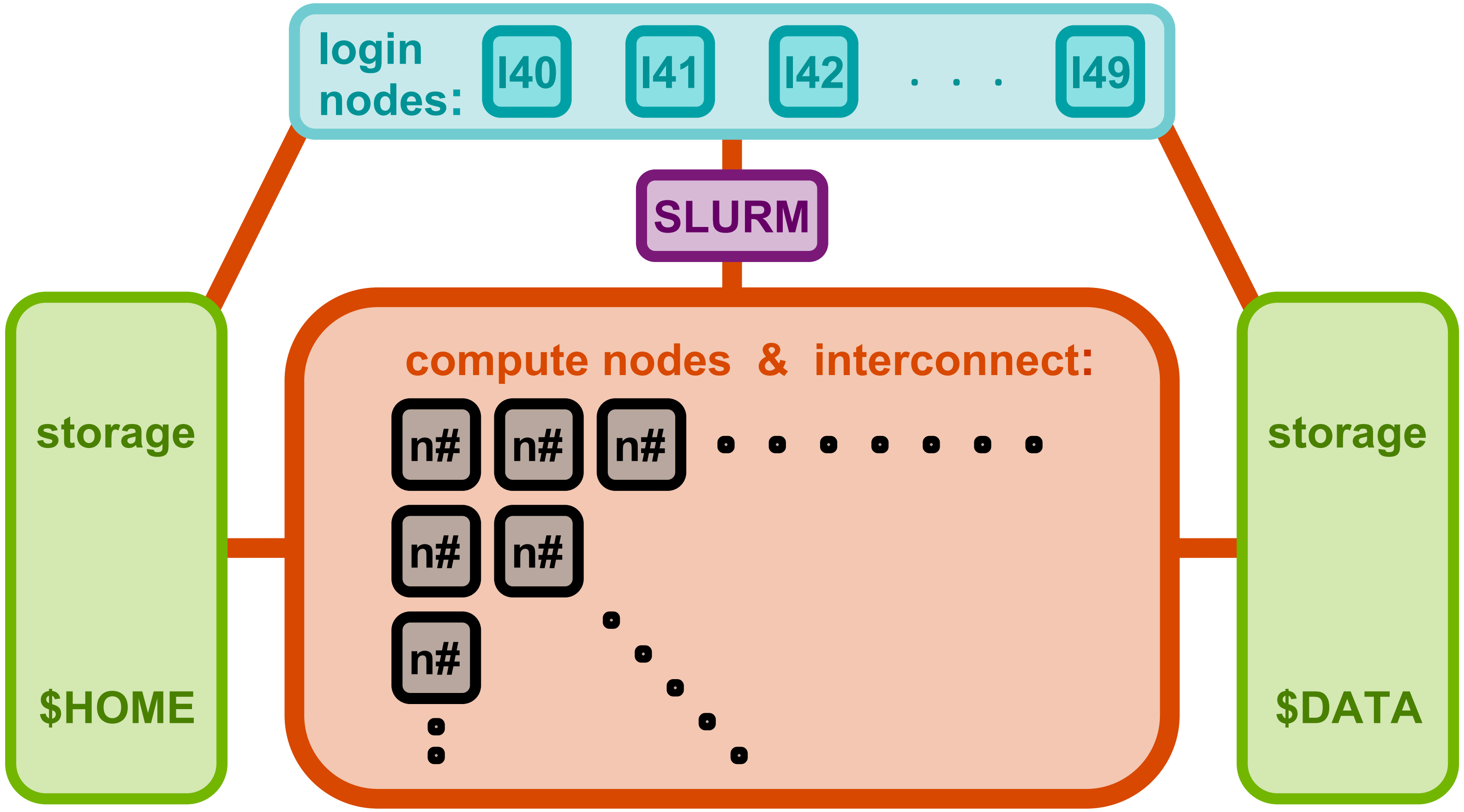
soon to come…

VIENNA SCIENTIFIC CLUSTER

https://vsc.ac.at
https://vsc.ac.at/training

# components of HPC clusters

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP



**login nodes**

↕

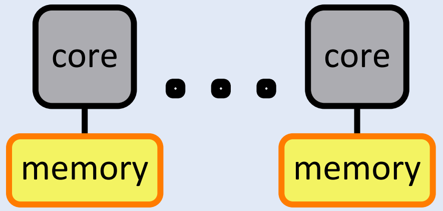
**compute nodes**

**shared** (login, storage)

↕

**user exclusive** (compute nodes)

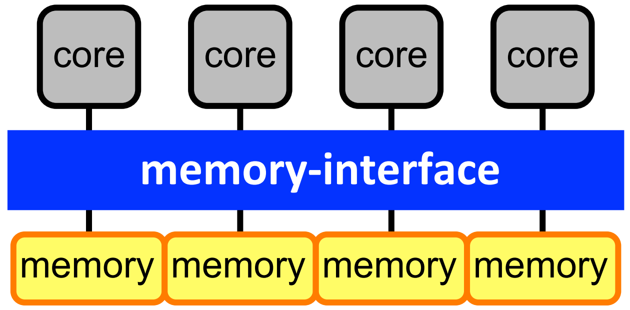
## parallel hardware

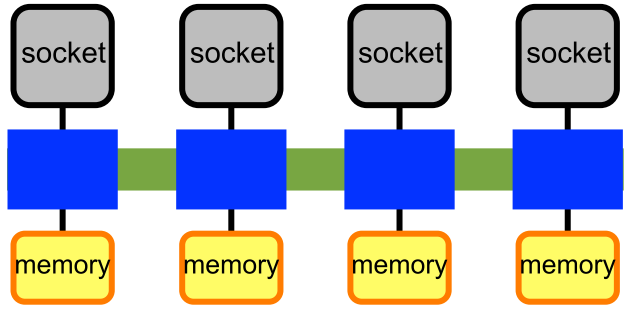


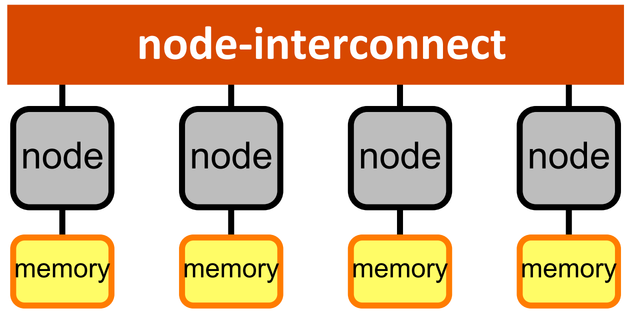
SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

### shared memory

### distributed memory







**socket: → memory-interface**
UMA (uniform memory access)
SMP (symmetric multi-processing)

**node: → hyper-transport**
ccNUMA (cache-coherent non-uniform…)
! first touch, pinning !

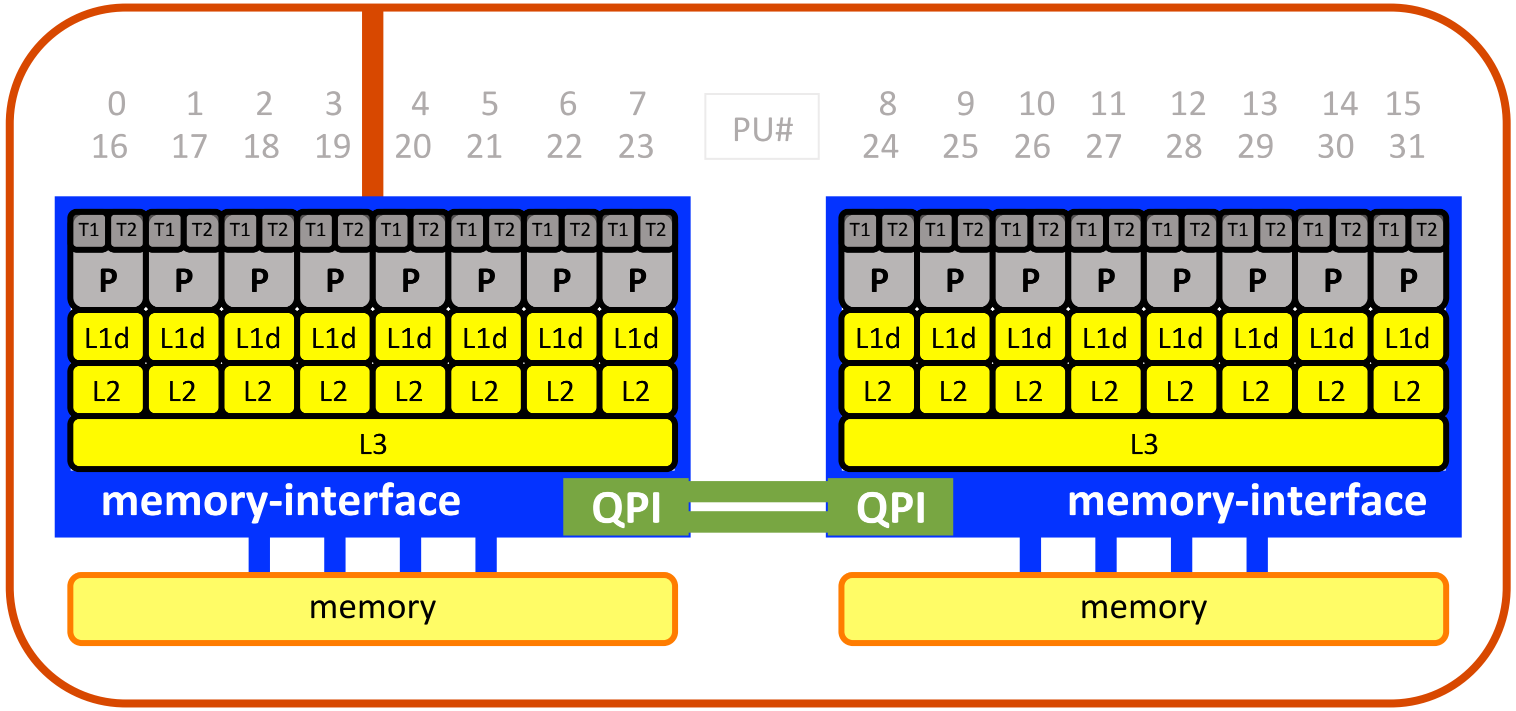
**cluster: → node-interconnect**
NUMA (non-uniform memory access)
! fast access only to its own memory !

**socket / CPU** **node** **cluster**

shared memory programming with **OpenMP**

**MPI** works everywhere

# compute nodes



**example:**

1 node

2 sockets (CPUs)

8 cores per socket (P)

2 threads per core (T1/T2)

1 HCA (host channel adapter)
(node-interconnect)

**info about nodes:**

| | |
|---|---|
| numactl --hardware | [Linux] |
| cpuinfo -A | [Intel] |
| likwid-topology -c -g | [LIKWID] |

# node-interconnect

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

**schematic figure:**

3-level fat tree

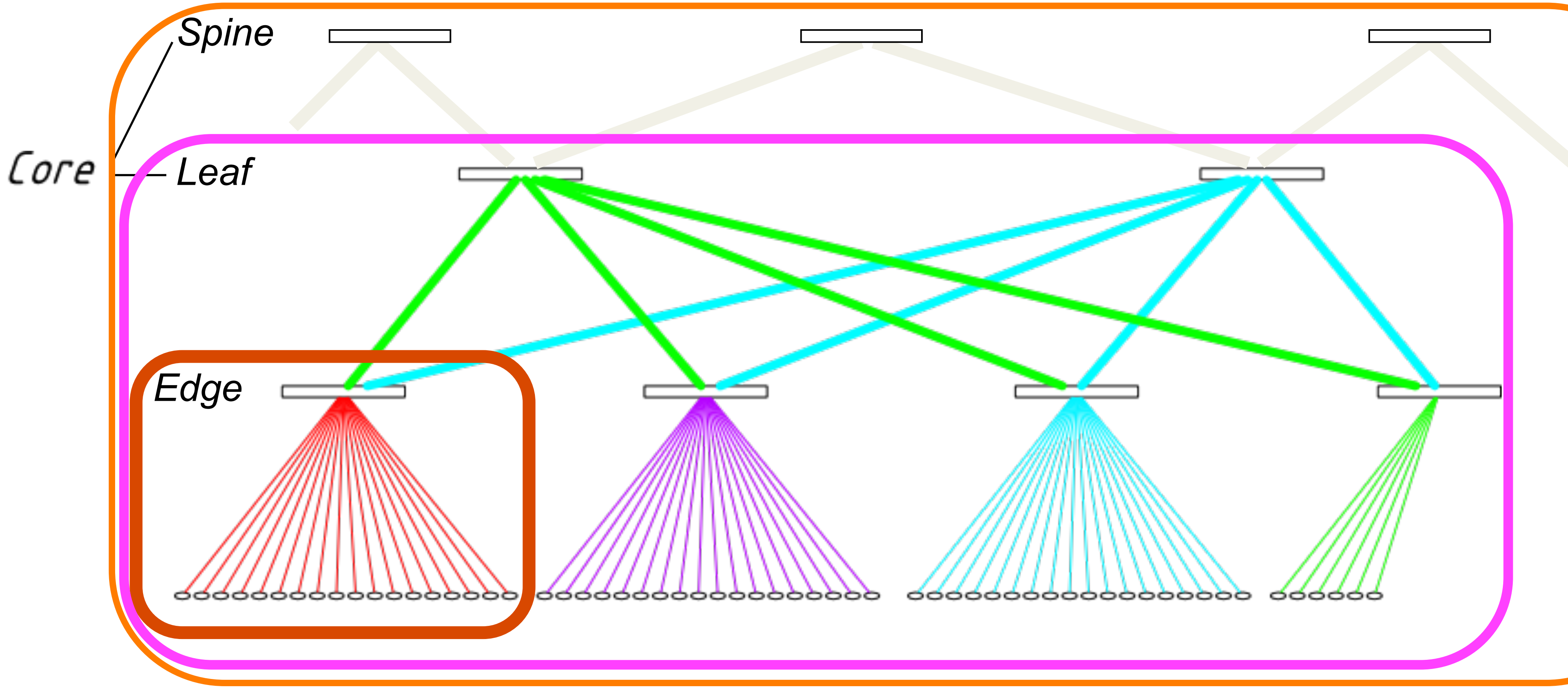
2-level fat-tree

1st level switches

compute nodes

attached to the

lowest level

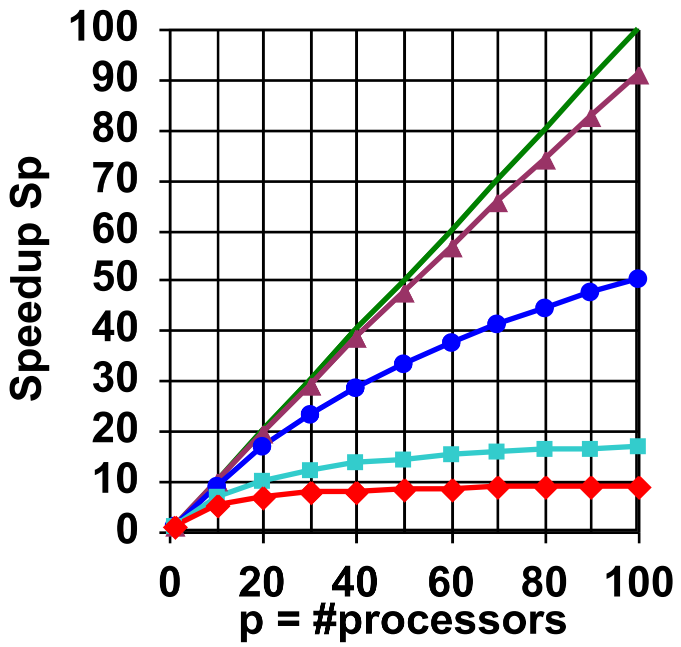


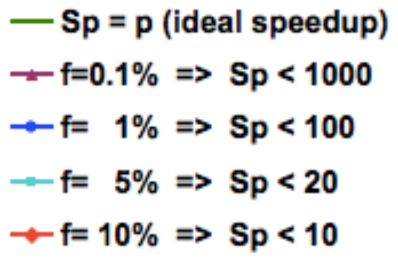
# Amdahl's Law

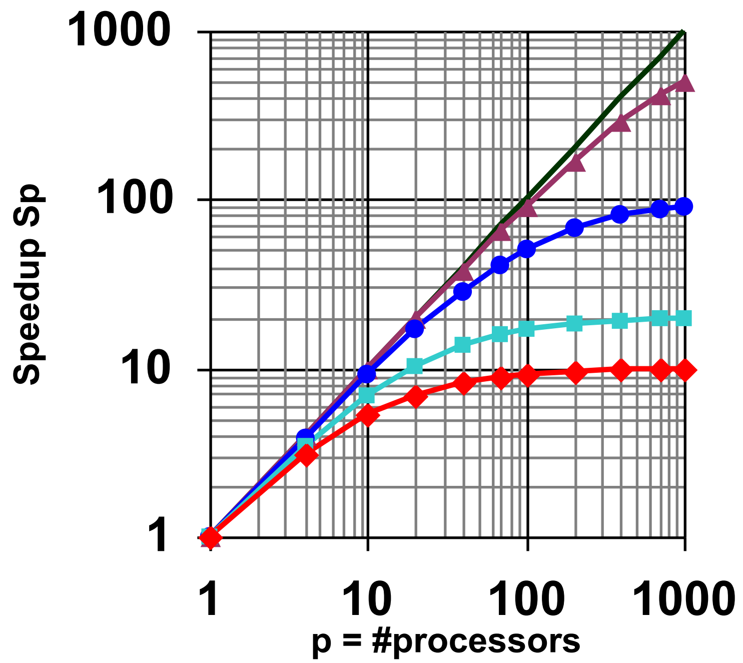
SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

$T_{parallel,\,p} = f \cdot T_{serial} + (1-f) \cdot T_{serial} \,/\, p$ f … sequential part of code **neglecting time for communication**

$S_p = T_{serial} \,/\, T_{parallel,\,p} = 1 \,/\, (f + (1-f) \,/\, p)$ Speedup is limited: $S_p < 1 / f$ **neglecting load imbalance**





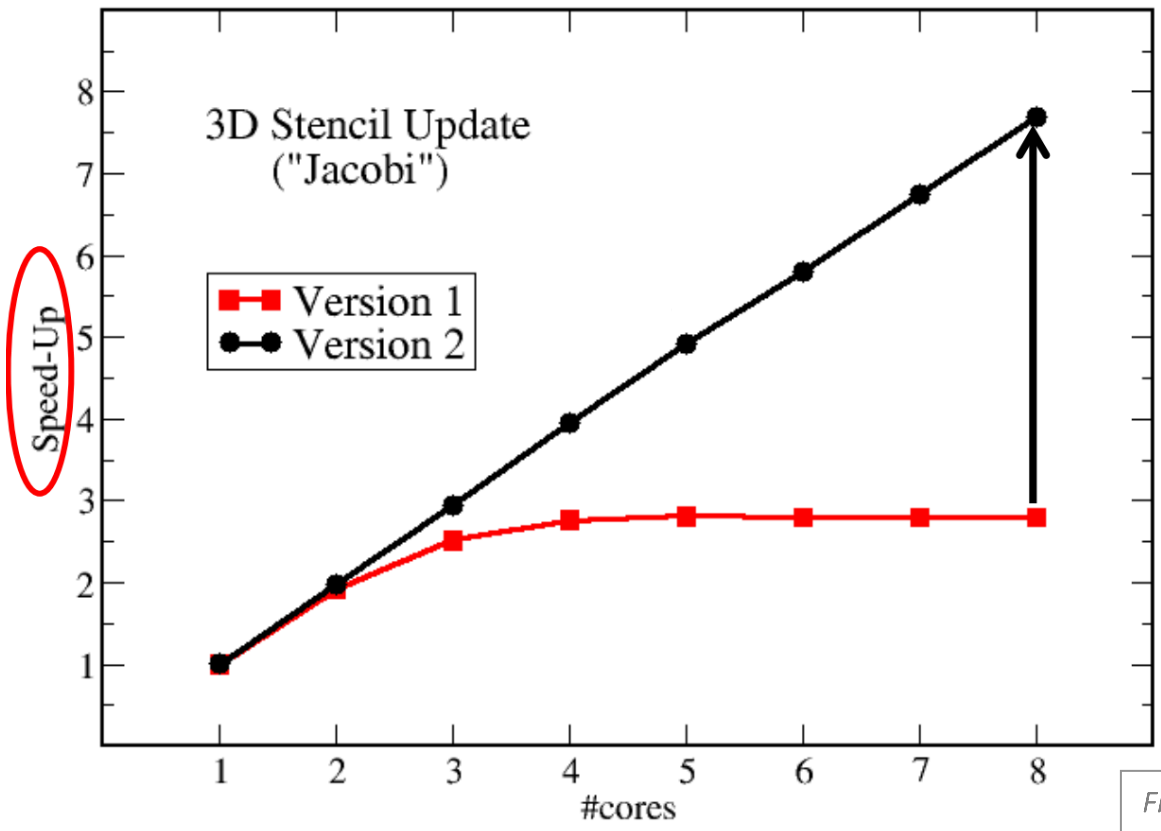


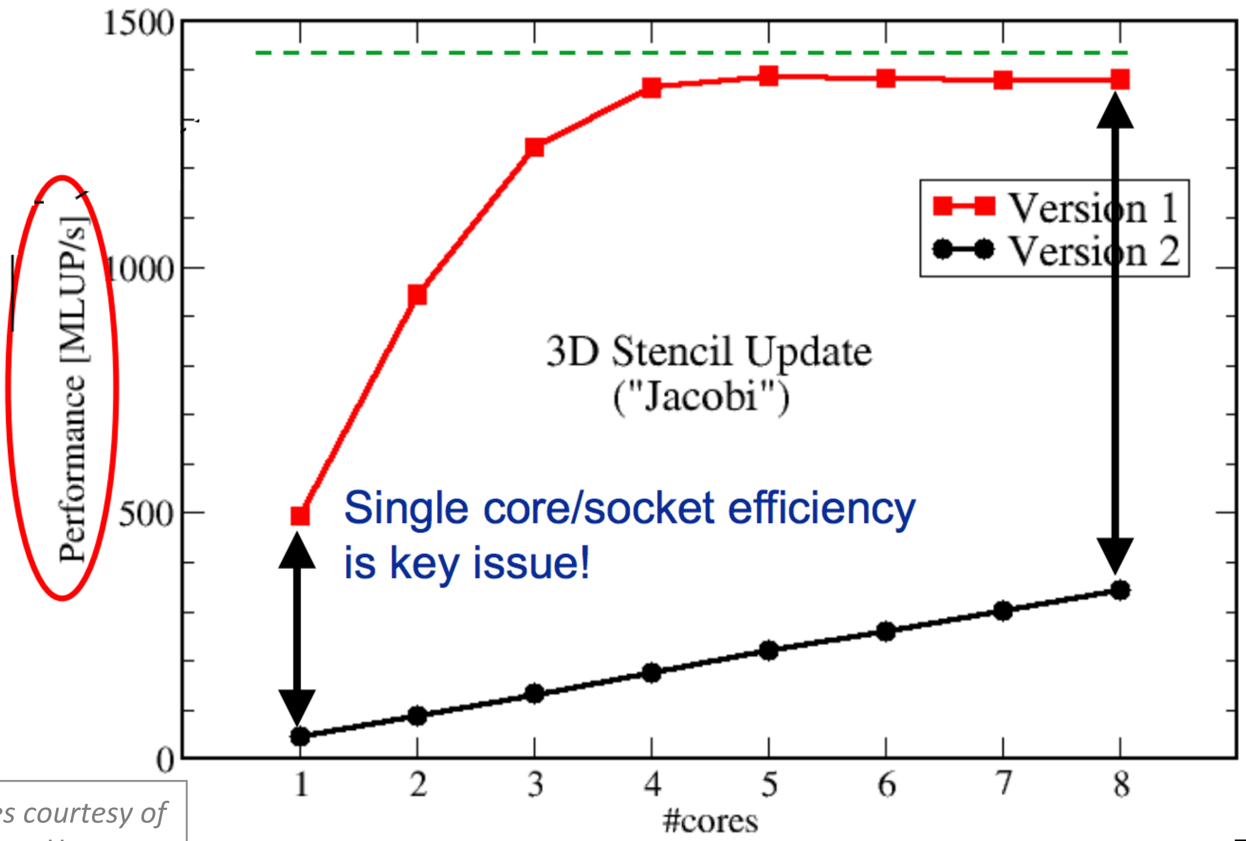
*Figures courtesy of Rolf Rabenseifner.*

**Speedup = ratio – no absolute performance !**

# scalability vs. performance

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP





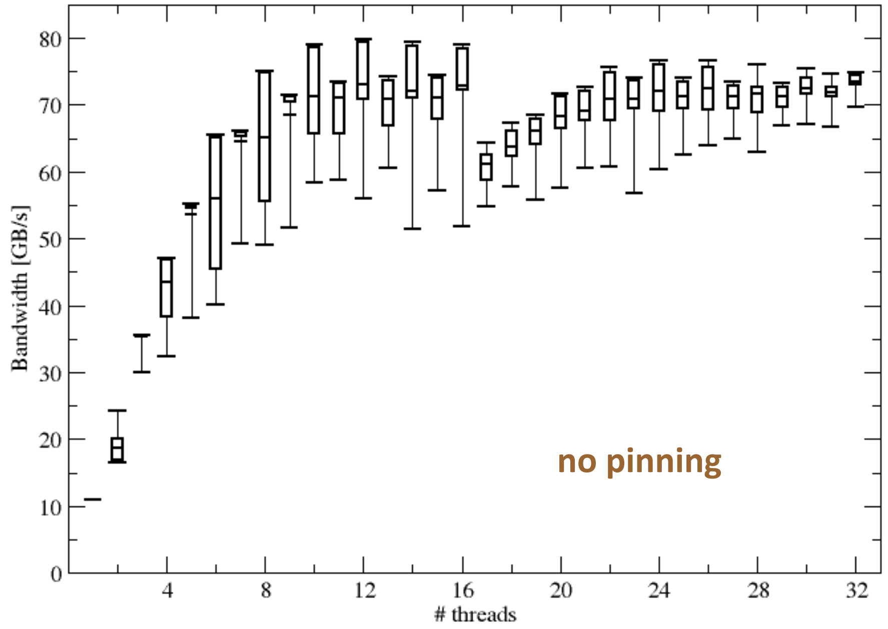
*Figures courtesy of Georg Hager.*

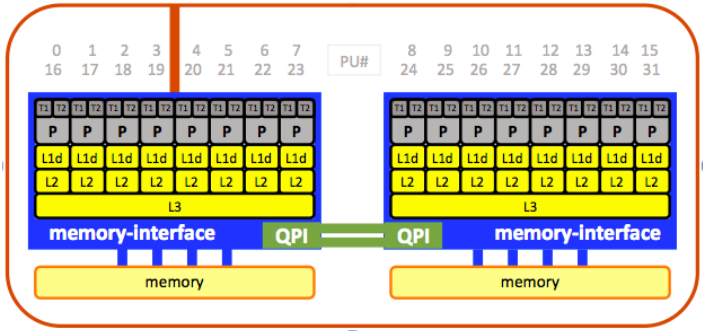
3D Stencil Update ("Jacobi"):

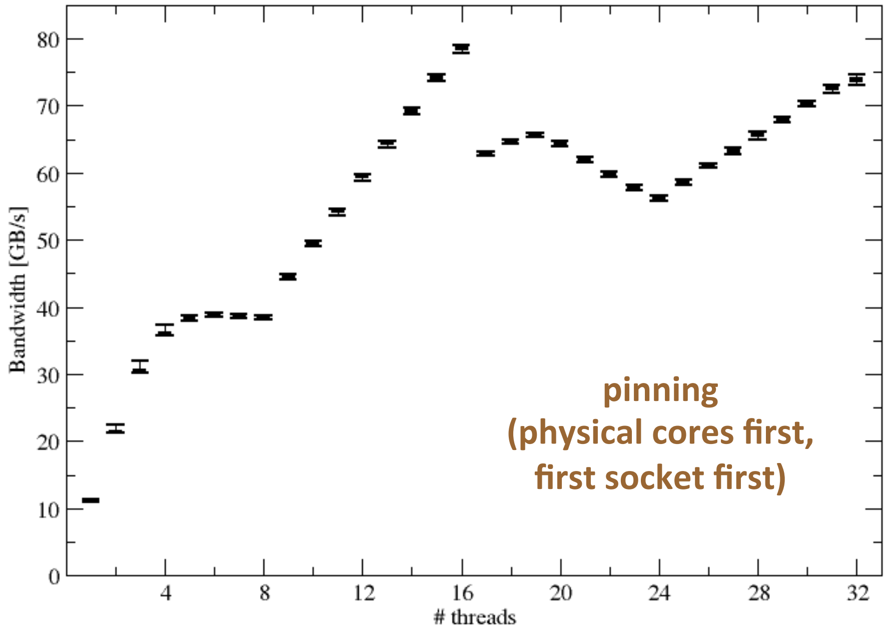
```
y(i,j,k) = b*(  x(i-1,j,k) + x(i+1,j,k) + x(i,j-1,k)
              + x(i,j+1,k) + x(i,j,k-1) + x(i,j,k+1))
```

# pinning ?

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP







OpenMP
STREAM benchmark

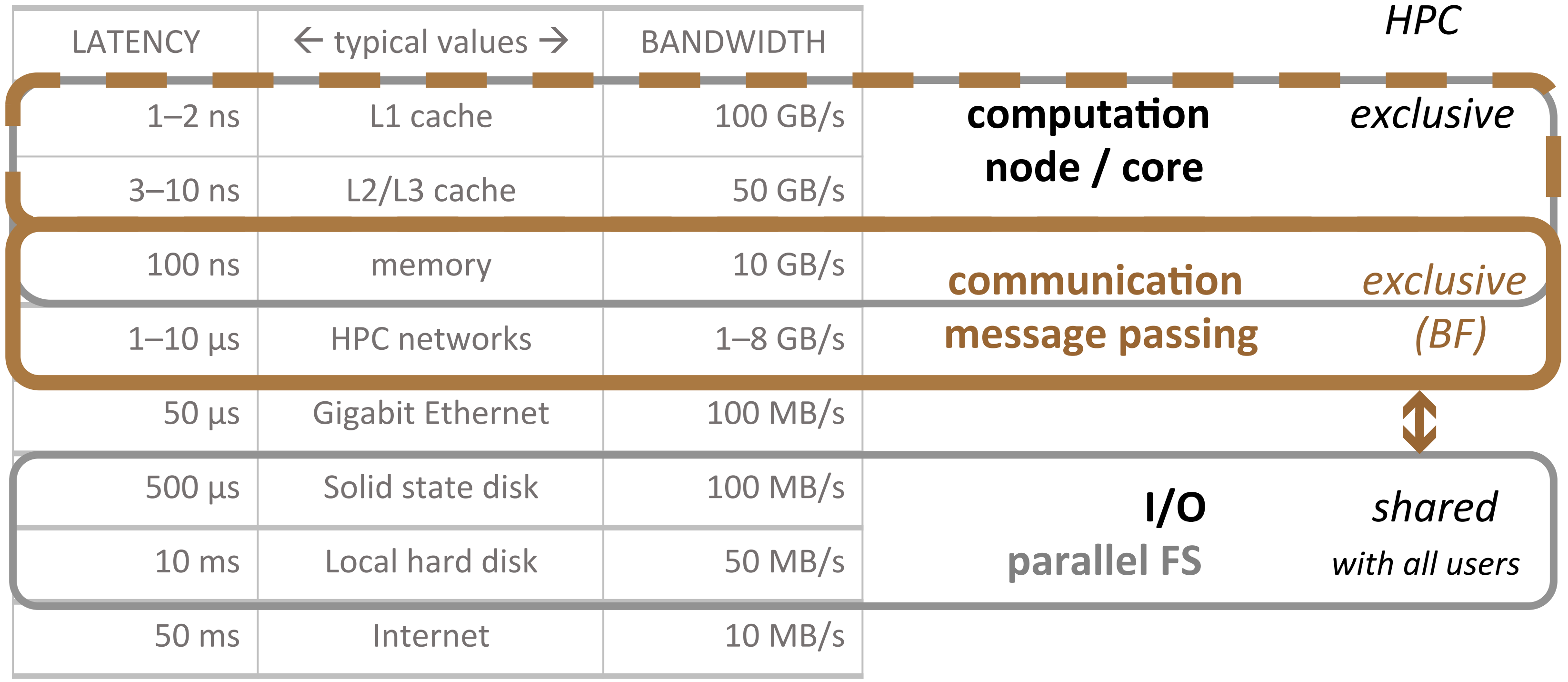
*Benchmark & plots courtesy of Georg Hager.*

**MPI** will give the very same picture **!**

why should we care about **pinning** ?

- eliminating performance variations
- making use of architectural features
- avoiding resource contention

## HPC = computation – communication – I/O



| LATENCY | ← typical values → | BANDWIDTH |
|---|---|---|
| 1–2 ns | L1 cache | 100 GB/s |
| 3–10 ns | L2/L3 cache | 50 GB/s |
| 100 ns | memory | 10 GB/s |
| 1–10 µs | HPC networks | 1–8 GB/s |
| 50 µs | Gigabit Ethernet | 100 MB/s |
| 500 µs | Solid state disk | 100 MB/s |
| 10 ms | Local hard disk | 50 MB/s |
| 50 ms | Internet | 10 MB/s |

Understand HW features!

Know your code!

Know the sys. environment!

→ Take control!

➔ Avoiding slow data paths is the key to most performance optimizations!

# login to a cluster

- **username and password** (ssh-keys)
- restricted IPs (firewall)
- two-factor authentication

- terminal: xterm, terminal, PuTTY
- ssh <username>@<cluster>

➢ Linux command-line access

➢ **graphical user interface (GUI)**

- X-server, XQuartz, Xming
- ssh -X <username>@<cluster>

- **NoMachine** (remote virtual desktop)

  → @viz.hpc.fs.uni-lj.si

# modules & compiling

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

- **module environment** [spack]

```
module list | purge | load | avail [2>&1 | less]
```

- **compiling with GCC**

```
module load foss/2019a

cc --version
mpicc --version

cc [-fopenmp] program.c
mpicc [-fopenmp] program.c
```

- **compiling with Intel**

```
module load intel
```

→ @viz.hpc.fs.uni-lj.si

```
icc --version
mpiicc --version

icc [-qopenmp] program.c
mpiicc [-qopenmp] program.c
```

## SLURM what kind of nodes

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

- partitions

```
sinfo -o %P
sinfo
```

- qos (quality of service)

```
sacctmgr show qos
```

- @VSC → `sqos -acc` & `sqos`

- use other than default

```
#SBATCH --qos=<qos>
#SBATCH --partition=<partition>
#SBATCH --account=<account>
```

- more detailed info

```
scontrol show partition <part.>
scontrol show node <node>
scontrol show reservation
```

# job submission via SLURM

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

- **SLURM** job script

| | |
|---|---|
| `#!/bin/bash` | → has to be a shell script |
| `#SBATCH` | → header lines for the job scheduler |
| `do_my_work` | → whatever needs do be done by the job |

- **SLURM** queuing system

| | |
|---|---|
| • `sbatch job.sh` | → submit |
| • `squeue -u $USER` | → check |
| • `scancel JOB_ID` | → cancel |
| • `slurm-*.out` | → output |

- **recommended** @ `~/.bashrc`

```
alias sq='squeue -u $USER'
export LC_CTYPE=en_US.UTF-8
export LC_ALL=en_US.UTF-8
unset LD_PRELOAD

source ~/.bashrc
```

SCtrain training week provides a node reservation (daily, 9 am – 5 pm) to avoid queuing times:

**`sbatch --reservation=sctrain job.sh`**

## SLURM job script (pure MPI)

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

```
#!/bin/bash                          # → @viz.hpc.fs.uni-lj.si

#SBATCH -J test                      # SLURM_JOB_NAME
#SBATCH -N 2                         # SLURM_JOB_NUM_NODES
#SBATCH --tasks-per-node=24          # SLURM_NTASKS_PER_NODE  [1 mpi/core]

# <do_my_work>
module purge                         # recommended to be done in all jobs !!!!!
module load foss/2019a               # load only modules actually needed by job

mpirun -n $SLURM_NTASKS ./a.out
```

Recommendation: reduce queuing times by allowing SLURM to do backfilling:

**`#SBATCH -time=hh:mm:ss    |    sbatch --time=hh:mm:ss job.sh`**

# demo HPC literacy

SCtrain | SUPERCOMPUTING KNOWLEDGE PARTNERSHIP

- **make yourself familiar with: login, modules & compiling, job submission**
- `cp -a ~cblass/HPC .` → copy the HPC exercises
- `cd ~/HPC` → go to the folder

```
module load foss/2019a
```
→ @viz.hpc.fs.uni-lj.si → GCC 8.2.0 & OpenMPI/3.1.3

```
export MPI_PROCESSES=4
export OMP_NUM_THREADS=6
```
→ **`co-co.c`** demo / hello world with conditional compilation

→ only here (built into coco.c): `-DUSE_MPI`

```
cc co-co.c
./a.out | sort -n
sbatch job-serial.sh
```

```
mpicc -DUSE_MPI co-co.c
mpirun -n $MPI_PROCESSES ./a.out | sort -n
sbatch job-mpi.sh
```

```
cc -fopenmp co-co.c
./a.out | sort -n
sbatch job-openmp.sh
```

```
mpicc -DUSE_MPI -fopenmp co-co.c
mpirun -n $MPI_PROCESSES ./a.out | sort -n
sbatch job-hybrid.sh
```

# Thank you for your attention!

http://sctrain.eu/

Univerza *v Ljubljani*

TECHNISCHE UNIVERSITÄT WIEN

CINECA consorzio interuniversitario

VSB TECHNICAL UNIVERSITY OF OSTRAVA | IT4INNOVATIONS NATIONAL SUPERCOMPUTING CENTER

Co-funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission.
This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.